Impact of Different Auto-Scaling Strategies on Adaptive Mobile Cloud Computing Systems

Preprint — The paper has been accepted for presentation at IEEE ISCC 2016

Michele Amoretti, Luca Consolini, Alessandro Grazioli, Francesco Zanichelli

Department of Information Engineering

Università degli Studi di Parma

Parma, Italy 43124

Email: {michele.amoretti,luca.consolini,francesco.zanichelli}@unipr.it, grazioli@ce.unipr.it

May 10, 2016

Mobile Cloud Computing (MCC) is an emerging paradigm aiming to elastically extend the range of resource-intensive tasks supported by mobile devices, leveraging upon broadband connectivity and cloud-based resources. In literature, almost all MCC models focus on mobile devices, considering the Cloud as a system endowed with unlimited resources. In this paper, we illustrate a novel MCC model characterized by the presence of adaptive loops, *i.e.*, feedback interactions between the mobile device and the Cloud, with the purpose to enforce adaptive behavior on both sides. Indeed, the Cloud adapts its resource allocation (number of activated virtual machines) to the workload provided by mobile devices. On the other hand, feedback from the Cloud allows mobile devices to improve offloading decisions. The performance of the whole system is heavily affected by the auto-scaling strategy adopted by the Cloud. By means of simulations, we have evaluated the impact of two very different auto-scaling strategies. Quantitative results are reported and discussed.

1 Introduction

Exploiting cloud infrastructures to mitigate resource scarcity of mobile devices, also known as Mobile Cloud Computing (MCC) [1], is an interesting concept that has been recently supported by theoretical frameworks (*e.g.*, MAUI [2] and CloneCloud [3]) and practical implementations (such as ThinkAir [4]). Abolfazli *et al.* [5] elaborated the entire history of MCC and presented the most interesting models and solutions. Their analysis points out that almost all MCC models focus on mobile devices, considering the Cloud as a system endowed with unlimited resources.

To contribute in filling this gap, we propose a modeling framework encompassing both sides of the MCC system. In particular, our framework is characterized by an autonomic strategy consisting of *adaptive loops* that involve mobile devices and the Cloud, aimed at automatically determining energy-time tradeoffs, and achieve better global performance. With respect to the reference strategy proposed by Kumar *et al.* [6], we take into account the dynamics of the Cloud (in this paper, we focus on only one type of cloud-based resources, which are distant giant data centers), with a concrete and effective approach. For example, when offloading decisions have to be taken for a given job, the mobile device can obtain up-to-date estimates of the Cloud speedup and available bandwidth. At the same time, the Cloud can periodically update the number of active VMs (*auto-scaling* [8]), based on workload characterization, in order to maintain a given level of service with optimized costs.

In particular, we analyze the impact of two alternative auto-scaling strategies on the whole MCC system. The first one, based on control theory, adapts the number of active VMs based on a *standard integral* *controller*, to the purpose of reducing latency. The second strategy, an extension of the recently proposed approach by Jiang *et al.* [7], consists in solving a *global optimization* problem, with two conflicting goals, namely reducing cost and latency.

The paper is organized as follows. In Section 2, the MCC system model is presented. In Section 3, the two alternative auto-scaling strategies for the Cloud are illustrated. In Section 4, the performance evaluation of the whole MCC system, with respect to realistic scenarios, is reported. In Section 5, other MCC models and auto-scaling strategies are recalled and compared to the proposed ones. Finally, Section 6 concludes the paper with a discussion of achieved results and future work.

Notation	
p_{off}	Offloading probability
N_t	Number of tasks forming a job (r.v.)
n_c	Avg number of CPU cores, for mobile devices
α	Avg clock frequency ratio (VM versus mobile device)
T	Network throughput [Byte $\cdot$ s^{-1}] (r.v.)
C	Job size in terms of number of instructions (r.v.)
S	Cloud speedup (nominal or estimated at runtime)
$c(T,C,S)$	Energy balance, on the mobile device
M	Job execution speed (instructions per second) [s^{-1}]
D	Amount of transferred data (r.v.) [Byte]
R	Required RAM space (r.v.) [Byte]
W_m	Avg processing power, on the mobile device [W]
W_i	Avg idle power, on the mobile device [W]
W_{off}	Avg job offloading power, on the mobile device [W]
$t_m^{(j)}$	Expected job execution time, on the mobile device [s]
$t_{off}^{(j)}$	Expected job offloading time [s]
$t_{ec}^{(j)}$	Expected job execution time, on the Cloud [s]
k	Number of cloud VMs $\in [k_{min}, k_{max}]$
ϕ_c	Max waiting time / max execution time (for a task)
s	Setpoint for ϕ_{tc}
δ	Parameter of the standard integral controller
f_c	Fraction of completed tasks on the Cloud
$L = \delta / f_c$	Gain of the standard integral controller
t_{sc}	Expected task service time on the Cloud [s]
t_{wc}	Expected task waiting time on the Cloud [s]
t_{ec}	Expected task execution time on the Cloud [s]
t_{off}	Expected task offloading time [s]
T_{th}	Threshold for t_{ec} [s]
$F(k)$	Cost of having k active VMs
γ	Importance ratio of VM cost and average task latency

2 MCC System Model

The MCC system we consider is illustrated in Figure 1. Users execute jobs on their mobile devices. We assume that a job is a set of independent tasks, whose size N_t is a random variable (throughout the paper, for the sake of simplicity, it will be assumed to be uniform). For each job, its tasks are executed either sequentially or in parallel (if n_c cores are available within the CPU) on the mobile device, while they are always executed in parallel in the Cloud. Hence, the nominal Cloud speedup[1] is approximated as $S = \alpha N_t / n_c$, where α is the ratio between the clock frequency of a VM's processor and the clock frequency of the mobile device's cores, since other architectural differences between processors are not considered.

Other works consider possible dependencies among tasks belonging to the same job [2, 3, 9, 10, 11], modeling such dependencies as a graph and analyzing how to partition the application tasks on the mobile

[1] In this context, the speedup is defined as the ratio between the sequential execution time and the parallel execution time of the job. Task execution time is approximated as the sum of the *waiting time* before being processed and the processing time (denoted as *service time*, throughout the paper).

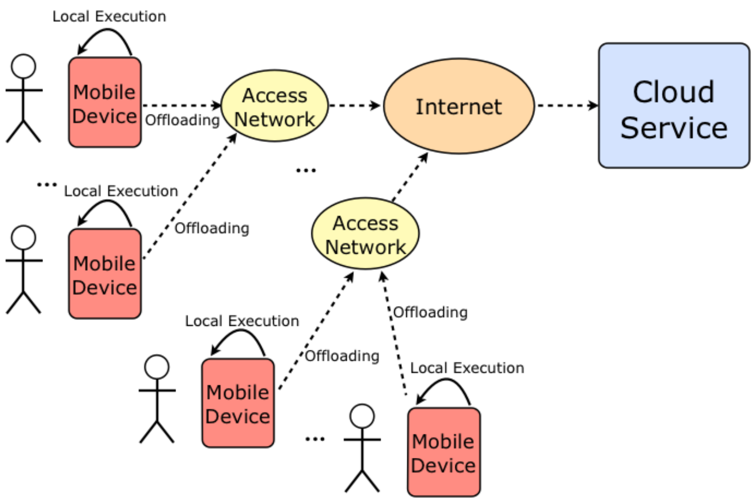

Figure 1: The MCC system.

and cloud resources. The tasks that have to be offloaded are sent to the Cloud with different priorities, *i.e.*, definitively, with different deadlines [12].

2.1 Offloading Probability

The offloading probability is defined as

$$p_{off} = P\left\{\bigcup_i cond_i\right\} \tag{1}$$

where $cond_i$ is the i-th offloading condition. For example, a foremost condition for MCC (introduced by Kumar *et al.* [6]) is related to energy savings for the mobile device. Given a computational job, offloading takes place if the energy required to transfer it to the Cloud is lower than the energy necessary to execute it on the mobile device, *i.e.* the device energy balance is favorable:

$$c(T,C,S) = \frac{C}{M}(W_m - \frac{W_i}{S}) - W_{off}\frac{D}{T} > 0 \tag{2}$$

being C the job size in terms of number of instructions, M the job execution rate (instructions per second) on the mobile device, W_m the average power consumed to execute jobs on the mobile device, W_i the average power consumed by the mobile device when idle, S the (nominal or estimated at runtime) Cloud speedup, W_{off} the average power consumed by the job transfer (*i.e.*, the data exchange between the mobile device and the Cloud, mostly), D the amount of transferred data, T the throughput that is available to the mobile device. Condition (2) means that the energy cost of local execution is higher than the offloading one.

The expected job execution time can thus be estimated by means of the offloading probability:

$$t_e^{(j)} = p_{off}(t_{ec}^{(j)} + t_{off}^{(j)}) + (1 - p_{off})t_m^{(j)} \tag{3}$$

where $t_{ec}^{(j)}$ is the expected *job execution time on the Cloud* (*i.e.*, the time interval between the arrival of the first task and the completion of the last task of the job, on the Cloud), $t_{off}^{(j)}$ is the expected time required by the offloading process, $t_m^{(j)}$ is the expected *job execution time on the mobile device*. It should be noted that p_{off} may depend (among others) on a constraint on the job execution time.

2.2 Network Model

Once offloading for a job has been evaluated as beneficial, its tasks are sent to the Cloud. Assuming that task offloading is performed over TCP connections, we model the wireless access network and the Internet as a delay center (like Ahmed *et al.* [16] and Cardellini *et al.* [15] did). Given the Round Trip Time (RTT) between the mobile device and the Cloud, we compute the throughput T. User mobility may cause intermittent connectivity or handover between different wireless access networks. Temporary disconnections are taken into account as a delay t_{disc} with probability p_{disc}. Handover is taken into account as a delay t_h with probability p_h. Thus, the expected time required by the offloading process is computed as:

$$t_{off}^{(j)} = \frac{D}{T} + p_{disc}t_{disc} + p_h t_h \qquad (4)$$

2.3 Dispatching and Scheduling Strategies in the Cloud

Modeling data centers is a very challenging task, given the interaction and complexity of internal processes and external workloads [14, 13]. Like several other authors [18, 14, 15], we use a Queueing Network (QN) model (illustrated in Figure 2).

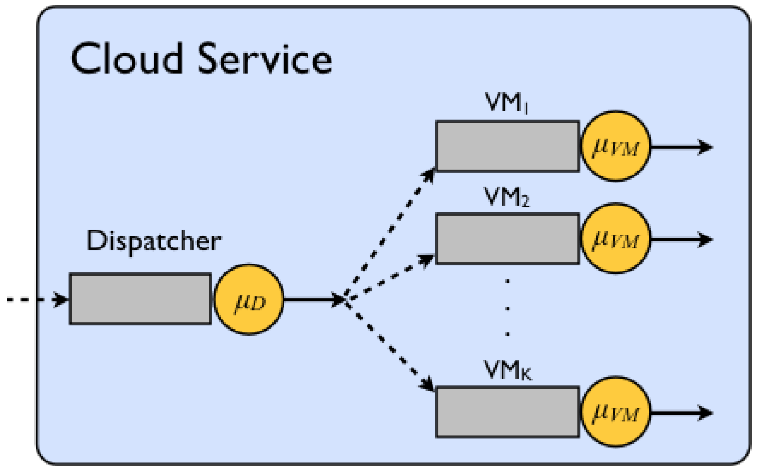

Figure 2: QN model of the data center.

Tasks are assigned to VMs by a Dispatcher that adopts strategies like RANDOM, where each task is assigned to a randomly selected VM, or Shortest Queue First (SQF), where each task is assigned to the VM with less waiting tasks.

Then, each VM uses the First-Come First-Served (FCFS) strategy to serve tasks in the order they arrive into the queue of the VM. An alternative scheduling strategy is Shortest Job First (SJF), where the waiting task with the smallest service time is given highest priority.

2.4 Adaptive Loops

Within the proposed MCC model, we introduce the possibility to define adaptive loops between the single mobile device and the Cloud, to the purpose of improving efficiency on both sides. An example is illustrated in Figure 3 and described in the following. The mobile device periodically obtains updated Cloud speedup S and throughput T estimations at runtime, to be used for assessing the energy cost trade-off $c(T,C,S)$ and the expected job execution time on the Cloud $t_e^{(j)}$, in order to make a decision about job offloading. At the same time, the Cloud maintains workload statistics, useful to optimize the number k of active VMs and to provide refined speedup predictions. With several concurrent adaptive loops — one for each mobile device — and a stationary workload, the MCC system should be able to quickly converge to a stable state. Workload modifications should have the effect of driving the system to different stable states. The reactivity of the system will ultimately depend on the adaptation processes implemented in the mobile devices and the Cloud.

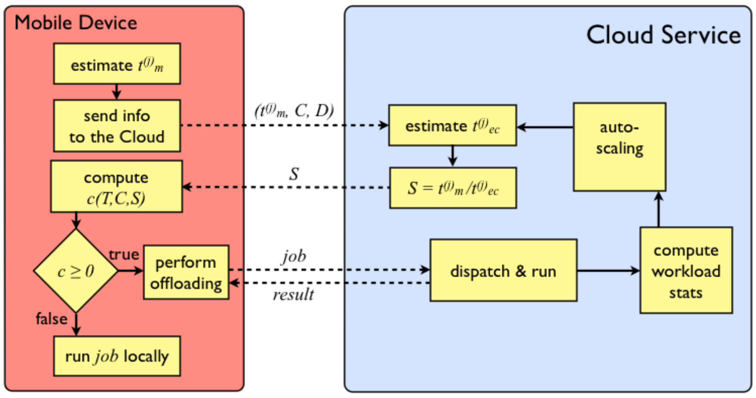

Figure 3: Example of adaptive loop in the MCC system.

In Eq. (2), using the nominal speedup value S is not realistic, as the actual real value of S depends on the overall workload the Cloud is subject to. Hence, it is better that each offloading decision is based on an estimated S provided by the Cloud upon request. In detail, each mobile device determines the expected time to complete the jobs in its queue, and adds it to the estimated local execution time of the target job. The resulting $t_m^{(j)}$ is communicated to the Cloud, along with a descriptor of the target job (*i.e.*, number of independent tasks, clock cycles of each task, etc.). Then, the Cloud estimates $t_{ec}^{(j)}$ as the time it would require to execute the target job, taking into account the possibility to run its component tasks in parallel. Finally, the estimated speedup is computed as $S = t_m^{(j)}/t_{ec}^{(j)}$ and returned to the mobile device, to be used for computing $c(T,C,S)$ with eq. 2.

3 Auto-scaling Strategies for the Cloud

As the statistics of the workload may change over time, the Cloud should be able to adapt the number k of active VMs, in order to satisfy the service level agreements (SLAs) with the clients. An accurate review of auto-scaling techniques for elastic applications in cloud environments has been recently proposed by Lorido-Botran *et al.* [8]. In the following, we illustrate two alternative strategies to periodically compute the updated value of k, by taking into account performance indicators, as well as system parameters. The first strategy is based on control theory — in particular, a standard integral controller with adaptive gain is used. The second strategy, based on global optimization, is an extension of the recently proposed approach by Jiang *et al.* [7].

3.1 Auto-scaling with Standard Integral Controller

The controlled discrete-time system is illustrated in Figure 4. There, $k(t)$ is the number of VMs at time t; $\phi_c(t)$ is the ratio between the average maximum task waiting time and the maximum task execution time (*i.e.*, waiting time t_{wc} plus service time t_{sc}; also denoted as latency of the cloud) at time t; $s \in [0,1]$ is the set point for ϕ_c; $e(t) = \phi_c(t) - s$. The block B forces $k(t)$ between k_{min} and k_{max}.

In theory, to map the task execution time constraint T_{th} (*i.e.*, the maximum allowed t_{ec}), defined in the Service Level Agreement (SLA) between user and cloud provider, on the set point s, it is sufficient to estimate the maximum task service time t_{sc}. Then, it is possible to set the maximum allowed task waiting time as $t_{wc} = t_{ec} - t_{sc}$ and consequently set $s = t_{wc}/t_{ec}$. In practice, this approach for setting s is not sufficiently prudent (as shown by the simulation results presented in Section 4).

A standard integral controller (SIC) is characterized by the following transfer function (with $z \in \mathbb{C}$):

$$I(z) = L\frac{z}{z-1} \tag{5}$$

Thus:

$$K(z) = I(z)E(z) = L\frac{z}{z-1}E(z) \tag{6}$$

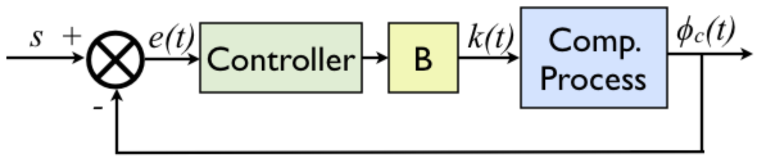

Figure 4: The controlled discrete-time system.

which can be written as:

$$(z-1)K(z) = LzE(z)$$

In the time domain, the latter becomes:

$$k(t+1) - k(t) = Le(t+1)$$

which is equivalent to:

$$\begin{aligned} k(t) &= k(t-1) + Le(t) \\ &= k(t-1) + L(\phi_c(t) - s) \end{aligned}$$

Gain L can be tuned by means of the well-known Ziegler-Nichols heuristic method [19].

In this paper, we consider the alternative approach where L is self-tuning. More precisely, $L = \delta / f_c$, where $\delta \in \mathbb{R}$ is a system parameter, and $f_c \in [0,1]$ is the fraction of completed tasks versus arrived tasks. When L becomes stable, the aforementioned expression for $k(t)$ is exact. We observe that the term $L(\phi_c(t) - s)$ must be rounded to next upper integer. Moreover, we observe that, in any case, $k(t)$ cannot be updated to a value that is higher than k_{max} or lower than k_{min}.

3.2 Auto-scaling by global optimization

Extending the recently proposed approach by Jiang *et al.* [7], we formulate the auto-scaling problem as a global optimization (GO) one, by periodically exploiting the following cost-latency trade-off objective function:

$$arg\min_{k} \Gamma(\lambda, k) = \gamma F'(k) + (1-\gamma) t'_{ec}(\lambda, k) \quad (7)$$

subject to:

$$F(k) \le F_{max}; t_{ec}(\lambda, k) \le T_{th} \quad (8)$$

where λ is the rate of the tasks entering the cloud, k is number of active VMs, $\gamma \in [0,1]$ reflects the importance ratio of cost and latency, $F(k)$ is the cost of having k active VMs, F_{max} is the maximum allowed cost per period, $t_{ec}(\lambda, k)$ is the latency of the cloud, and T_{th} is the latency threshold (defined in the SLA). $F'(k)$ and $t'_{ec}(\lambda, k)$ are the normalized cost and latency, respectively.

With respect to the global optimization approach proposed by Jiang *et al.* [7], here t'_{ec} does not depend only on k, but also on the task rate, which, in general, is a function of time: $\lambda = \lambda(t)$. In simulation, λ can be easily periodically measured. Thus, solving the global optimization problem is quite straightforward, and even a brute force approach takes a reasonable amount of time.

The Dispatcher and the VMs are modeled as $M/M/1$ queues with FCFS scheduling strategy, assuming $\lambda < \mu_D$ (being μ_D the service rate of the Dispatcher). This model fits with the type of compute-intensive workload we consider, as shown later in Section 4. Assuming that the average queue length is short, the SQF and RANDOM dispatching strategy are almost equivalent, in terms of load distribution over the k active VMs. Indeed, when new VMs are activated, the SQF strategy sends tasks to them, rather than to long-running VMs. But, after a short transitory, all VMs have queues of similar length. At this point, the Dispatcher behaves as if it was provided with the RANDOM strategy.

In summary, each active VM should face a task rate $\lambda/k < \mu_{VM}$. Being all VMs equivalent and working

Table 1: Power demand of different mobile devices.

Model	WiFi [W]	CPU [W]	Idle [W]
Openmoko Neo Freerunner [20]	0.7	0.67	0.25
T-Mobile G1 [21]	$0.6 \div 1.3$	n.a.	$0.1 \div 0.5$
HTC Magic [21]	$0.6 \div 1.3$	n.a.	$0.1 \div 0.5$
HTC Nexus One [22]	0.35	0.55	0.1
HTC Galaxy i7500 [23]	0.628	0.606	n.a.
HTC Nexus S [23]	0.455	0.886	n.a.
HTC HD2 [24]	0.817	n.a.	n.a.
HTC EVO [25]	n.a.	1.005	n.a.
Motorola ATRIX 4G [25]	n.a.	0.918	n.a.
Motorola Droid [25]	n.a.	0.944	n.a.

in parallel, the resulting latency of the Cloud (considering the Dispatcher and the VMs) is:

$$t_{ec} = \frac{1/\mu_D}{1-\rho_D} + \frac{1/\mu_{VM}}{1-\rho_{VM}} \tag{9}$$

where ρ is the utilization of the queue. More precisely:

$$\rho_D = \lambda/\mu_D \tag{10}$$

$$\rho_{VM} = (\lambda/k)/\mu_{VM} \tag{11}$$

The normalized latency is defined as:

$$t'_{ec}(\lambda,k) = t_{ec}(\lambda,k)/T_{th} \tag{12}$$

The cost of having k active VMs may be modeled with a linear function $F(k) = \beta k + F_0$, or with any other function. The normalized cost of having k active VMs is defined as:

$$F'(k) = \frac{F(k) - F_{min}}{F_{max} - F_{min}} \tag{13}$$

4 Performance Evaluation

To evaluate the MCC model and the auto-scaling strategies, we used the DEUS simulation tool [17]. Considered mobile devices are characterized by 1, 2 or 4 CPU cores (with the following reasonable distribution: 30%, 60% and 10%, respectively), and the following parameters:

- $W_m = 0.6 \div 0.9$ W (0.5 W being the power consumed by the operating system and other processes executed in background with respect to the considered app);
- $W_i = 0.25$ W;
- $W_{off} = 0.7$ W;
- $M = 800$ or 1400 MHz (equally distributed);
- available RAM = 1 GB.

These values take into account both outdated devices and more recent ones (in Table 1, we summarize our reference data).

We assumed WiFi connections with communication RTT $\in [125, 261]$ ms, the range we measured in our University Campus when accessing Amazon cloud services.

We considered jobs with multiple tasks ($N_t = U(1,10)$), each task being characterized by:

- number of instructions: $C/N_t = U(5.3 \cdot 10^9, 3.3 \cdot 10^{10})$
- required RAM space: $R = U(1,5)$ MB
- input data size: $D/N_t = U(1,3)$ MB
- allowed task completion time: $t_{max} = 40$ s

where U(x,y) means uniform distribution between x and y. The range of the number of instructions has been empirically estimated considering an application that processes image files ranging from 1600×1200 to 4000×3000 pixels. In case of offloading, the task completion time includes the offloading time t_{off}, which can be roughly estimated as $D/N_t T$, where T is the estimated network throughput (see Section 2.B). For simplicity, we considered $p_{disc} = p_h = 0$. In the worst case, $t_{off} = 5$ s. Thus, the resulting threshold for task execution time to be imposed in the SLA is $T_{th} = t_{max} - t_{off} = 35$ s. The SLA is the same for all users, as we suppose it is contracted between the (unique) mobile app provider and the cloud provider.

Every job (as a set of tasks) is assigned either to the Local Engine or to the Offloader of the mobile device. A job may also be dropped, if the mobile device estimates that the deadline of the job cannot be met neither locally nor remotely. The decision process is based on the following algorithm, where four offloading conditions are considered:

```
if ((c(T,C,S) >= 0)
     OR !meetDeadlineLocally(job)
     OR !hasSufficientRam(job))
{
  if (meetDeadlineOnCloud(task))
    return CLOUD;
  else
    return DROP;
}
else
  return LOCAL_ENGINE;
```

Importantly, $c(T,C,S)$ is computed using the estimated speedup of the cluster, not the nominal one. This means that $c(T,C,S)$ may vary over time, even if the job statistics do not change.

When a job is marked for offloading, its tasks are sequentially sent to the Cloud. The Cloud's Dispatcher handles tasks in FCFS order, and assigns them to VMs using the SQF strategy. Every VM executes its tasks in FCFS order. We assumed that VMs are provided with a virtual processor with 2 GHz clock.

To evaluate the performance of the MCC system, we measured the following indicators:

- number of active VMs k;
- offloading probability p_{off} (measured as the relative frequency of offloaded jobs);
- percentage of completed tasks;
- percentage of met task deadlines.

Indicators k and p_{off} are plotted versus time, to compare the effectiveness and efficiency of the auto-scaling strategies. A parametric analysis is provided, as well.

The auto-scaling strategy is executed every 300 s, with the purpose to compute the optimal k value and to update the number of active VMs accordingly.

SIC parameters are $\delta \in \{5,10,20\}$, $s \in \{0.1,0.3,0.5,0.7,0.9\}$, $k_{min} = 5$ and $k_{max} = 120$. Parameter δ directly influences the gain L of the controller. Parameter δ impacts on system responsiveness and stability. In theory, considering the worst possible t_{sc} corresponding to the largest input image, setpoint $s = 0.5$ is the one that meets the $T_{th} = 35$ s constraint, forcing the situation in which, for each VM, the task waiting time is equal to the task service time, *i.e.*, the maximum queue length is 1. Higher s values correspond to less restrictive configurations.

GO parameters are $\gamma \in \{0.1,0.5,0.9\}$, $T_{th} = 35$ s, $F(k) = \beta k + F_0$ with $F_0 = 10$ and $\beta = 10$, $F_{max} = 1210$ (thus k's upper limit is 120, like for the standard integral controller). Such a γ choice allows to study different scenarios, with emphasis on cost ($\gamma = 0.9$), latency ($\gamma = 0.1$) or both ($\gamma = 0.5$). The task dispatching rate is assumed to be high (μ_D=100), while the task service rate μ_{VM} is evaluated at runtime.

4.1 Simulation Results

All simulations have been performed on a MacBook Pro with 16 GB of 1067 MHz DDR3 RAM and a 2.4 GHz Intel Core 2 Duo processor, and repeated with 25 different seeds for the random number generator.

Initially, we analyzed the behavior of the MCC system with a stationary workload (800 mobile devices producing an overall job rate of 1.5 job/s), considering a time period spanning 24 hours.

Figure 5 illustrates the time variation of the number of active VMs and offloading probability, when SIC with $\delta = 10$ is adopted. Three different setpoints are considered, namely $s \in \{0.1, 0.5, 0.9\}$. The analysis is completed by Table 2, showing the percentages of completed tasks, met task deadlines and number k of activated VMs averaged over time. The configuration with $\delta = 10$ and $s = 0.1$ outperforms the others, in terms of percentage of met task deadlines, at the expense of a higher number of active VMs. Setpoint $s = 0.5$, instead, does not allow to achieve an acceptable percentage of met task deadlines. Importantly, the convergence to the steady-state values of average number of activated VMs and offloading probability is not affected by the initial conditions. Given a setpoint, we observed that the average number of active VM with $\delta = 5$, $\delta = 10$ and $\delta = 20$ is the same. However, with $\delta = 20$ the Cloud is less stable, meaning that the number of active VMs changes too much over time, leading to worst performance and offloading probability, consequently. We omit the graphs related to $\delta = 5$ and $\delta = 20$ because of lack of space.

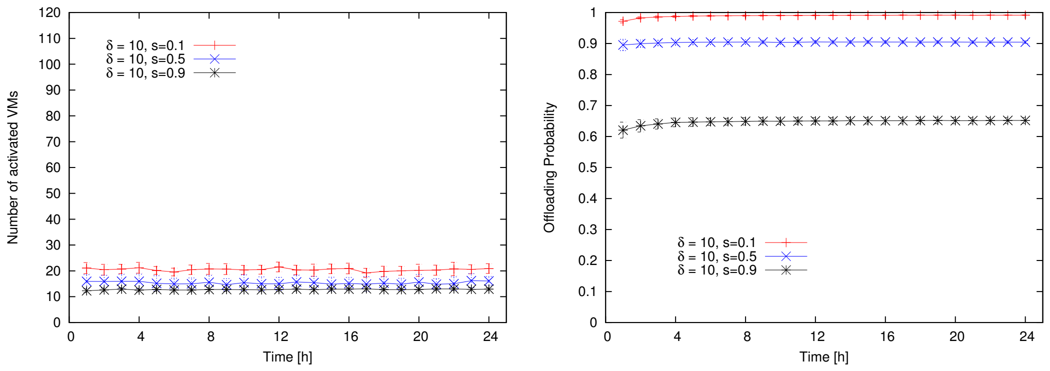

Figure 5: Time variation of the number of active VMs and offloading probability, when SIC with $\delta = 10$ s is adopted.

Table 2: Task statistics vs. k, with SIC

Configuration	% completed	% met deadlines	time-avg. k
$s = 0.1, \delta = 10$	99.89 ± 0.01	99.32 ± 0.16	20.48 ± 1.94
$s = 0.3, \delta = 10$	99.58 ± 0.03	87.74 ± 0.7	16.73 ± 2.38
$s = 0.5, \delta = 10$	98.84 ± 0.07	65.78 ± 1.24	15.35 ± 2.53
$s = 0.7, \delta = 10$	97.81 ± 0.08	41.67 ± 0.84	14.55 ± 2.38
$s = 0.9, \delta = 10$	96.77 ± 0.09	31.89 ± 0.88	12.78 ± 0.67

Figure 6 illustrates the time variation of the number of activated VMs and offloading probability, when the global optimization strategy with task latency threshold $T_{th} = 35$ s is adopted. As expected, such a latency constraint does fits with any γ values. In other words, if the emphasis is on reducing cost for the cloud administrator (with $\gamma = 0.9$), then the cloud becomes less attractive for users ($p_{off} = 0.81$). Conversely, if the emphasis is on reducing latency (with $\gamma = 0.1$), the offloading probability is higher ($p_{off} = 0.98$). The analysis is completed by Table 3, showing the percentages of completed tasks, met task deadlines and number k of activated VMs averaged over time. It is worth noting that the most unstable cloud behavior, corresponding to ($\gamma = 0.1, T_{th} = 35$), appears to be the one with best performance, from the users' point of view. Not surprisingly, the best tradeoff is achieved with the ($\gamma = 0.5, T_{th} = 35$) configuration. Importantly, the behavior of the system is totally unaffected by the initial number of activated VMs, as the GO strategy always explores all the possible k values, to find the one that minimizes the objective function.

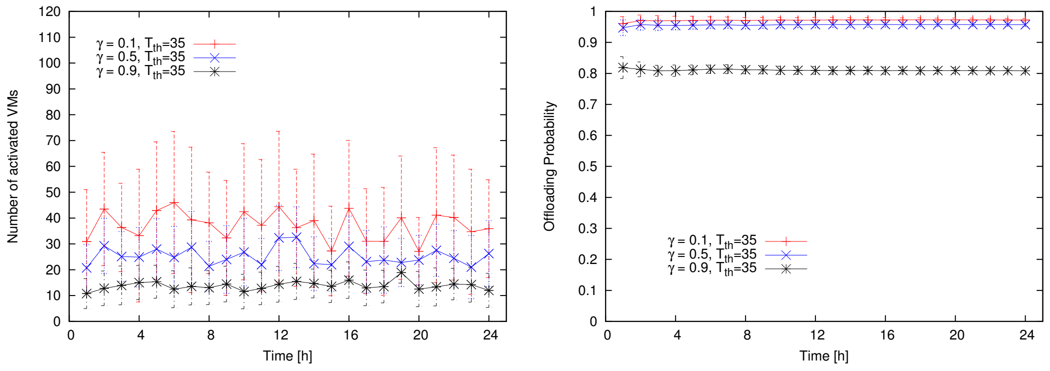

Figure 6: Time variation of the number of active VMs and offloading probability, when the global optimization strategy with latency threshold $T_{th} = 35$ s is adopted.

Table 3: Task statistics vs. k, with GO

Configuration	% completed	% met deadlines	time-avg. k
$T_{th} = 35, \gamma = 0.1$	99.68 ± 0.06	94.48 ± 0.97	37.26 ± 23.06
$T_{th} = 35, \gamma = 0.5$	99.51 ± 0.07	91.69 ± 0.94	25.27 ± 11.28
$T_{th} = 35, \gamma = 0.9$	97.74 ± 0.1	71.63 ± 1.35	13.81 ± 6.56

Next, we compared the two auto-scaling strategies with respect to a dynamic workload, where the job rate doubles after 8 hours (becoming (3 job/s)) and returns to the initial value after 16 hours. In Figure 7, the results related to the best configurations of the two auto-scaling strategies are illustrated. In detail, regarding SIC we considered $(\delta, s) = (10, 0.1)$, $(\delta, s) = (10, 0.5)$, $(\delta, s) = (10, 0.9)$ and $(\delta, s) = (5, 0.1)$, while for the GO approach we considered $(T_{th}, \gamma) = (35, 0.1)$, $(T_{th}, \gamma) = (35, 0.5)$ and $(T_{th}, \gamma) = (35, 0.1)$. The analysis is enriched by Table 4, showing the percentages of completed tasks and met deadlines. The best result in terms of performance, cost savings and stability is provided by SIC with $(\delta, s) = (10, 0.1)$. We observe that $\delta = 5$ does not allow for a responsive SIC, leading to a waste of resources. The GO strategy provides acceptable performance results only when γ is very low, at the expense of high costs and reduced stability. Interestingly, when $\gamma = 0.9$, performance is so bad that the abrupt decrease of p_{off} reduces the number of required VMs, in countertrend with respect to the other GO scenarios.

Table 4: Task statistics, comparing SIC to GO

Configuration	% completed	% met deadlines
$s = 0.1, \delta = 10$	99.86 ± 0.01	98.44 ± 0.2
$s = 0.5, \delta = 10$	98.78 ± 0.05	64.7 ± 0.92
$s = 0.9, \delta = 10$	96.52 ± 0.16	35.05 ± 0.71
$s = 0.1, \delta = 5$	99.91 ± 0.01	99.27 ± 0.34
$T_{th} = 35, \gamma = 0.1$	99.77 ± 0.04	96.15 ± 0.59
$T_{th} = 35, \gamma = 0.5$	99.6 ± 0.06	93.04 ± 1.09
$T_{th} = 30, \gamma = 0.9$	95.05 ± 0.14	63.19 ± 0.72

In summary, despite GO's advantage of enabling to select between high performance and reduced cost, SIC automatically provides the best tradeoff. However, it is worth noting that SIC's parameters δ and s have to be carefully set, as they have a high impact on the final result.

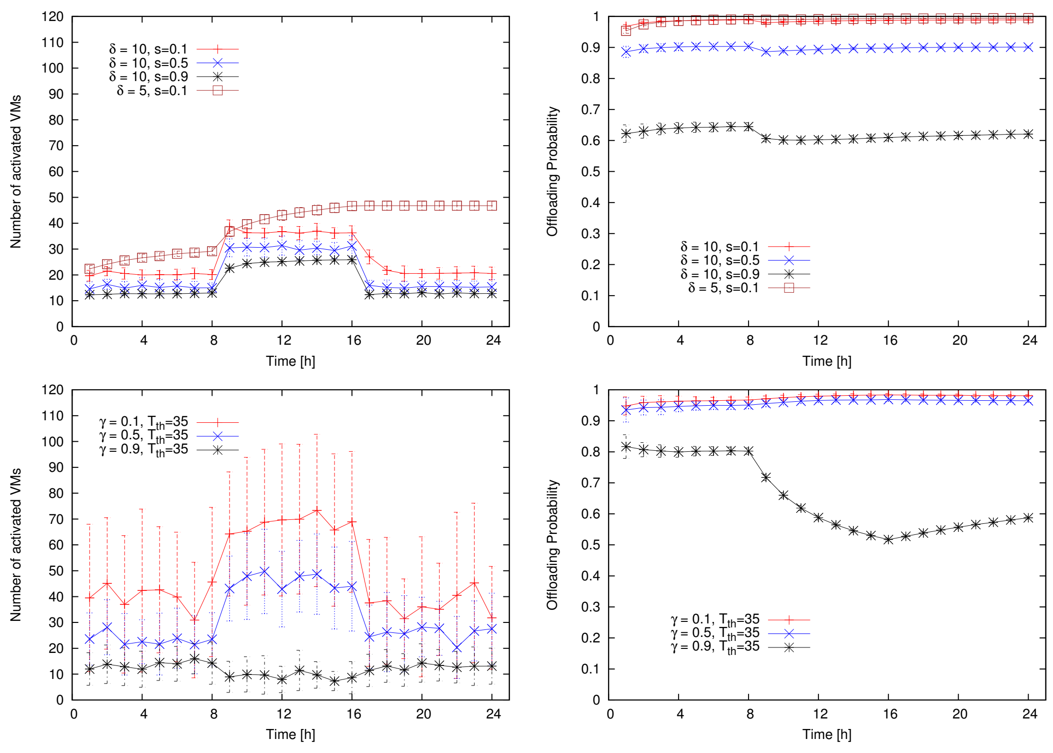

Figure 7: Time variation of the number of active VMs and offloading probability, considering a dynamic workload. Upper graphs refer to SIC, while lower ones refer to GO.

5 Related Work

In Section 2, we recalled the reference equation introduced by Kumar *et al.* [6], representing the energy costs for a mobile device that is enabled to offload tasks. The GO approach to cloud auto-scaling presented in Section 3 extends the one recently proposed by Jiang *et al.* [7]. Other references are discussed in the following.

Liang *et al.* [26] proposed an adaptive resource allocation strategy, based on a semi-Markov decision process, opposed to the traditional greedy approach. By considering the benefits and expenses of both Cloud and mobile devices, the proposed strategy is able to dynamically allocate different numbers of VMs to mobile applications, to obtain the maximal system rewards and to achieve various QoS levels for mobile users. Our approach has similar purposes, and performance indicators, although we consider parallelizable jobs and we provide a more detailed characterization of the Cloud, in terms of task dispatching and scheduling policies.

More recently, Netto *et al.* [27] evaluated reactive, conservative and predictive auto-scaling strategies. To this purpose, they introduced a new performance metric, namely the Auto-scaling Demand Index (ADI), which penalizes differences between actual and desired resource utilization levels. The proposed approach is interesting, although the adopted Cloud model is a bit simplistic, laking details about the internal structure of the system (dispatching policies, scheduling policies). Furthermore, auto-scaling is approached in terms of VM utilization only, without reference to SLA constraints (*e.g.*, latency) and VM costs, in general.

SLA-aware resource management is frequently formulated as an optimization problem. Cardellini *et al.* [15] defined a non-cooperative game-theoretic offloading strategy, considering a challenging three-tier MCC architecture which consists of mobile devices, cloudlets, and Cloud. However, the decentralized strategy proposed by Cardellini *et al.* does not involve adaptive loops between mobile devices and servers. For this reason, it is difficult to compare the two approaches quantitatively.

6 Conclusion

In this paper we have illustrated the comparative analysis of two different cloud auto-scaling strategies, in the context of an MCC system. One strategy, denoted as SIC, takes into account performance objectives only, while the other one, denoted as GO, considers also energy costs. Despite its apparent simplicity, the SIC strategy automatically provides the best tradeoff in terms of cloud stability, offloading probability and performance.

Regarding future work, we plan to further improve our study of MCC systems based on the adaptive loop approach, taking into account also the following topics: VM consolidation, communicating VMs, task workflow management. Last but not least, we are interested in extending the adaptive loop approach to the case of three-tier architectures, with cloudlets between mobile devices and cloud data centers [15].

References

[1] Z. Sanaei, S. Abolfazli, A. Gani, R. Buyya, Heterogeneity in Mobile Cloud Computing: Taxonomy and Open Challenges, IEEE Comm. Surveys Tutorials, vol. 16, no. 1, pp. 369–392, 2014.

[2] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, MAUI: making smartphones last longer with code offload, in: ACM 8th Annual International Conference on Mobile Systems, Applications and Services (MobiSys'10), San Francisco, CA, USA, 2010.

[3] B. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, CloneCloud: Elastic execution between mobile device and cloud, in: The European Professional Society on Computer Systems (EuroSys'11), Salzburg, Austria, 2011.

[4] S. Kosta, A. Aucinas, P. Hui, R. Mortier, Z. Xinwen, ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading, in: IEEE INFOCOM, Orlando, FL, USA, 2012, pp. 945–953.

[5] S. Abolfazli, Z. Sanaei, E. Ahmed, A. Gani, R. Buyya, Cloud-Based Augmentation for Mobile Devices: Motivation, Taxonomies, and Open Challenges, IEEE Communications Surveys Tutorials, vol. 16, no. 1, pp. 337–368, 2014.

[6] K. Kumar, Y.-H. Lu, Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?, Computer, vol. 43, no. 4, pp. 51–56, 2010.

[7] J. Jiang, J. Lu, G. Zhang, G. Long, Optimal Cloud Resource Auto-Scaling for Web Applications, 13th IEEE/ACM Int.'l Symp. on Cluster, Cloud, and Grid Comp., Delft, Netherlands, May 2013.

[8] T. Lorido-Botran, J. Miguel-Alonso, J. A. Lozano, *A Review of Auto-scaling Techniques for Elastic Applications in Cloud Environments*, J. of Grid Computing, vol. 14, no. 2, pp. 559–592, 2014

[9] E. Abebe, C. Ryan, *Adaptive application offloading using distributed abstract class graphs in mobile environments*, Journal of Systems and Software, vol. 85, no. 12, pp. 2755–2769, 2012.

[10] T. Verbelen, T. Stevens, F. De Turck, B. Dhoedt, *Graph partitioning algorithms for optimizing software deployment in mobile cloud computing*, Fut. Gen. Comp. Systems, vol. 29, no. 2, 2013.

[11] J. Liu, E. Ahmed, M. Shiraz, A. Gani, R. Buyya, A. Qureshi, *Application partitioning algorithms in mobile cloud computing: Taxonomy, review and future directions*, Journal of Network and Computer Applications, vol. 48, pp. 99–117, 2015.

[12] Y.-H. Kao, B. Krishnamachari, M.-R. Ra, F. Bai, *Hermes: Latency Optimal Task Assignment for Resource-constrained Mobile Computing*, IEEE INFOCOM'15, Hong Kong, 2015.

[13] A. Iosup and M. N. Yigitbasi and D. H. J. Epema, On the Performance Variability of Production Cloud Services, in: 11th IEEE/ACM Int'l Symp. on Cluster, Cloud, and Grid Computing (CCGrid), Newport Beach, CA, USA, 2011.

[14] H. Khazaei, J. Misic, V. B. Misic, Performance Analysis of Cloud Computing Centers Using $M/G/m/m+r$ Queueing Systems, IEEE Trans. on Par. and Distr. Systems, vol. 23, no. 5, pp. 936–943, 2012.

[15] V. Cardellini, V. De Nitto Personè, V. Di Valerio, F. Facchinei, V. Grassi, F. Lo Presti, V. Piccialli, A game-theoretic approach to computation offloading in mobile cloud computing, Mathematical Programming, available online since April 2015.

[16] E. Ahmed, A. Akhunzada, M. Whaiduzzamn, A. Gani, S. H. Ab Hamid, R. Buyya, *Network-centric performance analysis of runtime application migration in mobile cloud computing*, Simulation Modelling Practice and Theory, vol. 30, pp. 42-56, 2015.

[17] M. Amoretti, M. Picone, F. Zanichelli, G. Ferrari, Simulating Mobile and Distributed Systems with DEUS and ns-3, in: Int.'l Conf. on High Perf. Comp. and Sim., Helsinki, Finland, 2013.

[18] I. Moschakis, H. D. Karatza, Performance and Cost evaluation of Gang Scheduling in a Cloud Computing System with Job Migrations and Starvation Handling, in: IEEE Symposium on Computers and Communications (ISCC), Kerkyra, Greece, 2011.

[19] J. G. Ziegler, N. B. Nichols, Optimum settings for automatic controllers, Trans. of the ASME, 64 (1942) 759–768.

[20] A. Carroll, G. Heiser, An analysis of power consumption in a smartphone, in: 2010 USENIX Annual Technical Conference (USENIX ATC '10), Boston, MA, USA, 2010.

[21] A. Rice, S. Hay, Decomposing power measurements for mobile devices, in: IEEE International Conference on Pervasive Computing and Communications (PerCom), Mannheim, Germany, 2010.

[22] C. Yoon, D. Kim, W. Jung, C. Kang, H. Cha, AppScope: Application Energy Metering Framework for Android Smartphones using Kernel Activity Monitoring, in: 2012 USENIX Annual Technical Conference (USENIX ATC '12), Boston, MA, USA, 2012.

[23] L. Ardito, G. Procaccianti, M. Torchiano, G. Migliore, Profiling Power Consumption on Mobile Devices, in: 3rd International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies (ENERGY 2013), Lisbon, Portugal, 2013.

[24] L. Corral, A. B. Georgiev, A. Sillitti, G. Succi, A Method for Characterizing Energy Consumption in Android Smartphones, in: 2nd International Workshop on Green and Sustainable Software (GREENS), San Francisco, CA, USA, 2013.

[25] X. Ma, Z. Deng, M. Dong, L. Zhong, Characterizing the Performance and Power Consumption of 3D Mobile Games, Computer Magazine, vol. 46, no. 4, pp. 76–82, April 2013.

[26] H. Liang, T. Xing, L. X. Cai, D. Huang, D. Peng, Y. Liu, Adaptive Computing Resource Allocation for Mobile Cloud Computing, International Journal of Distributed Sensor Networks, 2013.

[27] M. A. S. Netto, C. Cardonha, R. L. F. Cunha, M. D. Assunção, Evaluating Auto-scaling Strategies for Cloud Computing Environments, 22nd IEEE International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS 2014), Paris, France, September 2014.