GMTA: A Geo-Aware Multi-Agent Task Allocation Approach for Scientific Workflows in Container-Based Cloud

Meng Niu, Bo Cheng, *Member, IEEE*, Yimeng Feng, and Junliang Chen

***Abstract*—Scientific workflow scheduling is one of the most challenging problems in cloud computing because of the large-scale computing tasks and massive data volumes involved. A cloud system is a distributed system that follows the on-demand resource provisioning and pay-per-use billing model. Therefore, practical scheduling approaches are essential for good workflow performance and low overheads. This paper proposes a novel workflow allocation approach, the *Geo-aware Multiagent Task Allocation Approach* (GMTA), which aims to optimize large-scale scientific workflow execution in container-based clouds. GMTA is an agent-based workflow allocation method that includes a market-like agent negotiation mechanism and a dynamic workflow restructuring strategy. It decreases workflow makespans and traffic overheads by reasonable task replications. Furthermore, the performance of GMTA is verified on real scientific workflows in the CloudSim environment.**

***Index Terms*—Geo-aware, scientific workflow, task allocation, multi-agent system, container cloud.**

I. Introduction

Scientific workflows are widely used in large-scale scientific computing in various fields, such as bioinformatics, astronomy, and physics [1]. Generally, large-scale scientific computing requires massive computational capabilities. Fortunately, by leveraging virtualization, cloud computing provides a high-performance environment for scientific workflow execution. Cloud vendors such as Amazon, Google, Microsoft, and IBM have globally distributed data centers (DCs) and can provide sufficient resources for scientific workflows [2].

Access to cloud resources is provided following the pay-per-use billing model. This model requires scheduling mechanisms to ensure cost-effective resource use. Resource scheduling problems have been studied for many years. However, the scientific workflow context introduces new challenges [3].

A workflow is a series of tasks that work together to achieve a goal. During this cooperative process, the tasks need to exchange data and synchronize their status information. However, because of resource limitations, the various tasks of a workflow may be dispersed throughout the cloud in accordance with the resource distribution. Data transmission between such geo-dispersed tasks takes time and is expensive. Moreover, the hosts must idle while waiting for data. Therefore, the geographic factor is a critical concern for workflow allocation mechanisms. Traditional cloud-based workflow allocation algorithms allocate tasks to virtual machines (VMs) while satisfying given deadline and resource constraints [4]–[6]. Moreover, a novel approach has been proposed in which the critical tasks of a workflow are replicated to refine the workflow structure and decrease the workflow makespan. However, replicating tasks in a VM-based cloud is a complicated process because rebuilding a VM and the environment on which a task depends takes considerable time.

Fortunately, at present, there is an emerging virtualization technology known as a container [7]. A container is a standard unit of tasks that includes only a program project and its necessary dependencies. In contrast to VMs, containers do not require virtualized infrastructure, and there is no virtual operating system layer. Therefore, containers are lighter than VMs and can be migrated and restarted faster [8], [9].

Consequently, in a container-based cloud, it is easy to replicate tasks on the critical path, thereby improving the parallelism of a workflow [10], [11]. However, the extra container instances of such replicated tasks require additional resources. Moreover, blindly making replications will cause unnecessary extra traffic costs. A multiagent system (MAS) is a standard negotiation mechanism in a dynamic resource-limited distributed environment [12]–[14]. In a MAS, multiple individual agents cooperate, coordinate, and negotiate to balance resource overheads and makespans.

Moreover, partitioning a workflow before allocation can simplify the dependencies among tasks and improve the performance of workflow allocation algorithms [15], [16].

This paper proposes the Geo-aware Multiagent Task Allocation Approach (GMTA), which is a geo-aware multiagent-based approach for scientific workflow allocation in container-based clouds. GMTA is based on an MAS, in which agents negotiate to allocate the tasks of a workflow. Through an additional middleman agent, GMTA transforms the workflow allocation problem into a task auction problem. Moreover, GMTA partitions the workflow before allocation.

Manuscript received December 10, 2019; revised March 26, 2020; accepted May 18, 2020. Date of publication May 21, 2020; date of current version September 9, 2020. This work is supported by the National Key Research and Development Program of China under grant 2018YFB1003804, in part by the National Natural Science Foundation of China under grant 61921003, 61972043. The associate editor coordinating the review of this article and approving it for publication was M. J. Khabbaz. *(Corresponding author: Bo Cheng.)*

The authors are with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100000, China (e-mail: mengniu@bupt.edu.cn; chengbo@bupt.edu.cn).

Digital Object Identifier 10.1109/TNSM.2020.2996304

The tasks in each partition are independent and do not depend on other tasks in the same partition. The task agents and host agents follow the improved contract net protocol (CNP) to complete the task auctions for each partition as a batch.

Moreover, GMTA is a dynamic mechanism; the auctioning of tasks proceeds gradually as the workflow advances. The host agents bid based on the real-time status of the hosts. However, when a task fails, it needs to be re-auctioned. Furthermore, a strategy is used to reasonably insert local replications of critical tasks to eliminate geo-dispersed communication bottlenecks and speed up the workflow. To this end, GMTA adopts a geo-aware cost model to weigh and balance the cost of replications and the makespan. GMTA aims to decrease the makespan while using resources efficiently.

Finally, this paper examines the actual execution results of GMTA for real scientific workflows based on CloudSim. GMTA decreases the dependencies among partitions and refines the workflow structure. Moreover, GMTA improves workflow parallelism and decreases the makespan of the whole workflow.

The contributions of this paper are as follows:

- Proposing a MAS-based dynamic scientific workflow allocation approach, GMTA, which optimizes the makespan while ensuring cost efficiency;
- Proposing a container-based critical task replication strategy to eliminate the bottlenecks caused by communication among geo-dispersed entities;
- Proposing a partition-based workflow preprocessing scheme for GMTA to refine the dependencies of a workflow and improve the parallelism of the MAS;
- Proposing a geo-aware cost model for GMTA to help agents weigh the overheads of traffic and resources to determine a strategy for task allocation.

The rest of this paper is organized as follows. Section II introduces related work and the improvements offered by GMTA. Section III presents the problem statement and derives the related formulas and models. Section IV introduces the MAS-based geo-aware workflow allocation mechanism and related algorithms. In Section V, an evaluation of the performance of GMTA on the CloudSim platform is reported. Section VI concludes the paper by summarizing the features of GMTA and providing an outlook on future work.

II. Related Work

There have been many works that have attempted to efficiently schedule scientific workflows in the cloud [17]–[19].

Some of the proposed methods are static algorithms that create a scheduling plan before any workflow tasks are run. HCOC is an example that attempts to maintain an execution time that is lower than a given deadline constraint to optimize monetary execution costs [20]. Other examples include G. Jun's work [21], FTWS [22] and IC-PCP [23]. The main disadvantage of these approaches is that they are susceptible to execution delays and cannot adapt to changeable environment.

As an alternative approach, dynamic algorithms have also been developed. One example is J. Sahni's proposal of a dynamic, cost-effective, time-limited heuristic algorithm for scheduling scientific workflows in a public cloud [24]. Other implementations include particle swarm optimization (PSO) [25] and the methods developed by Poola *et al.* [26] and Stavrinides and Karatza [27].

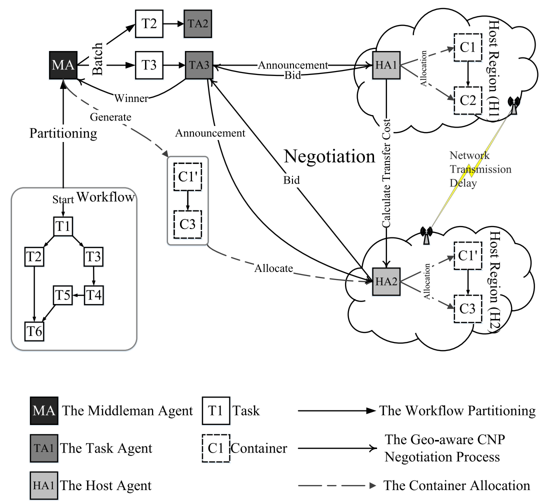


Fig. 1. GMTA Workflow Allocation.

Moreover, the allocation of workflows in a distributed cloud also requires the consideration of geographical characteristics. The traffic between geodistributed hosts can affect the performance of a workflow allocation approach.

Li *et al.* proposed an algorithm that minimizes the traffic between DCs by predicting jobs' makespans. Their proposal can reduce the traffic between DCs by 55% by aggregating all data to a single DC [28]. Chen *et al.* proposed a stream workflow allocation algorithm based on transformations that minimizes the cost of handling the flows of big data in geographically distributed DCs [29].

However, the above workflow allocation mechanisms are based on VMs, which are heavy and awkward to migrate. Recently, a novel virtualization technology called a container has emerged, which is lighter in weight and offers better schedulability than a VM [30]. Hence, container-based workflow allocation strategies can be more flexible [31]. A novel approach has been developed for the container-based cloud environment in which tasks are reallocated to reduce task delays and improve the concurrency number [32]. Chen has proven that selective task replication can mitigate the impact of time-consuming tasks on workflows' makespans [33].

Although task replication can speed up a workflow, it should be noted that extra task replications also require additional resources. Thus, an effective mechanism is needed to maintain a suitable balance. However, related research is lacking.

After investigating the resource distribution, Sim suggested that the agent-based method can be used to reasonably allocate resources in the cloud [34]. A MAS consists of agents that follow game-theory protocols such as the CNP [35]–[37] and negotiate with each other to achieve consensus [38], [39]. Zhu *et al.* proposed a MAS-based real-time task allocation mechanism called ANGEL, which can add VMs for scheduling to improve schedulability [14]. Other works [40]–[42] have proven that a MAS can help to balance multiple objectives during the cloud resource allocation process.

There are two ways for scientific workflow allocation: one is the static allocation (e.g., FTWS, IC-PCP), another is the dynamic allocation (e.g., heuristic, MAS). Dynamic allocation schedules tasks during workflow execution, hence it can respond to changes in the environment in time. Besides, traditional workflow allocations are based on VMs. This paper introduces a more flexible and scalable virtualization platform, the container. This paper proposes GMTA, which adopts a critical task replication strategy to accelerate workflow in a container-based cloud. Moreover, GMTA adopts a MAS to balance the resource cost of task replication and the makespan of a workflow. However, a traditional MAS cannot be directly applied for scientific workflow scheduling because of the complex dependencies among workflow tasks. Hence, GMTA introduces a middleman agent to manages and partitions workflow for allocation. Finally, GMTA applies an improved geo-aware CNP strategy that helps agents weigh resource costs and remain cost-effective.

III. Problem Formulation and Models

This section introduces the formulas for container-based workflow allocation and the notations and terminology used throughout the paper. For ease of reference, the main notations are summarized in Table I.

A. Problem Statement and Motivation

A scientific workflow is a set of tasks that cooperate to complete a scientific computing objective. Because the amount of computing required is large, such a workflow demands multiple hosts in the cloud to cooperate to support it. Thus, an efficient mechanism is required to reasonably allocate the tasks of a workflow and guarantee its makespan. Moreover, the distributed execution of workflows can cause significant data transfer delays, especially in the case of different cloud regions or geodistributed hosts. Excessive delays due to data transmission among geo-dispersed entities can result in not only high traffic costs but also bottlenecks in workflow performance.

For the container-based cloud environment, a novel approach has been proposed in which local surplus resources are used to replicate container instances of critical tasks to eliminate such bottlenecks. However, blindly generating such replications will result in an unnecessary waste of resources and additional traffic costs. Therefore, an efficient mechanism is required to balance the costs of geo-dispersed traffic and new container instances.

Based on the MAS concept, this paper proposes GMTA, which aims to allocate workflows in a container-based cloud efficiently. In accordance with the status of the real-time environment, the agents in GMTA adjust their workflow allocation strategies over time. Moreover, based on a geo-aware cost model, GMTA suitably weighs the two types of costs and reasonably replicates tasks to maximize resource utilization while achieving the optimal makespan.

B. Workflow Allocation Overview

This section mathematically describes the common goal of workflow allocation.

TABLE I
Definitions of Main Notations

Notation	Definition
DC_i	The i^{th} DC in the set $DCRs$
H_i	The i^{th} host in the set $Hosts$
BW_{vu}	The bandwidth between hosts H_v and H_u
$MIPS_v$	The processing power of H_v
WF	The DAG-based workflow description
t_i	The i^{th} task in the set $Tasks$
D_{ij}	The dependency between tasks $t_i \prec t_j$ in the workflow
TD_i	The size of the intermediate data generated by t_i
$TaskLength_i$	The total amount of computation required for task t_i
C_i	The i^{th} container in the container set C
x_{ij}	$x_{ij} = 1$ if C_i is assigned to H_j; otherwise, $x_{ij} = 0$
AT_{iv}	The set of available times at which H_v can provide sufficient resources for C_i
M_v	The amount of resources available on H_v
R_i	The resource demand of C_i
$HR_v(t)$	The function describing the resource usage over time of H_v
$TaskLength_{C_i}$	The total amount of computation to be performed by C_i
s_{iv}	The start time of C_i on H_v
f_{iv}	The completion time of C_i on H_v
e_{iv}	The execution time of C_i on H_v
RT_{iv}^{Host}	The host ready time for C_i on H_v
RT_{iv}^{C}	The task ready time of C_i on H_v
$Cost_p^r$	The cost of transferring data from a remote instance of t_p
$Cost_p^l$	The cost of reallocating a local container instance of t_p

The container-based cloud considered in this paper consists of geodistributed DC regions $DCRs = \{DC_1, DC_2, \ldots DC_u \cdots\}$. In *DCRs*, each DC region consists of many hosts. The host set is $Hosts = \{H_1, \ldots, H_v, \ldots, \}$. The following constraint holds:

$$DC_\alpha = \{H_{\alpha 1}, H_{\alpha 2}, \ldots, H_{\alpha i} \cdots \mid H_{\alpha i} \in Hosts\}. \quad (1)$$

Each host belongs to one and only one DC region. $MIPS_v$ denotes the amount of computation that H_v can process per unit time. Moreover, a 2-D matrix BW is defined, where BW_{vu} denotes the bandwidth between hosts H_v and H_u.

In this paper, the workflow structure is described in the form of a directed acyclic graph (DAG). Let *WF* = (*Tasks*, *D*) denote the workflow. Here, $Tasks = \{t_1, t_2, \ldots, t_i, \ldots, t_j \cdots\}$ denotes the set of tasks, with $t_i = (TaskLength_i, TD_i)$ representing the i^{th} task in the workflow, where $TaskLength_i$ denotes the total amount of computation required for task t_i and TD_i is the size of the resultant data generated by t_i. In addition, *D* is a 2-D matrix that describes the dependencies between tasks:

$$D_{ud} = \begin{cases} 1, & t_u \prec t_d; \\ 0, & otherwise. \end{cases} \quad t_u, t_d \in Tasks \quad (2)$$

Here, $t_u \prec t_d$ denotes that task t_d depends on the result of task t_u. Hence, task t_d cannot start before task t_u is completed. When $D_{ud} = 1$, t_u is said to be an ***upstream task*** of t_d, and t_d is said to be a ***downstream task*** of t_u.

In the container-based cloud, tasks are mapped to containers, which run on hosts and perform corresponding tasks. The container is the minimum unit of resource scheduling. Once a container acquires resources, it cannot be preempted.

$C = \{C_1, C_2, \ldots, C_n \cdots\}$ denotes the set of containers. $C_i = a$ denotes that C_i performs task t_a. Any container can perform only one task, but many containers may perform the same task. R_i denotes the resource demand of C_i, which includes the requirements of the task and the overhead of the container itself. The resource demands of containers that perform the same task are the same.

A 2-D matrix x is defined to record the mapping between containers and hosts:

$$x_{iv} = \begin{cases} 1, & C_i \text{ is allocated to } H_V; \\ 0, & otherwise. \end{cases} \tag{3}$$

Here, $x_{iv} = 1$ denotes that container C_i runs on host H_v. Notably, any container can be allocated to only one host.

$$\sum_{v=1}^{|Hosts|} x_{iv} = 1 \text{ or } 0, \quad i \in [1, |C|]. \tag{4}$$

GMTA aims to find a suitable x with the shortest possible makespan and the lowest possible traffic overhead. Hence, GMTA needs to find appropriate running intervals and suitable hosts for each task in the workflow.

C. Constraints on Workflow Allocation

This section describes the constraints on how tasks can be allocated to hosts.

Suppose that C_i will perform task t_α. If C_i is to run on host H_v, there must be sufficient resources available on H_v during its execution. M_v denotes the total amount of resources on H_v, and R_i is the resource demand of C_i. $HR_v(t)$ is a function describing the resource usage over time on H_v. Accordingly, AT_{iv} denotes the set of times at which H_v has sufficient resources for C_i:

$$AT_{iv} = \{t \in R \mid M_v - HR_v(t) > R_i\}. \tag{5}$$

Moreover, RT_{iv}^{Host} denotes a time interval during which H_v can always provide sufficient resources for C_i:

$$RT_{iv}^{Host} = \{\ [t, t + e_{iv}] \mid [t, t + e_{iv}] \subset AT_{iv}\}. \tag{6}$$

RT_{iv}^{Host} is called H_v's ready time for C_i. In the expression above, e_{iv} is the execution time of C_i on H_v, which is

$$e_{iv} = \frac{TaskLength_{C_i}}{MIPS_v}. \tag{7}$$

Here, $TaskLength_{C_i}$ denotes the total amount of computation to be performed by C_i. If C_i performs task t_α, then $TaskLength_{C_i} \approx TaskLength_\alpha$. Note that e_{iv} is an estimated value that will change in accordance with the actual situation.

In addition, because of the nature of workflow dependencies, a task cannot start before its upstream tasks are completed. Moreover, a task must receive its upstream tasks' results through the network. Hence, C_i cannot start until it has received the data from all of the upstream tasks of t_α. Accordingly, RT_{iv}^{C} denotes the time at which C_i on H_v has collected all of the results of t_α's upstream tasks:

$$RT_{iv}^{C} = \left\{ t \in R \;\middle|\; t > \max_{\substack{C_i=\alpha \\ \forall\ t_\beta \in Tasks}} \times \left\{ D_{\beta\alpha} \times \min_{C_j=\beta} \left\{ \underbrace{\left(f_{ju} + \frac{TD_\beta}{BW_{vu}} \right)}_{H_u \in Hosts \text{ and } X_{ju}=1} \right\}\right\}\right\}. \tag{8}$$

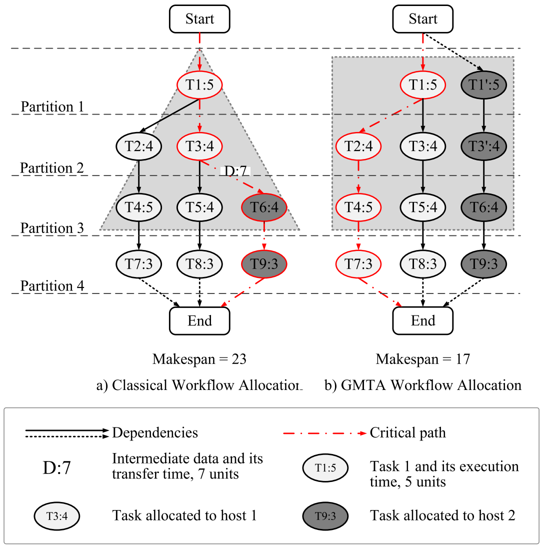


Fig. 2. Two Workflow Allocation Examples.

If t_β is an upstream task of t_α, then $D_{\beta\alpha} = 1$. Notably, there may be many containers on different hosts performing the task t_β. Therefore, C_i will obtain the necessary data from the most convenient of these instances. In this paper, RT_{iv}^{C} is called C_i's ready time on H_v.

f_{ju} denotes the completion time of C_j on H_u. Similarly, the completion time of container C_i on H_v is denoted by f_{iv}.

$$\begin{aligned} f_{iv} &= s_{iv} + e_{iv}; \\ s_{iv} &\in RT_{iv}^{Host} \bigcap RT_{iv}^{C}. \end{aligned} \tag{9}$$

s_{iv} denotes the time at which C_i can start on H_v. In summary, if C_i is to be allocated to H_v, it is necessary to find a time interval $[s_{iv}, f_{iv}] \subset \{RT_{iv}^{Host} \bigcap RT_{iv}^{C}\}$. The corresponding formulation is as follows:

$$iff\ :\ [s_{iv}, f_{iv}] \subset \left\{ RT_{iv}^{Host} \bigcap RT_{iv}^{C} \right\},\ then\ x_{pv} = 1. \tag{10}$$

Hence, it is to solve a linear programming (LP) problem.

D. Geo-Aware Cost Model

When a suitable time interval with sufficient resources is available on a host, the corresponding container instance can be allocated to this host. However, due to resource limitations, tasks will typically be allocated in a distributed manner throughout the geodistributed cloud.

Fig. 2 shows the locations of tasks assigned through two workflow allocation mechanisms. The light-colored tasks are allocated to one host, and the dark-colored tasks are allocated to another. The red line in Fig. 2 represents the critical path of the corresponding workflow allocation.

Suppose that $TC = \{tc_1, \ldots, tc_i, \ldots,\}$ denotes the tasks that constitute the critical path. Et_i is the execution time of task tc_i, and Dt_{ij} denotes the data transmission time between upstream task tc_i and downstream task tc_j. The makespan *WMT* of the workflow is determined by the execution times of the tasks in *TC* and the data transmission times between the tasks in *TC*.

$$\begin{aligned}&If\ \ tc_i,\ tc_j \cdots \in TC,\\ &WMT = Et_i + D_{ij} * Dt_{ij} + Et_j + \cdots\ ,\\ &\quad = \sum_{i}^{tc_i \in TC} (Et_i) + \sum_{i}^{tc_i \in TC} \sum_{j}^{tc_j \in TC} \big(D_{ij} * Dt_{ij}\big). \quad (11)\end{aligned}$$

According to Equation (11), the makespan of the workflow depends on the total execution times and data transmission times of the critical tasks on the critical path. Traditional workflow allocation algorithms can assign an appropriate execution time for each task. For the allocation shown in Fig. 2a), the makespan of the workflow is 30 units of time.

However, when task $T3$ sends its intermediate data to task $T6$ through the network, this data transmission costs 7 units of time. Data transfer between geodistributed hosts presents a challenge for workflow allocation because it increases the makespan of the workflow.

If the upstream and downstream tasks are allocated to the same host, then data can be directly and locally transferred between them, thus reducing the data transmission time. Based on this idea, this paper presents a workflow allocation mechanism called GMTA. As shown in Fig. 2b), GMTA re-executes tasks T_1 and T_3 on host 2 to reduce the intermediate data traffic. GMTA eliminates communication bottlenecks through reasonable task replication, thus accelerating workflows.

However, any newly created container instances occupy extra resources and also need to transfer data from their upstream tasks. It needs to weigh the local task replications reduced overhead and introduced new traffic. Therefore, this paper establishes a geo-aware cost model that quantifies the costs of replicating tasks against the overheads of data transfer.

Consider a task t_α on host H_v. t_α needs to receive data from its upstream task t_p. It is possible to re-execute t_p on H_v by creating a **local replication** of t_p for t_α; in this paper, the corresponding overhead, $Cost_p^l$, is called the local overhead of t_p. By contrast, $Cost_p^r$ denotes the overhead of transferring data from a remote instance of t_p. Since the network delay within a DC is much smaller than that between geodistributed DC regions, the network delay within a DC is neglected in this paper. By comparing $Cost_p^r$ and $Cost_p^l$, GMTA can determine the more favorable decision.

Consider a container C_α that is to perform t_α on H_v, $C_\alpha = \alpha$. For all upstream tasks of t_α, the following constraint holds:

$$If\ \ t_p \prec t_\alpha,\ D_{p\alpha} = 1,\ then\ t_p\ is\ t_\alpha' s\ upstream\ task. \quad (12)$$

The calculation process for $Cost_p^r$ and $Cost_p^l$ is as follows:

1) Determine whether a local instance of t_p exists.
If $x_{pu} = 1$, and $H_u \in DC_\alpha \wedge H_v \cap H_u \subseteq DC_\alpha$, then the local instance of t_p, C_p, exists.
2) If C_p exists, then C_α can directly obtain the upstream data from this local instance C_p without incurring any Internet overhead. Then, $Cost_p^l = f_{pv}$.
3) If C_p does not exist, then GMTA needs to calculate the remote cost $Cost_p^r$ and re-execution cost $Cost_p^l$ of t_p.
4) The cost $Cost_p^r$ of acquiring t_p's data from the remote DC is calculated as follows:
$$Cost_p^r = \min_{\substack{\forall H_u \in Hosts\\ x_{pu}=1}} \left\{ f_{pu} + \frac{TD_p}{BW_{uv}} \right\}. \quad (13)$$
5) $Cost_p^l$ is the time required to collect data from the upstream tasks of t_p on H_v plus the time required to execute t_p on H_v. It is evident that $Cost_p^l = f_{pv}$, which is obtained by solving Equation (14):
$$\begin{cases} s_{pv} = \min\big\{\ RT_{pv}^{Host} \bigcap RT_{pv}^{C}\ \big\};\\ f_{pv} = s_{pv} + e_{pv};\\ t \in \big[s_{pv},\ f_{pv}\big];\\ Cost_p^l = f_{pv}. \end{cases} \quad (14)$$
6) Step 5 is a recursive process that can continue all the way back to the initial task, which has no upstream tasks. However, it is also possible to terminate the recursion in a timely manner in accordance with the actual situation.
7) $Cost_p^r$ and $Cost_p^l$ are compared to determine the more affordable solution.
8) The above process is repeated until all upstream tasks t_p have been considered.

After processing all of the upstream tasks of t_α, t_α obtains the most suitable start time and completion time on H_v.

$$\begin{cases} s_{av} \in RT_{iv}^{Host};\\ s_{av} > \max \underbrace{\Big\{\min\Big\{Cost_p^r, Cost_p^l\Big\}\Big\}}_{\forall t_p\ \prec\ t_\alpha};\\ \min\{f_{av}\} = \min\{s_{av}\} + e_{av}. \end{cases} \quad (15)$$

Notably, the real environment is changeable. Therefore, GMTA adopts a MAS to implement the geo-aware cost model. The agents negotiate with each other and adjust their strategy in accordance with the actual environment to maintain a balance between resource consumption and makespan.

IV. Geo-Aware MAS-Based Workflow Allocation

By replicating critical tasks, GMTA eliminates communication bottlenecks and preserves the parallelism of the workflow. A workflow consists of numerous interdependent tasks; it would be overwhelming to expose the whole workflow to the allocator at the same time. Moreover, because of the influence of the real environment, a static allocation scheme is inappropriate because it cannot deal with unexpected events. For example, as mentioned earlier, the container execution time e_{iv} is simply an estimate, and it is subject to change [43]. Its exact value can only be determined after actual execution. In addition, sometimes hosts may suddenly drop out of the cloud. Once the host resource changes, it may no longer be able to perform pre-assigned tasks.

GMTA allocates tasks as the workflow progresses. Thus, the agents can promptly detect and respond to such unexpected situations. Moreover, GMTA can also detect and eliminate communication bottlenecks in real time.

As shown in Fig. 1, GMTA is separated into three stages:

Stage 1 (Workflow Partitioning): GMTA first divides the workflow into independent partitions to refine the effects of

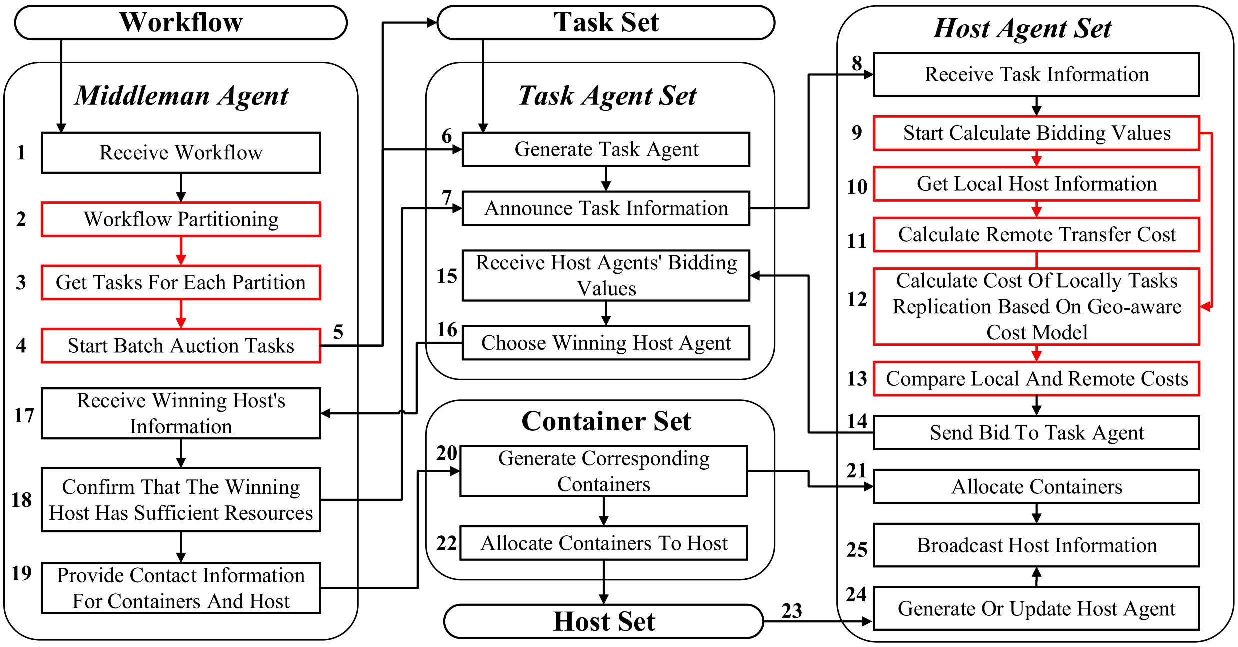


Fig. 3. Basic Interactions Between Agents.

task dependencies. There are no interdependencies between tasks in the same partition. The tasks in one partition depend only on the tasks of previous partitions that have been allocated in the previous auction. Workflow partitioning facilitates the next bidding process.

Stage 2 (Geo-Aware CNP): The agents of GMTA follow CNP to negotiate with each other. Agents announce-bid to allocate tasks. Moreover, agents calculate their bidding values based on the geo-aware cost model. They need to consider the costs of waiting for remote upstream task data and local re-execution. These values are affected by real-time resource usage and network conditions.

Stage 3 (Container Allocation): After the task agents choose the winners of the auctions, GMTA generates and allocates corresponding container instances. If a winning host chooses to reallocate upstream tasks, their corresponding instances are generated at the same time.

A. The Multi-Agents System

The GMTA is based on MAS, where agents calculate their benefit functions and adjust strategies based on the geo-aware cost model. All agents are rational, and they pursue the maximization of benefit. Interaction problems such as cooperation and coordination among agents can be modeled as strategy games where for any finite-strategy game, there is at least a mixed Nash equilibrium. Nash equilibrium is a special strategy profile. Under this profile, any agent unilaterally changing its strategy cannot increase its benefits. When an agent's behavior is inconsistent, the benefit function will force it to switch to the appropriate behavior. Therefore, in the continuous adjustment, the benefits have reached the optimal. In the end, the decisions of all agents form a stable and balanced strategy pattern, and no agent is willing to deviate from this pattern, that is, the Nash Equilibrium. From the stability of the Nash equilibrium, once agents' behavior reaches equilibrium, no member can break this balance to obtain higher benefits, thereby ensuring the smooth progress of the task.

The MAS in GMTA consists of three kinds of agents.

1) The middleman agent is "*the King's Hand*" of the workflow allocation. It manages workflow allocation, including receiving the workflow, generating task agents, managing the announcement-bidding process, and allocating containers to hosts. Moreover, the CNP is usually adopted for single independent task allocation. Therefore, the middleman agent need to divide the workflow into single task auctions through workflow partition, as the red part on the left of Fig. 3.

2) The task agents are responsible for the auctioning of corresponding tasks. A task agent announces the information and requirements of a task, including the task ID, the task length, and the size of the intermediate data, to all host agents. Then, it collects all bids and chooses a winner.

3) The host agents are generated in correspondence with the hosts and synchronize information with their corresponding hosts in real time. Moreover, the host agents continuously communicate and exchange host information with each other. After receiving auction information from a task agent, the host agents calculate their bids in accordance with the real-time status of their corresponding hosts and the geo-aware cost model, as the red part on the right of Fig. 3. Then, the host agents communicate their bids to the corresponding task agent.

The next section detail the interaction between agents.

B. CNP-Based Interactions

Fig. 3 displays the three stages of GMTA and the interactions between the three kinds of agents.

1) The Workflow Partition:

(1-2) The middleman agent manages the entire workflow allocation process. After receiving a workflow, it first

Algorithm 1 Algorithm for the Middleman Agent

Input: Workflow information *wf*
Output: $Containers\_Location\_Map$

```
1:  Initialize the workflow as an adjacency matrix D
2:  //Partition the workflow D into a partition array Partitions
3:  Partitions ← WORKFLOWPARTITION(D)
4:  // Announce the tasks in each partition in a batch
5:  for PartitionArray in Partitions do
6:      for task_i in PartitionArray do
7:          Generate a task agent for task_i and announce task_i
8:      end for
9:  end for
10: (H_winner, Tasks) ← Obtain the winning bidder and the allocation plan for task_i
11: Double-Check(H_winner, Tasks)
12: //Allocate Tasks to H_winner
13: Generate corresponding containers Containers for Tasks
14: Allocate Containers → H_winner
15: Containers_Location_Map ← (Containers, H_winner)
```

divides the workflow into partitions to refine the dependencies between tasks.

(3-4) Then, the middleman agent determines the tasks in each partition and auctions those tasks in a batch.

2) Geo-aware CNP

(5-6) To auction tasks, the middleman agent generates corresponding task agents in the MAS and sends information to the cloud to prepare the containers' environment.

(7) The task agents are responsible for the process of auctioning corresponding tasks. When a task agent is generated, it announces the corresponding task's information and requirements, such as the task ID, the task length, the size of the intermediate data, and the tasks' dependencies, to all host agents.

(8-9) The host agent is generated with the host and synchronizes the hosts' real-time information with other agents. When host agents receive announcement information, they start to calculate their bidding value for the task.

(10-13) GMTA has a strategy that re-execute the upstream tasks locally to reduce network transmission. All of the upstream tasks of the current tasks have already been allocated. Hence, the host agents need to consider data transmission only from already-allocated tasks. Through the geo-aware cost model, the host agents weigh the benefit of the critical task replication strategy. The host agent needs to calculate the cost of transferring data from the remote upstream task instance and the cost of replicating critical tasks locally. The calculation process follows Equation (12)–(16). Then, host agents balance the cost and makespan time and make a decision.

(14) The host agents send their bidding values to the corresponding task agent.

(15-16) Each task agent collects all bids from the host agents and chooses a winner. Then, the task agents send the winning host's information to the middleman agent.

(17-18) The middleman agent receives the auction results. However, before allocating containers to a winning host, the middleman agent confirms that the winning host has sufficient resources. If it does not, the task will be auctioned again.

3) Container Allocation

(19-20) If the winner has sufficient resources, the middleman agent will create a contract for the host and related containers. Then, the contract is sent to the cloud platform.

(21-22) After receiving the contract from the middleman agent, the cloud platform will allocate containers on the corresponding host.

(23-25) After the allocation of the newly added containers, the corresponding host agent will update the relevant information and broadcast the latest status of the host.

The agents communicate as described above and generate a reasonable allocation solution for each workflow task. Next, the details of the behavior of each agent will be introduced.

C. *The Middleman Agent and Workflow Partitioning*

Algorithm 1 presents the pseudocode for the middleman agent.

When it receives a workflow, the middleman agent partitions the workflow through *WorkflowPartition*. On line 4-9, the middleman agent generates corresponding task agents to announce the tasks in each partition in a batch. The middleman agent takes each task array from *Partitions* in turn and generates all corresponding task agents simultaneously for all tasks in the current partition. These task agents are responsible for announcing task information and receiving the bids. Subsequently, on line 10, the middleman agent receives the winning bidders' information. On line 11, the middleman agent performs a double-check to ensure that no host has been assigned tasks that cannot be completed. Although tasks are announced in batches, the middleman agent allocates tasks one by one. Then, on lines 12-14, the middleman agent generates and allocates containers for corresponding tasks to the winning host. Finally, the middleman agent records the containers' locations and shares them with the other agents.

Algorithm 2 shows the pseudocode for the workflow partitioning function. Workflow partitioning is performed on the basis of topological sorting, which is a standard graph theory algorithm. First, the middleman agent transcribes the workflow *wf* into the DAG adjacency matrix *D*. Then, *WorkflowPartition* pushes tasks that do not depend on other tasks into an array *partition*. Subsequently, these tasks and their edges are removed from *D*, and *partition* is pushed into *Partitions*. This process is repeated until there are no tasks remaining in *D*. Finally, *WorkflowPartition* returns a two-dimensional array *Partitions*, in which each item is an array of tasks corresponding to a partition.

Algorithm 2 Algorithm for Workflow Partitioning

Input: Workflow information adjacency matrix D
Output: Partition array *Partitions*
1: Initialize the partition array *Partitions*
2: //Topologically sort D into the partition array *Partitions*
3: **while** $D \neq \varnothing$ **do**
4: Initialize the array *partition*
5: **for** *task* in D **do**
6: **if** *task* does not depend on other tasks **then**
7: $partition.append(task)$
8: **end if**
9: **end for**
10: Remove all tasks $\in partition$ and their edges from D
11: $Partitions.append(partition)$
12: **end while**

Algorithm 3 Algorithm for a Task Agent

Input: Task information t_i
Output: Winning bidder H_{winner}
1: Initialize $valueList \leftarrow \varnothing$
2: Obtain the list of host agents HA
3: **for** $HA_j \in HA$ **do**
4: Send t_i's announcement information to HA_j
5: $b_{ij} \leftarrow$ Obtain the bidding value from host agent HA_j
6: $valueList.append\big(b_{ij}\big)$
7: **end for**
8: **if** $valueList \neq \varnothing$ **then**
9: Select the winning bidder H_{winner} from $valueList$
10: Send the information on H_{winner} and the tasks that will be allocated to middleman agent $\rightarrow H_{winner}, Tasks$
11: **end if**

Algorithm 4 Algorithm for a Host Agent

Input: Task t_i announced by AT_i
Output: Bidding value b_{ij} of HA_j
1: Initialize $HR_v(t)\_List$
2: $HR_v(t)\_List$ is list of two-dimensional array
3: $HR_v(t)\_List[x] = [t_s, t_e, R_O]$ represents the resource occupation R_O in time interval $[t_s, t_e]$
4: //Calculate the ready time r_{ij} of t_i
5: $r_{ij}, Tasks \leftarrow$ GETREADYTIME(t_i)
6: $s_{ij} \leftarrow r_{ij}$
7: $\alpha = \beta = 0$
8: **while** $\alpha, \beta < length(HR_v(t)\_List)$ **do**
9: $t_s \leftarrow HR_v(t)\_List[\alpha]$
10: $t_e \leftarrow HR_v(t)\_List[\beta]$
11: **if** $s_{ij} < t_s[0]$ **then**
12: $s_{ij} \leftarrow t_s[0] + 1$
13: **else**
14: **if** $t_e[1] < s_{ij}$ **then**
15: $\alpha++; \beta++$
16: **else**
17: **if** $t_e[2] + R_{t_i} < M_v$ **then**
18: **if** $\big(s_{ij} + e_{ij}\big) < t_e[1]$ **then**
19: **break**;
20: **else**
21: $\beta++$
22: **end if**
23: **else**
24: $\alpha = ++\beta$
25: **end if**
26: **end if**
27: **end if**
28: **end while**
29: Send $(b_{ij} = f_{ij} = s_{ij} + e_{ij},\ Tasks)$ to TA_i

D. The Task Agents

Algorithm 3 presents the pseudocode for the task agents. Task agent TA_i is responsible for the auctioning of task t_i. As shown on line 4, the task agent sends the announcement information about task t_i to all host agents and waits for their bidding values. After collecting all bids, agent TA_i chooses the lowest bidder as the winner of the auction. Then, TA_i sends the middleman agent the information on the auction winner H_{winner} and the tasks that H_{winner} wants to reallocate.

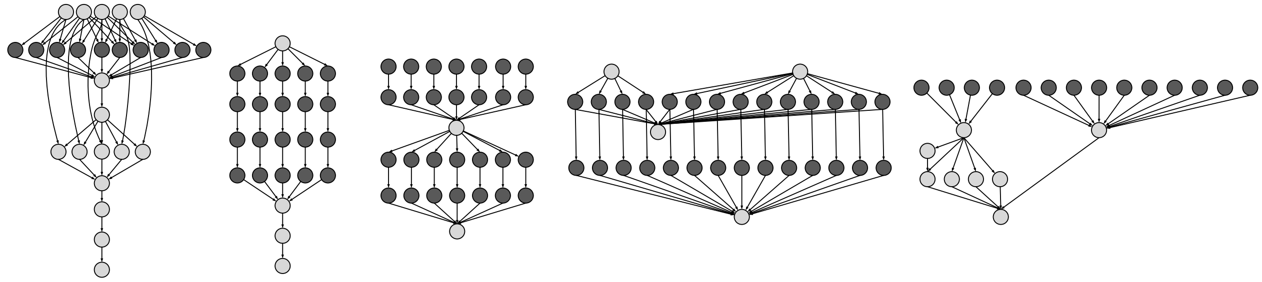
E. The Host Agent

Algorithm 4 presents the pseudocode for the host agents. After receiving the information of task t_i from task agent TA_i, host agent HA_j of host H_j will calculate a bidding value b_{ij} based on the geo-aware cost model.

A two-dimensional array $HR(t)\_List$ records the resource occupation of H_j in real time. As shown on line 5, based on the geo-aware cost model, the host agents determine their most economical allocation plans. Using Equation (15), HA_j calculates the time r_{ij} by which the data from all upstream tasks of t_i can be collected on H_j and the $Tasks$ that HA_j needs to reallocate. Based on a sliding window algorithm, HA_j traverses $HR_v(t)\_List$ starting at r_{ij} to find an appropriate execution window in which t_i can obtain sufficient resources. As shown on line 29, HA_j sends the earliest $f_{ij} = \min\{s_{ij}\} + e_{ij}$ to TA_i as its bid b_{ij}.

Because of task dependencies, each task in the workflow must collect the results of all of its upstream tasks before starting to run. Algorithm 5 specifies how to calculate the ready time r_{ij} of task t_i, i.e., the time at which t_i can be executed on H_j. As shown in Equation (16), GMTA relies on a geo-aware resource consumption model. Based on the geo-aware cost model, the host agent, HA_j, chooses the most appropriate way to handle t_i's upstream tasks. As shown on line 2, HA_j processes all of the upstream tasks of t_i in a loop. As shown on line 4, HA_j calculates the cost of locally re-executing upstream task t_p.

This is a recursive process. To calculate local re-execution costs, host agents sometimes need to work back to the start task of the workflow. In this recursive process, the host agent needs to comprehensively consider how many layers of upstream tasks need to be re-executed based on the cost. On lines 7-15, HA_j determines whether a local container instance of t_p already exists and calculates the minimum cost of transferring data from a remote instance. Finally, HA_j compares $Cost_p^r$ and $Cost_p^l$, and if reallocation is more



a) Montage **b) Epigenomics** **c) LIGO** **d) CyberShake** **e) SIPHT**

Fig. 4. Topologies of Scientific Workflows

Algorithm 5 Determine the Ready Time r_{ij} of t_i on H_j

Input: Task t_i
Output: Ready time r_{ij} of t_i on H_j and $Tasks$

Initialize r_{ij} and $Tasks$
for $t_p \in t_i$'s upstream tasks **do**
  Initialize the parameters $Cost_p^l$ and $Cost_p^r$
  $Cost_p^l \leftarrow$ Obtain the cost of locally reallocating t_p
  Retrieve information on the locations of t_p's containers
  $TaskLocation_p \leftarrow Containers\_Location\_Map[t_p]$
  for $H_t \in TaskLocation_p$ **do**
    if H_j and H_t are local && $Cost_p^l \geq f_{pt}$ **then**
      $Cost_p^l \leftarrow f_{pt}$
      $LocalFlag = True$
      break
    end if
    //Determine the cost of collecting data from H_t
    $Cost_p^r \leftarrow \min\left\{Cost_p^r,\ f_{pt} + \frac{TD_p}{BW_{tj}}\right\}$
  end for
  if $LocalFlag \neq True$ && $Cost_p^r > Cost_p^l$ **then**
    Locally reallocate t_p and update $HR_v(t)\_List$
    $Tasks$.append(t_p)
  end if
  $r_{ij} \leftarrow \max\left\{r_{ij}, \min\left\{Cost_p^r, Cost_p^l\right\}\right\}$
end for
return r_{ij}, $Tasks$

affordable, HA_j will append a replication of t_p to the $Tasks$ list. After processing all upstream tasks, HA_j obtains r_{ij}.

After receiving the bids from all host agents, the task agent will select the winner and report it to the middleman agent. The middleman agent will then notify the winning host agent to report its current real-time status. Because the above calculations performed by the host agents are estimates, it is possible that the winner may not, in fact, have sufficient resources to perform the task as the environment changes. After the middleman agent confirms that there are sufficient resources available on the winning host, it will allocate the corresponding container instances for the *Tasks* set submitted with the bid. After the allocation of *Containers*, H_{winner} will update its host information and broadcast it to the other agents.

The agents synchronize information with each other, allowing them to handle unexpected events promptly. The agents cooperate and negotiate with each other to allocate all tasks of a workflow in turn.

TABLE II
THE CLOUDSIM TESTBED

#DCs	*#hosts*	CPU/(MIPS)	Bandwidth	Task length/(MI)
8	16-40	100-300	50-500	10000-30000

V. PERFORMANCE EVALUATION

A. Simulation Testbed Setup

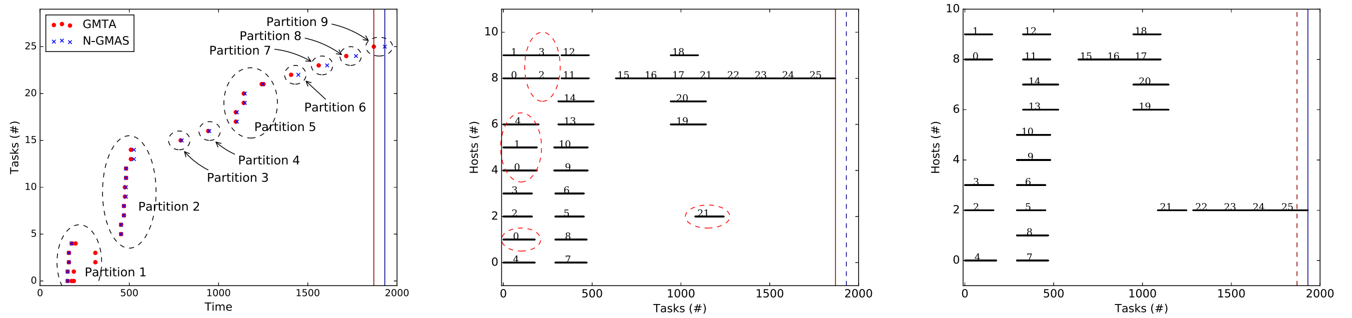
This section presents an evaluation of the performance of GMTA. To ensure the repeatability of the experiments, CloudSim was chosen to simulate the testbed, which consisted of 8 DCs, each consisting of 2-5 hosts. The agents were built in Python. To simulate a workflow, a matrix was constructed to record the dependencies between tasks. The Python-built agents communicated with each other to control the process of workflow allocation based on the real-time state of the testbed and the dependency matrix.

In Fig. 4, five real scientific workflows are depicted: *Montage* [44], *Epigenomic* [45], *LIGO* [46], *CyberShake* [47], and *SIPHT* [48]. These workflows were employed to evaluate GMTA. The workflows were assumed to arrive after system initialization. Notably, the host agents can then request additional containers.

The characteristics of the simulation environment are specified in Table II.

B. Baseline Algorithms

To highlight the contributions of GMTA, three baseline workflow allocation mechanisms were chosen for comparison: PSO [25], ANGEL [14], and N-GMAS. PSO is a classic heuristic algorithm that can quickly find a feasible solution. N-GMAS is similar to GMTA except that the task replication mechanism is removed to highlight the contribution of this mechanism. ANGEL is an effective MAS-based dynamic task allocation mechanism. For this evaluation, the proposed partitioning process was incorporated into the ANGEL mechanism to adapt it for workflow allocation. ANGEL has a bidirectional announcement–bidding mechanism. Moreover, it includes a strategy for creating a new VM for a task whose auction has failed when surplus resources are available to achieve



(a) The Finish Time of tasks (b) The Trace of Workflow Tasks(GMTA) (c) The Trace of Workflow Tasks(N-GMAS)

Fig. 5. Tasks Trace of Montage on CloudSim.

improved schedulability and flexibility. However, in this evaluation, the number of VMs was fixed; therefore, ANGEL was allowed to borrow VMs from the same host region to execute failed tasks. The differences among the four algorithms are shown in Table III.

TABLE III
DIFFERENCES BETWEEN GMTA AND THE THREE BASELINES

	GMTA	**N-GMAS**	**ANGEL**	**PSO**
Virtualization	Container	Container	VM	Container
Protocol	G-CNP[1]	G-CNP[1]	Bi-CNP[2]	Heuristic
Partitioning	Yes	Yes	Yes	No
Replication of Tasks	Yes	No	No	No
When Auction Fails	Re-auction	Re-auction	*	—

[1] *The CNP based on the geo-aware cost model.*
[2] *The CNP based on a bidirectional announcement–bidding process.*

C. Algorithm Objectives

To evaluate the effectiveness of the GMTA algorithm, three optimization indicators are adopted, as introduced below.

1) *Workflow Makespan Time (WMT)* Minimizing the makespan is the goal of all workflow allocation algorithms; GMTA is no exception. WMT is defined as the completion time of the last completed task of all tasks in the workflow:

$$WMT = \max_{\substack{i\in[1,|C|] \\ v\in[1,|Hosts|]}} \left\{ f_{iv} \times x_{iv} \right\}. \tag{16}$$

2) *Network Traffic (NWT)* Because of data dependencies, a container typically must collect data from upstream tasks. If the container instances of the upstream and downstream tasks are not allocated to the same host, data will need to be transferred over the network. NWT represents the cross-DC traffic, which can affect workflow performance.

$$NWT = \sum_{t=1}^{|Tasks|} \left(TD_t \times \underbrace{\sum_{j}^{|C|} \sum_{u}^{|Hosts|} D_{t|C_j|} \times x_{ju}}_{\substack{iff \ \nexists \ C_i = t \ \wedge \ x_{iu}=1 \ \wedge \\ H_v \cap H_u \subseteq DC_\alpha}} \right). \tag{17}$$

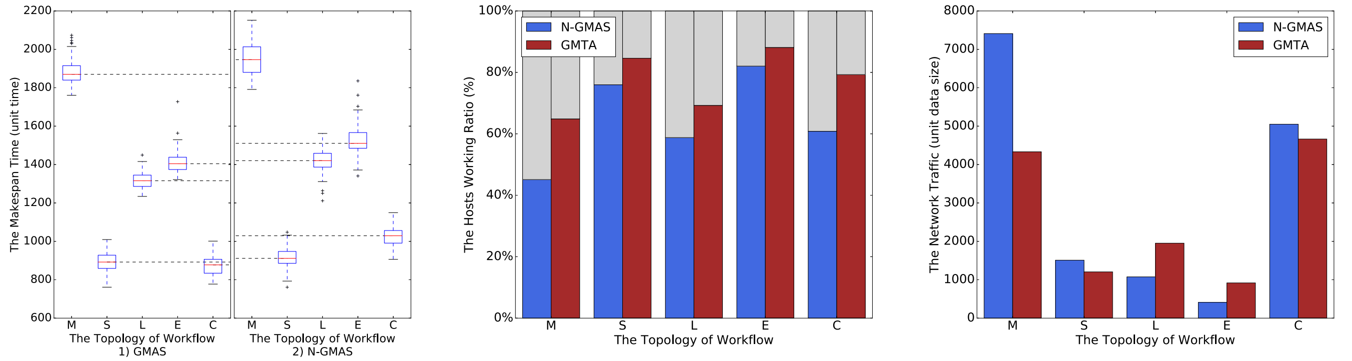
3) *Host Working Ratio (HWR)* According to practical experience, inefficient workflow allocation can cause task processes to be blocked and need to wait for data transfer. Such idle computing capacity is not cost effective. GMTA partitions workflows into a parallel form by replicating tasks, thereby increasing the proportion of working time, denoted by *HWR*, and reducing the effect of network latency.

$$HWR = \frac{\sum_{v=1}^{|Hosts|} \sum_{i=1}^{|C|} (x_{iv} \times (f_{iv} - s_{iv}))}{\sum_{v=1}^{|Hosts|} \left(\max\limits_{i\in[1,|C|]} \{x_{iv} \times f_{iv}\} \right)}. \tag{18}$$

D. Performance Traces on CloudSim

To preliminarily evaluate the practicality and efficiency of GMTA, this section presents the traces of the tasks for the Montage workflow on CloudSim. Montage is a scientific workflow that is used to process astronomical data. In this example, the Montage workflow was allocated using GMTA, which applies a geo-aware optimization strategy, and N-GMAS, which applies a traditional MAS-based strategy.

In Fig. 5, the traces of the Montage workflow tasks as allocated using these two strategies are shown. In Fig. 5a, the red dots and blue crosses represent the completion times of the tasks. The red line represents the *WMT* for GMTA, and the blue line represents the *WMT* for N-GMAS. Moreover, each dashed ellipse corresponds to a partition. The Montage workflow was batch allocated, partition by partition. Each task in a partition needs to receive data from its upstream tasks in the previous partition. However, there is no task dependency within a partition. When GMTA is used, local replications of upstream tasks are generated for some tasks. Consequently, as shown in Fig. 5a, some tasks are performed twice, possibly at different times. These task replications gradually accelerate the workflow as the partitions progress. As a result, the *WMT* of GMTA is earlier than that of N-GMAS. As shown in Fig. 5b and 5c, with the GMTA allocation, the hosts idle less and have a higher *HWR* because lower costs are incurred



(a) WMT (b) HWR (c) NWT

Fig. 6. Performance Impact of Different Workflow Topologies *(M = Montage, S = SIPHT, L = LIGO, E = Epigenomic, C = CyberShake).*

by creating local replications from which to collect upstream data. Without such geo-aware optimization, the network factor has a more significant impact. With GMTA, hosts that have greater computing capabilities have more spare capacity to perform additional tasks. Tasks are more likely to be aggregated on these hosts, thereby reducing transmission overheads. Consequently, the *NWT* metric of GMTA is also lower.

E. Performance Impacts of Topologies and Task Replication

Fig. 6 shows the *WMT*, *HWR*, and *NWT* metrics for the five scientific workflows depicted in Fig. 4. This set of experiments was conducted to evaluate the optimization effects of GMTA's replication strategy for different workflow topologies. GMTA can optimize workflow performance by replicating tasks and refining task dependencies. However, the optimization gains are different for different workflow topologies. For example, although the number of tasks is similar in each of these five workflows, the optimization gains in terms of *WMT* are different, ranging from 3% to 20%.

The topologies of the workflows affect GMTA's performance. Fig. 2a shows an example of a **triangle** topology, in which the tasks and their dependency connections form a triangle. In a triangle topology, a task nearer the top of the triangle has fewer upstream tasks than downstream tasks, so it can easily become a bottleneck during execution. When processing workflows, GMTA can perceive such triangle bottlenecks and eliminate them by replicating upstream tasks. After task replication, the task itself and its replications, along with its downstream tasks and their connections, form a **rectangle**, as shown in Fig. 2b. In such a topology, the downstream tasks can obtain data faster and more easily from nearby replications. In this way, GMTA eliminates triangle bottlenecks. As shown in Fig. 6, GMTA effectively optimizes the Montage, Epigenomic, and CyberShake workflows in terms of *WMT*, *HWR*, and *NWT*. However, some tasks and dependencies form an **inverted triangle**. This occurs when a task depends on multiple upstream nodes. Due to resource constraints, it is usually impossible to locally allocate all of these upstream tasks. In such situations, GMTA may not achieve ideal results.

In summary, the more triangular dependencies there are in a workflow, the better the optimization that GMTA can achieve. For example, for the Montage workflow, its *WMT* is significantly improved. The ratio *HWR* is also higher because GMTA improves the parallelism of such workflows.

F. Performance Impacts of Network and Hosts Performance

This section examines the performance of GMTA in different network environments and under different host performance conditions. The five scientific workflows were allocated using GMTA and the three selected baseline workflow allocation mechanisms, PSO, ANGEL, and N-GMAS. Table III summarizes the differences among these four mechanisms. N-GMAS follows the traditional MAS rule, whereas GMTA and ANGEL implement more flexible resource scheduling mechanisms based on virtualization.

In this section, the concurrency number (CN) is used to represent the performance of the hosts because the more resources a host has, the more tasks it can execute in parallel. For each CN value, multiple values of network latency were simulated in CloudSim to analyze the *WMT*, *HWR*, and *NWT* metrics of the allocation results for the five scientific workflow under different network and host performance conditions. Fig. 7 displays the curves of *WMT*, *HWR*, and *NWT* for different CN values as functions of network latency. GMTA significantly decreases *WMT* for all five scientific workflows.

As Fig. 7, the fluctuation of PSO is higher than the others. Although all the four mechanisms are dynamic, the heuristic method takes time to iterate and converge, and can not respond to environmental changes as fast as GMTA. GMTA is based on agents that can sense changes in the environment more quickly and respond accordingly. Hence, the GMTA allocated workflow is smoother than PSO.

As shown in the left column of Fig. 7, the *WMT* values of GMTA are better than those of the baselines under the same conditions. GMTA's scheduling strategy can reduce the impact of the network latency on *WMT*. The higher the CN value is, the better the effect of GMTA. However, if the CN is too high, for example, when CN is 4 for Epigenomic (Fig. 7j, Fig. 7k and Fig. 7l), there is little room for optimization with GMTA. In this scenario, the entire workflow can be executed in parallel on a single host, and optimization strategies are useless. However, a real environment will always have resource limitations. The tasks of a workflow will be scattered

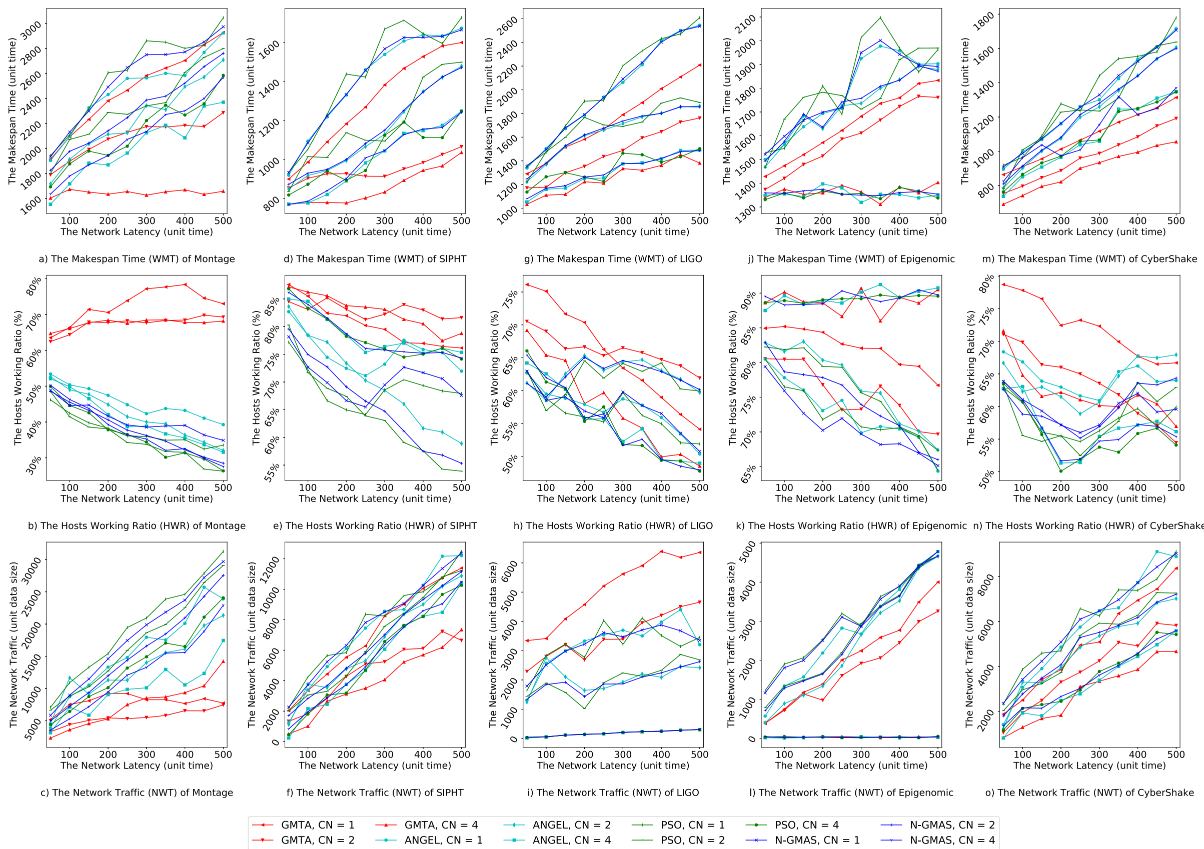


Fig. 7. Performance Impacts of Network and Hosts Performance.

throughout the cloud, and the *WMT* of the workflow will depend on the network latency. Thus, GMTA can use nearby idle resources to allocate local task replications and decrease *WMT* for workflows.

As mentioned in the previous section, the more regular triangular dependencies appear in a workflow, the better the optimization that GMTA achieves. However, the inverted triangle topology also affects the optimization performance of GMTA. For example, there are substantial numbers of inverted triangles in the CyberShake and SIPHT topologies. GMTA cannot eliminate the bottlenecks represented by these inverted triangles by replicating tasks. As shown in the second and fifth rows of Fig. 7, *WMT*, *HWR*, and *NWT* all increase linearly with increasing network latency. The tasks at the bottom of a large inverted triangle need to collect upstream data from tasks that are distributed in the cloud. Such network transmission is necessary and unavoidable. However, GMTA optimizes at least what it can, and there are also some regular triangles in these two topologies. By adjusting these parts of a workflow topology, GMTA can still achieve some optimization. Consequently, the GMTA results are still better than those of the baselines.

Meanwhile, there are many triangles in the Montage topology, on which GMTA works best. For example, when CN is two and the network latency is higher than 200, the network delay has little effect on the Montage workflow as allocated by GMTA. There are many opportunities for GMTA to use local surplus resources to replicate tasks to change triangle bottlenecks into rectangles. These local replications refine the dependencies in the workflow and increase its parallelism.

As shown in Fig. 7b, the *HWR* values of the baselines gradually decrease as the network latency increases. By contrast, after GMTA optimization, *HWR* can even increase as the network conditions deteriorate. There are two reasons for this behavior. On the one hand, GMTA generates some task replications that occupy additional run time on hosts. On the other hand, more importantly, the generated local upstream task replications reduce the wait times of downstream tasks and the idle times of hosts. As shown in Fig. 7c, the *NWT* values of the baselines also increase linearly as the network latency increases. This causes increasing transmission delay between tasks, which increases the overall workflow overhead. However, the network delay does not affect the GMTA-allocated Montage workflow because once the network delay becomes too significant, in accordance with the geo-aware model, GMTA will tend to create local task replications, allowing data to be directly transferred locally. This decreases the effects of network latency.

Notably, GMTA is most concerned with *WMT*. Sometimes, to decrease *WMT*, GMTA may generate more traffic. As shown in Fig. 4c, there is only one node in the third line of the LIGO topology, which is also the vertex of a triangle topology and the lower point of an inverted triangle. Replicating this task can reduce the traffic between it and its downstream tasks. At the same time, a replication of this task will need to receive data from all of its corresponding upstream tasks, which will generate more data traffic. In this case, this additional traffic is worthwhile; however, managing such decisions requires a careful strategy. As shown in Fig. 7g, GMTA ultimately reduces the *WMT* of LIGO.

On the one hand, increasing the amount of data traffic does not necessarily cause *WMT* to increase because data can be transmitted in parallel. On the other hand, obtaining data from a remote task instance may cost time, and it may be more convenient to instead obtain data from the upstream tasks of that remote task. It may take less time to collect the data from the upstream tasks of the remote task and re-execute the remote task locally than to wait for the remote task to finish and then transmit its data over the network. GMTA evaluates these costs through the geo-aware cost model.

GMTA's agents adopt the geo-aware cost model to evaluate the costs of these two allocation strategies based on the real-time status of the network and hosts. Finally, the middleman agent uses this feedback information to schedule the tasks of the workflow to achieve efficient workflow execution.

VI. Conclusion

This paper proposes a geo-aware multiagent workflow allocation approach, GMTA, which aims to allocate the tasks of scientific workflows in a container-based cloud environment. Based on a MAS, GMTA dynamically allocates tasks during workflow execution, which allows GMTA to sense and respond to changes in the environment in a timely manner. The dependencies among the tasks of a scientific workflow affect its parallelism and slow down its execution. Based on the excellent migration properties of containers, this paper proposes a critical task replication mechanism for refining the dependency structure of scientific workflow and improving the concurrency of its tasks.

GMTA is built on stone. It includes three novel mechanisms for improving scientific workflow allocation: workflow partitioning, task replication, and a geo-aware cost model. The workflow partitioning process simplifies and clarifies the dependencies between workflow tasks by separating the workflow into partitions. There are no dependencies within a partition, thereby facilitating the subsequent allocation process.

However, due to resource constraints, the tasks of a workflow will typically be distributed throughout the cloud. Current tasks may require data from offsite tasks. Hence, the network transmission performance affects the progress of the whole workflow. To address this challenge, GMTA adopts a critical-task replication strategy. Reasonable local replication of an upstream task can reduce the network transmission delay between it and its downstream tasks.

Nevertheless, extra task replications also take up additional host resources. Moreover, the replications of upstream tasks also require the data from the upstream tasks of those replicated upstream tasks. Therefore, although replicating tasks reduces the transmission delay between a replicated task and its downstream tasks, it also introduces a new transmission delay between the replicated task and its upstream tasks. Therefore, this paper proposes a geo-aware cost model and a CNP-based MAS. When allocating tasks, the agents will calculate the cost that would be incurred by replicating upstream tasks in accordance with the actual situation of the environment. Notably, this calculation is a recursive process. The geo-aware cost model will trace back the upstream tasks and find the most reasonable replication scheme. The agents then weigh the reduction in latency achieved with the critical-task replication strategy against the newly introduced latency. Then, the agents decide whether to adopt a task replication strategy based on its cost. Although replicated tasks may indeed introduce more traffic, the replications that are confirmed to be worthwhile by the geo-aware cost model will reduce the overall data transmission delay and, thus, the workflow makespan.

During the workflow allocation process, GMTA identifies bottlenecks in the execution of a scientific workflow and eliminates these bottlenecks by task replications. Based on the geo-aware CNP, the MAS coordinates resource consumption and ensures fast and efficient execution of workflows.

There are still some aspects in which GMTA can be improved. The MAS is based on a market-like mechanism that can comprehensively consider various types of constraints. However, in an actual run-time environment, excessive communication for negotiation between agents will affect efficiency. In addition, the agents will generate additional network traffic to synchronize their information. Further improvements are needed in these areas.

References

[1] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, "Characterizing and profiling scientific workflows," *Future Gener. Comput. Syst.*, vol. 29, no. 3, pp. 682–692, 2013.

[2] R. Buyya, "Cloud computing: The next revolution in information technology," in *Proc. 1st Int. Conf. Parallel Distrib. Grid Comput. (PDGC 2010)*, Solan, India, 2010, pp. 2–3.

[3] M. A. Rodriguez and R. Buyya, "Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds," *IEEE Trans. Cloud Comput.*, vol. 2, no. 2, pp. 222–235, Apr.–Jun. 2014.

[4] A. C. Zhou, B. He, X. Cheng, and C. T. Lau, "A declarative optimization engine for resource provisioning of scientific workflows in geo-distributed clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 3, pp. 647–661, Mar. 2017.

[5] T. A. Genez, L. F. Bittencourt, and E. R. Madeira, "Workflow scheduling for SaaS / PaaS cloud providers considering two SLA levels," in *Proc. IEEE Netw. Oper. Manag. Symp.*, Maui, HI, USA, 2012, pp. 906–912.

[6] M. Zhu, Q. Wu, and Y. Zhao, "A cost-effective scheduling algorithm for scientific workflows in clouds," in *Proc. IEEE 31st Int. Perform. Comput. Commun. Conf. (IPCCC)*, Austin, TX, USA, 2012, pp. 256–265.

[7] *Docker*. Accessed: Nov. 2019. [Online]. Available: https://www.docker.com/

[8] S. Fu, J. Liu, X. Chu, and Y. Hu, "Toward a standard interface for cloud providers: The container as the narrow waist," *IEEE Internet Comput.*, vol. 20, no. 2, pp. 66–71, Mar./Apr. 2016.

[9] L. Li, T. Tang, and W. Chou, "A REST service framework for fine-grained resource management in container-based cloud," in *Proc. IEEE 8th Int. Conf. Cloud Comput.*, New York, NY, USA, 2015, pp. 645–652.

[10] P. Eitschberger and J. Keller, "Fault-tolerant parallel execution of workflows with deadlines," in *Proc. 25th Euromicro Int. Conf. Parallel Distrib. Netw. Based Process. (PDP)*, St. Petersburg, Russia, Mar. 2017, pp. 78–84.

[11] R. Tudoran, A. Costan, and G. Antoniu, "OverFlow: Multi-site aware big data management for scientific workflows on clouds," *IEEE Trans. Cloud Comput.*, vol. 4, no. 1, pp. 76–89, Jan.–Mar. 2016.

[12] K. M. Sim, "Agent-based cloud computing," *IEEE Trans. Services Comput.*, vol. 5, no. 4, pp. 564–577, 4th Quart., 2012.

[13] H. Luo, X.-J. Hu, and X.-X. Hu, "Multi agent negotiation model for distributed task allocation," in *Proc. 2nd IEEE Int. Conf. Inf. Manag. Eng.*, Chengdu, China, 2010, pp. 54–57.

[14] X. Zhu, C. Chen, L. T. Yang, and Y. Xiang, "ANGEL: Agent-based scheduling for real-time tasks in virtualized clouds," *IEEE Trans. Comput.*, vol. 64, no. 12, pp. 3389–3403, Dec. 2015.

[15] K. Bousselmi, Z. Brahmi, and M. M. Gammoudi, "Energy efficient partitioning and scheduling approach for scientific workflows in the cloud," in *Proc. IEEE Int. Conf. Serv. Comput. (SCC)*, San Francisco, CA, USA, Jun. 2016, pp. 146–154.
[16] Z. Wen, R. Qasha, Z. Li, R. Ranjan, P. Watson, and A. Romanovsky, "Dynamically partitioning workflow over federated clouds for optimising the monetary cost and handling run-time failures," *IEEE Trans. Cloud Comput.*, early access, Aug. 26, 2016, doi: 10.1109/TCC.2016.2603477.
[17] X. Zhu, J. Wang, H. Guo, D. Zhu, L. T. Yang, and L. Liu, "Fault-tolerant scheduling for real-time scientific workflows with elastic resource provisioning in virtualized clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 12, pp. 3501–3517, Dec. 2016.
[18] H. M. D. Kabir, A. S. Sabyasachi, A. Khosravi, M. A. Hosen, S. Nahavandi, and R. Buyya, "A cloud bidding framework for deadline constrained jobs," in *Proc. IEEE Int. Conf. Ind. Technol. (ICIT)*, Melbourne, VIC, Australia, Feb. 2019, pp. 765–772.
[19] H. R. Faragardi, M. R. S. Sedghpour, S. Fazliahmadi, T. Fahringer, and N. Rasouli, "GRP-HEFT: A budget-constrained resource provisioning scheme for workflow scheduling in IaaS clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 6, pp. 1239–1254, Jun. 2020.
[20] L. F. Bittencourt and E. R. M. Madeira, "HCOC: A cost optimization algorithm for workflow scheduling in hybrid clouds," *J. Internet Serv. Appl.*, vol. 2, no. 3, pp. 207–227, 2011.
[21] G. Jung and H. Kim, "Optimal time-cost tradeoff of parallel service workflow in federated heterogeneous clouds," in *Proc. IEEE 20th Int. Conf. Web Serv.*, Santa Clara, CA, USA, Jun. 2013, pp. 499–506.
[22] S. Jayadivya, J. S. Nirmala, and M. S. S. Bhanu, "Fault tolerant workflow scheduling based on replication and resubmission of tasks in cloud computing," *Int. J. Comput. Sci. Eng.*, vol. 4, no. 6, pp. 996–1006, 2012.
[23] S. Abrishami, M. Naghibzadeh, and D. H. Epema, "Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds," *Future Gener. Comput. Syst.*, vol. 29, no. 1, pp. 158–169, 2013.
[24] J. Sahni and D. P. Vidyarthi, "A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a cloud environment," *IEEE Trans. Cloud Comput.*, vol. 6, no. 1, pp. 2–18, Jan.–Mar. 2018.
[25] A. Verma and S. Kaushal, "Bi-criteria priority based particle swarm optimization workflow scheduling algorithm for cloud," in *Proc. Recent Adv. Eng. Comput. Sci. (RAECS)*, Chandigarh, India, Mar. 2014, pp. 1–6.
[26] D. Poola, K. Ramamohanarao, and R. Buyya, "Enhancing reliability of workflow execution using task replication and spot instances," *ACM Trans. Auton. Adaptive Syst. (TAAS)*, vol. 10, no. 4, p. 30, 2016.
[27] G. L. Stavrinides and H. D. Karatza, "A cost-effective and QoS-aware approach to scheduling real-time workflow applications in PaaS and SaaS clouds," in *Proc. 3rd Int. Conf. Future Internet Things Cloud*, Rome, Italy, 2015, pp. 231–239.
[28] P. Li *et al.*, "Traffic-aware geo-distributed big data analytics with predictable job completion time," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 6, pp. 1785–1796, Jun. 2017.
[29] W. Chen, I. Paik, and P. C. K. Hung, "Transformation-based streaming workflow allocation on geo-distributed datacenters for streaming big data processing," *IEEE Trans. Services Comput.*, vol. 12, no. 4, pp. 654–668, Jul./Aug. 2019.
[30] D. Bernstein, "Containers and cloud: From LXC to docker to kubernetes," *IEEE Cloud Comput.*, vol. 1, no. 3, pp. 81–84, Sep. 2014.
[31] R. Zhou, Z. Li, and C. Wu, "Scheduling frameworks for cloud container services," *IEEE/ACM Trans. Netw.*, vol. 26, no. 1, pp. 436–450, Feb. 2018.
[32] L. Yin, J. Luo, and H. Luo, "Tasks scheduling and resource allocation in fog computing based on containers for smart manufacturing," *IEEE Trans. Ind. Informat.*, vol. 14, no. 10, pp. 4712–4721, Oct. 2018.
[33] H. Chen, X. Zhu, D. Qiu, L. Liu, and Z. Du, "Scheduling for workflows with security-sensitive intermediate data by selective tasks duplication in clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 9, pp. 2674–2688, Sep. 2017.
[34] K. M. Sim, "Agent-based approaches for intelligent intercloud resource allocation," *IEEE Trans. Cloud Comput.*, vol. 7, no. 2, pp. 442–455, Apr.–Jun. 2019.
[35] H. Liang and F. Kang, "A novel task optimal allocation approach based on contract net protocol for agent-oriented UUV swarm system modeling," *Optik Int. J. Light Electron Opt.*, vol. 127, no. 8, pp. 3928–3933, 2016.
[36] G. D. Jules, M. Saadat, and S. Saeidlou, "Holonic ontology and interaction protocol for manufacturing network organization," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 45, no. 5, pp. 819–830, May 2015.
[37] J. O. Gutierrez-Garcia and K. M. Sim, "Agent-based cloud service composition," *Appl. Intell.*, vol. 38, no. 3, pp. 436–464, 2013.
[38] S. Sikdar, S. Givigi, and K. Rudie, "A resource allocation mechanism using matching and bargaining," *IEEE Trans. Autom. Control*, vol. 62, no. 11, pp. 5909–5914, Nov. 2017.
[39] W. Liu, W. Gu, J. Wang, W. Yu, and X. Xi, "Game theoretic non-cooperative distributed coordination control for multi-microgrids," *IEEE Trans. Smart Grid*, vol. 9, no. 6, pp. 6986–6997, Nov. 2018.
[40] Y. Jiang, "A survey of task allocation and load balancing in distributed systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 2, pp. 585–599, Feb. 2016.
[41] R. Duan, R. Prodan, and X. Li, "Multi-objective game theoretic schedulingof bag-of-tasks workflows on hybrid clouds," *IEEE Trans. Cloud Comput.*, vol. 2, no. 1, pp. 29–42, Jan.–Mar. 2014.
[42] K. Dehghanpour and H. Nehrir, "An agent-based hierarchical bargaining framework for power management of multiple cooperative microgrids," *IEEE Trans. Smart Grid*, vol. 10, no. 1, pp. 514–522, Jan. 2019.
[43] H. Chen, X. Zhu, G. Liu, and W. Pedrycz, "Uncertainty-aware online scheduling for real-time workflows in cloud service environment," *IEEE Trans. Services Comput.*, early access, Aug. 21, 2018, doi: 10.1109/TSC.2018.2866421.
[44] *What is Montage?* Accessed: Nov. 2019. [Online]. Available: http://montage.ipac.caltech.edu/
[45] *Epigenome*. Accessed: Nov. 2019. [Online]. Available: http://epigenome.usc.edu
[46] *Ligo Project, LIGO-Laser Interferometer Gravitational Wave Observatory*. Accessed: Nov. 2019. [Online]. Available: http://www.ligo.caltech.edu/
[47] *Cybershake*. Accessed: Nov. 2019. [Online]. Available: https://scec.usc.edu/scecpedia/CyberShake
[48] J. Livny, H. Teonadi, M. Livny, and M. K. Waldor, "High-throughput, kingdom-wide prediction and annotation of bacterial non-coding RNAs," *PLoS ONE*, vol. 3, no. 9, p. e3197, Sep. 2008.

Meng Niu received the B.S. degree from the Beijing University of Posts and Telecommunications, Beijing, China, in 2015, where he is currently pursuing the Ph.D. degree with the State Key Laboratory of Networking and Switching Technology. His current research interests include network function virtualization, services and resource allocation managements.

Bo Cheng (Member, IEEE) received the Ph.D. degree in computer science from the University of Electronics Science and Technology of China in 2006. He is currently a Professor with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. His research interests include multimedia communications and services computing.

Yimeng Feng received the B.S. degree from the Beijing University of Posts and Telecommunications, Beijing, China, and the Queen Mary University of London, London, U.K., in 2016. She is currently pursuing the dual Ph.D. degree program with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China, and the School of Electrical and Data Engineering, University of Technology Sydney, Ultimo, NSW, Australia. Her current research interests include intelligent transportation systems, Internet of Things, and wireless sensor network technology.

Junliang Chen is a Professor with the Beijing University of Posts and Telecommunications. His research interests are in the area of service creation technology. He was elected as a member of the Chinese Academy of Science in 1991, and the Chinese Academy of Engineering in 1994.