Achieving Load Balance in a Hierarchical Peer-to-Peer Architecture

S.C. WANG, Y.H. SU, S.S. WANG*, K.Q. YAN*, S.F. CHEN
Chaoyang University of Technology
168, Jifeng E. Rd., Wufeng District, Taichung 41349
TAIWAN, R.O.C.
{scwang; s10033905; sswang*; kqyan*; s9514624}@cyut.edu.tw
*: Corresponding author

Abstract: - With the fast development of networking, the demands made of computers are greater than ever before. Determining how to utilize the resources of networking to reach the objective of cooperative computing has remained an important topic in recent years. In addition, with the great advances in technology of the Internet, Peer-to-Peer (or P2P) computing has gradually become the mainstream of distributed applications; it not only provides enormous resources for complicated computing that a single computer cannot solve, but also integrates resources more effectively. A P2P architecture relies primarily on the computing power and bandwidth of the participants in the network rather than concentrating the work in a relatively limited number of servers. P2P architectures typically are used for connecting nodes via large-scale connections. The topology is useful for many purposes. Furthermore, every joint in a P2P computing system has its own resources. Determining how to take the different characteristics of every joint set into consideration for loading assignments is an important topic. However, in this study, a three-phase scheduling algorithm under P2P architecture is advanced. The proposed scheduling algorithm is composed of BTO (Best Task Order), TOLB (Threshold-based Opportunistic Load Balancing) and TLBMM (Threshold-based Load Balance Min-Min) scheduling algorithm that can better utilize executing efficiency and maintain the load balancing of system.

Key-Words: - Distribution system, Peer-to-Peer computing, Scheduling, Load balance

1 Introduction

In recent years, as technology advances and develops, the complexity and scale of problems to be solved via computing become increasingly. However, some issues may be too complicated for a single computer, a PC cluster, or even a specially designed supercomputer to handle. Main function of the network is to connect many computers or resources in the various geographical locations and to play as the communication pipeline between these computers or resources [1].

When suffer from insufficient computing ability, problem solving of large-scaled issues would have to resort to a distributed systems. This approach is usually facilitated by integrating various systems (or nodes) to complete a related network application or duplication of files. There are several options for the establishment of distributed system; among them, cluster system and distributed system are the most frequently used [2].

The cluster system integrates several personal computers or workstations via high-speed network within a certain region into a computing environment of high-performance. However, the most significant weakness of cluster system is the limitation within a fixed area, hampering the flexibility of the system [3].

The distributed system can be divided into two categories: client/server system and P2P computing [4]. The client/server system has a central server that can provide service such as varied information, e-mail or information search etc. However, the weakness of client/server system is that when the server breaks down, the whole system should be crashed. P2P computing utilizes the network to integrate resources available on many nodes scattered in every region to facilitate a distributed application. P2P computing is a kind of internet technology and also called symmetric internet technology. P2P computing executes applications via computing ability of all members of network and each node can act as client or server [5,6].

Moreover, the usage of network has granted P2P computing with more and more attention. P2P computing is better than conventional distributed systems or client/server system in that the former provides large-scaled resource sharing, enhances resource utilization, and facilitates application over remote or distant network connection. Thus, limitation of space confronting conventional distributed and client/server systems can be overcome, achieving cross-platform operation and thorough exploitation of the abundant.

Thus, using P2P architecture can avoid the burden of server too heavy while only one central

server is provided in traditional client/server system. Moreover, the P2P system can also improve the computing and storing capability of the traditional client/server system. Otherwise, the distributed resource in each node can be used to collaborate completing a massive computation or application.

Therefore, how to utilize the advantage of P2P computing and make each task to obtain the required resources in the shortest time is an important topic. However, in this study, a three-phase load-balancing scheduling algorithm that composes of BTO, TOLB and TLBMM scheduling algorithm is proposed. Moreover, the agent mechanism is used to collect the related information of node to achieve efficient utilization resource and to enhance work efficiency. The rest of this paper is organized as follows. The literature review is discussed in Section 2. The proposed three-phase scheduling algorithm is presented in Section 3. The simulations of the proposed scheduling algorithm are shown in Section 4. Finally, the conclusions and the future works are discussed in Section 5.

2 Related Works

In this section, the related studies on the topologies of P2P computing and scheduling algorithms are introduced.

2.1 P2P topology

In a P2P computing environment, a network structure is constructed as long as nodes can connect each other via communication media. However, the shape of connection between each node is named topology. The topology can be classified by the node shape of align, such as start topology, ring topology, and hierarchical topology [3].

A star topology is designed with each node connected directly to a central node (server). Data on a star network topology passes through the server before continuing to its destination. The server manages and controls some functions of the network which continue provide service for each node. However, this server cannot supply each node directly data storage, it only provides the location of the needed data.

There is only one server in star topology, thus the service to client is limited. A ring topology can be solved the problem through linked multi-server. Namely each node connects to exactly two other nodes, forms a circular pathway for signalling a ring. Moreover, a ring topology provides only one pathway between any two nodes; it may be disrupted by a failure link. Therefore, the major disadvantage of this topology is that if a lot of nodes want to search information, all nodes connected to that node would be overloaded.

Hierarchical system is used in the relevant network application a long time, such as Domain Name Server (DNS). A hierarchical network includes a root sever to responsible verify mechanism, each lower level node must be verified by upper level node. The advantage is only record location of heighten level, thus node data can be reduced effectively [2].

In consequence of the properties of P2P computing, a hierarchical topology is adopted to our investigate framework.

2.2 Scheduling algorithm

Due to the different characteristics of each scheduling algorithm, the characteristics and the suitable applications of each scheduling algorithm need to be visited [7,8].

Opportunistic Load Balancing (OLB) attempts to make each node keep busy, therefore does not consider the present workload of each node. The advantage is simple and reaching load balance but its shortcoming is never considering the expected execution time of each task, therefore the whole completion time (makespan) is very poor.

Minimum Completion Time (MCT) assigns each task in arbitrary order to the nodes with minimum expected completion time of the task. The completion time is simply, but this is a much more successful heuristic as both execution times and node loads are considered.

Min-Min scheduling algorithm establishes the minimum completion time for every unscheduled task, and then assigns the task with the minimum completion time to the node that offers it this time. The minimum completion time for all tasks at each round is considered by Min-Min scheduling algorithm; hence, it can schedule the task that will increase the overall makespan. In addition, the spirit of Min-Min is that every composed of the best is all minimum completion time for allocation resource. However, the biggest weakness of Min-Min scheduling algorithm is it only considers the completion time of each task at the node but does not consider the work loading of each node.

Because of OLB is simply and easy to implement and each node often keep busy. In our research, the OLB is improved to assign the task and balance the load in P2P architecture. In addition, in order to provide the working performance and load balance of each node in the system, the Min-Min will be

improved in this investigates on which it expects to reduce the execution time of each node efficiently.

3. Research Methods

In previous studies, P2P architecture, as related to distributed network topology, included star, ring and hierarchical topology [9]. However, the application of a network is diversifying; the research results of the past cannot fit the requirements [10]. Hence, a topology of P2P is proposed in the study that can satisfy the requirements of task and maintain a stable network service by selecting manager and clustering nodes.

In order to guarantee that each task entering into the system can be completed quickly, the node resources are allocated to each task in order to achieve load balance; this research proposes a *three-phase load-balancing scheduling algorithm*. In the first phase, the Best Task Order (BTO) is used to provide the best order for the task that entering into the system by evaluating the demand and priority grade of each task. In the second phase, the Threshold-based Opportunistic Load Balancing (TOLB) uses a threshold value mechanism to allocate the task equally. In the third phase, the Threshold-based Load Balance Min-Min (TLBMM) is involved to choose the node that will complete the task in the shortest time, in order to enhance the performance of a system and to reach the load balance effectively. The proposed scheduling algorithm is described in detail in the following subsections.

3.1 Peer-to-Peer topology

In the initial state of a P2P network, each node is geographically distributed in a random fashion. This may cause a flooding phenomenon when the network fills with sensing data. A hierarchical P2P topology is proposed in the study that can avoid the flooding phenomenon. However, two mechanisms are proposed to construct the topology: capability value function and dynamic adjust mechanism.

In the capability value function, the properties of task to be processed are comprised of decision variables such as bandwidth, CPU capability, and memory capability. The relative value of each decision variable can be derived using equations or figure correspondence tables. To search for a node that can best meet the demands of task, various weight values are provided to the nodes in accordance with the level of preference for the task. By the progression, the capability value of each node can be determined. The capability function is shown as equation (1).

$$V_j = \sum_{i}^{n} w_i f(x_{i,j}) = w_1 f(x_{1,j}) + w_2 f(x_{2,j}) + \ldots + w_n f(x_{i,j}) \quad (1)$$

$$\sum_{i}^{n} w_i = w_1 + w_2 + \ldots + w_n = 1 \ ; \ 1 \le j \le N; \ 0 \le f(x_{i,j}) \le 1$$

Whereas

$f(x_{i,j})$: The value of the decision variable i in node j.
V_j: The function value of node j.
i: The decision variable i adopted in this capability function with n decision variables.
j: The node j in this P2P environment and there are N nodes in this P2P environment.
w_i: The weight of each decision variable i.

In equation (1), the capability value of each node can be calculated, and the node with the highest capability value is elected as manager. The objective of the capability function is used to elect the manager.

In a three-level P2P topology, the executive nodes in Level 3 are used to execute the subtasks. The Level 2 holds the sub-managers employed to divide the task into logical independent subtasks while the Level 1 consists of a manager that assigns the task to a suitable sub-manager. A three-level P2P topology is shown in Fig. 1.

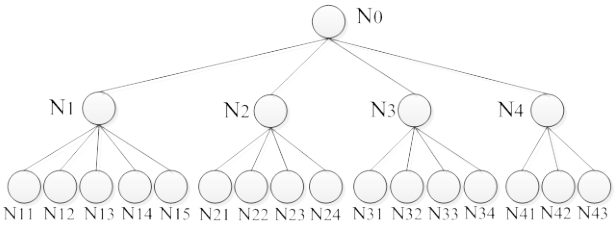

Fig. 1 Three-level P2P topology

Due to the varying requirements of each task, some tasks require the highest processing speed while others require a high rate of bandwidth and transmission. Therefore, according to the properties of the nodes, some nodes are formed as a group via a clustering mechanism. There are three steps of the clustering mechanism:

Step (1) In the initial stage, the capability value of each node is computed; the selected decision variables are depended on the requirements of each application. However, the value of the decision variable for each node is normalized and the value is set between 0 and 1.
Step (2) According to the capability value of each task, the nodes are clustered as a group.

For instance, if the capability value of node is more than 0.8, the node will be allocated to the first group; if the capability value of node is less than 0.6 and more than or equal to 0.4, the node will be allocated to the second group, and so on.

Step (3) After each node is clustered into a group, the node with the mean value of all nodes' capability values in the same group will be elected as the sub-manager of the group.

After clustering, the sub-managers in the Level 2 will be connected with the manager in the Level 1 to build a hierarchical framework by data transmission. Then, the relationship between the higher and the lower levels will be obtained. In other words, the objective of the message transmission mechanism is to establish the entire hierarchical framework through message exchanges.

3.2 Three-phase load balancing scheduling algorithm

In order to reach the load balance and reduce the execution time of each node in the P2P environment, a *three-phase load-balancing scheduling algorithm* is proposed in this study. In the first phase, the BTO scheduling algorithm is used to determine the execution order of each task. In the second phase, the TOLB scheduling algorithm is used to assign a suitable sub-manager for allocation of the executive node. In the third phase, the TLBMM scheduling algorithm is used to guarantee that a suitable executive node will be assigned to execute the task in the minimum execution time.

In the first phase, the best order of task executed will be arranged by the BTO scheduling algorithm. According to the characteristics of each task, an execution order is given and stored in job queue by manager (root node). Therefore, in this study, both job demand and service property of each task are considered and the shouldering algorithm is name BTO (Best Task Order), thereby ensuring that a suitable order for the tasks is assigned by BTO.

(1) Job demand. In a P2P environment, a node is assigned to execute tasks $T_i = \{T_1, T_2, T_3, \ldots, T_m\}$ that enter into system. However, the limited capability of the node may not properly address the task assigned. In order to allow each task to allocate fairly and execute quickly, the execution requirements of a task are considered. In addition, the execution requirements of each task have been stored in the job queue. However, according to the execution requirements that include bandwidth, transmission rate, CPU capability and memory capability, as shown in equation (2), the job demand (JD_i) for each task is calculated. Because each decision variable considered in this research has a different unit in the actual environment, the decision variables need to be normalized. Equation (3) shows an instance of memory normalization.

$$JD_i=\{Bandwidth_i,\ Transmission\ rate_i,\ CPU\ capability_i,\ Memory\ capability_i,\ \ldots\},\quad 0<JD_i\leq 1,\ 1\leq i\leq m \tag{2}$$

$$Memory\ capability_i = Memory\ capability\ of\ task\ T_i\ required/\ Memory\ capability\ of\ max,\quad 0<Memory\ capability\leq 1, \tag{3}$$

In a real environment, there are many tasks need to be executed. In this study, a task with the highest job demand (*JD*) is executed first; then the waiting time of the entire system can be reduced. In addition, each task is completed simultaneously when scheduled in parallel.

(2) Service property. The consideration of this study is not only the requirements of tasks, but also the service property of each task. In many related applications, the response time is one of the most important factors for QoS (Quality of Service). Many related applications for pervasive computing are real-time tasks that must obey a limited response time. Otherwise, the results of the execution will be futile. On the other hand, the related applications of non-real time tasks will not strictly the response time, although the results are still useful when tasks exceed the time limit [1].

Therefore, in this study, a higher priority is given to real time tasks, while the lower priority is given to non-real time tasks. According to the application of tasks, task priority is divided into four grades: Best Effort service (BE), Non-Real-Time Polling Service (nrtPS), Real-Time Polling Service (rtPS), and Unsolicited Grant Service (UGS) [11]. As the fourth grade, a UGS task is very strict with time and has the highest priority. The Priority (P_i) grade for each task can be calculated by equation (4).

$$P_i=The\ priority\ of\ task\ T_i\ /4(grades),\quad 0<P_i\leq 1,\ 1\leq i\leq m \tag{4}$$

In brief, a task with both the highest job demand and the highest priority is severed first. Hence, the order value (O_i) determines the entering sequence of a task The order value is determined by job demand (JD_i) and priority (P_i) grade. The order value (O_i) is

shown in equation (5). After calculating the order value of each task, the task with the maximum order value ($Max\{O_1, O_2, ..., O_m\}$) can enter into system first. In short, the tasks are dispatched to the system from higher to lower order.

$$O_i = JD_i * P_i, 1 \le i \le m \quad (5)$$

By using the BTO scheduling algorithm, the most suitable order is obtained in accordance to the job demand and the service priority of each task. Thus, the waiting time of each task can be reduced.. However, the performance of the entire system can be promoted.

In the second phase, the Threshold-based OLB (TOLB) is proposed in the study that integrates a traditional OLB and a *sub-manager threshold* mechanism.

(1) OLB scheduling algorithm. Traditional OLB assigns tasks to the nodes in an arbitrary order. For instance, to carry out the task, an *m*-by-*n* matrix of two-dimensions is employed where *m* is the total number tasks $T_i = \{T_1, T_2, T_3, \ldots, T_m\}$ and *n* is the total number of nodes $N_j = \{N_1, N_2, N_3, \ldots, N_n\}$. In the *m*-by-*n* matrix, task T_i is assigned in free order to execute at present node N_j, which is usable by the OLB scheduling algorithm. The OLB scheduling algorithm allows each node to keep busy. In other words, the load balance of each node is considered by OLB [12]. However, it has several shortcomings:

- The execution time of each task in node cannot be forecasted.
- The complete time will be extended when system loading is high.
- The suitable between the capability of tasks and nodes does not be considered.

In this research, the OLB scheduling algorithm will be used to assign tasks to nodes in order to reach load balance. However, there are different requirements for each task. The task may not be executed continuously if the node assignment is unsuitable. Therefore, a *sub-manager threshold* is used to filter the adaptive sub-manager in order to guarantee the quality of service.

(2) Sub-manager threshold. The ***sub-manager threshold*** is used to evaluate the capability of each sub-manager. Because the computing complexity of each task is different, it is important to avoid a situation in which a complex task is assigned to a node where the capability is poor. This will result in a situation that an unsuitable sub-manager is chosen and the performance of the system is reduced. Therefore, the decision variables according to the complexity of the task need to be considered. Specifically, the decision variables should be based on the resources required of the task to maintain system performance.

In this study, the assumptions of the computing capability of a node and the execution time of task in each node are given. In addition, the storage capability of a node is addressed, too. The decision variables can be defined as follows:

V_1 = CPU capability;
V_2 = Memory capability.

After evaluating the capability of node with a *sub-manager threshold*, the sub-manager with sufficient resource capability to support the tasks is numbered in *k*, $\{N_1, N_2, N_3 \ldots . N_k | T_i\}$, in order to address the problem that some tasks do not need be supported by a node with high capability that these tasks are actually supported by a higher capability node to carry out.

Therefore, this study selects all nodes that can carry out a task (T_i). Among these nodes, the node with the lowest capability is elected to carry out task T_i, $Min\{N_1, N_2, N_3 \ldots . N_k | T_i\}$. Using this method, the nodes with better capability will be reserved for the more complex tasks. Each task will be assigned to the proper node to enhance system performance. The tasks are dispatched to several clusters according to the requirements of CPU capability and memory capability. For easy clustering, each task is given a job demand value (*JD*) as calculated by equation (2). Through normalization, the values ranged between zero and one. For example, the value of cluster 1 is more than 0.8 while the value of cluster 2 is ranged from 0.6 and 0.8, and so on. In addition, the task is assigned to cluster 3 if the task requirements matched the restricted conditions.

However, the restricted condition for each task is not the same. Because of the necessary restricted condition depends on the application of tasks. Therefore, the different restricted condition of each task based on the different applications need to be considered.

The TOLB scheduling algorithm is proposed that combines the traditional OLB and the proposed *sub-manager threshold*. The TOLB scheduling algorithm not only dispatches the task to the most suitable sub-manager according to the property of task requiring execution, but also maintains the advantage of traditional OLB to reach the load balance.

In the third phase, a Threshold-based Load Balance Min-Min (TLBMM) algorithm is proposed that combines a Load Balance Min-Min (LBMM) scheduling algorithm [2] and an *executive-node threshold* that will guarantee the task is assigned a suitable node to carry out the minimum execution time. The LBMM and the *executive-node threshold* are explained as follows.

(1) LBMM scheduling algorithm. The Load Balance Min-Min (LBMM) scheduling algorithm takes the characteristics of a Min-Min scheduling algorithm. However, an LBMM scheduling algorithm addresses the shortcomings created by the load unbalance of a Min-Min. The objective of a LBMM algorithm is to obtain the shortest completed time for each task and the load balance of each node.

Hence, the LBMM algorithm can improve the Min-Min load unbalance and effectively reduce execution time. The idea behind an LBMM scheduling algorithm is to distribute subtasks among each sub-manager to be executed in a suitable executive node, and to allow each node to keep busy. Initially, the sub-manager is used to divide the task T_i into some logical independent subtasks (T_{i1}, T_{i2}, …T_{ij}, …) where T_{ij} is the subtask of T_i, $1 \le j \le m$ and m is the total number of subtasks of T_i. An m-by-n matrix with m subtasks and n nodes is created by LBMM scheduling algorithm; the matrix value considers the execution time of each subtask at each executive node. The execution time of each subtask at different executive nodes is evaluated by agent. According to the information gathering by the agent, each sub-manager chooses the executive node with the shortest execution time to execute various subtasks and record them into the Min-Time ser. Finally, the Min-Time ser for each subtask will be recorded; this is a set of minimal execution time on certain executive nodes. In the meantime, the sub-manager chooses the executive node from the Min-time ser. This means the αth subtask on the executive node γ is performed first. Therefore, the subtask α will be distributed to executive node γ. Since the subtask α has been distributed to the executive node γ to be performed, the subtask α will be deleted from the subtask queue. Meanwhile, the Min-Time ser will be rearranged and the executive node γ put last in the Min-Time ser, allowing the node without a task to be prioritized. The progression of the LBMM scheduling algorithm is shown in Table 1.

Table 1 The progression of LBMM scheduling

Step 1	The execution time of each subtask in different executive node is determined by an agent.
Step 2	According to the requirements of a subtask, the executive node with minimal execution time is chosen as a Min-Time executive node, and recorded into the Min-Time set.
Step 3	To select the executive node γ from the Min-time set in which executive node γ has the shortest execution time, $1 \le \gamma \le n$.
Step 4	Assign subtask αth to executive node γ.
Step 5	Remove the subtask α that has been executed completely from the required executed task set.
Step 6	Rearrange the Min-Time set, and the executive node γ is placed in the last order.
Step 7	Repeat Step 1 to Step 6, until all subtasks execute completely.

In accordance with the abovementioned steps for the LBMM algorithm, each subtask is dispatched to a suitable executive node to execute, but the capability value of each node is different. While some nodes can complete tasks quickly because they have good capability, other nodes require longer execution times to perform tasks, which results in longer completion times for the entire system. In order to solve the problem, an *executive-node threshold* is proposed as discussed below.

(2) Executive-node threshold. In a P2P environment, the composition of nodes is dynamic: each node can enter a busy state at any time, thus increasing its execution time and lowering its performance level. Thus, the "threshold of executive node" is used to choose the best executive node. The progression is divided into four steps as follows:

Step 1: To compute the average execution time in different executive node for each subtask.

Step 2: If the required execution for a subtask with a minimal execution time in the Min-Time set is less than or equal to the average execute time of a subtask at all executive nodes, then carry out the subtask to execute normally.

Step 3: If the required execution time of a subtask with a minimal execution time in a Min-Time ser is greater than the average execute time of a subtask at all executive nodes, then the executing time is set to ∞ (in this case

the execution time is too long and cannot to be considered). The executive nodes that have been executed subtasks completely will re-enter the system to participate in the execution of the subtask.

Step 4: Repeat Step 1 to Step 3 until all subtasks have been completely executed.

Setting the *executive-node threshold* can remove an executive node that has taken too long to execute. If the executive node has poor capability, a weaker executive node cannot execute the subtask.

However, in the third phase, the TLBMM scheduling algorithm is used to choose the best executive node and then the *executive-node threshold* is used to guarantee that the executive node carries out the task in the shortest time. Thus, the job can be allocated effectively and the best resource allocation is provided.

The *three-phase load-balancing algorithm* is employed in the P2P hierarchical topology. The proposed scheduling algorithm can schedule the task according to the task demand. Thus, each task can be completed effectively and quickly in the P2P topology.

4. Experiment

In our experiment, the network simulation (Network Simulation 2, NS2) designed by the Telecommunications Industry Association is used to simulate the network environment [13]. Besides NS2, other tools such as nsBench and Jbuilder are employed to assist in the experiment.

To create a real-like P2P environment, the following assumptions are made in advance:

1. The requirement of each task is normally distributed. That is, the tasks will not always be dispatched to the same group with the highest or lowest capability ensuring that some groups are not always idle.
2. Each independent and dismembered task is divided into several subtasks and they are completed independently.
3. The necessary execution time of a task can be forecasted. In addition, the task has a different execution time at different nodes.
4. The requirements of all tasks can be processed by the nodes. In other words, all tasks can be carried out if they enter into the system.
5. The capability of each node is normally distributed, namely that each task can be supported in different nodes.

However, NS2 is employed to simulate the network environment of fifty nodes and fifty tasks. The node attributes include Node_ID, Bandwidth, CPU capability, memory capability, and transmission rate. In addition, each task has a priority grade, and the priority grade identifies the order of tasks for entering the system.

Equation (1) is used to calculate the capability value of the fifty nodes. Through a procedure of normalization, the capability value for each node is in the range between 0 and 1. The node that has the highest capability value becomes the manager. Then, these nodes are divided into five groups according to the node capability value. The sub-manager is elected that has the mean of capability value.

In this experiment, we assumed that the most important resources for a P2P network application are CPU capability and memory capability. Thus, the nodes are divided into five groups according to these capabilities. The nodes with the highest capability value are clustered in Group 1; in particular, the capability values of CPU and memory must be bigger than or equal to 0.8. The nodes with lower capability value are assigned to Group 2; the capability values of CPU and memory must be larger than 0.6 and less than 0.8. In addition, Group 3 has nodes with capability value of the CPU and memory in a range between 0.4 and 0.6, and so on.

Distributed scheduling algorithms such as OLB or Min-Min (MM) are widely employed in researches. In our experiment, a TOLB and TLBMM scheduling algorithm with an OLB and Min-Min (MM) scheduling algorithm are compared. The design of experiment is shown in Table 2. However, the tasks are assigned to the nodes based on the three scripts. Each script is combined with two scheduling algorithms, i.e. Script 1 is combined with OLB and MM.

Table 2 The experiment design

Scheduling / Script design	Sub-manager of second phase	Executive-node of third phase
Script 1	OLB	MM
Script 2	TOLB	MM
Script 3	TOLB	TLBMM

The OLB scheduling algorithm and the proposed TOLB scheduling algorithm is used to dispatch the tasks by the manager in the Level 1 to the sub-manager in the Level 2, respectively. Then the MM and the proposed TLBMM is employed in order to assign the subtasks by the sub-manager in the Level 2 to the executive-node in the Level 3. For instance, in Script 1, the OLB scheduling algorithm is employed to allocate the tasks of a sub-manager in

advance. Then, the MM scheduling algorithm is used to assign the subtasks to the executive-node. In the experimental design, the script for combining OLB and TLBMM is not considered. In this case, we cannot find a fit executive node to carry out the subtask, which will result in an endless loop. Therefore, this study only considers the combinations of scheduling algorithms shown in Table 2.

In the simulation, the effectiveness of the proposed method is evaluated by load balance and makespan. These factors will assist in proving the usefulness of the scheduling algorithm.

Load balance: The experiment results are shown in Fig. 2~6.

Fig. 2 shows the state of a load for each node in Group 1 and the results of three kinds of scripts mentioned in Table 2 are compared. The y-axis of the figure represents the makespan: the total execution time for all subtasks in Group 1. The x-axis of the figure represents the assigned identifier for each node in the network topology: the order from the left side to the right side base on the capability value of node from highest to lowest. In particular, node N_{37} has the highest capability to support subtasks that enter into the system, while node N_3 has the lowest capability in Group 1. The figure block shows three script types combined with various scheduling algorithms. The first script combines OLB with MM. The second script combines TOLB with MM, while the third script combines TOLB with TLBMM.

In Fig. 2, the load balance for each node using the three scripts mentioned above is similar. In addition, the tasks aggregate to the left of the y-axis. The main reason is that the nodes with the lowest capability (i.e., N_{35}, N_{23} and N_3), cannot satisfy the requirements of tasks that require higher execution capability value. Because the executive nodes with the best capability value are clustered in Group 1, the tasks that require higher execution capabilities are also assigned to Group 1. In this situation, the worst capability node may not execute any tasks resulting in all tasks aggregating to the left of the y-axis. On the other hand, because of subtask is divided randomly in this study, when the number of subtasks is not large, the subtasks will be executed by the executive nodes with the shortest execution time (such as N_{37}, $N_{13,}$ N_9…). Therefore, the load of the nodes with higher capability value will be heavy.

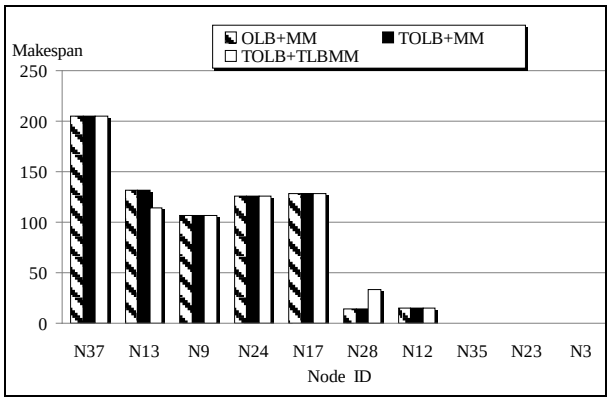

Fig. 2 The comparison of load balance in Group 1

Fig. 3 shows the state of load balance for each node in Group 2. Here, the second script (TOLB+MM) has the worst load state when load gap is used as the evaluated factor that the highest load and the lowest load of each executive node is compared. The load gap for the second script is 130 (147-17). The next worse load state is the first script which combines OLB with MM. Here, the load gap is 122 (139-17). The third script (TOLB+TLBMM) has a better load for each node with a load gap (81-34=47). Because of the TOLB is combined with MM scheduling algorithm. The TOLB scheduling algorithm chooses the sub-manager with the lowest capability value for carrying out a task by using the *sub-manager threshold*. In some cases, the selected sub-manager might just slightly exceed the threshold capability value. Thus, the subtask is dispatched to the group without the best capability value by the *sub-manager threshold*. The subtask may not be executed in the executive node with the shortest time because it may not employ the *executive-node threshold* to remove the inappropriate subtask in the MM scheduling algorithm. The above problem will lead to a situation with the worst load balance.

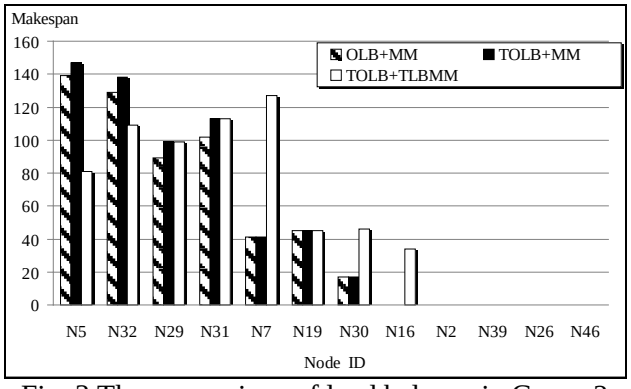

Fig. 3 The comparison of load balance in Group 2

Fig. 4 shows the state of load balance for each node in Group 3. In the figure, the first script shows

the worst load state with a load gap (317-38=279). The next worse load state is the second script that combines TOLB with MM. The load gap is 222 (260-38). The third script has the best load for each node with a load gap (161-38=123). On the other hand, the nodes N_{48}, N_{36}, and N_8 have no subtasks to execute when using the scheduling algorithm of the first and second script. There are not any subtask is assigned to node N_8 by the third script. The main reason for the bad result of the first and second script is that both of them use the MM scheduling algorithm. Thus, all subtasks will select the node (N_{45}, N_{42}) with the highest capability to execute. This situation will leave the nodes (N_{48}, $N_{36,}$ N_8) of worse capability are idle. In the case of the third script, there is only one node will be idle. It can be concluded that the third script has the best load balance according to the comparison results of Group 3.

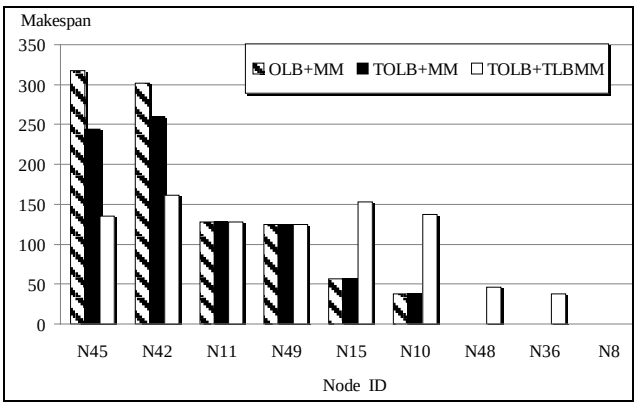

Fig. 4 The comparison of load balance in Group 3

Fig. 5 shows the state of load balance for each node in Group 4. In Group 4, each node has a better load balance because the executive nodes in Group 4 have lower capability value. A subtask entered into the system simply needs a lower execution requirement. Therefore, the node N_{34} with the lowest capability could also execute the subtask. In Group 4, the first script combining OLB with MM has the best load balance because the OLB scheduling algorithm dispatches a task to the sub-manager randomly. Therefore, the task may be assigned to a group that has the best capability to carry out the task quickly. However, this situation is not arisen in every case.

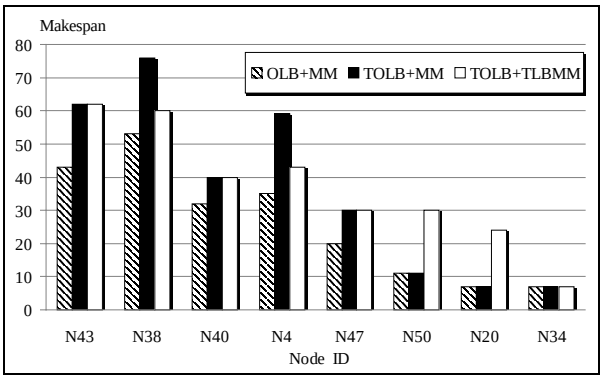

Fig. 5 The comparison of load balance in Group 4

Fig. 6 shows the state of load balance for each node in Group 5. The first script, which combines OLB with MM, has the worst load balance and the load gap is 382(409-27). The next worse load state is the second script that combines TOLB with MM. Here, the load gap is 87 (114-27). The load gap for the third script is 13 (74-61) .The third script displays a perfect circumstance. To compare the result of Group 5 and Group 4, it obvious that the first script is an unreliable scheduling algorithm. The first script sometimes presents a nice situation for the load balance (such as in Group 4) while it can also display the worst load balance (such as in Group 5).

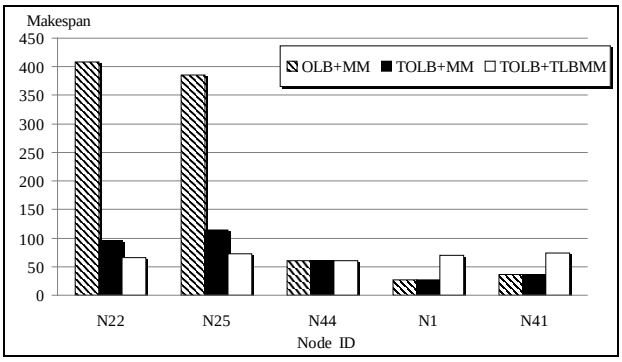

Fig. 6 The comparison of load balance in Group 5

The above experimental results for load balance in the five groups show that the proposed third script surpasses the other two in every assumed situation. In other words, the proposed scheduling algorithm can assign each task to a suitable node and reach a better load balance than other two scripts.

Makespan: The comparison of makespan: the total execution time for all subtasks of each script is shown in Fig. 7.

The y-axis of the figure represents the makespan, while the x-axis of the figure represents the scripts

of the scheduling algorithm composed of various scheduling, and displayed in Table 2. In Fig. 7, the makespan of the first script combining OLB with MM is 409 unit time. The makespan of the second script combining TOLB with MM is 260 unit time. In addition, the makespan of the third script combining TOLB with TLBMM proposed in this study is 205 unit time.

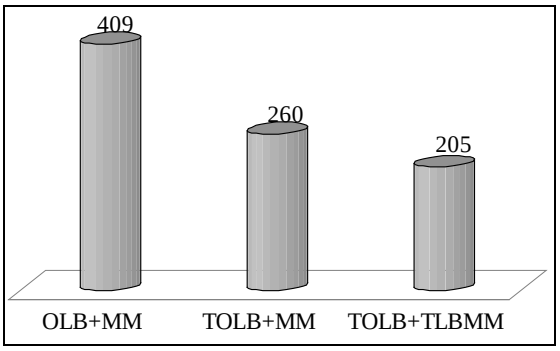

Fig. 7 The comparison of complete time (Makespan) in each scheduling

In the comparison of makespan for the three scripts, (TOLB+TLBMM) is faster than the (OLB+MM) with 55 unit time and faster than the (TOLB+MM) with 204 unit time. The main reason is that the (TOLB+TLBMM) not only considers the load balance for each node but also employs the threshold necessary to remove the unsuitable nodes. How to decide the threshold value is a big challenge for this research. If the threshold value is set too low, then resources may be wasted. If the threshold value is set too high, then the subtask cannot to be executed. Therefore, in our study, the threshold value is designed by the following rules:

(1) If the executive node will finish the working subtask in the shortest time, then the subtask that needs to be executed is waiting.
(2) If the executive node needs cost too much time to execute the subtask that needs to be executed, then the executive node is filtered.

By employing the proposed algorithm, the system can reach the highest execution performance and achieve the load balance of nodes.

The simulation results for the three scripts in the two designed experiments have been evaluated that the scheduling method combining TOLB with TLBMM is more effective than other scheduling approaches for reducing the completion time of a task (i.e. current OLB and MM that have been extensively used to calculate the minimum value). It is superior in the load balance of nodes, the makespan of all subtasks, and at enhancing the execution performance of the system.

5. Conclusion

In the topology of P2P, to ensure that each task entering the P2P system can be completed on time and the resource can be allocated quickly, the topology of a hierarchical P2P is constructed by using a capability calculation mechanism and a dynamical adjusting mechanism in this study. Moreover, the three-phase scheduling algorithm is integrated that included BTO, TOLB and TLBMM scheduling algorithm. The allocation of tasks was based on the related information from nodes collected by the agent. The experiment results of the makespan showed that the proposed script combining TOLB with TLBMM obviously could enhance the performance of a system. In other words, the load balance of nodes is reorganized and the entire execution performance of a system is enhanced by the proposed scheduling algorithm.

In this study, the *executive-node threshold* is set by the average execution time. It is not the best solution for setting threshold. Thus, in the future, the ideal *executive-node threshold* will be determined by simulating the ideal situation for the execution performance of a system. In the meantime, the clustering method of P2P network and the number of nodes within the clusters will be discussed continuously.

References:

[1] E.A. Sanchez, "Future Convergent Telecommunications Services: Creation, Context, P2P, QoS, and Charging," IEEE Communications Magazine, Vol. 49, Issue 1 2011, pp. 58-59.
[2] K.Q. Yan, S.C. Wang, S.F. Chen, S.S. Wang, "Reaching Efficient Load Balancing by Two-phase Scheduling in Three-level Peer-to-peer Network," 18th Annual Conference International Information Management Association (IIMA), Beijing, China, Oct. 16-19, 2007, 1-13.
[3] N. Olifer, V. Olifer, Computer Network: Principles, Technologies and Protocols for Network Design, 2006, John Wiley & Sons.
[4] L.Y. Tseng, Y.H. Chin, S.C. Wang, "A Deadline-based Task Scheduling with Minimized Makespan," International Journal of Innovative Computing, Information and Control, Vol. 5, No 6, June 2009, pp. 1665-1679.
[5] G. Muneeswari, K.L. Shunmuganathan, "Agent Based Load Balancing Scheme using Affinity Processor Scheduling for Multicore Architectures," WSEAS Transactions on Computers, Vol. 10, Issue 7, 2011, pp. 247-258.
[6] E.L. Silva, P. Linington, "A P2P based

Scheduler for Home Grids," Communications in Computer and Information Science, Vol. 162, Part 3, 2011, pp. 325-336.

[7] H.C. Hsiao, H. Liao, S.T. Chen, K.C. Huang, "Load Balance with Imperfect Information in Structured Peer-to-Peer Systems," IEEE Transactions on Parallel and Distributed Systems, Vol. 22, Issue 4, 2011, pp. 634-649.

[8] M.L. Pinedo, Scheduling Theory, Algorithms, and Systems, 2011, Springer.

[9] M. Tripathy, C.R. Tripathy, "A Distributed Shared Memory Cluster Architecture with Dynamic Load Balancing," WSEAS Transactions on Computers, Vol. 11, Issue 5, 2012, pp. 121-130.

[10] G. Xu, J. Pang, X. Fu, " A Load Balancing Model based on Cloud Partitioning for the Public Cloud," Tsinghua Science and Technology, Vol. 18, Issue 1, 2013, pp. 34-39.

[11] C.H. Jiang, T.C. Tsai, "Token Bucket based CAC and Packet Scheduling for IEEE 802.16 Broadband Wireless Access Networks." Proceedings of Conference on Consumer Communications and Networking (CCNC 2006), 2006, Vol. 1, pp. 183-187.

[12] M. Tripathy, C.R. Tripathy, "A Distributed Shared Memory Cluster Architecture with Dynamic Load Balancing," WSEAS Transactions on Computers, Vol. 11, Issue 5, 2012, pp. 121-130.

[13] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, H. Yu, "Advances in Network Simulation." IEEE Computer, 2000, Vol. 33, No. 5, pp. 59-67.