symmetry

MDPI

Article

Research on the Task Assignment Problem with Maximum Benefits in Volunteer Computing Platforms

Ling Xu [1,2,*], Jianzhong Qiao [1], Shukuan Lin [1] and Xiaowei Wang [3]

1 School of Computer Science and Engineering, Northeastern University, Shenyang 110819, China; qiaojianzhong@mail.neu.edu.cn (J.Q.); linshukuan@ise.neu.edu.cn (S.L.)
2 School of Software Engineering, Dalian University of Foreign Languages, Dalian 116044, China
3 School of Vehicle and Mobility, Tsinghua University, Beijing 100084, China; xiaoweiwang@mail.tsinghua.edu.cn
* Correspondence: xuling@dlufl.edu.cn; Tel.: +86-158-4097-5069

Received: 17 April 2020; Accepted: 21 May 2020; Published: 24 May 2020

Abstract: As a type of distributed computing, volunteer computing (VC) has provided unlimited computing capacity at a low cost in recent decades. The architecture of most volunteer computing platforms (VCPs) is a master–worker model, which defines a master–slave relationship. Therefore, VCPs can be considered asymmetric multiprocessing systems (AMSs). As AMSs, VCPs are very promising for providing computing services for users. Users can submit tasks with deadline constraints to the VCPs. If the tasks are completed within their deadlines, VCPs will obtain the benefits. For this application scenario, this paper proposes a new task assignment problem with the maximum benefits in VCPs for the first time. To address the problem, we first proposed a list-based task assignment (LTA) strategy, and we proved that the LTA strategy could complete the task with a deadline constraint as soon as possible. Then, based on the LTA strategy, we proposed a maximum benefit scheduling (MBS) algorithm, which aimed at maximizing the benefits of VCPs. The MBS algorithm determined the acceptable tasks using a pruning strategy. Finally, the experiment results show that our proposed algorithm is more effective than current algorithms in the aspects of benefits, task acceptance rate and task completion rate.

Keywords: volunteer computing; task assignment; deadline constraints; maximum benefit; asymmetric multiprocessing system

1. Introduction

As a type of distributed computing, volunteer computing (VC) is the use of idle computing capacity that comes from volunteer digital devices, such as desktops, laptops and smartphones, for large-scale scientific computing over volunteer computing platforms (VCPs). A well-known open VCP is the Berkeley Open Infrastructure for Network Computing (BOINC) [1], and most VC projects use BOINC, such as SETI@home [2], Climateprediction.net and Einstein@home. There are also VC projects using other VCPs, such as Folding@home [3] and ATLAS@home [4]. Generally speaking, the distributed computing system can be one of two types according to the architecture: a symmetry multiprocessing system (SMS) or a asymmetric multiprocessing system (AMS) [5]. The AMS defines a master–slave relationship, and the master processor schedules tasks to the slave processors. In contrast to AMS, SMS has no master–slave relationship in processors. In fact, the architecture of most volunteer computing platforms (VCPs) is a master–worker model, which defines a master–slave relation in nodes [6]. Therefore, VCPs can be considered an asymmetric multiprocessing system (AMS).

Surveys show that early VC projects such as SETI@home attracted one million volunteers. As there is no effective incentive mechanism at present, the volunteers have shrunk to approximately 200,000 [1]. Therefore, VCPs face the challenges of recruiting and retaining more volunteers. Presently, the performance of volunteer digital devices becomes more advanced. Therefore, the potential computing power of VC is much larger than you intend. Besides, according to the prior works [7,8], for the same computing power, the cost of Amazon cloud computing platforms is 420 times that of VC. Therefore, recruiting and retaining more volunteers to participate in volunteer computing will get huge computing power at a low cost. One method to recruit and retain volunteers in VCPs is to use BOINC credits as a proof-of-work for blockchain systems or other commercially distributed computing systems, and use the benefits issued by these systems, such as tokens, as rewards for participating [1,9]. In this paper, we go one step further and propose an application scenario where VCPs can provide computing services. Users pay an amount of money to use VCPs to compute their tasks. If the VCPs complete a task within its deadline constraint, the VCPs will obtain an amount of money and assign the benefits to volunteers.

On this basis, we propose research on the maximum benefits for VCPs. To solve the maximum benefit problem, we need to select tasks from users to complete and assign tasks to appropriate workers. From the perspective of VCPs, having the goal of a maximum benefit problem leads to an interesting task assignment problem under the assumption that computing services from VCPs can obtain financial benefits. How to assign tasks to appropriate workers has been extensively studied in prior works [10–18]. In these prior works, some studies are static task scheduling algorithms and others are dynamic task scheduling algorithms. Our study belongs to the second category. Specifically, we focused on the following problem: assigning independent tasks under deadline constraints dynamically and maximizing the benefits of VCPs.

Some prior works [19,20] studied dynamic task assignment under deadline constraints, but their task assignment goals were different from our work. At the same time, they did not consider the reliability. Usually, in distributed systems, reliability refers to the probability of executing tasks successfully [21]. In this paper, the definition of reliability can be narrowed down to the degree that VCPs can execute tasks correctly under the disturbance of the worker's actual available time. This is because, in VCPs, the available time of the workers is often disturbed, for instance, by the non-VC application CPU load exceeding a threshold, which may cause the task to exceed its deadline. Therefore, the VCP cannot complete the task correctly and achieve its matching benefit. Thus, designing a task assignment algorithm that is reliable against the disturbance is an important research. Consequently, we also considered the reliability when assigning tasks. Specifically, in this paper, we adopted a list-based task assignment (LTA) strategy to reduce the impact of the disturbance. We proposed a new task assignment algorithm, which aimed at maximizing the benefits of the VCPs under deadline constraints and considering the reliability. The studies most related to our work are the following [22,23].

In the work [22], the authors studied the problem of how to assign tasks when the disturbances of the workers' available times occurred and proposed two algorithms based on the heterogeneous earliest-finish-time (HEFT) [10]. One was the HEFT with availability constraint (HEFT-AC) algorithm, and the other was the HEFT AC with the unavailability (HEFT-ACU) algorithm. The goals of the two algorithms were to minimize the makespan. The HEFT-AC algorithm mainly extended the HEFT and allowed it to apply to the workers' available times with availability constraints. The HEFT-ACU algorithm's basic idea was to reduce the impact of the disturbance by using two reputation parameters. By using the method, the HEFT-ACU algorithm reduced the gap between the expected makespan and the actual one and improved the performance of the scheduling. We noticed that the two reputation parameters could not predict the availability time of a worker. Besides, they assigned the task with shortest execution time to the worker with the best computational ability, which might cause a task with a longer execution time to miss its deadline.

To dynamically assign tasks under deadline constraints and the disturbances coming from the uncertainty of distributed systems, the authors in [23] proposed a new task assignment algorithm called maximum on-time completions (MOC). In the MOC algorithm, they defined the stochastic robustness measure by using the stochastic task completion time. To maximize the number of tasks that meet their individual deadlines, they discarded executing tasks that miss their deadline constraints by considering two conditions: (1) executing tasks that cannot be discarded; (2) executing tasks that can be discarded. However, their goal of task assignment was different from ours.

Motivated by this, in this paper, we extended the use of the reliability model proposed by Xu et al. [24] and proposed a list-based task assignment (LTA) strategy. At the same time, we proved the advantage of LTA strategy in scheduling tasks with deadline constraints and considering the availability time of a worker being interrupted. The rationality of our work was to make full use of the availability time of a worker and to reduce the disturbances in assigning through using an LTA strategy. In this way, the expected maximum benefits from a task would be closer to the actual maximum benefits.

The potential application of our work is cloud computing platforms. As is known to all, cloud computing adopts a resource pool to provide computing services for a fee. Some cloud computing platforms are centralized; i.e., the architecture is a master–worker model. The architecture of our task assignment algorithm is also centralization (master–worker model). Simultaneously, our task assignment goal is to maximize the benefit of the system. If the cloud computing platform adopts our mechanism to schedule the tasks from users, the cloud computing platform can achieve more benefits. Therefore, our task assignment algorithm could be used as a primary scheduling mechanism for cloud computing. In this paper, we addressed the task assignment problem with the maximum benefits in VCPs and proposed an efficient task assignment algorithm that takes into account the reliability and tasks with deadline constraints. To summarize, the main contributions of this paper are as follows:

(1) To the best of our knowledge, the task assignment problem with the greatest benefit in VCPs was proposed for the first time in this paper, and we gave the evaluation criteria for the benefits.
(2) We proposed a new list-based task assignment (LTA) strategy while considering the reliability of the VCPs, and we proved the LTA strategy could complete the task with a deadline constraint as soon as possible.
(3) Based on the (LTA) strategy, we proposed a new task assignment algorithm while considering the task deadline constraints.
(4) We evaluated the efficiency of the proposed algorithm based on the simulation experiments. The experimental results showed that our proposed algorithm had a greater benefit than the current algorithms.

The rest of this paper is organized as follows: In the next section, we review the related work. In Section 3, we introduce the definition of the problems. In Section 4, we illustrate our task assignment algorithm. In Section 5, we give the experimental results and the analysis of the proposed task assignment algorithm. In Section 6, we conclude this paper.

2. Related Works

Task assignment in distributed computing refers to how to assign tasks to workers. For example, if there are m tasks and n workers, there will be m^n task assignment plans possible in distributed computing. How to find the optimal solution from the task assignment plans according to a certain assignment goal is a typical NP complete problem [25]. In recent years, many scholars have done a great deal of work regarding the task assignment problem in distributed computing. In this section, we summarize the prior works that are related to ours and classify them into two categories: one category for static task assignment algorithms, and the other for dynamic task assignment algorithms.

2.1. Static Task Assignment Algorithms

The static task assignment algorithms have the task execution order and the task assignment plan specified in advance, and the task execution order cannot be changed until all tasks have been completed. In the past few decades, static task assignment algorithms have been extensively studied and can be classified into two categories: heuristic-based and random-search-based scheduling algorithms [10]. The random-search-based scheduling algorithms use the random search strategy to find a suboptimal solution in the given problem space. Recently, studies demonstrated that random-search-based scheduling algorithms had excellent performance in task assignment problems. For example, genetic algorithms (GAs) [26–28] are typical random-search-based scheduling algorithms. However, the computational complexity is often much higher than the heuristic-based scheduling algorithms. The heuristic-based scheduling algorithms use certain heuristic rules to find the solutions of task assignment problems, and they can be classified into three categories: list scheduling algorithms (LSAs), clustering scheduling algorithms (CSAs) and task duplication scheduling algorithms (TDSAs).

The basic method of the LSA [10,29–31] is to calculate the task priority according to specific rules to build a task scheduling list, and to loop to execute the following two steps until all tasks are scheduled: (1) fetch the first task from the task scheduling list; and (2) assign the selected task to the processor that can compute it in the shortest time. For example, HEFT [10] and earliest deadline first (EDF) [30] are classical LSAs. The main differences between them are that the ways of calculating the task priority and processor assignment strategy are different. The CSA [32–34] is a scheduling algorithm for an unbounded number of processors, which has three major phases: the task clustering phase, the mapping task clusters to processors phase and the task execution phase.

The first phase merges tasks with high communication overhead into the same cluster, and the number of clusters must be less than the number of processors. The second phase assigns tasks in the same cluster to the same processors. The last phase computes the tasks according to certain rules. A TDSA [35–37] reduces the communication overhead between tasks by assigning tasks redundantly to different processors. Although the TDSA reduces the communication overhead, it increases the computation overhead. Therefore, it is mainly used for task scheduling with a high communication overhead.

In the above three categories of algorithma, the LSA has the best performance, and is simple to implement with a low computational complexity [10]. Therefore, as a baseline algorithm, it is often extended and applied to many application scenarios. This paper is a substantially extended version of the LSA. However, the disadvantage of the LSA is that it does not adequately consider the dynamic of distributed computing. Motivated by this, based on the LSA, we proposed a dynamic task assignment algorithm called maximum benefit scheduling (MBS), that can dynamically adjust the task scheduling order of the scheduling list. In contrast to the LSA, we adopted a new pruning strategy to determine the acceptance of newly-arrived tasks, and adjust the task scheduling list dynamically. In practical applications, tasks are usually dynamically updated; therefore, our algorithm has wide applicability. In the next section, we introduce the related works regarding dynamic scheduling algorithms in detail.

2.2. Dynamic Task Assignment Algorithms

In a static task assignment algorithm, we cannot schedule tasks until we know all task dependency in advance. However, we cannot know all task dependency in advance under practical applications. For example, conditional branching is a program structure that can cause uncertainty. For conditional branching, we cannot determine the successor tasks until the algorithm has been executed. Therefore, it is necessary to design an efficient task assignment algorithm that can assign tasks dynamically while the application is being executed. Usually, these kinds of algorithms are called dynamic task assignment algorithms. In contrast to the static task assignment algorithms, the dynamic task assignment algorithms have the task execution order and the task assignment plan determined dynamically according to the application's execution [38–40]. To execute parallel programs more efficiently in VCPs, many scholars have proposed many dynamic task assignment algorithms [16,18,41] in recent decades.

In prior works, the goal was either to maximize the number of task completions or to minimize the makespan. For example, based on a genetic algorithm, Estrada et al. [41] proposed a dynamic task assignment algorithm in a volunteer computing system to maximize the number of task completions. To automatically generate scheduling policies, the algorithm considered a large number of possible scheduling strategies and used if-then-else rules to create large-space searches for each scheduling strategy. Guler et al. [16] proposed a dynamic task assignment algorithm under the constraints of the power price, and their goal was also to maximize the number of task completions. In the paper [18], the authors proposed a dynamic algorithm which used the estimation of the spot-check rate to minimize the makespan. These works are different from ours, as our goal is to maximize the benefit of the VCPs.

These works did not consider deadline constraints. For the deadline constraints, there are also some prior works [17,42], but they did not consider the reliability. Ghafarian et al. [17] proposed a cloud-aware workflow scheduling algorithm. In their algorithm, they first partitioned a workflow into sub-workflows to minimize the data dependencies and scheduled these sub-workflows to the volunteers according to the proximity of resources and the load balancing policy. Second, they estimated the execution time of each sub-workflow, if the sub-workflow missed its deadline, they would consider re-scheduling of the sub-workflow in the public cloud resources. Finally, they proved their algorithm can increase the percentage of workflow within its deadline. Their algorithm can increase the percentage of workflow within its deadline, but they need to use the public cloud resources, which may influence the reliability of computing. Xu et al. [42] proposed two algorithms called deadline preference dispatch scheduling (DPDS) and improved dispatch constraint scheduling (IDCS). The goal of the two algorithms was to maximize the number of completed tasks, which is different from our goal. Moreover, the two algorithms did not consider the reliability.

For the reliability, there are also many prior works [22–24]; however, their optimization goals are different from ours. The works [22,23] have been described before, so we do not reiterate them here. Xu et al. [24] proposed two algorithms with the goal of minimizing the maksepan of tasks, which is different to our goal. In summary, the existing algorithms cannot solve the dynamic task assignment problem under the disturbance of availability time, and the goals of existing algorithms are different from our algorithm, which aims to maximize the benefit of VCPs. Motivated by this, in this paper, we proposed a dynamic task assignment algorithm that considered deadline constraints and improved the reliability on the basis of the LTA strategy proposed in this paper, as the goal of our algorithm was to maximize the benefit of VCPs.

3. Problem Description

In this section, we describe the task assignment problem. For the ease of description, Table 1 summarizes the main notations used in this paper.

In this paper, we mainly focused on the task assignment problem with the maximum benefit. Following prior works [17,22,23,43], we assumed that there was a server and nodes (workers) in a VCP. The server is responsible for receiving and assigning tasks from users, and the nodes compute the tasks and return their results to the server. Usually, the parameters of the tasks from the users are known in advance, such as deadline constraints, benefits, and computing costs, and the server can further divide the large task from users into small tasks [44]. At the same time, when a node joins in a VCP, the expected availability time of the node is known in advance, and it is often disturbed, as mentioned before.

For the ease of understanding the task assignment problem, we denote a task set in $VCPs$ by the set $T = \{t_1, t_2, \ldots, t_{|T|}\}$. Following the prior work [43], we suppose that each task t_i can be completed by a node n_j or nodes in $VCPs$. In addition, for different tasks, we assume that the nodes provide the same computing power. A node-set is denoted by the set $N = \{n_1, n_2, \ldots, n_{|N|}\}$. Next, we give the definitions for the tasks and nodes as follows:

Table 1. The descriptions of the main notation.

Notation	Definition
T	The set of tasks
$T^{'}$	The acceptable task set
t_i	The ith task of the set T
$\|T\|$	The number of tasks in the set T
N	The set of nodes
n_j	The jth node of the set N
$\|N\|$	The number of nodes in the set N
h_j	The node n_j contributes h_j hours
$n_j.trust$	The trust value of the node n_j
$t_i.cost$	The completion time needed of t_i
$t_i.benefit$	The benefit of completing t_i within its deadline
$t_i.deadline$	The deadline of t_i
B	The total benefit of $VCPs$
$B_.ideal$	The ideal total benefit of $VCPs$
λ	The compensation coefficient
$Range_n[l_a, l_b]$	The confidence interval of the node n_j
$ratio_n$	The ratio of the actual availability time of node n to its expected availability time
$probability_n$	The probability of different ratios
$D(n_j)$	The total time assigned to the node n_j

Definition 1 (Task). *Task t_i is a four dimensional array that is denoted by $(id, t_i.cost, t_i.deadline, t_i.benefit)$. The id represents the order of the tasks joined; $t_i.cost$ represents the time that a node needs $t_i.cost$ hours to complete t_i and the unit of $t_i.cost$ is an hour; $t_i.deadline$ represents the deadline constraint of t_i and the unit of $t_i.deadline$ is also an hour. If t_i is executed at time l_1, the deadline constraint of t_i represents that t_i must be completed at $l_1 + t_i.deadline$; otherwise the task is not completed; $t_i.benefit$ represents the benefit from completing t_i within the deadline constraints, and the unit of t_i is cents.*

For example, as shown in Figure 1a, $t_1.cost = 2$, which means that the node n_j will take two hours to complete the task t_1. The deadline constraint of task t_1 is two hours and begins at time l_1, which means that task t_1 must be completed before l_1 + 2. The benefit of t_1 is 100, which means that the VCP will obtain 100 cents, if t_1 has been completed before the deadline constraints.

Definition 2 (node). *Node n_j is a double dimension array, which is denoted by (id, h_j). The h_j represents the expected availability time of the node n_j is h_j hours, and id represents the order of the node join in VCPs.*

For example, as shown in Figure 1b, $h_2 = 4$ means the availability time of node n_2 is four hours.

T	$t_i.cost$	$t_i.benefit$	$t_i.deadline$
t_1	2	100	2
t_2	3	120	1
t_3	1	80	3
t_4	2	110	1
t_5	6	300	4
t_6	5	200	4

(**a**)

N	h_j
n_1	4
n_2	4
n_3	3
n_4	4
n_5	3

(**b**)

Figure 1. Task set and node set at time l_1. (**a**) Task set at time l_1; (**b**) node set at time l_1.

The goal of our task assignment was to maximize the benefit for the VCPs under the available resource constraints and deadline constraints of the tasks. The available resource constraint was as follows:

$$D(n_j) \leq h_j, \tag{1}$$

where $D(n_j)$ represents the total time assigned to the node n_j. Equation (1) ensures that the total time assigned to the node n_j must be less than or equal to the expected available time.

There may be many tasks submitted to a VCP in a period. The submitted tasks are denoted by set T, and the acceptable task set is denoted by T'. For any task, t_i' of T, if it can be completed within its deadline constraints, it is called a positive benefit; otherwise, it is called a negative benefit. For a negative benefit of task t_i', the VCP needs to compensate λ t_i.benefit. Therefore, the total benefit denoted by B of the VCP is defined as follows:

$$B = \sum_{t_i' \in T', t_i' \text{ is a positive benefit}} t_i'.benefit - \sum_{t_i' \in T', t_i' \text{ is anegative benefit}} \lambda t_i'.benefit \quad (0 \leq \lambda \leq 1). \tag{2}$$

The nodes that participated in the VCPs are not always available, for instance, due to the non-VC application CPU load exceeding its threshold, which interrupts the expected available time of a node. For the task with deadline constraint, the disturbance may cause the task to miss its deadline, which will cause the VCP to not obtain the benefit for completing the task. Therefore, the disturbance will mean that the actual benefit is less than the expected benefit, which is a main challenge in maximizing the benefit for the VCPs. To address this challenge and reduce the impact of the disturbance, in this paper, we use a new task assignment strategy based on a list, which ensures the expected maximum benefit is approximated to the actual benefit.

4. Algorithm Description

In this section, we first introduce a new list-based task assignment (LTA) strategy (Section 4.1), and then we introduce a task assignment algorithm with a maximum benefit based on the LTA strategy in detail (Section 4.2).

4.1. The List-Based Task Assignment (LTA) Strategy

For ease of understanding the LTA strategy, we introduce our definitions as follows:

Definition 3 (List). *For a task set T, list L is a permutation of tasks in set T. List L specifies the execution order of the tasks. Specifically, for a given list L,* $L = \{t_1, t_2, \ldots, t_{|T|}\}$, $\forall t_i \in L (0 < i < \mid T \mid)$*; then* $COM(t_i) < COM(t_i + 1)$*, where* $COM(t_i)$ *is the makespan of* t_i*.*

For a given list L, we proposed a new list-based task assignment (LTA) strategy, which contained three phases:

- Phase 1: The LTA strategy first fetches the first task of the list L and then assigns it to the appropriate node. Specifically, during the task assignment, nodes are not always available due to the reasons mentioned before, which makes the actual benefit to VCPs less than the expected benefit. Although the available time of a node is grouped by a large number of discrete time intervals, a statistical analysis shows that it is possible to obtain a relatively accurate availability interval using probabilistic law [22,24,45]. Based on this, in the LTA strategy, we first select the time interval when the node remains available to assign tasks, and we adopt the confidence interval proposed by Xu et al. [24] to predict the time interval. The definition of the confidence interval is as follows:

Definition 4 (Confidence Interval). *The confidence interval of a node n_j at time l_1 is denoted by $Range_j[l_a, l_b]$, which means that n_j can remain available before time l_a and may be unavailable at any time in the time interval $[l_a, l_b]$.*

Briefly, the upper bound of the confidence interval about node n_j was h_j, and the lower bound of the confidence interval can be calculated by interpolation. Specifically, we counted the actual availability time and the expected availability time of n_j nearly 100 times and obtained ratios of the actual availability time and expected availability time about n_j, and presence probabilities of different ratios, as shown in Figure 2a. In Figure 2a, $ratio_n$ represents the ratio of the actual availability time of n_j to the expected availability time and the $probability_n$ represents the presence probability of the different ratios.

Therefore, according to Figure 2a, for any given h_j of n_j, the lower bound of the confidence interval and the presence probability of different actual availability times can be calculated by interpolation. For example, if the expected availability time of n_j is 4 h, then according to Figure 2a, the lower bound of the confidence interval of n_j is 2 h. Similarly, we can determine any node in the VCPs according to interpolation, and the confidence intervals of the nodes at time l_1 are shown as Figure 2b.

$ratio_i$	$probability_i$
<=50%	*100%*
60%	*90%*
70%	*80%*
80%	*70%*
90%	*60%*
100%	*50%*

(**a**)

V	Confidence interval
v_1	$Range_1[3.4,4]$
v_2	$Range_2[2.6,4]$
v_3	$Range_3[2.2,3]$
v_4	$Range_4[3,4]$
v_5	$Range_5[2,3]$

(**b**)

Figure 2. The statistics of the actual availability time and expected availability time of the node n_j and confidence intervals of the nodes at time l_1. (**a**) Statistics of the actual availability time and expected availability time of the node n_j; (**b**) confidence intervals of nodes at time l_1.

- Phase 2: According to the definition of the list, task $t_i + 1$ could not be executed until that task t_i was completed. Therefore, to make full use of the computing resources in VCPs, during the assigning task t_i, we divided it into $|N|$ equal task slices and assigned these task slices to nodes in the VCPs, and the size of each task slice was $t_i.cost/|N|$. Specifically, we supposed that task t_i was executed at time m, and the confidence interval of node n was $Range_n[l_a, l_b]$. During assigning these task slices, for each node n in the VCPs, if $m + t_i.cost/|N| \leq l_a$; then all the nodes could complete the task slice and assign the task slice whose size was $t_i.cost/|N|$ to each node n. Otherwise, we assigned the task slice whose size was $l_a - t_i.cost/|N|$ to each node n, and we added the rest of task t_i to the head of the list and stopped assigning tasks to node n.
- Phase 3: Finally, if the list L is null, we assume that all tasks have been completed; if all time intervals of nodes keep available have been assigned, and there are still uncompleted tasks, the LTA strategy would not stop selecting time intervals until all nodes disconnect or tasks are completed.

For example, given a task set T and a node set N, as shown in Figure 1, the confidence intervals of the nodes are as shown in Figure 2a, and the given list $L = \{t_2, t_4, t_1, t_3, t_5, t_6\}$. Suppose the current time m = l_1 = 0. According to the LTA strategy:

Step one: We first assign task t_2. For any node in VCPs, $m + t_2.cost/|N| < l_a$, so all nodes can complete t_2, and m is updated to 0.6. Since $m < t_2.deadline$, the VCPs can complete task t_2 and obtain the benefit.

Step two: Next, we assign task t_4. For any node in VCPs, $m + t_4.cost/|N| < l_a$; thus, all nodes can complete t_4, and m is updated to 1. Since $m < t_4.deadline$, the VCPs can complete task t_4 and obtain the benefit.

Step three: We continue to assign task t_1. For any node in the VCPs, $m + t_1.cost/|N| < l_a$, so all nodes can complete t_1, and m is updated to 1.4. Since $m < t_1.deadline$, the VCPs can complete task t_1 and obtain the benefit.

Finally, According to the LTA strategy, we continually select tasks in the list to assign until the node N is null or the list is null.

The specific task assignment is shown in Figure 3. According to Figure 3, we can know that the tasks t_2, t_4, t_1, t_3 can be completed and that task t_6 cannot be completed, and we cannot determine whether VCPs can complete t_5. For uncompleted tasks, such as t_6, the VCPs may need to compensate the user. Therefore, accepting such tasks will reduce the benefit of VCPs. Next, we give the definition of a valid list.

Definition 5 (Valid List). *For a task set T, list L is a permutation of tasks in set T, and list* $L = \{t_1, t_2, \ldots, t_{|T|}\}$ *specifies the execution order of the tasks. For a given list L, if the VCPs are used the LTA strategy to assign tasks,* $\forall t_i \in L(0 < i < |T|)$, $COM(t_i) < t_i.deadline$, *then list L is a valid list.*

Next, we prove the advantage of the LTA strategy in assigning tasks with deadline constraints.

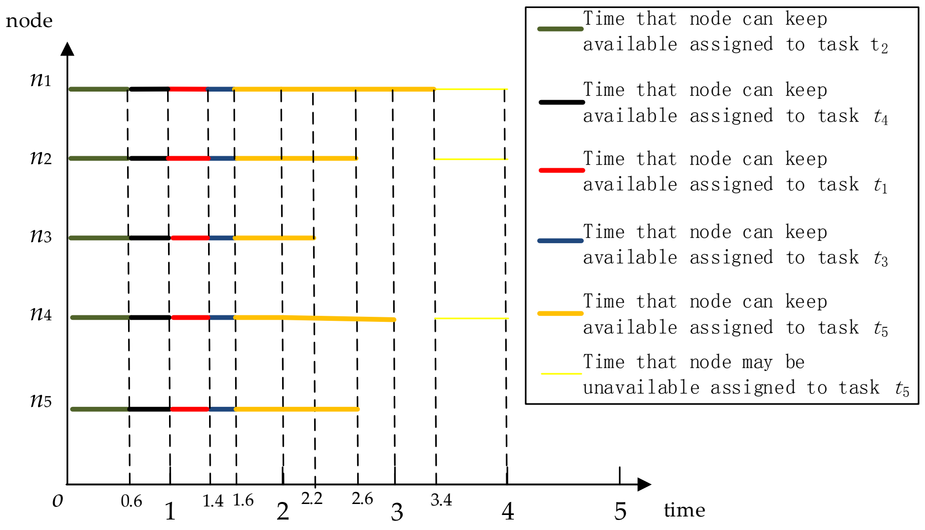


Figure 3. The specific task assignment of the list-based task assignment (LTA) strategy.

Theorem 1. *For a given task set T and a given list L, using the LTA strategy to assign tasks can guarantee: for any task t_i in set T, the task t_i can be completed in the shortest time, satisfying the definition of list L.*

Proof of Theorem 1. To facilitate the calculation in the proof process, we suppose that the expected available time provided by all nodes is equal to the actual available time. Given a list $L = \{t_1, t_2, \ldots, t_{|T|}\}$ and the current time m, at which the VCPs start calculating task t_1, and list L specifying: any task t_i that cannot be executed until all tasks $t_k (1 \le k < i)$ have been completed. Therefore, the earliest makespan of task t_i is $m + \sum_{i \in [1,k]} t_i.cost/|N|$. If the LTA strategy is used for assignment, the earliest makespan of task t_i is also $m + \sum_{i\in[1,k]} t_i.cost/|N|$. The theorem can be proven. □

Given a list L, according to the Theorem 1, the LTA strategy proposed in this paper has an advantage in the completion time when scheduling tasks with deadline constraints. Therefore, using the LTA strategy cannot guarantee that all tasks in the list are completed within their deadline constraints; however, other strategies also cannot guarantee that all tasks are completed within their deadline constraints. Otherwise, the task list is not a valid list.

4.2. The Maximum Benefit Scheduling (MBS) Algorithm

For a task set T and a given a list L, all tasks may not be completed using the LTA strategy, as shown in Figure 3. For example, for the task t_5, it is impossible to determine whether it can be completed. Therefore, accepting a task such as t_5, not only causes a waste of system computing resources but also may reduce the benefits of the VCPs. To maximize the benefits of the VCPs, to quickly determine the set of tasks that can be accepted and to find a valid list of tasks, the MBS algorithm based on the LTA strategy was proposed in this section. The MBS algorithm is a task assignment algorithm for a limited number of volunteer nodes, which has three major phases: (1) a task order computing phase for computing the task execution order of all tasks; (2) a task selection phase for selecting the accepted tasks, i.e., constructing a valid list; and (3) a task assignment phase for scheduling each accepted task based on the LTA strategy.

A task order computing phase: The MBS algorithm first sorted the tasks in descending order according to the benefit ratio; i.e., a task with a greater benefit ratio would be scheduled first. As the computing power of the VCPs is typically limited, the more intuitive method of calculating the benefit ratio is the benefit per unit time. However, large tasks and emergency (short deadline) tasks can also affect the benefits of the VCPs. Therefore, when we calculated the benefit ratio, we mainly considered three aspects as follows:

(1) As the computing power of the VCPs is limited, the benefit ratio of the task mainly used the benefit per unit time.
(2) If a task is too large, computing it will affect receiving other tasks, which may reduce the benefits of the VCPs. Therefore, we should consider penalizing such large tasks when calculating the task benefit ratios.
(3) If the deadline constraint of a task is too small, it is easy to miss its deadline constraint, which may affect the benefits of the VCPs. Therefore, we should consider punishing such emergency tasks (short deadline) when calculating the benefit ratio of the task.

After considering the above three factors, the benefit ratio of task t_i can be calculated as follows:

$$BR(t_i) = \frac{t_i.benefit}{t_i.cost \times Lj(t_i) \times Dj(t_i)} \tag{3}$$

where $BR(t_i)$ represents the benefit ratio of the task t_i, and $Lj(t_i)$ and $Dj(t_i)$ represent the punishment coefficients of large tasks and short-deadline tasks, respectively. Next, we will introduce the calculation of them in detail.

For a task t_i in task set T, if the calculation cost is larger than the average calculation cost of all the tasks in task set T, it is a large task; otherwise, we do not consider it a large task. Therefore, the definition of the $Lj(t_i)$ is as follows:

$$Lj(t_i) = \begin{cases} 1 + \dfrac{t_i.cost - \overline{T.cost}}{t_i.cost} & t_i \text{ is a large-task} \\ 1 & t_i \text{ is not a large-task} \end{cases} \tag{4}$$

where $\overline{T.cost}$ represents the average calculation cost of all the tasks in the task set T, and $\overline{T.cost} = \frac{\sum_{t_i \in T} t_i.cost}{|T|}$.

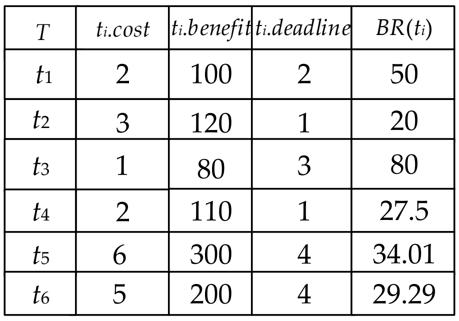
For a task t_i in the task set T, if the deadline constraints are smaller than the middle time of all the tasks in task set T, it is an emergency task; otherwise, we do not consider it an emergency task. Therefore, the definition of the $Dj(t_i)$ is as follows:

$$Dj(t_i) = \begin{cases} 1 + \left|\frac{t_i.deadline - \overline{T.deadline}}{t_i.deadline}\right| & t_i \text{ is a emergency task} \\ 1 & t_i \text{ is not a emergency task} \end{cases} \tag{5}$$

where $\overline{T.deadline}$ represents the middle time of all the tasks in the task set T, and $\overline{T.deadline} = \frac{t_i.deadline^{max}}{2}$ and $t_i.deadline^{max}$ are the largest deadline constraints of the tasks in task set T.

For example, a given task set T at time l_1 is shown in Figure 1a. We can calculate $\overline{T.cost} \approx 3.17$ and $\overline{T.deadline}$ = 2. As the calculation costs of tasks t_5 and t_6 are larger than $\overline{T.cost} \approx$, task t_5 and t_6 are large tasks. In addition, the deadline constraints of tasks t_2 and t_4 are smaller than $\overline{T.deadline}$; therefore, tasks t_2 and t_4 are emergency tasks.

According to Equations (3) and (4), $Dj(t_1) = 1$, $Dj(t_2) = 2$, $Dj(t_3) = 1$, $Dj(t_4) = 2$, $Dj(t_5) = 1$ and $Dj(t_6) = 1$; $Lj(t_1) = 1$, $Lj(t_2) = 1$, $Lj(t_3) = 1$, $Lj(t_4) = 1$, $Lj(t_5) \approx 1.47$ and $Lj(t_6) \approx 1.37$. Therefore, according to Equations (3)–(5), $BR(t_1) = \frac{t_1.benefit}{t_1.cost \times Lj(t_1) \times Dj(t_1)} = \frac{100}{2 \times 1 \times 1} = 50$. Similarly, we can calculate the benefit ratio of the other tasks in task set T, as shown in Figure 4.



T	$t_i.cost$	$t_i.benefit$	$t_i.deadline$	$BR(t_i)$
t_1	2	100	2	50
t_2	3	120	1	20
t_3	1	80	3	80
t_4	2	110	1	27.5
t_5	6	300	4	34.01
t_6	5	200	4	29.29

Figure 4. The benefit ratio of each task in task set T.

A task selection phase: To select the accepted tasks and construct a valid list, we used the MBS algorithm to propose theorems as follows:

Theorem 2. *For a given valid list* $L = \{t_1, t_2, \ldots, t_{|T|}\}$*, the current time* $m = 0$ *and a new task* t_{new}*, if* $\frac{t_{new}.cost}{|N|} + COM(t_i) > t_i.deadline$*, then a new list* L'*, which is constructed by inserting a new task* t_{new} *in any position of the interval* [1, *i*]*, is not a valid list.*

Proof of Theorem 2. The earliest completion time of the task t_{new} is $\frac{t_{new}.cost}{|N|}$; therefore, inserting a new task t_{new} in any position of the interval [1, i] will cause task t_i to not be completed within its deadline constraint. A new list L' constructed by inserting a new task t_{new} is not a valid list. The theorem can be proven. □

Theorem 3. *For a given valid list $L = \{t_1, t_2, \ldots, t_{|T|}\}$, and a new task t_{new}, if $\frac{t_{new}.cost}{|N|} + COM(t_i) > t_{new}.deadline$, then a new list L', which is constructed by inserting a new task t_{new} in any position of the interval $[i + 1, \mid T \mid + 1]$ is not a valid list.*

Proof of Theorem 3. Suppose that we can insert a new task t_{new} in any position of the interval $[i+1, |T|+1]$; then, the earliest completion time of the task t_{new} is $\frac{t_{new}.cost}{|N|} + COM(t_i)$. Clearly, task t_{new} cannot be completed within its deadline constraint. Therefore, a new list L', which is constructed by inserting a new task t_{new} in any position of the interval [i+1, ∣T∣+1], is not a valid list. The theorem can be proven. □

According to the Theorems 2 and 3, we can select the accepted tasks and construct a valid list. Specifically, the function called EffectiveList, which can construct a valid list, is described in detail in Algorithm 1. Next, we illustrate the implementation of Algorithm 1. Examples are shown as follows:

Algorithm 1 The EffectiveList function (N_l, T_l, m, L).

```
Input: node set $N_l$, task set $T_l = \{t_1, t_2, \ldots, t_{|T|}\}$, current time $l = m$, the current valid list $L$.
Output: valid list set $E_{|T|}$.
1: Calculate the benefit ratio of each task in task set $T_l$ according to the Equations (3)–(5).
2: Construct the new task set $T_l$'in descending order according to the benefit ratio of tasks in task set $T_l$.
3: $E_0 = \{L\}$ // $E_0$ is a valid list set
4: for each $t'_i \in T'_l (i \geq 1)$ do
5:   $E_i = \emptyset$
6:   for each $L' \in E_{i-1}$ do
7:     Find the head position $b$ and tail position $e$ where $t'_i$ can be inserted according to Theorems 2 and 3.
8:     if $b \leq e$ then
9:       for each $j \in [b, e]$ do
10:        Insert task $t'_i$ into the position $j$ of the effective list $L'$ to construct a new list $L''$, and calculate the completion time of task $t'_i$ according to the LTA strategy.
11:        if $L''$ is a valid list then
12:          Insert $L''$ into $E_i$
13:        end if
14:      end for
15:    end if
16:  end for
17:  if $E_i = \emptyset$ then
18:    $E_i = E_{i-1}$
19:  end if
20: end for
21: return valid list set $E_{|T|}$.
```

Given a benefit ratio of each task in task set T_{l_1} at time l_1, as shown in Figure 4, on this basis, we can construct the new task set $T'_{l_1} = \{t_3, t_1, t_5, t_6, t_4, t_2\}$ in descending order according to the benefit ratio of the tasks in task set T_l. For ease of calculation, we suppose the current times $l_1 = 0$ and $L = \emptyset$.

Step one: According to line 5 of Algorithm 1, we first select task t_3 and determine whether it can be accepted. According to line 7 of Algorithm 1, we can know $L' = \emptyset$ at the moment. According to Theorems 2 and 3, the head position b and tail position e where t'_i can be inserted is 1; i.e., $b = e = 1$.

Therefore, $L'' = \{t_3\}$. According to the LTA strategy, it can be known that $COM(t_3) = \frac{t_3.cost}{5} = 0.2$, and $COM(t_3) < l_1 + t_3.deadline$. Therefore, $L'' = \{t_3\}$ is a valid list, and $E_1 = \{\{t_3\}\}$.

Step two: Next, we select task t_1 and determine whether it can be accepted. According to the line 7 of Algorithm 1, we can know $L' = \{t_3\}$ at the moment. According to Theorems 2 and 3, the head position b and tail position e are 1 and 2, respectively; i.e., $b = 1, e = 2$. Suppose that task t_1 can be inserted in position 1; then $L'' = \{t_1, t_3\}$. According to the LTA strategy, it can be known that $COM(t_1) = \frac{t_1.cost}{5} = 0.4$, and $COM(t_3) = COM(t_1) + \frac{t_3.cost}{5} = 0.6$. As both tasks t_1 and t_3 can be completed within their deadline constraints, $L'' = \{t_1, t_3\}$ is a valid list and $E_2 = \{\{t_1, t_3\}\}$. Suppose that task t_1 can be inserted in position 2; then $L'' = \{t_3, t_1\}$. According to the LTA strategy, it can be known that $COM(t_3) = \frac{t_3.cost}{5} = 0.2$, and $COM(t_1) = COM(t_3) + \frac{t_1.cost}{5} = 0.6$. As both tasks t_1 and t_3 can be completed within their deadline constraints, $L'' = \{t_3, t_1\}$ is a valid list and E_2 is equal to $\{\{t_1, t_3\}, \{t_3, t_1\}\}$ at the moment; i.e., $E_2 = \{\{t_1, t_3\}, \{t_3, t_1\}\}$. Similarly, it can be known that the final valid list set $E_{|T|}$ is equal to $\{\{t_5, t_1, t_3\}, \{t_1, t_5, t_3\}, \{t_1, t_3, t_5\}, \{t_5, t_1, t_1\}, \{t_3, t_5, t_1\}, \{t_3, t_1, t_5\}\}$.

A task assignment phase: In this phase, the MBS algorithm selects a valid list from $E_{|T|}$ and schedules each task of the valid list based on the LTA strategy.

However, in practical applications, nodes can join and exit freely, and new tasks are allowed to insert. Therefore, to assign tasks dynamically, the MBS algorithm sets a monitoring mechanism, which cannot be triggered until meeting the following two conditions simultaneously:

(1) There are new nodes or tasks arriving at the VCPs;
(2) There are no tasks that are being executed.

The MBS algorithm selects tasks that can be accepted according to Theorems 2 and 3 and assigns tasks according to the LTA strategy, which first selects the time interval to assign tasks, and the time interval keeps nodes available. Based on this, all accepted tasks can be completed within their deadline constraints. Therefore, we did not consider the node's exit calculation when setting the monitoring mechanism. To improve the utilization of the computing resources and the performance of the algorithm, in the future, we will consider it. After the monitoring mechanism is triggered, the MBS algorithm recalls the EffectiveList function to select the tasks and assigns the selected tasks according to the LTA strategy. The MBS algorithm is described in detail in Algorithm 2.

Next, we illustrate the implementation of the Algorithm 2. Examples are shown as follows:

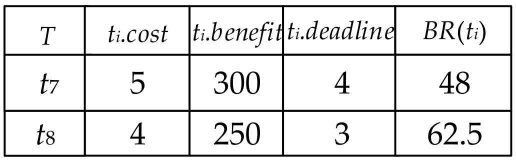
For example, for the ease of calculation, a given task set T_l and a node set N_l at time l are shown in Figure 1, and the current time $l = m = 0$; then, the MBS algorithm calls the function EffectiveList (N_l, T_l, l), and we can obtain the valid list set $E_{|T|} = \{\{t_5, t_1, t_3\}, \{t_1, t_5, t_3\}, \{t_1, t_3, t_5\}, \{t_5, t_1, t_1\}, \{t_3, t_5, t_1\}, \{t_3, t_1, t_5\}\}$. Suppose that the MBS algorithm selects the valid list $L = \{t_1, t_3, t_5\}$ to assign. According to line 5 of the MBS algorithm, we first assign task t_1. According to line 6 of the MBS algorithm, the task slice of task t_1 is l', and $l' = \frac{t_1.cost}{|N_l|} = 0.4$. For all nodes, since $m + l' < l_a$, the task slice whose size is 0.4 is assigned to n_1, n_2, n_3, n_4, n_5. $COM(t_1) = 0.4 < t_1.deadline$, and thus t_1 has been completed. At the moment, m is updated to 0.4 and $B = 100$. Next, we continuously select task t_3 to assign. According to line 6 of the MBS algorithm, the task slice of task t_3 is l', and $l' = \frac{t_3.cost}{|N_l|} = 0.2$.

For all nodes, as $m + l' = 0.6 < l_a$, the task slice whose size is 0.2 is assigned to n_1, n_2, n_3, n_4, n_5. $COM(t_3) = 0.6 < t_3.deadline$, so t_3 has been completed. At the moment, m is updated to 0.6 and $B = 180$. Next, we continuously select task t_5 to assign. According to line 6 of the MBS algorithm, the task slice of task t_5 is l', and $l' = \frac{t_5.cost}{|N_l|} = 1.2$. For all nodes, since $m + l' = 1.8 < l_a$, the task slice whose size is 1.2 is assigned to n_1, n_2, n_3, n_4, n_5. $COM(t_5) = 1.8 < t_5.deadline$, so t_5 has been completed. At the moment, m is updated to 1.8 and $B = 480$.

For the ease of calculation, we supposed that there were two new tasks that arrived at time $l = 1$ and that there are no new nodes or tasks that arrived at other times. The specific parameters of the new tasks that arrived at the time $l = 1$ are shown in Figure 5.

Algorithm 2 The MBS algorithm.

```
Input: node set $N_l$, task set $T_l = \{t_1, t_2, \ldots, t_{|T|}\}$, current time $l = m$, the current valid list $L$.
Output: the benefit of VCPs $B$
 1: Wait until the monitoring mechanism is triggered
 2: Call the EffectiveList function($N_l, T_l, m, L$)
 3: Select a valid list $L$ set from $E_{|T|}$
 4: while $L \neq \emptyset$ and $N_l \neq \emptyset$ do
 5:     Take the first task $t'$ from list $L$
 6:     $l' = t'.cost/N_l$ // calculate the size of task slice
 7:     for each $n \in N_l$ do
 8:         if $l_a > m + l'$ then
 9:             add $< n, l' >$ to $T.assign$ //$T.assign$ represents the task assignment plan
10:             $t'.cost = t'.cost - l'$
11:         else
12:             add $< n, l_a - m >$ to $T.assign$
13:             $t'.cost = t'.cost - (l_a - m)$
14:             delete node $n$ from $N_l$
15:         end if
16:     end for
17:     $m = m + l'$
18:     if $t'.cost == 0$ then
19:         delete task $t'$ from $L$
20:         $B = B + t'.benefit$
21:     end if
22: end while
23: return $B$
```



T	$t_i.cost$	$t_i.benefit$	$t_i.deadline$	$BR(t_i)$
t_7	5	300	4	48
t_8	4	250	3	62.5

Figure 5. New tasks that arrived at time $l = 1$.

As task t_5 is being executed at the time $l = 1$, the monitoring mechanism of the MBS algorithm cannot be triggered until task t_5 is completed. The task t_5 was completed at time $l = 1.8$, and the monitoring mechanism of the MBS algorithm was triggered at the time $l = 1.8$. After the monitoring mechanism is triggered, the MBS algorithm recalls the function EffectiveList (N_l, T_l, l) to calculate the valid list set. According to line 4 of the Algorithm 1, we can know $E_0 = \emptyset$.

According to line 5 of Algorithm 1, we first select task t_8 and determine whether it can be accepted. According to line 7 of Algorithm 1, we can know $L' = \emptyset$ at the moment. According to Theorems 2 and 3, the head position b and tail position e where t_8' can be inserted are 1; i.e., $b = e = 1$. Therefore, $L'' = \{t_8\}$. According to the LTA strategy, it can be known that $COM(t_8)$ = 3.2, and $COM(t_8)$ < 1 + $t_8.deadline$. Therefore, $L'' = \{t_8\}$ is a valid list, and $E_1 = \{\{t_8\}\}$.

Next, we select task t_7 and determine whether it can be accepted. According to line 7 of Algorithm 1, we can know $L' = \{t_8\}$ at the moment. According to Theorems 2 and 3, the head position b and tail position e where t_8' can be inserted are 1 and 2, respectively; i.e., $b = 1, e = 2$.

Suppose that task t_7 can be inserted in position 1; then $L'' = \{t_7, t_8\}$. According to the LTA strategy, it can be known that $COM(t_7) = \frac{t_1.cost}{|N|} + COM(t_8) > 1 + 3.2 > 1 + t_8.deadline$. As task t_8 cannot be completed within its deadline constraint, task t_7 cannot be inserted in position 1. Suppose that task t_7 can be inserted in position 2; then $L'' = \{t_8, t_7\}$. According to the LTA strategy, it can be known that $COM(t_7) = \frac{t_1.cost}{|N|} + COM(t_8) = 5 + 3.2 > 1 + t_7.deadline$. As task t_7 cannot be completed within its deadline constraint, task t_7 cannot be inserted in position 2.

Based on the above analysis, we can know that the VCPs can accept task t_8 to assign; the specific task assignment is shown as Figure 6. As shown in Figure 6, we can know that tasks t_1, t_3, t_5, t_8 can be completed within their deadline constraints, and the benefit B of the VCPs is 720.

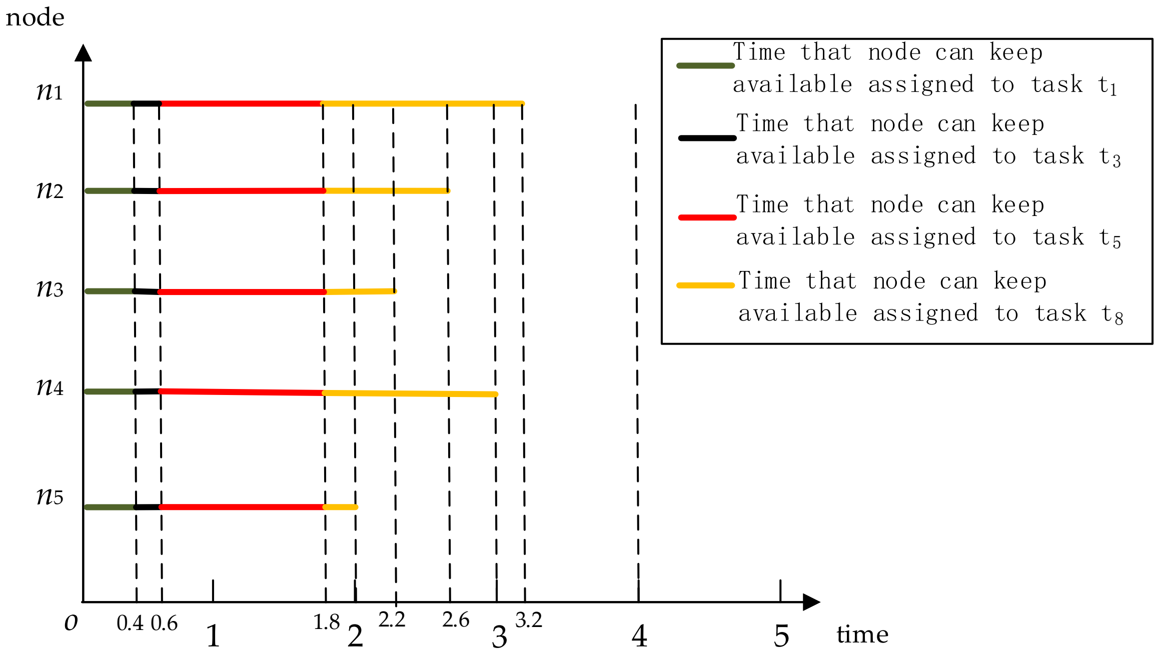


Figure 6. The specific task assignment of the maximum benefit scheduling (MBS) algorithm.

5. Experimental Evaluation

In this section, we evaluate the performance of our proposed algorithm through simulation experiments. Specifically, we used a static task set, a dynamic task set and different average connection speed to compare the proposed algorithm with MOC algorithm [23], HEFT-AC and HEFT-ACU [22] as mentioned before. The simulation experiment environment consisted of one server node and fifty sub nodes. All nodes were configured with Intel Core i7 4790 CPU@3.4 GHz, 8 GB DDR3 memory, 1 TB hard disk and a Windows 10 operating system. At the same time, we adopted the interpolation method to obtain the confidence interval of each subnode.

5.1. Experimental Results and Analysis on Static Task Sets

In the experiment of this section, we tested the performance of proposed algorithm on static task sets. Specifically, we used three common tasks: word frequency statistics, inverted index and distributed Grep. The input files, the data and dump files, were provided by Wikipedia (the main contents are entries, templates, picture descriptions, basic meta-pages, etc.). In the experimental results and analysis on static task sets, we mainly considered the influences of the three main parameters as follows:

- The task set scale; i.e., the number of tasks included in the task set T.

- The average size of tasks in task set T, denoted by S, which was measured by the number of input file fragments. Specifically, in our experiment, $\forall t_i \in T$, the size of the task t_i was a random value of the interval $[0.5S, 2.5S]$.
- The average deadline constraints of tasks in T, denoted by D. Specifically, in our experiment, $\forall t_i \in T$, the deadline constraint of the task t_i was a random value of the interval $[0.8D, 1.6D]$.

In addition, in our experiment, $\forall t_i \in T$, the average deadline constraint of the task was a random value of the interval $[800, 1600]$, $t_i.benefit$ was a random value of the interval $[400, 600]$ and $\lambda = 0.5$. Table 2 shows the default values and ranges of the main parameters.

Table 2. Experimental default parameters.

Parameter	Default	Value Range
Average size of tasks (MB)	128	64–320
Task set scale	50	20–100
Average deadline constraints(s)	1000	800–1600

In this paper, we considered the benefit of the VCPs to be the primary performance index. In addition, we also used the task completion rate and the task acceptance rate to measure the performance of the algorithm more comprehensively. Specifically, the task completion rate and task acceptance rate are defined as follows:

$$task\ completion\ rate = \frac{the\ number\ of\ completion\ tasks\ within\ deadline\ constraints}{the\ number\ of\ the\ accepted\ tasks}, \tag{6}$$

$$task\ acceptance\ rate = \frac{the\ number\ of\ accepted\ tasks}{the\ number\ of\ the\ task\ set\ T}. \tag{7}$$

The proposed algorithm is the first algorithm to provide the maximum benefit from independent tasks with deadline constraints in VCPs. Although there are some similar prior works, they solved different problems. For comparison, on the one hand, we chose the MOC algorithm as the comparison algorithm, which is one of the latest works designed for independent tasks with deadline constraints in distributed systems and is more effective than many typical algorithms. However, the goal of the MOC algorithm mainly focuses on the number of completed tasks, and it does not consider the benefit in VCPs; thus, we adjusted it to deal with the assignment with benefits in VCPs so that we could compare the results with our proposed algorithm. On the other hand, we chose the HEFT-AC and HEFT-ACU algorithms as the comparison algorithms, which are the latest ones designed for independent tasks in VCPs under the disturbance that occurred. However, the goals of the two algorithms mainly focus on the makespan of tasks, and they do not consider the benefit in VCPs; thus, we adjusted them to deal with the assignment with benefits so that we could compare the results with our proposed algorithm. The three algorithms [22,23] have been described before, so we do not reiterate them here. Besides, to more comprehensively evaluate the performance of the proposed algorithm, we introduced the ideal benefit,; i.e., the VCPs could compute a timeout task without any compensation.

Specifically, for any task set T, the current time is l_1 and the task set $T' =$ is arranged in descending order according to the benefit ratio of each task. The ideal benefit of the VCPs is defined as finding the m tasks so that the completion time of the mth task is just greater than $l_1 + t_m.deadline$, and then the ideal benefit can be defined as follows:

$$B.ideal = \begin{cases} \sum_{i=1}^{m-1} t_i.benefit + \dfrac{N.remain}{t_m.cost} & N.remain < t_m.cost \\ \sum_{i=1}^{m} t_i.benefit & N.remain \geq t_m.cost \end{cases} \tag{8}$$

where $N.remain$ represents the rest computing resources of the VCPs after completing the $(m-1)$th task, and it can be defined as follows:

$$N.remain = \sum_{i=1}^{|N|} h_j - COM(t_{m-1}). \tag{9}$$

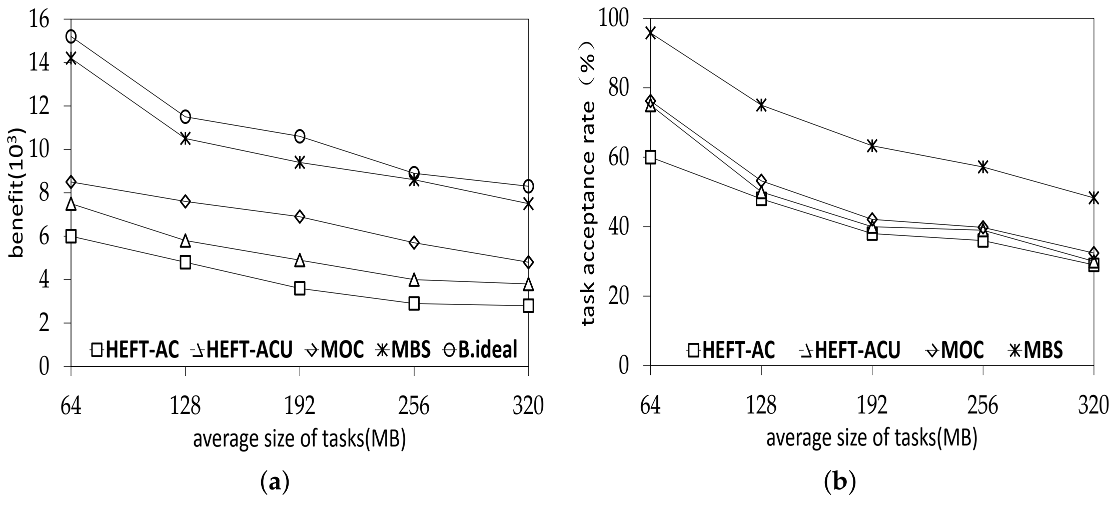
According to the definition above, $B.ideal$ is an ideal value. For example, the task set and node set at time l_1 are shown in Figure 1, and the task set is arranged in descending order of the benefit ratio to obtain $T' = t_3, t_1, t_5, t_6, t_4, t_2$. Suppose $l_1 = 0$, according to the above definition of the ideal value of the benefit; then, it can be known that m is 5. Thus, according to Equations (8) and (9), we can know that $B.ideal = 80 + 100 + 300 + 200 + 110 = 790$.

5.1.1. The Impact of the Average Size of Tasks

In this experiment, we tested the benefit, task acceptance rate and task completion rate under the different average sizes of tasks. Specifically, the task set scale was set to 50 and the average deadline constraint was set to 1000 s. We varied the average size of tasks from 64 to 320 MB. Figure 7 shows the experimental results. In Figure 7, we noticed that, regardless of the different average sizes of tasks, the performances of MBS and MOC were better than those of HEFT-AC and HEFT-ACU. This is because HEFT-AC and HEFT-ACU did not consider the deadline constraints of tasks, which caused some tasks to miss their deadline constraints. We also noticed that, with the average size of tasks increased, the four algorithms' benefit and task acceptance rate decreased. The reason is that the computing power of the VCPs consisted of the bounded nodes, when the average size of the tasks increased, task acceptance rates gradually decreased. Another reason is that, when the average size of the tasks increased, the node will be idle until the task data are stored. Therefore, the benefit and task acceptance rate decreases.

As shown in Figure 7a, $B.ideal$ represented the ideal benefit of the system. In Figure 7a,b, VCPs had a lesser benefit and lower task acceptance rate using the MOC than by using our proposed algorithm MBS, and the benefit of our proposed algorithm was closer to the ideal benefit. The reason is that the MBS algorithm, based on the LTA strategy, could complete tasks faster than the MOC. At the same time, MOC mapped the task with the lowest completion time to the computer with maximum robustness each time, which cannot guarantee the benefit of the task is significant. Besides, HEFT-ACU outperforms HEFT-AC; this is because that HEFT-ACU considered the disturbance of the nodes, which could complete more tasks on time than HEFT-AC.

In Figure 7c, we tested the impacts of the average tasks on the task completion rate of the algorithm. The task completion rate did not change significantly when the average size of the task increased. The MBS algorithm could almost complete all accepted tasks. As MOC discarded the tasks that missed the deadline, the task completion rate of MOC was lower than MBS. Besides, as HEFT-AC and HEFT-ACU did not consider the deadline constraints, the task completion rate was lower than MBS and MOC. As can be seen from Figure 7b,c, MBS performed better than MOC in both the task acceptance rate and the task completion rate, which is also the reason why MBS had a greater benefit.



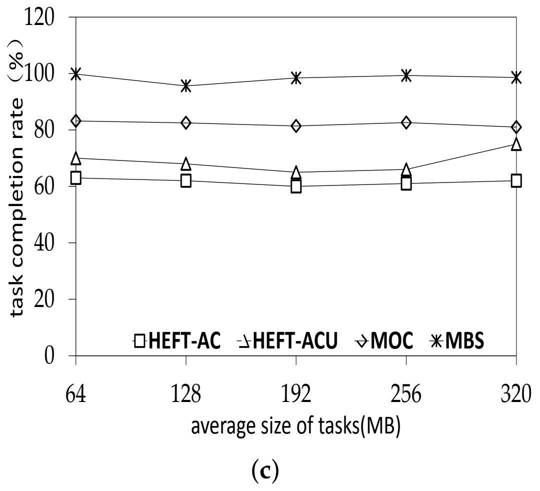


Figure 7. The impacts of average sized tasks on the performances of the algorithms: (**a**) The impact of the average size of the tasks on the benefit. (**b**) The impact of the average size of the tasks on the task acceptance rate. (**c**) The impact of the average size of the tasks on the task completion rate.

5.1.2. The Impact of the Average Deadline Constraints of Tasks

In this experiment, we tested the benefit, task acceptance rate and task completion rate under the different average deadline constraints. Specifically, the task set scale was set to 50, and the average size of tasks was set to 128 MB. We varied the average deadline constraints of tasks from 800 s to 1600 s. Figure 8 shows the experimental results. As can be seen from Figure 8, with the increase of the average deadline constraints of the tasks, the performances of the four algorithms gradually increased. The reason is that, with the increase of the average deadline constraints of the tasks, more tasks could be completed before their deadline constraints. We also noticed that MBS and MOC performed better than HEFT-AC and HEFT-ACU. The reason is that MBS and MOC will not compute tasks that missed the deadline. In contrast to MBS and MOC, despite the task missing its deadline, HEFT-AC and HEFT-ACU still computed it. Therefore, the performance of HEFT-AC and HEFT-ACU is lower than MBS and MOC. However, the task completion rate did not change significantly when the average deadline constraints of the tasks increased.

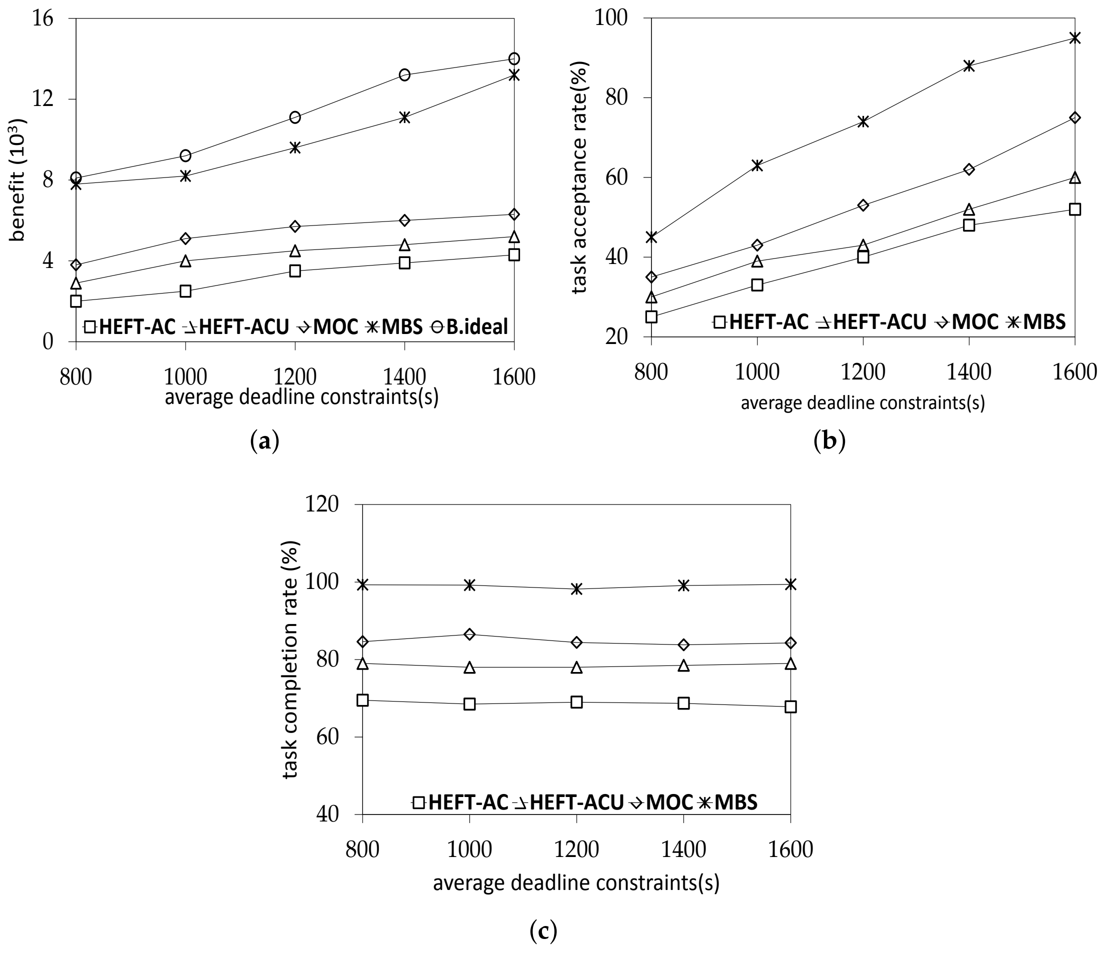
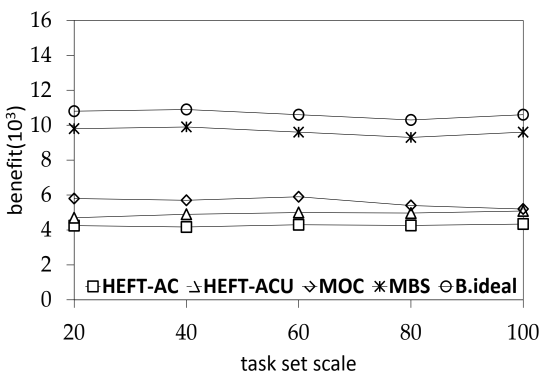


Figure 8. The impacts of the average deadline constraints on the performances of the algorithms: (**a**) The impact of the average deadline constraints on the benefit. (**b**) The impact of the average deadline constraints on the task acceptance rate. (**c**) The impact of the average deadline constraints on the task completion rate.

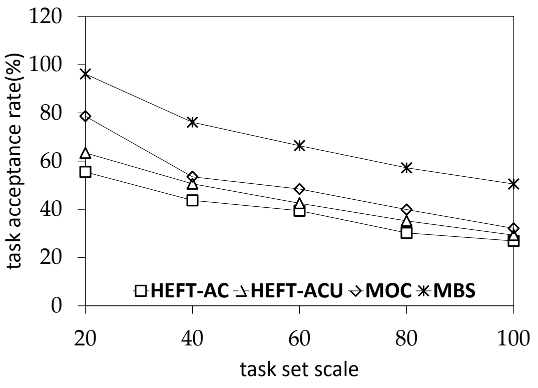
5.1.3. The Impact of the Task Set Scale

In this experiment, we tested the benefit, task acceptance rate and task completion rate under the different task set scale. Specifically, the average deadline constraint was set to 1000 s, and the average size of tasks was set to 128 MB. We varied the task set scale (i.e., the number of tasks) from 20 to 100. Figure 9 shows the experimental results. We noticed that MBS outperforms the other three algorithms. As can be seen from Figure 9a, when the number of tasks increased, the benefits did not significantly change. As the computing power of nodes was absolute in a period, the benefits did not change substantially. We also observed that MBS and MOC outperformed the other two algorithms. The reason is that MBS and MOC considered deadline constraints, which could compute more tasks than the two algorithms.

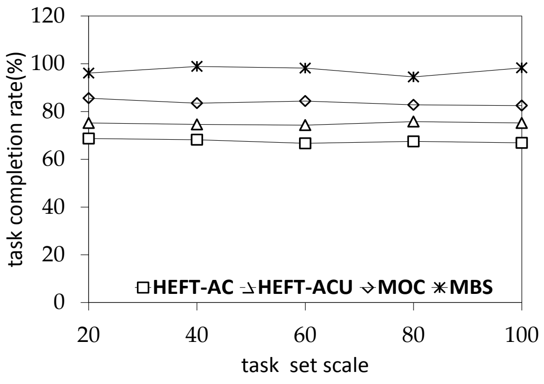
As shown in Figure 9b,c, when the number of tasks increased, the task acceptance rate and task completion rate decreased. The reason is that the deadline constraints and total computing power did not change. Therefore, the accepted tasks of VCPs did not change. Therefore, the task acceptance rate decreased. Besides, for MOC, with the number of tasks increased, more tasks wait on each computer; thus, the arriving tasks will miss their deadline constraints. Therefore, the task completion rate decreased, and it is lower than the MBS.



(**a**)



(**b**)



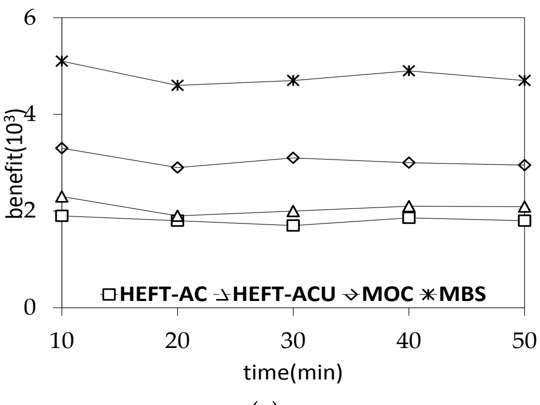
(**c**)

Figure 9. The impacts of the task set scale on the performances of the algorithms: (**a**) The impact of the task set scale on the benefit. (**b**) The impact of the task set scale on the task acceptance rate. (**c**) The impact of the task set scale on the task completion rate.

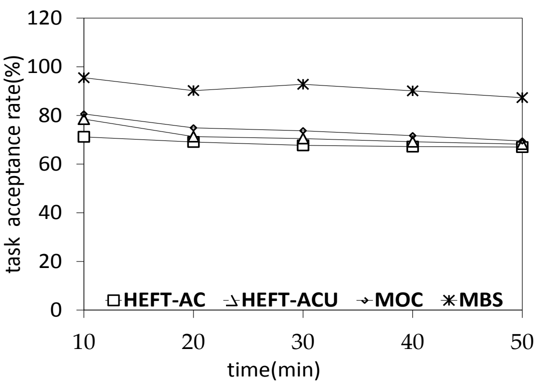
5.2. Experimental Results and the Analysis on Dynamic Task Sets

To validate the time efficiency of our proposed algorithm, we tested the performance on a dynamic set of applications under the same computing power, average size of tasks, and task set scale. The average size of tasks was 64 MB; task set scale is 50; the average deadline constraint of the task was a random value of the interval $[800, 1600]$, and $t_i.benefit$ was a random value of the interval $[400, 600]$. In the experiment, the system was monitored every 10 min to obtain the benefit of the VCPs, the task acceptance rate and task completion rate.

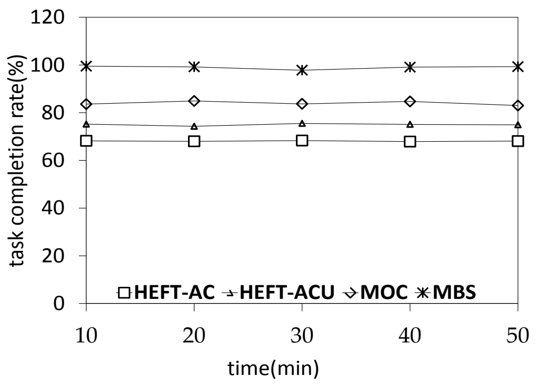
Figure 10 shows the experimental results. The experimental results show that the benefit, the task acceptance rate and the task completion rate of MBS outperform the others algorithms during the same time period. The reason is that MBS can complete tasks as soon as possible based on the LTA strategy and dynamically assign the tasks with a maximum benefit-ratio each time. In particular, the MBS algorithm's benefit and task acceptance rate in the first 10 min was higher than the MOC algorithm, and this is mainly because all the nodes could be used directly at the beginning. However, in other periods, the nodes needed to compute the uncompleted tasks from the previous period. Therefore, the benefit and the task acceptance rate of the VCPs are then lower than in the first 10 min.



(**a**)



(**b**)



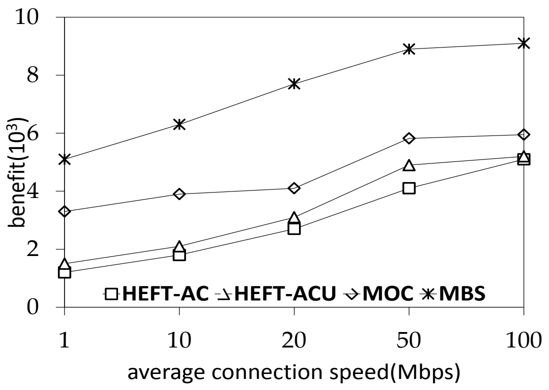
(**c**)

Figure 10. Performance comparisons of the algorithms on the dynamic task set: (**a**) Comparison of the benefit of the VCPs. (**b**) Comparison of the task acceptance rate. (**c**) Comparison of the task completion rate.

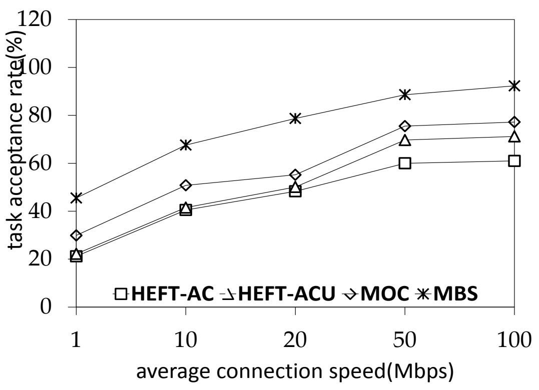
5.3. Experimental Results and the Analysis on Average Connection Speed

To validate the time efficiency of our proposed algorithm, we tested the performance of the proposed algorithm in dynamic task sets under the same computing power, average size of tasks and task set scale. Specifically, the average size of tasks was 64 MB, task set scale was 50, the average deadline constraint of the task was a random value of the interval $[800, 1600]$ and $t_i.benefit$ was a random value of the interval $[400, 600]$. We varied the average connection speed from 1Mbps to 100 Mbps. The experimental results are averaged over 100 runs.

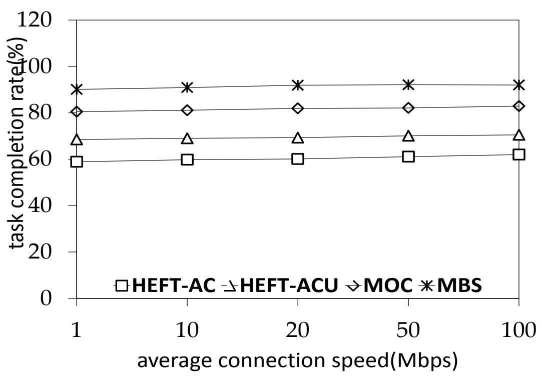
As we expected, Figure 11 shows that MBS outperforms the comparison algorithms in the same statistics time. In addition, we observed that MBS and MOC perform better than HEFT-AC and HEFT-ACU in benefit, task acceptance rate and task completion rate. The reason is that MBS and MOC assigned tasks enough time to meet tasks' deadline constraints by a pruning strategy and dropping tasks, respectively. We observed that with average connection speed increases, the performance of the four algorithms improved. The reason is that, when the average connection speed increases, it will take less time for tasks to store their required data into the assigned computers. Therefore, the number of accepted tasks and completed tasks on time will increase; thus, the benefit and task acceptance rate will increase. However, the task completion rate did not change significantly when the average connection speed.



(**a**)



(**b**)



(**c**)

Figure 11. Performance comparisons of the algorithms with different average connection speeds: (**a**) Comparison of the benefits of the VCPs. (**b**) Comparison of the task acceptance rates. (**c**) Comparison of the task completion rates.

6. Conclusions

In this paper, to maximize the benefit of the VCPs under considering the reliability, first, we proposed a new list-based task assignment (LTA) strategy, which adopted a reliability model proposed in prior work to predict the time interval that the node can keep available. The LTA strategy considered the deadline constraints of tasks, and we proved that the LTA strategy could complete the task with a deadline constraint as soon as possible. Secondly, on this basis, we proposed a new task assignment algorithm, called MBS, that can maximize the benefits of the VCPs while considering the deadline constraints of tasks. In the MBS algorithm, we first proposed the method of calculating the benefit ratio of tasks; then, we proposed Theorems 2 and 3 and adopted them to select tasks, and we assigned each selected task based on the LTA strategy. Finally, we conducted simulation experiments to prove that our proposed algorithm was more effective than the current algorithms in terms of the benefits, task acceptance rate and task completion rate. As a basic scheduling strategy, our algorithm can be used in other distributed computing applications, such as cloud computing. In the future, we will integrate VC with other distributed computing systems and consider other constraints of VCPs in task scheduling, such as energy constraints.

Author Contributions: L.X. designed and wrote the paper; J.Q. supervised the work; L.X. performed the experiments; S.L. and X.W. analyzed the data. All authors have read and approved the final manuscript.

Funding: This work was supported by the National Social Science Foundation of China (number 15BYY028) and Dalian University of Foreign Languages Research Foundation (number 2015XJQN05).

Acknowledgments: We thank Qiang Zhang for his help in the revision of this manuscript.

Conflicts of Interest: The authors declares no conflict of interest.

References

1. Anderson, D.P. BOINC: A Platform for Volunteer Computing. *arXiv* **2019**, arXiv:1903.01699.
2. Anderson, D.P.; Cobb, J.; Korpela, E.; Lebofsky, M.; Werthimer, D. SETI@home: An experiment in public-resource computing. *Commun. ACM* **2002**, *45*, 56–61. [CrossRef]
3. Beberg, A.L.; Ensign, D.L.; Jayachandran, G.; Khaliq, S.; Pande, V.S. Folding@home: Lessons from Eight Years of Volunteer Distributed Computing. In Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Processing, Rome, Italy, 23–29 May 2009.
4. Adambourdarios, C.; Wu, W.; Cameron, D.; Lancon, E.; Filipčič, A. *ATLAS@Home: Harnessing Volunteer Computing for HEP*; IOP Publishing: Bristol, UK, 2015.
5. Jiang, J. C.; Wang, T. Q . An Operating System Architecture Design for Heterogeneous Multi-Core Processor Based on Multi-Master Model. *Adv. Mater. Res.* **2011**, *187*, 190–197. [CrossRef]
6. Filep, L. Model for Improved Load Balancing in Volunteer Computing Platforms. In Proceedings of the European, Mediterranean, and Middle Eastern Conference on Information Systems, Limassol, Cyprus, 4–5 October 2018; pp. 131–143.
7. Ostermann, S.; Iosup, A.; Yigitbasi, N.; Prodan, R.; Fahringer, T.; Epema, D. A Performance Analysis of EC2 Cloud Computing Services for Scientific Computing. In Proceedings of the International Conference on Cloud Computing, Bangalore, India, 21–25 September 2009; Springer: Berlin/Heidelberg, Germany, 2009.
8. Kondo, D.; Javadi, B.; Malecot, P.; Cappello, F.; Anderson, D. Cost-benefit analysis of Cloud Computing versus desktop grids. In Proceedings of the IEEE International Symposium on Parallel and Distributed Processing, Rome, Italy, 23–29 May 2009.
9. Gridcoin. The Computation Power of a Blockchain Driving Science and Data Analysis. 2018. Available online: https://gridcoin.us/assets/img/whitepaper.pdf (accessed on 6 November 2019).
10. Topcuoglu, H.; Hariri, S.; Wu, M.Y. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distrib. Syst.* **2002**, *13*, 260–274. [CrossRef]
11. Ding, S.; Wu, J.; Xie, G.; Zeng, G. A hybrid heuristic-genetic algorithm with adaptive parameters for static task scheduling in heterogeneous computing system. In Proceedings of the 14th IEEE International Conference on Embedded Software And Systems, Sydney, Australia, 1–4 August 2017; pp. 761–766.
12. Boveiri, H. R.; Khayami, R. Static Homogeneous Multiprocessor Task Graph Scheduling Using Ant Colony Optimization. *Ksii Trans. Internet Inform. Syst.* **2017**, *11*, 3046–3070.
13. Zhou, N.; Qi, D.; Wang, X.; Zheng, S. A static task scheduling algorithm for heterogeneous systems based on merging tasks and critical tasks. *J. Comput. Methods Sci. Eng.* **2017**, *17*, 1–18. [CrossRef]
14. Liu, T.; Liu, Y.; Song, P. DScheduler: Dynamic Network Scheduling Method for MapReduce in Distributed Controllers. In Proceedings of the IEEE International Conference on Parallel Distributed Systems, Wuhan, China, 13–16 December 2017.
15. Anderson, D.P.; McLeod, J. Local scheduling for volunteer computing. In Proceedings of the 2007 IEEE International Parallel and Distributed Processing Symposium, Rome, Italy, 26–30 March 2007; pp. 1–8.
16. Guler, H.; Cambazoglu, B.B.; Ozkasap, O. Task allocation in volunteer computing networks under monetary budget constraint. *Peer Netw. Appl.* **2015**, *8*, 938–951. [CrossRef]
17. Ghafarian, T.; Javadi, B. Cloud-aware data intensive workflow scheduling on volunteer computing systems. *Future Gener. Comput. Syst.* **2015**, *51*, 87–97. [CrossRef]
18. Miyakoshi, Y.; Yasuda, S.; Watanabe, K.; Fukushi, M.; Nogami, Y. Dynamic Job Scheduling Method Based on Expected Probability of Completion of Voting in Volunteer Computing. In Proceedings of the Second International Symposium on Computing and Networking, Hokkaido, Japan, 8–11 December 2015; pp. 2132–2140.
19. Canon, L.C.; Chang, A.K.W.; Robert, Y.; Vivien, F. Scheduling Independent Stochastic Tasks Under Deadline and Budget Constraints. In Proceedings of the International Symposium on Computer Architecture and High Performance Computing, Lyon, France, 24–27 September 2018; pp. 33–40.
20. Chuprat, S; Salleh, S. A deadline-based algorithm for dynamic task scheduling with precedence constraints. In Proceedings of the Conference on Iasted International Multi-Conference: Parallel and Distributed Computing and Networks, Innsbruck, Austria, 13–15 February 2007; ACTA Press: Calgary, AB, Canada, 2007.

21. Yin, P.Y.; Yu, S.S.; Wang, P.P.; Wang, Y.T. Task allocation for maximizing reliability of a distributed system using hybrid particle swarm optimization. *J. Syst. Softw.* **2007**, *80*, 724–735. [CrossRef]
22. Essafi, A.; Trystram, D.; Zaidi, Z. An efficient algorithm for scheduling jobs in volunteer computing platforms. In Proceedings of the Parallel Distributed Processing Symposium Workshops (IPDPSW), Phoenix, AZ, USA, 19–23 May 2014; pp. 68–76.
23. Salehi, M.A.; Smith, J.; Maciejewski, A.A. Stochastic-based robust dynamic resource allocation for independent tasks in a heterogeneous computing system. *J. Parallel Distrib. Comput.* **2016**, *97*, 96–111. [CrossRef]
24. Xu, L.; Qiao, J.; Lin, S.; Qi, R. Task Assignment Algorithm Based on Trust in Volunteer Computing Platforms. *Information* **2019**, *10*, 244. [CrossRef]
25. Kang, Q.; He, H.; Wei, J. An effective iterated greedy algorithm for reliability-oriented task allocation in distributed computing systems. *J. Parallel Distrib. Comput*. **2013**, 73, 1106–1115. [CrossRef]
26. Omara, F.A.; Arafa, M.M. Genetic algorithms for task scheduling problem. *J. Parallel Distrib. Comput.* **2010**, *70*, 13–22. [CrossRef]
27. Wu, A.S.; Yu, H.; Jin, S.; Lin, K.; Schiavone, G. An incremental genetic algorithm approach to multiprocessor scheduling. *IEEE Trans. Paral. Distrib. Syst.* **2004**, *15*, 824–834. [CrossRef]
28. Page, A.J.; Naughton, T.J. Framework for Task Scheduling in Heterogeneous Distributed Computing Using Genetic Algorithms. *Artif. Intell. Rev.* **2005**, 24, 415–429. [CrossRef]
29. Chai, S.; Li, Y.; Wang, J.; Wu, C. A List Simulated Annealing Algorithm for Task Scheduling on Network-on-Chip. *J. Comput.* **2014**, *9*, 176–182. [CrossRef]
30. Li, J.; Luo, Z.; Ferry, D.; David, F.; Agrawal, K.; Gill, D.; Lu, C. Global EDF scheduling for parallel real-time tasks. *Real Time Syst.* **2015**, *51*, 395–439. [CrossRef]
31. Zhou, N.; Qi, D.; Wang, X.; Zheng, Z.; Lin, W. A list scheduling algorithm for heterogeneous systems based on a critical node cost table and pessimistic cost table. *Concurr. Comput. Pract. Exp.* **2017**, 29, 1–11. [CrossRef]
32. Liu, W.; Li, H.; Du, W.; Shi, F. Energy-Aware Task Clustering Scheduling Algorithm for Heterogeneous Clusters. In Proceedings of the IEEE/ACM International Conference on Green Computing and Communications; ACM: New York, NY, USA, Chengdu, China, 4–5 August 2011; pp.34–37.
33. Maurya, A.K.; Tripathi, A.K. ECP: A novel clustering-based technique to schedule precedence constrained tasks on multiprocessor computing systems. *Computing* **2019**, *101*, 1015–1039. [CrossRef]
34. Kanemitsu, H.; Hanada, M.; Nakazato, H. Clustering-Based Task Scheduling in a Large Number of Heterogeneous Processors. *IEEE Trans. Parallel Distrib. Syst*. **2016**, *27*, 3144–3157. [CrossRef]
35. Tang, X.; Li, K.; Liao, G.; Li, R. List scheduling with duplication for heterogeneous computing systems. *J. Parallel Distrib. Comput*. **2010**, *70*, 323–329. [CrossRef]
36. Bansal, S.; Kumar, P.; Singh, K . An improved duplication strategy for scheduling precedence constrained graphs in multiprocessor systems. *IEEE Trans. Parallel Distrib. Syst*. **2003**, *14*, 533–544. [CrossRef]
37. Hu, M.; Luo, J.; Wang, Y.; Veeravalli, B. Adaptive Scheduling of Task Graphs with Dynamic Resilience. *IEEE Trans. Comput*. **2017**, *66*, 17–23. [CrossRef]
38. Nayak, S.K.; Padhy, S.K.; Panigrahi, S.P. A novel algorithm for dynamic task scheduling. *Future Gener. Comput. Syst.* **2012**, *28*, 709–717. [CrossRef]
39. Juarez, F.; Ejarque, J.; Badia, R.M. Dynamic energy-aware scheduling for parallel task-based application in cloud computing. *Future Gener. Comput. Syst.* **2018**, *78*, 257–271. [CrossRef]
40. Andrew, J.; Thomas, J. Dynamic Task Scheduling using Genetic Algorithms for Heterogeneous Distributed Computing. In Proceedings of the 19th International Parallel and Distributed Processing Symposium (IPDPS 2005), Denver, CO, USA, 4–8 April 2005.
41. Estrada, T.; Flores, D. A.; Taufer, M.; Teller, P. J.; Kerstens, A.; Anderson, D. P. The Effectiveness of Threshold-Based Scheduling Policies in BOINC Projects. IEEE International Conference on E-science and Grid Computing, Auckland, New Zealand, 24–27 October 2006; IEEE Computer Society: Washington, DC, USA, 2006.
42. Xu, L.; Qiao, J.; Lin, S.; Zhang, W. Dynamic Task Scheduling Algorithm with Deadline Constraint in Heterogeneous Volunteer Computing Platforms. *Future Internet* **2019**, *11*, 121. [CrossRef]
43. Sakai, T.; Fukushi, M. A Reliable Volunteer Computing System with Credibility-based Voting. *J. Inform. Process.* **2016**, *24*, 266–274. [CrossRef]

44. Bazinet, A.L.; Cummings, M.P. Subdividing Long-Running, Variable-Length Analyses Into Short, Fixed-Length BOINCWorkunits. *J. Grid. Comput.* **2016**, *14*, 1–13. [CrossRef]
45. Javadi, B.; Kondo, D.; Vincent, J.M. Discovering statistical models of availability in large distributed systems: An empirical study of seti@ home. *IEEE Trans. Parallel Distrib. Syst.* **2011**, *22*, 1896–1903. [CrossRef]