Scheduling of Sporadic Tasks with Deadline Constrains in Cloud Environments

Florin Pop, Ciprian Dobre, Valentin Cristea
Computer Science Department
University Politehnica of Bucharest, Romania
Emails: florin.pop@cs.pub.ro, ciprian.dobre@cs.pub.ro, valentin.cristea@cs.pub.ro

Nik Bessis
School of Computing & Maths
University of Derby
United Kingdom
Email: N.Bessis@derby.ac.uk

Abstract—The mobile interaction in Cloud systems became a fancy behavior. Data processing on demand or data transfers requests are usually sporadic tasks. In a public environment like a Cloud, events are processed according to specific conditions. Each event has one or more tasks that will be scheduled and executed in Cloud. This paper addresses the problem of remote scheduling of aperiodic and sporadic tasks with deadline constrains in Cloud Environments. Starting from classical addressed scheduling techniques and considering asynchronous mechanism to handle tasks, we analyze the possibility of decoupling event listening from task creation and scheduling, actions that can be put into a peer-peer relation over a network or to client-server in Cloud. We consider multiple independent tasks sources that follow with a specific distribution. We will prove in this paper that for a scheduler in a Cloud these independent sources could be considered as a single one. More, we will prove that the resource allocation process respects the same distribution. We created a simulation experiment in MONARC that highlights the capability of tasks migration in order to respect the deadlines.

Keywords—Task Scheduling, Cloud Environments, Sporadic Tasks, Real-Time Systems

I. INTRODUCTION

In the context of Cloud Computing where resources are offered for different users in different way, from physical resource to software (as services), we have major challenges for resource management with optimization constrains. If the systems deal with real-time constrains, resource management and task scheduling become critical. Most real-time systems are hybrid: they deal with both periodic (or time-driven) tasks and aperiodic (or event-driven) tasks. An example of a hybrid system could be a factory controller that periodically executes critical control loops and is also responsible for treating sporadic user interaction. A successful functioning of such a system must ensure responsiveness for both the hard periodic task as well as for the aperiodic tasks, which could have soft or hard deadlines.

Scheduling in distributed computing is an intensely studied problem [1]. It deals with the problem of assigning tasks, sometimes of different types, to a set of resources, sometimes with different characteristics. The tasks can be with or without dependencies, IO intensive or computationally intensive. In the specialty literature there are several approaches to this problem. It is known that this problem is NP-complete [2]. For this type of problems, genetic algorithms are a solution that can give good results for specific optimization criteria. Although genetic algorithms are sometimes slow, if the genetic operators and the fitness function are well implemented than it can be obtained a good solution, quite close to the optimal one.

In a Cloud, where the transparency is an important issue, 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 remained after periodic tasks are executed. The disadvantage of this approach is experienced in the case of high periodic loads, when the resulting aperiodic response time can be too long. This makes background scheduling suitable only when periodic processor utilization is not high and aperiodic activities do not have hard timing constraints. 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.

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. In [4] is presented a solution of scheduling bag of tasks. Here the users receive guidance. They 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. In [5] is presented 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 [6]. 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 takes into account the
heterogeneity of the system. The paper [7] presents a scheduling model which takes in consideration both budget and deadline constraints. The level of user interaction here is very high. The user can change dynamically what he wants from the scheduler. The interventions can be made every schedule round. This is an interesting model because the user can choose to pay more or less depending on the scenario. The sure thing is that the user has more power. Here is also made an estimation of the tasks and the rule after this estimation is made is First Come First Served.

The paper presents in Section II the classical approaches of sporadic and aperiodic task scheduling. Then, in Section III, we introduce the problem of multiple sporadic tasks sources and we prove that the scheduler could consider these sources as a single one, so a system based on queues is adequate for this problem. Section IV presents several experiments, considering four regional centers as a sporadic task sources, and we describe the migration behavior in order to ensure the deadlines. The paper ends with conclusion and future work.

II. CLASSICAL APPROACHES OF SPORADIC SCHEDULING FOR CLOUDS

There are several classical approaches for this 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 Polling Server functions as follows: at regular time intervals the server activates and services aperiodic tasks within its capacity. If no aperiodic task is waiting for execution, the server will suspend itself until the next period. Also, the Polling Server does not preserve its capacity, meaning that the remaining, unused capacity after the execution spree in each period will be lost [12].

The Deferrable Server (DS) algorithm was first introduced by Lehoczky, Sha and Stosnider in [13]. This technique is derived from the PS, and manifests improved response times for aperiodic tasks. As the PS, the DS algorithm creates a periodic task for servicing aperiodic requests. However, DS preserves its capacity if no requests are pending upon the invocation of the server.

The Priority Exchange Server (PES) algorithm is another implementation of a periodic server task used for servicing aperiodic requests. In this case, 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. At the beginning of each server period, the capacity of the server is replenished at full value. If aperiodic requests are pending and the server is ready, the requests are serviced using the available capacity. Otherwise, CS is exchanged for the execution time of the active periodic task with the highest priority. When this exchange occurs, the periodic task executes at the server’s priority level while the server accumulates a capacity at the priority level of the periodic task.

The Sporadic Server (SS) algorithm was introduced by Sprunt [14] in order to enhance the average response time of aperiodic tasks without degrading the utilization bound of the aperiodic task set. A server implementation, the algorithm is also based on a servicing task created for aperiodic requests. Like the DS, the algorithm preserves the server capacity at its high-priority level until an aperiodic request occurs. The main difference between SS and DS is the manner in which the server replenishes its capacity. In the case of DS and PES, the replenishment sets capacity at its full value, at the beginning of each server period.

A particular scheduling technique for aperiodic requests is the Slack Stealing (SSt) algorithm, introduced by Lehoczky and Ramos-Thuel in [15]. 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.

If we consider a set of n tasks \( \{ T_i \}_{1 \leq i \leq n} \) and having \( d_i \) the deadline of task \( T_i \), we can 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 a sporadic and 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 systems performance, the optimization methods must take into consideration the maximization of slacks. For a systems that will respect all deadlines (like a Cloud System with respect to Service Level Agreement) we have at any moment of time \( t \):

\[
\forall T_i, slack_i(t) \geq 0, c_i(d_i) \leq 0
\]

III. SPORADIC TASKS APPROACH WITH DEADLINE CONSTRANS

For concurrent access for tasks execution, Cloud environments could be modeled at PaaS level as a queuing system. For such type of systems, the number of task arrivals in a given interval of time is a random variable with a Poisson distribution [9], [10].

Let’s consider \( \tau \) the time between two successive arrivals and \( T \geq 0 \) a time threshold. We have the following result [11]:

\[
Prob(\tau \leq T) = 1 - \exp \{ -\lambda T \},
\]
so, if \( T \) is fixed a priori, the probability has a constant value.

Let’s consider a time interval with length \( t \). The Poisson distribution for the number of tasks arrivals in an arbitrary time interval with length equal with \( t \) is:

\[
\text{Prob}(N(t) = n) = \exp \left\{ -\lambda t \right\} \frac{(\lambda t)^n}{n!}, n = 0, 1, \ldots
\]

where \( \lambda t \) is the shape parameter which indicates the average number of events (tasks arrivals) in the given \( t \) time interval.

Let’s consider the 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: \( \forall T_i, d_i - a_i \leq t \).

Let’s consider \( m \) different tasks sources (different users that submit for execution a set of tasks in a sporadic way). For each source \( k \) we have a specific \( n_k \) number of tasks for an interval \( t \), and a specific \( \lambda_k \) parameter for Poisson distribution. Considering \( N_k(t) \) the number of tasks submitted by source \( k \) in the \( t \) interval, we have:

\[
\text{Prob}(N_k(t) = n_k) = \exp \left\{ -\lambda_k t \right\} \frac{(\lambda_k t)^{n_k}}{n_k!}.
\]

If \( n = \sum n_k \) is the total number of given tasks and \( N(t) = \sum N_k(t) \) is the total number of tasks arrived in the \( t \) interval, we have:

\[
\text{Prob}(N(t) = n) = \text{Prob}(N_1(t) = n_1, N_2(t) = n_2, \ldots).
\]

We have the following result:

\textbf{Theorem 1:} Let’s consider \( m \) sources of sporadic tasks with specific parameters \( (\lambda_k, n_k)_{1 \leq k \leq m} \) for Poisson distribution. For a scheduling system the \( m \) inputs appear as a single one with specific parameter \( (n = \sum n_k, \lambda = \sum \lambda_k) \) for Poisson distribution.

\[
\text{Proof:}\text{The proof is based on mathematical induction and it is similar with approach presented in [11].}
\]

For \( m = 1 \) we have one single task source, so the result is trivial.

For \( m = 2 \) we have \( n = n_1 + n_2 \) and \( \lambda = \lambda_1 + \lambda_2 \) and:

\[
\text{Prob}(N(t) = n) = \sum_{n_1=0}^{n} \frac{(\lambda_1 t)^{n_1}}{n_1!} \exp \left\{ -\lambda_1 t \right\} \frac{(\lambda_2 t)^{n-n_1}}{(n-n_1)!} \exp \left\{ -\lambda_2 t \right\} =
\]

\[
n^\lambda \exp \left\{ -\lambda \right\} \frac{1}{n!} \sum_{n_1=0}^{n} \frac{n!}{(n-n_1)!} \lambda_1^{n_1} \lambda_2^{n-n_1} =
\]

\[
n^\lambda \exp \left\{ -\lambda \right\} \frac{1}{n!} (\lambda_1 + \lambda_2)^n = \exp \left\{ -\lambda t \right\} \frac{(\lambda t)^n}{n!}.
\]

For \( m = p - 1 \) we consider the theorem to be demonstrated and we try to proof for \( m = p, p \geq 3 \). In this step we have:

\[
n = n_1 + n_2 + \ldots + n_{p-1} + n_p = \tilde{n}_{p-1} + n_p,
\]

so, we follow the same proof like in the case of \( m = 2 \). At the end we conclude that:

\[
\text{Prob}(N(t) = n) = \exp \left\{ -\lambda t \right\} \frac{(\lambda t)^n}{n!},
\]

so, all \( m \) tasks sources can be seen as a single one for a sporadic scheduler.

\textbf{Remark 1.} The result of \textit{Theorem 1} allow us to consider a queueing system for scheduling with a local coordinator for a regional center. In each regional center we have multiple task sources with different characteristics (different \( \lambda \) and \( n \) for a Poisson distribution).

For each source we can consider the following model for deadline scheduling. Let’s each task 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 \( A = \min \{a_i\} \) and global deadline \( D = \max \{d_i\} \). Considering \( f \) the fraction of input data that is given as output, we have \( output_i = f * data_i \). Now, let’s introduce the \( c_r \) the execution cost and \( c_{com} \) the communication cost. 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:

\[
TotalCost = \frac{1}{n_{res}} \sum_i (data_i c_r + output_i c_r) + \sum_i output_i c_{com}.
\]

where \( n_{res} \) is the number of resources in a regional center.

Considering all of these assumption, we have the following result:

\textbf{Theorem 2:} For a request \( Q \) and a homogeneous regional center with \( n_{res} \) resources, considering a schedule with deadline constrains, then:

\[
n_{res} \geq \frac{(1 + f)c_r \sum_i data_i}{D - A - f c_{com} \sum_i data_i}.
\]

\textbf{Proof:} A schedule with deadline constrains means:

\[
A + \text{TotalCost} \leq D,
\]

so, considering the defined \( \text{TotalCost} \), we have:

\[
A + \frac{1}{n_{res}} \sum_i (data_i c_r + output_i c_r) + \sum_i output_i c_{com} \leq D,
\]

\[
A + \frac{(1 + f)c_r}{n_{res}} \sum_i data_i + f c_{com} \sum_i data_i \leq D,
\]

\[
\frac{(1 + f)c_r}{n_{res}} \sum_i data_i \leq D - A - f c_{com} \sum_i data_i,
\]

\[
n_{res} \geq \frac{(1 + f)c_r \sum_i data_i}{D - A - f c_{com} \sum_i data_i}.
\]
Remark 2. If we have \( c_{com} = 0 \) (no communication) and defining \( \text{SerialCost}(Q) = (1 + f) c_r \sum_i data_i \), then:

\[
\begin{align*}
n_{res} & \geq \frac{\text{SerialCost}(Q)}{D - A} .
\end{align*}
\]

This remark could establish the minimum number of resources for BaTs Scheduling without communication.

Remark 3. The result of Theorem 2 allow as set the number of resources in a regional center as:

\[
\begin{align*}
n_{res} &= \left[ \frac{(1 + f) c_r \sum_i data_i}{D - A - f_{com} \sum_i data_i} \right] + 1
\end{align*}
\]

and, if we need more resources we will considering migration between regional centers. This assumption is based on the maximization of slacks approach.

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

For a heterogeneous environments the, we introduce \( c_r(T_i, R_j) \) a cost function that establish the execution cost of task \( T_i \) scheduled on resource \( R_j \). We can consider the expected time to compute \( ETC[i][j] = c_r(T_i, R_j) \) indicates the expected execution time of task \( i \) in resource \( j \) [16]. Following the same model, we can introduce a cost function that establish the communication cost to transfer \( data_i \) from resource \( R_i \) resource \( R_k \). \( R_j \) and \( R_k \) can be situated in different regional centers. Let’s \( c_{com}(T_i, R_j, R_k) \) to be the communication cost.

**Theorem 3:** \( n_{res}^h \geq n_{res}^o \).

**Proof:** In this theorem \( h \) means heterogeneous and \( o \) means homogeneous. For a \( h \)-regional center we can consider \( c_r = \max_j \{ c_r(T_i, R_j) \} \) and \( c_{com} = \max_{j,k} \{ c_{com}(T_i, R_j, R_k) \} \). These assumptions consider the worst case for scheduling problem. But, for a regional center, we can consider the same communication cost between resources so \( c_{com}(T_i, R_j, R_k) = c_{com} \) (a InfiniBand or Myrinet solution). The results of Theorem 1 can be reformulated as:

\[
\begin{align*}
n_{res}^h & \geq \frac{(1 + f) \sum_i c_r data_i}{D - A - c_{com} \sum_i data_i}.
\end{align*}
\]

Then, considering \( c_r = \min_j \{ c_r \} \) the execution cost for a \( o \)-regional center build with the best resources from a \( h \)-regional center, we have \( \forall_i, c_r \geq c_r \), so \( \sum_i c_r data_i \geq \sum_i c_r data_i \) and

\[
\begin{align*}
n_{res}^o & \geq \frac{(1 + f) \sum_i c_r data_i}{D - A - \sum_i data_i} \geq \\
& \frac{(1 + f) \sum_i c_r data_i}{D - A - \sum_i data_i} = \frac{(1 + f) c_r \sum_i data_i}{D - A - \sum_i data_i} = n_{res}^o
\end{align*}
\]

**Remark 4.** In general, in a \( h \)-regional center we need a higher number of resources for the same request \( Q \) in order to respect the deadline constrains. The \( o \)-regional centers are also always built with high processing capacity machines and high speed network.

IV. EVALUATION SCENARIOS AND RESULTS

INTERPRETATION

A. MONARC Simulator

MONARC 2 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 [3]. 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 centers: the system is composed of several interconnected regional centers. The model allows the simulation of various Grid architectures. For example, the centers can logically be hierarchical organized. The simulation framework includes several components, providing the necessary simulation components to accurately model various distributed systems experiments. A first set of components was created for describing the physical resources of the distributed system under simulation. The largest one is the regional center, which contains a farm of processing nodes (CPU units), database servers and mass storage units, as well as one or more local and wide area networks. Another set of components model the behaviour of the applications and their interaction with users. Such components are the “Users” or “Activity” objects which are used to generate data processing jobs based on different scenarios. The task is another basic component, simulated with the aid of an active object, and scheduled for execution on a CPU unit by a “Task Scheduler”.

B. Task Scheduling in MONARC

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. In this section we present the task scheduler component. The task scheduler is responsible with decisions such as whether its possible to execute a particular task at the current simulation moment of time or it should be placed in a waiting queue for later execution, on which CPU unit it must be executed or even to which other regional center the task should be transferred for remote execution. MONARC 2 provides a basic task scheduler class named taskScheduler. This class can be extended in order to implement other scheduling algorithms. By extending this class the user can make use of its own scheduling algorithm. The basic taskScheduler class allows the tasks to be executed only in the regional center where they have been submitted. There are two scheduling classes that are straightforward available to be
used. The first one implements a basic task scheduler, while the second one implements a very simple distributed scheduling algorithm. In the following we present the basic task scheduler, as well as the distributed scheduler implementations.

The basic task scheduler implements a rudimentary decision making algorithm. As output, the 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 later resubmission. The rudimentary scheduling implementation considers only the set of processing units belonging to the local regional center. According to the algorithm, when a new task is submitted, the scheduler first tries to find an available CPU unit to execute the task. The task might need to be executed on a specific CPU unit, and in this case the scheduler doesn’t search anywhere it knows exactly where to send it. The decision about the CPU unit is based on the total amount of memory used by the tasks that are already running on the CPU. A task can be executed on a CPU unit if the memory needed by the task doesn’t exceed the total amount of memory available on that particular CPU unit.

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. The scheduler looks for the CPU unit having the minimum load and that has enough memory to execute the new task. If it doesn’t find such a CPU, or if the task must be executed on a specific CPU and that CPU is unavailable, the task is placed into a waiting queue. The waiting queue keeps the tasks ordered by their priorities, so the first task that will be extracted from the queue will be the having the highest priority. A task is extracted from the queue when a processing unit becomes available, possible because of other tasks finishing their execution.

MONARC 2 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 removes any unnecessary execution loops from the scheduling algorithm. This basic distributed algorithm is provided only as an example of the possibilities offered by MONARC 2 to handle various scheduling algorithms. The user can easily extend this model to include various new conditions, new resource considerations or performance metrics, in order to test the behavior of new scheduling technologies.

Various scheduling procedures were successfully tested using MONARC 2 and we present in the next sections several of the most successful obtained results [18].

C. Simulation Setup

This simulation tests the migration of the sporadic tasks scheduling between regional centers. We used 4 regional 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 sporadic behaviour. The time intervals respect 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 you can set from the configuration file in the simulation are:

- 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.

These experiments demonstrate the capability of MONARC 2 to validate various scheduling technologies. The section also presents an experiment that involved a comparison of several real-world scheduling decision algorithms. These experiments demonstrate the capability of MONARC 2 to perform, based on various analyzed
characteristics, a complete survey of particular scheduling technology solutions.

Figures 1 and 2 show the evolution of submitted and finished jobs for 2 regional centers from UPB (UPB_01 and UPB_02). We can observe that there is a periodicity in tasks submission and a slow increasing at the end of the period. The migration process start here. The same behavior can be seen for DERBY regional centers (DERBY_01 and DERBY_02) in Figures 5 and 6.

Figures 3 and 4 show the evolution of running and waiting tasks for UPB regional centers. We can see that there are some 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. A similar behavior can be seen for DERBY regional centers in Figures 7 and 8. Here during the interval that regional centers work at full capacity the number of waiting task are higher than full capacity, so scheduler will activate the migration function.
Figures 9 and 10 and Figures 11 and 12 highlight the migration function. In the time period when UPB and DERBY regional centers worked at full capacity, if we the tasks remained in the waiting queues the deadline constrains are not satisfied. So, all regional centers start the tasks migration and several tasks become submitted tasks for other regional centers. All the submitted tasks in the initial phase or in the migration phase are sporadic tasks that were submitted in the regional centers during the experimental interval. All tasks flow were modeled according with Poisson distribution.

V. DISCUSSION AND CONCLUSIONS

This paper presents the classical approach for sporadic and aperiodic task scheduling considering a scheduling
system for periodic tasks. An important result that describe the behavior of scheduler in the context of multiple heterogeneous tasks source and the possibility to considers the tasks source as a single one. This make the scheduling system a queuing one. The optimization of slacks was introduced and, based on this idea, the experimental results introduced the migration function. There is no possibility to have an optimal scheduler. In fact, no optimal scheduling scheme does exist for aperiodic tasks, as the Tia-Liu-Shankar theorem in [17] states: “For any set of periodic tasks ordered on a given fixed-priority scheme and aperiodic requests ordered according to a given aperiodic queueing discipline, there does not exists any valid algorithm that minimizes the response time of every soft aperiodic request”.

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 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.

As future work we will consider the tasks’ assignment in a regional center, considering statistical methods. Also, a broker architecture and a communication protocol will be considered for hybrid Clouds.

ACKNOWLEDGMENT

This work was supported by project “ERRIC - Empowering Romanian Research on Intelligent Information Technologies/FP7-REGPOT-2010-1”, ID: 264207. The work has been co-funded by the Sectorial Operational Program Human Resources Development 2007-2013 of the Romanian Ministry of Labor, Family and Social Protection through the Financial Agreement “POS-DRU/89/1.5/S/62557”. Also, the first author would like to thank UNITE FP7- 248583 for their secondment support.

REFERENCES


