Maejo Int. J. Sci. Technol. **2011**, *5*(02), 215-230

Maejo International Journal of Science and Technology

ISSN 1905-7873
Available online at www.mijst.mju.ac.th

Full Paper

Agentless robust load sharing strategy for utilising heterogeneous resources over wide area network

Natthakrit Sanguandikul* and Natawut Nupairoj

Department of Computer Engineering, Chulalongkorn University, Bangkok 10110, Thailand

* Corresponding author, e-mail: ramza_th@hotmail.com

Received: 2 November 2010 / Accepted: 20 June 2011 / Published: 22 June 2011

Abstract: Resource monitoring and performance prediction services have always been regarded as important keys to improving the performance of load sharing strategy. However, the traditional methodologies usually require specific performance information, which can only be collected by installing proprietary agents on all participating resources. This requirement of implementing a single unified monitoring service may not be feasible because of the differences in the underlying systems and organisation policies. To address this problem, we define a new load sharing strategy which bases the load decision on a simple performance estimation that can be measured easily at the coordinator node. Our proposed strategy relies on a stage-based dynamic task allocation to handle the imprecision of our performance estimation and to correct load distribution on-the-fly. The simulation results showed that the performance of our strategy is comparable or better than traditional strategies, especially when the performance information from the monitoring service is not accurate.

Keywords: load sharing strategies, self-scheduling strategies, high performance computing, distributed systems, heterogeneous systems, load balancing, task assignment

INTRODUCTION

For the past decades, there has been an emergence of the technologies for utilising a large number of computing resources over wide area network such as grid [1] and cloud computing [2]. A low-cost, high-performance system for computing-intensive applications [3] can be built by aggregating multiple computing clusters which may consist of either real physical resources or virtualised resources from external providers. Hence, the number of computing nodes and the complexity of the underlying system have been dramatically increased. In order to efficiently utilise

Maejo Int. J. Sci. Technol. **2011**, *5*(02), 215-230

the available computing power, several resource monitoring frameworks and performance prediction methodologies have been proposed [4-7]. Although accurate performance prediction can greatly improve overall resource utilisation, these methodologies usually require a proprietary monitoring service to be implemented on all participating resources for collecting specific performance information. Hence, this requirement of implementing proprietary monitoring service might prevent the utilisation of cheaper or better computing resources due to the differences in their implemented systems or policies. In addition, predicting the execution time of a fine-grained task can be difficult on non-dedicated computing resources [8]. These limitations will hinder the possible applications in the upcoming computing technology such as those described in many-task computing (MTC) [9] where each job can consist of a large number of tasks which can be executed within a small computational time.

To address the problems in the traditional work, we propose a new load sharing strategy called agentless robust self-scheduling strategy (ARSS). As its name implies, our strategy can be used to assign the workload without the necessity of implementing any additional monitoring service in the computing resources while still being able to address the dynamic behaviour in the computing system. ARSS is based on self-scheduling strategies [10-11] and makes load decision according to performance metrics estimated at the coordinator node. The metrics used in our strategy are simple ones that represent how fast each computing resource can process the submitted workloads. Since these metrics can be obtained quickly and easily at the coordinator node which is responsible for assigning the workload, ARSS can use these estimations to make the load decision across different computing systems without any need to implement monitoring services in the participating resources. To compensate for the imprecision of these rough metrics, ARSS performs a dynamic task allocation to adjust the load distribution on-the-fly. The dynamic allocation is stage-based, consisting of both increasing and decreasing stages. The increasing stages are for improving the performance estimation accuracy while minimising the run-time by overlapping between computation and communication overheads over the wide area network (WAN). The decreasing stages are for smoothing the load imbalance near the end of the execution due to an inaccurate performance prediction. In addition, ARSS also includes an on-the-fly load distribution correction mechanism which performs the job-stealing on the leftover workload between stages in order to further minimise the effects of abrupt changes in computing power and of the performance estimation inaccuracy. Using this mechanism, ARSS does not require any agents installed on the participating resources while being very robust against the changes in the available computing power of the underlying system.

RELATED BACKGROUND AND ASSUMPTIONS

In this section, we will describe the multi-organisational computing environment including the application model and other related work in the past.

Maejo Int. J. Sci. Technol. **2011**, *5*(02), 215-230

Multi-Organisational Computing Environment

Throughout this work, we assume that a computing system consists of several heterogeneous computing resources from different organisations. Let us assume that there are a total of N computing nodes aggregated from different M groups or clusters $\{C_1, C_2 \ldots C_M\}$. These computing clusters communicate with each other over WAN with a propagation delay and bandwidth specified as α_W and β_W respectively. The propagation delay and bandwidth within each cluster are defined to be α_L and β_L respectively. The computing heterogeneity within our model will come from differences in the computing power between participating clusters while the computing nodes within the same cluster are homogeneous as illustrated by Chau and Fu [12]. Each cluster is assumed to have one local gateway which is responsible for distributing workloads submitted by users to the computing nodes within its own cluster based on local workload assignment strategy and which also handles the inter-cluster communications. In addition, one of the local gateways will also serve as the coordinator node which manages submitted jobs and assigns workloads to other clusters based on the global workload assignment strategy. Note that the local strategy can be varied depending on the owner of that particular cluster. Figure 1 illustrates an example of the multi-organisational computing system which involves multiple clusters from different organisations.

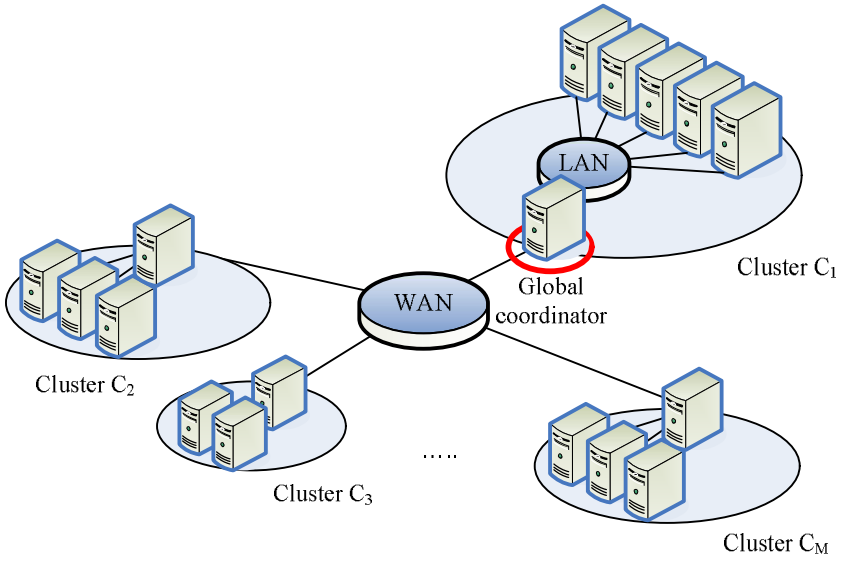


Figure 1. Multi-organisational computing environment

Application Model

We define our application model to represent computing intensive applications including those from many-task computing. This class of application contains a large number of small tasks which can be from thousands to billions. This common pattern can be found in many scientific applications from a wide range of domains such as astronomy, physics, astrophysics, pharmaceuticals, bioinformatics, biometrics, neuroscience, medical imaging, chemistry, climate modelling, economics and data analytics [13]. This amount of tasks will cause the performance

Maejo Int. J. Sci. Technol. **2011**, *5*(02), 215-230

degradation to time-consuming decision-making strategies due to high queuing and dispatching overheads. Since the expected run-time of small tasks is difficult to predict, the problem of inaccurate prediction will also affect the overall performance. Although we can group small tasks together to address this problem, this methodology will make it even harder to make the load decision given heterogeneous computing resources.

Our work focuses on fine-grained computationally intensive applications where the data sets are not large or have been prepared beforehand. Each task within a submitted application can be either independent or dependent on other tasks. If there are dependency between tasks, we assume each task can retrieve necessary information from the result files created by previously executed tasks similar to loosely-coupled applications defined by Zhang et al [14]. Hence, we define an application model consisting of U unit tasks whose computation and communication sizes vary. The distribution of computation sizes of unit tasks can be grouped into four distinct classes, viz. uniform, increasing, decreasing, and random distributions. These classes can represent popular applications, e.g. Matrix Multiplication, Successive Over-Relaxation, Reverse Adjoint Convolution, LU Decomposition and Gauss Jordan Elimination [15].

Related Work

The self-scheduling strategy (SS) has been famous for its simplicity for making load decision during the execution. This strategy dynamically assigns only one unit task for each request for an idle computing resource. With this behaviour, it can achieve an almost perfect load balancing because every computing resource will finish within one task of each other. However, this strategy suffers from high communication overheads. To address this problem while maintaining simplicity, many variations of SS have been proposed [16-19]. One of them, called 'factoring self-scheduling' (FSS), is famous for its robustness [20]. This strategy assigns a workload into multiple stages. In the first stage, FSS distributes the largest chunk and proportionally decreases the chunk size in the subsequent stages. During each stage, every processor will receive an equal chunk size of workload. The FSS can reduce communication overheads by sending large chunks at the beginning while achieving sub-optimal run-time by sending small chunks near the end of computation.

To further address heterogeneity within the computing system, 'weighted factoring self-scheduling" (WFSS) [21] was 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 utilises pre-execution information of the computing resources as weighted values to assign workloads allocated within each stage. One of the major weaknesses of this strategy is that the load decision is made based on static information. Thus, WFSS performs quite poorly in the dynamic multi-organisational computing environment.

One of the descendants of FSS, called 'adaptive weighted factoring' (AWF) [22], addresses this problem by extending WFSS with an adaptive weighted value called 'weighted average performance' (WAP). This weighted value is re-calculated at every stage using the newly obtained computing rates of each resource. Therefore, the pre-execution information will be used as a weighted value during the first stage only. With this average value, AWF can address the dynamic behaviour of the heterogeneous computing system. However, since AWF assigns half of the available

Maejo Int. J. Sci. Technol. **2011**, *5*(02), 215-230

workloads during the first stage, the problem of inaccurate pre-execution information can still affect the performance of this strategy.

In addition to SS, our proposed strategy is also based on the concept of increasing and decreasing stages. Utilising increasing and decreasing stages has been introduced in one of the traditional strategies [23]. However, that strategy requires specific information about the underlying system for determining the appropriate chunk sizes during the increasing and decreasing stages. Hence, it needs the monitoring service to be installed on the participating resources and its performance can also be highly affected by information inaccuracy. Although the load sharing strategy that focuses on practical usage [24] addresses inaccurate performance information by assigning the N smallest tasks to each node in order to compare their real performance, this behaviour requires the knowledge about the computation size of each task for creating the performance ranking of the computing resources.

Description of ARSS

Our proposed strategy aims for two important goals. First, the strategy must be non-intrusive such that it can assign the workload to participating resources using performance estimation at the coordinator node without relying on any monitoring services at the computing nodes. Second, the strategy must be robust enough against information inaccuracy due to the performance estimation and dynamic behaviour of the underlying system. In order to achieve these goals, our proposed strategy divides the entire unit tasks and assigns them in multiple stages. At each stage, the allocated unit tasks for that particular stage will be further divided into chunks and assigned to participating clusters with respect to their performance. ARSS will begin with the increasing stages and end with the decreasing stages. The increasing stages will allow ARSS to overlap communication overheads over WAN with computation overheads and to also collect more accurate performance metrics of the participating clusters, while the decreasing stages will ensure the robustness against both the information inaccuracy and the dynamic behaviour of the underlying system. Equation 1 illustrates how our strategy calculates the amount of unit tasks allocated for stage j (S_j) where U is the total number of unit tasks and j starts from 1 until it reaches the last stage, which is $2(\log_2 U - 1)$. An example of how ARSS assigns 8,192 unit tasks for each stage is shown in Figure 2.

$$S_j = \begin{cases} \left\lceil \dfrac{U}{2^{(\log_2 U - j + 1)}} \right\rceil & j \le \log_2 U - 1 \\ \left\lceil \dfrac{U}{2^{(j + 2 - \log_2 U)}} \right\rceil & otherwise \end{cases} \quad (1)$$

During each stage, the coordinator node will assign tasks to participating clusters and estimate their performance based on the number of assigned tasks and the interval time between requests. Since this information is derived based on the interval time between requests, the coordinator node does not have these metrics at the beginning of the execution. Thus, the coordinator node will assign an equal number of unit tasks to all clusters at the first stage. In the

Maejo Int. J. Sci. Technol. **2011**, *5*(02), 215-230

subsequent stages, ARSS will dynamically adjust the assigned tasks based on the pre-defined number of tasks in each stage and the relative performance of the computing resources.

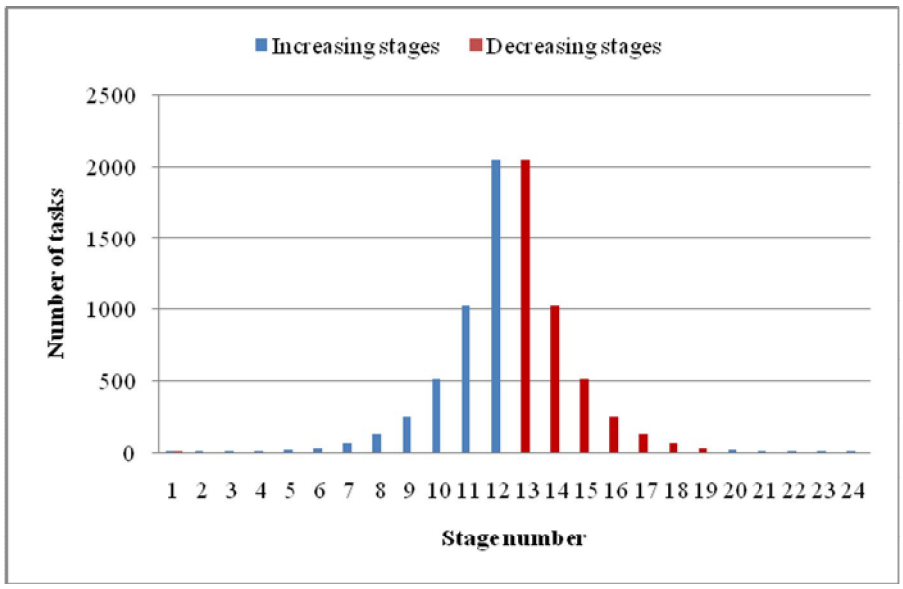


Figure 2. Number of unit tasks allocated within each stage

Let $R_{i,j}$ represent the number of tasks executed per second by cluster i during stage j. In an ideal case, all resources will finish their execution of all assigned tasks of stage j at the same time before entering the next stage, $j+1$. However, because $R_{i,j}$ is an estimation at the coordinator node, it can be inaccurate. Thus, while some clusters may complete the execution of stage j and be ready to move further to stage $j+1$, other clusters may still be in the middle of the execution of stage j. This behaviour will create a load imbalance and can degrade the overall execution time. To address this problem, our strategy performs a job-stealing mechanism called "stage-warping". By including the leftover tasks of the previous stage when allocating tasks for the next stage, this technique will make sure that every cluster will progress through each stage at the same pace until the end of the execution. Given L_{j-1} as the number of leftover tasks from the previous stages $j-1$, the number of tasks assigned to cluster i during the current stage j ($A_{i,j}$) can be specified as

$$A_{i,j} = \frac{R_{i,j-1}}{\sum_{k=1}^{M} R_{k,j-1}} \left(L_{j-1} + S_j \right) \tag{2}$$

From equation 2, we can see that the leftover workload in the previous stage $j-1$ will be reassigned together with other tasks allocated for the current stage j. In other words, this behaviour will allow other computing resources to steal the workload from computing resources which have been slower than expected due to the estimation error of our performance metrics.

Maejo Int. J. Sci. Technol. **2011**, *5*(02), 215-230

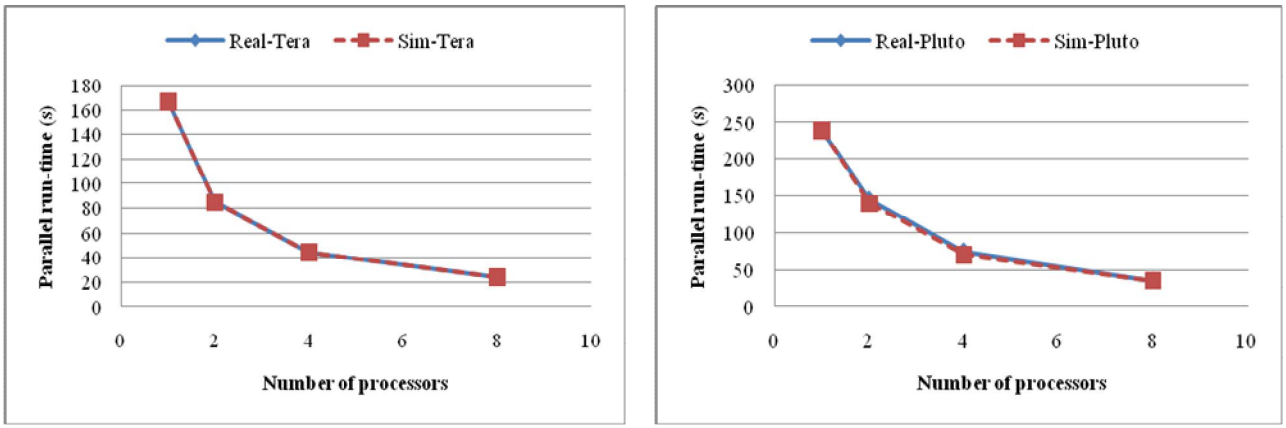
RESULTS AND DISCUSSION

Simulation Environment for Performance Evaluation

The performance of our strategy was evaluated by creating a large-scale computing system using ns-2 simulator [25]. To ensure the validity of our simulation experiments, we compared the results with those from the actual computing environments, i.e. TERA and PLUTO clusters in Thaigrid [26]. TERA cluster belongs to Kasetsart University while PLUTO cluster belongs to Chulalongkorn University. Table 1 presents the parameters for our simulations collected from the actual environment. Note that the unit time represents the computation time for one computing node in each cluster to execute one row multiplication of the submitted matrix multiplication program. Using parameters from real environments, we simulated the SS strategy in our simulator and compared the results of using the strategy on the TERA and PLUTO clusters as presented in Figure 3. The obtained results clearly show that our model can accurately predict the parallel performance of computationally intensive application over the computing clusters.

Table 1. Simulation parameters for evaluating the accuracy of the test environment

Variable	Value
Unit time (TERA)	0.083s
Unit time (PLUTO)	0.117s
Number of unit tasks	2000
LAN propagation delay	30μs
LAN bandwidth	1000Mbps



(a) (b)

Figure 3. Comparison between real and simulated environments: a) TERA; b) PLUTO

Since this work focuses on the global strategy, the local strategy in all clusters is defined as SS. We define a uniformly distributed random variable to represent the actual computing power within each node to simulate the randomness of the available computing power as proposed by Casanova [27]. The specified computing power was varied within ±30% from the expected value.

Maejo Int. J. Sci. Technol. **2011**, *5*(02), 215-230

The computing heterogeneity was simulated by changing the expected value of the computing power. Let H represent the computing heterogeneity, which is the computing ratio between the fastest and slowest clusters in the system ($p_{fastest} / p_{slowest}$) [28]. Unless specified otherwise, the environment in our simulation is assumed to be highly heterogeneous where H is specified as 10 [29]. We assume that it takes 1 second to execute a task whose computation size is 1 task unit on the computing resource with computing power specified as 1 power unit. As for the prediction error from the monitoring service, a uniform distribution random variable is used to represent the prediction error. The range of the prediction error is defined to be ±30%. The other related parameters for simulating a large-scale multi-organisational computing environment are summarised in Table 2.

Table 2. Related parameters for simulating multiple-cluster environments

Variable	Value
Number of clusters (M)	4
Total computing nodes (N)	128
Total computing power (P)	128
Inter-cluster propagation delay (α_W)	30ms
Inter-cluster bandwidth (β_W)	2Mb/s
Intra-cluster propagation delay (α_L)	1ms
Intra-cluster bandwidth (β_L)	100Mb/s
Number of tasks (U)	16,384
Total computation size (W)	16,384
Total communication size (V)	16,384 kB

Overall Performance Comparison

To evaluate the performance of load sharing strategies, the ARSS and four other load sharing strategies, namely chunk self-scheduling (CSS), one-time, WFSS and AWF, were simulated on two different computing environments, i.e. homogeneous and highly heterogeneous environments. In our simulation, CSS always assigned a fixed-size chunk of eight unit tasks to the requesting cluster. For one-time strategy, the entire workload was proportionally assigned all at once to participating resources with respect to the performance information obtained from the monitoring service. Finally, the performance of the predecessor of AWF, which is WFSS, was also evaluated. Unlike AWF, WFSS uses only the estimators obtained from the monitoring service for distributing the workload at every stage.

As illustrated in Figure 4, all strategies performed better in a homogeneous environment. The parallel run-times of CSS suffered from large communication overheads and load imbalance, especially when the underlying system was highly heterogeneous. Although we could reduce the communication overheads by assigning the workload within only one round, the performance of the

Maejo Int. J. Sci. Technol. **2011**, *5*(02), 215-230

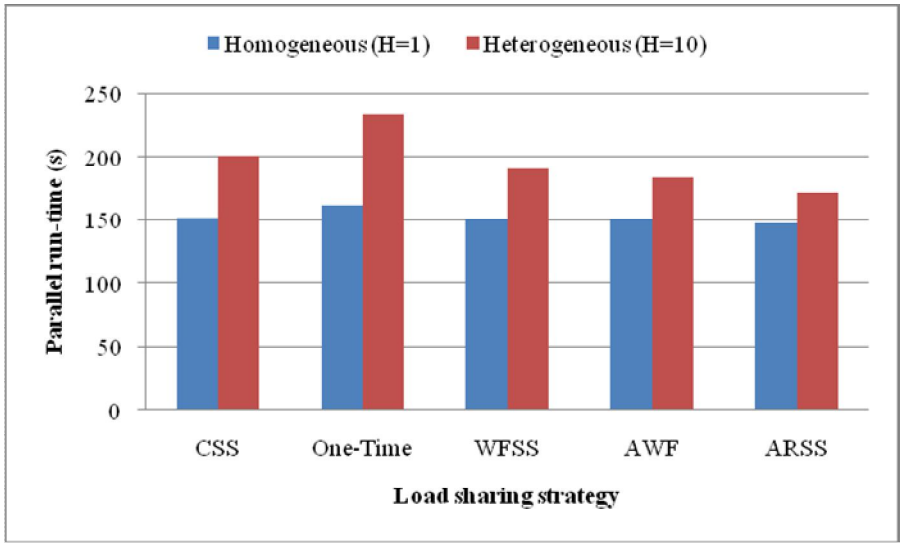


Figure 4. Parallel run-times over two different environments

one-time strategy was the worst on both systems because of an excessive load imbalance due to inaccurate information. WFSS and AWF performed equally well in a homogeneous system. However, AWF outperformed WFSS in a highly heterogeneous system because of its ability to adjust weight values for making load decision during the execution. Finally, ARSS performed as well as WFSS and AWF over a homogeneous system although it had to spend additional time during the increasing stages for obtaining accurate performance metrics. With exponentially increasing and decreasing task allocation during the increasing and decreasing stages, ARSS did not suffer from large communication overheads like CSS. The prediction error from the monitoring service did not affect the performance of ARSS because it used its own simple performance metrics calculated during the execution. Therefore, ARSS achieved the best parallel run-time over a highly heterogeneous system, given inaccurate performance information from the monitoring service.

Effect of Information Inaccuracy from Monitoring Services

The behaviour of the traditional strategies which rely on performance information from the monitoring service was evaluated. As mentioned earlier, we modelled the prediction errors as upper- and lower-bound percentages of the uniformly distributed random variable.

In Figure 5, it is quite obvious that AWF performs best when the estimated value is accurate (zero prediction error). As the prediction error increases, AWF will perform worse. The reason behind this behaviour is the characteristic of AWF. Since AWF assigns half of the available workload during the first stage, the problem of inaccurate information given a large prediction error can affect the performance of this strategy despite its effort to adjust the accuracy of the weighted values during the execution in the subsequent stages. On the contrary, the prediction error from the monitoring service does not affect the performance of our strategy. This is due to the fact that ARSS relies only on its simple performance metrics obtained by the coordinator node during the execution instead of using the values measured by the monitoring services.

Maejo Int. J. Sci. Technol. **2011**, *5*(02), 215-230

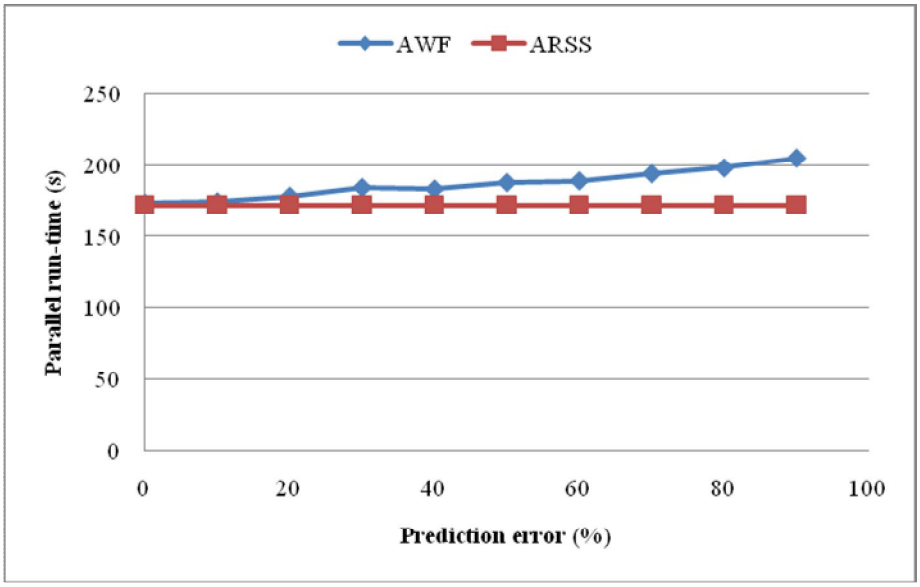


Figure 5. Parallel run-times with different prediction errors from monitoring service

Effect of Computing Heterogeneity

The differences in the computing power between participating clusters can increase the risk of load imbalance. Unintentionally sending one additional task to a computing cluster ten times slower than other clusters can result in a very bad parallel run-time because other clusters have to wait until that cluster finishes. Figure 6 illustrates the performance of load sharing strategies with different values of computing heterogeneity (*H*).

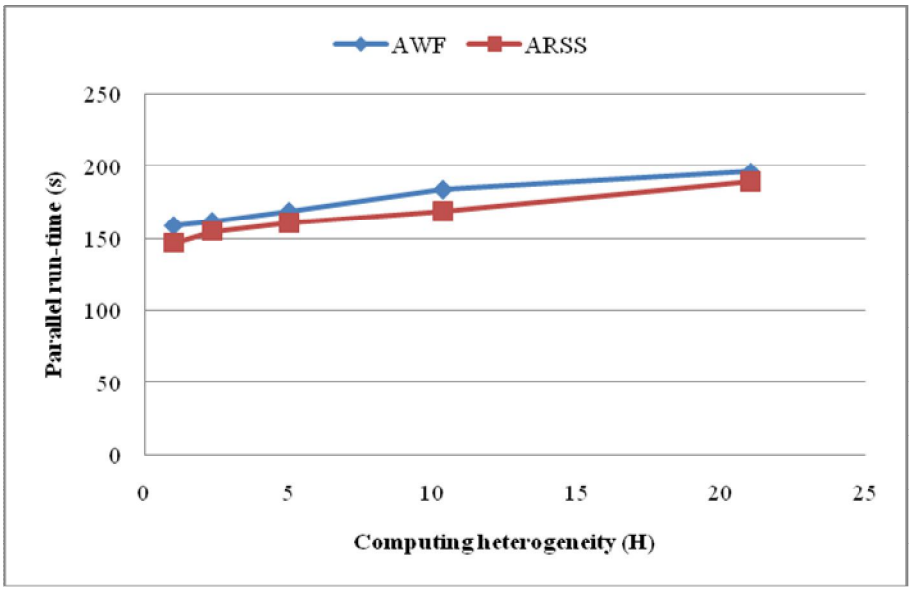


Figure 6. Parallel run-times at different values of computing heterogeneity

Maejo Int. J. Sci. Technol. **2011**, *5*(02), 215-230

As shown in Figure 6, the parallel run-times of both strategies increases when the underlying system becomes more heterogeneous. However, since ARSS utilises job-stealing technique to ensure that every cluster enters the same stage throughout the entire execution, it can perform better than AWF over different values of computing heterogeneity.

Effect of Communication Overheads

Large communication overheads can degrade the performance of load sharing strategies. The effect of communication overheads was evaluated by focusing on two parameters: the propagation delay in WAN and the communication size of the submitted application. The effect of propagation delay in WAN on different load sharing strategies is shown in Figure 7.

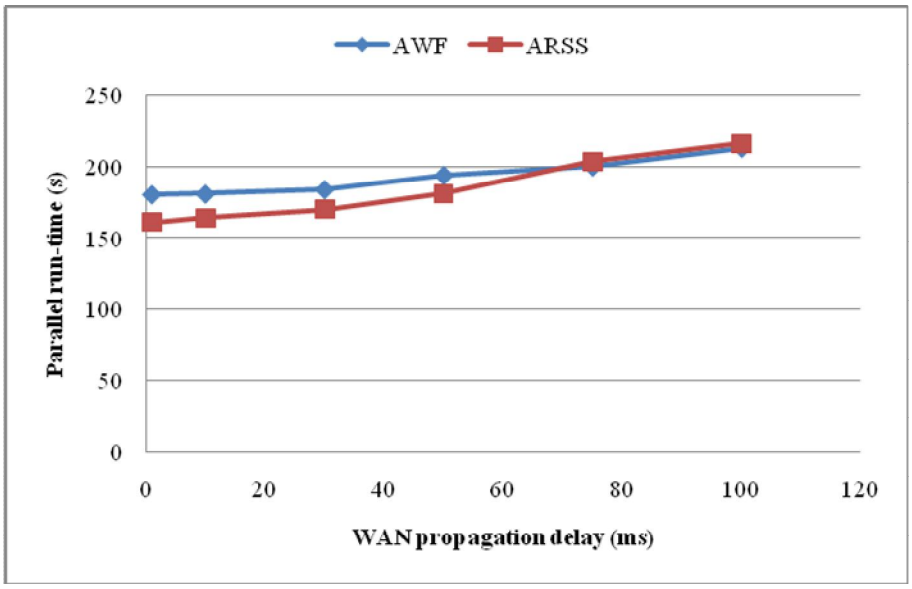


Figure 7. Parallel run-times at different WAN propagation delays

The obtained results illustrate that the performance of ARSS is better than that of AWF when the propagation delay in WAN is low. Since the accuracy of simple performance metrics used by ARSS decreases with propagation delay, the parallel run-times of both strategies become comparable when the WAN propagation delay is large.

From Figure 8, it can be seen that the performance of ARSS is much better than that of AWF when the communication size is large. The explanation for this behaviour is that AWF assigns workload in a decreasing fashion. Therefore, it suffers from a large amount of communication overheads during the first stage. Since ARSS starts the execution with the increasing stages, it can overlap the communication overheads with the computation time for each cluster. Note that, in our experiments, we do not consider the data-intensive applications with large communication sizes. In real life, large data sets which may be up to terabytes in total are usually pre-fetched to the computing resources before the execution begins in order to hide the communication overheads. Therefore, it is uncommon to assume that these large data sets will be transferred during run-time.

Maejo Int. J. Sci. Technol. **2011**, *5*(02), 215-230

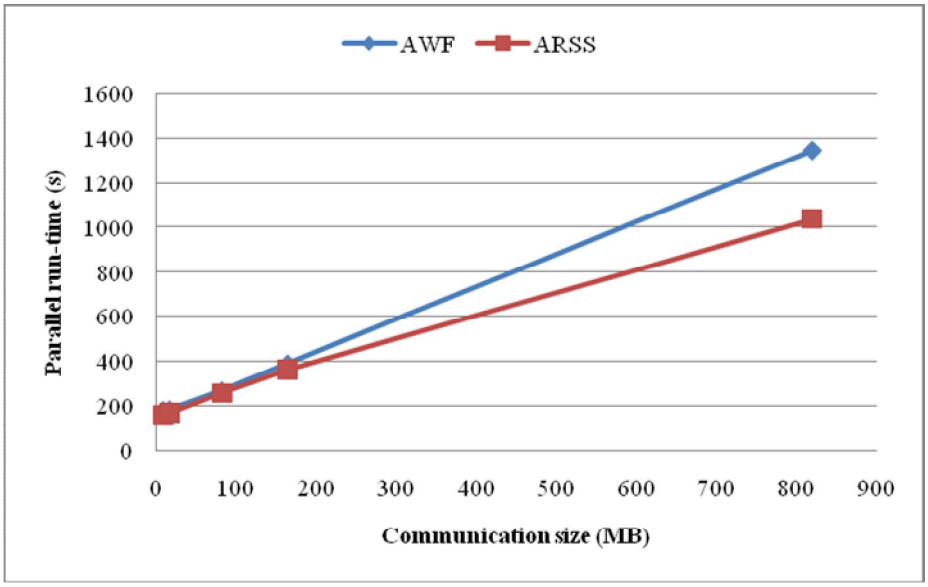


Figure 8. Parallel run-times at different communication sizes

Effect of Application Pattern

Since ARSS will use the first half of total unit tasks to collect performance metrics, the characteristic of the submitted application can affect its performance. However, as shown in Figure 9, it can be seen that the performance of ARSS is barely changed given four different classes of applications. If an application has constant-size tasks, ARSS can perform better than AWF because the performance metrics obtained during the increasing stages are accurate. In the case of an application with increasing-size tasks, the performance of AWF is not good since the size of the unit tasks near the end of the execution, which will be used to balance the workload assigned during the previous stage, is large. In contrast to AWF, ARSS gradually increases the chunk size at the beginning and also gradually decreases it near the end of the execution. Although this behaviour will add a small amount of communication overheads, it provides stability to ARSS even when the computation size of each task is not the same.

Effect of Total Number of Unit Tasks

The number of unit tasks in a submitted application is also one of the important factors because it can affect the accuracy of the obtained performance metrics used by ARSS. Since we keep the same total computation size for every test, the computation size of each task will be larger when the total number of unit tasks is decreased. From Figure 10, it can be seen that the performance of ARSS is still comparable or even better than AWF given a limited number of unit tasks. This behaviour shows that ARSS is robust enough to obtain good results even with a small number of unit tasks.

Maejo Int. J. Sci. Technol. **2011**, *5*(02), 215-230

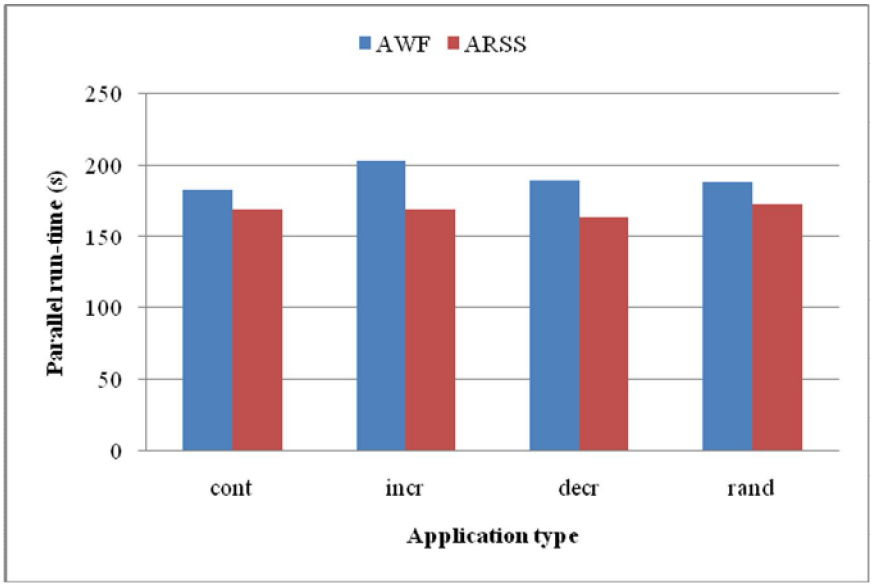


Figure 9. Parallel run-times for four different applications: those with constant-size (cont), increasing-size (incr), decreasing-size (decr), and randomized-size (rand) tasks

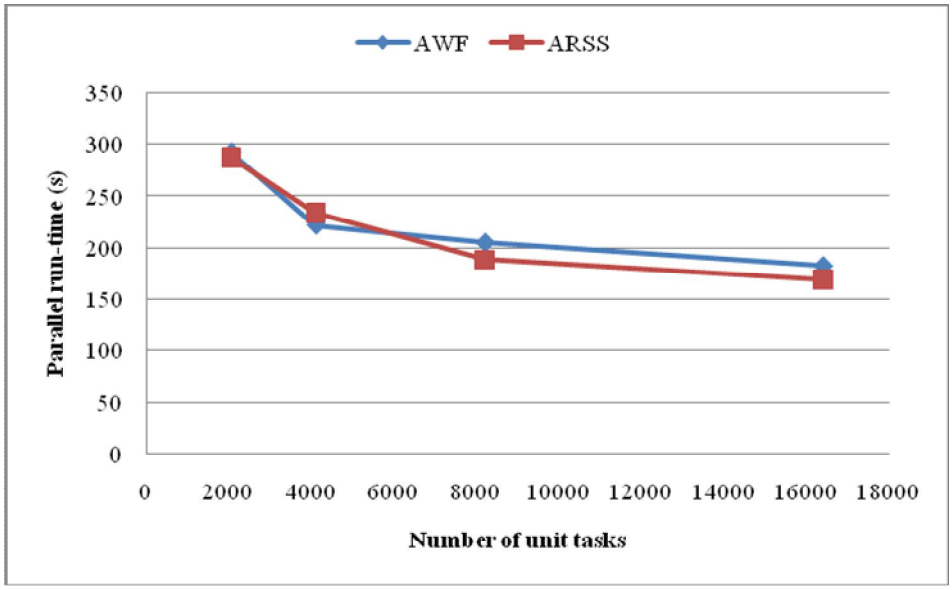


Figure 10. Parallel run-times for different number of unit tasks

CONCLUSIONS

We have proposed a new global strategy called agentless robust self-scheduling strategy (ARSS) for large-scale multi-organisational computing systems. Unlike other traditional strategies, our proposed strategy can make load decision without any proprietary monitoring services installed at the participating resources. In order to address communication overheads in WAN and dynamic behaviour over a heterogeneous computing system, our strategy divides an entire computation into multiple stages. The increasing stages during the beginning of the execution are for obtaining an

accurate estimation of the computing power of each resource and also for overlapping the communication overheads. After that, the decreasing stages will eliminate the load imbalance until the end of the execution. During each stage, our strategy addresses the dynamic behaviour of the underlying system by assigning workload according to the performance metrics obtained in the previous stages. We also introduce a stage-warping technique to further handle the performance estimation errors. This technique will allow clusters to steal workload from those slower-than-expected clusters. The experimental results have shown that the proposed strategy can achieve a comparable or better performance compared to that of other traditional strategies. Our strategy is therefore non-intrusive and efficient at utilising heterogeneous resources over WAN, which definitely will be served as the computing platform for the next generation.

REFERENCES

1. I. Foster, C. Kesselman and S. Tuecke, "The anatomy of the grid: Enabling scalable virtual organizations", *Int. J. High Perform. Comput. Appl.*, **2001**, *15*, 200-222.
2. R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility", *Future Generat. Comput. Syst.*, **2009**, *25*, 599-616.
3. R. Moreno-Vozmediano, R. S. Montero and I. M. Llorente, "Elastic management of cluster-based services in the cloud", Proceedings of the 1st Workshop on Automated Control for Datacenters and Clouds, **2009**, Barcelona, Spain, pp.19-24.
4. J. S. Vetter and D. A. Reed, "Real-time performance monitoring, adaptive control, and interactive steering of computational grids", *Int. J. High Perform. Comput. Appl.*, **2000**, *14*, 357-366.
5. B. Tierney, B. Crowley, D. Gunter, J. Lee and M. Thompson, "A monitoring sensor management system for grid environments", *Cluster Comput.*, **2001**, *4*, 19-28.
6. R. Biswas, M. Frumkin, W. Smith and R. Van der Wijngaart, "Tools and techniques for measuring and improving grid performance", *Lect. Notes Comput. Sci.*, **2002**, *2571*, 45-54.
7. R. Wolski, "Experiences with predicting resource performance on-line in computational grid settings", *Perform. Eval. Rev.*, **2003**, *30*, 41-49.
8. X. H. Sun and M. Wu, "Grid harvest service: A system for long-term, application-level task scheduling", Proceedings of the 17th IEEE International Parallel and Distributed Processing Symposium, **2003**, Nice, France, pp.25-33.
9. I. Raicu, I. T. Foster and Y. Zhao, "Many-task computing for grids and supercomputers", Proceedings of ACM Workshop on Many-Task Computing on Grids and Supercomputers, **2008**, Austin, USA, pp.1-11.
10. C. P. Kruskal and A. Weiss, "Allocating independent subtasks on parallel processors", *IEEE Trans. Software Eng.*, **1985**, *11*, 1001-1016.
11. P. Tang and P. C. Yew, "Processor self-scheduling for multiple-nested parallel loops", Proceedings of the International Conference on Parallel Processing, **1986**, Harrisburg, USA, pp. 528-535.

12. S. C. Chau and A. W. C. Fu, "Load balancing between heterogeneous computing clusters", *Lect. Notes Comput. Sci.*, **2004**, *3032*, 75-82.
13. R. Buyya, M. Murshed, D. Abramson and S. Venugopal, "Scheduling parameter sweep applications on global Grids: a deadline and budget constrained cost-time optimization algorithm", *Softw. Pract. Exper.*, **2005**, *35*, 491-512.
14. Z. Zhang, A. Espinosa, K. Iskra, I. Raicu, I. Foster and M. Wilde, "Design and evaluation of a collective IO model for loosely coupled petascale programming", Proceedings of ACM Workshop on Many-Task Computing on Grids and Supercomputers, **2008**, Austin, USA, pp.1-10.
15. Y. W. Fann, C. T. Yang, S. S. Tseng and C. J. Tsai, "An intelligent parallel loop scheduling for multiprocessor systems", *J. Inform. Sci. Eng.*, **2000**, *16*, 169-200.
16. C. D. Polychronopoulos and D. J. Kuck, "Guided self-scheduling: A practical scheduling scheme for parallel supercomputers", *IEEE Trans. Comput.*, **1987**, *36*, 1425-1439.
17. T. H. Tzen and L. M. Ni, "Trapezoid self-scheduling: A practical scheduling scheme for parallel compilers", *IEEE Trans. Parallel Distr. Syst.*, **1993**, *4*, 87-98.
18. A. T. Chronopoulos, R. Andonie, M. Benche and D. Grosu, "A class of loop self-scheduling for heterogeneous clusters", Proceedings of the IEEE Annual International Conference on Cluster Computing, **2001**, Newport Beach, USA, pp.282-291.
19. K. W. Cheng, C. T. Yang, C. L. Lai and S. C. Chang, "A parallel loop self-scheduling on grid computing environments", Proceedings of the International Symposium on Parallel Architectures, Algorithms and Networks, **2004**, Kaohsiung, Taiwan, pp.409-414.
20. S. F. Hummel, E. Schonberg and L. E. Flynn, "Factoring: A method for scheduling parallel loops", *Comm. ACM*, **1992**, *35*, 90-101.
21. S. F. Hummel, J. Schmidt, R. N. Uma and J. Wein, "Load-sharing in heterogeneous systems via weighted factoring", Proceedings of the ACM Symposium on Parallelism in Algorithms and Architectures, **1996**, Padua, Italy, pp.318-328.
22. I. Banicescu and V. Velusamy, "Performance of scheduling scientific applications with adaptive weighted factoring", Proceedings of the 15th IEEE International Parallel and Distributed Processing Symposium, **2001**, San Francisco, USA, pp.791-801.
23. Y. Yang and H. Casanova, "RUMR: Robust scheduling for divisible workloads", Proceedings of the 12th High Performance Parallel and Distributed Computing, **2003**, Seattle, USA, pp. 114-123.
24. Y. C. Lee and A. Y. Zomaya, "Practical scheduling of bag-of-tasks applications on grids with dynamic resilience", *IEEE Trans. Comput.*, **2007**, *56*, 815-825.
25. S. McCanne and S. Floyd, "ns-2 network simulator", http://www.isi.edu/nsnam/ns/. (Accession date: 20 Feb. **2006**)
26. V. Varavithya and P. Uthayopas, "ThaiGrid: Architecture and overview", *NECTEC Tech. J.*, **2000**, *2*, 219-224.
27. H. Casanova, "Simgrid: A toolkit for the simulation of application scheduling", Proceedings of the 1st IEEE/ACM International Symposium on Cluster Computing and the Grid, **2001**, Brisbane, Australia, pp.430-437.

Maejo Int. J. Sci. Technol. **2011**, *5*(02), 215-230

28. C. T. Yang, K. W. Cheng and W. C. Shih, "On development of an efficient parallel loop self-scheduling for grid computing environments", *Parallel Comput.*, **2007**, *33*, 467-487.
29. K. Lu, R. Subrata and A. Y. Zomaya, "On the performance-driven load distribution for heterogeneous computational grids", *J. Comput. Syst. Sci.*, **2007**, *73*, 1191-1206.