Mapping DAG-based Applications to Multiclusters with Background Workload [*]

Ligang He, Stephen A. Jarvis, Daniel P. Spooner, David Bacigalupo, Guang Tan
and Graham R. Nudd
Department of Computer Science, University of Warwick
Coventry, CV4 7AL, United Kingdom
liganghe@dcs.warwick.ac.uk

Abstract

Before an application modelled as a Directed Acyclic Graph (DAG) is executed on a heterogeneous system, a DAG mapping policy is often enacted. After mapping, the tasks (in the DAG-based application) to be executed at each computational resource are determined. The tasks are then sent to the corresponding resources, where they are orchestrated in the pre-designed pattern to complete the work. Most DAG mapping policies in the literature assume that each computational resource is a processing node of a single processor, i.e. the tasks mapped to a resource are to be run in sequence. Our studies demonstrate that if the resource is actually a cluster with multiple processing nodes, this assumption will cause a misperception in the tasks' execution time and execution order. This will disturb the pre-designed cooperation among tasks so that the expected performance cannot be achieved. In this paper, a DAG mapping algorithm is presented for multicluster architectures. Each constituent cluster in the multicluster is shared by background workload (from other users) and has its own independent local scheduler. The multicluster DAG mapping policy is based on theoretical analysis and its performance is evaluated through extensive experimental studies. The results show that compared with conventional DAG mapping policies, the new scheme that we present can significantly improve the scheduling performance of a DAG-based application in terms of the schedule length.

1. Introduction

Clusters are becoming popular platforms for the processing of scientific and commercial applications. Multiple separate clusters can be further interconnected to obtain multicluster computing architectures (or grids) [8]. These constituent clusters may locate within a single organization or across wide geographical sites [2][9].

In this paper, a DAG mapping algorithm is presented for multicluster architectures. Each constituent cluster is shared by background workload (from other users) and has its own independent local scheduler. Such an architecture is often encountered in grid environments.

Studies on the mapping of DAG-based applications to heterogeneous systems have received a good deal attention [4][5][6][7][10]. Most DAG mapping algorithms can be classified into two categories: list scheduling [5][7] and graph partitioning [1][4][16].

A list scheduling algorithm gathers all current schedulable tasks (a task is schedulable if all of its parents have completed execution or it has no parent) and maps each task to a suitable resource according to a certain policy; this policy differentiates the list scheduling algorithms from one another. A large number of list scheduling algorithms for heterogeneous systems have been presented in the literature [3][5][7][11][14][15]. However, they are not suitable for solving the scheduling problems in the scenario considered in this paper.

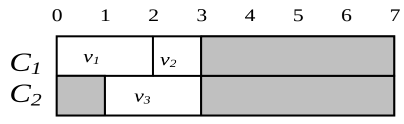
First, these previous approaches consider each resource as a single processor, and tasks are mapped to every single resource in the heterogeneous system. In this paper, however, a cluster consisting of multiple processing nodes has its own independent local scheduler and the DAG mapping algorithm cannot specify the processing node which a task should be mapped to.

Second, if a cluster is regarded as a single resource to which tasks are allocated, these algorithms assume that the tasks mapped to each resource are to be run in sequence. This assumption may cause a misperception in the tasks' execution time and execution order [12]; this is illustrated in this paper through a supportive case study.

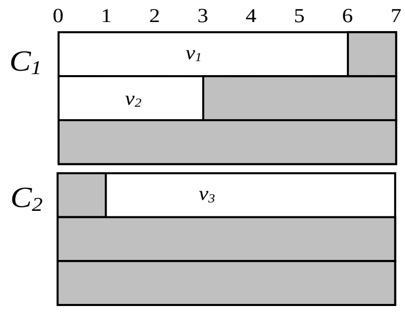
Finally, in the scheduling scenario presented here, background workload and the tasks from the DAG compete for the cluster resources. This complicates the mapping design and makes the list scheduling algorithms presented in the literature even less effective.

* 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).

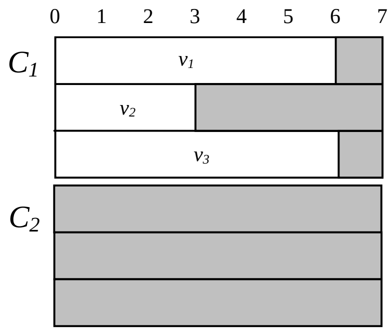
A case study is described to illustrate the misperception caused by regarding a cluster as a single processing node. Suppose three tasks v_1, v_2, v_3 are schedulable. Their computational volume is 6, 3 and 6 respectively. Now consider a multicluster consists of two clusters C_1 and C_2, each with 3 processing nodes, and assume the service rate of each processing node is 1. We also assume that mapping a task to C_2 incurs the inter-cluster communication cost of 1 time unit.



(a) Expected mapping and execution pattern



(b) Actual mapping and execution



(c) Optimal mapping and execution

Fig.1. A case study for the misperception caused by regarding each cluster as a single processing node.

If each cluster is regarded as a single processing node, whose service rate is 3, the execution time of the three tasks (the execution time equals the computational volume divided by the service rate) will be 2, 1 and 2, respectively.

From this perspective, the optimal mapping and execution of these three tasks should be as in Fig.1.a, where v_3 starts from time 1 due to the inter-cluster communication delay. The expected schedule length is 3 (the schedule length is the duration from the time when the first task starts running to the time when all tasks are completed). However, if these three tasks are mapped as in Fig.1.a, their actual execution will be as in Fig.1.b, if both C_1 and C_2 consist of 3 processing nodes with the service rate of 1. As a result, the actual schedule length is 7. Moreover, the algorithm expects that v_2 finishes first, followed by v_1 and v_3. In the actual execution, however, the order in which the tasks finish is v_1, v_2 and v_3.

Considering the parallelism provided by the resource C_1, the optimal mapping and corresponding execution should be as in Fig.1.c, where the schedule length is 6 and is therefore better than that seen in Fig.1.b.

Graph partitioning adopts another approach to mapping DAGs to heterogeneous systems. It analyses the DAG's topology and partitions the graph into several sub-graphs according to a certain metric (e.g., the least amount of communication costs among the different sub-graphs). Each sub-graph is then mapped to a resource for execution. Graph partitioning policies have also been presented in a number of research papers [1][4][16]. However, these graph partitioning algorithms also assume that the sub-graph can be mapped to a specified resource, and that the tasks in the sub-graph will be run in sequence, which is not necessary the case for the scheduling scenariosconsidered in this paper. Furthermore, these graph partitioning algorithms do not take into account background workload on the resources.

The multicluster DAG mapping algorithm presented here can be categorized as a list scheduling algorithm. The algorithm aims to achieve the optimised schedule length, which is defined as the duration between the time when the first task starts and the time by which all tasks are completed. It performs the mapping operation before the application starts running. After mapping, the tasks to be run in each cluster are determined. The tasks are then sent to the corresponding clusters where they are further scheduled by a local scheduler together with any background workload.

An approach is developed in this paper to compute the finish time of a task in a cluster with background workload. An admission control mechanism is also introduced to map as many tasks as possible to the same cluster until a pre-defined condition is broken. The goal for this is to fully utilize the parallel processing capability, and at the same time reduce the inter-cluster communication.

The rest of this paper is organised as follows. Section 2 presents the system and workload model. The multicluster DAG mapping algorithm is proposed in Section 3. A performance evaluation is conducted in Section 4. Finally, Section 5 concludes the paper.

2. System and workload model

An application consists of a set of tasks with precedence constraints, which is modelled as a Directed Acyclic Graph (DAG). A DAG is denoted as $J=\{V, E\}$, where $V=\{v_1, v_2,\dots, v_r\}$, which defines r tasks that constitute the application. α_p is denoted as v_p's computational volume; E represents the communication relationship and the precedence constraints among tasks; $e_{pq}=(v_p, v_q)\in E$ represents a message sent from task v_p to v_q and it also suggests v_q can start running only after v_p is complete and v_q receives message e_{pq}; v_p is called v_q's parent and β_{pq} is denoted as message e_{pq}'s volume. It is assumed in this paper that a DAG has only one entry task, which has no parents, and one exit task, which has no children.

The multicluster consists of n clusters, $C_1, C_2, \ldots, C_n$; where a cluster C_i ($1\le i\le n$) consists of m_i homogeneous processing nodes (the size of cluster C_i is m_i), each with a service rate of u_i. The nodes in the different clusters may have different performance. The processing nodes in a homogeneous cluster are connected by an intra-cluster network. In cluster C_i, each communication link among processing nodes is weighted l_i, which models the time it takes to transfer one unit of a message between two processing nodes. Two clusters are connected by an inter-cluster communication network. The communication link between cluster C_i and C_j is weighted w_{ij}.

It is assumed that a local scheduler is located at each cluster and adopts a centralised queueing architecture, i.e., a single waiting queue is used by each local scheduler to accommodate the DAG-based application and the workload from other users (called the background workload). The mean arrival rate of background workload arriving at cluster C_i is λ_i and their mean task size is δ_i. Each local scheduler schedules the tasks in the waiting queue based on a First-Come-First-Served policy; where the task at the head of the queue is sent to a free processing node for execution.

All tasks placed in the waiting queue are ready to be executed. When a set of tasks with precedence constraints (including the tasks in the DAG-based application) arrive, these tasks are first placed into a schedule queue. A task is placed into the waiting queue as soon as it becomes schedulable (a task is considered schedulable if all of its parents have finished and the task has received all messages from its parents).

3. DAG mapping algorithm for multiclusters

In this section, an approach is presented to predict the finish time of a task in the DAG. An admission control mechanism is also introduced to exploit the parallel processing capabilities of the resources and guarantee that a cluster does not become overloaded. The DAG mapping algorithm for multiclusters is consequently presented.

3.1 Calculating a task's finish time in a cluster

Many list scheduling algorithms need to know a task's start/finish time to make scheduling decisions [3][5][7][11]. If the tasks are scheduled to every single processing node, the start/finish time of a task is easy to compute since the tasks at the processing node are run in sequence, which is the case for the list algorithms in the literature. In this paper, however, the tasks mapped to a cluster are further scheduled by the local scheduler. It is therefore a non-trivial task to compute the start/finish time in a cluster. The task is further complicated by the presence of background workload. Theoretical analysis is conducted and a new approach is developed in this subsection to predict a task' finish time in a cluster with background workload.

Suppose the tasks currently allocated to cluster C_i are ordered in increasing time when they become schedulable. The sequence of tasks is denoted as $<v_{i1}, \ldots v_{ik}>$.

sum_et[*i*][*j*] and *sum_msg*[*i*][*j*] are the sums of the execution time and the communication time of tasks $v_{i1}, \ldots v_{ij}$ ($1\le j\le k$) in cluster C_i. *idle_cap*[*i*][*j*] is the idle processing capability (measured by time unit) before the time when task v_{ij} becomes schedulable in cluster C_i.

The algorithm for computing the finish time of a task in cluster C_i is shown in Algorithm 1.

Algorithm 1. Computing the finish time of task v_p, denoted as $ft(v_p)$, in cluster C_i
Input: array *sum_et*, *sum_msg*, *idle_cap* and j (task v_{ij} is the last task in the sequence of tasks $<v_{i1}, \ldots v_{ik}>$ that becomes schedulable before v_p in C_i)
Output: $ft(v_p)$

```
1.  get the maximum of the finish time of $v_p$'s parents, denoted as $ft\_par$;
2.  if($\lambda_i\delta_i\times ft\_par+sum\_et[i][j]\le m_i u_i\times(ft\_par-\frac{sum\_msg[i][j]}{l_i})-idle\_cap[i][j]$)
3.      $st(v_p)=ft\_par+\max\{\frac{\beta_{qp}}{bwid} \mid v_q\ is\ v_p's\ parents\}$,
        where $bwid$ is $l_i$ if $v_q$ is scheduled to $C_i$ and $bwid$ is $w_{ji}$ if $v_q$ is scheduled to $C_j$ ($i\ne j$);
4.  else
5.      $st(v_p)=ft\_par+(\lambda_i\delta_i\times ft\_par+sum\_et[i][j]-m_i u_i\times(ft\_par-\frac{sum\_msg[i][j]}{l_i})+idle\_cap[i][j])/m_i u_i+\frac{\max(\beta_{qp} \mid v_q\ is\ v_p's\ parents)}{l_i}$;
6.  endif
7.  $ft(v_p)=st(v_p)+\frac{\alpha_p}{u_i}$;
```

Algorithm 1 is analyzed as follows. It computes C_i's total computing capabilities (i.e., total computational volume that C_i is able to finish) and the idle computing capabilities (because the system utilization is less than 1) during the period between the time when starting mapping the DAG-based application and the time when task v_p becomes schedulable. The algorithm also calculates the newly generated workload (including the tasks from the DAG and background workload) during this period. If the generated workload is less than the available computing capabilities of C_i, which is the difference between C_i's total and idle computing capabilities, the workload can be completed in a timely fashion and therefore, task v_p's actual start time equals its earliest start time, which is the time when v_p becomes schedulable. Otherwise, the unfinished workload will queue (in the waiting queue) so as to delay the start of task v_p.

When computing the generated workload in Algorithm 1, all tasks from the DAG which become schedulable before task v_p are counted. The calculation is correct for the following reason. The local scheduler in each cluster places a task in the DAG into the central waiting queue as soon as it becomes schedulable. Therefore, task v_p's actual start time will be later than those tasks that become schedulable before v_p (although v_p's finish time may not be later since the tasks are run in parallel).

3.2 Admission control mechanism

Since the inter-cluster communication is less efficient than the intra-cluster communication, the multicluster DAG mapping algorithm tries to map as many tasks as possible to the same cluster until the capacity of the cluster is reached. This policy can also make full use of the parallel processing capabilities in a cluster.

In this paper, an admission control mechanism is introduced to ensure that a cluster's parallel processing capability can be effectively exploited, while at the same time ensuring that the cluster is not overloaded in terms of a particular metric. The metric used here is the *expected schedule length* of a DAG. The expected schedule length is changeable and it may be updated throughout the mapping procedure.

The multicluster mapping algorithm maps as many tasks as possible to a cluster as long as the current expected schedule length is not exceeded. If the expected schedule length cannot be met, the mapping algorithm seeks to find another cluster which can do so. If no cluster can be found, the task is mapped to the cluster which offers the smallest excess and the expected schedule length is then updated.

The initial value of the expected schedule length is important. If the initial value is set too high, the admission control mechanism will not be effective, while a cluster's parallel processing capability cannot be fully exploited if the value is set too low.

The lower bound of a DAG's schedule length is the longest path in the DAG (called the critical path). When only the tasks in the critical path are submitted to a cluster with background workload, their schedule length can be viewed as the lower bound of the DAG's schedule length in that cluster.

Suppose a DAG's critical path consists of k tasks, v_1^c, v_2^c, …, v_k^c. If these k tasks are submitted to be run in cluster C_i, then their schedule length SL can be computed as in Eq.1, where W_0 is the mean waiting time encountered by the first task in the critical path.

$$SL=\begin{cases}\sum_{p=1}^{k-1}\left(\frac{\alpha_p}{u_i}+\frac{\beta_{p,p+1}}{l_i}\right)+\frac{\alpha_k}{u_i} & \lambda_i\delta_i\le(m_i-1)u_i \quad (a)\\ W_0+\left(\frac{\lambda_i\delta_i-(m_i-1)u_i}{m_iu_i}+1\right)\sum_{p=1}^{k-1}\left(\frac{\alpha_p}{u_i}+\frac{\beta_{p,p+1}}{l_i}\right)+\frac{\alpha_k}{u_i} & \lambda_i\delta_i>(m_i-1)u_i \quad (b)\end{cases} \qquad (1)$$

The equation is explained as follows. At any one time, only one task in the critical path can be executed because of precedence constraints. Hence the remaining (m_i-1) processing nodes are still available for processing background workload. Suppose task v_k is running. If the background workload is low, all background workload arriving during v_k's execution can be completed by these (m_i-1) processing nodes. Therefore, v_k's child can be executed as soon as it becomes schedulable. This is the case for Eq.1.a. However, if the background workload is too high to be completed in a timely fashion, the tasks will be queued in the waiting queue so as to delay the execution of v_k's child. This is the case for Eq.1.b. Eq.1 also gives the conditions for differentiating between these two cases.

In Eq.1.b, W_0 is the mean waiting time encountered by the first task in the critical path. There are two approaches for obtaining the value of W_0. First, if the arrival process of the background workload follows a certain probability distribution (e.g., Poisson process), the value of W_0 can be obtained through theoretical analysis (for example, if background workload follows the Poisson arrival, W_0 is calculated as in [13]). Second, W_0 can be computed by gathering the workload information in the cluster just before mapping the DAG.

The lower bound of the DAG is set to be the initial value of the expected schedule length. This is reasonable since it is possible that the actual schedule length of the DAG is just the lower bound in a cluster.

In order to meet the initial expected schedule length, any task v_p in the DAG has a latest finish time $lt(v_p)$ (because of precedence constraints), which is calculated in Eq.2, where $et'(v_p)$ is the earliest finish time of task v_p, which itself is a task in the critical path. $et'(v_p)$ can be calculated when computing the lower bound of the schedule length of the DAG using Eq.1.

$$lt(v_p)=\begin{cases}et'(v_p) & v_p\ \textit{is a task in the critical path}\\ \max\{lt(v_q)-\frac{\alpha_q}{u_i}-\frac{\beta_{pq}}{l_i}\mid v_q\ \textit{is}\ v_p\textit{'s child}\} & \textit{otherwise}\end{cases} \qquad (2)$$

If a task's actual finish time is after its latest finish time, the current expected schedule length cannot be met. The actual finish time can be calculated using Algorithm 1. The new value of the expected schedule length is updated using its current value plus the excess of the task's actual finish time over its latest finish time. The new value of the latest finish time of each remaining un-mapped task is also updated using its current value plus the excess. The feasibility of these calculations is shown in Theorem 1 (the proof is omitted).

Theorem 1. Suppose task v_p's latest finish time is $lt(v_p)$ and the expected schedule length of the DAG is sl. If v_p's actual finish time is $lt(v_p)+a$ ($a>0$), then the earliest possible schedule length of the DAG is $sl+a$, and in order to meet this earliest possible schedule length, the latest finish time of an arbitrary task v_q is $lt(v_q)+a$.

When task v_p is allocated to a cluster, v_p may become schedulable before some of the tasks that have been previously allocated to the cluster. Hence the execution of these tasks may be delayed since a cluster has its independent local scheduler and task v_p will be placed into the waiting queue before those tasks. Hence, when the current expected schedule length of the DAG is computed, the impact of v_p on other previously allocated tasks should be taken into account.

The multicluster mapping algorithm is outlined in Algorithm 2. Its time complexity is $O((n(r+1)r/2+3r+g)$, where n is the number of clusters, r is the number of tasks and g is the number of edges in the DAG.

Algorithm 2. The multicluster DAG mapping algorithm

```
1.  calculate the lower bound of the critical path in each
    cluster using Eq.1;
2.  calculate tasks' latest finish time using Eq.2;
3.  obtain the cluster with the least lower bound of the
    schedule length, suppose it is $C_1$;
4.  initialize all emements in arrays $sum\_et$, $sum\_msg$
    and $idle\_cap$ to zero;
5.  $i$=1; $excess=0$;
6.  if there are schedulable tasks
7.      get task $v_p$, which becomes schedulable first in
        cluster $C_i$ among all current unallocated and schedul-
        able tasks;
8.      $count$=1; $T=\{C_i\}$;
9.      while($count \le n$)
10.         get the greatest $j$ so that $v_{ij}$ becomes schedul-
            able before $v_p$;
11.         call Algorithm 1 to compute $ft^i(v_p)$;
12.         call Algorithm 1 to compute $ft^i(v_{iq})$, $1 \le q \le j$;
13.         $excess_i=\max(ft^i(v_s)-(lt(v_s)+excess))$,where
            $v_s \in \{v_p\} \cup \{v_{i1},\ldots, v_{ij}\}$;
14.         if($excess_i<0$)
15.             task $v_p$ is mapped to $C_i$;
16.             break;
17.         else
18.             get such $C_k$ that the bandwidth between $C_k$
                and $C_i$ is the highest and $C_k$ is not in $T$;
19.             $i=k$; count++; $T=T \cup \{C_k\}$;
20.     end while
21.     if($count>n$)
22.             obtain the cluster $C_k$ with the least $excess_i$, $v_p$
                is mapped to $C_k$;
23.             $i=k$; $excess$+=min{$excess_i$};
24.     insert task $v_p$ into the task sequence $<v_{i1},\ldots, v_{ik}>$
        and update arrays $sum\_et$, $sum\_msg$ and $idle\_cap$;
25.     update schedulable tasks;
26.     go to Step 6;
```

4. Experimental Studies

An experimental simulator has been developed to evaluate the performance of the Multicluster DAG Mapping algorithm (denoted as MDM) presented in this paper. The experiments are conducted under a wide range of system configurations and workload levels.

The multicluster consists of a collection of clusters; and the number of processing nodes in each cluster is uniformly chosen between *MIN_M* and *MAX_M*. In every cluster a central computer acts as the local scheduler and schedules workload on a First-Come-First-Served basis.

In a DAG, task v_p's execution time on cluster C_i is calculated as α_p / u_i. Similarly, message e_{pq}'s communication time is β_{pq} / l_i, if both v_p and v_q are scheduled to C_i; otherwise, the communication time is β_{pq} / w_{ij}, if v_p is scheduled to C_i and v_q to C_j. Cluster C_i's service rate is uniformly chosen between *MIN_U* and *MAX_U*. This range reflects the level of computational heterogeneity. The bandwidth of a intra-cluster network is uniformly chosen between *MIN_L* and *MAX_L*. This range reflects the level of communicational heterogeneity in the clusters. The ratio of the bandwidth of a inter-cluster network to the mean bandwidth of all local networks is uniformly chosen between *MIN_W* and *MAX_W*.

In the experiments, a DAG has a randomly generated topology with a given number of tasks. The number of a task's children is uniformly chosen between *MIN_CH* and *MAX_CH*, which reflects the degree of parallelism. The greater the value of *MAX_CH*, the more tasks in the DAG can potentially be run in parallel. A task's computational volume is uniformly chosen between *MIN_CV* and *MAX_CV* and the volume of a message among tasks is uniformly chosen between *MIN_ MV* and *MAX_MV*.

In the experimental studies, the background workload (sequential tasks) arrives at cluster C_i following a Poisson process and the workload's computational volume follows an exponential distribution. The average arrival rate is uniformly chosen from *MIN_ARV* and *MAX_ARV* and the mean workload volume from *MIN_VOL* and *MAX_VOL*. The system utilization (*SU*) provided by the background workload for cluster C_i is used as the metric for measuring the background workload level in C_i.

The values of the simulation parameters are given in Table 1 unless otherwise stated.

Table 1. Parameters for the simulation studies

Parameter	Explanation	Value
MAX_U/MIN_U	Max/min service rate	1.0/0.2
MAX_M/MIN_M	The number of processing nodes in a cluster	16/4
MAX_L/MIN_L	Max/min bandwidth for intra-cluster network	1.0/0.5
MAX_W/MIN_W	Max/min ratio of intra- to inter-cluster bandwidth	10.0/1.0
MAX_CV/MIN_CV	Max/min computation volume	25/1
MAX_MV/MIN_MV	Max/min message volume	5/1
SU	Utilization by background workload	0.5
TASKNUM	Task number in a DAG	60
Multicluster size	The number of clusters in the multicluster	4
MAX_CH/MIN_CH	Max/min degree of parallelism used to generate a DAG	16/1

The performance metric evaluated in these experiments is the schedule length of a DAG. The experimental results demonstrate the performance advantages of the multicluster DAG mapping algorithm over the scheduling policies that regard each resource as a single processor. The DAG scheduling algorithm presented in [7] (denoted as SDS) is selected as a representative. The SDS also aims to reduce a DAG's schedule length. It schedules a task to the single-processor node that is able to offer the shortest response time in a heterogeneous cluster.

In the experimental studies, the SDS algorithm regards a cluster as a single processor node, whose service rate is the total service rate of all processing nodes in the cluster. The schedule length of a DAG is also obtained by scheduling all tasks to the same cluster, that is the one with the greatest value of ($m_i u_i$-$\lambda_i \delta_i$). The schedule length is used as a base line to measure the extent to which the performance is improved by the multicluster DAG mapping algorithm presented in this paper.

4.1 Degree of parallelism

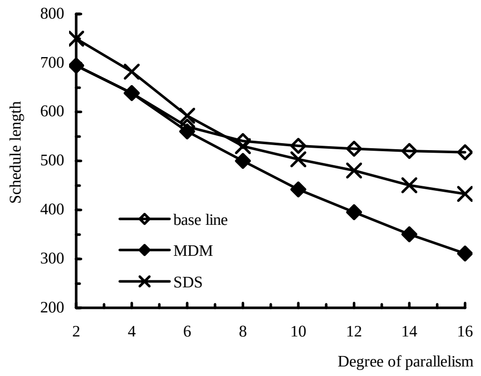


Fig.2. Performance comparison under the increasing degree of parallelism

The degree of parallelism in a DAG determines whether its tasks can be effectively run in parallel. Fig.2 shows the impact of the degree of parallelism on the schedule length under the different scheduling policies.

The first observation from Fig.2 is that the schedule length of the DAG decreases as its degree of parallelism increases, as is to be expected. As can be observed further from Fig.2, the multicluster DAG mapping algorithm (MDM) achieves the same performance as the base line when the degree of parallelism is low (from 2 to 4). This is because when the degree of parallelism is low, MDM also schedules all tasks in the DAG to the same cluster. This is verified by our experimental results. However, as the degree of parallelism increases further, the MDM schedule increasingly allocates more tasks to the other clusters so that the tasks are effectively run using a higher degree of parallelism. This reduces the schedule length of the DAG.

Another interesting observation is that the schedule length achieved by the SDS algorithm is worse than that achieved by scheduling all tasks to the same cluster (the base line) when the degree of parallelism in the DAG is low (from 2 to 6). This result can be explained as follows. The SDS algorithm regards a cluster as a single processor node and in so doing assumes that all tasks scheduled to a cluster will be run sequentially. Because of this, the algorithm may schedule some tasks to different clusters to achieve a higher degree of parallelism. However, the parallelism is achieved at the expense of higher communication costs (inter-cluster communication). If these tasks can be scheduled to the same cluster, they can be run in parallel with lower communication costs (intra-cluster communication).

It can be observed from Fig.2 that MDM outperforms SDS significantly in all cases. This is because that the MDM algorithm takes into account the parallel processing capability of a cluster and calculates the impact of background workload with a greater degree of sensitivity.

4.2 Background workload

Fig.3 shows the impact of the background workload on the schedule length. The level of background workload is measured by the observed system utilization. Each cluster is presented with the same level of background workload in the experiments.

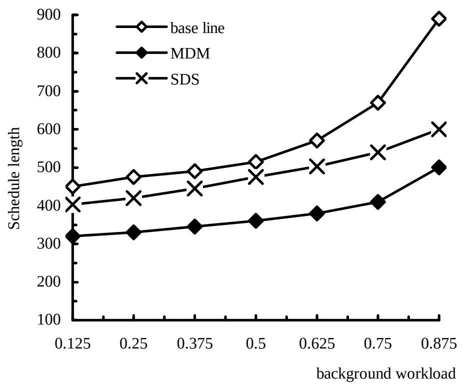


Fig.3. The impact of the background workload on the schedule length

It can be seen from Fig.3 that the schedule length increases as the background workload increases, as is to be expected. However, the increase ranges are different for different policies. The curve for the base line is the sharpest, while the other two curves are relatively even. This can be explained as follows. The background workload and the tasks from the DAG compete for the resources. As the background workload in a cluster increases, it becomes increasingly difficult for the DAG tasks to find enough free processing nodes so as to be run in parallel. Hence, the performance of the base line deteriorates sharply. However, the MDM and SDS algorithm can schedule the tasks to different clusters so as to gain a greater chance of being run in parallel.

As can be observed from Fig.3, MDM performs significantly better than SDS under all levels of background workload. This result again shows the benefit of developing this new DAG mapping algorithm for multiclusters.

4.3. Task and message volume

Fig.4 shows the impact of the ratio of the task volume to message volume on the schedule length. The task volume is uniformly chosen from the range [*MIN_CV*, *MAX_CV*] and the task volume from [*MIN_MV*, *MAX_MV*]. In the experiments, both *MIN_CV* and *MIN_MV* are fixed to be 1. The *MAX_CV*/*MAX_MV* ratio varies from 25/5 to 5/25, which indicates that the task volume in the DAG decreases and the message volume increases while their volume sum remains unchanged.

It can be observed from Fig.4 that the schedule length decreases as the task-volume/message-volume ratio increases in all cases. This may be caused by the fact that the tasks from the DAG compete for computational resources with the background workload. As the task volume decreases, the competition is gradually moderated so that the schedule length is improved.

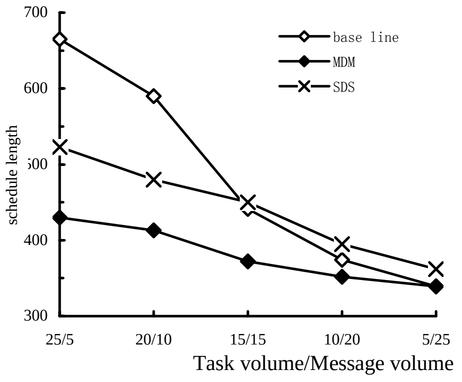


Fig.4. The impact of the ratio of task volume to message volume on the schedule length

A further observation is that the advantage of MDM over the base line becomes less prominent as the ratio of task volume to message volume decreases. This is because as the message volume becomes gradually larger, MDM tends to schedule the tasks to the same cluster so as to reduce the communication cost via the inter-cluster network with lower bandwidth. This leads to similar scheduling results as those of the base line, which are verified in the experimental results. Under MDM, computation-intensive applications can achieve a higher degree of parallelism than communication-intensive applications.

As can be observed from Fig.4, when the ratio of task-volume to message-volume is small (less than 15/15), the performance achieved by SDS is worse than that of the base line. This is again because the SDS algorithm treats a cluster as a single processing node of higher service rate. Hence when SDS schedules a task to a cluster, the task is always completed at a much later finish time than that expected by SDS. This may cause less effective cooperation among tasks (e.g., the tasks' children have to wait longer than expected). The situation becomes increasingly worse when the actual degree of parallelism in running tasks is high on one cluster, but low on another, which is more likely to happen when the message volume is large compared with the task volume. Hence, the performance is impaired.

4.4. Heterogeneity of inter-cluster and intra-cluster communication

Fig.5 shows the impact of communication heterogeneity among the intra- and inter-cluster networks. The communication heterogeneity is measured by the ratio of the average bandwidth of the inter-cluster network to that of the intra-cluster network (the average bandwidth of the intra-cluster remains unchanged).

It is clear that the performance of the based line is not influenced by the communication heterogeneity since all tasks are scheduled to the same cluster.

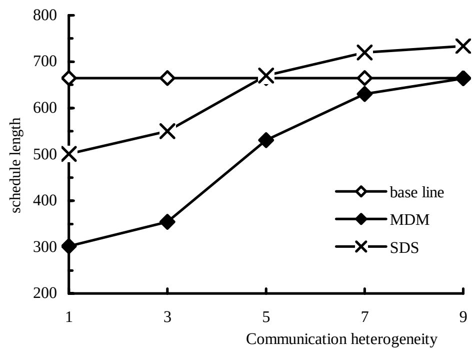


Fig.5. The impact of the computational heterogeneity among the local and inter-cluster networks

It can be observed from Fig.5 that under MDM, the schedule length increases and approaches that of the base line as the communication heterogeneity increases. This is because it incurs a higher communication cost to schedule the tasks to different clusters as the communication heterogeneity increases. As a result, the tasks are more likely to be scheduled to fewer clusters so as to reduce the actual degree of parallelism in the running tasks. This result suggests that the MDM algorithm is more beneficial in multicluster architectures with a smaller heterogeneity between the inter-cluster and intra-cluster communication.

As can be observed from Fig.5, the SDS algorithm achieves the worse performance as compared to the base line when the communication heterogeneity is high (higher than 5 in Fig.5).This is consistent with the experimental results documented in Fig.4, where the performance of SDS is worse than that of the base line when the message volume is large.

It can be observed from Fig.5 that MDM consistently outperforms SDS and the advantage becomes increasingly prominent as the communication heterogeneity decreases. This is because the potential of the MDM algorithm is better exploited as the heterogeneity decreases. These experimental results imply once again that the MDM mapping algorithm is a better choice than SDS in multicluster architectures.

5. Conclusion

This paper presents a DAG mapping algorithm for multiclusters with background workload. Each cluster has its own local scheduler. An approach is developed to calculate the finish time of a task in a cluster. An admission control mechanism is also introduced to exploit the parallel processing capability and guarantee that a cluster is not overloaded. Simulation experiments demonstrate that the multicluster DAG mapping algorithm significantly improves the scheduling performance in terms of the schedule length.

References

[1] A Aleta, J Codina, J Sanchez, and A Gonzalez, "Graph-partitioning based instruction scheduling for clustered processors", *Proceedings of the 34th Annual International Symposium on Microarchitecture*, 2001

[2] R Buyya and M Baker, "Emerging Technologies for Multicluster/Grid Computing," *Proceedings of the 2001 IEEE International Conference on Cluster Computing*, 2001.

[3] J Cao, SA Jarvis, S Saini and GR Nudd, "GridFlow Workflow Management for Grid Computing", *3rd International Symposium on Cluster Computing and the Grid*, 2003

[4] M Chu, K Fan and S Mahlke, "Region-based hierarchical operation partitioning for multicluster processors", *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, 2003

[5] MM Eshaghian, YC Wu, "Mapping heterogeneous task graphs onto heterogeneous system graphs", *6th Heterogeneous Computing Workshop*, 1997

[6] B Fields, R Bodik, MD Hill, "Slack: maximizing performance under technological constraints", *29th international Symposium on Computer Architecture*, 2002

[7] L He, SA Jarvis, DP Spooner, GR Nudd, "Dynamic, capability-driven scheduling of DAG-based real-time jobs in heterogeneous clusters", *International Journal of High Performance Computing and Networking*, 2004

[8] L He, SA Jarvis, DP Spooner, X Chen, GR Nudd, "Dynamic Scheduling of Parallel Jobs with QoS Demands in Multiclusters and Grids", *5th IEEE/ACM International Workshop on Grid Computing (Grid2004)*, Pittsburgh, USA, Nov 8, 2004

[9] L He, SA Jarvis, DP Spooner, GR Nudd, "Optimising static workload allocation in multiclusters", *Proceedings of 18th IEEE International Parallel and Distributed Processing Symposium* (IPDPS'04), April 26-30, 2004, Santa Fe, New Mexico.

[10] L He, SA Jarvis, DP Spooner, GR Nudd, "Dynamic Scheduling of Parallel Real-time Jobs by Modelling Spare Capabilities in Heterogeneous Clusters", *Proceedings of IEEE International Conference on Cluster Computing* (Cluster03), pp. 2-10. Hong Kong, Dec 1-4, 2003

[11] M Iverson, F Ozguner, "Dynamic, Competitive Scheduling of Multiple DAGs in a Distributed Heterogeneous Environment", *Proceedings of the Seventh Heterogeneous Computing Workshop*, 1998

[12] SA Jarvis, L He, DP Spooner, GR Nudd, "The impact of predictive inaccuracies on execution scheduling", to appear in *International Journal of Performance Evaluation special issue on Performance Modelling and Evaluation of High-performance Parallel and Distributed Systems*, 2004

[13] L Kleinrock, *Queueing system*, John Wiley & Sons, 1975.

[14] X Qin, H Jiang, "Dynamic, Reliability-driven Scheduling of Parallel Real-time Jobs in Heterogeneous Systems," *Proceedings of the 30th International Conference on Parallel Processing (ICPP 2001)*, pp.113-122, Valencia, Spain, September 3-7, 2001.

[15] A Radulescu, A van Gemund: "Fast and Effective Task Scheduling in Heterogeneous Systems", *9th Heterogeneous Computing Workshop*, 2000

[16] AG Ranade, "Scheduling loosely connected task graphs", *Journal of Computer and System Sciences*, 2003