Towards Energy-Aware Placement of Real-Time Virtual Machines in a Cloud Data Center

Nima Khalilzad, Hamid Faragardi, Thomas Nolte
MRTC/Mälardalen University, Västerås, Sweden
{nima.m.khalilzad, hamid.faragardi, thomas.nolte}@mdh.se

***Abstract*—Cloud computing is an evolving paradigm which is becoming an adoptable technology for a variety of applications. However, cloud infrastructures must be able to fulfill application requirements before adopting cloud solutions. Cloud infrastructure providers communicate the characteristics of their services to their customers through Service Level Agreements (SLA). In order for a real-time application to be able to use cloud technology, cloud infrastructure providers have to be able to provide timing guarantees in the SLAs. In this paper, we present our ongoing work regarding a cloud solution in which periodic tasks are provided as a service in the Software as a Service (SaS) model. Tasks belonging to a certain application are mapped in a Virtual Machine (VM). We also study the problem of VM placement on a cloud infrastructure. We propose a placement mechanism which minimizes the energy consumption of the data center by consolidating VMs in a minimum number of servers while respecting the timing requirement of virtual machines.**

Keywords—Real-time cloud; energy aware allocation; VM placement.

I. Introduction

Cloud computing is widely referred as the next generation of computing systems in which dynamically scalable and often virtualized resources are provided as services over the Internet [1]. Service sharing and utility computing are the main characteristics of Cloud Computing Systems (CCS), which distinguish CCS from Grid, Cluster computing and other types of distributed systems. Nowadays, a wide range of services are served by the cloud providers such as computational resources for high performance computing applications, web services, social networking, and telecommunications services. Most of these services are real-time in nature, i.e, there are some strict timing requirements referred as deadlines in such services which must be met by cloud providers. The timing constraints are stipulated as one the Service Level Objectives (SLOs) in the Service Level Agreement (SLA). The SLA provides a facility to agree upon minimum requirements between end-users and cloud providers. Other SLOs examples are security or availability. If the SLOs are violated in the sense that the services are not executed within the negotiated Quality of Service (QoS), agreed upon consequences (usually penalty payments) go into effect. Therefore, in order to avoid SLA violation, the cloud providers should do their best to fulfill the SLOs.

Scalability and heterogeneity are two key reasons that make fulfilling timing requirements in cloud data centers a challenging issue. From a scalability perspective, the deadlines of applications must be met not only when few applications are running in the system but it also when a lot of applications are executing in the system. In addition, from a heterogeneity perspective, different applications located on different guest operating systems should be able to meet their deadlines while they may be running on different types of servers in a cloud environment. This complexity should be taken into consideration in three different levels: (i) the scheduler of the guest OS; (ii) the placer component of the hypervisor located on top of servers of a data center (iii) the scheduler of the host OS in each server. The scheduler of a guest OS schedules a set of periodic tasks of a given application. We assume a one-to-one application to VM mapping. The placer component of the hypervisors decide how the provided set of VMs should be placed among the available servers of a data center. The host OS scheduler, on the other hand, schedules the VM executions by allocating the VMs to the cores and running a real-time scheduler per core. In this paper, all these three levels are taken into account as a comprehensive solution to deal with the real-time requirements of applications in cloud environments from a cloud provider's perspective.

Another goal in this work is to consider energy consumption as one of the significant topics in cloud computing systems. There might be several solutions which all of them can meet the application timing requirements but among them the solution consuming the minimum energy is the most desirable one. The amount of money paid by end-users to utilize a service is a considerable factor. If a service supplied by a cloud provider is more expensive than that provided by others, the users may not wish to utilize the services. Therefore, cloud providers should struggle to supply services within the admissible QoS with the minimum possible cost. Energy consumption is a dominant factor which directly affects the cost of services [2]. Hence, in this paper energy consumption is also taken into account besides timing constraints.

Contributions. Although a wide range of studies are carried out to investigate the running of real-time services in the context of cloud computing, the contributions of this paper are: (i) We provide a cohesive solution considering three scheduling levels whereas, most of existing works only consider one of these levels; (ii) We adapt the compositional analysis provided in [3] for abstracting VM specification from task set parameters; (iii) An energy aware placement algorithm is introduced which consolidates VMs in a minimum number of servers.

II. Model

We approach the problem from a cloud provider point of view. The provider is responsible for running a set of real-time applications in a timely manner according to the SLAs. We assume that the applications are supposed to run on a single

The research leading to this paper has received funding from the Swedish Research Council (Vetenskapsrådet) under the project ARROWS and the Swedish Foundation for Strategic Research (SSF) via the research project PRESS.

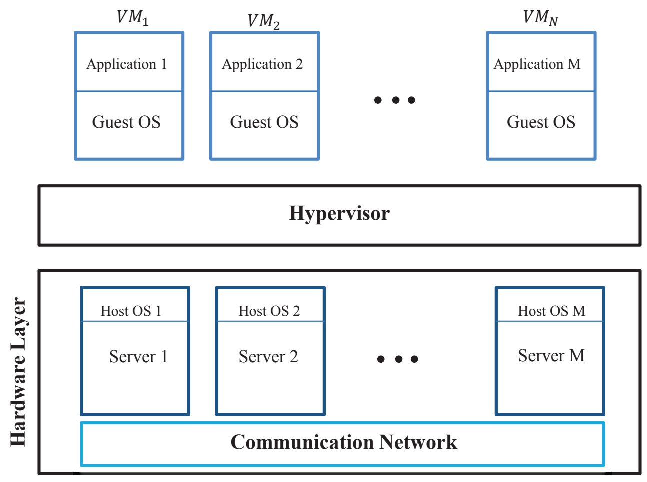

Fig. 1: Target system architecture.

cloud data center with a set of homogeneous servers. The cloud provider is interested in scheduling the application such that (i) the timing requirements are met (ii) memory constraints are satisfied (iii) the minimum number of servers are used so that the remaining servers can be turned off to reduce energy consumption.

Application model. We assume that the provider has received N applications. Each application $\mathcal{A}_i$ consists of n_i real-time tasks τ_j^i where i and j denote the application and task indices respectively. A task is activated periodically with period P_j^i, execution cost C_j^i and deadline D_j^i. The memory requirement of tasks is denoted using h. Periodic workloads are found in multimedia [4] and telecommunication applications that can be provided as a service in the Software as Service (SaS) model. Moreover, embedded devices (e.g, surveillance cameras) may use *Computation Offloading* to execute computational-intensive tasks on the cloud [5]. Online video gaming is another example in which resources have to be provided in a timely fashion [6]. In such applications, one of the main challenges from cloud providers' point of view is inherent in the scheduling of the tasks of applications such that the tasks meet their deadlines while other system requirements are satisfied. In our model, tasks are scheduled using a *task-scheduler*. This scheduling is performed within VMs by the guest operating system. Let's suppose Earliest Deadline First (EDF) as the algorithm for task-scheduling.

Data center model. We assume a data center consisting of M identical servers. Each server applies a multi-core processor with m processing cores. Let's assume, processing cores of a server have identical processing power (homogeneous multi-core). The available memory capacity of each server is denoted by H. Our assumed architecture is presented in Figure 1.

III. Solution

In this section we present our solution. We assume that the applications are submitted to the cloud provider beforehand and that the provider makes placement and allocation decisions off-line. Our approach works in three steps.

1) We derive the specification of each VM using the tasks' parameters within that VM.
2) The next step is to place VMs on servers. The placement algorithm has to take into account both the resource capacity of servers and the resource requirements of VMs. This placement is performed in the hypervisor level. The objective of the VM to server placement is to minimize the number of used servers, for energy saving purposes. In the following the VM placement problem is expressed as an optimization problem and then a heuristic solution is introduced to deal with the optimization problem.
3) Finally, VMs assigned to a server should be allocated among different processing cores on that server. We provide an algorithm for this mapping. VMs may utilize partial bandwidth of a physical core. Hence, multiple VMs may be allocated on a single core at the same time. In our architecture, the VM-scheduler is responsible for coordinating the execution of VMs that share the same physical core. This scheduling is performed in the level of host OS located on physical servers. The VM-scheduler only uses the VM specifications for making scheduling decisions, i.e, the scheduler is unaware of the task set information within the VMs.

Note that we use the phrase "placement" referring to the VM to server placement step. While the word "allocation" is used for VM to core allocation step.

A. *From application to VM specification*

We use global EDF (gEDF) for scheduling tasks within VMs (task-scheduling). gEDF works as follows. At each scheduling instance the scheduler selects a task with the earliest deadline and it assigns it on an idle processor. Since the task-scheduling is performed within VMs, the scheduler selects an idle virtual processor at scheduling points. We are interested in deriving the specifications of VMs given the specification of tasks that are assigned to the VM. We use the analysis framework proposed by Easwaran et. al [3] for calculating the VM specifications. The result of the analysis is the following parameters: (i) the period of the VMs denoted by Π (ii) the total budget that has to be provided within each period (Θ) to the VMs (iii) the minimum number of processors that has to be allocated to a VM ($\lceil \frac{\Theta}{\Pi} \rceil$), and (iv) the maximum number of processors that can contribute in providing the total budget m'. This specification means that the underlying virtualization mechanism has to make sure that the corresponding VM receives Θ units of processor time every Π time units using $\lceil \frac{\Theta}{\Pi} \rceil$ to m' physical processors. Such a mechanism is supported by the RT-Xen hypervisor [7]. It should be noted that VMs may be split on more than their minimum number of processors due to unavailability of the demanded number of processors. The processor utilization of each VM is defined as follows: $u = \frac{\Theta}{\Pi}$. The memory requirement of each application is equal to the sum of memory requirement of its tasks. In summary, the specifications of the i^{th} VM is represented using the following tuple:

$$< \Pi_i, u_i, m'_i, h_i > .$$

The VM specification generation process starts by assuming a period for the VM. The period of the task with the shortest period is often selected as the period of VMs. The parallelism level m' is then selected using a binary search. For each value of m' the smallest budget is selected such that the schedulability of the task within the VM is preserved. Finally, the most efficient interface, i.e, the one with the lowest u, is selected as the VM specification.

To evaluate each candidate solution for the placement of VMs on servers in the problem space we need an energy model. This gives us the ability to calculate the amount of energy consumption of a placement solution. An energy model is suggested in the following.

Energy model. Let's suppose that the given real-time applications running on the system have unlimited lifetime then we attempt to minimize the power consumption of servers to reach energy minimization. However, it is worth noting that minimizing power consumption does not necessarily lead to energy minimization in all situations. For example, if the lifetime of the given workload is limited and the servers are heterogeneous, then power reduction may increase the length of execution time of the given workload which would yield a non-optimal energy consumption.

The power consumption of a host consists of the power consumed by CPU, memory, disk storage and network interfaces. It is shown by [8] that power consumption of the CPU dominates the overall power consumption of a host. Accordingly, power consumption of a host could be achieved based on the CPU utilization by a linear model defined in Eq.1 [9].

$$P(U) = k \times P^{max} + (1-k)P^{max} \times U \tag{1}$$

where $P(U)$ is the power consumption of a host when its CPU utilization is U, P^{max} is the maximum power of a fully utilized host and k is the fraction of power consumed by an idle host. It is cost-effective to turn the host off when its utilization is equal to zero. It should be noted that a host still consumes energy when it is turned off. Thus, it should be taken into consideration to provide a more accurate energy model off-consumption (i.e., the consumption of plugged-host when it is off). It is observed by [10] that off-consumption is 15% of the idle consumption. Hence, with the assumption that a server is always turned off whenever it is idle (utilization is equal to zero) then Eq.2 can model power consumption of the ith host.

$$P_i(U) = \begin{cases} k \times P_i^{max} + (1-k)P_i^{max} \times U_i & U_i > 0 \\ 0.15 \times P_{idle} & U_i = 0 \end{cases} \tag{2}$$

Power consumption of a host. The above mentioned power model is originally proposed for a uni-processor however, most of the common servers use a multi-core processor. There are three ways to adapt the mentioned model for a multi-core processor. The most straightforward way is to consider each core as an independent processor which in this case, the power model could be too pessimistic as the cores are located on the same chip, and therefore their power consumption is significantly lower than a multi-processor where processors do not share the same chip. The second way to apply the mentioned model for a multi-core processor is to assume that a multi-core processor is a single processor with a higher computation capacity which consumes more power. In this case, as we assumed that the multi-core processor in each server is homogeneous (all processing cores are identical) then the total utilization of the workload assigned to this server can be divided by the number of cores within this server and then we can apply Eq.2 to calculate the power consumption of this server. In fact, we assume that the assigned workload to this server is uniformly distributed among its processing cores. Nevertheless, we need a load-balancing allocation algorithm to assign VMs to the cores of a server in a balanced manner. The third power model is to investigate the effect of different loads of the processing cores on the power consumption of the multi-core chip. The third choice can be covered in future work. In this paper the second way is adopted to model the power consumption of a multi-core processor similar to [11]. Therefore, the power consumption of a server is derived by Eq.2 where the server utilization is calculated by the following

$$U_j(\mathcal{X}) = \frac{\sum_{i=1}^{N} u_i x_{ij}}{m} \tag{3}$$

where the set $\mathcal{X}$ refers to a particular VM placement. A placement is defined as $\mathcal{X} = \{x_{1,1}, \ldots, x_{N,M}\}$ where $x_{i,j}$ denotes the existence of the i^{th} VM on the j^{th} server. If VM i is placed on server j, then $x_{i,j} = 1$, otherwise, we have $x_{i,j} = 0$. In addition, $U_j(\mathcal{X})$ denotes the utilization of the jth host corresponding to the placement $\mathcal{X}$.

B. VM to core allocation

Let's suppose that the replacement algorithm has already assigned a subset of VMs to a server. The VMs have to be allocated to the cores of the server in such a way that the VM specification is respected. For instance, the VM to core allocator is not allowed to split a VM to more than its maximum parallelism level m'. Since the utilization of VMs may be more than one, we first split the VMs into a number of smaller utilizations. We investigate two alternatives for the splitting: (i) compact splitting (ii) balanced splitting. The compact splitting is performed in the following manner. Each VM is split into $\lfloor u_i \rfloor$ full processor utilizations and one utilization equal to $u_i - \lfloor u_i \rfloor$ if $u_i \neq \lfloor u_i \rfloor$. For instance, a VM with utilization equal to 2.3 will be divided to the following three splits: 1, 1 and 0.3. The balanced splitting, however, creates m' splits with utilization equal to u_i/m'. For instance a VM with utilization equal to 1.5 and $m' = 3$ is split to three splits with utilization equal to 0.5.

Thereafter, the split VMs are sorted based on decreasing utilization. Finally, we allocate the split VMs on the cores using the Worst Fit (WF) heuristic. WF works as follows. The algorithm treats the splits in a sequence. Note that the splits are ordered. At each allocation step, we select a core in which after allocating the current split, maximum free space is left in the core. If the algorithm fails to allocate the current split, then it discards the current split and it continues with the next one in the sequence. The rational behind using the WF heuristic is that this algorithm allocates the utilization among the cores in a balanced manner. The balanced core utilization, in turn, minimizes the consumed power by the server. This is because the idle cores are not shut down. The algorithm returns a value indicating the amount of failures. The allocation failure degree ρ_i^u is the sum of the utilization of all VM splits that the allocation heuristic fails to allocate on the cores. We present a simple example to elaborate this parameter. Suppose that we want to allocate three VMs with utilization equal to 0.6 on a dual core server. The allocation failure degree is equal to 0.6. Note that although the sum of utilization of VMs is less than the available processing power of the multiprocessor ($1.8 < 2$), the allocation fails ($\rho_i^u = 0.6$). A failure degree equal to zero ($\rho_i^u = 0$) means that all the placed VMs can be executed successfully on the server without any constraint violation.

Optimization problem. In this subsection the problem is mathematically formulated as an integer linear optimization problem. There are a lot of tools and algorithms which then can be applied to solve the optimization problem. The goal function is to minimize the total power of the set of servers.

$$\textbf{Minimize:} \qquad TP(\mathcal{X}) = \sum\nolimits_{i=1}^{M} P_i^{Server}(\mathcal{X}), \tag{4a}$$

$$\textbf{Subject to:} \qquad \sum\nolimits_{i=1}^{N} h_i x_{ij} \le H \qquad \forall j \in [1 \dots M], \tag{4b}$$

$$\rho_i^u \le 0 \qquad \forall i \in [1 \dots M], \tag{4c}$$

$$\sum\nolimits_{j=1}^{M} x_{ij} \le 1 \qquad \forall i \in [1 \dots N]. \tag{4d}$$

where $TP(\mathcal{X})$ indicates the total power of servers for the assignment $\mathcal{X}$, and $P_i^{Server}(\mathcal{X})$ denotes the power of the ith server for the assignment $\mathcal{X}$. It should be mentioned that Eq.4b implies the memory constraint which according to that the sum of the memory requirements of the VMs assigned to each server should not exceed the available memory in that server. Eq.4c reflects the effect of the utilization constraint on the optimization problem. This formula is formed based on the output of the worst fit algorithm mentioned as the mechanism of the allocation of VMs to cores. The last constraint (Eq.4d) represents the *no redundancy* restriction, meaning that each VM should be assigned to no more than one server.

In the context of integer linear optimization it is highly desirable to integrate the goal function and all constraints into one function which can be used as the fitness function for optimization problem solvers. In the following we suggest a single fitness function comprising the whole optimization problem.

$$Fitness(\mathcal{X}) = TP(\mathcal{X}) + \alpha \sum_{j=1}^{M} \rho_j^{mem}(\mathcal{X}) + \beta \sum_{i=1}^{N} \rho_i^{u}(\mathcal{X}) \tag{5}$$

where $\rho_j^{mem}(\mathcal{X})$ is the penalty function defined by Eq.6 which is applied to measure satisfiability of a given assignment from a memory perspective. It should be noted that if the value of both the second and the third terms in Eq.5 are zero, then the assignment $\mathcal{X}$ satisfies all the memory and deadline constraints. Otherwise, some of the deadlines are missed or some memory violations happen. α and β are the penalty coefficient used to guide the search towards valid solutions. These coefficients tune the weight of the penalty functions with regards to both the range of the goal function and the importance of violation of each constraint. For example, in a soft real-time system, where missing a small number of deadlines may be tolerable, β can be set to a lower value.

$$\rho_j^{mem}(\mathcal{X}) = \max(0, \sum\nolimits_{i=1}^{N} h_i x_{ij} - H) \tag{6}$$

C. VM placement algorithm.

In this subsection an energy aware algorithm to placement of VMs onto the servers is suggested. As the problem is NP-hard, finding an exact solution needs an exhaustive search which is dramatically time-consuming and can not be applied for medium or large instances of the problem. Therefore, we suggest a powerful meta-heuristic evolutionary algorithm which is able to find a near-optimal solution in a reasonable execution time. This evolutionary algorithm is a Max-Min Ant System [12] being leveraged by an asynchronous parallel version of the Simulated Annealing (SA) algorithm. A flowchart scheme of this framework is given by Fig. 2. As is seen in the flowchart, most parts of this algorithm are implemented in a multi-threading manner where the threads can be executed concurrently on different cores, and thus we exploit the potential of multi-core processors to accomplish a highly efficient search. Some further details regarding the framework configuration are described below:

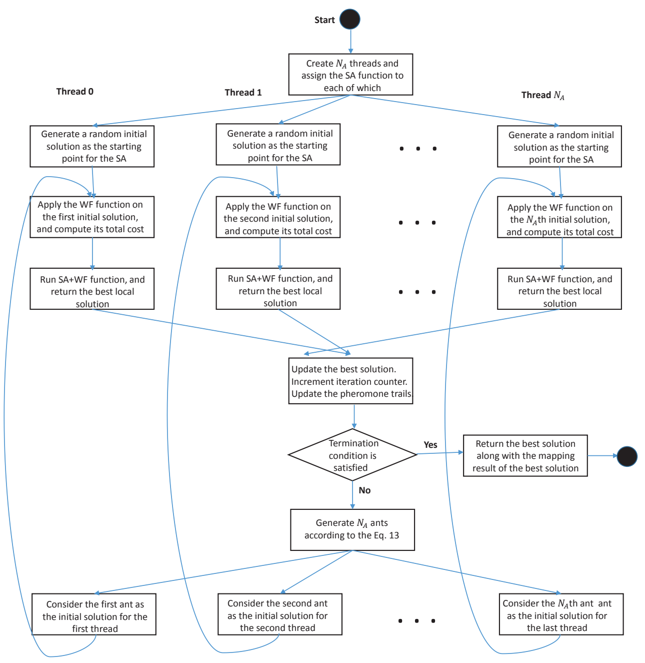
Start
Create N_A threads and assign the SA function to each of which
Thread 0
Thread 1
Thread N_A
Generate a random initial solution as the starting point for the SA
Apply the WF function on the first initial solution, and compute its total cost
Apply the WF function on the second initial solution, and compute its total cost
Apply the WF function on the N_Ath initial solution, and compute its total cost
Run SA+WF function, and return the best local solution
Update the best solution. Increment iteration counter. Update the pheromone trails
Termination condition is satisfied
Yes
No
Return the best solution along with the mapping result of the best solution
Generate N_A ants according to the Eq. 13
Consider the first ant as the initial solution for the first thread
Consider the second ant as the initial solution for the second thread
Consider the N_Ath ant ant as the initial solution for the last thread

Fig. 2: Flowchart representation for the proposed solution framework.

- *Problem space:* The set of all possible allocations for a given set of VMs and servers is called the problem space.
- *Solution representation:* Each point in the problem space is corresponding to an assignment of VMs to the servers that potentially could be a solution for the problem. The solution representation strongly affects the algorithm performance. We represent each allocation solution with a vector of N elements, and each element is an integer value between one and M. The vector is called Placement Representation (PR). Fig. 3 shows an illustrative example for a placement solution. The third element of this example is two, which means that the third VM (corresponding to the third application) is assigned to the second server. Furthermore, this representation causes satisfaction of the no redundancy constraint.

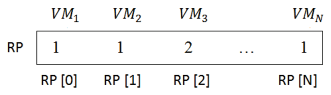

Fig. 3: Representation for assignment of VMs to servers.

- *Neighborhood structure used by the SA function:* SA constitutes a sub set of the problem space that is reachable by moving any single VM to any other server as the neighbors of the current solution. Therefore, each solution has $N(M-1)$ different neighbors, because each VM can run on one of the other $M-1$ servers.

- *Selecting neighbor in SA:* SA in each step, instead of considering all neighbors (i.e., $N(M-1)$ neighbors), selects one VM randomly and then it examines all neighbors of the current solution in which the selected VM is assigned to another server. Hence, it visits $M-1$ neighbors, and then the best solution of this subset is designated irrespective of whether it is better than the current solution. We call this process *stochastic-systematic selection*, because we use a combination of systematic and stochastic process to select a neighbor.
- *Cooling schedule in SA:* There are two common types of cooling schedules, namely, monotonic and non-monotonic. The cooling schedule of the SA in this paper is assumed monotonic in the sense that the temperature of the current iteration is equal to $\mu \times$ the temperature in the previous iteration, where μ is a real value between zero and one.
- *Stopping condition of SA:* The algorithm terminates when the current temperature ψ_i becomes less than the final temperature ψ_f.
- *SA+WF:* The fitness function is used as the tool to evaluate each candidate in problem space. The function mentioned by Eq.5 is applied to this purpose nevertheless, as is seen, in this equation the WF function should be invoked to calculate the value of the utilization penalty and thus, we call the whole process SA+WF.
- *Updating the pheromone trails:* Real ants use trails of a chemical substance to communicate with other ants to inform them about the directions in which food can be found. Actually, the pheromone trails are a kind of distributed numeric information which is modified by the ants to reflect their experience achieved during solving a particular problem. In order to apply the ant system to VM placement problems, a pheromone matrix Ph with the size of $M \times N$ is required where the element Ph_{ij} is corresponding to the assignment of the ith VM to the jth server. Updating the pheromone trails is done first by lowering the pheromone trails by a constant factor (called evaporation) and then by allowing the best ant to deposit pheromone on the direction that it has visited (called reinforcement). In particular, the update can be performed by

$$Ph_{ij}^{I+1} = \varpi \times Ph_{ij}^{I} + \frac{x_{ij}^{best}}{Fitness(PR^{best})} \quad (7)$$

  where ϖ denotes the evaporation factor, Ph_{ij}^{I+1} indicates the pheromone value for the next iteration, x_{ij}^{best} is a binary variable which is equal to one if in the best solution the ith VM is assigned to the jth server, otherwise it is set to zero, and $Fitness(PR^{best})$ denotes the fitness value for the best solution.
- *Generation of ants:* The ants are created based on a probabilistic decision relevant to the pheromone values. In other words, if the pheromone value for the element Ph_{ij} is a large value then the ith VM will probably be assigned to the jth server in the next ants. This concept is reflected by the following formula

$$Prob^k(x_{ij}) = \frac{Ph_{ij}}{\sum_{l=1}^{N_A} Ph_{il}} \quad (8)$$

  where $Prob^k(x_{ij})$ denotes the probability of assigning the ith VM to the jth server in the kth ant.
- *Stopping condition of the main algorithm:* The algorithm terminates after a specific number of iterations, denoted by υ.

It is worth noting that the algorithm is developed in such a way that only some light instructions are located in the non-parallel part such as updating the pheromone trails and selecting the next generation of ants while the CPU-intensive functions like the fitness function is handled in the parallel part.

IV. Related Work

Virtualization techniques have been widely studied in the real-time scheduling community. Early works have been evolved around virtualizing the computational capacity of single-processor hardware. Such a virtualized hardware is refereed as a Virtual Processor (VP). For instance, the periodic resource model uses period and budget [13] for characterizing the VPs. Several authors have proposed different virtualization models targeting multiprocessors. Multiprocessor models also need to specify the maximum parallelism level in their interface. Shin *et al.* proposed Multiprocessor Periodic Resource (MPR) model [3]. Lipari and Bini suggested the Bounded-Delay Multipartition (BDM) model [14]. Leontyev and Anderson [15] proposed a model that only specifies bandwidth w in the component interface. The Generalized MPR (GMPR) model [16] reduces pessimism of the MPR model. There is often a trade-off between simplicity and accuracy in virtualization models. In other words, simple interfaces tend to be pessimistic and introduce more resource loss than the more detailed yet complex models. We opted the MPR model because of its simplicity. Scheduling of VMs compliant with the MPR model is implemented in the RT-Xen hypervisor [7]. The authors have implemented both global and partitioned multiprocessor scheduling algorithms.

From an architectural point of view, different alternatives have been investigated. In order to avoid replicating software layers through hardware visualization, the operating system can be virtualized in [17]. In this approach all user applications are run on top of a single operating system. The periodic real-time model is used for scheduling both batch and interactive VMs using the EDF scheduling algorithm [18]. In the context of the IRMOS project, a hierarchical scheduler is developed for providing timing guarantees for the VMs [19]. Q-Cloud uses a control theoretic approach for providing dynamic resources to the VMs [20]. In this framework, the VMs communicate their Quality-of-Service (QoS) to the hypervisor. The hypervisor then adjusts the amount of resource provision considering the current QoS as well as the SLA.

From a placement perspective, a wide range of studies have been carried out to place a set of VMs into the physical servers to minimize operation cost of a cloud data center by maximizing its energy efficiency [21], [11]. The majority of these works consider real-time aspects of VMs while the real-time requirement is either refereed as the SLA [22] or explicitly mentioned as deadlines [23], [24].

Recently, [9], [25] considered the placement of VMs to servers in a cloud data center to minimize energy consumption by using a consolidation approach while the live migration was also taken into account. In addition, in some of these works, e.g [26], energy consumption of network equipment is also taken into consideration. On the other hand, there are a wide range of studies that focus on the energy minimization problem in cloud data centers using the Dynamic Voltage Scaling (DVS) technique. They strive to reduce the frequency of processors within the servers to consume lower energy. In fact, the problem in such studies is finding a good compromise between the QoS requirements and energy consumption.

Although extensive studies have been carried out in the context of energy aware VM placement onto the cloud data centers, the structure of their real-time VMs are completely different with that considered in this paper. The VMs in the mentioned works consist of a set of ordinary real-time tasks where there is only one instance for each task whereas in this paper we consider periodic real-time tasks (or sporadic with a known minimum inter-arrival time) where a set of identical instances of a task is released periodically. It leads us to an additional complexity which must be handled in a holistic manner.

V. Conclusions and Future work

In this paper we studied the problem of allocation of a set of real-time applications where each of them are comprising a set of periodic tasks onto a cloud data center. We approached the problem from a cloud provider point of view. The periodic tasks are first mapped to a set of VMs. The VM specification is abstracted based on the property of tasks assigned to the VM. VMs are then placed on servers. Not only the placement algorithm considers the timing requirements of the real-time applications running within the VMs but it also attempts to minimize energy consumption by reducing the number of used servers. To deal with this problem, an integer linear optimization problem is first introduced and after that a two-level placement framework was introduced.

In future we will study the behavior of our proposed solution through extensive simulation studies. In addition we would like to relax the following assumptions. (i) We assumed that all of the VMs are defined in advance and we provided a solution for off-line placement of VMs. We intend to investigate on-line VM placement given that the VMs are allowed to leave/join the data center during runtime. (ii) We assumed that VM migration is not allowed due to the off-line nature of our approach. We will enable VM migrations while considering the migration overhead on the scheduability of real-time VMs. (iii) We only considered periodic VMs containing periodic tasks with unbounded life-time in this paper. We will enable coexistence of VMs which only need to run for one instance before a given deadline together with periodic VMs on the same server. (iv) Finally, we would like to assume limited life time for the periodic VMs. This assumption, in turn, will trigger the need for on-line VM migration (as discussed above).

References

[1] G. Gruman and E. Knorr, "What cloud computing really means," *InfoWorld*, vol. 37, p. 13, 2008.

[2] J. G. Koomey, "Estimating total power consumption by servers in the us and the world," 2007.

[3] A. Easwaran, I. Shin, and I. Lee, "Optimal virtual cluster-based multiprocessor scheduling," *Real-Time Systems*, vol. 43, no. 1, pp. 25 – 59, September 2009.

[4] T. Cucinotta, K. Oberle, M. Stein, P. Domschitz, and S. Mullender, "Run-time support for real-time multimedia in the cloud," in *REACTION'13*, December 2013.

[5] A. Toma and J.-J. Chen, "Server resource reservations for computation offloading in real-time embedded systems," in *ESTIMedia'13*, October 2013, pp. 31–39.

[6] M. Garca-Valls, T. Cucinotta, and C. Lu, "Challenges in real-time virtualization and predictable cloud computing," *Journal of Systems Architecture*, vol. 60, no. 9, pp. 726–740, 2014.

[7] S. Xi, M. Xu, C. Lu, L. T. Phan, C. Gill, O. Sokolsky, and I. Lee, "Real-time multi-core virtual machine scheduling in Xen," in *EMSOFT'14*, October 2014.

[8] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *ACM SIGARCH Computer Architecture News*, vol. 35, no. 2. ACM, 2007, pp. 13–23.

[9] A. Beloglazov, J. Abawajy, and R. Buyya, "Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing," *Future Generation Computer Systems*, vol. 28, no. 5, pp. 755–768, 2012.

[10] A.-C. Orgerie, L. Lefevre, and J.-P. Gelas, "Demystifying energy consumption in grids and clouds," in *GCC'10*, 2010, pp. 335–342.

[11] H. R. Faragardi, A. Rajabi, R. Shojaee, and N. Yazdani, "Towards energy-aware resource scheduling to maximize reliability in cloud computing systems," in *HPCC'13*, 2013.

[12] T. Stützle and H. H. Hoos, "Max–min ant system," *Future generation computer systems*, vol. 16, no. 8, pp. 889–914, 2000.

[13] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," in *RTSS'03*, December 2003, pp. 2–13.

[14] G. Lipari and E. Bini, "A framework for hierarchical scheduling on multiprocessors: From application requirements to run-time allocation," in *RTSS'10*, December 2010, pp. 249–258.

[15] H. Leontyev and J. Anderson, "A hierarchical multiprocessor bandwidth reservation scheme with timing guarantees," in *ECRTS'08*, July 2008, pp. 191–200.

[16] A. Burmyakov, E. Bini, and E. Tovar, "Compositional multiprocessor scheduling: the gmpr interface," *Real-Time Systems*, vol. 50, no. 3, pp. 342–376, 2014.

[17] J. Sacha, J. Napper, S. Mullender, and J. McKie, "Osprey: Operating system for predictable clouds," in *DSN-W'12*, June 2012, pp. 1–6.

[18] B. Lin and P. A. Dinda, "Vsched: Mixing batch and interactive virtual machines using periodic real-time scheduling," in *SC'05*, 2005, p. 8.

[19] F. Checconi, T. Cucinotta, D. Faggioli, and S. S. S. Anna, "Hierarchical multiprocessor CPU reservations for the linux kernel." in *OSPERT'09*, June 2009.

[20] R. Nathuji, A. Kansal, and A. Ghaffarkhah, "Q-clouds: Managing performance interference effects for qos-aware clouds," in *EuroSys'10*, April 2010, pp. 237–250.

[21] I. Pietri, M. Malawski, G. Juve, E. Deelman, J. Nabrzyski, and R. Sakellariou, "Energy-constrained provisioning for scientific workflow ensembles," in *CGC'13*, 2013, pp. 34–41.

[22] K. H. Kim, A. Beloglazov, and R. Buyya, "Power-aware provisioning of cloud resources for real-time services," in *MGC'09*. ACM, 2009, p. 1.

[23] A. Rajabi, H. R. Faragardi, and N. Yazdani, "Communication-aware and energy-efficient resource provisioning for real-time cloud services," in *CADS'13*, 2013, pp. 125–129.

[24] Y. Gao, Y. Wang, S. K. Gupta, and M. Pedram, "An energy and deadline aware resource provisioning, scheduling and optimization framework for cloud systems," in *CODES'13*, 2013, p. 31.

[25] A. Beloglazov and R. Buyya, "Adaptive threshold-based approach for energy-efficient consolidation of virtual machines in cloud data centers," in *MGC'10*, 2010, pp. 4–.

[26] J. A. Pascual, T. Lorido-Botrán, J. Miguel-Alonso, and J. A. Lozano, "Towards a greener cloud infrastructure management using optimized placement policies," *Journal of Grid Computing*, pp. 1–15, 2014.