An Instance Reservation Framework for Cost Effective Services in Geo-Distributed Data Centers

Kaiyang Liu, *Student Member, IEEE*, Jun Peng, *Member, IEEE*, Boyang Yu, *Student Member, IEEE*, Weirong Liu, *Member, IEEE*, Zhiwu Huang, *Member, IEEE*, and Jianping Pan, *Senior Member, IEEE*

Abstract—Infrastructure-as-a-Service clouds in geo-distributed data centers offer various pricing options, including on-demand and reserved instances, which provide an elastic and cost-effective infrastructure to support High Performance Computing (HPC) applications. In this paper, we propose an instance reservation based cloud service framework, modeling the cost-minimizing reservation decision issue as an NP-hard integer programming problem for distributed data centers. To ease its computation complexity, two algorithms are proposed to minimize the HPC service cost with the worst-case performance guarantees: an offline heuristic-greedy algorithm, and a rolling-horizon based online algorithm when only short-term demand prediction is available. Facing fluctuating demands, instance reservation in a single data center may incur the highly underutilized capacity. To address this issue for further cost reduction, we extend the scheme with a novel cloud broker federation based resource sharing mechanism, reallocating already reserved but unused instances to computation-intensive and short-lived tasks for continuous execution without interruption. Extensive evaluations driven by large-scale trace-based datasets demonstrate that the proposed mechanism can effectively handle large volumes of service requests, saving considerable service costs with higher reservation resource utilization.

Index Terms—Geo-distributed data centers, high performance computing, instance reservation, resource sharing, cost minimization

1 Introduction

Nowadays cloud services have attracted people to migrate ever-increasing resource-intensive High Performance Computing (HPC) applications to geo-distributed data centers for high reliability, scalability and cost saving benefits [1], [2]. As an infrastructure service, Infrastructure-as-a-Service (IaaS) providers provision resources in the form of Virtual Machines (VMs) based on time-varying needs elastically. Multiple VMs are organized as a cloud-based virtual cluster, realizing the parallel services of HPC applications. Due to the high demand on computation resources, HPC with big data processing can incur big cost now. According to IDC predictions, 71 percent of data center hardware expenditure will be from the big data processing, which will hit $203 billion in 2020 [3]. Therefore, it is imperative to provide cost-effective computing services in geo-distributed data centers.

To attract different cloud users, most IaaS cloud service providers offer various types of purchasing options, referred to as the on-demand and reservation instances [4]. On-demand instance offers a flexible resource service based on user demands, while it may incur higher financial cost than reservations due to the relatively higher price. In contrast, reservation allows cloud users to prepay a price to reserve instances for a certain long period (could be weeks, months or years). Generally, reserved instances are often cost-effective for the cloud user. Cloud providers, e.g., Amazon EC2, Windows Azure and Rackspace, charge the reservations with a significant discount to ensure the long-term risk-free income. For example, the reservation can save the cost up to 65 percent when reservations are fully utilized [5]. The cloud providers prefer steadier workloads for easier resource provisioning [6]. In fact, HPC workload shows fluctuating characteristics, and is driven by a lot of short-lived tasks [7]. It is challenging to solve the instance reservation problem facing fluctuating demands.

Thus, utilizing the pricing gap between reserved and on-demand plans, we proposed a cloud instance reservation-based framework to realize the cost-effective HPC in geo-distributed data centers. Starting with the single data center scenario, we build an integer programming optimization framework to determine when and how many instances to reserve in each data center. Unfortunately, such an instance reservation problem is hard to be solved due to its non-convex, nonlinear and NP-hard nature. Therefore, an efficient approximation algorithm is proposed, which incurs less than twice of the optimal cost given user demand information. An

- *K. Liu is with the School of Information Science and Engineering, Central South University, Changsha 410075, China and with the Department of Computer Science, University of Victoria, Victoria, BC V8W 2Y2, Canada. E-mail: liukaiyang@csu.edu.cn.*
- *J. Peng, W. Liu, and Z. Huang are with the School of Information Science and Engineering, Central South University, Changsha 410075, China. E-mail: {pengj, frat, hzw}@csu.edu.cn.*
- *B. Yu and J. Pan are with the Department of Computer Science, University of Victoria, Victoria, BC V8W 2Y2, Canada. E-mail: {boyangyu, pan}@uvic.ca.*

Manuscript received 20 Sept. 2017; revised 7 Feb. 2018; accepted 10 Mar. 2018. Date of publication 22 Mar. 2018; date of current version 7 Apr. 2021. (Corresponding author: Jun Peng.)
Digital Object Identifier no. 10.1109/TSC.2018.2818121

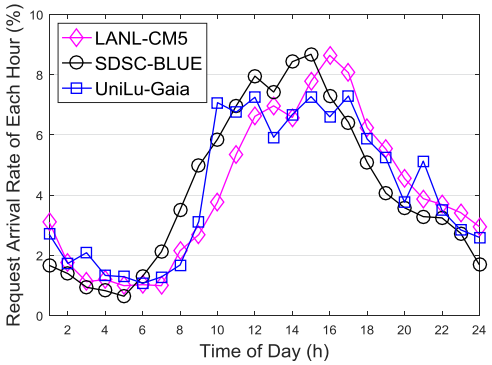


Fig. 1. The normalized request arrival rate of each hour on average during a day based on real-world HPC workload logs in three geo-distributed data centers (shifted for time zone difference), i.e., server UniLu Gaia located in Luxemburg (GMT +1), LANL CM5 located at Los Alamos (GMT −7) and SDSC-BLUE located at San Diego (GMT −8) [8].

effective rolling-horizon based online algorithm is also proposed, which only relies on the short-term demand prediction. Theoretical analysis indicates that the proposed online algorithm is 3-competitive.

As the instance reservation over individual data center may cause highly underutilized capacity, a novel broker federation mechanism is investigated then to realize unused capacity sharing among geo-distributed data centers. Fig. 1 illustrates the normalized request arrival rate per hour during a day on average (in percentage) from three distributed data centers [8]. Due to the time-zone differences, the workloads among geo-distributed data centers may not reach their peaks and valleys at the same time [9]. The data centers authorize the broker federation reallocating already reserved but unused instances to computation-intensive and short-lived tasks for continuous execution without interruption. The proposed task-aware instance sharing algorithm makes the instance reservation service more competitive in cost saving and resource utilization.

Furthermore, large-scale evaluations driven by real-world HPC workloads [8] demonstrate that the proposed instance reservation strategies among federated data centers are highly efficient to handle large volumes of workloads, reducing the total expenses by about 40 percent for cloud users. Besides, the resource utilization can be improved about 10 percent for distributed data centers.

The rest of the paper is organized as follows: Section 2 presents the system model. Section 3 formulates the instance reservation problem in a single data center, and proposes the heuristic-greedy approximation based offline reservation algorithm, along with the rolling-horizon based online reservation strategy. Section 4 investigates the broker federation based real-time resource sharing in distributed data centers. Section 5 presents evaluation results. Section 6 reviews the related work. Section 7 draws the conclusion and lists the future work.

2 System Model

In this section, we introduce the geo-distributed cloud service system with the resource reservation strategy, the cloud pricing scheme and the cloud-based virtual clusters for HPC job execution. The major notations used in this paper are summarized in Table 1.

TABLE 1
Notations

Symbol	Definition
$\mathcal{N}$	The set of geo-distributed data centers
t	Time slot, $t = \{1, \ldots, T\}$
p_i^{r}, p_i^{o}	Prices of reservation and on-demand plan
τ	Valid length of reservations
λ	Length of short-term demand prediction
$\mathcal{H}_i(t)$	Short-term demand prediction, $t = \{1, \ldots, \lambda\}$
$d_i(t)$	Demand estimations at time slot t
$r_i(t)$, $u_i(t)$	Reservation decision and available reservations at time slot t
$e_i(t)$, $o_i(t)$	Idle reserved instances and extra needed instances at time slot t
κ	Submitted HPC tasks, $\kappa \in \mathcal{K}_i$
$\mathcal{F}$	Data center federation
$\theta_{\kappa,j}$	Number of reservations from data center j shared by task κ
φ_i	Price of unit outgoing traffic from data center i
$\mathcal{I}_i(t)$	Divided time slots each with length τ, $t = \{1, \ldots, T - \tau + 1\}$
$\nu_i(t)$	Reservation priority of time slot $\mathcal{I}_i(t)$
C_i	Total cost of users belong to data center i
$C_i^{\mathcal{F}}$	Total cost of users belong to data center i with federation

2.1 Geo-Distributed Cloud Service Systems

As shown in Fig. 2, we consider a geo-distributed cloud service system that consists of a set of data centers $\mathcal{N}$ distributed at different geographical locations (with size $N = |\mathcal{N}|$). Each cloud data center owns a cloud controller module, a demand monitor and a predictor, a cloud broker and several resource-rich servers (e.g., computing servers and storage servers). Computing servers provision cloud services in the form of VMs according to HPC user requests. All data files are stored in the storage servers. Inside a data center, all these components are linked with high-speed switches and LAN networks.

First, the demand monitor aggregates the request information of HPC tasks in previous periods, such as usage statistics of CPU, memory and bandwidth. Using the virtualization technology, the cloud controller schedules the tasks of each user into VM instances for execution. To fulfill the operation of resource-intensive HPC applications, multiple VMs are organized as cloud-based virtual clusters for

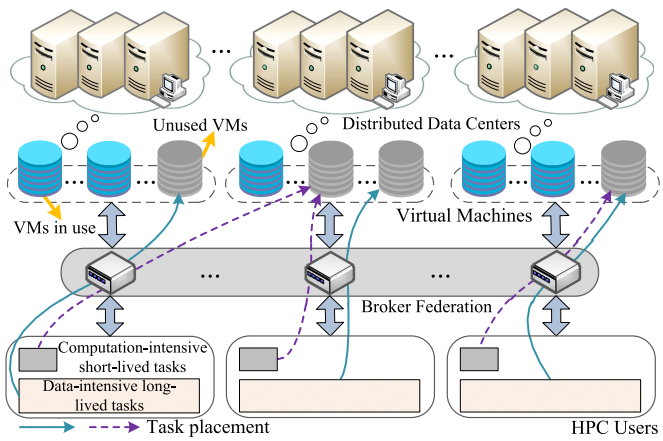
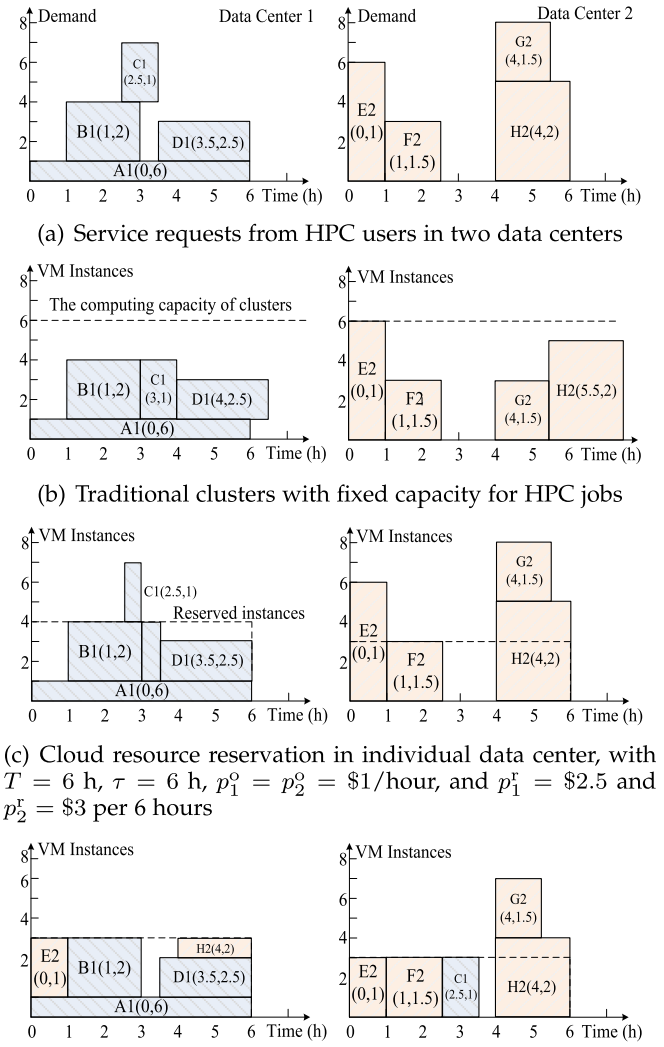


Fig. 2. The geo-distributed cloud computing architecture.

the parallel execution. The system charges service fee in a time-slotted fashion, for $t = 1, \ldots, T$. Then by utilizing the historical request information, the demand predictors make predictions, i.e., the required number of instances in the future periods, for each cloud provider.

To realize instance reservation in each data center, the cloud broker module is adopted. We assume that all data centers and brokers belong to one IaaS cloud provider, which facilitates the resource sharing without generating complex trust issues. The case between different IaaS cloud providers will be considered in our follow-on work. The broker module is deployed upon a virtual infrastructure management architecture (e.g., the OpenNebula manager and its external-resource lease manager Haizea [10]) to manage the life cycles of VMs. Instead of trading directly with data centers, cloud users can purchase instances from the broker. The broker reserves a certain number of instances in advance to accommodate a major part of user requests. To ensure the SLA, on-demand instances may be launched if reservations are not enough to satisfy the burst demand.

The workload analysis indicates that HPC tasks show both computation and data-intensive, long and short-lived characteristics [8]. The broker federation aggregates the request and reservation information of all distributed data centers, allowing data centers to share unused capacities for the bursty requests. If the computation-intensive and short-lived task can be executed uninterruptedly in one data center, the overheads of live VM migration to run a task can be avoided. Furthermore, the service cost can be reduced, and the resource utilization can be improved simultaneously. It is worth noting that the proposed instance sharing model supports heterogeneous instance types. For example, Amazon EC2 can modify the type of reserved instances according to the changing demand during the service period [5]. And we can employ an elementary resource measure as the building block for various VM configurations, e.g., the Amazon EC2 compute units [11]. Without loss of generality, we focus on a single type of VM instance for simplicity in this paper.



(a) Service requests from HPC users in two data centers

(b) Traditional clusters with fixed capacity for HPC jobs

(c) Cloud resource reservation in individual data center, with $T = 6$ h, $\tau = 6$ h, $p_1^{\mathrm{o}} = p_2^{\mathrm{o}} = \$1$/hour, and $p_1^{\mathrm{r}} = \$2.5$ and $p_2^{\mathrm{r}} = \$3$ per 6 hours

(d) Cloud resource sharing in federated clouds for HPC jobs

Fig. 3. An example of the cloud-based HPC task execution in distributed data centers.

2.2 Cloud Pricing Schemes

Pricing method plays a key role in the cloud marketplace. For the reservation plan, users can pay a lower price p_i^{r} to reserve instances in data center i for certain time slots τ, no matter whether the instances are used or not. Reservation pricing strategies may be different among cloud providers. Some providers will give discounts when a large number of instances are purchased, such as, Amazon EC2 [5]. But on most occasions, the cost of reserved VMs is constant, for example, the cloud services offered by GoGrid and Rackspace. Furthermore, the cloud provider may offer various reservation options with different unit reservation length τ. Taking Amazon EC2 as an example, the per hour cost of reserved instances with a three-year contract is less than that with a one-year contract [5]. For simplicity, this paper will focus on fixed reservation costs with a constant τ. This assumption can be eased by dividing cloud users into different groups according to the configuration of reservation length τ, and making reservation decisions for each group of users individually. Before making the reservation decisions, users in each group prefer to utilize already reserved but unused instances from other groups to improve the resource utilization, just as they leverage the on-demand and reservation instances in this paper.

For the on-demand plan, users pay a relatively higher price to launch extra VMs at the beginning of every time slot dynamically. The on-demand plan allows cloud users to pay a fixed price in each time slot without any performance guarantees. For example, if the time slot is one hour, the hourly price of an on-demand VM is p_i^{o}. The fee of the instance that has been launched for n hours is $n \cdot p_i^{\mathrm{o}}$. Obviously, the cloud user can enjoy the cost-saving benefits only when the expenses of unit reservation are cheaper than that of the on-demand plan. We give Assumption 1 as follows:

Assumption 1. *If we want to achieve cost saving through instance reservation, the cloud pricing scheme must satisfy* $p_i^{\mathrm{r}}/\tau \leq p_i^{\mathrm{o}}$.

2.3 Cloud-Based HPC Job Execution

We illustrate the superiority of cloud-based model for HPC task execution. As shown in Fig. 3a, the HPC tasks from multiple users are depicted as rectangles, whose horizontal and vertical sides represent the requested execution time and computing resources, respectively. In the rectangle,

every task is marked with its arrival time and requested execution time. The HPC task execution over traditional physical clusters with fixed computing capacity is illustrated in Fig. 3b. With the batch scheduling mechanism, task C1, D1 and H2 are delayed because the computing resources are not enough, which generates an average waiting time of 0.83 hours. While task C1, D1 and H2 are waiting, the computing resources are not fully utilized. The average resource utilization is only 53.6 percent. This means that the traditional batch scheduling based HPC task execution over physical clusters may result in a waste of resources.

Then, we migrate HPC tasks into clouds for execution. The cloud provider organizes the resources in the form of VM instances, and multiple instances are organized as cloud-based virtual clusters. Using the MapReduce framework, the resource intensive HPC tasks are split into a number of smaller tasks, realizing the parallel task execution on the virtual clusters. With HPC-oriented cloud execution in individual data centers in Fig. 3c, each task can be allocated with an appropriate number of instances upon arrival, leading to nearly no SLA violation. We assume that each data center can make cloud resource reservations individually, with $T = 6$ h, $\tau = 6$ h, $p_1^{\mathrm{o}} = p_2^{\mathrm{o}} = \$1$ per hour, and $p_1^{\mathrm{r}} = \$2.5$ and $p_2^{\mathrm{r}} = \$3$ per 6 hours. For the maximum cost saving, the optimal solutions are to reserve 4 and 3 instances in the two data centers, respectively. The reserved resource utilization is 76.2 percent.

In Fig. 3d, considering the characteristics of HPC tasks, the resource sharing model in federated cloud data centers is illustrated. The broker federation aggregates all submitted jobs for scheduling. With the distributed computing technologies across data centers [12], [13], [14], part of small tasks belonging to E2 are scheduled into data center 1 by the broker federation, with no live VM migration overheads. Similarly, the whole C1 and part of H2 are reallocated to utilize unused reserved instances. The resource utilization is increased to 91.7 percent. The HPC job execution with cloud resource sharing has considerable advantages in saving cost and improving resource utilization.

3 Instance Reservation in a Data Center

In this section, we investigate the instance reservation problem in a single data center without the resource sharing at first. If the long-term request prediction is available, an offline heuristic-greedy algorithm is proposed to approximate optimal solutions with competitive ratio $2 - \frac{p_i^{\mathrm{r}}}{\tau \cdot p_i^{\mathrm{o}}}$. Furthermore, if only short-term demand estimations are available, a rolling-horizon based online resource reservation strategy is investigated with competitive ratio 3.

3.1 Instance Reservation Problem Formulation

We assume that each demand predictor module can estimate the required number of VM instances $d_i(t)$ for data center i, $i = 1, \ldots, N$. If long-term prediction is available, the demand prediction covers all T time slots, then an offline reservation strategy is proposed to determine how many instances $r_i(t)$ to reserve at the beginning of each time slot t, $r_i(t) \geq 0$, $t = 1, \ldots, T$. All reserved instances $r_i(t)$ will remain effective from time t to $t + \tau - 1$, which means the valid period of reserved instances is τ. At time t, the earliest instances that still remain effective were reserved at time $t - \tau + 1$. Therefore, the number of total available reservations $u_i(t)$ at t is given by

$$u_i(t) = \sum_{j=t-\tau+1}^{t} r_i(j), \tag{1}$$

where $r_i(j) := 0$ for all $j \leq 0$. If $u_i(t) \geq d_i(t)$, cloud user demands can be satisfied by reservation instances only. On the contrary, if $u_i(t) < d_i(t)$, the reservation instances are not enough to meet the demand $d_i(t)$, so the extra on-demand instances $o_i(t)$ need to be launched

$$o_i(t) = (d_i(t) - u_i(t))^+. \tag{2}$$

We define

$$(x)^+ := \max\{0, x\}. \tag{3}$$

The total fee C_i that cloud users should pay to data center i can be calculated as

$$C_i = \sum_{t=1}^{T} r_i(t) \cdot p_i^{\mathrm{r}} + o_i(t) \cdot p_i^{\mathrm{o}}, \tag{4}$$

where $\sum_{t=1}^{T} r_i(t) \cdot p_i^{\mathrm{r}}$ represents the total fee for the reserved instances, and $\sum_{t=1}^{T} o_i(t) \cdot p_i^{\mathrm{o}}$ represents the total fee of using on-demand instances. Therefore, the critical problem is to determine the number of reserved instances $r_i(t)$ to minimize the total service cost

$$\begin{aligned} \min_{r_i(t) \in \mathbb{Z}^+} \; & C_i = \sum_{t=1}^{T} r_i(t) \cdot p_i^{\mathrm{r}} + o_i(t) \cdot p_i^{\mathrm{o}} \\ \text{s.t.} \quad & \forall r_i(t) \leq \max_{t \in [1,T]} d_i(t), \end{aligned} \tag{5}$$

where the constraint means that the number of reserved instances should not exceed the maximum number of demands for the cost and resource saving. The optimization problem is an integer programming problem where optimization variables $r_i(t)$ should be nonnegative integers.

3.2 Hardness Analysis

We rigorously examine the hardness of the formulated problem in Eq. (5) by showing its non-convexity and nonlinearity, and then proving its NP-hardness.

Theorem 1. *The formulated problem in Eq. (5) is a non-convex nonlinear programming problem.*

The proof is deferred to Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSC.2018.2818121. In general, solving non-convex nonlinear programming problems is a proven difficult problem. At a fundamental level, the difficulty of globally solving non-convex nonlinear programming lies in the fact that the multiple of its local optimal solutions may not necessarily be global optimal solutions [15]. Therefore, for the formulated problem, relaxing the integer constraint $r_i(t) \in \mathbb{Z}^+$ to continuous constraint $r_i(t) \in \mathbb{R}^+$, and then rounding the obtained results to the nearest integers is not a simple or feasible solution.

Theorem 2. *The formulated problem in Eq. (5) is NP-hard.*

The proof is deferred to Appendix B, available in the online supplemental material. Due to its NP-complete nature, we cannot expect optimal and polynomial time solutions for this integer programming problem. Meanwhile, the non-convex optimization objective function makes this problem inapplicable to general-purpose solutions for integer programming, e.g., the branch-and-bound method [15]. Furthermore, in real-world cloud systems, the schedulers need to be fast and efficient. So for further complexity reduction, a highly efficient heuristic-greedy algorithm is proposed to address this issue.

3.3 Offline Solutions to Instance Reservation

To solve the integer programming problem above, if the long-term request prediction is available, a highly efficient heuristic-greedy reservation algorithm is proposed to obtain offline solutions. If the request prediction is within a single reservation period $T \leq \tau$, the proposed algorithm can reveal optimal reservation decisions for each data center. If the request prediction lasts for more than one reservation period $T > \tau$, the heuristic-greedy algorithm overcomes the prohibitive complexity of the integer programming, and achieves close-to-optimal solutions with the worst-case performance guarantee in the meantime.

3.3.1 The Optimal Heuristic Solution for $T \leq \tau$

As discussed above, the valid period of reserved instances lasts for τ time slots. If all demands are within a single reservation period, i.e., $T \leq \tau$, it is rational to make all the reservations at the beginning $t = 1$, because the reservation decision will remain valid over all time slots. Then we present a heuristic solution for optimal reservation decisions in the case $T \leq \tau$. To calculate the number of slots with demands greater than 0, we start off by introducing variable $\rho_i(t)$, which is defined as

$$\rho_i(t) := \begin{cases} 1, \text{ if } d_i(t) \geq 1, \\ 0, \text{ if } d_i(t) = 0. \end{cases} \tag{6}$$

Then variable ν_i is introduced to denote the number of time slots whose demand is greater than 0, $t \in [1, T]$

$$\nu_i := \sum_{t=1}^{T} \rho_i(t), \tag{7}$$

where ν_i determines whether service cost saving can be achieved through reservation. The $p_i^{\mathrm{r}}/p_i^{\mathrm{o}}$ can be treated as the heuristic factor for decision making. Mathematically, for cost saving, the cloud broker will reserve instances only if $\nu_i \geq p_i^{\mathrm{r}}/p_i^{\mathrm{o}}$ satisfies. Let σ be the number of reservations in the current step, which equals the minimum number of demands greater than 0, i.e., $\sigma = \min_{t\in[1,T],d_i(t)>0} d_i(t)$. Then we update the reservation decision with $r_i(1) = r_i(1) + \sigma$ and subtract satisfied demands with $d_i(t) = (d_i(t) - \sigma)^+$. The whole process is executed repeatedly until $\nu_i < p_i^{\mathrm{r}}/p_i^{\mathrm{o}}$. With the derived reservation decision $r_i(1)$, the overall expenditure of each cloud broker C_i can be calculated with Eq. (4).

It is easy to check that when $T \leq \tau$, Algorithm 1 degenerates into a heuristic solution which yields optimal decisions. It is worth noting that Algorithm 1 is highly efficient that only requires $\Theta(\max_{t\in[1,T]} d_i(t))$ execution time and $\Theta(T)$ searching space.

Algorithm 1. The Heuristic-Greedy Reservation

Input: Demand prediction $d_i(t), t \in [1, T]$.
Output: Reservation decision $r_i(t)$, $t = 1$ if $T \leq \tau$, and $t \in [1, T - \tau + 1]$ if $T > \tau$.
Initialization: $r_i(t) \leftarrow 0, t \in [1, T - \tau + 1]$.

```
if T ≤ τ then
  repeat
    σ ← min_{t∈[1,T],d_i(t)>0} d_i(t);
                          ▷ Get temporary decision
    d_i(t) ← (d_i(t) − σ)^+, t ∈ [1, T];
                          ▷ Update demands
    r_i(1) ← r_i(1) + σ;  ▷ Update reservations
  until ν_i < p_i^r/p_i^o
else if T > τ then
  repeat
    t* ← arg max_{t∈[1,T]} ν_i(t); ▷ Select interval I_i(t*)
    σ ← min_{t∈I_i(t*),d_i(t)>0} d_i(t);
                          ▷ Get temporary decision
    d_i(t) ← (d_i(t) − σ)^+, t ∈ I_i(t*);
                          ▷ Update demands in I_i(t*)
    r_i(t*) ← r_i(t*) + σ; ▷ Update reservations for t*
  until ν_i(t) < p_i^r/p_i^o, ∀t ∈ [1, T]
end if
Calculate service cost C_i using Eq. (4).
```

3.3.2 The Heuristic-Greedy Approximation for $T > \tau$

When demands last for more than one reservation period, i.e., $T > \tau$, we extend the above heuristic solution with the greedy strategy. The heuristic-greedy reservation algorithm is proposed for case $T > \tau$. To begin with, we divide the whole service period $\{1, 2, \ldots, T\}$ into $T - \tau + 1$ different time slots $\mathcal{I}_i(t)$, each with length τ

$$\mathcal{I}_i(t) := \{t, t + 1, \ldots, t + \tau - 1\}. \tag{8}$$

In each time slot $\mathcal{I}_i(t)$, we can apply the above optimal heuristic solution in Section 3.3.1. Similar to Eq. (7), variable $\nu_i(t)$ is introduced to reflect the reservation priority of each time slot $\mathcal{I}_i(t)$

$$\nu_i(t) := \sum_{j=t}^{t+\tau-1} \rho_i(j), \ t = 1, \ldots, T - \tau + 1. \tag{9}$$

The larger the value of $\nu_i(t)$, the more the service cost can be saved through reservation. At the same time, fewer reserved instances will be wasted. Namely, the larger value of $\nu_i(t)$, the higher reservation priority will be for time slot $\mathcal{I}_i(t)$. Based on the prior knowledge, the heuristic-greedy reservation is proposed as in Algorithm 1. Algorithm 1 summarizes the process of the heuristic-greedy reservation strategy for case $T > \tau$. Guided by the greedy method, the time slot $\mathcal{I}_i(t^*)$ with high priority will be selected first

$$t^* = \arg\max_{t\in[1,T]} \nu_i(t). \tag{10}$$

Especially, time slots with the same priority will be selected from the front to back along the time line. In the

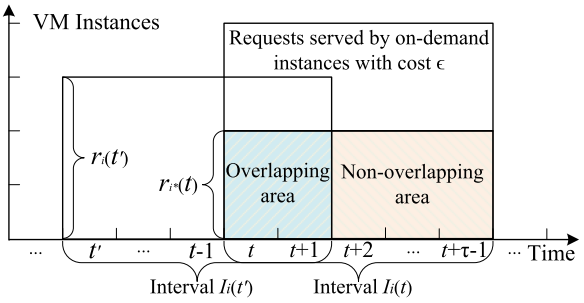


Fig. 4. The overlapping intervals when the proposed heuristic-greedy reservation algorithm reveals the worst-case cost-saving performance.

selected time slot $\mathcal{I}_i(t^*)$, the proposed heuristic solution is deployed for the reservation decisions in the current step, i.e., $\sigma = \min_{t\in\mathcal{I}_i(t^*),d_i(t)>0} d_i(t)$. Then we update the reservation decision at time t^* with $r_i(t^*)+=\sigma$, and subtract all satisfied demands in $\mathcal{I}_i(t^*)$ with

$$d_i(t) = (d_i(t) - \sigma)^+,\ t = t^*, t^*+1, \ldots, t^*+\tau-1. \tag{11}$$

We repeat the greedy selection process until no $\nu_i(t) \geq p_i^{\mathrm{r}}/p_i^{\mathrm{o}}$ exists for all time slots, which means no cost saving can be achieved then. The following Theorem 3 bounds the worst-case performance of the proposed heuristic-greedy reservation algorithm.

Theorem 3. *Under the circumstance of $T > \tau$, the proposed heuristic-greedy reservation algorithm incurs no more than $2 - \frac{p_i^{\mathrm{r}}}{\tau\cdot p_i^{\mathrm{o}}}$ times of the optimal service cost.*

Proof. We assume that the optimal reservation decisions are already known. Then on the basis of the optimal reservation decisions, the whole demands can be divided into the same number of fragments each with time length τ. Let $r_i^*(t)$ be one arbitrarily selected optimal reservation decision, and $\mathcal{I}_i(t)$ is the fragment that the reservations remain valid from time slot t to $t+\tau-1$. In fragment $\mathcal{I}_i(t)$, as discussed before, we can always find $\lceil p_r/p_o \rceil$ numbers of time slots with demands exceeding $r_i^*(t)$ for cost-saving reasons. If the number of time slots with demands exceeding $r_i^*(t)$ is less than $\lceil p_r/p_o \rceil$, the service cost can be further reduced through launching more on-demand instances. Let ϵ represent the possible on-demand cost, $\epsilon \geq 0$. The total optimal cost of fragment $\mathcal{I}_i(t)$ is given by

$$C_i^*(t) = r_i^*(t)\cdot p_r + \epsilon. \tag{12}$$

Then we investigate the performance of the heuristic-greedy reservation algorithm to evaluate its worst-case cost-saving performance. In fragment $\mathcal{I}_i(t)$, the proposed algorithm can make the reservation in three different types:

- The proposed algorithm makes only one reservation decision $r_i(t')$, and $t' = t$. Therefore, reservation fragment $\mathcal{I}_i(t)$ and $\mathcal{I}_i(t')$ are aligned so that the heuristic-greedy reservation algorithm yields the same reservation decision $r_i(t') = r_i^*(t)$, and the same service cost $C_i(t) = C_i^*(t)$.
- The proposed algorithm makes only one reservation decision $r_i(t')$, while $t' \neq t$. Just as shown in Fig. 4, fragment $\mathcal{I}_i(t)$ must overlap with at least one reservation fragment $\mathcal{I}_i(t')$. The non-overlapping part of requests will be served by on-demand instances only.
- The proposed algorithm makes several reservation decisions which remain valid in fragment $\mathcal{I}_i(t)$, so fragment $\mathcal{I}_i(t)$ overlaps with several reservation fragments.

When fragment $\mathcal{I}_i(t)$ overlaps with only one fragment $\mathcal{I}_i(t')$ and $t' \neq t$, the proposed algorithm yields the maximum service cost for fragment $\mathcal{I}_i(t)$. Because under the circumstances, we have the largest number of demands which are satisfied with on-demand instances. Just as shown in Fig. 4, the heuristic-greedy reservation algorithm yields the worst-case cost-saving performance.

Furthermore, we give the quantitative analysis of the worst-case service cost. It is worth noting that $r_i(t') \geq r_i^*(t)$ must hold. It is easy to understand that if $r_i(t') < r_i^*(t)$, additional reservations will be launched at time slot t for further cost reduction. For the same reason, the non-overlapping area in Fig. 4 has at most $\lceil p_r/p_o \rceil - 1$ time slots with demand greater than 0. With all discussions above, the worst-case service cost $C_i(t)$ of fragment $\mathcal{I}_i(t)$ incurred by the proposed algorithm is given by

$$C_i(t) = \left(\tau - \left\lceil\frac{p_r}{p_o}\right\rceil + 1\right)\cdot r_i(t^*)\cdot\frac{p_r}{\tau} + \left(\left\lceil\frac{p_r}{p_o}\right\rceil - 1\right)\cdot r_i(t^*)\cdot p_o + \epsilon, \tag{13}$$

In Eq. (13), the first part means the reservation cost of the overlapping area, the second part represents the on-demand cost of the non-overlapping area, and the last part ϵ indicates the possible on-demand cost that the optimal and the proposed heuristic-greedy reservation algorithms both have. For fragment $\mathcal{I}_i(t)$, we can conclude that

$$\begin{aligned}\frac{C_i(t)}{C_i^*(t)} &= \frac{\left(\tau - \left\lceil\frac{p_r}{p_o}\right\rceil + 1\right)\cdot r_i(t^*)\cdot\frac{p_r}{\tau} - \left(\left\lceil\frac{p_r}{p_o}\right\rceil - 1\right)\cdot r_i(t^*)\cdot p_o + \epsilon}{r_i(t^*)\cdot p_r + \epsilon} \\ &= 1 + \frac{\left(\left\lceil\frac{p_r}{p_o}\right\rceil - 1\right)\cdot r_i(t^*)\cdot\left(p_o - \frac{p_r}{\tau}\right)}{r_i(t^*)\cdot p_r + \epsilon} \\ &< 1 + \frac{p_r\cdot r_i(t^*)\cdot\left(1 - \frac{p_r}{\tau\cdot p_o}\right)}{r_i(t^*)\cdot p_r + \epsilon} \\ &\leq 2 - \frac{p_r}{\tau\cdot p_o}.\end{aligned} \tag{14}$$

For all fragments divided by the optimal reservation decisions, condition (14) holds. Therefore, the proposed heuristic-greedy reservation algorithm incurs less than $2 - \frac{p_r}{\tau\cdot p_o}$ times of the optimal cost. The proof completes. □

When $T > \tau$, Algorithm 1 is still highly efficient that only requires $O((T-\tau+1)\cdot\max_{t\in[1,T]} d_i(t))$ execution time and $O(T)$ searching space.

3.4 Online Solutions to Instance Reservation

Due to the sporadic characteristic of HPC user demands, many current cloud forecast models focus on the short-term predictions. Let λ be the time window length of the short-term prediction, $\lambda < T$. In order to fulfill the online

reservation for all time slots T, we employ the rolling-horizon strategy. At the beginning of time slot t, the demand predictor in data center i makes a precise short-term forecast in time window $\mathcal{H}_i(t)$

$$\mathcal{H}_i(t) := \{t, t+1, \ldots, t+\lambda-1\}. \tag{15}$$

In each window $\mathcal{H}_i(t)$, if $\lambda \leq \tau$, the heuristic solution in Section 3.3.1 can be implemented for reservation decision making. If $\lambda > \tau$, the heuristic-greedy method in Section 3.3.2 is leveraged in the time window. Then we move to the next time slot $t+1$, update the prediction information $\mathcal{H}_i(t+1)$, and repeat the process until $t = T-\lambda+1$. The proposed Algorithm 2 can be treated as a greedy strategy from the front to back along the time line. With the time window moving forward, the proposed algorithm reserves instances greedily until $\nu_i(t) < p_i^{\mathrm{r}}/p_i^{\mathrm{o}}, \forall t \in \mathcal{H}_i(t)$. Due to the fact that for cost saving, instances will be reserved only when the reservation priority $\nu_i(t)$ is no less than $p_i^{\mathrm{r}}/p_i^{\mathrm{o}}$, the proposed online algorithm remains valid for

$$\lceil p_i^{\mathrm{r}}/p_i^{\mathrm{o}} \rceil \leq \lambda \leq T. \tag{16}$$

Algorithm 2. The Rolling-Horizon Based Online Reservation

Input: Short-term prediction $d_i(t), t \in \mathcal{H}_i(t)$.
Output: The online reservation decision $r_i(t), t = [1, T-\tau+1]$.
Initialization: $r_i(t) \leftarrow 0, t = [1, T-\tau+1]$
1: **for** Time $t = 1$ to $T-\lambda+1$ **do**
2: Invoke Algorithm 1 for reservation decision $r_i(t)$ in time window $\mathcal{H}_i(t)$;
3: $d_i(t) \leftarrow (d_i(t) - r_i(t))^+, t \in \mathcal{H}_i(t)$;
▷ Update demands in $\mathcal{H}_i(t)$
4: **end for**
5: Calculate the service cost C_i using Eq. (4).

Compared with the offline Heuristic-Greedy reservation with global information, the online algorithm with only local information incurs a performance loss in $\mathcal{H}_i(t)$ for decision making.

Theorem 4. *Algorithm 2 incurs no more than 3 times of the optimal service cost.*

Proof. We assume that the optimal reservation decisions are already known, and the optimal cost is defined as C_i^*. Then we segments the demand period T into non-overlapping intervals $\{I_k\}$, each with length τ. We define

$$I_k := [(k-1)\tau + 1, k\tau], k = 1, 2, \ldots. \tag{17}$$

The heuristic solution can be utilized in intervals $\{I_k\}$ for local optimal solutions. The total service cost of the non-overlapping interval based method is defined as C_i^I. In [6], the authors proved that $C_i^I \leq 2C_i^*$ always holds.

Let $\{r_i(t)\}$ be the reservation decisions of the proposed online algorithm. We define a reservation interval-aligned if instances are reserved at the very beginnings of intervals $\{I_k\}$. Meanwhile, we call the rest of other reservations interval-overlapped. It is worth noting that a reservation can overlap two intervals at most. Based on the aligned and overlapped information, we segment user demands into two different parts: interval-aligned demands $\{d_i^a(t)\}$, and interval-overlapped demands $\{d_i^o(t)\}$. We have

$$d_i(t) = d_i^a(t) + d_i^o(t), t = 1, \ldots, T. \tag{18}$$

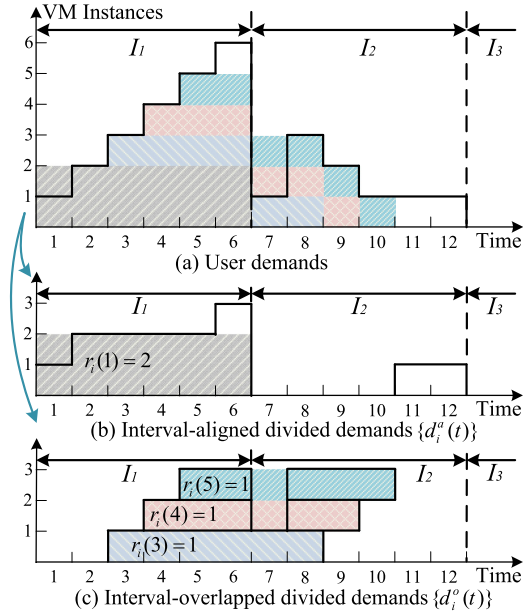


Fig. 5. An example of the online instance reservation is shown, with $p_r = \$0.75/\text{hour}$, $p_o = \$1/\text{hour}$, $\tau = 6$, $\lambda = 6$ and $T = 12$. The reservation decisions $r_i(1) = 2$, $r_i(3) = 1$, $r_i(4) = 1$ and $r_i(5) = 1$ are highlighted as the colored areas. The segmentation of user demands is also shown.

As shown in Fig. 5, we give an example to illustrate the segmentation method. With the reservation decisions derived by the online instance reservation algorithm, the demands served by interval-aligned reservations, e.g., $r_i(1)$, and possible on-demand instances are categorized into interval-aligned demands. Only the demands served by interval-overlapped reservations, e.g., $r_i(3)$, $r_i(4)$ and $r_i(5)$, are categorized into interval-overlapped demands. As $d_i^a(t) \leq d_i(t)$ for all t, the service cost of interval-aligned demands $C_i^a \leq C_i^I$ always holds. Moreover, the online algorithm reveals optimal reservation decisions for interval-overlapped demands. The service cost of interval-overlapped demands $C_i^o \leq C_i^*$ also holds. Therefore, $C_i = C_i^a + C_i^o \leq 3C_i^*$. The proof completes. □

Note that the costs of $\{d_i^a(t)\}$ and $\{d_i^o(t)\}$ will not reach their maximum values simultaneously. So the 3-competitiveness is only a loose bound. Evaluation results show that competitive ratio of the online algorithm is about 2. Algorithm 2 requires $O((T-\lambda+1)\cdot\lambda\cdot\max_{t\in[1,T]}(d_i(t)))$ execution time and $O(\lambda)$ searching space.

4 Task-Aware Instance Sharing in Federated Geo-Distributed Data Centers

In this section, facing submitted HPC jobs, the cloud broker federation is formed which aggregates all reserved but unused instances, and schedules all tasks in a real-time manner. For further service cost reduction and reservation resource utilization improvement, the broker federation allows all data centers to borrow spare capacity during peaks for short-lived and computation-intensive jobs, and

share unused capacities during valleys. With the proposed instance reservation strategy for cloud data center i, the broker has $e_i(t)$ already reserved but unused reservations in time slot t

$$e_i(t) = (u_i(t) - d_i(t))^+, \tag{19}$$

while the broker for other data center j may have $o_j(t)$ resource shortage. To avoid resource wastage, all cloud brokers form a stable federation to cooperate among geo-distributed data centers. The cloud federation exchanges the demand and reservation information of participating data centers, aiming at satisfying all requests with reserved instances as much as possible. This in turn reduces the service cost and improves the resource utilization as well. We define the cloud federation $\mathcal{F}$ as a set of cloud brokers where the corresponding data centers have agreed to share their unused capacities. Moreover, if all cloud brokers form one federation, i.e., $|\mathcal{F}| = |\mathcal{N}|$, we call it the grand federation.

We define a tuple $(\mu_\kappa^s, \mu_\kappa^e, d_\kappa, \omega_\kappa)$ for the submitted HPC task κ, $\kappa \in \mathcal{K}_i$, $i \in \mathcal{N}$, where μ_κ^s and μ_κ^e mean the start and end time, and d_κ means the required number of VM instances. In the distributed computing framework, e.g., Hadoop, the Hadoop Distributed File System (HDFS) usually breaks down very large files into data blocks with the same default size, which facilitates the efficiency of parallel task processing and reduces the traffic of data transfer [16]. So we let ω_κ represent the average used disk space per VM instance. As the compression techniques might be utilized when migrating tasks, the disk space usage can be considered as the upper bound of the traffic when migrating tasks.

As discussed above, the number of reserved instances is $u_i(t)$ for data center i at time t. The submitted HPC tasks preferentially utilize reserved instances in the local data center. If the local reserved instances are not enough, we may borrow spare capacity from other data centers. We directly set the instance sharing price as p_i^{r}/τ, which means the price of unused instances among data centers is equal to the reservation price per time unit. With the simple but efficient pricing scheme, no data center will maliciously compromise the cooperation by reserving needless instances for sale. Note that other dynamic pricing models can also be applied in our proposed instance sharing framework. Let $\theta_{\kappa,j}$ be the part of demands in d_κ that can be transferred to data center j for the largest possible cost saving, which can be calculated by the optimization problem as follows:

$$\begin{aligned} \min & \sum_{t=\mu_\kappa^s}^{\mu_\kappa^e} ((d_\kappa - \theta_{\kappa,j} - u_i(t))^+ + (\theta_{\kappa,j} - e_j(t))^+) \\ & \cdot p_i^{\mathrm{o}} + (e_j(t), \theta_{\kappa,j})^- \cdot \frac{p_i^{\mathrm{r}}}{\tau} + \theta_{\kappa,j}\omega_\kappa\varphi_i \\ \text{s.t.} \quad & 0 \le \theta_{\kappa,j} \le d_\kappa, \theta_{\kappa,j} \in \mathbb{Z}^+, \end{aligned} \tag{20}$$

where $\sum_{t=\mu_\kappa^s}^{\mu_\kappa^e} (d_\kappa - \theta_{\kappa,j} - u_i(t))^+ \cdot p_i^{\mathrm{o}}$ means the additional cost of launching on-demand instances in the local data center, $\sum_{t=\mu_\kappa^s}^{\mu_\kappa^e} (\theta_{\kappa,j} - e_j(t))^+ \cdot p_i^{\mathrm{o}} + (e_j(t), \theta_{\kappa,j})^- \cdot p_i^{\mathrm{r}}/\tau$ represents the cost of using on-demand and unused instances in data center j, and $\theta_{\kappa,j}\omega_\kappa\varphi_i$ means the largest possible data transfer cost.

We define $(x, y)^- := \min(x, y)$. As only one variable exists in the optimization problem, optimal θ_κ can easily be obtained by exhaustive searching. It is worth noting that $\theta_{\kappa,j}$ will be 0 if no cost can be saved through instance sharing. This ensures that only computational-intensive tasks will be transferred due to the cost of data transfer. The data-intensive tasks which require simple computation will be processed locally. Furthermore, as the reserved but unused instances show fragmentation characteristics which are not always available for all tasks, the short-lived tasks have higher opportunity to be transferred. The total cost $\Theta_{\kappa,j}$ cloud user should pay for instance sharing is given by

$$\begin{aligned} \Theta_{\kappa,j} = & \sum_{t=\mu_\kappa^s}^{\mu_\kappa^e} (\theta_{\kappa,j} - e_j(t))^+) \cdot p_i^{\mathrm{o}} \\ & + (e_j(t), \theta_{\kappa,j})^- \cdot p_i^{\mathrm{r}}/\tau + \theta_{\kappa,j}\omega_\kappa\varphi_i, \end{aligned} \tag{21}$$

which includes the cost cloud user should pay to j for using idle reservations and on-demand instances, plus the expenditure of data transfer. Similarly, the potential revenue of selling the unused reservations for j is given by $\sum_{t=\mu_\kappa^s}^{\mu_\kappa^e} (e_j(t), \theta_{\kappa,j})^- \cdot p_i^{\mathrm{r}}/\tau$. Then we update the demand by $d_\kappa = d_\kappa - \theta_{\kappa,j}$, and repeat the process until all data centers are considered. As part of demands in d_κ may be distributed to other data centers, the unused reservations in i should be updated with $e_i(t) = (u_i(t) - d_i(t) + \sum_{j \in \mathcal{N}} \theta_{\kappa,j})^+, \quad \mu_\kappa^s \le t \le \mu_\kappa^e$. Furthermore, if the reserved capacity from other data centers is not enough for θ_κ, on-demand instances $\sum_{t=\mu_\kappa^s}^{\mu_\kappa^e} (d_\kappa - u_i(t))^+$ from the local data center will be launched for the task. The real-time task-aware instance sharing is given by Algorithm 3 as shown above. It is worth noting that the algorithm runs for all data centers in parallel. With the proposed mechanism, the real-time resource sharing is achieved among distributed data centers, without incurring the overheads of live VM migration or the SLA violation to run HPC tasks.

The performance gain of Algorithm 3 is analyzed by the following examples. We assume that $C_i = \sum_{t=1}^{T} d_i(t) \cdot p_i^{\mathrm{o}} = \sum_{t=1}^{T} r_i(t) \cdot p_i^{\mathrm{r}}$ is satisfied for $\forall i$, which means all demands are already satisfied with local reserved instances. There is no need to transfer tasks among data centers. The performance gain of Algorithm 3 is 0 in this case. Then we consider another case that $C_i = \sum_{t=1}^{T} d_i(t) \cdot p_i^{\mathrm{o}} = \sum_{t=1}^{T} r_i(t) \cdot p_i^{\mathrm{r}} + o_i(t) \cdot p_i^{\mathrm{o}}$ for $\forall i$, which means no service cost reduction can be achieved with the instance reservation in a single data center. Then, with the instance sharing mechanism, for the maximum cost saving, it is possible that all reserved instances can be fully utilized, and no additional on-demand instances are needed with task transfer. Therefore, the performance gain of Algorithm 3 is in

$$\left(0, \sum_{i=1}^{N} \sum_{t=1}^{T} d_i(t) \cdot p_i^{\mathrm{o}} - r_i(t) \cdot p_i^{\mathrm{r}}\right). \tag{22}$$

The computational complexity of Algorithm 3 in data center i is given by $O(\max_{\kappa \in \mathcal{K}_i} d_\kappa \cdot N \cdot \|\mathcal{K}_i\|)$.

5 Performance Evaluation

In this section, we perform extensive evaluations driven by large volumes of real-world HPC workload traces, i.e., the

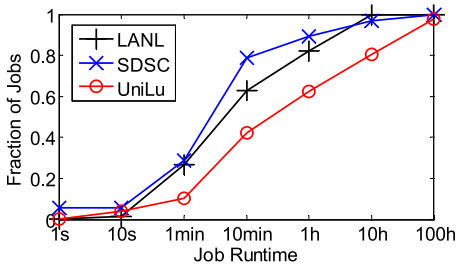


Fig. 6. The CDF of the job runtime derived from the three data centers.

Parallel Workloads Archive [8], to evaluate the performance of the proposed resource reservation algorithms in federated cloud data centers. The accurate prediction of cloud workload can be achieved as demonstrated in previous studies [17], [18]. Hence, in this evaluation we simply adopt the workload as the predicted request information.

Algorithm 3. Real-Time Task-Aware Instance Sharing

Input: Available reserved instances $u_i(t)$, unused reservations $e_i(t)$ at t, HPC task tuple $(\mu_\kappa^s, \mu_\kappa^e, d_\kappa, \omega_\kappa)$, $i \in \mathcal{N}$, $\kappa \in \mathcal{K}_i$, the price of unit outgoing traffic φ_i.

Output: Service cost $C_i^{\mathcal{F}}$ in the grand federation $\mathcal{F}$, $i \in \mathcal{N}$.

Initialization: $\theta_{\kappa,j} \leftarrow 0$, $C_i^{\mathcal{F}} \leftarrow \sum_{t=1}^{T} r_i(t) \cdot p_i^{\mathrm{r}}$; ▷ Cost of reserved instances

for $\kappa = 1$ to $\|\mathcal{K}_i\|$ **do**
  $d_\kappa \leftarrow d_\kappa - \min\{d_\kappa, \min_{\mu_\kappa^s \le t \le \mu_\kappa^e} u_i(t)\}$;
  $u_i(t) \leftarrow u_i(t) - \min\{d_\kappa, \min_{\mu_\kappa^s \le t \le \mu_\kappa^e} u_i(t)\}$, $\mu_\kappa^s \le t \le \mu_\kappa^e$; ▷ Serve with local reservations
  Sort data centers in ascending order based on the transfer delay between the user and data centers;
  for Data center $j = 1$ to N, $j \neq i$ **do**
    if $d_\kappa > 0$ **then**
      Get the demand $\theta_{\kappa,j}$ that can be transferred to data center j by Eq. (20);
      $d_\kappa \leftarrow d_\kappa - \theta_{\kappa,j}$; ▷ Update demands
      $C_i^{\mathcal{F}} \leftarrow C_i^{\mathcal{F}} + \Theta_{\kappa,j}$; ▷ Cost of instance sharing
      $C_j^{\mathcal{F}} \leftarrow C_j^{\mathcal{F}} - \sum_{t=\mu_\kappa^s}^{\mu_\kappa^e} (e_j(t), \theta_{\kappa,j})^- \cdot p_i^{\mathrm{r}}/\tau$; ▷ Revenue of instance sharing
      $u_j(t) \leftarrow u_j(t) - \theta_{\kappa,j}$, $\mu_\kappa^s \le t \le \mu_\kappa^e$; ▷ Update reservations in j
    end if
  end for
  $e_i(t) \leftarrow (u_i(t) - d_i(t) + \sum_{j \in \mathcal{N}} \theta_{\kappa,j})^+$, $\mu_\kappa^s \le t \le \mu_\kappa^e$; ▷ Update unused reservations
  $C_j^{\mathcal{F}} \leftarrow C_j^{\mathcal{F}} + \sum_{t=\mu_\kappa^s}^{\mu_\kappa^e} (d_\kappa - u_i(t))^+$; ▷ Cost of on-demand instances
end for

5.1 Datasets Description and Preprocessing

Data Sets. From the total available 36 real parallel workloads, three frequently used workload logs are selected for performance evaluation. They are: 1) The UniLu-Gaia containing 51,987 jobs. 2) The SDSC-BLUE containing 243,306 jobs. 3) The LANL-CM5 containing 122,060 jobs. For the workload in each data center, we extract 600 hours to perform the evaluation (i.e., 25 days, $T = 600$ h), and set the time slot to 1 hour. During the extracted period, the total numbers of needed instances are 671,360, 802,234 and 1,013,280 for the three data centers, respectively. For each HPC task, the request information, e.g., the user ID, submit time, requested number of processors and memory sizes and actual runtime of jobs, can be obtained from the trace. As shown in Fig. 6, the cumulative distribution function (CDF) of the job runtime is illustrated. Note that tasks running for less than one hour count for more than 62.35, 89.10 and 82.10 percent of all HPC jobs for the selected three parallel workloads, respectively. These short-lived jobs have higher opportunity to utilize idle reservations with the broker federation for uninterrupted execution.

For the reason of confidentiality, most publicly available workloads, including the utilized Parallel Workloads Archive, do not specify the detailed application for each submitted job. It is worth noting that the disk space usage of HPC applications ranges from a few Megabytes to Gigabytes [19]. Without loss of generality, the average used disk space per VM instance for each task is randomly generated following the power-law distribution [20]. The total data sizes of tasks range widely from 5 Megabytes to 65 Gigabytes, which can reflect different characteristics of HPC tasks to some degree, i.e., computation-intensive or data-intensive. The disk space usage can be considered as the upper bound of the traffic.

Instance Scheduling. We employ Amazon EC2 Cluster Compute Instances to evaluate the performance of proposed instance reservation mechanism (by simulation). For simplicity, we consider a uniform distribution of VM instances. Our evaluation is based on the compute-optimized `t2.small` instances, each instance with 1 processor and 2 GB memory. Then we take the selected three parallel workloads as input, scheduling all jobs into instances based on the required processors and memory sizes. We assume HPC jobs can share the same VM in one time slot only if they belong to the same cloud user for data security. The demand curves of the three data centers are illustrated as in Fig. 7, indicating how many instances are required by HPC jobs in each hour.

Pricing. The per hour prices of on-demand instances are listed in Table 2, which are the same as Amazon EC2 small instances. We assume each reservation is effective for 24 hours, $\tau = 24$. The reservation prices p_i^{r} are almost equal to purchasing on-demand instances for half a reservation period. If all demands are satisfied by reserved instances, the maximum cost saving rate will be $1 - \frac{p_i^{\mathrm{r}}}{\tau \cdot p_i^{\mathrm{o}}} \approx 50\%$. In addition, the prices of outgoing traffic for data transfer among data centers are also listed in Table 2.

Performance Baselines. To verify the superiority of the proposed offline heuristic-greedy reservation (HGR) algorithm, the 2-Approximation Heuristic (2AH) algorithm proposed in [6] is introduced for a fair comparison in service cost, resource utilization and data transfer size aspects. The 2AH algorithm segments the demand period into several non-overlapping intervals. Each interval has the same length τ as the reservation period. Then the cloud broker makes reservation decisions separately only at the very beginning of each interval. The 2AH algorithm reveals local optimal results for each interval, and incurs no more than twice of the minimum overall cost.

Furthermore, to verify the performance of the proposed rolling-horizon based online instance reservation (RHO) algorithm, the online algorithm proposed in [6] that makes

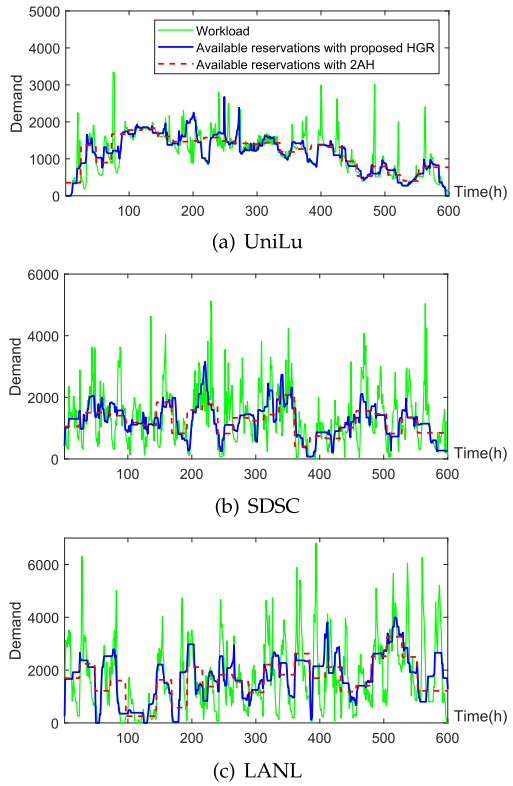


Fig. 7. The demand curves of the three geo-distributed data center logs are shown. The available numbers of reserved instances achieved by the 2AH and proposed HGR algorithm are also depicted when the cloud broker reserves instances individually for each data center.

reservation decisions based only on history is also introduced. An instance will be reserved if the accumulated cost incurred by the use of on-demand instances in the past reservation period, i.e., from time $t-\tau+1$ to time t, is no less than the cost of a reserved instance p_i^r. The online algorithm in [6] incurs at most 4 times of the minimum cost.

5.2 Performance of the Offline Resource Reservation

Now we evaluate the performance of the proposed reservation strategies from three aspects: the cost saving, resource utilization and data transfer size. In particular, when long-term demand predictions are reliable, the offline HGR algorithm is applied among geo-distributed data centers.

In Fig. 7, the real workloads from three distributed parallel systems are depicted. With the reservation service by the cloud broker, the 2AH algorithm and our proposed HGR algorithm are adopted for reservation decision making. To solve the NP-hard integer programming for all the three data centers, the 2AH algorithm only needs 1 s to converge. The proposed HGR algorithm needs a little bit longer time about 2.5 s to converge and yields close-to-optimal

TABLE 2
Prices of Geo-Distributed Clouds

Data Center	UniLu	LANL	SDSC
On-demand instance ($/hour)	0.060	0.065	0.075
Reserved instance ($/day)	0.750	0.715	0.800
Outgoing traffic ($/GB)	0.020	0.010	0.015

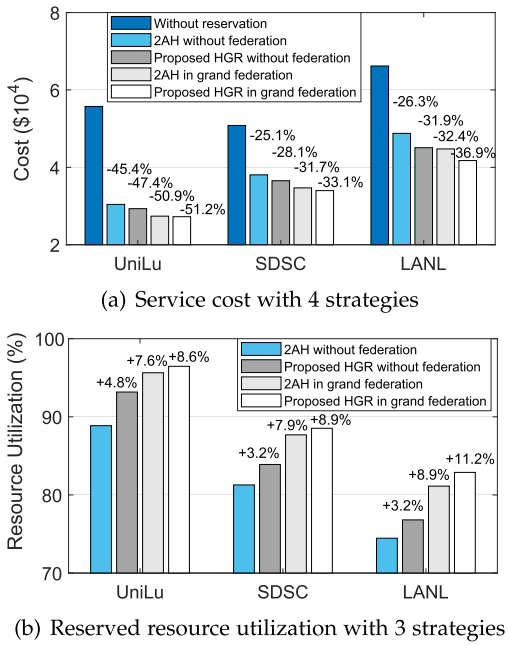


Fig. 8. The performance of different reservation strategies in each distributed data center.

reservation decisions. The numbers of available reservations achieved by two algorithms are also plotted in Fig. 7. In contrast, HGR is a more fine-grained strategy that can start to reserve new instances at any time. As shown in Fig. 7, the number of available reservations determined by the proposed HGR algorithm tracks the trends of user demands better. This means the proposed HGR strategy can provide cheaper and more resource-efficient cloud services.

The detailed performance comparison of the algorithms is given as in Figs. 8a and 8b. Without instance reservation, the service costs are $55,745, $50,831, and $66,194, respectively. The 2AH reservation strategy can bring cost saving of 45.4, 25.1 and 26.3 percent with reserved resource utilization 88.9, 81.3 and 74.5 percent. However, the coarse-grained 2AH strategy ignores the variation tendency of user demands and only makes reservation decisions at the beginnings of each time interval. Therefore, we can see that the fine-grained HGR algorithm outperforms the 2AH strategy in both service cost saving and resource utilization. It is worth noting that the reservation benefits are different for various data centers. The demand fluctuation degrees are 0.49, 0.66 and 0.78 for UniLu-Gaia, SDSC-BLUE and LANL-CM5, respectively. The reservation prefers steady demand patterns. When user requests are steady, they are largely served by reserved instances. The steadier the user demand patterns are, the more benefits the reservation can enjoy.

Facing widely fluctuating user demands in the three data centers, the grand cloud federation is formed. The federation smooths out all service requests through coordination, exploiting the benefits of reserved instances better. As shown in Fig. 8, if we extend 2AH with the proposed broker federation mechanism, the service cost can be reduced to $27,388, $34,693 and $44,756, and the reserved resource utilization is improved to 95.6, 87.7 and 81.1 percent, respectively. Moreover, with HGR in federation, the service cost can be further reduced to $27,240, $33,986 and $41,750, and the reserved resource utilization is improved to 96.5, 88.5

TABLE 3
Total Transferred Data Size (GB)

Data transfer path	with 2AH	with HGR
UniLu to SDSC	323.04	317.46
UniLu to LANL	280.72	367.02
SDSC to UniLu	397.17	149.89
SDSC to LANL	746.90	614.53
LANL to UniLu	415.94	220.08
LANL to SDSC	577.18	422.53

and 82.9 percent, respectively. Furthermore, the total sizes of transferred data among the three data centers can be seen in Table 3. It is worth noting that the proposed HGR algorithm can achieve a smaller data transfer size in total than that of 2AH. Compared with HGR, the non-overlapping interval based 2AH yields more fragmented demands that need more on-demand instances along with more unused reservations. Therefore, more data in total will be transferred with 2AH. The federation benefits are also different for various data centers: LANL-CM5 enjoys the highest benefit improvement, while UniLu-Gaia enjoys the lowest improvement. This is because with a steady demand pattern, most user requests are already satisfied by reservations, leading to less federation benefit improvement. In general, considerable performance improvements can be achieved with the proposed HGR algorithm.

As we cannot ensure the demand prediction is always accurate, how the prediction error affects the performance of HGR is quantitatively analyzed here. The accurate demand prediction is given by $d_i(t)$. Denote ε as the prediction error upper bound. The real predicted demand in each time slot is randomly generated in interval

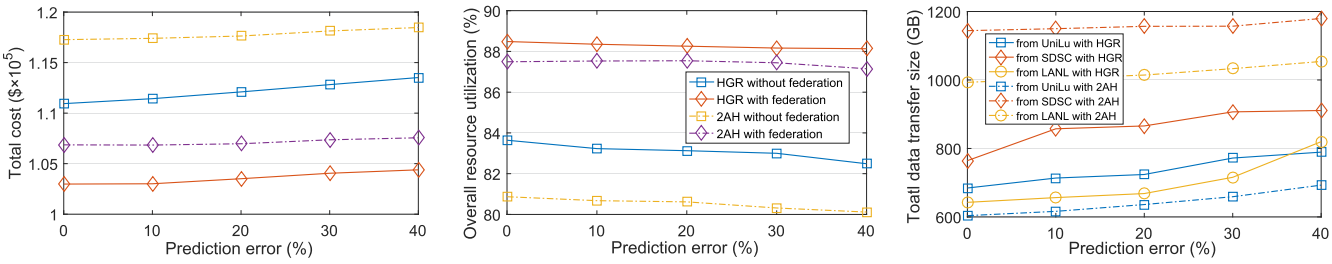
$$[(1-\varepsilon)d_i(t), (1+\varepsilon)d_i(t)].$$

As shown in Fig. 9, with the increase of prediction error ε, the instance reservation may not accurately track the trends of user demands. Therefore, the total service cost of the three data centers increases, and the overall reserved resource utilization decreases slightly. This also demonstrates that the proposed algorithm is robust to the prediction errors. Furthermore, with the increase of ε, more tasks may not be satisfied by local reservations. Meanwhile, more reservations may be underutilized in a single data center. So tasks have more chances to be transferred. For example, for HGR, with ε increasing from 0 to 40 percent, more cost reduction can be achieved with instance sharing (from 7.1 to 8.2 percent). This means the instance reservation will be more beneficial with instance sharing.

5.3 Performance of the Online Instance Reservation

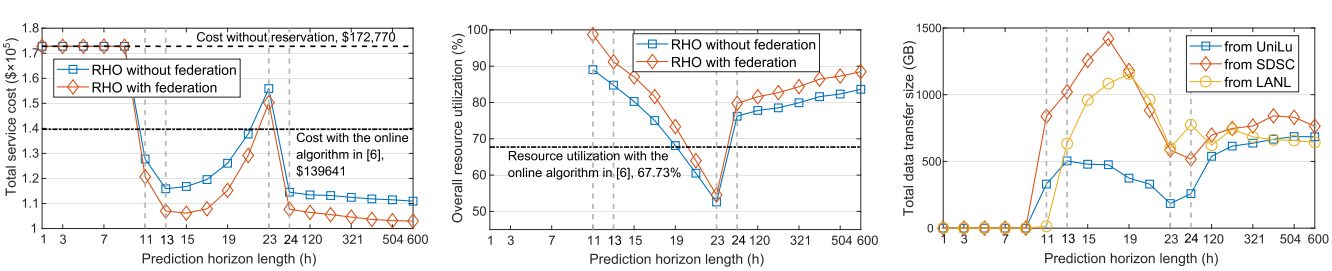
As shown in Fig. 10, we evaluate the performance of the proposed rolling-horizon-based online reservation (RHO) algorithm with the variation of demand prediction length λ. As discussed in Section 3, for cost saving, instances will be reserved only when the number of time slots whose demand is greater than 0 in prediction window λ is no less than the ratio between on-demand and reservation price p_i^r/p_i^o. As shown in Fig. 10, when $1 \le \lambda \le 10$, only on-demand instances are purchased. No reserved instances will be utilized. When $\lambda = 11$, as $\lceil p_1^r/p_1^o \rceil = \lceil p_3^r/p_3^o \rceil = 11$, users from UniLu and LANL begin to utilize reserved instances. Then when $\lambda = \lceil p_2^r/p_2^o \rceil = 13$, all users from the three data centers can enjoy the benefits of instance reservation.

When $\lambda = 13$, instances will be reserved only when the predicted demands in all 13 time slots are above 0. Meanwhile, the reserved instances will be fully utilized at least for the first 13 time slots. In this case, RHO tries to utilize fewer reserved instances (93,891) with a higher resource utilization (84.8 percent) to serve user requests. Then with the increase of λ from 13 to 23, it is unlikely that the reserved instances can always be fully utilized from time slots t to $t+\lambda-1$. In turn,



(a) Total service cost (b) Overall reserved resource utilization (c) Total transferred data size in federation

Fig. 9. The impact of the prediction error ε on HGR.



(a) Total service cost (b) Overall reserved resource utilization (c) Total transferred data size in federation

Fig. 10. The performance of the proposed RHO algorithm.

TABLE 4
The Performance of RHO ($\lambda = 13$) with the Prediction Error ε

Without federation	Cost ($)	Utilization	With federation	Cost ($)	Utilization	Traffic size (GB)
ε = 0%	115,874	84.8%	ε = 0%	107,014	91.2%	2,158.3
ε = 20%	116,840	83.7%	ε = 20%	108,017	90.6%	2,488.7
ε = 40%	119,227	82.3%	ε = 40%	110,519	89.8%	2,607.4

with a lower resource utilization, more instances should be reserved to serve the user requests. This means that the cost saving declines for every reserved instance. Therefore, with λ increases from 13 to 23, the service cost rises, and the resource utilization declines, just as shown in Figs. 10a and 10b. When $\lambda = 23$, the service cost and the total number of reservations reach the maximum \$155,890 and 193,717, respectively, with utilization 52.6 percent. As shown in Fig. 10c, with more available reservations in the federation, the transferred data size increases at the beginning. Then with more demands have already been satisfied with the local reservations, the transferred data size declines.

Furthermore, the valid length of reserved instances is $\tau = 24$. This means when $\lambda < 24$, due to the lack of demand information for all τ time slots, we cannot ensure RHO always makes appropriate reservation decisions. On the contrary, when $\lambda = 24$, as discussed in Section 3, RHO ensures the reservation decision is optimal with the largest possible cost saving for the current τ time slots. Therefore, the service costs reduce dramatically when λ increases from 23 to 24. Then, along with the further increase of λ, more time slots of demand predictions are achieved. Therefore, RHO has more choices to make more appropriate reservation decisions from t to $t + \lambda - \tau$. Fewer reserved instances are launched to accommodate more user requests from $\lambda = 24$ (126,769) to $\lambda = 600$ (106,661). The service cost with RHO declines, and the reserved resource utilization increases gradually with the increase of λ from 24 to 600. While $\lambda = 600$, RHO degenerates to the proposed offline HGR algorithm with the highest cost saving benefits.

Fig. 10a demonstrates that the competitive ratio of the proposed RHO algorithm is about 2. With the global demand information, HGR can approximate the optimal reservation decision with the competitive ratio $2 - \frac{p_i^r}{\tau \cdot p_i^o} \approx 1.5$. So the optimal service costs are in \$[68,653, 102,980]. When $\lambda = 23$, RHO yields the maximum cost \$155,890. It is easy to check that the competitive ratio is about 2. Fig. 10 also demonstrates that the federation service can always bring more service cost reduction and resource utilization improvement. As the online algorithm in [6] makes reservation decisions based only on the history, we cannot ensure whether the instance sharing is beneficial or not in the future. Therefore, we do not extend the online algorithm in [6] with the instance sharing mechanism.

In general, the prediction window length plays a key role affecting the performance of online instance reservation. When $\lambda = 13$, the total expense is \$115,874 with reserved resource utilization 84.8 percent, which is close to the results in the proposed offline HGR algorithm where the cost and resource utilization are \$110,938 and 83.9 percent, respectively. In fact, the shorter the length λ is, the more accurate the demand prediction can be achieved. According to the observations in Fig. 10, we can make short-term predictions to ensure considerable benefits of instance sharing.

The impact of the prediction error ε on RHO is also quantitatively analyzed. Let us fix $\lambda = 13$. As listed in Table 4, with the increase of ε from 0 to 40 percent, the total service cost increases slightly by about 3 percent, and the overall reserved resource utilization decreases by about 2 percent. The total size of data transfer traffic among data centers is increased by 20.8 percent. This demonstrates that the proposed RHO algorithm is also robust to the randomized prediction errors.

6 Related Work

6.1 HPC-Oriented Cloud Computing

Recently, HPC-oriented cloud computing has received much attention from both academia and industry communities. For example, people explored the feasibility of HPC-optimized clouds, such as Magellan [21], Amazon EC2 Cluster Compute [22] and Google Cloud Platform [23]. Wang et al. [12] designed G-Hadoop, a MapReduce framework that aims to realize large scale distributed computing across multiple data centers. The optimization of service cost and resource utilization were considered as one of the most important research topics in cloud-based HPC application scheduling [13], [18], [24]. Garg et al. [13] proposed several heuristic algorithms for HPC workload scheduling in distributed cloud data centers. As a follow-up study, Kessaci et al. [24] proposed a meta-heuristic approach to solve the same problem. However, they mainly focus on reducing the energy consumption of data centers to increase the profit of cloud providers. From another point of view, Niu et al. [18] investigated the reservation strategy to build variable-size virtual clusters for cost-effective HPC cloud resource provisioning. Inspired by previous studies, for further service cost reduction and instance utilization improvement, we establish a broker federation to realize the reservation resource sharing for HPC jobs among geo-distributed data centers.

6.2 Resource Sharing in Cloud Federation

Early approaches to cloud federation can be found in [25], [26], [27], [28]. Buyya et al. [25] illustrated the vision, architecture, and challenges in cloud federation environments. Celesti et al. [26] described a cloud federation scenario where the home cloud provider could rent resources from foreign clouds if demands of home users are not satisfied. But the incentives of the foreign clouds for helping are not specified clearly. Along the same lines, Goiri et al. [27] investigated a profit-driven cloud federation that allows a provider to dynamically insource or outsource resources to other providers based on demand variations. Samaan et al. [11] proposed a repeated game based capacity sharing mechanism in a federation of hybrid cloud providers, with the objective of maximizing the profits when the future workload fluctuations are uncertain. From another perspective, Tang et al. [28]

considered the history resource allocation information, investigating the hierarchical resource sharing with long-term fairness.

However, all these previous studies ignore the task characteristics in the resource sharing, which decreases the resource utilization. Zhang et al. [29] designed a novel truthful online auction mechanism to support the heterogeneous user demands. Yi et al. [30] proposed a new pricing framework, exploiting the task elasticity to achieve the fairness, resource efficiency, and revenue maximization simultaneously. Hindman et al. [31] pointed out that short-lived tasks allow diverse cluster computing frameworks for efficient resource sharing. Liu et al. [32] considered the job size, and leveraged the complementary features of job requirements on different resource types in the job packing for resource sharing in clouds. Yi et al. [33] proposed a container-based framework *Cocoa* to pack small and short jobs into the proper group buying deals. *Cocoa* allows flexible resource sharing among different users, achieving the cost effectiveness for cloud users and the resource efficiency for the provider. In this paper, dynamic instance reservation strategies based on time varying user demands are proposed. Unlike previous studies, we investigate the sharing mechanism for reserved but unused instances. In fact, cloud provider, i.e., Amazon EC2, has already laid down guidelines about reselling unused reservations when users have owned the reserved instances after 30 days [34]. Unlike the existing long-term reservation sharing, the fragmented unused reservations during the service period are reused for short-lived jobs to improve the resource utilization in federated data centers.

6.3 Mathematical Programming Method

Through cloud federation, the capacity planning and cost optimization can be addressed efficiently to provide dependable cloud services. In this area, mathematical programming methods have been adopted widely. For instance, Chaisiri et al. [35] applied the stochastic programming approach for resource provisioning offered by multiple cloud providers. This algorithm can optimally make the tradeoff between the reservation and on-demand plan with uncertain demand and price. Wang et al. [6] formulated the resource reservation as a dynamic programming problem, using a set of recursive Bellman equations to achieve the minimal expenditure, and developing several approximation algorithms to obtain near-optimal results. Khatua et al. [36] formulated the resource reservation as an integer programming problem, and proposed several heuristic-based polynomial time algorithms to find the near optimal solutions. Now few researches on resource management have considered utilizing already reserved but unused instances. The nonlinear integer programming has NP-hard complexity in obtaining the solution, which restricts its wide application. This paper proposes an efficient heuristic-greedy algorithm to ease the complexity with the worst-case performance guarantee.

7 Conclusion and Future Work

In this paper, dynamic cloud instance reservation strategies are investigated among federated clouds for the joint optimization of HPC service cost and resource utilization. We pursue a mathematical programming optimization framework, and propose two highly efficient approximation algorithms to determine when and how many instances to reserve in each data center. Then by utilizing the complementary feature of HPC demand patterns among geo-distributed data centers, the cloud broker federation is formed that can smooth out the demand curve through coordination. To exploit the benefits of federated resource sharing, the broker federation reallocates the reserved but unused instances to computation-intensive and short-lived jobs from another data center for uninterrupted execution, without incurring the overheads of live instance migration or SLA violations. Finally, we evaluated the proposed schemes through extensive performance evaluations.

In this paper, the grand formation of data center federation is adopted aiming at providing cost-effective cloud-based HPC services. In the future work, for-profit data center federation and competitive cloud providers will be studied for more flexible federation formation, exploiting the benefits of cooperative instance reservation better.

Acknowledgments

The authors would like to acknowledge that this work was partially supported by the National Natural Science Foundation of China (Grant Nos. 61772558, 61379111, 61672537 and 61672539), and in part by NSERC, CFI and BCKDF. The work is extended from [37]. This new paper designs novel algorithms with performance analysis, proofs, and evaluation results.

References

[1] G. Mateescu, W. Gentzsch, and C. J. Ribbens, "Hybrid computing—Where HPC meets grid and cloud computing," *Future Generation Comput. Syst.*, vol. 27, no. 5, pp. 440–453, 2011.

[2] A. G. Carlyle, S. L. Harrell, and P. M. Smith, "Cost-effective HPC: The community or the cloud?" in *Proc. IEEE 2nd Int. Conf. Cloud Comput. Technol. Sci.*, 2010, pp. 169–176.

[3] Double-digit growth forecast for the worldwide big data and business analytics market through 2020 led by banking and manufacturing investments. 2016. [Online]. Available: http://www.idc.com/getdoc.jsp?containerId=prUS41826116

[4] M. Mazzucco and M. Dumas, "Reserved or on-demand instances? A revenue maximization model for cloud providers," in *Proc. IEEE Int. Conf. Cloud Comput.*, 2011, pp. 428–435.

[5] Amazon EC2 Reserved Instances. 2018. [Online]. Available: https://aws.amazon.com/ec2/pricing/reserved-instances/

[6] W. Wang, D. Niu, B. Liang, and B. Li, "Dynamic cloud instance acquisition via IaaS cloud brokerage," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 6, pp. 1580–1593, Jun. 2015.

[7] M. Vouk, "Cloud computing issues, research and implementations," in *Proc. Int. Conf. Inf. Technol. Interfaces*, 2008, pp. 31–40.

[8] D. G. Feitelson, "Parallel workloads archive: Standard workload format," 2013. [Online]. Available: http://www.cs.huji.ac.il/labs/parallel/workload/swf.html

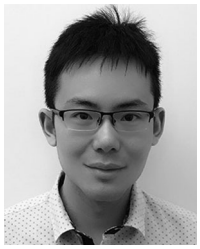
[9] N. Laoutaris, G. Smaragdakis, P. Rodriguez, and R. Sundaram, "Delay tolerant bulk data transfers on the Internet," *IEEE/ACM Trans. Netw.*, vol. 21, no. 6, pp. 1852–1865, Dec. 2013.

[10] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Virtual infrastructure management in private and hybrid clouds," *IEEE Internet Comput.*, vol. 13, no. 5, pp. 14–22, Sep./Oct. 2009.

[11] N. Samaan, "A novel economic sharing model in a federation of selfish cloud providers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 1, pp. 12–21, Jan. 2014.

[12] L. Wang, J. Tao, R. Ranjan, H. Marten, A. Streit, J. Chen, and D. Chen, "G-Hadoop: MapReduce across distributed data centers for data-intensive computing," *Future Generation Comput. Syst.*, vol. 29, no. 3, pp. 739–750, 2013.

[13] S. Garg, C. Yeo, A. Anandasivam, and R. Buyya, "Environment-conscious scheduling of HPC applications on distributed cloud-oriented data centers," *J. Parallel Distrib. Comput.*, vol. 71, no. 6, pp. 732–749, 2011.
[14] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica, "Low latency geo-distributed data analytics," in *Proc. ACM Conf. Special Interest Group Data Commun.*, 2015, pp. 421–434.
[15] D. P. Bertsekas, *Nonlinear Programming*. Belmont, MA, USA: Athena Scientific, 1999.
[16] HDFS architecture guide. (2018). [Online]. Available: https://hadoop.apache.org/
[17] J. Liu, H. Shen, and L. Chen, "CORP: Cooperative opportunistic resource provisioning for short-lived jobs in cloud systems," in *Proc. IEEE Int. Conf. Cluster Comput.*, 2016, pp. 90–99.
[18] S. Niu, J. Zhai, X. Ma, X. Tang, W. Chen, and W. Zheng, "Building semi-elastic virtual clusters for cost-effective HPC cloud resource provisioning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 7, pp. 1915–1928, Jul. 2016.
[19] R. Cheveresan, M. Ramsay, C. Feucht, and I. Sharapov, "Characteristics of workloads used in high performance and technical computing," in *Proc. ACM Annu. Int. Conf. Supercomput.*, 2007, pp. 73–82.
[20] M. E. Crovella and A. Bestavros, "Self-similarity in World Wide Web traffic: Evidence and possible causes," *IEEE/ACM Trans. Netw.*, vol. 5, no. 6, pp 835–846, Dec. 1997.
[21] K. Yelick, S. Coghlan, B. Draney, and R. S. Canon, "The Magellan report on cloud computing for science," U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research (ASCR), Washington, DC, USA, Dec. 2011.
[22] High Performance Computing (HPC) on AWS. (2018). [Online]. Available: http://aws.amazon.com/hpc-applications
[23] R. Prodan, M. Sperk, and S. Ostermann, "Evaluating high-performance computing on Google app engine," *IEEE Softw.*, vol. 29, no. 2, pp. 52–58, Mar./Apr. 2012.
[24] Y. Kessaci, N. Melab, and E. Talbi, "A Pareto-based metaheuristic for scheduling HPC applications on a geographically distributed cloud federation," *Cluster Comput.*, vol. 16, no. 3, pp. 451–468, 2012.
[25] R. Buyya, R. Ranjan, and R. Calheiros, "InterCloud: Utility-oriented federation of cloud computing environments for scaling of application services," in *Proc. Int. Conf. Algorithms Archit. Parallel Process.*, 2010, pp. 13–31.
[26] A. Celesti, F. Tusa, M. Villari, and A. Puliafito, "How to enhance cloud architectures to enable cross-federation," in *Proc. IEEE Int. Conf. Cloud Comput.*, 2010, pp. 337–345.
[27] I. Goiri, J. Guitart, and J. Torres, "Characterizing cloud federation for enhancing providers' profit," in *Proc. IEEE Int. Conf. Cloud Comput.*, 2010, pp. 123–130.
[28] S. Tang, B. S. Lee, and B. He, "Fair resource allocation for data-intensive computing in the cloud," *IEEE Trans. Serv. Comput.*, vol. 11, no. 1, pp. 20–33, Jan./Feb. 2018.
[29] H. Zhang, H. Jiang, B. Li, F. Liu, A. V. Vasilakos, and J. Liu, "A framework for truthful online auctions in cloud computing with heterogeneous user demands," *IEEE Trans. Comput.*, vol. 65, no. 3, pp. 805–818, Mar. 2016.
[30] X. Yi, F. Liu, Z. Li, and H. Jin, "Flexible instance: Meeting deadlines of delay tolerant jobs in the cloud with dynamic pricing," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, 2016, pp. 415–424.
[31] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. Joseph, R. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center," in *Proc. USENIX Conf. Netw. Syst. Des. Implementation*, 2011, pp. 295–308.
[32] J. Liu, H. Shen, and H. S. Narman, "CCRP: Customized cooperative resource provisioning for high resource utilization in clouds," in *Proc. IEEE Int. Conf. Big Data*, 2016, pp. 243–252.
[33] X. Yi, F. Liu, D. Niu, H. Jin, and J. Lui, "Cocoa: Dynamic container-based group buying strategies for cloud computing," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 2, no. 2, pp. 8–38, 2017.
[34] Selling in the Reserved Instance Marketplace. (2018). [Online]. Available: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ri-market-selling-guide.html
[35] S. Chaisiri, L. Bu-Sung, and D. Niyato, "Optimization of resource provisioning cost in cloud computing," *IEEE Trans. Serv. Comput.*, vol. 5, no. 2, pp. 164–177, Apr.-Jun. 2012.
[36] S. Khatua, P. K. Sur, R. K. Das, and N. Mukherjee, "Heuristic-based resource reservation strategies for public cloud," *IEEE Trans. Cloud Comput.*, vol. 4, no. 4, pp. 392–401, Oct.–Dec. 2016.
[37] K. Liu, J. Peng, W. Liu, P. Yao, and Z. Huang, "Dynamic resource reservation via broker federation in cloud service: A fine-grained heuristic-based approach," in *Proc. IEEE Global Commun. Conf.*, 2014, pp. 2338–2343.

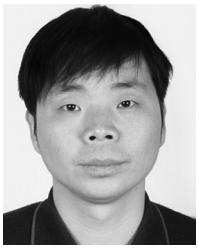


Kaiyang Liu received the BS degree from the School of Information Science and Engineering, Central South University, in 2012. He is currently working toward the PhD degree in the School of Information Science and Engineering, Central South University. His general research interests cover the broad area of wireless communication and computer networking, with special emphasis on resource management and scheduling in cloud computing systems. He is a student member of the IEEE.

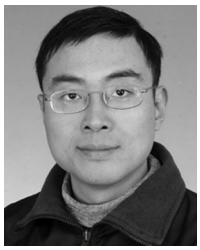
Jun Peng received the BS degree from Xiangtan University, the MSc degree from the National University of Defense Technology, China, in 1987 and 1990, respectively, and the PhD degree from Central South University, in 2005. She is a professor in the School of Information Science and Engineering, Central South of University, China. From 2006 to 2007, she was with the School of Electrical and Computer Science, University of Central Florida, as a visiting scholar. Her research interests include cooperative control, cloud computing, and wireless communications. She is a member of the IEEE.

Boyang Yu received the bachelor's and master's degrees in computer science from Nankai University, China, in 2006 and 2009, respectively, and the PhD degree from the Department of Computer Science, University of Victoria, Canada, in 2016. His current research areas include networked systems, distributed systems, and cloud computing, with special focus on the analysis and optimization in the data-intensive services. He is a student member of the IEEE.

Weirong Liu received the BE degree in computer software engineering, the ME degree in computer application technology from Central South University, Changsha, China, in 1998 and 2003, respectively, and the PhD degree in control theory and control engineering from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 2007. Since 2008, he has been a faculty member with the School of Information Science and Engineering, Central South University, where he is currently an associate professor. From 2016 to 2017, he was a visiting scholar with the Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, Canada. His current research interests include reinforcement learning, neural networks, wireless sensor networks, network protocol, smart grid, and microgrid. He received the Best Paper Award from the 7th Chinese Conference on Cloud Computing in 2016. He is a member of the IEEE.



Zhiwu Huang received the BS degree in industrial automation from Xiangtan University, in 1987, the MS degree in industrial automation from the Department of Automatic Control, University of Science and Technology Beijing, in 1989, and the PhD degree in control theory and control engineering from Central South University, in 2006. He is a professor with the School of Information Science and Engineering, Central South of University, China. From 2008 to 2009, he was with the School of Computer Science and Electronic Engineering, University of Essex, United Kingdom, as a visiting scholar. His research interests include fault diagnostic technique and cooperative control. He is a member of the IEEE.



Jianping Pan received the bachelor's and PhD degrees in computer science from Southeast University, Nanjing, Jiangsu, China. He is currently a professor of computer science with the University of Victoria, Victoria, British Columbia, Canada. He did his postdoctoral research with the University of Waterloo, Waterloo, Ontario, Canada. He also worked with Fujitsu Labs and NTT Labs. His area of specialization is computer networks and distributed systems, and his current research interests include protocols for advanced networking, performance analysis of networked systems, and applied network security. He received the IEICE Best Paper Award in 2009, the Telecommunications Advancement Foundation's Telesys Award in 2010, the WCSP 2011 Best Paper Award, the IEEE Globecom 2011 Best Paper Award, the JSPS Invitation Fellowship in 2012, the IEEE ICC 2013 Best Paper Award, and the NSERC DAS Award in 2016, and has been serving on the technical program committees of major computer communications and networking conferences including IEEE INFOCOM, ICC, Globecom, WCNC, and CCNC. He was the Ad Hoc and Sensor Networking Symposium co-chair of IEEE Globecom 2012 and an associate editor of the *IEEE Transactions on Vehicular Technology*. He is a senior member of the IEEE and ACM.