Multi-job Associated Task Scheduling Based on Task Duplication and Insertion for Cloud Computing

Yuqi Fan(✉), Lunfei Wang, Jie Chen, Zhifeng Jin, Lei Shi, and Juan Xu

School of Computer Science and Information Engineering,
Anhui Province Key Laboratory of Industry Safety and Emergency Technology,
Hefei University of Technology, Hefei 230601, Anhui, China
{yuqi.fan,2019170960,shilei,xujuan}@hfut.edu.cn,
{cxwlf,jie.chen}@mail.hfut.edu.cn

Abstract. The jobs processed in cloud computing systems may consist of multiple associated tasks which need to be executed under ordering constraints. The tasks of each job are run on different nodes, and communication is required to transfer data between nodes. The processing and communication capacities of different components have great heterogeneity. For multiple jobs, simple task scheduling policies cannot fully utilize cloud resources and hence may degrade the performance of job processing. Therefore, careful multi-job task scheduling is critical to achieve efficient job processing. The performance of existing research on associated task scheduling for multiple jobs needs to be improved. In this paper, we tackle the problem of associated task scheduling of multiple jobs with the aim to minimize jobs' makespan. We propose a task Duplication and Insertion based List Scheduling algorithm (DILS) which incorporates dynamic finish time prediction, task replication, and task insertion. The algorithm dynamically schedules the tasks based on the finish time of scheduled tasks, replicates some of the tasks on different nodes, and inserts the tasks into idle time slots to expedite successive task execution. We finally conduct experiments through simulations. Experimental results demonstrate that the proposed algorithm can effectively reduce the jobs' makespan.

Keywords: Task scheduling · Associated tasks · Job priority · Makespan

1 Introduction

Cloud computing is an increasingly essential platform for various applications, since cloud computing can achieve scalability and economy of scale. In order

This work was partly supported by the National Key Research Development Plan of China under Grant 2018YFB2000505 and the Key Research and Development Project in Anhui Province under Grant 201904a06020024.

D. Yu et al. (Eds.): WASA 2020, LNCS 12384, pp. 109–120, 2020.
https://doi.org/10.1007/978-3-030-59016-1_10

to enable on-demand resource provisioning and allocation, cloud platform is built on various hardware, i.e. computing and network components, which show great heterogeneity. Users submit jobs to the cloud platform for execution. Each job processed by the cloud is split into multiple tasks which are assigned to multiple servers to execute in a parallel and distributed manner, and hence communication is required to transfer the data between the servers so that the tasks can get the required input data.

The tasks of a job may need to be executed under ordering constraints, and the associated tasks of a job are represented by a directed acyclic graph (DAG), where each node is a task and each directed arc is the ordering constraint between two consecutive tasks [1,2]. The DAG-based jobs are widely found in some real applications such as DNA detection in genetic engineering, image recognition, climate prediction, geological exploration, etc. [3].

Extensive research has been conducted on associated task scheduling. A heuristic algorithm based on genetic algorithms and task duplication was proposed in [4]. A task duplication based scheduling algorithm was introduced in [5]. A heterogeneous scheduling algorithm with improved task priority (HISP) which calculates task priority based on standard deviation with improved magnitude as computation weight and communication weight was proposed; the algorithm adopts an entry task duplication selection policy and an idle time slots insertion-based optimizing policy [6]. A list-based scheduling algorithm called Predict Earliest Finish Time (PEFT) was proposed to introduce an optimistic cost table (OCT), which was used for task ranking and processor selection [7]. The joint problem of task assignment and scheduling considering multidimensional task diversity was modeled as a reverse auction with task owners being auctioneers; four auction schemes were designed to satisfy different application requirements [8,9]. A solution was proposed to detect and remove both short-term and long-term traffic redundancy through a two-layer redundancy elimination design [10]. A stochastic load balancing scheme was designed to provide probabilistic guarantee against the resource overloading with virtual machine migration, while minimizing the total migration overhead [11]. An algorithm was proposed to schedule tasks by calculating the priority of each task; the algorithm first processes the tasks with higher priority to meet the deadline [12]. A Heterogeneous Earliest-Finish-Time (HEFT) algorithm was proposed to select the task with the highest upward rank value at each step and assign the selected task to the processor, which minimizes earliest finish time of the task with an insertion-based approach [13]. A task scheduling mechanism based on two levels of load balance was proposed to meet the dynamic task requirements of users and improve the utilization of resources [14]. A performance effective task scheduling (PETS) algorithm was proposed to calculate the priority of each task based on task communication cost and average computation cost and select the processor with the minimum earliest finish time for each task [15].

During the execution of each job, there may be some idle time slots on the processors. For multiple jobs, simple task scheduling policies cannot fully utilize the idle resources, such that the performance of job processing may be degraded.

Therefore, careful multi-job task scheduling is critical for achieving high performance of cloud computing systems. The performance of existing research on associated task scheduling for multiple jobs needs to be improved. In this paper, we deal with the multi-priority associated task scheduling problem with the objective of minimizing the jobs' makespan, when the underlying computing and communication components of the cloud computing platform have great diversity.

The main contributions of this paper are as follows. We formulate the problem of prioritized associated task scheduling with the aim to minimize the jobs' makespan. We then propose a task Duplication and Insertion based List Scheduling (DILS) algorithm which incorporates dynamic finish time prediction, task replication, and task insertion. The algorithm dynamically predicts the remaining execution time for each task according to the scheduling of previously scheduled tasks, replicates some of the tasks on different nodes, and inserts the tasks into idle time slots to expedite task execution. We also conduct experiments through simulations. Experimental results demonstrate that the proposed algorithm can effectively reduce the jobs' makespan.

The rest of the paper is organized as follows. The problem is defined in Sect. 2. We present the proposed algorithm in Sect. 3. The performance evaluation of the proposed algorithm is given in Sect. 4. Finally, Sect. 5 concludes the paper.

2 Prioritized Associated Task Scheduling Model

Assume the finite set of jobs with different priorities to be processed is $J = \{J_1, ..., J_k, ...\}$, where J_k is the k-th DAG-based job consisting of multiple associated tasks. Job J_k is represented by a tuple $J_k =< T_k, E_k >$, where T_k is the task set of job J_k and E_k is the directed arc set $\{e^k_{i,j} | t^k_i, t^k_j \in T_k\}$. Each arc $e^k_{i,j} \in E_k$ signifies the ordering constraint of task t^k_i and task t^k_j, where t^k_i and t^k_j are the i-th task and the j-th task of job J_k, respectively. To be specific, arc $e^k_{i,j}$ signifies that task t^k_j cannot be executed until the execution of task t^k_i is completed, i.e. task t^k_j is the successive task of task t^k_i and task t^k_i is the preceding task of task t^k_j. Arc $e^k_{i,j}$ also indicates that task t^k_j requires the data generated by task t^k_i, and the data need to be transmitted to the server executing task t_j. The weight $c(t^k_i, t^k_j)$ of arc $e^k_{i,j}$ is the time for data transmission from the server running task t^k_i to the server executing task t^k_j. When the two tasks are scheduled on the same server, $c(t^k_i, t^k_j) = 0$; that is, the communication within a single server is negligible.

A DAG-based job example is shown in Fig. 1. In the DAG, task set $T_k = \{t^k_1, ..., t^k_i, ..., t^k_{10}\}$ consists of 10 tasks. The directed arc from node t^k_2 to node t^k_8 indicates task t^k_2 is the preceding task of task t^k_8, and the weight of the directed arc specifies that the communication time from task t^k_2 to task t^k_8 is 3, if the two tasks are assigned on different servers.

The execution time of a task may be different on different servers due to the diversity of servers' computing capacities. Assuming $P = \{p_1, ..., p_s...\}$ is the set of servers which will run the jobs, we use an execution time matrix

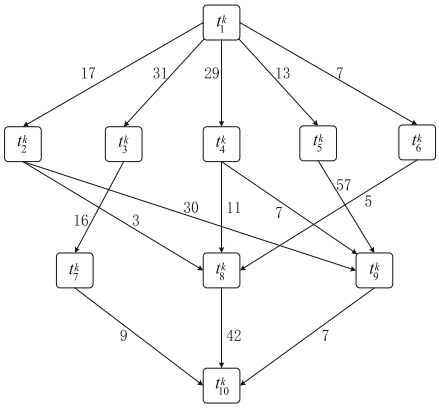


Fig. 1. DAG-based job example.

Table 1. Execution time matrix example

	p_1	p_2	p_3
t_1^k	22	21	36
t_2^k	22	18	18
t_3^k	32	27	43
t_4^k	7	10	4
t_5^k	29	27	35
t_6^k	26	17	24
t_7^k	14	25	30
t_8^k	29	23	36
t_9^k	15	21	8
t_{10}^k	13	16	33

$W_k = T_k \times P$ to list all the possible mapping between task t_i^k and server p_s. Each element $w(t_i^k, p_s)$ in matrix W_k is the execution time of server p_s running task t_i^k. An execution time matrix example for the DAG shown in Fig. 1 is described in Table 1. Assuming the server set $P = \{p_1, p_2, p_3\}$ consists of 3 servers, the element of mapping task t_2^k and server p_2 indicates that the execution time of task t_2^k on server p_2 is 18.

Definition 1. *In DAG-based job J_k, ingress task t_{in}^k is the task without any preceding tasks, and egress task t_{out}^k is the task without any successive tasks.*

For the DAG-based job shown in Fig. 1, the ingress task and egress task are tasks t_1^k and t_{10}^k, respectively. If a job includes more than one ingress task, we add a virtual task node with zero computation cost as the virtual ingress task, and add a directed arc from the virtual ingress task to each of the original ingress task nodes with zero weight.

Definition 2. *$EST(t_i^k, p_s)$, the earliest start time (EST) that task t_i^k can be executed on server p_s, is defined via Eq. (1), where $EAT(p_s)$ is the earliest time that server p_s is available, $pre(t_i^k)$ is the set of the preceding tasks of task t_i^k, and $AFT(t_j^k)$ is the actual finish time of task t_j^k.*

$$EST(t_i^k, p_s) = max\{\ EAT(p_s), max_{t_j^k \in pre(t_i^k)}\{AFT(t_j^k) + c(t_j^k, t_i^k)\}\}, \forall t_i^k \in T_k, \forall p_s \in P. \tag{1}$$

Definition 3. *$EFT(t_i^k, p_s)$, the earliest finish time (EFT) that server p_s can complete the execution of task t_i^k, is calculated via Eq. (2).*

$$EFT(t_i^k, p_s) = EST(t_i^k, p_s) + w(t_i^k, p_s), \forall t_i^k \in T, \forall p_s \in P. \tag{2}$$

Definition 4. *For job set J in which the jobs have different priorities, the makespan of job set J, $\Gamma(J)$, is the completion time of all the jobs and can be calculated via Eq.* (3).

$$\Gamma(J) = \max_{J_k \in J} AFT(t_{out}^k). \tag{3}$$

Given job set J, the DAG of each job, server set P, and the execution time of each job on each server, we need to schedule the jobs in J on the servers in P, so as to minimize $\Gamma(J)$, i.e. the makespan of job set J.

3 Multi-job Task Scheduling Algorithm

It is known that the single-job associated task scheduling problem with the aim to minimize the makespan is an NP-hard problem. Obviously, the multi-job associated task scheduling problem is also NP-hard [16].

In this section, we propose a task Duplication and Insertion based List Scheduling (DILS) algorithm which incorporates dynamic finish time prediction, task replication, and task insertion. The algorithm dynamically predicts the remaining execution time for each task according to the scheduling of previously scheduled tasks. The algorithm then schedules the task with the latest remaining time on the server which can minimize the remaining time. After each task is scheduled, the algorithm adopts task duplication and task insertion to advance the start time of this task. The remaining execution time of each to-be-scheduled task is updated upon the scheduling of a task. Algorithm DILS shown in Algorithm 1 consists of five components: task remaining time calculation, selection of the task to be scheduled, server allocation for the task to be scheduled, task duplication, and task insertion.

Algorithm 1. *Algorithm DILS*

Input: Server set P, job set J with each job's DAG, and execution time matrices
Output: Makespan of the jobs

```
for each job J_k ∈ J do
    Calculate the predicted remaining time of each task via Eq. (6) and create the
    Prediction of Remaining Time (PRT) table;
    Create an empty ready task list and add the ingress task to the list;
    while ready task list is not empty do
        for each task t_i^k in ready task list do
            Compute the average path length of task t_i via Eq. (8);
        end for
        Select the task with the maximum average path length with Eq. (9);
        Assign the server leading to the minimum estimated path length via Eq. (10)
        to the task;
        Task duplication and task insertion;
        Update the ready task list;
    end while
end for
return the completion time of the last scheduled task.
```

3.1 Task Remaining Time Calculation

Each task is assigned a weight which is the total computation and communication time of all the subsequent tasks. The weights of all the tasks are maintained in a Predicted Remaining Time (PRT) table, and each element $PRT(t_i^k, p_s)$ in table PRT is the predicted remaining time required for executing all the subsequent tasks of task t_i^k $(1 \leq i \leq N)$, if it is allocated to server $p_s \in P$. The weight of task t_i^k is closely related to its successive tasks and the number of servers, and we let

$$A_{i,s}^k = max_{t_j^k \in suc(t_i^k)}\{min_{p_t \in p}\{\ PRT(t_j^k, p_t) + w(t_j^k, p_t) + c(t_i^k, t_j^k)\}\}, \tag{4}$$

where $suc(t_i^k)$ is the set of successive task of task t_i^k, and let

$$B_i^k = \frac{\sum_{t_j^k \in suc(t_i^k)} \frac{\sum_{p_t \in P} PRT(t_j^k, p_t)}{M}}{M}. \tag{5}$$

Note that the weight of the egress task of each job is 0, no matter which server runs the egress task. That is, for any $p_t \in P$, $PRT(t_{out}^k, p_t) = 0$.

$$PRT(t_i^k, p_s) = max\{A_{i,s}^k, B_i^k\}, \forall t_i^k \in T, \forall p_s \in P. \tag{6}$$

The weight of task t_i^k is calculated via Eq. 6. Starting from the egress task to the ingress task in each DAG, algorithm DILS recursively calculates backwards the weight of each task on each server, and obtains the PRT table.

3.2 Selection of the Task to Be Scheduled

We call task t_i^k is *ready*, when all the preceding tasks of task t_i^k are scheduled. We create a ready task list (RTL) to maintain all the tasks which are ready. Initially, only the ingress task of each DAG-based job is ready, and hence the ready task list contains only one task, t_{in}^k. We calculate the EST of each task which is in the ready task list. The EST of task t_i^k on server p_s, $EST(t_i^k, p_s)$, is calculated via Eq. (1). The estimated path length (EPL) when task t_i^k is assigned to server p_s, $EPL(t_i^k, p_s)$, can be calculated via Eq. (7).

$$EPL(t_i^k, p_s) = EST(t_i^k, p_s) + w(t_i^k, p_s) + PRT(t_i^k, p_s), \forall t_i^k \in T_k, \forall p_s \in P. \tag{7}$$

Each task may be placed on each $p_s \in P$. For task t_i^k which is ready, $APL(t_i^k)$, the average path length (APL) of task t_i^k, is calculated with Eq. (8).

$$APL(t_i^k) = \frac{\sum_{p_s \in P} EPL(t_i^k, p_s)}{M}, \forall t_i^k \in T_k. \tag{8}$$

We select task t_i^k which is ready and has the maximum average path length as the task to be scheduled by considering the difference of the task execution time on different paths; that is,

$$t_i^k = argmax_{t_j^k \in RTL}\{APL(t_j^k)\}. \tag{9}$$

The selection is related to the task's EST which dynamically changes with the previous scheduling result. Therefore, the selection of to-be-scheduled task is dynamically changed during the associated task scheduling process.

3.3 Allocation of the Server for the Task to Be Scheduled

We allocate to-be-scheduled task t_i^k to server p_s which leads to the minimum estimated path length to reduce the makespan, i.e.

$$p_s = argmin_{p_t \in P}\{EPL(t_i^k, p_t)\}. \tag{10}$$

If multiple servers achieve the same minimum estimated path length for task t_i^k, we randomly assign task t_i^k to one of the servers. The actual start time of task t_i^k is calculated by Eq. (2). After the server allocation for the task to be scheduled is completed, some other tasks may be ready to scheduled, and hence we update the ready task list.

3.4 Task Duplication

Task duplication makes multiple copies of task t_i^k and assigns the task copies on different processors to reduce the data transmission time between tasks. In this way, the direct successive tasks of task t_i^k can be started right after the execution of t_i^k, without waiting for the data generated by task t_i^k to be transmitted through network. Task duplication works as follows.

(1) Assign each task t_j^k in $pre(t_i^k)$ a time weight τ_j^k which is the time when the data generated by the preceding task of task t_i^k are sent to processor p_s where task t_i^k is to be executed.
(2) Sort all the preceding tasks in $pre(t_i^k)$ according to non-ascending order of the time weights.
(3) Process each of the preceding tasks in $pre(t_i^k)$ iteratively. Schedule a copy of task $t_j^k \in pre(t_i^k)$ in the earliest idle time slot (ITS) on processor p_s, if the following conditions are satisfied: (I) $\tau_j^k > EAT(p_s)$, that is, τ_j^k is greater than the earliest available time of processor p_s; (II) there is an idle time slot for the copy of t_j^k; (III) task t_i^k can start earlier by duplicating t_j^k.

3.5 Task Insertion

Task insertion inserts a task into an idle time slot on a processor which is occupied by some tasks after the time slot. Task insertion works as follows.

1) When task t_i^k is to be scheduled on processor p_s, we search all idle time slots on processor p_s, and the idle time slot chosen to insert task t_i^k meets the following conditions: (I) $EST(t_i^k, p_s)$, the earliest start time of task t_i^k, is no ealier than the start time of the idle time slot; (II) $EFT(t_i^k, p_s)$, the earliest finish time of task t_i^k on processor p_s, is no later than the end time of the idle time slot;

2) When multiple idle time slots meet the conditions above, we select the ITS with the smallest difference between the length of the ITS and the execution time of task t_i^k.

4 Simulation

In this section, we evaluate the performance of the proposed algorithm DILS against two state-of-the-art algorithms PEFT [7] and HSIP [6]. We also investigate the impact of important parameters on the performance of algorithm DILS.

The scheduling results of the three algorithms for job set $J_s = <J_1, J_2>$ are illustrated in Fig. 2, where DAGs J_1 and J_2 are the same as the DAG shown in Fig. 1 and Table 1. The gray shadowed block represents the task that algorithms DILS and HSIP choose to replicate. We can see that algorithm DILS achieves better results than algorithms HSIP and PEFT in this example.

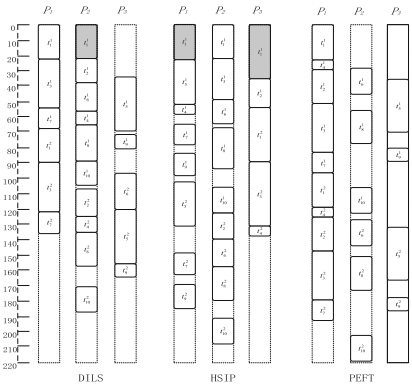


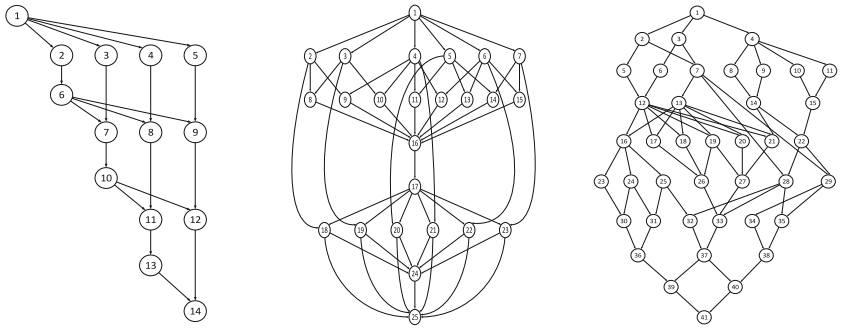
Fig. 2. The example of scheduling result

4.1 Simulation Setup

In this paper, we evaluate the performance of the three algorithms in terms of Schedule Length Ratio (SLR) which is the ratio of the scheduling length to the minimum scheduling length by ignoring the communication time as defined via Eq. (11).

$$SLR = \frac{\sum_{J_k \in J} makespan}{\sum_{J_k \in J} \sum_{t_i^k \in CP_{MIN}} min_{p_s \in P} \left\{ w\left(t_i^k, p_s\right) \right\}} \tag{11}$$

where CP_{MIN} is the minimum length of the critical path in the DAG-based job after ignoring the communication time between tasks, and the critical path of a DAG-based job is the longest path from its ingress task to its egress task.



(a) Gaussian Elimination. (b) Montage Workflow. (c) Molecular Dynamics.

Fig. 3. Real-world DAGs

We use both DAG-based jobs randomly generated by a DAG generator and real-world DAG-based jobs in the simulations, where the parameters used by the generator are consistent with those in [17,18]. The DAG topology parameters are as follows:

(1) DAG average calculation time $\overline{c_{DAG}}$: indicating the average execution time of the tasks in the DAG, which is randomly set during the simulation.
(2) Communication Calculation Ratio CCR: the ratio of the average communication time and the average execution time; the larger the value, the more communication-intensive the DAG-based job; the smaller the value, the more computation-intensive the DAG-based job.
(3) Heterogeneous parameter α: representing the task execution time range on different processors; the larger the value, the more heterogeneous the processors.

We select three classic DAG-based jobs from the real-world applications in the simulations.

(1) Gaussian Elimination: it is used in linear algebraic programming for solving linear equations as shown in Fig. 3(a). The number of nodes is $N = \frac{\beta^2+\beta-2}{2}$ according to matrix parameter β.
(2) Montage Workflow: it is applied to construct astronomical image mosaic. An example of Montage Workflow is shown in Fig. 3(b).
(3) Molecular Dynamics Code: it is an algorithm to implement the atomic and the molecular physical motion. An example of Molecular Dynamics Code is depicted in Fig. 3(c).

4.2 Performance of Algorithm DILS and Impact of Parameters

Figure 4 shows the makespan performance of the three algorithms by varying the number of DAG-based jobs, when the number of processors is set as 4, the DAG-based jobs are randomly generated by the generator, and the parameters CCR

and α are both set as 1. It can be seen that the makespan increases as the number of DAG-based jobs increases. In general, algorithm DILS always achieves the best performance among the three algorithms, and algorithm HSIP performs better than algorithm PEFT. With 10 DAG-based jobs, the performance improvement of algorithm DILS on algorithms HSIP and PEFT reaches up to 4.5% and 7.9%, respectively.

Figure 5 depicts the average SLR performance with different number of DAG-based jobs, when the number of processors is 8, the DAG-based jobs are randomly generated by the generator, and the parameters CCR and α are both set as 1. The average SLR generated by the three algorithms of DILS, HSIP and PEFT decreases with the increase of the number of DAG-based jobs, since more idle time slots are utilized. Algorithm DILS leads to the least average SLR among the three algorithms since algorithm DILS can make the best use of idle time slots. Algorithm DILS outperforms algorithms HSIP and PEFT from 16.3% to 17.6% and from 17.9% to 25.0%, respectively, when the number of DAG-based jobs increases from 2 to 10.

Figure 6 illustrates the average SLR performance versus different matrix size parameter β for the Gaussian Elimination jobs when the number of processors is 2. It can be observed that the average SLR of the three algorithms increases with the increase of β. There are more tasks in the jobs with a larger β than that with a smaller β, which requires more computation and communication time to execute all the associated tasks. The performance improvement of algorithm DILS on algorithms HSIP and PEFT is up to 11.9% and 12.1%, respectively.

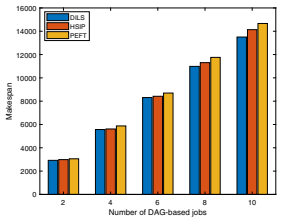


Fig. 4. The makespan with the different number of DAG-based jobs

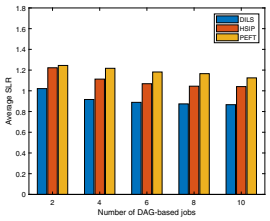


Fig. 5. The average SLR with the different number of DAG-based jobs

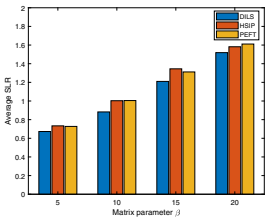


Fig. 6. The average SLR with the different matrix parameter β

Table 2 describes the average SLR for Montage Workflow and Molecular Dynamics Code. Parameter α varies in $\{01, 0.5, 1, 2\}$, when parameter $CCR = 1$ and the numbers of DAG-based jobs and processors are 5 and 4, respectively. Parameter CCR increases from 0.2 to 10, when $\alpha = 1$ and the number of DAG-based jobs and processors is 5 and 8, respectively. The average SLR increases with the increase of CCR, since the communication between tasks consumes more time as parameter CCR increases. Algorithm DILS always achieves the best performance in all the three algorithms.

Table 2. Average SLR with Montage Workflow and Molecular Dynamics Code

DAG type	Algorithm	Parameter α				Parameter CCR					
		0.1	0.5	1	2	0.2	0.5	1	2	5	10
Montage Workflow	DILS	1.00	1.04	1.12	1.15	0.53	0.58	0.61	0.62	0.67	0.93
	HSIP	1.22	1.33	1.43	1.50	0.74	0.75	0.84	0.87	0.92	1.13
	PEFT	1.25	1.36	1.38	1.48	0.73	0.75	0.83	0.84	0.98	1.22
Molecular Dynamics Code	DILS	1.25	1.36	1.46	1.53	0.66	0.70	0.75	0.80	0.87	1.01
	HSIP	1.31	1.45	1.56	1.67	0.73	0.79	0.87	1.03	1.21	1.33
	PEFT	1.29	1.44	1.62	1.61	0.74	0.76	0.92	1.06	1.31	1.58

5 Conclusion

Careful multi-job task scheduling is critical to achieve efficient job processing. In this paper, we studied the problem of associated task scheduling of multiple jobs with the aim to minimize jobs' makespan. We proposed a task Duplication and Insertion based List Scheduling algorithm (DILS) which incorporates dynamic finish time prediction, task replication, and task insertion. The algorithm dynamically schedules the tasks based on the finish time of scheduled tasks, replicates some of the tasks on different nodes, and inserts the tasks into idle time slots to expedite successive task execution. The simulation results demonstrate that the proposed algorithm can effectively reduce the jobs' makespan.

References

1. Chen, W., Xie, G., Li, R., Bai, Y., Fan, C., Li, K.: Efficient task scheduling for budget constrained parallel applications on heterogeneous cloud computing systems. Future Gener. Comput. Syst. **74**(C), 1–11 (2017)
2. Arabnejad, H., Barbosa, J.: Fairness resource sharing for dynamic workflow scheduling on heterogeneous systems. In: 2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications (ISPA), Leganes, Spain, 10–13 July 2012, pp. 633–639 (2012)
3. Panda, S.K., Jana, P.K.: Efficient task scheduling algorithms for heterogeneous multi-cloud environment. J. Supercomput. **71**(4), 1505–1533 (2015). https://doi.org/10.1007/s11227-014-1376-6
4. Tsuchiya, T., Osada, T., Kikuno, T.: A new heuristic algorithm based on GAs for multiprocessor scheduling with task duplication. In: Proceedings of 3rd International Conference on Algorithms and Architectures for Parallel Processing, pp. 295–308. IEEE (1997)
5. Bajaj, R., Agrawal, D.P.: Improving scheduling of tasks in a heterogeneous environment. IEEE Trans. Parallel Distrib. Syst. **15**(2), 107–118 (2004)
6. Wang, G., Wang, Y., Liu, H., Guo, H.: HSIP: a novel task scheduling algorithm for heterogeneous computing. Sci. Programm. **2016**, 1–11 (2016)
7. Hamid, A., Barbosa, J.G.: List scheduling algorithm for heterogeneous systems by an optimistic cost table. IEEE Trans. Parallel Distrib. Syst. **25**(3), 682–694 (2014)
8. Duan, Z., Li, W., Cai, Z.: Distributed auctions for task assignment and scheduling in mobile crowdsensing systems. In: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), pp. 635–644 (2017)

9. Cai, Z., Duan, Z., Li, W.: Exploiting multi-dimensional task diversity in distributed auctions for mobile crowdsensing. IEEE Trans. Mob. Comput. (2020)
10. Yu, L., Shen, H., Sapra, K., Ye, L., Cai, Z.: CoRE: cooperative end-to-end traffic redundancy elimination for reducing cloud bandwidth cost. IEEE Trans. Parallel Distrib. Syst. **28**(2), 446–461 (2017)
11. Yu, L., Chen, L., Cai, Z., Shen, H., Liang, Y., Pan, Y.: Stochastic load balancing for virtual resource management in datacenters. IEEE Trans. Cloud Comput. **8**(2), 459–472 (2020)
12. Choudhari, T., Moh, M., Moh, T.-S.: Prioritized task scheduling in fog computing. In: Proceedings of the ACMSE 2018 Conference, pp. 1–8 (2018)
13. Topcuoglu, H., Hariri, S., Wu, M.-Y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Trans. Parallel Distrib. Syst. **13**(3), 260–274 (2002)
14. Fang, Y., Wang, F., Ge, J.: A task scheduling algorithm based on load balancing in cloud computing. In: Wang, F.L., Gong, Z., Luo, X., Lei, J. (eds.) WISM 2010. LNCS, vol. 6318, pp. 271–277. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16515-3_34
15. Ilavarasan, E., Thambidurai, P., Mahilmannan, R.: Performance effective task scheduling algorithm for heterogeneous computing system. In: 4th International Symposium on Parallel and Distributed Computing (ISPDC 2005), Lille, France, 4–6 July 2005, pp. 28–38 (2005)
16. Ullman, J.D.: NP-complete scheduling problems. J. Comput. Syst. Sci. **10**(6), 384–393 (1975)
17. Cordeiro, D., Mounié, G., Swann, P., Trystram, D., Vincent, J.-M., Wagner, F.: Random graph generation for scheduling simulations. In: Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques (SIMUTools 2010), Torremolinos, Malaga, Spain, 15–19 March 2010 (2010)
18. Fan, Y., Tao, L., Chen, J.: Associated task scheduling based on dynamic finish time prediction for cloud computing. In: The 39th IEEE International Conference on Distributed Computing Systems (ICDCS 2019), Dallas, Texas, USA, 7–10 July 2019 (2019)