Advanced Reservation-based Scheduling of Task Graphs on Clusters

Anthony Sulistio[1], Wolfram Schiffmann[2], and Rajkumar Buyya[1]

[1] Grid Computing and Distributed Systems Lab
Dept. of Computer Science and Software Engineering
The University of Melbourne, Australia
{anthony, raj}@csse.unimelb.edu.au
[2] Dept. of Mathematics and Computer Science
University of Hagen, Germany
Wolfram.Schiffmann@FernUni-Hagen.de

Abstract. A Task Graph (TG) is a model of a parallel program that consists of many subtasks that can be executed simultaneously on different processing elements. Subtasks exchange data via an interconnection network. The dependencies between subtasks are described by means of a Directed Acyclic Graph. Unfortunately, due to their characteristics, scheduling a TG requires dedicated or uninterruptible resources. Moreover, scheduling a TG by itself results in a low resource utilization because of the dependencies among the subtasks. Therefore, in order to solve the above problems, we propose a scheduling approach for TGs by using advance reservation in a cluster environment. In addition, to improve resource utilization, we also propose a scheduling solution by interweaving one or more TGs within the same reservation block and/or backfilling with independent jobs.

1 Introduction

A Task Graph (TG) is a model of a parallel program that consists of many subtasks that can be executed simultaneously on different processing elements (PEs). Subtasks exchange data via an interconnection network. The dependencies between subtasks are described by means of a Directed Acyclic Graph (DAG). Executing a TG is determined by two factors: a *node weight* that denotes the computation time of each subtask, and an *edge weight* that corresponds to the communication time between dependent subtasks [1]. Thus, to run these TGs, we need a target system that is tightly coupled by fast interconnection networks. Typically, cluster computers provide an appropriate infrastructure for running parallel programs.

Scheduling TGs in a cluster environment is a challenging process because of the following constraints: *Firstly*, a TG requires a fixed number of processors for execution. Hence, a user needs to reserve the exact number of PEs. *Secondly*, due to communication overhead between the subtasks on different PEs, each subtask must be completed within a specific time period. *Finally*, each subtask needs to

wait for its parent subtasks to finish executing in order to satisfy the required dependencies.

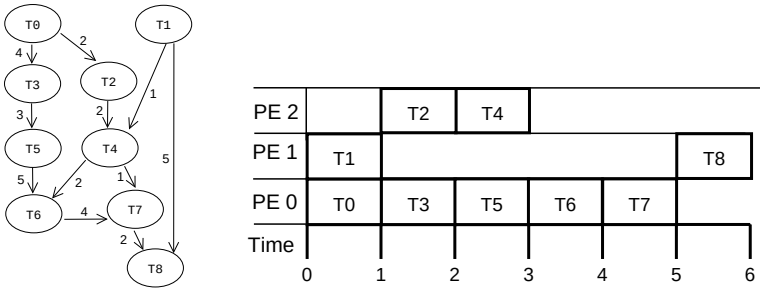


Fig. 1. Illustration of a task graph (left) and its schedule (right) on 3 PEs.

Scheduling a TG on a resource can be visualized by a time-space diagram as shown in Figure 1. In this figure, a TG consists of 9 subtasks ($T0 - T8$), and as an example it was scheduled using 4 target processing elements (TPEs). Each subtask has a node weight of 1 time unit, and its edge weight is also shown on Figure 1 (left) in a number next to the arrow line. In order to minimize the schedule length (overall computation time) and the communication costs of a TG, its subtasks must be assigned to appropriate PEs and they must be started after their parent subtasks. In this example, $T6$ depends on $T4$ and $T5$, so it must wait for both subtasks to finish and it will be scheduled on $PE0$ in order to minimize the communication cost. However, this schedule does not make an efficient use of the given TPEs. Although this schedule assigned the subtasks to 3 PEs, only 2 PEs are actually needed. In general, the right number of schedule's PEs can not be determined in advance. Thus, the resulting schedules might not be able to make an efficient use of the available PEs. Therefore, in this paper, we will talk about how this problem can be improved upon by means of an advanced reservation-based scheduling.

If we consider DAGs with different node and edge weights, the general scheduling problem is NP-complete [2]. Thus, in practice, heuristics are most often used to compute optimized (but not optimal) schedules. Unfortunately, time-optimized algorithms do not make an efficient use of the given PEs. In this context, the *efficiency* is measured by the ratio of the *total node weight* in relation to the *overall available* processing time. As an example, in Figure 1, the efficiency of this TG schedule is 9/18 or 50%, which is quite low because $PE1$ and $PE2$ are mostly idling. If there are no idle PEs at all time, then the efficiency can be said to be optimal (100%).

In [3], a comprehensive test bench (comprised of 36,000 TGs with up to 250 nodes), is used to evaluate the schedule's efficiency of several popular heuristics, such as such as DLS [4], ETF [5], HLFET [6] and MCP [7]. Essentially, it reveals that the efficiency of the DAG-schedules is mostly below 60%, which means a

lot of the provided computing power is wasted. The main reason is due to the constraints of the schedule as demonstrated in the previous example.

The contribution of this paper is as follows. We propose an approach to schedule TGs by using advance reservation in a cluster environment. Moreover, to improve the efficiency or to maximize the CPU utilization, we also propose a scheduling solution by interweaving one or more TGs within the same reservation block and/or backfilling with other independent jobs.

The rest of this paper is organized as follows. Section 2 mentions some related work in this area. Section 3 describes the proposed model, whereas Section 4 evaluates the effectiveness of the scheduling solution. Finally, Section 5 concludes the paper and gives some future work.

2 Related Work

Some systems are available for running DAG applications in the Grid or cluster computing environment, such as Condor [8, 9], GrADS [10], Pegasus [11], Taverna [12] and ICENI [13]. However, only ICENI provides a reservation capability in its scheduler [14]. In comparison to our work, the scheduler inside ICENI does not consider backfilling other independent jobs with the reserved DAG applications. Hence, ICENI resource scheduler does not consider the efficiency of the reserved applications towards CPU utilization.

With regards to the efficiency analysis of functional parallel programs, i.e. executing two or more tasks concurrently, there are only few works done so far. In [15], the authors analyze the efficiency of TG schedules, such as ECPFD [16], DLS [4] and BSA [17] with respect of different Communication-to-Computation (CCR) values. The authors report that a resource efficiency drops down if the CCR value is increased and it also depends on the network topology. Moreover, they find that for coarse grained parallel programs (low CCR), the efficiency achieved is lower than 50%. In [15], the efficiency is defined as speedup of a TG schedule divided by number of processors, where the speedup denotes a ratio of measured parallel execution time to sequential execution time. However, it can be easily shown that this definition of efficiency is equivalent to the one already given in the previous section. Hence, the above findings are similar with [3] as mentioned earlier, except that in [15], the experiments were conducted on a real system because the model accuracy should be evaluated. Therefore, the main goal of our work is to increase the scheduling efficiency of these TGs.

3 Description of the Model

3.1 System Model

Figure 2 shows the open queueing network model of a resource applied for our work. In this model, there are two queues: one is reserved for TGs while the other one is for parallel and independent jobs. Each queue has a finite buffer with size S to store objects waiting to be processed by one of P independent CPUs or

PEs. All PEs are connected by a high-speed network. PEs in a resource can be homogeneous or heterogeneous. For this paper, we assume that a resource has homogeneous PEs, each having the same processing power.

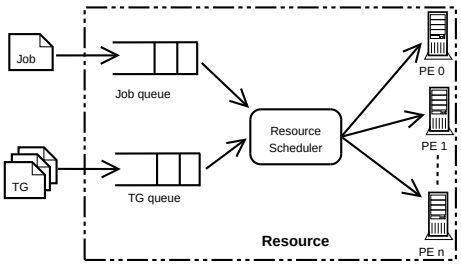


Fig. 2. Overall model where a user submits a set of task graphs on a resource.

In this model, as shown in Figure 2, we assume that we have already known the optimal schedules for each TG in the queue and that their run times are also identified. With this assumption, the resource scheduler only needs to reserve and run these TGs. Moreover, the resource scheduler can perform futher optimization methods that will be discussed later on.

3.2 User Model

A user provides the following parameters during submission:

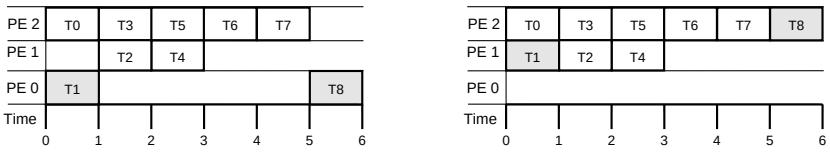
- $TG = \{T1, T2, ..., Tn\}$: Task Graph (TG) that consists of a set of dependent subtasks, where each subtasks has a node and edge weight.
- $List = \{TG_1, TG_2, ..., TG_k\}$: a collection of TGs.
- PE : number of CPUs requested.
- $start$: reservation start time.
- $finish$: reservation finish time.

A user needs to make a reservation by specifying a tuple $< PE, start, finish >$ to a resource. Once a reservation has been confirmed, then the user sends $List$ to the resource before the start time, otherwise the reservation will be cancelled. More details on the states of Advance Reservation can be found on [18].

3.3 Scheduling Model

The aim of our reservation-based scheduler is to improve the efficiency of each TG. Therefore, for executing TGs, we propose the following approaches:

1. **Rearranging subtasks**: This is done by rearranging all subtasks in a TG based on the total number of subtasks executed on each PE. For example,



(a) Rearranging subtasks (b) Moving subtasks

Fig. 3. Scheduling a task graph. The shaded subtasks denote the before (a) and after (b) a moving operation.

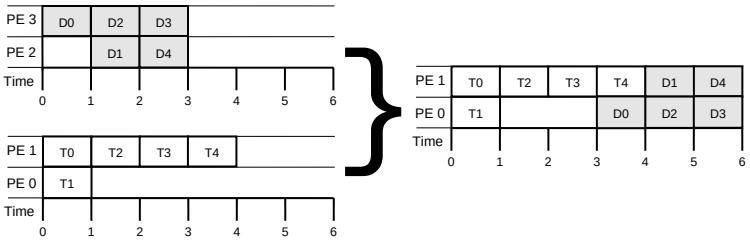


Fig. 4. Combining the execution of two TG by interweaving.

we relocate all subtasks of $PE0$, $PE1$ and $PE2$ as depicted in Figure 1 to $PE2$, $PE0$ and $PE1$ respectively as shown in Figure 3(a). This fundamental step is required as a basis for the next step.

2. **Moving subtasks**: This is done by moving one or more subtasks from one PE to another as long as there are empty slots. For example, we move $T1$ and $T8$ as mentioned in Figure 3(a) from $PE0$ to $PE1$ and $PE2$ respectively as shown in Figure 3(b). With this approach, the best case scenario would result in the reduction of the schedule's PEs (SPEs). Hence, the available PEs can be used to run another TG by interweaving and/or backfilling with independent jobs as discussed in the next step.

3. **Interweaving TGs**: This can be done by combining two or more TGs from $List$ and still keeping the original allocation and dependencies untouched.

   For example, in Figure 4, two TGs that require the same number of PEs are interlocked. In general, the number of PEs do not matter. Each TG has an earliest task to start with. Without the loss of generality, this TG can be placed on a PE that will be availabe next. Due to the time relation in a schedule, we can now look if the second earliest TG "row" can be placed on another PE. If yes, we can proceed in this way until the second TG is completely placed. If there are no PEs available to fit the time relations, we

delay all the previously placed task rows of that schedule appropriately. Of course this will create gaps of idle processor-cycles. But these gaps can be hopefully closed by the following backfilling step.

4. **Backfilling a TG or remaining gaps between interweaved TGs**: This can be done if there are smaller independent jobs that can be fit in and executed without delaying any of the subtasks of a TG.

   In this step, we try to close the gaps by using (independent) jobs from another queue. In contrast to the interweaving step, the best fitting jobs should be selected out of this queue. We start with the first gap and look for the job that has an estimated schedule length lower or (best) equal to the gap's length. Jobs that can not be used to fill enough gaps must be scheduled after all the parallel programs are executed. As an example, there is enough gap on $PE0$ in Figure 4 to put 2 small independent jobs, each runs for 1 time unit.

4 Performance Evaluation

In order to evaluate the performance of our advanced reservation-based scheduler (AR), we compare it with two standard algorithms, i.e. First Come First Serve (FCFS) and EASY backfilling (Backfill) [19]. We use GridSim toolkit [18] to conduct the experiment with different parameters. We simulate the experiment with three different target systems that consist of clusters with varying number of processors, i.e. 16, 32 and 64 PEs. Then, we run the experiment by submitting both TGs and other jobs (taken from a workload trace) into these systems.

4.1 Experimental Setup

Test Bench Structure In this experiment, we use the same test bench (created by a task graph generator), as discussed in [1] and [3], to evaluate the performance of our scheduler. Therefore, we briefly describe the structure of the test bench. More detailed explanation of the test bench can be found in [1].

TGs with various properties are synthesized by a graph generator whose input parameters are varied. The directory tree that represents the structure of test bench are shown in Figure 5. The total number of TGs at each level within a path of the tree is shown on the right side. The parameters of a TG is described as follows (from top to bottom level in Figure 5):

- Graph Size (GS): denotes the number of nodes or subtasks for each TG. In Figure 5, The parameters of a generated TG are grouped into three categories: 7 to 12 nodes (GS7_12), 13 to 18 nodes (GS13_18) and 19 to 24 nodes (GS19_24).
- Meshing Degree (MD) or Number of Sons (NoS): denotes the number of dependencies between the subtasks of each TG. When a TG has a low, medium

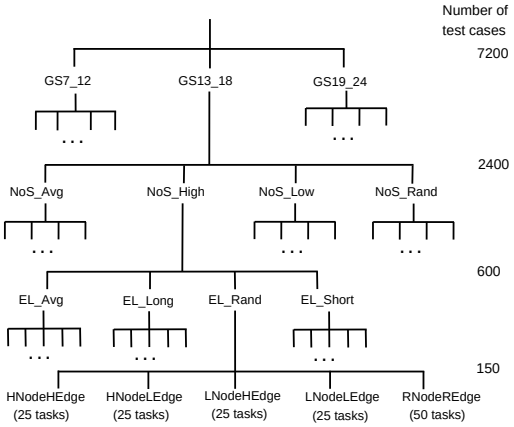


Fig. 5. Structure of the test bench.

and strong meshing degree, the NoS in Figure 5 are NoS_Low, NoS_Avg and NoS_High respectively. TGs with random meshing degrees are represented as NoS_Rand.

- Edge Length (EL): denotes the distance between connected nodes. When a TG has a short, average and long edge length, Figure 5 depicts the notation as EL_Short, EL_Avg and EL_Long respectively. TGs with random edges are represented as EL_Rand.
- Node- and Edge-weight: denotes the Computation-to-Communication Ratio with a combination of heavy (H), light (L) and random (R) weightings for the node and edge.

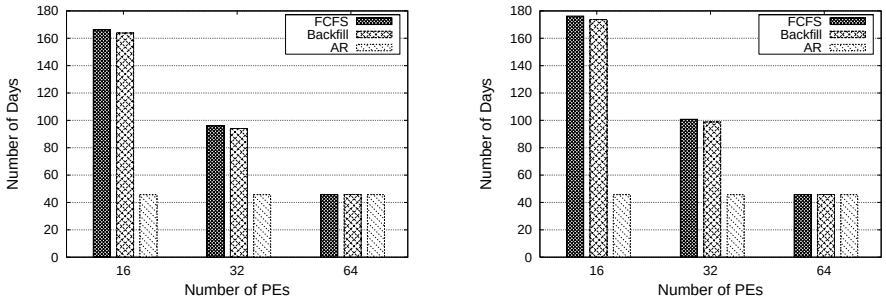
From this test bench, we also use the optimal schedules for the branches of GS7_12 and GS13_18 for both 2 and 4 TPEs. Each branch contains 2,400 task graphs, hence the maximum number of task graphs that we use is 9,600. These optimal schedules were computed and cross-checked by two independent informed search algorithms (branch-and-bound and A*) [1]. Note that at the time of conducting this experiment, the optimal schedules of GS19_24 for 4 TPEs are not yet completed. Therefore, we do not incorporate the schedules of GS19_24 for 2 TPEs into the experiment for consistency.

Workload Trace We also take two workload traces from the Parallel Workload Archive [20] for our experiment. We use the trace logs from DAS2 fs4 (Distributed ASCI Supercomputer-2 or DAS in short) cluster of Utrecht University, Netherlands and LPC (Laboratoire de Physique Corpusculaire) cluster of Universite Blaise-Pascal, Clermont-Ferrand, France. The DAS cluster has 64 CPUs with 33,795 jobs, whereas the LPC cluster has 140 CPUs with 244,821 jobs. The detailed analysis for DAS and LPC workload traces can be found in [21] and [22] respectively. Since both original logs recorded several months of run-time period

with thousands of jobs, we limit the number of submitted jobs to be 1000, which is roughly a 5-days period from each log. If the job requires more than the total PEs of a resource, we set this job to the maximum number of PEs.

In order to submit 2,400 TGs within the 5-days period, a Poisson distribution is used. 4 TGs arrive in approximately 10 minutes for conducting the FCFS and Backfill experiments. When using the AR scheduler, we set the limit of each reservation slot to contain only 5 TGs from the same leaf of the test bench tree from Figure 5. Hence, only 480 reservations were created during the experiment, where every 30 minutes a new reservation is requested. If there are no available PEs, then the resource scheduler will reserve the next available ones.

4.2 Results



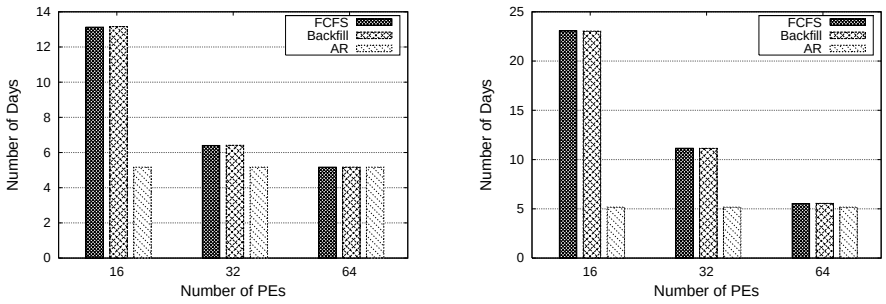
(a) Using GS7_12 branch

(b) Using GS13_18 branch

Fig. 6. Total completion time on the DAS trace with 4 TPEs (lower is better).

Figure 6 and 7 show a huge gain for using AR scheduler for the total completion time on 4 TPEs for both the DAS and LPC trace respectively, especially on a resource that has 16 PEs. Note that for 2 TPEs, the results are similar, hence they are being omitted in this paper.

There are two main reasons that the AR scheduler manages to complete much earlier. The first reason is because a set of TGs in a single reservation slot can be interweaved successfully, as shown in Table 1. For TGs on a GS7_12 branch fot 4 TPEs, the initial reservation duration time is reduced up to 23.74% on the HNode LEdge branch. For TGs on a GS13_18 branch for 4 TPEs, the maximum reduction is 26.31% on the HNode HEdge branch. In constrast, the reduction is much smaller for 2 TPEs on the same branches. The reduction in the reservation duration time can also be referred to as an increase in the efficiency of scheduling TGs in this experiment. Overall, these results show that the achievable reduction depends on the size of the TGs and their graph properties as well.



(a) Using GS7_12 branch

(b) Using GS13_18 branch

Fig. 7. Total completion time on the LPC trace with 4 TPEs (lower is better).

Table 1. Average percentage of reduction in a reservation duration time

Task Graph Parameters	2 TPEs (% reduction)			4 TPEs (% reduction)		
	GS7_12	GS13_18	Avg	GS7_12	GS13_18	Avg
MD Low	2.06	2.15	2.10	14.99	22.80	18.89
MD Avg	6.59	7.73	7.16	13.68	19.87	16.78
MD High	9.66	9.61	9.64	12.33	16.55	14.44
MD Rand	5.35	4.68	5.02	15.80	23.54	19.67
EL Long	0.21	0.00	0.11	9.52	11.85	10.69
EL Short	11.92	13.99	12.96	16.89	23.04	19.96
EL Avg	3.64	3.03	3.34	13.83	22.55	18.19
EL Rand	7.89	7.15	7.52	16.55	25.32	20.94
LNode LEdge	4.02	3.99	4.00	8.42	10.94	9.68
LNode HEdge	6.80	8.01	7.41	9.73	12.62	11.17
HNode LEdge	5.75	5.47	5.61	23.74	25.72	24.73
HNode HEdge	7.57	6.69	7.13	18.78	26.31	22.55
RNode REdge	5.67	6.05	5.86	12.26	24.60	18.43

Table 2. Average of total backfill time on the DAS trace (in seconds)

Task Graph Parameters	2 TPEs			4 TPEs		
	GS7_12	GS13_18	Avg	GS7_12	GS13_18	Avg
MD Low	1,089.00	432.00	760.50	711.33	209.67	460.50
MD Avg	4,499.00	2,301.33	3,400.17	2,121.33	2,585.33	2,353.33
MD High	598.67	145.00	371.83	197.67	614.33	406.00
MD Rand	943.33	1,041.67	992.50	698.67	644.33	671.50
EL Long	2,834.67	1,627.33	2,231.00	1,574.33	491.33	1,032.83
EL Short	1,811.33	1,114.00	1,462.67	467.33	2,469.33	1,468.33
EL Avg	2,263.67	379.67	1,321.67	777.33	329.00	553.17
EL Rand	220.33	799.00	509.67	910.00	764.00	837.00
LNode LEdge	1,760.67	865.33	1,313.00	981.33	329.67	655.50
LNode HEdge	602.67	74.67	338.67	436.67	9.33	223.00
HNode LEdge	620.67	102.00	361.33	201.67	146.67	174.17
HNode HEdge	1,259.67	382.00	820.83	509.33	962.67	736.00
RNode REdge	2,886.33	2,496.00	2,691.17	1,600.00	2,605.33	2,102.67

The second reason is because there are many small independent jobs that can be used to fill in the "gaps" within a reservation slot, as depicted in Table 2 and 3. However, on average, the AR scheduler manages to backfill more jobs from the LPC trace into the reservation slot compare to the DAS trace. This is due to the characteristics of workload jobs themselves. The first 1000 jobs from the LPC trace are primarily independent jobs that require only 1 PE with an average runtime of 23.11 seconds. In contrast, the first 1000 jobs from the DAS trace contain a mixture of independent and parallel jobs that require on average 9.15 PEs with an average runtime of 3676.70 seconds. These phenomena also explain why the total completion time on the DAS trace took much longer than the LPC one.

An interesting observation to note from Figure 6 and 7 is that, the total completion time for the AR scheduler is the same for a resource with 16, 32 and 64 PEs. The FCFS and Backfill algorithm only manage to finish within the same time as the AR scheduler when a resource has 64 PEs. Hence, the AR scheduler executes these jobs and TGs more efficiently.

5 Conclusion and Future Work

In this paper, we have presented a novel approach to schedule TGs by using advance reservation in a cluster environment. In addition, to improve resource utilization, we proposed a scheduling solution (AR scheduler) by interweaving one or more TGs within the same reservation block and/or backfilling with other independent jobs.

The results showed that the AR scheduler performs better than the standard FCFS and Easy backfilling algorithms for reducing both the reservation duration time and the total completion time. The AR scheduler managed to interweave

Table 3. Average of total backfill time on the LPC trace (in seconds)

Task Graph Parameters	2 TPEs			4 TPEs		
	GS7_12	GS13_18	Avg	GS7_12	GS13_18	Avg
MD Low	2,451.67	1,640.67	2,046.17	1,136.00	815.67	975.83
MD Avg	883.00	474.00	678.50	718.00	2,874.33	1,796.17
MD High	1,902.33	1,916.67	1,909.50	2,334.00	678.00	1,506.00
MD Rand	2,474.67	1,698.67	2,086.67	2,172.00	1,020.33	1,596.17
EL Long	2,018.67	1,611.33	1,815.00	1,889.00	1,419.33	1,654.17
EL Short	1,830.67	1,835.00	1,832.83	1,610.00	1,846.33	1,728.17
EL Avg	2,469.00	1,213.67	1,841.33	1,218.33	455.00	836.67
EL Rand	1,393.33	1,070.00	1,231.67	1,642.67	1,667.67	1,655.17
LNode LEdge	1,578.33	978.00	1,278.17	1,459.33	1,419.00	1,439.17
LNode HEdge	1,126.33	1,051.33	1,088.83	1,387.67	541.67	964.67
HNode LEdge	2,114.33	683.00	1,398.67	828.00	940.33	884.17
HNode HEdge	1,121.67	1,529.33	1,325.50	838.00	1,011.00	924.50
RNode REdge	1,771.00	1,488.33	1,629.67	1,847.00	1,476.33	1,661.67

a set of task graphs with a reduction of up to 23.74% and 26.31% on 7–12 nodes and 13–18 nodes with 4 target processing elements (TPEs) respectively. However, much smaller reduction is noticed for 2 TPEs on same nodes. These results also showed that the achievable reduction depends on the size of the task graphs and their graph properties as well. Finally, the results showed that when there are many small independent jobs, the AR scheduler accomplished to fill these jobs into the reservation blocks.

An extension to this work is to consider scheduling task graphs with an economy model in order to see how efficient the AR scheduler is in terms of resource profits and user costs. Moreover, the AR scheduler can be extended to interweave task graphs from different reservation slots within a specified time block.

Acknowledgement
We would like to thank Chee Shin Yeo, Uros Cibej and anonymous reviewers for their comments on the paper.

References

1. Hoenig, U., Schiffmann, W.: A comprehensive test bench for the evaluation of scheduling heuristics. In: Proc. of the 16th International Conference on Parallel and Distributed Computing and Systems (PDCS'04), Cambridge, USA (2004)
2. Coffman, E.G., ed.: Computer and Job-Shop Scheduling Theory. Wiley (1976)
3. Hoenig, U., Schiffmann, W.: Improving the efficiency of functional parallelism by means of hyper-scheduling. In: Proc. of the the 35th International Conference on Parallel Processing (ICPP - in print), Ohio, USA (2006)
4. Sih, G.C., Lee, E.A.: A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. IEEE Transactions on Parallel and Distributed Systems **4**(2) (1993) 75–87

5. Hwang, J.J., Chow, Y.C., Anger, F.D., Lee, C.Y.: Scheduling precedence graphs in systems with interprocessor communication times. SIAM Journal on Computing **18**(2) (1989) 244–257
6. Adam, T.L., Chandy, K.M., Dickson, J.: A comparison of list scheduling for parallel processing systems. Communications of the ACM **17** (1974) 685–690
7. Wu, M.Y., Gayski, D.D.: Hypertool: A programming aid for message-passing systems. IEEE Transactions on Parallel and Distributed Systems **1**(3) (1990) 330–343
8. Tannenbaum, T., Wright, D., Miller, K., Livny, M.: Condor – a distributed job scheduler. In Sterling, T., ed.: Beowulf Cluster Computing with Linux. MIT Press (2001)
9. Thain, D., Tannenbaum, T., Livny, M.: Distributed computing in practice: the Condor experience. Concurrency – Practice and Experience **17**(2-4) (2005) 323–356
10. Berman, F., Chien, A., Cooper, K., Dongarra, J., Foster, I., Gannon, D., Johnsson, L., Kennedy, K., Kesselman, C., Mellor-Crummey, J., Reed, D., Torczon, L., Wolski, R.: The GrADS project: Software support for high-level grid application development. International Journal of High Performance Computing Applications **15**(4) (2001) 327–344
11. Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Patil, S., Su, M.H., Vahi, K., Livny, M.: Pegasus: Mapping scientific workflow onto the grid. In: Across Grids Conference 2004, Nicosia, Cyprus (2004)
12. Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M., Wipat, A., Li, P.: Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics **20**(17) (2004) 3045–3054
13. McGough, S., Young, L., Afzal, A., Newhouse, S., Darlington, J.: Performance architecture within ICENI. UK e-Science All Hands Meeting (2004) 906–911
14. McGough, S., Young, L., Afzal, A., Newhouse, S., Darlington, J.: Workflow enactment in ICENI. UK e-Science All Hands Meeting (2004) 894–900
15. Sinnen, O., Sousa, L.: On task scheduling accuracy: Evaluation methodology and results. Journal of Supercomputing **27**(2) (2004) 177–194
16. Ahmad, I., Kwok, Y.K.: On exploiting task duplication in parallel program scheduling. IEEE Transactions on Parallel and Distributed Systems **9**(8) (1998) 872–892
17. Kwok, Y.K., Ahmad, I.: Link contention-constrained scheduling and mapping of tasks and messages to a network of heterogeneous processors. Cluster Computing: The Journal of Networks, Software Tools, and Applications **3**(2) (2000) 113–124
18. Sulistio, A., Buyya, R.: A grid simulation infrastructure supporting advance reservation. In: Proc. of the 16th International Conference on Parallel and Distributed Computing and Systems (PDCS'04), Cambridge, USA (2004)
19. Mu'alem, A.W., Feitelson, D.G.: Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling. IEEE Transactions on Parallel and Distributed Systems **12**(6) (2001) 529–543
20. Feitelson, D.: Parallel workloads archive. http://www.cs.huji.ac.il/labs/parallel/workload (2006)
21. Li, H., Groep, D., Walters, L.: Workload characteristics of a multi-cluster supercomputer. In Feitelson, D.G., Rudolph, L., Schwiegelshohn, U., eds.: Job Scheduling Strategies for Parallel Processing. Springer-Verlag (2004) 176–193 Lect. Notes Comput. Sci. vol. 3277.
22. Medernach, E.: Workload analysis of a cluster in a grid environment. In Feitelson, D.G., Rudolph, L., Schwiegelshohn, U., eds.: Job Scheduling Strategies for Parallel Processing. Springer-Verlag (2005) 36–61