Evaluation of Eligible Jobs Maximization Algorithm for DAG Scheduling in Grids

Tomasz Szepieniec[1] and Marian Bubak[1,2]

[1] Academic Computer Centre CYFRONET AGH,
ul. Nawojki 11, 30-950 Kraków, Poland
[2] Institute of Computer Science, AGH,
al. Mickiewicza 30, 30-059 Kraków, Poland
t.szepieniec@cyfronet.pl, bubak@agh.edu.pl
Phone: (+48 12) 617 43 35, Fax: (+48 12) 633 80 54

Abstract. Among many attempts to design DAG scheduling algorithms that would face grid environment requirements, the strategy of number of eligible jobs maximization seems promising. Therefore, this paper presents the results of thorough analysis and evaluation of this strategy and its implementation called PRIO. We have analysed a large space of random DAGs and various resources parameters to compare results of PRIO algorithm with standard critical path length prioritization, FIFO prioritization as well as with quasi-optimal solution. Results of this comparison, in terms of the makespan and robustness, are supplemented by a theoretical and specific case analysis. We conclude with an assessment of usefulness of the current implementation of eligible jobs maximization strategy.

Keywords: DAG, list scheduling, application scheduling, Internet-based computing, eligible jobs maximization.

1 Introduction

Modern, large scale applications require efficient execution on available resources. The structure of many of these application may be represented by DAGs. Many attempts to the DAG-based application scheduling for grid environments were enhancements of existing solutions [1,2], however, one may argue that they address the requirements only partially or require knowledge about the environment that is hard to obtain. The uncertainty level typical for *Internet-based computing* [3] precludes an accurate identification of a critical path. An observation done from a perspective of a user or an application scheduler in large computation environments, that free resources quickly become busy if not allocated immediately, provide foundation for an idea of keeping an application ready to use resources immediately when they become available. In case of DAGs, keeping applications 'ready' means scheduling jobs in a way which maximises the number of jobs that are eligible for mapping to new resources when they appear. An implementation of this strategy was a subject of a series of papers [3,4,5,6]. An

M. Bubak et al. (Eds.): ICCS 2008, Part I, LNCS 5101, pp. 254–263, 2008.

algorithm was introduced to provide Internet-Computing (IC) optimal schedule for a large class of DAGs. However, in practice an algorithm that may be applied for all possible DAGs is needed, so a heuristic algorithm was designed to provide an IC optimal schedule if it exists, while for the rest of DAGs, the heuristics take steps to enhance eligible jobs ratio [7].

The heuristics were implemented with integration with Condor DAGMan, under name *PRIO Tool*. This implementation was used in [7] for comparison with the FIFO ordering. The results of the evaluation were promising, however, while FIFO ordering is rather a basic strategy, we believe that a comparison with stronger algorithms is still needed to understand usefulness of the Malewicz's heuristics.

The goal of this paper is to provide a comparison with more advanced techniques and better understanding of the usability of PRIO algorithm for the DAGs scheduling in contemporary grids. Specifically, we focused on following aspects:

- assessment of PRIO algorithm applicability to real grid systems; we tried to identify possible strengths and weaknesses of this algorithm;
- statistical evaluation of the algorithm with comparison to the standard DAG's job prioritization method, FIFO ordering and optimal schedules, in order to clarify how successful the algorithm is and how much robust results it provides;
- case analysis of a few typical results to understand characteristics of schedules provided by PRIO algorithm prioritization.

Before we describe results of an analysis in Section 3, we give a brief overview of related works in Section 2. Next, we introduce a practical evaluation by describing a grid model implemented for simulation in Section 4. A methodology we have used is given in Section 5. Statistical evaluation and case analysis are presented in Sections 6 and 7. Finally, we summarize our evaluation in Section 8.

2 Related Works

Extensive overview of DAG scheduling algorithms related to grids is given in [8]. In a grid environment, new challenges are faced. The most important ones are the heterogeneity of resources and dynamic changes of several parameters of the environment, like resources availability and transfer time between nodes. Several attempts were made to adapt previous DAG scheduling heuristics, like HEFT [9] and FCP [10] to grid environment [1,2], however none of them fully addresses new requirements.

An important issue in the list scheduling, like HEFT or SDC [11], is its sensitivity to the method of job performance on nodes evaluation [13]. From a practical point of view, in a large environment it is complicated to gather relevant data, even if we use less sensitive *hybrid* algorithm [12]. All the mentioned above heuristics do not consider the heterogeneity of the network parameters. For applications which are sensitive to them, clustering heuristics would be a better solution [8]. Finally, some of proposed algorithms e.g. JDCS [14], provide sophisticated mechanisms such as back-trace techniques to reduce the data

preloading delay, but they usually require rarely available services, like the grid performance prediction or the resource reservation, and heavily depend on their quality.

Currently available evaluations of PRIO [7] are limited to comparison with FIFO-based ordering. While the latter method, in practice, means 'no prioritization', we consider such an evaluation insufficient. However, PRIO proved to be substantially better than FIFO. It is worth to mention that comparison was made on DAGs with almost equal execution time (random changes up to 30 per cent) and for resources nearly homogeneous.

3 Analysis of the Eligible Jobs Maximization Algorithm

The heuristics mentioned in the previous section, like HEFT, focus directly on reducing makespan. On the contrary, PRIO maximises number of eligible jobs in each step, hoping that this strategy results in increasing resource usage and, eventually, in decreasing the overall DAG makespan.

PRIO Tool is an implementation of a heuristic algorithm proposed by Malewicz et al. [3]. The heuristic includes the results of research on algorithms, that find optimal IC schedules for some classes of DAGs [3,4,5,6]. PRIO takes advantage of the idea that it is possible to derive an IC-optimal schedule for a complex DAG by decomposing it into simple components, scheduling each component independently, and then combining the resulting schedules [3]. The tool provides a prioritization of DAG jobs, which is typically the first stage of list scheduling algorithms. Mapping jobs to resources is expected to be done according to prioritized list.

The target platform of PRIO is the application scheduling in grids in which we have concurrent schedulers. In such environments, if heavily used, resources that become available are allocated immediately to jobs that are eligible. So, from the point of view of application level, schedulers resources are lost if there are no eligible jobs available at the moment when resources appear. The aim of the PRIO is to minimise probability of kind of situation, called *a gridlock*.

The important advantage of PRIO algorithm is that we do not require estimation of time needed to complete either jobs or transfers. The only input data that PRIO requires is a structure of a DAG. Therefore, in environments in which there are no means of obtaining estimation of node and links efficiency (in some cases this efficiency could vary for different jobs and/or change over time) or we do not have knowledge about jobs' reference execution time, PRIO would be a better choice than methods that are very sensitive to the accuracy of performance data [13].

However, in many cases, taking into consideration the structure of DAGs only, would weaken the results. Especially, it is worth to note that a sequence of jobs obtained by the PRIO algorithm is, in fact, intended for jobs completion, while at job completion time new eligible jobs are triggered. PRIO proposes this order for submission, which means that an assumption is made that all jobs and transfers take the same time or, at least, an order of completion remains the

same. It touches the heuristics applicability to heterogeneous environments and limits its usage for DAGs composed with jobs having different execution time. In such case a schedule done according PRIO would suffer from:

- the aim of maximising eligible jobs would be missed as a result of changes in jobs' completion sequence,
- gridlock probability increase in case of long jobs are scheduled while short jobs would provide new eligible jobs earlier.

4 Simulation Method and Related Grid Model

While the PRIO algorithm was designed to provide an application level scheduling, in the simulation environment we try to model resources which are available for single user in a computational grid environment organized similarly to WLCG/EGEE grid [15]. In WLCG resources consist of about 250 computation clusters, called 'sites', of size between several to thousands CPUs. Jobs at each site are scheduled by a Local Resource Management System (LRMS) according to local policy of supporting set of *virtual organizations* which are mapped onto queues at LRMS. A resource broker service chooses suitable services according to a job's owner membership in a *virtual organization* and job requirements. This decision is done immediately according to a resources list that is ranked typically by the expected waiting time in a local queue.

In such a grid architecture, requesting a resource for a job at the moment when it becomes eligible, introduces unacceptable overheads. More reasonable solution is to use *lazy scheduling*, in which resources are requested in advance according to expected need, but allocation to jobs is done at the moment when resources become available. When no jobs are ready for execution at this time, resources are freed by the application, while usually keeping resources that we do not actively use is against rules. Such conditions creates a room for application scheduling in which the environment is seen as a *stream of resources* ready for the job allocation and usage.

We use a simple, discrete-event simulator, similar to one applied in the first evaluation of PRIO algorithm. Resources are modelled as a probabilistic stream of free workers (CPUs) parameterized by: the average time between *resources-appear events* and the average number of workers available in such events. Computational resources are parameterized by heterogeneity level ranged from homogeneous resources to the level in which some workers can be 4 times more efficient than others, which is the value observed in WLCG.

While in our evaluation we do not focus on brokering resources to a specific site but on prioritization only, data transfers are not modeled separately. We considered it included in overall cost of executing job and in the heterogeneity of resources. We also assume that resources broker takes resources in random order, regardless of their efficiency.

5 Evaluation Set-Up

The aim of our evaluation was to understand PRIO usability by comparing its results with other solutions for a wide range of cases.

Below we describe prioritization methods chosen for the comparison. In each case a motivation for adding this algorithm to our comparison is provided.

1. BTIME – a classic approach to prioritize jobs according to the maximal sum of execution costs on a path from a current job to one of sinks in DAG, based on an estimated computing and communication costs of DAG jobs. A comparison with this approach is the most interesting as this method is commonly used.
2. FIFO – jobs are scheduled in the order in which they are made eligible. When more that one job is eligible at the same time, the order from DAG definition is taken. The reason for adding this algorithm to comparison was twofold: (1) this model represents no prioritization, so we could measure added values of PRIO, and (2) this algorithm was used in previous PRIO evaluations it gave us an opportunity for validating our results.
3. Quasi-optimal – post mortem searching for optimal prioritization based on exact availability of resources. We applied the algorithm that test all possible prioritizations with some optimizations that speed-up the process (e.g. detecting schedules already tested). However, it was still necessary to limit both execution time and a solution buffer size. In cases when the algorithm was not able to complete within defined time, we consider as a result the best of: temporal result of optimal algorithm and other algorithms used in the comparison. Motivation for adding this algorithm to comparison was to estimate a room for further improvement.

Random DAGs generation was done using modified DAGGEN tool [16]. We generated DAGs of different characteristics, parametrized by fatness (FAT), density of communication (DENS), regularity (REG), jumps between level (JUMP), difference in cost between jobs and overall size of DAG (CCR). Values of parameters used are shown in upper part of Table 1. We had 3000 different configurations for DAG generation, each was used 10 times in the process of generating DAGs.

Table 1. Values for DAG parameters and environment parameters

Parameter	Values
FAT	0.05, 0.1, 0.2, 0.3, 0.5
DENS	0.05, 0.1, 0.2, 0.3, 0.4
REG	0.01, 0.05, 0.1, 0.2, 0.4
JUMP	1, 3, 5
CCR	0, 1, 2, 3
size of DAGs	10, 30
TBE	5, 10, 20, 40, 80, 160
RS	1, 3, 5, 7, 9, 11
HI	1, 2, 3, 4

Table 2. Ratio of wins for each prioritization algorithm in nontrivial cases

	PRIO	BTIME	FIFO
summary of wins	59.0%	93.7%	8.6%
individual wins	3.9%	34.0%	0.9%

Table 3. Normalized average makespan and its standard deviation achieved by each method in nontrivial cases

	PRIO	BTIME	FIFO
average makespan	1.111	1.097	1.129
standard deviation	0.164	0.158	0.174

Resources are characterized by average time between resources-appear events (TBE), average resources size (RS) in each event and heterogeneity index (HI) of the system, defined as maximum performance ratio between two machines in a system. A list of parameters used is collected in the lower part of Table 1. There were 90 parameters combination each applied 10 times to every generated DAG. In general, there were 2.7M simulations performed for every prioritization method.

It is worth mentioning that in a mapping process we mapped all the eligible jobs in prioritization order, so the overall mapping order could be different from the prioritization list. As we focus on evaluation of prioritization algorithms, the simplest method of resource mapping was used. To make the competition fair, the efficiency of workers (heterogeneity index) mapped onto the same job remains the same for the whole methods.

The simulations were performed on Zeus cluster in ACC CYFRONET AGH. Limiting the optimal algorithm execution time to 1 minute, we were able to collect all the results in less that 72 hours using 150 cores of 4-core Intel Xeon 2.33 GHz.

6 Statistical Analysis of Results

From analysis presented, we excluded *trivial cases*, which we consider schedules that provided the same results for all algorithms. We had two classes of such cases: (1) resources were so limited that the whole process got almost sequential, (2) availability of resources allowed for scheduling all eligible jobs virtually immediately. In our experiment 82% of all cases were classified as trivial.

Table 2 presents a summary of the comparison, assuming that we are interested only in the best algorithm for each case. For all cases we chose an algorithm that provides the smallest makespan, which we call a winner. In many cases more than one method provided the best result. In general, the most successful prioritization method was BTIME. PRIO proved to be substantially better that FIFO, but failed to provide result comparable to BTIME.

Table 4. Selected parameter's value correlation with makespan and range of changes of average makespan results for PRIO and BTIME

	PRIO		BTIME	
	Correlation	Max.change	Correlation	Max.change
density	0.2	0.1%	97.6	0.0%
allowed jumps between levels	0.99	2.6%	0.99	3.0%
allowed jumps between levels	0.99	2.6%	0.99	3.0%
performance ratio between jobs	-0.95	1.4%	-0.85	1.3%
heterogeneity of environment	-0.95	1.4%	-0.85	1.3%

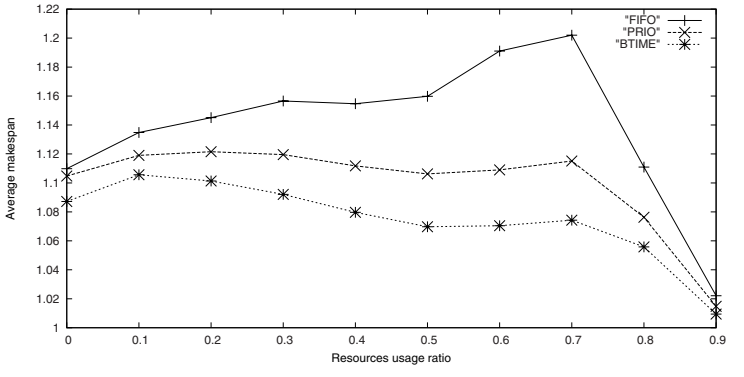

Fig. 1. Average normalized makespan in function of resource usage ratio

For better understanding of usability of PRIO algorithm we analysed parameters in class of 3.9% cases in which PRIO provide the best result. The results show the same distribution of parameters, similar to distribution of parameters in whole nontrivial cases set. Therefore, we can assume that success rate of PRIO algorithm, depends on a specific structure of DAG and its relation to the resources stream.

The rest of the analysis is based on makespan values normalized according to quasi-optimal method results. Normalization was done to achieve the same impact to presented data from long and short schedules.

General summary of average makespan of each algorithm and its standard deviation is presented in Table 3. We see that PRIO is situated between the other two algorithms both in makespan and robustness measured in terms of standard deviation of makespan [17]. Additionally, we could observe that each prioritization method produced average makespan about 10% worse than quasi-optimal. Taking into account also standard deviation that exceeds 15%, we have a picture of substantial room for further improvement in this field.

We also provided an analysis on how simulation parameters influence the results, but even if there was a strong correlation between parameter values and makespan, overall impact for makespan in range of parameters we evaluated does not exceed 3% (Table 4).

Table 5. Average stall ratio

	PRIO	BTIME	FIFO
average stall ratio	0.412	0.408	0.415
standard deviation	0.263	0.261	0.259

In Figure 1 average makespan in function of average resource usage is presented to check how schedules depend on value that reflects the availability of resources. We shall note, that in every class of each parameter BTIME prioritization outperformed PRIO. Additionally, we can conclude that scheduling of DAGs prioritized by evaluation algorithm depends strongly on specific structure of a DAG and resources availability.

Another question that we tried to answer in this evaluation was if the PRIO algorithm was able to provide more eligible jobs. If so, the ratio of 'stalls', defined as a number of cases where there were no eligible jobs when a resource event appeared in the system, should be smaller than in the other algorithms. Unfortunately, as we can see in Table 5 stall ratio was higher that in BTIME case. It could be connected with the fact, that bad schedule decisions lead to stalls while waiting for a job that is being executed longer than average. Huge value of standard deviation is caused by various parameters of resource flow.

7 Typical Cases Analysis

In the previous section we presented the results of evaluation that shows that PRIO prioritization provides results statistically worse than simple BTIME prioritization. In this section, we will provide analysis of two cases, taken from the experiment, illustrated in Figure 2, in order to better understand the results. On both figures, the number of jobs remaining both for submission and for completion their execution are shown in function of time. A space between these two lines was filled to better illustrate running jobs.

In the left graph, we can observe that, an application scheduled according to PRIO prioritization gained more resources and complete more jobs in first stage ($t<150$) of computation process. This was enough to start next stage substantially earlier and save about 6% of overall makespan. So, in this case PRIO algorithm provide results according to expectations.

In the right graph we can observe that for most of the time both prioritizations were equally efficient. Sequences proposed, although different in second half of makespan, not caused differences in allocating resources that were appearing. What is more, PRIO gained two slots in the last stage, substantially before the other algorithm. Surprisingly, at the end scheduling according to BTIME prioritization wins, while using such priorities it was possible to start the prelast job not waiting for completion of others running. So, in this particular case the strategy, that PRIO is build on, clearly provide bad solutions.

Summing up, we should note, that overall results depend on subtle relations between jobs timing and resource appeared-events. Concerning applicability of

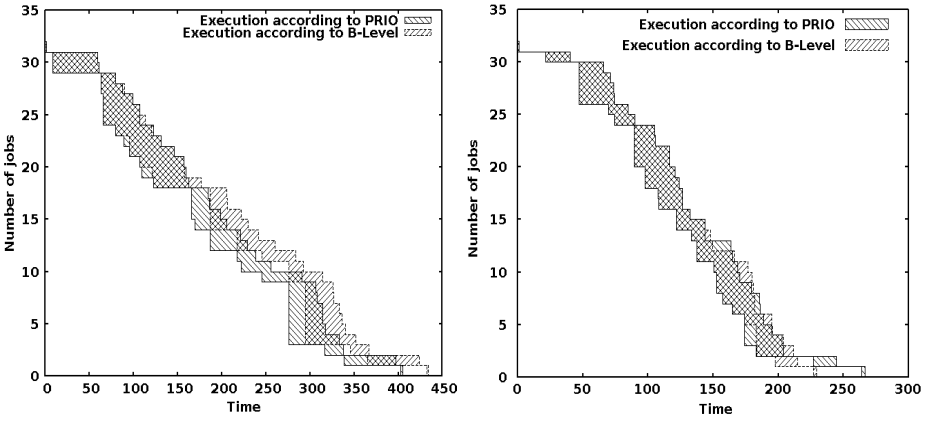

Fig. 2. Traces of two different schedules according PRIO and BTIME prioritizations. Graphs present number of jobs remaining for submission (lower line) and for completion (upper line) in function of time. Dashed area between them illustrates jobs in execution. Overlapping dashed area not impose the same set of jobs!

strategy used in PRIO, we could conclude that gaining more resource give good results until it is done in final part of the schedule. In final stage of DAG execution it is important to keep concurrent running, avoiding last jobs to be left at the end alone.

8 Conclusions

In this paper we considerably extended knowledge about heuristics that maximize eligible jobs (the PRIO algorithm) by comparison with other heuristics and analyse the applicability of this algorithm in real usage. The main conclusion is that, the tested implementation of eligible jobs maximization strategy is not mature enough to be used in the described environment. At this moment, simpler mechanisms, like BTIME heuristics, provide better schedules. However, there is a significant room for improvements, where strategy for maximizing eligible jobs could be useful, in improving existing solutions. To achieve this, we will continue our research towards elimination of weakness of the PRIO algorithm identified in this paper.

Acknowledgments. We would like to thank Grzegorz Malewicz for providing us the implementation of PRIO tool and a simulator, on which the presented evaluation was obtained. Simulations were processed on Zeus cluster in ACC CYFRONET AGH. This work was partly supported by EU IST CoreGRID Project.

References

1. You, S.Y., Kim, H.Y., Hwang, D.H., Kim, S.C.: Task Scheduling Algorithm in GRID Considering Heterogeneous Environment. In: Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA 2004, Nevada, USA, June 2004, pp. 240–245 (2004)
2. Ma, T., Buyya, R.: Critical-Path and Priority based Algorithms for Scheduling Workflows with Parameter Sweep Tasks on Global Grids. In: Proc. of the 17th International Symposium on Computer Architecture and High Performance Computing, Rio de Janeiro, Brazil (October 2005)
3. Malewicz, G., Rosenberg, A., Yurkewych, M.: Towards a Theory for Scheduling Dags in Internet-Based Computing. IEEE Transactions on Computers 55(6), 757–768 (2006)
4. Rosenberg, A.L.: On scheduling mesh-structured computations for Internet-based computing. IEEE Trans. Comput. 53, 1176–1186 (2004)
5. Rosenberg, A.L., Yurkewych, M.: Guidelines for scheduling some common computation-dags for Internet-based computing. IEEE Trans. Comput. 54, 428–438 (2005)
6. Cordasco, G., Malewicz, G., Rosenberg, A.L.: Advances in IC-Scheduling Theory: Scheduling Expansive and Reductive Dags and Scheduling Dags via Duality. IEEE TPDS 18(11) (November 2007) ISSN: 1045-9219
7. Malewicz, G., Foster, I., Rosenberg, A., Wilde, M.: A Tool for Prioritizing DAGMan Jobs and Its Evaluation. In: 15th IEEE International Symposium on High Performance Distributed Computing (HPDC-15), pp. 156–167 (2006)
8. Dong, F., Akl, S.G.: Scheduling Algorithms for Grid Computing: State of the Art and Open Problems. Technical Report of Queen's University School of Computing, 2006-504 (January 2006)
9. Topcuoglu, H., Hariri, S., Wu, M.Y.: Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. IEEE Transactions on Parallel and Distributed Systems 13(3), 260–274 (2002)
10. Radulescu, A., van Gemund, A.J.C.: On the Complexity of List Scheduling Algorithms for Distributed Memory Systems. In: Proc. of 13th International Conference on Supercomputing, Portland, Oregon, USA, ovember 1999, pp. 68–75 (1999)
11. Shi, Z., Dongarra, J.J.: Scheduling workflow applications on processors with different capabilities. Future Generation Computer Systems 22(6), 665–675 (2006)
12. Sakellariou, R., Zhao, H.: A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems. In: Proc. of 18th International Parallel and Distributed Processing Symposium (IPDPS 2004), Santa Fe, New Mexico USA, April 2004, pp. 111–123 (2004)
13. Zhao, H., Sakellariou, R.: An Experimental Investigation into the Rank Function of the Heterogeneous Earliest Finish Time Scheduling Algorithm. In: Kosch, H., Böszörményi, L., Hellwagner, H. (eds.) Euro-Par 2003. LNCS, vol. 2790, pp. 189–194. Springer, Heidelberg (2003)
14. Dong, F., Akl, S.G.: A Joint Data and Computation Scheduling Algorithm for the Grid. In: Kermarrec, A.-M., Bougé, L., Priol, T. (eds.) Euro-Par 2007. LNCS, vol. 4641, pp. 587–597. Springer, Heidelberg (2007)
15. Wordwide LHC Grid Computing. Web page: `http://lcg.web.cern.ch/LCG/`
16. Daggen – Synthetic DAG Generation. `http://www.loria.fr/~suter/dags.html`
17. Canon., L.C., Jeannot, E.: A Comparison of Robustness Metrics for Scheduling DAGs on Heterogeneous Systems. In: 6th Int. Workshop on Algorithms, Models and Tools – HeteroPar 2007, Austin (2007)