Peer-reviewed version available at Appl. Sci. 2018, 8, 1385; doi:10.3390/app8081385

Article

High 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, Taiwan ROC; {mlchiang, yfahuang, azongtsai}@cyut.edu.tw
2 Department of Information and Communication, Hsing Wu University of Technology, New Taipei City, Taiwan, ROC; luckyeva.hsieh@gmail.com
* Correspondence: yfahuang@cyut.edu.tw; Tel.: +886-423323000-7243

Featured Application: Authors are encouraged to provide a concise description of the specific application or a potential application of the work. This section is not mandatory.

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 system. Therefore, a Three-Layer Cloud Dispatching (TLCD) architecture is proposed to enhance the performance of task scheduling. In first layer, the tasks need to be distinguished to different types by their characters. Subsequently, the Cluster Selection Algorithm is proposed to dispatch the task to appropriately service cluster in the secondly layer. Besides, a new scheduling algorithm is proposed to dispatch the task to a suitable server in a server cluster to improve the dispatching efficiency in the thirdly layer. Basically, the TLCD architecture can obtain better task completion time than previous works. Besides, our algorithm and can achieve load-balancing and reliability in cloud computing network.

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 more storage requirement 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.

Furthermore, the cloud computing network can be divided into three basic service categories: The Software as a Service (SaaS), the Platforms as a Service (PaaS) and the Infrastructure as a Service (IaaS) [2][3[4]. In SaaS, the software is provided by the software vendor, such as Gmail and Google Driver. For the PaaS, it provides a platform for users for programming purpose. Google APP Engine is one kind of platforms of PaaS. In the last category of service called the IaaS, the hardware resources are proposed to support users for constructing the framework, such as Cloud server [2][3[4] No matter what kind of cloud service is applied to the cloud computing network, there has a common character: each server has different ability and computing power. Therefore, propose an efficient scheduling algorithm for dispatching the tasks to appropriate server nodes of cloud becomes an important challenge in cloud computing network.

Peer-reviewed version available at Appl. Sci. 2018, 8, 1385; doi:10.3390/app8081385

Basically, in the traditional cloud clustering architecture, system only considers the heterogeneity of tasks while executing scheduling procedure 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 the heterogeneous nodes and tasks exist in the cloud system at the same time. For the first layer, Category Assignment Cluster (CAC) [6][7] layer was proposed to reduce the task delay and the overloading by classifying the heterogeneous tasks. In CAC layer, the various tasks can be classified as three types according to the IaaS, SaaS, and PaaS categories. Subsequently, the homogeneous tasks can be dispatched to corresponding service category clusters in the second layer.

In the second layer, called the Cluster Selection (CS) layer, the homogenous task can be assigned to appropriate cluster by Cluster Scheduling Algorithm (CSA) to enhance the reliability of system. Besides, the cost and completion time of task scheduling can be reduced in this layer.

Finally, tasks can be dispatched to service nodes by scheduling algorithm in the third layer, Server Nodes Selection (SNS) layer. In this layer, an Advanced Cluster Sufferage Scheduling (ACSS) algorithm is proposed to enhance the resource utilization and to achieve load balancing [2][3][8][9][10][11].

The rest of this paper is organized as follows. Section 2 will describe the related works of scheduling algorithms in the cloud computing network. Section 3 gives the explanation of the TLCD architecture and the proposed algorithm. In section 4, an example is provided to describe the overall procedure of ACSS. Finally, the conclusions is given in Section 5.

2. Materials and Methods

In general, scheduling algorithm is a mapping mechanism and is being divided into two modes: the real-time mode and the batch mode. The main difference between these two modes is the timing of dispatching. Basically, tasks in the real-time mode will be assigned to server nodes immediately. In contrast to real-time mode, tasks will be assigned to server once the number of tasks is accumulated to a certain amount under the batch mode. In other words, the real-time scheduling algorithm only focuses on the results of a single task assignment, while the batch scheduling algorithm considers the assignments results of all tasks. Therefore, the batch mode scheduling algorithm has better performance in load balancing and completion time than the real-time scheduling algorithm [11][12][13].

So far, many mechanisms have been proposed to ensure the quality of service in the cloud computing network, and an appropriate task scheduling algorithm is one of the most important methods to achieve these goals. During the last several decades, there has been an increase in the number of publications on task dispatching algorithms [2][3][8][9][10][12][14]. Basically, these algorithms, such as Min-Min [8][9][12][14], Max-Min[8][12][14]14, Sufferage [2][8][10][14] and MaxSufferage algorithm [2][3][8][12], only consider the expected completion time (ECT) as a factor while designing the scheduling algorithm. The load status of the node is not considered. Therefore, the completion time is not as expected.

For example, in Min-Min algorithm, the tasks of the smallest sets are selected. With the selected set, the task which has the smallest ECT value will be assigned to the server nodes. It starts with a set T of all unassigned tasks and the task with minimum completion time is selected as min_ECT_i. Then, the task with overall minimum completion time from min_ECT_i is selected and assigned to that server node. Finally, the newly mapped task is removed from T and the process repeats until all tasks are assigned. Under such algorithm, the network system will be unbalanced while there have too many tasks waiting for dispatching.

Similar to Min-Min algorithm, the completion time is also an estimated factor for dispatching tasks under Max-Min algorithm. The difference between these two algorithms is that Max-Min algorithm selects the tasks with overall maximum completion time instead of the minimum completion time. However, the large tasks are always to be assigned in advance in Max-Min scheduling algorithm. Hence, the overall completion time will increase significantly [8][12][14]. Basically, the server nodes with higher ability are easily to be assigned more tasks than the server nodes with lower ability in the above algorithms. Therefore, the workloads of server nodes are unbalanced and the completion time will increase significantly. As a result, the Sufferage algorithm [1][14][15][16] is proposed to improve the load balancing of the network system. Here, the main concept of Sufferage algorithm is dispatching the tasks to server nodes by computing the Sufferage Value(SV) which is calculated by the second earliest completion time minus the earliest completion time. Subsequently, task *i* with largest SV value will be assigned to the appropriate server nodes with minimum ECT. However, the completion time between tasks cannot be reduced effectively and the load balancing cannot be improved while there are too many tasks waiting for dispatching.

Based on the reason above, the MaxSufferage algorithm 3 is proposed to improve the defect of Sufferage algorithm, and the proposed protocol is divided into three phases. In the first phase called the SV_i calculation phase, the SV value is calculated among all of tasks. Following is the MSV_i calculation phase, and the task *i* with the second earliest ECT will be elected as MaxSufferage Value(MSV) value while task *i* has the maximum SV value among all SV values. For the last phase called the task dispatch phase, task *i* with minimum ECT can be dispatched to appropriate server node *j* when MSV_i $MSV_i > ECT_{ij}$ of server node *j*. Conversely, task *i* with a maximum ECT value can be dispatched to server node *j*. Unfortunately, the large tasks are easy to be dispatched to server nodes with poor ability in MaxSufferage algorithm under the heterogeneous environments

For solving the problem above, the AMS algorithm[17] to improve the drawback of 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. Besides, all the tasks be dispatched to the appropriate server nodes in the cloud computing network even if the server nodes are located in heterogeneous environment.

Subsequently, the related description of our algorithm and explained as follow.

3. Three-layer cloud dispatching Architecture

Since the traditional dispatch algorithm of cluster architecture does not dispatch task by its capacity of cluster. It may cause the drawback of task delay, low reliability and high MakeSpan. Therefore, a Three Layer Cloud Dispatching (TLCD) architecture and related scheduling algorithm are proposed to apply to cluster-based cloud environment, as shown in Figure. 1. The description of TLCD is shown as follows.

Peer-reviewed version available at Appl. Sci. 2018, 8, 1385; doi:10.3390/app8081385

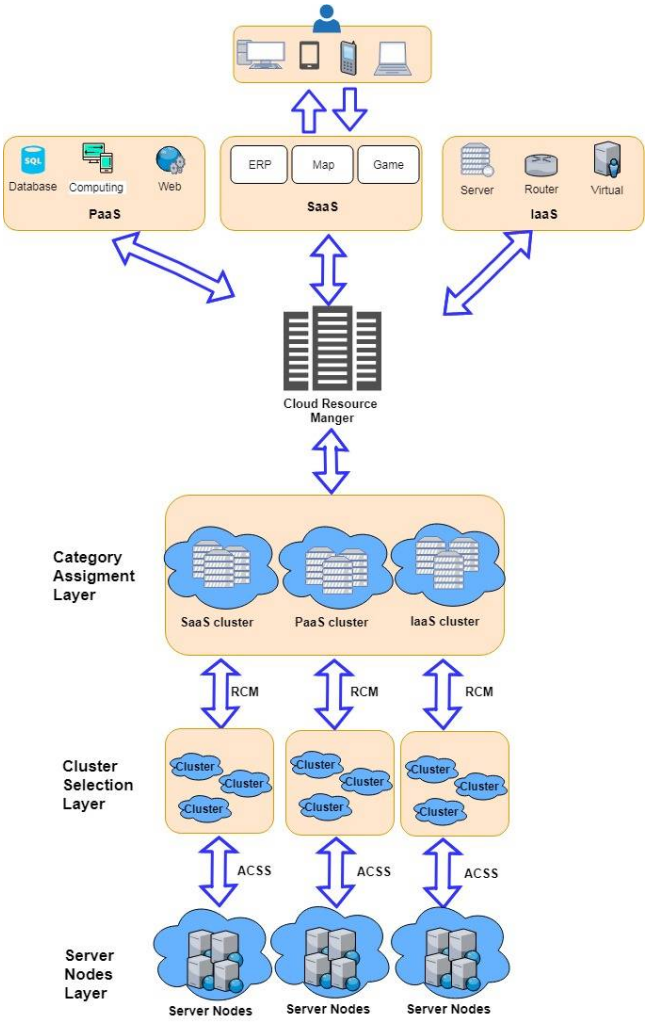

Figure. 1 .Three Layer Cloud Dispatching Architecture

3.1 Category Assignment cluster layer

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

Besides, the diversity of tasks increases the delay of 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 Category Assignment Cluster (CAC) layer [6][7]. The category cloud clusters can be divided into three types: SaaS, PaaS and IaaS. Through these three categories of classification, the difficulty of scheduling on 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 cluster by using the factors of Reliability (R_i), Cost (C_i) and MakeSpan (M_i) [3][18]. MakeSpan is the length of time to complete a task. Basically,

when the MakeSpan value is getting bigger, 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][18]. Finally, for the cost factor, it is defined as the cost needed for a task to be sent and responded. When taking these three factors into account, tasks can be assigned to suitable cluster 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. Because, the objectives of this example are configured for reliability and cost. Subsequently, we propose an example to explain this algorithm.

$$R_i = (\sum_{K=1}^{C} Sub\_R_k n_k) / n \quad (1)$$

$$C_i = (\sum_{k=1}^{L} Sub\_C_k n_k) \quad (2)$$

$$M_i = (\sum_{K=1}^{C} Sub\_M_k n_k)/n \quad (3)$$

k = cluster k.
i = assignment i.
n = the total number of tasks
l = the total number of clusters
n_k= the number of tasks assigned to the k cluster.
Sub_R_k = the reliability of the cluster k.
Sub_M_k= the makespan of the cluster k.
Sub_C_k= the cost of the cluster k.

In Line (4), we arranged the combination of tasks in all clusters. Cluster will choose an appropriate task combination and then help node to adjust these tasks. Furthermore, line (5) to (8) are proposed to check if the $R_i$and C_i of each assignment i agree with $R_i \geq R_s$ && $C_i \geq C_s$. Then among those passed assignments, the one with the smallest M_i is scheduled. If there are more than two groups eligible, 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 CSA algorithm and the related assumptions are showing in Table 1.

Algorithm 1-Cluster Selection Algorithm

R_s = the threshold value of the overall reliability.

C_s = the threshold value of the overall cost

A_i = assignment i

R_i = the reliability of the assignment i

C_i = the reliability of the assignment i

M_i = the task completion time of assignment i

n = the total number of tasks

n_k = the number of tasks assigned to the k cluster

1: for **total task n**

2: for total cluster k

3: get Sub_R_k 、Sub_Mk and Sub_C_k of each cluster

Peer-reviewed version available at Appl. Sci. 2018, 8, 1385; doi:10.3390/app8081385

4: Arrange all tasks in the cluster and assign the number of Ai

5: for calculating the R_i 、 $C_i$and Mi of the assignment A_i

6: if $R_i \geq R_s$ && $C_i < C_s$ 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 1. Example of Cluster Selection Algorithm

	Cluster1			Cluster2			Cluster3			(R_i, M_i,C_i)
	Sub_R_k	Sub_C_k	Sub_M_k	Sub_R_k	Sub_C_k	Sub_M_k	Sub_R_k	Sub_C_k	Sub_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: R_s and C_s are set by user. In this example, R_s and C_s. are 22 and 260 respectively.

Step 2: Calculate R_i, C_i and M_i of each allocation combination. According to Table 1, assignment A1 assign five tasks to Cluster 1, two tasks to Cluster 2 and three tasks to Cluster 3 respectively We used formula (1) and (2) to calculate the average of R_i, C_i and M_i, and showing as follows.

$$R_1 = \frac{22 \times 5 + 24 \times 2 + 25 \times 3}{10} = 23.3$$

$$C1 = 25 \times 5 + 28 \times 2 + 32 \times 3 = 242$$

$$M_1 = \frac{30 \times 5 + 36 \times 2 + 15 \times 3}{10} = 26.7$$

The same procedure goes to other assignments too.

Step 3: Select the schedule that meets the condition of $R_i \geq 22$ and $C_i < 260$; Here, A1 、A2 and A6 are selected.

Step 4: According to the conditions of step 3, we choose A1 with the highest reliability and smallest MakeSpan. Since the result of assignment A1 is better than others, thus assignment A1 is elected as 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 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 cluster. However, tasks may be assigned to inappropriate server nodes when there exists a large number of tasks. As a result, system will become unbalanced and will have higher completion time. To solve these problems, the Advanced Cluster Sufferage Scheduling (ACSS) algorithm is proposed to improve the defect of MaxSufferage algorithm [12][14][15] under the heterogeneous cloud computing network. The main concept of ACSS algorithm is making the tasks dispatched to the appropriate server nodes by using the average ECT of server node denoted as S_j to reduce the influence of inappropriate assignment. The detail of ACSS algorithm is shown in Algorithm 2.

Basically, the ACSS algorithm is divided into three phases. In the first phase called the SV_j calculation phase, system will find the $EECT_i$ (The earliest expected completion time) and $SEECT_i$ (The second earliest expect completion time) of the S_j to calculate the SV value, and the detail procedure is shown in lines (8) ~ (9) in Figure 3. Subsequently, the MSV value will be set to the second earliest ECT value of task i in the second phase which is called as the MSV_i calculation phase while task i has the maximum SV value among all SV values.

In the third phase called the task dispatching phase, task i will be dispatched to Sj while $MSV_i > ECT_{ij}$ and $EECT_i \mathrm{ECT}_i^{1st} > AECT_j$ of S_j. Conversely, when $EECT_i \mathrm{ECT}_i^{1st}$ $\mathrm{ECT}_i^{1st} < AECT_j$, task i can be dispatched to server node j where the ECT_i is approximate to $AECT_j$ and the ECT_ineeds to be larger than $AECT_j$. Basically, the procedure is different from Sufferage and MaxSufferage algorithm, and the detail algorithm is shown in line (11) and (12) in Algorithm 2.

Algorithm 2 Advanced Cluster Sufferage Scheduling

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 of server node SN_j will become ready to execute for next task;

$AECT_j$= the expected completion time of Average time in the server node S_j;

$EECT_i$ = the earliest expected completion time of tasks 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;

1: *for* all unassigned task i

2: *for* all server nodes SN_j

3: $TCT_{ij} = ECT_{ij} + r_j$

4: *do scheduling all job assignments*

5: mark all server nodes as unassigned;

6: *for* each task i in S_j;

7: find server node SN_j that gives the earliest completion time;

8: calculating the Sufferage Value (sv = $EECT_i - SEECT_i$);

9: If the maximum value of SV_i has two or even more the same

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 22 July 2018 doi:10.20944/preprints201807.0404.v1

Peer-reviewed version available at Appl. Sci. 2018, 8, 1385; doi:10.3390/app8081385

then chose the assignment $SEECT_i$ to MSV with maximum;

else the assignment i with maximum $SECT_i$ can be compared to other $EECT_i$;

10: If($MSV_i < ECT_{ij}$ of S_j)

Then the task i with maximum ECT can be dispatched to server nodes SN_j;

11: 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 ;

12: 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 SN_j can be dispatched to server nodes S_j;

13: end if;

14: end for

15: $r_j = r_j + ECT_{ij}$

16: update $TCT_{ij} = e_{ij} + r_j$

17: *end do*

According to the operations above, the completion time and load balancing of the heterogeneous cloud computing network can be improved efficiently. For the security issue, we can hire a MAC algorithm [15][19] with session key to confirm that the message come from the truly sender and has not been modified. Subsequently, the example is provided to help understanding the ACSS algorithm of server node selection layer.

4. Example and comparison results

In this paper, four heterogeneous environments including HiHi (High heterogeneity task, High heterogeneity server node), HiLo, LoHi and LoLo [4, 10] will be discussed. Basically, HiHi is the most complex problem. Hence, in this paper, we will use four nodes and twelve tasks for task assignment under the HiHi's heterogeneous environment as an example. Subsequently, the ACSS algorithm assignment task process can be divided into three phases to show our processes. Firstly, the SV value and the MSV value can be calculated in the SV_j calculation phase and the MSV_i calculation phase. The dispatching procedure is executed in the task dispatching phase by using $AECT_j$ of SN_j. The parameters of HiHi environment are shown in Table 2.

Table 2. Illustrations of the expected execution time of tasks

	Node A	Node B	Node C	Node D
Task *a*	19020	21453	22350	24003
Task *b*	17003	18036	18643	20320
Task *c*	24697	25123	25753	26302
Task *d*	8001	8503	8927	9657
Task *e*	13603	14563	15310	16530
Task *f*	27846	28123	28634	29763
Task *g*	22330	23493	24302	25300
Task *h*	43321	44698	46756	49631
Task *i*	31068	31864	32013	34650
Task *j*	16652	17897	18675	20159
Task *k*	13896	14569	15634	17650
Task *l*	12035	13256	14890	15890

Peer-reviewed version available at Appl. Sci. 2018, 8, 1385; doi:10.3390/app8081385

- *The SVi calculation phase*

Step 1. List the expected execution times for all task i on *Sj*, as shown in Table 2.

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 task *b* to *l* to calculate the SV values. The calculation results are shown in Table 3.

Table 3. Calculation of the SV value 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

- *The MSVi calculation phase*

Step 1. The second earliest ECT value of Task *i* can be selected as MSV value while task *i* has the maximum SV_i value among all SV values. As shown in Table 4, the task with the largest SV_i value is Task *g*, thus the second earliest ECT (Here is 21486) of ECT_{gB} is selected as the MSV value.

Table 4. Calculation of the MSV value 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	

- *The task dispatching phase*

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 22 July 2018 doi:10.20944/preprints201807.0404.v1

Peer-reviewed version available at Appl. Sci. 2018, 8, 1385; doi:10.3390/app8081385

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 founded 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_j. Therefore, task d is dispatched to Node D while the $MSV_i > ECT_{dD}$ and $ECT_{dD} > AECT_j$ in table 5 and table 6.

Table 5. Comparison of the MSV value 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	
Avg		19562		

Table 6. Comparison of the average ECT 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	
Avg		19562		

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

Compare the MSV value founded 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 7 because its ECT_{lA} is greater than the *MSV* value (43336> 21486).

Table 7. Comparison of the average *ECT* of tasks in Node A under case2

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 22 July 2018 doi:10.20944/preprints201807.0404.v1

Peer-reviewed version available at Appl. Sci. 2018, 8, 1385; doi:10.3390/app8081385

	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 founded in the *MSV_i calculation phase* with the earliest expected completion time of other tasks. Task *i* in table 8 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 the $MSV_i > ECT_{ij}$ and $EECT_i < AECT_j$ of S_j. Therefore, task *j* is assigned to Node B in table 9 because ECT_{jB} is bigger than and closer to $AECT_B$.

Table 5. Comparison of the average *ECT* 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	
Avg		46400		

Table 9. Assign 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	
Avg		46400		

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 22 July 2018 doi:10.20944/preprints201807.0404.v1

Peer-reviewed version available at Appl. Sci. 2018, 8, 1385; doi:10.3390/app8081385

Based on the examples above, the comparisons of MakeSpan and load balancing among the Sufferage, MaxSufferage, and ACSS algorithms are shown in Figure 2 to 5. In Figure. 2, the proposed algorithm has better MakeSpan than others. In addition, the load balancing index can be calculated by using the following formula [1][12].

Load balance index $=r_{min}/r_{max}$

r_{min} =The shortest completed task time of all tasks.

r_{max} =The longest completed task time of all tasks.

Basically, the value of load balancing index will be a number between 0 and 1. Here, 0 represents the worst load balancing and 1 represents the best.

As shown in the Figure. 3 to 5, ACSS algorithm can obtain best load balancing index (0.88) which is better than 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 Sufferage, but the load balancing results are similar. This is because that MaxSufferage did not consider the load status of the node during the selection of tasks. Thus, the proposed ACSS algorithm is more adapted to the heterogeneous cloud computing network than other algorithms.

Besides, the formula $RU = \frac{\sum_{j=1}^{N} TC_j}{Nm} \times 100\%$is used to calculate ratio of resource utilization to show whether the use of resource in this paper is maximized. In factor RU, the TC_{ij} 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. And, 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 others. It is because that the average value is used to consider allocation status of nodes in ACSS algorithm

Subsequently, the parameter of matching proximity is used to evaluate the degree of proximity of vary schedule algorithms. In Figure. 7, the MET (Minimum Execution Time) and ECT (Expected Compute Time) are used to estimate whether the task can be to quickly matched. A large value for matching proximity means that a large number of tasks are assigned to the machine that executes them faster [10]. The formula (4) is shown as follow.

$$\text{Matching Proximity} = \frac{\sum_{i \in Tasks} ECT[i][S[i]]}{\sum_{i \in Tasks} ECT[i][MET[i]]} \quad (4)$$

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 of algorithms can be compared in Table 10. The results of 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.

Peer-reviewed version available at Appl. Sci. 2018, 8, 1385; doi:10.3390/app8081385

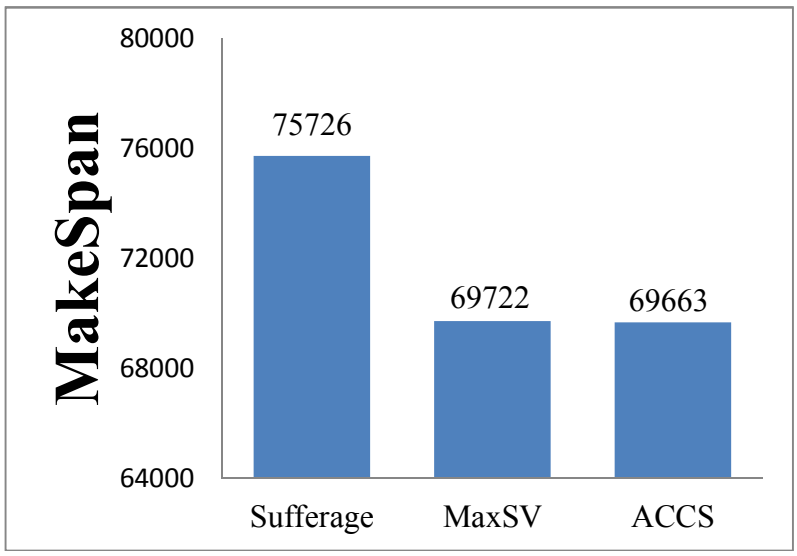

Figure. 2. The comparison results of MakeSpan

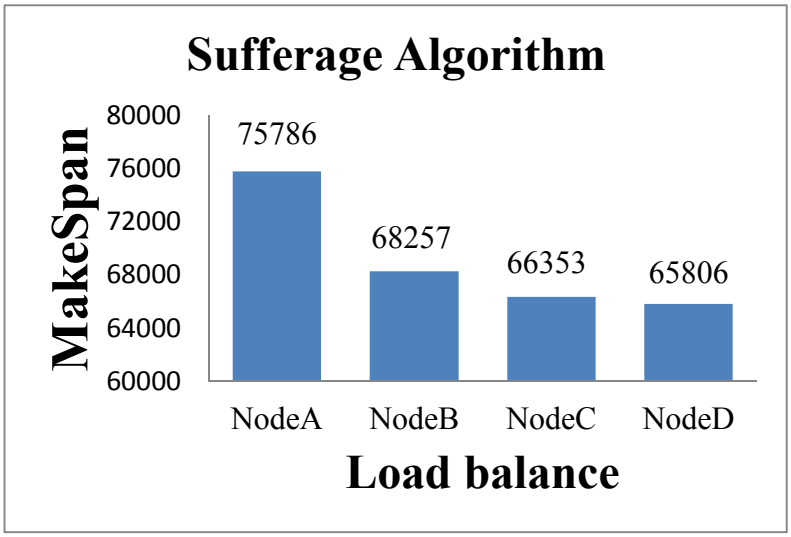

Figure. 3. The load balancing index in Sufferage scheduling algorithm

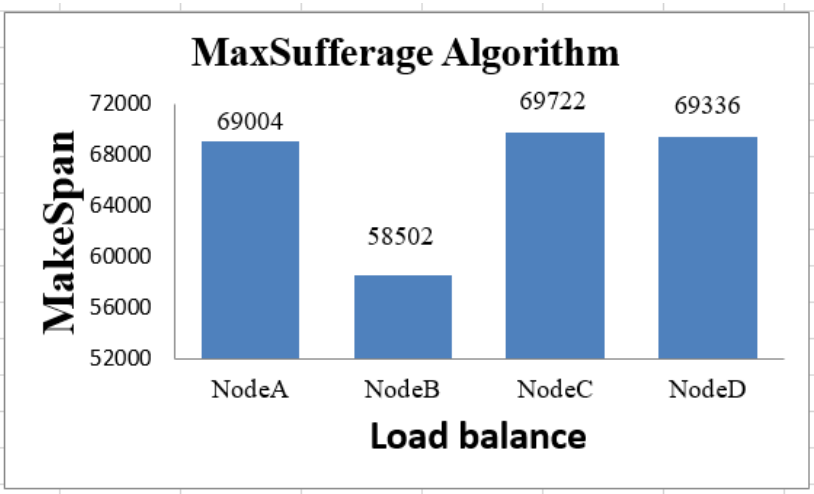

Figure. 4. The load balancing index in MaxSufferage scheduling algorithm

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 22 July 2018 doi:10.20944/preprints201807.0404.v1

Peer-reviewed version available at Appl. Sci. 2018, 8, 1385; doi:10.3390/app8081385

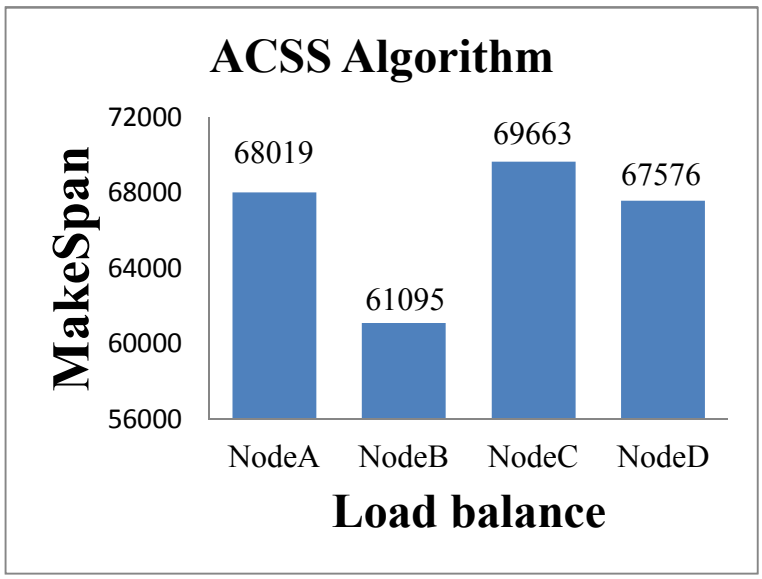

Figure. 5. The load balancing index in ACSS scheduling algorithm

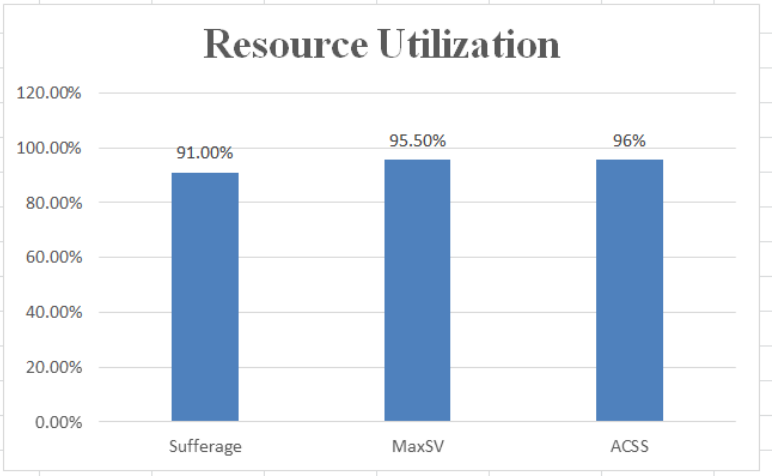

Figure. 6. The ratio of Resource Utilization in ACSS scheduling algorithm

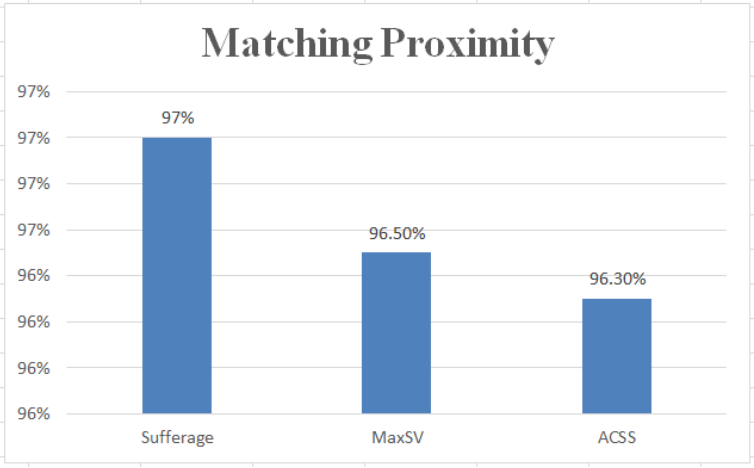

Figure. 7. The ratio of Matching Proximity in all of scheduling algorithms

Table 10. The performance comparison of all of algorithms

		Sufferage	MaxSufferage	ACSS
Experiments	MakeSpan	75726	69722	69663
	Load Balance	0.87	0.84	0.88
	Resource Utilization	91%	95.5%	96%
	Matching Proximity	0.97	0.965	0.963

5. Conclusions

Recently, cloud service users gradually increased, how to provide an efficient service for users is still an important issue. In this study, the TLCD architecture is proposed to provide secure and reliable scheduling and to improve the defect of the slow response of the cloud system.

Basically, TLCD includes three layers of procedure. In the first layer which is called the CAC layer, system can dispatch the heterogeneous tasks into appropriate category clusters to reduce task delay and overloading. Subsequently, a CSA algorithm is proposed in CS layer to dispatch the task to appropriate Cluster to enhance the reliability and reduce the cost and completion time. In the final layer which is defined as the SNS layer, system can improve the load balancing and reduce the completion time by elements of MSV and the average ECT of S_j.

Finally, as shown in Table 10, the proposed algorithms can obtain best results among all algorithms in evaluation factors including makespan, load balance, resource utilization and matching proximity under the heterogeneous environments.

Author Contributions: A and B designed the framework and wrote the manuscript. C and D verified the results of our work and conceived the experiments together. All authors discussed the results and contributed to the final manuscript.

References

1. Petkovic, I. CRM in the cloud. In Proceedings of the IEEE 8th International Symposium on Intelligent Systems and Informatics, 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, 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; pp. 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, June 2009; pp. 1-11.
5. 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(8), 777.
6. 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, Sept 2015; 151-158.
7. Salton, G.; Buckley, C. Term-weighting approaches in automatic text retrieval. Inf. Process. Manage. Aug 1988, 24, 5, 513–523.

8. Reda, N.M.; Tawfik, A.; Marzok, M.A.; Khamis, S.M. Sort-Mid tasks scheduling algorithm in grid computing. Journal of Advanced Research. November 2015, 6, 6, 987-993.
9. Anousha, S.; Ahmadi, M. An improved Min-Min task scheduling algorithm in grid computing. Lecture Notes in Computer Science Grid and Pervasive Computing. May 2013, 7861, 103-113.
10. Merajiand, S.; Salehnamadi, M.R. A batch mode scheduling algorithm for grid computing. Journal of Basic and Applied Scientific Research 2013, 3, 4, 173-181.
11. Meraji, S.; Reza Salehnamadi, M. A Batch Mode Scheduling Algorithm for Grid Computing. Journal of Basic and Applied ScientificResearch 2013,3, 4, 174-176.
12. Maheswaran, M.; Ali, S.; Siegel, H.J.; Hensgen, D.; Freund, R.F. Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems. Journal of Parallel and Distributed Computing November 1999, 59, 2, 107-131.
13. 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) DC, USA, May 1999; pp. 15-29.
14. Etminani, K.; Naghibzadeh, M. A Min-Min Max-Min selective algorithm for grid task scheduling. In Proceeding of the Third IEEE/IFIP International Conference in Central Asia on Internet, September 2007; pp. 138-144.
15. Lan, J.; Zhou, J.; Liu, X. An area-efficient implementation of a Message Authentication Code (MAC) algorithm for cryptographic systems. In Proceeding of the IEEE Region 10 Conference (TENCON), Nov 2016; pp. 1977-1979.
16. 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), Aug 2016; pp. 98-103.
17. 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 Proceeding of the IEEE 8th International Conference on Awareness Science and Technology (iCAST2017), Taichung Taiwan, 8-10 Nov. 2017; pp. 61.
18. 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, Dec 2010; pp. 1601-1605.
19. 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), Dec. 2009; pp. 398-400.