ONLINE HIERARCHICAL JOB SCHEDULING ON GRIDS

Andrei Tchernykh
Computer Science Department, CICESE Research Center, Ensenada, BC, México
chernykh@cicese.mx

Uwe Schwiegelshohn
Robotics Research Institute, Technische Universität Dortmund, Dortmund, Germany
uwe.schwiegelshohn@udo.edu

Ramin Yahyapour
IT and Media Center, Technische Universität Dortmund, Dortmund, Germany
ramin.yahyapour@udo.edu

Nikolai Kuzjurin
Institute of System Programming RAS, Moscow, Russia
nnkuz@ispras.ru

Abstract In this paper, we address non preemptive online scheduling of parallel jobs on a Grid. The 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 applied to each machine independently. We discuss strategies based on various combinations of allocation strategies and local scheduling algorithms. Finally, we propose and analyze a relatively simple 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.

Keywords: Grid Computing, Online Scheduling, Resource Management, Algorithmic Analysis, Job Allocation

1. Introduction

Due to the size and dynamic nature of Grids, allocating computational jobs to available Grid resources requires an automatic and efficient process. Various scheduling systems have been proposed and implemented in different types of Grids. However, there are still many open issues in this field, including the consideration of multiple layers of scheduling, dynamicity, and scalability. Grids are typically composed of heterogeneous resources which are decentralized and geographically dispersed. Academic studies often propose a completely distributed resource management system, see, for instance, Uppuluri et al. [13] while real installations favor a combination of decentralized and centralized structures, see, for instance, GridWay [4]. A hierarchical multilayer resource management can represent such a system well. Therefore, we use this model to find a suitable tradeoff between a fully centralized and a fully decentralized model. The highest layer is often a Grid-level scheduler that may have a more general view of the resources while the lowest layer is the local resource management system that manages a specific resource or set of resources, see Schwiegelshohn and Yahyapour [10]. Other layers may exist in between. At every layer, additional constraints and specifications must be considered, for instance, related to the dynamics of the resource situation. Thus, suitable scheduling algorithms are needed to support such multilayer structures of resource management.

Grids are typically based on existing scheduling methods for multiprocessors and use an additional Grid scheduling layer [10]. 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 scheduling in Grids is almost exclusively addressed by practitioners looking for suitable implementations. There are only very few theoretical results on Grid scheduling and most of them address divisible load scheduling like, for instance, Robertazzi and Yu [7].

In this paper, we propose new Grid scheduling approaches and use a theoretical analysis to evaluate them. As computational Grids are often considered as successors and extensions of multiprocessors or clusters we start with a simple model of parallel computing and extend it to Grids. One of the most basic models due to Garey and Graham [2] assumes a multiprocessor with identical processors as well as independent, rigid, parallel jobs with unknown processing times, where a suitable set of concurrently available processors exclusively executes this job. Although this model neither matches every real installation nor all real applications the assumptions are nonetheless reasonable. The model is still a valid basic abstraction of a parallel computer and many applications.

Our Grid model extends this approach by assuming that the processors are arranged into several machines and that parallel jobs cannot run across multiple machines. The latter assumption is typically true for jobs with extensive communication among the various processors unless special care has been devoted to optimize the code for a multisite configuration.

While a real Grid often consists of heterogeneous parallel machines, one can argue that an identical processor model is still reasonable as most modern processors in capacity computing mainly differ in the number of cores rather than in processor speed. Our model considers two main properties of a Grid: separate parallel machines and different machine sizes. Therefore, the focus of this paper is on these properties of Grids.

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 is commonly represented by the objective of makespan minimization. Although the makespan objective is mainly an offline criterion and has some shortcomings particularly in online scenarios with independent jobs, it is easy to handle and therefore frequently used even in these scenarios, see, for instance, Albers [1]. Hence, we also apply this objective in this paper. For such a model, Schwiegelshohn et al. [9] showed that the performance of Garey and Graham's list scheduling algorithm is significantly worse in Grids than in multiprocessors. They 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. The offline non-clairvoyant version of this algorithm has an approximation factor of 3. This "one-layer" algorithm can be implemented in centralized fashion or use a distributed "job stealing" approach. Although jobs are allocated to a machine at their submission times they can migrate if another machine becomes idle.

In this paper, we use a two layer hierarchical online Grid scheduling model. Once a job is allocated to a machine it must be scheduled and executed on this machine, that is, migration between machines is not allowed. Tchernykh et al. [11] considered a similar model for the offline case and addressed the performance of various 2-stage algorithms with respect to the makespan objective. They present algorithms with an approximation factor of 10.

In Section 2, we present our Grid scheduling model in more details. The algorithms are classified and analyzed in Section 3. We propose a novel adaptive two-level admissible scheduling strategy and analyze it in Section 4. Finally, we conclude with a summary and an outlook in Section 5.

2. Model

As already discussed, we address an online scheduling problem with the objective of minimizing the makespan: n parallel jobs $J_1, J_2, \ldots$ must be scheduled on m parallel machines $N_1, N_2, \ldots, N_m$. m_i denotes the number of identical processors of machine N_i. W.l.o.g. we index the parallel machines in ascending order of their sizes $m_1 \leq m_2 \leq ... \leq m_m$, and introduce $m_0 = 0$.

Each job J_j is described by a triple (r_j,$size_j$,p_j): its release date $r_j \geq 0$, its size $1 \leq size_j \leq m_m$ that is referred to as its *degree of parallelism*, and its execution time p_j. The release date is not available before a job is submitted, and its processing time is unknown until the job has completed its execution (non-clairvoyant scheduling).

We assume that job J_j can only run on machine N_i if $size_j \leq m_i$ holds, that is, we do not allow multi-site execution and co-allocation of processors from different machines. Finally, $g(J_j) = N_i$ denotes that job J_j is allocated to machine N_i. Let n_i be the number of jobs allocated to the machine N_i.

We assume a space sharing scheduling mode as this is typically applied on many parallel computers. Therefore, a parallel job J_j is executed on exactly $size_j$ disjoint processors without preemptions.

Let $p_{\max}$ be $\max_{1 \leq j \leq n}\{p_j\}$. Further, $W_j = p_j \cdot size_j$ is the work of job J_j, also called its area or its resource consumption. Similarly, the total work of a job set I is $W_I = \sum_{J_j \in I} W_j$. $c_j(S)$ denotes the completion time of job J_j in schedule S. We omit schedule S if we can do so without causing ambiguity.

All strategies are analyzed according to their competitive factor for makespan optimization. Let $C^*_{\max}$ and $C_{\max}(A)$ denote the makespan of an optimal schedule and of a schedule determined by strategy A, respectively. The competitive factor of strategy A is defined as $\rho_A = \sup \frac{C_{\max}(A)}{C^*_{\max}}$ for all problem instances.

The notation GP_m describes our Grid machine model. In the short three field notation *machine model-constraints-objective* proposed by Graham et al. [3], this problem is characterized as $GP_m | r_j, size_j | C_{\max}$. We use the notation MPS to refer to this problem while the notation PS describes the parallel job scheduling on a single parallel machine ($P_m | r_j, size_j | C_{\max}$).

3. Classification of Algorithms

In this section, we first split the scheduling algorithm into an allocation part and a local scheduling part. Then we introduce different strategies to allocate jobs to machines. We classify these strategies depending on the type and amount of information they require. Finally, we analyze the performance of these algorithms.

3.1 Two Layer MPS Lower Bound

Before going into details we add some general remarks about the approximation bounds of MPS. We regard MPS as 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. 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 best 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 best PS algorithm are identical as there is no need for any allocation stage.

But clearly, an unsuitable allocation strategy may produce bad competitive factors. Just assume that all jobs are allocated to a single machine in a Grid with k identical machines. Obviously, the competitive factor is lower bounded by k.

The best possible PS online non-clairvoyant algorithm has a tight competitive factor $2 - 1/m$ with m denoting the number of processors in the parallel system; see Naroska and Schwiegelshohn [6]. Hence, the lower bound of a competitive factor for any general two-layer online MPS is at least $2 - 1/m$.

Schwiegelshohn et al. [9] showed that there is no polynomial time algorithm that guarantees schedules with a competitive bound < 2 for $GP_m|size_j|C_{\max}$ and all problem instances unless $P = NP$. Therefore, the multiprocessor list scheduling bound of $2 - 1/m$, see Garey and Graham [2] for concurrent submission as well as Naroska and Schwiegelshohn [6] 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 [9, 11].

3.2 Job Allocation

Now, we focus on job allocation with the number of parallel machines and the information about the size of each machine being known. Further, we distinguish four different levels of additionally available information for job allocation.

Level 1: The job load of each machine, that is the number n_i of jobs waiting to run on machine N_i, is available. We use the job allocation strategies Min_L and Min_LP. Min_L allocates a job to the machine with the smallest job load. This strategy is similar to static load balancing. Min_LP takes into account the number of processors and selects the resource with the lowest job load per processor ($\arg\{\min_{1\leq i\leq m}\{\frac{n_i}{m_i}\}\}$). Note that neither strategy considers the degree of parallelism or the processing time of a job.

Level 2: Additionally to the information of Level 1, the degree of parallelism $size_j$ of each job is known. The Min_PL strategy selects a machine with the smallest parallel load per processor ($\arg\{\min_{1\leq i\leq m}\{\sum_{g(J_j)=N_i}\frac{size_j}{m_i}\}\}$).

Level 3: In addition to the information of Level 2, we consider clairvoyant scheduling, that is, the execution time of each job is available. The Min_LB strategy allocates a job to the machine N_i with the least remaining total workload of all jobs already allocated to this machine, that is

$$\arg\{\min_{1\le i\le m}\{\sum_{g(J_j)=N_i}\frac{size_j\cdot p_j}{m_i}\}\}.$$

If the actual processing time p_j of job J_j is not available, we may use an estimate of the processing time instead. Such an estimate may be generated automatically based on history files or provided by the user when the job is submitted.

Level 4: We have access to all information of Level 3 and to all local schedules as well. The Min_CT job strategy allocates job J_j to the machine with the earliest completion time of this job using the existing schedules and the job queues [8, 14].

Levels 1 and 2 describe non-clairvoyant problems. Strategies of Levels 1 to 3 are based only on the parameters of jobs, and do not need any information about local schedules.

3.3 Two Layer MPS Strategies

In this section, we analyze a two layer online MPS for different PS algorithms and the MPS-allocation strategies Min_L, Min_LP, Min_PL, Min_LB, and Min_CT.

Tchernykh et al. [11] discussed the combination of allocation strategies with the simple online scheduling algorithm $FCFS$, that schedules jobs in the order of their arrival times. Clearly, this strategy cannot guarantee a constant approximation factor as $FCFS$ is already not able to do this. Even if all jobs are available at time 0 and they are sorted in descending order of the degree of parallelism, we cannot achieve a constant competitive factor [11, 9].

Let us consider now the case of an arbitrary online local PS algorithm. Based on the simple example considered by Tchernykh et al. [11] for offline strategies and Schwiegelshohn et al. [9] for online strategies, it can be shown that Min_L, Min_LP, and Min_PL allocation strategies combined with PS cannot guarantee a constant approximation factor of MPS in the worst case.

Let us now consider the two allocation strategies Min_LB and Min_CT that take into account job execution times. Fig. 1 shows an example of sets of machines and a set of jobs for which constant approximation factors are not guaranteed for $Min\_LB + PS$ and $Min\_CT + PS$. In this figure, the vertical axis represents time while the bars and their widths denote machines and their numbers of processors, respectively. The example has an optimal makespan of 2, see Fig. 1. If the jobs are released in ascending order of their degrees of parallelism, algorithms Min_LB+PS and Min_CT+PS allocate them to machines as shown in Fig. 2. If the processing time is identical for all jobs the makespans of algorithms $Min\_LB + PS$ and $Min\_CT + PS$ equal the number of job groups with different degrees of parallelism. Additional information about the schedule in each machine (application of Min_CT) does not help to improve the worst case behavior of MPS. Results are the similar for the offline [11] and the online [9] cases.

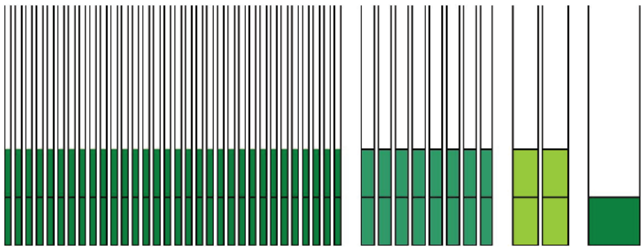

Figure 1: Optimal schedule of a bad instance.

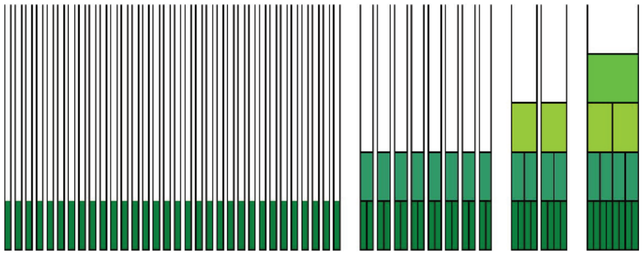

Figure 2: $Min\_LB + PS$ schedule for the instance of Fig.1.

4. Adaptive Admissible Allocation Strategy

Based on the example shown in Figure 2, it can be seen that one reason of the inefficient online job allocation is the occupation of large machines by sequential jobs causing highly parallel jobs to wait for their execution.

Tchernykh et al. [11] proposed a relatively simple scheme named *admissible selection* that can be efficiently implemented in real systems. This scheme excludes certain machines with many processors from the set of machines available to execute jobs with little parallelism.

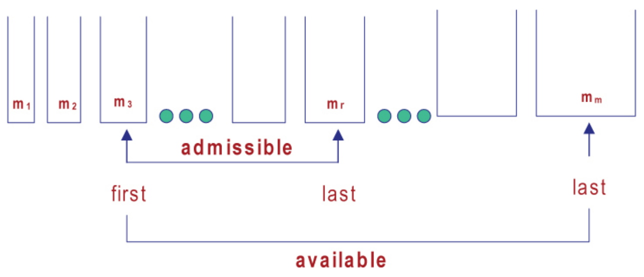

Figure 3: Concept of the admissible model

Let the machines be indexed in non-descending order of their sizes ($m_1 \leq m_2 \leq ... \leq m_m$). We define $f(j) = \text{first}(j)$ to be the smallest index i such that $m_i \geq size_j$ holds for job J_j. Note that due to our restriction $size_j \leq m_m \forall J_j$, we have $l \leq m$. The set of available machines $M_{available}(j)$ that are available for allocation of job J_j corresponds to the set of machine indexes $s(f(j), m) = \{f(j), f(j)+1, \ldots, m\}$, see Fig. 3. Obviously, the total set of machines $M - total$ is represented by the integer set $s(1, m) = 1, \ldots, m$. $m(f, l) = \sum_{i=f}^{l} m_i$ is the total number of processors that are in machines m_f to m_l.

Tchernykh et al. [11] defined the set $M_{admissible}(j)$ of admissible machines for a job J_j to be the machines with their indexes being in the set $s(f(j), r(j))$, see Fig. 3. $r(j)$ is the smallest number with $m(f(j), r(j)) \geq \frac{1}{2} m(f(j), m)$. In this paper, the definition is generalized by introducing a new parameter $0 \leq a \leq 1$ that parameterizes the admissibility ratio used for the job allocation. Hence, we call $s(f(j), r(j))$ the index set of admissible machines if $r(j)$ is the minimum index such that $m(f)j), r(j)) \geq a \cdot m(f(j), m)$ holds. The choice $a = 0.5$ produces the original definition of Tchernykh et al. [11].

A worst case analysis of adaptive admissible selection strategies is presented in Section 4.2.

4.1 Workload

Before going into details of admissible job allocation strategies, we define different types of possible workloads. First, we combine all machines of the same size into group. Let i be a machine index such that $m_{i-1} < m_i$. Then group G_i contains all machines M_j with $m_j = m_i$. The size of G_i is the total number of processors of all machines in G_i. Further, we can partition the set of all jobs into sets Y_i such that $J_j \in Y_i$ if and only if $m_{i-1} < size_j \leq m_i$ holds, that is, all jobs of Y_i can be executed by machines of G_i but do not fit on any machine of a group G_h with $h < i$. Note that some sets Y_i may be empty.

The workload is *balanced* for a set of machines $G = \bigcup_{i=1}^{k} G_i$ with some $k > 0$ if the ratio of the total work of set Y_i and the size of group G_i is the same for all groups of G.

The workload is *perfect* for G if it is balanced and each set Y_i can be scheduled in a nondelay fashion on G_i such that makespans of all machines in G are identical. Fig. 1 shows an example of a balanced workload for each set of machines that do not include the last machine.

4.2 Analysis

In this section, we consider the allocation strategies of Section 3.3 for admissible machines. Formally, we denote this extension by appending the letter a to the strategy name, for instance, $Min\_L - a$. Tchernykh et al. [11] showed that strategies $(Min\_L - a, Min\_PL - a) + FCFS$ cannot guarantee constant approximations for $a = 0.5$. This result also holds for arbitrary a and algorithm $Best\_PS$. We already showed in Section 3.3 that Min_LB cannot guarantee a constant approximation even in combination with the $Best\_PS$. We now consider the case when the selection of a suitable machine for executing job J_j is limited by its admissible set of machines $M_{admissible}(j)$.

4.2.1 Online Allocation and Online Local Scheduling. First, we determine the competitive factor of algorithm $Min\_LB - a + Best\_PS$.

THEOREM 1 *Assume a set of machines with identical processors, a set of rigid jobs, and admissible allocation range* $0 \le a \le 1$. *Then algorithm* $Min\_LB - a + Best\_PS$ *has the competitive factor*

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

with $1 \le f_0 \le f \le m$ *being parameters that depend on the workload, see Fig. 4.*

Proof. Let us assume that the makespan of machine N_k is also the makespan $C_{\max}$ of the Grid. Then let job J_d be the last job that was added to this machine. We use the notations $f = f(d)$ and $r = r(d)$. $I_f, \ldots, I_r$ are the sets of jobs that had already been scheduled on machines $N_f, \ldots, N_r$ before adding job J_d. Remember that machines $N_f, \ldots, N_r$ constitute the set $M_{admissible}(d)$. Since J_d was added to machine N_k, $Min\_LB - a$ guarantees $\frac{W_{I_k}}{m_k} \le \frac{W_{I_i}}{m_i}$ for all

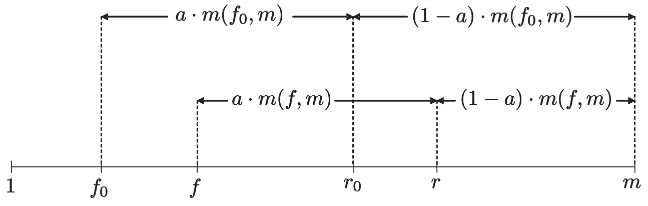

Figure 4: Admissible allocation with factor a

$i = f, \ldots, r$. Therefore, we have

$$\begin{aligned} W(f,r) &= \sum_{i=f}^{r} W_{I_i} = \sum_{i=f}^{r} \frac{W_{I_i}}{m_i} m_i \\ &\geq \sum_{i=f}^{r} \frac{W_{I_k}}{m_k} m_i = \frac{W_{I_k}}{m_k} \sum_{i=f}^{r} m_i = \frac{W_{I_k}}{m_k} m(f,r). \end{aligned}$$

Let W_{idle}^{opt} be the idle workload space of the optimal solution on the machine N_k. We use the notation $W_i^{'} = W_i + W_{idle}^{opt}$ and obtain

$$\begin{aligned} W^{'}(f,r) &= \sum_{i=f}^{r} (W_i + W_{idle}^{opt}) = \sum_{i=f}^{r} W_i^{'} = \sum_{i=f}^{r} \frac{W_i^{'}}{m_i} m_i \\ &\geq \sum_{i=f}^{r} \frac{W_k^{'}}{m_k} m_i == \frac{W_k^{'}}{m_k} \sum_{i=f}^{r} m_i = \frac{W_k^{'}}{m_k} m(f,r). \qquad (1) \end{aligned}$$

It is known from the literature [5, 12] that in the schedule of machine N_k, there are two kinds of time slots which can be combined conceptually into two successive intervals C_1 and C_2, see Fig. 5.

Let $size_{\max}$ be the maximum size of any job assigned to machine N_k. Then the intervals correspond to the parts of the schedule when at most $size_{\max} - 1$ processors are idle and when strictly more than $size_{\max} - 1$ processors are idle, respectively.

Tchernykh et al. [12] showed that C_2 is limited by the maximum job execution time $p_{\max}$ and that for an arbitrary list schedule, $W_k \geq (m_k - size_{\max} + 1)C_1 + C_2$ yields the competitive bound $\frac{2m_k - size_{\max}}{m_k - size_{\max} + 1}$.

Algorithm $Best\_PS$ produces the makespan $C = \frac{W_k + W_{idle}^{opt} + W_{idle}}{m_k}$ with W_{idle} being the additional idle space due to $Best\_PS$. To be $2 - \frac{1}{m}$ competitive, algorithm $Best\_PS$ must generate schedules that increase the idle space of the

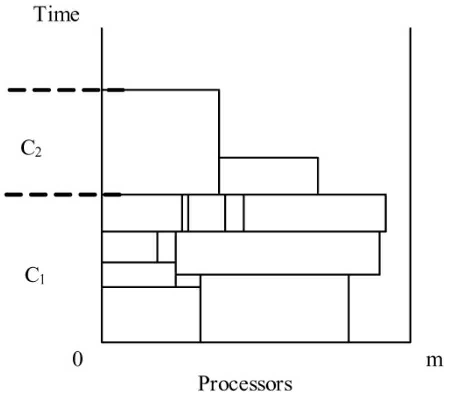

Figure 5: Scheduling Rigid Jobs in Space Sharing Mode [12]

optimal schedule by not more than
$W_{idle} \leq p_{\max} \cdot (m_k - 1)$.

Hence, for $W_k^{'} = W_k + W_{idle}^{opt}$ and $W_k^{'} \geq m_k \cdot C_1 + C_2$, we obtain an upper bound of the total completion time:

$$C \leq \frac{W_k^{'} + p_{\max} \cdot (m_k - 1)}{m_k} = \frac{W_k^{'}}{m_k} + p_{\max} \cdot (1 + \frac{1}{m_k}) \qquad (2)$$

Due to $C^*_{\max} = \frac{W_k^{'}}{m_k}$ and $C^*_{\max} \geq p_{\max}$, Equation 2 implies a competitive bound $2 - \frac{1}{m}$ for single machine scheduling.

Let J_b be the job having the smallest size among all jobs executed at machines $N_f, \ldots, N_r$. We use the notation $f_0 = f_b$. Hence jobs packed at $N_f, \ldots, N_r$ cannot be allocated to a machine with a smaller index than f_0. As J_b is executed on one of the machines $N_f, \ldots, N_r$ we have $r_b \geq f$, see Fig. 4 and $C^*_{\max} \geq \frac{W^{'}(f,r)}{m(f_0,r)}$. Substituting Equation 1 in this formula, we have

$$C^*_{\max} \quad \geq \quad \frac{W_k^{'} m(f,r)}{m_k m(f_0,m)}.$$

Finally, we consider two cases:

- $a \leq \frac{m(f,m)}{m(f_0,m)}$

  From our definition $m(f,r) \geq a \cdot m(f,m)$, we obtain $m(f_0,m) \leq m(f,r)/a^2$. This yields

$$C^*_{\max} \geq \frac{W_k^{'} \cdot m(f,r)}{m_k \cdot m(f_0,m)} \quad \geq \quad \frac{W_k^{'} \cdot a^2}{m_k}$$

As we have $C^*_{\max} \geq p_{\max}$, Equation 2 implies

$$\rho \leq \frac{W'_k}{m_k C^*_{\max}} + p_{\max} \frac{1 - \frac{1}{m_k}}{C^*_{\max}} \leq 1 + \frac{1}{a^2} - \frac{1}{m(1,m)}$$

- $a > \frac{m(f,m)}{m(f_0,m)}$

  We have $m(f_0, m) \leq m(f,m) + a \cdot m(f_0, m)$, see Fig. 4. This yields

$$C^*_{\max} \geq \frac{W'_k \cdot m(f,r)}{m_k \cdot m(f_0,m)} \geq \frac{W'_k \cdot a \cdot m(f,m)}{m_k \cdot \frac{m(f,m)}{1-a}} = \frac{W'_k \cdot a \cdot (1-a)}{m_k}$$

  and

$$\rho \leq \frac{W'_k}{m_k C^*_{\max}} + p_{\max} \frac{1 - \frac{1}{m_k}}{C^*_{\max}} \leq 1 + \frac{1}{a \cdot (1-a)} - \frac{1}{m(1,m)}$$

(End)

Note that both bounds produce the same result $\rho = 5 - \frac{1}{m(1,m)}$ for $a = 0.5$.

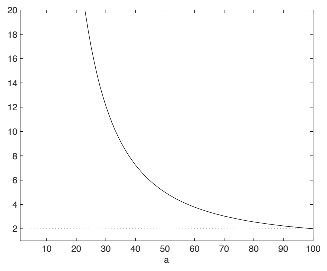

Figure 6: $\rho \leq 1 + \frac{1}{a^2} - \frac{1}{m(1,m)}$

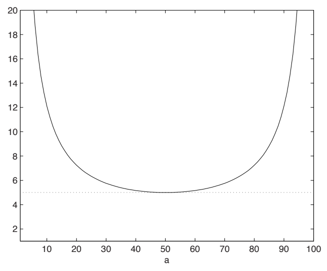

Figure 7: $\rho \leq 1 + \frac{1}{a(1-a)} - \frac{1}{m(1,m)}$

Fig. 6 to 8 show the bounds of the competitive factor of strategy $Min\_LB - a + Best\_PS$ as a function of the admissible value a in percent.

4.2.2 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 only few cases of workload characteristics to determine workload dependent worst case deviations.

- $f = m$ and $f_0 = 1$ produce $\frac{m_m}{\sum_{i=1}^{m} m_i} \leq a \leq 1$ and $\rho \leq 1 + \frac{1}{a(1-a)} - \frac{1}{m(1,m)}$. These characteristics are normal for a balanced workload. Clearly,

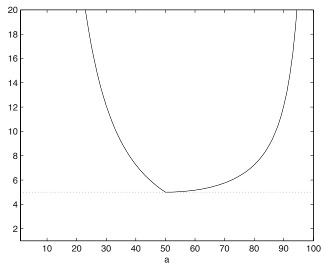

Figure 8: $\rho \leq 1 + \frac{1}{a^2} - \frac{1}{m(1,m)}$ for $a \leq 0.5$ and $\rho \leq 1 + \frac{1}{a(1-a)} - \frac{1}{m(1,m)}$ for $a > 0.5$

if $a = 1$ holds, as in traditional allocation strategies, a constant approximation cannot be guaranteed. The example in Fig. 2 shows such a schedule in which highly parallel jobs are starving due to jobs with little parallelism. However, a constant approximation $\rho = 5 - \frac{1}{m(1,m)}$ 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 $\rho \leq 1 + \frac{1}{a^2} - \frac{1}{m(1,m)}$. For $a = 1$, we obtain $\rho = 2 - \frac{1}{m(1,m)}$. This bound is equal to the bound of list scheduling on a single machine with the same number of processors. Hence, for this type of workload, Min_LB is the best possible allocation algorithm.

- If $f = f_0 = m$ holds we say that the workload is *predominantly parallel*. In such a case, we have $\rho \leq 1 + \frac{1}{a^2} - \frac{1}{m(1,m)}$. Again $a = 1$ yields $\rho = 2 - \frac{1}{m(1,m)}$. Therefore, Min_LB is also the best possible allocation algorithm for this type of workload.

- 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 optimal 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.

5. 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, different machine 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. In future work, we 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.

References

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

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

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

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

[5] E. Lloyd. Concurrent task systems, Operational Research 29(1):189-201, 1981.

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

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

[8] G. Sabin, R. Kettimuthu, A. Rajan, and P. Sadayappan. Scheduling of Parallel Jobs in a Heterogeneous Multi-Site Environment, Proceedings of the 8th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), pages 87-104, 2003.

[9] U. Schwiegelshohn, A. Tchernykh, and R. Yahyapour. Online Scheduling in Grids, Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS'2008), CD-ROM, 2008.

[10] U. Schwiegelshohn and R. Yahyapour. 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, pages 41-52, 2003.

[11] A. Tchernykh, J. Ramírez, A. Avetisyan, N. Kuzjurin, D. Grushin, and S. Zhuk. Two Level Job-Scheduling Strategies for a Computational Grid. In Parallel Processing and Applied Mathematics, Wyrzykowski et al. (Eds.): Proceedings of the Second Grid Resource Management Workshop (GRMW'2005) in conjunction with the Sixth International Conference

on Parallel Processing and Applied Mathematics - PPAM 2005. LNCS 3911, Springer-Verlag, pages 774-781, 2006.

[12] A. Tchernykh, D. Trystram, C. Brizuela, and I. Scherson. Idle Regulation in Non-Clairvoyant Scheduling of Parallel Jobs to be published in Discrete Applied Mathematics, 2008.

[13] P. Uppuluri, N. Jabisetti, U. Joshi, and Y. Lee. P2P Grid: Service Oriented Framework for Distributed Resource Management. Proceedings of the 2005 IEEE International Conference on Services Computing (SCC'05), pages 347-350, 2005.

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