On-line Hierarchical Job Scheduling on Grids with Admissible Allocation

Andrei Tchernykh[1]*, Uwe Schwiegelshohn[2], Ramin Yahyapour[3], Nikolai Kuzjurin[4]

[1] Computer Science Department, CICESE Research Center Ensenada, BC, México 22830, chernykh@cicese.mx

[2] Robotics Research Institute, Technische Universität Dortmund, 44221 Dortmund, Germany, uwe.schwiegelshohn@udo.edu

[3] IT and Media Center, Technische Universität Dortmund, 44221 Dortmund, Germany, ramin.yahyapour@udo.edu

[4] Institute of System Programming RAS, Moscow, nnkuz@ispras.ru

SUMMARY

In this paper, we address non preemptive online scheduling of parallel jobs on a Grid. Our Grid consists of a large number of identical processors that are divided into several machines. We consider a Grid scheduling model with two stages. At the first stage, jobs are allocated to a suitable machine while at the second stage, local scheduling is independently applied to each machine. We discuss strategies based on various combinations of allocation strategies and local scheduling algorithms. Finally, we propose and analyze a scheme named adaptive admissible allocation. This includes competitive analysis for different parameters and constraints. We show that the algorithm is beneficial under certain conditions and allows an efficient implementation in real systems. Furthermore, a dynamic and adaptive approach is presented which can cope with different workloads and Grid properties.

KEY WORDS: Grid Computing, Online Scheduling, Resource Management, Job Allocation

1. Introduction

An increasing number of scientific disciplines collaborate in virtual organizations to jointly address complex research problems and share computational resources. The dynamic nature of such virtual organizations requires flexible resource provisioning that is realized with Computational Grids and Clouds. Due to the size and dynamicity of Grids, we need an automatic and efficient process to allocate computational jobs to available resources in the Grid. Various scheduling systems have already been proposed and implemented in different types of Grids (Krauter, Buyya, and Maheswaran 2002, Rodero et al. 2005, Rodero et al. 2008, Elmroth and Tordsson 2005, Avellino et al. 2003). However, there are still many open issues in this field, including the consideration of multiple layers of scheduling. Most academic studies either propose a completely distributed resource management system, see, for instance (Ernemann and Yahyapour 2003), or suggest a central scheduler, see (Ernemann, Hamscher et al. 2002), while real installations favor a combination of decentralized and centralized structures, see (Vazquez-Poletti et al. 2007). A hierarchical multilayer resource management can represent such a combined system well (Schwiegelshohn and Yahyapour 2003, Kurowski et al. 2008). The highest layer is often called a Grid-level scheduler that may have a more general view of the job requests while specific details about the state of the resources remain hidden from it. The management of a specific resource is task of a local resource management system that only knows all details of its machine and only about the jobs that are actually forwarded to it. In practice, other layers may exist in between. At every layer, different constraints and specifications must be considered. Thus, an efficient resource management system for Grids requires a suitable combination of scheduling algorithms that support such multilayer structures of resource management.

In this paper, we discuss a basic two layer online Grid scheduling model. At the first layer, we allocate a job to a suitable machine using a given selection criteria. At the second layer, potentially different parallel scheduling algorithms are applied to these allocated jobs at each machine. Typically, Grid resources are only connected by wide area networks and do not share the

same management system. Therefore, job migration between different resources may incur significant overhead and is technically challenging. Hence, we do not allow a job migration after a job has been allocated to a machine, independent whether it has been started or not. That is, an allocated job must be executed on the assigned machine. Similarly, we do not consider multi-site execution.

The scheduling of jobs on multiprocessors is generally well understood and has been studied for decades. Many research results exist for different variations of this single system scheduling problem; some of them provide theoretical insights while others give hints for the implementation of real systems. However, the online machine allocation problem has rarely been addressed so far. Unfortunately, it may result in inefficient machine utilization in the worst case; see (Tchernykh et al. 2005). One of the structural reasons for the inefficiency in online job allocation is the occupation of large machines by jobs with small processor requirements causing highly parallel jobs to wait for their execution. This problem is addressed in this paper.

To this end, we use a simple model that focuses on some key aspects of Grids. The jobs are submitted over time and must be allocated to a machine immediately after submission. In order to hide transmission latencies, the job transfer is initiated as soon as possible even if the job cannot start immediately. Further, we assume rigid parallel jobs, that is, the jobs have a given degree of parallelism and must be assigned exclusively to a specified number of processors or cores during their whole executions. This is typical for optimized and communication intensive parallel applications. While the machines in a real Grid often exhibit different forms of heterogeneity, like different hardware, operating system and software, we restrict ourselves to machines with different numbers of the same processors or cores. Due to the advance of virtualization, differences in the operating systems become less important. Moreover, processors tend to differ mainly in the number of cores while the architectures of individual cores and their clock frequency are rather similar. Therefore, we believe that the focus in our model is reasonable although it neither matches every real installation nor all real applications.

From a system point of view, it is typically the goal of a Grid scheduler to achieve some kind of load balancing in the Grid. In scheduling theory, this goal is commonly represented by the objective of makespan minimization. Although the makespan objective is mainly an offline criterion and has shortcomings particularly in online scenarios with independent jobs, see (Schwiegelshohn, 2009), it is easy to handle and frequently used in theoretical evaluations, see, for instance, (Albers 1999). As we want to compare our results with results of multiprocessor scheduling research, we also apply the makespan objective in this paper. Similarly, for reasons of comparability, we use a worst case analysis by determining the competitive factor of online algorithms. Similar to the approximation factor in NP-hard deterministic scheduling problems the competitive factor is the ratio between objective values of the determined online schedule to the best possible schedule in the worst case.

We continue this paper by formally presenting our Grid scheduling model in Section 2. After a discussion of related work in Section 3 we introduce our algorithms and classify them in Section 4. Then we discuss an adaptive two-level scheduling strategy and analyze it. Finally, we conclude with a summary and an outlook in Section 6.

2. Model

We address an online scheduling problem with the objective of minimizing the makespan: n parallel jobs J_1, J_2, … must be scheduled on m parallel machines N_1, N_2, … , N_m. Let m_i be the number of identical processors of machine N_i also called the size of machine N_i. Assume without loss of generality that the parallel machines are arranged in non-descending order of their sizes $m_1 \le m_2 \le ... \le m_m$.

Each job J_j is described by a triple $(r_j, size_j, p_j)$: its release date $r_j \ge 0$, its size $1 \le size_j \le m_m$ that is referred to as its processor requirements or degree of parallelism, and its execution time p_j. Further, $w_j = p_j \cdot size_j$ is the work of job J_j, also called its area in the schedule or its resource consumption. The properties of a job only become known at its release date. The jobs are submitted over time and must be immediately and irrevocably allocated to a single machine. This machine must execute the job by exclusively allocating exactly $size_j$ processors for an uninterrupted period of time p_j to it. As we do not allow multi-site execution and co-allocation of processors from different machines, a job J_j can only run on machine N_i if

$size_j \le m_i$ holds. We use $g_j = i$ to denote that job J_j is allocated to machine N_i. If possible without causing confusion we use the index i to specify machine N_i. Further, let Z_i be the job set allocated to machine N_i. Therefore, the total work of a given job set Z is $W_Z = \sum_{J_j \in Z} w_j$.

The completion time of job J_j of instance I in a schedule S is denoted by $C_j(S, I)$. As already mentioned, we determine the makespan $C_{\max}(S, I) = \max_{J_j}\{C_j(S, I)\}$ of a schedule S and an instance I. The optimal makespan of instance I is denoted by $C^*_{\max}(I)$. Where possible without causing ambiguity we will omit instance I.

The competitive factor of algorithm A is defined as $\rho_A = \sup_I \frac{C_{\max}(S_A, I)}{C^*_{\max}(I)}$ over all problem instances. Again, we omit algorithm A if it does not cause any ambiguity.

We describe our Grid machine model by GP_m. In the short three field notation *machine_model*|*constraints*|*objective* proposed by (Graham et al. 1979), the scheduling problem is characterized as $GP_m \mid r_j, size_j \mid C_{\max}$. We use the notation *MPS* (Multiple Parallel Scheduling) to refer to this problem while the notation *PS* (Parallel Scheduling) describes the parallel job scheduling on a single parallel machine $P_m \mid r_j, size_j \mid C_{\max}$.

3. Related Work

Before going into details we add some general remarks about the competitive bounds of *MPS*. We regard *MPS* as a two stage scheduling strategy: *MPS*=*MPS_Alloc*+*PS*. At the first stage, we allocate a suitable machine for each job using a given selection criterion and the *MPS_Alloc* strategy. At the second stage, algorithm *PS* is applied to each machine independently for jobs allocated during the previous stage. It is easy to see that the competitive factor of the *MPS* algorithm is lower bounded by the competitive factor of the *PS* algorithm. Just consider a degenerated Grid that only contains a single machine. In this case, the competitive factors of the *MPS* algorithm and the *PS* algorithm are identical as there is no need for any allocation stage. But clearly, an unsuitable allocation strategy may produce worse competitive factors. Just assume that all jobs are allocated to a single machine in a Grid with *k* identical machines. Obviously, in this case the competitive factor is upper bounded by the competitive factor of the *PS* algorithm times *k*.

The best *PS* online non-clairvoyant algorithm known so far has a tight competitive factor 2-1/m with *m* denoting the number of processors in the parallel system; see (Naroska and Schwiegelshohn 2002). Hence, the lower bound of a competitive factor for any general two-layer online *MPS* is at least 2-1/m.

(Schwiegelshohn et al. 2008) showed that there is no polynomial time algorithm that guarantees schedules with a competitive bound <2 for $GP_m \mid r_j, size_j \mid C_{\max}$ and all problem instances unless *P*=*NP*. Therefore, the multiprocessor list scheduling bound of 2-1/m for concurrent submission, see (Garey and Graham 1975), as well as for online submission, does not apply to Grids. Even more, list scheduling cannot guarantee a constant competitive bound for all problem instances in the concurrent submission case (Tchernykh et al. 2005).

(Schwiegelshohn et al. 2008) showed that the performance of Garey and Graham's list scheduling algorithm is significantly worse in Grids than in multiprocessors. They also present an online non-clairvoyant algorithm that guarantees a competitive factor of 5 for the Grid scenario, where all available jobs can be used for local scheduling, that is, migration between machines is allowed. Hence, this approximation algorithm guarantees to generate a schedule with completion time being within a constant ratio 5 of the optimal solution.

The offline non-clairvoyant version of this algorithm has a competitive factor of 3. This "one-layer" algorithm can either be implemented in centralized fashion or by a distributed "job stealing" approach. Although jobs are allocated to a machine at their submission times they can migrate before they have been started if another machine becomes idle.

(Pascual et. al. 2008) addressed a related problem, modeled an offline system consisting of *N* clusters with exactly *m* identical processors each, and proposed an algorithm with a guaranteed worst-case performance ratio on the global makespan equal to 4. They also showed that a better bound 3 can be obtained in a specific case when the last completed job requires at most *m*/*2* processors.

(Tchernykh et al. 2005; Zhuk et al. 2004) considered the offline case and addressed the performance of various 2-stage algorithms with respect to the makespan objective. They present algorithms with a competitive factor of 10. One of the known reasons of the inefficient online job scheduling in a two layer hierarchical online Grid scheduling model is the occupation of large machines by jobs with small processor requirements causing highly parallel jobs to wait for their execution. (Tchernykh et al. 2008) addressed this problem and presented an algorithm showing a competitive factor variation between 5 and infinity for specific assumptions of the workload characteristics and changes of the so called admissible factor. In our work, we extend these results considering a more general model.

4. Classification of Algorithms

As already discussed, the Grid scheduling algorithm can be split into a global allocation part and a local scheduling part. We can distinguish different strategies depending on the type and amount of information they require. This classification allows a first performance analysis.

4.1. Job Allocation

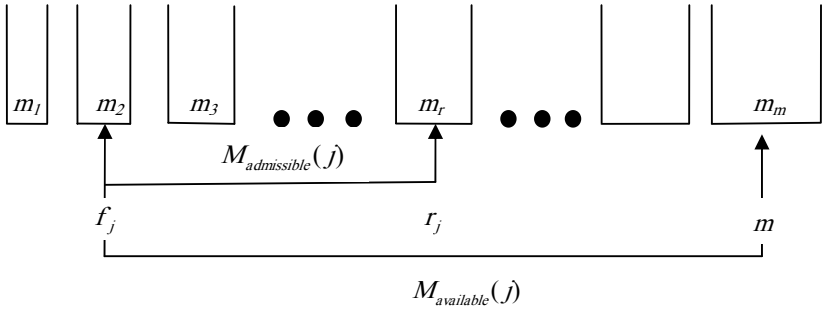

Figure 1. Concept of the available and admissible machines for allocation of job J_j

Remember that machines are indexed in non-descending order of their sizes. We define $f_j = i$ if machine N_i is the first machine with $m_i \geq size_j$. Note that due to our restriction $size_j \leq m_m \; \forall J_j$, we have $i \leq m$. The set of machines $M_{available}(j)$ that are available for allocation of job J_j corresponds to the set of machine indexes $\{f_j, f_j + 1, ..., m\}$, see Fig. 1. Obviously, the total set of machines M_{total} is represented by the integer set $\{1, ..., m\}$. An algorithm may not be allowed to use all available machine for allocation of job J_j. Therefore, we define a set of admissible machines $M_{admissible}(j)$ that is a subset of $M_{available}(j)$. In the example of Fig. 1, $M_{admissible}(j)$ is represented by the index set $\{f_j, ..., r_j\}$.

Remember that the number and the sizes of the parallel machines are known. Further, we distinguish three different levels of additionally available information for job allocation.

Level 1: There is no information on the processing time of any job, that is, we consider non clairvoyant scheduling. Obviously, no strategy can guarantee a constant competitive factor in our model. However, it is possible to achieve a constant competitive factor if machines may exchange jobs that are not yet started. An example for such a job stealing model has been analyzed by (Schwiegelshohn et al. 2008).

Level 2: Once a job has been submitted its parallelism $size_j$ and its processing time p_j are known. However, the GRMS (Grid Resource Management and Brokering Service) has no information on the local schedule of the machines. But it may use information on the already allocated load to each machine.

Level 3: We have access to all information of Level 2 and to all local schedules as well. The GRMS may allocate a job to the machine where it has the earliest completion time.

In this paper, we show that Level 2 algorithms can already achieve constant competitive factors.

4.2. Parallel Machine Scheduling

Once a job has been allocated to a parallel machine, the local resource management system (*LRMS*) of this machine will generate a schedule. Different scheduling algorithms may be used by the LRMS: Many real systems apply First-Come-First-Serve (*FCFS*) that schedules jobs in the order of their arrival times. Although this algorithm performs reasonably well in practice it obviously cannot guarantee a constant competitive factor. This is also true for the various versions of backfilling that usually improve the performance of FCFS for real workloads. Alternatively, list scheduling produces a competitive factor of $2-1/m$ for a parallel machine with m processors even in the non-clairvoyant case; see (Naroska and Schwiegelshohn, 2002). To the best of our knowledge no better competitive factor is known for the clairvoyant case of scheduling rigid parallel jobs. But the best lower bound for the online scheduling of sequential jobs on parallel machines is already 1.88; see (Rudin, 2001). In the remaining parts of this paper, we assume that the LRMS use an online parallel scheduling algorithm with the competitive factor 2.

5. Online Scheduling of Parallel Jobs in a Grid

In this section we provide a competitive analysis for two heuristics. The first heuristic is based on the online load balancing model of (Bar-Noy et al. 2002). We show how this model can be adapted to online makespan scheduling of parallel jobs in a Grid. The second heuristic extends a relatively simple scheme called *admissible allocation* that was proposed by (Tchernykh et al. 2005) and can be efficiently implemented in real systems.

The bad case example given in (Tchernykh et al. 2005) shows that one reason of the inefficient online job allocation may be the occupation of large machines by jobs with a small degree of parallelism causing highly parallel jobs to wait for their execution.

Both approaches exclude certain machines with many processors from the set of machines available to execute jobs with little parallelism.

Let $m_{f,r}=\sum_{i=f}^{r} m_i$ be the total number of processors belonging to machines m_f to m_l. (Tchernykh et al. 2005) defined the set $M_{admissible}(j)$ for a job J_j to be the machines with indexes $\{f_j,...,r_j\}$, where r_j is defined as the smallest machine index with $m_{f_j} \geq \frac{1}{2}\cdot m_{f_j,m}$ (see Fig. 1). Note, that $r_j \leq m$ always holds.

In this paper, the definition is generalized by introducing a new parameter $0 < a \leq 1$ that parameterizes the admissibility ratio used for the job allocation. Hence, we call $\{f_j,...,r_j\}$ the index set of admissible machines if r_j is the minimum index with $m_{f_j} \geq a \cdot m_{f_j,m}$. Note that at least one machine is admissible as a is strictly larger than 0 while $a=1$ defines all available machines.

5.1. Analysis of the Load Balancing Scheme

In this section, we provide a theoretical worst case analysis of online scheduling for our model. As already mentioned, the analysis extensively uses the results of (Bar-Noy et al. 2002). They addressed online load balancing in a hierarchical server model with m identical machines. In their model, all servers have different priorities out of the set $\{1,2,...,m\}$. A job J_j can only be executed on its eligible servers, which are its specified target server s_j and all servers with a higher priority than s_j. Further, each job J_j contributes a weight w_j to its allocated server. It is the objective to minimize the maximum weight on the most loaded server. For our problem, the fractional and the weighted integral model are of specific interest. In the fractional model, the weight of a job can be arbitrarily distributed to any eligible server for this job. This model is similar to the divisible load model in makespan scheduling. In the weighted integral model, a job must be allocated to a single eligible server. (Bar-Noy et al. 2002) proposed a uniform memoryless algorithm for the fractional model. A uniform memoryless algorithm uses the same allocation scheme for all jobs independent of the actual job and the previously allocated job. A function $u(s)$

assigns to every server s a number of the interval $[0,1]$. Processing the servers in priority order and starting with s_j, the fraction $u(s)$ of the job weight w_j is executed on server s provided that the remaining weight is sufficiently large. (Bar-Noy et al. 2002) showed in Proposition 7 that the assignment function $u(s) = \frac{1}{m+1-s}$ with m being the highest priority server achieves a competitive factor of e.

There are two main differences between our model and the model of (Bar-Noy et al. 2002):

1. We apply the makespan criterion.
2. We use parallel machines and rigid parallel jobs without multisite allocation.

Theorem 1 *Online scheduling of rigid parallel jobs on Grids with identical processors can be achieved with the competitive factor* $2e+1$.

Proof: (Naroska and Schwiegelshohn 2002) described the lower bound

$$C^*_{\max} \geq \hat{C}^*_{\max} = \max_{0 \leq t \leq \max_j \{p_j + r_j\}} \{t + \frac{1}{m} \sum_{J_j | p_j + r_j > t} size_j \cdot \min\{p_j, p_j + r_j - t\}\}$$

for makespan scheduling of rigid parallel jobs on m identical machines.

In the Grid, we have m machines with potentially many processors. Remember that we arrange the machines in order of increasing numbers of processors. Then we apply the hierarchical server model of (Bar-Noy et al. 2002) to assign pair wise different processor priorities $\{1,2,...,m_{1,m}\}$ using this order. The priority of a machine is the smallest priority of its processors. Therefore, if a machine is eligible to execute a job J_j then all of its processors are eligible to execute part of J_j and each machine with a higher priority is also eligible to execute J_j. Next, we adapt the above mentioned lower bound to Grids:

$$\hat{C}^*_{\max} = \max_{0 \leq t \leq \max_j \{p_j + r_j\}} \{t + \max_i \{\frac{1}{m_{i,m}} \sum_{J_j | p_j + r_j > t, f_j \geq i} size_j \cdot \min\{p_j, p_j + r_j - t\}\}\}$$

We extend our problem instance I to a new problem instance I' by adding new sequential jobs ($r_j = 0$, $size_j = 1$, sufficiently small $p_j, f_j = 1$) such that $\hat{C}^*_{\max} = \frac{1}{\hat{m}} \sum_{J_j} size_j \cdot p_j$ holds.

We interpret $size_j \cdot p_j$ as the weight w_j of job J_j and apply the fractional algorithm of (Bar-Noy et al. 2002) on instance I' directly. The algorithm produces a maximum processor load $L^{frac}_{\max}(I')$ with

$$\hat{C}^*_{\max} \leq L^{frac}_{\max}(I') \leq e \cdot \hat{C}^*_{\max}$$

The fractional model can easily be transformed into a weighted integral model with the help of a reference load. However, this approach prohibits the existence of temporary jobs, that is, the reference load on any processor must never decrease. Therefore, we assume an instance I' to generate the reference model:

1. Whenever a new job J_j with $f_j = i$ arrives we first determine the new reference load of the processors:
   a. We add the load $size_j \cdot p_j$ of job J_j.
   b. Let $L_i(t)$ be the total load on machines N_k at time t with $k \leq i$. If the new release date r_j is larger than the previous release date r' then we add the minimum amount of load to machine 1 such that for each $1 \leq i \leq m$ we have

$$\frac{1}{m_{1,m}}(L_k(r_j) - \sum_{J_h | p_h + r_h > r_j, f_h \leq k} size_h \cdot \min\{p_h, p_h + r_h - r_j\}) \geq r_j$$

   Remember that we may have added some load already in previous steps.

c. The new reference load of each processor is obtained by using the fractional algorithm to add the new load to the processors.

2. The new reference load of each machine is the sum of the reference loads of its processors.
3. A machine is overloaded if its load is greater than its reference load.
4. Job J_j is allocated to the eligible machine with the lowest priority that is not overloaded.

(Bar-Noy et al. 2002) have shown in Proposition 8 that this method guarantees for any processor s that the sum of the fractional loads of all processors s' with $s' \geq s$ is never smaller than the sum of the integral weighted loads of the same servers. Therefore, there is at least one eligible machine that is not overloaded for job J_j. Note that no machine is overloaded if for each overloaded machine we remove the last job allocated to it.

We apply list scheduling to schedule the jobs allocated to a machine. Clearly, the maximum load on a processor of N_i is not larger than the average load of all its processors. Therefore, if machine N_i is not overloaded we have

$$C_{\max}(N_i, I) < 2 \cdot \hat{C}^*_{\max}(N_i, I) \leq 2 \cdot \hat{C}^*_{\max} \leq 2 \cdot L^{frac}_{\max}(I') \leq 2e \cdot \hat{C}^*_{\max}$$

Note that the first inequality is due to Theorem 4 in (Naroska and Schwiegelshohn 2002).
Finally, we need to add at most one job to each machine to produce the final schedule. This cannot increase the makespan of this machine by more than $\hat{C}^*_{\max}$ leading to

$$C_{\max}(I) \leq (2e+1) \cdot \hat{C}^*_{\max} \leq (2e+1) \cdot C^*_{\max}(I).$$

■

5.1. Analysis of the Admissible Allocation Scheme

In this section, we consider the allocation strategy *Min_LB*$_a$ that allocates a new job J_j to the least loaded machine in $M_{admissible}(j)$ (Tchernykh et al. 2005) already showed that this scheme cannot guarantee a constant competitive factor if all eligible machines of a job are also admissible.

Let us therefore assume that job J_j is allocated to a machine from the set of admissible machines $f,...,r$ that contains $a \cdot m_{f,m}$ processors (see Fig. 2). Hence, $(1-a) \cdot m_{f,m}$ processors are excluded from $m_{f,m}$ processors available for allocation of job J_j.

Let us also assume that a job J_o with minimum processor requirements among all jobs allocated to these machines has the smallest machine index f_o. Hence, all jobs allocated to machines $f,...,r$ can be scheduled only on machines $f_0,...,m$. In this case the makespan of an optimal schedule of all jobs in a Grid is lower bounded by an optimal schedule of jobs set $Z_f \ldots Z_r$ on machines $f_0,...,m$.

Let schedules S_i, for machines $i = 1,...,m$ be the schedules generated by *Min_LB*$_a$*+PS* scheme. C_i denotes the makespan of schedule S_i. Further, we use $\hat{C}^*_i, \hat{C}^*_{f..r}$, and $\hat{C}^*_{f_o..m}$ to describe the lower bounds of the optimal schedules for machine *i*, machines $f,...,r$, and machines $f_0,...,m$ respectively. Remember that for $\hat{C}^*_{f..r}$, and $\hat{C}^*_{f_o..m}$, we consider only the job sets $Z_f \ldots Z_r$.

Remember that $\hat{C}^*_{\max}$ is based on an average workload and an offset t. Adding more processors will obviously distribute the workload but not the offset. This yields the relation

$$\hat{C}^*_{f_o..m} \geq \hat{C}^*_{f..r} \cdot \frac{m_{f,r}}{m_{f_o,m}}.$$

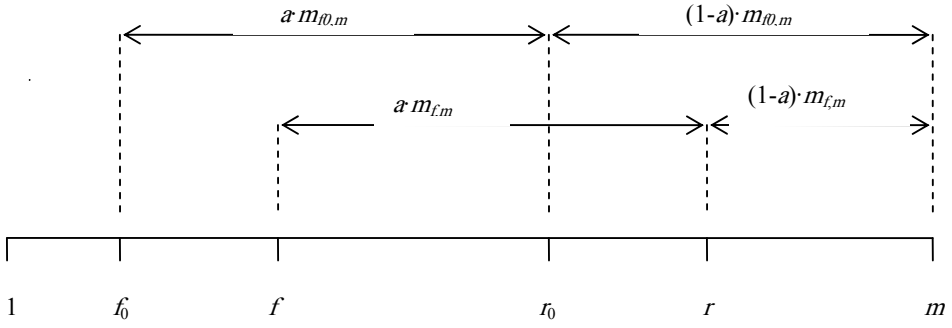

Figure 2. An example of admissible processors for allocation of jobs with factor a

Theorem 2. *Online scheduling of rigid parallel jobs on Grids with identical processors using the scheme Min_LB$_a$+PS with an admissible allocation range* $0 < a \le 1$ *has the competitive factor*

$$\rho \le \begin{cases} 1+\dfrac{4}{a^2} & \text{for } a \le \dfrac{m_{f,m}}{m_{f_0,m}} \\ 1+\dfrac{4}{a(1-a)} & \text{for } a > \dfrac{m_{f,m}}{m_{f_0,m}}, \end{cases}$$

where $1 \le f_0 \le f \le m$ (*see Fig. 2*) *are parameters that depend on the machine configuration and workload.*

Proof. Let us assume that the makespan of machine k is also the makespan $C_{\max}$ of the Grid, that is $C_k = C_{\max}$. Further, let job J_d be the last job that was added to this machine. We can assume that the completion time of job J_d determines the makespan C_k as otherwise we can delete job J_d without reducing the competitive factor. Machines $f = f_d, ..., r = r_d$ constitute the set $M_{admissible}(d)$. Since J_d was added to machine k, Min_LB$_a$ guarantees $\frac{W_k}{m_k} \le \frac{W_i}{m_i}$ for all $i = f,\ldots,r$. Note that W_k does not include the work of job J_d.

Let J_b be the job having the smallest size among all jobs allocated at machines $f, ..., r$. Hence, jobs packed at $f, ..., r$ cannot be allocated to a machine with a smaller index than $f_b = f_0$. As J_b is executed on one of the machines $f, ..., r$, see Fig. 2, we have $r \ge f$.

Remember that jobs allocated to machines $f, ..., r$ can be scheduled only on machines $f_0, ..., m$. Adding machines $1, ..., f_0 - 1$ and jobs allocated to machines $1, ..., f-1$, $r+1, ..., m$ does not increase $\hat{C}^*_{f_o,m}$ and therefore cannot reduce the competitive factor. We need to discuss the relation between $\hat{C}^*_k$ and $\hat{C}^*_{f,r}$. $\hat{C}^*_{f,r}$ is only smaller than $\hat{C}^*_k$ if we can redistribute workload from machine k to other machines. However, we have $\frac{W_k}{m_k} \le \frac{W_i}{m_i}$ for all $i = f,\ldots,r$. Therefore, we can only redistribute workload for $t \ne 0$. Particularly, $t \ge \frac{W_k}{m_k}$ is required to redistribute the whole workload of machine k. Note that this redistribution is limited by the processing time of jobs submitted after t. This is particularly true if $\hat{C}^*_k > t + \frac{W_k}{m_k}$ holds. Therefore, we have $\hat{C}^*_{f,r} > \frac{1}{2}\hat{C}^*_k$. Alternatively, we can determine the allocation of jobs according to the minimum of $\hat{C}^*_i$ of all admissible machines. This scheme yields $\hat{C}^*_{f,r} \ge \hat{C}^*_k$. However, it is more cumbersome to determine $\hat{C}^*_i$ than W_i.

Finally, remember that the workload of job J_d has not been considered yet.

For $a \le \frac{m_{f,m}}{m_{f_0,m}}$, we obtain $m_{f,m} \le \frac{m_{f,r}}{a}$ and $m_{f_0,m} \le \frac{m_{f,r}}{a^2}$.

This yields $\hat{C}^*_{f_o,m} \ge \hat{C}^*_{f,r} \cdot \frac{m_{f,r}}{m_{f_o,m}} > \hat{C}^*_k \cdot \frac{a^2}{2}$

For $a > \frac{m_{f,m}}{m_{f_0,m}}$, $r_0 \ge f$ implies $m_{f_0,m} \le m_{f,m} + a \cdot m_{f_0,m}$ (see Fig. 2).

This yields $\hat{C}^*_{f_o,m} \ge \hat{C}^*_{f,r} \cdot \frac{m_{f,r}}{m_{f_o,m}} > \hat{C}^*_k \cdot \frac{a(1-a)}{2}$

As scheme *Min_LB*$_a$ uses W_k without including the work of job J_d we must consider job J_d in addition. In the worst case C_k is increased by $p_d \le C^*_{\max}$ resulting in $C_{\max} \le C_k + C^*_{\max}$ and $\rho \le 1 + \frac{C_k}{C^*_{f_0,m}}$. Due to $C_k \le 2 \cdot \hat{C}^*_k$, see Section 4.2, we have $\rho \le 1 + 2 \cdot \frac{\hat{C}^*_k}{\hat{C}^*_{f_o,m}}$.

The competitive factor turns out to be

$$\rho \le \begin{cases} 1 + \frac{4}{a^2} & \text{for } a \le \frac{m_{f,m}}{m_{f_0,m}} \\ 1 + \frac{4}{a(1-a)} & \text{for } a > \frac{m_{f,m}}{m_{f_0,m}}. \end{cases}$$

■

Remarks: Fig. 3 and 4 show the bounds of the competitive factor of strategy *Min_LB*$_a$*+PS* for the admissible value $a \le \frac{m_{f,m}}{m_{f_0,m}}$ and $a > \frac{m_{f,m}}{m_{f_0,m}}$, respectively. One can see that if $a \le \frac{m_{f,m}}{m_{f_0,m}}$ the worst case bounds change from ∞ to 5 as a function of the admissible value $0 < a \le 1$. If $a > \frac{m_{f,m}}{m_{f_0,m}}$ the worst case bounds change from ∞ to ∞ with minimum value for $a = 0.5$.

Fig. 5 shows the resulting bound that is a maximum of worst case bounds presented in Fig. 3, and Fig. 4, as a function of the admissible value $0 < a \le 1$. Note that the bound produces $\rho \le 17$ for $a = 0.5$.

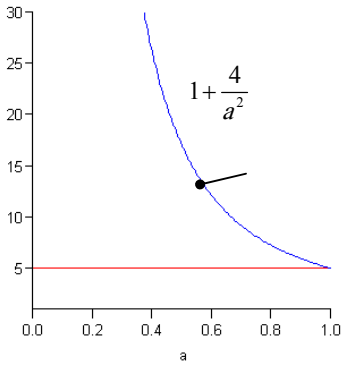

Figure 3. The competitive factor of strategy *Min_LB*$_a$*+PS* for $a \le \frac{m_{f,m}}{m_{f_0,m}}$.

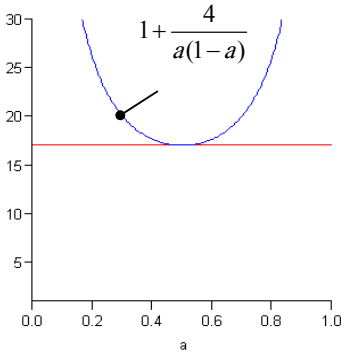

Figure 4. The competitive factor of strategy *Min_LB*$_a$*+PS* for $a > \frac{m_{f,m}}{m_{f_0,m}}$.

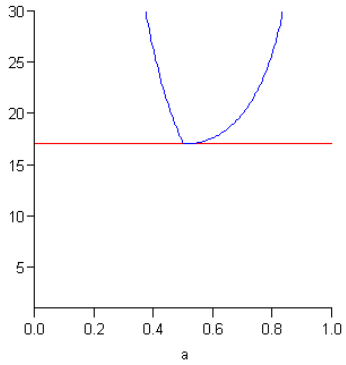

Figure 5. The competitive factor of strategy Min_LB_a+PS

5.1.3. Worst case performance tune up

Finally, we analyze the worst case performance for various workload types. We consider two intervals for the admissible factor a: $(0, \frac{m_{f,m}}{m_{f_0,m}}]$ and $(\frac{m_{f,m}}{m_{f_0,m}}, 1]$. We distinguish here only few cases of workload characteristics to show workload dependent worst case deviations.

- $f = m$ and $f_0 = 1$, produce $\frac{m_m}{m_{1,m}} \le a \le 1$, and $\rho \le 1 + \frac{4}{a(1-a)}$. Clearly, if $a = 1$ holds, as in traditional allocation strategies, a constant competitive factor cannot be guaranteed and $\rho \to \infty$ (see Fig. 4). The example in Sec. 3.3. shows such a schedule in which highly parallel jobs are starving due to jobs with little parallelism. However, a constant competitive factor $\rho \le 17$ can be achieved with $a = 0.5$.
- If $f = f_0 = 1$ holds, we say that the workload is *predominantly sequential*. In such a case, we have $\rho \le 1 + \frac{4}{a^2}$. For $a = 1$, we obtain $\rho \le 5$. However, if $a \to 0$ (jobs are allocated to their first available machines) $\rho \to \infty$ (see Fig. 3).
- If $f = f_0 = m$ holds we say that the workload is *predominantly parallel*. In such a case, we have $\rho \le 1 + \frac{4}{a^2}$. Again $a = 1$ yields $\rho \le 5$.

In a real Grid scenario, the admissible factor can be dynamically adjusted in response to the changes in the configuration and/or the workload. To this end, the past workload within a given time interval can be analyzed to determine an appropriate admissible factor *a*. The time interval for this adaptation should be set according to the dynamics in the workload characteristics and in the Grid configuration. One can iteratively approximate the optimal admissible factor.

6. Concluding remarks

Scheduling in Grids is vital to achieve efficiently operating Grids. While scheduling in general is well understood and has been subject of research for many years, there are still only few theoretical results available. In this paper, we analyze the Grid scheduling problem and present a new algorithm that is based on an adaptive allocation policy. Our Grid scheduling model uses a two layer hierarchical structure and covers the main properties of Grids, for instance, machines with different sizes, and parallel jobs. The theoretical worst-case analysis yields decent bounds of the competitive ratio for certain workload configurations. Therefore, the proposed algorithm may serve as a starting point for future heuristic Grid scheduling algorithms that can be implemented in

real computational Grids. While the scope of this work is the theoretical analysis of Grid scheduling, in future work we also intend to evaluate the practical performance of the proposed strategies and their derivatives. To this end, we plan simulations using real workload traces and corresponding Grid configurations. Further, we will compare our approach with other existing Grid scheduling strategies which are typically based on heuristics.

Acknowledgements. This work is partly supported by CONACYT (Consejo Nacional de Ciencia y Tecnología de México) under grant no.48385, DAAD (Deutscher Akademischer Austauschdienst) Section: 414, A/09/03178.

References

Albers, S. (1999). Better bounds for online scheduling. *SIAM Journal on Computing*, 29(2), 459-473.

Avellino, G., Barale, S., Beco, S., Cantalupo, B., Colling, D., Giacomini, F., Gianelle, A., Guarise, A., Krenek, A., Kouril, D., Maraschini, A., Matyska, L., Mezzadri, M., Monforte, S., Mulac, M., Pacini, F., Pappalardo, M., Peluso, R., Pospisil, J., Prelz, F., Ronchieri, E., Ruda, M., Salconi, L., Salvet, Z., Sgaravatto, M.,Sitera, J., Terracina, A., Vocu, M., & Werbrouck, A. (2003). The EU DataGrid workload management system: towards the second major release. In CHEP 2003, La Jolla, CA, March 2003.

Bar-Noy, A., Freund, A. and Naor, S. (2002) On-line load balancing in a hierarchical server topology. *SIAM Journal on Computing*, 2 (31), 527-549.

Elmroth, E., & Tordsson, J. (2005). An interoperable standards-based grid resource broker and job submission service, e-Science 2005. In First IEEE conference on e-science and grid computing (pp. 212¬220). Los Alamitos: IEEE Computer Society Press. computers: modeling the characteristics of rigid jobs. Journal of Parallel and Distributed Computing, 63(11), 1105- 1122.

Ernemann, C., Yahyapour, R (2003) Applying Economic Scheduling Methods to Grid Environments",in book "Grid Resource Management - State of the Art and Future Trends", Kluwer Academic Publishers, 491-506.

Ernemann, E., Hamscher, V., Schwiegelshohn, U., Streit, A., Yahyapour R. (2002) On Advantages of Grid Computing for Parallel Job Scheduling", Proceedings of 2nd IEEE International Symposium on Cluster Computing and the Grid (CC-GRID 2002), 39-46.

Garey, M. and Graham, R. (1975). Bounds for multiprocessor scheduling with resource constraints. *SIAM Journal on Computing*, 4(2), 187-200.

Graham, R., Lawler, E., Lenstra, J., and Kan, A. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 15, 287-326.

Huedo, E., Montero, R. and Llorente, I. (2007). A modular meta-scheduling architecture for interfacing with pre-WS and WS Grid resource management services. *Future Generation Computing Systems* 23(2), 252-261.

Krauter, K., Buyya, R., and Maheswaran, M. (2002), A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing, Internation Journal of Software: Practice and Experience (SPE), Wiley Press, 32:135–164.

Kurowski, K., Nabrzyski, J., Oleksiak A., Weglarz, J. (2008). A Multi-criteria Approach to Two-level Hierarchy Scheduling in Grids. Journal of Scheduling, 11(5), 371-379.

Naroska, E. and Schwiegelshohn, U. (2002). On an online scheduling problem for parallel jobs. *Information Processing Letters*, 81(6), 297-304.

Pascual, F., Rzadca, K., Trystram, D. (2008). Cooperation in multi-organization scheduling. In *Concurrency and Computation: Practice and Experience*. doi: 10.1002/cpe.1378.

Robertazzi, T. and Yu, D. (2006). Multi-Source Grid Scheduling for Divisible Loads. Proceedings of the 40th Annual Conference on Information Sciences and Systems, pp. 188-191.

Rodero, I., Corbalan, J., Badia, R. M., & Labarta, J. (2005). eNANOS grid resource broker. In P. M. A. Sloot, et al. (Eds.), Advances in Grid Computing - EGC 2005, European Grid Conference, 2005.

Rodero, I., Guim, F., Corbalan, J., Labarta, J., Oleksiak, A., Kurowski, K., Nabrzyski, J. (2008). Integration of the eNANOS Execution Framework with GRMS. Achievements in European Research on Grid Systems, CoreGRID Integration Workshop 2006 (Selected Papers), Springer-Verlag, pp 25-39, 2008.

Rudin III, J. (2001). Improved bounds for the on-line scheduling problem. Ph.D. thesis, The University of Texas at Dallas.

Schwiegelshohn, U. and Yahyapour, R. (2003) Attributes for communication between grid scheduling instances. In J. Nabrzyski, J. Schopf, and J. Weglarz (Eds.), *Grid Resource Management - State of the Art and Future Trends*, Kluwer Academic pp. 41-52.

Schwiegelshohn, U., Tchernykh, A., and Yahyapour, R. (2008) Online Scheduling in Grids, Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS 2008), pp. 1-10.

Schwiegelshohn, U. (2009) An Owner-centric Metric for the Evaluation of Online Job Schedules, Proceedings of the 2009 Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2009), pp. 557-569.

Tchernykh, A., Ramirez, J., Avetisyan, A., Kuzjurin, N., Grushin, D., and Zhuk, S. (2006) Two Level Job-Scheduling Strategies for a Computational Grid. in Wyrzykowski et al. (Eds.). Parallel Processing and Applied Mathematics, LNCS 3911, Springer-Verlag , pp. 774-781.

Tchernykh, A., Schwiegelshohn, U., Yahyapour, R., Kuzjurin, N. (2008) Online Hierarchical Job Scheduling in Grids, in Thierry Priol and Marco Vanneschi (Eds.) *From Grids to Service and Pervasive Computing*, Springer-Verlag, pp. 77-91.

Vazquez-Poletti, J.L. Huedo, E. Montero, R.S. Llorente, I.M. (2007) A comparison between two grid scheduling philosophies: EGEE WMS and Grid Way, Journal Multiagent and Grid Systems, IOS Press, VOL 3; NUMB 4, 429-440.

Zhuk, S., Chernykh, A., Kuzjurin, N., Pospelov, A., Shokurov, A., Avetisyan, A., Gaissaryan, S., Grushin, D. (2004) Comparison of Scheduling Heuristics for Grid Resource Broker. Proceedings of the Third International IEEE Conference on Parallel Computing Systems (PCS2004), 388-392.