Journal of Computers Vol. 30 No. 4, 2019, pp. 113-121
doi:10.3966/199115992019083004010

# Task Scheduling Algorithm Based on Improved Data Gridding

Xiang-Kui Jiang[1*], Yong-Qing Fan[1], Zhi-Cang Wang[1], He-Jun Xuan[2]

[1] School of Automation, Xi'an University of Posts and Telecommunications,
Chang'an West Street, Chang'an District, Xi'an 710121, China
{jiangxiangkui@xupt.edu.cn, fanyongqing@xupt.edu.cn, zhcwang@qq.com}

[2] School of Computer and Information Technology, Xinyang Normal University,
Xinyang, 464000, China
xuanhejun0896@126.com

Received 13 January 2019; Revised 14 January 2019; Accepted 12 March 2019

Abstract. Task scheduling in distributed environments is an important research field. To address the issue of effective resource scheduling, a resource scheduling model using data gridding is proposed in this paper. First, we use data gridding on the attributes of resources to divide different types of resources into different data subspaces. We then search the gridding data for resources satisfying the minimum requirements of the task, which reduces the blindness of the allocation of resources and decreases the time needed for resource matching and scheduling. Finally, a globally optimized resource allocation strategy to minimize the task execution time is adopted to choose from among the resources satisfying the minimum task requirements. Results of experimental and theoretical analyses show that the proposed algorithm has a higher task-matching speed and shorter task-execution time than traditional algorithms.

Keywords: data gridding, distributed computing, heterogeneous network, task scheduling

## 1 Introduction

The preparation of manuscripts which are to be reproduced by photo-offset requires special care. Papers submitted in a technically unsuitable form will be returned for retyping, or canceled if the volume cannot otherwise be finished on time. With the advent of the age of big data, the effective use of parallel and distributed systems for big data processing has become a popular topic of scientific research. Task scheduling in distributed systems [1-2] is the typical method for highly efficient handling of big data. By considering the performances of various resource nodes, task requirements, and related factors, task scheduling allocates various tasks to the corresponding resource nodes in an approach for achieving particular goals or meeting certain expectations [3-4]. The effectiveness of the task scheduling algorithm is an important factor affecting the quality of system services; therefore, the manner of choosing appropriate resources from the system and for assigning tasks are key problems to be solved when designing scheduling algorithms for executing tasks.

Different user tasks have different levels of service demands [2]. Algorithms for task scheduling have been widely studied [5-8]. The current research focus area for task scheduling algorithms is the scheduling of tasks to minimize the overall response time, which is minimization of the time needed to complete the job. One such algorithm is the so-called min-min algorithm. This algorithm can rapidly schedule tasks. However, the scheduling algorithm always schedules sub-tasks in advance before carrying out the main task. Moreover, some resources are frequently assigned tasks, leading to an imbalance in the processor load.

Scheduling algorithms based on intelligent algorithms, such as genetic algorithms and particle swarm optimization (PSO) algorithms [9-10], can obtain better scheduling results during static scheduling. In addition, they can achieve faster response times for tasks. However, for real-time tasks, good scheduling

* Corresponding Author

results have not been demonstrated. In a grid environment with rich resources, algorithms based on resource clustering [11-12] can narrow the scope of task scheduling during resource selection, improve the resource selection accuracy, choose the appropriate processor for task allocation, and reduce the task execution time. Although these scheduling algorithms can improve the speed of task resource matching, the result is not necessarily optimal clustering. This is because of the parameter settings at the moment of clustering and the limitations of the clustering methods. Preferred tasks are unable to find resources to meet the given conditions, resulting in a reduced quality of system service.

To decrease the time required for matching tasks and resources, and to improve the efficiency of resource allocation, a resource matching algorithm based on improved handling of grid data is herein proposed. The distributed system resources, which fulfill the minimum demand, are quickly engaged when a user task arrives, thus improving the speed of matching tasks and resources. To reduce the task execution time, a global optimal search strategy is used to minimize the completion time of the user task.

## 2 Problem Statement and Preliminaries

In a distributed system—which has a greater number of processor resources—let $R=\{R_1,R_2,\ldots,R_n\}$ be the set of processors in the distributed system resources, where $n=|R|$ is the number of processor resources in the system. For the $i$ th processor $R_i$ in the resource pool of the distributed system, we use the triad $R_i=\{R_i^{\mathrm{id}},R_i^{\mathrm{user}},R_i^{\mathrm{cap}}\}$ to represent the attributes of processor $R_i$, in which $R_i^{\mathrm{id}}$ is the ID number of the processor, $R_i^{\mathrm{user}}$ means that the processor is subordinate to the providing distributed system, and $R_i^{\mathrm{cap}}$ is the computing power, storage capacity, reliability, security level, and other characteristic values of the processor.

A heterogeneous distributed system is one in which the attributes of different processors differ in their distributed system resources, such as their calculation speed, communication speed, storage capacity, reliability, and security. Therefore, attribute $R_i^{\mathrm{cap}}$ of processor $R_i$ can be expressed as $R_i^{\mathrm{cap}}=\{r_1^i,r_2^i,\ldots,r_N^i\}$, where $r_j^i\ (j=1,2,\ldots,N)$ is the jth attribute value in the attribute $R_i^{\mathrm{cap}}$ of processor $R_i$, and N is the number of $R_i^{\mathrm{cap}}$ attributes of processor $R_i$.

With the expansion of distributed applications, an increasing number of application tasks have certain requirements in terms of system resources [13]. For example, real-time tasks require a certain minimum computing speed, a network link to the processor, and an adequate storage device speed to achieve instantaneous execution of tasks. Furthermore, the levels of reliability, fault-tolerant capability, and security performance of processors must also be high to satisfy the reliability and safety demands of tasks.

For a task in a distributed system, let $T=\{T_1,T_2,\ldots,T_m\}$ be the task set, where $m=|T|$ is the number of tasks in the set. In this task set, the ith task $T_i$ can be expressed as $T_i=\{t_{\mathrm{a}}^i,t_{\mathrm{d}}^i,t_1^i,t_2^i,\ldots,t_N^i\}$, where $t_{\mathrm{a}}^i$ is the arrival time of task $T_i$, $t_{\mathrm{d}}^i$ is the absolute deadline for task $T_i$, and $t_p^i\ (p=1,2,\ldots,N)$ is the minimum requirement for properties $r_p\ (p=1,2,\ldots,N)$ of the processor resource. Consequently, if task $T_i$ has been assigned to be performed on processor $R_j$, then $t_p^i\le r_p^j\ (p=1,2,\ldots,N)$; that is, the value of the $p$ th attribute of processor $R_j$ is not less than the minimum required for task $T_i$.

## 3 Improved Scheduling Model Using Data Gridding

Decreasing the time required for matching tasks and resources, and quickly selecting the appropriate resources to be assigned to a task, are important strategies for minimizing the response time of a task. When a task arrives, the system resources meeting the minimum demands of the task are quickly selected from the distributed system. Appropriate resources are selected and assigned to tasks submitted by users, thus minimizing the task execution time. To address the problem of how resources meeting the minimum requirements of a task can be rapidly selected from the distributed system, we herein propose a scheme

based on improved data gridding. The scheme is designed to improve the speed and accuracy of the allocation of resources.

### 3.1 Improved Resource Matching via Data Gridding

Data gridding is a means of processing discrete data that is widely used in light detection and ranging (LiDAR) point cloud data filtering, target recognition, and other research fields [14-15]. The data gridding method divides a high-dimensional space into subspaces $n_1 \times n_2 \times \cdots \times n_N$ (where N is the number of data dimensions), and then maps the data to the corresponding subspaces. This method is simple and convenient. Moreover, it is easily applicable to discrete data and data having no rules. Thus, the attribute information for resources in the distributed system and the task requirements for these properties can be abstracted as data points. The discrete points can then be mapped to the subspaces of the data space.

#### 3.1.1 Grid Width Selection and Data Coordinate Determination

Each attribute value or task requirement can be regarded as a data point in data space. In this space, the dimensions are represented by the data point coordinates, and the appropriate mapping function is chosen to map each data point to the corresponding subspace. To perform this mapping, an appropriate grid width for each dimension must be selected to divide the high-dimensional data space. In traditional data gridding on LiDAR data, it is valid to conduct a mesh partition with a uniform distribution of data points, where each dimension has the same grid width. However, for data representing the attributes of distributed system resources, which have a non-uniform distribution, this method is not suitable. When data are more concentrated in a dimension, the large grid width will cause more data points to map to the same grid, which is not conducive to the gridding of resource attribute data. Therefore, we propose an improved data gridding method for matching resources that meet the minimum task demands.

Assuming that the ith data point is expressed by $x^i = \left(x_1^i, x_2^i, \ldots, x_N^i\right)$, the width of the initial grid and the coordinates of the data points in the data grid can be calculated via the following steps.

Calculate the data space volume by:

$$V_N = \prod_{i=1}^{N} \left(x_{\max}^i - x_{\min}^i\right). \tag{1}$$

where $x_{\max}^i = \max\limits_{1 \le k \le N_p} \{x_k^i\}$ and $x_{\min}^i = \min\limits_{1 \le k \le N_p} \{x_k^i\}$ are the maximum and minimum, respectively, of the data on the ith dimension, and $N$ is the dimensionality of the data space.

(2) Calculate the average data volume held by data points according to:

$$V_N' = V_N / N_p \ . \tag{2}$$

where $V_N$ is the volume of the data space and $N_p$ is the number of data items in the space.

(3) Calculate grid width $a^i$ on the ith dimensional space by:

$$a^i = k \times s^i, \quad i = 1, 2, \ldots, N \ . \tag{3}$$

where $s^i$ is the variance of the data in the ith dimension. In addition, the proportionality coefficient $k$ is the specific value for the ith grid width $a^i$ and the variance $s^i$. Once the grid width $a^i$ is determined, the information pertaining to the data distributed in the ith dimension is considered. The grid width of data in the ith dimension and the data distribution in the dimension are directly proportional to the intensity; that is, $a^i = k \times s^i$. Thus, the distribution of data in the dimension can effectively determine the grid width: when the data distribution is dense, grid width is small; otherwise, the width is larger.

(4) Calculate the coefficient, $k$, by Eqs. (4) and (5):

$$V_N' = \prod_{i=1}^{N} a^i = \prod_{i=1}^{N} \left(k \times s^i\right) = k^N \times \prod_{i=1}^{N} s^i \,. \tag{4}$$

$$k = \sqrt[N]{V_N \Big/ \prod_{i=1}^{N} s^i} = \sqrt[N]{V_N' \Big/ \left(N_p \times \prod_{i=1}^{N} s^i\right)}. \tag{5}$$

(5) Calculate the position $p^i = \left(p_1^i, p_2^i, \ldots, p_N^i\right)$ of the ith data point in the grid according to:

$$p_k^i = \left\lfloor \left(x_k^i - x_{\min}^i\right) \Big/ a^i \right\rfloor + 1, \quad k = 1, 2, \ldots, N \,. \tag{6}$$

After Steps 1 to 5, the width of the initial mesh and the coordinates of the data have been identified. To make each data grid contain only one data point, it is necessary to modify the grid width (generally, one to three revisions are needed). In this paper, the final determinations of grid width and data points are completed according to the method shown in Fig. 1.

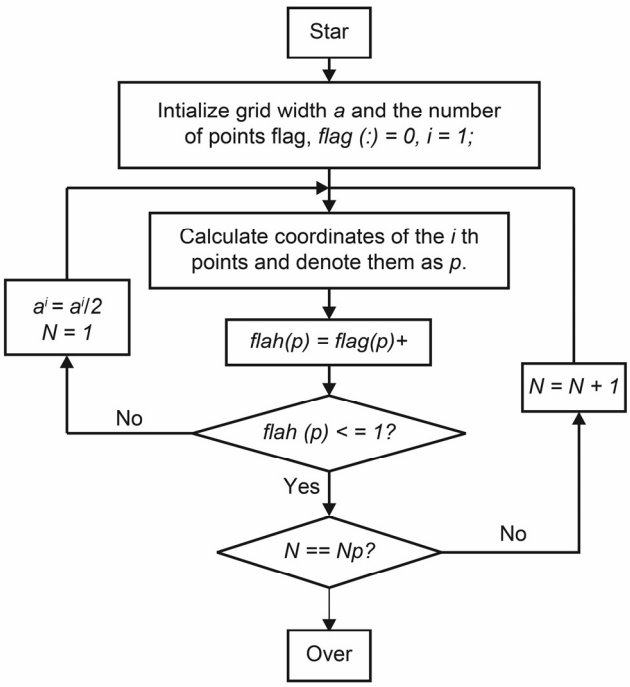


**Fig. 1.** Calculation of the grid width and data point coordinates

In Steps 3 and 4 of the improved data gridding method, the distribution information in each dimension in the data is considered. The grid width for the dimension and the intensive distribution of data in the dimension are directly proportional; that is, $a^i = k \times s^i$, where $s^i$ is the variance of data in the ith dimension. When the data distribution is dense, the grid width is small; otherwise the width is larger. In this case, the distribution information of data is conducive to the mapping of only one point in each grid. Traditional data gridding methods only determine the grid width once, and multiple data points are mapped to each grid. Therefore, to cause only one resource data point to be mapped to each subspace, a correction process for determining the grid width of the data is incorporated into the improved gridding method; that is, $a^i = a^i / 2$, as shown in Fig. 1. Although the decrease in grid width will increase the number of grids, only grids having data points mapped to them are stored at the time of storage. Thus, this modification will not increase the storage space or reduce the speed of access.

### 3.1.2 Coarse Matching of Resources

When task $T_i = \left\{t_{\mathrm{a}}^i, t_{\mathrm{d}}^i, t_1^i, t_2^i, \ldots, t_N^i\right\}$ is assigned to execute in processor $R_j = \left\{R_j^{\mathrm{id}}, R_j^{\mathrm{user}}, r_1^j, r_2^j, \ldots, r_N^j\right\}$, Eq. (7) is established, which states that the $p$ th attribute value of processor $R_j$ is not less than the minimum required by task Ti.

$$t_p^i \le r_p^j \quad (p = 1, 2, \ldots, N). \tag{7}$$

After the data gridding, the subspace position of the resource requests for task $T_i = \left\{t_{\mathrm{a}}^i, t_{\mathrm{d}}^i, t_1^i, t_2^i, \ldots, t_N^i\right\}$ is $p^{\mathrm{req}} = \left(p_1^{\mathrm{req}}, p_2^{\mathrm{req}}, \ldots, p_N^{\mathrm{req}}\right)$ on the condition that the data subspace position of processor $j$ assigned to the task is $p_j = \left(p_1^j, p_2^j, \ldots, p_N^j\right)$, where $p_k^{\mathrm{req}}$ and $p_k^j$ can be obtained by Eqs. (8) and (9).

$$p_k^{\mathrm{req}} = \left\lfloor \left(t_k^i - r_k^{\min}\right) / a^k \right\rfloor + 1, \quad k = 1, 2, \ldots, N. \tag{8}$$

$$p_k^j = \left\lfloor \left(r_k^i - r_k^{\min}\right) / a^k \right\rfloor + 1, \quad k = 1, 2, \ldots, N. \tag{9}$$

In these equations, $r_k^{\min}$ is the minimum value of the data $k\left(k = 1, 2, \ldots, N\right)$ in the preprocessing resource.

During data gridding, which is a kind of linear transformation from the property data to grid coordinates, the size relationships of the data are preserved, and Eq. (10) can be obtained by Eq. (7), (8), and (9).

$$p_k^j \ge p_k^{\mathrm{req}}, \quad k = 1, 2, \ldots, N. \tag{20}$$

Using Eq. (10), the resources that satisfy the lowest requirement of the given task are at the top right in the subspace of the task after the resource data gridding. Thus, when a task arrives, the subspace locations are calculated according to Eq. (8). Then, the processor that satisfies the minimum requirement for the task resource can be found with degree of complexity $o(1)$.

### 3.2 Resource Scheduling Model

With the improved data gridding method, the matching of the task and resources can be carried out quickly when a task arrives, thereby finding the correct resources from among those that meet the minimum task demands. To minimize the task execution time, the appropriate processor must be determined. In a distributed system, the completion time of a task submitted by a user depends on the finishing time, which is the time of the last accomplishment of the task. It is also the time of the end of the latest completed task execution. Let the completion time of a task of user $u$ be $t\left(T^u\right)$, and let $t\left(T_j^u\right)$ be the completion time of task $T_j^u$.

$$t\left(T_j^u\right) = \max\left\{t\left(R_i\right), t_{\mathrm{a}}^j\right\} + t\left(T_{i,j}^u\right). \tag{31}$$

In Eq. (11), $t\left(R_i\right)$ represents the finishing time of the previous task in processor $R_i$, $t_{\mathrm{a}}^j$ is the arrival time of user task $T_j$, and $t\left(T_{i,j}^u\right)$ is the execution time of task $T_j$ in processor $R_i$. Then, $t\left(T^u\right)$ can be expressed by Eq. (12), which is the latest completion time of all tasks of this user.

$$t\left(T^u\right) = \max_{j \in U}\left\{t\left(T_j^u\right)\right\}. \tag{42}$$

In Eq. (12), $U$ is the set of all tasks of user $u$. To make the execution time of the tasks of user $u$ the earliest, $t\left(T^u\right)$ must be made the earliest. Thus, we employ Eq. (13) to determine the processor to which the task will be assigned for execution for ensuring the earliest possible finishing time for the task.

$$\begin{cases} num\_P = \arg\min\limits_{i} \left\{ \max\limits_{j \in U} \left\{ t\left(T_j^u\right) \right\} \right\} \\ \text{such that} \\ \quad t\left(T_j^u\right) \le t_{\mathrm{d}}^j \\ \quad i \in R^U \end{cases} \tag{53}$$

In Eq. (13), $t_{\mathrm{d}}^j$ is the deadline for completion of the task, and $R^U$ is the set of processors meeting the requirements of task $T_j$. By solving this equation, the processor $num\_P$ to which task $T_j$ is assigned is obtained, which will ensure the shortest processing time for the task.

## 4 Experiment Results and Analysis

To decrease the time needed for resource matching, we propose a strategy for resource matching based on improved data gridding. The proposed method accelerates the search for resources of the distributed system that meets the minimum requirements of a task. A global search strategy is adopted to improve the distribution of processors to execute the task in the shortest possible time. To verify the validity of this method, the comparison of algorithms mainly focuses on the time needed to search for resources that meet the minimum requirements of a task (i.e., the time for coarse resource matching) and on the execution time of the task. Section 4.1 provides an analysis of the time complexity of coarse matching for resources to verify the speed achieved when matching resources using the improved data gridding method. In Section 4.2, the algorithm proposed herein (task scheduling based on data gridding, or TSBDG) is compared with the min-min algorithm and the fuzzy clustering and two-level-based task scheduling (FCTLBS) algorithm [12] in completing simulation experiments on different tasks. We thereby demonstrate that the TSBDG algorithm results in execution times are shorter than those of other algorithms.

### 4.1 Time for Coarse Matching of Resources

In the method of the comparison of resources, the task having the lowest demand for resources is found in a distributed system with $m$ resources by a search of $n_t$ tasks for the task having the lowest requirements; the time complexity for this search is $o(m \times n_t)$. Using the clustering method, the complexity of dividing $m$ resources in the distributed system into $N$ classes ($N$ being the number of resource properties) is $o(m \times IterateTimes)$, where $IterateTimes$ is the maximum number of iterations during clustering. The complexity of searching for resources that meet the minimum requirements of a task from among preferred resources is $o(m_i)$, where $m_i$ is the number of resources of class $i$. As a result, using a clustering search for resources meeting the lowest requirements of $n_t$ tasks in a distributed system, the time complexity is $o(m_i \times n_t + m \times IterateTimes)$. Moreover, when the quantity $n_i$ of tasks is a larger value, the complexity of this method is $o\left(m_i \times n_t\right)$.

Using data gridding methods, the complexity of partitioning $m$ resources in a distributed system into subspaces is $o(m \times MaxTimes)$, where $MaxTimes$ is the maximum number of revisions. The maximum number of revisions is generally one to three, and it is significantly smaller than the maximum number of iterations for the clustering method. The complexity of searching for resources that meet the minimum requirements of tasks in the data grid is $o(1)$. By using the improved data gridding, the time complexity of searching for resources meeting the lowest requirements of $n_t$ tasks in a distributed system is $o(n_t + m \times MaxTimes)$, where $1 \le MaxTimes \le 3$. Therefore, when the quantity of tasks $n_t$ is a larger value, the time complexity of coarse matching of resources using the improved data gridding method proposed herein is $o(n_t)$.

Table 1 shows the time complexity of the three methods of coarse matching of resources, where $m$ is the number of resources in the distributed system, $m_i$ is the number of resources in the distributed system

of kind $i$ after clustering of resources, and $n_t$ is the number of tasks that need resource matching. It is evident from the time complexity analysis for resource matching algorithms that the complexity of resource matching using the data gridding method based on the improvement proposed herein is smaller. Consequently, the matching speed will be higher.

**Table 1.** Time complexity for coarse matching of resources

| Resource comparison method | Resource clustering method | Data gridding method |
|---|---|---|
| $o(m \times n_t)$ | $o(m_i \times n_t)$ | $o(n_t)$ |

### 4.2 Completion Time of Tasks

The goal of task scheduling is to achieve a reasonable allocation of resources to tasks to improve the system efficiency as a whole and reduce the response time of tasks, which entails minimizing the task execution time. Among task scheduling algorithms, the min-min algorithm is the basis for many scheduling models. The strategy for allocation of resources using improved data gridding is to minimize the execution time of tasks. Therefore, the proposed TSBDG algorithm was compared in an experiment with the min-min algorithm and the FCTLBS algorithm proposed in Ref. [12] to demonstrate the TSBDG algorithm effectiveness.

To increase the objectivity of the experiment, the processor had a calculation ability, communication ability, and storage capacity. The simulation system ran on an Intel(R) Core(TM) i7 CPU with 2.93 GHz, 8 GB of RAM, and the 64-bit Windows 7 operating system. The simulation system consisted of the resource generator, task generator, and task scheduler. The resource generator handled the generation of resource attribute data in the distributed system. The task generator handled the generation of user tasks. The task scheduler assigned a task submitted by a user to the appropriate processing machine. The minimum requirements for the resources related to each task were randomly generated. The generated task identified resources meeting the minimum requirements in the resource pool. The computation goal was from the range $[t_{\text{comp}}, C_{\text{diff}} \times t_{\text{comp}}]$, the network throughput memory space was from $[t_{\text{BW}}, B_{\text{diff}} \times t_{\text{BW}}]$, and the memory space was from $[t_{\text{store}}, S_{\text{diff}} \times t_{\text{store}}]$, where $t_{comp}$, $t_{BW}$, $t_{store}$ $C_{\text{diff}}$, $B_{\text{diff}}$ and $S_{\text{diff}}$ were random numbers in [1, 25]. Fig. 2 and Fig. 3 show the execution times for resource allocation using the three algorithms (TSBDG vs. min-min and FCTLBS) when the number of tasks submitted by users was 50 to 500 and 500 to 1,500, respectively.

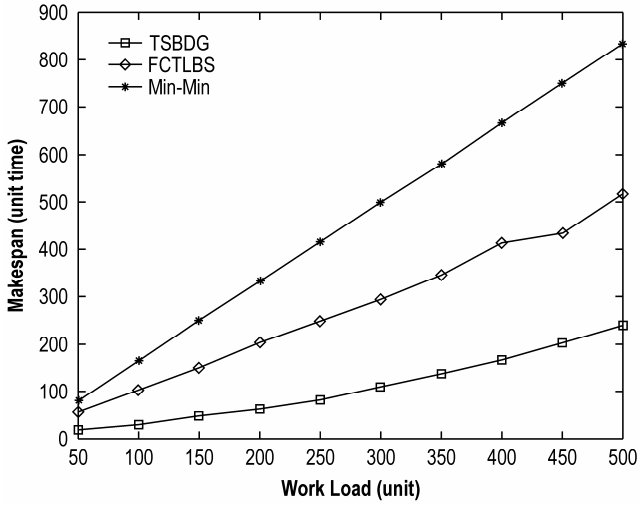


**Fig. 2.** Completion time when the quantity of tasks is 50 to 500

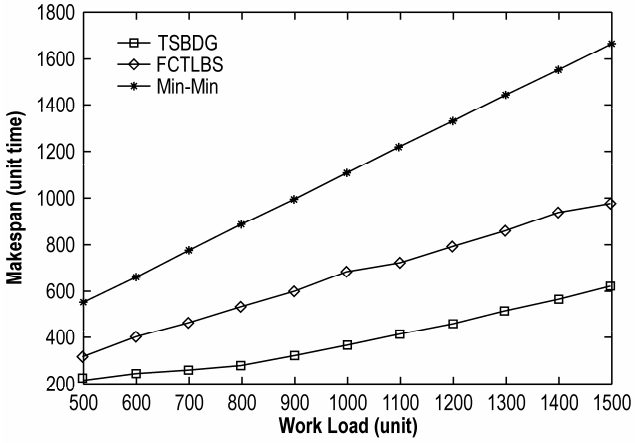


**Fig. 3.** Completion time when the quantity of tasks is 500 to 1500

In the min-min algorithm, the small task was executed first during scheduling. Resources that were frequently assigned tasks caused an imbalance in the processor load so that the completion time of user-submitted tasks increased. In the FCTLBS algorithm, resources were first categorized. Next, task preferences were computed. Then, tasks were assigned to the appropriate resources from among the resources having the correct preference type. Because it found a suitable processor from only resources preferring the corresponding task type, it could not achieve the global optimal solution (i.e., the most appropriate allocation of resources). Furthermore, the resource load imbalance led to an increase in task completion time.

In contrast, the proposed TSBDG algorithm rapidly identified all the resources meeting the minimum requirements of the task in the resource data after gridding. Then, in searching for resources that fulfill the minimum task demands, the global optimal search strategy for resource allocation provided the shortest possible completion time of user-submitted tasks.

### 4.3 Experimental Analysis

The time complexity analysis given in Section 4.1 demonstrates that the proposed method based on improved data gridding has a higher speed of resource matching than the resource comparison method and the method based on resource clustering. In Section 4.2, the results of the simulation experiment showed that the proposed TSBDG algorithm had a better task completion time than the min-min algorithm and FCTLBS algorithm. The results are displayed in Fig. 2 and Fig. 3. Accordingly, during coarse matching of resources by the proposed algorithm, the complexity was small, which narrowed the resource matching time. Meanwhile, the simulation results showed that the proposed algorithm had a shorter task completion time in task scheduling.

## 5 Conclusions

To decrease the search scope and increase the efficiency of resource matching, we proposed a strategy based on improved data gridding to conduct coarse matching of resources. The proposed algorithm could quickly and accurately find distributed system resources that meet the minimum task requirements. Thus, using global optimal searching for processor allocation to minimize the task completion time is an ideal approach to performing task scheduling. Simulation experiments and a process analysis showed that the proposed algorithm guaranteed rapid resource matching and task scheduling, which resulted in the shortest possible completion time of the user-submitted tasks.

## Acknowledgements

This work received financial assistance from Shaanxi Provincial Science and Technology Department Key R&D Plan General Project(No. 2017NY-129), Shaanxi Provincial Department of Education Special

Project (No. 17JK0712) The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

## References

[1] V. Bharadwaj, D. Ghose, T.G. Robertazzi, Divisible load theory: a new paradigm for load scheduling in distributed systems, Cluster Computer 6(1)(2003) 7-17.

[2] F. Ramezani, J. Lu, F. Hussain, Task-based system load balancing in cloud computing using particle swarm optimization, International Journal of Parallel Programming 42(5)(2014) 739-754.

[3] D.S. Yadav, S. Rana, S. Prakash, An efficient resources allocation strategy for survivable WDM network under static lightpath demand, Optik 124(2013) 722-728.

[4] W. Zhao, J. Zhao, S. Zhao, Y. Li, Y. Dong, C. Dong, X. Li, Resources scheduling for data relay satellite with microwave and optical hybrid links based on improved niche genetic algorithm, Optik 125(2014) 3370-3375.

[5] X. Wang, Y. Wang, Y. Cui, A new multi-objective bi-level programming model for energy and locality aware multi-job scheduling in cloud computing, Future Generation Computer System 36(1)(2014) 91-101.

[6] O. Beaumont, L. Eyraud-Dubois, C.T. Caro, H. Rejeb, Heterogeneous resource allocation under degree constraints, IEEE Transaction on Parallel and Distributed Systems 24(5)(2013) 926-937.

[7] H. Zhu, Y. Wang, Constrained multi-objective grid task security scheduling model and algorithm, Journal of Electronics & Information Technology 32(4)(2010) 988-992.

[8] A. Kargarian, Y. Fu, System of systems based security-constrained unit commitment incorporating active distribution grids, IEEE Transaction Power System 29(5)(2014) 2489-2498.

[9] X. Wang, Y. Wang, Y. Cui, Energy and locality aware load balancing in cloud computing, Integrated Computer-Aided Engineering 20(4)(2013) 360-374.

[10] X. Zuo, G. Zhang, W. Tan, Self-adaptive learning pso-based deadline constrain task scheduling for hybrid iaas cloud, IEEE Trans on Automation Science and Engineering 11(2)(2014) 564-573.

[11] S. Adabi, A. Movaghar, A.M. Rahmani, Bi-level fuzzy based advanced reservation of Cloud workflow applications on distributed Grid resources, International Journal of Supercomputing 67(1)(2014) 175-218.

[12] W. Li, Q. Zhang, L. Ping, X. Pan, Cloud Scheduling algorithm based on fuzzy clustering, Journal of Communication 33(3)(2012) 146-154.

[13] X. Zhu, X. Qin, M. Qiu, QoS-aware fault-tolerant scheduling for real-time tasks on heterogeneous clusters, IEEE Transaction on Computers 60(6)(2011) 800-812.

[14] L. Cheng, W. Zhao, P. Han, W. Zhanga, J. Shan, Y. Liu, M. Lia, Building region derivation from LiDAR data using a reversed iterative mathematic morphological algorithm, Optics Communications 28(6)(2013) 244-250.

[15] J. Song, J. Wu, Y. Jiang, Extraction and reconstruction of curved surface buildings by contour clustering using airborne LiDAR data, Optik 126(2015) 513-521.