International Journal On Engineering Technology and Sciences – IJETS™
ISSN(P): 2349-3968, ISSN (O): 2349-3976
Volume III, Issue IV, April- 2016

COMD: Computation Offloading for Mobile Devices

***Rashmila GV*[1], *Navya EK*[2]**
Dept. Of Computer Science
Malabar Institute of Technology,
Anjarakandy.
rashmilagv@gmail.com[1],*navya.06rimaan@gmail.com*

Abstract

Potential for boosting the performance of mobile devices by offloading computation-intensive parts of mobile applications is becoming famous. Computation offloading to multiple devices is an effective method to reduce energy consumption and to enhance performance for mobile applications. Android provides mechanisms for creating mobile applications but it lacks a native scheduling system for determining where the code should be executed. In this paper, I present the design and implementation of the COMD system which bridges this gap by providing computation offloading as a service to mobile devices. COMD efficiently manages cloud resources for offloading the requests to both to improve offloading performance seen by mobile devices and to reduce the cost per request to the provider. COMD also effectively allocates and schedules offloading requests to resolve the contention for resources in cloud.

1. Introduction

Task offloading from smart phones to the cloud is a promising strategy to enhance the capability of computing of smart phones and prolong their battery life. However, task offloading introduces a communication cost for those devices. Therefore, the consideration of the communication cost is crucial for the effectiveness of task offloading. To make task offloading beneficial, one of the challenges is to estimate. The idea of offloading computation from mobile devices to remote computing resources is to improve performance and to reduce energy consumption. A key challenge in computation offloading lies in the mismatch between how individual mobile devices demand and access computing resources and how cloud providers offer them. Offloading requests from a mobile device require quick response and may not be very frequent. Therefore, the ideal computing resources suitable for computation offloading should be immediately available upon request and be quickly released after execution. In contrast, cloud computing resources have long setup time and are leased for long time quanta. For example, it takes about 27 seconds to start an Amazon EC2 VM instance. The time quantum for leasing an EC2 VM instance is one hour. If an instance is used for less than an hour, the user must still pay for one-hour usage. This mismatch can thus hamper offloading performance and/or incur high monetary cost. In this paper I propose COMD, a system that bridges the above-discussed gaps by providing computation offloading as a service. The key premise of COMD is that an intermediate service between a commercial cloud provider and mobile devices can make the properties of underlying computing and communication resources transparent to mobile devices and can reorganize these resources in a cost-effective manner to satisfy offloading demands from mobile devices. The COMD system receives mobile user computation offload demands and allocates them to a shared set of compute resources that it dynamically acquires (through leases) from a commercial cloud service provider. The goal of COMD is to provide the benefit of computation offloading to mobile devices while at the same time minimizing the compute resource leasing cost.

The goal in this paper is to develop a design for COMD, implement it and evaluate its performance. At the heart of COMD are two types of decisions: 1) whether a mobile device should offload a particular computation to COMD and 2) how COMD should manage the acquisition of compute resources from the commercial cloud provider.

The rest of the paper is organized as follows. Section 2 presents some background material and presents the problem statement and optimization formulation. Section 3 presents an overview of COMD's architecture. Section 4 presents the design details of the COMD system. Section 5 presents the implementation details and Section 6 presents the conclusion and future work.

2. Background and Problem Statement

2.1. Background

2.1.1 Computation Offloading

A basic computation-offloading system is composed of a client component running on the mobile device and a server component running in the cloud. The client component has three major functions. First, it monitors and predicts the network performance of the mobile device. Second, it tracks and predicts the execution requirements of mobile applications in terms of input/output data requirements and execution time on both the mobile device and the cloud. Third, using this information the client component chooses some portions of the computation to execute in the cloud so that the total execution time is minimized. The server component executes these offloaded portions immediately after receiving them and returns the results back to the client component so that the application can be resumed on the mobile device.

2.1.2 Cloud Computation Resources

Cloud computation resources are usually provided in the form of virtual machine (VM) instances. To use a VM instance, a user installs an OS on the VM and starts it up, both incurring delay. VM instances are leased based on a time quanta. e.g., Amazon EC2 uses a one hour lease granularity. If a VM instance is used for less than the time quanta, the user must still pay for usage. A cloud provider typically provides various types of VM instances with different properties and prices.

2.2 Problem Statement

The basic idea of COMD is to achieve good offloading performance at low monetary cost by sharing cloud resources among mobile devices. Specifically, in this paper my goal is to minimize the usage cost of cloud resources under the constraint that the speedup of using COMD against local execution is larger than $1-\delta$ of the maximal speedup that it can achieve using the same cloud service where $\delta \in (0, 1)$. If O_k is offloaded to the ith VM instance, $I(i, k) = 1$. Otherwise, $I(i, k) = 0$. Similarly, if it is locally executed, $I_l(k) = 1$. Otherwise, $I_l(k) = 0$. The key symbols are summarized in Table 1.

M_i	the type of the i^{th} VM instance
T_i	the leasing periods of the i^{th} VM instance
$\psi(M, T)$	the pricing function for leasing a VM instance
O_k	the k^{th} computational task
$I(i, k)$	the indicator function for offloading task O_k to VM i
$I_l(k)$	the indicator function for executing task O_k locally
$L(O_k)$	the local execution time of the task O_k
$R_i(O_k)$	the response time of offloading the task O_k to VM i

International Journal On Engineering Technology and Sciences – IJETS™
ISSN(P): 2349-3968, ISSN (O): 2349-3976
Volume III, Issue IV, April- 2016

3. COMD System

Figure 1 provides a high-level overview of the COMD system. It consists of three components: a COMD Master running on a VM instance that manages cloud resources and exchanges information with mobile devices; a set of active COMD Servers each of which runs on a VM instance and executes offloaded tasks; and a COMD Client on each mobile device that monitors application execution and network connectivity and makes offloading decisions.

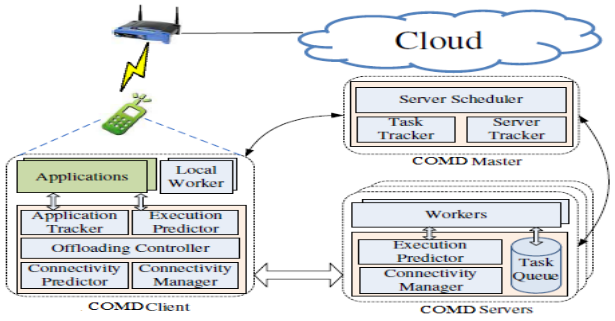

Figure 1: The architecture of the COMD system.

The COMD Master is the central component for cloud resource management. It periodically collects information of computation tasks from COMD Clients through task tracker and the workloads of COMD Servers through server tracker. Using this information, its server scheduler decides the number and type of active COMD Servers over time. Note that when a COMD Server is turned on/off, its corresponding VM instance is also turned on/off. An active COMD Server is responsible for executing offloaded computation tasks. Each COMD Server has one task queue and multiple workers the number of which equals to its number of CPU cores. Computation tasks are executed on a first-come-first-serve basis. The COMD Server also estimates and provides the workload information upon request by predicting the execution time of all tasks in the queue through the execution predictor. A COMD Client tracks all applications running on its mobile device, makes offloading decisions for them, and allocates tasks to COMD Servers. When a mobile application starts, an application tracker monitors the application execution and identifies its compute-intensive. When it reaches the entry point of such a task, the offloading controller obtains the computation speedup from the execution predictor and the communication delay from the connectivity predictor and decides if it should offload the task based on them. If it decides to offload, the COMD Client allocates the task to an active COMD Server. Finally the COMD Client offloads the task and waits to receive the result. If the COMD Client cannot obtain the results before a deadline, it executes the task on a local worker.

4. Design Details

4.1. Cloud Resource Management

The cloud resource management has two major mechanisms: how to select the type of VM instance and when to start and stop COMD Servers.

4.1.1Resource Selection

COMD strives to minimize the cost per offloading request under the constraint that the offloading speedup is large enough. Our goal is to achieve speedup of at least $1 - \delta$ of the maximally possible. Therefore, the resource selection algorithm selects the least cost VM instance whose CPU frequency is larger than $1 - \delta$ of the most powerful VM instance

4.1.2 Server Scheduling

Server scheduling is the key mechanism to balance the usage cost of VM instances and the offloading performance. Its basic operations are as follows: The COMD Master periodically collects the number of offloading requests and the workloads of COMD Servers (every 30 seconds in our implementation). When the workloads are too large, it turns on new COMD Servers. When a time quantum of a COMD Server is to expire, it turns off the COMD Server if the remaining COMD Servers are enough to handle the offloading requests.

4.2 Offloading Decision

The offloading controller uses the information from the connectivity and execution predictors to estimate the potential benefits of the offloading service. Ideally, if future connectivity and execution time can be accurately predicted immediately after the mobile application starts, the offloading controller can make the global optimal offloading decision. However, such global optimum is unavailable in reality. Instead, the offloading controller uses a greedy strategy to make the offloading decision. Every time an off loadable task is initiated, the offloading controller determines if it is beneficial to offload it. Because of the uncertainties inherent in the mobile environment, the offloading decision takes risk into consideration. In case a bad decision has been made, it will also adjust its strategy with new information available.

4.3. Task Allocation

When a COMD Client is to offload a task, it must decide which COMD Server should execute the task. I consider three heuristic methods. The first method is that the COMD Master maintains a global queue to directly accept offloading requests and allocates tasks when a COMD server has idle cores. Although it should have high server utilization, the network connecting the COMD Master may become a bottleneck. The second method is that the COMD Client queries the workloads of a set of COMD servers and randomly chooses one with low workload to allocate the new task. Although tasks are directly sent to COMD Servers in this method, it will cause huge control traffic. In addition, it will cause extra waiting time. The third method is that the COMD Master provides each COMD Client a set of active COMD Servers and informs it the average workloads of all COMD Servers. Each COMD Client randomly chooses a server among them to offload the task. This method has minimal control overhead. As the resource-management mechanism ensures that the workloads of COMD Servers are low, it should also have good performance. Thus, COMD uses the third method for task allocation.

5. System Implementation

I implemented the COMD Server on Android x86.To run COMD Servers on Amazon EC2; I use an Android-x86 AMI to create EC2 instances. Since Android-x86 is a 32- bit OS, three types of EC2 instances can be used for COMD Servers. Based on our resource selection algorithm, High-CPU On-Demand Medium instances are used to run COMD Servers. The COMD Master runs on an EC2 instance running Ubuntu 12.04. It uses the Amazon EC2 API tools to start and stop EC2 VM instances on which COMD Servers run. A COMD Client runs on an Android device equipped with both WiFi and 3G connections. It uses the Java reflection techniques to enable the offloading of computation tasks. I modified three existing Android applications to use COMD, including: FACEDETECT is a face detection application that uses APIs in the Android SDK. I collected a data set of pictures containing faces from Google Image. VOICERECOG is an Android port of the speech recognition program PocketSphinx .For simplicity of experiments, I also modified it to use audio files as input. DROIDFISH is an Android port of the chess engine Stockfish that allows users to set the strength of the AI logic

6. Conclusion and Future Work

In this paper, I proposed COMD, a system that provides computation offloading as a service to resolve the mismatch between how individual mobile devices demand computing resources and how cloud providers offer them. COMD solves two key challenges to achieve this goal, including how to manage and share the cloud resources and how to handle the variable connectivity in making offloading decision. I have implemented COMD and conducted an extensive evaluation. The experimental results show that COMD enables effective computation offloading at low cost.

There are some future directions to extend COMD. I will explore how to extend COMD to provide the computation offloading services in a manner that optimizes the energy consumption of mobile devices. It requires two major changes. First, the offloading controller should make the offloading decision based on energy consumption. It should delay computation offloading until the network connectivity is good. Second, the cloud resources can be used in a more efficient way. Instead of immediately executing each offloaded task, COMD should wait until enough tasks are aggregated. In addition, I will investigate the proper pricing model for COMD. There are several possible methods. Users pay monthly service fees and can use COMD as frequently as they want. Alternately, an offloaded task could be charged according to its execution time. It's even possible for mobile devices to bid for computation-offloading. When the number of offloading requests is small, COMD could charge a lower price to attract more requests and thereby avoid wasting cloud resources.

References

[1] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. MAUI: Making smartphones last longer with code offload. In MobiSys, 2010.

[2] B. Chun, S. Ihm, P. Maniatis, M. Naik, A. Patti. CloneCloud: Elastic Execution between Mobile Device and Cloud. In ACM Eu-roSys, 2011.

[3] H. Qian and D. Andresen. Jade: An Efficient Energy-aware Computation Offloading System with Heterogeneous Network In-terface Bonding for Ad-hoc Networked Mobile Devices. In Pro-ceedings of the 15th IEEE/ACIS International Conference on Soft-ware Engineering, Artificial Intelligence, Networking and Paral-lel/Distributed Computing (SNPD), 2014.

[4] H. Qian and D. Andresen. Extending Mobile Device's Battery Life by Offloading Computation to Cloud. In Proceedings of the 2nd ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft), 2015.

[5] H. Qian and D. Andresen. An Energy-saving Task Scheduler for Mobile Devices. In Proceedings of the 14th IEEE/ACIS Interna-tional Conference on Computer and Information Science (ICIS), 2015.

[6] H. Qian and D. Andresen. Emerald: Enhance Scientific Work-flow Performance with Computation Offloading to the Cloud. In Proceedings of the 14th IEEE/ACIS International Conference on Computer and Information Science (ICIS), 2015.

[7] Q. Chen, H. Qian et al. BAVC: Classifying Benign Atomicity Violations via Machine Learning. In Advanced Materials Research, Vols 765-767, pp. 1576-1580, Sep, 2013.

[8] F. Wei, S. Roy and S. Ou. Amandroid: A Precise and General Inter-component Data Flow Analysis Framework for Security Vet-ting of Android Apps. In proceedings of the 2014 ACM Conference on Computer and Communications Security. 2014.

[9] L. Peng, Y. Yang et al. Highly Accurate Video Object Identifi-cation Utilizing Hint Information. In proceedings of the International Conference on Computing, Networking and Communications (ICNC), 2014.