Joint Online Optimization of Task Rescheduling and Data Redistribution

Yao Song[1,2], Limin Xiao[1,2], Liang Wang[1,2], Wei Wei[3], Jinquan Wang[1,2]*

[1] State Key Laboratory of Software Development Environment, Beihang University, China
[2] School of Computer Science and Engineering, Beihang University, China
[3] School of Computer Science and Engineering, Xi'an University of Technology, China
songyao@buaa.edu.cn, xiaolm@buaa.edu.cn, lwang20@buaa.edu.cn, weiwei@xaut.edu.cn, derekjqwang@buaa.edu.cn

Abstract

Wide-area distributed computing environment is the main platform for storing large amounts of data and conducting wide-area computing. Tasks and data are jointly scheduled among multiple computing platforms to improve system efficiency. However, large network latency and limited bandwidth in wide-area networks may cause a large delay in scheduling information and data migration, which brings low task execution efficiency and a long time waiting for data. Traditional works mainly focus on allocating tasks based on data locality or distributing data replications, but optimizing task allocation or data placement alone is insufficient from a global perspective. To mitigate the impact of large network latency and limited bandwidth on system performance, joint online optimization of task rescheduling and data redistribution is proposed in this study. The task allocation and data placement can be adjusted collaboratively during the system running process through the task stealing and backfilling mechanism and the data replication placement mechanism. The experimental results indicate that compared with the state-of-the-art method, the proposed method improves the system throughput and computing resource utilization by 20.67% and 20.26% respectively, and can significantly reduce the global data migration costs.

Keywords: Distributed computing, Joint scheduling, Task rescheduling, Data redistribution

1 Introduction

With the increase in data volume and computing demands in various fields, companies and organizations have established and organized large computing platforms such as data centers and supercomputing centers to build wide-area distributed computing environments and jointly process widely distributed data sets efficiently [1]. In the wide-area distributed computing environments such as XSEDE [2], EGI [3], CNGrid [4], and multi-cloud systems [5], tasks and data are jointly scheduled for better efficiency and user experience [6]. However, the large network latency and limited bandwidth in the wide-area network (WAN) may cause a large delay in scheduling information and data migration. The inefficient scheduling schemes bring low task execution efficiency and a long time waiting for data. To improve system throughput, designing an online method to adjust and optimize the distribution of tasks and data during the system running process is necessary.

The existing optimization methods during the system running process can be classified into task rescheduling methods [7-10] and data redistribution methods [6, 11-14]. The task rescheduling methods tend to optimize the utilization of computing resources by finding idle resources during the system running process and enabling smaller tasks to run in advance through the backfilling mechanism. However, the data placement information is seldom considered in the task rescheduling methods, and the advantages of data replication in multiple data centers are not fully utilized [15], resulting in limited resource utilization improvement and poor optimization effect. The data redistribution methods can reduce the data migration cost through data redundancy placement based on the data popularity and storage resource capacity. However, the neglect of the computing resource state may lead to unbalanced data placement and task allocation in wide-area environments, and reduce the computing efficiency. In recent years, there have been several efforts to accomplish task rescheduling or data redistribution through machine learning and heuristic algorithms, but again, they do not consider the relationship between tasks and data [16-19]. To summarize, the aforementioned optimization methods mainly focus on optimization through one aspect of task rescheduling or data redistribution rather than on a comprehensive approach, which cannot meet the demands of global performance optimization. Therefore, they are not well applicable for improving throughput during the system running process in the wide-area distributed computing environment.

In this paper, we propose a Joint Online optimization method of Task rescheduling and Data redistribution (JOTD). The task allocation and data placement are adjusted collaboratively based on system feedback to optimize the global resource utilization, thus improving the system throughput. Compared with the existing methods, the proposed method adjusts the task allocation and data placement collaboratively, avoiding the problem of incomplete optimization in the aforementioned methods. The JOTD jointly combined the task stealing and backfilling mechanism and the data replication placement adjustment mechanism, improving the system throughput which is influenced by the delayed scheduling

*Corresponding Author: Liang Wang; E-mail: lwang20@buaa.edu.cn
DOI: 10.53106/160792642023012401002

information and data migration.

The main contributions of this study are summarized as follows:

- A Joint online optimization method of task rescheduling and data redistribution is proposed, which can effectively mitigate the large delay in scheduling information and data migration caused by the large network latency and limited bandwidth in WAN.
- A locality-aware wide-area task stealing and backfilling mechanism is applied to optimize the utilization of computing resources by perceiving data distribution characteristics and resource idle state.
- A popularity- and cost-aware adaptive data replication placement adjustment mechanism is applied to optimize the utilization of storage resources by sensing the data access popularity and data replication costs.
- The task allocation and data placement are adjusted collaboratively based on the system feedback to optimize system performance.
- Through the evaluation of using real workload traces, the proposed method can significantly improve the system throughput compared to other optimization methods.

2 Related Works

To mitigate the system performance degradation caused by limited network bandwidth and large latency in the WAN environment, optimizing task allocation and data placement during the system running process is an effective approach. The goal of existing optimization methods is to achieve reasonable task allocation to improve computing resource utilization through task rescheduling methods such as task backfilling, or to optimize the data placement based on system feedback information to reduce data migration cost and improve the execution process.

Task rescheduling methods have been widely used in job management systems such as SLURM [20], TORQUE [21], and PBSpro [22] to improve execution efficiency. Moreover, in the wide-area distributed computing environment, it is particularly important to optimize task allocation through task backfilling to solve the problem of low efficiency caused by large scheduling information delays. Several task rescheduling methods have been proposed in the past few decades [7-10, 23-34]. In [7], basic strategies such as FCFS and SPTF are integrated for task backfilling, scheduling system chooses different rules for reordering and backfilling tasks in the waiting queue to optimize computing resource utilization in computing platforms. To obtain a more accurate task backfilling strategy to further optimize the computing resource utilization, Carastan-Santos [8] adopts a dynamic rescheduling method based on simulation and machine learning. The users' history information in the system is collected, and the task characteristics are analyzed through a nonlinear model to predict the behavior of tasks. Thus, the task allocation can be optimized more precisely with the proposed method. In [16], authors propose GARLSched, which uses reinforcement learning to take several task informations into account, and can be optimized for different workloads. In [17], authors propose an efficient running time prediction model, referred to as online learning and KNN-based predictor with correction mechanism, called OKCM. In [18], authors propose RLSchert, a job scheduler based on deep reinforcement learning and remaining runtime prediction, which estimates the state of the system by using a dynamic job remaining runtime predictor and learns the best policy to select or kill jobs based on the state by imitation learning and approximate policy optimization algorithms. However, in the wide-area distributed computing environment, data is often replicated among multiple computing platforms to improve reliability and access performance, while most task backfilling methods tend to ignore the performance improvement that can be brought by using data replication placement [15, 24]. Therefore, although task rescheduling methods can optimize the computing resource utilization to some extent, the improvement of system performance is still limited because the data replication placement is not fully optimized and utilized.

The problem of large data migration costs in WAN environments has always been a hot topic for researchers. Data migration is often optimized through bandwidth allocation, load balancing, and other data transmission mechanisms [35-36]. Data replication placement is also a common technique to improve data reliability and access performance in storage systems [6, 11-14, 35-40]. Kosar [41] designed the STORK storage resource management system to improve system performance in large-scale data processing, and the efficient access of storage resources is realized through the methods of data replication placement and data transmission management. In [6] and [13], the data placement method based on data popularity feedback is proposed. The popularity of data blocks, storage size required by data, and storage resource capacity are considered comprehensively to optimize and adjust the data placement in the system, and then task scheduling is carried out based on the data placement information. In [11], four data replication policy is aggregated to support efficient data access, and data replication placement is adjusted dynamically and adaptively based on reliability, cost, and popularity, optimizing storage resource utilizing and reducing data migration cost. However, the data replication placement methods usually only focus on the capacity and load of storage resources, as well as data popularity, and pays little attention to the state of computing resources. As a result, the subsequent task scheduling based on data placement information will cause unbalanced task allocation in the system, reducing system throughput. In [19], the authors propose a metaheuristic approach using a non-dominated sorting genetic algorithm II in order to address the latency and traffic differences between different hardware nodes and provide an automated method to manage the transmission of data copies, including their deployment in a fog cloud environment.

To summarize, the existing optimization methods during the system running process mainly focus on optimization through one aspect of task rescheduling or data redistribution rather than on a comprehensive approach, which cannot meet the demands of global performance optimization. Therefore, this paper proposes a joint online optimization method for task rescheduling and data redistribution. Compared with the existing methods, the proposed method adjusts the task allocation and data placement in the system running process based on the collaborative application of task rescheduling

and data redistribution mechanisms, avoiding the problem of incomplete optimization in the aforementioned methods, and effectively improving the system performance.

3 Joint Online Optimization

JOTD is an optimization method during the system running process based on dynamic task rescheduling and data redistribution. The locality-aware wide-area task stealing and backfilling mechanism is applied in JOTD to modify the inefficient scheduling schemes caused by the delay of scheduling information. On the premise of making full use of data locality, the appropriate tasks are selected by considering information such as task requirements and priority, and the task stealing and backfilling mechanism among task queues in a wide-area is carried out to optimize the utilization of computing resources. Moreover, the popularity- and cost-aware adaptive data replication placement adjustment mechanism is applied to mitigate the long data waiting time caused by delayed data migration. Based on the feedback on the data replication cost and the data popularity, the evaluation model of the data replication scheme is established, and the data replication placement in a wide-area is adjusted adaptively to optimize the utilization of storage resources. The framework of JOTD is shown in Figure 1.

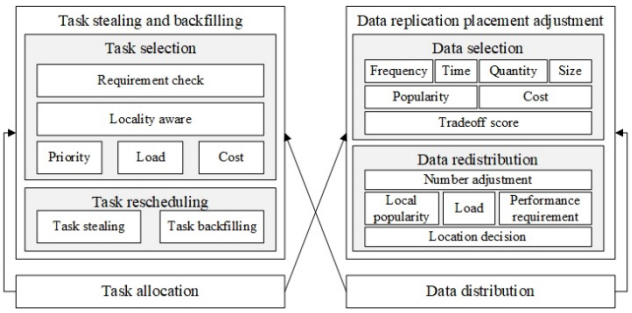

Figure 1. JOTD framework

The task allocation and data placement are adjusted collaboratively in JOTD. In terms of information coordination, the two mechanisms share the resource information, the data placement information, and the task allocation information, and make decisions based on this shared information. In terms of decision coordination, the adjustment of task allocation will affect the data access popularity in the data replication placement adjustment mechanism and then affect the data redistribution. Besides, the adjustment of data placement will affect the data locality and migration cost in the task stealing and backfilling mechanism, and then affect the task rescheduling. JOTD avoids the problem of a single optimization direction in the aforementioned optimization methods, and can effectively optimize the global resource utilization during the system running process, thus improving the system throughput.

3.1 Locality-aware Wide-area Task Stealing and Backfilling Mechanism

Due to the delay in scheduling information in the WAN environment, the scheduling scheme may not be accurate, and the time costs such as task execution time, data migration time, and queue waiting time may differ from the actual situation. To avoid the unbalanced task allocation caused by the delayed scheduling information, we propose a task stealing and backfilling mechanism based on data locality and resource state feedback in the wide-area distributed computing environment. When a computing platform has idle resource feedback, the task allocation is adjusted among multiple task queues in the wide-area to improve the execution efficiency, thus improving the system throughput. The overview of the locality-aware wide-area task stealing and backfilling mechanism is shown in Figure 2.

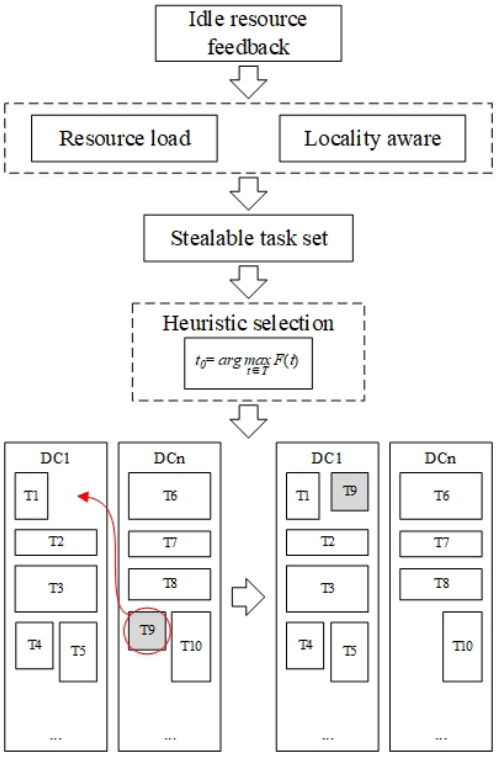

Figure 2. Locality-aware wide-area task stealing and backfilling mechanism

3.1.1 Task Selection

When receiving feedback from the system, informing the computing platform j has idle resources, the mechanism selects tasks that are queuing in other computing platforms to generate a task set by sensing data locality and global resource load. All tasks whose requirements can be met by computing platform j, such as its computing capacity requirement and application deployment, are set as a task set T_{all}. In T_{all}, all tasks whose required data replication are stored in platform j are set as a task set T_{data}. Once a task in T_{data} is stolen and backfilled, it can be executed immediately without going through the data migration process and queue waiting process. Other tasks in T_{data} are set as a task set T_{comp}, which means $T_{comp} = T_{all} - T_{data}$. Tasks in T_{comp} need to undergo a data migration process after stealing and backfilling, so the task allocation can be adjusted with some data migration costs.

Since we need to select a suitable task for stealing in a short time, task selection set T is defined from T_{all} to narrow

the search domain. First, let $T = T_{data}$, and redefine $T = T_{comp}$ only when none of the tasks in T meet the filtering criteria. That is, T_{data} is an empty set or the target platform queue will be affected if any task in T is stolen. To select a more suitable task for stealing, we select the task with a higher priority in the computing platform with a higher load, thus achieving a higher resource utilization and balancing the load among platforms. Besides, the data replication required by the selected task should be afore stored in platform j to avoid the data migration cost, reducing the network load and ensuring the timeliness of task stealing and backfilling. For a task $t \in T$, its stealing adaptability parameter is:

$$F(t) = \begin{cases} \dfrac{priority}{|\vec{R}_i|}, T = T_{data} \\ \dfrac{priority}{|\vec{R}_i| \times T_{trans,t}}, T = T_{comp} \end{cases} . \quad (1)$$

In Eq. (1), *priority* indicates the priority of tasks, which is defined based on the computing quantity of the task. In this case, smaller tasks are preferentially selected to make full use of idle resources and reduce the average waiting time in the system. $|\vec{R}_i|$ represents the idle state of the storage and computing resources in the computing platform i which holds task t. The larger the idle state is, the less likely the task t is to be stolen.

$$|\vec{R}_i| = \left| \frac{Stor_{remain,i}}{Stor_{all,i}}, \frac{Comp_{avail,i}}{Comp_{all,i}} \right| . \quad (2)$$

Besides, when $T = T_{comp}$, data migration time $T_{trans,t}$ should be added to the filtering condition. When the data migration cost is higher, the task is less likely to be stolen, thus avoiding a large amount of additional data migration.

In this mechanism, a mountain climbing algorithm is used to find the sub-optimal solution in the task set T to improve the search efficiency. In addition, to ensure the effectiveness of task stealing and backfill mechanism, filtering conditions should be set as follows:

$$t_0 = \arg\max_{t \in T} F(t) . \quad (3)$$

$$T_{exec,t_0} < T_{wait,t} . \quad (4)$$

$$T_{trans,t_0} < T_{wait,t_0} . \quad (5)$$

Task t_0 represents the stealing target and task t represents a task in the target idle computing platform. The filtering condition in Eq. (4) ensures that the execution time of the stolen task on target platform j is shorter than the expected waiting time of any task t' in platform j. In this case, when the stolen task is backfilled, the waiting time for tasks on the target platform will not be prolonged. Besides, it is necessary to ensure that the data migration cost of the stolen task is less than its original queue waiting time. Since T_{trans,t_0} of the tasks in T_{data} is 0, this condition applies to the tasks in T_{comp}.

3.1.2 Task Rescheduling

After a task t_0 is selected, the resource information of platform j is locked, the selected task t_0 is added to the task queue of the platform j, and task t_0 is deleted from the original task queue. If $t_0 \in T_{comp}$, the data required by t_0 will be migrated. The data may have multiple replications in the system, and the scheme with the lowest data migration cost should be selected. For platform j, although its task queue may not be empty, the idle resources on it cannot meet the needs of any task in the queue. Therefore, the task rescheduled to the platform will be executed in advance without affecting the queuing of the existing tasks, thus effectively improving the utilization of computing resources. For the selected task t_0, the execution of the task will be advanced regardless of whether the required data replication is stored in platform j, thus improving the system efficiency.

The locality-aware wide-area task stealing and backfilling mechanism is shown in Algorithm 1.

Algorithm 1. Locality-aware wide-area task stealing and backfilling

Input: Task collection T_{all}
Output: Stolen task t_0

```
for all i in computing platforms do
  for all t in T_all do
    // note the distinction of data distribution
    calculate F(t) by Eq. (1);
  end for
end for
F(t)_max=0;
for all t in T_all do
  if F(t) < F(t)_max then
    continue;
  else if T_exec,t < T_wait and T_trans,t < T_wait,t then
t_0=t;
F(t)_max= F(t_0);
  end if
end for
```

The value of $F(t)$ is first calculated for each platform with each task, and it is important to note whether required data exists in the current platform. Then a local optimal solution is obtained using the hill climbing algorithm. The task of this local optimal solution needs to satisfy the condition that the task execution time is less than the waiting time of every tasks in the backfill queue to avoid prolonging the tasks.

To summarize, when there are idle resources and small tasks in the system, the locality-aware wide-area task stealing and backfilling mechanism can optimize the task allocation, thus improving the execution efficiency and the system throughput.

3.2 Popularity- and Cost-aware Adaptive Data Replication Placement Adjustment Mechanism

Due to the high latency and low bandwidth of WAN, data migration may be unstable and slow, resulting in tasks long time waiting for data. Data replication placement is a common method to improve data access performance in this case. Replication and placement of hot data can effectively aggregate WAN bandwidth, however, storage and network resources occupied by multiple replications may degrade system performance to some extent. Therefore, the popularity- and cost-aware adaptive data replication placement adjustment mechanism is proposed. The number of data replications in the wide-area distributed computing environment is adjusted based on the tradeoff of data popularity and replication cost. Moreover, the locations of replications are adjusted based on the state of global computing resources to avoid an unbalanced load. Thus, data placement can be adjusted adaptively to optimize the utilization of storage resources, reducing data migration costs and improving system throughput. The overview of the popularity- and cost-aware adaptive data replication placement adjustment mechanism is shown in Figure 3.

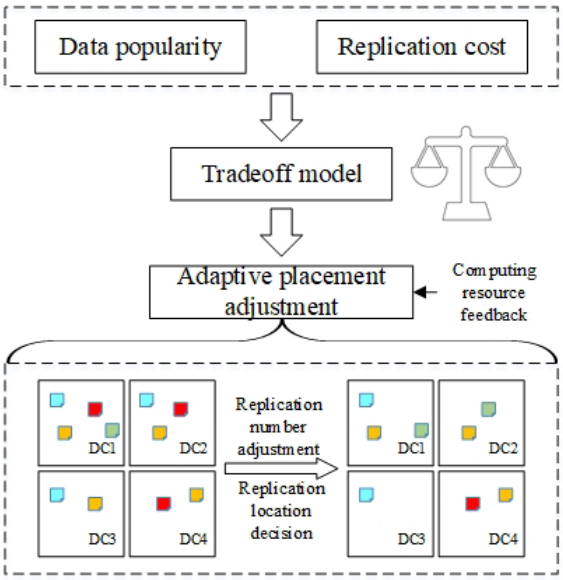

Figure 3. Popularity- and cost-aware adaptive data replication placement adjustment mechanism

3.2.1 Data Selection

The replication mechanism collects information on the data popularity and replication cost in the system. When storage resources in the system are idle, the replication mechanism periodically selects data whose replication placement needs to be adjusted based on the system feedback. Firstly, for data d, according to the collected data access information, the data popularity is calculated and feedback to the system. Data popularity is divided into global data popularity P_d and local data popularity $p_{i,d}$ in platform i. Data popularity is defined as:

$$P_d = F_d \times \left(1 - \frac{\Delta t_d - \Delta t_{\min,d}}{\Delta t_{\max,d} - \Delta t_{\min,d}}\right). \tag{6}$$

$$p_{i,d} = f_{i,d} \times \left(1 - \frac{\Delta t_{i,d} - \Delta t_{\min,i,d}}{\Delta t_{\max,i,d} - \Delta t_{\min,i,d}}\right). \tag{7}$$

In the definition, Data access ratio F_d is the ratio of the access times of data d to global access times of all data, and $f_{i,d}$ is the ratio of the access times of data d in platform i to global access times of data d. Meanwhile, the data access time interval Δt is max-min normalized. When the data access time interval is smaller and the access ratio is higher, it indicates that the data is more popular.

Although data replication can effectively improve data reliability and access performance, however, the maintenance of multiple replications will cause network and storage costs, which are defined as replication cost C_d. Replication cost mainly consists of the replication storage cost, the data migration cost during replication creation, and the consistency cost among multiple replications.

$$C_d = C_{stor,d} + C_{tran,d} + C_{cons,d}. \tag{8}$$

As the costs are mainly considered from the perspective of resource occupation, all costs in Eq. (8) are defined as the product of occupied resource quantity and occupied time. Assuming that there are n replications of data d in the system and the existence time of the xth replication d_x is T_{dura,d_x}, we analyze the costs of each item.

The replication storage cost is mainly caused by the occupation of storage resources by multiple replications. However, the occupation of storage resources by data replications may affect the placement of other data. Therefore, an inappropriately large number of replications may cause performance degradation. The replication storage cost is defined as:

$$C_{stor,d} = \sum_{x=1}^{n} S_d \times T_{dura,d_x}. \tag{9}$$

Where S_d represents the Size of data d.

The migration cost during replication creation is only incurred at creation time. It is worth mentioning that for the data without replications, the replication mechanism will be triggered after it is migrated, to determine whether a permanent replication of the data should be generated in the platform according to the data popularity. Because the data migration is complete at this point, generating a replication can reap the benefits of data replication while avoiding the extra data migration cost. The migration cost during replication creation is defined as:

$$C_{tran,d} = \sum_{x=2}^{n} S_d \times T_{tran,d_x} = \sum_{x=2}^{n} S_d \times \frac{S_d}{B_{i,j,d_x}}. \tag{10}$$

Where T_{tran,d_x} represents the transmission time of the replication d_x, and B_{i,j,d_x} represents the bandwidth between platform i and j occupied by the migration of replication d_x.

The consistency cost mainly comes from the cost of network resources consumed by the consistency synchronization between multiple replications. The broadcast protocol and master-slave replication mechanism which are commonly used in storage systems are applied in this case. After data is changed, the changes are broadcasted to all replications of the data for synchronization. It is assumed that data changes of every τ time, then the consistency cost is calculated based on the existence time T_{dura,d_x}.

$$C_{cons,d} = \sum_{x=2}^{n} \frac{T_{dura,d_x}}{\tau} \times \triangle S_d \times \frac{\triangle S_d}{B_{l,j,d_x}} . \tag{11}$$

Then, there needs to be a tradeoff between the global data popularity P_d and the replication cost C_d. For data with high popularity, this mechanism tends to maintain replications in the system. Meanwhile, for data with high replication costs, this mechanism tends to reduce the number of replications.

Max-min normalization is performed on the replication cost C_d:

$$C_{d,c} = \frac{C_d - C_{\min}}{C_{\max} - C_{\min}} . \tag{12}$$

Where C_{max} and C_{min} are the maximum and minimum replication costs of all data. The level of replication cost of data d among all data is expressed by relative replication cost $C_{d,c}$. Then the tradeoff score between global data popularity and replication cost is defined as:

$$Score_d = \sqrt{P_d - (1 - C_{d,c})} . \tag{13}$$

The tradeoff score is a value in the range of (0, 1). For the data with relatively low replication cost and high data popularity, the score is higher, and the mechanism is more inclined to add the replication number in the system. The upper and lower thresholds are set for the tradeoff score. For the data exceeding the thresholds, the mechanism will adjust the number and location of replications. The range of tradeoff scores, which is ($Score_l$, $Score_u$), is concluded from experiments to ensure better performance of most replications. The tradeoff scores are sorted from large to small. The data whose tradeoff score is less than $Score_l$ and ranked in the bottom 20 percent, and the data whose tradeoff score is greater than $Score_u$ and ranked in the top 20 percent are selected for data redistribution.

3.2.2 Data Redistribution

In [11], a calculation formula for the appropriate number of replications in the system is proposed according to the data size, storage resource state, application parameters, and stability parameters. However, the delay of system information in the wide-area environment may cause an inaccurate result, and the large-scale adjustment of the replications may cause a surge in network load. In this mechanism, we only implement modest adjustment of replications, and the constant polling of the replication mechanism makes the replication placement in an appropriate state. We propose a solution named increase and decrease of γ, that is, for a system with k computing platforms, n replications of data d have been distributed in the system, then γ is the number of replications added or reduced in the data redistribution stage.

$$\gamma = \begin{cases} \min\left(\left\lfloor \frac{n}{2} \right\rfloor, \left\lfloor \frac{k-n}{2} \right\rfloor\right), Score_d > Score_u \\ \left\lfloor \frac{n}{2} \right\rfloor, Score_d < Score_l \end{cases} . \tag{14}$$

The data replication placement is adjusted adaptively based on local data popularity $p_{i,d}$ and the idle state of the storage and computing resources $|\vec{R}_i|$, which are defined in Eq. (7) and Eq. (2) separately. Data popularity is relevant to the performance improvement brought by replication placement. Besides, considering the load of storage and computing resources can avoid the problem of unbalanced task allocation in subsequent scheduling.

The data popularity and resource idle state are multi-objective rankings, and n' computing platforms with the highest rank are selected to generate a new replication scheme. QoS checks are then performed, when the computing platforms specified as replication locations by the users are not in the scheme, the mechanism will replace the last ordered platforms in the scheme with the specified platforms.

Popularity- and cost-aware adaptive data replication placement adjustment mechanism is shown in Algorithm 2.

Firstly the algorithm calculates P_d and $p_{i,d}$ representing global data access popularity and local data access popularity by the frequency of data access, and calculates the cost of data storage, data transmission, and data consistency maintenance by the number of data replications, and later synthesizes the replication cost C_d. After that, the maximum and minimum values of data replication cost are searched, and each replication cost value is normalized. Finally, the increase and shrinkage policies of data are executed by the thresholds set in advance, and the platforms that perform special operations are specified by the local data access popularity.

To summarize, when there are sufficient storage and network resources in the system, the proposed mechanism can bring a large performance improvement. Otherwise, the mechanism may cause extre load on storage and network resources. The popularity and cost-aware adaptive data replication placement adjustment mechanism can optimize the data placement to reduce data migration costs and avoid long time waiting for data, thus improving the system throughput.

Algorithm 2. Popularity- and cost-aware adaptive data replication placement adjustment

Input: Data information
Output: Data placement decision

for all data d **do**
  calculate P_d by Eq. (6);
  for all i in computing platforms **do**
    calculated $p_{i,d}$ by Eq. (7);
  end for
  calculate $C_{stor,d}$ for data d by Eq. (9);
  calculate $C_{trans,d}$ for data d by Eq. (10);
  calculate $C_{cons,d}$ for data d by Eq. (11);
  $C_d = C_{stor,d} + C_{trans,d} + C_{cons,d}$;
end for
find the maximum and minimum values in C_d;
for all data d **do**
  normalize C_d by Eq. (12);
  calculate $Score_d$ by Eq. (13);
end for

for all data d **do**
  if $Score_d > Score_u$ **then**
    Add data replications by Eq. (14) and $p_{i,d}$;
  else if $Score_d < Score_l$ **then**
    Remove data replications by Eq. (14) and $p_{i,d}$;
  end if
end for

4 Experiments and Analysis

In this section, we evaluate the performance of JOTD and the comparison methods in a simulation environment. The Simgrid is used to simulate the wide-area distributed computing environment and implement different optimization methods in this platform. The system throughput, resource utilization, and global data migration cost are evaluated.

4.1 Experimental Setup

The experimental environment is established based on Simgrid [42], which simulates a wide-area distributed computing environment consisting of 5 geographically distributed computing platforms. Various parameters in the environment are configured (shown in Table 1), including the computing and storage capability of computing platforms, and network capability between computing platforms.

Table 1. Computing platform configuration

Computing platform	Core number	Core computing capability	Storage capability
1	3500	22.89 GFlops	30 TB
2	3200	29.43 GFlops	25 TB
3	2600	14.55 GFlops	20 TB
4	2700	22.89 GFlops	10 TB
5	2800	16.55 GFlops	8 TB

The datasets used for the experiments are published real workload traces from French Atomic Energy Commission (CEA) [43] and Facebook [44]. The tasks in CEA trace have the largest computing requirements, which can occupy nearly the computing resources of a whole computing platform, so it may impose a heavy burden on the system. Meanwhile, the computing requirements of tasks in the Facebook trace are relatively small and impose a steady load on the system.

In terms of optimization methods, runData [13], CAMS [11] and GCSS [45] are chosen as the comparison methods. The runData applies a dynamic data redistribution mechanism and considers data locality in the task offloading process, but does not make data replication. CAMS adopts a dynamic replication redistribution mechanism and tends to assign tasks to the computing platform which has the required data replication and the highest computing performance. When no computing platform meets the condition, then random scheduling is adopted. GCSS is a collaborative scheduling of tasks and data, which applies data replication and task rescheduling, but does not carry out dynamic redistribution of replication. JOTD is an optimization method during the system running process based on dynamic task rescheduling and data redistribution. Moreover, JOTD applies data replication and collaborative scheduling of tasks and data. Moreover, the complexity of each method is analyzed according to the published paper. The characteristics of the methods being compared are shown in Table 2.

Table 2. Characteristics of the methods being compared

Method	runData	CAMS	GCSS	JOTD
Data redistribution	✓	✓		✓
Task rescheduling	✓		✓	✓
Data replication		✓	✓	✓
Collaborative scheduling	✓	✓	✓	✓
Complexity	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$

4.2 Experimental Results

We analyze the experimental results in terms of system throughput, resource utilization, and global data migration cost. System throughput is the most important evaluation index we pay attention to, while resource utilization and global data migration cost can also help analyze the impact of different optimization mechanisms on system performance.

4.2.1 System Throughput

System throughput refers to the number of tasks completed during a certain period. We experiment in 20000 to 100000 seconds using Facebook trace and CEA trace as input.

Figure 4 indicates that JOTD has the best system throughput when replaying both Facebook trace and CEA trace. When replaying Facebook trace for 100000 seconds, JOTD outperforms runData, CAMS, and GCSS by 26.26%, 30.66%, and 27.58%, respectively. Meanwhile, JOTD respectively outperforms runData, CAMS, and GCSS by 26.64%, 33.19%, and 20.67% when replaying the CEA trace. JOTD optimizes the system performance through task rescheduling and data redistribution and dynamically adjusts the task allocation and data replication placement according to the information feedback in the system. Meanwhile, JOTD pays attention to cost, popularity, performance, and other objectives during collaborative optimization to make the system more efficient.

Therefore, JOTD achieves the highest system throughput. The runData dynamically adjusts the data placement based on data access information and resource load state during the system running process, then runData reschedules tasks to balance the load of each computing platform according to data placement, so runData method can achieve higher system throughput and performance than CAMS and GCSS. Due to the lack of a data replication mechanism, the access performance of hot data is limited, thus runData has a lower throughput than JOTD. Meanwhile, CAMS and GCSS optimize the system performance through data redistribution and task rescheduling respectively. However, single optimization direction leads to the limitation of optimization, CAMS generates random task scheduling schemes in some cases, and the replication mechanism in GCSS cannot be dynamically adjusted, so there is still room for improvement in the system throughput of CAMS and GCSS.

It can also be observed that JOTD has a greater performance improvement in system throughput when replaying CEA Trace. After comparing the two traces, it can be observed that the tasks in CEA Trace have larger core requirements, and some tasks require up to 2000 cores for running, which means that there may be high-demand tasks waiting to be executed in the system from time to time. Therefore, fragmented resources may exist in the system. These resources cannot meet the requirements of tasks in queues of the local computing platform but can meet the requirements of tasks in queues of other computing platforms. In this case, the locality-aware wide-area task stealing and backfilling mechanism introduced in section 3.1 can be triggered and brings a large performance improvement.

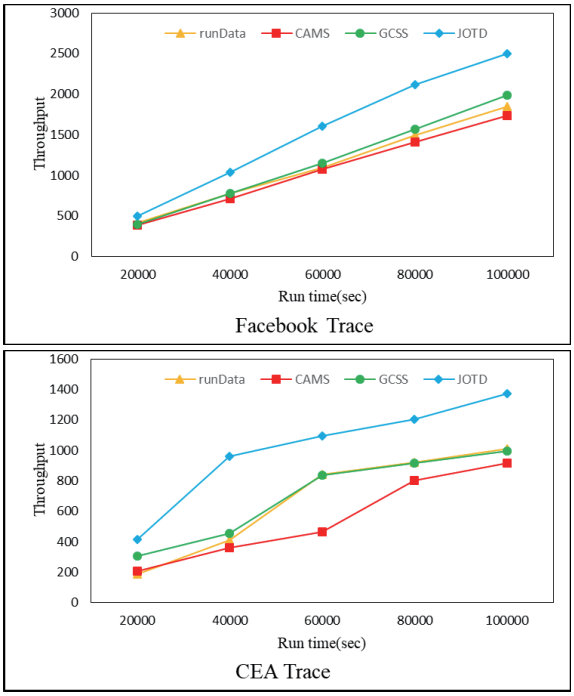

Figure 4. System throughput observed for different traces

4.2.2 Resource Utilization

Resource utilization is defined as the ratio of effective resource usage to total resource usage. It is calculated as the sum of all the used resources multiplied by their running time divided by the total resources multiplied by the system running time. It is worth mentioning that in section 4.1, we set the computing platform with sufficient storage resources, that is, the storage resource will not become the bottleneck, so we only analyze the utilization of computing resources here.

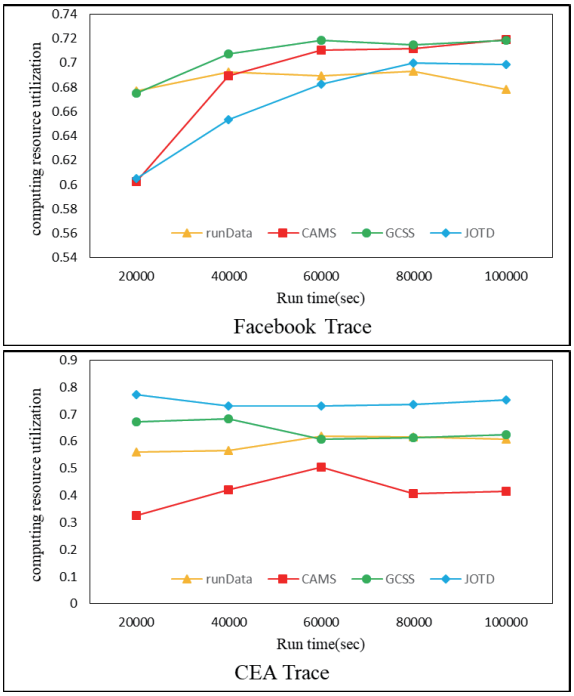

Figure 5. Computing resource utilization observed for different traces

It can be observed from Figure 5 that the computing resource utilization of JOTD is not the highest when replaying Facebook trace. However, when the system load is high, that is, when replaying CEA Trace, JOTD achieves better performance, outperforming runData, CAMS, and GCSS by 20.26%, 44.21%, and 14.00% on average respectively. When replaying Facebook trace, the load in the system is relatively stable, and there are few tasks with large core requirements, so the fragmented resources caused by resource reservation for large tasks are rarely generated. Task stealing and backfilling mechanism in JOTD is rarely triggered, so the computing resource utilization of each methods is relatively close. When replaying the CEA trace, as analyzed in section 4.2.1, the task stealing and backfilling mechanism in the JOTD method achieves good results as the load in the system is high. The locality-aware wide-area task stealing and backfilling mechanism can be triggered by idle resources in the system. In this case, the idle resources cannot meet the requirements of tasks in the local queue but can meet the requirements of tasks in queues of other computing platforms. Through this mechanism, appropriate tasks are selected to be stolen and backfilled, and the selected tasks will not affect the execution of other tasks in the computing platform with idle resources, that is, the selected tasks are executed in advance, while other tasks in the system will not be postponed. Therefore, task stealing and backfilling can effectively improve the utilization of computing resources and system throughput in the system. Similarly, GCSS with task stealing mechanism also has a good performance in computing resource utili-

zation. The runData and CAMS have poor performance in computing resource utilization compared to other methods. This occurs because the ignorance of data replication mechanism in runData may cause a long time waiting for data, and CAMS do not consider the optimization of computing resource utilization.

4.2.3 Global Data Migration Cost

We calculated the global data migration cost from the time dimension, that is, the sum of the time spent on data transmission during system running. We take 2500 tasks from both Facebook trace and CEA trace as input. The global data migration cost can reflect the performance advantage of data replication placement more intuitionistic than the storage resource utilization.

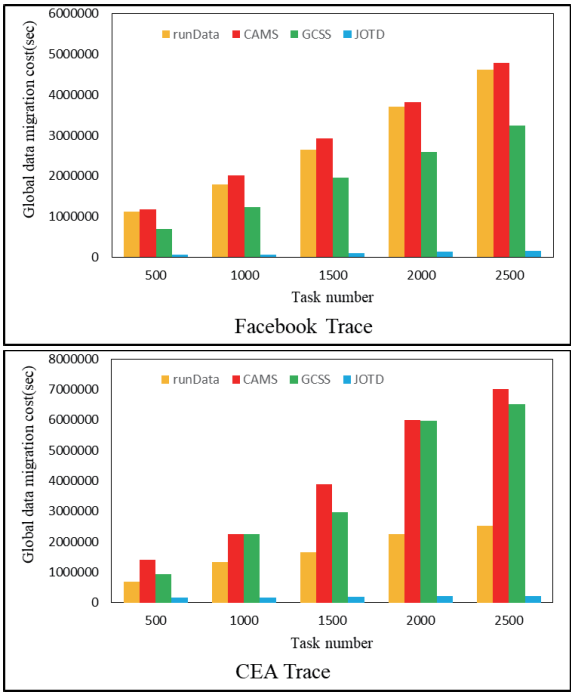

Figure 6. Global data migration cost observed for different traces

It can be observed From Figure 6 that the JOTD can significantly reduce the global data migration cost in each case. When replaying Facebook trace, JOTD outperforms runData, CAMS, and GCSS by 94.94%, 95.21%, and 94.13%, respectively. When replaying the CEA trace, JOTD outperforms runData, CAMS, and GCSS by 77.19%, 88.95%, and 83.01%, respectively. CAMS has the highest global data migration cost in all cases. In the collaborative scheduling of tasks and data process, CAMS prefers to schedule a task to the computing platform with required replication and high computing capability, and when no scheduling scheme can meet the condition, random scheduling is adopted to balance the load. Note that the schemes generated by random scheduling will cause remote data access, so the global data migration cost increases significantly. On the other hand, runData and GCSS consider data placement state and data locality when scheduling tasks and data, thus both have less global data migration cost than CAMS. However, data replication mechanism is an effective way to optimize data migration in WAN. The non-replicaiton mechanism and static replication mechanism in runData and GCSS can still be optimized to achieve a better performance in data migration. Meanwhile, in JOTD, the data replication placement can be adjusted dynamically and adaptively according to the replication cost and data popularity, so storage resources can be effectively utilized and the global data migration cost can be greatly mitigated. In the case of sufficient storage resources in the system, it is even possible to make a full replication decision on hot small data. Such data will not incur data migration costs when accessed by tasks, but only the cost of maintaining replication consistency. Therefore, the global data migration cost of JOTD is much lower than that of other methods.

5 Conclusion

This work proposes a joint online optimization of task rescheduling and data redistribution. In this method, a locality-aware wide-area task stealing and backfilling mechanism and popularity- and cost-aware adaptive data replication placement adjustment mechanism is applied. The task allocation and data placement are adjusted collaborative during the system running process to mitigate the impact of large network latency and limited bandwidth on system performance. The simulation results indicate that JOTD outperforms the state-of-the-art methods in terms of system throughput, computing resource utilization, and global data migration costs. The current realization of JOTD does not take into account the optimization of dynamic bandwidth allocation among different transport works. In the future, we would like to further improve the system performance and optimize network resource load through the bandwidth allocation methods.

Acknowledgements

This work was supported by the National Natural Science Foundation of China under Grant No. 62104014 and Grant No. 61772053, and the fund of the State Key Laboratory of Software Development Environment under Grant No. SKLSDE-2022ZX-07. This job is also supported by Natural Science Foundation of Shaanxi Province of China (2021JM-344) and Shaanxi Key Laboratory of Intelligent Processing for Big Energy Data (No. IPBED7).

References

[1] L. Cheng, Y. Wang, Q. Liu, D. H.-J. Epema, C. Liu, Y. Mao, J. Murphy, Network-Aware Locality Scheduling for Distributed Data Operators in Data Centers, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 32, No. 6, pp. 1494-1510, June, 2021.

[2] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson, R. Roskies, J. R. Scott, N. Wilkins-Diehr, XSEDE: Accelerating Scientific Discovery, *Computing in science and engineering*, Vol. 16, No. 5, pp. 62-74, September/ October, 2014.

[3] F. Gagliardi, The European Grid Infrastructure EGEE Project, in: M. Dayde, J. Dongarra, V. Hernandez, M. L.

M. Palma (Eds.), *Proceedings of the 6th International Conference on High Performance Computing for Computational Science*, Valencia, Spain, 2004, pp. 194-203.

[4] X. Xie, N. Xiao, Z. Xu, L. Zha, W. Li, H. Yu, CNgrid software 2: service oriented approach to grid computing, *Proceedings of the UK e-Science All Hands Meeting*, Nottingham, UK, 2005, pp. 701-708.

[5] S. Kang, B. Veeravalli, K. Aung, Dynamic scheduling strategy with efficient node availability prediction for handling divisible loads in multi-cloud systems, *Journal of Parallel and Distributed Computing*, Vol. 113, pp. 1-16, March, 2018.

[6] C. Li, J. Bai, J. Tang, Joint optimization of data placement and scheduling for improving user experience in edge computing, *Journal of Parallel and Distributed Computing*, Vol. 125, pp. 93-105, March, 2019.

[7] E. Gussier, J. Lelong, V. Reis, D. Trystram, Online Tuning of EASY-Backfilling using Queue Reordering Policies, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 29, No. 10, pp. 2304-2316, October, 2018.

[8] D. Carastan-Santos, R. Y. De-Camargo, Obtaining dynamic scheduling policies with simulation and machine learning, *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis*, Denver, Colorado, USA, 2017, pp. 1-13.

[9] M. T. Chung, J. Weidendorfer, P. Samfass, K. Fuerlinger, D. Kranzlmuller, Scheduling across Multiple Applications using Task-Based Programming Models, *Proceedings of 2020 IEEE/ACM Fourth Annual Workshop on Emerging Parallel and Distributed Runtime Systems and Middleware*, Georgia, USA, 2020, pp. 1-8.

[10] Z. Li, V. Chang, H. Hu, H. Hu, C. Li, J. Ge, Real-Time and Dynamic Fault-Tolerant Scheduling for Scientific Workflows in Clouds, *Information Sciences*, Vol. 568, pp. 13-39, August, 2021.

[11] M. Breitbach, D. Schafer, J. Edinger, C. Becker, Context-Aware Data and Task Placement in Edge Computing Environments, *Proceedings of 2019 IEEE International Conference on Pervasive Computing and Communications*, Kyoto, Japan, 2019, pp. 1-10.

[12] S. Wang, X. Zhang, K. Yang, L. Wang, W. Wang, Distributed edge caching scheme considering the tradeoff between the diversity and redundancy of cached content, *Proceedings of 2015 IEEE/CIC International Conference on Communications in China*, Shenzhen, China, 2015, pp. 1-5.

[13] Y. Jin, Z. Qian, S. Guo, S. Zhang, L. Jiao, S. Lu, runData: Re-distributing Data via Piggybacking for Geo-distributed Data Analytics over Edges, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 33, No. 1, pp. 40-55, January, 2022.

[14] K. Qu, L. Meng, Y. Yang, A dynamic replica strategy based on Markov model for hadoop distributed file system (HDFS), *Proceedings of the 2016 4th International Conference on Cloud Computing and Intelligence Systems*, Beijing, China, 2016, pp. 337-342.

[15] K. Wang, K. Qiao, I. Sadooghi, X. Zhou, T. Li, M. Lang, I. Raicu, Load-balanced and locality-aware scheduling for data-intensive workloads at extreme scales, *Concurrency and computation: practice and experience*, Vol. 28, No. 1, pp. 70-94, January, 2016.

[16] J. Li, X. Zhang, J. Wei, Z. Ji, Z. Wei, GARLSched: Generative adversarial deep reinforcement learning task scheduling optimization for large-scale high performance computing systems, *Future Generation Computer Systems*, Vol. 135, pp. 259-269, October, 2022.

[17] J. Li, X. Zhang, L. Han, Z. Ji, X. Dong, C. Hu, OKCM: improving parallel task scheduling in high-performance computing systems using online learning, *The Journal of Supercomputing*, Vol. 77, No. 6, pp. 5960-5983, June, 2021.

[18] Q. Wang, H. Zhang, C. Qu, Y. Shen, X. Liu, J. Li, RLSchert: An HPC Job Scheduler Using Deep Reinforcement Learning and Remaining Time Prediction, *Applied Sciences*, Vol. 11, No. 20, Article No. 9448, October, 2021.

[19] J. Taghizadeh, M. Ghobaei-Arani, A. Shahidinejad, A metaheuristic-based data replica placement approach for data-intensive IoT applications in the fog computing environment, *Software: Practice and Experience*, Vol. 52, No. 2, pp. 482-505, February, 2022.

[20] A. B. Yoo, M. A. Jette, M. Grondona, Slurm: Simple linux utility for resource management, *Proceedings of Workshop on Job Scheduling Strategies for Parallel Processing*, Seattle, WA, USA, 2003, pp. 44-60.

[21] G. Staples, TORQUE resource manager, *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, Tampa, Florida, USA, 2006, pp. 8-es.

[22] B. Nitzberg, J. M. Schopf, J. P. Jones, In: J. Nabrzyski, J. M. Schopf, J. Węglarz (Eds), PBS Pro: Grid computing and scheduling attributes, *International Series in Operations Research & Management Science*, Vol. 64, Springer, Boston, MA, 2004, pp. 183-190.

[23] P. Valledor, A. Gomez, P. Priore, J. Puente, Modelling and Solving Rescheduling Problems in Dynamic Permutation Flow Shop Environments, *Complexity*, Vol. 2020, pp. 1-17, July, 2020.

[24] W. Wei, X. Fan, H. Song, X. Fan, J. Yang, Imperfect Information Dynamic Stackelberg Game Based Resource Allocation Using Hidden Markov for Cloud Computing, *IEEE Transactions on Services Computing*, Vol. 11, No. 1, pp. 78-89, January-February, 2018.

[25] Z. Zhong, J. He, M. A. Rodriguez, S. Erfani, R. Kotagiri, R. Buyya, Heterogeneous Task Co-location in Containerized Cloud Computing Environments, *Proceedings of the 23rd International Symposium on Real-Time Distributed Computing*, Nashville, TN, USA, 2020, pp. 79-88.

[26] T. J. Ikonen, K. Heljanko, I. Harjunkoski, Reinforcement learning of adaptive online rescheduling timing and computing time allocation, *Computers & Chemical Engineering*, Vol. 141, pp. 1-17, October, 2020.

[27] A. Daoud, F. Balbo, P. Gianessi, G. Picard, ORNInA: A decentralized, auction-based multi-agent coordination in ODT systems, *Ai Communications*, Vol. 34, No. 1, pp. 37-53, 2021.

[28] Y. Guo, J. Zhao, V. Cave, V. Sarkar, SLAW: a scalable locality-aware adaptive work-stealing scheduler for multi-core systems, *Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Paral-*

lel Programming, Bangalore, India, 2010, pp. 341-342.
[29] R. Yadav, W. Zhang, O. Kaiwartya, P. R. Singh, I. A. Elgendy, Y. Tian, Adaptive Energy-aware Algorithms for Minimizing Energy Consumption and SLA Violation in Cloud Computing, *IEEE Access*, Vol. 6, pp. 55923-55936, 2018.
[30] R. Yadav, W. Zhang, K. Li, C. Liu, M. Shafiq, N. K. Karn, An adaptive heuristic for managing energy consumption and overloaded hosts in a cloud data center, *Wireless Networks*, Vol. 26, No. 3, pp. 1905-1919, April, 2020.
[31] R. Yadav, W. Zhang, K. Li, C. Liu, A. Laghari, Managing overloaded hosts for energy-efficiency in cloud data centers, *Cluster Computing*, Vol. 24, pp. 2001-2015, 2021.
[32] R. Yadav, W. Zhang, MeReg: Managing Energy-SLA Tradeoff for Green Mobile Cloud Computing, *Wireless Communications and Mobile Computing*, Vol, 2017, Article No. 6741972, December, 2017.
[33] R. Yadav, W. Zhang, H. Chen, T. Guo, MuMs: Energy-Aware VM Selection Scheme for Cloud Data Center, *2017 28th International Workshop on Database and Expert Systems Applications (DEXA)*, Lyon, France, 2017, pp. 132-136.
[34] G. Muthusamy, S.-R. Chandran, Cluster-based Task Scheduling Using K-Means Clustering for Load Balancing in Cloud Datacenters, *Journal of Internet Technology*, Vol. 22, No. 1, pp. 121-130, January, 2021.
[35] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, I. Stoica, Managing data transfers in computer clusters with orchestra, *ACM SIGCOMM Computer Communication Review*, Toronto, Ontario, Canada, 2011, pp. 98-109.
[36] L. Wang, Y. Zhang, J. Xu, G. Xue, MAPX: Controlled Data Migration in the Expansion of Decentralized Object-Based Storage Systems, *Proceedings of the 18th USENIX Conference on File and Storage Technologies*, Santa Clara, CA, USA, 2020, pp. 1-11.
[37] G. Zhang, Z. Huang, X. Ma, S. Yang, Z. Wang, W. Zheng, RAID+: Deterministic and balanced data distribution for large disk enclosures, *Proceedings of the 16th USENIX Conference on File and Storage Technologies*, Oakland, CA, USA, 2018, pp. 279-294.
[38] D. Sun, G. Zhang, S. Gao, Data Management across Geographically-Distributed Autonomous Systems: Architecture, Implementation, and Performance Evaluation, *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, Zhangjiajie, China, 2019, pp. 2284-2292.
[39] S. Kadekodi, F. Maturana, S. J. Subramanya, J. Yang, K. V. Rashmi, G. R. Ganger, PACEMAKER: Avoiding HeART attacks in storage clusters with disk-adaptive redundancy, *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation*, Virtual Event, 2020, pp. 369-385.
[40] W. Zhang, R. Yadav, Y. Tian, S. K. S. Tyagi, I. A. Eelgendy, O. Kaiwartya, Two-Phase Industrial Manufacturing Service Management for Energy Efficiency of Data Centers, *IEEE Transactions on Industrial Informatics*, February, 2022.
[41] T. Kosar, M. Balman, A new paradigm: Data-aware scheduling in grid computing, *Future Generation Computer Systems*, Vol. 25, No. 4, pp. 406-413, April, 2009.
[42] H. Casanova, A. Legrand, M. Quinson, SimGrid: a Generic Framework for Large-Scale Distributed Experiments, *Proceedings of the 10th International Conference on Computer Modeling and Simulation*, Cambridge, UK, 2008, pp. 126-131.
[43] D. G. Feitelson, D. Tsafrir, D. Krakov, Experience with using the Parallel Workloads Archive, *Journal of Parallel and Distributed Computing*, Vol. 74, No. 10, pp. 2967-2982, October, 2014.
[44] Y. Chen, A. Ganapathi, R. Griffith, R. Katz, The Case for Evaluating MapReduce Performance Using Workload Suites, *2011 IEEE 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*, Singapore, 2011, pp. 390-399.
[45] Y. Song, L. Xiao, L. Wang, G. Qin, B. Wei, B. Yan, C. Zhang, GCSS: a global collaborative scheduling strategy for wide-area high-performance computing, *Frontiers of Computer Science*, Vol. 16, No. 5, pp. 1-15, October, 2022.

Biographies

Yao Song received the B.S. in spacecraft design and engineering from Beihang University, Beijing, China, in 2016. He is currently pursuing a Ph.D. degree in cyber-space security at Beihang University. His main research interests include cyber-space security, parallel and distributed file system, high performance computing, and scheduling system.

Limin Xiao received the Ph.D. degree in computer science from Institute of computing, Chinese Academy of Sciences, Beijing, China, in 1998. He is a professor of the School of Computer Science and Engineering, Beihang University, Beijing, China. His main research areas are computer architecture, high performance computing, and cloud computing.

Liang Wang received the Ph.D degree in Computer Science and Engineering from The Chinese University of Hong Kong in 2017. He is an assistant professor with the School of Computer Science and Engineering, Beihang University, China. His research interests include power-efficient and reliability-aware design for network-on-chip and many-core system.

Wei Wei received his Ph.D. degree from Xi'an Jiaotong University in 2011. He is an associate Professor at Xi'an University of Technology. He is an ACM&IEEE Senior Member. His research interests include Wireless Networks and Wireless Sensor Net-works Application, Mobile Computing, Distributed Computing, and Pervasive Computing.

Jinquan Wang received the B.S. in software engineering from Hunan University, Changsha, China, in 2021. He is currently pursuing a Ph.D. degree in Computer Architecture at Beihang University. His main research interests include parallel and distributed file systems, high performance computing, and distributed management system.