Resource Allocation for Real-Time Tasks using Cloud Computing

Karthik Kumar, Jing Feng, Yamini Nimmagadda, and Yung-Hsiang Lu
School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, 47907.

***Abstract*—This paper presents a method to allocate resources for real-time tasks using the "Infrastructure as a Service" model offered by cloud computing. Real-time tasks have to be completed before deadlines, and cloud computing offers selection of resources with different speeds and costs. In cloud computing, resource allocations can be scaled up based on the requirements; this is called *elasticity* and is the key difference from existing multiprocessor task allocation. Scalable resources make economical allocation of resources an important problem. We analyze the problem of allocating resources for a set of real-time tasks such that the economic cost is minimized and all the deadlines are met. We formulate the problem as a constrained optimization problem and propose a polynomial-time solution to allocate resources efficiently. We compare the economic costs and performance provided by our solution with the optimal solution and an EDF (earliest deadline first) method. We show how the cost varies based on the distribution of the tasks.**

***Index Terms*—resource allocation; cloud computing; scheduling;**

I. Introduction

Cloud computing offers a user the service (called "Infrastructure as a Service" - IaaS) of renting computing resources over the Internet. The user can select from different types of computing resources based on the requirements. For example, Amazon's EC2 Cloud service provides different options for selecting resources as shown in Table I. The user can rent an arbitrarily large number of resources: this is called *scalable computing* or *elasticity*, since the number can be scaled dynamically to meet the requirements.

One set of applications that can benefit from scalable computing are mixed-parallel applications [5], [12], [16], [18], [20]. These applications exhibit high task and data parallelism. Many of these applications are real-time [5], [12], [20] and require their workloads to be completed before deadlines; examples include voice and object recognition, image and video retrieval, navigation systems, financial systems, and mission-critical systems. For example, an object recognition engine [20] may be hosted on the cloud. Each task is an object query; the object must be recognized within a specified time duration to be of value to the user. Since the cloud offers scalable computing, resource allocation can be scaled based on the arrival times, workloads, and deadlines of the tasks.

In cloud computing, a resource is a virtual machine that guarantees a certain level of performance to the user. For example, Amazon's EC2 cloud service defines virtual machines with speeds in "compute units"; this provides a hardware-independent definition of speed for the virtual machine by abstracting away variations in the underlying physical hardware. The types and amounts of virtual machines allocated determine the cost paid by the user. Amazon's costs for different virtual machines are shown in Table I. There are three different types– standard, high memory, and high CPU, including different amounts of processors, memory, and storage at different costs. Each virtual machine is rented multiple hours, and the user is charged a fixed cost irrespective of the virtual machine's utilization within the hours. This motivates the need to find a cost-efficient allocation for a given set of tasks. In the rest of this paper, we use the terms "resources", "virtual machines (VMs)", and "processors" interchangeably.

Virtual Machine type	Compute Units	Memory	Storage	Cost/ hour
High Memory	6.5	17.1 GB	420 GB	$0.69
Standard	8.0	15.0 GB	1690 GB	$1.04
High CPU	20.0	7.0 GB	1690 GB	$1.24

TABLE I
Virtual machine types available in Amazon's EC2 Cloud Service (size Extra Large, US-N California). http://aws.amazon.com/ec2/pricing/

Several researchers have developed efficient allocations for real-time tasks on multi-processor systems [7], [23], [13]. However, previous studies schedule tasks on a *fixed number* of processors. For scalable computing, the virtual machines (VMs) are rented, and *can be scaled up to any number*. This creates some fundamental changes in the problem. First, it implies that if a task is sufficiently parallelizable, its deadline can always be met since more VMs can be allocated to complete the task before its deadline. This is different from previous studies that examine the schedulability on a given number of processors. Second, since the number of available VMs is nearly infinite, at every time instant, there are options using different numbers and types of VMs, based on their computing speeds and costs. For example, to finish a task, a user can select a larger number of slower, cheaper VMs or a smaller number of faster, more expensive VMs, or a combination between them. Third, acquiring VMs by rent implies a fixed charge for a given rental period. Suppose a user rents a VM for an hour and the task completes before the end of the hour, the VM becomes available for running other tasks that arrive within the hour. Thus the allocation of resources for one task at the *present* may affect the selection of resources for *future* tasks. Recent studies on allocating cloud VMs for real time tasks [19], [12], [14], [3] focus on different aspects like the infrastructures to enable real-time tasks on VMs, selection of VMs for power management in the data center, etc. Unfortunately, none of these studies considers how the user can make a cost-efficient selection from a set of different VMs for real-time tasks.

In this paper, we develop an algorithm for allocating VMs to applications with real-time tasks. The allocation is formulated as a constrained optimization problem. Since an exhaustive search for solutions has exponential complexity, we propose a polynomial-time heuristic to solve the problem. We compare the cost obtained by our heuristic with the optimal solution, and an Earliest Deadline First (EDF-greedy) strategy. We show the conditions when our heuristic outperforms the EDF-greedy strategy. We also perform a sensitivity analysis for the

978-1-4577-0638-7 /11/$26.00 ©2011 IEEE

Papers	Heterogeneous Processors	Workload	Number of Processors
[9]	Yes	Fixed	Fixed
[22], [4]	No	Probabilistic	Fixed
[7], [23], [13]	No	Fixed	Fixed
This paper	Yes	Fixed	Scalable

TABLE II
COMPARISON OF DIFFERENT STUDIES ON REAL-TIME SCHEDULING.

parameters of the problem.

The remainder of this paper is organized as follows: Section II describes the types of applications that can be used for our problem, and highlights our contributions by comparing it with related work. Section III describes our problem formulation, and the proposed solution. Section IV describes the evaluation of the proposed solution, and Section V concludes the paper.

II. Background and Related Work

A. Scheduling Parallel Applications

Scheduling mixed-parallel applications [16], [18] on multiprocessor systems is a known NP-Complete problem [16]. Many of these parallel applications are real-time [12], [5], [20]. The tasks in these applications have two important characteristics: (1) highly parallelizable [11], [2], [16] and (2) real-time constraints (or deadlines) [15], [17]. Each task can be partitioned into smaller and parallel units. We use p as the smallest unit of computation. For example, *image retrieval* may compare images with a query to find a match, and the smallest unit of computation is comparing a single image with the query. The task finds a match for a query image, called img, from a collection of one million images. This task must be completed within d seconds. We may use one million VMs and each compares only one image. If all VMs can execute simultaneously and the storage system can provide a sufficient bandwidth, this would be the fastest approach but it would also use the largest number of VMs. Since we only need to find a match for img (yes/no), the results can be merged very quickly in logarithmic time. Another option uses 1000 VMs and each VM compares 1000 images. It is also possible to use a single VM for the one million images; this will take much longer. The actual selection of VMs may be constrained by the deadline d, the speeds and cost of different types of VMs. In this paper, we consider cost-efficient VM selection for such applications.

B. Scheduling for Multiprocessors

Previous studies have considered energy-efficient scheduling. Uniprocessor scheduling schemes like Earliest-Deadline First (EDF) [8] and Rate Monotonic (RM) scheduling [1] are adapted to multiprocessor systems. Many researchers [7], [4], [23], [13] consider real-time task allocation on multiprocessor systems. Several papers have studied Dynamic Voltage and Frequency Scaling (DVFS). We do not consider DVFS in this paper as it is a well studied problem. Table II shows how our work differs from the current real-time multiprocessor scheduling techniques.

C. Virtual Machine Allocation for Cloud Computing

This includes two different problems: (1) selecting virtual resources by the user and (2) mapping virtual resources to physical resources by the service provider. This paper focuses on the first problem. Several studies [10], [21], [6] consider cost-effective resource selections for cloud systems. The common focus of all these works is cost-efficient VM allocation; the key difference from the above works is that we consider real-time tasks.

Recent studies have been performed on allocation of VMs for real time tasks. Aymerich et al. [3] develop an infrastructure for deploying real-time financial tasks on cloud systems. Tsai et al. [19] discuss real-time partitioning of database tasks on cloud infrastructures. Liu et al. [14] show how to schedule real-time tasks with different utility functions; however they do not consider different types of VMs. The closest work to ours is that of Kim et al. [12]; they consider scheduling real-time tasks on cloud systems. However, in their work the real-time constraint is specified in a service level agreement (SLA). In such cloud models, the user does not select individual VMs and VM allocation is left to the service provider. Their work examines power management while allocating VMs to meet the SLA. In our work, we consider the IaaS model where the user selects and pays the cost for the VMs (similar to Amazon's model), and we propose a scheme to reduce cost for the user. Thus the work of Kim et al. [12] benefits the service provider and our work benefits the user.

This paper has the following contributions: (1) This is the first paper to consider cost-efficient resource allocation in heterogenous cloud for real-time tasks. (2) We formulate resource allocation as a constrained optimization problem. (3) We propose a polynomial-time algorithm to allocate VMs while meeting tasks' real-time constraints and we show how the cost varies based on the distributions of the task sets.

III. Real-time Cloud Scheduling

This section describes the problem of scheduling real-time tasks. Sections III-A and III-B define the problem, and show a simple example. Section III-C formulates a constrained optimization problem. Section III-D describes a greedy solution and Section III-E presents our algorithm to solve the problem.

A. Problem Definition

The application has a set of tasks T. Each task $\alpha_i \in T$ has an arrival time a_i and a deadline d_i (specified in minutes). We use *cycles* to quantify the task's workload w_i; the usage of *cycles* in this context is a general measure of the amount of computation required for the workload. The workload of each task is parallelizable into smaller units (subtasks); the size of the smallest possible subtask is p. R is the set of available VMs. Each VM $v_j \in R$ has a computation speed s_j and the corresponding cost c_j. The speed s_j is the number of cycles the VM can complete per minute. The user is charged the cost c_j for renting v_j for D minutes continuously, regardless of the utilization within the interval. D is the minimum time unit for renting. We assume that task α_i can always meet its deadline d_i if enough VMs are allocated. The condition is $\forall i, \exists j, |d_i - a_i| \geq |\frac{p}{max(s_j)}|$; in other words, the fastest VM $max(s_j)$ can compute the smallest subtask p before the deadline. The problem can be stated as follows: *find an offline mapping from T onto a subset of R to minimize the overall cost while meeting the deadlines of all tasks.*

B. Motivating Example

We describe a simple motivating example. Table III shows the speeds and the costs of the resources available for selection. The costs are in the same ratio as table I. The value of D is 60 minutes and p is 10. We consider a single task α_1 with an arrival time a_1 of 0 min, workload w_1 of 600 cycles,

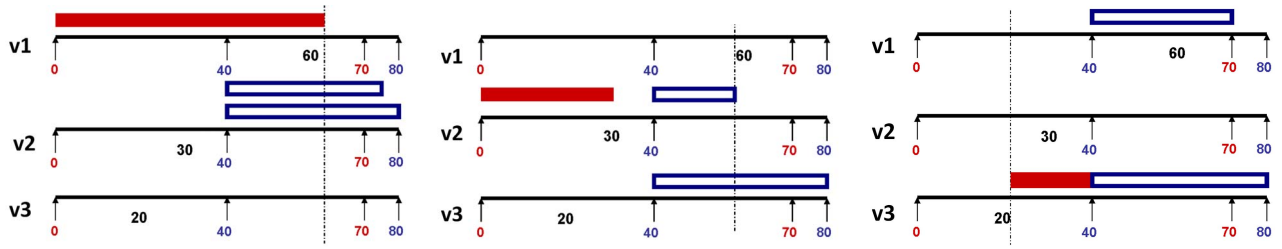


Fig. 1. Allocation of three resources v_1, v_2, and v_3 for two tasks: α_1 is shown in solid rectangles, and α_2 in empty rectangles. (a) Assuming v_1 is allocated to α_1 from 0 to 60 min, allocating two of v_2 at 40 min to α_2. The cost is $1 + 2 \times 1.5 = 4$. (b) If v_2 is allocated to α_1, v_2 can be also used for α_2 from 40 min to 60 min. This along with one of v_3 at 40 min can complete α_2. The cost is $1.5 + 2 = 3.5$. (c) If v_3 is allocated to α_1 , v_3 can also be used for α_2 from 40 min to 80 min. This along with one of v_1 at 40 min can complete α_2 with the lowest total cost $2 + 1 = 3$.

and a deadline d_1 of 70 min. Task α_1 takes 60 ($=\frac{w_1}{s_1}$), 30 ($=\frac{w_1}{s_2}$), and 20 ($=\frac{w_1}{s_3}$) minutes on v_1, v_2, and v_3 respectively. The objective is to reduce the cost while meeting the deadline. A greedy schedule selects the resources with the lowest cost while meeting α_1's deadline. Renting v_1 from 0 min to 60 min can finish α_1 before d_1 with the lowest cost. For each task α_i, we use c_{α_i} to denote the cost for the task to meet its deadline and $c_{\alpha_1} = 1$.

VM v_j	$s_j = \frac{\text{cycle}}{\text{min}}$	$c_j = \frac{\text{cost}}{\text{hour}}$
v_1	10	1.0
v_2	20	1.5
v_3	30	2.0

TABLE III
RESOURCES AVAILABLE FOR SELECTION

Task α_i	Arrival a_i	Deadline d_i	Workload w_i
α_1	0	70	600
α_2	40	80	1500

TABLE IV
TASKS TO BE COMPLETED

Next, we consider task α_2 with arrival time a_2 = 40 min, workload w_2 = 1500 cycles, and deadline d_2 = 80 min. We use the selection v_1 made earlier for α_1, and examine how to allocate resources to execute α_2. We need to complete α_2 in $d_2 - a_2$ = 40 minutes; this can be accomplished with four v_1 ($\frac{1500}{10}$=150 minutes, $\lceil\frac{150}{40}\rceil = 4v_1$), two v_2 or, two v_3. The corresponding costs for both tasks are 1+4=5, 1+3=4, and 1+4=5 respectively. If we consider using more than one type of VM for α_2, using one v_3 and one v_2 reduces the cost to 1+(2+1.5) = 4.5. Using one v_3 and one v_1 further reduces the cost to 1+(2+1) = 4 and this is the lowest total cost. Figure 1(a) shows the example with $c_{\alpha_1} + c_{\alpha_2}$=4, using v_1 for α_1 and two of v_2 for α_2.

If we consider both tasks simultaneously, a better solution allocates v_2 to α_1, and v_3 to α_2, shown in Figure 1(b). This results in $c_{\alpha_1} + c_{\alpha_2}$ =1.5 + 2 = 3.5, while meeting both deadlines. The cost is lower because α_1 utilizes v_2 from 0 min to 30 min, and α_2 can be run on v_2 from 40 min to 60 min, thus completing 400 cycles of α_2. One v_3 can be rented at 40 min, and the remaining 1100 cycles of α_2 can be run on v_3 before the deadline d_2. The example shows that a greedy strategy may not give the lowest total cost. The tasks must be considered *simultaneously* to minimize the total cost. In the next section, we formulate resource allocation as a constrained optimization problem.

C. Problem Formulation

We formulate the problem described in Section III-A as a constrained optimization problem. Each type of VM $v_j \in R$ has speed s_j and cost c_j, j = 1, 2, ... x. We may select any number of VMs of a given type and we use indicator variables $\delta(j,k)$ to represent the selection of VMs. More specifically, $\delta(j,k)$ = 1 if the k VMs of type v_j are used, where $k = 1, 2, 3....$ For example, $\delta(2,3) = 1$ indicates that three v_2's are used. The objective function is to minimize the total cost. This may be given by summing the costs of all the selected VMs as shown in equation (1):

$$\min \sum_{j=1}^{x} \sum_{k=1}^{y} c_j \times \delta(j,k) \tag{1}$$

The outer loop in equation (1) corresponds to the x different types of VMs, and the inner loop summed to y corresponds to the number of available VMs of each type. Since cloud computing offers the selection of an arbitrary number of VMs, the value of $y = \infty$. In reality, we can derive an upper bound for y for a given set of tasks T. The total computation for T is given by $C = \sum_{i=1}^{|T|} w_i$. The computation C is partitioned among the VMs. If we consider the extreme case that each VM performs the smallest unit p, then the maximum number of VMs that can be used for T is $|\frac{C}{p}|$. The $(|\frac{C}{p}| + 1)^{th}$ VM cannot be used as each of the previous $|\frac{C}{p}|$ VMs is already performing the smallest unit of computation; there is no more computation for any additional VMs. Thus we have an upper bound for y to be

$$y \leq |\frac{C}{p}| = |\frac{1}{p} \times \sum_{i=1}^{|T|} w_i|. \tag{2}$$

Equation (1) represents the total cost to be minimized and is the objective function. We now examine the constraints: all tasks need to meet their deadlines. We need to select sufficient VMs in equation (1) such that each task α_i can be completed after its arrival time a_i and before its deadline d_i. The δ in equation (1) only indicates the type and the number of VMs allocated; they do not indicate *when* the VMs are allocated. The time of allocation is important to ensure that the VMs allocated for α_i are available after a_i and before d_i. In order to consider the time of allocation, we make a restriction (for now) that each allocated VM is available for 1 minute (D=1), and we introduce a new set of indicator variables θ that includes

the temporal information: $\theta(j,k,m)$ = 1, if the k VMs of type j are available at the m^{th} minute.

Reconsider the example in Section III-B. For task α_1, $a_1 = 0$, $d_1 = 70$, $w_1 = 600$, and $p = 10$. To ensure this task is completed before its deadline, VMs must be allocated such that the number of cycles completed in the interval between a_1 (0 min) and d_1 (70 min) is at least w_1 (600 cycles). The number of cycles completed at each minute in this interval depends on the speeds of the VMs allocated at each minute within the interval; formulating this using θ gives

$$\sum_{j=1}^{x}\sum_{k=1}^{y} s_j \times \theta(j,k,m) \tag{3}$$

cycles at the m^{th} minute. In this example, x is 3 because there are three different types of VMs. From equation (2), the value of y cannot exceed $|\frac{1}{10} \times 600| = 60$. This means that the maximum number of VMs we can use is 60, with each computing 10 cycles. In equation (3), if $\theta(2,1,1) = 1$ and $\theta(1,k,1) = \theta(3,k,1) = 0$ for $1 \leq k \leq 60$, the speed at the 1^{st} minute is $s_2 = 20$ cycles/min. The amount of work that can be done during this minute is 20 cycles. To ensure that at least 600 cycles are completed for the task, we need to make sure the summation of work done at each minute from 0 to 70 is at least 600. To generalize this across the duration of task $\alpha_i \in T$, we obtain the following requirement, for each α_i:

$$\sum_{m=a_i}^{d_i}\sum_{j=1}^{x}\sum_{k=l_i}^{y} s_j \times \theta(j,k,m) \geq w_i \tag{4}$$

The lower limit of k for the i^{th} task equals l_i because some θ are already allocated for previous i-1 tasks (up to task α_{i-1}); this ensures that θ for the current task α_i does not overlap with the θ allocated for tasks α_0, α_1 ... α_{i-1}. This prevents sharing VMs among tasks; however, our assumption that $D = 1$ ensures that each VM is already fully utilized at the minute it is available.

Equation (4) is based on the assumption that D=1. In order to generalize our formulation such that a VM is available for D minutes, we need to define a function to represent a time interval of D minutes. We use the window function $u(m) - u(m-D)$, where $u(m)$ is the unit step function. We now need to find the number of "D minute windows" to perform the same amount of computation as the allocations for θ in equation (4). In order to do this, we define equations (5) and (6) for each VM type j. At each time instant m, $\theta(j,k,m)$ should be less than or equal to $n_{(m-z)j}$; $n_{(m-z)j}$ is the smallest number of D minute windows required to do the same work as $\theta(j,k,m)$ at time m. We use a subscript of $m-z$ because a D-minute window that begins z minutes *before* m, can be used at m, as long as $z \leq D$. The value of m is between the arrival of the first task a_1 and the deadline of the last task d_N.

$$\begin{aligned} &f(m,z) = u(m-z) - u(m-z+D) \\ &\sum_{m=a_1}^{d_N}(\sum_{k=1}^{y}\theta(j,k,m) - \sum_{z=0}^{D} n_{(m-z)j} f(m,z)) \leq 0 \end{aligned} \tag{5}$$

To find the total number of VMs of each type j that are required, we define

$$t_j = \sum_{m=a_1-D}^{d_N} n_{mj} \tag{6}$$

Speed	Cost	VMs used
10	1.0	v_1
20	1.5	v_2
30	2.0	v_3
40	3.0	v_1 + v_3
50	3.5	v_2 + v_3
60	4.0	v_3 + v_3
70	5.0	v_1 + 2 × v_3
80	5.5	v_2 + 2 × v_3
90	6.0	v_3 + 2 × v_3
...		

TABLE V
LOOK-UP TABLE FOR DIFFERENT SPEED REQUIREMENTS AND THE CORRESPONDING COSTS AND VMS.

Equation (6) sums all the n_{mj} at every possible m, ranging from D minutes before the arrival of the first task, to deadline of the last task. This sum t_j gives the total number of VMs of type j that are required. In order to make this selection a minimum cost selection in equation (1), we set

$$\delta(j,k) = 1 \quad \forall k \leq t_j. \tag{7}$$

The solution to this ILP formulation is intractable. In Sections III-D and III-E, we describe a greedy algorithm and our polynomial time heuristic to solve this problem.

D. EDF-Greedy Algorithm

The first solution in Section III-B can be described as a greedy strategy based on EDF (earliest deadline first). The tasks are considered in the order of their deadlines. The strategy first tries to allocate a task to VMs available from the allocations for previous tasks. If these VMs are insufficient to complete the task before its deadline, the strategy selects the cheapest set of VMs that can complete the task before the deadline.

To find the cheapest set of VMs for each task, we construct a lookup table based on the speeds and costs in Table III. The lookup table contains a range of possible computing speeds constructed from the different types of VMs. In Table III, the VMs have speeds $\{10, 20, 30\}$. Using these speeds as the base, we can obtain the combinatorial set of speeds $S = \{10, 20, 30, 40, 50, 60, ...\}$. For each speed in S, the VM set to give that speed is identified and stored with that speed in Table V. For example, to achieve speed of 50, we can use one v_2 and one v_3 with cost = $2 + 1.5 = 3.5$, or two v_2's and one v_1 with cost = $2 \times 1.5 + 1 = 4$. The former has a lower cost and it is a better allocation. The entries in the lookup table are sorted in order of their speeds.

For a given task's workload, the greedy strategy searches the lookup table to find the lowest speed that can finish the workload before the deadline. For task α_1 in Section III-B, the lowest speed needed to finish the task of $w_1 = 600$ cycles in $d_1 - a_1 = 70$ minutes is 10 cycles/min. If d_1 is changed to 10 min, then the lowest speed needed to compute 600 cycles in 10 min in the table is 60 cycles/min. If this lowest speed in the lookup table is s_{α_i}, the corresponding cost is c_{α_i}. The lookup table is finite because the number of rows is bounded by the highest speed needed to satisfy any task. This bound is computed by

$$\max(\frac{w_i}{d_i - a_i}), \forall \alpha_i \in T. \tag{8}$$

This guarantees that the greedy strategy can find a VM allocation for all the tasks. The complexity of computing the

lookup table depends on the number of types of VMs x. In the above example, $x = 3$ and it is observed that the VMs have speed-cost ratios of 10, 13.33, and 15 for v_1, v_2, and v_3 respectively. We construct the lookup table by trying to allocate VMs in decreasing order of their speed-cost ratios, i.e., we try to allocate as many of v_3 before we allocate v_2, and so on. To get a speed s_x, we can compute the required VMs to be $\lfloor \frac{s_x}{s_3} \rfloor$ = t_3 of v_3, $\lfloor \frac{s_x-(t_3 \times s_3)}{s_2} \rfloor$ = t_2 of v_2, and $\lfloor \frac{s_x-(t_3 \times s_3+t_2 \times s_2)}{s_1} \rfloor$ of v_1. Thus a speed of 80 is obtained by using $\lfloor \frac{80}{30} \rfloor$ = 2 of v_3 (t_3=2), and $\lfloor \frac{80-2\times30}{20} \rfloor$ = 1 of v_2, and $\lfloor \frac{80-(2\times30+1\times20)}{10} \rfloor$ = 0 of v_1, giving $v_2 + 2 \times v_3$, as seen in Table V. Similar formulations may be obtained for different types of VMs with different speeds and costs.

E. Allocation Considering Temporal Overlaps

The optimization problem formulated in Section III-C may be solved by exhaustive combinations of allocating different types of resources to all tasks at various time instants. This approach unfortunately is intractible. The greedy algorithm in Section III-D allocates VMs for each task separately. When the tasks overlap in time, this greedy strategy may produce under-utilized and higher-cost allocations. This section describes our polynomial-time algorithm that considers all tasks together in the order of their deadlines. For each task α_i, the algorithm (1) identifies overlapping tasks in the future and (2) allocates resources considering these tasks.

(1) *Identifying Overlapping Tasks:*
For a given task α_i, we consider the tasks in the future whose deadlines are after d_i and examine whether these tasks temporally overlap with α_i. Task α_j overlaps temporally with α_i if $a_j < d_i + D$. This means that a VM can be allocated such that it is shared by α_i and α_j. We use T_i to denote the set of tasks that overlap with α_i:

$$T_i \leftarrow \{\alpha_j | d_j \geq d_i, \text{ and } a_j < d_i + D\} \tag{9}$$

(2) Allocating Resources:
First, we use Table V to obtain the lowest speed s_{α_i} required to compute task α_i (Section III-D). Next, we examine if allocating a VM with speed greater than s_{α_i} for task α_i can benefit the overlapping tasks in T_i. We consider each VM speed $s \geq s_{\alpha_i}$ in the lookup table and compute the VM overlap time t_{ij} for each task $\alpha_j \in T_i$. We define t_{ij} as the amount of time the VM allocated for α_i is still available for running α_j. The value of t_{ij} depends on the arrival times, workloads, and deadlines of α_i and α_j, and the speed s of the VM. It is calculated by:

$$t_{ij} = \begin{cases} min(D - D_{ij}, d_j - a_j) & \text{if } a_j - a_i \geq \frac{w_i}{s} \\ D - D_{ij}, & \text{if } a_j - a_i < \frac{w_i}{s}, d_j - a_i \geq D \\ d_j - (a_i + D_{ij}), & \text{if } a_j - a_i < \frac{w_i}{s}, d_j - a_i < D \end{cases}$$

where

$$D_{ij} = \frac{w_i}{s} + max(a_j - d_i, 0). \tag{10}$$

In the above equation, the time to complete the α_i is $\frac{w_i}{s}$ min. The first case shows the condition when the arrival times of the two tasks are sufficiently apart such that α_i is completed before the arrival time of α_j, i.e. ($a_j - a_i \geq \frac{w_i}{s}$). Task α_j can use the VM for t_j min. The value of t_j is given by the lesser of two terms, $D - D_{ij}$, and $d_j - a_j$. The first term $D - D_{ij}$ denotes remaining time for the VM for α_j, after taking into account the time it is used by α_i, = $\frac{w_i}{s}$, and a possible gap $a_j - d_i$ between the deadline of the first task and the arrival of the second task; during this time, the VM cannot be used by both tasks. For example, assume the first task uses $\frac{w_1}{s} = 10$ min of the VM, must be completed by $d_1 = 20$ min, and the second task arrives at time $a_2 = 40$ min, and must be completed by $d_2 = 60$ min. Then the value of D_{12} from equation (10) is $10 + max((40 - 20), 0)$=30, and $D - D_{12}$ is $60 - 30$=30. However, $d_2 - a_2$=$60 - 40$=20. Thus, the VM that is available for 30 min at a_2 is used only for min(30,20) minutes, and this is t_{12} for the first case in the above equation. The other two cases are similarly constructed. Note that the VM may not be used for the entire t_{ij} if task α_j does not have sufficient workload w_j to utilize the VM for t_{ij} min. In such a scenario, the temporal overlap by α_j is reduced from t_{ij} to $\frac{w_j}{s}$ where s is the speed of the VM being considered, $s \geq s_{\alpha_i}$. We use the O to represent the actual overlap.

$$O \leftarrow min(\frac{w_j}{s}, t_{ij}), \text{ for } s \geq s_{\alpha_i} \tag{11}$$

The smaller of $\frac{w_j}{s}$ and t_{ij} is the amount of time a VM allocated for a previous task can be actually used for a future task. Based on the overlap O, we compute a revised cost for the current task α_i. The revised cost c'_{α_i} is *not* the actual cost paid for the resources, and is used only for the purpose of decision making in our algorithm. The cost c'_{α_i} is calculated

$$c'_{\alpha_i} \leftarrow \min\{c_{\alpha_i} \times (1 - \frac{O}{D})\}, \forall \alpha_j \in T_i, \forall s \geq s_{\alpha_i}. \tag{12}$$

In the above equation, the cost c'_{α_i} is obtained by reducing c_{α_i} based on how much the VM overlaps with tasks in the future $(1 - \frac{O}{D})$. Since we select the resources corresponding to c'_{α_i} for each task α_i, this encourages selection of VMs that can be used by other tasks in the future. Our algorithm restricts the set of speeds searched to the size of the lookup table. Further, our algorithm considers the set T_i (all tasks that overlap with the current task) while making an allocation for α_i. A more exhaustive analysis could further consider that each task $\alpha_j \in T_i$ has a corresponding set of tasks T_j that is also affected by the allocation for α_i, and so on. Considering this would make our algorithm closer to exhaustive search.

To illustrate our algorithm, we consider the tasks α_1 and α_2 from Section III-B and the value of $D = 60$. Originally, we had assigned v_1 to α_1 since it has the lowest cost (c_{α_1}= 1), and finishes before the deadline. We now examine the costs for VM allocation to α_1 for the temporal-overlap algorithm. The overlap between α_1 and α_2 begins at 40 min. If we allocate v_1 for α_1, the cost remains the same ($t_{12} = 0, O = 0$ and c'_{α_1}=$c_{\alpha_1} = 1$) since α_1 uses v_1 for the entire 60 min, and v_1 cannot be made available for the overlapping task α_2. Allocating v_2 for α_1 earlier resulted in $c_{\alpha_1} = c_2 = 1.5$. If we allocate v_2 for α_1 in the temporal-overlap scheme, allocating it from 10 min to 70 min results in v_2 being available for α_2 from 40 min to 70 min ($t_{12} = 30$). The temporal overlap $O = min(30, \frac{1500}{20}) = 30$. The cost for this allocation is $c'_{\alpha_1} = c_2 \times (1 - \frac{O}{D}) = 1.5 \times (1 - \frac{30}{60}) = 0.75$, which is less than the previous option.

Allocating v_3 for α_1 earlier resulted in $c_{\alpha_1} = c_3 = 2$. For the temporal-overlap scheme, if we allocate v_3 from 20 min to 80 min, α_1 is completed from 20 min to 40 min, and v_3 is

available for α_2 from 40 min to 80 min. This results in a cost of $c'_{\alpha_1} = c_3 - c_3 \times (\frac{40}{60}) = 2 - 2 \times (\frac{40}{60}) = 0.67$, thus making the greedy selector choose v_3 for α_1 over v_1 and v_2. Assigning v_3 to α_1 as shown in Figure 1(c) results in 1200 cycles of α_2 being completed from 40 min to 80 min; allocating v_1 for α_2 at 40 min can complete the task with $c_{\alpha_2} = 1$. The actual cost for α_1 and α_2 using v_1 and v_3 is $c_{\alpha_1} + c_{\alpha_2}$= 2+1=3, lower than the total costs obtained in Section III-B.

IV. Evaluation

In this section, we describe how we evaluate the different scheduling algorithms. Section IV-A describes the evaluation setup. Sections IV-B compares the costs obtained by the algorithms. Sections IV-C and IV-D describe our sensitivity analysis, and Section IV-E compares the complexity.

A. Setup

We consider three algorithms for our evaluation: (1) EDF-greedy (called "greedy") described in Section III-D, (2) temporal-overlap (called "overlap") in Section III-E, and (3) exhaustive search, considering all possible combinations to obtain a minimum-cost allocation. It uses nested loops of the following, (1) each task, (2) each type of VM, (3) any number of resources of a given type, and (4) every possible temporal allocation of each VM.

The inputs to the schemes are the speeds and costs of the resources available, and the arrival times, workloads, deadlines of the tasks. We use the speeds and costs shown in Table III for our evaluation. We choose a Poisson distribution with parameter λ to model the arrival times of the tasks since it is a natural way to model a sequence of events "randomly spaced in time". We consider up to 500 tasks for our evaluation and λ ranging from 0.5 to 500. We use randomly generated deadlines of the tasks between (d_l,d_u) minutes and different values of d_l and d_u, ranging from 10 to 1500. The workloads for the tasks are randomly generated within a range of (p, max(w)) cycles, where the lower limit is p=10 cycles. We use a range of upper limits for max(w), from 1500 to 4500. As described in Section III-D, the value of q and d_l determine the number of entries in the lookup table. Accordingly, we construct the lookup table for the resources.

B. Cost Analysis

We observe the costs returned by the schemes. The exhaustive search considers all possible allocations of resources, and returns the lowest cost. However, since exhaustive search takes several hours, we compare the cost returned by exhaustive search for only up to ten tasks. The greedy and overlap schemes perform within 200% of the results from exhaustive search for upto ten tasks. For a greater number of tasks, we only compare the greedy and the overlap schemes

Figure 2 (a) shows the cost for the greedy and overlap schemes, for different number of tasks ranging from 100 to 500, with $\lambda = 1.5$, $d_l = 10$, and $d_u = 1500$. The values in the figure are averaged over 100 trials. The figure shows that the overlap scheme performs consistently better than the greedy scheme. For 500 tasks, the overlap scheme returns an average cost of 546, 22% reduction when compared to the greedy value of 698. This is because the overlap scheme considers sharing VMs temporally for overlapping tasks.

C. Varying Poisson parameter λ

The value of λ determines the distribution of the tasks. A lower value results in a more concentrated distribution, with more tasks with a common arrival time. Figure 2 (b) shows the histogram of the arrival times of a small set of tasks from our evaluation, for different values of λ. It is observed that the set of tasks with lower values of λ have more overlap in their arrival times. This overlap in the arrival times corresponds to an overlap between the durations of the tasks. Thus lower values of λ correspond to greater overlap between the tasks.

We vary λ from 0.5 to 500 and observe the cost returned by the greedy and overlap schemes in Figure 2 (c). The number of tasks is 100. The cost returned by the greedy scheme does not vary significantly with different λs because the greedy scheme does not consider the tasks simultaneously. For the overlap scheme, the cost obtained increases for higher values of λ. For example, for $\lambda = 0.5$, the cost returned by the overlap scheme is 27.2% less than the greedy scheme; for $\lambda = 500$, the improvement is reduced to 7.2%. This is because as λ increases, there is less overlap between the tasks, and hence considering the tasks simultaneously has less benefit. An extreme case would be a scenario when there is no overlap between the tasks. In this case, the greedy scheme would return the optimal solution.

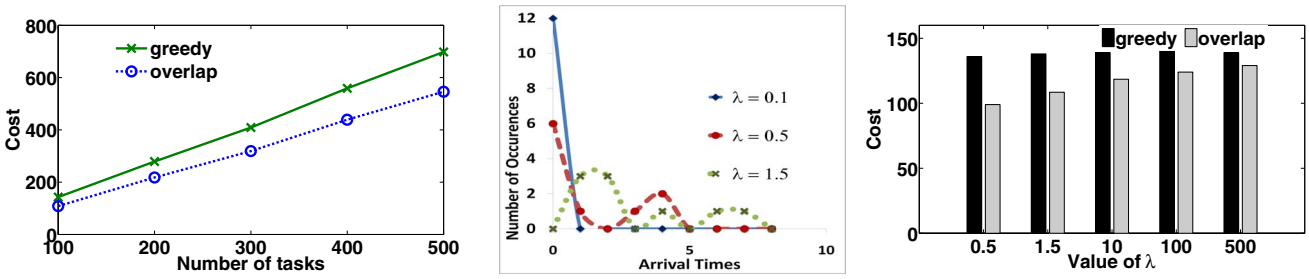
D. Varying workload

Increasing the workloads of the tasks results in a proportionate increase in the cost function. We vary the limits of the random generator (p, q) and observe the variation in the costs. Figure 3(a) shows the cost with $p = 10$, for different values of q. Figure 3(b) shows the cost with $q = 4500$ and different values of p. It is observed that the cost for both schemes increases with the workload.

E. Complexity analysis

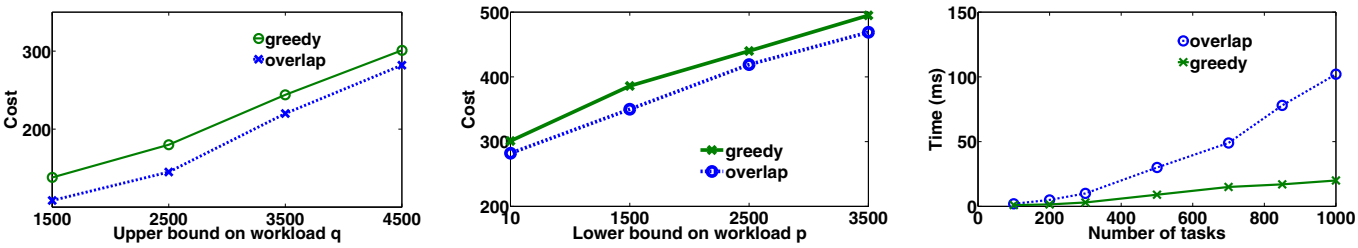
Exhaustive search has exponential complexity and is not a scalable solution. The greedy scheme has the following steps (1) Construct the lookup table. The number of entries is $|\max(\frac{w_i}{d_i - a_i})|$ and each entry is linear in the number of different types of VMs. Thus the complexity of this step is given by $O(x \times |\max(\frac{w_i}{d_i - a_i})|)$. (2) Visit each task once and search the entries of the lookup table to find the allocation. Since the table is sorted, searching can be done in $O(|\max(\frac{w_i}{d_i - a_i})|)$ time. Since there are N tasks, this results in a complexity of $O(N \times |\max(\frac{w_i}{d_i - a_i})|)$. The overall complexity is the sum of the above steps (1) and (2). If $N >> x$ and $|\max(\frac{w_i}{d_i - a_i})|$ is small, the complexity of the greedy solution is linear in the number of tasks, i.e., $O(N)$.

For the overlap scheme, we have the same steps (1) and (2). In addition, for each task, we consider all the tasks with deadlines greater than the current task and examine the sharing of VMs between tasks. This results in each task allocation having a complexity of O($N - 1$); since there are N such task allocations, the overall complexity is given by $O(N \times (N - 1)) = O(N^2)$. Figure 3(c) shows the time taken by the greedy and overlap schemes for different numbers of tasks. We do not show the exhaustive search in the figure because it is exponential in the number of tasks, number of resources, and types of resources and takes several hours for a set of 20 tasks on a 3 GHz system with 4 GB RAM. We fit the data points in Figure 3(c), and we verify that the greedy and



(a) (b) (c)

Fig. 2. (a) Costs obtained by the greedy and overlap schemes, for different numbers of tasks (lower cost is better). The value of λ is 1.5. (b) Histogram of arrival times of the tasks for different values of λ. The points in the histogram are interpolated to show the distribution of the arrival times. (c) Costs obtained by the greedy scheme and the overlap scheme for different values of λ for 100 tasks. As the value of λ increases, the amount of overlap decreases and thus the the cost obtained by the overlap scheme increases.



(a) (b) (c)

Fig. 3. (a) the cost of the greedy and overlap schemes with lower bound $p = 10$, for different values of upper bound q. (b) the cost of the greedy and overlap schemes with upper bound $q = 4500$ and different values of lower bound p. (c) performance of the greedy and overlap schemes for different numbers of tasks.

overlap schemes have linear and quadratic time complexities respectively.

V. Conclusion

We analyze the problem of allocating resources for real-time tasks such that the cost is minimized and all the deadlines are met. We formulate the problem as a constrained optimization problem. As an optimal solution has exponential complexity, we propose an EDF-greedy scheme and a scheme consider temporal overlapping to allocate resources efficiently. Our future work includes extending the analysis by considering tardiness of tasks, relaxed or soft real-time constraints, and considering bulk discount pricing.

References

[1] T. A. AlEnawy et al. Energy-Aware Task Allocation for Rate Monotonic Scheduling. In *IEEE Real Time and Embedded Technology and Applications Symposium*, pages 213–223, March 2005.

[2] U. Ali et al. Video based Parallel Face recognition using Gabor filter on homogeneous distributed systems. In *IEEE International Conference on Engineering of Intelligent Systems*, pages 1–5, 2006.

[3] F. Aymerich et al. A real time financial system based on grid and cloud computing. In *ACM symposium on Applied Computing*, pages 1219–1220, 2009.

[4] J. Cong et al. Energy Efficient Multiprocessor Task Scheduling under Input-dependent Variation. In *Design, Automation and Test in Europe*, pages 411–416, 2009.

[5] R. Datta et al. Image Retrieval: Ideas, Influences, and Trends of the New Age. *ACM Computing Surveys*, 40:1–60, April 2008.

[6] E. Deelman et al. The cost of doing science on the cloud: the montage example. In *ACM/IEEE Conference on Supercomputing*, pages 1–12, 2008.

[7] K. Funaoka et al. Energy-Efficient Optimal Real-Time Scheduling on Multiprocessors. In *IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, pages 23–30, 2008.

[8] S. Funk et al. Energy Minimization Techniques for Real-Time Scheduling on Multiprocessor Platforms. In *Technical Report 01-30*, Computer Science Department, University of North Carolina-Chapel Hill, 2001.

[9] M. Goraczko et al. Energy-optimal software partitioning in heterogeneous multiprocessor embedded systems. In *Design Automation Conference*, pages 191–196, 2008.

[10] R. Huang et al. Automatic resource specification generation for resource selection. In *ACM/IEEE Conference on Supercomputing*, pages 1–11, 2007.

[11] O. Kao et al. Scheduling aspects for image retrieval in cluster-based image databases. In *IEEE/ACM Symposium on Cluster Computing and Grid*, pages 329–336, 2001.

[12] K. H. Kim et al. Power-aware provisioning of cloud resources for real-time services. In *International Workshop on Middleware for Grids, Clouds and e-Science*, pages 1–6, 2009.

[13] W. Lee. Energy-Saving DVFS Scheduling of Multiple Periodic Real-Time Tasks on Multi-core Processors. In *IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications*, pages 216–223, 2009.

[14] S. Liu et al. On-Line Scheduling of Real-Time Services for Cloud Computing. In *World Congress on Services*, pages 459–464, 2010.

[15] C. Nastar et al. Real-time face recognition using feature combination. In *IEEE International Conference on Automatic Face and Gesture Recognition*, pages 312–317, 1998.

[16] T. N'takpé et al. A comparison of scheduling approaches for mixed-parallel applications on heterogeneous platforms. In *International Symposium on Parallel and Distributed Computing*, page 35, 2007.

[17] K. Pua et al. Real time repeated video sequence identification. *Computer Vision and Image Understanding*, 93(3):310–327, 2004.

[18] F. Suter. Scheduling Delta Critical Tasks in mixed-parallel applications on a national grid. In *IEEE/ACM International Conference on Grid Computing*, pages 2–9, 2007.

[19] W. Tsai et al. Real-Time Service-Oriented Cloud Computing. In *World Congress on Services*, pages 473–478, 2010.

[20] P. Viola et al. Robust real-time object detection. *International Journal of Computer Vision*, 57(2):137–154, 2002.

[21] G. Wei et al. A game-theoretic method of fair resource allocation for cloud computing services. *The Journal of Supercomputing*, pages 1–18, 2009.

[22] C. Xian et al. Energy-aware scheduling for real-time multiprocessor systems with uncertain task execution time. In *Design Automation Conference*, page 669, 2007.

[23] G. Zeng et al. Practical Energy-Aware Scheduling for Real-Time Multiprocessor Systems. In *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 383–392, 2009.