Stage-Warping Load Sharing Strategy for Fine Grain Applications over Grid Environments

Natthakrit Sanguandikul and Natawut Nupairoj
Department of Computer Engineering,
Chulalongkorn University, Bangkok, 10110 Thailand
E-mail: Natthakrit.S@student.chula.ac.th

Abstract

A Load sharing strategy is one of important keys to improve the performance of computing systems. Nowadays, large scale computing system can be created by aggregating multiple computing clusters from different organizations using Grid technology. However, it is difficult to define a practical load sharing strategy due to the computing heterogeneity and dynamic behavior of Grid resources. In this work, we introduce a load sharing strategy for distributing workloads among participating clusters. The proposed strategy implements a new job-stealing technique called "stage-warping", for dynamically adjusting the amount of assigned workloads for each cluster during an execution. In our strategy, the entire workloads are divided into stages which enable the total control of workload assignment during an execution, while still being highly robust against performance fluctuation and information inaccuracy of the computing resources. During execution, faster-than-expected clusters which will finish the assigned workloads during each stage before other clusters will steal left-over workloads from other clusters and let them skip or warp to the foremost stage. This will make all clusters to be fully utilized by finishing their assigned workloads almost at the same time near the end of each stage, resulting in a better overall parallel performance of load sharing strategy. We evaluate our proposed strategy using a set of simulation experiments based on the parameters from the available computing resources in Thaigrid, as well as, fine-grain applications which can serve as an example of computationally intensive applications. The results show that our proposed strategy can achieve better parallel runtimes when being compared to other existing methods, especially when the estimator of the underlying system is not accurate.

Keywords: Distributed Computing, Grid Technology, Load Sharing Strategy

1. Introduction

The current trend of parallel processing is to create a low-cost supercomputer by aggregating multiple PCs together. For example, Grid computing focuses on aggregating the computing resources geographically distributed across different organizations [1]. Therefore, the number of computing nodes and the complexity of the underlying system will be dramatically increased. In addition, the

performance of Grid resources can change abruptly because they are not dedicated resources and are subject to the local resource policy. In order to utilize this vast amount of computing power effectively, we must employ a load sharing strategy in order to keep every computing resource busy until the end of an execution. Although many load sharing strategies have been proposed in the past [2-3], these strategies are aimed for single cluster computing environments only. Therefore, they fail to address new characteristics in multiple-cluster environments [4] such as communication structure between clusters, large overhead over WAN, high computing heterogeneity, and the dynamic behavior of non-dedicated resources. Moreover, most load sharing strategies proposed for large scale computing environments [5-6] still require some specific performance indicators of computing resources for making load decisions. These indicators are difficult to derive as there are various parameters which can affect the overall runtime such as processor speed, system architecture, communication bandwidth, and submitted applications. Thus, it is very difficult to collect all necessary internal information within each cluster and define a sophisticated resource model that can truly predict the performance of the computing system [7-8]. Given the growing complexity in computing environments and applications, the traditional methodologies are no longer practical.

To address this foreseeable problem, we propose a load sharing strategy with new job-stealing technique for distributing workloads among participating clusters. Our strategy first divides total workloads into different stage sizes before further assigning workloads allocated within each stage to the participating clusters according to their performance indicators. By implementing our new job-stealing technique called "stage-warping", faster-than-expected clusters can execute additional tasks by stealing workloads to be executed by other clusters. Hence, those clusters can catch up with the faster-than-expected clusters by skipping or warping to the same current stage. This behavior will give additional robustness against the performance fluctuation of highly heterogeneous resources in Grid environments. In addition, our performance indicator is just a rough performance estimator of the underlying system, which a global coordinator can calculate during an execution. Therefore, there is no need to implement any monitoring services to gather information from the computing clusters at all. To increase the accuracy of this performance indicator, our strategy consists of increasing stages and decreasing stages. The increasing stages are for obtaining accurate performance indicators of the underlying systems and the decreasing stages are for reducing load imbalance near the end of an execution.

Although the proposed strategy in [9] makes load decision based on the performance indicators obtained during an execution like our proposed strategy, it uses job replication to address dynamic behavior of Grid resources, which is more suitable for coarse grain applications. Moreover, given a fine grain computing application, our proposed strategy can balance the workloads between each cluster dynamically which eliminates load imbalance near the end of an execution. Therefore, we do not have to implement job replication, and we can also save some computing cycles by doing so. The organization of this paper is as follows. Section 2 provides the details about the models within our work. Section 3 describes other related works. In section 4, we propose our stage-warping load sharing strategy. The performance evaluations are shown in section 5. And finally, we conclude our work in section 6.

2. Models

2.1 Grid Model

We assume that the computing system is a multiple-cluster computing environment. It consists of N computing nodes which will be grouped together into L clusters $\{C_1, C_2 \ldots C_L\}$. These computing clusters communicate with each other over WAN while the intra-cluster communication will be made over LAN. The computing nodes within the same cluster will be considered homogeneous and have the same computing power. With this assumption, we can increase the computing heterogeneity within our system by specifying some clusters to have more total computing power than the others. Within each cluster, there is one local gateway which is responsible for distributing workloads submitted by Grid users to other computing nodes in the same cluster and also handles the inter-cluster communications. In addition, one of the local gateways will also serve as the global coordinator which manages submitted jobs and assigns workloads to the other clusters. Global strategy will be used for assigning workloads among participating clusters while local strategy is for assigning workloads within each cluster. Note that the local strategy can be specified differently depending on the local administrator of that cluster.

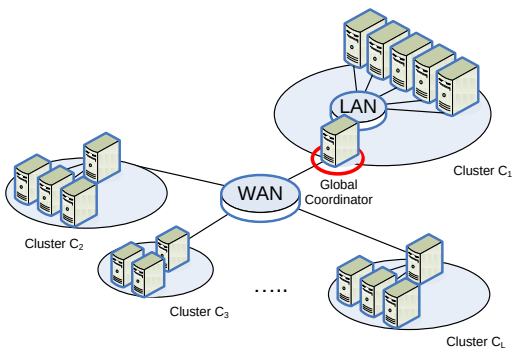

Figure 1 The system model of Grid environment.

2.2 Application Model

We define an application model based on fine-grain computationally intensive application which consists of U unit tasks where the computation and communication size of each task can be specified differently. This model represents parameter sweeping applications which are considered as the killer applications for Grid technology [10]. The examples of real life applications that belong to this type are radiation equipment calibration analysis, searching for extra-terrestrial intelligence, protein folding, molecular modeling for drug design, human-genome sequence analysis, brain activity analysis, high-energy physics event analysis, ad hoc network simulation, crash simulation, tomography, financial modeling, and M-cell simulations. Moreover, we also further divide the fine-grain applications into four distinct classes of applications, including uniform, increasing, decreasing, and random distribution [11]. These classes can represent popular applications where the computation size of each unit task can be different during an execution such as Matrix Multiplication, SOR, Reverse Adjoint Convolution, LU Decomposition, and Gauss Jordan Elimination.

3. Related Works

The Self-Scheduling strategy (SS) [12] dynamically assigns only one unit task per each request for an idle computing resource. With this behavior, it can achieve almost perfect load balancing because every computing resource will finish within one task of each other. However, this strategy also suffers from high communication overhead. To address this problem while keeping its simplicity, many variations of SS have been proposed [13-14]. One of them, which is famous for robustness is called "Factoring" (FSS) [15]. This strategy assigns workloads into multiple stages. In the first stage, FSS distributes the largest

chunk and decreases the chunk size in the subsequence stages proportionally. During each stage, every processor will receive an equal chunk size of workloads. FSS can reduce communication overhead by sending large chunks at the beginning while it achieves sub-optimal runtime by sending small chunks near the end of computation. To address heterogeneity within the computing system, "Weighted Factoring" (WFSS) [16] is proposed as an extension of FSS. In this strategy, the amount of total unit tasks allocated during each stage is the same as in FSS. However, unlike FSS, WFSS utilizes pre-execution information of the computing resources as weighted values to assign workloads allocated within each stage. We will call a strategy that utilizes pre-execution information an "explicit strategy".

One of the major weaknesses of explicit strategies is that these strategies rely on static knowledge. Thus, they perform quite poor in a dynamic environment like Grid computing. One of the descendants of FSS called "Adaptive Weighted Factoring" (AWF) [17], addresses this problem by further extending WFSS with an adaptive weighted value called "Weighted Average Performance" (WAP). This weighted value will be re-calculated every stage using the newly obtained computing rates of each resource. Therefore, the explicit information will be used as a weighed value during the first stage only. With this average value, AWF can address the dynamic behavior of the heterogeneous computing system. However, since AWF assigns half of the available workloads during the first stage, the problem of inaccurate explicit information can still affect the performance of this strategy.

4. Proposed Load Sharing Strategy

In this work, we propose a new strategy called "stage-warping load sharing strategy" (SWS). Our proposed strategy is aimed to address the computing heterogeneity and dynamic behavior of Grid resources by dynamically adjusting workload assignment based on the estimation of the computing power of the participating clusters. However, the estimation can be inaccurate since the performance of Grid resources can change dramatically. To address this problem, our proposed strategy requires a job-stealing technique in order to minimize the inefficiency from inaccurate estimation. To define a practical job-stealing technique, we divide the entire workloads into different stages. The workloads in each stage will be further divided for the participating clusters according to their computing power. In an ideal case, all computing clusters are supposed to finish their assigned workloads nearly at the same time during the end of each stage. This will fully utilize all computing resources and leads to the minimal execution time. In reality, however, some clusters will finish their assigned workloads before other clusters and request more workloads by entering the next stage immediately. This behavior will result in differences of stage numbers of these clusters which will make clusters complete their execution at different times. Thus, this will lead to underutilization of the computing resources and eventually decrease overall parallel performance. To address this problem, we introduce the stage-warping technique that allows faster-than-expected clusters to execute additional tasks by stealing workloads to be executed by other clusters. Slower-than-expected clusters can then catch up with the other clusters by skipping or warping to the same current stage. With this approach, the completion time of each cluster can be controlled to be finished almost at the same time near the end of each stage despite the dynamic behavior of the underlying resources. We can see this behavior as another form of job-stealing technique since

the faster-than-expected clusters steal workloads which are supposed to be executed by other clusters.

Our proposed strategy uses rough performance estimation, called "consuming rate" (*cr*), which is calculated from the amount of assigned workloads and the interval time between requests. Thus, overall execution aspects including the relationship between the submitted application and the underlying system, in term of performance, can be evaluated through this performance indicator during the execution. Therefore, the problem of misleading parameters like assigning I/O intensive applications to the computing cluster with fast processing speed can be avoided. Moreover, this indicator can be calculated at the coordinator node that is responsible for assigning workloads. Hence, our strategy can make load decisions without implementing the monitoring service within the participating clusters.

4.1 Stage Size Assignment

In our proposed strategy, application workloads are divided into stages. The amount of workloads assigned to the participating clusters during each stage will be further adjusted with respect to the performance of the requesting cluster and the performance of other clusters. Moreover, we also divide entire computing stages into two groups, which are the increasing stages and the decreasing stages. The purpose of the increasing stages is to obtain accurate performance indicators before entering the decreasing stages. Although we can obtain accurate consuming rate from the beginning of an execution by sending a large chunk of workloads, it is also too risky to assign workloads without having an accurate estimator of the requesting cluster first. To address this problem, our strategy sends small workloads during the first stage and increases the stage size exponentially. After that, the remaining half of the total workloads will be used to balance the workloads between each cluster. The stage number of every cluster starts from a negative value specified as $-\left\lfloor \log_2\left(\frac{U}{2}\right)\right\rfloor$. Then the stage number of each cluster will be gradually increased by one value every time that cluster requests more workloads until it becomes zero, which is the last stage of the increasing stages. After that, the cluster will enter the decreasing stages, and its stage number will still be increased by one value for every request until the end of an execution. An example of stage number sequence from one participating cluster during the entire execution can be given as {-3,-2,-1,0,1,2,3,4}. Equation (1) describes how we allocate workloads during both increasing and decreasing stage m (u_m) given the total number of tasks (U) and constant ratio (δ) which we specify as 2 throughout this work. An example of how our strategy allocates workload for each stage is shown in Fig. 2.

$$u_m = \begin{cases} \left\lceil \frac{U}{2 * \delta^{|m|+1}} \right\rceil & if\ m \le 0 \\ \left\lceil \frac{U}{2 * \delta^{m}} \right\rceil & otherwise \end{cases} \quad (1)$$

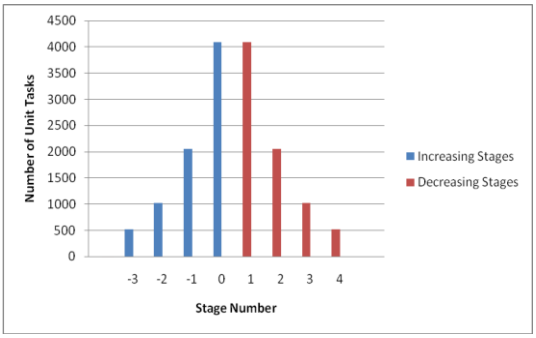

Figure 2 The number of unit tasks allocated for each stage.

Given the stage size of both increasing and decreasing stages, the global coordinator can determine $k_{i,m}$, which is the chunk size to be assigned for cluster C_i

during stage *m*. This value will be calculated from the performance indicator of the requesting cluster ($cr_{i,m}$), compared with those of the other clusters ,together with the predefined stage size (u_m) as shown in Eq. (2). Note that the workloads allocated in the first stage will be assigned equally to all clusters because there is no consuming rate available yet.

$$k_{i,m} = \frac{cr_{i,m}}{\sum_{q=1}^{L} cr_{q,m-1}} * u_m \qquad (2)$$

Keeping all clusters to execute at the same stage for the entire execution period is very important as it prevents a load imbalance, which can lead to poor performance. In an ideal case where the estimator of each cluster is always accurate, tasks will be assigned to all clusters, such that all clusters will enter the same stage, complete all tasks in the predefined stage chunk, and then move to the next stage at the same time throughout the entire execution. In reality, this behavior will never happen. Given the inaccuracy, some clusters will be over-estimated (or slower than expected) while the others will be under-estimated (or faster than expected). Under-estimated clusters will enter the following stage and continue requesting workloads for the new stage while over-estimated clusters are still in the previous stage. This behavior will create additional load imbalance in the system especially when there are some clusters still in the increasing stages at the end of an execution. Moreover, this problem will become worse in a multiple-cluster environment as the underlying resources can be highly heterogeneous.

4.2 Stage-Warping Technique for Grid Environment

In order to keep every cluster to be at the same stage throughout an execution, our strategy introduces a job-stealing technique called "stage-warping". This technique allows the clusters whose stage numbers are behind the others to skip (or warp) from their current stages to the foremost stage. The newly derived stage will include the remaining workloads in the previous stages into the foremost stage. With this behavior, the leftover workloads in the previous stages will be re-assigned again to every cluster and the effect of inaccurate estimators, which causes some clusters to stay behind, can be reduced. In other words, the faster-than-expected clusters absorb the load imbalance effects by stealing workloads from those clusters that warp to catch up to the foremost stage. Equation (3) illustrates how we can calculate the new stage size (u'_m) with stage-warping technique, by first finding the number of assigned tasks during the previous stages from the remaining tasks at the beginning of the current stage *m* (ω'_m), before combining the leftover tasks with the pre-allocated tasks. An example of this behavior can be seen in Fig. 3 where the tasks inside stage -1 and 0 are moved into stage 1 with our stage-warping technique.

$$u'_m = \begin{cases} 2\left\lceil \frac{U}{2*\delta^{|m|+1}} \right\rceil + \omega'_m - U & \text{if } m < 1 \\ \left\lceil \frac{U}{2*\delta^{m}} \right\rceil + \left(\frac{U}{2} + \omega'_m - U \right) & \text{if } m = 1 \\ \left\lceil \frac{U}{2*\delta^{m}} \right\rceil + \left(\frac{U}{2} + \sum_{q=1}^{m-1} \left\lceil \frac{U}{2*\delta^{q}} \right\rceil + \omega'_m - U \right) & \text{if } m > 1 \end{cases} \qquad (3)$$

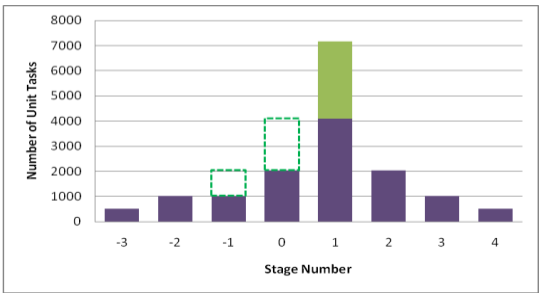

Figure 3 Behavior of stage-warping technique.

5. Performance Evaluation

In this section, we will evaluate the performance of load sharing strategies by simulating a real computing environment with NS2 [18]. We choose two computing clusters in Thaigrid[19], TERA and PLUTO, as representatives of computing clusters. TERA cluster belongs to Kasetsart University while PLUTO cluster is belonged to Chulalongkorn University. We collect the computing power of each cluster by using a simple Matrix-Multiplication program with different matrix sizes. The communication-related parameters are defined based on our preliminary tests and the available specifications. The parameters that we use to create our test environment from the real environment are shown in Table 1. Note that the unit time represents the computation time for one computing node in each cluster to execute only one multiplication. Hence, the entire execution will consist of M^3 multiplications where M is the matrix size during each run.

Figure 4 and 5 illustrate the accuracy of our simulation by comparing the parallel runtimes from both real and simulated single cluster environment using SS as load sharing strategy with matrix size specified as 2000 x 2000.

Table 1 The parameters from real environment

Variables	Values
Unit time (TERA)	20.834ns
Unit time (PLUTO)	29.223ns
LAN Latency	30μs
LAN Bandwidth	1000Mbps
WAN Latency	25ms
WAN Bandwidth	1.5Mbps

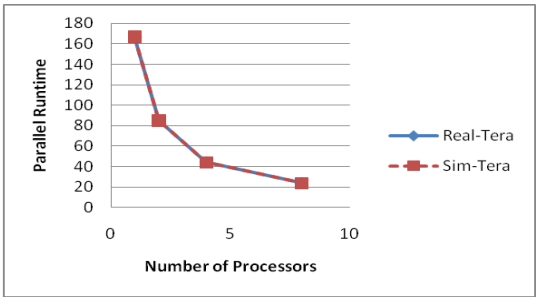

Figure 4 The comparison of parallel runtime over TERA.

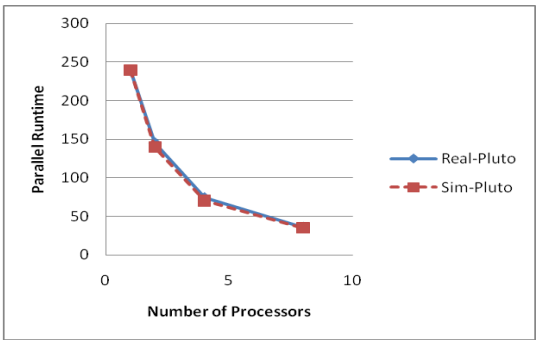

Figure 5 The comparison of parallel runtime over PLUTO.

To simulate heterogeneous computing environment, we create four clusters where one of them consists of 64 nodes while the other clusters will consist of only 16 nodes. The first cluster will use the computing power collected from TERA cluster and the rest will be from PLUTO cluster. Since this work focuses on the global strategy which assigns workloads among participating clusters, the local strategy within each cluster is defined based on SS. As for the submitted application, the matrix size for the simulated experiments will be specified as 10000 x 10000 and the communication size of each unit task is defined based on data within the Matrix Multiplication program, which is in floating point. Therefore, our submitted application can be considered as a computing-intensive application where the communication overhead will not affect overall performance of load sharing strategies, except SS, which assigns only single task per request for obtaining better performance stability.

5.1 Effect of the Computing Heterogeneity

First, we evaluate the effect of computing heterogeneity within an underlying system by comparing the parallel runtimes of load sharing strategies over both homogeneous and heterogeneous systems. We create a homogeneous system within our simulation by assigning every cluster to have the same number of computing nodes with the same computing power. Note that the homogeneous system will have the same total computing power as in the heterogeneous system. We choose SS to represent the simplest form of load sharing strategy that can assign workload without using any performance indicators about an underlying system. WFSS can be considered as an example of explicit strategy that uses only the information obtained from an external source, where AWF will also adjust to the dynamic of the computing system by re-calculating the weighted value throughout an execution. Figure 6 shows the performance of load sharing strategies over computing systems with different computing heterogeneities.

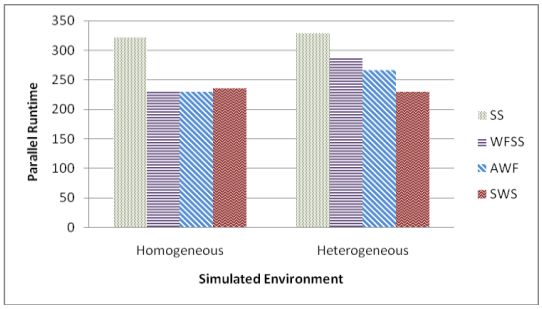

Figure 6 Parallel runtimes over different computing systems.

Experiment results show that SS has the worst results on both homogeneous and heterogeneous system because of an excessive communication overhead for each request. As for WFSS and AWF, we can see that AWF performs better than WFSS in both cases. This behavior is the result of how AWF can adjust the performance indicator of each computing resource during an execution. This is the reason why AWF is considered to be one of the best load sharing strategies for grid computing systems. However, while SWS achieves a comparable parallel performance with AWF over a homogeneous system, its parallel performance will be a lot better than AWF over a heterogeneous system. The reason behind this behavior is how our strategy can reduce the load imbalance between the fast and slow clusters better than AWF by allowing under-estimated clusters to steal workload of other over-estimated clusters. Therefore, our strategy can avoid a bad situation when there are some tasks left over in the slow clusters, while other clusters idle near the end of an execution.

5.2 Effect of the Dynamic Behavior of the Underlying Resources

Since Grid technology allows local administrators to have control over their resources, the available computing power of Grid resources can change abruptly throughout an execution because the local policy allows other jobs to be executed on the same computing nodes. To simulate this behavior, we specify the computing power of each computing node with normal distribution where the max-min boundaries are specified as a percentage of the average value as mentioned in [20].

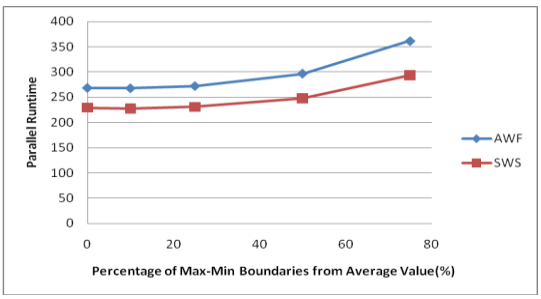

Figure 7 Parallel runtimes over different fluctuation in available computing power.

We can see from the results in Fig. 7 that while the parallel performance of both strategies decrease when the

fluctuation of the computing power increases, our proposed strategy can achieve better parallel runtime than AWF. This behavior shows that our strategy can tolerate a fluctuation of computing power more than AWF because we define our strategy so that it can address the changes in computing system throughout an entire execution with our stage-warping technique.

5.3 Effect of Inaccurate Information

Our strategy can obtain the available computing power by itself. Thus, it does not need an estimated value of the underlying system prior to execution. This behavior not only eliminates the need for implementing the monitoring service over participating clusters, but also makes our strategy independent from information inaccuracy of each resource when we start the execution. On the contrary, most load sharing strategies, such as AWF, rely on the estimated performance of the underlying systems. Thus, their performances are subjected to the accuracy of the estimators. To illustrate this effect, we define a random variable which represents the estimated computing power used by AWF during the first stage. The value of this variable will be selected from a normal distribution, where different max-min boundaries are specified as a percentage of the real computing power. The effect of inaccurate information over the performance of load sharing strategies is shown in Fig. 8.

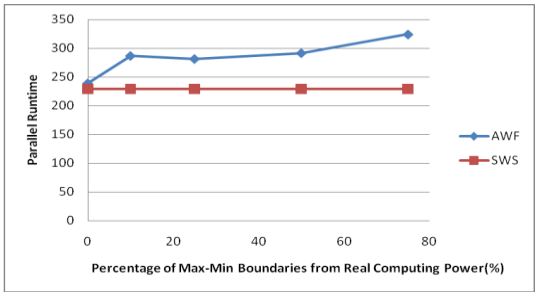

Figure 8 Parallel runtimes over different fluctuation with information inaccuracy.

The experiment results show that our strategy is not affected by inaccurate information like AWF. This behavior is the results of how we define our strategy to collect and adjust the estimated computing power of an underlying resource by itself without using an estimated value from other sources. We can see that the performance of AWF will decrease greatly when the estimated information is not accurate.

5.4 Effect of Application Classes

Different types of applications can also affect the performance of load sharing strategy. The computation size of each task can be uniform, random, increasing, or decreasing throughout the entire execution.

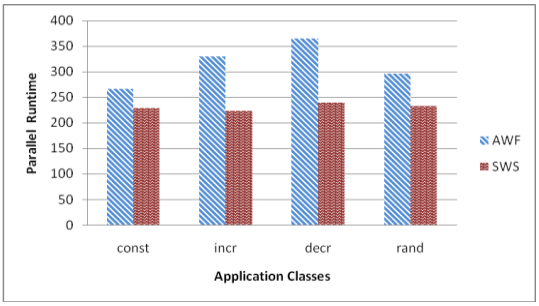

Figure 9 Parallel runtimes over different application classes.

Figure 9 shows that our strategy can achieve better parallel runtimes than AWF on every application, especially when the computation size of each task decreases during an execution. This is due to the fact that for applications with decreasing computation size, the correct task assignment is very crucial for the first stage, as it includes larger tasks, which contribute the majority of the execution time. Thus, this will greatly impact the system using AWF to distribute workloads, as it assigns half of the entire workloads during the first stage.

6. Conclusion

In this work, we propose a robust load sharing strategy for large-scale cluster-based computing systems. By implementing a stage-warping technique, our strategy can make load decisions based on a rough performance estimator which can be collected at the global coordinator during an execution, and steal jobs from slower-than-expected clusters to prevent a load imbalance near the end of the execution. Hence, there is no need to implement a monitoring service or worry about its accuracy at all. From our experiments, we show that our proposed strategy can address computing heterogeneity, fluctuation in available computing power, and different classes of submitted applications. Therefore, our proposed global strategy is simple yet efficient in a large scale computing environment which definitely will serve as the computing platform for the next generation.

7. Reference

[1] Foster, I. et al., The Anatomy of the Grid, Int. J. Supercomput. Appl. High Perform. Comput., Vol. 15, pp. 200-222, 2001.

[2] Kruskal, C. P. and Weiss, A., Allocating Independent Subtasks on Parallel Processors, IEEE Trans. Software Eng., Vol. 11, pp. 1001-1016, 1985.

[3] Balasubramanian, J. et al., Evaluating the Performance of Middleware Load Balancing Strategies, EDOC, pp. 135-146, 2004.

[4] Shen, K. et al., Cluster Load Balancing for Fine-grain Network Services, IPDPS, pp. 51-58, 2002.

[5] Chronopoulos, A. T. et al., Scalable Loop Self-scheduling Schemes for Heterogeneous Clusters, Int. J. Comput. Eng. Sci., pp. 353-359, 2002.

[6] Shih, W. C. et al., A Performance-based Parallel Loop Self-scheduling on Grid Environments, Lect. Notes Comput. Sci., Vol. 3779, pp. 48-55, 2005.

[7] Biswas, R. et al., Tools and Techniques for Measuring and Improving Grid Performance, Lect. Notes Comput. Sci., Vol. 2571, pp. 45-54, 2002.

[8] Wolski, R., Experiences with Predicting Resource Performance On-line in Computational Grid Settings, Perform. Eval. Rev., Vol. 30, pp. 41-49, 2003.

[9] Lee, Y. C. and Zomaya, A. Y., Practical Scheduling of Bag-of-Tasks Applications on Grids with Dynamic Resilience, IEEE Trans. Comput., Vol. 56, pp. 815-825, 2007.

[10] Buyya, R. et al., Scheduling Parameter Sweep Applications on Global Grids: a Deadline and Budget Constrained Cost-time Optimization Algorithm, Software Pract. Ex., Vol. 35, pp. 491-512, 2005.

[11] Fann, Y. W. et al., IPLS: An Intelligent Parallel Loop Scheduling for Multiprocessor Systems, ICPADS, pp. 775-782, 1998.

[12] Tang, P. and Yew, P. C., Processor Self-scheduling for Multiple-nested Parallel Loops, ICPP, pp. 528-535, 1986.

[13] Polychronopoulos, C. D. and Kuck, D. J., Guided Self-scheduling: A Practical Scheduling Scheme for Parallel Supercomputers, IEEE Trans. Comput., Vol. 36, pp. 1425-1439, 1987.

[14] Tzen, T. H. and Ni, L. M., Trapezoid Self-scheduling: A Practical Scheduling Scheme for Parallel Compilers, TPDS, Vol. 4, pp. 87-98, 1993.

[15] Hummel, S. F. et al., Factoring: A Method for Scheduling Parallel Loops, Comm. ACM, Vol. 35, 1992.

[16] Hummel, S. F. et al., Load-sharing in Heterogeneous Systems via Weighted factoring, SPAA, pp. 318-328, 1996.

[17] Banicescu, I. and Velusamy, V., Performance of Scheduling Scientific Applications with Adaptive Weighted Factoring, IPDPS, pp. 84, 2001.

[18] McCanne, S. and Floyd, S., VINT Network Simulator - ns (version 2), http://www.mash.CS.Berkeley.EDU/ns/.

[19] Varavithya, V. and Uthayopas, P., ThaiGrid: Architecture and Overview, NECTEC Tech. J., Vol. 2, 2000.

[20] Casanova, H., Simgrid: A Toolkit for the Simulation of Application Scheduling, CCGrid, pp. 430-441, 2001.