Computing
https://doi.org/10.1007/s00607-020-00838-1

REGULAR PAPER

Constraint aware profit maximization scheduling of tasks in heterogeneous datacenters

Chinmaya Kumar Swain[1] · Bhawana Gupta[1] · Aryabartta Sahu[1]

Received: 16 March 2020 / Accepted: 5 August 2020

Abstract
The extensive use of cloud services in different domains triggers the efficient use of cloud resources to achieve maximum profit. The heterogeneous nature of data centers and the heterogeneous resource requirement of user applications create a scope of improvement in task scheduling. The resource requirements in terms of task constraints must be fulfilled for the tasks to be admitted to the system. Once a task admitted to the system, it may violate service level agreement and incurs penalty due to the disproportionate resource allocation at run time. The latency-sensitive and short-lived workloads need effective scheduling to gain more profit. In this work, we propose Heuristic of Ordering and Mapping for Constraint Aware Profit Maximization (HOM-CAPM) problem for efficient scheduling of tasks with constraints and deadlines to gain maximum profit. The HOM-CAPM approach considers estimation of task execution time in a heterogeneous environment, efficient task ordering, and profit-based task allocation to maximize the overall profit of the cloud system. To gain maximum profit the proposed heuristic considers two cases, (a) not allowing the tasks for execution if it expected to miss its deadline and (b) allowing the task which earns substantial profit even though it is expected to miss its deadline. The results of the extensive simulation using Google trace data as input show that our proposed HOM-CAPM approach generates more profit than other state-of-the-art approaches.

Keywords Datacenter · Cloud computing · Scheduling · Constraint · Profit

Mathematics Subject Classification 68T20 · 68W40 · 68Q15 · 97K50

✉ Chinmaya Kumar Swain
chinmayaswain@iitg.ac.in

Bhawana Gupta
bhawana.gupta@intel.com

Aryabartta Sahu
asahu@iitg.ac.in

[1] Department of Computer Science and Engineering, Indian Institute of Technology Guwahati, Guwahati, India

1 Introduction

Over the years, cloud computing gains popularity due to its unique properties like flexibility, elasticity, availability of unlimited computational resources, and pay-as-you-use pricing model [1,2]. This induces many clients to transfer their business to the cloud. The key feature of cloud computing is pay-as-you-use, which means the users need to pay for the resources they consume for the entire time of usage [3,4].

According to the present practices of different cloud service providers (Amazon EC2, Microsoft Azure, etc.), the customers' demand for CPU, memory, etc., are provided to the customer in terms of virtual machines (VMs) using virtualization technology. In the virtualization method, flexibility is higher as any requirement of VM in terms of OS, kernel, library, and architecture, etc can be provided on top of the host machines. But the overhead associated with virtualization is around 15–40% [5], which decreases the profit of the cloud service provider. To get rid of the huge overhead, the container-based operating-system-level virtualization with less overhead (i.e., $\leq 5\%$) is being popularly used [6,7]. In container-based virtualization, VM of the same OS, kernel, and architecture can run on top of host machines. So we need to segregate the user requested VMs into categories based on the available categories of host machines and map the requested VMs on corresponding categories of host machines.

Data centers get more diverse in terms of hardware and software because old systems get replaced by new ones with different specifications [8]. This introduces heterogeneity in a data center, which is the backbone of the cloud system. Heterogeneity ignorance leads to massive inefficiencies as applications are sensitive to hardware architectures [9]. As the jobs submitted to the cloud system are from diverse sources, the resource requirements for each of the tasks may vary significantly, which complicates the process of efficient resource management for the service provider. The jobs/tasks may also have different preferences of machines, where it may need specific configurations or specialized accelerators (GPUs, FPGAs, etc.) or a machine with a particular kernel version. Constraints restrict a task for its execution on a specific set of machines. Apart from the constraints, applications need the assurance of meeting quality of service (QoS) and service level agreement (SLA). The preferences and constraints associated with the tasks further complicate the scheduling decisions [10].

Due to the hourly billing scheme, the user needs to pay for the resources by the hour even if he does not utilize the allocated resources in the whole billing cycle [12]. This leads to an increase in the cost of the user to a certain extent. To minimize the resource wastage and the charged cost to users, short-term (on-demand) renting scheme for the user and long-term (reserved) renting scheme for the service provider is evolved [13]. This kind of billing scheme requires three tiers architecture of the cloud system. The three tiers' architecture of the cloud system consists of infrastructure providers, service providers, and users (Fig. 1). The user submits its job to a service provider and pays for it based on the amount of computational resources used for the entire time of usage. The service provider (broker/ middleman) rents the resources on a reserved basis from the infrastructure provider and provides the service to the users. Based on the user requirement the service provider configures the containers dynamically so as to minimize the resource wastage [6,7]. This technique is more preferred due to

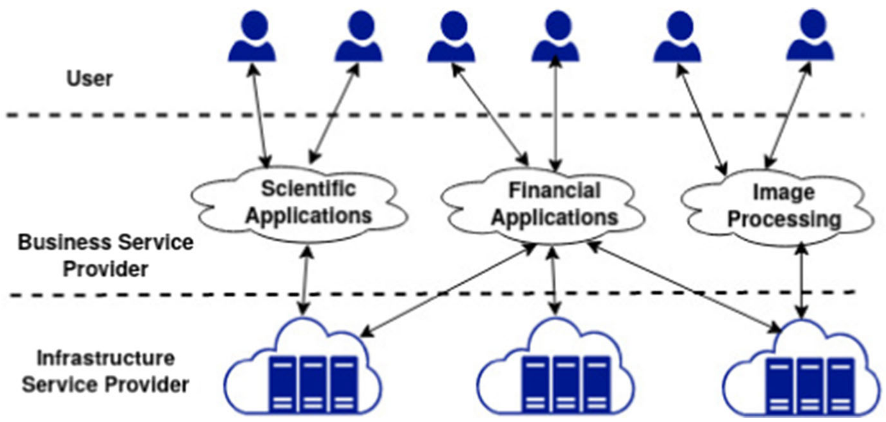


Fig. 1 Cloud architecture [11]

the large gap between the amount of resource request, and the amount of resource used (based on the study done by Reiss et al. [8] on Google cluster trace [14]), which leads to huge resource wastage. The infrastructure provider maintains the hardware and software facilities (data centers) for renting purposes.

To make a profit with sustainable growth, the service provider needs to be competitive among peer service providers. Also, the processing of applications within specific deadlines has become more important due to the introduction of QoS and time-dependent pricing model [15,16]. Due to the task constraints, the delay associated with the scheduling of tasks increases by a factor of 2 to 6, which induces deadline miss of the tasks [17]. Deadline miss of the task increases the penalty, and hence reduce the profit of the service provider.

In this work, our objective is to maximize the profit considering task placement constraint and completion of the task execution before its deadline. Based on the application characteristics the service provider creates a virtual data center by renting the heterogeneous physical resources from the infrastructure service provider. In this case, we are not using processor virtualization due to huge overhead even though we do not have to handle the resource constraints as requested by the users. However, we use OS-level virtualization, where we need to handle the resource constraint to gain maximum profit. In this work, we deploy applications with the least infrastructure using the concept of containerization, where computing resources are split up dynamically and shared using the operating system level virtualization. So, the physical machines are segregated based on hard constraints by exploiting the knowledge of application characteristics. The scheduler equipped with such information is capable of making a better decision for current and upcoming user requests. An efficient approach is being proposed, which can understand application requirements, machine characteristics, and uses this information to maximize the overall profit of the system.

Our contributions in this work are:

– We develop Heuristic of Ordering and Mapping for Constraint Aware Profit Maximization (HOM-CAPM) scheduling approach for the allocation of tasks associated

with constraints and deadlines to heterogeneous machines, where we segregate the user tasks and host machines based on their characteristics.
- We survey different pricing models, penalties and used proper pricing and penalty model for the user tasks where every task associated with their deadlines.
- The proposed approach maximizes the profit using a three-level profit maximization scheme, (a) profit-based task ordering for task admission, (b) profit-based task allocation of the admitted task, and (c) admission of the tasks which are expected to miss their deadlines but still generate extra profit.
- We then analyze the possibility of admission of the tasks which are expected to miss their deadlines to increase the profit with manageable QoS degradation.
- We validate our work through simulation and report a comparative study with other related approaches. For the simulation, we use the real-world Google cluster traces.
- Further, we report a comparative study of simulated annealing approach with our proposed heuristic for the CAPM problem.

The paper is organized as follows. Section 2 presents a summary of the related work. Section 3 describes the machine environment, task environment, revenue model, cost model, penalty model, and the overall problem statement. Section 4 presents our considered system architecture. Section 5 presents the solution approach. Experimental setup and result analysis are reported in Sect. 6. Finally we concludes the paper and present the future research directions in Sect. 7.

2 Related work

The presence of a number of private and public cloud providers with their distinct features (VM types, pricing scheme, etc.) makes it challenging for the user to choose the right service. The broker (service provider) mechanism is used to assist the user in a better way by transforming the heterogeneous cloud system into a commodity-like service [18]. Most of the datacenters (infrastructure service provider) which are the backbone of cloud system are intrinsically heterogeneous in terms of CPU (i.e., instruction set architecture (ISA), clock frequency), memory (i.e., capacity, bandwidth), network (i.e., technology, bandwidth) and storage (i.e., redundancy, technology, capacity) configurations [19]. Cloud service providers allow the tasks to request for heterogeneous resources using task constraints. Many researchers have studied the task constraints with their impacts on resource management in datacenters [17]. Thus any scheduling approach which is aware of different constraints can benefit substantially for improving the system performance.

Sharma et al. [17] integrated machine characteristics and task placement constraints into a performance benchmark of Google compute clusters to evaluate the impact of changes in machine configuration and application demands. They found that incorporating placement constraints showed a reduction of 13% in average task execution delays and 5% improvement in machine resource utilization. Thinakaran et al. [19] proposed a constraint-aware hybrid scheduler, which addressed the problems of constraint consciousness and decreasing queuing delay in heterogeneous datacenters. Delimitrou et al. [9] presented a scheduler named Paragon which uses a collaborative filtering technique to handle both heterogeneity and interference for datacenters.

In real-time systems, the primary concern for the task is to meet its deadlines (which is a form of QoS). Different approaches are being proposed to provide better QoS in the virtualized cloud environment where different applications share common resources [20–22]. As the cloud system is used as utility service, profit maximization along with QoS and SLA is a key driving feature of any cloud system. Profit of the cloud service provider depends on many factors (price, system configuration, market demand, customer satisfaction, and so on), among which task urgency is one of them. Service providers wish to gain higher profit without compromising the QoS [11]. Mei et al. [11] designed the double renting scheme to increase the profit of the cloud service provider with guaranteed quality of service. Lee et al. [23] and Cao et al. [16] used a static pricing model, where the price of the service is fixed, and it is known to the customer in advance. However, in a dynamic pricing model, the system delays the pricing decision until the customer demand is revealed [24].

A similar kind of assumption is being made in our proposed approach for the calculation of the penalty. Li et al. [25] proposed the non-preemptive and preemptive profit and penalty aware (PP-aware) scheduling algorithms for maximizing the system's total accrued profit. The well-known cloud schedulers like Mesos [26] and Omega [27] follows the hierarchical design and considers only task placement constraints. These schedulers do not consider hard and soft constraints for the task placement in heterogeneous data centers. The contemporary container-based schedulers like Mesosphere [28] and Kubernetes [7] neither consider task constraint nor does dynamic task ordering for optimizing the profit. Belal Ali Al-Maytami et al. [29] proposed a scheduling algorithm employing Principle Components Analysis (PCA) for the improvement of make span and reduction of Expected Time to Compute (ETC) matrix of the tasks represented in DAG. Yawen Wang et al. [30] proposed a task-VM mapping algorithm based on dynamic Heterogeneous Earliest Finish Time (HEFT) which is to accelerate defense strategy switching and improve workflow efficiency. The work carried out in [31] focuses on a requirement model for the run time execution and control of an intention-oriented Cloud-Based Application. The research work Cloud-SEnergy [32] was proposed to optimize the energy consumption without adversely affecting performance. This uses a bin-packing technique to generate the most efficient service composition plans to mitigate the ecological impact.

All the above discussed well-known schedulers do not consider both constraint and deadline of the task. In our case, we consider real-time scheduling of tasks, considering the constraint and deadline of the task. We compare our approach with the PP-NP [25] and GUS [33] because these two models are used to maximize the profit considering the deadline of the task.

The scheduling challenges in terms of task constraints, deadline constraints, and profit maximization have been extensively studied individually. But scheduling approach considering all the three aspects are not being considered. The proposed approach very much addresses the problem of deadline miss minimization and profit maximization with constraint awareness in a heterogeneous cloud system.

3 Problem formulation

3.1 Machine environment

We consider the cloud system that contains m number of heterogeneous physical machines (PMs) i.e. $M = \{M_1, M_2, M_3, \ldots, M_m\}$. Each PM, M_j is characterized by $\langle arch_j,\ plat_j,\ kern_j,\ core_j,\ mem_j,\ nbw_j,\ clk_j,\ dbw_j \rangle$. Here the terms, $arch_j$, $plat_j$, $kern_j$, $core_j$, mem_j, nbw_j, clk_j and dbw_j represents architecture (ISA may be X86, X86_64, ARM, or Power PC etc.), platform (OS may be MS Windows, Linux, etc), kernel (Kernel version of OS), number of cores, memory, network bandwidth, clock speed and disk bandwidth respectively available with machine M_j.

3.2 Task environment

Given with a set of n independent and non splittable tasks with deadline $T = \{T_1, T_2, T_3, \ldots, T_n\}$, where each task T_i is characterized by $\langle a_i,\ e_i,\ d_i, arch_i,\ plat_i,\ kern_i,\ core_i,\ mem_i,\ nbw_i,\ clk_i,\ dbw_i \rangle$. Here a_i, e_i and d_i represents arrival time, execution time and deadline of task T_i respectively. Other terms $arch_i$, $plat_i$, $kern_i$, $core_i$, mem_i, nbw_i, clk_i and dbw_i represents architecture, platform, kernel, number of cores, amount of memory, network bandwidth, clock speed and disk bandwidth required for the task T_i.

The constraints associated with each task T_i can be broadly categorized as a hard constraint and soft constraint [19]. Hard constraints are the strict requirements of a task without which the task cannot run, on the other hand, soft constraints can be relaxed or negotiated by task performance in terms of extended execution time (or extended finish time). In this work, we consider architecture, platform, kernel, and the number of cores are the hard constraints. Similarly, memory, network bandwidth, clock speed, and disk bandwidth are considered as soft constraints. Each of the soft resource constraints of type (k) is provided with a degradation factor of α_k. It is verified experimentally that inappropriate soft resource allocation degrades overall application performance significantly [34]. However, just as a thumb rule, at least 70% of all the individual soft resource requirements must be made available to a task so that the task can execute without much performance degradation. The net execution time e_i^{net} of a task T_i for the case, where allocation is made without meeting the soft constraint requirement can be written as:

$$e_i^{net} = e_i \times \left(1 + \sum_{k=1}^{p} \alpha_k \times \frac{R_i^k - A_i^k}{R_i^k} . (R_i^k > A_i^k)\right) \tag{1}$$

where, R_i^k is the required amount of resource, and A_i^k is the allocated amount of resource, of type k for the task T_i and p is the number of soft resource constraints. The term $R_i^k > A_i^k$ get evaluates to 1 if the condition is true else evaluates to 0. The units of A_i^k varies and depends on the type of resource. These are (a) core in numbers, (b) memory in GB, (c) network bandwidth in KB/s, (d) clock speed in MHz, and (e) disk bandwidth in MB/s. If all the soft constraints are satisfied for a task then $e_i^{net} = e_i$.

The execution time estimate, which is represented by Eq. 1 is taken from [35] and the similar kind of assumptions are also discussed in the paper [36]. In the paper [36], the estimation of the execution time computed based on the resource usage of various computing resources. They designed the platform-independent model which takes the sum of the product terms. However, the final form of the estimation of execution time linearly depends on the resource usage of various applications. The evaluation of the proposed model in [36], done by taking many real-life workloads like production web server, Fibonacci number generator, standalone and distributed variants of weather-research-forecasting applications, etc.. Even though experimentation with other classes of applications would be necessary before a generic conclusion, but the Eq. 1 can be a reasonable estimate for arbitrary applications. Also, many researchers proposed different linear models to predict the slow-down of applications due to disproportionate allocation of resources at run time [9,37–39]. Eq. 1 dynamically estimates the net execution time based on the availability of resources with a machine where the task gets allocated.

3.3 Revenue model

The revenue model adopted here considers the user has to pay according to the amount of resources (such as CPU, memory, etc.) requested by him. The similar kind of assumptions is considered in [13], where the tasks are allocated to containers that are dynamically created through OS-level virtualization. Here we consider the service provider to configure the containers as per the resource request of the user.

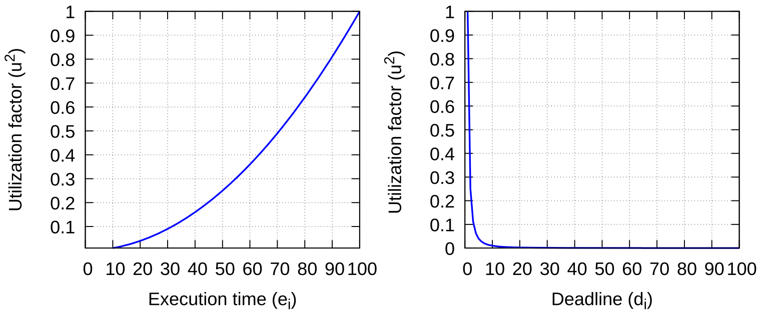
Here we consider the fine-grained billing time unit (BTU) for the collection of revenue from the customers. The revenue collected from the customer depends on the amount of resources it requests and duration of use, i.e., the time to execute the task T_i. The revenue ($revenue_i$) collected for the execution of the task T_i can be defined as;

$$revenue_i = e_i \times \left(\sum_{k=1}^{q} R_i^k \times C_k^c \right) \times (1 + u_i^2) \tag{2}$$

where R_i^k is the requested amount of physical resources of k^{th} type, by the task T_i, C_k^c is per unit cost per unit time charged for the k^{th} resource to the user and q is the number of chargeable resources. All the resources in the cloud system are not chargeable resources. The term u_i can be defined as follows:

$$u_i = \frac{e_i}{d_i - a_i} \tag{3}$$

where a_i, e_i, and d_i are the arrival time, execution time and deadline of the task T_i respectively. The utilization factor u_i^2, which controls the cost, based on latency sensitivity of task (T_i). More urgency of a task (whose $d_i - a_i$ is smaller) implies



(a) u_i^2 verses e_i, where d_i and a_i are constants $(a_i = 0, d_i = 100)$.

(b) u_i^2 verses d_i, where e_i and a_i are constants $(a_i = 0, e_i = 1)$.

Fig. 2 Utilization factor (u^2) vs execution time and deadline

higher revenue and the profit. Figure 2 represents the utilization factor which controls the cost model of the system. As shown in Fig. 2a, utilization factor increases due to an increase in execution time of a task and that contributes more to the revenue.

Figure 2b represents the utilization factor vs deadline keeping execution time and arrival time constant. The value of u_i can be from 0 (when $d_i = \infty$) to 1 (when $e_i = d_i - a_i$), so the revenue from a task can be up to two times, if it is a tight deadline task as compared to completely relax task where $d_i = \infty$.

3.4 Cost model

The actual cost spent by the service provider can be defined as:

$$cost_i = e_i^{net} \times \left(\sum_{k=1}^{q} A_i^k \times C_k^s \right) \tag{4}$$

where C_k^s is the actual cost spent per unit physical resources per unit time of the allocated k^{th} resource, A_i^k is the quantity of allocated k^{th} type physical resource to task T_i. The units of A_i^k varies and depends on the type of resource. These are (a) core in numbers, (b) memory in GB, (c) network bandwidth in KB/s, (d) clock speed in MHz, and (e) disk bandwidth in MB/s. There are two cost factors that affect the profit for the service provider. One of the cost is the charged cost (C_k^c), which is charged to the user by the service provider and the other one is the actual cost (C_k^s) spent by the service provider by paying the rent for the physical resources taken from the infrastructure service provider. The service provider creates a virtual datacenter by renting the reserved VMs and provide the services to the user on-demand basis. The typical value of C_k^c is 10–20% more than C_k^s.

Table 1 Penalty model of international cloud service providers [40]

Cloud provider	Calculation method	Service credit	Penalty cap
Amazon EC2	Ratio of Total Charge	$< 99.95\% - 10\%$ $< 99\% - 30\%$	N/A
IBM Softlayer	Ratio of downtime	Each 30 min downtime, 5% of the fees	N/A
Windows Azure	Ratio of Total Charge	$< 99.95\% - 10\%$ $< 99\% - 25\%$	N/A
VPS.net	Ratio of downtime	$10 \times downtime$	100%
Google GCE	Ratio of total charge	$< 99.95\% - 10\%$ $< 99\% - 25\%$ $< 95\% - 50\%$	N/A
Rackspace	Ratio of downtime	Each 30 min downtime, 5% of the fees	100%
GoGrid	Ratio of downtime	$100 \times downtime$	100%

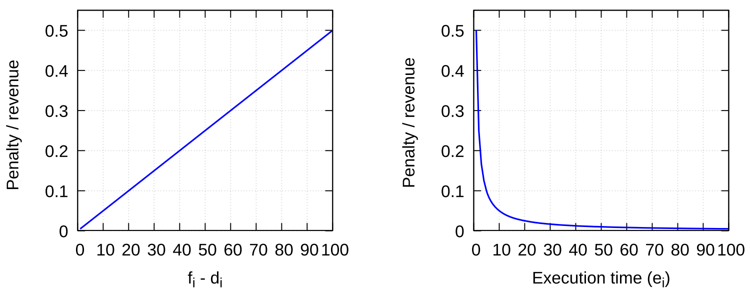
3.5 Penalty model

Suppose a task T_i misses the deadline, the service provider pays back the penalty to compensate for the SLA violation. Most popularly used penalty calculation methods by international cloud providers are listed in Table 1. Mostly three methods are used to compute the penalty based on different violation levels (i.e., downtime or unavailability rate). Those three methods are (a) a specific ratio of downtime, usually greater than 1 (b) fixed value at different violation levels and (c) a certain percentage of total charge paid to consumers based on different violation levels [40,46,47]. Penalty cap refers to be the maximum value of the penalty that the service provider payback for the violation of SLA. In our model, we consider the SLA violation as the deadline miss case.

Here, we have considered the penalty model which is very much similar to the penalty models of real cloud service providers and can be defined as:

$$penalty_i = \begin{cases} \beta . \left(\frac{f_i - d_i}{e_i} \right) .revenue_i & \text{if } f_i > d_i \\ 0 & \text{if } f_i \leq d_i \end{cases} \tag{5}$$

where β is a positive constant which controls the penalty cap and e_i, f_i, and d_i are the execution time, finish time, and deadline of the task T_i respectively. The penalty to be paid (as defined in Eq. 5) if the task finishes its execution after its predefined deadline, otherwise no penalty will be paid for the task. The delay-sensitive tasks need to be finished before its deadline. If those tasks are delayed then the service provider needs to pay the penalty which is proportional to the difference between finish time and deadline. Figure 3 graphically illustrates the penalty with varied $(f_i - d_i)$ keeping e_i constant (Fig. 3a) and varied e_i keeping $(f_i - d_i)$ constant (Fig. 3b). As shown in Fig. 3a, the penalty linearly increases as the difference between finish time and deadline $(f_i - d_i)$ increases, keeping execution time (e_i) constant. So the cloud service provider will try to allocate the tasks in such a way that, the difference of finish time



(a) With constant e_i and $\beta = 0.5$ ($e_i = 100$).

(b) With constant $f_i - d_i$ and $\beta = 0.5$ ($f_i - d_i = 1$).

Fig. 3 $Penalty/revenue$ vs $f_i - d_i$ and e_i

and deadline must be minimum which leads to a minimum penalty. Figure 3b shows that the penalty decreases with the increase in execution time keeping $f_i - d_i$ to be constant. In this model, shorter tasks incur more penalty percentage at near-miss as compared to the longer tasks. The penalty cap value (β) of the cloud service provider is constant for all tasks. Mostly used notations used in this paper are represented in Table 2.

3.6 Problem statement

In this work, we want to schedule a set of n tasks with the deadline (as specified in Sect. 3.2) on m heterogeneous physical machines (as specified in Sect. 3.1), so that the overall profit is maximized. We call this problem to be **C**onstraint **A**ware **P**rofit **M**aximization (CAPM) problem. The CAPM problem can be defined as follows:

$$Maximize \sum_{i}^{\#\text{admitted tasks}} profit_i \tag{6}$$

where $profit_i$ is the profit of executing a task T_i and can be written as:

$$profit_i = revenue_i - cost_i - penalty_i \tag{7}$$

and the value of revenue, cost and penalty are calculated by Eqs. 2, 4 and 5 respectively.
subject to

$$arch_i = arch_j; plat_i = plat_j; kern_i = kern_j \tag{8}$$

$$core_i \geq core_j \tag{9}$$

Eq. 8 and Eq. 9 represents the hard constraint to be satisfied for the i^{th} task to run on j^{th} machine, whereas soft constraints can be relaxed with performance trade-off. This may be achieved possibly by executing the more urgent tasks before their deadlines and decreasing the penalty.

Table 2 Notations used in this work

Notation	Definitions
$arch_j$	Architecture of the host machine M_j
$plat_j$	Platform of the host machine M_j
$kern_j$	Kernel (Kernel version of the OS)
$core_j$	Number of cores of the host machine M_j
mem_j	Memory capacity of the host machine M_j
nbw_j	Maximum network badnwidth of the host machine M_j
clk_j	Clock speed of the host machine M_j
dbw_j	Disk bandwidth of the host machine M_j
a_i	Arrival time of the task T_i
e_i	Execution time of the task T_i
d_i	Deadline of the task T_i
$arch_i$	Architecture requirement of the task T_i
$plat_i$	Platform requirement of the task T_i
$kern_i$	Kernel (Kernel version of the OS) requirement of the task T_i
$core_i$	Number of cores requirement of the task T_i
mem_i	Memory requirement of the task T_i
nbw_i	Network badnwidth requirement of the task T_i
clk_i	Clock speed requirement of the task T_i
dbw_i	Disk bandwidth requirement of the task T_i
e_i^{net}	Net execution time of the task T_i
α_k	Degradation factor of soft resource constraint of type k
R_i^k	Required amount of resources of type k for task T_i
A_i^k	Allocated amount of resources of type k for task T_i
u_i	Utilization factor of task T_i
$revenue_i$	Revenue collected for execution of the task T_i
C_k^c	Cost per unit charged for the resource type k
q	Number of chargeable resources
$cost_i$	Actual cost
C_k^s	Actual per unit cost spent for the resource type k
f_i	Finish time of the task T_i
β	Factor which controls the penalty cap
$penalty_i$	Penalty of the task T_i
$profit_i$	Profit for executing the task T_i

4 System architecture

Here we consider the cloud system, where the service provider provides the services to the user through a virtual data center. The virtual data center contains the physical resources rented by the service provider from the infrastructure service provider. The service provider accepts the batches of tasks with deadlines. A batch of tasks

Table 3 Machine attributes

Abbreviation	Description	Domain	Constraint type
Arch	Architecture	X86/ARM	Hard
Plat	Platform family	Linux/Windows	Hard
Kern	Kernel version	2 Linux/Windows	Hard
Core	Number of cores	4/8/16/32/64	Hard
Mem	Memory available	2 to 512 MB	Soft
Nbw	Network bandwidth	upto 1 MB/s	Soft
Clk	CPU clock speed	300 to 3500 MHz	Soft
Dbw	Disk bandwidth	Upto 50 MB/s	Soft

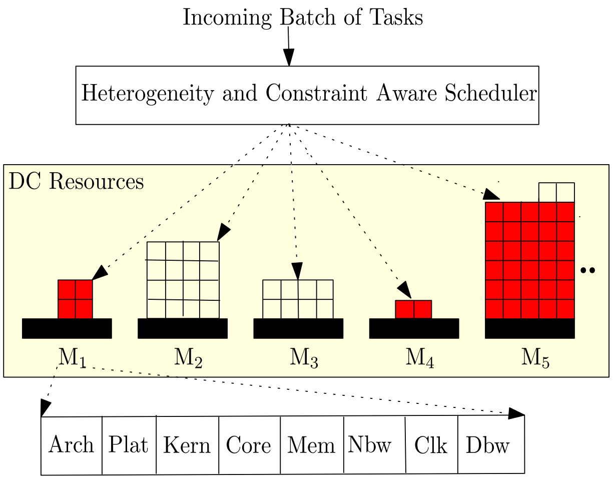
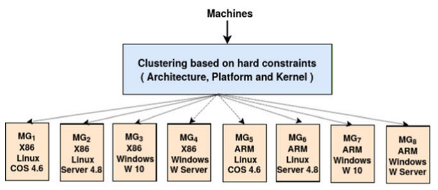


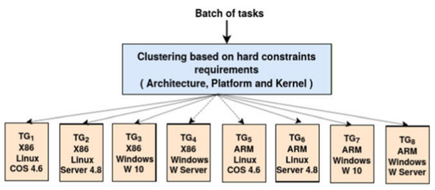
Fig. 4 System architecture

have the same arrival time, and for simplicity, we assume $a_i = 0$ for a batch of tasks. All the submitted tasks are associated with one or more constraints of any category (hard or soft constraint). For example, the semantics of individual constraints and their domain values of the Google cluster trace [14] are reported in Table 3. Some constraints are hard constraints ($arch$, $plat$, $kern$, and $core$), and some are soft constraints (mem, nbw, clk, and dbw). The assumed virtual datacenter contains sufficiently large number of machines, $M = \{M_1, M_2, M_3, \ldots, M_m\}$. These machines are heterogeneous, and they are mainly characterized by their architecture, platform, kernel version, number of cores, amount of memory, network bandwidth, clock speed, and disk bandwidth. The heterogeneity and constraint aware scheduler accepts a batch of tasks and efficiently schedule those tasks to machines available in the data center to maximize the profit.

The example datacenter in Fig. 4 has 5 machines: M_1, M_2, M_3, M_4 and M_5. The number of cores available at machines M_1, M_2, M_3, M_4 and M_5 are 4, 16, 8, 2 and



(a) Machine Clustering.



(b) Task Clustering.

Fig. 5 Machine and task grouping

32 respectively. Machines M_2 and M_3 are idle while some of the cores at M_1, M_4 and M_5 are executing some tasks.

5 Solution for CAPM

Here we propose a task ordering and mapping based heuristic approach to solve the CAPM problem. The proposed approach is called as Heuristic of Ordering and Mapping for CAPM problem (HOM-CAPM). As each task needs to be satisfied with their required hard constraints, so in our proposed approach, we group the set of machines based on hard constraints (architecture, kernel, and platform) except the number of cores. Similarly, tasks are grouped into the same number of groups as the machines, based on their requirements. As there is multiple types of resources available with each machine, it is profitable to allocate more tasks to one physical machine, where the task satisfies the constraints.

In our approaches, we segregate the set of machines based on hard constraints. As we are considering two types of architectures (X86 and ARM), two platform types (Linux and Windows) and each platform have two different types of kernels, there are 8 different options or group of hard constraints. So we make the machine groups (MGs) as, MG_1, MG_2, …, and MG_8 which is shown in Fig. 5a. Similarly, tasks are grouped based on their hard constraint requirements and formed the task groups (TGs) as, TG_1, TG_2, …, and TG_8 (Fig. 5b). All the tasks of a group need to be executed on top of the corresponding machine group. For example, all tasks of TG_1 need to be executed on MG_1 satisfying the hard constraints (except the number of cores). The grouping of machines and tasks are required because constraint induces a delay in scheduling decisions [8]. So if the tasks and machines are grouped appropriately based on constraints, then the scheduling delay can be minimized. So the Algorithm 1 has an important role in grouping the tasks and machines based on the constraints.

So, here our objective is to map all the tasks of the task group (TG) to machine group (MG) such that overall profit will be maximized. The number of nodes deployed for each MG is proportional to the number of tasks in each TG based on the history of previous batches. For example, if the number of tasks in TG_1 is more as compared to the other TGs, then the number of nodes in MG_1 will be more as compared to other MGs. Here we are not considering to minimize the number of nodes for a given cluster to meet deadlines of submitted tasks. But we try to execute the maximum number of

Algorithm 1: Constraint Aware Grouping and Scheduling

```
Input: Batch of Tasks, Machines
1: Group the machines based on hard constraints $MG_1, \dots, MG_l$
2: Group the tasks based on hard constraints $TG_1, \dots, TG_l$
3: For each task group $TG_i$ in $TG_1, TG_2, \dots, TG_l$
4:     Schedule tasks of $TG_i$ to $MG_i$ using Algorithm 2 or Algorithm 3
```

tasks before their deadlines keeping the constant number of deployed nodes to each cluster for a batch of tasks.

For the tasks of a TG to be scheduled on specific machines of MG, the problem can be simplified to a set of tasks need to execute on machines with their corresponding specifications. The simpler version of this problem, scheduling of n tasks $T_i(e_i, d_i)$ on m processors with deadline i.e. the $min\big(\sum w_i.U_i\big)$ (minimization of weighted deadline miss of a task set on multiprocessor) problem is NP-hard [41]. Therefore, in this work, we use heuristic approaches for solving this problem. Here we consider two cases, (a) profit maximization without deadline miss, i.e. we would not allow a task to be admitted and scheduled if we expect that the task may miss its deadline when admitted to the system, and (b) allow a task to be admitted, if it gives substantial profit even though it misses the deadline.

The overall approach of HOM-CAPM is given in the pseudocode of Algorithm 1. As we know constraints induces a delay in scheduling decisions [8], so the tasks and machines are grouped appropriately based on constraints to minimize the delay. In this approach, we group the tasks based on hard constraints, group the machines based on hard constraints except for the number of cores, and schedule tasks of task group TG_i on the machines of MG_i to maximize the profit of each group. Then the subsequent steps are to order the tasks (as defined in 5.1) and map the tasks (as defined in 5.2) to maximize the overall profit.

5.1 Ordering of tasks

The order in which tasks are scheduled affects the finishing time of tasks. If more number of high profitable tasks are finished before their deadlines, the profit incurred by a service provider will be more. Here to examine the effect of different ordering strategies for profit maximization, we explore some of the ordering strategies. Those ordering criteria are defined as follows:

- **Expected profit (EP):** The tasks having higher expected profit are assumed to be more profitable. Here, we arrange the tasks based on decreasing order of their expected profit (Eq. 7 without considering penalty).
- **Revenue (RV):** The tasks having higher expected revenue or charged cost (Eq. 2) are likely to give more profit if they are given priority while scheduling. In this case, we arrange the tasks based on the decreasing order of their expected revenue.
- **Revenue per unit execution time (RPE):** Tasks are arranged in the decreasing order of their revenue per unit of their execution times.

Algorithm 2: HOM-CAPM without allowing to be missed task

```
Input: $TG_i$ of a batch of Tasks, corresponding Machine Group $MG_i$ and Task ordering (one
       ordering from Sect. 5.1)
1  Order the tasks based on predefined ordering;
2  while $TaskGroup$ is not empty do
3      while T is not empty do
4          Select a task from T, say $T_i$;
5          Find subset $M_a$ of currently available machines in $MG$ which satisfies core constraint of $T_i$;
6          Select candidate machines $M_c$, where $T_i$ meet it's deadline;
7          if $M_c = \Phi$ then
8              Reject the task;
9          else
10             Rank the machines using Eq. 13, for all $M_j \in M_c$;
11             Select machine $M_j$ which gives highest profit;
```

- **Resource usage cost (RUC):** Here, we arrange the tasks based on decreasing order of their resource usage costs as defined in Eq. 10.

$$cost_i^u = e_i \times \left(\sum_{k=1}^{q} R_i^k \times C_k^c \right) \tag{10}$$

  where C_k^c is the charged cost for the per unit physical resources per unit time allocated k^{th} resource, R_i^k is the quantity of requested k^{th} type physical resource to task T_i.
- **Cost incurred by cloud provider (IC):** The tasks which cost more (Eq. 10) may not result in higher profit. So, in this ordering tasks expecting more costs are given less preference (reverse case of RUC).
- **Slack time (ST):** Slack time ($d_i - e_i$) is the time by which a task can be delayed so that it will finish on or before the deadline. Tasks having less slack time should be scheduled first to avoid penalties. So, here tasks are scheduled in the increasing order of their slack times.
- **Earliest due date (EDD):** In these criteria, tasks are ordered in the increasing sequence of their deadlines.
- **Shortest job first (SJF):** In this case, tasks are arranged in the increasing order of their specified execution times.

5.2 Task mapping without allowing deadline miss

After ordering the tasks based on the best ordering criteria, we need to map the tasks to machines dynamically looking at different constraints which will generate more profit. Algorithms 2 and 3 (as defined in Sect. 5.3) present the pseudo-code of those approaches. Algorithm 2 presents the pseudo-code of the approach where the system does not allow a task to execute (or do not admit the task) if the task is expected to miss its deadline. Most of the datacenters have heterogeneous physical machines, and every physical machine has some configuration including a number of processors, memory size, network bandwidth, clock speed, etc. The processors of the considered heterogeneous physical machine have similar compute power and assumed to depend on the frequency of operation. Here, we assume that each task needs to use only one PM for its execution. As task requests of a number of cores are allocated in a single physical machine, so we neglect the factor of data communication between processors. This approach selects tasks of the input task group TG_i in the given input ordering and tries to find suitable machines for each task one by one. Tasks are checked for their allocation to machines as per their constraint and deadline requirements. If no such machine available where the task meets its deadline, then the task gets rejected. Based on the soft constraint requirements and the traditional ranking criteria, the model rank the machines where the task T_i can achieve maximum compactness using Eq. 11.

$$rank(T_i, M_j) = \sum_{k=1}^{p} f(R_i^k) \tag{11}$$

where $rank(T_i, M_j)$ represents the ranking of the task T_i when allocated to the machine M_j with soft constraint requirements R_i^k for $k = 1$ to p. Lower the $rank(T_i, M_j)$ value, more preferred is the machine M_j for the task T_i. The $f(R_i^k)$ can be defined as:

$$f(R_i^k) = \begin{cases} 1 & \text{if } R_i^k \leq A_i^k \\ \dfrac{A_i^k}{R_i^k} & \text{if } R_i^k > A_i^k \end{cases} \tag{12}$$

where R_i^k and A_i^k are the required soft resource constraint and available soft resource constraint for the task T_i respectively. Here the compactness ranking assumes that the profit will be higher if we execute the task with less amount of resources than the required amount.

However, this will not guarantee the maximum profit, because if we allocate less amount of resources to a task, it's execution time increases and that increases the cost. Now from the set of candidate machines, the algorithm selects a machine where the task can gain maximum expected profit and allocate the task to that machine. The maximum expected profit (MEP) is calculated using Eq. 13 for task T_i.

$$MEP(T_i) = max\{g(T_i, M_j)\}, \forall M_j \in M_c \tag{13}$$

where M_c represents the set of candidate machines where the task T_i can be scheduled and $g(T_i, M_j)$ can be defined as:

$$g(T_i, M_j) = revenue_i - cost_i, \text{if } T_i \text{ is allocated to } M_j \tag{14}$$

where the $revenue_i$ and $cost_i$ are calculated using Eqs. 2 and 4 respectively. As we are not allowing the task to miss its deadline, so no penalty will be paid for that task.

Algorithm 3: HOM-CAPM with allowing to be missed task

```
Input: $TG_i$ of a batch of Tasks, corresponding Machine Group $MG_i$ and Task ordering (one
       ordering from Sect. 5.1)
1  Order the tasks based on predefined ordering;
2  while $TaskGroup$ is not empty do
3      while $T$ is not empty do
4          Select a task from T, say $T_i$;
5          Find subset $M_a$ of currently available machines in $MG$ which satisfies core constraint of $T_i$;
6          Select candidate machines $M_c$, where the task can meet it's deadline;
7          if $M_c = \Phi$ then
8              Select a machine $M_j \in M_a$, where profit is maximized using Eq. 15 and
               $profit_i \geq \rho.cost_i$, if no such machine found then reject the task;
9          else
10             Rank the machines using Eq. 13, for all $M_j \in M_c$;
11             Select machine $M_j$ which gives highest profit;
```

5.3 Task mapping with allowing deadline miss

In this case, we allow the tasks to be admitted to the system if the tasks can contribute to the overall profit even if they miss their deadlines (Ref. line 7, 8 of Algorithm 3). This may lead to an increase in profit for the cloud service provider. The task may incur some penalty due to run time allocation and for that we need to allocate the task to machine which gives maximum profit with penalty (MPP).

$$MPP(T_i) = max\{g^{'}(T_i, M_j)\}, \forall M_j \in M_c \tag{15}$$

where M_c represents the set of candidate machines where the task T_i can be scheduled and $g^{'}(T_i, M_j)$ can be defined as:

$$g^{'}(T_i, M_j) = revenue_i - cost_i - penalty_i, \text{ if } T_i \text{ is allocated to } M_j$$

where the $revenue_i$, $cost_i$ and $penalty_i$ are calculated using Eqs. 2, 4 and 5 respectively. In Algorithm 3, we present the pseudo-code of the given approach. The algorithm considers the MGs and TGs with predefined task ordering in each TG (using Algorithm 1) and schedules the TGs to MGs to maximize the overall profit of the system. In this case, we allow the task to execute on the system even if it expected to miss its deadline and contributes the substantial profit to the system. It checks for the profit to be greater than a particular threshold level (ρ), then only that task will be scheduled. This still results in more earning in terms of profit in spite of paying penalties for some tasks. The ρ value is set in such a way that, at least $revenue_i$ collected for the task T_i is $(\rho \times 100)\%$ more than $cost_i + penalty_i$. For example, if $\rho = 0.05$ then at least $revenue$ collected for the task must be 5% more than the $cost + penalty$ for that task.

Time Complexity: The constraint aware grouping and scheduling algorithm (Algorithm 1) either uses the Algorithms 2 or 3 to schedule the batch of tasks to machines, where profit will be maximized. The complexity of Algorithms 2 and 3 can be formulated as O($nlogn + nm$), where m represents the number of active machines and n

represents the number of tasks, and $n \gg m$. The complexity of the computations, for finding rank and expected profit used in these algorithms take constant time as they are independent of n and m. As the system accepts a large number of tasks, so the time complexity of the proposed algorithms is O($nlogn$).

5.4 Simulated annealing for CAPM (SA-CAPM)

We explore the popular meta-heuristic approach called simulated annealing to solve the CAPM problem for further analysis. The SA is considered as the most popular single solution based optimization algorithm [42,43]. As we discussed in Algorithm 1, the set of machines and a set of tasks are grouped and each task group TG_i will be scheduled to the set of machines in MG_i. For each task group and its corresponding machine group are the input to the SA algorithm and that generates the profit for that task group. The sum of the profits of all groups generates the overall profit of the system. The step-by-step procedure of SA is given in Algorithm 4. The SA algorithm begins with an initial random solution of S and generates its neighborhood solution $S^{'}$. The next step of SA is to compute the fitness value of S and $S^{'}$ and update the solution. If the $f(S) \leq f(S^{'})$ then set $S = S^{'}$ otherwise accept the solution $S^{'}$ with the probability P. The probability P value is defined by $P = e^{-\Delta E/kt}$, where $\Delta E = f(S) - f(S^{'})$, k the Boltzmann constant and t the value of temperature. If $P > random()$, then $S = S^{'}$, otherwise, the value of S will not change. The next step is to update the temperature value t using $t = \beta \times t$, where $\beta \in [0, 1]$ represents a random value.

Algorithm 4: Simulated Annealing for CAPM (SA-CAPM)

```
Result: Best solution ($S_b$) with maximum profit
1  Generate the initial solution $S$ and compute its fitness value $f(S)$;
2  Initialize the value of the temperature $t_0$ and total number of iterations $i_{max}$ ;
3  Set the best solution $S_b = S$ and $f(S_b) = f(S)$ ;
4  Set $i = 1$ and $t = t_0$ ;
5  while $i < i_{max}$ do
6      Find the neighborhood solution $S^{'}$ from $S$ ;
7      Compute the fitness value of $S^{'}$ as $f(S^{'})$ ;
8      if $f(S^{'}) > f(S)$ then
9          $S = S^{'}$;
10     else
11         Compute the $\Delta E = f(S) - f(S^{'})$ ;
12         if $P \leq random()$ then
13             $S = S^{'}$;
14         Update the value of the temperature $t$ using $t = \beta \times t$ ;
15         if $f(S) > f(S_b)$ then
16             $S_b = S$ ;
17             Set i = i + 1 ;
```

Table 4 Parameters for simulation studies

Parameter	Values
Architecture	X86 or ARM
Platform	Linux or Windows
Kernels	Windows 10, Windows Server, Linux centOS 4.6, Ubuntu 4.8
CPU	1, 2, 4, 8 or 16
RAM	2 to 512 GB
Network bandwidth	1 to 10000 KB/s
Clock speed	100 to 3500 MHz
Disk bandwidth	1 to 50 MB/s

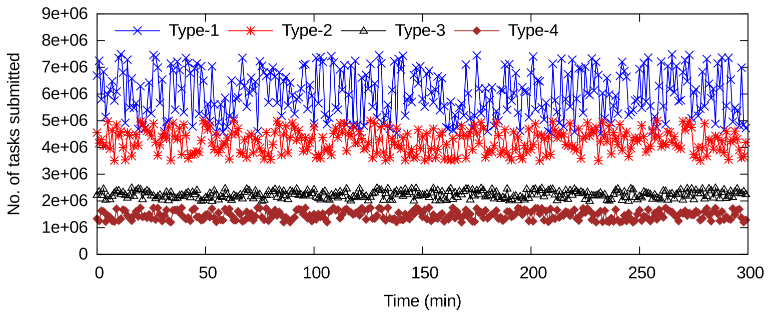


Fig. 6 Arrival of different types of tasks in Google traces

6 Experimental results

6.1 Simulation setup

We use the simulation platform similar to Eagle [44] and it provides many scheduling modules. The detail explanation of parameter setting for our simulation as follows:

6.1.1 Machine parameters

In our simulation, we consider all the PMs are heterogeneous. The values of different machine parameters are listed in Table 4.

6.1.2 Task parameters

In this work, we had used real-world public traces of Google to evaluate the performance of HOM-CAPM with other state-of-the-art approaches. The data set in Fig. 6 shows the arrival pattern of different types of tasks in the Google production cluster for 300 minutes in May 2011 [14]. The trace contains four types of tasks, and for our performance evaluation, we had used tasks of type 1. The simulation was conducted

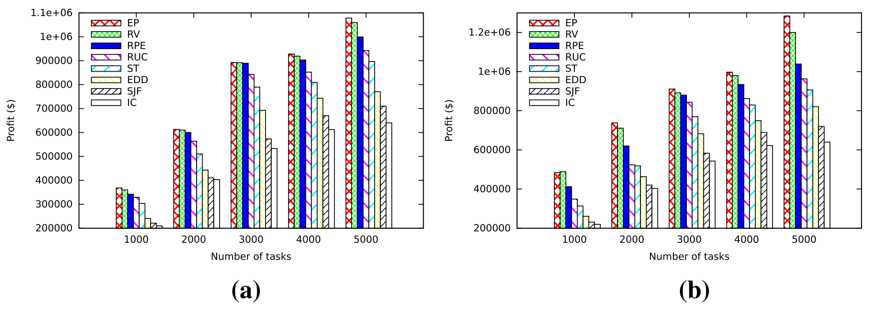


Fig. 7 Effect of task ordering on the profit. **a** HOM-CAPM without allowing to be missed tasks. **b** HOM-CAPM with allowing some of the to be missed tasks

varying the task count from 1000 to 10000. Each set of tasks are divided into batches based on the time interval (t'). Here, we had taken the time interval of $t' = 1$ minute, whereas we can also take other values. We had collected all the information from the Google cluster data except the deadline of the tasks. The deadline of the tasks are set as $d_i = a_i + e_i + random(baseTime, \mu \times baseTime)$, where $\mu = 5$ and $baseTime = 100$ only for simplification required in order to obtain numerical results, and the model can support any value for this parameter. The degradation factor (α_i) for each soft constraint is different but for the simplification of the computation, we had taken the value to be the same for all the soft constraints i.e. 0.1. For controlling the penalty, we set the value of β to be 0.5 (i.e., the penalty should not exceed 50% of the total cost). The charged cost and actual cost spent set as the values \$1.0 and \$0.75 per unit time respectively, and this value is being taken to get the numerical results, whereas it can support any value for these parameters.

6.2 Performance of different task ordering

The overall profit of the system greatly influenced by the ordering of the task submission to the system. There are many ordering techniques are available for task allocation in cloud system. We have explored many task ordering techniques which believe to produce similar profit for our CAPM problem. However we experimentally finetune the ordering which produces maximum profit. As the tasks are considered batch-wise, we had ordered the tasks within a batch based on our predefined ordering and calculated the profit gained by the system, which is shown in Fig. 7. Figure 7a reports the profit gained by the system when the deadline miss of task is not allowed and Fig. 7b reports the result for the deadline miss case. In both cases, the expected profit (EP) and revenue (RV) based orderings perform comparably. But expected profit (EP) based ordering performing best in most of the cases. Apart from EP and RV ordering, the other orderings with decreasing order of profit are revenue per unit execution time (RPE), resource usage cost (RUC), slack time (ST), earliest due date (EDD), shortest job first (SJF) and incurred cost (IC). The IC ordering is performing worst. In subsequent experiments, we had taken the best ordering (i.e., EP) for reporting our results.

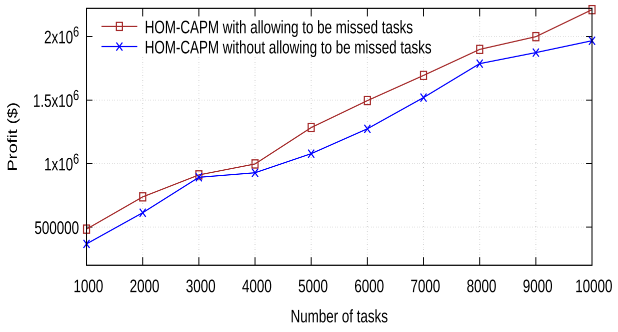


Fig. 8 Profit considering HOM-CAPM approaches with or without allowing to be missed tasks (with EP ordering)

Table 5 Profit in ($) of different scheduling approaches

No. of tasks	HOM-CAPM with missed tasks	HOM-CAPM without missed tasks	PP-NP	GUS	EDF
1000	368,074	484,090	336,011	311,073	309,076
2000	613,021	738,056	563,064	552,087	481,087
3000	892,092	911,107	694,041	667,073	532,080
4000	928,103	997,271	815,106	793,108	611,050
5000	107,873,0	1,083,870	924,137	821,162	682,211
6000	127,401,6	1,302,052	1,028,411	942,815	796,362
7000	152,010,5	1,623,069	119,227,4	1,092,321	876,123
8000	1,787,048	181,209,3	136,792,1	120,001,3	993,629
9000	187,402,4	1,945,109	1,586,006	1,496,321	1,023,761
10000	1,967,101	2,112,187	1,834,200	1,689,027	1,329,011

6.3 Performance of HOM-CAPM with or without allowing to be missed tasks

To investigate the benefits of the admission control mechanism, we ran our simulation with different loads (i.e., a varying number of tasks). We compare two cases of HOM-CAPM approach, wherein one case we will not allow the tasks to miss their deadline and in other cases we allow the task to miss its deadline if it contributes substantial profit, i.e., we allow the tasks to miss the deadline for the case where $profit_i \geq \rho.cost_i$. But for the case $profit_i < \rho.cost_i$, the task will get rejected as that will incurs a huge loss. We ran our simulation 10 times and took the average for consideration. As shown in Fig. 8, as the load goes on the increase, the profit difference between the two cases is significant, whereas, for low load case, that is not so significant. The X-axis of Fig. 8 represents the number of tasks, and Y-axis reports the profit of the system. As the number of tasks increases, the profit of the system increases. The profit gained for the case of CAPM with allowing to be missed tasks varies 3–7% as compared to the case where the system does not allow the tasks to miss their deadlines.

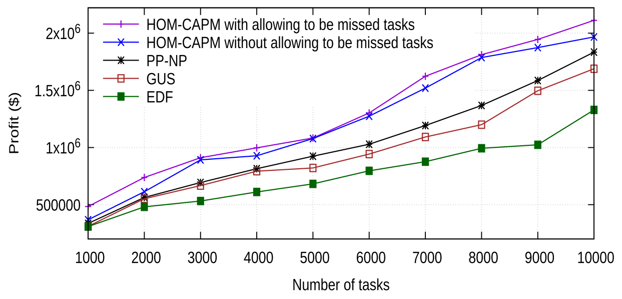


Fig. 9 Performance of different scheduling approaches

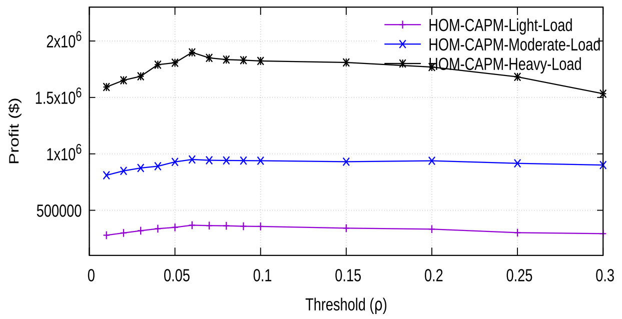


Fig. 10 Impact of ρ on system performance with different loads

6.4 Performance of different scheduling approaches

This section reports the performance of our proposed approaches (HOM-CAPM with or without allowing to be missed tasks) against state-of-the-art UA-aware (GUS) [33], Non-preemptive profit penalty aware (PP-NP) [25] and EDF scheduling approaches. From utility accrual-aware scheduling, we choose the GUS algorithm defined in [33], for which the tasks with the largest potential utility density [45] is scheduled first. Without loss of generality, $(d_i - a_i)/e_i$ value is used to order the set of tasks for the GUS approach. Non-preemptive PP-Aware Scheduling as defined in [25] orders the tasks based on its risk factor. Here for simplicity we had taken the $riskfactor_i = loss_i/gain_i$, where $loss_i = cost_i + penalty_i$ and $gain_i = revenue_i$ of the task T_i. If the value of the $riskfactor_i > 1$, then that task will be dropped otherwise it will be scheduled.

The approaches EDF, GUS, and PP-NP, do not consider the soft constraints while allocating tasks to machines, whereas HOM-CAPM considers the constraints for task allocation. In this experiment, we had taken tasks randomly from Google traces (1000 to 10000 tasks). For each case, we had reported the average results as shown in Fig. 9 and also in Table 5. From Fig. 9, it is being observed that when a number of tasks are less, all the approaches are performing well as compared to the case of more number of tasks admitted to the system.

The proposed approach HOM-CAPM (with or without allowing to be missed tasks) outperforms all other approaches (EDF, GUS, and PP-NP) because it considers the

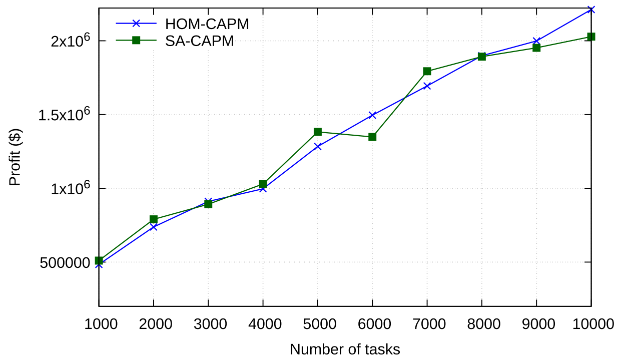


Fig. 11 Performance of HOM-CAPM against SA-CAPM

constraints associated with the task while scheduling and allocates the tasks to appropriate machines such that profit will be maximized. In HOM-CAPM with deadline miss approach, we incorporate threshold (ρ) value to control the profit, which is not there with other approaches, except PP-NP. PP-NP and GUS approaches are performing comparably with 2–5% of the difference in performance.

6.5 Impact of threshold (ρ)

We further extend our study to know the potential impact of the threshold (ρ) value to control the profit in HOM-CAPM by allowing to be missed tasks. For this study we fix the threshold (ρ) value to be 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.15, 0.2, 0.25 and 0.3, and based on that we allow the tasks to miss their deadline if it can add profit of at least ρ% of cost for the task. We conduct three sets of experiments, first with a light load case (number of tasks is 1000), second with moderate load case (number of tasks is 5000), and third with heavy load case (number of tasks is 10000). The results are reported in Fig. 10 varying the threshold ρ value from 0.01 to 0.3.

As shown in Fig. 10, when the system with lower ρ value gains less profit and continue to increase till the ρ value is 0.07. After that ρ value, the profit goes on decreasing. A similar pattern is observed for all types of situations (i.e., lightly loaded, moderately loaded and heavily loaded). The decrease in the profit is sharper for the case when the system is heavily loaded. The variation of profit is more in case of the heavily loaded case due to the more variation of tasks. The ρ value controls the profit level nicely based on the requirement of the system. Although our result reported in Fig. 10, gives a general guideline to choose ρ value judiciously to maximize the profit, but still, the optimal system performance needs a complex understanding of all the unmeasured factors which directly or indirectly defines the system, specifically the QoS.

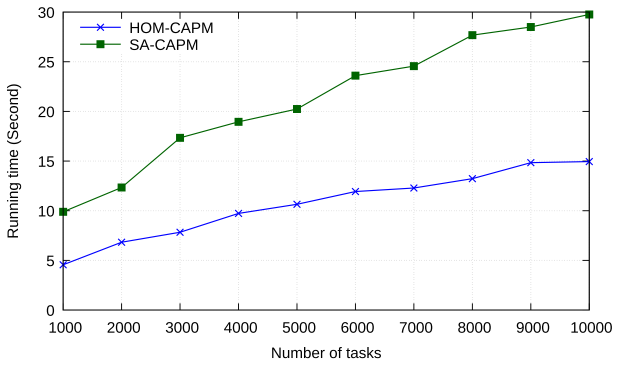


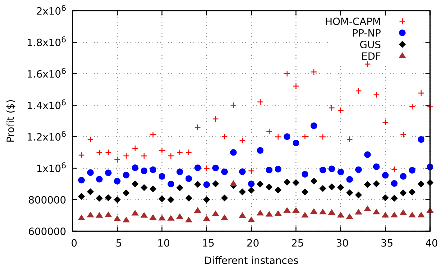
Fig. 12 Performance of HOM-CAPM against SA-CAPM in terms of running time

6.6 Performance of SA-CAPM and HOM-CAPM

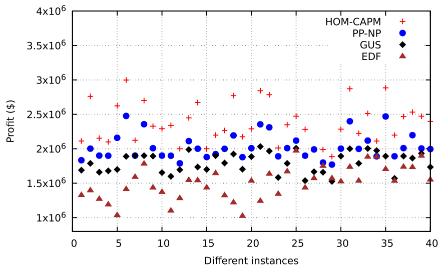
To evaluate our HOM-CAPM approach with SA based approach (SA-CAPM), we had taken all the task and machine parameters as discussed in Sects. 6.1.2 and 6.1.1. Apart from that, the extra parameters required for SA are $i_{max} = 100$, $t = 1.0$, and $\beta = 0.8$. Figure 11 illustrates the profit gained by HOM-CAPM and SA-CAPM as the number of tasks increases. The profit of the SA-CAPM algorithm is larger than HOM-CAPM in most of the cases. However, the running time of SA-CAPM is quite larger (all most double the time) than the HOM-CAPM approach as shown in Fig. 12. The slow convergence of SA-CAPM in most of the cases still produces a better result than HOM-CAPM in terms of profit.

6.7 Validity of the simulation result

To address this concern we had conducted the simulation for 40 different instances of the Google traces. We had considered two cases, in one case we consider 5000 tasks and other cases 10000 tasks. The average profit of HOM-CAPM, PP-NP, GUS, and EDF for 5000 tasks is 1250886, 9977725, 858294, and 706931 respectively. Similarly, for 10000 tasks, the average profit of CAPM, PP-NP, GUS, and EDF are 2373159, 2030895, 1799743, and 1511717 respectively. Fig. 13 reports the performance of different approaches at each instance. The X-axis of Fig. 13 represent different instances of Google traces, from instance 1 to instance 40, and Y-axis represents the profit. The significance threshold was set at 0.05 (5%). It is being observed from the repetitive simulation result reported in Fig. 13, that the proposed HOM-CAPM approach performs better than other approaches. The performance difference is statistically significant ($p = 0.023$). There seem to be many validity threats to the simulation study. The choice of data is limited (only from Google traces), which can further be extended with other real-world traces. This would give more detailed statistical analysis as well as allowing broader coverage in the field of study.



(a) 5000 tasks.



(b) 10000 tasks.

Fig. 13 Performance of different instances of Google trace

7 Conclusion and future work

In this paper, we proposed a heuristic of ordering and mapping for CAPM, for scheduling the latency-sensitive tasks with constraints to gain maximum profit. The proposed approach schedules the tasks to machines in a heterogeneous cloud system to finish most of the tasks before their deadline and maximize the profit for the cloud service provider. The HOM-CAPM approach is evaluated based on different task ordering (EP, RV, RPE, RUC, IC, ST, EDD, and SJF) criteria and expected profit (EP) ordering gives a better result than other ordering criteria. The simulation results with Google cluster data demonstrate that the proposed approach can greatly increase the profit as compared to other approaches like PP-NP, GUS, and EDF. Further, we reported a comparative study of simulated annealing, a meta-heuristic approach with our proposed approach. We found that most of the cases, SA performs better than HOM-CAPM at the expense of long execution time. In the future, we would like to extend our work using different optimization techniques. We would like to explore the gap between our optimal solution and the theoretical optimal solution. Also, we would like to evaluate our proposed model with a real cloud system to explore its efficiency. For further extension, we will consider the constraints apply to the collection of tasks and their impact on task execution time and profit.

References

1. Li K, Mei J, Li K (2018) A fund-constrained investment scheme for profit maximization in cloud computing. IEEE Trans Serv Comput 11(6):893–907
2. Nesmachnow S, Iturriaga S, Dorronsoro B (2015) Efficient heuristics for profit optimization of virtual cloud brokers. IEEE Comput Intell Mag 10(1):33–43
3. Buyya R et al (2009) Cloud computing and emerging it platforms: vision hype and reality for delivering computing as the 5th utility. Future Gener Comput Syst 25(6):599–616
4. Chen J, Wang C, Zhou BB, Sun L, Lee YC, Zomaya AY (2011) Tradeoffs between profit and customer satisfaction for service provisioning in the cloud. In: Proceedings of international symposium on high-performance parallel and distributed computing, pp 229–238
5. García-Valls M, Cucinotta T, Lu C (2014) Challenges in real-time virtualization and predictable cloud computing. J Syst Archit 60(9):726–740
6. Bhimani J et al (2018) Docker container scheduler for I/O intensive applications running on NVMe SSDs. IEEE Trans Multi-Scale Comput Syst 4(3):313–326

7. Bernstein D (2014) Containers and cloud: from LXC to Docker to Kubernetes. IEEE Cloud Comput 1(3):81–84
8. Reiss C, Tumanov A, Ganger GR, Katz RH, Kozuch MA (2012) Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In: ACM SoCC ’12
9. Delimitrou C, Kozyrakis C (2013) Paragon: Qos-aware scheduling for heterogeneous datacenters. In: ASPLOS ’13, pp 77–88
10. Cuomo A, Di Modica G, Distefano S, et al. (2013) An SLA-based broker for cloud infrastructures. J Grid Comput
11. Mei J, Li K, Ouyang A, Li K (2015) A profit maximization scheme with guaranteed quality of service in cloud computing. IEEE Trans Comput 64(11):3064–3078
12. Wang W, Niu D, Li B, Liang B (2013) Dynamic cloud resource reservation via cloud brokerage. In: IEEE international conference on distributed computing systems, pp 400–409
13. Mei J, Li K, Tong Z, Li Q, Li K (2019) Profit maximization for cloud brokers in cloud computing. IEEE Trans Parallel Distrib Syst 30(1):190–203
14. Google Cluster Data. http://code.google.com/p/googleclusterdata/
15. Xu M, Alamro S, Lan T, Subramaniam S (2017) CRED: cloud right-sizing with execution deadlines and data locality. IEEE Trans Parallel Distrib Syst 28(12):3389–3400
16. Cao J, Hwang K, Li K, Zomaya AY (2013) Optimal multiserver configuration for profit maximization in cloud computing. IEEE Trans Parallel Distrib Syst 24(6):1087–1096
17. Sharma B, Chudnovsky V, Hellerstein JL, Rifaat R, Das CR (2011) Modeling and synthesizing task placement constraints in Google compute clusters. In: Proceedings of SoCC
18. Rogers O, Cliff D (2012) A financial brokerage model for cloud computing. J Cloud Comput 1(1):1–12
19. Thinakaran P, Gunasekaran JR, Sharma B, Kandemir MT, Das CR (2017) Phoenix: a constraint-aware scheduler for heterogeneous datacenters. In: IEEE ICDCS, pp 977–987
20. Tao F, LaiLi Y, Xu L, Zhang L (2013) FC-PACO-RM: a parallel method for service composition optimal-selection in cloud manufacturing system. IEEE Trans Ind Inform 9(4):2023–2033
21. Tao F, Cheng Y et al (2014) CCIoT-CMfg: cloud computing and internet of things-based cloud manufacturing service system. IEEE Trans Ind Inform 10(2):1435–1442
22. Zheng X, Martin P, Brohman K, Xu LD (2014) Cloud service negotiation in internet of things environment: a mixed approach. IEEE Trans Ind Inform 10(2):1506–1515
23. Lee YC, Wang C, Zomaya AY, Zhou BB (2012) Profit-driven scheduling for cloud services with data access awareness. J Parallel Distrib Comput 72(4):591–602
24. Cachon GP, Feldman P (2010) Dynamic versus static pricing in the presence of strategic consumers
25. Li S, Ren S, Yu Y, Wang X, Wang L, Quan G (2012) Profit and penalty aware scheduling for real-time online services. IEEE Trans Ind Inform 8(1):78–89
26. Hindman B, Konwinski A, Zaharia M, Ghodsi A, Joseph AD, Katz RH, Shenker S, Stoica I (2011) Mesos: a platform for fine-grained resource sharing in the data center. NSDI 11:22–22
27. Schwarzkopf M, Konwinski A, Abd-El-Malek M, Wilkes J (2013) Omega: flexible, scalable schedulers for large compute clusters. In: Proceedings of ACM European conference on computer systems, pp 351–364
28. Google Mesosphere (2013) https://github.com/mesosphere
29. Al-Maytami BA, Fan P, Hussain A, Baker T, Liatsis P (2019) A task scheduling algorithm with improved makespan based on prediction of tasks computation time algorithm for cloud computing. IEEE Access 7:160916–160926. https://doi.org/10.1109/ACCESS.2019.2948704
30. Wang Y, Guo Y, Guo Z, Baker T, Liu W (2020) CLOSURE: a cloud scientific workflow scheduling algorithm based on attack-defense game model. Future Gener Comput Syst 111:460–474
31. Baker T, Mackay M, Randles M, Taleb-Bendiab A (2013) Intention-oriented programming support for runtime adaptive autonomic cloud-based applications. Comput Electr Eng 39(7):2400–2412
32. Baker T, Aldawsari B, Asim M, Tawfik H, Maamar Z, Buyya R (2018) Cloud-SEnergy: a bin-packing based multi-cloud service broker for energy efficient composition and execution of data-intensive applications. Sustain Comput Inform Syst 19:242–252
33. Li P (2004) Utility accrual real-time scheduling: models and algorithms. Ph.D. dissertation, Virginia Polytechnic Inst. State Univ
34. Wang Q et al (2011) The impact of soft resource allocation on n-tier application scalability,. In: 2011 IEEE international parallel and distributed processing symposium, Anchorage, AK, pp 1034–1045
35. Sadjadi SM et al (2008) A modeling approach for estimating execution time of long-running scientific applications. In: IEEE international symposium on parallel and distributed processing, pp 1–8

36. Shimizu S, Rangaswami R, Duran-Limon HA, Corona-Perez M (2009) Platform-independent modeling and prediction of application resource usage characteristics. J Syst Softw 82(12):2117–2127
37. Delimitrou C, Kozyrakis C (2014) Quasar: resource-efficient and QoS-aware cluster management. In: Proceedings of the 19th international conference on architectural support for programming languages and operating systems (ASPLOS '14)
38. Delimitrou C, Kozyrakis C (2016) HCloud: resource-efficient provisioning in shared cloud systems. In: Proceedings of ASPLOS
39. Swain CK, Sahu A (2018) Interference aware scheduling of real time tasks in cloud environment. In: IEEE 20th international conference on high performance computing and communications; IEEE 16th international conference on smart city; IEEE 4th international conference on data science and systems (HPCC/SmartCity/DSS), Exeter, UK
40. Xiaoyong Y, Hongyan T, Ying L, Tong J, Tiancheng L, Zhonghai W (2015) A competitive penalty model for availability based cloud SLA. In: IEEE 8th international conference on cloud computing, pp 964–970
41. Baev ID, Meleis WM, Eichenberger AE (1999) Algorithms for total weighted completion time scheduling. In: Proceedings of ACM-SIAM symposium on discrete algorithms
42. Bertsimas D, Tsitsiklis J (1993) Simulated annealing. Stat Sci 8(1):10–15
43. Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220(4598):671–680
44. Delgado P, Didona D, Dinu F, Zwaenepoel W (2016) Job-aware scheduling in eagle: divide and stick to your probes. In: Proceedings of ACM symposium on cloud computing
45. Clark RK (1990) Scheduling dependent real-time activities. Ph.D. dissertation, Carnegie Mellon Univ., Pittsburgh, PA
46. Zhang R, Wu K, Li M, Wang J (2016) Online resource scheduling under concave pricing for cloud computing. IEEE Trans Parallel Distrib Syst 27(4):1131–1145
47. Amazon EC2 SLA. https://aws.amazon.com/cn/ec2/sla