applied sciences

MDPI

Article

Highly Reliable and Efficient Three-Layer Cloud Dispatching Architecture in the Heterogeneous Cloud Computing Environment †

Mao-Lun Chiang [1], Yung-Fa Huang [1,*], Hui-Ching Hsieh [2] and Wen-Chung Tsai [1]

[1] Department of Information and Communication Engineering, Chaoyang University of Technology, Taichung City 41349, Taiwan; mlchiang@cyut.edu.tw (M.-L.C.); azongtsai@cyut.edu.tw (W.-C.T.)
[2] Department of Information Communication, Hsing Wu University, New Taipei City 24452, Taiwan; luckyeva.hsieh@gmail.com
* Correspondence: yfahuang@cyut.edu.tw; Tel.: +886-4-2332-3000 (ext. 4419)
† This paper is an extended version of our paper published in 2017 IEEE 8th International Conference on Awareness Science and Technology (iCAST2017), Taichung, Taiwan, 8–10 November 2017.

Received: 21 July 2018; Accepted: 13 August 2018; Published: 16 August 2018

Abstract: Due to the rapid development and popularity of the Internet, cloud computing has become an indispensable application service. However, how to assign various tasks to the appropriate service nodes is an important issue. Based on the reason above, an efficient scheduling algorithm is necessary to enhance the performance of the system. Therefore, a Three-Layer Cloud Dispatching (TLCD) architecture is proposed to enhance the performance of task scheduling. In the first layer, the tasks need to be distinguished into different types by their characters. Subsequently, the Cluster Selection Algorithm is proposed to dispatch the tasks to appropriate service clusters in the second layer. Besides this, a new scheduling algorithm is proposed in the third layer to dispatch the task to a suitable server in a server cluster to enhance the scheduling efficiency. Basically, the best task completion time can be obtained in our TLCD architecture. Furthermore, load balancing and reliability can be achieved under a cloud computing network environment.

Keywords: cloud computing; reliability; load balancing; Sufferage; task dispatching

1. Introduction

Due to the rapid development and popularity of the Internet, cloud computing has become an indispensable and highly demanded application service [1]. In order to meet the greater storage requirements of Big Data, the system will continue to upgrade and expand its capabilities, resulting in a large number of heterogeneous servers, storage, and related equipment in a cloud computing environment.

The server type of cloud computing can be classified into three service types: Software as a Service (SaaS), Platforms as a Service (PaaS), and Infrastructure as a Service (IaaS) [2–4]. In SaaS, the frequently used software and programs are provided by the Internet vendor, such as Gmail and Google Maps [5]. In PaaS, a platform for users is provided for programs, such as the Google APP Engine, one kind of PaaS platform. In the last type of service, the IaaS, the user can rent the hardware resources to build their own framework, such as a Cloud server [2–4,6]. No matter what kind of cloud service is applied to the cloud computing network, there is a common character: each cloud server has different capabilities and computing abilities. According to the reason above, there is a need for an efficient scheduling algorithm that can dispatch tasks to the appropriate cloud server, which is an important challenge in current cloud computing environments.

Basically, in the traditional cloud clustering architecture, the system only considers the heterogeneity of tasks while executing scheduling procedures and ignores the heterogeneity of tasks which come from different platforms, and its categories are not distinguishable by nodes. Therefore, heterogeneous tasks have become major flaws in traditional cloud architectures. In addition, cloud service types are very diverse. As a result, the cloud computing environment becomes complicated and the reliability is greatly reduced. So, the scheduling of cloud environments is more difficult.

Therefore, the Three-Layer Cloud Dispatching (TLCD) architecture is proposed to handle the scheduling problem while heterogeneous nodes and tasks exist in the cloud system at the same time. For the first layer, a Category Assignment Cluster (CAC) [7,8] layer was proposed to reduce the task delay and the overloading by classifying the heterogeneous tasks. In the CAC layer, the various tasks can be classified into three types according to the IaaS, SaaS, and PaaS categories. Subsequently, the homogeneous tasks can be dispatched to the corresponding service category clusters in the following layer.

In the second layer, called the Cluster Selection (CS) layer, the homogenous tasks can be assigned to appropriate clusters by the Cluster Scheduling Algorithm (CSA) to enhance the reliability of the system. Besides this, the cost and completion time of task scheduling can be reduced in this layer. Finally, tasks can be dispatched to service nodes by the scheduling algorithm in the third layer, the Server Nodes Selection (SNS) layer. In this layer, an Advanced Cluster Sufferage Scheduling (ACSS) algorithm is proposed to enhance resource utilization and to achieve load balancing [2,3,9–11].

The rest of this paper is organized as follows. Section 2 illustrates the previous works on scheduling algorithms. Subsequently, the details of the TLCD architecture are shown in Section 3 and examples are given in Section 4. Finally, Section 5 gives the conclusions of this paper.

2. Materials and Methods

So far, many scheduling algorithms been proposed for various application scenarios [12–14]. For example, in [12], the authors propose an energy-efficient adaptive resource scheduler architecture for providing real-time cloud services to Vehicular Clients. The main contribution of the proposed protocol is maximizing the overall communication-plus-computing energy efficiency while meeting the application-induced hard Quality of Service (QoS) [14] requirements on the minimum transmission rates, maximum delays, and delay jitters. In [14], the authors propose a joint computing-plus-communication optimization framework exploiting virtualization technologies to ensure users the QoS, to achieve maximum energy savings, and to attain green cloud computing goals in a fully distributed fashion. Basically, these proposed algorithms and architectures can help with rethinking the design of scheduling algorithms in depth.

Generally, there are two types of scheduling algorithm can be provided based on the time of scheduling, namely, the real-time type and the batch type. Basically, the received tasks are immediately assigned to cloud server nodes in the real-time type. Conversely, in the batch type, the received tasks are accumulated for a period of time and subsequently dispatched to cloud server nodes. A scheduling algorithm of the batch type can obtain better performance than one of the real-time type. This is because the assignment results of all tasks can be considered in a batch-based scheduling algorithm [11,15,16].

So far, many scheduling algorithms [2,3,9–11,15,17] have been proposed to dispatch the task to cloud server nodes in the cloud computing network, such as the Min-Min [9,10,15,17], Max-Min [9,15,17], Sufferage [2,9,11,17], and MaxSufferage algorithms [2,3,9,15]. However, these algorithms only consider the factor of the expected completion time (*ECT*) without considering the load status of the server node. Therefore, the performance is not as good as expected, and the minimum completion time cannot be obtained.

For example, in the Min-Min algorithm, the task i with the minimum *ECT* of the unassigned tasks T is called *min_ECT*. Subsequently, the task which has minimum *ECT* can be elected and dispatched to corresponding server node. Then, the task which is newly matched is eliminated from a set T and the

procedure is repeated until all tasks are dispatched. Under this circumstance, the workload will easy to unbalance while there are too many tasks waiting to be scheduled.

In the Max-Min algorithm, the *ECT* is also to be used for dispatching tasks. Conversely, the task with overall maximum *ECT* in Max-Min algorithm is always to be selected to assign; thus, the overall *ECT* will be increased significantly [9,15,17]. Besides this, both of above algorithms easily cause higher-capacity server nodes to be assigned more tasks than lower-capacity server nodes. The workloads of cloud server nodes are unbalanced and inefficient. Therefore, an improvement algorithm, the Sufferage algorithm [1,17–19], was proposed to reduce the workloads of the cloud server nodes. In the Sufferage algorithm, the Sufferage Value (*SV*), which is calculated by the second earliest *ECT* minus the earliest *ECT*, is used as an estimated factor to dispatch the task. Then, the task which has largest *SV* value can be selected and dispatched to the corresponding cloud server node which has minimum *ECT*. However, the Sufferage algorithm cannot obtain efficient performance when the number of waiting tasks is very large.

Therefore, the MaxSufferage algorithm [3], which is improved from the Sufferage algorithm, has three phases to solve the above problem. At first, the SV values of all tasks need to be calculated in the SV_i calculation phase. Subsequently, in the MSV_i calculation phase, the second earliest *ECT* of task i which has the maximum *SV* value will be elected as the MaxSufferage Value (*MSV*) value. In the final phase, the task dispatch phase, task i which has maximum *SV* can be dispatched to corresponding server node j when the ECT_{ij} of server node j is less than MSV_i. Conversely, task i with maximum ECT_{ij} value can be dispatched to server node j. However, in this algorithm, the large tasks are easily dispatched to low-capability server nodes under the heterogeneous environments.

For solving the problem above, the Advanced MaxSufferage (AMS) algorithm [20] is proposed to improve this drawback of the MaxSufferage algorithm. However, the AMS only considers the task scheduling of service nodes, regardless of the cluster and type of service. As a result, an incremental algorithm is proposed to solve the scheduling service types, clusters, and service nodes simultaneously. In addition, all tasks can be dispatched to the appropriate server nodes in the cloud computing network even if the server nodes are in a heterogeneous environment.

Subsequently, our algorithm is described and explained as follows.

3. Three-Layer Cloud Dispatching Architecture

The traditional dispatch algorithm of cluster architecture does not dispatch tasks by the capacity of clusters, which may cause the drawbacks of task delay, low reliability, and high makespan. Therefore, a Three-Layer Cloud Dispatching (TLCD) architecture and related scheduling algorithm are proposed for application to cluster-based cloud environments, as shown in Figure 1. Before introducing the detail of the proposed protocol, all notation and their descriptions as used in the algorithm are organized in Table 1.

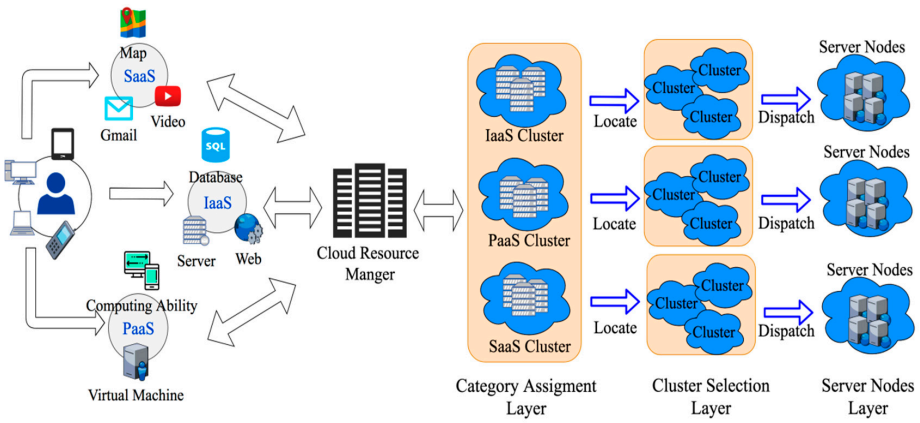

Figure 1. Three-Layer Cloud Dispatching Architecture.

Table 1. The descriptions of notation in the cluster selection layer.

Notations	Description
N	The total number of tasks
L	The total number of clusters
k	The cluster k
n_k	The number of tasks assigned to the k cluster
T_r	The threshold value of the overall reliability
T_c	The threshold value of the overall cost
A_i	The assignment i
R_k	The reliability of the cluster k.
M_k	The makespan of the cluster k
C_k	The cost of the cluster k

In the cloud computing environment, improving the accuracy of cloud service search is challenging [7]. Thus, to improve the overall performance, the first thing in the proposed protocol is identifying cloud service categories. Based on [7], the system will randomly select a cloud service task as the core task of the first cluster. After that, the similarities between this randomly selected cloud service task and other cloud service tasks will be calculated. Here, all cloud service tasks which have larger similarity scores than the predefined threshold will be added to the first cluster. Then these tasks will be removed from the candidate set of cloud service tasks. At this time, the system will continue to select another core task randomly from the remaining cloud service tasks to generate a second cloud cluster. Again, the system will apply a similar selection process to add similar cloud service tasks to this cluster. The selection process will repeat until all cloud service tasks are categorized into a cluster. After identifying and categorizing the tasks, all classified tasks will continuously be executed in the proposed TLCD architecture. The details of TLCD are shown as follows.

3.1. Category Assignment Cluster Layer

The traditional cluster architecture collects and distributes tasks to the cluster by way of cloud resource managers. However, the allocation process may be affected by cluster heterogeneity, causing the task to be insignificant in terms of scheduling. This is because that the tasks are allocated to an idle cluster by cloud resource managers. Therefore, the scheduling result is not ideal. This will increase the complexity of the cloud computing system.

Besides this, the diversity of tasks increases the delay in processing time. To reduce the delay and the complexity of scheduling, the heterogeneity task can be classified into different categories according to demand defined in the Category Assignment Cluster (CAC) layer [7,8]. The category cloud clusters can be divided into three types: SaaS, PaaS, and IaaS. Through these three categories of classification, the difficulty of scheduling heterogeneous tasks and scheduling delays can be reduced.

3.2. Cluster Selection Layer

After completing the classification of the category assignment cluster layer, the classified tasks can be dispatched to the corresponding category cluster. Subsequently, a Cluster Selection Algorithm (CSA) is proposed to assign the tasks to appropriate clusters by using the factors of Reliability (R_i), Cost (C_i), and MakeSpan (M_i) [3,21], as shown in Equations (1)–(3). MakeSpan is the length of time to complete a task. Basically, when the MakeSpan value increases, the system will need longer operation time. Due to the similarity of the fault tolerance of clusters, the computing power will increase as the Reliability gets higher. Therefore, we need to focus on clusters' computing power [3,21]. Finally, the cost factor is defined as the cost needed for a task to be sent and responded to. When taking these three factors into account, tasks can be assigned to suitable clusters and the system efficiency can be enhanced. In addition, users and service providers can customize those three factors based on their own requirements. According to the above description, in the following example, we customize the

Reliability (R_i) and Cost (C_i), and arrange the tasks to the suitable clusters. The reliability and the cost of assignment i are expressed by

$$R_i = (\sum_{k=1}^{C} R_k n_k)/N \quad (1)$$

and

$$C_i = \left(\sum_{k=1}^{L} C_k n_k \right) \quad (2)$$

respectively. Moreover, the makespan of assignment i is expressed by

$$M_i = (\sum_{k=1}^{C} M_k n_k)/N \quad (3)$$

Subsequently, we propose an example to explain Algorithm 1. In Line (4) of Algorithm 1, we arrange the combination of tasks in all clusters. The cluster will choose an appropriate task combination and then help the node to adjust these tasks. Furthermore, Lines (5) to (8) are proposed to check if the R_i and C_i of each assignment i agree with $R_i \geq T_r$ && $C_i \geq T_c$. Then, among those assignments, the one with the smallest M_i is scheduled. If there are more than two eligible groups, we compare R_i and M_i and choose A_i as the combination of the highest reliability and the least time.

In CSA, users can customize the quality of services by reliability, cost, and MakeSpan factors. Thus, algorithms can meet the requirements of various users and can enhance the efficiency of job scheduling. Subsequently, an example is shown to explain the CSA algorithm and the related assumptions are shown in Table 2.

Algorithm 1. Cluster Selection Algorithm

```
1: for total tasks N
2:   for total clusters L
3:     get R_k, M_k, and C_k of each cluster k
4:     Arrange all tasks in the cluster and assign the number of A_i
5:       for calculating the R_i, C_i, and M_i of the assignment A_i
6:         if R_i ≥ T_r && C_i < T_c in assignment A_i then A_i is candidate assignment
7:       end for
8:       choose the smallest M_i in candidate assignment A_i
9:   end for
10: end for
11:End
```

Table 2. Example of Cluster Selection Algorithm.

	Cluster 1			**Cluster 2**			**Cluster 3**			(R_i, C_i, M_i)
	R_k	C_k	M_k	R_k	C_k	M_k	R_k	C_k	M_k	
	(22,25,30)			**(24,28,36)**			**(25,32,15)**			
A_1		5			2			3		(23.3, 242, 26.7)
A_2		8			2			0		(22.4, 256, 31.2)
A_3		4			4			2		(23.4, 276, 29.4)
A_4		3			6			1		(23.5, 275, 32.1)
A_5		5			1			4		(23.4, 281, 24.6)
A_6		10			0			0		(22, 250, 30)

We assumed that there are 10 tasks need to be sent to 3 cloud clusters; the process is the following:
Step 1: T_r and T_c are set by the user. In this example, T_r and T_c are 22 and 260, respectively.
Step 2: Calculate R_i, C_i, and M_i of each allocation combination.

According to Table 2, assignment A_1 assigns five tasks to Cluster 1, two tasks to Cluster 2 and three tasks to Cluster 3. We used Equations (1)–(3) to calculate the average of R_i, C_i, and M_i, and the results are $R_1 = \frac{22\times5+24\times2+25\times3}{10} = 23.3$; $C_1 = 25 \times 5 + 28 \times 2 + 32 \times 3 = 242$; $M_1 = \frac{30\times5+36\times2+15\times3}{10} = 26.7$.

The same procedure is followed for the other assignments, too.

Step 3: Select the schedule that meets the condition of $R_i \geq 22$ and $C_i < 260$; Here, A_1, A_2, and A_6 are selected.

Step 4: According to the conditions of Step 3, we choose A_1 with the highest reliability and smallest MakeSpan. Since the result of assignment A_1 is better than others, assignment A_1 is elected as the combination to dispatch in this example.

The above procedure is the most suitable solution when the MakeSpan is the main concern. However, when the reliability is the main concern, the MakeSpan and Cost become the masking factors to filter out the schedule with the best reliability. After finished the CSA layer, the tasks can be dispatched to the corresponding clusters in the cloud cluster section layer. Subsequently, the appropriate server nodes need to be elected to complete the task in the next layer.

3.3. Server Nodes Selection Layer

After finishing the first two layers, the homogeneous tasks can be dispatched to homogeneous cloud clusters. However, the cloud workload can be unbalanced when a large number of tasks exits due to inappropriate task assignment. Based on the reasons above, this paper proposed a novel algorithm, the Advanced Cluster Sufferage Scheduling (ACSS), to solve the drawback of the MaxSufferage algorithm [12,17,18], especially in the heterogeneous environment. In ACSS, each task can be assigned to the appropriate server nodes by S_j which is calculated from the average *ECT* of the server node to reduce the influence of inappropriate assignment. The notation and details of the ACSS algorithm are shown in Table 3 and Algorithm 2.

In general, there are three phases in the ACSS algorithm. At first, the $EECT_i$ (the earliest expected completion time) and $SEECT_i$ (the second earliest expect completion time) of the S_j are found in order to calculate the *SV* value in the SV_j calculation phase; the detailed process is shown in Lines (8) and (9) in the ACSS algorithm. Then, the MSV value will be set to the second earliest *ECT* value of task *i* in the second phase, which is called the MSV_i calculation phase, while task *i* has the maximum SV value among all *SV* values.

In the final phase, the task dispatching phase, task *i* will be dispatched to S_j while $MSV_i > ECT_{ij}$ and $EECT_i > AECT_j$ of S_j. Conversely, when $EECT_i < AECT_j$, task *i* can be dispatched to server node *j* where the ECT_i is approximate to $AECT_j$ and the ECT_i needs to be larger than $AECT_j$. As a result, the main concept is different from those of previous algorithms; the detail of the algorithm is shown in Line (11) and (12) of Algorithm 2.

Table 3. The notations of ACSS algorithm in the server nodes selection layer.

TCT_{ij}	The task completion time of the task *i* in the server node S_j
ECT_{ij}	The expected completion time of task *i* in the server node S_j
r_j	The expected time server node S_j will become ready to execute for next task
$AECT_j$	The average expected completion time in the server node S_j
$EECT_i$	The earliest expected completion time of task *i*
$SEECT_i$	The second earliest expected completion time of task *i*
SV_i	The Sufferage Value of task *i*
MSV_i	The MaxSufferage Value of task *i*

Algorithm 2. Advanced Cluster Sufferage Scheduling

1: for all unassigned tasks i
2: for all server nodes S_j
3: $TCT_{ij} = ECT_{ij} + r_j$
4: do scheduling for all job assignments
5: mark all server nodes as unassigned;
6: for each task i in S_j
7: find server node S_j that gives the earliest completion time;
8: calculate the Sufferage Value ($SV = EECT_i - SEECT_i$);
9: If the maximum value of SV_i has two or even more the same
then choose the assignment $SEECT_i$ to MSV with maximum;
else the assignment i with maximum $SECT_i$ can be compared to other $EECT_i$;
10: *end if*;
11: If ($MSV_i < ECT_{ij}$ *of* S_j)
then the task i with maximum ECT can be dispatched to server nodes S_j;
12: else if ($MSV_i > ECT_{ij}$) && ($EECT_i > AECT_j$ *of* S_j)
then the task i can be dispatched to server nodes S_j;
13: else if ($MSV_i > ECT_{ij}$) && ($EECT_i < AECT_j$ *of* S_j)
then assignment task $i > AECT_{j\_AVG}$ *of* S_j && *task* $i \approx AECT_j$ *of* S_j can be
dispatched to server nodes S_j;
14: end if;
15: end for
16: end do
17: end for
18: $r_j = r_j + ECT_{ij}$
19: **update** $TCT_{ij} = ECT_{ij} + r_j$
20: end for
21:End

Based on the procedure above, the completion time and load balancing of the workload can be reduced efficiently in the heterogeneous cloud computing network. For the security issue, we can employ a Message Authentication Code (MAC) algorithm [18,22] with a session key to confirm that the message truly comes from the sender and has not been modified. Subsequently, the example is provided to help understanding of the ACSS algorithm in the server node selection layer.

4. Example and Comparison Results

In this section, there are four heterogeneous environments that can be discussed, such as HiHi (High heterogeneity task, High heterogeneity server node), LoLo, HiLo, and LoHi [8,19]. In general, HiHi is the most complex case among all of environments. As a result, an example of task assignment including four server nodes and twelve tasks is discussed under the HiHi heterogeneous environment. At first, the SV value and the MSV value can be calculated in the SV_j calculation phase and the MSV_i calculation phase separately. Subsequently, the $AECT_j$ of S_j is calculated in the task dispatching phase. In addition, the related parameters of the HiHi environment are shown in Table 4.

Table 4. Illustration of the expected execution times of tasks.

	Node A	**Node B**	**Node C**	**Node D**
Task *a*	22345	23526	24323	25328
Task *b*	16667	17930	18696	20187
Task *c*	31083	31897	32034	34678
Task *d*	24712	25156	25774	26330
Task *e*	17018	18069	18664	20348
Task *f*	12050	13289	14911	15918
Task *g*	19035	21486	22371	24031
Task *h*	13911	14602	15655	17678
Task *i*	8016	8536	8948	9685
Task *j*	13618	14596	15331	16558
Task *k*	27861	28156	28655	29791
Task *l*	43336	44731	46777	49659

4.1. The SVi Calculation Phase

Step 1. List the expected execution times for all tasks i on S_j, as shown in Table 3.

Step 2. Calculate the SV value for each task. For example, the SV value of Task a is equal to $SEECT_i$ minus $EECT_i$, which is 23526 − 22345 = 1181, and the same procedures will be executed for tasks b to l to calculate the SV values. The calculation results are shown in Table 5.

Table 5. Calculation of the SV values of tasks.

	Node A	**Node B**	***SV***
Task *a*	22345	23526	1181
Task *b*	16667	17930	1263
Task *c*	31083	31897	814
Task *d*	24712	25156	444
Task *e*	17018	18069	1051
Task *f*	12050	13289	1239
Task *g*	19035	21486	2451
Task *h*	13911	14602	691
Task *i*	8016	8536	520
Task *j*	13618	14596	978
Task *k*	27861	28156	295
Task *l*	43336	44731	1395

4.2. The MSVi Calculation Phase

Step 1. The second earliest ECT value of task i can be selected as the MSV value while task i has the maximum SV_i value among all SV values. As shown in Table 6, the task with the largest SV_i value is task g; thus, the second earliest ECT (here, it is 21486) of ECT_{aB} is selected as the MSV value.

Table 6. Calculation of the MSV values of tasks.

	Node A	**Node B**	***SV***	***MSV***
Task *a*	22345	23526	1181	-
Task *b*	16667	17930	1263	-
Task *c*	31083	31897	814	-
Task *d*	24712	25156	444	-
Task *e*	17018	18069	1051	-
Task *f*	12050	13289	1239	-
Task *g*	19035	21486	2451	21486
Task *h*	13911	14602	691	-
Task *i*	8016	8536	520	-
Task *j*	13618	14596	978	-
Task *k*	27861	28156	295	-
Task *l*	43336	44731	1395	-

4.3. The Task Dispatching Phase

Basically, there are three cases that need to be discussed in different heterogeneous environments, and the cases are illustrated and discussed as follows, step by step.

Case 1. $MSV_i > ECT_{ij}$ of S_j and $EECT_i > AECT_j$ of S_j

Compare the *MSV* value found in the MSV_i calculation phase with the earliest expected completion time of other tasks. Task *i* can be dispatched to the appropriate server node *j* while $MSV_i > ECT_{ij}$ of S_j and $EECT_i > AECT_j$ of *S*. Therefore, task *d* is dispatched to Node D while $MSV_i > ECT_{dD}$ and $ECT_{dD} > AECT_j$ in Tables 7 and 8.

Table 7. Comparison of the MSV values of tasks.

	Node C	Node D	*SV*	*MSV*
Task *a*	52978	25328	27650	-
Task *b*	47351	20187	27164	-
Task *d*	54429	26330	28099	54429
Task *e*	47319	20348	26971	-
Task *f*	43566	15918	27648	-
Task *g*	51026	24031	26995	-
Task *h*	44310	17678	26632	-
Task *j*	37603	9685	27918	-
Task *l*	43986	16558	27428	-
Average	-	19562	-	-

Table 8. Comparison of the average *ECT* values of tasks in Node D under Case 1.

	Node C	Node D	*SV*	*MSV*
Task *a*	52978	25328	27650	-
Task *b*	47351	20187	27164	-
Task *d*	54429	26330	28099	54429
Task *e*	47319	20348	26971	-
Task *f*	43566	15918	27648	-
Task *g*	51026	24031	26995	-
Task *h*	44310	17678	26632	-
Task *j*	37603	9685	27918	-
Task *l*	43986	16558	27428	-
Average	-	19562	-	-

Case 2. $MSV_i < ECT_{ij}$ of S_j

Compare the *MSV* value found in the MSV_i *calculation phase* with the earliest expected completion time of other tasks. Task *i* with maximum *ECT* can be dispatched to appropriate server node *j* when $MSV_i < ECT_{ij}$ of server node *j*. Therefore, task *l* is assigned to Node A in Table 9 because its ECT_{lA} is greater than the *MSV* value (43336 > 21486).

Table 9. Comparison of the average *ECT* values of tasks in Node A under Case 2.

	Node A	Node B	*SV*	*MSV*
Task *a*	22345	23526	1181	-
Task *b*	16667	17930	1263	-
Task *c*	31083	31897	814	-
Task *d*	24712	25156	444	-
Task *e*	17018	18069	1051	-
Task *f*	12050	13289	1239	-
Task *g*	19035	21486	2451	21486
Task *h*	13911	14602	691	-
Task *i*	8016	8536	520	-
Task *j*	13618	14596	978	-
Task *k*	27861	28156	295	-
Task *l*	43336	44731	1395	-

Case 3. $MSV_i > ECT_{ij}$ of S_j and $EECT_i < AECT_j$ of S_j

Compare the MSV value found in the MSV_i *calculation phase* with the earliest expected completion time of other tasks. Task *i* in Table 10 will be dispatched to server node *j* where the ECT_i is approximate to $AECT_j$ and the ECT_i is larger than $AECT_j$ under the conditions that $MSV_i > ECT_{ij}$ and $EECT_i < AECT_j$ of S_j. Therefore, task *j* is assigned to Node B in Table 11 because ECT_{jB} is bigger than and closer to $AECT_B$.

Table 10. Comparison of the average *ECT* values of tasks in Node D under Case 3.

	Node A	Node B	*SV*	*MSV*
Task *b*	60023	49827	10196	-
Task *e*	60374	49966	10408	-
Task *f*	55406	45186	10220	-
Task *h*	57267	46499	10768	-
Task *i*	51372	40433	10939	51372
Task *j*	56974	46493	10481	-
Average	-	46400	-	-

Table 11. Assigning tasks to the server nodes.

	Node A	Node B	*SV*	*MSV*
Task *b*	60023	49827	10196	-
Task *e*	60374	49966	10408	-
Task *f*	55406	45186	10220	-
Task *h*	57267	46499	10768	-
Task *i*	51372	40433	10939	51372
Task *j*	56974	46493	10481	-
Average	-	46400	-	-

Subsequently, the above examples can be simulated in our experiment. At first, the number of tasks is set to 50 to 100 and the computing ability of the 5 cloud server nodes is set to {500~1000} units to adapt to the high-heterogeneity environment. Besides this, the comparisons of MakeSpan and load balancing among the Sufferage, MaxSufferage, and ACSS algorithms are simulated 50 times and the

average value of makespan is taken, as shown in Figures 2–5. In Figure 2, the proposed algorithm has a better MakeSpan than the others, especially in the large tasks environment. In addition, the load balancing index can be defined by r_{min}/r_{max}, where r_{min} is the shortest completed task time of all tasks and r_{max} is the longest completed task time of all tasks [1,15].

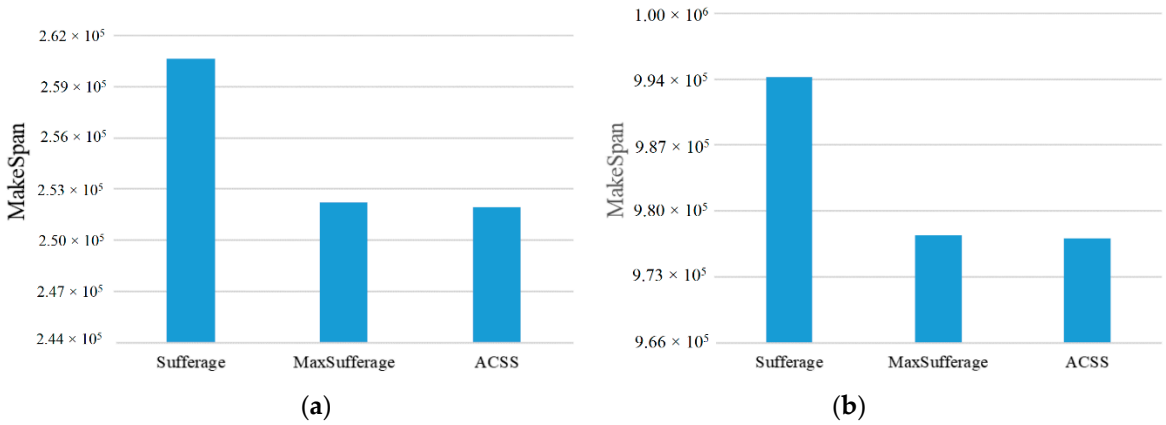

Figure 2. The comparison results of MakeSpan among Sufferage, MaxSufferage, and ACSS algorithms for (**a**) n = 4 and number of tasks = 50 and (**b**) n = 4 and number of tasks = 100.

In general, the value of the load balancing index is a number between 0 and 1, with 0 being the worst load balance and 1 being the optimal load balance.

As shown in Figures 3–5, the ACSS algorithm can obtain the best load balancing index (0.88) over Sufferage (0.87) and MaxSufferage (0.83). Because the ACSS algorithm uses the distribution of the average value, the MakeSpan of each node can achieve similar results. However, MaxSufferage completed time is better than that of Sufferage, but the load balancing results are similar. This is because MaxSufferage did not consider the load status of the node during the selection of tasks. As a result, the proposed ACSS algorithm can obtain the best results of complete time and load balance among these algorithms even if the heterogeneous cloud computing network is complex.

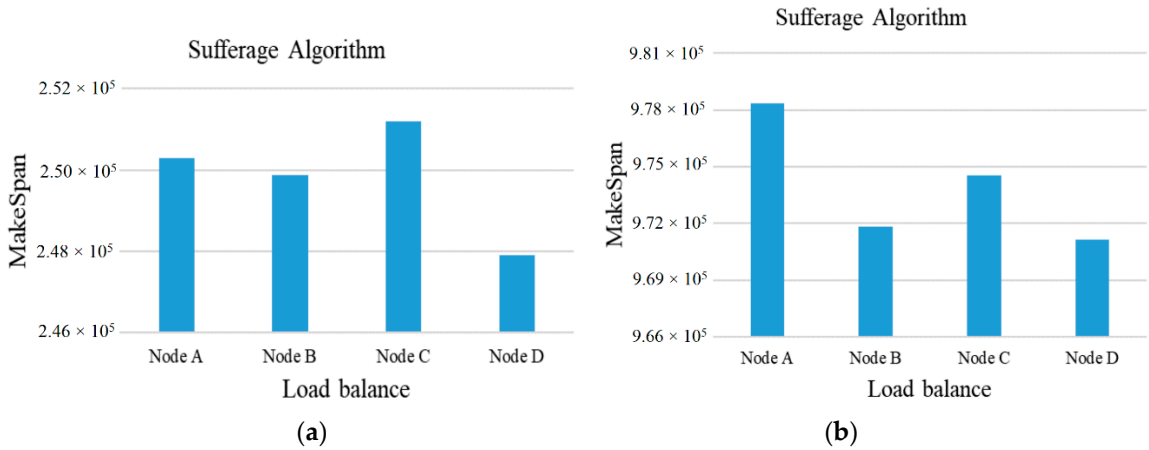

Figure 3. The results of load balancing index in the Sufferage scheduling algorithm for (**a**) n = 4 and number of task = 50 and (**b**) n = 4 and number of tasks = 100.

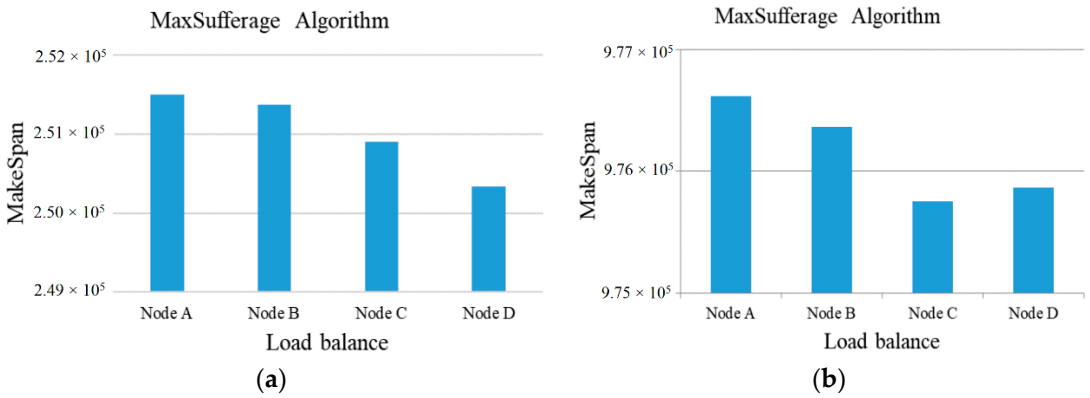

Figure 4. The results of load balancing index in the MaxSufferage scheduling algorithm for (**a**) n = 4 and number of tasks = 50 and (**b**) n = 4 and number of tasks = 100.

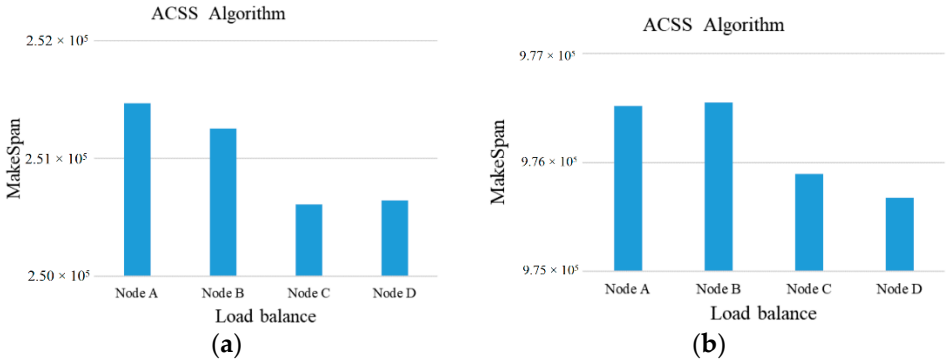

Figure 5. The results of load balancing index in the Advanced Cluster Sufferage Scheduling (ACSS) scheduling algorithm for (**a**) n = 4 and number of tasks = 50 and (**b**) n = 4 and number of tasks = 100.

Besides this, the formula $RU = \frac{\sum_{j=1}^{N} TC_j}{Nm} \times 100\%$ is used to calculate the ratio of resource utilization to show whether the use of resource in this paper is maximized. In factor RU, the TC_j represents the total expected completion time by virtual machine j, N represents the number of virtual machines, and m represents the final completion time of the virtual machine. The related ratio results of resource utilization are shown in Figure 6. In Figure 6, the ratio of resource utilization of ACCS can reach 89%, and this result is better than for other algorithms. This is because that the average value is used to consider the allocation status of nodes in the ACSS algorithm

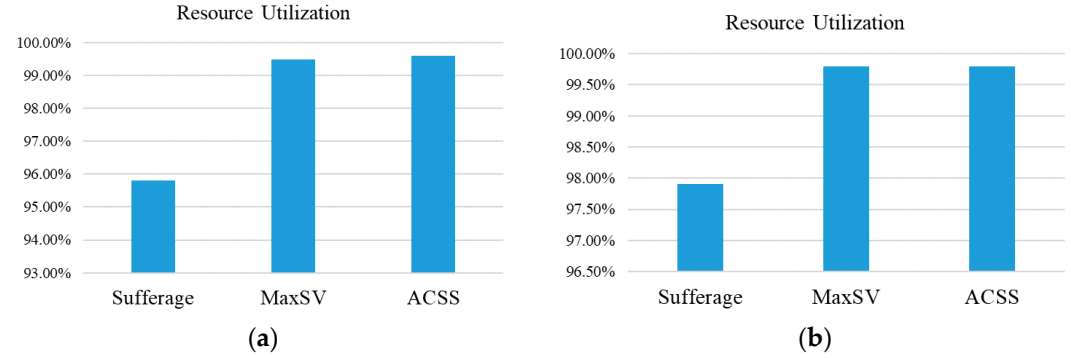

Figure 6. The ratio of Resource Utilization in the ACSS scheduling algorithm for (**a**) n = 4 and number of tasks = 50 and (**b**) n = 4 and number of tasks = 100.

Subsequently, the parameter of matching proximity [11] is used to evaluate the degree of proximity of various scheduling algorithms. In Figure 7, the MET (Minimum Execution Time) and ECT (Expected Computing Time) are used to estimate whether the task can be quickly matched. A large value for matching proximity means that a large number of tasks are assigned to the machine that executes them faster, as expressed by

$$\text{Matching Proximity} = \frac{\sum_{i \in Tasks} ECT_i S_i}{\sum_{i \in Tasks} ECT_i MET_i}. \quad (4)$$

As show in Figure 7, the matching ratio of the three algorithms is close to 1. These three algorithms have good matching efficiency. Subsequently, the performance and complexity of algorithms can be compared in Table 12. The results of the comparison table show that ACSS can obtain the best performance among all algorithms in evaluation factors including makespan, load balance, resource utilization, and matching proximity. Based on [9], the Big O notation is used to estimate the complexity of these algorithms. Then, the results of complexity of MaxSufferage and ACSS are O(n^2rm) because parameter r indicates that the alternative condition is selected when the condition of ($MSV_i > ECT_{ij}$) && ($EECT_i < AECT_{ij}$ of S_j) is satisfied. As a result, the complexity of the ACC algorithm is approximately equal to that of the Sufferage algorithm.

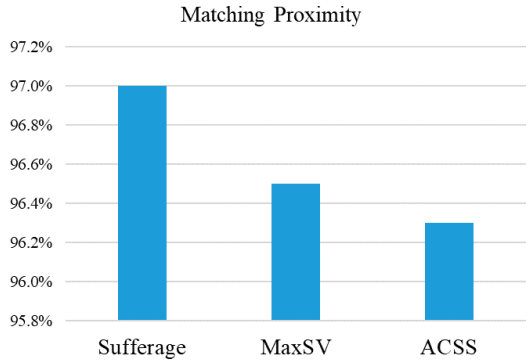

Figure 7. The ratio of Matching Proximity in all of scheduling algorithms.

Table 12. The performance and complexity comparisons of all of algorithms.

		Sufferage	MaxSufferage	ACSS
Experiments	MakeSpan	7.57×10^4	6.97×10^4	6.96×10^4
	Load Balance	0.87	0.84	0.88
	Resource Utilization	91%	95.5%	96%
	Matching Proximity	0.97	0.965	0.963
	Complexity	O (n^2m)	O (n^2rm)	O (n^2rm)

5. Conclusions

In this study, the TLCD architecture is proposed to provide secure and reliable scheduling and to improve the defect of slow response in cloud systems. Basically, TLCD includes three layers of procedures. In the first layer, which is called the CAC layer, the system can dispatch the heterogeneous tasks into appropriate category clusters to reduce task delay and overloading. Subsequently, a CSA algorithm is proposed in the CS layer to dispatch the task to an appropriate cluster to enhance the reliability and reduce the cost and completion time. In the final layer, which is defined as the SNS layer, the system can improve the load balancing and reduce the completion time by elements of MSV and the average ECT of S_j. Simulation results show that the proposed algorithms can obtain the best results among all algorithms in evaluation factors including makespan, load balance, resource utilization, and matching proximity under heterogeneous environments.

Author Contributions: M.-L.C. and Y.-F.H. designed the framework and wrote the manuscript. H.-C.H. and W.-C.T. verified the results of our work and conceived the experiments together. M.-L.C. and Y.-F.H. discussed the results and contributed to the final manuscript.

Funding: This research was funded by Ministry of Science and Technology of Taiwan under Grant MOST 106-2221-E-324-020.

Conflicts of Interest: The founding sponsors had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, and in the decision to publish the results.

References

1. Petkovic, I. CRM in the cloud. In Proceedings of the IEEE 8th International Symposium on Intelligent Systems and Informatics, Subotica, Serbia, 10–11 September 2010; pp. 365–370.
2. Casanova, H.; Legrand, A.; Zagorodnov, D.; Berman, F. Heuristics for scheduling parameter sweep applications in grid environment. In Proceedings of the 9th Heterogeneous Computing Workshop, Cancun, Mexico, 1 May 2000; pp. 349–363.
3. Chiang, M.L.; Luo, J.A.; Lin, C.B. High-Reliable Dispatching Mechanisms for Tasks in Cloud Computing. In Proceedings of the BAI2013 International Conference on Business and Information, Bali, Indonesia, 7–9 July 2013; p. 73.
4. Buyya, R.; Ranjan, R.; Calheiros, R.N. Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit: Challenges and opportunities. In Proceedings of the International Conference on High Performance Computing & Simulation, Leipzig, Germany, 21–24 June 2009; pp. 1–11.
5. Jones, M.T. Google's Geospatial Organizing Principle. *IEEE Comput. Graph. Appl.* **2017**, *27*, 8–13. [CrossRef]
6. Lee, Y.H.; Huang, K.C.; Wu, C.H.; Kuo, Y.H.; Lai, K.C. A Framework of Proactive Resource Provisioning in IaaS Clouds. *Appl. Sci.* **2017**, *7*, 777. [CrossRef]
7. Alfazi, A.; Sheng, Q.Z.; Qin, Y.; Noor, T.H. Ontology-Based Automatic Cloud ServiceCategorization for Enhancing Cloud ServiceDiscovery. In Proceedings of the IEEE 19th International Enterprise Distributed Object Computing Conference, Adelaide, SA, Australia, 21–25 September 2015; pp. 151–158.
8. Salton, G.; Buckley, C. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manag.* **1988**, *24*, 513–523. [CrossRef]
9. Reda, N.M.; Tawfik, A.; Marzok, M.A.; Khamis, S.M. Sort-Mid tasks scheduling algorithm in grid computing. *J. Adv. Res.* **2015**, *6*, 987–993. [CrossRef] [PubMed]
10. Anousha, S.; Ahmadi, M. An improved Min-Min task scheduling algorithm in grid computing. *Lect. Notes Comput. Sci. Grid Pervasive Comput.* **2013**, *7861*, 103–113.
11. Merajiand, S.; Salehnamadi, M.R. A batch mode scheduling algorithm for grid computing. *J. Basic Appl. Sci. Res.* **2013**, *3*, 173–181.
12. Shojafar, M.; Cordeschi, N.; Baccarelli, E. Energy-efficient Adaptive Resource Management for Real-time Vehicular Cloud Services. *IEEE Trans. Cloud Comput.* **2018**. [CrossRef]
13. Shojafar, M.; Javanmardi, S.; Abolfazli, S.; Cordeschi, N. FUGE: A joint meta-heuristic approach to cloud job scheduling algorithm using fuzzy theory and a genetic method. *Clust. Comput.* **2015**, *18*, 829–844. [CrossRef]
14. Shojafar, M.; Canali, C.; Lancellotti, R.; Abawajy, J. Adaptive Computing-plus-Communication Optimization Framework for Multimedia Processing in Cloud Systems. *IEEE Trans. Cloud Comput.* **2018**. [CrossRef]
15. Maheswaran, M.; Ali, S.; Siegel, H.J.; Hensgen, D.; Freund, R.F. Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems. *J. Parallel Distrib. Comput.* **1999**, *59*, 107–131. [CrossRef]
16. Braun, T.D.; et al. A comparison study of static mapping heuristics for a class of meta-tasks on heterogeneous computing systems. In Proceedings of the Heterogeneous Computing Workshop (HCW '99), San Juan, Puerto Rico, 12 April 1999; pp. 15–29.
17. Etminani, K.; Naghibzadeh, M. A Min-Min Max-Min selective algorithm for grid task scheduling. In Proceedings of the Third IEEE/IFIP International Conference in Central Asia on Internet, Tashkent, Uzbekistan, 26–28 September 2007; pp. 138–144.
18. Lan, J.; Zhou, J.; Liu, X. An area-efficient implementation of a Message Authentication Code (MAC) algorithm for cryptographic systems. In Proceedings of the IEEE Region 10 Conference (TENCON), Singapore, 22–25 November 2016; pp. 1977–1979.

19. Li, S.; Liu, J.; Wang, S.; Li, D.; Huang, T.; Dou, W. A Novel Node Selection Method for Real-Time Collaborative Computation in Cloud. In Proceedings of the International Conference on Advanced Cloud and Big Data (CBD), Chengdu, China, 13–16 August 2016; pp. 98–103.
20. Chiang, M.L.; Hsieh, H.C.; Tsai, W.C.; Ke, M.C. An Improved Task Scheduling and Load Balancing Algorithm under the Heterogeneous Cloud Computing Network. In Proceedings of the IEEE 8th International Conference on Awareness Science and Technology (iCAST2017), Taichung, Taiwan, 8–10 November 2017; p. 61.
21. Deng, J.; Huang, S.C.H.; Han, Y.S.; Deng, J.H. Fault-Tolerant and Reliable Computation in Cloud Computing. In Proceedings of the IEEE Globecom 2010 Workshop on Web and Pervasive Security, Miami, FL, USA, 6–10 December 2010; pp. 1601–1605.
22. Yoon, E.J.; Yoo, K.Y. An Efficient Diffie-Hellman-MAC Key Exchange Scheme. In Proceedings of the Fourth International Conference on Innovative Computing, Information and Control (ICICIC), Kaohsiung, Taiwan, 7–9 December 2009; pp. 398–400.