MECCA: Mobile, Efficient Cloud Computing Workload Adoption Framework using Scheduler Customization and Workload Migration Decisions

by

Kakadia Dharmesh, Prasad Saripalli, Vasudeva Varma

in

The 14th ACM International symposium on Mobile Ad Hoc Networking and Computing

Bangalore, India

Report No: IIIT/TR/2013/-1

Centre for Search and Information Extraction Lab
International Institute of Information Technology
Hyderabad - 500 032, INDIA
July 2013
MECCA: Mobile, Efficient Cloud Computing Workload Adoption Framework using Scheduler Customization and Workload Migration Decisions

Dharmesh Kakadia
International Institute of Information Technology, Hyderabad, India
dharmesh.kakadia@research.iiti.ac.in

Prasad Saripalli
IBM Cloud Center of Excellence
prasadsar@in.ibm.com

Vasudeva Varma
International Institute of Information Technology, Hyderabad, India
vv@iiit.ac.in

ABSTRACT
The availability of increasingly richer applications is providing surprisingly wide range of functionalities and new use cases on mobile devices. Even though mobile devices are becoming increasingly more powerful, the resource utilization of richer application can overwhelm resources on these devices. At the same time, ubiquitous connectivity of mobile devices also opens up the possibility of leveraging cloud resources. Seamless and flexible path to mobile cloud computing requires recognizing opportunities where the application execution on cloud instead of mobile device. In this paper we propose a cloud aware scheduler for application offloading from mobile devices to clouds. We used learning based algorithm for predicting the gain attainable using performance monitoring and high level features. We evaluated prototype of our system on various workloads and under various conditions.

Categories and Subject Descriptors
C.2.4 [Distributed Systems]: Client/Server

Keywords
Mobile Cloud Computing, Profiling, Cloud Workload Migration, Scheduling

1. INTRODUCTION
The exponential growth of Mobile Cloud Computing (MCC) is underpinned by a triad of challenging requirements around three main groups of ubiquity, trust, and energy efficiency. Large variation in platforms, variations in network technologies and quality as well as the devices causes horizontal heterogeneity and complicates MCC more than cloud computing [12]. Ubiquity requires seamless connectivity, context awareness and data integrity. Energy efficiency encompasses remote execution of mobile workflows, latency time reduction, fidelity adaptation, mobile mashup application and resource aware Algorithms.

We propose a Mobile OS scheduler customization and workload migration decision framework for MCC, which can be used to configure and administer the MCC systems dynamically during runtime, such that a majority of the above requirements can be met in an adaptive fashion. This framework is designed and implemented along four modules:

1. Resource usage profiling/fingerprinting
2. Job completion and resource usage prediction
3. Cloud aware scheduler for opportunistic offloading of tasks
4. Dynamic modeling of gain

For job completion and resource usage prediction using learning based algorithms, we use the MADMAC framework we had developed earlier [13]. The idea is to collect data when the app runs natively and under different circumstances, and using these data to predict the resource requirement and then predict the completion time. Multi-attribute Decision Making (MADM) has been shown to provide a rational basis to aid decision making in such scenarios. We present a MADMAC framework for cloud adoption, consisting of 3 Decision Areas (DA) referred to as the MCC requirements triad: trust, ubiquity and energy efficiency as above. It requires definition of attributes, alternatives and attribute weights, to construct a decision matrix and arrive at a relative ranking to identify the optimal alternative. Decision Support System (DSS) presented showing algorithms derived from MADMAC can compute and optimize Cloud Adoption (CA) decisions separately for the three stages, where the attributes differently influence CA decisions.

In the context of mobile clouds, scheduling algorithms’ function is to dynamically allocate threads, processes or data flows to particular nodes in the MCC architecture, which consist of mobile devices and cloud servers (VMs), given access to the state of system resources such as the processor time and communications bandwidth. Goal here is to migrate resource intensive processes, threads or data flows from the mobile device (where resource availability typically is limited) to the cloud servers, to achieve a target quality of service. The need for a scheduling algorithm arises from the
requirement for deciding which processes and workflows to migrate, when and how to merge the remotely executed processes back with the mobile-based workflow. For example, a graphics intensive or media intensive application running on the mobile device could be offloaded to a remote server on the cloud, and the results merged back into the mobile workflow. The scheduler makes such runtime decision based mainly on key resources such as the throughput which can be measured in terms of the total number of processes that complete their execution per time unit. Similarly, latency is typically measured as the total turnaround time between submission of a process and its completion. Also important to usability (user experience) is the response time, which is the amount of time taken between the submission of a request and first response is produced on the mobile device.

A modified Wide-band Delphi method is proposed for assessing the relative weights for each attribute, by workload. Relative ranks are calculated using these weights, and the Simple Additive Weighting (SAW) method is used to generate value functions for all the alternatives, and rank the alternatives by their value to finally choose the best alternative. The above algorithms are implemented in a OpenStack based MCC framework proof of concept, using Android as the OS on the mobile device and KVM Linux as the OS for the remote server cloud. For Resource usage profiling, we use established techniques in non-intrusive and scalable way.

The contributions of our work may be summarized as follows:

- A novel scheduling algorithm is proposed for MCC which optimizes on the Gain value of any cloud resource to augment the resources for mobile applications.
- We establish its inclusion in principle into a more elaborate multi-attribute decision framework (MADMAC) for optimizing across multiple, sometimes conflicting criteria.
- We demonstrate the use of learning based algorithm with features which includes not only the system resource utilization and performance related features but also higher level features.
- We proposed an algorithm for offloading decision based on the expected gain while migrating to the cloud.
- We built a prototype of the proposed system and evaluated its performance under various scenarios.

The rest of the paper is organized as follows. Section 2 briefly summarizes the related efforts in mobile cloud computing. Section 3 explains our system model and various components and interactions. In section 4, we detail our proposed solution and implementation of our prototype. Our experimental setup and results are described in 5. We conclude the paper in section 6.

2. RELATED WORK

Mobile cloud computing has recently received considerable attention by researchers mainly from academia, but industry-interest is also growing. This has resulted in various efforts by multiple research groups in the area.

Bahls et al. [5] provides an excellent overview of various suggested techniques for mobile cloud environment and unanswered challenges. They predict that mobile applications will become more context aware and personalized, but only envision traditional use cases of social media and content sharing etc. We believe that mobile device have opportunity to provide extremely personal experience if not only the apps, but device operating system also adapts the user. Kovachev D. et al. [7] provides an excellent survey on advances in mobile cloud computing and indicates alternative models of execution for mobile applications. Our approach is similar to the Augmented Execution Model mentioned by them, which uses cloned replica of VM on cloud.

ClonesCloud [6] provides a framework and runtime for offloading part of the application execution on cloud. They use thread based application partitioning for elastic execution between mobile and cloud. Internet Suspend Resume (ISR) [8] project uses execution and customized state on distributed storage. They used layered virtual machine technology for improving efficiency independent of hardware.

Wanghong Yuan et al. [15] proposed a soft real time scheduler for improving multimedia system on mobile devices. They mainly focus on codec for multimedia applications and focus on soft performance guarantees. Multiple research efforts have focused on special scheduling techniques to reduce battery usage and many solutions have emerged in this space. Aki Saarinen et al. designed SmartDiet [11] for offloading mobile applications to cloud for energy savings using execution traces.

Ayaj et al. [14] suggested a logging-based method to achieve VM migration to cloud. They used cryptographic hashing techniques to identify the divergent VM state and use compression based techniques to transfer and replay the state on cloud. The method proposed by them helps in reducing network transfer required, they do not address how to identify migration opportunities.

MADMAC system [13] has defined a framework to evaluate multi-attribute decision making for cloud adoption in various areas. They provide a framework for evaluate various attributes and alternatives for decision making. While they have not applied the framework for mobile cloud, the framework is general and extendable.

3. SYSTEM MODEL

Figure 1 shows our system model and interaction among the components for our system. Scheduler and profiler runs on mobile device and leverages cloud resources through cloud API. One of the goals of our system was to be transparent to the end user and developer. Thus, both scheduler and profiler run as part of mobile operating system and are transparent. We explain main components of the system in following sections.

3.1 Mobile Operating System

We chose Android as operating system for mobile device. Android is a Linux based operating system for mobile and tablet devices. Android is currently the leading open source operating system (with more than 75% market-share). Android kernel was forked from the linux kernel and runs each process as a separate user, isolating applications from other applications. This isolation makes it easier to profile and migrate applications to cloud.

3.2 Profiler
The profiler module is responsible for collecting all the statistics (described in based on which the offloading decision is made. It not only collects performance metrics for all user applications, but also gathers higher level features. We used combination of tools like perf and native api for sensors to get the required data which is then sent to the scheduler. Scheduler also controls the overhead of profiling by limiting the profiler invocations at different rate for different features.

### 3.3 Scheduler

Scheduler is core part of the system and responsible for offloading decisions. Mobile cloud aware scheduler has additional responsibility of including cloud parameters (cost, network connectivity etc.) in scheduling decision. We modified the android scheduler to include these parameters while making offloading decision.

### 3.4 Cloud API

Cloud API provides the functionality of interacting with the cloud infrastructure. Ideally, we would like to use standardized and provider-independent API so that the user can leverage resources from best possible (nearest, cheapest) cloud. While there are efforts in this direction [2], we still do not have the all-agreed cloud standards. We used OpenStack [4] as our cloud infrastructure management OS and used widely used EC2 [1] APIs which are supported by multiple cloud providers. We used Infrastructure as a Service (IaaS) model for using cloud resources as they provide the most flexibility.

We leverage the ability to run custom images for virtual machines by cloud providers and used custom image, with required softwares pre-installed.

### 4. PROPOSED SOLUTION

In this section, we describe our proposed solution for offloading decision and execution of migration to cloud.

#### 4.1 Profiling and Monitoring

We used following features to model the application and the user interactions with application.

##### 4.1.1 Dynamic Features

- High level features: this comprises of features that are concerned to user. It includes battery status, date and time, user location (moving/stable), etc.
- Application features: this captures application usage habits including frequency of usage of the application, stretch of usage, use of local and remote data, etc.
- Network Status: network condition between cloud and mobile device. This includes multiple parameters including bandwidth, latency and stability.
- Resource usage by other applications running on device: This is a combined vector of all individual applications. This is to capture the perceived use of applications and resources.

#### 4.1.2 Non-Dynamic Features

- Device Configuration: this captures all the hardware and software configuration of the device. This includes cpu frequency, cpu power steps, operating frequency, etc.
- Cloud Configuration: This captures characteristics of the cloud provider. This includes cost of running a VM for various hardware/software configuration.

### 4.2 Gain Model and Offload decision

The decision to leverage cloud depends on the benefits that can be achieved using cloud. To estimate the gain, we use following model.

\[
Gain = \sum \left( \frac{w_i \times (m_i - c_i)/m_i}{\sum w_i} \right)
\]

(1)

where \(w_i\) is the weight of \(i\) the feature gain and are normalized to unity. \(m_i\) and \(c_i\) are costs of running the application on mobile device and cloud respectively and range from 0 to 1. Costs \(m_i\) and \(c_i\) are for each feature described earlier. For example, cost for battery might higher when running on mobile \((m_i)\) compared to running on cloud \((c_i)\). On the other hand, cost of network usage can be higher while application runs on cloud \((c_i)\) than running on mobile device \((m_i)\) for highly interactive applications. The reason for using \(w_i\) in above model is to allow giving different importance to different features while estimating the gain.

#### Algorithm 1 Offloading Decision

```plaintext
for each application i do
    Update the model with current metrics
    predict \(c_i\) values for all the features using model
    \(Gain_p\) according to Eq. 1 predicted values of \(c_i\) and measured values of \(m_i\)
    if \(Gain_p \geq \text{significance\_threshold}\) then
        Execute the \(p\) remotely on cloud.
    else
        continue executing \(p\) locally.
    end if
end for
```

Offloading algorithm is described in Algorithm 1. For each application, the model is updated in online setting with current metrics for various features and new values for \(c_i\)s are predicted by the scheduler prediction module. The \(Gain_p\) is calculated with new values of \(c_i\) and measured values of \(m_i\) and compared against \(\text{significance\_threshold}\). If the \(Gain_p\) is sufficiently high, the process is migrated to cloud and user
interactions continue remotely. Otherwise next application is tried for possible optimization. The choice of parameter \textit{significance threshold} controls aggressiveness of offloading decisions. Very high values of \textit{significance threshold} indicate that algorithm will run most of the applications locally while lower values will allow applications to be offloaded to cloud even if gain achieved is not very high. We suggest its value between 0.25 to 0.65.

4.3 Migration

We ran applications in isolated process in android environment which simplifies the migration of application on cloud. We would like to note that focus of our work is not to improve migration of applications to cloud and relied on the simple standard techniques for migrating user application from the mobile device to cloud. We used remote frame buffer protocol [10] to continue the user interaction after migration to cloud. We used VNC viewer on the mobile side and ran VNC server on the cloud side. The algorithm runs at each epoch.

5. EXPERIMENTS AND EVALUATION

In this section we describe our prototype implementation and experiments to validate our approach. Figure 2 describes our implementation. We used Vowpal Wabbit (vw) [9] for learning weights ($w_i$). Vowpal Wabbit is a fast machine learning package. We considered the problem as a regression problem and used online learning with square loss function. Because of the online learning we can also control the overhead of profiling by varying profiler invocation frequency. We used traffic control utility (tc) [3] for emulating performance with various network scenarios. We only have one parameter for user (\textit{significance threshold}) which makes it easy to use. This results were noted at \textit{significance threshold} set to 0.35. Table 5 describes our experimental setup. We evaluated our system for representative workloads from 4 different classes shown in Table 5.

As described theses workloads cover range of applications various levels of interactivity, resource usage and runtime.

We start with uniform weights and each time scheduler decides to offload to the cloud, we asked user to approve the decision and showed the gain and other related information.

The users approval or disapproval is fed back into the learning algorithm as training data and algorithm uses them to modify the importance of features ($w_i$). As this process continues the scheduler becomes more personalized about user preferences for various features. For example, some users might prefer to get results fast by running on cloud while others might be more concerned about the monetary cost of using cloud. So the personalized scheduler becomes aware of context and user preferences.

Table 1: Experimental Setup

<table>
<thead>
<tr>
<th>Parameter</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cloud Operating System</td>
<td>Ubuntu 12.04(kernel 3.2)</td>
</tr>
<tr>
<td>Cloud VM configuration</td>
<td>4 GB, 2.66GHz</td>
</tr>
<tr>
<td>Device Operating System</td>
<td>Android 4.2</td>
</tr>
<tr>
<td>Device Configuration</td>
<td>1GB, 1.5 GHz</td>
</tr>
</tbody>
</table>

Table 2: Workloads

<table>
<thead>
<tr>
<th>Workload</th>
<th>Description</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel</td>
<td>kernel download + build</td>
<td>long + resource intensive</td>
</tr>
<tr>
<td>GIMP</td>
<td>Image editing + applying image filters</td>
<td>interactive + little intensive</td>
</tr>
<tr>
<td>Video conversion</td>
<td>download &amp; convert a (500MB) video</td>
<td>short + resource intensive</td>
</tr>
<tr>
<td>Browser</td>
<td>browsing 5 sites</td>
<td>interactive</td>
</tr>
</tbody>
</table>

Figure 3: Decision Time for various Applications under various network scenarios

Figure 3 shows the output of decision and time taken for making it for various workloads under various network scenarios. We emulated three classes of bandwidth cable(0.375/6), Digital Subscriber Line (DSK 0.75/3) and Evolution-Data Optimized (EVDO 1.2/3.8) where numbers in bracket specifies the uplink and downlink bandwidths in Megabits per sec. This is plotted against the the % of time application run when the decision was made about offloading. The decision of offloading is shown on top of bar. For kernel workload algorithm was able to decide early (within 2%) that it is suitable for cloud offloading. Similarly for browser workload also, it was able to understand the unsuitability of offloading early-on (within 5%). For GIMP workload, the algo-
Algorithm spent time in understanding the user interaction and decided not to offload in case of poor network condition but decided to offload if network condition is good. This is important as poor network condition will result in poor user experience for interactive workloads. For video conversion workload the algorithm decided to offload even if the network condition is not very good. This is again expected as the video conversion workload is non-interactive and intensive, the algorithm rightly decided to offload.

Figure 4 shows the overhead of our approach for various number of applications. We measured overhead as the percentage increase in the resource utilization with and without running our system.

![Figure 4: Overhead of running our system for increasing number of applications](image)

6. CONCLUSION AND FUTURE WORK

We proposed a scheduling algorithm which tries to make use of cloud resource to augment the resources for mobile applications. We used learning based algorithm with features which includes not only the system resource utilization and performance related features but also higher level features. We proposed an algorithm for offload decision based on the expected gain while migrating to the cloud. We built a prototype of the proposed system and evaluated its performance under various scenarios. We found that our approach is able to identify high level workload characteristics and make appropriate decision about running application locally or on cloud.

In future, we would like to evaluate our approach on even more workloads. We would also like to expand our approach with more features and decision making where mobile user has access to multiple clouds with different characteristics.

Acknowledgments

The authors would like to thank the anonymous reviewers for their valuable feedback and suggestions for improvements which made final version of paper much better. They would also like to thank students and faculty of Search and Information Extraction Lab (SIEL) for their extensive support throughout this work.

7. REFERENCES