Mobiles on Cloud Nine: Efficient Task Migration Policies for Cloud Computing Systems

Lazaros Gkatzikis
KTH Royal Institute of Technology
Stockholm, Sweden

Iordanis Koutsopoulos
Athens University of Economics and Business (AUEB) and
Centre for Research and Technology Hellas (CERTH), Greece

***Abstract*—Due to limited processing and energy resources, mobile devices outsource their computationally intensive tasks to the cloud. However, clouds are shared facilities and hence task execution time may vary significantly. In this paper, we investigate the potential of task migrations to reduce contention for the shared resources of a mobile cloud computing architecture in which local clouds are attached to wireless access infrastructure (e.g. wireless base stations or access points). We devise online migration strategies that at each time make migration decisions according to the instantaneous load and the anticipated execution time. We explicitly take into account the interaction of co-located tasks in a server and the cost of migrations. We propose three classes of migration policies, ranging from fully uncoordinated ones, in which each user or server autonomously makes its migration decisions, up to cloud-wide ones, where migration decisions are made by the cloud provider. The key underlying idea is that a migration should occur *only if it is beneficial for the processing time of the task, including the migration delay*.**

I. Introduction

Mobile applications (e.g. augmented reality, speech recognition, games) have become more sophisticated than ever in terms of computing requirements and data generation. The proliferation though of 4G wireless access technologies, such as LTE, that support high communication rates and Quality of Service (QoS) enables mobile devices to offload their computationally intensive tasks to the cloud. The later asset has given rise to what is known as *mobile cloud computing* (MCC).

We consider the architecture depicted in Fig. 1 that brings into stage cloud facilities together with mobile wireless devices. Mobile devices access the cloud through readily available hubs like 4G/LTE base stations (BS) or WiFi access points. Cloud servers are attached to the points of wireless access, forming local clouds that are directly accessible by mobile users and hence avoid the additional communication delay of the Internet.

Virtualization is a key ingredient of MCC that enables agile consolidation and parallel execution of diverse mobile applications on the same physical machine (server), each hosted in a separate Virtual Machine (VM). The problem that arises is how to best leverage cloud resources, in particular computation capacity, so as to minimize execution time of mobile tasks. We suggest that this goal can be efficiently realized through VM migration policies that enable dynamic reconfiguration of the cloud. The key underlying idea is that a migration should occur *only if it is beneficial for the processing time of the task, including the migration delay*.

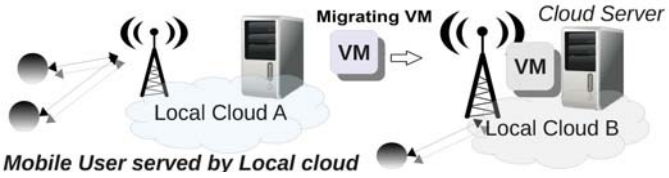


Fig. 1. Mobile Cloud Computing architecture

In order to demonstrate the potential of VM migrations, we provide the following simple motivating example. Consider a mobile CPU intensive task of 10^{10} CPU cycles. Initially the user uploads the task for execution to its current cloud server of available CPU capacity 10^6 cycles/sec. Once the task has been completed by 50% the user moves in range of a local-cloud/BS of double the available CPU capacity, 2×10^6 cycles/sec. If the task stays on the initial server it will be completed after 5000 sec or 1.39 hours. Instead, given an interconnection link of 10Mbps between the two BSs and a current data volume of 10^{10} bits a migration to the new server will cost 1000 sec, and another 2500 sec for execution. Thus, the optimal strategy is to migrate the task to the local server immediately, leading to a completion time of 3500 sec or 0.97 hours, i.e. an improvement by $(1.39 - 0.97)/1.39 \times 100\% \simeq 30\%$.

A. Mobile Cloud Computing challenges

Efficient management of the cloud infrastructure is a challenging task due to the highly dynamic and unpredictable nature of the system.

First, the pattern of instantaneous resource demand varies with time and location as new tasks arise continually at various locations, while others complete service. Second, available processing capacity is varying with time, due to the unpredictable effect of coexistence of VMs on the same physical server and their varying patterns of access to shared resources (e.g. CPU, caches, network, I/O) at different phases of their execution. This results in a multitenancy overhead that is task-dependent and difficult to model or predict.

Third, tasks are usually accompanied with a time-varying data volume, that may be increasing, decreasing or constant with time, depending on whether the task generates new information, discards some data as processing proceeds or does not modify data respectively. The time required to migrate a task to another server depends decisively on this data volume.

Finally, user mobility introduces additional challenges, since

at each time instant a user has direct access only to a certain subset of servers, namely those attached to her current BS. In general, communication with a remote cloud server takes place over WAN connections and hence introduces significant delay. Thus, migrations should be such that the task "follows" the mobile user, i.e., the server that executes the task should be "close" to the user in terms of anticipated delay for retrieving the results of the task.

B. Related work

Virtualization allows diverse tasks to run over a shared hardware platform. Each task is hosted on its own VM, which provides an isolated execution environment. However, operation over the same physical machine introduces significant contention for shared system resources. The problem of noisy-neighbours, where co-located tenants cause significant and unpredictable performance degradation (a.k.a. multitenancy cost), has been reported by several cloud customers (e.g. [1], [2]) and has even led companies to abandon the cloud [3]. Similar performance limitations have been observed when computationally intensive scientific tasks were executed on existing cloud facilities [4].

Several works attempt to perform analytical [5] or experimental (e.g.[6],[7]) estimation of the multitenancy effect. The former though require a priori knowledge of the resource requirements (CPU, Memory, I/O) of each task, while the latter perform extensive profiling of different types of cloud tasks. However, trace data from the Google cloud facility [8] reveal that significantly diverse tasks exist in the cloud and hence a characterization through profiling is impractical. Thus, we propose that multitenancy effect has to be predicted through online measurements as tasks are being executed.

The issues above motivate the need for *task migration*. Migration mechanisms enable dynamic reconfiguration of clouds and hence have attracted the interest of both the research community and virtualization companies. Indicatively, the capability of VM migration across remote servers [9] was recently incorporated in the latest version of vSphere, the commercial virtualization platform of VMware. In vMotion, which is the migration scheduler of vSphere, migrations are performed so as to keep average CPU and memory utilization balanced across servers. Instead, we propose that migrations should be performed according to execution time as this is estimated at running time.

Recent work [10] provides an experimental evaluation of the performance benefits of migrations, whereas max-weight inspired policies are proposed by [11] to maximize throughput through VM allocation. However, the multitenancy effect, the evolving data footprint and the cost of migration that significantly affect the actual execution time are not considered. In [12], the authors propose a system that monitors resource usage and performs a migration whenever a Service Level Agreement (SLA) is violated for a sustained period. Existing schemes perform the VM assignment/migrations according to the required resources or assume that multitenancy cost is a priori known [13],[14]. Instead, we perform task migrations according to the actual performance of the task as this is observed online, which enables us to capture the impact of multitenancy.

In addition, the impact of mobility has been considered only in the scenario where a single task may migrate to a remote server, instead of being executed locally at the mobile terminal, a mechanism generally known as offloading. The resulting energy and time savings have been analyzed in [15], where a Markovian control framework is proposed. In a similar setting the authors of [16] propose the CloneCloud offloading system. Contrary to these works, we explicitly model the interaction of multiple tasks and take into account the impact of migrations on the cloud system as a whole.

A survey of the challenges and potential of VM migration scheduling in the context of MCC are presented in [17]. Here, we formulate the VM allocation problem and devise mechanisms that can be applied in real systems.

C. Our contribution

In order to adhere to a realistic scenario, we address the *online* version of the problem, in which *information about the system dynamics is not available a priori, but rather it is presented to the migration control engine, just before the control decision is made*. For example, the rate of arrival of new tasks in each server is generally unknown. To counteract this, we propose that from time to time the cloud facility has to be reconfigured through VM migrations that reassign VMs to the cloud servers.

The key contributions of this paper are as follows:

1) We develop lightweight task migration mechanisms that capture the following cloud scenarios (a) a centrally coordinated setup, where all migration decisions are taken by a central migration control engine. This setup represents the cloud-provider objective and rationale. (b) a server-centric setup where a migration control engine resides in each server and taken the migration decisions for tasks on that server. (c) a task-centric one, where the migration is decided autonomously by each task.
2) We develop criteria for migration that factor (i) the anticipated delay to migrate the accompanying data volume between servers, (ii) the anticipated remaining execution time of the task at the new server.
3) We explicitly model the coexistence of several VMs on the same cloud server through an associated performance overhead, which depends on the number and the types of coexisting tasks. This is continuously measured and fed back to the entity that performs task migrations, generally called hypervisor or VM monitor.
4) We capture the impact of mobility on the migration strategy. The main idea is to migrate the task closer to the user as execution approaches its end.

The rest of the paper is organized as follows. In Section II we model interaction of tasks/VMs within the cloud. Section III describes the proposed migration mechanisms that capture mobility and interaction of co-located tasks. Numerical results quantifying the performance of our schemes are presented in Section IV. Section V concludes our study.

II. System Model

A. Cloud architecture

We consider a Mobile Cloud Computing architecture that consists of a set $\mathcal{K}$ of K cloud servers, which are attached to wireless access infrastructure (Fig. 1). Such small-scale local clouds provide processing capacity that is at the proximity of mobile users. For notational simplicity, we assume that each local cloud consists of a single server. Each server/local-cloud j has fixed processing capacity C_j flops.

Any mobile user in range of a BS may directly access the corresponding server. In addition, each server ℓ is connected over the Internet with any other server k through an overlay link of bandwidth $W_{\ell k}$ (in bits/sec).

B. Application tasks

Application tasks are continually generated by mobile users at various locations. We use the term *task lifetime* T_i to refer to the total time that task i spends in the cloud. The task lifetime consists of the following stages:

(i) **Task Upload**: Once a new task is generated by a user, the source code and any input data required for the initialization of the VM are uploaded to the cloud through the wireless link between the user and the corresponding point of wireless access. Then, the task execution is initiated.

(ii) **Execution / Migration**: This phase amounts to the actual processing within the cloud. A task may be transferred from its current server to a new one to continue execution there. This process is known as task migration and may occur multiple times during task execution.

(iii) **Download**: In this phase the task is completed and the mobile user retrieves the final results through its current BS. If the task host server is not in range of the mobile user, data have to be transferred to a server that is accessible by the user.

Task i is characterized by its total processing requirements B_i (in flops, or CPU cycles) and carries with it a progress indicator $x_i \in [0,1]$ that denotes the percentage of completion. Task i is also accompanied by an evolving volume of data in bits, $d_i(x_i)$. This pattern may be increasing, decreasing or constant, modeling different types of applications that generate new data or compress it as they evolve. We refer to this accompanying volume of data as the *data footprint* of the task. A typical CPU-intensive task of decreasing footprint is video compression. Starting from an initial raw video of several gigabytes we end up with a compressed video of hundreds megabytes. On the other hand, many tasks such as decompression of a data archive is characterized by an increasing data footprints, since new data are continuously produced.

For a task i that is executed at server j, and that presumably uses its entire capacity C_j, it would take B_i/C_j time to execute, and hence the progress indicator would evolve as $x_i(t) = \min\left\{1, \frac{C_j}{B_i}t\right\}$. The remaining processing requirement for the task is given by $b_i(x_i) = B_i(1-x_i)$ while its data footprint evolution is given by $d_i(x_i)$.

C. Virtualization in cloud servers

A VM provides an isolated environment for the execution of a single task. Without loss of generality, we assume that each task consists of a single VM and that each user generates a single task. Hence, we may use the same index i to refer to a user, his task and the corresponding VM. Upon migration of a task, a new VM is created for this task on the destination server and the execution starts, while the VM at the server where the task executed initially is ceased and removed.

Although there does not exist a constraint on the maximum number of VMs that can be co-located on the same server, *this coexistence intuitively affects the execution performance experienced by each VM*. This is because of two factors : (i) the fact that resources in the physical machine need to be shared among multiple VMs, (ii) coexistence of VMs leads to a graceful reduction in effective available processing capacity, due to overhead induced by the interaction of VMs in an effort to coordinate resource allocation. This overhead strongly depends on the number and types of VMs (tasks) that reside on the same server. For instance, the overhead from 3 coexisting CPU-intensive VMs is larger than that for 2 CPU-intensive VMs. Also the overhead for 2 CPU-intensive VMs is different than the one we would have for 1 CPU-intensive VM and 1 memory-intensive VM.

We model this VM interdependence as follows. Let $\mathcal{A}_j(t)$ denote the set of active tasks of server j at some time t. For any server j with processing capacity C_j and $|\mathcal{A}_j| = n$, we assume that an amount of its processing capacity $\varepsilon(n)$ is effectively lost due to the multitenancy effect, i.e. due to cross-VM management and contention. To model this impact of multitenancy, $\varepsilon(n)$ is taken to be increasing in number of VMs n. We assume that the remaining capacity is equally shared among co-located VMs. Hence, each task on server j obtains an effective share of processing capacity:

$$c_j(n) = \frac{C_j}{n} - \varepsilon(n)\,. \tag{1}$$

Since the effective processing share that each VM enjoys is unknown, due to unknown parameter $\varepsilon(n)$, we propose a mechanism for estimating this amount of overhead and feeding it back to the software entity that is responsible for management of cloud resources and migration decisions, generally referred to as *hypervisor*. Fix attention to computing the overhead $\varepsilon(n)$ due to n coexisting VMs on a server. Let $\mathcal{S}_n$ denote the subset of servers that host n VMs. At each server $j \in \mathcal{S}_n$ and at each decision epoch, we take M measurement samples of the effective processing capacity, $\{C_{jm} : m = 1, \ldots, M\}$ that each hosted task enjoys. CPU consumption measurements are possible by using off-the-shelf cloud monitoring tools like Ganeti [18]. This leads to sample values $\varepsilon_{jm}(n) = \frac{C_j}{n} - C_{jm}$, $m = 1, \ldots, M$ of the multitenancy overhead at server j.

Subsequently, the collected measurements are passed to the hypervisor aggregates these values to a sample mean estimate as follows:

$$\tilde{\varepsilon}(n) = \frac{1}{M|\mathcal{S}_n|} \sum_{j \in \mathcal{S}_n} \sum_{m=1}^{M} \varepsilon_{jm}(n) \tag{2}$$

This quantity serves as an estimate of the overhead for each server with n tasks, and it will be used in migration decisions. In particular, all decisions regarding migrations are performed by the hypervisor based on the estimated share for a server j with n coexisting tasks, given by $\tilde{c}_j(n) = \frac{C_j}{n} - \tilde{\varepsilon}(n)$.

D. Migration

A task may migrate several times to different servers during its execution. The need for task migration arises from the dynamics of the system in terms of new task arrivals, task completions and varying delays due to varying processing speeds. A migration should be performed whenever it improves task lifetime. In order to decide whether a migration is beneficial in terms of reducing the execution time, we compare two delays:

(a) the anticipated execution time at the current server,

(b) the expected completion time at the new server, including the time required to transfer the data volume from the current to the new server.

Assume that task i resides and is executed at server k. Its residual processing requirement is $b_i = B_i(1 - x_i)$ and the amount of accompanying data is d_i. Let there be n_k active tasks on server k. This means that the effective available processing capacity for task $c_k(n_k)$ is given by (1) and the remaining processing time for tentative case of no migration (a) is:

$$D_i^{\mathrm{a}}(n_k) = \frac{b_i}{c_k(n_k)} \tag{3}$$

Next, we determine whether migrating to server ℓ that hosts n_ℓ tasks will lead to shorter task execution time for task i (case (b)). In order to resume execution at server ℓ, the entire volume of accompanying data needs to be moved to the new server over the overlay link of capacity $W_{k\ell}$, incurring a communication cost (delay) equal to $\frac{d_i}{W_{k\ell}}$. If task i moves there, it will receive a processing share of $\tilde{c}_\ell(n_\ell + 1) = \frac{C_\ell}{n_\ell + 1} - \tilde{\varepsilon}(n_\ell + 1)$. Thus, the estimated remaining execution time at ℓ is:

$$D_i^{k \rightsquigarrow \ell}(n_\ell) = \frac{d_i}{W_{k\ell}} + \frac{b_i}{\tilde{c}_\ell(n_\ell + 1)} \tag{4}$$

Notice that throughout the paper, we use the superscript $^{\mathrm{a}}$ to refer to migration case (a), while case (b) is denoted by the superscript $^{k \rightsquigarrow \ell}$ indicating a task migrating from server k to ℓ. Notice that a migration is beneficial only if the new server can provide significantly better effective processing speed so as to accommodate migration delay.

E. Mobility

The mobility pattern of user i can be represented as a sequence of servers $\{j_i(t)\}$ that denotes her current point of wireless access for each timeslot t. As the user moves from one BS to another, a different subset of servers is directly accessible, namely the ones comprising the local cloud of the corresponding BS. A task is *completed*, once its data have been transferred to the mobile user. If the VM is completed in a remote server k, the final data have to be transferred to the user's current point of access j_i. Thus, for mobile users we need also to include the corresponding delay in migration decisions. Intuitively, as the task proceeds to its completion, the migration strategy should favor the local clouds that are closer to the mobile user.

For the no-migration case (a) the remaining lifetime of task i hosted at server k becomes:

$$T_i^{\mathrm{a}}(n_k) = D_i^{\mathrm{a}}(n_k) + \mathbb{1}_{\{k \neq j_i\}} \frac{d_i(1)}{W_{kj_i}}, \tag{5}$$

where $D_i^{\mathrm{a}}(n_k)$ is given by (3) and $d_i(1)$ is the volume of the accompanying data once the execution is completed, i.e. for $x_i = 1$. Notice that for the calculation of total lifetime in the cloud, we have assumed that user i will be at her current point of wireless access, namely j_i.

In order to decide whether migration of task i to server ℓ is beneficial, we derive the following metric capturing the expected lifetime in case of migration case (b):

$$T_i^{k \rightsquigarrow \ell}(n_\ell) = D_i^{k \rightsquigarrow \ell}(n_\ell) + \mathbb{1}_{\{\ell \neq j_i\}} \frac{d_i(1)}{W_{\ell j_i}} \tag{6}$$

where $D_i^{k \rightsquigarrow \ell}$ is given by (4) while the second term captures the delay cost of transferring the final results to the current point of access of user i.

III. Algorithms for Efficient Task Migration in the Cloud

The execution time of a task in a cloud computing system depends on a number of dynamic and unpredictable parameters, such as task arrivals, user mobility and the effect of multitenancy which are revealed to the hypervisor in an online manner. We consider a model where migration decisions are taken in discrete time intervals. A period of 5 minutes is typical in commercial virtualization platforms [9]. We propose three online migration mechanisms. Our schemes make use of control messages that are circulated within the cloud, including the number of tasks running on each server and an estimate of the multitenancy overhead parameter $\tilde{\varepsilon}(n)$. Using this information the estimated processing share per task $\tilde{c}_k(n_k)$ at any server k can be computed.

A. Cloud–wide task migration

Consider the *fully coordinated scenario*, where the cloud provider needs to decide which tasks to migrate and where. Consider a task under tentative migration from its current server to a new one. Given that co-located tasks compete for the same resources, the migration affects the performance of all tasks running at the current and the new server. Thus, a migration should be considered only if it is beneficial for *the system as a whole*.

Under processor sharing, each task receives a portion of processing capacity inversely proportional to the number of tasks running on this server. Thus, a migration improves the performance of co-located tasks in the host server. Fewer tasks have to share the available physical resources, which also leads to reduced multitenancy cost. On the other hand, the addition of a new task at the destination server would lead to a larger number of tasks sharing a given processing capacity, while the multitenancy overhead would also increase. Finally, the residual execution time of the task under migration may increase or decrease, depending on the load at the host and

the destination server, the capacity of the interconnecting link and the volume of accompanying data.

At each decision epoch, all active tasks are candidates for migration to any other server. We focus on task i currently hosted by server k and consider the impact of its migration to server ℓ. Initially, the lifetime (execution + migration time) of the involved tasks is used as the performance metric for the case of no migration. This is given by:

$$M^{\mathrm{a}} = \sum_{l \in \mathcal{A}_k} T_l^{\mathrm{a}}(n_k) + \sum_{l \in \mathcal{A}_\ell} T_l^{\mathrm{a}}(n_\ell). \quad (7)$$

Both terms come from (5) and capture the total lifetime at each of the servers k, ℓ if no migration is performed.

Next, the remaining lifetime is estimated for the case of migration of task i to server ℓ:

$$M_i^{k \rightsquigarrow \ell} = \sum_{l \in \mathcal{A}_k \setminus \{i\}} T_l^{\mathrm{a}}(n_k - 1) + T_i^{k \rightsquigarrow \ell}(n_\ell) + \sum_{l \in \mathcal{A}_\ell} T_l^{\mathrm{a}}(n_\ell + 1). \quad (8)$$

The first term corresponds to the improved lifetime of tasks at host server and is given by (5), while the second term is the expected lifetime of the migrating task described by (6). The third term captures the degraded performance at the destination server ℓ and is given by (3). The difference in performance $M^{\mathrm{a}} - M_i^{k \rightsquigarrow \ell}$ if positive, indicates the reduction in total execution time of tasks by virtue of migration of task i from server k to server ℓ.

Since a migration to any other server is possible, this calculation has to be carried out for each possible destination server ℓ. Finally, out of all the possible migrations the one of maximum reduction is performed. This process is repeated until no more beneficial migrations can be found. This online mechanism is executed in every epoch. Within an epoch new tasks arrive while others complete execution. In general, the algorithm intuititively reassigns tasks from overloaded servers to underutilized ones. However, the selection of the task to migrate is performed by jointly considering the following guidelines:

- Migrations of tasks with increasing data volume pattern are given priority, since the migration cost of any such task increases as its execution proceeds.
- Tasks of significant residual processing burden are preferred to migrate since the benefit of a migration is an increasing function of the remaining burden of the task. A task of substantial remaining processing time can exploit the available capacity at the destination server more efficiently, while migrating a task that is close to completion, may not be beneficial even if the destination server is idle.
- Tasks that experience significant multitenancy cost are selected for migrations. Although this cost is generally increasing in the number of co-located tasks, its exact impact depends also on the type of co-located tasks.
- prefer servers that are closer to the user, as a task moves towards completion.

B. Server-centric task migration

Next, we devise a migration mechanism that is of lower complexity than the first one. Each server periodically checks whether the execution time of its active tasks can be improved through a migration to a new server and autonomously selects which of its active tasks should migrate and where. The anticipated reduction in execution time for each possible migration to any new server needs to be estimated, which for task i migrating from server k to server ℓ is defined as:

$$L^{\mathrm{a}} - L_i^{k \rightsquigarrow \ell} = \sum_{l \in \mathcal{A}_k} T_l^{\mathrm{a}}(n_k) - \sum_{l \in \mathcal{A}_k \setminus \{i\}} T_l^{\mathrm{a}}(n_k - 1) - T_i^{k \rightsquigarrow \ell}(n_\ell)$$

In this case, we consider the total reduction in execution time of the tasks hosted by server k. The first term is the total execution time for the case of no migration, while the second term is the total execution time of the $n_k - 1$ tasks remaining at server k and the third term corresponds to the expected execution time of migrating task i in the new server ℓ. Since migrations are initiated by the host server, the impact of the migration on the tasks located at the destination server is unknown and hence not considered. The migration of maximum gain is performed. This is repeated until no more migrations of positive expected gain can be found.

This algorithm requires no synchronization among servers, in the sense that each server may autonomously decide when to check for beneficial migrations. This could be triggered whenever a server considers itself overloaded, e.g. compared to the average server load of the cloud.

C. Task-centric migration

In contemporary cloud systems, migration decisions are made by the cloud provider. However, each task/user may autonomously decide its migration strategy towards minimizing its own execution time. Next, we consider the scenario of a user that may offload its task to one out of several possible cloud providers/servers, but is only aware of the advertised capacity of each.

Due to multitenancy the effective capacity of each server k at timeslot t is lower than the advertised one. The actual capacity is only revealed once a task migrates to a server. Thus, we propose the following heuristic. From time to time, any task i may autonomously check whether a tentative migration from its current server k to a new one, say ℓ, would decrease its estimated lifetime, i.e. whether $T_i^{k \rightsquigarrow \ell}(n_\ell) < T_i^{\mathrm{a}}(n_k)$. Then, the task migrates to the server that results in the maximum reduction in task lifetime.

IV. Numerical Evaluation

In order to compare the performance of the proposed schemes, we use an event-driven simulator for a cloud system of 50 servers, each of capacity $C \in [0.1, 2]$ Tera–flops, interconnected through wireline communication links of mean available link capacity $W \in [1, 10]$ Mbps. The link and processing capacities are assumed to be i.i.d. across time.

New tasks arrive at each server i according to an inhomogeneous poisson arrival process of parameter $\lambda_i(t)$ arrivals/sec. The processing burden follows a heavy tailed pareto-like probability density function (pdf) with $P[B_i > x] = \min\{1, (1/x)^{1.25}\}$, where x is in teraflops, which is typical for CPU intensive tasks [19].

We consider tasks of increasing or decreasing data foot-

print evolving according to $d_i(x_i) = \min\{0, \alpha x_i + \beta\}$ with $\alpha \sim U(-10, 10)$, $\beta \sim U(0, 100)$ for the increasing and $\beta \sim U(10, 1000)$ for the decreasing ones (in MBytes). The multitenancy cost is modeled as a Gaussian random variable with mean $\mu = \frac{k-1}{2k}$ and variance $\sigma^2 = \frac{1}{16k^2}$. Throughout this section, we will use the no-migration strategy and a one-shot placement scheme as performance benchmarks. Placement follows the rationale of our cloud-wide approach, but now the server that will host the task is selected only once, when the task arrives at the system. The depicted values are averages over 100 instances.

Initially, we investigate the population of tasks N hosted by each server, which indicates the load balancing behaviour of each algorithm. Since N is a random variable, we depict in Fig. 2(a) its cumulative distribution function (cdf), i.e. the probability $P[N \leq k] \ \forall k \in \mathbb{N}$. The slopes of the curves indicate how balanced the cloud is. In the cloud-wide approach a server hosts at most 3 VMs, while the probability of having more than 3 VMs running on a server is 10% for the server-level approach and 25% for the task-initiated one. In the latter case the probability of having more than 20 tenants in a server is non-negligible ($\sim 4\%$). Nevertheless, compared to the no-migration case the load is more balanced. We depict in Fig. 2(b) the exact distribution of tasks on the servers through the probability mass function (p.m.f.) of N. We see that the cloud-wide migration strategy lead to a more balanced network, where in most of the cases the number of tasks N ranges from 1 to 3. Moving from centrally coordinated to server- and task-centric approaches, we observe that we get higher variance. For example, in the user–centric scheme the probability that a server is empty is significant and even for $N > 20$ the probability is non-negligible.

Next, we consider how the distribution of tasks is affected by the link capacity of the cloud interconnection links. Thus, we depict in Fig. 2(c) the pmf of the cloud wide approach for three different scenarios, where the cloud servers are interconnected by high/medium/low capacity links. As the link capacity decreases the pmf broadens indicating that it is not always optimal to perform a perfect load balancing. This is justified by the fact that in low link capacity regime, the migration cost becomes significant and hence it is preferable to keep the cloud unbalanced.

In Fig. 3(a) we quantify the impact of processing capacity on average lifetime of tasks. As expected, the performance degrades as we move from the centralized approach that has system–wide information to decentralized ones that rely only on local information. Increasing server capacity causes the performance gap of the proposed schemes to diminish, which indicates that careful migration decisions are most important in overcommitted clouds. Interestingly, it is only the task-initiated approach that performs worse than one-shot placement. Our results reveal that although task-initiated approach does not perform much better than no-migration in terms of average lifetime, the migrating tasks benefit significantly.

In Fig. 3(b) we depict the impact of data footprint on average task lifetime. As the mean footprint, increases the performance gap between the proposed algorithms decreases, since the more the data that a task carries, the greater the migration cost is. Finally, we consider the frequency of migrations for each algorithm in Fig. 3(c) for two different scenarios. For tasks of medium data footprint we observe that more migrations are performed, compared to a system serving data-intensive tasks.

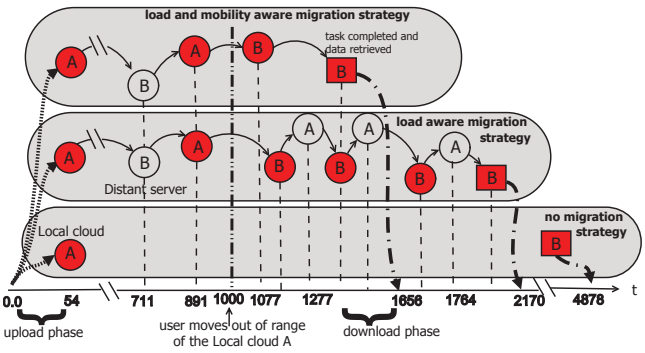
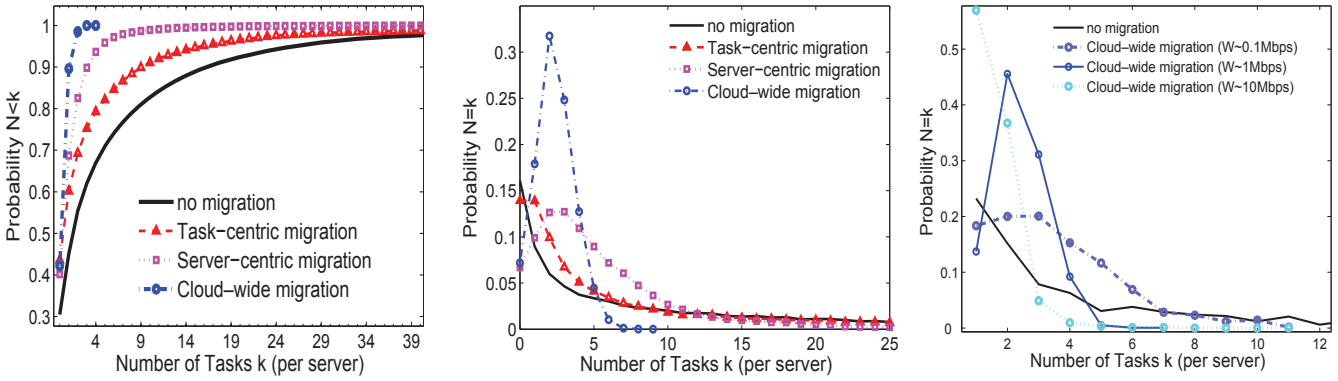


Fig. 4. The lifetime of a mobile task under different migration strategies

In order to stress the impact of mobility in migration decisions, we depict in Fig. 4 the lifetime of a task that is initially uploaded to local cloud A by a mobile user. We consider a task of increasing data footprint that can be executed either at the directly accessible local cloud A or a distant one, say B. We depict the scenarios of a) no migration, b) a strategy that does not consider migration cost and download time and c) the proposed task-initiated migration strategy that is mobility aware. For each time instance, we depict in colour the closest to the user cloud facility in terms of communication delay. In the no-migration strategy execution takes place in local cloud A and hence exhibits the worst performance. Initially, the migration cost is negligible since the data footprint of the task is small. Thus, as long as the user is in range of local cloud A and migrations are costless, the other two migration strategies perform identically. Once the user moves out of range of BS A to a place that is closer to cloud B, the mobility-aware approach moves the task there. In contrast, the load-aware policy constantly migrates the task to the least-loaded server, in an attempt to exploit the available processing capacity, without considering though the increasing cost of each subsequent migration and the additional cost of downloading the final data from a remote server. Hence, it is outperformed by the mobility-aware one.

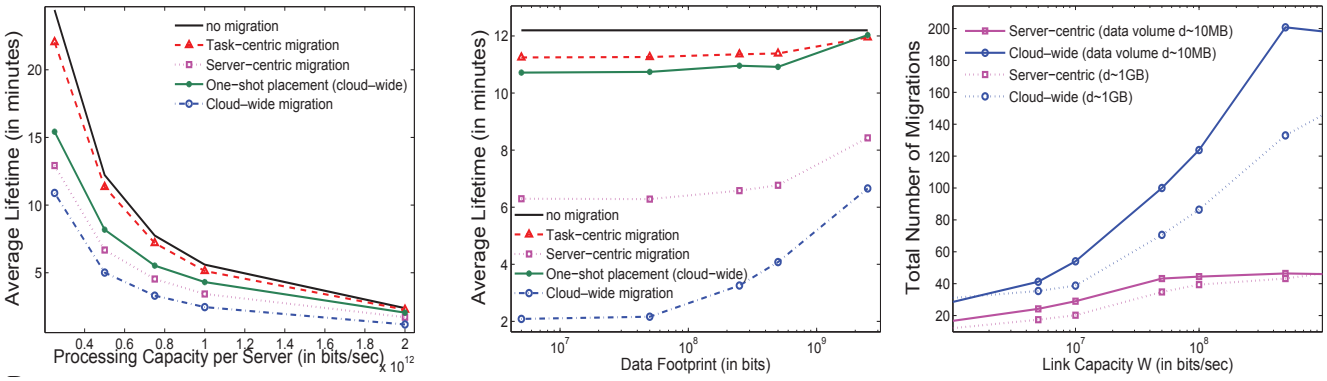
V. Conclusion

We considered the problem of minimizing execution time of tasks in the cloud. We developed migration policies that opportunistically exploit available processing capacity at cloud servers. Our techniques explicitly take into account into account the interaction of co-located tasks in a server and the cost of migrations. In contrast to the migration policies currently applied in commercial virtualization platforms like vSphere [9], we demonstrate that migrations should be performed based on the criterion of estimated execution time and not based



(a) cdf of the number of tasks per server (b) pmf of the number of tasks per server (c) pmf for different link rates

Fig. 2. Load balancing behaviour of the proposed algorithms



(a) Available processing capacity (b) Data footprint (c) Available link capacity

Fig. 3. The impact of system/task parameters on cloud performance

on the requested resources of each task/VM. In this work, we followed an algorithmic approach to the problem of task migration. We are currently in the process of implementing these techniques in a real cloud cluster of tens of nodes and we expect to report findings in future work.

VI. Acknowledgements

This work was supported by ERC08-RECITAL project, co-financed by Greece and the European Union (European Social Fund) through the Operational Program "Education and Lifelong Learning" - NSRF 2007-2013.

References

[1] "Rethink it: Getting rid of noisy neighbours," http://blogs.vmware.com/rethinkit/2010/09/getting-rid-of-noisy-cloud-neighbors.html.

[2] "Has amazon ec2 become over subscribed?" http://alan.blog-city.com/has_amazon_ec2_become_over_subscribed.htm.

[3] "Mixpanel: Why we moved off the cloud," http://code.mixpanel.com/2011/10/27/why-we-moved-off-the-cloud/.

[4] A. Iosup, S. Ostermann, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema, "Performance analysis of cloud computing services for many-tasks scientific computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 6, Jun. 2011.

[5] S.-H. Lim, J.-S. Huh, Y. Kim, G. M. Shipman, and C. R. Das, "D-factor: a quantitative model of application slow-down in multi-resource shared systems," in *ACM SIGMETRICS/PERFORMANCE 2012*.

[6] X. Pu, L. Liu, Y. Mei, S. Sivathanu, Y. Koh, and C. Pu, "Understanding Performance Interference of I/O Workload in Virtualized Cloud Environments," in *IEEE CLOUD 2010*, July 2010, pp. 51 –58.

[7] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen, and C. Pu, "An analysis of performance interference effects in virtual environments," in *IEEE ISPASS 2007*, April 2007, pp. 200 –209.

[8] A. K. Mishra, J. L. Hellerstein, W. Cirne, and C. R. Das, "Towards characterizing cloud backend workloads: insights from google compute clusters," *ACM SIGMETRICS*, vol. 37, pp. 34–41, March 2010.

[9] "vSphere Resource Management Guide," http://www.vmware.com/support/pubs/vsphere-esxi-vcenter-server-pubs.html.

[10] K. Srinivasan, S. Yuuw, and T. J. Adelmeyer, "Dynamic VM migration: assessing its risks and rewards using a benchmark," in *ICPE '11*.

[11] S. Maguluri, R. Srikant, and L. Ying, "Stochastic models of load balancing and scheduling in cloud computing clusters," in *Proceedings IEEE INFOCOM 2012*, March 2012, pp. 702 –710.

[12] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Black-box and gray-box strategies for virtual machine migration," in *NSDI'07*.

[13] K. Li, H. Zheng, and J. Wu, "Migration-based virtual machine placement in cloud systems," in *IEEE CloudNet*, 2013, pp. 83–90.

[14] A. Roytman, A. Kansal, S. Govindan, J. Liu, and S. Nath, "PACMan: Performance Aware Virtual Machine Consolidation," in *IEEE ICAC 13*, 2013.

[15] S. Gitzenis and N. Bambos, "Joint task migration and power management in wireless computing," *IEEE Trans. on Mob. Comp.*, vol. 8, no. 9, pp. 1189 –1204, Sept. 2009.

[16] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *EuroSys '11*.

[17] L. Gkatzikis and I. Koutsopoulos, "Migrate or not? exploiting dynamic task migration in mobile cloud computing systems," *IEEE Wireless Communications*, vol. 20, no. 3, 2013.

[18] "Ganeti : Cluster-based virtualization management software," http://code.google.com/p/ganeti/.

[19] M. Harchol-Balter and A. B. Downey, "Exploiting process lifetime distributions for dynamic load balancing," *ACM Trans. on Comp. Syst. 1997*.