mathematics

MDPI

Article

Extension of Divisible-Load Theory from Scheduling Fine-Grained to Coarse-Grained Divisible Workloads on Networked Computing Systems

Xiaoli Wang [1,*], Bharadwaj Veeravalli [2], Kangjian Wu [1] and Xiaobo Song [3]

[1] School of Computer Science and Technology, Xidian University, Xi'an 710071, China
[2] Department of Electrical and Computer Engineering, National University of Singapore, 4 Engineering Drive 3, Singapore 119077, Singapore
[3] The 20th Research Institute of China Electronics Technology Group Corporation, Xi'an 710068, China
* Correspondence: wangxiaoli@mail.xidian.edu.cn

Abstract: The big data explosion has sparked a strong demand for high-performance data processing. Meanwhile, the rapid development of networked computing systems, coupled with the growth of Divisible-Load Theory (DLT) as an innovative technology with competent scheduling strategies, provides a practical way of conducting parallel processing with big data. Existing studies in the area of DLT usually consider the scheduling problem with regard to fine-grained divisible workloads. However, numerous big data loads nowadays can only be abstracted as coarse-grained workloads, such as large-scale image classification, context-dependent emotional analysis and so on. In view of this, this paper extends DLT from fine-grained to coarse-grained divisible loads by establishing a new multi-installment scheduling model. With this model, a subtle heuristic algorithm was proposed to find a feasible load partitioning scheme that minimizes the makespan of the entire workload. Simulation results show that the proposed algorithm is superior to the up-to-date multi-installment scheduling strategy in terms of achieving a shorter makespan of workloads when dealing with coarse-grained divisible loads.

Keywords: divisible load; coarse-grained workload; multi-installment scheduling; networked computing; 68W15

Citation: Wang, X.; Veeravalli, B.; Wu, K.; Song, X. Extension of Divisible-Load Theory from Scheduling Fine-Grained to Coarse-Grained Divisible Workloads on Networked Computing Systems. *Mathematics* **2023**, *11*, 1752. https://doi.org/10.3390/math11071752

Academic Editor: Theodore E. Simos

Received: 20 February 2023
Revised: 30 March 2023
Accepted: 5 April 2023
Published: 6 April 2023

1. Introduction

During the past few decades, the volume of available data worldwide has grown at an unprecedented rate. In order to fully mine and analyze such massive data, the subject of "big data" has become a hot topic. Various applications related to big data are still emerging, such as smart cities, smart medical care, intelligent transportation systems, etc. The IMARC Group expects the global big data market to expand from USD 162.6 billion in 2021 to USD 314.1 billion by 2027, at a CAGR of approximately 11% over the forecast period [1]. Like computers and the Internet, big data is very likely to trigger a new round of technological revolution. However, on the other hand, in the face of the explosive growth of data, the computing capacity of a single server has been stretched. Therefore, we urgently need networked computing systems to coordinate the computing power of multiple servers that is able to take on an extremely complex computing workload including big data. In order to efficiently accomplish workload computation and fully utilize compute resources, studying how to schedule workloads reasonably in networked computing systems has become a fundamental problem for researchers.

Although the amount of big data is remarkably immense, many big data workloads can be regarded as divisible loads, even fine-grained ones. That is to say, the total workload can be divided into arbitrary load partitions of any size. There is no interdependence between them, no data transmission is required, and there is no particular order of task

execution. These divisions are supposed to be distributed to multiple servers in networked computing systems by a competent divisible-load scheduling strategy to perform parallel computing, so as to shorten the total makespan. This kind of divisible-load scheduling strategy is collectively referred to as the Divisible-Load Theory (DLT) [2]. DLT has been deeply studied with regard to various topologies of networked computing systems, such as the tree network [3], bus network [4], complete b-Ary network [5], Gaussian, mesh, torus network [6], cloud-computing platform [7], edge-computing architecture [8] and so on.

Existing studies have put forward various divisible-load scheduling models and algorithms to address numerous distinct constraints emerging in a realistic networked computing environment. For instance, the literature [9] considers communication and computation startup overheads in the divisible-load scheduling models and proves that startup overheads have an unneglectable impact on the makespan. The authors of [10] designed a multi-installment scheduling model to deal with servers that have a limited number of memory buffers and cannot hold the whole load package. To address the time-varying issues of computing speeds and transmitting speeds, the authors of [11] proposed two recursive algorithms and one iterative algorithm coping with two different situations: one in which workloads arrive and depart based on a previously known schedule, and one in which nothing occurs. The authors of [12] considered scheduling divisible loads in systems with hierarchical memory, which has different time and energy efficiencies for varying levels of memory. Core memory is time-efficient but too small to hold the whole workload, while external memory is more costly both in terms of time and energy. Hence, a multi-installment scheduling strategy was proposed to avoid using out-of-core storage. One study [13] found that the computation and communication-rate-cheating problem has a considerable impact on the applications of divisible-load scheduling. The authors of [14] studied the influence of available server times on the makespan of divisible loads in networked systems. The authors of [15] presented a next-generation novel multi-cloud architecture with arbitrary server release times. Based on this architecture, a dynamic scheduling strategy was proposed, which allows servers to estimate their ready times for receiving and processing load fractions in order to guarantee load balancing and high performance of the system. Besides server release times, the authors of [16] also considered finite-size buffer capacity when scheduling computationally intensive divisible loads in bus networks. One study [17] designed an efficient scheduling algorithm for fault workload redistribution in response to frequent server failures.

The above studies are all at the theoretical level. In contrast, at the practical level, DLT works spectacularly well in many big-data-related applications, such as real-time video encoding [18], large-scale image processing [19], signature searching [20], data-intensive flow optimization [21], dynamic voltage and frequency regulation [22] and so on. However, successful applications of DLT are all based on fine-grained divisible loads [23–26] igu. For example, in [18], when solving the problem of image processing, an image is regarded as being composed of numerous pixels so that a large-scale image of several gigabytes can be abstracted as a fine-grained divisible workload, and further be divided and allocated on several servers in networked computing systems for parallel processing. However, in recent years, with the diversity of big data applications and the development of deep learning technology, people sometimes need to mine the correlation between data, so workloads cannot simply be divided into independent load partitions and distributed to separate servers for parallel computing. For example, when dealing with the problem of emotional analysis [27], in order to extract and analyze people's opinions on different products or services, it is usually necessary to read through the whole text rather than just relying on a word or character to complete emotional analysis. That is, data in this problem cannot be abstracted into fine-grained divisible loads (a word or a character as a unit) but can only be abstracted into coarse-grained divisible loads (a paragraph or a chapter as a unit). Another intuitive example is that when addressing the problem of image classification [28], it is almost impossible to classify the image correctly by merely observing a single pixel or a part of the image. Similarly, data in this problem can no longer be arbitrarily divided and

processed separately. Hence, this problem should be treated with regard to coarse-grained divisible-load scheduling.

Existing studies have proved that even fine-grained divisible-load scheduling problems in distributed computing systems are NP-hard [29], not to mention coarse-grained load scheduling. Moreover, if we directly apply fine-grained scheduling models on coarse-grained divisible loads and inadvertently assign fragmented load partitions to servers without considering their granularity constraints, it will inevitably lead to partial loads left to be completed. Even if we first round up the load partitions according to coarse granularity and then distribute them, the efficiency of load processing is bound to be affected on networked computing systems. Furthermore, it is likely to cause time conflicts between two adjacent scheduling installments. Based on this background, in this paper, we design a new multi-installment scheduling model for coarse-grained loads with the objective of minimizing the makespan of the entire workload.

The rest of this paper is structured as follows. We first give a mathematical description of the coarse-grained divisible-load scheduling problem on networked computing systems in Section 2, followed by a new multi-installment scheduling model. Section 3 puts forward a heuristic algorithm to obtain a feasible load partitioning scheme, and Section 4 evaluates its performance through rigorous simulation experiments. Section 5 provides the conclusions.

2. Coarse-Grained Divisible-Load Scheduling Model

Consider a networked computing system that consists of $n+1$ servers connected in a star topology, where p_0 denotes the master server and $\{p_1, p_2, \ldots, p_n\}$ represents computing servers. Server p_0 is accountable for dividing workloads and assigning load fractions to servers $\{p_1, p_2, \ldots, p_n\}$ in several installments, while the latter is responsible for load computation. Each server has a constant computation startup overhead, denoted as s_i for server p_i, representing the time it takes to start specific components or programs needed for load computation. It takes server p_i time w_i to compute the unit size of load, so for a workload with size x, it requires time $s_i + xw_i$. The master connects each server via communication links, denoted as $\{l_1, l_2, \ldots, l_n\}$. Link l_i has a constant communication startup overhead, defined as o_i. It takes link l_i time z_i to transmit the unit size of load, and thus it requires time $o_i + xz_i$ to transmit a workload with size x. The master p_0 distributes load fractions to servers in the order of p_1 to p_n. Each server starts computing when it receives the whole part of its assignment from p_0.

Figure 1 shows the Gantt chart for coarse-grained multi-installment scheduling. As can be seen, the whole scheduling process is composed of $m+1$ installments, where the first m installments, termed as internal installments, are uniform. The last installment is different from internal installments as it guarantees all servers finish computing at the same time as much as possible. As for fine-grained divisible loads, to achieve the shortest makespan, all servers involved in workload computation are supposed to complete their computing simultaneously. If not, one can always transfer partial load fractions distributed to the server that completes computation later to the server that finishes computing earlier, thus decreasing the makespan of the entire workload. This is referred to as the *Optimality Principle* in DLT [9]. However, when it comes to coarse-grained divisible loads, the Optimality Principle is no longer applicable. Still, we can find a feasible load partitioning scheme in the last installment to make the completion time of all servers as close as possible.

Suppose the coarse-grained divisible load with size W arrives at the master at time $t = 0$. Let granularity d represent the minimum scale of load partition. For example, d could be the size of an image when dealing with image classification. The master assigns server p_i load $\alpha_i d$ and $\beta_i d$ in every internal installment and the last installment, respectively, where α_i and β_i are load partitioning coefficients that are integers greater than 0; that is, $\alpha_i \in N^+$ and $\beta_i \in N^+$. In each internal installment and the last installment, the total size of load partitions that all servers are supposed to complete is $\lceil W/(m+1)d \rceil$. Table 1 lists the commonly used notations in this paper for the reader's convenience.

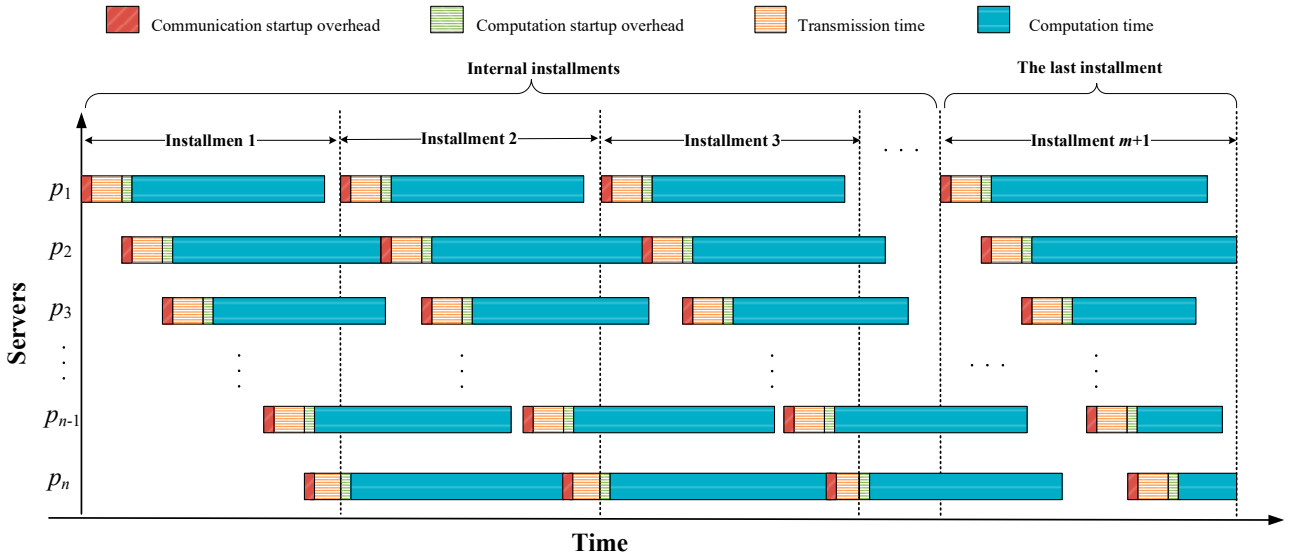

Figure 1. Multi-installment scheduling for coarse-grained divisible loads.

Table 1. Notations and their meanings.

Notations	Meanings
W	Total size of the workload waiting to be processed
n	Total number of computing servers
$\{p_1, p_2, \ldots, p_n\}$	Computing servers
$\{l_1, l_2, \ldots, l_n\}$	Communication links
o_i	Communication startup overhead of server p_i
z_i	The time it takes for link l_i to transmit the unit size of load
s_i	Computation startup overhead of server p_i
w_i	The time it takes for server p_i to complete the unit size of load
d	Granularity d represents the minimum scale of load partition
m	Total number of internal installments
$A = (\alpha_1, \alpha_2, \ldots, \alpha_n)$	The master assigns load $\alpha_i d$ to server p_i in each internal installment, where $\alpha_i \in N^+$
$B = (\beta_1, \beta_2, \ldots, \beta_n)$	The master assigns load $\beta_i d$ to server p_i in the last installment, where $\beta_i \in N^+$
TI_{ij}^{start}	The start time of server p_i in the jth internal installment
TI_{ij}^{end}	The end time of server p_i in the jth internal installment
TL_i^{start}	The start time of server p_i in the last installment
TL_i^{end}	The end time of server p_i in the last installment
T	Makespan of the entire workload
i	A subscript with a value ranging from 1 to n, corresponding to the ith server.
j	A subscript with a value varying from 1 to m, corresponding to the jth installment.

The total makespan depends upon the last server completing its assigned load fractions. The shorter the makespan, the better the scheduling strategy. Therefore, our goal in this paper is to search for a feasible load partitioning scheme that minimizes the makespan of the entire coarse-grained workload.

As for fine-grained divisible loads, there is no time interval for any server between any two adjacent internal installments. By contrast, the load partitioning scheme is subject to integer constraints for coarse-grained loads, and thus internal installments may not be closely connected to each other. Therefore, in order to achieve the shortest makespan of coarse-grained divisible loads, it is necessary to minimize the time gap between any two adjacent internal installments. Let TI_{ij}^{start} and TI_{ij}^{end} be the start and end time of server p_i in the jth internal installment, respectively. As for the first server p_1, the start time of its first installment is $TI_{11}^{start} = 0$, while the start time TI_{1j}^{start} of its jth internal installment depends on two factors: whether it has finished its computation in the $j-1$th installment

and whether the master has finished transmitting load partitions for all servers in the $j-1$th installment. That is,

$$TI_{1,j}^{start} = \max\left\{TI_{1,j-1}^{end},\ TI_{n,j-1}^{start} + o_n + z_n\alpha_n d\right\},\ j = 2,\ \ldots,\ m. \tag{1}$$

The end time of server p_i in the jth internal installment equals its start time in the jth internal installment plus the load transmission time and load computation time, as follows:

$$TI_{1,j}^{end} = TI_{1,j}^{start} + o_1 + s_1 + (z_1 + w_1)\alpha_1 d,\ j = 2,\ \ldots,\ m. \tag{2}$$

Server p_i starts its first installment when its previous server p_{i-1} finishes load transmission from the master. That is,

$$TI_{i,1}^{start} = TI_{i-1,1}^{start} + o_{i-1} + z_{i-1}\alpha_{i-1} d,\ i = 2,\ \ldots,\ n. \tag{3}$$

The start time $TI_{i,j}^{start}$ of server p_i in the jth internal installment is determined by the end time of its previous installment and the time when its previous server finishes receiving its load fraction from the master. Hence, we have

$$TI_{i,j}^{start} = \max\left\{TI_{i,j-1}^{end},\ TI_{i-1,j}^{start} + o_{i-1} + z_{i-1}\alpha_{i-1} d\right\},\ i = 2,\ldots,n,\ j = 2,\ \ldots,\ m. \tag{4}$$

Therefore, the end time TI_{ij}^{end} of server p_i in the jth internal installment is

$$TI_{i,j}^{end} = TI_{i,j}^{start} + o_i + s_i + (z_i + w_i)\alpha_i d,\ i = 2,\ldots,n,\ j = 2,\ \ldots,\ m. \tag{5}$$

The start time TL_1^{start} of the first server p_1 in the last installment should be no earlier than the end time of its internal installments or the time at which the master finishes load transmitting for all servers in the internal installments. That is,

$$TL_1^{start} = \max\left\{TI_{1,m}^{end},\ TI_{n,m}^{start} + o_n + z_n\alpha_n d\right\}. \tag{6}$$

The end time of server p_1 in the last installment equals its start time in the last installment plus the load transmission time and load computation time:

$$TL_1^{end} = TL_1^{start} + o_1 + s_1 + (z_1 + w_1)\beta_1 d \tag{7}$$

Similarly, the start time and end time of server p_i in the last installment are as follows:

$$TL_i^{start} = \max\left\{TI_{i,m}^{end},\ TL_{i-1}^{start} + o_n + z_n\beta_n d\right\},\ i = 2,3,\ldots n. \tag{8}$$

$$TL_i^{end} = TL_i^{start} + o_i + s_i + (z_i + w_i)\beta_i d,\ i = 2,3,\ldots n. \tag{9}$$

The total makespan T depends upon the last server completing its assigned load fractions in the last installment. Hence, we can obtain

$$T(A,B) = \max_{i=1\sim n}\left\{TL_i^{end}\right\} \tag{10}$$

where $A = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ and $B = (\beta_1, \beta_2, \ldots, \beta_n)$.

Based on the above analysis, we build a new multi-installment scheduling model for coarse-grained divisible loads, aiming at a minimum makespan T on networked computing systems:

$$\min_{A,B} T(A,B) = \min_{A,B}\left\{\max_{i=1\sim n}\left\{TL_i^{end}\right\}\right\}, \tag{11}$$

subject to

(a) $\alpha_i \in N^+,\ \beta_i \in N^+.$

(b) $\sum_{i=1}^{n} \alpha_i d \geq \frac{W}{m+1},\ \sum_{i=1}^{n} \beta_i d \geq \frac{W}{m+1}.$

where

(1)

$$\begin{cases} TI_{11}^{start} = 0. \\ TI_{1,j}^{start} = \max\left\{TI_{1,j-1}^{end},\ TI_{n,j-1}^{start} + o_n + z_n\alpha_n d\right\},\ j = 2,\ \ldots,\ m. \\ TI_{1,j}^{end} = TI_{1,j}^{start} + o_1 + s_1 + (z_1 + w_1)\alpha_1 d,\ j = 2,\ \ldots,\ m. \end{cases}$$

(2)

$$\begin{cases} TI_{i,1}^{start} = TI_{i-1,1}^{start} + o_{i-1} + z_{i-1}\alpha_{i-1} d,\ i = 2,\ \ldots,\ n. \\ TI_{i,j}^{start} = \max\left\{TI_{i,j-1}^{end},\ TI_{i-1,j}^{start} + o_{i-1} + z_{i-1}\alpha_{i-1} d\right\},\ i = 2, \ldots, n,\ j = 2,\ \ldots,\ m. \\ TI_{i,j}^{end} = TI_{i,j}^{start} + o_i + s_i + (z_i + w_i)\alpha_i d,\ i = 2, \ldots, n,\ j = 2,\ \ldots,\ m. \end{cases}$$

(3)

$$\begin{cases} TL_1^{start} = \max\left\{TI_{1,m}^{end},\ TI_{n,m}^{start} + o_n + z_n\alpha_n d\right\}. \\ TL_1^{end} = TL_1^{start} + o_1 + s_1 + (z_1 + w_1)\beta_1 d. \end{cases}$$

(4)

$$\begin{cases} TL_i^{start} = \max\left\{TI_{i,m}^{end},\ TL_{i-1}^{start} + o_n + z_n\beta_n d\right\},\ i = 2, 3, \ldots n. \\ TL_i^{end} = TL_i^{start} + o_i + s_i + (z_i + w_i)\beta_i d,\ i = 2, 3, \ldots n. \end{cases}$$

As given in the proposed model, we aim to minimize the total makespan. This model involves two sets of variables: load partitioning scheme $A = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ for internal installments and load partitioning scheme $B = (\beta_1, \beta_2, \ldots, \beta_n)$ for the last installment. Additionally, this model has two specified constraints. The first constraint indicates that every load partition assigned to servers must be a positive integer since the scheduling problem considered in this paper focuses on coarse-grained divisible loads. The second constraint means that the total amount of load completed by all servers in each installment should be at least as large as $W/(m+1)$ because the scheduling process is supposed to be periodic for computational simplicity.

3. Heuristic Scheduling Algorithm

As for fine-grained workloads, one can obtain a closed-form solution to an optimal load partitioning scheme for internal installments based on the assumption that the internal scheduling time of each server is absolutely the same. Meanwhile, one can also obtain a closed-form solution to an optimal load partitioning scheme for the last installment since all servers finish computing simultaneously. However, for coarse-grained loads, the above two conditions for obtaining an optimal load partition cannot be satisfied.

In order to obtain the shortest makespan of coarse-grained divisible loads, the time consumed by each server in every internal installment should be as close as possible. Let U be an upper bound for the completion time of servers in each internal installment. We can obtain a solution to load partitioning scheme $A = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ by looking for a feasible value of U. The upper bound U satisfies

$$o_i + s_i + (z_i + w_i)\alpha_i d \leq U < o_i + s_i + (z_i + w_i)(\alpha_i + 1)d. \tag{12}$$

Rearranging Equation (12) yields

$$\frac{(U - o_i - s_i)}{(z_i + w_i)d} - 1 < \alpha_i \leq \frac{(U - o_i - s_i)}{(z_i + w_i)d}. \tag{13}$$

According to constraints (a) and (b) in the proposed model, we have

$$\sum_{i=1}^{n} \alpha_i = \left\lceil \frac{W}{(m+1)d} \right\rceil, \ \alpha_i \in N^+. \tag{14}$$

Substituting Equation (13) into Equation (14) results in

$$\sum_{i=1}^{n} \left\lceil \frac{U - o_i - s_i}{(z_i + w_i)d} \right\rceil - n < \left\lceil \frac{W}{(m+1)d} \right\rceil \leq \sum_{i=1}^{n} \left\lceil \frac{U - o_i - s_i}{(z_i + w_i)d} \right\rceil \tag{15}$$

Rearranging Equation (15), one can obtain the upper and lower bounds of U as follows:

$$\begin{cases} \underline{U} = \frac{1}{n \sum\limits_{i=1}^{n} \frac{1}{(z_i+w_i)d}} \left(\left\lceil \frac{W}{(m+1)d} \right\rceil + \sum\limits_{i=1}^{n} \frac{o_i+s_i}{(z_i+w_i)d} \right). \\ \overline{U} = \frac{1}{n \sum\limits_{i=1}^{n} \frac{1}{(z_i+w_i)d}} \left(\left\lceil \frac{W}{(m+1)d} \right\rceil + n + \sum\limits_{i=1}^{n} \frac{o_i+s_i}{(z_i+w_i)d} \right). \end{cases} \tag{16}$$

A feasible solution to load partitioning scheme $A = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ that meets the constraint in Equation (14) can be obtained via a binary search of U on interval $\left[\overline{U}, \underline{U}\right]$.

Similarly, let Y be an upper bound for the completion time of servers in the last installment. By searching for the value of Y, one can obtain a feasible solution to load partitioning scheme $B = (\beta_1, \beta_2, \ldots, \beta_n)$. The upper bound Y satisfies

$$TL_i^{end}(\beta_i) \leq Y < TL_i^{end}(\beta_i + 1) \tag{17}$$

where $TL_i^{end}(\beta_i)$ and $TL_i^{end}(\beta_i + 1)$ represent the end times of server p_i in the last installment when it has been assigned load fractions β_i and β_{i+1}, respectively.

By substituting Equation (7) into Equation (17), we obtain

$$TL_1^{start} + o_1 + s_1 + (z_1 + w_1)\beta_1 d \leq Y < TL_1^{start} + o_1 + s_1 + (z_1 + w_1)(\beta_1 + 1)d. \tag{18}$$

Rearranging Equation (18) yields

$$\beta_1 = \left\lfloor \frac{Y - TL_1^{start} - o_1 - s_1}{(z_1 + w_1)d} \right\rfloor \tag{19}$$

Likewise, by bringing Equations (8) and (9) into Equation (17), we shall arrive at

$$\beta_i = \left\lfloor \frac{Y - \max\left\{ TI_{i,m}^{end}, \ TL_{i-1}^{start} + o_n + z_n \beta_n d \right\} - o_i - s_i}{(z_i + w_i)d} \right\rfloor \tag{20}$$

According to constraints (a) and (b) in the proposed model, we have

$$\sum_{i=1}^{n} \beta_i = \left\lceil \frac{W}{(m+1)d} \right\rceil, \ \beta_i \in N^+. \tag{21}$$

Hence, a feasible solution to load partitioning scheme $B = (\beta_1, \beta_2, \ldots, \beta_n)$ that meets the constraint in Equation (21) can be obtained via a binary search of Y on interval $\left[\overline{Y}, \underline{Y}\right] = \left[TI_{n,m}^{end}, \ 2TI_{n,m}^{end}\right]$.

To sum up, via a binary search of U and Y, one can obtain a feasible solution to load partitioning schemes A and B, respectively, thus solving the proposed model. Algorithm 1 shows the framework of this heuristic algorithm in detail.

Algorithm 1 A Heuristic Algorithm for Scheduling Coarse-Grained Divisible Loads

Input: W, m, o_i, s_i, w_i, z_i with $i = 1, 2, \ldots, n$.
Output: A feasible load partitioning scheme $A = (\alpha_1, \alpha_2, \ldots, \alpha_n)$ for internal installments and $B = (\beta_1, \beta_2, \ldots, \beta_n)$ for the last installment.
Step 1: Obtain the upper and lower bounds of U by Equation (16);
Step 2: $U = (\underline{U} + \overline{U})/2$;
Step 3: Obtain load partition A for internal installments by Equation (13);
Step 4: If $\sum_{i=1}^{n} \alpha_i > V$, then $\overline{U} = U$; else if $\sum_{i=1}^{n} \alpha_i < V$, then $\underline{U} = U$;
Step 5: If A does not satisfy Equation (14), go to step 2;
Step 6: $\overline{Y} = 2TI_{n,m}^{end}$, $\underline{Y} = TI_{n,m}^{end}$;
Step 7: $Y = (\underline{Y} + \overline{Y})/2$;
Step 8: Obtain load partition B for the last installment by Equations (19) and (20);
Step 9: If $\sum_{i=1}^{n} \beta_i > V$, then $\overline{Y} = Y$; else if $\sum_{i=1}^{n} \beta_i < V$, then $\underline{Y} = Y$;
Step 10: If B does not satisfy Equation (21), go to step 7;
Step 11: Return A and B.

Steps 1 to 5 in Algorithm 1 obtain a valid load partitioning scheme for internal installments via a binary search of U on interval $\left[\overline{U}, \underline{U}\right]$ given in Equation (16), while steps 6 to 10 obtain this for the last installment via a binary search of Y on interval $\left[\overline{Y}, \underline{Y}\right] = \left[TI_{n,m}^{end}, 2TI_{n,m}^{end}\right]$. The complexity of steps 1, 3, 4, 5, 8 and 9 is $O(n)$,while the complexity of step 6 is $O(nm)$. Therefore, in the best-case scenario, steps 2 to 5 and steps 7 to 10 only cycle once to find an optimal solution, and the overall time complexity of the proposed heuristic scheduling algorithm is $O(n + nm)$.

4. Experiments and Result Analysis

We will compare in this section the proposed model and algorithm with an up-to-date periodic multi-installment scheduling model and algorithm proposed in [30] (abbreviated as PMIS) under the scenarios of coarse-grained workloads. In the following simulations, $n = 15$; that is, there are 15 servers involved in workload computation. The parameters of the heterogenous networked computing system are given in Table 2.

Table 2. Parameters of the heterogeneous networked computing system.

Servers	o_i	s_i	z_i	w_i
p_1	3.66	0.02	0.8	8.63
p_2	7.76	5.62	0.20	8.41
p_3	7.60	1.61	0.11	17.20
p_4	10.18	2.50	0.82	15.37
p_5	1.43	1.52	0.63	13.41
p_6	7.11	3.57	0.12	16.04
p_7	0.67	1.68	0.48	18.35
p_8	8.23	3.28	0.61	6.03
p_9	1.64	1.73	0.87	6.79
p_{10}	6.18	2.48	0.01	9.58
p_{11}	6.89	1.83	0.75	9.04
p_{12}	2.68	4.57	0.38	14.76
p_{13}	3.79	3.18	0.19	5.69
p_{14}	5.17	4.98	0.56	14.81
p_{15}	8.53	3.95	0.32	5.18

We conduct three sets of experiments. In the first one, the size of workloads and its granularity are set to be 10,000 and 8, respectively. That is, $W = 10000$ and $d = 8$. Additionally, we round up the number of load partitions obtained by PMIS to meet the coarse granularity constraints. Figure 2 shows the makespan obtained by the two algorithms with different numbers of installments varying from 1 to 20.

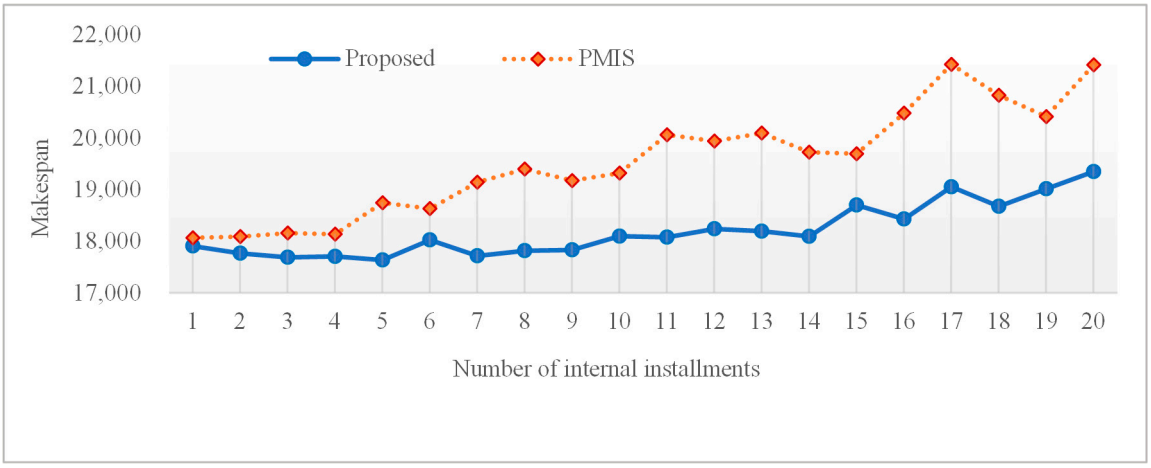

Figure 2. Makespan versus the number of installments.

In Figure 2, the X axis denotes different numbers of internal installments, while the Y axis represents the total makespan. As seen from this figure, the algorithm proposed in this paper outperforms PMIS in solving coarse-grained divisible-load scheduling problems as it can obtain a shorter makespan of workloads. Additionally, it is worth mentioning that for fine-grained workloads, it has been proved in [30] that the makespan first decreases and then increases with an increase in the number of installments. However, one can observe from Figure 2 that for coarse-grained divisible loads, the relationship between the makespan and the number of installments no longer follows the above rule.

Next, we compare the proposed algorithm with PMIS for different sizes of workloads ranging from 9000 to 10,000 with the same granularity, $d = 8$. Figure 3 shows the makespan versus the size of workloads. As can be seen from this figure, the proposed algorithm is superior to the existing algorithm in minimizing the makespan no matter how large the workload is.

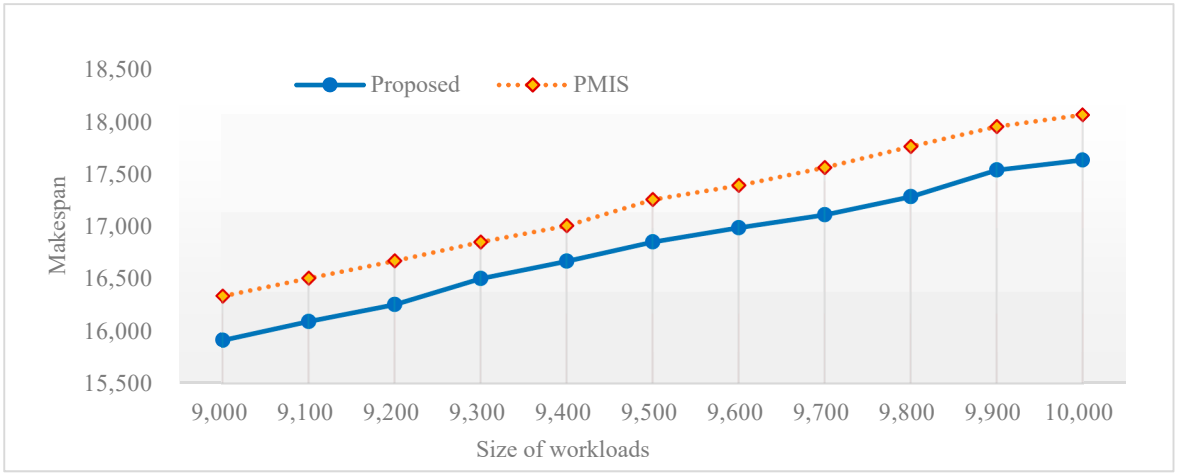

Figure 3. Makespan versus the size of workloads.

Additionally, Figure 4 shows a comparison of the results between the proposed algorithm and PMIS under the same workload size ($W = 10000$) with various coarse granularities varying from 1 to 20. On the one hand, we can observe from this figure that for any granularity, the makespan obtained by the proposed algorithm is shorter than that achieved by PMIS. On the other hand, with an increase in coarse granularity, the makespan shows an upward trend. This is because the greater the granularity, the higher the possibility that the gap between adjacent installments of each server turns out to be large. For fully

fine-grained workloads, the gap reaches the minimum; that is, there is no time gap between any adjacent internal installments. By contrast, for coarse-grained workloads, with an increase of granularity, the time gap between internal installments may become larger, and additionally, the differences in the end times of servers in the last installment may widen.

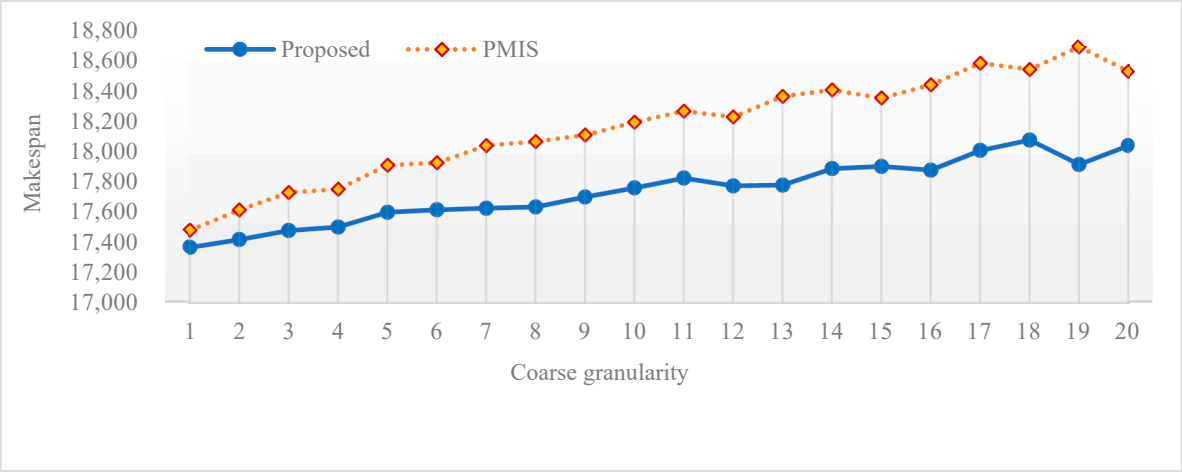

Figure 4. Makespan versus coarse granularity.

5. Conclusions

In this paper, we first described a normalized mathematical description of the coarse-grained divisible-load scheduling problem under heterogeneous networked computing systems. To solve this problem, we established a new multi-installment scheduling model with the objective of achieving a minimized makespan of workloads. The scheduling process comprises two distinct parts: multiple internal installments and the last installment. Taking full advantage of coarse granularity, we obtained valid variation ranges for the upper bounds of the completion time of servers in each internal installment and the last installment through strict mathematical derivation. Through a binary search, a feasible load partitioning scheme could be found efficiently. Finally, simulation results show that the proposed algorithm is superior to the existing scheduling algorithm in obtaining a shorter makespan when dealing with coarse-grained workloads under various conditions. Extensions of this work could consider two aspects. The first is how to find an optimal number of installments for coarse-grained workloads, and the second is how to accelerate the convergence speed of the proposed heuristic algorithm. These are expected to be open-ended problems at this juncture.

Author Contributions: Conceptualization, X.W. and B.V.; methodology, X.W. and B.V.; software, K.W.; data curation, X.S.; writing—original draft preparation, X.W. and K.W.; writing—review and editing, B.V.; funding acquisition, X.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China (No. U22A2098, No. 62172457, No. 62272367), the Key Research and Development Program of Shaanxi Province (No. 2022ZDLGY01-06 and No. 2022ZDLGY01-01) and the Key Laboratory of Cognitive Radio and Information Processing, Ministry of Education, Guilin University of Electronic Technology (CRKL220206).

Data Availability Statement: Not applicable.

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

References

1. IMARC Group. Big Data Software Market: Global Industry Trends, Share, Size, Growth, Opportunity and Forecast 2022–2027. 2021. Available online: https://www.imarcgroup.com/big-data-software-market (accessed on 20 February 2023).
2. Yang, C.; Fei, W.; Robertazzi, T. Integrating Amdahl-like Laws and Divisible Load Theory. *Parallel Process Lett.* **2021**, *31*, 2150008.
3. Ghanbari, S.; Othman, M.; Bakar, M.R.A.; Leong, W.J. Multi-objective method for divisible load scheduling in multi-level tree network. *Future Gener. Comput. Syst.* **2015**, *54*, 132–143. [CrossRef]
4. Carroll, T.E.; Grosu, D. A Strategyproof Mechanism for Scheduling Divisible Loads in Bus Networks without Control Processors. In Proceedings of the 20th IEEE International Parallel & Distributed Processing Symposium, Rhodes Island, Greece, 25–29 April 2006; pp. 1–8.
5. Chen, C.Y. Divisible Nonlinear Load Distribution on Complete b-Ary Trees. *IEEE Trans. Aerosp. Electron. Syst.* **2020**, *56*, 998–1013. [CrossRef]
6. Zhang, Z.; Robertazzi, T.G. Scheduling Divisible Loads in Gaussian, Mesh and Torus Network of Servers. *IEEE Trans. Comput.* **2015**, *64*, 3249–3264. [CrossRef]
7. Wu, Z.; Sun, J.; Zhang, Y.; Zhu, Y.; Li, J.; Plaza, A.; Benediktsson, J.A.; Wei, Z. Scheduling-Guided Automatic Processing of Massive Hyperspectral Image Classification on Cloud Computing Architectures. *IEEE Trans. Cybern.* **2020**, *51*, 588–3601. [CrossRef]
8. Chen, W.; Zhu, Y.; Liu, J.; Chen, Y. Enhancing Mobile Edge Computing with Efficient Load Balancing Using Load Estimation in Ultra-Dense Network. *Sensors* **2021**, *21*, 3135. [CrossRef]
9. Bharadwaj, V.; Li, X.; Chi, C.K. Design and analysis of load distribution strategies with start-up costs in scheduling divisible loads on distributed networks. *Math. Comput. Model.* **2000**, *32*, 901–932. [CrossRef]
10. Berlińska, J.; Drozdowski, M. Heuristics for multi-round divisible loads scheduling with limited memory. *Parallel Comput.* **2010**, *36*, 199–211. [CrossRef]
11. Fei, W.; Yang, C.; Robertazzi, T. Optimal Divisible Load Scheduling for Resource-Sharing Network. *arXiv* **2019**, arXiv:1902.01898.
12. Marszalkowski, J.; Drozdowski, M.; Singh, G. Time-energy trade-offs in processing divisible loads on heterogeneous hierarchical memory systems. *J. Parallel Distrib. Comput.* **2020**, *144*, 206–219. [CrossRef]
13. Ghanbari, S.; Othman, M. Time Cheating in Divisible Load Scheduling: Sensitivity Analysis, Results and Open Problems. *Procedia Comput. Sci.* **2018**, *125*, 935–943. [CrossRef]
14. Hu, M.; Veeravalli, B. Requirement-Aware Strategies with Arbitrary Processor Release Times for Scheduling Multiple Divisible Loads. *IEEE Trans. Parallel Distrib. Syst.* **2011**, *22*, 1697–1704. [CrossRef]
15. Kang, S.; Veeravalli, B.; Aung, K. Dynamic scheduling strategy with efficient node availability prediction for handling divisible loads in multi-cloud systems. *J. Parallel Distrib. Comput.* **2018**, *113*, 1–16. [CrossRef]
16. Veeravalli, B.; Barlas, G. Scheduling Divisible Loads with Processor Release Times and Finite Size Buffer Capacity Constraints in Bus Networks. *Cluster Comput.* **2003**, *6*, 63–74.
17. Tong, W.; Xiao, S.; Li, H. Fault-Tolerant Scheduling Algorithm with Re-allocation for Divisible Loads on Homogeneous Distributed System. *IAENG Int. J. Comput. Sci.* **2018**, *45*, 450–457.
18. Li, P.; Veeravalli, B.; Kassim, A. Design and implementation of parallel video encoding strategies using divisible load analysis. *IEEE Trans. Circuits Syst. Video Technol.* **2005**, *15*, 1098–1112.
19. Aali, S.N.; Bagherzadeh, N. Divisible load scheduling of image processing applications on the heterogeneous star and tree networks using a new genetic algorithm. *Concurr. Comput.* **2020**, *10*, 1–15.
20. Ying, Z.; Robertazzi, T.G. Signature Searching in a Networked Collection of Files. *IEEE Trans. Parallel Distrib. Syst.* **2014**, *25*, 1339–1348. [CrossRef]
21. Zhang, J.; Shi, L.; Liu, Y.; Robertazzi, T.G. Optimizing Data Intensive Flows for Networks on Chips. *Parallel Process. Lett.* **2021**, *31*, 2150013. [CrossRef]
22. Yu, H.; Ha, Y.; Veeravalli, B.; Chen, F.; El-Sayed, H. DVFS-Based Quality Maximization for Adaptive Applications with Diminishing Return. *IEEE Trans. Comput.* **2020**, *70*, 803–816. [CrossRef]
23. Tan, X.; Golikov, P.; Vijaykumar, N.; Pekhimenko, G. GPUPool: A Holistic Approach to Fine-Grained GPU Sharing in the Cloud. In Proceedings of the 31st International Conference on Parallel Architectures and Compilation Techniques, Chicago, IL, USA, 8–12 October 2022; pp. 317–332.
24. Li, Y.; Wen, M.; Fei, J.; Shen, J.; Cao, Y. A Fine-Grained Modeling Approach for Systolic Array-Based Accelerator. *Electronics* **2022**, *11*, 2928. [CrossRef]
25. Souri, A.; Zhao, Y.; Gao, M.; Mohammadian, A.; Shen, J.; Al-Masri, E. A Trust-Aware and Authentication-Based Collaborative Method for Resource Management of Cloud-Edge Computing in Social Internet of Things. *IEEE Trans. Comput. Soc. Syst.* **2023**, *early access*. [CrossRef]
26. Choudhary, A.; Govil, M.; Singh, G.; Awasthi, L.; Pilli, E. Energy-aware scientific workflow scheduling in cloud environment. *Cluster Comput.* **2022**, *25*, 3845–3874. [CrossRef]
27. Birjali, M.; Kasri, M.; Beni-Hssane, A. A comprehensive survey on sentiment analysis: Approaches, challenges and trends. *Knowl. Based Syst.* **2021**, *226*, 107134. [CrossRef]
28. Sharma, S.; Guleria, K. Deep learning models for image classification: Comparison and applications. In Proceedings of the IEEE International Conference on Advance Computing and Innovative Technologies in Engineering, Greater Noida, India, 28–29 April 2022; pp. 1733–1738.

29. Wang, X.; Veeravalli, B.; Song, J. Multi-Installment Scheduling for Large-Scale Workload Computation with Result Retrieval. *Neurocomputing* **2021**, *458*, 579–591. [CrossRef]
30. Wang, X.; Veeravalli, B. Performance Characterization on Handling Large-Scale Partitionable Workloads on Heterogeneous Networked Compute Platforms. *IEEE Trans. Parallel Distrib. Syst.* **2017**, *28*, 2925–2938. [CrossRef]