Dynamic job scheduling on heterogeneous clusters

J. Barbosa
Universidade do Porto, Faculdade de Engenharia
Departamento de Engenharia Informática
Lab. de Int. Artificial e Ciência dos Computadores
Rua Dr. Roberto Frias, 4200-465 Porto, Portugal
jbarbosa@fe.up.pt

Belmiro Moreira
Universidade do Porto, Faculdade de Engenharia
Rua Dr. Roberto Frias, 4200-465 Porto, Portugal
belmiro.moreira@fe.up.pt

Abstract

This paper addresses the problem of scheduling dynamically multi-user and independent jobs on clusters, both homogeneous and heterogeneous. The dynamic behavior means that the scheduler is able to adapt the scheduling when new jobs are submitted and also when processors availability changes. The scheduler has two main features comparing to other solutions: it considers a job as being described by a direct acyclic graph (DAG) and it is able to schedule parallel tasks, when appropriate, instead of the common dynamic mapping approach that assigns an entire job to a processor or a fixed set of processors. The scheduling method is divided in a scheduling strategy and a scheduler algorithm, so that other scheduling algorithms can be incorporated. In this paper two static DAG schedulers for heterogeneous machines are considered. The results show the behavior of the scheduling method for the short completion time of a batch of jobs. These results show better performance when compared to the common schedulers strategies that fix the number of processors per job or assign one processor per job.

Keywords: *static and dynamic scheduling, parallel task, list scheduling, cluster computing.*

1. Introduction

A computer cluster in this paper is intended to be a set of processors connected by an high speed network, possibly with different processing capabilities, resulting in this case an heterogeneous computer. The task scheduling problem consists in allocating resources (processors) to the tasks and to establish an order for the tasks to be executed on the resources [6], [18]. There are two different types of task scheduling: static and dynamic. Static strategies define a schedule at compile or at launch time based on the knowledge of the processors availability and tasks to execute. The aim is usually to minimize the schedule length (or *makespan*) of the tasks [2], [9], [10], [18].

Dynamic strategies, on the other hand, are applied when the tasks arrival time is not known a priori and therefore the system needs to schedule tasks as they arrive. The aim of dynamic scheduling may be different from minimizing the schedule length as in [17] where the aim is to obtain the short completion time of a batch of tasks, but to minimize some global performance measure related to the quality of service (QoS). In [6] it is defined a value V for each completed task and the aim is to maximize the sum of the values of completed tasks in a given interval of time. The value V is based on task priority, task deadline and a task performance metric.

The algorithms proposed for scheduling DAGs in heterogeneous systems consider only one DAG (or job) to schedule and therefore they are all static approaches [2], [18]. The dynamic scheduling on the other hand considers independent jobs (or tasks) and assign one job to one processor, since it is not considered that a job may have several tasks [6], [17]. In this paper it is proposed a dynamic scheduling method that schedules dynamically multiple independent DAGs (jobs) on heterogeneous clusters.

The dynamic scheduling methods consist in a scheduling strategy and a scheduling algorithm [6], [17]. The scheduling strategy defines the instants when the scheduling algorithm is called to produce a schedule based on the machine information and tasks waiting in the queue at the time it is called. The dynamic behavior is achieved by calling the static scheduling algorithm along time. As described in [12], the dynamic scheduling strategies can be divided in two categories: immediate mode and batch mode. In immediate mode only the new task arrived is considered by the scheduler; in batch mode the scheduler considers the new task and the tasks schedule before but that are waiting to be processed. It was shown that the batch mode allows to achieve higher performance. Therefore, in this research, it is considered the batch mode.

The aim of the work herein presented is to improve the performance of clusters in the processing of multiple applications (or jobs) composed by a set of dependent tasks. The performance is evaluated for the short completion time of a batch of tasks as in [17]. The common scheduling approach is to consider a fixed number of available processors to schedule the set of tasks [9], [10], [13], [15], [18] which

978-0-7695-3680-4/09 $25.00 © 2009 IEEE
DOI 10.1109/ISPDC.2009.19

on a multi-user environment corresponds to fix the number of processors available for each user (or job). Users try to allocate as much capacity as possible and there is not a global management. The results presented compare to that strategy.

The target computer platform is a cluster, either homogeneous or heterogeneous, with a dedicated network. Such clusters can be private clusters of some organization but also they can be the nodes of a Grid infrastructure which to have a good performance requires, at least, that the end clusters have also a good performance. An example where such a batch of jobs can occur is for example in processing image sequences where objects contained in the frames are identified and followed along frames. The objects form may vary along frames and therefore the problem is known as deformable object tracking [14]. This is applied in video surveillance, microscopic imaging processing, biomechanical analysis, etc. In this context a multi-user environment may be for example the input from several cameras.

The remaining of the paper is organized as follows: section 2 defines the scheduling problem and revises related work in scheduling parallel and non-parallel tasks as well as dynamic scheduling methods. Section 3 presents the computational model and the methodology used in this paper. Section 4 presents the scheduling method proposed in this paper. Finally, section 5 presents results and section 6 conclusions.

2. Problem definition and related work

The problem addressed in this paper is the dynamic scheduling of a batch of jobs, or applications, represented by directed acyclic graphs (DAGs) on a distributed memory computer (i.e. a cluster). Former work on dynamic scheduling for heterogeneous clusters consider exclusively independent tasks and assign one task to one processor (sequential tasks) [6], [12], [17]. In [17] it was implemented a genetic algorithm that can achieve the short completion time of a batch of independent tasks. In [12] it was compared the performance of immediate versus batch mode for a set of independent task scheduling heuristics, and concluded that batch mode achieves better performance when minimizing the total schedule length. In [6] it is tested a set of independent task scheduling heuristics with the aim of maximizing a Quality of Service (QoS) measure defined as the value accrued of completed tasks in a given interval of time. The value of a task is calculated based on the priority of the task and the completion time of the task with respect to its deadlines.

In this paper it is addressed a more complex problem that is to consider each job composed by a DAG. The application of a genetic algorithm as in [17] is not consider here because it needs to be reformulated to schedule dependent tasks which will increase the time complexity of the algorithm. The scheduling method incorporates two DAG scheduling algorithms that where proved to be efficient in heterogeneous clusters [2], [18]. The scheduling method has two parts, one related to the scheduling strategy and another to the scheduling algorithm that determines a schedule for a given instant of time. In the scheduling instants the algorithm considers the tasks ready to execute from all jobs and applies DAG scheduling, so that the scheduling problem here can be expressed with the classical formulation [9], [15], [16], [18] resumed next.

A DAG $G = (V, E)$, where V is the set of v nodes of the graph, representing the tasks to be processed, and E is the set of e edges, representing precedence among tasks and communication costs. For each node v_i it is defined a schedule start-time ($ST(v_i)$) and a finish-time ($FT(v_i)$), being the schedule length given by $max_i\{FT(v_i)\}$. Here the DAG is the master DAG that represents all active jobs. Therefore, the goal of scheduling a batch of jobs at any scheduling instant is to minimize $max_i\{FT(v_i)\}$. This definition is valid either for homogeneous or heterogeneous machines and either for parallel tasks (executed on several processors) and non-parallel tasks (executed on one processor).

3. Computational model

The computational platform considered is a distributed memory machine composed by P processors of possibly different processing capacities (heterogeneous cluster), connected by a switched network. It supports simultaneous communications between different pairs of machines. It is assumed that the applications are represented by DAGs and the execution time of the tasks can be estimated at compile time or before starting the execution. The communications required to complete a task are included in the computation time as a function of the processors p used by that task. The inter-task communication is defined as a function of the computational time of the sender task and it is represented by the edges weight in the DAG.

The computational model that supports the estimation of the processing time, for each task, is based on the processing capacity S_i of processor i ($i \in [1, P]$) measured in $Mflop/s$, the network latency T_L, and the bandwidth ω measured in $Mbit/s$. The total computation time is obtained by summing the time spent communicating, T_{comm}, and the time spent in parallel operations, $T_{parallel}$. The time required to transmit a message of b elements is $T_{comm} = T_L + b\omega^{-1}$. The time required to compute the pure parallel part of the code, without any sequential part or synchronization time, on p processors is $T_{parallel} = f(n)/\sum_{i=1}^{p} S_i$. The numerator $f(n)$ is the cost function of the algorithm, measured in floating point operations, depending on problem size n.

As an example, for a matrix multiplication of (n, n) matrices, using the algorithm described in [7], the number of floating point operations is estimated to be $f(n) = 2n^3$.

The total amount of data required to be transmitted in order to complete the algorithm on a grid of processors $P = r \times c$ is $n^2(r-1)$ across rows of processors and $n^2(c-1)$ across columns of processors, resulting in the total of $n^2(r+c-2)$ data elements. If the broadcast over a column or a row of processors is considered sequential, then they are transformed in $(r-1)$ and $(c-1)$ messages, respectively.

Finally, the time function for the matrix multiplication algorithm is given by:

$$T = T_{comm} + T_{parallel} = \frac{n^2(r+c-2)}{w} + T_L + \frac{2n^3}{\sum_{i=1}^{p} S_i} \quad (1)$$

This expression is computed for $p = 1$ to P and the number of processors that minimize the processing time is determined. The computation of the best processor grid for linear algebra kernels, on a heterogeneous machine, was discussed in [3].

4. Scheduling method

The scheduling method is divided in two parts, namely, a scheduling strategy and a scheduling algorithm.

4.1. Scheduling strategy

The scheduling strategy determines the type of dynamic scheduling, immediate or batch mode, and when the scheduling algorithm is called to produce a schedule, which defines a scheduling cycle. The scheduling strategy is shown in Algorithm 1.

Algorithm1.

```
1. while(1)
2.    Wait for new job
3.    Insert the job into the master dag
4.    Call the scheduling algorithm
```

A new scheduling cycle starts when a new job arrives. First, it is inserted in a master DAG as shown in Figure 1, connected to the zero time root node, so that it has the same priority as any other. In this instant the scheduler algorithm is called with the master DAG as input and reschedules all tasks from all jobs, including the tasks scheduled before but not yet processed. If the machine capacity is exceeded some tasks are postponed, as described next.

Whatever the scheduling algorithm used in the second step, no static reservation of processing nodes is made per job. This is an important feature of the dynamic scheduler presented here, because, as shown in the results section, it will improve the cluster performance concerning the total schedule length. The difference for the static reservation schedule is that if a job does not need that capacity at a given time, it will be available for other jobs.

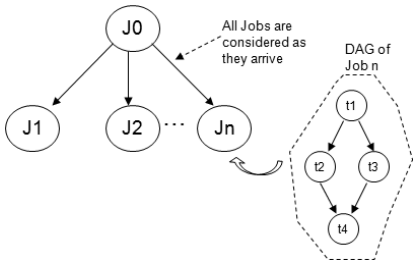

Figure 1. Master DAG example; jobs are inserted in the root node in order to have the same priority

4.2. Scheduling algorithm

In the second part of the scheduling method, the scheduling algorithm is called to produce a schedule based on the information available at that instant concerning the ready tasks and the machine information. Here, two algorithms for scheduling DAGs on heterogeneous clusters are considered, namely, Heterogeneous Earliest-Finish-Time (HEFT) [18] and Heterogeneous Parallel Task Scheduler (HPTS) [2]. HEFT was presented as the best algorithm for DAG scheduling on heterogeneous systems which assigns an entire task to a processor. On the other hand, HPTS optimizes the schedule of a DAG composed by parallel tasks on heterogeneous systems. In [1] it was demonstrated that, when parallel tasks can be used, HPTS allows a better cluster performance than HEFT. The use of the parallel task scheduling depends on the machine load, i.e. for heavy loaded machines, HPTS tends to assign one task to one processor as the HEFT. Note that other algorithms can be used in this step.

The scheduling algorithms applied to the global DAG are based on the list scheduling technique [9] which consists in the following steps: a) determine the available tasks to schedule, b) define a priority to them and c) until all tasks are scheduled, select the task with higher priority and assign it to the processor that allows the earliest start-time. For parallel tasks the last step selects not one processor but several processors that allow the earliest start-time [2]. Note that at this step we refer to tasks that result from all jobs.

Two frequently used attributes to define the tasks priorities are the *t-level* (top-level) and the *b-level* (bottom-level). The *t-level* of a node n_i is defined as the length of the longest path from an entry node to n_i (excluding n_i). The *b-level* of a node n_i is the length of the longest path from n_i to an exit node. The nodes along the DAG with higher *b-level* belong to the critical path. The HEFT algorithm is described in Algorithm 2.

Algorithm2: HEFT

1. Set the computation cost of tasks and comm. costs of edges with mean values
2. Compute b-level
3. Sort tasks by nonincreasing b-level
4. while there are unscheduled tasks
5. Select the first task n_i
6. for each processor p_k in the processor set
7. Compute EFT_{n_i,p_k} using insertion policy
8. Assign task n_i to the processor p_j that minimizes EFT of task n_i

The HEFT algorithm is a heuristic that schedules tasks by decreasing *b-level* value, and assigns them to the processor that allows the Earliest Finish Time (EFT). It implements an insertion policy in order to schedule a task in a earliest idle time slot between two already scheduled task on a processor. In our dynamic method, tasks from all jobs are considered to schedule in step 4.

The HPTS algorithm is a different approach of scheduling DAGs because it considers that a task can execute in several processors [4], [8], [11], [19]. In this case the following assumption are considered. The execution time $t_{i,p}$ of task i is considered to be non-monotonic so that there is a number p of processors for which $t_{i,p} < t_{i,p-1}$ and $t_{i,p} < t_{i,p+1}$. Let $t^*_{i,p}$ be the minimum processing time of task i on the heterogeneous machine, which is achieved when the fastest p processors are used. Other combination of p processors will result in less computational capacity and consequently more processing time. The specific best processor layout should be a parameter of the tasks so that it can be considered in the optimal ($t^*_{i,p}$) processing time computation. From this definition we can estimate a lower bound for the *makespan* which is the sum of the minimum processing time of the tasks on the critical path: $t_\infty = \sum_i t^*_{i,p}$, which is the time required to solve all tasks assuming an unbounded number of processors [19], and in this case it means that any task has the cluster fully available for it.

The expected *makespan* is higher because not all concurrent tasks can execute on the fastest processors which may change dynamically the critical path. Lower priority tasks, after being scheduled, can be transformed in critical path tasks if the capacity of the machine is lower than the required capacity to obtain $t^*_{i,p}$ for all concurrent tasks. Therefore, the algorithm [2] evaluates dynamically the *b-level* of the tasks being scheduled and makes scheduling corrections in order to reduce the maximum *b-level* of those tasks.

The processing capacity required to achieve $t^*_{i,p}$ for task i considers the fastest processors and is defined as $S^*_i = \sum_{j=1}^{p} S_j$. It is obvious that if slower processors are used, the capacity that achieves minimum time for task i is $S'_i < S^*_i$, resulting $t'_i > t^*_i$. Since more processors are required to obtain S^*_i they would imply more communications and consequently more processing time; therefore, the minimum processing time achievable will be certainly higher than the estimated t^*_i. Algorithm 3 shows the HPTS scheduling algorithm.

Algorithm3: HPTS

1. while tasks $\neq \oslash$
2. Compute the set of ready tasks
3. For each ready task i
4. Compute the optimal capacity S^*_i
5. if $\sum_i S^*_i > S_{max}$
6. For each ready task i
7. $S'_i = (S_{max}/\sum_j S^*_j)S^*_i$
8. else
9. For each ready task i $S'_i = S^*_i$
10. Compute tl
11. while ready tasks $\neq \oslash$
12. Select the minimum tl
13. Select processors that allow tl
14. while $S_i < S'_i$ and $t_{i,p} < t_{i,p-1}$ add processor
15. Compute bl
16. while true
17. Select task k with highest bl
18. Select task r with minimum bl
19. if r has been maximum then break
20. Reduce one processor to task r
21. Assign it to task k
22. Re-evaluate processing time of tasks r and k
23. Re-evaluate bl of tasks r and k

In algorithm 3 *t-level* and *b-level* were replaced by `tl` and `bl` respectively. $S_{max} = \sum_{i=1}^{P} S_i$ is the total capacity of the heterogeneous machine computed as the sum the the individual capacity of each node. The while cycle at line 1 refers to all tasks of the Master DAG. In line 2 the ready tasks are those for which the predecessors have finished processing. From line 3 to 9 the algorithm determines the computational capacity that minimizes each ready tasks S^*_i, according to the computational model and by assuming that the fastest processors are used. The number of processors is not important here and it is not registered. Then if the machine capacity S_{max} is exceeded, the capacity assigned to each tasks is limited to the relative weight of each task. From line 10 to 14 the algorithm schedules all ready tasks trying to assign to them the processing capacity determined before. The minimum assignment is one processor to a task and if there is not available processors to start in the *t-level* of the task, it will be implicitly postponed because the processor selection considers the time to process the tasks assigned so far. The selected processors allow the tasks to start on their earliest start-time, but it also verifies if starting later, with more processors, conducts to an earlier finish time. The processing time needs to be tested since in general the processors used are not the fastest ones and consequently the minimum processing time is achieved with less processing capacity, although higher than $t^*_{i,p}$. From line 15 to 23 the algorithm tries to correct the last schedule by assigning more processors to the tasks that have higher *b-level*. The computation of *b-level* on line 15 and 23 uses $t^*_{i,p}$ for the tasks on ahead levels (not processed yet) and for tasks of the actual level, it is used the time computed on line 14 and 22 respectively. The algorithm stops if the task with minimum *b-level* has been maximum during the

minimization procedure. At line 22 the computation time of tasks r and k are re-evaluated considering the new set of processors assigned which have resulted from the transference of one processor from the set of r to the set of k.

The resulting time complexity of algorithms 2 and 3 is $O(v^2 \times P)$ [2], [18].

5. Results and discussion

In this section it is evaluated the dynamic scheduling method proposed in this paper and it is compared to the alternative available scheduling which is: a) to execute each job on a single processor; b) to use several processors per job but the number of processors used in each job is constant. The latter corresponds to the common approach were the user specifies the number of processor to be used to process the job.

The results shown below are obtained from a simulation setup but based on measures taken in the target cluster. The procedure to estimate computation and communication times were presented and analyzed before [3].

5.1. Parallel machine

Although the dynamic algorithm was designed to work on heterogeneous machines, it will be considered here also the homogeneous case in order to have an unbiased comparison of the scheduler behavior and to evaluate its performance in these systems.

The machines considered are composed by 10 and 20 processors, connected by a $1Gbit$ switched Ethernet. In the homogeneous case the relative capacity is 1 for all processors and table 1 shows the relative processing capacities for the heterogeneous case.

Relative capacity	1	0.75	0.5
Machine 1 processors	4	3	3
Machine 2 processors	8	6	6

Table 1. Relative processing capacities and number of processors used in the Heterogeneous Machines

The main characteristic of the network is that it allows simultaneous communications between different pairs of machines. For parallel tasks this is an important characteristic because to complete a task the involved processors (group) have to exchange data. In the general case, when accounting for the amount of communication involved, we need to ensure that inside the group there is no communication conflicts. Otherwise, it would be very difficult to synchronize communications, in different parallel tasks, to avoid conflicts.

5.2. DAGs and tasks

There were used 12 DAGs (jobs) generated based on the algorithm presented in [5] which can be resumed as follows: there are Na nodes with no predecessors and only successors, with ids ranging from 1 to Na; Nb nodes with both predecessors and successors, with ids ranging from Na+1 to Na+Nb; Nc nodes with only predecessors and ids ranging from Na+Nb+1 to Na+Nb+Nc. Here we considered Na=Nc=4 and Nb equal to the remaining nodes. The minimum and maximum out node degree is 2 and 5 respectively. We also make all edges pointing from smaller id nodes to larger id nodes.

The randomly generated DAGs [5] can represent a collection of jobs from one or several users that are organized in a master DAG as expressed on section 4, and representing real applications. In this paper the tasks that form all DAGs are linear algebra kernels namely tridiagonal factorization (TRD), matrix Q computation, QR iteration and correlation (C). The size of each task is randomly generated in the interval $[200, 1200]$. The processing times estimated in the scheduling algorithm are based on real values measured on the target processors. Table 2 shows the relative computation and intra-task communication weight of the tasks. The DAG edges are assigned a inter-task communication cost of 30% of the computation time of the precedent task (computation to communication ratio of 0.3).

Task type	TRD	Q	QR	C
Task relative computational weight	1	0.82	2	3
Task relative communication weight	1	0.125	0.25	0.50

Table 2. Relative computational and intra-communication weights of tasks

5.3. Experimental results

To evaluate the dynamic method proposed in this paper a simulation setup was made in order to compare to the available scheduling policies that are usually used. These policies are: a) to use a fixed set of processors to process a job from the first to the last task, processors that are commonly defined by the user; b) to assign one job to a single processor, which is a frequent utilization when no DAG scheduling is applied. The first case corresponds to the simple usage of independent DAG schedulers that receive as parameter the processors to use. The second case corresponds to the dynamic scheduling of independent tasks as in [6], [17]. The main difference of the proposed algorithm is to implement DAG scheduling considering available tasks from all jobs waiting to be processed.

It were considered the processing of 12 jobs, totalizing 200 tasks, and that jobs arrive with an interval of 10 seconds. The computational cost of tasks are computed based on the computational model. The first experiment considers the heterogeneous machine M1. The results are shown in Figure 2.

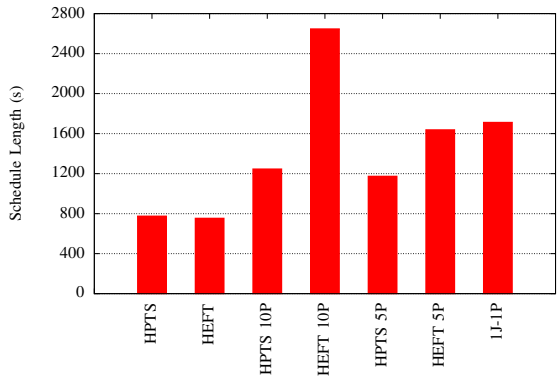

Figure 2. Schedule length obtained for heterogeneous machine M1 with 10 processors

The HPTS and HEFT corresponds to the dynamic implementation proposed in this paper. HPTS 10P and HEFT 10P are the results when 10 processors are used to process each job. In this case, since the machine only had 10 processors, only one job was executed at a time. HPTS 5P and HEFT 5P are the results when 5 processors are used per job. The 1J-1P corresponds to the assignment of one job to a single processor. It is clear that the reservation of processors per job penalizes considerably the performance of the cluster. Both scheduling algorithms, HPTS and HEFT, achieve the best performances with the scheduling strategy proposed.

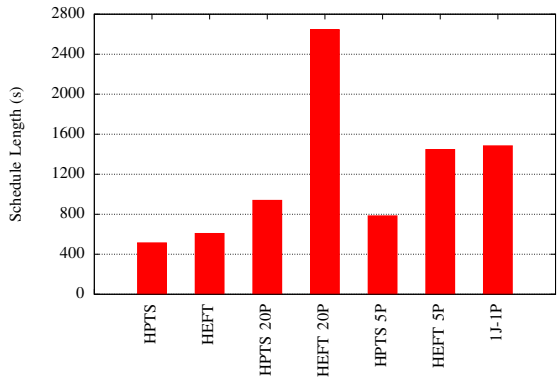

Figure 3. Schedule length obtained for heterogeneous machine M2 with 20 processors

Figure 3 shows the same simulation but on a heterogeneous machine with 20 processors, M2. The labels on the x axis have the same meaning as above. The test conditions were: no reservation, reservation of 20 and 5 processors, and one job one processor. The behavior is identical to the machine M1, i.e. once again the dynamic method proposed here achieves a significant better performance.

Another important conclusion is that not all DAG schedulers work well with a reservation policy as well as its performance depends on the number of processors reserved, as is the case of HEFT. The HPTS on the other hand, ensures the most regular performance. Table 3 shows the mean and the standard deviation of the schedule length.

Machine	Algorithm	Mean (s)	STD (s)
M1	HPTS	1066	253.3
	HEFT	1680	947.8
	1J-1P	1713	0
M2	HPTS	746	214.8
	HEFT	1567	1023.5
	1J-1P	1484	0

Table 3. Mean and Standard Deviation of the schedule length obtained under several conditions on heterogeneous machines (average performance)

To complete the evaluation and to verify if in homogeneous clusters the dynamic scheduling method is also useful, two experiments with the same conditions as for the heterogeneous case were carried out. Figure 4 and 5 present the results. Again, in both cases, the no reservation policy achieves a significant schedule length reduction. In the homogeneous case the schedule length is generally smaller because the machines have more computational power.

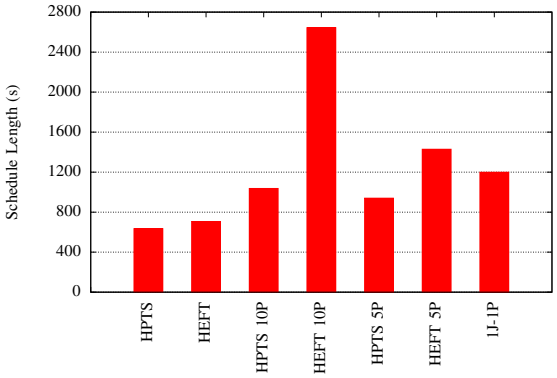

Figure 4. Schedule length obtained for a homogeneous machine with 10 processors

Table 4 shows the mean and standard deviation for the homogeneous case. The performance of the algorithms are similar to the heterogeneous case, so that we can say that the dynamic scheduling method, that do not set the number of

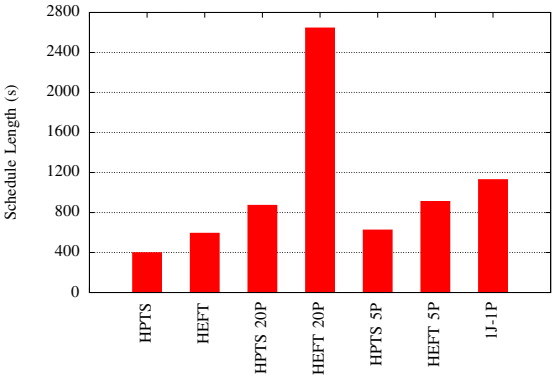

Figure 5. Schedule length obtained for a homogeneous machine with 20 processors

processors to be used to process a job, achieves better performance either on homogeneous and heterogeneous systems.

Machine	Algorithm	Mean (s)	STD (s)
10 Processors	HPTS	871	209.5
	HEFT	1593	978.6
	1J-1P	1199	0
20 Processors	HPTS	631	237.0
	HEFT	1382	1104.4
	1J-1P	1128	0

Table 4. Mean and Standard Deviation of the schedule length obtained under several conditions on homogeneous machines (average performance)

6. Conclusions

The scheduling method presented in this paper can improve the cluster utilization and the response time once we allow a variable computing power (number of processors) assigned for a job. It was shown a better performance for both homogeneous and heterogeneous machines although more tests should be conducted in the future that consider a greater variety of heterogeneous machines. The aim as defined before, is to minimize the completion time of a batch of jobs.

The dynamic method proposed does not require a fixed subdivision of processors. When scheduling a set of ready tasks the machine is viewed has a whole, independently of the groups of processors formed in the last level, thus allowing a better use of the machine and consequently achieving improvements in processing time.

It was also concluded that, when scheduling DAGs, the parallel task scheduler, HPTS, achieved better performance and more regular behavior for different conditions, i.e. for non reservation policies but also for the classical policy that reserves processors to the jobs.

References

[1] J. Barbosa and A.P. Monteiro. A list scheduling algorithm for scheduling multi-user jobs on clusters. In *VECPAR 2008*, volume LNCS 5336, pages 123–136. Springer-Verlag, 2008.

[2] J. Barbosa, C. Morais, R. Nobrega, and A.P. Monteiro. Static scheduling of dependent parallel tasks on heterogeneous clusters. In *Heteropar'05*, pages 1–8. IEEE Computer Society, 2005.

[3] J. Barbosa, J. Tavares, and A.J. Padilha. Linear algebra algorithms in a heterogeneous cluster of personal computers. In *Proceedings of 9th Heterogeneous Computing Workshop*, pages 147–159. IEEE CS Press, May 2000.

[4] J. Blazewicz, M. Machowiak, J. Weglarz, M. Kovalyov, and D. Trystram. Scheduling malleable tasks on parallel processors to minimize the makespan. *Annals of Operations Research*, (129):65–80, 2004.

[5] Sameer Shivle et al. Mapping of subtasks with multiple versions in a heterogeneous ad hoc grid environment. In *Heteropar'04*. IEEE Computer Society, 2004.

[6] Jong-Kook Kim *et al.* Dynamically mapping tasks with priorities and multiple deadlines in a heterogeneous environment. *Journal of Parallel and Distributed Computing*, 67:154–169, 2007.

[7] R. Geijn and J. Watts. Summa: Scalable universal matrix multiplication algorithm. Technical Report CS-95-286, University of Tennessee, Knoxville, 1995.

[8] K. Jansen. Scheduling malleable parallel tasks: An asymptotic fully polynomial time approximation scheme. *Algorithmica*, 39:59–81, 2004.

[9] Y. Kwok and I. Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Computing Surveys*, 31(4):406–471, December 1999.

[10] Y. Kwok and I. Ahmad. On multiprocessor task scheduling using efficient state space search approaches. *Journal of Parallel and Distributed Computing*, 65:1515–1532, 2005.

[11] R. Lepère, G. Mounié, and D. Trystram. An approximation algorithm for scheduling trees of malleable tasks. *European journal of Operational Research*, (142):242–249, 2002.

[12] M. Maheswaran, S. Ali, H.J. Siegel, D. Hensgen, and R.F. Freund. Dynamically mapping of a class of independent tasks onto heterogeneous computing systems. *Journal of Parallel and Distributed Computing*, 59:107–131, 1999.

[13] Gyung-Leen Park. Performance evaluation of a list scheduling algorithm in distributed memory multiprocessor systems. *Future Generation Computer Systems*, (20):249–256, 2004.

[14] S. Sclaroff and A. Pentland. Modal matching for correspondence and recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(6):545–561, 1995.

[15] B. Shirazi, M. Wang, and G. Pathak. Analysis and evaluation of heuristic methods for static task scheduling. *Journal of Parallel and Distributing Computing*, 10:222–232, 1990.

[16] O. Sinnen and L. Sousa. List scheduling: extension for contention awareness and evaluation of node priorities for heterogeneous cluster architectures. *Parallel Computing*, (30):81–101, 2004.

[17] Wei SUN, Yuanyuan ZHANG, and Yasushi INOGUCHI. Dynamic task flow scheduling for heterogeneous distributed computing: Algorithm and strategy. *IEICE Trans. INF. & SYST.*, E90-D(4):736–744, April 2007.

[18] H. Topcuoglu, S. Hariri, and M.-Y Wu. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems*, 13(3):260–274, March 2002.

[19] Denis Trystram. Scheduling parallel applications using malleable tasks on clusters. In *15th International Conference on Parallel and Distributed Processing Symposium*, 2001.