Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Mobile Cloud Middleware

Huber Flores[*], Satish Narayana Srirama

University of Tartu, Institute of Computer Science, Mobile Cloud Lab., J. Liivi 2, Tartu, Estonia

ARTICLE INFO

Article history:
Received 30 September 2012
Received in revised form 9 September 2013
Accepted 9 September 2013
Available online 23 September 2013

Keywords:
Mobile Cloud Computing
Task delegation
Code offloading
Interoperability

ABSTRACT

Mobile Cloud Computing (MCC) is arising as a prominent research area that is seeking to bring the massive advantages of the cloud to the constrained smartphones. Mobile devices are looking towards cloud-aware techniques, driven by their growing interest to provide ubiquitous PC-like functionality to mobile users. These functionalities mainly target at increasing storage and computational capabilities. Smartphones may integrate those functionalities from different cloud levels, in a service oriented manner within the mobile applications, so that a mobile task can be delegated by direct invocation of a service. However, developing these kind of mobile cloud applications requires to integrate and consider multiple aspects of the clouds, such as resource-intensive processing, programmatically provisioning of resources (Web APIs) and cloud intercommunication. To overcome these issues, we have developed a Mobile Cloud Middleware (MCM) framework, which addresses the issues of interoperability across multiple clouds, asynchronous delegation of mobile tasks and dynamic allocation of cloud infrastructure. MCM also fosters the integration and orchestration of mobile tasks delegated with minimal data transfer. A prototype of MCM is developed and several applications are demonstrated in different domains. To verify the scalability of MCM, load tests are also performed on the hybrid cloud resources. The detailed performance analysis of the middleware framework shows that MCM improves the quality of service for mobiles and helps in maintaining soft-real time responses for mobile cloud applications.

1. Introduction

Mobile Cloud Computing (MCC) is arising as a prominent research area that is seeking to bring the massive advantages of the cloud to the constrained smartphones and to enhance the telecommunication insfrastructures with self-adaptive behavior for the provisioning of scalable mobile cloud services. MCC focuses on the benefits that can be achieved by the mobile resources when a mobile operation such as data storage or processing is delegated or offloaded to the cloud (Satyanarayanan et al., 2009; Flores et al., 2011; Flores and Srirama, 2013). These benefits include extended battery lifetime, improved storage capacity and increased processing power, thus enriching the mobile applications along with the mobile user experience. Moreover, MCC focuses on finding an optimal configuration of a mobile cloud infrastructure in order to handle the oscillating telecommunication loads (scale-out), to facilitate the process of deploying services without managing the underlying technology (on the fly) and to reduce operational and provisioning costs (pay-as-you-go model) (Flores and Srirama, 2012).

Mobile cloud applications (Flores et al., 2012) are considered as the next generation of mobile applications, due to their promise of bonded cloud functionality that augment processing capabilities on demand, power-aware decision mechanisms that allow to utilize efficiently the resources of the device, and their dynamic resource allocation approaches that allow to program and utilize cloud services at different levels (SaaS, IaaS, PaaS). However, adapting the cloud paradigm for mobile devices is still in its infancy and several issues are yet to be answered. Some of the prominent questions are; how to decide from the smartphone, the deployment aspects (e.g. type of instance) of a mobile task delegated to the cloud? How to decrease the effort and complexity of developing a mobile application that requires accessing distributed hybrid cloud architectures? How to handle a multi-cloud operation without overloading the mobile resources? How to keep the properties (e.g. memory use, application size, etc.) of a mobile cloud application similar to that of a native one?

Hybrid cloud and cloud interoperability are essential for mobile scenarios in order to foster the de-coupling of the handset to a specific cloud vendor, to enrich the mobile applications with the variety of cloud services provided on the Web and to create new business opportunities and alliances (Zeng et al., 2004). However, developing a mobile cloud application involves adapting different Web APIs from different cloud vendors within a native mobile platform. Vendors generally offer the Web API as an interface that

* Corresponding author. Tel.: +372 56090142.
E-mail addresses: huber@ut.ee (H. Flores), srirama@ut.ee (S.N. Srirama).

0164-1212/$ – see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2013.09.012

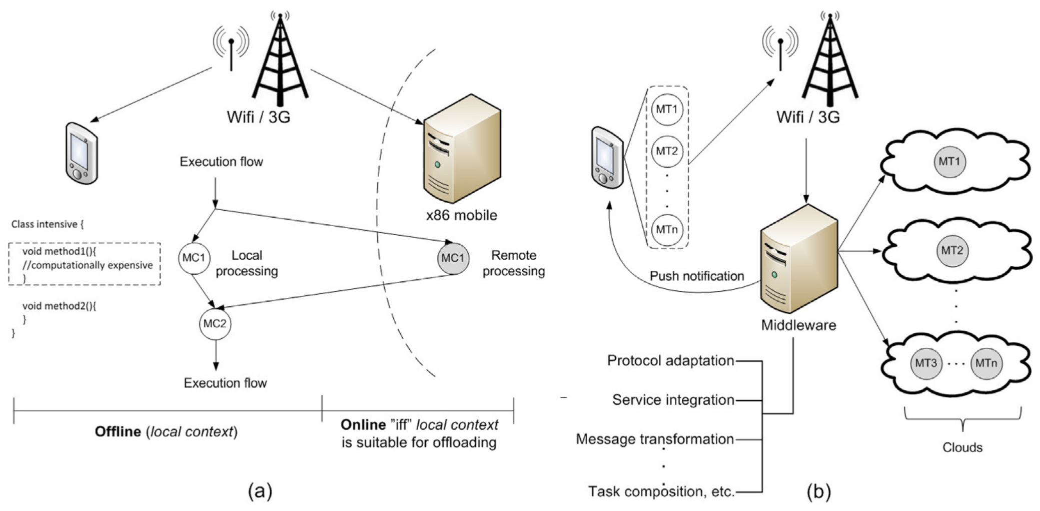


Fig. 1. (a) Code offloading architecture and (b) task delegation architecture.

allows programming the dynamic computational infrastructure that support massively parallel computing. Deploying a Web API on a handset is demanding for the mobile operating system due to many reasons like compiler limitations, additional dependencies, code incompatibility etc., and thus in most of the cases the deployment just fails. Moreover, adapting a Web API requires specialized knowledge of low level programming techniques and most of the solutions are implemented as ad-hoc.

In terms of data storage facilitation of the cloud, existing mobile cloud approaches such as data synchronization (via SyncML) allow the deployment of a Web API on the device for retrieving data from the cloud. For instance, Funambol (Onetti and Capobianco, 2005) or gdata-calendar Web API can be integrated to an Android application to synchronize calendar information (e.g. alarms, tasks, etc.). However, this approach focuses on replicating the data located in the cloud with the data located in the handset, which is not a real improvement. Alternatively, cloud services can be encapsulated as Web services that can be invoked directly using a simple REST mobile client. However, due to the time consuming nature of a cloud request, this can cause an overhead in the mobile resources, without a proper asynchronous communication mechanism. Moreover, by using a REST mobile client, the device is forced to perform multiple transactions and to handle the results of those transactions locally, which is costly from the energy point of view of the handset. The number of transactions are directly associated with the number of cloud services utilized in the mobile cloud application.

To counter the problems such as the interoperability across multiple clouds, invocation of data-intensive processing from the handset, dynamic configuration of execution properties of a delegated task in the cloud, and to introduce the platform independence feature for the mobile cloud applications, we propose the Mobile Cloud Middleware (MCM) (Flores et al., 2011). The middleware abstracts the Web API of different/multiple cloud levels and provides a unique interface that responds (JSON-based) according to the cloud services requested (REST-based). MCM provides multiple internal components and adapters, which manage the connection and communication between different clouds. Since most of the cloud services require significant time to process the request; it is logical to have asynchronous invocation of the cloud service. Asynchronicity is added to the MCM by using push notification services provided by different mobile application platforms and by extending the capabilities of a XMPP-based IM infrastructure (Flores and Srirama, 2013).

Furthermore, MCM fosters a flexible and scalable approach to create hybrid cloud mobile applications based on declarative service composition. Service composition is considered for representing each mobile task to be delegated as a MCM delegation component that is depicted graphically in an Eclipse plugin so that the cloud services that conform a mobile cloud application can be modeled as a data-flow structure based on user driven specifications. Once developed, a composed task is deployed within the middleware for execution that is triggered by a single invocation from the mobile. Finally, MCM prototype was extensively analyzed for its performance and scalability under heavy loads, and the details are addressed in further sections.

The rest of this paper is organized as follows: Section 2 addresses the related work and highlights the issues and challenges that need to be investigated in order to implement a mobile cloud architecture based on delegation. Section 3 introduces the concept of Mobile Cloud Middleware. Section 4 discusses the MCM hybrid cloud composition mechanism. Section 5 presents a scalability analysis of the framework and Section 6 concludes the paper with future research directions.

2. Related work and state of the art

Recently, there has been a growing interest to bind cloud resources to low-power devices such as smartphones in order to provide PC-like functionality to the mobile users (Balan et al., 2002). This loose integration happens through a mediator (aka middleware) that controls every aspect in the communication and coordinates the interaction (online/offline) between the back-end infrastructure and the mobile device. Currently, two main middleware criterias are utilized to bring the cloud to the vicinity of a mobile; offloading and delegation. The architecture for each approach is shown in Fig. 1.

In a delegation model, a mobile device utilizes the cloud in a service oriented manner in order to integrate services running at different cloud levels within the mobile applications so that a mobile task (MT) can be delegated by invoking a cloud service from the handset. This kind of approach requires to have always available network connectivity and conceptually a mobile task is delegated when it is computationally unfeasible for the mobile resources (Flores et al., 2011). In contrast, in an offloading model, a mobile application may be partitioned (e.g. methods, classes) and analyzed *a priori* (at development stages) or *a posteriori* (at runtime) so that the most computational expensive operations (aka mobile components) at code level can be identified and offloaded for remote processing (Gu et al., 2004; Li et al., 2001). A mobile component (MC) may be offloaded or not, depending on the impact of its execution over the mobile resources. Conceptually, offloading is preferable only if a mobile component requires high amounts of computational processing and the same time, low amounts of data need to be sent in the communication. Otherwise, offloading is not encouraged (Kumar and Lu, 2010). Moreover, while delegation enhances the mobile with different functionalities that target multiple aspects of cloud infrastructure, offloading just enhances the computational operations running at the handset when a suitable mobile context is sensed.

Multiple research works have proposed solutions to bind computational cloud services to mobiles (Cuervo et al., 2010; Chun et al., 2011) from an offloading perspective, mostly due to virtualization technologies and their synchronization primitives, enabling transparent migration and execution of intermediate code. However, in an offloading model much of the advantages of cloud computing are left unexploited and poorly considered. For example, a cloud does not just mean a virtual machine or a pool of servers which are accessible from the Internet. It has its own intrinsic features like elasticity, utility computing, fine-grained billing, illusion of infinite resources, parallelization of tasks, etc. We also have explored the offloading model, and to counter most of these issues we proposed the EMCO framework; its details are addressed in a different publication (Flores and Srirama, 2013).

However, a mobile application may enrich its functionality from other strategies that consider multiple cloud sources for delegating and triggering resource-intensive tasks (e.g MapReduce based) in which their execution can be configurable and parallelizable among multiple servers (aka multi-cloud operation). For example, sensor data (e.g accelerometer) may be collected in cloud storage from multiple devices, and later this information can be utilized to train a classifier of complex human activities so that when a mobile application needs to recognize an activity, it can delegate the mobile task to the cloud through the middleware (Paniagua et al., 2012).

We focus in this paper, the delegation of mobile tasks and their composition into multi-cloud operations. A multi-cloud operation consists of delegating mobile tasks to a diversity of cloud services (e.g. from the infrastructure level, platform level, etc.) located on different clouds (e.g. public, private, etc.) and orchestating (e.g. parallel, sequential, etc.) those transactions for achieving a common purpose. Developing this kind of mobile cloud applications requires, from the cloud perspective, the interoperability among cloud architectures; from the mobile perspective, a considerable effort to manage the complexity of working with distributed cloud services, a specialized knowledge to adapt each particular Web API to a specific mobile platform and an efficient approach that avoids the unnecessary data transfer.

From a delegation perspective, current solutions try to overcome the problems of multi-cloud service integration in mashups that can be accessible from the handset. However, they just focus on primitive services at SaaS level, which are not data-intensive (Wang and Deters, 2009). Finally, middleware approaches for mobile task delegation similar to MCM have been addressed in the literature. MCCM (Mobile Cloud Computing Middleware) (Wang and Deters, 2009) is a framework that stands between mobile and cloud, for the creation of mobile mashup applications that are limited to the use of SaaS. Moreover, the middleware uses synchronous communication in order to invoke and process a cloud transaction. Cloud agency (Aversa et al., 2010) 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.

Unlike previous solutions, we proposed in this paper a framework (MCM) that uses asynchronous resource-intensive task delegation for augmenting the computational and storage capabilities of the mobile devices. MCM addresses the issues of interoperability across multiple clouds and dynamic allocation of cloud infrastructure. Moreover, MCM tries to use service composition mechanisms in order to decrease the energetic effort (minimize communication channels) of utilizing multiple clouds in a mobile mashup application.

However, since MCM defines a new interaction model for mobile cloud applications, there are several infrastructural challenges that must be overcome. Consequently, the rest of this section examines those issues in detail and highlights the opportunities of designing the mobile cloud architectures of the future.

2.1. Middleware for Mobile Cloud Computing: challenges and opportunities

In the emerging ecosystem of Mobile Cloud Computing, a rich mobile application is one, in which through a soft real-time interactivity, huge amounts of data are processed and presented to the user as a single result. Performing such operations in a mobile phone is difficult due to computational limitations of the handset. Thus, computation offloading/delegation is needed for augmenting on demand the capabilities of the mobile resources.

2.1.1. Delegation of mobile tasks to cloud

Computational/storage delegation has become a common operation that is supported by any mobile platform through various mechanisms (e.g. Web sockets, REST-based requests, etc.). Different mobile platforms or versions of the same mobile platform implement different approaches for managing network communication. For instance, Android platform level-10 handles REST-based requests synchronously. In contrast, Android platform level-16 forces the developer to extend any network communication with the *AsyncTask Class* running on a different thread so that it will be executed asynchronously in the mobile background. This guarantees to keep the real-time interactivity of a mobile application and to maintain the normal execution of a mobile application, if an exception arises.

A mobile application that requires resource-intensive functionality of the cloud can benefit from this asynchronous communication in order to delegate and monitor the status of a time consuming operation. However, this approach introduces several drawbacks such as energy consumption (e.g. keeping an open connection while transaction is performed), reliability in the communication (e.g. what happens if the connection fails?) and recoverability of a cloud transaction among others. Consequently, asynchronous middleware support is encouraged for delegating data-intensive tasks to the cloud (Flores et al., 2011). In this asynchronous process, when a mobile application sends a request to access a cloud service, 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 so that the mobile device can continue with other activities. Once the process is finished at the cloud, an asynchronous message 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.

Mobile application may rely on push technologies (aka notification services) for dealing with remote executions, and thus avoiding the effect of polling caused by protocols such as HTTP. However, these mechanisms are considered as black box services which have certain constraints and limitations such as being platform specific (e.g. AC2DM, Google Inc, 2013a/GCM, Google Inc, 2013b for Android, APNS, Apple Inc, 2013a for iOS, MPNS, Microsoft Inc, 2013 for Windows Phone 7, etc.), the size of the message that can be pushed into the device (e.g. 1024 bytes for Android, 256 bytes for iOS, 4096 bytes for Windows Phone 7, etc.), the number of the messages that can be sent to a single handset (e.g. 200,000 for AC2DM, 500 for MPNS without authentication) and the maintenance of a particular infrastructure (e.g. application server, mobile clients) that relies on mobile platform features (e.g. Broadcast receivers for Android, URI-channels for Windows Phone 7 etc.) and its related cloud vendor technology (e.g. authentication mechanisms, communication protocols, etc.). 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.

2.1.2. Cloud service integration for mobile applications

While cloud infrastructure is programmable through the utilization of the Web API, different clouds present different levels of granularity for configuring the cloud resources. Depending on the cloud vendor architecture, a Web API may be used for deploying applications from the scratch (e.g. MapReduce) or for accessing existent functionality, which can be integrated with other software applications. For instance, Amazon API and typica API (typica, 2013) allow to manage EC2 instances (run scripts, attached volumes, etc.), jetS3t (jets3t, 2013) API provides access to S3/Walrus and GData API (Google and Inc., 2013c) enables configuring services such as calendar, analytics, etc. Consequently, software applications that require cloud intercommunication are forced to implement multiple Web APIs.

Since a Web API may suffer from disuses, changes or replacements with time due to many reasons such as new Web API releases, improvements, etc., the development of applications becomes tightly coupled and difficult to port, to reuse and to maintain. To address most of these problems, several open source projects have been started. For instance, jclouds (2013) is a multi-cloud library that claims transparent communication with different cloud providers and the reuse of the source code while working with different services. Jclouds provides a core library which contains the core functionality and a set of libraries which handle the communication with any particular cloud. Current version of jclouds supports Amazon, GoGrid, VMWare, Azure and Rackspace. Other projects such as Apache Libcloud (Foundation, 2013) and Dasein Cloud API (dasein, 2013) also provide a Web API that abstracts away differences among multiple cloud providers, however currently, jclouds API is the one that supports more cloud vendors. Other projects such as deltacloud (Delta Cloud, 2013) focus on managing particular services with the same Web API. For instance, deltacloud allows utilizing the same code routines for starting an instance in Amazon and in Rackspace.

Even though there are many Web APIs that can be used for abstracting the communication with different cloud vendors, most of them are not deployable within a mobile platform due to several drawbacks such as additional dependencies, incompatibility with the mobile platform, integration with the compiler, etc. For instance, the dalvik virtual machine of Android offers just a set of java functionality. Consequently, the richness of the language can not be exploited and libraries such as jclouds or typica can not be executed on mobile platforms.

Furthermore, the development of mobile cloud applications using Web APIs, increases dramatically the effort of implementing simple operations such as offloading a file to remote storage, etc. For instance, we have ported jetS3t API for Android platform level-10 and its apk file requires approximately 4.55 Mb of storage on the mobile. Moreover, the application uses synchronous communication for delegating data to cloud storage (S3/Walrus). Therefore, the mobile resources get frozen while the transaction is being completed ($\approx$ 6 s when uploading a file of 3 Mb using a bandwidth with an upload rate of $\approx$ 1409 kbps). Even more, the source code is not compatible with higher versions of Android. Consequently, the application is not portable and a complete re-implementation is needed for using it within other Android versions.

Currently, Web APIs for mobiles are still under development and often target simple services such as storage. For instance, at the time of writing this paper, Amazon has just released the beta version of the Web API for accessing S3 for both Android and iOS platforms. Thus, more testing is needed over this Web APIs, before they can actually be embedded into the mobile applications that can be distributed in the mobile market.

3. Mobile Cloud Middleware

MCM is introduced as an intermediary between the mobile phones and the clouds for managing asynchronous delegation of mobile tasks to cloud resources. MCM hides the complexity of dealing with multiple cloud providers by abstracting the Web APIs from different clouds in a common operation level so that the service functionality of the middleware can be added based on combining different cloud services. Moreover, MCM enables the development of customized services based on service composition, in order to decrease the number of mobile-to-cloud transactions needed in a mobile cloud application. The architecture is shown in Fig. 2. When a mobile application tries to delegate a mobile task to a cloud, it sends a request to the TP-Handler component of the middleware, which can be based on several protocols like the Hypertext Transfer Protocol (HTTP) or the Extensible Messaging and Presence Protocol (XMPP). The request is immediately followed by an acknowledgement from MCM (freeing the mobile) and it consists of a URL with the name of the server, the service being requested and the configuration parameters, which are applied on the cloud resources for executing the task. For instance, http://ec2-x-x-x-x.compute-1.amazonaws.com:8080/MCM/SensorAnalysis represents a processing task that triggers a MapReduce activity recognition algorithm over a set of sensor data (accelerometer and gyroscope) collected by the mobile in order to discover reading patterns. The request also includes information regarding the cloud vendor, type of instance, region, image identifier and the rest of parameters associated with that particular service. Notice that different services involves the utilization of different parameters within a request. Once at the MCM, the request is then processed by the MCM-Manager for creating the adapters that will be used in the transactional process with the clouds. Fig. 3 shows the interaction logic of the components of MCM.

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. 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 a multi-cloud operation. Based on the request, the service transaction is managed by the Interoperability-API-Engine or the Composition-Engine (single or compose service invocation, respectively).

In the case of a single service invocation, the request is handled by the Interoperability-API-Engine, which selects, at runtime, based on request the Web API to utilize in a cloud transaction. The engine extends the interoperability features to the Adapter-Servlets component, which contains the set of routines/functions that are used to invoke a specific cloud service. The MCM-Manager encapsulates the API and the routine in an adapter for performing the

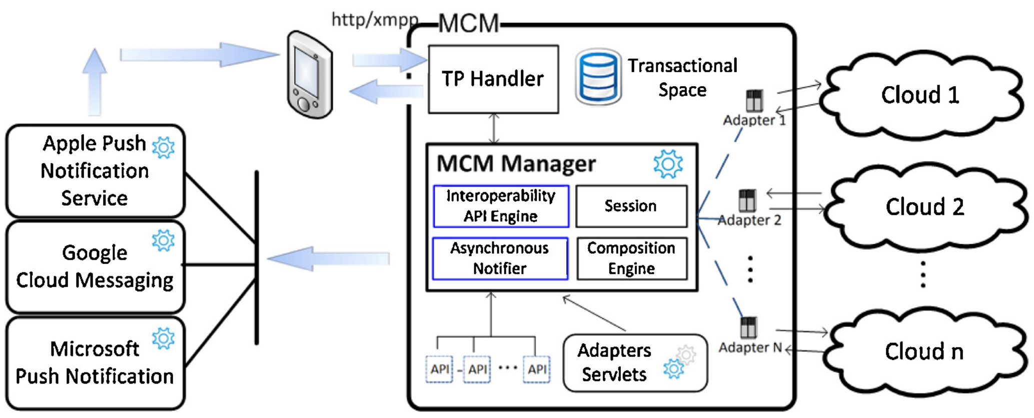


Fig. 2. Architecture of the Mobile Cloud Middleware.

transaction and accessing the XaaS. Basically, an adapter is a runnable abstract class that provides a generic behavior for a mobile task. We utilized Gson google-gson (2013) for loading and serializing mobile tasks.

In contrast, if the request consists of a composite service invocation, the Composition-Engine (explained in detail in Section 4) interprets the service schema and acquires the adapters needed for executing the services from the Interoperability-API-Engine. The hybrid cloud property of an adapter is achieved by using the Clojure (Hickey, 2008) component that encapsulates several Web APIs. Each adapter keeps the connection alive between MCM and the cloud and monitors the status of each task running at the cloud. An adapter can store data in the transactional space, in order to be used by another adapter.

Once the single/composite service transaction is completed, the result is sent back to the handset in a JSON (JavaScript Object Notation) format. MCM-Manager uses the asynchronous notification feature to push the response back to the handset. Asynchronicity is added to the MCM by implementing the Google Cloud Messaging for Android (GCM), the Apple Push Notification Services (APNS) and the Microsoft Push Notification Service (MPNS) protocols for Android, iOS and Windows Phone 7 respectively. APNS messages are sent through binary interface that uses streaming TCP socket design. Forwarding messages to device happens through constantly open IP connection. MPNS messages are delivered through URI channels that can be invoked via REST/POST with the possibility of create up to 30 different channels for pushing data to the applications (one application corresponds to one channel).

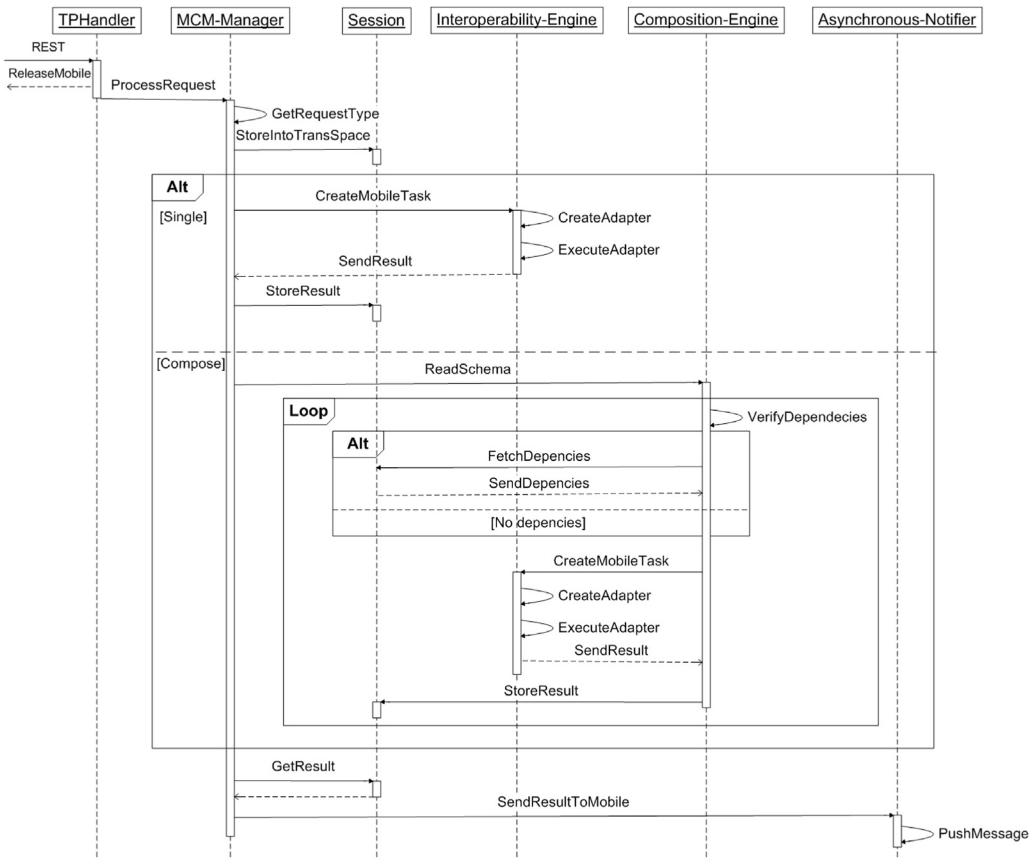


Fig. 3. Interaction logic of the components of Mobile Cloud Middleware.

Similarly, GCM mechanism lets to push a message into a queue of a third party notification service, which is later sent to the device. MCM also supports previous Android Cloud to Device Messaging Framework (AC2DM), which was recently deprecated by Google, but shares similar implementation with GCM. Once the message is received, the system wakes up the application via Intent Broadcast, passing the raw message data received straight to the application.

Alternatively, MCM also has support for sending messages using the Mobile Cloud Messaging Framework based on XMPP (Flores and Srirama, 2013), which extends an ejabberd (ejabberd, 2013) infrastructure for delivering messages to any smartphone that implements an XMPP mobile client. Basically, the framework reuses the IM infrastructure features (e.g. JIDs, authentication, etc.) for sending notification based on the JID. Messages consist of XML stanzas with a new attribute that differentiates the message from the other stanzas in the mobile client.

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. Web APIs are encapsulated using Clojure, and thus are accessed by a common API. This encapsulation guarantees updating deprecated Web APIs with newer versions which are released constantly by the cloud vendor. Moreover, Clojure is also considered as its distributed nature introduces flexibility for scaling the applications horizontally, and thus augmenting the faul-tolerant properties of the overall system. Moreover, Clojure also provides portability in the design. Thus, decreasing the effort of migrating the whole architecture to more suitable telecommunication programming languages.

Hybrid cloud services from Amazon EC2, S3, Google and Eucalyptus based private cloud are considered. Jets3t API enables the access to the storage service of Amazon and Google from MCM. Jets3t is an open source API that handles the maintenance for buckets and objects (creation, deletion, modification). A modified version of the API was implemented for handling the storage service of Eucalyptus, Walrus. Latest version of jets3t also handles synchronization of objects and folders from the cloud. Typica API and the Amazon API are used to manage (turn on/off, attach volumes) the instances from Eucalyptus and EC2 respectively. MCM also has support for SaaS from facebook, Google, AlchemyAPI.com and face.com.

MCM and the resource intensive tasks can easily be envisioned in several scenarios. For instance, we have developed several mobile applications that benefit from going cloud-aware. Zompopo (Flores et al., 2012), consists of the provisioning of context-aware services for processing data collected by the accelerometer with the purpose of creating an intelligent calendar. CroudSTag (Srirama et al., 2012), consists of the formation of a social group by recognizing people in a set of pictures stored in the cloud. Finally, Bakabs (Flores et al., 2012) is an application that helps in managing the cloud resources themselves from the mobile, by applying linear programming techniques for determining optimized cluster configurations for the provisioning of Web-based applications.

4. Hybrid cloud service composition of MCM

While delegating a mobile task to the cloud may enrich the mobile applications with sophisticated functionality and release the mobile resources of heavy processing, frequent delegation rates (mobile-to-cloud transactions) may introduce costly computational expenses for the handset. Therefore, approches that enables to avoid unnecessary communication overhead such as those based on service composition must be encouraged. Service composition consists of the integration and re-utilization of existent distributed services for building more complex and thus more rich service structures. Most of these structures are developed graphically as control or data flow based models. For instance, YahooPipes is a composition tool based on the concept of Unix pipes. A pipe depicts a data resource on the Web (e.g. RSS feeds, etc.) that can be filtered, sorted or translated. Several pipes can be joined together into a single result for extracting information according to the needs of the user.

Service composition enriches a single service invocation by adding, executing, orchestrating and joining multiple service requests (treating each service as an individual task) in a common operation. In order to foster a flexible and scalable approach, to compose hybrid cloud services and to bring the benefits of service composition to the mobiles, MCM implemented a service composition mechanism that enables to automate the execution of a workflow structure with a single mobile offloading. Moreover, tasks are composed using a declarative approach, where the workflow is modeled graphically and deployed in the middleware for execution by using an Eclipse plugin.

4.1. Hybrid cloud service implementation

The MCM composition editor is developed as a plugin for the Indigo version of Eclipse. Basically, after configuring the plugin with the remote location of the MCM, it retrieves the list of services that are available at the middleware for mobile delegation (e.g. sensor analysis, text extraction, etc.) so that each service (cloud transaction) can be depicted graphically as a *MCM delegatiOn Component* (MOC). The plugin is developed by combining the capabilities of GEF (Foundation, 2013) (Graphical Editing Framework) and EMF (Foundation, 2013a) (Eclipse Modeling Framework). GEF is used for creating the graphical editor that consists of a palette of tools (MOC, connector and mouse pointer) and a blank frame in which the MOC is dragged and dropped multiple times for building the work flow. Each MOC provides standard inputs and standard outputs that represent the receiving/sending of messages in JSON format that are used for the intercommunication of components. Thus, the combination of MOCs is tied to the matching between inputs and outputs. Since an individual service usually is triggered for execution by a REST request that responds with a JSON payload, when passing parameters between MOCs, the JSON payload is analyzed and all the necessary parameters are extracted from it for creating a request that matches the input of the next MOC. This request is loaded into the MOC using Gson so that it can be executed.

A MOC is drawn by extending the draw2d (Foundation, 2013) library (Node class) with a label object. When the component is active on the frame (focus on), its properties view pops up so that the component can be customized. The properties view consists of (1) a description category that is used to specify the MCM service which is selected from a list (previously retrieved) contained in a ComboBoxPropertyDescriptor; once the service is selected, the service name is set in the label object with its respective URL value as attribute, (2) an inputs category that describes the list of inputs of the service selected in (1), which basically represents the execution properties that depend on the Web API being utilized, along with the required data for executing the task at the cloud. Input parameters may be dynamic or static. A dynamic input value is one that is obtained from a connected MOC. In contrast, a static input value is defined by the user as plain value (e.g. file path), (3) an outputs category that describes the list of outputs of the service selected in (1), which represents the results of the cloud transaction. Both (2) and (3) also provide information related to other MOC which may be connected to its inputs/outputs.

EMF is used for creating an XML-based representation of the work flow (serialization of the model). This serialization consists of describing each MOC and connection relations as data type and flow conditions, respectively, so that they can be deployed and published at the middleware for execution using the

MCM-Composition engine. Notice that the XML description is utilized mainly for validating the execution of the workflow and for checking the dependencies prior to the execution of a MOC. In the XML description, each data type is described by mapping its graphical representation into object properties (e.g. height, weight, etc.) and input/output attributes (e.g. bucket name, security group identifier, etc.). For instance, the MOC *Start Instance* by default establishes a height of 50px and width of 150px, it is invoked by performing a request to the URL with value http://ec2-x-x-x-x.compute-1.amazonaws.com:8080/MCM-/StartInstance and it requires as input parameters, an image id (e.g. ami-xxxxxxxx), a provider name (e.g. amazon), an instance size (e.g. large, small, etc.), a region (e.g. us-east-1c) and an username, in order to generate an Instance object of typica library as output. This Instance object is passed to another MOC (e.g. RunScript) as serialized dynamic parameter along with the path of the file that is defined by the user as static input.

Once the XML description of the composed service is deployed at the middleware, the MCM-Composition engine is in charge of performing three tasks for executing the new composed service. These tasks consist of publishing, converting and executing. The publishing task is performed once the file is deployed. It consists of assigning a unique URL for invoking the service via the TP Handler. The service is named according to the name of the file which is deployed (if name is already existing then service cannot be deployed). The converting task is performed once the service has been requested. It consists of parsing the XML file for getting the individual information of each MOC. This information is passed to the Interoperability API engine for creating the adapters (as described in Section 3) and performing the cloud transaction. Notice that in runtime a MOC may have one or more associated adapters. Finally, the executing task is in charge of mapping the entire file (from top to bottom) and executing each service using the adapters created previously. The execution follows the structure of the workflow considering any parallel or sequential tasks. When a service is executed, the task is monitored by the MCM from start to end. Once the result of each task is obtained, a request is created with it and is redirected to the next MOC.

4.2. Hybrid mobile cloud application – demo scenario

To demonstrate the composition feature of MCM, a hybrid mobile cloud application has been developed using MCM and its composition tools. We have developed the application in Android platform due to its popularity in the mobile market and unrestricted uses. However, the application could be developed for iOS or Windows Phone 7. The application benefits from its multi-cloud nature for performing a variety of cloud analysis, which are invoked by a single transaction. The aim of the application is to figure out whether the user likes or not the content of the Web pages that the user is reading on the mobile, so that the text of the most interesting articles can be extracted along with some keywords, which are obtained through machine learning analysis. Whether the user likes a particular page is calculated based on approximating the angle at which the phone is held to a fixed threshold which is set by obtaining stable gyroscope measurements. Later, from that point, it is sensed whether the handset experiments show intense movement or not. The information extracted from the analysis is stored in a Web document on the cloud for being accessed later through a URL using the Web browser of a mobile or a standalone computer.

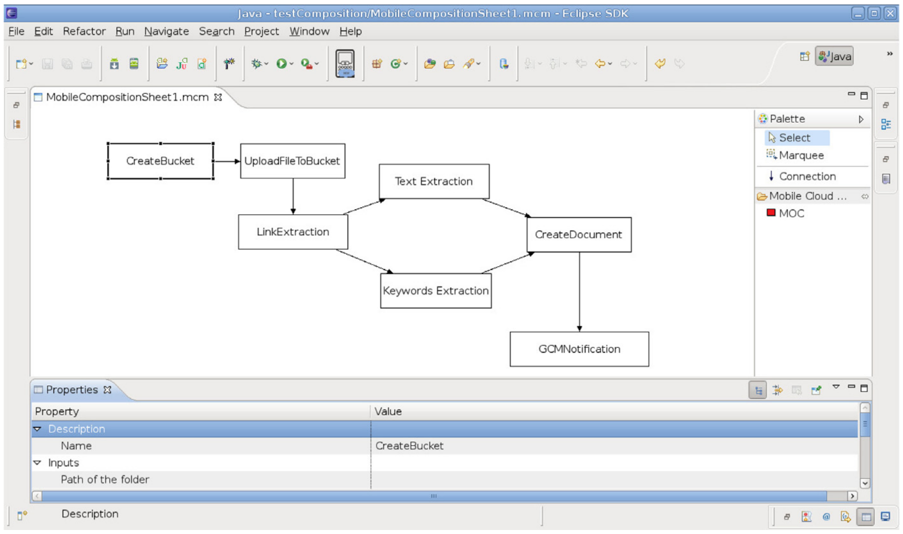
Since most of the functionality of the mobile application is located on multiple clouds and is managed by the MCM, the client application running on the mobile is lighter and simple to build. The application consists of an EditText for typing an URL and a WebView for displaying its content. Once packed in an apk file, the application requires $\approx$ 512 kb of storage on the mobile. When the application is launched, a concurrent process is triggered in the background. This process is used to store the information sensed by the accelerometer and the gyroscope sensors along with the active URL of the WebView, into a SQLite database. With each measurement written to the database, one tuple of the form, <t_i, [x_i, y_i, z_i], [$g1_i$, $g2_i$, $g3_i$], URL > is stored. Where t_i represents a timestamp measured in seconds, [x_i, y_i, z_i] represents the data of the three axes of the accelerometer measured at time t_i and [$g1_i$, $g2_i$, $g3_i$] is the data of the three directions of the gyroscope measured at time t_i.

When the application is closed (goes to the background), an *Async task* is executed on the *OnPause* method of the application. This task consists of a unique offloading to the URL of the compose service published at MCM. This offloading contains (1) the data that will be used in the multi-cloud analysis (in this case, the database file), (2) the execution properties that allow to configure the cloud resources in runtime (bucket name, Amazon image id, instance size, region, Amazon username, eucalyptus image id, eucalyptus username, sensor analysis provider, text analysis provider, keyword analysis provider and document name) and (3) a GCM request for registering the mobile at Google notification servers. The registration is mandatory for sending messages to the mobile using MCM and it is required only once. On this state, the application (activity) is also terminated so that the user can continue with other activities. However, the application will be re-activated via Broadcast Intent once a message from the notification service arrives with the result of the composition (URL of the document). The application uses MCM for invoking the services from Amazon, Eucalyptus, Google and AlchemyAPI.com. The cloud services are defined for composition in the Eclipse plugin as follows.

After adding a file with extension **.mcm* in the Java perspective of Eclipse, the composition tools of the plugin get enabled. The following services implemented at MCM are considered for the composition; *CreateBucket* represents a service that makes use of the Web Amazon API for creating a bucket in S3 and requires a bucket name as parameter. *UploadFileToBucket* uses jetS3t to locate objects in a specific Amazon bucket and requires a bucket name target. *StartInstance*, *EndInstance* and *RunScript* use typica for handling computational instances in Amazon and Eucalyptus. Here, the infrastructure parameters are used (instance size, region, amazon username, amazon image id, eucalyptus username, etc.). *KeywordExtraction* can be assumed as a black box service for the text analysis that is available by using the Web Alchemy API (text analysis provider). *CreateDocument* uses the gdata-docs API for creating the document, whose name is passed as parameter. Finally, *GCMNotification* represents the push notification service of Google.

Before creating the workflow of the application, some subcomposition is needed first. The subcomposition process consists of creating the *LinkExtraction* and *TextExtraction* services. The LinkExtraction is an Amazon computational service that applies a MapReduce activity recognition algorithm over the sensor data in order to understand how the user was holding the handset, and thus extracting the more interesting URLs (explained in Appendix A). LinkExtraction is created by connecting the MOCs StartInstance, RunScript and EndInstance. Similarly, TextExtraction is a service running on Eucalyptus that implements the BulletParser (LightCrawler, 2013) for extracting the text of a set of URLs (Web pages). TextExtraction shares the same logic as LinkExtraction and is created by connecting the MOCs StartInstance, RunScript and EndInstance. However, notice that RunScript differs on both services as the property file path varies.

The workflow is structured by connecting the mentioned MOCs. The logic of the workflow is shown in Fig. 4 and it considers the following. After the database is offloaded to MCM, the middleware creates a bucket to locate the file (CreateBucket) and the location of the file (URL) is passed to the LinkExtraction service to obtain a



Property	Value
Description	
Name	CreateBucket
Inputs	
Path of the folder	

Description

Fig. 4. Workflow executed by MCM and triggered by a single service invocation.

list of URLs. Later, the list of URLs is passed in parallel to the TextExtraction and KeywordExtraction service. Once these services are finished and MCM obtains their results, MCM connects to Google docs to create a document with that information. Finally, MCM sends a message via GCM to the mobile device. The message contains the URL of the document which can be viewed in the browser of the mobile.

On the basis of the functional prototype of the mobile cloud application presented, we can derive that it is possible to handle process intensive hybrid cloud services from the smartphones, via the MCM. Fig. 5 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 delegation time for handling a multi-cloud operation asynchronously, is:

$$T_{mcs_a} \cong T_{tr} + T_m + \Delta T_m + \sum_{i=1}^{n}(T_{te_i} + T_{c_i}) + T_{pn} \quad (1)$$

where T_{tr} is the transmission time taken across the radio link for the invocation between the mobile phone and the MCM. The value

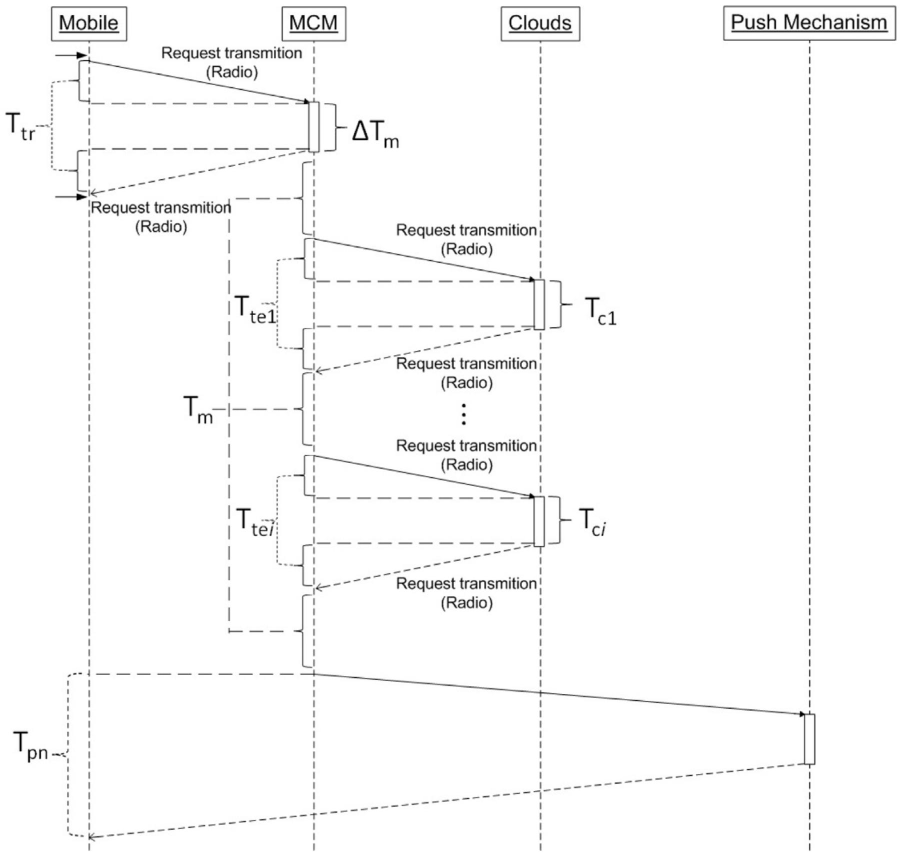

Mobile
MCM
Clouds
Push Mechanism
Request transmition (Radio)
T_{tr}
ΔT_m
Request transmition (Radio)
Request transmition (Radio)
T_{te1}
T_{c1}
Request transmition (Radio)
T_m
Request transmition (Radio)
T_{tei}
T_{ci}
Request transmition (Radio)
T_{pn}

Fig. 5. Timestamps of the application scenario.

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 Transmission Control Protocol (TCP) (Comer, 1995) 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 represents the latency of receiving a request for delegation and sending a response to the mobile about its status. ΔT_m is the extra performance time added by the components of MCM for processing the request. 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 sigma is considered for the composite service case, which involves several mobile cloud service invocations. However, in other cases the access to multiple cloud services may actually happen in parallel. 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 ($Max_{i=1}^{n}(T_{te_i} + T_{c_i})$). Finally, T_{pn}, represents the push notification time, which is the time taken to send the response of the mobile cloud service to the device. With the introduction of support for push notification services at the MCM, the mobile phone just sends the request and gets the acknowledgement back once the multi-cloud operation is performed. However, in this case, the delays completely depend on external sources like the latencies with GCM/APNS/MPNS frameworks and the respective clouds Flores and Srirama (2013).

To analyze the performance of the application, a 5 MB of sensor data was stored in a Amazon bucket. Samsumg Galaxy S II (i9100) with Android 2.3.3, 32 GB of storage, 1 GB of RAM, support for Wi-Fi 802.11 a/b/g/n was considered. Wifi connection was used to connect the mobile to the middleware. So, test cases were taken in a network with an upload rate of $\approx$ 1409 kbps and download rate of $\approx$ 3692 kbps, respectively. However, as mentioned already, estimating the true values of transmission capabilities achieved at a particular 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 of the mobile cloud service invocation of the complete scenario is shown in Fig. 6. The value of $T_{tr} + \Delta T_m$ is quite short (< 870 ms), which is acceptable from the user perspective. So, 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 (workflow) taken for handling the cloud services at MCM, T_{Cloud} ($\sum_{i=1}^{n}(T_{te_i} + T_{c_i})$), is also logical and higher as expected. Cloud processing time also considers provisioning latency of computational resources. This latency represents the time of submitting a request for launching a resource and obtaining the resource in an active state. Figs. 7 and 8 show the execution time for each service that participate in the mashup. Cloud services created from the scratch with IaaS were evaluated on different underlying hardware. Based on the results, we can obseve that the allocation of cloud resources affects the execution time of a delegated mobile task, which is configurable dynamically by the middleware.

Finally, T_{pn} varies depending on current traffic of the GCM service and has an average of $\approx$1.56 s. This notification average is obtained specifically to GCM through an 8 h experiments. In the experiment,messages are sent 1 per second for 15 s in sequence, then with a 30 min sleep time, later followed by another set of 15 messages, repeating the procedure for 8 h (240 messages in total).

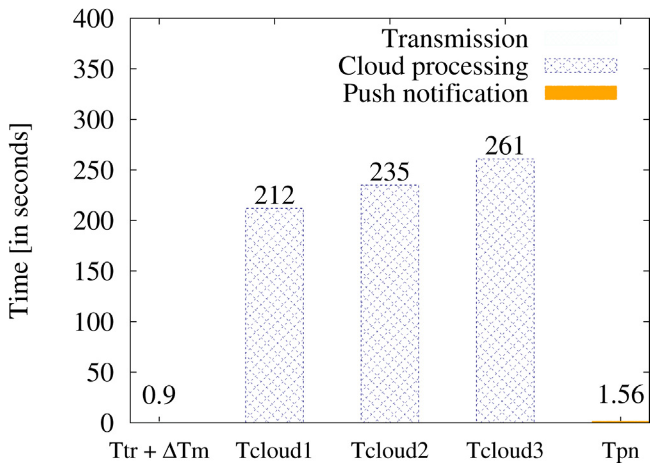


Fig. 6. Mobile cloud service invocation timestamps. Different underlying hardware is considered for the execution of infrastructure services in *Tcloud*. Tcloud1, Tcloud2 and Tcloud3 use large, medium and small instances, respectively. Moreover, Tcloud also considers the provisioning time which has an average of $\approx$150 s of any infrastructure type.

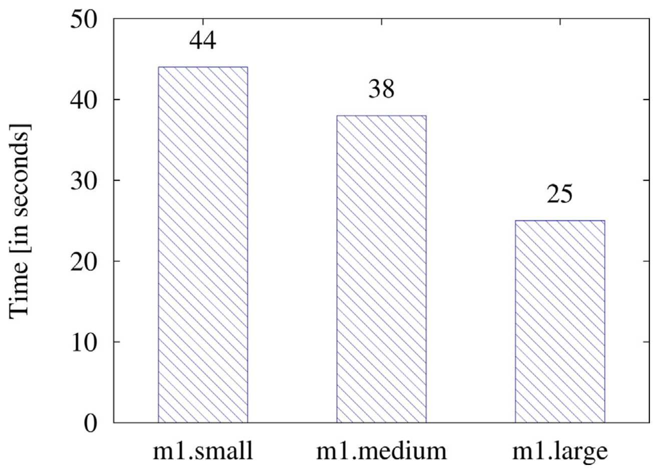


Fig. 7. Execution time of sensor analysis service (*LinkExtraction*) on different instances: service at IaaS level.

The frequency of the messages is set in this way, in order to mitigate the possibility of being detected as a potential attacker (e.g. Denial of Service) to the cloud vendor and to refresh the notification service from a single requester and possible undelivered data. Moreover, the duration of the experiments guarantee to have an overview of the service under different mobile loads, which may arise during different hours of the day. Experiments were conducted in a Wi-Fi network. Results are shown in Fig. 9. We have noticed that GCM messages tend to arrive in a predictable interval when the service is not utilized periodically. Consequently, the reliability of GCM tends to decrease as the number of requests for sending notifications are increased (Flores and Srirama, 2013).

4.3. Hybrid cloud service composition analysis

In order to analyze how MCM service composition adds value to the multi-cloud delegation process of a mobile cloud application, let us consider a similar analysis as the one presented by Kumar and Lu (2010). In their work, they claimed that a single offloading benefits the mobile resources if the mobile component, which is offloaded to the cloud, requires huge amounts of computational resources to be processed and at the same time, the offloading process requires small amounts of data to be sent in the communication. Otherwise, it is preferable not to offload the mobile component and process it locally. We tried to apply same principle in a delegation model as the mobile cloud communication is the one which introduces high overheads in the device (Flores et al., 2012). In this context, a mobile cloud application that uses hybrid cloud services; it has to handle all the invocation logic locally, which is translated into multiple mobile cloud transactions. Moreover, the handset may also be forced to use extra processing power as each mobile task is delegated. This extra processing consists of data manipulation on the results acquired per each cloud transaction. Data manipulation may be needed for joining all the results collected from the cloud services or simply for re-converting the data in a suitable format for triggering the next service.

Due to the resource-intensive/time-consuming nature of the cloud services and the multiple frameworks that enable performing parallel processing on the cloud (e.g. Hadoop), for this analysis we do not consider that a cloud task can be performed on the mobile resources if the above condition is not met. We rather focus on how to decrease the number of mobile cloud communication required for delegating mobile tasks to hybrid cloud resources. With these assumptions in mind, the following example provides a simple analysis.

Suppose that E_w is the total amount of energy wasted by the mobile when executing a mobile cloud application. n is the number of hybrid cloud services in a mobile application (time to offload

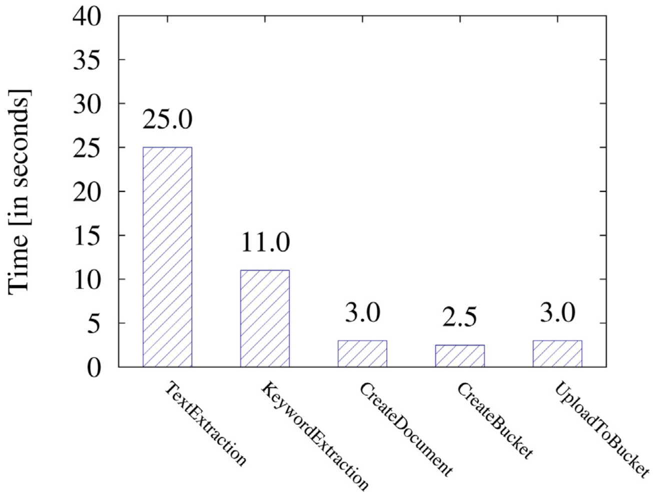


Fig. 8. Execution time of the different cloud services that participate in the workflow: services at SaaS level.

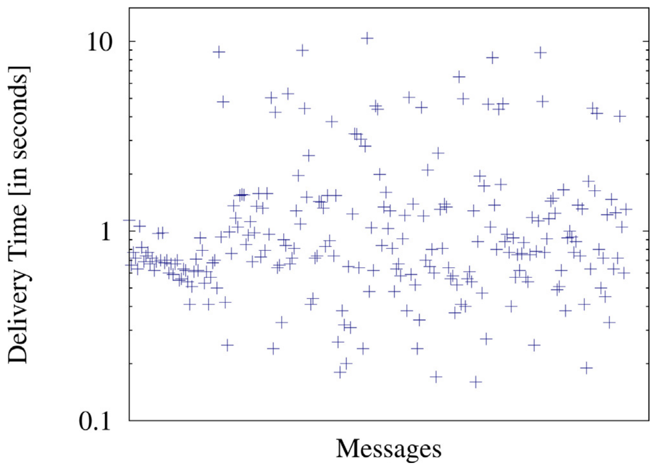


Fig. 9. Delivery rate for GCM. Experiment over a sample of 240 messages.

data). Let B be the bandwith used in the communication between the mobile and the cloud and D is the size of the data in bytes that are exchanged. In each delegation, the mobile will consume (in watts), P_c for the processing performed by the mobile when handling the results of each cloud transaction, P_{tr} for transmitting and receiving data. For this analysis, we consider that transmission and receiving power are the same. However, depending on the approach for sending the result back to the mobile (e.g. notification services, real-time protocols such as XMPP, etc.), both will differ.

Conceptually, if the mobile cloud application is handled by the mobile resources using any approach discussed in Section 2.1, the total time of energy consumed in the multi-cloud offloading process will be:

$$E_w \cong \sum_{j=1}^{n} \left(\left(P_{tr} \times \frac{D_j}{B} \right) + P_{c_j} \right) \tag{2}$$

In contrast, when using MCM and its service composition mechanism, the hybrid cloud service integration occurs at the middleware, and thus n becomes equal to 1. Therefore, one data offloading is needed to trigger a bunch of different cloud services.

$$E_w \cong \left(P_{tr} \times \frac{\sum_{j=1}^{m}(D_j)}{B}) + P_c \right) \tag{3}$$

where $\sum_{j=1}^{m}(D_j)$ represents the data sent per each cloud service requested (if any). Notice that in some cases, no data is sent as the output of one service may be the triggered input of the next service. So in Eq. 3, m ($m \leq n$) is the number of services that participate in the composite service and require input from the mobile. The main purpose of the composition is to alliviate the mobile cloud service invocation T_{mcs_a} from unnecessary latency in the communication and to decrease the transfer of data.

5. Scalability of MCM

While MCM was successful in handling a multi-cloud operation from a mobile cloud application, the capabilities of MCM for handling heavy loads depend on its deployment aspects in the cloud and the dynamic configuration of those at runtime. Dynamic cloud reconfiguration mainly focuses on the distribution of application components that alleviate the entire system, when it is facing a certain condition (e.g. High CPU utilization, etc.), thus, increasing its capabilities for managing concurrency.

To verify the scalability of the middleware, MCM is located in a public cloud (Amazon EC2), in a cluster-based configuration that consists of a front-end node (load balancer – LB) and multiple end-nodes (MCM servers). Fig. 10 shows the deployment scenario. Basically, the front-end node distributes the load between the back-end servers. Therefore, the LB requires a powerful CPU to handle the heavy demand. HAProxy (2013) is considered as the LB as it allows dynamic behavior to the architecture and new MCM servers can be added while the system is running (hot reconfiguration). Back-end servers can be considered as commodity servers which can be replaced without affecting the overall performance of the cluster. Once, the scenario was set up, different mobile loads were simulated using benchmarking tools for testing the horizontal scalability properties of the middleware.

5.1. Scalability analysis of the MCM

Load testing of MCM was performed using Tsung (2013) (open source multi-protocol load testing software). Tsung was deployed in a distributed cluster composed of three nodes running on separated instances (one primary and two secondary nodes). The primary node is in charge of executing the test plan and collecting

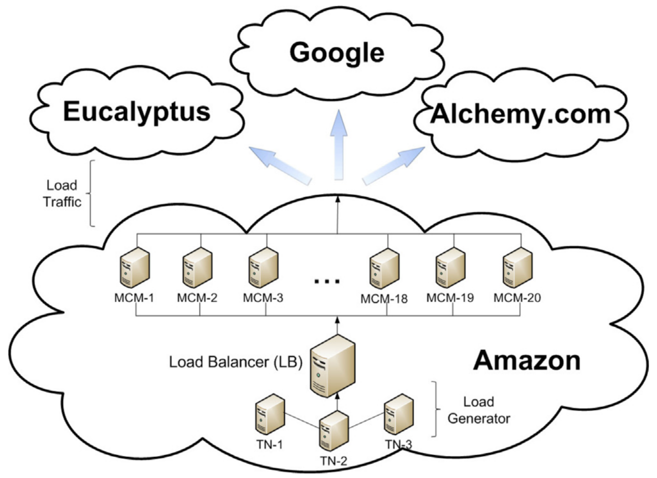


Fig. 10. Load test setup for the MCM.

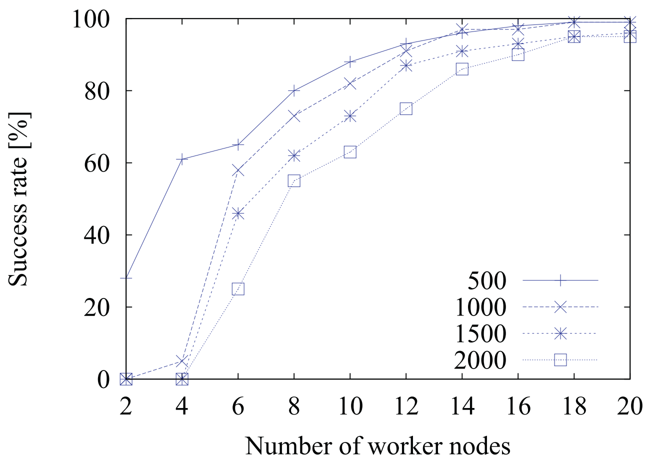


Fig. 11. Success rate of concurrent requests over multiple server nodes.

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 consists of three parts, the server/client configuration part, in which the machines' information is defined. The load part that contains the information related with the mean inter-arrival time between new clients and the phase duration. Here, the number of concurrent users is defined. For instance, for generating a load of three hundred users in 1 s a mean of 0.0033 was used (Tsung primary node divided the load in equal parts among the available Tsung nodes). And finally, the session' part, in which the testing scenario is configured and consists of describing the clients' request (captured using Tsung-recorder).

A single client request consists of simulating the hybrid mobile cloud application described in Section 4.2. The launch/terminate time of an Amazon instance is simulated by connecting to a large capacity running instance that performs the sensor processing. This time is simulated as launching a high number of instances incurs in large utility costs (not feasible for a single user). The time is the average calculated from a set of individual samples times (launching/terminating) that were taken during the day for a small instance size. This simulation does not affect the overall results, since the communication with the service is established and the transaction is performed over the cloud resources. Same occurs with Alchemy.com service, which is constrained by a free request quota. This issue was solved by taking an estimation about the time that it takes for the service to process a request. GCM and other services did not present any problem. In the case of GCM, messages were routed to a single device.

Load traffic is simulated by n concurrent threads, where n varied between ≈ 150 and 650 per Tsung node (TN), making 500–2000 concurrent requests on the load balancer, thus, simulating a large number of concurrent 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 was increased from 2 to 20. Initial amount of nodes was 2 and 2 nodes were added each time the setup needed to scale. Servers running on Amazon EC2 infrastructure were using EC2 m1.small instances. A small instance has 1.8 GB of memory and up to 10 GB of storage. One EC2 computational instance is equivalent to a CPU capacity of 2.66 GHz Intelˆ® Xeon^{TM}processor (CPU capacity of an EC2 compute unit do change in time). Servers were running on 64 bit Linux platforms (Ubuntu). Finally, 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, the aim of the experiment is to measure (1) how the access policies of the framework are enhanced by scaling the infrastructure horizontally and (2) 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. Similarly, the success rate indicates the number of requests from all performed requests that have succeeded. The results of the experiments are shown in Fig. 11. From the diagram it can be observed that the success rate follows a logistic function, with the number of nodes. The performance of eight nodes drops to ≈ 75% after receiving 1500 concurrent requests, however, 18 nodes can handle this load with almost 100% success rate. It can also be seen that with current test architecture adding more worker nodes does not show any visible improvement in the performance after 18 nodes in contrast when the setup was composed of 2, 4 and 6 nodes.

To sum it up, with current MCM implementation, pair nodes deployment may handle around 100–150 concurrent requests with almost 100% success rate. An addition of two nodes adds roughly the capacity of handling another 100 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.

6. Conclusions and future research directions

Mobile and cloud computing are converging as the leading technologies that are fostering the change to the post-pc era. Mobile devices are looking towards cloud-aware techniques that allow to bind transparently cloud resources and mobile applications. In this context, hybrid cloud integration based on service composition is a prominent approach required in a mobile cloud application, in order to alliviate the mobile resources from unneccessary data transfer. Moreover, a multi-cloud operation based delegation may empower the device with innovative services based on business collaboration.

We tried to address in this paper the challenges of delegating a mobile task to multiple clouds and configuring its deployment aspects for execution by encapsulating multiple Web APIs in a common operation level. We proposed MCM as an intermediary layer in the communication for handling the asynchronous delegation of a multi-cloud operation, reducing the complexity of working with distributed hybrid cloud services, decreasing the offloading times to the cloud from the handset and fostering a scalable and flexible self-service approach based on composition that enables to reuse, to maintain and to automate cloud services.

The architecture of the MCM is explained in detail along with its composition approach through the study of a mobile cloud application that uses cloud services from Amazon, Alchemy.com, Eucalyptus and Google. To demonstrate the horizontal scaling of the MCM, a scalability analysis is presented. The results show that MCM can handle reasonable loads with significant ease. While the prototype is working fine with the traditional web technologies like the HTTP and servlets, our future research will consist in extending the architecture to better suite the cellular network by re-implementating the middleware using Erlang Armstrong et al. (2013) since it is more suitable for managing high concurrency and for deploying components on the fly. Erlang also enables adding and replacing code, while the system is running. Hence, the development and deployment of new MCM functionality can be done without restarting the entire system.

Acknowledgments

This research is supported by the Tiger University Program of the Estonian Information Technology Foundation, the European Regional Development Fund through the EXCS, Estonian Science Foundation grant ETF9287, Target Funding theme SF0180008s12 and the European Social Fund for Doctoral Studies and Internationalisation Programme DoRa. The authors also thank the anonymous reviewers for their insightful comments.

Appendix A. Sensor classification algorithm using MapReduce

The integration of micromechanical sensor technologies within the smartphones makes it possible to enrich the usability experience of interacting with a mobile application by sensing the user's context and to understand certain human activities based on the tracking of the user's intention. Several prototypes and signal processing algorithms have been developed for human motion classification and recognition allowing reliable (more than 90% accurate) detection of basic movements (Preece et al., 2009).

The accelerometer simultaneously outputting tilt, is the most common sensor that is included within a modern mobile device as it allows tracking information that can be used for infering multiple human movements. Depending on the number of axes, it can gather the acceleration information from multiple directions. Generally a triaxial accelerometer is the most common in mobiles from vendors such as HTC, Samsung, Nokia. Therefore, acceleration can be sensed on three axes, 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.

Similarly, the gyroscope sensor is an actuator based on the principles of angular momentum conservation that is used for establishing position, navigation and orientation of the device, among others. It consists of three axes or freedom degrees (spinning, perpendicular and tilting) mounted in a rotor which are composed by two concentrically pivoted rings (inner and outer). The gyroscope is used within the mobile for enhancing techniques such as gesture recognition and face detection. Furthermore, the combination of accelerometer and gyroscope sensor data allows to increase the motion accuracy, and thus approaches such as video stabilization are implemented on the mobile.

We collected and synchronized in this paper, the measurements of the gyroscope and accelerometer along with the URL of the mobile web browser in order to identify repetitive patterns that can be classified as a specific human activity (e.g. walking, reading, etc.). Algorithm A.1 shows the parallelizable process of sensor analysis. We used MapReduce as the sensor information can be collected daily from the mobile and uploaded to the cloud, thus, creating a big repository of analyzable data that requires resource-intensive processing.

Algorithm A.1. Sensor Processing with MapReduce

- Map **Require:** CSV file
  - map function parameter is a < *key, value* > pair, where:
    - * *key* – line number
    - * *value* – line content
  - *value* is split into several variables (gyroscope's and accelerometer's values, timestamp, url)
  - gyroscope's values are examined, they have to be smaller than 0.05 (which means minimal rotation)
  - accelerometer's values are compared to the fixed threshold (to indicate that the user is holding the phone in his hand)
  - if the sensors' values are in range, the map will emit a < *key, value* > pair, where:
    - * *key* – timestamp in seconds
    - * *value* – url
- Reduce **Require:** *timestamp, list < url >*
  - timestamp consists of the relative time in which the measurement was taken
  - count the elements in the list and emit a < *key,value* > pair, where:
    - * *key* – time range
    - * *value* – url
  - sort the list in descending order.

References

Satyanarayanan, M., Bahl, P., Caceres, R., Davies, N., 2009. The case for vm-based cloudlets in mobile computing. IEEE Pervasive Computing 8 (4), 14–23.

Flores, H., Srirama, S.N., Paniagua, C., 2011. 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, pp. 87–94.

Flores, H., Srirama, S.N., 2013. Adaptive code offloading for mobile cloud applications: exploiting fuzzy sets and evidence-based learning. In: Proceeding of the 4th ACM Workshop on Mobile Cloud Computing and Services, pp. 9–16.

Flores, H., Srirama, S., 2012. Dynamic configuration of mobile cloud middleware based on traffic load. In: 2012 IEEE 9th International Conference on Mobile Adhoc and Sensor Systems (MASS), pp. 475–476, http://dx.doi.org/10.1109/MASS.2012.6502552.

Flores, H., Srirama, S., Paniagua, C., 2012. Towards mobile cloud applications: Offloading resource-intensive tasks to hybrid clouds. International Journal of Pervasive Computing and Communications 8 (4), 344–367.

Zeng, L., Benatallah, B., Ngu, A.H., Dumas, M., Kalagnanam, J., Chang, H., 2004. Qos-aware middleware for web services composition. IEEE Transactions on Software Engineering 30 (5), 311–327.

Onetti, A., Capobianco, F., 2005. Open source and business model innovation. The funambol case. In: International Conference on OS Systems Genova, 11–15th July, pp. 224–227.

Flores, H., Srirama, S.N., 2013. Mobile cloud messaging supported by xmpp primitives. In: Proceeding of the 4th ACM Workshop on Mobile Cloud Computing and Services, ACM, pp. 17–24.

Balan, R., Flinn, J., Satyanarayanan, M., Sinnamohideen, S., Yang, H.-I., 2002. The case for cyber foraging. In: Proceedings of the 10th Workshop on ACM SIGOPS European Workshop, ACM, pp. 87–92.

Gu, X., Nahrstedt, K., Messer, A., Greenberg, I., Milojicic, D., 2004. Adaptive offloading for pervasive computing. IEEE Pervasive Computing 3 (3), 66–73.

Li, Z., Wang, C., Xu, R., 2001. Computation offloading to save energy on handheld devices: a partition scheme. Proceedings of the 2001 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, ACM, 238–246.

Kumar, K., Lu, Y., 2010. Cloud computing for mobile users: can offloading computation save energy? Computer 43 (4), 51–56.

Cuervo, E., Balasubramanian, A., Cho, D., Wolman, A., Saroiu, S., Chandra, R., Bahl, P., 2010. Maui: making smartphones last longer with code offload. In: Proceedings of the 8th International Conference on Mobile Systems, Applications and Services, ACM, pp. 49–62.

Chun, B., Ihm, S., Maniatis, P., Naik, M., Patti, A., 2011. Clonecloud: elastic execution between mobile device and cloud. In: Proceedings of the Sixth Conference on Computer Systems, pp. 301–314.

Paniagua, C., Flores, H., Srirama, S.N., 2012. Mobile sensor data classification for human activity recognition using mapreduce on cloud. Procedia Computer Science 10, 585–592.

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

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

Google Inc., 2013a. AC2DM. http://code.google.com/android/c2dm/index.html

Google Inc, 2013b. GCM – Google Cloud Messaging for Android. http://developer.android.com/guide/google/gcm/index.html

Apple Inc, 2013a. APNS. http://developer.apple.com/library/ios/

Microsoft Inc, 2013. MPNS. http://msdn.microsoft.com/en-us/library/

typica, 2013. typica – A Java Client Library for a Variety of Amazon Web Services. http://code.google.com/p/typica/

jets3t, 2013. jetS3t – An Open Source Java Toolkit for Amazon S3 and CloudFront. http://jets3t.s3.amazonaws.com/toolkit/guide.html

Google, Inc., 2013c. Google Data Protocol http://code.google.com/apis/gdata/

jclouds, 2013. jclouds – Multi cloud Library. http://code.google.com/p/jclouds/

Foundation, T.A.S., 2013. Apache Libcloud a Unified Interface to the Cloud. http://libcloud.apache.org/

dasein, 2013. Project, dasein.org – The Dasein Cloud API. http://dasein-cloud.sourceforge.net/

Delta Cloud, 2013. Delta Cloud – Many Clouds. One API. No problem. http://incubator.apache.org/deltacloud/

google-gson, 2013. A Java Library to Convert JSON to Java Objects and Vice-versa. http://code.google.com/p/google-gson/

Hickey, R., 2008. The clojure programming language. In: Proceedings of the 2008 Symposium on Dynamic Languages, ACM, p. 1.

ejabberd, 2013. The Erlang Jabber/XMPP daemon. http://www.ejabberd.im/

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

T. E. Foundation, 2013. Gef (Graphical Editing Framework). http://www.eclipse.org/gef/

T. E. Foundation, 2013a. Eclipse Modeling Framework Project (emf) http://www.eclipse.org/modeling/emf/

LightCrawler, 2013. Open Source Crawler for Java. http://code.google.com/p/lightcrawler/

Comer, D., 1995. Internetworking with TCP/IP, vol. I, Principles, Protocols, and Architecture, vol.3. Prentice Hall Englewood Cliffs, NJ.

HAProxy, 2013. The Reliable, High Performance TCP/HTTP Load Balancer http://haproxy.1wt.eu/

Tsung, 2013. A Distributed Load Testing Tool. http://tsung.erlang-projects.org/

Armstrong, J., Virding, R., Wikstrom, C., Williams, M., 2013. Concurrent Programming in Erlang.

Preece, S., Goulermas, J., Kenney, L., Howard, D., Meijer, K., Crompton, R., 2009. Activity identification using body-mounted sensors-a review of classification techniques. Physiological Measurement 30 (4), R1.

Huber Flores is currently a PhD student at the Faculty of Mathematics and Computer Science, University of Tartu. He received his B.A. in Computer Science Engineering from the University of San Carlos of Guatemala, Guatemala and his M.Sc. in Software Embedded Systems Engineering from a combined program between the University of Tartu and Tallinn University of Technology, Estonia. His major research interests include mobile middleware architectures, high performance parallel computing on cloud and mobile cloud computing.

Satish Narayana Srirama is an Associate Professor and Head of the Mobile Cloud Lab at the Institute of Computer Science, University of Tartu. He received his PhD in Computer Science from RWTH Aachen University, Germany, in 2008. His current research focuses on mobile web services, cloud computing, mobile cloud, scientific computing on the cloud and mobile community support.