Noname manuscript No.
(will be inserted by the editor)

Deadline Scheduling for Aperiodic Tasks in inter-Cloud Environments: a New Approach to Resource Management

Florin Pop · Ciprian Dobre · Valentin Cristea · Nik Bessis · Fatos Xhafa · Leonard Barolli

Received: date / Accepted: date

Abstract In the Big Data era, the speed of analytical processing is influenced by the storage and retrieval capabilities to handle large amounts of data. While the distributed crunching applications themselves can yield useful information, the analysts face difficult challenges: they need to predict how much data to process and where, such that to get an optimum data crunching cost, while also respect deadlines and Service Level Agreements within a

*The research presented in this paper is supported by the following projects: "*SideSTEP - Scheduling Methods for Dynamic Distributed Systems: a self-* approach*", (PN-II-CT-RO-FR-2012-1-0084);"; *CyberWater* grant of the Romanian National Authority for Scientific Research, CNDI-UEFISCDI, project number 47/2012.
This paper is based on [?**]: Florin Pop, Ciprian Dobre, Valentin Cristea, and Nik Bessis. Scheduling of sporadic tasks with deadline constrains in cloud environments. In *Proceedings of the 2013 IEEE 27th International Conference on Advanced Information Networking and Applications*, AINA '13, pages 764–771, 2013. IEEE Computer Society.

Florin Pop - Principal Author
Associate Professor, Computer Science Department, Faculty of Automatic Control and Computers, University *Politehnica* of Bucharest, Romania, E-mail: florin.pop@cs.pub.ro

Ciprian Dobre - Corresponding Author
Associate Professor, Computer Science Department, Faculty of Automatic Control and Computers, University *Politehnica* of Bucharest, Romania, E-mail: ciprian.dobre@cs.pub.ro

Valentin Cristea
Professor, Computer Science Department, Faculty of Automatic Control and Computers, University *Politehnica* of Bucharest, Romania, E-mail: valentin.cristea@cs.pub.ro

Nik Bessis
Professor of Computer Science, School of Computing & Maths, University of Derby, United Kingdom, E-mail: N.Bessis@derby.ac.uk

Fatos Xhafa
Professor Titular d'Universitat, Hab. Full Professor, Department of Language & Infomatics Systems, Technical University of Catalonia Barcelona, Spain, E-mail: fatos@lsi.upc.edu

Leonard Barolli
Professor, Department of Information and Communication Engineering, Faculty of Information Engineering, Fukuoka Institute of Technology (FIT), E-mail: barolli@fit.ac.jp

limited budget. In today's data centers, data processing on demand and data transfers requests coming from distributed applications are usually expressed as aperiodic tasks. In this paper we challenge the problem of tasks scheduling with deadline constraints of aperiodic tasks within inter-Cloud environments. In massively multithreaded computing systems that deal with data-intensive applications, Hadoop and BaTs tasks arrive periodically, which challenges traditional scheduling approaches previously proposed for supercomputing. Here we consider the deadline as the main constrain, and propose a method to estimate the number of resources needed to schedule a set of aperiodic tasks, considering both execution and data transfers costs. Starting from classical scheduling techniques, and considering asynchronous tasks handling, we analyze the possibility of decoupling task arriving from task creation, scheduling and execution, sets of actions that can be put into a peer-to-peer relation over a network or over a client-server architecture in the Cloud. Based on a mathematical model, and using different simulation scenarios, we prove the following statements: (1) multiple source of independent aperiodic tasks can be considered similar to a single one; (2) with respect to the global deadline, the tasks migration between different regional centers is the appropriate solution when the number of estimated resources exceed a data center capacity; and (3) in a heterogeneous data center, we need a higher number of resources for the same request in order to respect the deadline constrains. We believe such results will benefit researchers and practitioners alike, who are interested in optimizing the resource management in data centers according to novel challenges coming from next-generation Big Data applications.

Keywords Deadline Scheduling · Aperiodic Tasks · Resource Allocation and Cloud Environments · Big Data

PACS Distribution theory, 02.50.Ng

Mathematics Subject Classification (2000) 68M20 · 68M14 · 68U20

1 Introduction

Cloud Computing is a new paradigm where resources, hardware or software, are offered to users remotely, in the form of services. Behind this vision, a Cloud middleware transparently provide support for reliability, scalability, security, and more. Because the middleware needs to support the distributed execution of complex applications, it also needs to provide guarantees for their execution. For deadline constraints, resource management and task scheduling become critical components in such systems. The problem is even more complicated for other types of real-time systems, either dealing with periodic (time-driven) tasks and/or aperiodic (event-driven) tasks. An example of such a real-time system can be a factory controller that periodically executes critical control loops, while being also responsible for treating aperiodic user interaction [**?**] or batch production scheduling in the process industries [**?**]. Such real-time systems, dealing simultaneous with multiple constraints, are called hybrid real-

time systems. They can, in fact, support a wide range of applications dealing with deadlines: meteorological prediction, genomic analysis, real-time complex physics simulations, monitoring watershed parameters through software services [**?**], and biological and environmental assistance. The execution in due time also affects Internet searches, finance and business informatics, and many more.

Another example relates to vehicular ad-hoc networks (VANETs) - such networks are often used in conjunction with composite very-large-scale neighborhood search algorithms, to solve the critical vehicle routing problem (see Agarwal et al. in [**?**]). Massively multiplayer online games, consisting of huge worlds populated by thousands of clients, far beyond the ability of a single server to maintain [**?**], are another example of deadline-constraint systems. In this case, to provide players with the illusion of a single large world, dedicated systems often divide their game world across servers and synchronize all nearby activity between them. In a network that support such type of applications an important challenge is selective contents broadcasting depending on users' preferences with node relay-based web cast. To meet deadlines, waiting time is reduced by receiving contents from several nodes. In [**?**], authors propose a scheduling method considering reconnection on selective contents delivery with node relay-based web cast that relay data among nodes.

Generally, scheduling in distributed systems deals with the problem of assigning tasks, sometimes of different types, to a set of resources, sometimes with different characteristics [**?**]. The tasks can be resource-intensive, where a resource is usually CPU, Memory, and I/O. It is known that the general scheduling problem is NP-complete [**?**] [**?**].

In Cloud, background scheduling is the simplest manner to handle the scheduling of a mixed set of periodic and aperiodic tasks, and executing the aperiodic tasks when no periodic task instance is ready to run. Aperiodic tasks can be scheduled and executed on free time slots remaining after periodic tasks are executed. The disadvantage of this approach is experienced in case of high periodic loads, when the resulting aperiodic response time can be quite long. Nevertheless, background scheduling has a great advantage in its simplicity having two queues: one for the periodic task set and the other for the aperiodic tasks, with the periodic queue having a higher priority than the aperiodic one. An algorithm for scheduling of aperiodic task systems with arbitrary deadlines on identical multiprocessor platforms is presented in [**?**]. The algorithm is based on the concept of semi-partitioned scheduling, in which most tasks are fixed to specific processors, while a few tasks migrate across processors. The solution proposed in [**?**] for scheduling of aperiodic tasks on multiprocessors uses the approximation of the exact demand bound function on uni-processor as a criterion and introduce a partitioned scheduling algorithm for a least-number processors and fixed-number processors respectively.

Moschakis (2012) studies the performance of a distributed Cloud Computing model, based on the Amazon Elastic Compute Cloud (EC2) architecture that implements a Gang Scheduling scheme (an efficient job scheduling algorithm for time sharing). In this approach virtual machines (VMs)

act as the computational units of the system. The authors prove that Gang Scheduling can be effectively applied in a Cloud Computing environments, both performance-wise and cost-wise [**?**]. Looking for performance, the optimum performance from the distributed computing system is achieved by using effective scheduling and load balancing strategy [**?**] [**?**]. The author propose a Mixed Task Load Balancing for cluster of workstation systems. In this strategy pre-tasks are assigned to each worker by the master to eliminate the worker's idle time.

The paper is structured as follow: in Section 2 the classical approaches of aperiodic task scheduling are presented and analyzed. Section 3 introduces the problem of aperiodic task scheduling with deadline constrains considering homogenous and heterogeneous datacenters for inter-Clouds environments. Section 4 presents simulation experiments and analyzes the migration behavior in order to meet deadlines. The paper ends with analysis, conclusions and future work. This paper is based on [**?**].

2 Aperiodic Scheduling. Classical Approaches

There are several approaches to the scheduling problem that were considered over time. These approaches consider different scenarios that take into account the types of applications, the execution platform, the execution platform type, the types of algorithms used and the constraints that users may require. [**?**] presents a solution of scheduling bag of tasks. In this case users receive guidance, and are able to choose the way the application is executed: with more money and faster or with less money but slower. Other important element in this method of scheduling is the phase of profiling. The basic scheduling is realized with a type of bounded knapsack algorithm. [**?**] presents the idea of scheduling based on scaling up and down the number of the machines in the cloud system. The users can also choose their own policies. This solution provides meeting the deadline with reducing the cost. A scheduling solution based on genetic algorithms is given in [**?**]. Here the scheduling is made on grid systems. They are not the same as the cloud systems, but the principle of assigning tasks to resources is the same. This solution of scheduling works with application that can be modeled as DAGs. The idea for this solution is minimizing the duration of the application execution while the budget is respected. This approach also considers the heterogeneity of the system. The paper [**?**] presents a scheduling model which takes in consideration both budget and deadline constraints.

There are several classical approaches for the scheduling problem, considering a central server [**?**]: polling server, deferrable server, priority exchange server, sporadic server, slack stealing. The *Polling Server* (PS) implies creating a periodic task – a server, which will service aperiodic tasks. The server task is created in order to emerge aperiodic task servicing from the background scheduling and therefore, to improve the average response time [**?**]. The *Deferrable Server* (DS) algorithm was introduced by Lehoczky, Sha and

Stosnider in [**?**]. The technique is derived from the PS, and manifests improved response times for aperiodic tasks. The DS algorithm creates a periodic task for servicing aperiodic requests and preserves server capacity if no requests are pending. The *Priority Exchange Server* (PES) considers that the server task usually has a high priority and differs from the other server-based algorithms in the way that it preserves its capacity, by converting it into execution time in a lower-priority periodic task [**?**]. The *Sporadic Server* (SS) algorithm was introduced by Sprunt [**?**] in order to enhance the average response time of aperiodic tasks without degrading the utilization bound of the aperiodic task set. A particular scheduling technique for aperiodic requests is the *Slack Stealing* (SSt) algorithm, introduced by Lehoczky and Ramos-Thuel in [**?**]. This technique offers great improvement in response time over the previously discussed service methods (PES, DS, SS). The SSt algorithm does not create a periodic task to service the aperiodic request, instead it creates a passive task, named Slack Stealer, that attempts to make time for servicing aperiodic tasks by stealing all the processing time it can from the periodic tasks without causing their deadlines to be missed. All algorithms used in presented models behave the same manner when there are enough aperiodic tasks to execute.

3 Aperiodic Task Scheduling with Deadline Constrains

Cloud Computing is one of the fastest evolving paradigm in the domain of Computer Science. Whether one wants to provide a simple file transfer service that consumes an insignificant amount of resources and time, or a parallel and distributed algorithm that defines a weather prediction model that requires high computing power or even a very strict and secure banking service, its implementation by means of a service has a great number of advantages. Tasks execution can address one Cloud or multiple Clouds, depending on users' requirements. So, hybrid Clouds will be considered and inter-Clods environments become the fundamental platform for tasks execution. For concurrent access we consider a queuing system for tasks submission. For such type of systems, the number of task arrivals in a given interval of time is a random variable with a Poisson distribution [**?**] [**?**]. In this section we describe the estimation method for the necessary resources to schedule a set of aperiodic tasks in parallel with periodic tasks.

Let's consider a time interval with length t and a set of n tasks $\{T_i\}_{1\leq i\leq n}$, each task having known the arrival time a_i and the deadline d_i in the considered time interval: $d_i - a_i \leq t, \forall T_i$. If we consider τ the time between two successive arrivals and $T \geq 0$ a time threshold. We have [**?**]: $Prob(\tau \leq T) = 1 - e^{-\lambda T}$ so, if T is fixed apriori, the probability has a constant value (similar approach with unitary processes [**?**] [**?**]). The Poisson distribution for the number of tasks arrivals from a source k in an arbitrary time interval with length equal with t, for $n_k = 0, 1, \ldots$ is:

$$Prob(N_k(t) = n_k) = e^{-\lambda_k t} \frac{(\lambda_k t)^{n_k}}{n_k!}.$$

where $\lambda_k t$ is the shape parameter which indicates the average number of events (tasks arrivals) in the given t time interval. For each tasks source (different users that submit for execution a set of aperiodic tasks) we have a specific n_k number of tasks for an interval t, and a specific λ_k parameter for Poisson distribution. If $n = \sum_k n_k$ is the total number of given tasks and $N(t) = \sum_k N_k(t)$ is the total number of tasks arrived in the t interval, we have:

$$Prob(N(t) = n) = Prob(N_1(t) = n_1, N_2(t) = n_2, \ldots).$$

We have the following result [**?**] *Let's consider m sources of aperiodic tasks with specific parameters* $(\lambda_k, n_k)_{1 \leq k \leq m}$ *for Poisson distribution. For a scheduling system the m inputs appear appear as a single one with* (λ, n) *specific parameters for Poisson distribution, where* $\lambda = \sum_k^m \lambda_k$ *and* $n = \sum_k^m n_k$. This result allow to consider a queuing system for scheduling with a local coordinator for a regional center. In each regional center we have multiple task sources with different submission characteristics.

Deadline Scheduling is NP-Complete in a strong sense, proofed in [**?**] by a pseudo-polynomial reduction from strongly NP-Complete 3-Partition Problem: for a set of $3m$ positive numbers $A = (a_1, a_2, \ldots, a_{3m})$ with $\sum a_i = mB$ and $B/4 < a_i < B/2$ for each i, is there a partition of A into $A_1, \ldots, A_m$ such that $\sum_{a_j \in A_i} a_j = B$, for each i. We can consider here a set of m map-reduce tasks (equal number of mappers and reducers), each set A_i encoding the processing time (p) for map and reduce task and add a "transition" task in order to satisfy the restriction of $\sum_{a_j \in A_i} a_j = B$ translated in $p_{map} + p_{reduce} + p_{transition} = B$, and B can be considered the total execution time of a set (map, reduce).

Now, for each source we can consider the following model for deadline scheduling. Each task is described as $T_i = (a_i, d_i, data_i)$, where a_i is arrival time, d_i is deadline and $data_i$ is the input data volume. We consider a soft-real time system and we introduce for a request $Q = \{T_i | T_i = (a_i, d_i, data_i), i = 1, 2, \ldots\}$ the *global arrival* time $A = \min_i \{a_i\}$ and *global deadline* $D = \max_i \{d_i\}$. Considering f the fraction of input data that is given as output, we have $output_i = f * data_i$. Now, let's introduce the c_{exec} the execution cost and c_{com} the communication cost (here, the cost is associated with processing time for a data unit). A similar model, for Hadoop jobs is presented in [**?**]. We consider for the beginning a homogeneous environments with the same computation cost for all resources and the same communication cost for all links. Then, the total cost for Q is:

$$TotalCost = \frac{1}{N_{res}} \sum_i (data_i * c_{exec} + output_i * c_{exec}) + \sum_i output_i * c_{com}$$

where n_{res} is the necessary number of resources in a regional center to support execution of Q set. We have the following result [**?**], based on presented assumptions: *For a request Q and a homogeneous regional center with* n_{res} *resources, considering a schedule with deadline constrains, then*:

$$N_{res} \geq \frac{(1+f) c_{exec} \sum_i data_i}{D - A - f c_{com} \sum_i data_i}.$$

This result allow to set the number of resources in a regional center as:

$$N_{res} = \left[\frac{(1+f)c_{exec} \sum_i data_i}{D - A - fc_{com} \sum_i data_i} \right] + 1$$

and, if we need more resources we will consider migration between regional centers. This assumption is based on the maximization of slacks approach, as follow. We define the slack for task T_i considering the remaining computation time $c_i(t)$ at the moment t, as: $slack_i(t) = d_i - (t + c_i(t))$. The slacks are used in the scheduling process especially for online scheduling, considering that whenever an aperiodic request is issued, the server (for example, in the SSt scenario) steals all the available slack from periodic tasks and uses it to service the aperiodic request as soon as possible. For a systems with deadline constrains, with a defined Service Level Agreement, we have at any moment of time t: $slack_i(t) \geq 0, \forall T_i$ and $c_i(d_i) \leq 0$.

A numerical example considers a request Q with 1000 tasks, each task having $1KB$ as input and $1KB$ as output, which means $\forall i, data_i = 1KB$ and $f = 1$. If $D - A = 100s$, $c_r = 1s/KB$ and we have no communication, then $n_{res} = 10$. So, the regional center must have minimum 10 CPU (virtual resources).

In general, in a heterogeneous h-regional center we need a higher number of resources for the same request Q in order to respect the deadline constrains. The homogeneous o-regional centers are also always built with high processing capacity machines and high speed network. We have the following result:

$$N_{res}^h \geq N_{res}^o$$

where the N_{res}^h and N_{res}^o represent the lower bound for number of necessary resources to be used for a specific set of tasks to be created in a heterogeneous, respectively in a homogeneous environment.

4 Evaluation Scenarios and Results Interpretation

The practical evaluation is presented in this section, and is represented by simulation experiments that show the behavior of the migration phase applied when the total number of estimated nodes exceeded the regional center capacity.

4.1 MONARC Simulator

MONARC simulator is built based on a process oriented approach for discrete event simulation, which is well suited to describe concurrent running programs as well as all the stochastic arrival patterns, specific for this type of simulation [**?**]. In order to provide a realistic simulation, all the components of the system and their interactions were abstracted. The chosen model is equivalent to the simulated system in all the important aspects. The simulation model is based on regional interconnected centers.

4.2 Distributed Task Scheduling based on Migration in MONARC Simulator

In MONARC, each regional center can also incorporate a task scheduler component. The scheduler is used to simulate the decisions making process regarding the allocation of resources for the execution of tasks based on various internal algorithms. The basic task scheduler implements a decision making algorithm. As output, scheduler can only make one of two decisions: either it assigns the task for execution on designated processing resources or, if there are no available resources, it places the task in a special waiting queue structure for migration or later resubmission. When there are more than one processing units that could handle the execution of a particular task, the task scheduler will choose the one having the minimal load. This value is computed based on the memory consumption and the number of tasks being already concurrently processed on that particular unit.

MONARC also includes a distributed task scheduler class, responsible with implementing a distributed scheduling decision algorithm. This means that in this case the scheduling decision can result in submitting the task for execution in other regional centers than the one they were originally submitted to by the user. The implemented distributed algorithm considers that each local scheduler unit decides where it is better to submit the task for execution.

The algorithm of the distributed task scheduler works as follows. If the load percentage of each CPU unit from the local regional center exceeds a certain value (given by a constant having the default value of 70%), the scheduler sends the task to another regional center. Then the regional center having the minimum average load is chosen to execute the task. If the regional center having the minimum load is a remote one, the task is sent there. Else, it will be executed in the local regional center. When a task is sent to another regional center, the task scheduler from that regional center is responsible with the effective execution of the task (it won‘t try to send it to another regional center, because this way the task could move from one center to another for ever). This model can easily extent to include various new conditions, new resource considerations or performance metrics, in order to test the behavior of new scheduling models and algorithms.

4.3 Simulation Setup

The simulation experiments evaluate the migration of the aperiodic task scheduling between regional centers. We used 4 regional heterogeneous centers (`UPB_01`, `UPB_02`, `DERBY_01` and `DERBY_02`). In each center, we submit a number of tasks with random time intervals between them, in order to simulate the aperiodic behaviour. The time intervals follow a normal distribution, and have different averages in different periods of the day. We defined three periods (morning, midday and evening), and the exact hours when they begin can be set from the configuration file. Each regional center has its own activity as a model for tasks execution, and each activity has several characteristics. The parameters

are set from a configuration file (Table 1 shows the actual values used for several of these parameters): **gmtOffset**: the time difference between the regional center and GMT (n hours); **numDays**: the number of days the simulation will last; **morningTime, lunchTime, eveningTime**: the hours that define the 3 periods of the day; **timeInt1,2,3**: the average time interval between tasks in the 3 periods; **numtasks1,2,3**: the number of tasks submitted in the 3 periods.

Table 1 Simulation Experiment Characteristics for Regional Centers

Regional Center	numDays	morningT	lunchT	eveningT	t1	t2	t3
UPB_01	5	7	7	13	1200	240	3600
UPB_02	5	7	7	13	600	120	1800
DERBY_01	5	7	11	17	1200	240	3600
DERBY_02	5	7	11	17	600	120	1800

All simulation results highlight the evolution for one regional center (UPB_01). For the other three regional centers the evolution is very similar. Figure 1 shows the evolution of submitted and finished jobs. One can observe that there is a periodicity in tasks submission and a slow increasing at the end of the period. The migration process starts here.

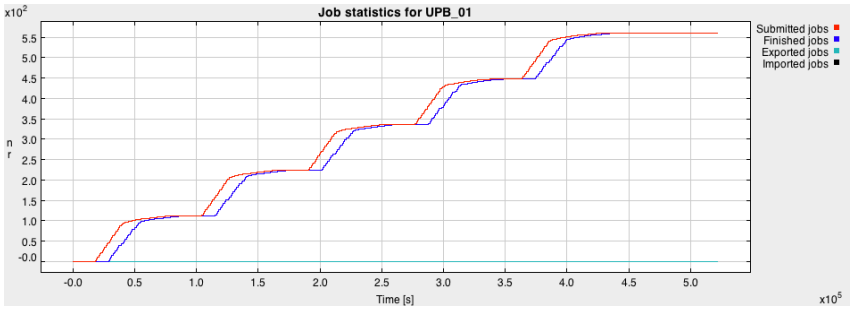

Fig. 1 Simulation Results. Job statistics.

Figure 2 shows the evolution of running and waiting tasks. As seen, there are time interval when the regional renters work at full capacity and there are several waiting tasks. The tasks will stay in the waiting queue only if the deadline constrains will be respected. Here, during the interval that regional centers work at full capacity, the number of waiting task is higher than full capacity, so scheduler will activate the migration function.

Figure 3 highlights the migration function. In the time period when regional centers work at full capacity, if tasks remaining in the waiting queues continue to stay there, the deadline constrains are not satisfied. Thus, all regional centers start the tasks migration and several tasks become submitted

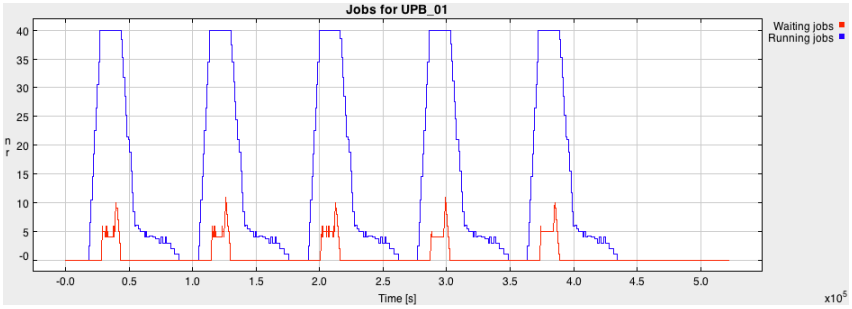

Fig. 2 Simulation Results. Running and Waiting tasks.

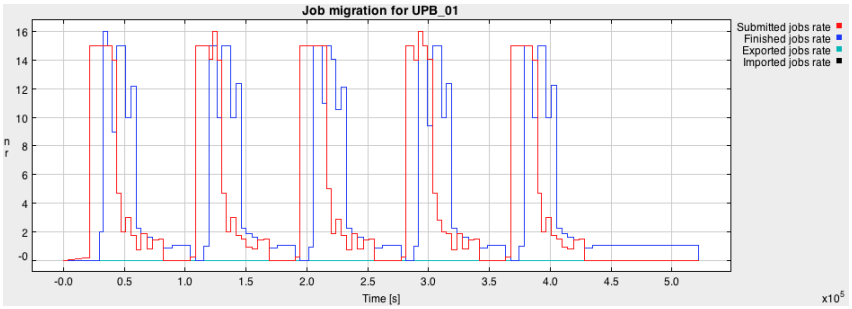

Fig. 3 Simulation Results. Job migration rate.

in other regional centers. All the submitted tasks in the initial phase or in the migration phase are aperiodic tasks.

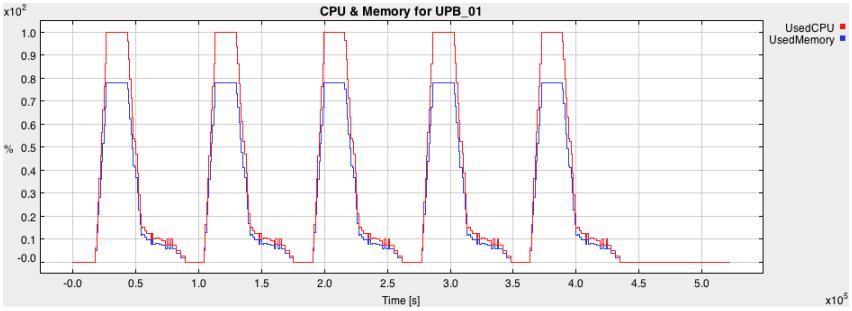

Fig. 4 Simulation Results. Percentage of used CPU and Memory.

The last measurements (see Figure 4) get the CPU usage and memory usage during the experiments. Once again, the figures confirm that regional centers are used at almost full capacity, and the need for migration. The memory usage respects the limitation, but follow the profile of CPU usage graphic.

5 Conclusions

This paper presents the classical approach for aperiodic task scheduling considering a scheduling system with different queues for periodic and aperiodic tasks. We prooved that multiple source of independent aperiodic tasks can be considered like a single one. As a support for deadline scheduling, the optimization of slacks was introduced and a migration function was introduce for regional centers with limited capacity. The paper presented a method to compute a lower bound for number of necessary resources to be used for a specific set of tasks. When this number exceeded the number of resource in a datacenter we will migrate several tasks to other datacenters.

The deadline constraints were presented and we obtained a result, which prove that in general, in a heterogeneous regional center we need a higher number of resources for the same request in order to respect the deadline constrains. The homogeneous regional centers are also always built with high processing capacity machines and high speed network. We establish in this paper a lower bound for dimension of a regional center (number of resources) in order to respect the deadline constrains. This bound depends of computation ans communication costs and also, depends on applications type.

The proved statements can be used as follow: *Statement* (1): multiple source of independent aperiodic tasks can be considered similar to a single one; *Applicability*: is it possible to consider a queuing system with multi-queues for task submission but a single queue for scheduling component. *Statement* (2): the tasks migration between different regional centers is the appropriate solution when the number of estimated resources exceed a data center capacity; *Applicability*: the resource management component implements this technique to distribute the load between different data centers in a inter-Clouds environment. *Statement* (3) in a heterogeneous data center, we need a higher number of resources for the same request in order to respect the deadline constrains; *Applicability*: if we have a pre-computed value for number of nodes necessary for a specific scheduling request (for a homogeneous cluster), we must increase these values for a heterogeneous data center where is more difficult to estimate the costs used in proposed model.

Acknowledgements We would like to thank the reviewers for their time and expertise, constructive comments and valuable insights.