Towards a Hybrid Load Balancing Policy
in Grid Computing System

Kuo-Qin Yan, Shun-Sheng Wang, Shu-Ching Wang*, Chiu-Ping Chang
Chaoyang University of Technology
168, Jifong E. Rd., Wufong Township, Taichung County 41349, Taiwan, R.O.C.
Email: {kqyan; sswang, scwang*; s9314614}@cyut.edu.tw
* Corresponding author to provide phone: +886-4-2330-000; fax: +886-2330-4902

Abstract

Grid computing has become conventional in distributed systems due to technological advancements and network popularity. Grid computing facilitates distributed applications by integrating available idle network computing resources into formidable computing power. As a result, by using efficient integration and sharing of resources, this enables abundant computing resources to solve complicated problems that a single machine cannot manage. However, Grid computing mines resources from accessible idle nodes and node accessibility varies with time. A node that is currently idle, may become occupied within a second of time and then be unavailable to provide resources. Accordingly, node selection must provide effective and sufficient resources over a long period to allow load assignment. This study proposes a hybrid load balancing policy to integrate static and dynamic load balancing technologies. Essentially, a static load balancing policy is applied to select effective and suitable node sets. This will lower the unbalanced load probability caused by assigning tasks to ineffective nodes. When a node reveals the possible inability to continue providing resources, the dynamic load balancing policy will determine whether the node in question is ineffective to provide load assignment. The system will then obtain a new replacement node within a short time, to maintain system execution performance.

Keywords: Grid computing, task scheduling, load balancing, distributed system

1. Introduction

In recent years, network bandwidth and communication quality have drastically improved. The various communication and computing responsibilities, in fields such as telecommunications, multimedia, information technology, and construction simulations, have been integrated and applied to the distributed computing environments. Cluster
computing and Grid computing are several options for establishing distributed computing (Foster and Kesselman, 1999). Cluster computing combines several personal computers or workstations, to conduct distributed applications through local networks in a given area. The main disadvantage of cluster systems is that they are limited to a fixed area. This gives rise to the application being inflexible in variation. Grid computing makes use of the network from different geographic locations and combines idle resources for distributed applications. Grid computing is characterized from conventional distributed computing and cluster computing by its focus. This is based upon large-scale resource sharing and innovative applications, in a widely connected network such as the Internet (Yang et al., 2003). Therefore, the limit of space on conventional distributed systems can be broken to gain cross-platform features and fully exploit the large resources of idle computers (Fujimoto and Hagihara, 2004).

It is acknowledged that Grid computing concept based projects are continuous, such as SETI@Home (Anderson et al., 2002), Folding@Home (Folding@Home, 2006) and Genome@Home (Genome@home, 2006). These projects combine the network idle resources made available by volunteer providers, to form super computing capability for the simultaneous execution of many projects. In a Grid environment, many nodes are capable of providing computing resources. Some of them are frequently idle with the ability to provide constant computing resources and others are not. Therefore, if the selected node cannot provide steady resources for assigning tasks, the tasks will be frequently reassigned and re-executed. This will cause reduction in system performance and task run time will become longer. The selection of efficient nodes to provide effective and stable resources is the goal of this research. As a result, we propose a hybrid load balancing policy to select efficient nodes in the shortest time to maintain system execution performance.

Section 2 of this study focuses on the literature review. Section 3 details the research method and describes the structure of the proposed system, including the system environment design and load balancing policy. Details of the simulation experiment are in Section 4. Section 5 provides the analysis and further verifies that the proposed system achieves the goal of exploiting resources and enhancing system performance. Finally, conclusions are presented in Section 6.

2. Literature Review

Computing usage in different science domains, allows scientists to use various task-scheduling algorithms to utilize resources effectively. This raises performance in enhancing execution performance and maintaining system load balancing. Still, resources are contributed by idle computers in the Grid computing environment. The
contributed resources will change with time. This happens when new resources join or old resources exit the Grid environment. In this dynamic and time-variant environment, designing a task-scheduling algorithm to dynamically change according to the variation in resources, requires considering numerous and complicated factors. This complex scheduling will cause greater load on the system, which seriously affects system performance. For this reason, a proper task-scheduling algorithm is very important in the Grid environment. The following sub-sections will show the conventional scheduling methods and the load balancing policy in a distributed system.

2.1 Conventional Task Scheduling Algorithms

Conventional scheduling algorithms focus on the number of factors to be considered, when scheduling resources to tasks. Some algorithms consider a single factor, while others consider multiple factors. Different algorithms are suitable for different applications. Consequently, each characteristic and suitable domain should be considered in selecting a proper scheduling algorithm.

2.1.1 Single Factor Scheduling Algorithm

In this method, only a single factor is considered, such as node resource or task execution demand. Based upon this knowledge, this scheduling method is simpler and does not increase the system load in practical applications. Because only one factor is considered, this method is suitable for pure and stable environments. It is not suitable for complex and multi-variant environments.

(1) First Come First Served (FCFS) Scheduling Algorithm

FCFS is the simplest form of a scheduling algorithm. It is based on the theory that resources are assigned to the primary proposed tasks. After the first task is completed, the resource is re-assigned to the next proposed task. This method uses the order of executed tasks in the order of submission. If a significant amount of time is required to complete the first task, the same amount of time is spent waiting for the execution of the next task. Therefore, a convoy effect is created and the entire system performance will be reduced (Saleem and Javed, 2000).

(2) Priority-Scheduling Algorithm

Priority scheduling algorithms give priority to the order of task execution. If more than two tasks have the same priority, then the FCFS method will be applied (Saleem and Javed, 2000). Since the order of execution is defined by the order of priority, the priority order decision is the biggest problem in the priority-scheduling algorithm. If the priority order decision is incorrect, then the resources will be continuously occupied by a high priority task, causing unlimited deadlocks or starvation.
2.1.2 Multiple Factors Scheduling Algorithm

The characteristics of this approach are the significance of simultaneous considerations of nodes and resource demand loads. Although multiple factors concurrently considered and tasks are efficiently completed, a greater load will be placed on the system. Therefore, this approach is more suitable for complex and volatile environments.

(1) Minimum Completion Time (MCT)

MCT assigns each task, to the node, in an arbitrary order with the minimum expected completion time for that task (Fujimoto and Hagihara, 2004). This method causes assignment of some tasks to nodes that do not have the minimum execution time for that task (Braun et al., 2001). The logic behind MCT is to combine the benefits of Opportunistic Load Balancing (OLB) and Minimum Execution Time (MET) (Ritchie and Levine, 2003), while preventing the conditions in which OLB and MET perform poorly. This is a much more successful exploratory problem-solving technique, as both execution times and node loads are considered.

(2) Min-Min

Min-min establishes the minimum completion time for every unscheduled task, in the same way as MCT. It then assigns the task to the node that offers the minimum completion time. Min-min uses the same intuition as MCT, but since it considers the minimum completion time for all tasks; it can schedule at each iteration, the task that will least increase the overall make-span, to help balance the nodes better than MCT (Braun et al., 2001).

As shown in the above task scheduling methods, using a single factor is simpler and does not add extra load to the system. However, this approach is not suitable in a complex environment. The consideration of multiple factors is suitable for a complex environment, but greater loads will be placed on the system. In designing an optimal scheduling method, the customization of the system environment and scheduling method burden on the system is important.

2.2 Load Balancing Policy

The Grid combines idle resources of nodes scattered in different locations on the network. The most critical issues pertaining to a distributed system is how to integrate and apply every computer resource node in the distributed system. The integration and application are important for the ability to enhance performance, enhance resource sharing, and increase availability. Load balancing is extremely important in the distributed environment. This is due to the reason that each node has a different processing speed and system resources in the distributed system. Load balancing must
perform a critical role in enhancing the utilization of each node and shortening the expenditure of time. In addition, policy and method of load balancing will directly affect the performance of the system. In general, load-balancing policies for distributed system can be categorized into either a static load balancing or a dynamic load balancing policy (Dandamudi, 1998).

2.2.1 Static Load Balancing Policy

Simple system data, such as the information from the system and the operation cycle are used in static load balancing. In relation to these data, tasks are distributed through mathematical formulas or other adjustment methods. This is so each node in the distributed system can process the assigned tasks until completed. The advantage of this method is that the requirement to collect system information at all times is not considered necessary. The system can be built still some of the nodes have lower utilization rates, because the process does not dynamically adjust with the system information and there is a certain degree of threshold on system performance (Braun et al., 2001, Dandamudi, 1998). The common static load balancing policies are the Speed-Weighted Random Splitting Policy and the Load-Dependents Static Policy.

2.2.2 Dynamic Load Balancing Policy

Dynamic load balancing policy refers to either the current state of the system or the most recent state at system time. This is used to decide how to assign tasks to each node in the distributed system. When the system is overloaded, the goal of a dynamic balance is to move the overloading task to other nodes and processed. However, the migration of tasks to other nodes will induce extra overhead to the system (Dandamudi, 1998). This is because the system must reserve some resources to collect and maintain the system state. When overhead is controlled and limited to a certain reasonable range, dynamic load balancing policy has better performance than static load balancing in most conditions (Dandamudi, 1998). The general dynamic load balancing policies include the Shortest Queue policy, the Never Queue policy, the Maximum Throughput policy and the Adaptive Separable policy (Banawan and Zeidat, 1992).

There are distinct advantages and disadvantages to both static load balancing and dynamic load balancing policies. The advantage of static load balancing policy is it does not require an endless collection of system status and only analyzes with available related information. Therefore, no extra overhead will be created. A disadvantage is it cannot dynamically adjust with the state of the system, because related information is not continuously gathered. For this reason, it is unable to adjust the loads of each node satisfactorily.

The dynamic load balancing policy is able to adequately adjust the loads of each node and task assignment. However, real-time monitoring will cause system overhead
(Banawan and Zeidat, 1992, Dandamudi, 1998). In related research (Zhou, 1988), it has been shown that before the execution of tasks, if the completion time of a task is calculated in advance, static load balancing policy may have better effects. This is because the algorithm, adopted by static load balancing policy, is simpler and will not produce extra load on the system. If there are frequent variations in load conditions of the system, then dynamic load balancing policy is more suitable than static load balancing policy. The nodes are composed of idle nodes in the Grid environment, so provided resources are very dynamic and not resolvable with purely static load balancing policy or dynamic load balancing policy. Therefore, this study proposed a hybrid load balancing policy that adopts static load balancing policy. If any node is likely to be ineffective, dynamic load balancing is adopted to avoid the reduction of system performance due to the variation of nodes. Further details of the research method, the proposed system structure, the system environment design and the load balancing policy are shown in Section 3.

3. Hybrid Load Balancing Policy

In the Grid environment, many nodes are able to provide computing resources. Some nodes are always in an idle state and capable of providing resources steadily for a long time, while other nodes are not capable. The performance of the system deteriorates, because the system chooses ineffective nodes, which generates task re-distribution and re-execution. As a result, an important issue in the study of Grid computing is how to choose a set of effective nodes that provide resources steadily. This pairs with the issue of determining, when a node is ineffective, a new node be located in the shortest time to replace the ineffective nodes and maintain the execution performance of the system.

To solve these problems effectively, this research proposes a hybrid load balancing policy. This policy combines static load balancing policy and dynamic load balancing policy to reach load-balancing policy of the system. When a request for executing a task is called for, the system adopts the static load balancing policy. To begin, division of the proposed task into several subtasks must occur. Based on the lowest demand of the subtask, a table of effective nodes is created with the Value Function (VF) and needed nodes are selected from the table of effective nodes. At the same time, there is assignment of the subtasks, to the selected nodes to avoid the selection of ineffective nodes. When a node sends a message to the dispatcher, that indicates it is no longer able to provide resources or there is a delay in completion of the subtask, the causes the initiation of the dynamic load balancing policy. In the dynamic load balancing policy, the first action is for the agent to collect information from each node and compare the
collected information with records of nodes to define whether the node is effective or ineffective. If the node is effective, there will not be adjustments to the assigned subtasks, but the completion time of the subtask must be re-predict and re-calculated. If the node is ineffective, a selection will be made of the node with the highest value of VF, from the table of effective nodes, to replace the ineffective one. Then re-assigns the subtask to this new node, to reach system load balancing and maintain system performance.

3.1 Design of the System Environment

The structure of the proposed system will be explained in this subsection. The structure of the proposed system mainly consists of a dispatcher, which manages the system related task nodes, to join the Grid and provide resources. More to the point, in building a real Grid environment, conditions of the systems environment must be as follows:

1. Each new node that joins the Grid has to provide its hardware information, such as CPU speed, size of memory, etc.
2. The dispatcher is a manager in the Grid environment that administers the state of each node and selects nodes to execute tasks.
3. The time and the level of resource contribution of each node are recorded as a reference for decision-making in the future.
4. Related information, such as needed CPU and memory usages, of the resources should be provided when a user requests the dispatcher for resources (Tourino, 2005).
5. The user can precisely provide the information of the demanded resource.
6. The division of tasks into independent subtasks and the information of running tasks are provided.
7. The execution of each subtask is completed in a finite time.

3.1.1 The Dispatcher

The role of the dispatcher in the system is to manage tasks. These include load balancing, holding the status of each node in the Grid, node selection for task execution and assignment-adjustment of tasks for each node. For the dispatcher to accomplish its mission in the most efficient way, the dispatcher has the mechanisms of an agent at its disposal for node selection and effective node calculation. The details of the dispatcher role are shown below.

1. The Mechanism of an Agent

In this system environment, the agent manages the nodes to provide resources in this Grid environment. When the agent is dispatched, it will collect related information
from each node participating in the Grid environment. This information includes CPU utilization, remaining CPU capability, remaining memory, the execution condition of the assigned task, etc. After all these data are collected, they will be made available to the dispatcher, which will assist it in maintaining the system load balancing.

(2) The Mechanism of Using VF to Evaluate the Values of Each Effective Node

The composition of nodes in the Grid environment is dynamic and each node is likely to enter a busy state at any time, which lowers the performance of the system. As a result, when selecting nodes, the available resources of a node cannot be the sole factor of consideration. Other factors must be considered for effectiveness of nodes, such as the node’s historic level of contribution, the task completion rate, the possibility of continuous provision of the resource, etc. Therefore, a Value Function (VF) is proposed in this study to evaluate the value of each effective node and provide a reference for selecting effective nodes.

In this VF, the decision variable is the characteristic of the task to be processed. The compared differences between nodes cannot use direct values, because the derivation of adopted decision variables comes from different approaches. In this way, each decision variable results from a mathematical equation and the unit has to be standardized. Applied formulas can calculate the value of each node in a decision variable or the building of a correlation table to determine the corresponding value of each decision variable for comparison. Each task has different characteristics and the level of preference for each decision variable may be different. As examples, the need for a faster CPU for complex computing or the requirement of larger bandwidth when a large amount of messages have to be transmitted among different nodes. Different weight values are given to each node in accordance with the level of preference for the task. This includes the search for a node that meets the demand most and the selection of the nodes most suitable for the execution of the task. For these reasons, the Value Function (VF) is shown as Equation (1) (Yang et al., 2003):

$$V_j = \sum_{i}^{n} w_i f(x_{i,j}) = w_1 f(x_{1,j}) + w_2 f(x_{2,j}) + \ldots + w_n f(x_{n,j}) ; \sum_{i}^{n} w_i = 1, 1 \leq j \leq N, 0 \leq f(x_{i,j}) \leq 1$$

(1)

Where

- $x_{i,j}$ : the value of the decision variable $i$ in node $j$.
- $f(x_{i,j})$ : the function value of the decision variable $i$ in node $j$.
- $V_j$ : the estimated value of node $j$.
- $i$ : the decision variable $i$ adopted in this value function and the total of $n$ decision variables.
- $j$ : the node $j$ in this Grid environment and the total of $N$ nodes.
- $w_i$ : the weight value of each decision variable.
(3) The Mechanism of Using VF for Node Selection

A derived threshold is used to make the dispatcher select appropriate nodes effectively. This threshold comes from the demand for resource needed to execute the task and will evaluate all of the nodes in the system. The nodes that pass the threshold are regarded as effective and will be organized into a table of effective nodes. Through the mechanism of VF, the value of each effective node can be calculated and listed in ascending order. To attain the task distribution status, all assigned tasks and execution tasks will be incorporated into an execution aggregate (R). The ‘R’ will also predict the time required by each node to process the task. In contrast to the assigned nodes, the unassigned nodes will be incorporated into a waiting aggregate (W). When tasks are again re-distributed to find new nodes, the table of effective nodes is adjusted and ‘W’ becomes the main operation. When any node in the ‘R’ accomplishes its assigned task, it will be transferred to the ‘W’ to wait for another assignment.

Due to the reason when nodes are changed, the state of nodes cannot be predicted in advance. Therefore, it is very important for the table of effective nodes to be changed. If the table of effective nodes is adjusted frequently, the load on the dispatcher will be very tedious. However, if the table of effective nodes does not adjust at an appropriate time, the state of the nodes will not be known. Therefore, two (2) approaches are designed as follows.

(A) Steady State System Environment

When the state of nodes in the Grid environment is not frequently changed, the table of effective nodes must be updated and re-adjusted whenever a new node joins or an existing node exits the system. There are several advantages to this option, they include the prompt indication of effective nodes from which newly joined and nodes that are more effective have a chance to be selected. The disadvantage is the load on the dispatcher may increase, because the environment varies frequently. Consequently, this approach is not suitable for environments with high frequencies of variations.

(B) Frequently Changing System Environment

This approach is used, in the environments where states of nodes frequently change, to set the threshold for the table that frequently changes and may add extra loads on the dispatcher. Specifically, upon the exit of an effective node, the node will be recorded as an unavailable node on the node table. When a new node joins, the table of effective nodes will not be updated immediately, but will be updated when the node variation reaches the threshold.

3.1.2 Node

In this system, nodes play the role of assisting the execution of tasks. When any node is in an idle state and can provide its resource, a “join” message and related
hardware information will be transmitted to the dispatcher. When it can no longer provide its resource, it will transmit an “exit” message to the dispatcher.

3.2 Load Balancing Policy

To maintain system load balancing, this study proposes a hybrid load balancing policy, consisting of a static and dynamic load balancing policies. In the static load balancing state, when the request for executing a task is created, the task will be assigned to an appropriate node to achieve the goal of load balancing. In addition, in the state of dynamic load balancing, the system will adjust dynamically according to the load balancing until it reaches a balance.

3.2.1 Static Load Balancing

Since Grid environment resources are provided by idle nodes, when a request for executing a task is proposed, the task is divided into several independent subtasks by the dispatcher. The lowest requirement of subtasks needs for nodes is the threshold of the table of effective nodes. Nodes passing this threshold are considered as effective nodes. All effective nodes can be organized and built into a table of effective nodes, which are optimized for the proposed task and the number of required nodes is decided. If the total amount of nodes in the table is smaller than the needed amount, a portion of the subtasks will be assigned to effective nodes and the remaining subtasks will be processed when new nodes are added to the table of effective nodes.

For the reason that Grid environment resources are composed of idle resources, the performance of each node may not be the same. The execution time required for running a subtask by nodes with similar performance standards is almost equivalent. In contrast, the execution time required for running a subtask by nodes with different performance criteria is ultimately different. Based on this knowledge, the two (2) cases of performance with nodes of similar and different criteria are discussed as follows:

(1) **Ideal Case - Nodes with Similar Performance**

If Grid environment resources are composed of nodes of similar performance, task completion time is not affected by the subtask distribution method. As a result, it is only necessary to decide the number of needed nodes and distribute each subtask to the selected nodes for execution. For example, if total $k$ nodes are required to provide assistance for the execution of a specific task, the $k$ nodes with the highest values of VF will be selected from the table of effective nodes to assist in task execution.

(2) **General Case - Nodes with Different Performance**

If Grid environment resources are composed of nodes of different performance, the execution time of each node is different. Therefore, the method to distribute subtasks to nodes is an important issue. This study proposes a VF in which the value of a node is
determined according to the fitness level of the node and the subtask. In this manner, the nodes with the highest values of VF will provide the most stable resources. To complete the task securely, a less effective node is selected to estimate the execution time required for each subtask. Then the subtasks are classified by execution time and the subtasks are distributed to nodes using the node value and subtask execution time. The subtask with the longest execution time is distributed to the node with the highest value. Then the subtask of the second-longest execution time is distributed to the node of the second-highest value. To describe specifically, the task must be divided into $k$ subtasks and then the top $k$ nodes, with the highest values of VF, will be selected from the table of effective nodes. The $k$-st is a base to estimate the execution time for each divided subtask and subtasks are sorted by execution time. The largest subtask is distributed to node of the highest value, and the second largest subtask is distributed to node of the second highest value.

3.2.2 Dynamic Load Balancing

In a Grid environment, the effectiveness of nodes may vary with time at any time. Thus, the assignment of tasks must be dynamically adjusted in accordance with the variation in node status. The variation of the node status can be identified in two conditions; firstly, when the dispatcher receives the message that a certain node can no longer provide resources, and secondly, when the execution of a certain node exceeds the expected time. When any of the above situations is occurred and detected by the dispatcher, the dispatcher will launch the agent mechanism to collect the related data of all the nodes in the table of effective nodes and compare the collected data with historic ones, in order to confirm if the node is still effective. If the node remains effective, the distribution of the task will not be re-adjusted, but the time required for the execution of node of the task will be estimated again. If the node confirmation is ineffective, then a selection is made from an ineffective node, with the highest value of effectiveness.

4. Design Simulation

In this section, simulation experiments were conducted according to the hybrid load balancing policy we proposed, FCFS (Ritchie and Levine, 2003, Yang et al., 2003), LIFO (Yang et al., 2003), CPU-based (Yang et al., 2003), and Random-selection (Yang et al., 2003). Based on the demands for computing resources for the execution of a task and the transmission rate, simulation tests were conducted using four different conditions. Then the conditions were compared to other node-selection mechanisms in terms of performance. This was done to verify if the policy proposed by this study could produce a better performance.
4.1 The Simulation Process

This subsection will explain the process of simulations with the proposed hybrid load balancing policy, FCFS (Ritchie and Levine, 2003, Yang et al., 2003), LIFO (Yang et al., 2003), CPU-based (Yang et al., 2003), and Random-selection (Yang et al., 2003). Due to the different performances of nodes in a Grid environment, the subtask is assigned to nodes with the same performance and that require similar execution time. If the subtask is assigned to nodes with different performances, the resulting execution time will be different. As a result, this simulation was performed using two (2) separate cases: the ideal case and the general case. Experiment 1 focuses on the ideal case and simulation of the general case was conducted in Experiment 2.

4.1.1 Experiment 1 – The Ideal Case

The performances of the nodes in Experiment 1 are the same, so task completion time is not affected by the subtask to node distribution. Therefore, four different common algorithms will be employed in the simulation to compare with the hybrid load balancing policy. These include FCFS (Ritchie and Levine, 2003, Yang et al., 2003), LIFO (Yang et al., 2003), CPU-based (Yang et al., 2003), and Random-selection (Yang et al., 2003). The process is as follows:

Step1: The proposed task is divided into ten independent subtasks.
Step2: Ten nodes are chosen by different methods.
Step3: The subtasks are distributed to each selected node.
Step4: If the selected node is unable to complete the assigned subtask, a new node will be found by a different task-scheduling method and the subtask will be re-distributed and re-executed.

4.1.2 Experiment 2 – The General Case

Experiment 2 focuses on the performance of nodes that are different. As the algorithms of FCFS, LIFO, CPU-based, and Random-selection are only suitable for node selection, the execution order will not be considered. However, when there are different nodes of performance, task completion time may vary with the order of execution. To be precise, different sizes of subtasks will produce different combinations of nodes and subtasks. Therefore, the above methods are not suitable for this experiment. As a result, in Experiment 2, two task-scheduling algorithms, MCT (Ritchie and Levine, 2003) and Min-min (Ritchie and Levine, 2003), will be utilized and compared with the proposed method. The process is as follows:

Step1: The proposed task is divided into ten independent subtasks.
Step2: Ten nodes are selected and assigned to a task by three different task-scheduling methods.
Step3: If any of the nodes are unable to complete the assigned subtask, a new node
will be found by a different task-scheduling algorithm and the subtask will be re-distributed and re-executed.

4.2 The Design of a Hybrid Load Balancing Policy

The proposed hybrid load balancing policy is carried out in two phases. The first phase, a static load balancing policy selects the desired nodes. If any node is unable to complete the assigned subtask, a new node will be located using the dynamic load balancing policy.

4.2.1 The Static Load Balancing Policy

This method divides the task into several independent subtasks and designates the lowest resource demand of each subtask to be the threshold value. In accordance with the characteristics of the task, the decision variables in the VF are defined and a weighted value is set by the level of preference for the task. In the VF, decision variables are given a different setting, dependent of the focused factor in the actual application. After the decision variables are defined, the value of each node is estimated with the VF, to select the nodes most suitable for the execution of the task.

During the experiment, the available CPU capacity, the size of available memory, the transmission rate and the past completion rate are the four factors considered as the threshold for the VF in selecting the nodes and the decision variables for estimating node values. This is due to the Grid environment being composed of heterogeneous systems, where the structures of each system may be greatly different. As an example, the computing capability provided by the CPU and the available size of memory are different. Since Grid environment computing utilizes idle resources of each node, the available resources of each node may vary in a busy condition. Using the perspective of task completion time, the available CPU capacity and the size of available memory are the two decisive factors for the duration of execution. Therefore, in this experiment, the available CPU capacity and the size of available memory are taken as the threshold for VF and the decision variables for estimating node values.

The Grid environment is created by connecting nodes scattered in every region throughout the network and the data transmission time is closely related to the bandwidth of the linked nodes (Chen et al., 20014). In this experiment, since nodes may not provide as much transmission rate as the bandwidth due to ongoing execution of tasks, the transmission rate of each node is taken as the threshold for the VF to select the nodes and the parameters for estimating node values. The effectiveness of the nodes in the Grid environment may vary with time and the status of the nodes in the next moment may not be completely predicted. However, the historic records can be used to predict the time that nodes can provide resources. In this study, it is known that the
nodes are capable of completing the execution of the assigned task as expected. In addition, since the time of providing resources for Grid environment computing is longer, the past completion rate of the nodes was taken as the threshold for the VF to select the nodes and one of the variables for estimating node values.

After the decision variables for the VF are determined, each of the variables has to be quantified, to make every decision variable comparable. In this experiment, the variables of CPU capacity and the size of available memory are quantified using Equations (2) and (3). However, due to the fact that the transmission rate between the dispatcher and each node is limited to their network bandwidth, the network bandwidth of the dispatcher is taken as the denominator, as shown in Equation (4), to quantify the transmission rate of each node.

\[ CV = \frac{(CS-CU)}{CS} \]  

\[ MV = \frac{(MS-MU)}{MS} \]  

\[ RV = \frac{NR}{DW} \]

After the value of each node is calculated, a table of effective nodes can be built. The top ten nodes with the highest values are selected based on the design of this experiment. In Experiment 1, if the nodes have the same performance, the task distribution order is not considered and the subtasks are assigned to each selected node one by one. In Experiment 2, the nodes are of different performance, which causes different distribution orders to result in different completion time. To minimize the task completion time, the subtasks are organized by size, distributed to nodes by the execution time of the task and the value of nodes. This means that the largest subtask will be assigned to the node of the highest value of VF.

4.2.2 The Dynamic Load Balancing Policy

If any of the nodes cannot complete the assigned task, the node with the highest value of VF will be examined from the table of effective nodes, to replace the ineffective node and resume subtask execution.
4.3 The Design of the Simulated Environment

This subsection details the design of the environment of the simulative experiment. To create an authentic Grid environment, the following assumptions are made in advance:

1. The proposed task has to provide the information about the required resources, such as CPU, memory requirement and amount of data to be transmitted.
2. Each node operation is time-limited.
3. If any node is busy and unable to continue the operation, it has to send a message of exit to the dispatcher.

According to the previous assumptions, this experiment is carried out. However, the Grid environments with 100, 200, 300, 400, 500, 600, 700, 800, 900, and 1000 nodes are dynamically created by Network Simulation – Version 2 (NS-2) (The network simulator version 2, 2005). Data packages from the Grid environment are generated in random sizes and transmitted in Constant Bit Rate (CBR). The transmission rates between the dispatcher and every node are tested. To simulate the heterogeneity of nodes in the Grid environments, past task completion rates of each node are generated at random using the CPU capability, the memory size, the CPU usage and the memory usage. Included in this simulation, the effective time of each node is generated at random and then multiplied by the past task completion rate. This is used to reflect the relation between the past task completion rate and effective time for the node.

In this study, the experiment is conducted using the following four conditions. Which are according to the computing resource and amount of data transmission required to execute the task.

**Condition 1**: demand for computing resource is large \(25,000\text{GHz}\), and the amount of data transmission is small \(100\text{MB}\).

**Condition 2**: demand for computing resource is small \(1,000\text{GHz}\) and the amount of data transmission is large \(300,000\text{MB}\).

**Condition 3**: demand for computing resource is large \(25,000\text{GHz}\), and the amount of data transmission is large \(300,000\text{MB}\).

**Condition 4**: demand for computing resource is small \(1,000\text{GHz}\) and the amount of data transmission is small \(100\text{MB}\).

4.4 The Design of the Estimation of Factors

Based on the above four conditions, the task completion time and times of task re-distribution are factors for evaluation. The task completion time includes the time of task transmission to the node and the time that the node requires to complete the task. To verify that the nodes selected by VF performed better than those by other task
scheduling methods, the VF of different sets of weighted values are evaluated. Each set is simulated 100 times, to obtain objective data. Using the VF of different weighted values, three different variables will be compared on the task completion time and the number of task redistributions, to other task-scheduling methods. These variables are the worst weighted set that produces the longest completion time (VF-MAX), the average completion time of all weighted sets (VF-AVG) and the best-weighted set that produces the minimal completion time (VF-MIN).

5. Analysis of the Experiment

This section focuses on two experiments using different task dividing methods. The resultant analysis and discussion are below.

5.1 Experiment 1

This focuses on the simulation of the nodes with similar performances in four different conditions, with the analysis and discussion of experimental results.

5.1.1 Condition 1

When demand for computing resource is large and amount of data transmission is small, the available CPU capacity provided by the node will affect task completion. Therefore, the node-transmission-rate impact on the task completion time is not significant. The experimental results are shown in Figure 1 and Figure 2. These are due to the algorithms of FCFS and LIFO selecting the nodes by the order of node’s arrival and the related factors of the nodes are not considered. The algorithm of Random-selection indicates the nodes are selected at random. The CPU-based algorithm selects nodes by available CPU capacity. Due to selected nodes providing richer CPU resources, the completion time of each selected node is shortened. This means the task can be completed in the effective time, so the times of task re-distribution and task completion are fewer than the cases of FCFS, LIFO, and Random. Therefore, FCFS, LIFO, and Random selected nodes cannot complete the tasks in the effective time, which means that task re-distribution and re-execution will constantly occur. This will result in higher numbers of task re-distribution as shown in Figure 1 and longer task completion time as shown in Figure 2. The VF method proposed in this study, selects nodes by value. Due to the considerations of the various factors of the nodes, nodes with a better performance are chosen and the nodes with the longest effective time will be chosen by the past task completion rate. Consequently, the VF is better than other methods. This does not matter in the cases of VF-MAX or VF-AVG or VF-MIN, and it is better than other methods as shown in Figure 1 and Figure 2.
5.1.2 Condition 2

In the condition where demand for computing resource is large and the data transmission is large, the impact of the nodes-transmission-rate is more significant. Therefore, the impact of the nodes available CPU on task completion time is smaller. Figure 3 and Figure 4 show that when the FCFS, LIFO and Random algorithms select nodes for task assignment, the factors of nodes are not considered. Consequently, the experimental results are not much different from those derived in Condition 1. The CPU-based method only takes into account remaining usage of CPU, not the transmission rate of the node. As a result, when the amount of data transmission is large, a great deal of time is likely to be spent on data transmission. This makes the node unable to complete the assigned task in effective time. Transmission rate and past task completion rate of nodes are considered in the VF. Nodes that provide a better transmission rate and computing resources will be continuously selected. Even if the VF-MAX is required to have the fewest times of task re-distribution as shown in Figure 3 and shortest task completion time than other methods as shown in Figure 4.

5.1.3 Condition 3

In the condition where demand for computing resource and data transmission are large, the available CPU capacity and the data transmission rate become very important. Figure 7 and Figure 8 show that the FCFS, LIFO, and Random algorithms do not consider the node factors. Therefore, the task re-distribution time is more serious than in Conditions 1 and 2. The CPU-based method considers the remaining CPU capacity and chooses nodes with better computing performances, so a shorter completion time is needed. This allows the assigned task to be completed in the effective time. As a result, task re-distribution times are fewer and the task completion time is shorter, than in the cases of the FCFS, LIFO, and Random methods. Still, since the CPU-based method does not take into account the available size of memory, the transmission rate and the past task completion rate; then insufficient memory or interruptions are likely to occur during task execution. Therefore, large amounts of transmission time will be required. Consequently, it will be impossible to complete the task in an effective time and result in task re-distribution and re-execution. With the VF selecting necessary nodes by the properties of the task to be processed, due to the VF-MAX the times of task re-distribution and completion time are still fewer than other methods. The comparable methods, such as CPU-based, FCFS, LIFO, and Random are shown in Figure 5 and Figure 6.

5.1.4 Condition 4

When the computing resources and data transmission demand is small, the resources provided by the node, such as available CPU and memory capacity and the
data transmission rate do not have a significant impact. As shown in Figure 7 and Figure 8, the CPU-based method only considers the available CPU capacity and the derived result is similar to the FCFS, LIFO, and Random method results. The experimental results in Figure 7 and Figure 8 show that these three methods result in similar task re-distribution and completion times. The CPU-based result is shorter than those from the FCFS, LIFO, and Random methods, because the CPU-based method finds the node with the largest resources. The VF considers the various factors of the nodes and selects the nodes with better performance and longer provision time. Therefore, the nodes selected within any weight set only need a few times of re-distribution to complete the task and the completion time is shorter than other methods.

5.2 Experiment 2

This focuses on the simulation of the nodes with different performances in four different conditions, with the analysis and discussion of experimental results.

5.2.1 Condition 1

When demand for computing resource is large and amount of data transmission is small, the available CPU capacity and memory provided by the node will affect task completion. Therefore, the node-transmission-rate impact on the task completion time is not significant. The experimental results are shown in Figure 9 and Figure 10. These indicate that the nodes with the largest CPU resource are first selected, disregarding the execution order for subtasks in MCT. Min-min, in contrast, first selects the combination with the node and subtask that consume the shortest time making the required task completion time shorter than MCT. However, Min-min does not consider the memory provided by the node. This can cause subtasks to be re-distributed under insufficient memory. In which the VF considers not only the CPU capacity, but also the memory, the transmission rate and the past task completion rate of the node. Even if the task completion time with the VF-MAX is close to Min-min, the VF still consumes a shorter completion time than Min-min as shown in Figure 10 and times of task re-distribution are fewer than Min-min as shown in Figure 9. Therefore, the VF of this is more effective in maintaining the system load balance.

5.2.2 Condition 2

In the condition where demand for computing resource is large and the data transmission is large, the impact of the nodes transmission rate is more significant. The experimental results are shown in Figures 11 and 12. MCT and Min-min primarily consider the CPU capacity of a node and disregard the data transmission rate. When the data transmission is large, a large amount of time may be spent on transferring data. Which makes a node unable to complete the assigned subtask in the effective time and
the subtask has to be re-distributed and re-executed. As the VF considers these factors, under various weighted sets, fewer times of task redistribution as shown in Figure 11 and a shorter time of task completion as shown in Figure 12 are required.

5.2.3 Condition 3

In the condition where demand for computing resource and data transmission are large, the available CPU and data transmission rate become very import. In Figures 13 and 14, it is shown that MCT does not consider the execution order. The largest subtask may not be able to find the node with the largest resource. Which means the proper nodes cannot be searched out and assigned to the subtasks. Min-min selects a node by the shortest completion time instead of the order of subtasks. Therefore, it requires a shorter task completed time than MCT. As the VF considers multiple factors, nodes that can provide stable resources will be first selected. Even if in the VF-MAX, task completion time is longer than Min-min as shown in Figure 14, but task re-distribution time is less as shown in Figure 13. Specifically, the nodes selected by the VF may not be able to produce the best results in all weighted sets, but they are still effective in maintaining system load balance.

5.2.4 Condition 4

When the demand for computing resources and data transmission is small, the resources provided by a node such as available CPU and transmission rate do not have a significant impact. In Figures 15 and 16, it is shown that MCT and Min-min only consider CPU capability, not the memory and transmission rate. As a result, in the process of task execution, subtasks may be re-distributed and a longer task completion time might occur. The subtask may not be completed in the effective time, due to the time spent on data transmission. Therefore, the task has to be re-assigned to another node and as a consequence consumes more time. The VF considers multiple factors, so a shorter completion time is required; this is different from most all the other methods as shown in Figure 16. Included from the VF, task re-distribution is mostly not necessary. The task can be completed in the effective time in nearly all cases.

In a Grid environment, the nodes are composed of idle resources from different geographic locations. Since each node has a different hardware structure, the nodes cannot be selected based on a single condition, such as available CPU capacity. For that reason, the executable properties of the task must be considered. Based on the above results, it is shown that when selecting nodes, if the properties of the task and the resources that nodes can provide are not considered, the task to be executed will be re-assigned and re-executed continuously. Therefore, prolonging the task completion time and lowering the execution performance of the system. The VF proposed in this study lists the node resources, transmission rates and past completion rates under
consideration. By estimating the node value with these factors, the nodes that provide a longer time and larger resource will be selected. The results of the four different conditions in the two experiments have proven that the VF task-scheduling method is far more effective than other methods in reducing times of task re-distribution and completion time. Including the enhancing of the execution performance of the system, whether in terms of the VF-MIN, the VF-MAX, or the VF-AVG,

Figure 1. Condition 1: Times of Task Re-Distribution in the ideal case

Figure 2. Condition 1: Task Completion Time in the ideal case

Figure 3. Condition 2: Times of Task Re-Distribution in the ideal case

Figure 4. Condition 2: Task Completion Time in the ideal case

Figure 5. Condition 3: Times of Task Re-Distribution in the ideal case

Figure 6. Condition 3: Task Completion Time in the ideal case
Figure 7. Condition 4: Times of Task Re-Distribution in the ideal case

Figure 8. Condition 4: Task Completion Time in the ideal case

Figure 9. Condition 1: Times of Task Re-Distribution in the general case

Figure 10. Condition 1: Task Completion Time in the general case

Figure 11. Condition 2: Times of Task Re-Distribution in the general case

Figure 12. Condition 2: Task Completion Time in the general case
6. Conclusion

The resources in a Grid environment are composed of idle resource on the network and these nodes supply resources, which vary with time. This means a node may be idle in one moment, but may be busy in the next and therefore ceases to provide the resource. A new node has to be sought out for the allocated task; as a result, task re-distribution and re-execution may occur. For this reason, when choosing nodes for task assignment, it is necessary to consider both the available capacities, such as CPU availability, that a node can supply and other factors of the node. This is desired to choose a suitable and useful node to carry out the work process. To maintain the system load balancing condition of a Grid environment, this study proposes a hybrid load balancing policy.

This policy method integrates static and dynamic load balancing techniques to locate effective nodes, identify system imbalance in the shortest time when any node becomes ineffective and fill in the imbalance with a new node. Through two different
cases and four conditions, simulation experiments were conducted to prove that the methods in this research study. In creating the VF in the VF-MIN or the VF-MAX or the VF-AVG, it is shown that this method is far more effective than other methods in selecting nodes with better effectiveness and performance for task execution, reducing task completion time and avoiding the occurrence of task re-distribution and re-execution.

Accordingly, the proposed method can effectively maintain the load balance and enhance the system performance. The proposed method has the main advantage of locating proper resources according to the task properties. It can reduce the decline of system performance, which results from the incorrect selection of ineffective nodes and maintain the load balancing of the system.

References
Folding@Home (2006), http://folding.stanford.edu/.
Genome@home (2006), http://genomewathome.stanford.edu/.
independent jobs in heterogeneous computing environments. In Proceedings of 22nd Workshop of the UK Planning and Scheduling Special Interest Group, Glasgow (pp. 178-183).


