Understanding Performance Modeling for Modular Mobile-Cloud Applications

Ioana Giurgiu
Systems Group, Dept. of Computer Science, ETH Zurich
igiurgiu@inf.ethz.ch

ABSTRACT

Mobile devices are becoming the main entry points to the growing number of cloud applications and services. Unlike traditional approaches, we pursue a flexible architectural model where cloud hosted applications are distributed between mobile devices and the cloud in a bid to improve interaction performance. Given the increasing variety of mobile platforms or virtual instances, in this paper we approach the problem of estimating performance for such applications in two steps. First, we identify the factors that impact interaction response times, such as the application distribution schemes, workload sizes and intensities, or the resource variations of the mobile-cloud setup. Second, we attempt to find correlations between these factors and to understand how to build a unified and generic performance estimation model.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling Techniques

Keywords

Mobile cloud applications, performance models

1. INTRODUCTION

Mobile devices are becoming the main entry points to the growing number of cloud applications and services. The predominant architecture for offering such services to users are browser-based applications, where most, if not all, of the application software is hosted in the cloud. In such scenarios, performance depends only on the available bandwidth and connection latency between the mobile device and the cloud instance. To alleviate the network problem, we explore an alternative model for cloud applications, where the cloud instance dynamically migrates part of the application to the mobile device to improve user experience, by minimizing data transfers and overall interaction times. Our model uses *modularization* [7] to allow flexible distributed deployments of applications, from keeping only the user interface on the device to hosting the whole application locally.

ICPE'12, April 22–25, 2012, Boston, Massachusetts, USA

Given the increasing variety of mobile platforms and cloud instances, we cannot aasume to always have access to accurate measurements in all scenarios, by running the applications a-priori. Therefore, in this paper we address the following question: "*What is the best performance of an application, when distributed between a mobile device MD and a cloud virtual instance VI, with workload Y?*". The *mobile device* MD and *virtual instance* VI represent the setup to estimate the application's response time for, without actually running it. With the diversity of mobile devices, each with different resource capabilities, the application performance will vary accordingly. For an image processing application, one would experience higher response times on a Motorola Droid (i.e. 600MHz CPU) compared to an HTC Desire (i.e. 1GHz CPU), when using the same distribution scheme and cloud instance. Similar changes in response time are observed when different virtual instances are used.

Best performance is correlated with the application *distribution scheme* that achieves the lowest response time in the setup (MD, VI). In practice, it is hardly the case that the distribution with best response time for a specific (MD, VI) setup will be optimal for other setups, as well. For the image processing example, offloading less computational parts on the Motorola Droid results in better interaction times, because its CPU capability is lower. In addition, the type of *workload* a user inputs to such an application will impact performance dramatically. Imagine how the response time varies for a panorama application, where instead of submitting 3 images of 100kB each, a user sends 6 images of 500kB each. Both times spent in executing image processing algorithms and transferring data remotely become much higher.

To summarize, we identify relevant factors that impact the performance estimation of a modular mobile-cloud application in an (MD, VI) setup: (a) the application distribution scheme, (b) the workload size and intensity, and (c) the resource variations observed between (MD, VI) and *logged* setups for which the application was previously run. Given the variability of these factors, the problem becomes complex and requires one to understand what are the application demands for a specific resource and how they are impacted by workload types. In addition, one must find correlations between application demands and the resource variations of the (MD, VI) setup compared to logged setups, to understand which distribution scheme would indeed provide the best response time. We discuss the relevant factors and their correlations in Section 3, after introducing the current state of the art in Section 1.1 and modular mobile-cloud applications in Section 2. We conclude in Section 4.

1.1 State of the art

Performance modeling and estimation is studied especially for distributed and multi-tier applications. A preferred approach is based on queuing models of complex networked services. Stewart and Shen [9] predict throughput and mean response time based on performance profiles and M/G/1 queuing expressions. Urgaonkar et al. [10] describe a complex queuing network model for multi-tier applications. Their approach requires extensive calibration, but can be used for dynamic capacity provisioning, performance prediction, bottleneck identification and admission control. In [11], the authors look into performance modeling of virtualized resource allocation, based on probabilistic relationships between virtualized CPU allocation and application response time.

Another direction focuses on workload modeling. In Magpie [1], the authors exploit knowledge of application architecture to determine the resource demands of different transaction types. Stewart et al. [8] propose a model for performance prediction based on the nonstationarity character of transactions types, by relying solely on lightweight passive measurements. Rolia et al. [4] proposes a resource demand modeling approach, while others attempt to diagnose performance changes by comparing request flows from two executions [5], or to estimate performance for embedded systems based on discrete event simulations [3].

Some work has also been done in modeling and estimating performance in the context of mobile devices. In [6], the authors introduce a method based on linear regression and clustering to predict performance requirements of mobile devices tasks using hardware resource utilizations and input data. However, to the best of our knowledge all the existing performance models for mobile devices address only scenarios where applications are entirely run locally. Instead, we are tackling the performance estimation problem for distributed applications between mobile devices and cloud instances.

2. MOBILE-CLOUD APPLICATIONS

Mobile-cloud applications are cloud applications enabled to run on mobile platforms, by distributing their components between virtual instances and a mobile device. As with mobile applications, the user requires spontaneous and faster interactions for mobile-cloud applications, as well. Since applications are different in their resource demands, there is no unique distribution scheme that maximizes performance for all. Therefore, to add flexibility in how applications are distributed, we propose an architectural model that applies the modularity principle [7]. According to it, applications are organized as sets of processing modules that communicate with each other, each module encompassing a logical functionality or a set of highly cohesive functionalities.

Let us assume an application is composed of N modules, $\mathbb{M} = \{M_i | i = 1, 2, ..., N\}$, as in Figure 1a. M_1 represents the entry point in the application and the minimal code that needs to be installed on the mobile device that allows the user to start an interaction. Typically, M_1 implements the application user interface, while the remaining modules implement main logical functionalities. The communication between modules is modeled as a *directed acyclic graph* (DAG), where a module M_i issues requests to all the

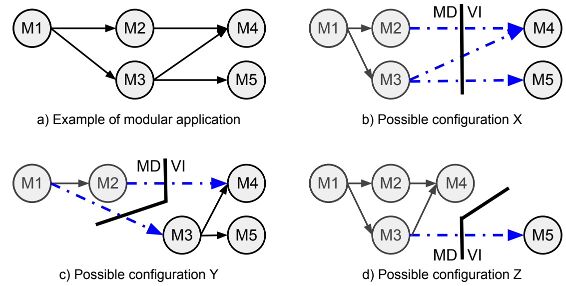

Figure 1: Abstract example of modular application and possible distribution partitionings between MD and VI.

modules M_j it logically depends on, $i \neq j$, until it reaches the last module M_N. In the simplest case, each such request is processed exactly once and the result is sent in reverse order until it reaches M_1, which then returns it to the user. More complex processing is possible when a request can visit a module *multiple* times. As an example, consider a keyword search which triggers queries on different tables in a database. In the sequential case, each request is issued once the processing of the previous request has finished. The more complex parallel case is not currently addressed.

Given the DAG representation of a modular application, we define a partitioning between the mobile device MD and the virtual instance VI as two non-overlapping sets of modules that cover all application modules, P_{MD} and P_{VI}, respectively. Therefore, $P_{MD} \cup P_{VI} = \mathbb{M}$ and $P_{MD} \cap P_{VI} = \emptyset$. Examples of possible partitionings are shown in Figures 1b–d. In [2] we have investigated how distribution partitionings impact application performance, especially when user inputs and number of requests per module change over time. This confirms the need to understand the performance modeling problem for mobile-cloud applications.

3. HOW TO MODEL PERFORMANCE OF MOBILE-CLOUD APPLICATIONS?

In the context of modular mobile-cloud applications, we define a *distribution setup* as a (MD_x, VI_y) pair, such that $MD_x \in \mathbb{MD} = \{MD_x | x = 1, 2, ..., X\}$ and $VI_y \in \mathbb{VI} = \{VI_y | y = 1, 2, ..., Y\}$, where $\mathbb{MD}$ and $\mathbb{VI}$ represent sets of diverse mobile devices and cloud instances, respectively. Our goal is to identify and model relevant application and infrastructure parameters, in order to accurately estimate the application's response time in a specific such setup.

First, we account for logged measurements of application executions in different setups (e.g. (MD_1, VI_2) = (HTC Desire, Amazon EC2-Small)) and distributions (e.g. $M_{1,2,3}$ on the mobile device and the remaining modules remote), to identify the application demands for the underlying resources. Second, we need to characterize how the new setup (e.g. (MD_2, VI_3) = (Motorola Droid, Amazon EC2-Large)) differs from the logged setups in terms of resources. Third, it is important to characterize workload size and intensity, and to understand how these factors can be correlated in a unified model.

3.1 Identifying application demands for underlying resources

To understand how resource demanding a modular application is, we represent it as a network of M queues Q_1, ...,

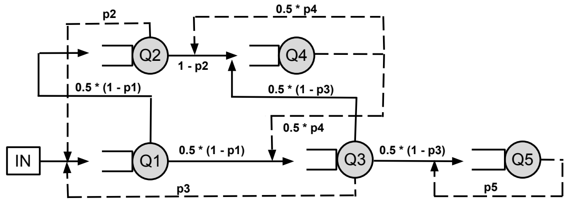

Figure 2: Modeling the example modular application using a network of queues.

Q_M. Each queue represents a specific application module and the underlying platform it runs on. When a request arrives at module M_i, it triggers one or more requests to modules M_j it depends on; recall the example of a keyword search that triggers multiple queries at different database tables. In our queueing model, we capture this by allowing a request to make multiple *visits* to a queue during its overall execution, and by introducing a transition from each queue to its predecessor. Figure 2 represents the queue model for the application in Figure 1a. After processing at queue Q_j, a request follows one of two paths based on the application communication model. Either it returns to one of the b queues from which it received the request, Q_i, with probability $\frac{p_j}{b}$, or it proceeds to one of the c queues to which it depends on, Q_k, with probability $\frac{1-p_i}{c}$. In practice, we cannot associate exact values to these probabilities, since application dataflows for different inputs or factors can be different. However, with enough collected measurements, we can compute aggregated medians of request visits per module and assign their values to the corresponding probabilities.

Let S_i denote the service time of a request at Q_i, $1 \leq i \leq M$, and W_i the waiting time from Q_i to one of the c queues it depends on. By using these basic queue properties and logged measurements from previous runs, we can model and identify the application demands for specific underlying resources. The service time at Q_i represents the execution time of module M_i, which is a measure of the CPU demand per module. The waiting time from Q_i is equivalent with the time required to transfer data to a subsequent queue, thus an expression of the network demand. By comparing the total service time with the total waiting time, we can understand how much of the total response time is spent performing computational tasks and transferring data, respectively.

Open problems. The queue network naturally models CPU and bandwidth resources. However, especially on mobile platforms, memory is a scarce resource that needs to be accounted for. One possible model extension is to assign to each application module a network of queues, where each queue corresponds to a specific resource. This way, by estimating the utilizations at the queues, we can understand how memory impacts application performance.

3.2 Application workload modeling

In mobile-cloud applications, the following observations about workload hold: (a) workload consists of *request-reply* interactions; (b) interactions have a limited number of *types* (e.g. browsing, online payment, printing for a ticket machine application); (c) interaction types influence resource demands; (d) interaction mix is *nonstationary*, meaning it changes over time. Nonstationary processes can be used to model workloads, but do not address the complementary problem of workload forecasting. However, as shown in [8] if accurate forecasts are available, they can be mapped to accurate performance predictions. Therefore, we make the following assumptions prior to formulating a workload model: (a) all previous application interactions are logged, (b) the modules CPU utilization and the bandwidth utilization between them are extracted from the measured execution and data transfer times.

To capture the application behavior for a specific workload, we observe a number of sequential interactions performed by a user over windows of fixed length T. Assuming that the application can process S interaction types, we denote by S_i the number of interactions of the i-th type, where $1 \leq i \leq S$. U_{r,M_j} is the average utilization of resource r at module M_j during the monitoring window. This representation is natural for CPU, because its utilization corresponds to execution times measured at module level. Instead, for network resources the average utilization is a property of all the incoming communication links to module M_j. Therefore, we define D_{i,M_j} as the average service time of interactions of type i at module M_j, while for network resources, we need to replace the *service* time with the summed *waiting* time on all incoming links to M_j. Based on the utilization law, for each monitoring window and resource r we obtain:

$$\sum_i S_i D_{i,M_j} = U_{r,M_j} T \tag{1}$$

In practice, it is infeasible to obtain accurate service or waiting times D_{i,M_j}. Thus, we consider the approximated costs of D_{i,M_j} for resource r (i.e. CPU or bandwidth) and denote it by C_{i,M_j}. An approximated utilization U'_{r,M_j} and the corresponding amount of time when resource r is used by specific application operations, can be computed as

$$U'_{r,M_j} = \frac{\sum_i S_i C_{i,M_j}}{T} \qquad T'_{r,M_j} = \sum_i S_i C_{i,M_j} \tag{2}$$

To solve the equation for the approximated costs C_{i,M_j}, several regression methods can be used. Typically, the goal is either to minimize the absolute error between U'_{r,M_j} and U_{r,M_j}, or their squared error over each monitoring window.

Open problems. It is worth investigating which regression methods are most suitable to be used for the application types we are targeting. Furthermore, an open question is how the monitoring window size and workload intensities impact the accuracy of the regression solution.

3.3 Inter- and intra- variations for mobile devices and virtual instances

Further we quantify how different is the current setup, for which we estimate the application's response time, from logged setups, in terms of resources capacities and usages.

Let us denote the current setup $S_{current} = (MD_c, VI_c)$ and the logged setup $S_{logged} = (MD_l, VI_l)$. For simplicity reasons, at this stage we only consider a single logged setup. In comparing $S_{current}$ and S_{logged}, two types of resource variations are used between MD_c and MD_l, as well as VI_c and VI_l, respectively. We denote by *inter-type variation*, the difference between the resources capacities when comparing distinct mobile devices and EC2 instances, respectively. An example is given in Table 1, where we want to compute the resources variation of the (Motorola Droid, EC2-Large) setup relative to the logged (HTC Desire, EC2-Small) setup.

In practice, the *inter-type variation* does not accurately reflect the difference between two distinct mobile devices or

Table 1: Resource-based comparison between two setups

	LOGGED		CURRENT	
Resources	HTC	EC2-S	Motorola	EC2-L
CPU	1 GHz	1 CU	600 MHz	4 CU
3G HSDPA	7.2 Mbps	–	10.2 Mbps	–
3G HSUPA	2 Mbps	–	5.76 Mbps	–
WiFi	802.11 b/g	–	802.11 b/g	–

virtual instances. For example, a mobile device usually runs several applications simultaneously, which means it cannot allocate 100% of the CPU or network capacities for an incoming application. Therefore, we define the *intra-type* variation to account for the actual resources usages on the underlying infrastructure. It only applies to compare mobile devices, since in real scenarios a third party has no access to information about the current resources usages in the cloud.

For instance, let us consider the HTC Desire in Table 1 has 70% CPU usage, while the Motorola Droid has 20% CPU usage. For a computational intensive application, it is possible that even though HTC Desire has a larger CPU capacity, its higher usage compared to the Droid would impact the application performance, making it slower in practice. We combine the inter- and intra- type variations to quantify how much faster or slower would the current mobile device be relative to the logged device, while executing computational modules or transferring data as follows:

$$C_r = \frac{r_{AB}(MD_c) * r_{USED}(MD_c)}{r_{AB}(MD_l) * r_{USED}(MD_l)} \quad (3)$$

where r_{AB} is the absolute capacity of the resource r (i.e. CPU and WiFi / 3G) and r_{USED} represents how much of r_{AB} in percentages is already used by other applications. If $C_r < 1$, then for the application parts that demand resource r, MD_c would require a longer execution time. The opposite applies for $C_r > 1$.

Open problems. In quantifying the inter- and intra- variations, we made the simplifying assumption that there is only one logged setup to compare against. In practice, this is hardly the case and therefore it is important to identify the logged setup that is closest to the current setup in terms of its resource capacities and usages. One possible direction is to treat resources as independent variables and apply regression-based methods.

3.4 Discussion

Finally, we want to correlate and combine all three factors discussed (Sections 3.1–3.3) in a unified estimation model. The queuing model proposed to identify application demands for underlying resources and the workload model based on nonstationary interactions combine naturally. In fact, the workload representation is an extension of the queuing model. Assuming that an interaction of type j lasts for k windows, we estimate the time required to execute operations that demand a specific resource r (i.e. computational steps require CPU, data transfers require bandwidth) over all N application modules, by combining the expressions from Eq. (2) and (3) as follows:

$$RT_r = \sum_{t=1}^{k} C_r^t \sum_{j=1}^{N} T'^{t}_{r,M_j} \qquad RT = RT_{CPU} + RT_{bandwidth} \quad (4)$$

In our current model that only considers CPU and bandwidth resources, it is easy to estimate the overall response time (RT) of an interaction by summing the times required to perform computational steps at module level (RT_{CPU}) and to transfer data between modules ($RT_{bandwidth}$). However, it requires further study to understand how to encompass additional resources, such as memory.

4. CONCLUSIONS

Modular mobile-cloud applications provide an alternative architectural model to flexibly distribute applications parts between a mobile device and a virtual instance to improve interaction response times. In this paper, we address the complex problem of estimating what would be the best response times for such applications in the case of specific workloads and mobile-cloud setups, without actually running them. We discuss how to model the impact of the application distribution scheme, as well as the workload size and intensity, and how they can be correlated with the resource variations of the given setups against logged setups. Future work will focus on finalizing a unified and generic model that encompasses all the above factors and evaluating its accuracy in real scenarios.

5. REFERENCES

[1] Barham, P., Donnelly, A., Isaacs, R., and Mortier, R. Using magpie for request extraction and workload modelling. In *Proc. of the 6th Conference on Symposium on Operating Systems Design and Implementation* (2004).

[2] Giurgiu, I., Riva, O., Juric, D., Krivulev, I., and Alonso, G. Calling the cloud: Enabling mobile phones as interfaces to cloud applications. In *Proc. of the 10th International Conference on Middleware* (2009).

[3] Madl, G., Dutt, M., and Abdelwahed, S. Performance estimation of distributed real-time embedded systems by discrete events simulations. In *Proc. of the International Conference on Embedded Software* (2007).

[4] Rolia, J., Kalbasi, A., Krishnamurthy, D., and Dawson, S. Resource demand modeling for multi-tier services. In *Proc. of the 1st International Conference on Performance Engineering* (2010).

[5] Sambasivan, R. R., Zheng, A. X., de Rosa, M., Krevat, E., Whitman, S., Stroucken, M., Wang, W., Xu, L., and Ganger, G. R. Diagnosing performance changes by comparing request flows. In *Proc. of the 8th USENIX Symposium on Networked Systems Design and Implementation* (2011).

[6] Schwarzer, S., Peschlow, P., Pustina, L., and Martini, P. Automatic estimation of performance requirements for software tasks of mobile devices. In *Proc. of the 2nd International Conference on Performance Engineering* (2011).

[7] Stevens, W., Myers, G., and Constantine, L. Structured design. In *IBM Systems Journal, 13(2)* (1974).

[8] Stewart, C., Kelly, T., and Zhang, A. Exploiting nonstationarity for performance prediction. In *Proc. of the European Conference on Computer Systems* (2007).

[9] Stewart, C., and Shen, K. Performance modelling and system management for multi-component online services. In *Proc. of the 2nd Conference on Networked Systems Design and Implementation* (2005).

[10] Urgaonkar, B., Pacifici, G., Shenoy, P., Spreitzer, M., and Tantawi, A. An analyical model for multi-tier internet services and its applications. In *Proc. of the International Conference on Measurement and Modeling of Computer Systems* (2005).

[11] Watson, B. J., Marwah, M., Gmach, D., dn Martin Arlitt, Y. C., and Wang, Z. Probabilistic performance modelling of virtualized resource allocation. In *Proc. of the 7th International Conference on Autonomic Computing* (2010).