Hybrid performance-based workload management for multiclusters and grids

L. He, S.A. Jarvis, D.P. Spooner, X. Chen and G.R. Nudd

Abstract: The paper addresses the dynamic scheduling of parallel jobs with quality-of-service demands (soft-deadlines) in multiclusters and grids. Three performance metrics (over-deadline, makespan and idle-time) are combined with variable weights to evaluate the scheduling performance. These three metrics are used to measure the extent to which jobs comply with their QoS demands, the resource throughput and the resource utilisation. Therefore, clusters situated in different administrative organisations can utilise different weight combinations to represent their different performance requirements. 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 allocate suitable workloads to each cluster to balance the overall system performance. At the local cluster level, an existing workload manager, called TITAN, utilises a genetic algorithm to further improve the scheduling performance of the jobs sent by MUSCLE. Extensive experimental studies are conducted to verify the effectiveness of the scheduling mechanism in MUSCLE. The results show that, compared with traditional distributed workload allocation policies, the comprehensive scheduling performance (in terms of over-deadline, makespan and idle-time) of parallel jobs is significantly improved across the multicluster.

1 Introduction

Clusters are increasingly being interconnected to create multicluster or grid computing architectures. These constituent clusters may be located within a single organisation or across different administrative organisations [1–3]. Workload management and scheduling are key issues in multicluster and grid computing and parallel jobs constitute a typical workload type. Parallel jobs can be classified into two categories: rigid and mouldable [4]. Rigid parallel jobs are run on a user-specified number of computational resources, while mouldable jobs can run on a variable number of resources, often determined at run-time [4].

Specific qualities of service (QoS) can be requested by jobs submitted to a grid system [5], an example of which includes specifying user-defined deadlines [2]. The QoS of a job is satisfied if it finishes before the specified deadline, while the QoS decreases as the excess (of completion time over deadline) increases. Therefore, over-deadline can be used to measure the extent to which the QoS demands of a job set is satisfied, where over-deadline is defined as the sum of the excess time of each job's finish time over its deadline. The scheduling of parallel jobs has been studied extensively in single cluster environments [4, 6–8]. In such a scenario, submitted jobs typically have no QoS requirements and resource utilisation is a commonly used system-oriented metric [4]. Idle-time in a resource can be viewed as the metric for measuring resource utilisation [4]. An additional goal in job scheduling in computational grid environments is high resource throughput. In these scenarios, makespan is often used to measure throughput [5]. Makespan is defined as the duration between the start time of the first job and the finish time of the last executed job.

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

In this work, the multicluster architecture is equipped with two levels of performance optimisation. A multicluster scheduling lever (MUSCLE) is developed at a global (multicluster) level to allocate jobs to constituent clusters. MUSCLE is designed to allocate parallel jobs with high packing potential (i.e. they can be packed more tightly) to the same cluster. It also takes the QoS demands of jobs into account and exploits a heuristic to control workload allocation among clusters, so as to balance the overall performance across the multicluster. When MUSCLE distributes jobs to individual clusters, it determines a seed schedule for the jobs allocated to each cluster. These seed schedules are sent to the corresponding clusters where an existing workload manager (TITAN [9]) uses a genetic algorithm to transform the schedule into one that improves the local (cluster) level comprehensive performance.

In grid systems, the global scheduler usually has no control over the local schedulers. This presents difficulties when designing effective scheduling schemes for such environments. The two-level optimisation architecture developed in this paper overcomes this difficulty. MUSCLE has no control over the operations of the TITAN scheduler

IEE Proceedings online no. 20041088

doi: 10.1049/ip-sen:20041088

Paper received 24th August 2004. Originally published online 1st November 2004

The authors are with the Department of Computer Science, University of Warwick, Coventry, United Kingdom CV4 7AL

at each local cluster. There is also no communication between global-level decision making (using MUSCLE) and local-level optimisation (using TITAN).

Another challenge in grid scheduling is that jobs can be submitted from different locations and so a grid scheduler generally lacks a complete view of the jobs in the grid. In this paper we preserve realistic usage patterns in that users are allowed to submit jobs to the multicluster through MUSCLE or through the TITANs of each local cluster. If a job is submitted through TITAN, another (existing) grid component called A4 (agile architecture and autonomous agents) [10, 11] informs MUSCLE as to the submission and provides summary metadata (e.g. job size, arriving time) for the job. A4 is responsible for resource advertisement and discovery as well as for transferring jobs (or job metadata) among schedulers or resources if necessary. This paper focuses on presenting the scheduling mechanism found in MUSCLE. A detailed design of the A4 component is addressed in [10, 11].

Job scheduling is extensively documented in the literature [2, 12–16]. Mechanisms of co-location and adaptive scheduling are presented in AppleS [12]. However, AppleS does not consider jobs' deadlines in scheduling. Condor [16] aims to achieve high throughput in a local network environment – a goal which is different from this work. Nimrod [2] takes jobs' deadlines into account. However, it is based on an economy model and therefore also has different concerns from this work. A QoS guided min–min heuristic is presented in [5] for grids. The QoS in [5] is based on the network bandwidth, and makespan is used to evaluate the scheduling performance. In this paper, multiple performance metrics are considered. Reference [17] also utilises multiple metrics (timeliness, reliability, security, etc.) to evaluate scheduling performance. However, the scheduling policy is static and schedules single processor jobs in a local cluster.

2 System and workload model

The multicluster architecture assumed in this work (shown in Fig. 1) consists of n clusters, $C_1, C_2, \ldots, C_n$, where cluster $C_i\,(1 \le i \le n)$ consists of m_i homogeneous computational resources, each with a service rate of u_i. There are two scheduling levels in the architecture: MUSCLE acts as the multicluster scheduler while TITAN schedules the jobs sent by MUSCLE within each local cluster. MUSCLE and TITAN are interconnected through the A4 system. Users can submit parallel jobs to the multicluster through MUSCLE or through TITAN. If a job is submitted via TITAN, MUSCLE is made aware of it through metadata sent by the A4 agents. Also, if the resources in the multicluster change (e.g. addition or deletion), MUSCLE is also informed of this via A4. Hence, the multicluster is

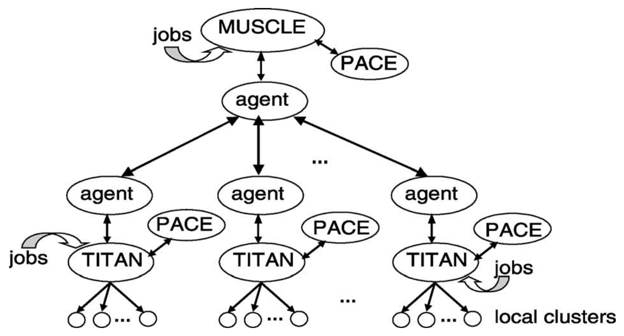

Fig. 1 *Multicluster job management architecture*

a virtual organisation from the perspective of users. Submitted jobs are judicially allocated by MUSCLE to suitable clusters for further scheduling and execution. The PACE (performance analysis and characterisation environment) toolkit [8, 18] is incorporated into the architecture to provide execution time predictions for the submitted jobs.

It is assumed that the parallel jobs submitted to the multicluster can be executed in any constituent cluster. Parallel jobs considered in this paper are rigid. 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 job size (i.e. the number of computational resources on which the job is requested to be run), e_{ij} is its execution time in cluster $C_j (1 \le j \le n)$ and d_i is its soft-deadline.

The jobs' deadlines can be determined through two approaches. On the one hand, the users are allowed to explicitly specify the submitted jobs' deadlines. On the other hand, a deadline can be assigned to a job by the system according to the service class which the job falls into [19]. For example, the jobs in the gold service class will be assigned shorter deadlines than those in the silver class.

3 Local-level scheduling via TITAN

This Section briefly describes the genetic algorithm used by TITAN [9]. A two-dimensional coding scheme is developed to represent a schedule of parallel jobs in a cluster. Each column in the coding specifies the allocation of the resources to a parallel job, while the order of these columns in the coding is also the order in which the corresponding jobs are to be executed. An example is given in Fig. 2 and illustrates the coding for a schedule as well as the execution of the corresponding jobs.

Three metrics (over-deadline, makespan and idle-time) are combined with variable weights to form a comprehensive performance metric (denoted by CP), which is used to evaluate a schedule. The CP is defined in (1), where Γ, ω and θ are makespan, idle-time and over-deadline, and 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:

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

A genetic algorithm is used by TITAN to find a schedule with a low CP. The algorithm first generates a set of initial schedules (one of these being the seed schedule sent by MUSCLE and the others are generated randomly). The performance of a schedule, evaluated by the CP, is normalised to a fitness function which is shown in (2), where $CP^{\max}$ and $CP^{\min}$ represent the best and the worst performance in the schedule set, and CP^k is the performance of the kth schedule:

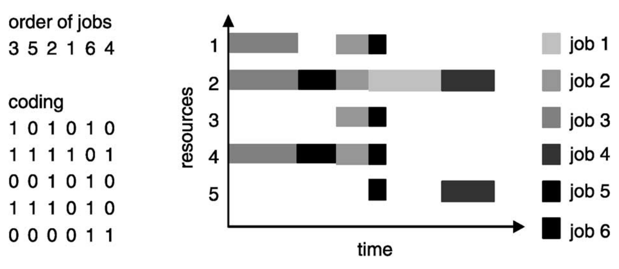

Fig. 2 *TITAN coding scheme for a schedule and corresponding execution map*

$$f_k = \frac{CP^{\max} - CP^k}{CP^{\max} - CP^{\min}} \quad (2)$$

In the genetic algorithm, the value of the fitness function of a schedule determines the probability that the schedule is selected to create the next generation. *Crossover* and *mutation* operations are performed to generate the next generation of the current schedule set. This procedure continues until the performance in each generation of schedule stabilises.

4 Multicluster-level scheduling via MUSCLE

The main operations performed by MUSCLE are as follows. First, MUSCLE determines which parallel jobs can be packed into a resource space with a given size (i.e. available resources of a given number). It is possible that with a set of parallel jobs, different compositions of jobs can be packed into a given resource space. These possible compositions of parallel jobs for packing into a given resource space are organised into a composition table. MUSCLE then searches this table for suitable parallel jobs when allocating jobs into an available resource space in a cluster. When the resource space is available in multiple clusters, MUSCLE orders the processing of the spaces using a heuristic.

4.1 Organising parallel jobs

Suppose the maximum cluster size in the multicluster is m_{MAX} and that, at some time point, p parallel jobs, $J_1, J_2, \ldots, J_p$, are collected by MUSCLE (into a queue). Algorithm 1 outlines the steps for constructing the composition table. The p jobs are filled into suitable rows in the table. When trying to fill job J_i (with size s_i) into the jth row, the algorithm checks if there exists such a composition in the $(j - s_i)$th row that no job in the composition has appeared in the jth row. If such a composition exists, it indicates that J_i and the jobs in the composition can be packed into the resource space with size j. Hence, J_i and these jobs are filled into this row.

Algorithm 1: Constructing the composition table

```
1. for each parallel job $J_i$, $1 \le i \le p$, to be scheduled do
2. for each $j$ satisfying $1 \le j \le m_{MAX}$ 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. $r \leftarrow j - s_i$;
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;
```

The composition table has m_{MAX} rows. A composition in the table is located by a pair (r, l) and the composition is denoted by $cps(r, l)$, where r is the row number in which the composition lies and l is the position subscript of the first job of the composition in the row. A job is not permitted to appear more than once in the same row. This rule is to guarantee that a job will not be allocated to a different resource space. There are two for-loops in Algorithm 1. 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($p^2 m_{MAX}$). Algorithm 1 is illustrated by the following case study. The cluster setting and the parameters of the jobs in the queue are listed in Table 1; these are used as the working example for all remaining case studies in this paper. Table 2 shows the composition table after filling these jobs.

Table 1: Working parameters of case studies demonstrated in this paper

Clusters	C_1	C_2				
Size	4	6				
Service rate of resources	1	1				
Jobs	J_1	J_2	J_3	J_4	J_5	J_6
Job size	2	1	4	3	1	2
Execution time	2	4	4	6	2	4
Slack	6	8	14	12	4	8

Table 2: Composition table case study; jobs J_1–J_6 are listed in Table 1 and m_{MAX} is 6

1	J_2	J_5	
2	J_1	J_5, J_2	J_6
3	J_2, J_1	J_4	J_6, J_5
4	J_3	J_4, J_2	J_6, J_1
5	J_3, J_2	J_4, J_1	
6	J_3, J_1	J_6, J_4, J_2	

4.2 Searching the composition table

The procedure of allocating jobs to a resource space with size r in a cluster proceeds as follows (Algorithm 3). First, it searches the composition table from the rth 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 resource space. If a job J_i, whose size is s_i, in the composition has been allocated, a function (Algorithm 2) is called to search the s_ith row for alternative jobs for J_i. The function is called recursively if a composition cannot be found in the s_ith 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 resource space with size r, it continues by trying to identify jobs to allocate to the resource space with size $r = r - 1$. The procedure continues until r reaches 0. After allocated jobs have been determined, the schedule for these jobs can also be computed (Step 11 in Algorithm 3).

Algorithm 2: Calculating the alternatives for the job allocation in composition $cps(r, l)$
Input: the position of the composition in the composition table (r, l); an array q (used to store alternative jobs).
Output: if success, return 1; otherwise, return 0; (array q contains partial alternative jobs).

```
1. $lc \leftarrow l$; $fail \leftarrow 0$;
2. while $lc$ does not reach the end of the composition and
      $fail$ equals 0
3. Get the job $J_i$ pointed to by $lc$;
4. if $J_i$ has not been allocated
5. Append $J_i$ to array $q$;
6. else if there is only one composition in the $s_i$th row of the
      composition table ($s_i$ is the size of $J_i$), which consists of
      $J_i$ itself
```

```
7. $fail \leftarrow 1$;
8. else
9. In the $s_i$th row (except the composition consisting of $J_i$
      itself), get such a composition $cps(s_i, l)$ in which the
      number of allocated jobs is minimum (if more than
      one composition has the same minimum number,
      select the first found);
10. if the number of allocated jobs in $cps(s_i, l)$ is 0
11. Append the jobs in the composition $cps(s_i, l)$ to array $q$;
12. else
13. Call Algorithm 2 with $(s_i, l)$ and array $q$ as input;
14. if its output is 0
15. $fail \leftarrow 1$;
16. $lc \leftarrow lc + 1$;
17. end while;
18. if $fail$ equals 0
19. return 1;
20. else
21. return 0;
```

Algorithm 3: Allocating jobs to a resource space (*t*, *r*, *cp*) in cluster C_i, where *t* is the time when the space is available, *r* is the size of the space and *cp* is the resource number that the space starts from
Input: the resource space (*t*, *r*, *cp*); an array *q* (used to contain jobs allocated to the resource space).

```
1. $rc \leftarrow r$;
2. while the $rc$th row is NULL
3. $rc \leftarrow rc - 1$;
4. Get such a composition of jobs in which the number of
      allocated jobs is minimum in the $rc$th row (if more
      than one composition has the same minimum number,
      select the one first found; suppose the composition
      found is $cps(r_0, l)$);
5. if the number of allocated jobs in the composition is 0
6. Put the jobs in the composition into array $q$;
7. else
8. Call Algorithm 2 with $(r_0, l)$ and $q$ as inputs;
9. if Algorithm 3 returns 0
10. $rc \leftarrow rc - 1$; Go to Step 2;
11. The starting time of these jobs is $t$; these jobs are
      allocated to the resources in the order specified in
      array $q$, starting from resource $cp$.
```

It can be seen that Algorithm 3 always attempts to identify the jobs that maximally occupy the given resource space. In doing so, the number of resources left idle is minimised. Consequently, the jobs sent to a cluster are packed tightly. The time complexity of Algorithm 3 (including Algorithm 2) 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 $O(p^2 n_{MAX})$, since the worst case involves searching the complete composition table.

4.3 Employing a heuristic to balance performance

In each local cluster, TITAN uses a genetic algorithm to adjust the seed schedule sent by MUSCLE, aiming to further improve the CP. Although MUSCLE has no control over the detailed operations of the genetic algorithm, it analyses the objective factors influencing the performance and allocates different levels of workload to each cluster through a heuristic ensuring the CP can be well balanced among the clusters.

The fundamental attributes of parallel jobs allocated to cluster C_i, include:

- p_i : the number of jobs;
- $etSum_i$: the sum of execution times of all jobs;
- $sizeSum_i$: total job sizes;
- $slkSum_i$: total slacks (the slack of a job is its deadline minus its execution time and its arrival time, which relates to the QoS of jobs).

The fundamental attribute of a cluster, C_i, is the number of resources m_i (the attribute of the service rate has been reflected in the execution times of jobs).

The scheduling performance achieved in a cluster is determined by the resultant forces of these attributes. In addition to this, the packing potential for the parallel jobs allocated to a cluster is also considered a critical factor. This problem, however, is solved in Algorithm 3, which ensures that jobs are selected to maximally occupy a given resource space.

When the value of the attributes of p_i, $etSum_i$ and $sizeSum_i$ in a cluster is higher, the cluster can be allocated more jobs, while the relation is opposite for $slkSum_i$ and m_i. These attributes are integrated to form a new metric (denoted by ε), shown in (3). When multiple clusters offer available resource space, the cluster with the smallest ε is given the highest priority and will be allocated the jobs. When more than one cluster has the same value of ε, the cluster with the greatest size is given the highest priority. In the case of equal sized clusters, one is selected randomly:

$$\varepsilon = \frac{p_i \times etSum_i \times sizeSum_i}{slkSum_i \times m_i} \tag{3}$$

The complete scheduling procedure for MUSCLE is outlined in Algorithm 4.

Algorithm 4: MUSCLE scheduling

```
1. while the expected makespans of the jobs yet to be
      scheduled in all clusters (provided by TITAN) are
      greater than a predefined threshold
2. Collect the arriving jobs in the multicluster;
3. Call Algorithm 1 to construct the composition table for
      the collected jobs;
4. do
5. for each cluster do
6. Calculate $\varepsilon$ using (3);
7. Get the earliest available resource space in cluster $C_i$
      which has the minimal $\varepsilon$;
8. Call Algorithm 3 to allocate jobs to this space;
9. Update the earliest available resource space and the
      attribute values of the workload in $C_i$;
10. while all collected jobs have not been allocated;
11. Go to Step 1;
```

The time of Algorithm 4 is dominated by Step 3 and Step 8 in the do-while loop. Their time complexities have been analysed in Sub-Sections 4.1 and 4.2.

A case study is presented below to illustrate Algorithm 4. The working parameters are listed in Table 1. The corresponding composition table is found in Table 2. Figure 3 shows the evolution of the job scheduling in C_1 and C_2 (*a*, *b* are for C_1; *c*, *d* for C_2). After Algorithm 4 is completed, the seed schedules in both clusters are determined, which are shown in Figs. 3*b*, *d*. It can be seen that these jobs have been packed tightly. These seed schedules will be sent to TITANs situated in C_1 and C_2 for further performance improvement.

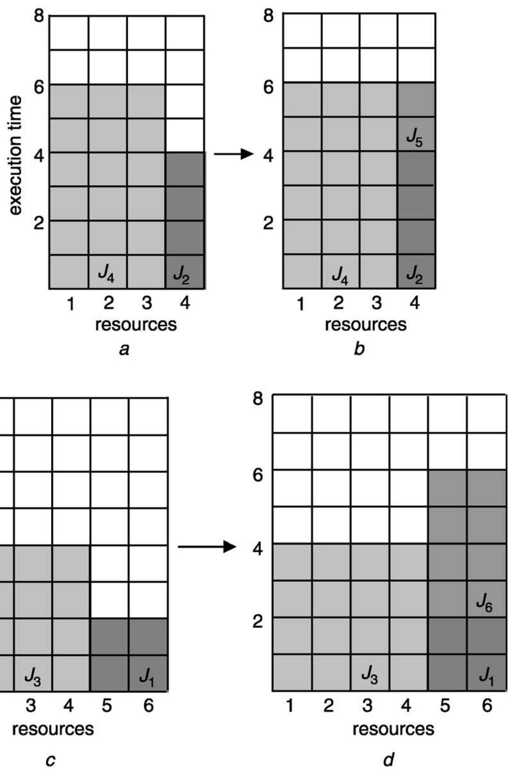

Fig. 3 *Evolution of job scheduling in cluster C_1 and C_2*
a, *b* are for C_1; *c*, *d* for C_2

5 Experimental studies

A simulator is developed to evaluate the performance of the scheduling mechanism in MUSCLE. The presented experimental results focus on demonstrating the performance advantages of the scheduling mechanism in MUSCLE over the traditional scheduling policies frequently used in distributed systems. Weighted random (WRAND) and dynamic least load (DLL) policies are two selected representatives.

In the experiments, the generated parallel jobs are submitted to the multicluster. MUSCLE, DLL or WRAND are used as the multicluster-level scheduling policies, respectively, while in all cases TITAN is used as the cluster-level scheduler in each local cluster. The combination of the weights for the over-deadline, the makespan and the idle-time is denoted by (W^o, W^m, W^i).

The workloads in the experiments are generated as follows. 20 000 parallel jobs are generated; the submissions of parallel jobs follow a Poisson process.

The execution times of the jobs in cluster C_1 follow a bounded Pareto distribution [20], shown in (4), where e_l and e_u are the lower and upper limit of the execution time x. It is shown in [20] that this distribution will represent a more realistic workload model than a simple exponential distribution,

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

If a job is scheduled to another cluster consisting of resources with a service rate s_j, the execution time is determined through multiplying by the ratio between s_l and s_j (i.e. s_l/s_j). A job's deadline d_i is determined by (5), where dr is the deadline ratio. dr follows a uniform distribution in $MIN\_DR, MAX\_DR]$. The range is used to measure the deadline range:

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

The job size can follow different probability distributions according to different application scenarios. Reference [21] studies jobs whose sizes follow uniform or normal distribution, while [22] shows that the probability that a job uses fewer than n processors is roughly proportional to $\log n$. These experiments are conducted under these different distributions to verify the effectiveness of the scheduling mechanism proposed in this paper. The results show similar patterns, and hence only the results for a specific representative distribution are presented in each experiment in this Section.

Dynamic least-load (DLL) is a scheduling policy extensively used in heterogeneous systems [3, 20]. The workload in cluster C_i, denoted as L_i, is computed using (6). When a parallel job is submitted to the multicluster, DLL schedules the job to the cluster whose workload is the least and whose size is greater than the job's size:

$$L_i = \sum_{Ji \in WQ_i} e_i s_i / m_i \quad WQ_i \text{ is the set of jobs allocated to cluster } C_i \tag{6}$$

The weighted random (WRAND) policy is another frequently used scheduling policy in distributed systems [3, 20]. When a job arrives at the multicluster, the WRAND policy first picks out all clusters whose sizes are greater than the job size (suppose these clusters are $C_{i1}, C_{i2}, \ldots, C_{ij}$), and then schedules the job to a cluster $C_{ik} (1 \le k \le j)$ with the probability proportional to its processing capability $(m_{ik} u_{ik})$.

The performance metrics evaluated in the experiments are the mean comprehensive performance (MCP) and performance balance factor (PB). Each time MUSCLE sends seed schedules to TITANs in individual clusters $C_i, 1 \le i \le n$. These TITANs further improve the performance in terms of the CP. When the performance improvement between two generations of schedule sets is less than a threshold, the CP performance for cluster C_i, denoted by CP_i, is recorded. The MCP is the average of $\mathrm{CP}_i, 1 \le i \le n$, calculated by (7), where p_i is the number of jobs allocated to C_i. The procedure continues until all generated jobs are processed. Each point in the performance curve is plotted as the average of the MCP calculated each time:

$$MCP = \sum_{i=1}^{n} CP_i \times \frac{p_i}{p} \tag{7}$$

The PB is defined as the standard deviation of CP_i, computed by (8):

$$PB = \sqrt{\frac{1}{n-1} \sum\nolimits_{i=1}^{n} (CP_i - MCP)^2} \tag{8}$$

5.1 Workload levels

Figures 4*a*, *b* compare the performance of MUSCLE, DLL and WRAND policies under different workload levels. The workload level is measured by the mean job number in the queue in the multicluster, whose setting is listed in Table 3.

It can be seen from Fig. 4*a* that MUSCLE outperforms DLL and WRAND under all workload levels. This is because the jobs are packed tightly in the seed schedules sent by MUSCLE to individual clusters. Therefore, the further improvement in each cluster is based on an excellent seed value. However, the jobs sent by DLL or WRAND to each cluster are random in terms of whether they can be

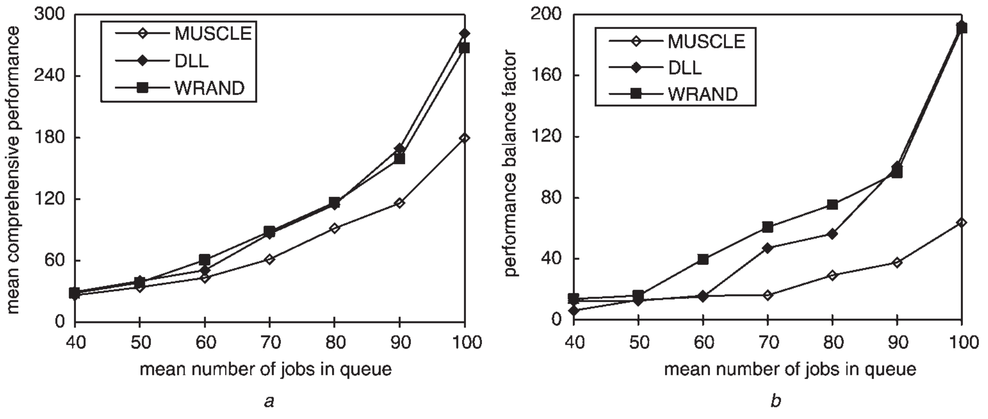

Fig. 4 *Performance comparison of MUSCLE, DLL and WRAND under different work-load levels in terms of mean comprehensive performance (MCP) and performance balance factor (PB)*

$(W^o, W^m, W^i) = (4, 3, 1)$; $e_l/e_u = 5/100$; $MIN\_S/MAX\_S = 1/10$ (uniform distribution); $MIN\_DR/MAX\_DR = 0/5$
a MCP
b PB

Table 3: 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

packed tightly or not. This reduces the possibility of achieving a high MCP in the multicluster.

A further observation from Fig. 4*a* is that the advantage of MUSCLE over other policies becomes increasingly pronounced as the workload increases. When the mean number of the jobs in queue is 40, MUSCLE outperforms DLL in terms of the MCP by 12.1% and outperforms WRAND by 8.4%. When the mean number of jobs is 100, the performance advantages increase to 56.7% and 48.9% compared with DLL and WRAND, respectively. This is because, when the number of jobs in the queue increases, MUSCLE can gather more information regarding parallel jobs and as a result make better allocation decisions among clusters.

As can be observed from Fig. 4*b*, when the mean number of jobs in the queue is less than 60, the PB achieved by MUSCLE is slightly worse than that by DLL. However, MUSCLE significantly outperforms DLL in all other cases. This can be explained as follows. When the workload is low, a small number of jobs miss their deadlines and the MCP is mainly caused by makespan and idle time. Therefore, DLL shows more balanced MCP performance. However, as the workload increases further, more jobs miss their deadlines. DLL ignores the QoS demands of these jobs. In contrast, MUSCLE takes the QoS demands into account so that the MCP performance remains balanced among the clusters when the over-deadline gradually becomes a significant factor in the MCP performance.

5.2 Deadlines

Figure 5 compares the performance of MUSCLE, DLL and WRAND under different deadline ranges, which is measured by the range of *dr* in (5). In Figs. 5*a*, *c*, the mean number job in the queue is 40, while in Figs. 5*b*, *d* this is set to 100.

A general observation from Figs. 5*a*, *b* is that MUSCLE performs better than DLL and WRAND in terms of MCP. Further observations show that the advantages are different under different combinations of workload levels and

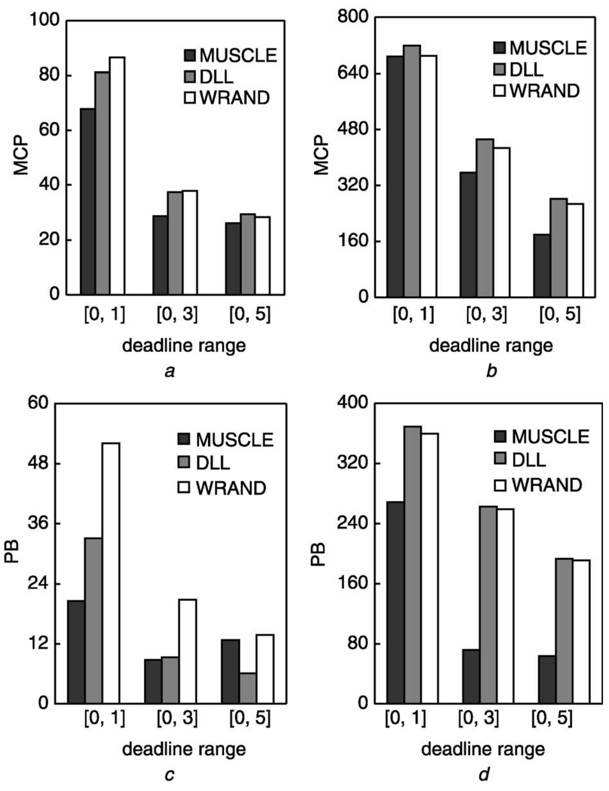

Fig. 5 *Performance comparison of MUSCLE, DLL and WRAND under different deadline ranges*

Mean number job in queue is 40 in (*a*) and 100 in (*b*); mean number of jobs in queue is 40 in (*c*) and 100 in (*d*); $(W^o, W^m, W^i) = (4, 3, 1)$; $e_l/e_u = 5/100$; $MIN\_S/MAX\_S = 1/10$ (uniform distribution); multicluster setting is listed in Table 3

deadlines. When the workload is low and the deadline is loose, the advantage is low. This is because in this case the over-deadline, makespan and idle-time are all low. Hence, the potential of MUSCLE cannot be fully released. When the deadlines become more urgent relative to the workload, the advantage of MUSCLE over DLL and WRAND becomes increasingly prominent. However, when the deadlines become even shorter relative to the workload, the advantage of MUSCLE diminishes. This is because the workload is saturated relative to the urgent deadlines and the finish times of many jobs exceed their deadlines by a large amount. This performance deterioration is due to overloading, and scheduling policies are able to do little in this situation.

As for the performance in terms of PB (shown in Figs. 5*c*, *d*), MUSCLE significantly outperforms DLL and WRAND under all workloads and deadline ranges (except when the workloads are low and the deadlines are loose). This is because, compared with DLL and WRAND, MUSCLE is able to distribute the workload evenly among the multicluster by taking the QoS demands of the jobs into account.

5.3 Different domain-level performance requirements

Figure 6 compares the performance of MUSCLE, DLL and WRAND in terms of MCP and PB when the constituent clusters in the multicluster utilise different weights, shown in Table 3, to calculate the MCP. The experiments are conducted for different mean numbers of jobs in the queue (40, 100 and 160).

As can be observed from Figs. 6*a*, *b* and *c*, the performance of MUSCLE is superior to other policies in all cases in terms of the MCP. Furthermore, the advantage of MUSCLE over DLL and WRAND is different under different workload levels. When the mean number of jobs in the queue is 40, the advantage of MUSCLE over DLL is 15.6%, while the advantage increases to 57.5% when the number of jobs is 100. When the workload increases further and the mean number of jobs in the queue is 160, the advantage of MUSCLE over DLL decreases to 13.7%. These results are consistent with the trends demonstrated in Sub-Section 5.2.

The performance comparison in terms of PB demonstrates a similar pattern to that shown in Figs. 5*c*, *d*. When the mean number of jobs in the queue is 40, the performance of MUSCLE in terms of PB is worse than that of DLL. Under other workloads, MUSCLE performs significantly better than DLL and WRAND. These are also consistent with the results in Figs. 4*b*, 5*c* and *d*.

5.4 Multicluster size

Figure 7 compares the performance of MUSCLE, DLL and WRAND under different multicluster sizes. In this experiment, we use a cluster pool of 10 clusters, where cluster sizes vary from 20 to 56 with increments of 4 and the service rates of all resources are equivalent. Initially, jobs are processed in the multicluster consisting of 4 clusters of the

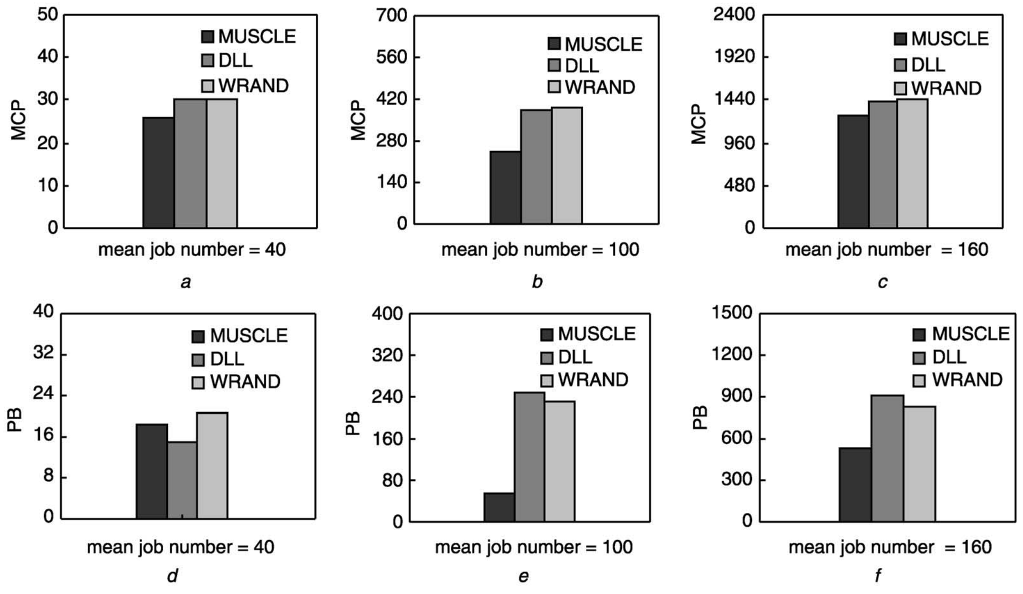

Fig. 6 *Performance comparison of MUSCLE, DLL and WRAND in terms of MCP and PB*

$e_l/e_u = 5/100$; $MIN\_S/MAX\_S = 1/10$ (uniform distribution); $MIN\_DR/MAX\_DR = 0/5$; multicluster setting is listed in Table 3 and weight combinations for C_1–C_4 are (12, 3, 1), (8, 3, 1), (4, 3, 1) and (1, 3, 1)

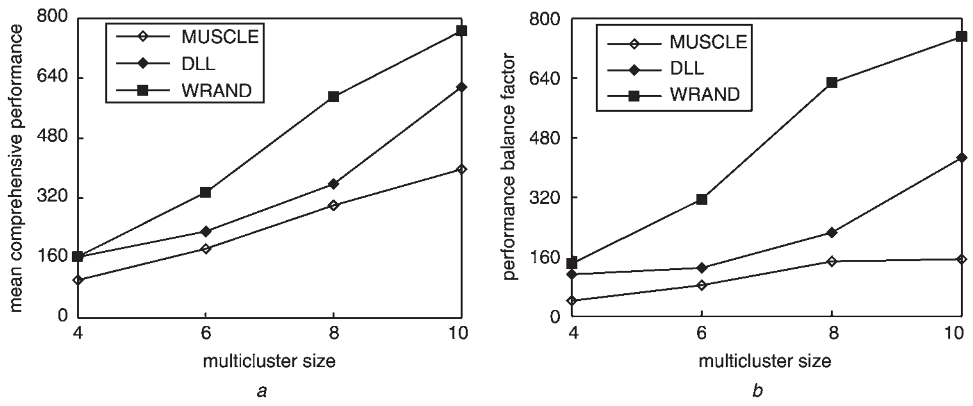

Fig. 7 *Performance comparison as multicluster size changes in terms of MCP and PB*

$(W^o, W^m, W^i) = (8, 3, 1)$; $e_l/e_u = 5/100$; $MIN\_S/MAX\_S = 1/m_{MAX}$ (proportional to $\log n_{MAX}$); $MIN\_DR/MAX\_DR = 0/5$; initial mean number of jobs is 80 and cluster sizes vary in [20, 56]

a MCP
b PB

smallest sizes (20–32); then two clusters of greater sizes are added to the multicluster each time until all 10 clusters are used. The workload level is also increased accordingly, so that the ratio of the mean number of jobs in the queue to the total processing capability remains unchanged. The results in Fig.7 are for the case where the initial mean number of jobs in the queue is 80 (the results for the other workload levels show similar patterns). The probability that a job requests fewer than m_{MAX} resources is proportional to $\log m_{MAX}$ (m_{MAX} is the size of the greatest cluster in the current multicluster).

As can be observed from Fig. 7, MUSCLE outperforms DLL and WRAND in terms of MCP and PB in all cases. A further observation is that the advantage of MUSCLE over WRAND becomes more pronounced as the cluster size increases, while the trend is not obvious for the case of DLL. This can be explained as follows. WRAND allocates the fixed proportion of workload to each cluster. As the number of clusters increases, this policy becomes increasingly incompetent. DLL considers both the workload and processing capability of each cluster. When the jobs sent by DLL to a cluster happen to have a good packing potential and their deadlines are not (on average) urgent, comparatively high performance can be achieved. However, DLL does not provide such a general scheme as that implemented in MUSCLE to take into account the packing potential and QoS requirements of jobs.

6 Conclusions

A multicluster-level scheduler, called MUSCLE, is described in this paper for the scheduling of parallel jobs with QoS demands in multiclusters, in which the constituent clusters may be located in different administrative organisations. Three metrics (over-deadline, makespan and idle-time) are combined with variable weights to evaluate the scheduling performance. MUSCLE is able to allocate jobs with high packing potential to the same cluster and further utilises a heuristic to control the workload distribution among the clusters. Extensive experimental studies are carried out to verify the performance advantages of MUSCLE. The results show that, compared with the traditional scheduling policies in distributed systems, the comprehensive performance (in terms of over-deadline, makespan and idle-time) is significantly improved and the jobs are well balanced across the multicluster.

7 Acknowledgments

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).

8 References

1 Barreto, M., Avila, R., and Navaux, P.: 'The MultiCluster model to the integrated use of multiple workstation clusters'. Proc. 3rd Workshop on Personal Computerbased Networks of Workstations, Cancun, Mexico, 2000, pp. 71–80
2 Buyya, R., and Baker, M.: 'Emerging technologies for multicluster/grid computing'. Proc. 2001 IEEE Int. Conf. on Cluster Computing, CA, USA, 2001
3 He, L., Jarvis, S.A., Spooner, D.P., and Nudd, G.R.: 'Optimising static workload allocation in multiclusters'. 18th IEEE Int. Parallel and Distributed Processing Symposium (IPDPS), New Mexico, USA, 2004
4 Shmueli, E., and Feitelson, D.G.: 'Backfilling with lookahead to optimize the performance of parallel job scheduling'. Proc. 9th Conf. on Job Scheduling Strategies for Parallel Processing, Seattle, WA, USA, 2003, pp. 228–251
5 He, X., Sun, X., and Laszewski, G.: 'QoS guided min-min heuristic for grid task scheduling', *J. Comput. Sci. Tech.*, 2003, **18**, (4)
6 Lawson, B.G., and Smirni, E.: 'Multiple-queue backfilling scheduling with priorities and reservations for parallel systems'. 8th Conf. on Job Scheduling Strategies for Parallel Processing, Edinburgh, UK, 2002, pp. 72–87
7 Mu'alem, A.W., and Feitelson, D.G.: 'Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling', *IEEE Trans. Parallel Distrib. Syst.*, 2001, **12**, (6), pp. 529–543
8 Talby, D., and Feitelson, D.G.: 'Supporting priorities and improving utilization of the IBM SP2 scheduler using slack-based backfilling'. Proc. 13th Int. Parallel Processing Symp., 1999, pp. 513–517
9 Spooner, D.P., Jarvis, S.A., Cao, J., Saini, S., and Nudd, G.R.: 'Local grid scheduling techniques using performance prediction', *IEE Proc. Comput. Digit. Tech.*, 2003, **150**, (2), pp. 87–96
10 Cao, J., Kerbyson, D.J., and Nudd, G.R.: 'Performance evaluation of an agent-based resource management infrastructure for grid computing'. Proc. 1st IEEE/ACM Int. Symp. on Cluster Computing and the Grid, Brisbane, Australia, 2001, pp. 311–318
11 Cao, J., Spooner, D.P., Jarvis, S.A., and Nudd, G.R.: 'Grid load balancing using intelligent agents', *Future Gener. Comput. Syst.*, special issue on Intelligent Grid Environments: Principles and Applications, 2004, (accepted for publication)
12 Casanova, H., Obertelli, G., Berman, F., and Wolski, R.: 'The AppLeS parameter sweep template: user-level middleware for the Grid'. Proc. Super Computing Conf. (SC), Texas, USA, 2000
13 He, L., Jarvis, S.A., Bacigalupo, D., Spooner, D.P., Chen, X., and Nudd, G.R.: 'Queueing network-based optimisation techniques for workload allocation in clusters of computers'. IEEE Int. Conf. on Services Computing (SCC), Shanghai, China, (accepted for publication)
14 He, L., Jarvis, S.A., Spooner, D.P., and Nudd, G.R.: 'Dynamic scheduling of parallel real-time jobs by modelling spare capabilities in heterogeneous clusters'. Proc. IEEE Int. Conf. on Cluster Computing (Cluster), Hong Kong, 2003, pp. 2–10
15 He, L., Jarvis, S.A., Spooner, D.P., and Nudd, G.R.: 'Dynamic, capability-driven scheduling of DAG-based real-time jobs in heterogeneous clusters', *Int. J. High Perform. Comput. Netw.*, 2004, (3), (accepted for publication)
16 Raman, R., Livny, M., and Solomon, M.: 'Matchmaking: distributed resource management for high throughput computing'. Proc. 7th IEEE Int. Symp. on High Performance Distributed Computing, Chicago, IL, USA, 1998
17 Dogan, A., and ÖzgünerI, F.: 'On QoS-based scheduling of a meta-task with multiple QoS demands in heterogeneous computing'. Int. Parallel and Distributed Processing Symp., Florida, USA, 2002
18 Nudd, G.R., Kerbyson, D.J., Papaefstathiou, E., Harper, J.S., Perry, S.C., and Wilcox, D.V.: 'PACE: a toolset for the performance prediction of parallel and distributed systems', *Int. J. High Perform. Comput. Appl.*, 2000, **14**, (4), pp. 228–251
19 Liu, Z., Squillante, M.S., and Wolf, J.L.: 'On maximizing service-level-agreement profits'. Proc. 3rd ACM Conf. on Electronic Commerce, Florida, USA, 2001, pp. 213–223
20 Tang, X.Y., and Chanson, S.T.: 'Optimizing static job scheduling in a network of heterogeneous computers'. Proc. 29th Int. Conf. on Parallel Processing, Toronto, Canada, 2000, pp. 373–382
21 Yu, C., and Das, C.R.: 'Limit allocation: an efficient processor management scheme for hypercubes'. Proc. Int. Conf. Parallel Processing (ICPP), North Carolina, USA, 1994, pp. 143–150
22 Feitelson, D.G., and Rudolph, L.: 'Metrics and benchmarking for parallel job scheduling'. 4th Workshop on Job Scheduling Strategies for Parallel Processing, Florida, USA, 1998, pp. 1–24
23 Cao, J., Kerbyson, D.J., Papaefstathiou, E., and Nudd, G.R.: 'Performance modeling parallel and distributed computing using PACE'. Proc. 19th IEEE Int. Performance, Computing, and Communications Conf., Arizona, USA, 2000, pp. 485–492