

Local Search for Deadline Driven Grid Scheduling

Dalibor Klusáček, Luděk Matyska and Hana Rudová

Faculty of Informatics, Masaryk University
Botanická 68a, 60200 Brno
Czech Republic
{xklusac, ludek, hanka}@fi.muni.cz

Abstract. This work describes application of local search based algorithms for job scheduling in the Grid environment where dynamic changes occur. The primary intent is to consider problems with the typical quality of service constraint taking into account and minimizing the number of late jobs. To achieve this goal a special instance of the Tabu search algorithm applicable to dynamic problems is proposed. Also a new dispatching rule generating the initial solution is introduced. Comparison with typical queue-based policies such as First Come First Served, Earliest Deadline First or Easy Backfilling is provided. Experimental results shows that local search based algorithms is a promising technique with better performance than queue-based algorithms while still fast enough to provide solutions in a reasonable time.

1 Introduction

In this paper we describe application of dispatching rule [9] and Tabu search [10] algorithm for job scheduling in the Grid [8] environment where nontrivial QoS is required. Unlike to common production systems like PBS [6], Condor [20], LSF [22] or meta-scheduling systems such as Grid Service Broker [21], GridWay [12] and GRMS [11] that all use *queue-based* scheduling policies we are using *schedule-based* [13] approach which allows us to precisely map jobs onto machines in time. While queue-based systems maintain information only about resources this approach uses information about the run time of all jobs [13]. This allows us to use advanced scheduling algorithms [17, 9] to optimize the schedule—in this case to minimize the number of delayed jobs. We focus on local search methods [9] and dispatching rules [17] to achieve this goal.

This work describes in detail the dispatching rule used to create initial schedule and Tabu search algorithm design to optimise it wrt. objective function. Single QoS based objective function focusing on maximizing the number of jobs that meet their deadline [5] is used. We use a *dynamic environment* [3, 15, 18] where jobs are arriving over the time and disappear at their completion time. For this *dynamic problem* local search based optimization seems to be very suitable but according to our knowledge it was only applied to *static problems* where the assumption is that all the jobs and resources are *known in advance*

and the local search for all jobs must be performed en bloc [1, 2]. Since this can be very time consuming operation we always use previously computed schedule as the starting point for a new solution computation when new job arrives in the system. This helps us to keep the run time short. Experimental evaluation performed in our Grid scheduling simulation environment Alea [14] showed interesting results for local search algorithms by means of improvement in the quality of objective function and still good run time in comparison to common queue-based scheduling policies.

The structure of the paper is following. Next section formulates the main features of the Grid scheduling problem. Following section describes the algorithms used to create and optimize the schedule. Then we present some experimental results, and the last section concludes the paper and discusses the future work.

2 Problem Description

Let us briefly describe characteristics of the Grid scheduling problem [7]. In the static case, all information about scheduled jobs and resources are known before any scheduling process starts. Here the problem of n jobs and m resources is considered. More realistic case includes *dynamic behavior* of the system. Resources may change, jobs are not known in advance and they appear while other jobs were already scheduled and are running. In this paper, we still expect the fixed set of resources but we allow changes in the set of jobs. As the time is running, new jobs may appear and processing of other jobs is completed.

Each job is characterized with the release date (arrival time) r_j representing the time when the job appears in the system. Job deadline d_j is understood as a desired job completion time which should be kept. Job j has known processing time $p_{i,j}$ which depends on the CPU speed of machine i . Job also requires $R_{i,j}$ number of CPUs for its execution ($R_{i,j} > 0$). Resources are computational machines with known capacity R_i representing the number of CPUs. All CPUs within one machine has the same speed s_i . Different machines may have different speeds.

Various objective functions can be considered such as makespan or average flow time. In this work the goal of the scheduler is to maximize the number of jobs that meet their deadline [5]. The higher this number is the higher QoS is provided to the users, i.e., job owners. This value is represented by *unit penalty* function $U = \sum_{j=1}^n U_j$ where $U_j = 1$ if $C_j > d_j$ holds otherwise U_j equals to 0. Since C_j represents the job completion time, the unit penalty represents the number of late jobs. An important objective function we also consider is the *tardiness* given as a sum of $T_j = \max(C_j - d_j, 0)$ over all jobs j .

3 Algorithms

Here we will concentrate on the description of the algorithms which are used to generate and optimize the schedule. The *schedule* is composed of several *resource schedules*. Each resource schedule is a list of jobs planned to be executed on this

resource. Schedule represents jobs that *will* be executed on available resources, jobs already sent to resources are not affected by the scheduling process.

The algorithms are designed to minimize the unit penalty function U in the dynamic situation when new jobs are arriving into the system. Let us note that the final unit penalty U_j of each job j is known when this job is completed. Because we need this information during scheduling process even when all jobs are not known in advance, we need to approximate *current* unit penalty as $U_{expected}$.

$$U_{expected} = \sum_{i=1}^m \left(U_C[i] + \sum_{j=1}^{|\text{schedule}[i]|} (\text{expected } U_{i,j}) \right). \quad (1)$$

$U_C[i]$ stores *known* values of U_j of jobs already completed on resource i while $\sum_{j=1}^{|\text{schedule}[i]|} (\text{expected } U_{i,j})$ approximates expected unit penalty of currently known jobs *planned* to be executed on resource i . The value of *expected* $U_{i,j}$ can be approximated because the processing time $p_{i,j}$ is known as well as other important parameters such as d_j , $R_{i,j}$, R_i and s_i . Using this information and the current schedule it is possible to compute expected completion time C_j for each job present in the system at this moment.

Dispatching rule called *Minimum Tardiness Earliest Deadline First (MT-EDF)* is used to create *initial schedule* using previously computed schedule. It places incoming job into such resource where the expected unit penalty or expected tardiness after adding this job will be minimal.¹ In this resource the job is placed according to EDF strategy [17]. Detailed implementation of the algorithm is described at Fig. 1.

Optimization of existing schedule is performed by *Tabu Search (TS)* [10, 9]. Our implementation of Tabu search (see Fig. 2) optimizes existing schedule by moving tardy jobs from resource schedule with higher expected tardiness ($res_schedule_{max}$) to the resource schedule with smaller expected tardiness ($res_schedule_{min}$). Tardy *job* is moved by *MoveJob* procedure (see Fig. 3). If this move decreases $U_{expected}$ or at least decreases $T_{expected}$ then it is accepted otherwise the *job* is placed back into $res_schedule_{max}$. This is expressed by boolean expression: $U_{best} > U_{expected}$ **or** ($U_{best} = U_{expected}$ **and** $T_{best} > T_{expected}$). If we used $U_{expected}$ only as decision criterion then some moves that decrease tardiness would not be accepted which led us to worse results than this approach.

Dispatching rule always produces acceptable solution therefore the *TS* optimisation can be stopped at any time if fast decisions are required. This approach is crucial because it allows to deliver solution on time and even more, its quality may be improved wrt. the available time.

4 Experimental Evaluation

The Grid environment was simulated by our Alea Simulator [14] which is based on modified and extended GridSim [4] toolkit written in Java. The experiments

¹ Expected tardiness $T_{expected}$ is computed similarly to $U_{expected}$.

Procedure MTEDF(*job*, T_C , U_C)
schedule := [*res_schedule*₁, ..., *res_schedule*_{*m*}];
res_schedule := null; *res_schedule*_{*best*} := null; *k* := 0;
 T_{best} := MAX_VALUE; $T_{expected}$:= MAX_VALUE;
 U_{best} := MAX_VALUE; $U_{expected}$:= MAX_VALUE;

while(*k* < *m* **and** T_{best} > 0) **do**
 k := *k* + 1;
 res_schedule := *schedule*[*k*];
 place *job* into *res_schedule* on position *l* such: ($d_{l-1} \leq d_l < d_{l+1}$);
 $T_{expected} = \sum_{i=1}^m \left(T_C[i] + \sum_{j=1}^{|\text{schedule}[i]|} (\text{expected } T_{i,j}) \right)$;
 $U_{expected} = \sum_{i=1}^m \left(U_C[i] + \sum_{j=1}^{|\text{schedule}[i]|} (\text{expected } U_{i,j}) \right)$;
 if ($U_{best} > U_{expected}$ **or** ($U_{best} = U_{expected}$ **and** $T_{best} > T_{expected}$)) **then**
 $T_{best} := T_{expected}$;
 $U_{best} := U_{expected}$;
 *res_schedule*_{*best*} := *res_schedule*;
 remove *job* from *res_schedule*;
 place *job* into *res_schedule*_{*best*} on position *l* such: ($d_{l-1} \leq d_l < d_{l+1}$); //EDF

Fig. 1. MTEDF dispatching rule used for initial schedule generation.

Procedure TS(*rounds*, T_C , U_C)
schedule := [*res_schedule*₁, ..., *res_schedule*_{*m*}];
*schedule*_{*avail*} := [*res_schedule*₁, ..., *res_schedule*_{*m*}];
*schedule*_{*cand*} := [*res_schedule*₁, ..., *res_schedule*_{*m*}];
 $T_{best} := \sum_{i=1}^m \left(T_C[i] + \sum_{j=1}^{|\text{schedule}[i]|} (\text{expected } T_{i,j}) \right)$;
 $U_{best} := \sum_{i=1}^m \left(U_C[i] + \sum_{j=1}^{|\text{schedule}[i]|} (\text{expected } U_{i,j}) \right)$;
*res_schedule*_{*max*} := null; *k* := 0; *tabu_jobs* := \emptyset ;

while(*k* < *rounds* **and** T_{best} > 0 **and** *schedule*_{*avail*} $\neq \emptyset$) **do**
 k := *k* + 1;
 *res_schedule*_{*max*} := *schedule*[*i*] such that (*schedule*[*i*] \in *schedule*_{*avail*} **and**
 $T_C[i] + \sum_{j=1}^{|\text{schedule}[i]|} (\text{expected } T_{i,j})$ is maximal **and**
 $i \in \{1, \dots, m\}$);
 if \exists *job* such that (*job* \in *res_schedule*_{*max*} **and** *job* \notin *tabu_jobs* **and** $T_{job} > 0$) **then**
 job := *job* such that (*job* \in *res_schedule*_{*max*} **and** *job* \notin *tabu_jobs* **and**
 T_{job} is maximal);
 MoveJob(*job*, *res_schedule*_{*max*}, *schedule*_{*cand*}, *tabu_jobs*, T_C , T_{best} , U_{best});
 else
 *schedule*_{*avail*} := *schedule*_{*avail*} \setminus {*res_schedule*_{*max*}};
 *schedule*_{*cand*} := [*res_schedule*₁, ..., *res_schedule*_{*m*}];

Fig. 2. Tabu search optimization procedure.

```

Proc. MoveJob(job, res_schedulemax, schedulecand, tabu_jobs, TC, Tbest, Ubest)
res_schedulemin := null; Texpected := MAX_VALUE; Uexpected := MAX_VALUE;
schedule := [res_schedule1, ..., res_schedulem];

res_schedulemin := schedule[i] such that (schedule[i] ≠ res_schedulemax and
    schedule[i] ∈ schedulecand and i ∈ (1, ..., m) and
     $T_C[i] + \sum_{j=1}^{|\text{schedule}[i]|} (\text{expected } T_{i,j})$  is minimal);
if res_schedulemin = null then
    tabu_jobs := tabu_jobs ∪ job; //replace oldest item if tabu_jobs is full
    schedulecand := [res_schedule1, ..., res_schedulem];
else
    remove job from res_schedulemax; //put in res_schedulemin using EDF
    place job into res_schedulemin on position l such that (dl-1 ≤ dl < dl+1);
     $T_{\text{expected}} = \sum_{i=1}^m \left( T_C[i] + \sum_{j=1}^{|\text{schedule}[i]|} (\text{expected } T_{i,j}) \right)$ ;
     $U_{\text{expected}} = \sum_{i=1}^m \left( U_C[i] + \sum_{j=1}^{|\text{schedule}[i]|} (\text{expected } U_{i,j}) \right)$ ;
    if (Ubest > Uexpected or (Ubest = Uexpected and Tbest > Texpected)) then
        Tbest := Texpected;
        Ubest := Uexpected;
        scheduleavail := [res_schedule1, ..., res_schedulem];
        schedulecand := [res_schedule1, ..., res_schedulem];
    else
        remove job from res_schedulemin;
        put job back into res_schedulemax;
        schedulecand := schedulecand \ {res_schedulemin};

```

Fig. 3. Procedure *MoveJob* managing local changes in the *schedule*.

were executed on Intel Pentium 4 2.6 GHz machine with 512 MB RAM. Simulation was performed for 1500 dynamically arriving jobs and 150 machines with different CPU count/speed. Both parallel and sequential jobs were simulated. Our primary optimization criterion takes into account the deadlines—the typical quality of service constraint which allows to guarantee the execution of a job before the given time instant. Experiments were processed on 7 different arrival-time distributions representing 7 different loads of the system each with 20 different data sets which were generated synthetically using exponential distribution with the generator from [5, 19]. In the first case jobs were arriving frequently so the number of jobs waiting for execution was very high. Remaining cases had stepwise higher average job inter-arrival time so the load of the system was lower. Tabu search performance was compared with *First Come First Served (FCFS)*, *Earliest Deadline First (EDF)* and *Easy Backfilling (BF)* [16, 19] as the representatives of scheduling policies used by current production queuing systems. The results showed that our schedule-based approach reduces average number of late jobs for all loads in comparison with the best results achieved by the queue-based algorithms (see Fig. 4). Although the Tabu search takes much longer than the queue-based algorithms, the absolute time to provide a solution

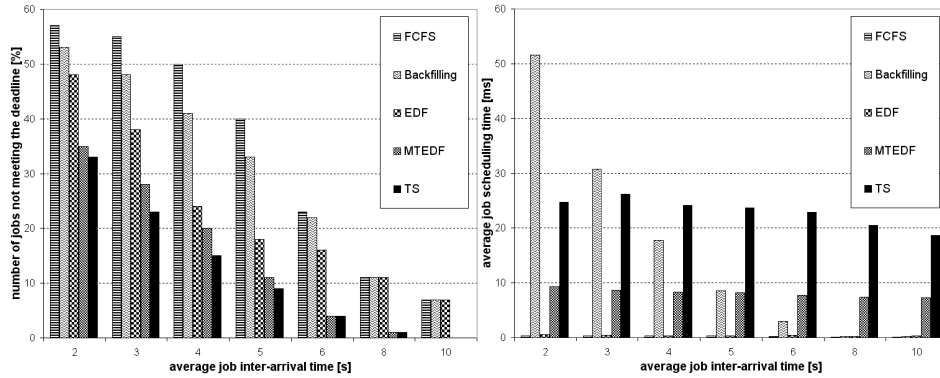


Fig. 4. Unit penalty U results (i.e., jobs not meeting their deadline – on the left) and average scheduling time i.e., algorithm run time (right).

is very acceptable (22.8 ms per job in average). This is in contrast to the Backfilling algorithm which was very time consuming when the queue contained a lot of jobs. We also focused on the algorithms scalability regarding queue/schedule size and algorithm run time. In this case the schedule-based solution proved to be quite stable in contrast to the linear grow of run time in the case of the Backfilling (see Fig. 5). It shows the correlation between queue/schedule size and time required to add or select the job. It is not surprising that addition of a job into the schedule or optimization itself takes longer than simple addition of a job into the queue. On the other hand selecting a job from the schedule is always fast operation in contrast to the Backfilling where the time grows lineary with the size of the queue.

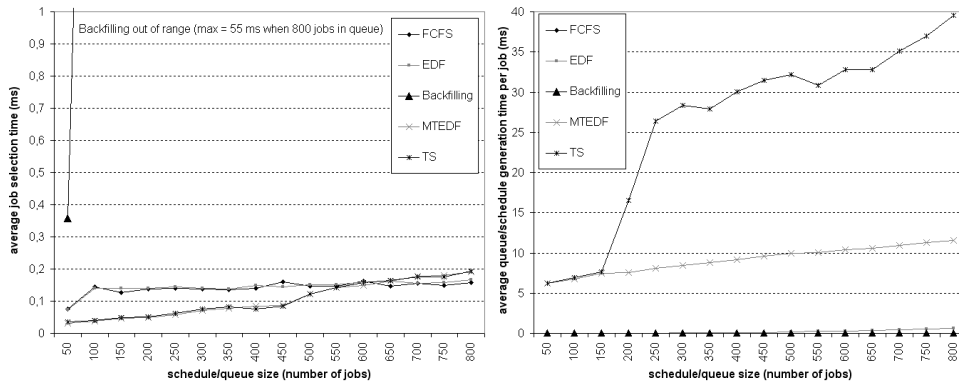


Fig. 5. Time required to select job to be run wrt. queue/schedule size. Notice quick growth (out of scale) of the Backfilling run time (left). Average time needed to modify queue/schedule when new job arrive or schedule is optimised by the TS (right).

5 Conclusion and Future Work

Incremental schedule-based algorithms demonstrated significant improvement when decreasing the number of late jobs with still very good run time of Tabu search over common queue-based policies. The Tabu search algorithm demonstrates promising direction in development of new efficient local search based algorithms applicable for Grid scheduling.

In the future we would like to optimize the local search algorithms to reflect parallel job features—information about the resource load can be used to move jobs to the time where the capacity of resource is not fully occupied by currently scheduled jobs. Also, we would like to introduce network simulation, failure tolerance, preemptivity and job migration. Moreover new objective functions such as makespan, resource usage or average flow time must be implemented. We also plan to compare Tabu search performance with different scheduling techniques such as Flexible Backfilling [19] or Convergent Scheduling [5].

Acknowledgment

This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic under the research intent No. 0021622419, and by the Grant Agency of the Czech Republic with grant No. 201/07/0205 which we highly appreciate. We would like to thank Ranieri Baraglia and Gabriele Capannini from CNR-ISTI for providing us the data set generator from [5, 19].

References

1. Vinicius A. Armentano and Denise S. Yamashita. Tabu search for scheduling on identical parallel machines to minimize mean tardiness. *Journal of Intelligent Manufacturing*, 11:453–460, 2000.
2. Ranieri Baraglia, Renato Ferrini, and Pierluigi Ritrovato. A static mapping heuristics to map parallel applications to heterogeneous computing systems: Research articles. *Concurrency and Computation: Practice and Experience*, 17(13):1579–1605, 2005.
3. Julien Bidot. *A General Framework Integrating Techniques for Scheduling under Uncertainty*. PhD thesis, Institut National Polytechnique de Toulouse, France, 2005.
4. Rajkumar Buyya and Manzur Murshed. GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing. *The Journal of Concurrency and Computation: Practice and Experience (CCPE)*, 14:1175–1220, 2002.
5. G. Capannini, R. Baraglia, D. Puppini, L. Ricci, and M. Pasquali. A job scheduling framework for large computing farms. In *SC07 International Conference for High Performance Computing, Networking, Storage and Analysis*, 2007. To appear.
6. Hanhua Feng, Vishal Misra, and Dan Rubenstein. PBS: A unified priority-based CPU scheduler. Technical report, Computer Science Department, Columbia University, 2006.

7. Pavel Fibich, Luděk Matyska, and Hana Rudová. Model of Grid Scheduling Problem, Exploring Planning and Scheduling for Web Services, Grid and Autonomic Computing, Papers from the AAAI-05 workshop. Technical Report WS-05-03, AAAI Press, 2005.
8. Ian Foster and Carl Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure, second edition*. Morgan Kaufmann, 2004.
9. Fred W. Glover and Gary A. Kochenberger, editors. *Handbook of Metaheuristics*. Kluwer, 2003.
10. Fred W. Glover and Manuel Laguna. *Tabu Search*. Kluwer Academic Publishers, 1998.
11. J. Huang, Y. Wang, N.R. Vaidyanathan, and F. Cao. GRMS: A global resource management system for distributed QoS and criticality support. In *ICMCS '97: Proceedings of the 1997 International Conference on Multimedia Computing and Systems (ICMCS '97)*, pages 424–432, Washington, DC, USA, 1997. IEEE Computer Society.
12. Eduardo Huedo, Rubén Montero, and Ignacio Llorente. The GridWay framework for adaptive scheduling and execution on Grids. *Scalable Computing: Practice and Experience*, 6(3):1–8, 2005.
13. Axel Keller and Alexander Reinefeld. Anatomy of a resource management system for HPC clusters. *Annual Review of Scalable Computing*, 3, 2001.
14. Dalibor Klusáček, Luděk Matyska, and Hana Rudová. Grid Scheduling Simulation Environment. In *Workshop on scheduling for parallel computing at the 7th International Conference on Parallel Processing and Applied Mathematics (PPAM 2007)*, Lecture Notes in Computer Science. Springer, 2007. To appear.
15. Waldemar Kocjan. Dynamic scheduling state of the art report. Technical report, SICS Technical Report T2002:28, 2002.
16. David A. Lifka. The ANL/IBM SP Scheduling System. In *IPPS '95: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*, pages 295–303, London, UK, 1995. Springer-Verlag.
17. Michael Pinedo. *Planning and Scheduling in Manufacturing and Services*. Springer, 2005.
18. Rizos Sakellariou and Henan Zhao. A low-cost rescheduling policy for efficient mapping of workflows on grid systems. *Scientific Programming*, 12(4):253–262, 2004.
19. A. D. Techiouba, Gabriele Capannini, Ranieri Baraglia, Diego Puppini, and Marco Pasquali. Backfilling strategies for scheduling streams of jobs on computational farms. In *CoreGRID Workshop on Grid Programming Model, Grid and P2P Systems Architecture, Grid Systems, Tools and Environments*. Springer, 2007. To appear.
20. Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: the Condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.
21. Srikumar Venugopal, Rajkumar Buyya, and Lyle Winton. A Grid Service Broker for scheduling distributed data-oriented applications on global Grids. In *MGC '04: Proceedings of the 2nd workshop on Middleware for grid computing*, pages 75–80, New York, NY, USA, 2004. ACM Press.
22. Ming Q. Xu. Effective metacomputing using LSF multicluster. In *CCGRID '01: Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, pages 100–105, Washington, DC, USA, 2001. IEEE Computer Society.