MAPCloud: Mobile Applications on an Elastic and Scalable 2-Tier Cloud Architecture

M. Reza Rahimi\(^1\), Nalini Venkatasubramanian\(^1\)*, Sharad Mehrotra\(^1\)*, and Athanasios V. Vasilakos\(^2\)

\(^1\)School of Information and Computer Science, University of California, Irvine, USA.
\(^2\)National Technical University of Athens, Athens, Greece.
\(*\)\{mrrahimi, nalini, sharad\}@ics.uci.edu, \(^1\)vasilako@ath.forthnet.gr

Abstract—The rise in popularity of mobile applications creates a growing demand to deliver richer functionality to users executing on mobile devices with limited resources. The availability of cloud computing platforms has made available unlimited and scalable resource pools of computation and storage that can be used to enhance service quality for mobile applications. This paper exploits the observation that using local resources in close proximity to the user, i.e., local clouds, can increase the quality and performance of mobile applications. In contrast, public cloud offerings (e.g., Amazon Web Services) offer scalability at the cost of higher delays, higher power consumption and higher price on the mobile device. In this paper we introduce MAPCloud, a hybrid, tiered cloud architecture consisting of local and public clouds and show how it can be leveraged to increase both performance and scalability of mobile applications. We model the mobile application as a workflow of tasks and aim to optimally decompose the set of tasks to execute on the mobile client and 2-tier cloud architecture considering multiple QoS factors such as power, price, and delay. Such an optimization is shown to be NP-Hard; we propose an efficient simulated annealing based heuristic, called CRAM that is able to achieve about 84% of optimal solutions when the number of users is high. We evaluate CRAM and the 2-tier approach via implementation (on Android G2 devices and Amazon EC2, S3 and CloudFront) and extensive simulation using two rich mobile applications (Video-Content Augmented Reality and Image processing). Our results indicate that MAPCloud provides improved scalability as compared to local clouds, improved efficiency (power/delay) (about 32% lower delays and power) and about 40% decrease in price in comparison to only using public cloud.

I. INTRODUCTION

The past two decades of explosive growth of wireless networking, mobile computing and web technologies has profoundly influenced society at large. Almost anyone with access to a mobile device has access to services on the Internet and has reaped the benefits of instant accessibility to Internet-enabled technologies such as mapping applications, media streaming applications, games, instant messaging and email. We argue that the next generation of mobile applications involves significant use of rich media with more stringent Quality of Service (QoS) needs. Examples of technologies changing the mobile landscape include web-based learning tools, serious games that blur the distinction between education and entertainment, digital libraries that provide information of various topics, and technologies to empower the public to collaboratively develop, maintain, and share information.

The emerging Cloud Computing environment enables a new framework that shifts the physical location of computation and storage into the network to reduce operational and maintenance costs [19]. This paper aims to synergistically exploit mobile and cloud computing to enable services that can enrich the experience and capabilities of mobile users in a pervasive environment. While mobile computing empowers users with anywhere, anytime access to the Internet, cloud computing harnesses the vast storage, computing, and software infrastructure resources of large organizations (e.g. Amazon, Google) into a single virtualized infrastructure within reach of the general population.

In the cloud market, infrastructure providers offer reliable and customized services by enabling Service Level Agreements (SLAs) with consumers that dictate resource levels and QoS (in terms of speed, size, bandwidth, delay) bounds. One of the main bottlenecks in ensuring mobile QoS is the level of wireless connectivity offered by last hop access networks such as 3G and Wi-Fi. These networks exhibit varying characteristics. For example, 3G networks offer wide area ubiquitous connectivity; however, 3G connections are known to suffer from latency increases with an increasing number of users in a local cloud [4], [14]. Resulting in increased power consumption and cost at the user side. In contrast, Wi-Fi deployments, e.g. 802.11 hotspots, exhibit low communication latencies/delays, connected to or collocated with Wi-Fi access points can be used to form a nearby local cloud [4], [14]. Using a local only solutions with Wi-Fi networks creates scalability issues; as the number of users increases the latency and packet losses increase causing a decrease in application performance. Fig. 1 illustrates actual delays incurred with an increasing number of users in a local Wi-Fi-based cloud network executing OCR (Optical Character Recognition) mobile application.

In this paper, we will consider a 2-Tier architecture for the mobile cloud that synergistically combines the capabilities of local clouds and public cloud offerings to increase the performance and scalability of mobile applications. Specifically, we will develop efficient techniques for discovering and allocating resources in such a tiered cloud architecture to meet the multidimensional QoS needs (price, power, delay) of diverse mobile applications in the system.

Key Contributions: The main contributions of this paper are as follows:
1) We design a 2-tier cloud architecture for rich mobile applications and develop a mathematical formulation of the tiered cloud resource allocation problem (Sec II). Here, each application is modeled as a workflow of tasks which could be optimally decomposed in the 2-tier architecture based on a utility metric that combines service price, power consumption and delay. The resulting optimization problem is shown to be NP-hard.

2) We propose an efficient simulated annealing based heuristic, CRAM (Cloud Resource Allocation for Mobile Applications) to achieve a near optimal solution to the tiered cloud resource allocation problem (Sec III).

3) We develop a prototype of the MAPCloud system using Amazon Web Services: EC2, S3, CloudFront as the public cloud, a local campus cloud and Android devices. We implement two real-world rich media mobile applications in MAPCloud, (a) an OCR-based text reader (OCRS) that involves intensive image and speech processing and (b) a video Content augmented reality (VCAR) application, which augments user’s video clip with real data. We profile their power/delay characteristics under different configurations. The results from profiling on the prototype system were used to drive a thorough simulation study.

4) The simulation results indicate that the CRAM heuristic achieves close to 84% of the optimal solution when the number of users is high. They also indicate that the 2-tier cloud architecture for mobile cloud computing decreases power consumption and delay in average 32% when the price is fixed for each users in comparison to using only public cloud. This implies that while the resources are high in public cloud the wireless connectivity and capacity plays a bottleneck for system performance. This 2-tier cloud architecture also decreases the average user’s price about 40% (with fixed value of delay and power consumption) in comparison to only using public cloud.

**Related Work:** The idea of remote execution of resource-intensive tasks to alleviate resource constraints in mobile device is not new in itself. The typical application runs a simple GUI on the mobile device and intensive-processing tasks on a remote server, [20], [22]. Efficient execution of mobile applications by leveraging grid computing platforms has been addressed in systems such as MAPGrid [21]. In MAPGrid [21], intermittently available resources on grid platforms have been used to intelligently process and cache data for rich mobile applications such as video streaming. However, adapting the above techniques to work in the current cloud framework brings in new challenges and constraints. The autonomy of cloud resources leads to challenges in using the cloud effectively for mobile applications. In a grid environment, a grid proxy can provide storage, computational and network resources and it is often enough to find one resource node to service a mobile request. However, in the cloud environment, e.g. Amazon cloud services, storage and computational resources may be provided independently and charged individually. A single resource discovery process (for a request) may now need to be partitioned into multiple requests, one for each type of resource. The fact that users have to pay for public cloud resources also impacts the utility of these resources in the overall framework. The Cloudlets [16] platform provides mechanisms for creation of resources near access points (AP) that provide computational and storage services for mobile users. Other efforts [17], [14], [4], use concepts from workflow technologies to partition applications between the mobile device and a local cloud. In particular, parameters such as code-size, allocated memory and computational needs of the application are shown to be crucial in effective partitioning of the workflow for high utility [17]. The MAUI [14], CloneCloud [4] and [2] systems enable fine-grained energy-aware offloading of mobile application to the infrastructure. In particular, CloneCloud uses static and dynamic application profilers to optimize execution of mobile applications in terms of energy consumption. Mechanisms to offload the execution tasks include method shipping (in MAUI) and on-demand delivery of execution state to pre-instantiated threads. In contrast to the above efforts, MAPCloud integrates the use of current public cloud technology with local resources, enabling us to scale the mobile cloud system effectively to large deployments (via the public cloud) and ensure continuous availability (via the local cloud). There are some other approaches base on parallel processing of mobile applications such as Hyrax and [12], [3]. In Hyrax [10], a system architecture based on MapReduce [13] architecture has been proposed. It uses the network of smart phones and infrastructure to do intensive computational tasks. Although it has a nice and scalable architecture, the performance of Hyrax is poor for CPU-bound tasks. In [12], they proposed the architecture based on group of mobile devices to upload the task. They claimed that this architecture could improve the mobile application performance but they did not considered the performance of the application such as power and delay which are critical for mobile applications. In WhereStore [11], the authors considered the data sharing application. They showed that the locality of these storage can significantly improve the performance of the application, specially for location-based data search and sharing. In this work they mainly target to
reduce the missing rate of replicas in such applications.

II. MODELING RESOURCE ALLOCATION ON THE TIERED CLOUD

Fig. 2 shows the 2-tier cloud architecture for mobile applications [1]. Tier 1 nodes in the system architecture represents public cloud resources such as Amazon Web Services [25] and Google Application Engine [27]. Services that are provided by these vendors provide high scalability and availability on demand, but do not have fine grain location granularity that are required for high performance mobile applications (in the best case they have city level granularity) [16]. The second tier, i.e. local cloud consists of nodes that are connected to access points - location information of these resources are available at finer levels of granularity (campus and street level). Mobile users are typically connected to local clouds through Wi-Fi (via access points) or cellular (via 3G cell towers) connectivity - this enables us to offload device tasks onto cloud nodes.

To manage and run mobile applications in this architecture we need a middleware broker which exploits this architecture efficiently. MAPCloud performs task/resource mapping at a middleware broker node. The broker maintains a registry of resources and services in both tiers of the cloud. For each incoming application request (modeled as a workflow of tasks) from a mobile device, the broker consults the registry of available resources/services and executes admission control that determines whether the task is schedulable or not. Once admitted, the broker runs a scheduling policy to schedule the various workflow tasks onto specific nodes in the 2-tier cloud architecture. This middleware could be located mainly in the second tier to warranty the efficiency of the system (low delay).

To more formally model concepts and services in the 2-tier mobile cloud architecture, we use terminology from the Service Oriented Architecture SOA [23], [18] literature. SOA provides a flexible framework for modeling, reusing and composing existing services based on a workflow model. Furthermore, SOA enables representation of QoS parameters at multiple levels of abstraction - i.e. using an atomic service level QoS and a composite system-wide QoS [23].

Table I: QoS parameters that will be used in mobile cloud computing environment

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_{price}(s_i, U_{loc})$</td>
<td>The price of using service $s_i$ when user is in location $U_{loc}$.</td>
</tr>
<tr>
<td>$q_{power}(s_i, U_{loc})$</td>
<td>The power consumed on user mobile device using $s_i$ when user is in location $U_{loc}$.</td>
</tr>
<tr>
<td>$q_{delay}(s_i, U_{loc})$</td>
<td>The delay of executing service $s_i$ when user is in location $U_{loc}$.</td>
</tr>
</tbody>
</table>

We start by defining the concept of service set as follows:

Def. 1: Cloud Service Set $U_s$, the set of all services (e.g. storage and computation capabilities) provided by local and public cloud providers. It is denoted as:

$$C_s \triangleq \{s_1, s_2, \ldots, s_{|S|}\}$$

Def. 2: Cloud Location $C_{loc}$ is the location of the cloud resource $C$ in 2-D space.

Def. 3: User Service Set $U_s$ is the set of all services that a user has on his device (e.g. decoders, language translators, image editors etc.) is represented as:

$$U_s \triangleq \{u_1^s, u_2^s, \ldots, u_{|U_s|}^s\}$$

Def. 4: User Location $U_{loc}$ is the user location in 2-D space.

A generic mobile application is modeled as a workflow. [23], [18] where a workflow is consists of a number of logical and precise steps, each of which is known as a task. A workflow begins at the start task and finishes in the final task. Tasks in a workflow can be composed together in different patterns. The SEQ pattern indicates a sequential execution of tasks. The AND pattern models the parallel execution of the tasks. XOR is a conditional execution of tasks and LOOP pattern indicates an iterative repetition of the tasks. Each task is associated with a set of services that are capable of realizing and implementing the task in the tiered cloud architecture.

Several Quality of Service (QoS) parameters such as delay, power and price are associated with each service. Table. I shows the quality of service parameters that we will use in our mobile cloud computing environment. As it can be seen from the table, these QoS factors depend on user location. This is primarily due to the fact that communication link characteristics (Wi-Fi, 3G) vary based on user location and this in turn has an effect on the delay, power and price of the services and hence impacts the QoS. The delay of the service is considered as the difference between the time when a service is called (on the mobile device or cloud) and when the service is terminated. If the service on the cloud is being used we also account for the network delay (Wi-Fi or 3G). Power consumption of the service refers to the power consumed on mobile device to execute the service. If the service executes on the cloud, power consumed includes the power overheads of the network connection and data transfer related to that service. Finally, the price of the service is the actual price/cost to the end user of executing the service on the public cloud.
For each task $T_i$ in workflow $W$ we define $\chi_{T_i}$ as:

$$\chi_{T_i} \triangleq \{S_k \mid S_k \in U_s \cup C_s, S_k \text{ implements } T_i\}$$

Intuitively $\chi_{T_i}$ is the set of all services that could realize task $T_i$. For the workflow $W$ consisting of $n$ tasks, the set $\Gamma$ describes all the feasible solutions or execution plans [23]. It is defined as:

$$\Gamma \triangleq \chi_{T_1} \times \chi_{T_2} \times \ldots \times \chi_{T_n}$$

Table II defines the QoS for the application workflow based on the execution plan $\vec{x} \in \Gamma$. The QoS of a workflow is evaluated based on the QoS of its atomic services while taking into account the composition patterns [23]. The QoS of a SEQ pattern is the sum of the QoSs of the constituent tasks for all QoS parameters (price, power, delay). In the case of the AND pattern, that models parallel task flow, each of the QoS parameters is calculated independently. The price (power) of an AND workflow is the sum of the price (power) of the constituent tasks; the delay of the workflow is set to be the maximum delay of the parallel flows. In the XOR pattern, the maximum among the constituent values determines the QoS value all QoS types; for iterative tasks (i.e., structured as a LOOP), the QoS is determined by the number of executions of the service.

In order to formally state the tiered cloud resource allocation problem, we require normalized values (for price, power, delay) that can be used to calculate the utility of a service set to an application. This process is necessary while power, price and delay have different units like dollar, joule and second. The normalization process [23] that generate a normalized values for price is as follows (the extrapolation to delay and power is straightforward):

- $C_{\text{max,price}}$: The total price of the services in workflow when the most expensive services are selected.
- $C_{\text{min,price}}$: The total price of the services in workflow when the cheapest services are selected.
- $||W_{\text{price}}(\vec{x}, U_{loc})||$: Normalized price of the workflow with specific service plan $\vec{x} \in \Gamma$ is defined as:

$$||W_{\text{price}}(\vec{x}, U_{loc})|| = \begin{cases} C_{\text{max,price}} - W_{\text{price}}(\vec{x}, U_{loc}) & \text{if } C_{\text{max,price}} \neq C_{\text{min,price}} \\ 1 & \text{else} \end{cases}$$

From the above definition, we see that the higher the normalized price, the cheaper is the real price. We define a conservative notion of utility of a service plan $\vec{x}$, at a given location as the minimum achievable performance of that service set at the location.

$$\text{Utility}(\vec{x}, U_{loc}) = \min\{||W_{\text{price}}(\vec{x}, U_{loc})||, ||W_{\text{power}}(\vec{x}, U_{loc})||, ||W_{\text{delay}}(\vec{x}, U_{loc})||\}$$

Based on these definitions we will define following optimization problem for resource allocation in the tiered cloud. Our objective is to maximize the utility function as mentioned above. In other words we try to maximize the minimum performance of the application or maximize the savings on price, power or delay. It can be formally stated as:

$$\max \text{Utility}(\vec{x}, U_{loc})$$

subject to:

$$W_{\text{price}}(\vec{x}, U_{loc}) \leq C_{\text{price}}$$ (1)
$$W_{\text{power}}(\vec{x}, U_{loc}) \leq C_{\text{power}}$$
$$W_{\text{delay}}(\vec{x}, U_{loc}) \leq C_{\text{delay}}$$

The first constraint says that the price of the workflow should not be greater than a budget. The second and the third constraints place limits on the total power consumption and delay of the workflow.

The above optimization problem is NP-Hard - with the Knapsack problem being a special case of it [23]. In the next section we propose CRAM, a heuristic algorithm for solving this problem efficiently.

### III. CRAM: A Simulated Annealing Based Heuristic for Resource Allocation in the Tiered Cloud

In this section, we develop CRAM (Cloud Resource Allocation for Mobile Applications), an efficient heuristic for tiered-cloud resource allocations for mobile applications. We begin by introducing some notational conventions. First, we will apply a normalization process [23] for services. We illustrate it in the context of power, but is easily generalized to price and delay.

- $P_{\text{pow}}(\chi_{T_i})$: The maximum power consumption of the services that could realize task $T_i$.
- $P_{\text{pow}}(\chi_{T_i})$: The minimum power consumption of the services that could realize task $T_i$.
- For each services $s \in \chi_{T_i}$ the normalized power could be defined as:

$$\delta_{\text{pow}} = \begin{cases} \frac{P_{\text{pow}}(\chi_{T_i}) - s_{\text{pow}}}{P_{\text{pow}}(\chi_{T_i}) - P_{\text{pow}}(\chi_{T_i})} & \text{if } P_{\text{pow}}(\chi_{T_i}) \neq P_{\text{pow}}(\chi_{T_i}) \\ 1 & \text{else} \end{cases}$$

For each services $s \in S_{T_i}$ the total normalized QoS is defined as:

$$\hat{s} = \sqrt{\delta_{\text{pow}}^2 + \delta_{\text{price}}^2 + \delta_{\text{delay}}^2}$$

In general the higher the $\hat{s}$ is, the better the QoS/price (small delay, power consumption and price) of the service. The CRAM algorithm is a greedy heuristic that generates a near-optimal solution to the tiered cloud resource allocation problem using a simulated
annealing based approach. A simulated annealing based approach typically starts out with an initial solution in the potential solution space and iteratively refines this to generate increasingly improved solutions. The efficacy of simulated annealing (i.e. the speed with which one approaches the optimal solution) is dictated by the choice of the initial solution. CRAM uses a greedy approach for initial solution selection. There are two intuitions behind such a greedy selection:

- It is known that services in close proximity to the user usually provide better QoS performance in terms of delay and power consumption.
- Using services with high QoS will increase system utility.

In our context, improved QoS can be realized using one of four metrics – normalized delay, power, price and total normalized QoS.

In CRAM, we facilitate a better initial solution by veering the service selection towards those services in close proximity to the user. This is realized by storing the services in broker directory service/registry using a structure that enable efficient retrieval of nearby user services. Specifically, we store services using an R-tree based data structure [15]. Such an R-tree based data structure has been used for storing geometrical data and has been shown to enable efficient search, insertion, deletion and updates. Fig. 3 shows a sample R-Tree data structure for services. The R-tree structure splits the search space into hierarchically nested, and possibly overlapping, minimum bounding rectangles. We next illustrate how efficient retrieval of services near a user can be realized using an R-Tree data structure. As an example suppose we are interested in query "Retrieve all services in distance d of point A" as shown in Fig. 3 (a). The system will create a minimum rectangle that contains a circle with center A and radius d. This rectangle is called $R_d$ in Fig. 3 (a). Then it will search and find all overlapping rectangle with $R_d$ which is in our case is $R_0$ and retrieve all services in $R_0$.

Table III contains pseudo code for the CRAM algorithm. While CRAM uses simulated annealing as the core approach in selecting and refining service selection; custom policies have been designed to make it efficient for the 2-tiered cloud architecture with mobile applications. CRAM begins with a service selection $\text{FindService}(W, C, U_{\text{loc}})$ function that returns the list of services near the user that could realize the workflow and satisfy the constraints (of price, power and delay). The utility function of this solution is computed in line 2. Following this, the CRAM algorithm will enter a loop which is the main core for every simulated annealing based algorithm. The difference between the initial utility function and current utility function is extracted in line 6. If it is positive, it will be then considered as the new service list else with probability it will be kept and the algorithm will go to the next round of iteration. The while loop is eventually terminated when the number of iterations exceeds a limit $it$. After the iterations are done the final utility and service set will be returned as the solution.

Table IV illustrate the $\text{FindService}$ routine, which is the main module of CRAM. This routine returns the candidate list of services that realize the workflow by using a nearest, best service policy. In other words, the routine selects services that (a) have a high normalized QoS and (b) are within close proximity to the user location. To begin with, we select a candidate set of services within a threshold distance $d = d_{th}$ from the user. Four sorted lists are generated from the candidate set, sorted based on the normalized price, power, delay and total QoS from high to low. CRAM performs a randomization selection of services from these lists; the random selection is evaluated for satisfaction of the the input constraints. If input constraints are satisfied, the list is returned; else the process is repeated with an increased search distance.

**IV. SYSTEM PROTOTYPING AND PROFILING**

OCR+Speech (OCRS) and video content augmented reality (VCAR) applications have been developed as the rich mobile applications to study the performance of the proposed algorithm and architecture. In the first application the user takes a picture of the text page and the application will return a file which contains the spoken text. In the second application the user captures the video from a mobile device and uploads the

---

**TABLE III**

<table>
<thead>
<tr>
<th>CRAM ALGORITHM PSEUDO CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CRAM</strong> $(W, S, C, U_{\text{loc}}, max_{iter})$</td>
</tr>
<tr>
<td>$W$: Work Flow, $S$: Service Set, $C$: Constraint Vector, $U_{\text{loc}}$: User Location, $max_{iter}$: Number of Iteration in Simulated Annealing</td>
</tr>
<tr>
<td><strong>Begin</strong></td>
</tr>
<tr>
<td>(1) $\text{CandidateService} = \text{FindService}(W, C, U_{\text{loc}})$</td>
</tr>
<tr>
<td>(2) $\text{Util}_0 = \text{ComputeUtility}(\text{CandidateServices})$</td>
</tr>
<tr>
<td>(3) For $i=1$ to $max_{iter}$ do</td>
</tr>
<tr>
<td>(4) $\text{CandidateService} = \text{FindService}(W, C, U_{\text{loc}})$</td>
</tr>
<tr>
<td>(5) $\text{Util}_i = \text{ComputeUtility}(\text{CandidateService})$</td>
</tr>
<tr>
<td>(6) $\Delta = \text{Util}_i - \text{Util}_0$</td>
</tr>
<tr>
<td>(7) If $\Delta &gt; 0$</td>
</tr>
<tr>
<td>(8) $\text{Util}_0 = \text{Util}_i$</td>
</tr>
<tr>
<td>(9) Else</td>
</tr>
<tr>
<td>(10) Replace $\text{Util}_0 = \text{Util}<em>1$ when $\text{exp}(max</em>{iter}) \geq U[0,1]$</td>
</tr>
<tr>
<td>$\text{Util}[0,1]$ means the uniform distribution function $\gamma$</td>
</tr>
<tr>
<td>(11) End if</td>
</tr>
<tr>
<td>(12) Return $\text{CandidateService}, \text{Util}_1$</td>
</tr>
<tr>
<td><strong>End</strong></td>
</tr>
</tbody>
</table>

---

**TABLE IV**

<table>
<thead>
<tr>
<th>FINDSERVICE Routine</th>
<th>Returns the list of services that realize the workflow</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FindService</strong> $(W, C, U_{\text{loc}})$</td>
<td></td>
</tr>
<tr>
<td><strong>Begin</strong></td>
<td></td>
</tr>
<tr>
<td>(1) $\text{CandidateService} = \text{FindService}(W, C, U_{\text{loc}})$</td>
<td></td>
</tr>
<tr>
<td>(2) $\text{Util}_0 = \text{ComputeUtility}(\text{CandidateService})$</td>
<td></td>
</tr>
<tr>
<td>(3) For $i=1$ to $max_{iter}$ do</td>
<td></td>
</tr>
<tr>
<td>(4) $\text{CandidateService} = \text{FindService}(W, C, U_{\text{loc}})$</td>
<td></td>
</tr>
<tr>
<td>(5) $\text{Util}_i = \text{ComputeUtility}(\text{CandidateService})$</td>
<td></td>
</tr>
<tr>
<td>(6) $\Delta = \text{Util}_i - \text{Util}_0$</td>
<td></td>
</tr>
<tr>
<td>(7) If $\Delta &gt; 0$</td>
<td></td>
</tr>
<tr>
<td>(8) $\text{Util}_0 = \text{Util}_i$</td>
<td></td>
</tr>
<tr>
<td>(9) Else</td>
<td></td>
</tr>
<tr>
<td>(10) Replace $\text{Util}_0 = \text{Util}<em>1$ when $\text{exp}(max</em>{iter}) \geq U[0,1]$</td>
<td></td>
</tr>
<tr>
<td>$\text{Util}[0,1]$ means the uniform distribution function $\gamma$</td>
<td></td>
</tr>
<tr>
<td>(11) End if</td>
<td></td>
</tr>
<tr>
<td>(12) Return $\text{CandidateService}, \text{Util}_1$</td>
<td></td>
</tr>
<tr>
<td><strong>End</strong></td>
<td></td>
</tr>
</tbody>
</table>

---

**Fig. 3.** R-Tree Data Structure: (a) Partitioning the 2-D space into rectangles (b) R-Tree structure of services
CRAM FIND SERVICE ALGORITHM PSEUDO CODE

crfindservice(W, C, Uloc)
begin
We assume that the directory service database contains information
on the normalized QoS of the service with R-Tree indexing.
if W: Work Flow, S: Service Set, Uloc: User Location,
then make 4 different lists sorted according to normalized
price, normalized, power, normalizes delay and normalized
total QoS from large to small.
randomly choose from the 4 list services.
check if it satisfies the constraints
else if yes return the service set
else
increase the search radios to d = dth + i * dr
end while
End

TABLE IV
CRAM FIND SERVICE ALGORITHM PSEUDO CODE

video to the server. On the server, ARToolkit [24] will be run
to infuse the 2D object in video. The resulted video will be
send back to the user. For both of these applications 9 different
services (RESTful Web Services) has been extracted such as
image filtering, noise cancelation, video format conversion,
etc. We measure the delay and power consumption of services
in different situation for both local and public cloud. For
measuring power PowerTutor[7] has been used.

To implement these applications and CRAM a middleware
has been implemented. Fig. 4 shows different components of
this middleware as described below [6]:

Broker: The broker serves as the point of contact for all
applications in the MAPCloud system - its key task is to
perform admission control, i.e. determine whether incoming
mobile application requests (structured as task workflows) can
be accepted or rejected based on the current system state and
available resources.

Service Directory: This module serves as a repository of
information on services available in the MAPCloud infrastruc-
ture, i.e. on local cloud, public cloud and user devices along
with their QoS. It is implemented using a MySQL DB [29] and
and R-Tree indexing structure (described earlier) for efficient
spatial querying.

Scheduler: This module implements policies for efficiently
(or optimally) decomposing the mobile application workflow
on the 2-tier cloud architecture using information on service
distribution and QoS from the directory service. In particular,
the CRAM algorithm is implemented in this module.

Mobile Client and Cloud Monitoring Modules: Given the
dynamic nature of mobile users and applications, optimized
allocation of resources requires constant and accurate moni-
toring of (a) service executions on user mobile client, such
as power consumption and processing delay, and (b) QoS
associated with each service on local or public cloud.

The operational flow through this module is simple - a
user request for a mobile application is forwarded to the
broker. If admitted, the scheduler module will compute and
determine the best allocation of services based using the
CRAM algorithm. The broker and scheduler modules con-
sult the directory service (which keeps accurate state of the
resources based on information gathered by the mobile client
and cloud monitoring modules. The allocation of services to
nodes is returned back as service plan (encoded in XML) - the
associated XML file also contains the URL of each service
in application workflow. In our prototype implementation,
we have used Android G2 devices as mobile client; a 64bit
Windows dual-core server with 8GB of memory and 500GB
of storage as the local cloud. Mobile users could connect to
local cloud using free Wi-Fi or 3G. For 3G we considered
T-Mobile 3G services for our experiment [28]. We have used
the Amazon Web Services (EC2, S3 and CloudFront) large
instance as the server on cloud with windows OS, this is
equivalent to a PC with 7.5GB of memory, 850 GB of storage.
In our experiment we used Amazon Web Services hourly price
and For 3G we used T-Mobile price plan [28]. It is reasonable
to consider free Wi-Fi connectivity and free local clouds (or
extremely cheaper than public cloud offering).

To test the performance of the architecture and proposed
algorithm, in particular we perform scalability studies on a
cloud simulation engine. In particular, we use CloudSim [5],
an open source cloud simulator which supports modeling
of data centers, virtual machines and resource provisioning
policies in a cloud computing environment. The experimental
result obtained by profiling real applications in the prototype
has been used to tune the simulation environment.

We first show the optimality of the CRAM algorithm
and 2-tier cloud architecture. This is followed by a detailed
performance study of the two sample applications on varying
configurations of the tiered cloud under different CRAM
settings. The basic simulation setup models a region with 225
cells. Local clouds have valid Wi-Fi in 5 cells around and
there exists 3G connectivity in whole region. A LAN provides
a backbone for local cloud connectivity and data transfer.
Fig. 5. CRAM Algorithm throughput according to different types of applications, number of users and different combination of cloud resources: (a) OCRS (b) VCAR

Users and local clouds are distributed uniformly in whole region, and the maximum number of CRAM iterations has been set to 20. In our experiments, we varied data sizes which were uniformly distributed from [2Mb, 4Mb]. Each simulation results is the average of 10 runs. Different experimental scenarios considered include (a) diverse applications such as OCRS, VCAR (b) varying numbers of local and public cloud resources.

**CRAM Optimality Study**: To measure CRAM performance we have defined the following metric as the CRAM algorithm throughput:

\[
CRAM_{Throughput} = \frac{CRAM \text{ output}}{Optimal \text{ Solution of Eq.1}} \times 100
\]

A brute force search has been used to find the optimal solution of Eq.1. Fig. 5 shows the CRAM algorithm throughput for different types and combination of applications for local and public cloud resources. In Fig. 5 (a) the OCRS application has been considered. We have considered up to 100 concurrent users in the system. Our results indicate that when the number of users is small, the CRAM algorithm performance is around 52% in average for each user. This is because of the large size of search space in comparison to the number of users. This performance will increase to 84% when about 100 concurrent users in the system. The performance of CRAM is similar for the VCAR application (See Fig. 5 (b) and (c)). This motivates that CRAM performs close to the optimal solution independent of application type. Table V shows the running time of CRAM and brute-force search method (on a 64bit Windows dual-core Intel with 8GB of memory and 500GB of hard). According to this table when the number of mobile users is small such as 20 CRAM execution time is 4 times faster than brute-force search method. This ratio increased to 20 when there is a large number of users in the system (about 100).

2-Tier Cloud Architecture Performance Study: In this section we try to study the performance of each cloud tier.

<table>
<thead>
<tr>
<th>Number of Users</th>
<th>20</th>
<th>40</th>
<th>80</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRAM (seconds per person)</td>
<td>2s</td>
<td>6s</td>
<td>15s</td>
<td>27s</td>
</tr>
<tr>
<td>Brute-Force Search (seconds per person)</td>
<td>8s</td>
<td>55s</td>
<td>240s</td>
<td>550s</td>
</tr>
</tbody>
</table>

**TABLE V**

**PROCESSING TIME OF CRAM AND BRUTE-FORCE SEARCH METHOD**

Fig. 6. CRAM algorithm real values for delay and power consumption on average input data size of 3Mb according to different types of applications (OCRS, VCAR), different number and types of clouds when there are 100 concurrent users in the system.

Fig. 6 (a) and (c) show the real delay of OCRS and VCAR applications according to different number of local and public cloud instances. The x axis presents number of public cloud instances (Amazon Large Instance) and y axis presents the average delay in seconds when there are 100 users in system. As it can be derived from graphs, by using large number of local cloud instances (e.g 16 instances), one could get about 60% decrease in delay in comparison to only using large number of public cloud instances (e.g 20). This indicates that communication delay and capacity is still a bottleneck in wireless environment. These figures also indicate that using local and public cloud resources could improve the performance in comparison to only using public cloud (in average around 32% by fixing the price for each user). The same is valid for power consumption as shown in Fig. 6 (a) and (c). By using only large number of local clouds (e.g 16), one could get 54% decrease in power in comparison to use only large number of public cloud instances. This decrease in power consumption is in average about 25% when using the combination of local and public cloud. It can be understood from figures that power consumption and delay are correlated quantities. This makes sense while long delay is usually because of long communication and processing time which results more power consumption.

Fig. 7 shows another important issue, which is the price of using Amazon Web Services. We have used the *pay as you go* model (per hour usage) [25]. The x axis presents the number of local cloud and Y axis presents the average price for each user (when there 100 users in system). When there are large number of local cloud instances in system (e.g 16) the average...
price for each user is low (0.06$). This increase to 0.75$ when the resources on public cloud are used (about 20 public cloud instances) or 1200% increase in price. By fixing the delay and power consumption of both OCRS and VCAR applications, we still could reach the same delay and power consumption when using 2-tier architecture in comparison to only using public cloud resources. For example according to the figures to achieve delay (20s) and power consumption (35 jole), we could use 20 instances of public cloud or 8 instances of local cloud and 8 instances of public cloud by paying only 0.31$ for each person which is decrease around 40% in comparison to only using public cloud.

V. CONCLUSIONS AND FUTURE DIRECTIONS

In this paper we showed how to use cloud platform to increase the performance of mobile applications. We introduce a tier architecture consisting of the local and public clouds, the CRAM algorithm, which efficiently decomposes mobile applications on mobile client and 2-tier cloud architecture. Several important QoS factors such as power, price and delay have been considered. Our results indicate that the CRAM heuristic achieves close to 84% of the optimal solution when the number of users is high. Our results also indicate that the 2-tier cloud architecture for mobile cloud computing decreases power consumption and delay, i.e. improves QoS (about 32% as compared to using the public cloud) and increases scalability (as compared to local clouds) and about 40% decrease in price in comparison to only using public cloud for rich mobile applications.

In future we try to extend our approach for streaming mobile applications and study how user trajectory will be used to design optimal algorithm and architecture.

VI. ACKNOWLEDGMENTS

The authors would like to thank the DSM and ISG group members in the university of California Irvine, for their valuable and constructive comments to improve the paper.

REFERENCES


