Hybrid Performance-oriented Scheduling of Moldable Jobs with QoS Demands in Multiclusters and Grids*

Ligang He, Stephen A. Jarvis, Daniel P. Spooner, Xinuo Chen and Graham R. Nudd

Department of Computer Science, University of Warwick
Coventry, United Kingdom CV4 7AL
liganghe@dcs.warwick.ac.uk

Abstract. This paper addresses the dynamic scheduling of moldable jobs with QoS demands (soft-deadlines) in multiclusters. A moldable job can be run on a variable number of resources. Three metrics (over-deadline, makespan and idle-time) are combined with weights to evaluate the scheduling performance. Two levels of performance optimisation are applied in the multicluster. At the multicluster level, a scheduler (which we call MUSCLE) allocates parallel jobs with high packing potential to the same cluster; MUSCLE also takes the jobs' QoS requirements into account and employs a heuristic to achieve performance balancing across the multicluster. At the single cluster level, an existing workload manager, called TITAN, utilizes a genetic algorithm to further improve the scheduling performance of the jobs allocated by MUSCLE. Extensive experimental studies are conducted to verify the effectiveness of the scheduling mechanism in MUSCLE. The results show that the comprehensive scheduling performance of parallel jobs is significantly improved across the multicluster.

1 Introduction

Separate clusters are increasingly being interconnected to create multicluster or grid computing architectures [1][2][6][7] and as a result, workload management for these architectures is becoming a key research issue. Parallel jobs that run in these domains can be classified into two categories: rigid and moldable [11]. In this paper, a mechanism is developed to schedule moldable jobs in multiclusters/grids. A moldable job is defined as a parallel job that can be run on a variable number of computers.

A job's execution time may not be inversely proportional to its size due to the presence of communication among execution components [10][12]. Consequently, the smallest product of a job's size and the corresponding execution time results in the least consumption of resources. This size is called the *preferable size* of the job.

In the multicluster architecture assumed in this paper, the constituent clusters may be located in different administrative organizations and as a result be managed with

* This work is sponsored in part by grants from the NASA AMES Research Center (administrated by USARDSG, contract no. N68171-01-C-9012), the EPSRC (contract no. GR/R47424/01) and the EPSRC e-Science Core Programme (contract no. GR/S03058/01).

different performance criteria. In this scheduling work, we combine three metrics (*over-deadline*, *makespan* and *idle-time*) with additional variable weights; this allows the resources in different locations to represent different performance scenarios. Over-deadline is defined as the sum of excess time of each job's finish time over its deadline; makespan is defined as the duration between the start time of the first job and the finish time of the last executed job [8][9].

In this work, the multicluster architecture is equipped with two levels of performance optimisation. A multicluster-level scheduler (called MUSCLE) is developed to allocate moldable jobs with QoS demands (deadlines) to constituent clusters. When a job is submitted to the multicluster, MUSCLE identifies the job's preferable size and corresponding execution time and then allocates jobs with high packing potential in terms of their preferable sizes to the same cluster. It also takes the QoS demands of jobs into account and exploits a heuristic to allocate suitable workload to each cluster. When MUSCLE makes scheduling decisions to distribute jobs to individual clusters, it also determines a *seed schedule* for the jobs allocated to each cluster assuming the jobs are run with their preferable sizes. These seed schedules are sent to the corresponding clusters where an existing workload manager (TITAN [12]) uses a genetic algorithm to transform the schedule into one that further improves the (local) comprehensive performance.

The rest of the paper is organized as follows. The system and workload model is introduced in Section 2. Section 3 briefly presents the genetic algorithm used in TITAN. The design of MUSLCE is proposed in Section 4. Section 5 presents the experimental results. Finally, Section 6 concludes the paper.

2 System and Workload Model

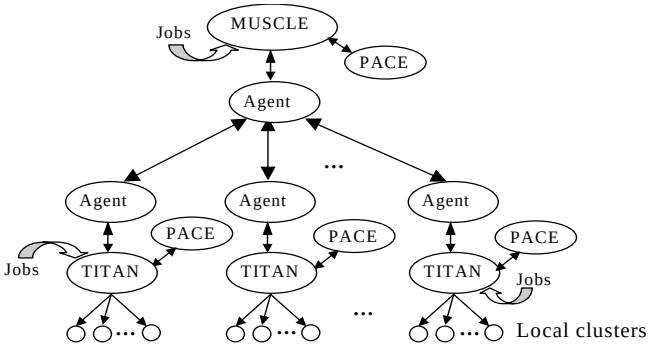


Fig. 1. The multicluster job management architecture

The multicluster architecture assumed in this work (shown in Fig. 1) consists of n clusters, C_1, C_2, …, C_n; where a cluster C_i ($1 \le i \le n$) consists of m_i homogeneous computers (i.e., the size of cluster C_i is m_i), each with a service rate of u_i. There are two scheduling levels in the architecture; MUSCLE acts as the global scheduler for the multicluster while TITAN schedules the jobs sent by MUSCLE within each local

cluster. MUSCLE and TITAN are interconnected through an agent system [3][4][5]. Users can submit parallel jobs to the multicluster through MUSCLE or through TITAN. PACE (Performance Analysis and Characterisation Environment) [10][15] is incorporated into the architecture to provide the jobs' preferable size and corresponding execution time. Parallel jobs are moldable, and a parallel job, denoted by J_i, is identified by a 4-tuple (a_i, s_i, e_{ij}, d_i), where a_i is J_i's arrival time, s_i is its preferable size, et_{ij} is its execution time in cluster C_j ($1 \le j \le n$) and d_i is its soft-deadline (QoS).

3 Local-level Scheduling via TITAN

This section briefly describes the genetic algorithm used in the TITAN workload manager [12]. Three metrics (over-deadline, makespan and idle-time) are combined with the additional weights to form a comprehensive performance metric (denoted by *CP*), which is used to evaluate a schedule. The CP is defined in Eq.1, where Γ, ω and θ are makespan, idle-time and over-deadline, respectively; W^i, W^m and W^o are their weights. For a given weight combination, the lower the value of CP, the better the comprehensive performance.

$$CP = \frac{W^i \Gamma + W^m \omega + W^o \theta}{W^i + W^m + W^o} \quad (1)$$

A genetic algorithm is used to find a schedule with a low CP. The algorithm first generates a set of initial schedules (one of these is the seed schedule sent by MUSCLE). *Crossover* and *mutation* operations are performed to transform the schedules in the current set and generate the next generation. This procedure continues until the performance in each generation of schedule stabilizes.

4 Global-level Scheduling via MUSCLE

The main operations performed by MUSCLE are as follows. First, MUSCLE determines which parallel jobs with preferable sizes can be packed into a *computer space* with a given size. These possible compositions of parallel jobs are organized into a *composition table*. After the composition table is constructed, MUSCLE searches the table for suitable parallel jobs to allocate to the available computer space in a cluster. When the computer space is available in multiple clusters, MUSCLE orders the processing of the space using a heuristic.

4.1 Organizing Parallel Jobs

Suppose the maximum cluster size in the multicluster is m_{MAX} and that at some time point, p parallel jobs, J_1, J_2, …, J_p are collected (into a queue) in the multicluster. Algorithm 1 outlines the steps for constructing the composition table. The p jobs are filled into suitable rows in the table.

```
Algorithm 1. Constructing the composition table
1. for each parallel job J_i (1≤i≤p) to be scheduled do
2.   for each j satisfying 1≤j≤ m_MAX do
3.     if s_i = j
4.       Append J_i to the tail of the j-th row of the
         table;
5.     if s_i < j
6.       r←j−s_i;
7.       if the r-th row in the table is not NULL
8.         if there is such a composition of parallel
           jobs in the r-th row in which no job is
           in the j-th row of the table;
9.           Append J_i as well as the parallel jobs in
             the composition from the r-th row to
             the tail of the j-th row;
```

There are two for-loops in Algorithm 1 and Step 8 searches a row for the qualified composition. In the worst case, the time taken by Step 8 is O(p). Hence, the worst-case time complexity of Algorithm 1 is O($p^2 m_{MAX}$).

4.2 Searching the Composition Table

The algorithm for allocating jobs to a computer space with size r in a cluster proceeds as follows. First, it searches the composition table from the r-th row up to the first row to obtain the first row that is not null. Then, in this row, the algorithm selects the composition in which the number of jobs having been allocated is the least. If the number is zero, these jobs are allocated to the computer space. If a job J_i, whose size is s_i, in the composition has been allocated, a function is called to search the s_i-th row for alternative jobs for J_i. The function is called recursively if a composition cannot be found in the s_i-th row in which no job in it is allocated. The recursive call terminates when there is only one composition in a searched row (i.e. there are no alternative jobs) or when the composition consisting of unallocated jobs is found. If the algorithm fails to allocate jobs to the computer space with size r, it continues by trying to identify jobs to allocate to the computer space with size r=r-1. The procedure continues until r reaches 0. After the allocated jobs have been determined, the schedule for these jobs can also be computed. The time complexity of the procedure is based on the number of jobs that are allocated. The best-case time complexity is O(1) while the worst-case time complexity is O($p^2 n_{MAX}$).

As can be seen from the above description, the job allocation algorithm always attempts to identify the jobs that maximally occupy the given computer space. In doing so, the number of computers left idle is minimized.

4.3 Employing a Heuristic to Balance Performance

A performance metric ε is formed by integrating the workload attributes in clusters. ε is defined in Eq.2, where p_i is the number of jobs, $etSum_i$ is the sum of the execution times of all jobs, $sizeSum_i$ represents the total job sizes and $slkSum_i$ is the total slack

(the slack of a job is its deadline minus its execution time and its arrival time). When multiple clusters offer available computer space, the cluster with the smallest ε is given the highest priority and will be allocated the jobs.

$$\varepsilon = \frac{p_i \times etSum_i \times sizeSum_i}{slkSum_i \times m_i} \quad (2)$$

The complete scheduling procedure for MUSCLE is outlined in Algorithm 2.

```
Algorithm 2. MUSCLE scheduling
1.   while the expected makespan of the jobs yet to be
     scheduled in clusters is greater than a prede-
     fined valve
2.     Collect the arriving jobs in the multicluster;
3.   Call Algorithm 1 to construct the composition ta-
     ble for the collected jobs;
4.   do
5.     for each cluster do
6.       Calculate ε using Eq.2
7.     Get the earliest available computer space in
       cluster C_i which has the minimal ε;
8.     Allocate jobs to this space;
9.     Update the earliest available computer space and
       the values of the workload attributes in C_i;
10.  while all collected jobs have not been allocated;
11.  Go to Step 1;
```

The time of Algorithm 2 is dominated by Step 3 and Step 8 in the do-while loop. Their time complexities have been analysed in subsection 4.1 and 4.2.

5 Experimental Studies

A simulator is developed to evaluate the performance of the scheduling mechanism in MUSCLE. The experimental results focus on demonstrating the performance advantages of the scheduling mechanism in MUSCLE over the scheduling policies frequently used in distributed systems. Weighted Random (WRAND) and Dynamic Least Load (DLL) policies are two selected representatives. The DLL policy schedules jobs to the resource with the least workload. In the WRAND policy, the probability that a job is scheduled to a resource is proportional to its processing capability.

40,000 parallel jobs are generated; the submissions of parallel jobs follow a Poisson process. A job's preferable size follows a uniform distribution in [*MIN_S*, *MAX_S*]. Given a job's preferable size s_i, suppose that the corresponding execution time is $e_{ij,si}$; then when the job's size is s_i+k, the corresponding execution time $e_{ij,si+k}$ is determined by Eq.3, where k can be positive or negative integer and φ is the factor that determines the scalability of the job regarding its size.

$$e_{ij,\,si\,+\,k} = ei_{j,\,si} \times \frac{s_i}{s_i + k} \times (1 + |\,k\,|\,\varphi) \quad (3)$$

When jobs take preferable sizes and the service rate is the average service rate of the constituent clusters in the multicluster, their execution time follow a bounded Pareto distribution, shown in Eq.4, where e_l and e_u are the lower and upper limit of the execution time x.

$$f(x) = \frac{\alpha e_l^{\alpha}}{1-(e_l / e_u)^{\alpha}} x^{-\alpha-1} \quad (4)$$

A job's deadline, d_i is determined by Eq.5, where dr is the deadline ratio. dr follows a uniform distribution in [*MIN_DR*, *MAX_DR*], which is used to measure the deadline range.

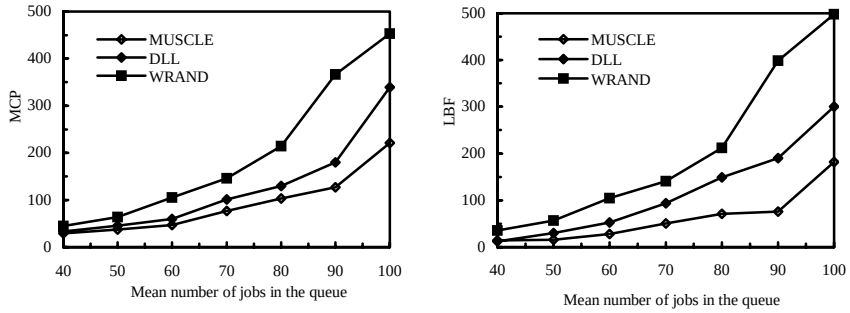
$$d_i = \max\{et_{ij}\} \times (1+dr) \quad (5)$$

The performance metrics evaluated in the experiments are the *Mean Comprehensive Performance* (MCP) and *Load Balance Factor* (LBF). MCP is the average of CP in each cluster and LBF is the standard deviation of these CP's.

5.1 Workload Levels

Table 1. The multicluster setting in Fig.4

Clusters	C_1	C_2	C_3	C_4
Size	20	16	12	10
Service rate ratio	1.0	1.2	1.4	1.6



(a) (b)

Fig. 2. The comparison of MUSCLE, DLL and WRAND under different workload levels in terms of (a) mean comprehensive performance (MCP), and (b) Load balance factor (LBF); (W^o, W^m, W^i)=(4, 3, 1); (e_l, e_u)=(5, 100); (MIN_S, MAX_S)=(1, 10); (MIN_DR, MAX_DR)=(1, 5); φ=0.05; The cluster size and service rate are shown in Table 1

Fig.2.a and Fig.2.b demonstrate the performance difference among MUSCLE, DLL and WRAND scheduling policies under different workload levels. The workload level is measured by the mean number of jobs in the queue (for accommodating the collected jobs) in the multicluster. It can be observed from Fig.2.a that MUSCLE

outperforms the DLL and WRAND policies under all workload levels. This is because the jobs are packed tightly in the seed schedules sent by MUSCLE to the individual clusters. Therefore, the further improvement by the genetic algorithm in each cluster is based on an excellent "seed". As can be observed from Fig.2.b, MUSCLE outperforms DLL and WRAND in terms of LBF except when the mean number of jobs is 40 (in that case, the performance achieved by MUSCLE is slightly worse than that by DLL). The reasoning behind this is as follows. When the workload is low, a small number of jobs miss their deadlines. DLL is beneficial to obtaining the balanced resource throughput and resource utilization. Therefore, DLL shows a more balanced MCP performance. However, as the workload increases further, more jobs miss their deadlines. MUSCLE takes the QoS demands into account so that the MCP performance remains balanced among the clusters.

5.2 Cluster Size

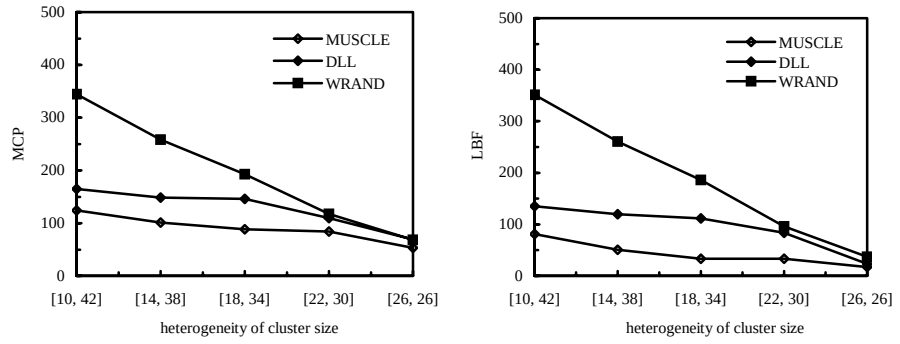


Fig. 3. Performance comparison of MUSCLE, DLL and WRAND under different heterogeneity levels of cluster size, measured by a range [a, b]; (e_l, e_u)=(5, 100); (MIN_S, MAX_S)=(1, a); (MIN_DR, MAX_DR)=(1, 5); φ=(a+1)/a; the mean number of jobs in the queue is 100

Parallel jobs have to be packed into the clusters. Cluster size is therefore an important parameter for parallel job scheduling. Fig.3.a and Fig.3.b compare the performance of MUSCLE, DLL and WRAND under different heterogeneity levels of cluster size. The heterogeneity levels of cluster size are measured by the scale of the range from which cluster sizes are selected. The multicluster consists of five clusters. Five sets of cluster sizes, all with the same average, are uniformly chosen from five ranges, [10, 42], [14, 38], [18, 34], [22, 30] and [26, 26]. The range [26, 26] means the multicluster is homogeneous in terms of cluster size. The service rates of computers in all clusters are set to be the same.

As can be observed from Fig.3.a and Fig.3.b, MUSCLE outperforms DLL and WRAND in terms of MCP and LBF in all cases. Further, MCP and LBF performance achieved by all three policies improves as the heterogeneity level decreases. This is

because the decrease in the heterogeneity of the cluster size is beneficial to achieving a balanced load for all policies. Thus, the MCP performance is also improved.

6. Conclusions

A multicluster-level scheduler, called MUSCLE, is described in this paper for the scheduling of moldable jobs with QoS demands in multiclusters. MUSCLE is able to allocate jobs with high packing potential (in terms of their preferable sizes) to the same cluster and further utilizes a heuristic to control the workload distribution among the clusters. Extensive experimental studies have been carried out to verify the performance advantages of MUSCLE.

References

1. M. Barreto, R. Avila, P. Navaux.: The MultiCluster model to the integrated use of multiple workstation clusters. Proc. of the 3rd Workshop on Personal Computerbased Networks of Workstations, 2000, pp. 71–80
2. R. Buyya, M. Baker.: Emerging Technologies for Multicluster/Grid Computing. Proceedings of the 2001 IEEE International Conference on Cluster Computing, 2001
3. J. Cao, D. J. Kerbyson, G. R. Nudd.: Performance Evaluation of an Agent-Based Resource Management Infrastructure for Grid Computing. Proc. of 1st IEEE/ACM International Symposium on Cluster Computing and the Grid, 2001
4. J. Cao, D. J. Kerbyson, E. Papaefstathiou, G. R. Nudd.: Performance Modeling of Parallel and Distributed Computing Using PACE. Proceedings of 19th IEEE Intl Performance, Computing, and Communications Conference, 2000
5. J. Cao, D. P Spooner, S. A Jarvis, G. R Nudd.: Grid load balancing using intelligent agents. To appear in Future Generation Computer Systems special issue on Intelligent Grid Environments: Principles and Applications, 2004
6. L. He, S. A. Jarvis, D. P. Spooner, G. R. Nudd.: Optimising static workload allocation in multiclusters. Proceedings of 18th IEEE International Parallel and Distributed Processing Symposium (IPDPS'04), April 26-30, 2004
7. X. He, X. Sun, G. Laszewski.: QoS Guided Min-Min Heuristic for Grid Task Scheduling. Journal of Computer Sci&Tech, Special Issue on Grid Computing, 18(4), 2003
8. B. G. Lawson, E. Smirni.: Multiple-queue Backfilling Scheduling with Priorities and Reservations for Parallel Systems. 8th Job Scheduling Strategies for Parallel Processing, 2002
9. A. W. Mu'alem, D. G. Feitelson.: Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. IEEE Trans. Parallel & Distributed Syst. 12(6), pp. 529-543, 2001
10. G. R. Nudd, D. J. Kerbyson, E. Papaefstathiou, J. S. Harper, S. C. Perry, D. V. Wilcox.: PACE: A Toolset for the Performance Prediction of Parallel and Distributed Systems. In The Intl Journal of High Performance Computing, 1999.
11. E. Shmueli, D. G. Feitelson.: Backfilling with lookahead to optimize the performance of parallel job scheduling. 9th Job Scheduling Strategies for Parallel Processing, 2003
12. D. P Spooner, S. A Jarvis, J Cao, S Saini, G. R Nudd.: Local Grid Scheduling Techniques using Performance Prediction. IEE Proc. Comp. Digit. Tech., 150(2):87-96, 2003