Simple Competitive Request Scheduling Strategies*

Petra Berenbrink, Marco Riedel, and Christian Scheideler
Department of Mathematics and Computer Science
Paderborn University, 33095 Paderborn, Germany
email: {pebe, barcom, chrsch}@uni-paderborn.de

Abstract

In this paper we study the problem of scheduling real-time requests in distributed data servers. We assume the time to be divided into time steps of equal length called rounds. During every round a set of requests arrives at the system, and every resource is able to fulfill one request per round. Every request specifies two (distinct) resources and requires to get access to one of them. Furthermore, every request has a deadline of d, i.e. a request that arrives in round t has to be fulfilled during round $t + d - 1$ at the latest. The number of requests which arrive during some round and the two alternative resources of every request are selected by an adversary. The goal is to maximize the number of requests that are fulfilled before their deadlines expire.

We examine the scheduling problem in an *online setting*, i.e. new requests continuously arrive at the system, and we have to determine online an assignment of the requests to the resources in such a way that every resource has to fulfill at most one request per round. We study both global (i.e. centralized) and local (i.e. distributed) scheduling strategies. In order to measure the performance of our scheduling strategies we apply competitive analyses. Previously, no non-trivial bounds have been known for the competitive ratio of scheduling strategies in our model. We present (partly matching) upper and lower bounds for several simple scheduling strategies.

1 Introduction

Recently, the progress in storage medium and communication technologies made it possible to store an enormous amount of data in distributed data servers. The stored information can be made accessible to many clients simultaneously in applications like video-on-demand, tele-teaching or online transaction processing. In order to have these applications running information often has to be sent under hard real-time assumptions.

In this paper we study the problem of scheduling real-time requests in distributed data servers. Such a server consists of a set of disks (called *resources* in the following) and a set of clients which are connected to the disks via a network. If a client wants to access a data item stored on one of the disks, it will issue a *request*. It is well known (see e.g. [MSF98, Kor97]) that an effective strategy to avoid hot spots at the disks is to have two copies of every data item which are stored on different disks. As the data server now has two possible options for choosing the disk that has to fulfill the request, it can perform a kind of load balancing among the disks.

In our model we assume a data server to work in synchronized rounds, and every resource can fulfill one request per round. Every request has a deadline that limits the rounds which can be used in order to fulfill the request. If a request cannot be processed before its deadline, it is canceled. Thus, the goal of a data server is to maximize the number of requests which are fulfilled before their deadlines expire. The number of requests arriving during some round and the two alternative resources of every of these requests are selected by an adversary. The advantage of choosing an adversarial model is that results within this model do not depend on particular probabilistic assumptions. This is important since there might sometimes be high correlations among the requested data items for applications such as video-on-demand. This might cause a set of requests to appear much more frequently within a short time interval than would be the case under idealized probabilistic assumptions.

We examine the scheduling problem in an *online setting*, i.e. new requests are continuously injected into the system and we have to determine online an assignment of the requests to the resources in such a way that every resource has to fulfill at most one request per round. This means that decisions have to be taken without any knowledge about future requests. We examine both *local* and *global* assignment strategies. In the global case the online algorithm is allowed to find the assignment using the whole set of information about the requests which are currently in the system. This can be regarded as having one central instance taking the assignment decisions. In the *local* case the decisions have to be taken locally at the clients and/or resources. Therefore, the resources/clients may have to gather information concerning the request situation at other resources. Gathering information from other clients or resources is expensive, since this requires communication, and therefore the gathering should be minimized.

To measure the performance of our scheduling strategies we make use of *competitive analyses*. The main idea of a competitive analysis is to compare the quality of the solution computed by the online algorithm with the solution of an optimal offline algorithm which knows the whole sequence of requests in advance.

* Authors supported in part by the DFG-Sonderforschungsbereich 376 "Massive Parallelität: Algorithmen, Entwurfsmethoden, Anwendungen", by the EU ESPRIT Long Term Research Project 20244 (ALCOM-IT), and by the DFG-Graduiertenkolleg "Parallele Rechnernetzwerke in der Produktionstechnik", GRK 124/2-96

1.1 Known results

The previous work that is related to our model can be divided into three topics: At first we very briefly state some results from the area of balls-into-bins games and graph matching. Then we give brief short survey of online scheduling results, especially about bipartite matching problems in online settings.

In the area of balls-into-bins games and related allocation processes it is well-known that the choice of one of several alternative resources can be very profitable for the distribution of the requests. In the case of balls-into-bins games the problem is to distribute m balls across n bins, and thereby minimize the maximum load of the bins. It is well-known that if each ball randomly chooses one bin and $m = n$, it will be likely that one of the bins receives $\Omega(\log n/ \log\log n)$ balls. Karp, Luby, and Meyer auf der Heide [KLM92] have been the first to showed that using several copies for every data item instead of one leads to an exponential improvement of the maximum load. They used the approach of several copies to construct fast distributed algorithms for the simulation of parallel random access machines (PRAMs) on distributed memory machines (DMMs). Azar et al. [ABKU94] study the problem of assigning requests to resources in a sequential setting, i.e. the requests arrive one by one. They show that, if every data item has c copies and $m = n$, there is a simple strategy that ensures a maximum load of $\Theta(\log\log n/ \log c)$ with high probability. They also present results for the case of m significantly larger than n. Much work has been done to analyze balls-into-bins games in its different flavors. For sequential balls-into-bins games see [Mit96, CS97, Mit97, Czu98, CFM+98]. For parallel balls-into-bins games see [ACMR95, Ste96, BMS97, ABS98].

Graph matching is a fundamental topic in graph theory. Let $G = (V, E)$ be a graph with nodes V and undirected edges E without multi-edges or self-loops. A matching of G is a subset $M \subset E$, such that the edges of M are not adjacent. Many algorithms which calculate matchings have been developed in the past. Please consult e.g. [LP86] for the history of matching algorithms. Simple methods with time complexity $O(|E|)$ can be used to calculate maximal matchings. The so far fastest algorithm known so far for finding a maximum cardinality matching is by Micali and Vazirani [MV80] which has a time complexity of $O(|E|\sqrt{|V|})$.

In the area of online scheduling there is a vast amount of literature. [Sga98] gives an encyclopedic survey. The study of bipartite matching problems in online settings has been initiated by Karp, Vazirani and Vazirani in [KVV90]. Their model distinguishes between a 'known' and an 'unknown' partition. Nodes of the unknown partition are revealed as the time passes and an online algorithm can put at most one edge of a just revealed node in the online matching. The simple GREEDY algorithm is an optimal deterministic online algorithm for this problem. It achieves a competitive ratio of 2. The key contribution of [KVV90] is the construction and analysis of an optimal randomized algorithm, called RANKING, which is $e/(e-1) \approx 1.58$-competitive. Various extensions and variants of this problem have been studied later. A comprehensive survey and further references can be found in [KP98].

A different online variant of the bipartite matching problem is introduced in [Rie99]. It is also motivated by a scheduling problem and is called *online request server matching* problem. The two partitions of the graph represent unit size requests and unit time intervals of a server, and an edge connects a request with a future service time. In contrast to our model, every request is allowed to specify an arbitrary set of edges. Each time step, one request is revealed and for the node representing the next time interval of the server a matching edge is determined. The author presents an optimal deterministic, 1.5-competitive online algorithm. Our model was first introduced in [MBLR97]. However, there the authors present only fairly simple and apparent bounds.

1.2 Our Model

In this section we present our scheduling model and the notation we need to describe and analyze our scheduling strategies.

We have a set $S_1 \ldots, S_n$ of n *resources* working in synchronized rounds. Every resource can fulfill one request per round. During every round, a set of uniform *requests* arrives. Every of these requests specifies two alternative resources and requires to access one of them. Furthermore, every request has a deadline of d which means that a request that arrives in round t cannot be fulfilled later than in round $t + d - 1$. The aim is to maximize the number of requests that are fulfilled before their deadlines expire. We assume that the number of requests arriving during some round and their alternative resources are generated by an adversary.

The scenario above can be modeled as a bipartite graph $G = (R \cup S, E)$. The set $R = \{r_1, r_2, \ldots\}$ represents the sequence of requests (which are generated by the adversary), ordered by 1) their entry round, and 2) a request identifier which numbers the requests arriving at the system during the same round. $S = \{s_{1,1}, s_{2,1}, \ldots, s_{n,1}, s_{1,2}, s_{2,2}, \ldots, s_{n,2}, \ldots\}$ represents the set of *time slots* of the resources. Time slot $s_{i,j}$ represents the ith resource during the jth round. Every node of the bipartite graph representing request $r_i \in R$ is *directed* (i.e. it has an edge) to the $2 \cdot d$ nodes of S which represent the time slots of the two alternative resources of r_i in which the request can be fulfilled. That is, if r_i arrives at round t and specifies the two resources S_j and S_k, the node representing r_i will be directed to the nodes representing $s_{j,t}, s_{j,t+1}, \ldots, s_{j,t+d-1}$ and $s_{k,t}, s_{k,t+1}, \ldots, s_{k,t+d-1}$.

A matching M of a graph is a subset of edges such that no two edges of M are adjacent. A node incident to an edge of M is called *matched* and all others are called *free*. A *maximal matching* M_{MAX} is a matching which cannot be enlarged by an additional edge without destroying the matching property. A graph may have several different maximal matchings and, especially, maximal matchings of different cardinality. A *maximum cardinality matching* M_{MCM} is a matching of maximum size, i.e. for all matchings $\tilde{M}$ of G it holds $|M_{\text{MCM}}| \geq |\tilde{M}|$.

Since a resource can fulfill at most one request per round, every solution to our scheduling problem can be modeled as a matching in the corresponding graph $G = (R \cup S, E)$: For every request r_i that is scheduled by resource S_j at time t, the matching contains the edge connecting r_i with $s_{j,t}$. In the following, we will say that this matching is the matching *induced* in G by the given solution. An optimal solution to the scheduling problem corresponds to a maximum matching in G.

In general, an online algorithm A cannot compute a maximum matching in G because it does not know the whole graph G in advance. A can only work on a subgraph G_t of G in round t. The node set of G_t consists of all nodes in S belonging to time slots $s_{i,t+j}$ with $1 \leq i \leq n$ and $0 \leq j \leq d-1$, and all nodes representing requests revealed up to round t without requests that have been already fulfilled or have expired deadlines. All edges of G which connect these two sets of nodes build the edge set of G_t. During every round t, A performs three steps:

1. Delete requests of which the deadline has expired and receive new requests.
2. Determine a matching on G_t. Here we will investigate different strategies.
3. Fulfill all requests scheduled at time slots $s_{i,t}$, $1 \leq i \leq n$.

In order to compare the solution computed by an online algorithm with an optimal solution, we compare a fixed maximum cardinality matching in G with the matching in G induced by the online algorithm. An efficient way to do this is to use *augmenting paths*:

Let (G, M) denote a matching $M \subset E$ in a graph $G = (V, E)$ and let $\overline{M} = E \setminus M$. A path (resp. circuit) $\mathcal{P}$ in (G, M) will be called *alternating* if every node of $\mathcal{P}$ (except of the two end nodes if $\mathcal{P}$ is a path) has one edge in $M \cap \mathcal{P}$ and one edge in $\overline{M} \cap \mathcal{P}$. For any two matchings M_1 and M_2 in G, let $M_1 \oplus M_2$ be the graph consisting of all nodes V of G and the edges $(M_1 \setminus M_2) \cup (M_2 \setminus M_1)$ (symmetric difference). All components of that subgraph are paths and circuits which are alternating for (G, M_1) and (G, M_2), because a node meets at most two edges in $M_1 \oplus M_2$: one in $M_1 \setminus M_2$ and one in $M_2 \setminus M_1$. Thus, every last node of a path is free (i.e. unmatched) either for M_1 or M_2.

An alternating path $\mathcal{P}$ in (G, M) connecting two free nodes contains one more edge of $\overline{M}$ than of M. This means that $M \oplus \mathcal{P}$ is a matching of G that is larger by one than M. Such a path is called *augmenting*. A matching M is not of maximum cardinality if, and only if (G, M) contains an augmenting path.

In the case of our graph G and a given matching M (computed by an online algorithm A) an augmenting path $\mathcal{P}$ of *order* ℓ has structure $\mathcal{P} = r_1 \text{ - } s_{q_1,t_1} \text{ - } r_2 \text{ - } s_{q_2,t_2} \text{ - } \ldots \text{ - } r_\ell \text{ - } s_{q_\ell,t_\ell}$. For $1 \leq i \leq \ell$, r_i represents a request and s_{q_i,t_i} represents round t_i at resource S_{q_i}. $\mathcal{P}$ starts with an unmatched request r_1 having S_{q_1} as alternative resource and ends with a time slot s_{q_ℓ,t_ℓ} not matched by M. For $2 \leq i < \ell$ it contains requests r_i that are matched to $s_{q_{i-1},t_{i-1}}$ by M and have S_{q_i} as alternative resource. The size of M increases by one when all request nodes of $\mathcal{P}$ are matched to their opposite time slots in $\mathcal{P}$, i.e. the time slots which are not matched in M. As an augmenting path of order ℓ contains exactly ℓ nodes representing resources and ℓ nodes representing time slots, A only fulfills $\ell - 1$ requests, but an optimal strategy is able to fulfill each of the ℓ requests.

As it has been already stated in the introduction we will use competitive analyses. Let OPT be an optimal offline algorithm for a payoff maximization problem. For any input sequence σ, we denote by $\text{perf}_A(\sigma)$ the performance of the algorithm A and by $\text{perf}_{\text{OPT}}(\sigma)$ the performance of OPT respectively. A will be called c*-competitive* if there is a constant α such that

$$\forall \sigma : \quad \text{perf}_{\text{OPT}}(\sigma) \ \leq \ c \cdot \text{perf}_A(\sigma) + \alpha \ .$$

The infimum over all values c such that A is c-competitive is called the *competitive ratio* of A. This analysis is independent of a stochastic model for the inputs and gives performance *guarantees*. On the other hand this kind of worst case analysis may sometimes be unrealistically pessimistic.

1.3 New Results

In this section we present our new results. In the first part we summarize our upper and lower bounds on the competitiveness of simple, global online strategies. In the second part we summarize our results for local online strategies.

Global Strategies

As the algorithms presented in this section are only roughly defined, they can be seen as a whole class of algorithms. Thus, in the following we will likewise use the name 'strategy' instead of 'algorithm'. We consider the following strategies:

$\boxed{A_{\text{fix}}}$: For every round t, choose any maximal matching in G_t with the property of 1) every request that is already matched to some time slot $s_{i,t'}$, $t' \geq t$, in G_{t-1} is also matched to this time slot in G_t and 2) a maximum number of requests generated at t is scheduled in G_t. Thus, no rescheduling of requests is allowed once they are scheduled.

$\boxed{A_{\text{current}}}$: For every round t, choose any maximum matching between all nodes representing requests not yet fulfilled and all nodes representing time slots of the current round. All nodes that belong to later time steps are not considered.

$\boxed{A_{\text{fix_balance}}}$: For every round t, choose any maximal matching in G_t with the property that 1) every request that is already matched to some time slot $s_{i,t'}$, $t' \geq t$, in G_{t-1} is also matched to this time slot in G_t and 2) the function $F = \sum_{j=0}^{d-1} X_{t+j} \cdot (n+1)^{d-j}$ is maximized. X_t denotes the number of matched nodes representing time slots $s_{i,t+j}$ with $1 \leq i \leq n$. F ensures that the new requests are assigned to the resources in such a way that the number of scheduled requests per resource is as balanced as possible.

$\boxed{A_{\text{eager}}}$: For every round t, choose any maximum matching in G_t with the property of 1) a maximum possible number of requests is scheduled at round t and 2) all previously scheduled requests remain scheduled (but are allowed to be moved to other time slots).

$\boxed{A_{\text{balance}}}$: For every round t, choose any maximum matching in G_t with the property of 1) the function F defined in $A_{\text{fix_balance}}$ is maximized and 2) all previously scheduled requests remain scheduled (but are again allowed to be moved to other time slots).

Table 1 summarizes our results for these strategies. The lower bounds are shown in Section 2 and the upper bounds are shown in Section 3. A denotes an arbitrary online strategy.

Local Strategies

Within the scope of local strategies we assume that every new request does not know anything about other requests and their alternative recourses at the beginning. In order to make reasonable decisions, the requests and resources are allowed to communicate with each other. During these *communication rounds* the resources and requests can exchange fixed size messages. With exchanging messages we mean that the requests can send a message to the resources and receive a message as the response to their message. This approach is also used in balls-into-bins games where, during each communication round, each ball is allowed to exchange messages with some bins (see [ACMR95]). We assume that the bandwidth of the system is sufficiently large such that up to d messages can reach a resource in each communication round. If more than d messages are sent to a resource in one communication round, it receives only d of them (selected according to the LDF (latest deadline first) rule). The senders of the other messages will be informed that their messages failed.

We measure the performance of our local strategies in terms of communication rounds. This is reasonable as the time to exchange information in a distributed system usually by far dominates the time of internal computations. Our aim is to find strategies that have a good competitive ratio and only require a constant number of communication rounds.

In Section 3.2 we present a local variant of the A_{fix} algorithm, called $A_{\text{local_fix}}$ that only requires two communication rounds. Furthermore we show that $A_{\text{local_fix}}$ is 2-competitive.

Algorithm	Lower Bound	Theorem	Upper Bound	Theorem
A_{fix}	$2-1/d$	2.1	$2-1/d$	3.3
A_{current}	$4/3$ for $d=2$ $e/(e-1)$ for $d\to\infty$	2.4, 2.2	$2-1/d$	3.3
$A_{\text{fix_balance}}$	$4/3$ for $d=2$ $3d/(2d+2)$ for $d>2$	2.4, 2.3	$4/3$ for $d=2$ $7/5$ for $d=3$ $2-2/d$ for $d>3$	3.4
A_{eager}	$4/3$	2.4	$4/3$ for $d=2$ $(3d-2)/(2d-1)$ for $d>2$	3.5
A_{balance}	$4/3$ for $d=2$ $(5d+2)/(4d+1)$ for $d=3x-1,\ x\in\mathbb{N}$	2.4, 2.5	$4/3$ for $d=2$ $6(d-1)/(4d-3)$ for $d>2$	3.6
A	$45/41$ for $d=3x, x\in\mathbb{N}$	2.6		

Table 1: Upper and lower bounds for the global strategies.

Moreover, we present a local variant of the A_{eager} algorithm, called $A_{\text{local_eager}}$, that requires 9 communication rounds. We show that $A_{\text{local_eager}}$ has a competitive ratio of at most $5/3$.

2 Lower Bounds

In this section we present lower bounds on the competitive ratio of the scheduling strategies defined in Section 1.3. The lower bounds are of existential nature. That is, we show that there are implementations of the strategies presented in Section 1.3 for which the lower bounds shown below hold. This indicates which strategies might have the potential to be efficient in any form of implementation.

At first we define 'block(a,d)', a basic input structure which will be used in the following proof. It consists of $a{\cdot}d$ requests which are all generated in the same round. The requests are directed to a different resources $S_0,\ldots,S_{a-1}$: For all $i\in\{0,\ldots,a-1\}$ there is a group of d requests directed to S_i and $S_{(i+1) \bmod a}$. All requests can be fulfilled using all possible d time slots of all a resources. Such a block(a,d) is very dense. Thus, it can *block* other requests and break dependencies in a sense that no further gain can be achieved when a resource involved in the block(a,d) fulfills another request. A frequently used structure is block$(2,d)$ which consists of $2d$ requests directed to two resources S_0 and S_1 (every request can be fulfilled by both of the two resources).

The first lower bound we present is the one for A_{fix}, which is not allowed to perform a reallocation.

Theorem 2.1 *If the number of resources is at least* 4 *the competitive ratio of* A_{fix} *is at least* $2-\frac{1}{d}$.

Proof: The lower bound proof uses a sequence of identical phases, each consisting of d rounds. For all $i\geq 1$, the ith phase starts in round $i\cdot d-1$. Initially, at round 0 the adversary generates a block$(2,d)$ consisting of requests directed to S_2 and S_3. Then it generates the same set of requests in the first two rounds of every phase.

In the first round of the first phase S_2 and S_3 are still busy. During that round (and every first round of a phase) the adversary generates $2d-2$ requests divided into two equally sized groups $\mathcal{R}_1$ and $\mathcal{R}_2$. The requests of $\mathcal{R}_1$ are directed to S_1 and S_2, and the requests of $\mathcal{R}_2$ are directed to S_3 and S_4. In the second round of every phase the adversary generates a block$(2,d)$ consisting of requests which are directed to S_2 and S_3.

The next phase starts one round after the deadlines of the requests in $\mathcal{R}_1$ and $\mathcal{R}_2$ expired. Thus, at the beginning of every phase, resources S_2 and S_3 are still busy for one round.

Obviously, the optimal strategy can fulfill every of the $4d-2$ requests injected in a phase. $\mathcal{R}_1$ is fulfilled by S_1 and $\mathcal{R}_2$ by S_4. The requests of the block are fulfilled by S_2 and S_3.

A_{fix}, however, can be implemented in a way that the adversary can force A_{fix} to compute a maximum matching in which $\mathcal{R}_1$ is scheduled at S_2 and $\mathcal{R}_2$ at S_3. Thus, as A_{fix} is not allowed to perform a reallocation, it can only fulfill $2d$ of the $4d-2$ requests. This yields a lower bound on the competitive ratio of at least $\frac{4d-2}{2d}=2-\frac{1}{d}$. ∎

The next lower bound holds for algorithm A_{current}, which is only allowed to compute a maximum cardinality matching on the part of G_t which consists of the nodes representing the current time slots and the unfulfilled requests connected to them.

Theorem 2.2 *For* $d=\ell!$ *(*$\ell\in\mathbb{N}$*) and at least* ℓ *available resources it holds: The competitive ratio of* A_{current} *is at least* $e/(e-1)\approx 1.58$ *for* $d\to\infty$.

Proof: The lower bound proof uses a sequence of identical phases, each consisting of d rounds, and ℓ resources with $d=\ell!$. During every phase the adversary generates ℓ groups $\mathcal{R}_1,\ldots,\mathcal{R}_\ell$ of d requests each. The first alternatives of all requests in $\mathcal{R}_i$ $(1\leq i<\ell)$ are evenly distributed among $S_1,\ldots,S_{\ell-i}$, and the second alternative of every request in $\mathcal{R}_i$ is $S_{\ell-i+1}$. Set $\mathcal{R}_\ell$ is equivalent to $\mathcal{R}_{\ell-1}$. Thus, for all $1\leq i<\ell$ and $1\leq j\leq \ell-i$, S_j is chosen as an alternative by $d/(\ell-i)$ requests of the ith request group and $S_{\ell-i+1}$ is chosen as an alternative of all d requests of that group.

All requests of a phase are injected in its first round. The optimal strategy schedules every request in $\mathcal{R}_i$ at S_i. Therefore, all requests (ℓd in every phase) can be fulfilled.

A_{current}, however, can be implemented in a way that the adversary can force A_{current} to first fulfill the requests in $\mathcal{R}_1$, then the requests in $\mathcal{R}_2$, and so on. This holds since A_{current} only tries to maximize the number of requests fulfilled in the current round, without considering the future. Thus, A_{current} can only fulfill requests belonging to the first k groups with $\sum_{i=1}^{k} d/(\ell-i+1)\leq d$. Some simple but lengthy computations yield that A_{current} can fulfill $(e-1)/e\cdot\ell d$ requests per phase for $d\to\infty$. This yields a lower bound on the competitive ratio of $e/(e-1)\approx 1.58$ for $d\to\infty$. ∎

The previous two lower bounds show that, in order to have an efficient schedule, it is too restrictive to prohibit the rescheduling of requests when only choosing any maximum matching for the new

requests, or to compute a maximum matching only for the time slots belonging to the current round.

Next we examine $A_{\text{fix_balance}}$, which tries to distribute the requests among the resources more evenly in order to avoid having overloaded resources as quickly as the previous two algorithms. Furthermore, function F used by $A_{\text{fix_balance}}$ ensures that the requests are fulfilled as early as possible. Like algorithm A_{fix}, algorithm $A_{\text{fix_balance}}$ is only allowed to extend the matching that is computed during the last round without rescheduling requests.

For the ease of presentation we assume for the next two theorems that d is even. The same bounds also hold for the case d being odd.

Theorem 2.3 *If the number of resources is at least* 6 *the competitive ratio of* $A_{\text{fix_balance}}$ *is at least* $\frac{3d}{2d+2}$.

Proof: The theorem is shown with the help of an infinite sequence of phases. At round 0 the adversary generates a block$(2, d)$ consisting of requests which are directed to S_1 and S_2 blocking the two resources for the next d rounds. Round $d/2$ is the beginning of the first phase. In this phase, S_1 and S_2 are blocked for the next $d/2$ rounds. Every phase starts $d/2+1$ rounds after the previous one.

At the beginning of the first phase the adversary generates two request groups $\mathcal{R}_1$ and $\mathcal{R}_2$, each consisting of $d/2$ requests. The requests in $\mathcal{R}_1$ are directed to S_1 and S_3, and the requests in $\mathcal{R}_2$ are directed to S_2 and S_4. One round later the adversary generates a block$(2, d)$ consisting of requests directed to S_3 and S_4.

The second phase starts at round $d+1$. As in the first phase two resources (S_3 and S_4) are blocked for the next $d/2$ rounds. Thus, we have the same situation as at the beginning of the first phase. Now S_3 and S_4 play the role of S_1 and S_2. S_5 and S_6 take over the part of S_3 and S_4. S_1 and S_2 remain empty as S_5 and S_6 in the first phase. This 'switching strategy' can be carried on in a round robin fashion for the next phases.

The optimal strategy can fulfill all $2d+2 \cdot d/2 = 3d$ requests injected in every phase. Obviously, an optimal solution has to schedule the block requests at the corresponding resources and the requests in $\mathcal{R}_1$ and $\mathcal{R}_2$ at the remaining alternative resources.

As $A_{\text{fix_balance}}$ has to balance the number of requests scheduled at the resources, it schedules the requests in $\mathcal{R}_1$ and $\mathcal{R}_2$ at S_3 and S_4. The algorithm is not allowed to reallocate any of the requests. Thus, only $d+2$ of the $2d$ block requests (arriving one round later) can be fulfill. Since altogether $3d$ requests are injected in every phase, this results in a lower bound on the competitive ratio of at least $3d/(2d+2)$. ∎

The following theorem presents a lower bound on the competitive ratio of A_{eager}. However, for $d = 2$ it can also be used for A_{current}, $A_{\text{fix_balance}}$, and A_{balance}.

Theorem 2.4 *If the number of resources is at least* 4 *the competitive ratio of* A_{eager} *is at least* $4/3$ *for any* $d \geq 2$.

Proof: The theorem is shown with the help of an infinite sequence of overlapping phases, each spanning $3d/2$ rounds. For all $i \geq 1$, phase i starts at round $i \cdot d + d/2$. Initially, at round 0 a block$(2, d)$ is generated consisting of requests directed to S_1 and S_4.

In the first round of every odd phase the adversary generates $2d$ requests divided into three groups $\mathcal{R}_1$, $\mathcal{R}_2$ and $\mathcal{R}_3$. $\mathcal{R}_1$ contains $d/2$ requests directed to S_1 and S_2, $\mathcal{R}_2$ contains $d/2$ requests directed to S_3 and S_4, and $\mathcal{R}_3$ contains d requests directed to S_2 and S_3. After $d/2$ rounds the adversary generates a block$(2, d)$ consisting of requests directed to S_2 and S_3. Every even phase works in the same way as an odd phase with the only difference that the requests in $\mathcal{R}_3$ and the block requests are directed to S_1 and S_4.

The optimal strategy can fulfill all $4d$ requests. In every odd phase it schedules the requests in $\mathcal{R}_1$ at S_1 and the requests in $\mathcal{R}_2$ at S_4. The requests in $\mathcal{R}_3$ and the block requests can be scheduled in S_2 and S_3. (A similar strategy works for even phases.)

Nevertheless, A_{eager} can be implemented in a way that it schedules in the first $d/2$ rounds of each phase the requests in $\mathcal{R}_1$ and $\mathcal{R}_2$. Then it is only able to schedule $2d$ of the $3d$ requests in $\mathcal{R}_3$ and the block. This yields a competitive ratio of at least $4/3$. ∎

The previous lower bound indicates that it might be profitable to compute a maximum matching over the whole known subgraph and to allow the rescheduling of requests. The next algorithm A_{balance} additionally tries to balance the number of requests which have to be fulfilled at the resources, using the same function as $A_{\text{fix_balance}}$. Like A_{eager}, the algorithm A_{balance} is allowed to compute a maximum matching over the whole subgraph G_t in round t.

Theorem 2.5 *For any* $d = 3x - 1$ *with* $x \in \mathbb{N}$ *it holds: The competitive ratio of* A_{balance} *is at least* $\frac{5d+2}{4d+1}$ *in the limit* $n \to \infty$.

Proof: The theorem is shown by a construction that uses an infinite sequence of intervals of $2x$ rounds. Every interval consists of two phases described below. Requests are only generated at the beginning of a phase. The key idea is to exploit the property that A_{balance} does not have a rule that prefers requests having a heavily requested resource as a second alternative.

The strategy uses $k = \lfloor (n-2)/3 \rfloor$ groups of three resources, S_1^i, S_2^i, and S_3^i, with $1 \leq i \leq k$ and two additional resources S' and S'', which are intended to be permanently blocked. Furthermore, we use a structure called 'block$(1, d)$'. block$(1, d)$ denotes a block of d requests which are directed to S' and to one other resource S_1^i. Initially, in round 0 a block$(2, d)$ is generated and directed to S' and S''. Furthermore, for every group i, a block$(1, d)$ is generated for S_1^i. Afterwards the following requests are injected for every group i.

Phase 1 starts in round x. The adversary generates two groups $\mathcal{R}_1$ and $\mathcal{R}_2$ of x requests each. The requests in $\mathcal{R}_1$ are directed to S_1^i and S_2^i and the requests in $\mathcal{R}_2$ are directed to S_2^i and S'. A_{balance} can be implemented in a way that it first fulfills the x requests in $\mathcal{R}_1$ (in round $x, \ldots, 2x-1$) and afterwards the requests in $\mathcal{R}_2$ (in round $2x, \ldots, 3x-1$) by S_2^i. Due to the block requests injected at round 0, S_1^i is still blocked in rounds $x, \ldots, 3x-2$. Hence, the schedule is conform to the balancing rule of A_{balance}. The requests in $\mathcal{R}_1$ are fulfilled after x rounds, where Phase 1 ends.

Phase 2 starts in round $2x$. The adversary ensures the continuous blocking of S' and S'' by additional $4x$ requests and generates a block$(1, d)$ at S_2^i. In the next x rounds A_{balance} can only fulfill altogether x of the requests in $\mathcal{R}_2$ and the block at S_2^i. In round $3x$ (x rounds after Phase 2 started) the resource S_2^i is blocked for another $2x-1$ rounds. This situation is similar to the beginning of Phase 1. The adversary therefore starts the next Phase 1 x rounds after Phase 2 started. This time, S_2^i plays the role of S_1^i, S_3^i becomes S_2^i, and S_1^i remains without new requests as S_3^i did before.

Since this strategy is played for k groups, the requests with alternatives S' and S'' do not influence the competitive ratio if n is arbitrarily large. Thus it suffices to count the remaining requests.

In the two phases above, A_{balance} fulfills $2x-1$ block requests at S_1 and the x requests in $\mathcal{R}_1$ and altogether x further requests from $\mathcal{R}_2$ and the block at S_2. An optimal solution first fulfills the requests in $\mathcal{R}_2$ at S_2 and the requests in $\mathcal{R}_1$ shortly before their deadlines expire at S_1 (in the above example: rounds $3x-1, \ldots, 4x-2$). Hence, it can fulfill all block requests. Summing up these numbers and using the fact that $x = (d+1)/3$ leads to a lower bound on the competitive ratio of

$$\frac{5x-1}{4x-1} = \frac{5d+2}{4d+1} \qquad \text{in the limit } n \to \infty. \quad \blacksquare$$

Finally, a general lower bound on the performance of any online algorithm is given. It shows that only a small number of resources ($n \geq 10$) is necessary to enforce a competitive ratio of almost 1.1.

Theorem 2.6 *If the number of resources is at least* 10*,* $3 \mid d$*, or* $d \to \infty$ *every deterministic online algorithm* A *has a competitive ratio of at least* $\frac{45}{41} = 1.\overline{09756}$.

Proof: The theorem is proven by an adversarial strategy which divides the time into intervals of d rounds, consisting of two phases each. At first we assume the case $3 \mid d$. Phase 1 lasts $d/3$ rounds and Phase 2 lasts $2d/3$ rounds. Requests are generated at the beginning of a phase only. The adversary uses ten resources divided into five groups (called $\mathcal{S}_{\text{I}}$, $\mathcal{S}_{\text{II}}$, $\mathcal{S}_{\text{III}}$, $\mathcal{S}_{\text{IV}}$, and $\mathcal{S}_{\text{V}}$) with two resources each.

Initially, in round 0, a block$(6, d)$ of $6d$ requests blocks the six resources in $\mathcal{S}_{\text{I}}$, $\mathcal{S}_{\text{II}}$, and $\mathcal{S}_{\text{III}}$.

Phase 1 starts in round $2d/3$ and ends in round $d-1$. The adversary injects $4d$ requests in three equally sized groups (they are called *colored* with 'blue', 'green', and 'red'). The first alternatives of the requests are evenly distributed among the four resources in $\mathcal{S}_{\text{IV}}$ and $\mathcal{S}_{\text{V}}$ (that is, each of these resources is assigned to exactly $d/3$ requests of each group). The second alternatives are evenly distributed among the resources in $\mathcal{S}_{\text{I}}$ for the blue requests, in $\mathcal{S}_{\text{II}}$ for the green, and in $\mathcal{S}_{\text{III}}$ for the red requests.

Phase 2 starts in round d and ends in round $d+2d/3-1$. Obviously, there exists a colored request group with a maximal number of not fulfilled requests (e.g. the red group). The adversary generates $6d$ requests to block that group, i.e. the block requests are directed to $\mathcal{S}_{\text{IV}}$ and $\mathcal{S}_{\text{V}}$, and to $\mathcal{S}_{\text{III}}$ with red requests only.

The deadlines of colored requests expire at the end of Phase 2. Now we have a similar situation like at the beginning of Phase 1: three resource groups are still blocked for the next $d/3$ rounds. After renaming the resource groups the whole strategy is repeated with a new Phase 1 starting in round $d + 2d/3$.

An optimal solution for the request sequence generated by the adversary fulfills all requests. The resources and time slots which have to be used to fulfill the block requests are unique. During Phase 2 two resource groups out of $\mathcal{S}_{\text{I}}$, $\mathcal{S}_{\text{II}}$, and $\mathcal{S}_{\text{III}}$ are not blocked. Each of them fulfills all $4d/3$ requests of one color (this color is unique). Requests of the third color are fulfilled during Phase 1 by $\mathcal{S}_{\text{IV}}$ and $\mathcal{S}_{\text{V}}$.

For purpose of the analysis of A we assume w.l.o.g. that all block requests are fulfilled. In Phase 1 at most $4d/3$ colored requests are fulfilled by $\mathcal{S}_{\text{IV}}$ and $\mathcal{S}_{\text{V}}$. Therefore, $4d - 4d/3 = 8d/3$ colored requests are left and an averaging argument yields the existence of a request group with at least $\lceil 8d/9 \rceil$ not fulfilled requests. As such a request group is blocked during Phase 2, we can conclude that at least $\lceil 8d/9 \rceil$ requests are not fulfilled by the online algorithm A within the two phases.

The number of requests in the input sequence is $4d$ for every Phase 1 (colored requests) and $6d$ for every Phase 2 (block$(6, d)$). The infinite repetition of this adversarial strategy leads to a lower bound on the competitive ratio of

$$\frac{10d}{10d - \left\lceil \frac{8}{9}d \right\rceil} \geq \frac{10d}{10d - \frac{8}{9}d} = \frac{45}{41} = 1.\overline{09756} \ .$$

In case $3 \nmid d$, the number of rounds of Phase 1 has to be rounded down to $\lfloor d/3 \rfloor$. At the same time the number of requests in every colored group has to be slightly decreased. However, both values differ from the above described strategy by small and *constant* numbers only. With increasing values for parameter d the number of requests in every phase increases, too. Hence, the influence of the constant deviation on the result disappears in the limit $d \to \infty$. Indeed, the adversary asymptotically achieves a lower bound of $45/41$ for the competitive ratio. $\blacksquare$

A careful analysis and the consideration of slightly different strategies for the adversary for a few special cases show that a lower bound of $12/11 = 1.\overline{09}$ for the competitive ratio can be guaranteed for all values of d.

3 Upper Bounds

In this section we prove upper bounds on the competitiveness of global and local variants of the scheduling algorithms presented in Section 1.3. Obviously, the simple Earliest Deadline First (EDF) strategy is optimal in the case of each request specifying only one resource. As every resource works independently from the other resources and schedules the jobs in the order of increasing deadlines (ties are broken arbitrarily), the EDF strategy is a local strategy.

Observation 3.1 *If each request has only one alternative the EDF strategy is* 1*-competitive.*

Proof: (sketch) In the following we call a time slot *lossy* if it is used by the optimal strategy, but not by EDF.

Clearly, any solution computed by an optimal strategy can be transformed into a so-called *greedy* solution in which every request is scheduled as early as possible, i.e. no time slot remains unused as long as there are requests which can be scheduled in the time slot. Then, using an inductive argument, it can easily be shown that for every round t the total number of active, not yet scheduled requests in EDF is at least the number of active, not yet scheduled requests in any greedy optimal solution. This implies that there cannot be a lossy time slot and EDF executes at least as many requests as any optimal solution. Thus EDF is 1-competitive. $\blacksquare$

As it is not difficult to show, this observation also holds for the case of the requests having different deadlines. From the above observation it follows that, if every request has two alternative resources, EDF has a competitive ratio of at most 2. In order to show why this holds we assume the two copies of a request specifying the alternative resources to be handled as being independent from each other. As EDF is 1-competitive for the copies of the requests and every request is fulfilled at most twice (i.e. at most one copy per request is fulfilled by EDF without further gain), the strategy has a competitive ratio of at most 2. This is also a lower bound, since there are simple examples for which EDF is exactly 2-competitive. Hence, the following observation holds.

Observation 3.2 *If each request has two alternative resources the EDF strategy is* 2*-competitive.*

Note that the observation will also hold if the requests have different deadlines. The above observation can be extended to the case of each request having $c \in \mathbb{N}$ alternatives. In this case, EDF is c-competitive. In the next subsection we present global and local strategies that have a competitive ratio of less than 2.

3.1 Global strategies

The next theorem shows that A_{fix} and A_{current} are better than EDF.

Theorem 3.3 *Algorithms* A_{fix} *and* A_{current} *have a competitive ratio of at most* $2 - \frac{1}{d}$.

Proof: In the following we only study A_{fix}. The same analysis, however, holds for A_{current}.

No request that fails in A_{fix} is the beginning of an augmenting path of order 1 (i.e. an unfulfilled request is directly connected to an unused time slot), since otherwise the rule of A_{fix} having to compute a maximal matching is violated. Thus, every augmenting path is of order of at least 2 and we can identify a uniquely scheduled request lying on the augmenting path of r for every failed request r. We choose the request connected to the last time slot of the path. In order to show the theorem it remains to find one more scheduled request that can be uniquely assigned to $d-1$ failed requests which are the beginning of augmenting paths. This is done as follows:

Consider any non-empty set $\mathcal{R}$ of failed requests that were injected in round t. We build a set of overloaded resources $\mathcal{S}$, beginning with a set $\mathcal{S}$ containing all alternative resources of these failed requests. Then we continue to add resources to $\mathcal{S}$ that are alternatives of requests that are injected in time t and scheduled at resources in $\mathcal{S}$, until $\mathcal{S}$ does not grow any further. Obviously, for every resource $S_i \in \mathcal{S}$, $s_{i,t+d-1}$ must be matched to a request injected at time t. This holds since these resources are overloaded. Otherwise the A_{fix} rule is again violated.

Since at most $d \cdot |\mathcal{S}|$ requests with injection time t can be scheduled in $\mathcal{S}$ it holds that, if $|\mathcal{R}| > (d-1)|\mathcal{S}|$, even an optimal solution will not be able to schedule all requests injected at time t. Hence, the number of requests in $\mathcal{R}$ that are the beginning of an augmenting path can be at most $(d-1)|\mathcal{S}|$. As all resources in $\mathcal{S}$ are overloaded, each of them has to schedule at least one request with injection time t. None of these requests can be connected to the last time slot of an augmenting path (even if it belongs to a different group of failed requests), since otherwise the rule that A_{fix} has to compute a maximal matching is violated. Thus we can identify at least d scheduled requests per $d-1$ failed requests, which results in a competitive ratio of at most $(d+(d-1))/d = 2-1/d$. ∎

From the lower bounds section we know that A_{fix} in general cannot have a competitive ratio of less than $(2-\frac{1}{d})$. Hence, the result above is tight for A_{fix}. As it is shown in the following theorem, $A_{\text{fix_balance}}$ has a better competitive ratio than A_{fix}.

Theorem 3.4 *Algorithm $A_{\text{fix_balance}}$ has a competitive ratio of at most $4/3$ for $d=2$, $7/5$ for $d=3$, and $2-\frac{2}{d}$ for any $d>3$.*

Proof: (sketch) First we need some definitions. Consider any non-empty set $\mathcal{R}_t$ of failed requests that are injected in round t. We define the set $\mathcal{S}_t$ of resources overloaded by $\mathcal{R}_t$ as follows. We start with a set $\mathcal{S}_t$ containing all alternative resources of the requests in $\mathcal{R}_t$. Then we continue to add the resources to $\mathcal{S}_t$ that are alternatives of requests that are injected in time t and scheduled at resources in $\mathcal{S}_t$, until $\mathcal{S}_t$ does not grow any further. Every set $\{s_{i,t'} \mid t \le t' < t+d\}$ with $i \in \mathcal{S}_t$ is called an *overloaded group*. Any execution of a request injected in round t at a resource that belongs to $\mathcal{S}_t$ is called an *overloaded execution*. All other executions are called *normal executions*. For any resource i, let $\mathcal{G}_i$ represent the union of all overloaded groups over time slots in $\{s_{i,t} \mid t \in \mathbb{N}\}$. Any maximal set of consecutive time slots in $\mathcal{G}_i$ is called an *overloaded interval*. Two groups $g_1 = \{s_{i,t'_1} \mid t_1 \le t'_1 < t_1+d\}$ and $g_2 = \{s_{i,t'_2} \mid t_2 \le t'_2 < t_2+d\}$ with $t_1 < t_2$ therefore will belong to different overloaded intervals if there is some time slot $s_{i,t'}$, $t_1 < t' < t_2$, that does not belong to any group in $\mathcal{G}_i$. For any overloaded interval covering time slots t_1 to t_2 of resource i for some $t_1 < t_2$, let time slot s_{i,t_1-1} denote its *head*.

Let $\mathcal{P}$ be the set of augmenting paths resulting from some outcome of $A_{\text{fix_balance}}$ and any fixed optimal solution. Obviously, the first time slot of any augmenting path in $\mathcal{P}$ must lie in some overloaded interval. Let the *exit slot* of an augmenting path be defined as the last slot it has in an overloaded interval. This time slot cannot be empty or used by an overloaded execution. Otherwise either $A_{\text{fix_balance}}$ is violated or the chosen exit slot is not the last slot the augmenting path has in an overloaded interval. Hence, there must be a normal execution at the exit slot of any augmenting path.

Now we describe as to how to count executions of requests. First of all, we count all requests executed in overloaded intervals. These requests, however, do not suffice to prove the bound on the competitive ratio given in the theorem. To find additional executions of requests, we take a closer look at overloaded groups.

For any group $g = \{s_{i,t} \mid t_1 \le t < t_1+d\}$, let its corresponding *fraction* be defined as the set of consecutive time slots from s_{i,t_1+d-1} to s_{i,t_2-1}, where s_{i,t_2} is the last time slot of the next younger overloaded group. We assign every augmenting path to the group fraction to which its exit slot belongs. This ensures that every augmenting path, and therefore every failed request, is considered in some group fraction. For every group fraction, we then bound its competitive ratio (in a way that we uniquely assign scheduled requests to each group fraction and compare this number with the number of augmenting paths with exit slots in this group fraction). The maximum competitive ratio over all group fractions will then be taken as an upper bound of the competitive ratio of the complete outcome of $A_{\text{fix_balance}}$. To bound the competitive ratio, we distinguish between 3 cases:

Case 1: The group fraction consists of only one time slot.
In this case it only consists of a time slot used by an overloaded execution. Therefore the group fraction cannot contain the exit slot of an augmenting path and thus has a competitive ratio of 1.

Case 2: The group fraction consists of two time slots.
If none of the time slots is the exit slot of an augmenting path, we will obtain a competitive ratio of 1. Otherwise, we can identify one additional execution for the time slot used by a normal execution in the same way as in Case 3b. This results in 3 scheduled requests and therefore in a competitive ratio of at most $4/3$.

Case 3: The group fraction consists of at least three time slots.
Here we will distinguish between two further cases.

Case 3a: Assume that the latest three time slots of the group fraction are used by at least two overloaded executions. If the size of the group fraction is k, $k \in \{3,\ldots,d\}$, this results in k executions of requests and at most $k-2$ augmenting paths and therefore in a competitive ratio of at most

$$\frac{d+(d-2)}{d} = 2-\frac{2}{d} \ .$$

Case 3b: Assume that only the latest time slot is used by an overloaded execution. In this case, the previous two time slots are used by normal executions. Consider any of these time slots and let it be called s_{i_0,t_0}. If s_{i_0,t_0} is not used as an exit slot of some augmenting path p, we obtain the same bound as in Case 3a. Otherwise, $A_{\text{fix_balance}}$ ensures that only the following three subcases can occur for the time slot, s_{i_1,t_1}, following s_{i_0,t_0} in p:

1. s_{i_1,t_1} is empty and s_{i_1,t_1-1} is used. In this case we count the request scheduled at s_{i_1,t_1-1}.

2. s_{i_1,t_1} is used and s_{i_1,t_1-1} is also used. In this case we also count the request scheduled at s_{i_1,t_1-1}.

3. s_{i_1,t_1} is used and s_{i_1,t_1-1} is empty (or not defined). Then the two requests scheduled at s_{i_0,t_0} and s_{i_1,t_1} must have been injected at the same time, otherwise $A_{\text{fix_balance}}$ is violated. In this case we follow p to the next time slot, s_{i_2,t_2}. It can be shown that, due to the balancing property of $A_{\text{fix_balance}}$,

for this time slot again the three subcases above suffice to be considered. Therefore we finally arrive at a time slot that either belongs to subcase 1 or subcase 2.

The time slots counted in the subcases above are disjoint, since they are always one round prior to a time slot belonging to an augmenting path. Unfortunately, it is possible that s_{i_1,t_1-1} belongs to an overloaded interval. In this case, however, it must be the last time slot of an overloaded interval since, according to the definition of the exit slot of an augmenting path, s_{i_1,t_1} cannot lie in an overloaded interval. To compensate such an overlap, instead of counting the request scheduled at s_{i_1,t_1-1} we count a different request. For this we will either choose the request scheduled at the head of the corresponding overloaded interval if the head is not empty, or otherwise any request of the group with earliest time slots in the interval (which all must be filled with overloaded executions). Requests scheduled at heads of overloaded intervals are counted only once, since none of the scheduled requests counted previously (within and outside overloaded intervals) can be scheduled at a head.

Hence we can identify an additional execution for each of two normal executions at the latest three time slots of the group fraction. If the size of the group fraction is k, $k \in \{3, \ldots, d\}$, this results in $k+2$ executions of requests and at most $k-1$ augmenting paths and therefore in a competitive ratio of at most

$$\frac{(d+2)+(d-1)}{d+2} = 2 - \frac{3}{d+2} \quad .$$

Taking the maximum over all competitive ratios yields a competitive ratio of $\max\{2-2/d, 2-3/(d+2), 4/3\}$ for any $d \geq 2$. ∎

Even better competitive ratios can be shown for A_{eager} and A_{balance}.

Theorem 3.5 *Algorithm A_{eager} has a competitive ratio of at most $\frac{3d-2}{2d-1}$.*

Proof: We first show that no request that fails in A_{eager} is the beginning of an augmenting path of order 1 or 2. Again, it holds that no canceled request is connected to an unused time slot (i.e., there cannot be an augmenting path of order 1). Assume that we have an augmenting path q - $s_{i,t}$ - r - $s_{j,t'}$ of order 2. The case $t' < t$ is a contradiction to the rule that the matching computed at round t' has to be maximum for round t'. The case $t \leq t'$ is a contradiction to the rule that A_{eager} computes a maximum matching in round t because the whole augmenting path is known in that round. Thus, every augmenting path is of order at least 3.

Hence, for every augmenting path we can identify at least two unique, scheduled requests lying on this path (take the requests connected to the last two time slots of the augmenting path). In order to show the theorem it remains to find one more scheduled request that can be assigned to $d-1$ failed requests. This can be done similar to the proof of Theorem 3.3. As a consequence, for every set of $d-1$ failed requests at least $2(d-1)+1$ requests can be identified that are scheduled by A_{eager}, which results in a competitive ratio of at most $(3d-2)/(2d-1)$. ∎

From the lower bounds section we know that A_{eager} cannot have a competitive ratio of less than $4/3$. Hence, for $d = 2$ the result above is tight. For A_{balance}, we can prove a slightly better competitive ratio.

Theorem 3.6 *Algorithm A_{balance} has a competitive ratio of at most $4/3$ for $d = 2$ and $\frac{6(d-1)}{4d-3}$ for any $d > 2$.*

Proof: (sketch) We use the same notation as in the proof of Theorem 3.4. W.l.o.g. we assume in the following that A_{balance} guarantees for every normal execution in some overloaded interval that the request scheduled there has not been moved since it was scheduled the last time before it belonged to an overloaded group. It can be shown that any other outcome can be transformed to such a situation without changing the number of scheduled requests or creating augmenting paths of order below 3.

As in the proof of Theorem 3.4, the *exit slot* of an augmenting path is defined as the last time slot it has in an overloaded interval. Again, this slot can only be used by a normal execution. Analogous to the proof of Theorem 3.5 it can be shown that, even if the exit slots are declared as the first time slots of the augmenting paths, all augmenting paths are of order at least 3. We always count the first two time slots of any augmenting path, starting with the exit slot. In order to identify further scheduled requests we use the same three cases as in the proof of Theorem 3.4. Case 1 results in a competitive ratio of 1.

Case 2: The group fraction consists of two time slots.
If none of the time slots is the exit slot of an augmenting path, we will obtain a competitive ratio of 1. Otherwise, we can count 3 scheduled requests (two due to the augmenting path) and therefore will obtain a competitive ratio of at most $4/3$.

Case 3: The group fraction consists of at least three time slots.
Here we will distinguish between two further cases.

Case 3a: Assume that the latest three time slots of the group fraction are used by at least two overloaded executions. If the size of the group fraction is k, $k \in \{3, \ldots, d\}$, this results in k executions of requests and at most $k-2$ augmenting paths of order at least 3 and therefore in a competitive ratio of at most

$$\frac{d+2(d-2)}{d+(d-2)} = \frac{3d-4}{2d-2} \quad .$$

Case 3b: Assume that only the latest time slot is used by an overloaded execution. Then consider the second latest time slot and let it be called s_{i_0,t_0}. If s_{i_0,t_0} is not used as an exit slot of an augmenting path p, we obtain the same bound as in Case 3a. Otherwise, the balancing rule of A_{balance} together with our assumption on the rescheduling of normal executions at the beginning of the proof ensure that only the following two subcases can occur for the second next time slot, s_{i_2,t_2}, of p:

1. s_{i_2,t_2} is empty and s_{i_2,t_2-1} is used. Then it can be shown that s_{i_2,t_2-1} is either the last time slot of an overloaded interval, or belongs to an augmenting path of order larger than 3, or can be uniquely counted for p.

2. s_{i_2,t_2} is used. Then s_{i_2,t_2} is counted for p.

For subcase 1 we can identify further cases: Let the time slot following s_{i_0,t_0} in p be called s_{i_1,t_1}. If s_{i_1,t_1+1} does not belong to an overloaded interval then it either belongs to an augmenting path of order larger than 3, or can be uniquely counted for p. Otherwise s_{i_1,t_1+1} is the first time slot of an overloaded interval. Let g be the overloaded group to which s_{i_1,t_1+1} belongs. Clearly, the request scheduled at s_{i_0,t_0+1} was injected at the same time as the requests belonging to overloaded executions in g.

If the request scheduled at s_{i_1,t_1+d-2} was injected at the same time as the request scheduled at s_{i_0,t_0} then we can consider the same subcases as above in case that s_{i_1,t_1+d-2} is the exit slot of an augmenting path.

If the request scheduled at s_{i_1,t_1+d-2} is older, then time slot s_{i_1,t_1-1} must have been used. This slot is either the last time slot

of an overloaded interval, or belongs to an augmenting path of order larger than 3, or can be counted uniquely for p.

Considering all these cases yields an (amortized) competitive ratio of $6(d-1)/(4d-3)$. (The worst case is reached if half of the overloaded groups at the beginning of overloaded intervals fall into Case 3a and the other half has an overloaded interval above s_{i_1,t_1} and s_{i_2,t_2}.)

Taking the maximum over all competitive ratios yields the theorem. ∎

3.2 Local strategies

In this section we apply the proof techniques for the global strategies above in order to construct fast local strategies with low competitive ratio.

First we present a local variant of the A_{fix} algorithm, called $A_{\mathrm{local_fix}}$. Assume t to be the current round of the scheduling problem. As in A_{fix}, all requests generated before t have been scheduled or failed.

$\boldsymbol{A_{\mathrm{local_fix}}}$ consists of two communication rounds.

Communication Round 1 Every request injected at time t (in the following called *new* request) is sent to its first alternative resource. If all requests that arrive at some resource S_i can be scheduled at S_i without any failure (i.e. they can be matched to nodes $s_{i,t'}$ with $t' \in \{t, \ldots, t+d-1\}$ that are not yet matched), all of the requests will be accepted. Otherwise S_i only accepts a maximal selection of requests that can be matched to yet unmatched nodes $s_{i,t'}$.

Communication Round 2 All failed requests are sent to their second alternative resource. The resources again accept a maximal selection of requests that can be matched to yet unmatched time slots.

The following tight bound on the competitive ratio is shown for $A_{\mathrm{local_fix}}$.

Theorem 3.7 *If the number of resources is at least 4, the competitive ratio of $A_{\mathrm{local_fix}}$ is precisely 2.*

Proof: It is easy to check that all requests that fail to be scheduled by $A_{\mathrm{local_fix}}$ must have all of their alternative time slots occupied by other requests. Otherwise the $A_{\mathrm{local_fix}}$ rule is violated. Hence, as in A_{fix}, every failed request that can be scheduled by an optimal strategy has an augmenting path of order at least 2. This implies the upper bound.

An adversarial strategy to show the lower bound divides the time into intervals of d rounds. New requests are generated in the first round of every interval only. The adversary uses four resources S_1, S_2, S_3, and S_4. It presents three groups of requests $\mathcal{R}_1$, $\mathcal{R}_2$, and $\mathcal{R}_3$. The d requests in $\mathcal{R}_1$ are directed to S_1 and S_2, $\mathcal{R}_2$ contains d requests which are directed to S_3 and S_4, and $\mathcal{R}_3$ contains $2d$ requests directed to S_1 and S_3. The adversary manages that requests of $\mathcal{R}_1$, respectively $\mathcal{R}_2$, are sent to resource S_1, respectively S_3, in the first communication round. All of them are accepted. Simultaneously, the requests of $\mathcal{R}_3$ are sent to S_1 within this first communication round and all of them are rejected. The same happens in the second communication round because S_3 has already accepted a maximal number of requests for the next d rounds.

The optimal solution fulfills all d requests in $\mathcal{R}_1$ at S_2, all d requests in $\mathcal{R}_2$ at S_4, and S_1 and S_3 each fulfill d requests of $\mathcal{R}_3$. Using this strategy in infinitely many intervals therefore yields a competitive ratio of at least 2 for $A_{\mathrm{local_fix}}$. ∎

A comparison of the results of Observation 3.2 and Theorem 3.7 leads to the following conclusion. In contrast to the simple EDF strategy, $A_{\mathrm{local_fix}}$ avoids the multiple service of a single request. Nevertheless, $A_{\mathrm{local_fix}}$ cannot achieve an improvement in a worst case situation.

Next we present an algorithm, called $A_{\mathrm{local_eager}}$, which has a better competitive ratio than $A_{\mathrm{local_fix}}$. As above, assume that t is the current round of the scheduling problem.

$\boldsymbol{A_{\mathrm{local_eager}}}$ consists of three phases. We assume that round t to start with the previously determined scheduling for the time slots of rounds $t, \ldots, t+d-2$.

Phase 1: This phase works like $A_{\mathrm{local_fix}}$ with one difference: *all* requests (newly injected and older ones) which are not yet scheduled are sent to their resources. This takes two communication rounds.

Phase 2: Every request which is accepted by a resource at a time $t' > t$ is sent to its alternative resource. Every resource S_i with an unused current time slot $s_{i,t}$ arbitrarily chooses one request and sends back an acknowledgment. After this first communication round, every request which receives an acknowledgment sends a canceling message to its previous resource. Hence, it moves to its other alternative and decreases its service time from round t' to t. This phase requires two communication rounds.

Phase 3: Every request q with non-elapsed deadline which is not yet scheduled is sent to its first alternative resource $S_{q,1}$. Phase 1 ensures that this resource has a request r scheduled at round t. Each of these requested resources selects one of the requests (called *rivaling* in the following) and sends a return message to it including request r, its alternative resource S_r, and a *tag* for highest priority which can be used the next time q communicates with $S_{q,1}$.

After this first communication round, q sends request r to S_r. Every resource, such as S_r, accepts as many requests as it can schedule and returns acknowledgments.

An acknowledgment received by q implies that request r is scheduled by S_r and can be canceled at $S_{q,1}$. In a third communication round such a successful request q sends a message to $S_{q,1}$ to exchange r with q in its schedule. It uses its high priority tag to ensure that this message is received by $S_{q,1}$.

All requests that are still unsuccessful repeat the previously given steps of this third phase for their second alternative resource, starting in the third communication round. The overlapping of the messages in the third communication round does not evoke problems. As a resource S_i receives at most one request with a high priority tag (this request occupies the first time slot $s_{i,t}$), S_i can receive a message for all other $d-1$ time slots.

The whole Phase 3 requires at most 5 communication rounds. Clearly, the total number of communication rounds required for $A_{\mathrm{local_eager}}$ in each round is at most 9.

Note: It is possible to save one communication round. For this, it is necessary to increase the maximum number of messages a resource can receive in one round to $2d-2$. In this case, the last communication round of Phase 2 and the first one of Phase 3 can be performed simultaneously.

Furthermore, the following upper bound on the competitive ratio is shown.

Theorem 3.8 *$A_{\mathrm{local_eager}}$ is $5/3$-competitive.*

Proof: Phase 1 of the protocol ensures the non-existence of augmenting paths of order 1 in the final solution.

This can be shown similar to the proof of Theorem 3.7: As all not yet scheduled requests are sent to their alternative resources all of their alternative time slots have to be occupied by other requests. Note that the priority rule for receiving messages (select messages

according to the latest deadline first rule) ensures that Phase 1 of $A_{\text{local_eager}}$ accepts a maximum number of requests.

Phase 2 and Phase 3 handle augmenting paths of order 2. Let $\mathcal{P} = q$ - $s_{q,t}$ - r - $s_{r,t'}$ be such an augmenting path. In the following we distinguish two cases $t > t'$ (Phase 2) and $t \leq t'$ (Phase 3).

Case $t > t'$: In Phase 2 of round t' request r tries to decrease its service time by requesting S_r, from which it follows that S_r cannot be unused in time slot t'. Note that the following Phase 3 is only allowed to exchange the request at $s_{r,t'}$. Thus, such an augmenting path cannot exist.

Case $t \leq t'$: This case is divided into two subcases: $t' \in \{t, t+1\}$ and $t' > t+1$. In the first subcase we can, similar to Case $t > t'$, exclude the existence of such an augmenting path, whereas this cannot be guaranteed for the second subcase.

For $t' = t$ request q (or another of q's rivaling requests) causes in Phase 3 of round t' that request r is sent to S_r. Thus, as S_r receives at least one request, time slot $s_{r,t'}$ does not remain unused.

For $t' = t+1$ request q (or another of q's rivaling requests) causes in Phase 3 of round t that request r is sent to S_r. Thus, as the priority rule ensures that S_r receives enough requests which can be fulfilled by $s_{r,t'}$, a request is assigned to $s_{r,t'}$. Furthermore, $s_{r,t'}$ cannot become unused in the next round t': Phase 1 only assigns requests, Phase 2 cannot move the request scheduled at $s_{r,t'}$ to an earlier time slot, and again Phase 3 only exchanges requests. This yields a contradiction to the assumption that $\mathcal{P}$ exists.

For subcase $t' > t+1$ the above line of arguments does not work in one special situation: In round t another request p is scheduled at $s_{r,t'}$. Later in Phase 2 of round $\tilde{t} \in \{t+1, \ldots, t'-1\}$, p is moved in to its alternative resource S_p and $s_{r,t'}$ remains unused. In the following we intend to count the execution of p to $\mathcal{P}$.

Request p cannot be the second request of an augmenting path, i.e. the following structure does not exist: $\tilde{q}$ - $s_{p,\tilde{t}}$ - p - $\ldots$. This holds as $s_{p,\tilde{t}}$ is free directly before the beginning of Phase 2 of round $\tilde{t}$. Then, Phase 2 moves request p to $s_{p,\tilde{t}}$. But this leads to a contradiction to Phase 1 of round $\tilde{t}$ because the unfulfilled request $\tilde{q}$ is sent to S_p.

Now the theorem is inferred as follows: Augmenting paths of order 1 do not exist. An augmenting path of order 2 with $t > t'$ or $t' \in \{t, t+1\}$ does not exist, too.

In case of an augmenting path $\mathcal{P}$ of order two with $t' > t+1$ we assign a unique request p to $\mathcal{P}$. Unfortunately, request p can be an element of an augmenting path $\mathcal{P}'$ itself. However, we have shown that it cannot be the second request in this path from which it follows that $\mathcal{P}'$ is of order at least three.

Sharing the gain of p among $\mathcal{P}$ and $\mathcal{P}'$ yields 1.5 fulfilled requests for $\mathcal{P}$. As $\mathcal{P}'$ also is of order at least three and p cannot be the second request, we can count at least 1.5 fulfilled requests for $\mathcal{P}'$, too. Thus, in the worst case we have 1.5 requests fulfilled by $A_{\text{local_eager}}$ compared to 2.5 requests fulfilled by the optimal solution. This yields a competitive ratio of $5/3$. ∎

References

[ABKU94] Yossi Azar, Andrei Z. Broder, Anna R. Karlin, and Eli Upfal. Balanced allocations (extended abstract). In *Proceedings of the 26th Symposium on Theory of Computing*, pages 593–602, 1994.

[ABS98] Micah Adler, Petra Berenbrink, and Klaus Schröder. Analyzing an infinite parallel job allocation process. In *Proceedings of the 6th European Symposium on Algorithms*, pages 417–428, 1998.

[ACMR95] Micah Adler, Soumen Chakrabarti, Michael Mitzenmacher, and Lars Rasmussen. Parallel randomized load balancing. In *Proceedings of the 27th Symposium on Theory of Computing*, pages 234–247, 1995.

[BMS97] Petra Berenbrink, Friedhelm Meyer auf der Heide, and Klaus Schröder. Allocating weighted balls in parallel. In *Proceedings of the 9th Symposium on Parallel Algorithms and Architectures*, pages 302–310, 1997.

[CFM+98] Richard Cole, Alan Frieze, Bruce M. Maggs, Michael Mitzenmacher, Andrea W. Richa, Ramesh K. Sitaraman, and Eli Upfal. On balls and bins with deletions. In *Proceedings of the 2nd International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 145–158, 1998.

[CS97] Artur Czumaj and Volker Stemann. Randomized allocation processes. In *Proceedings of the 37th Symposium on Foundations on Computer Science*, pages 194–203, 1997.

[Czu98] Artur Czumaj. Recovery time of dynamic allocation processes. In *Proceedings of the 10th Symposium on Parallel Algorithms and Architectures*, pages 202–211, 1998.

[KLM92] Richard Karp, Michael Luby, and Friedhelm Meyer auf der Heide. Efficient PRAM simulations on a distributed memory machine. In *Proceedings of the 24th Symposium on Theory of Computing*, pages 318–326, 1992.

[Kor97] Jan Korst. Random duplicated assignment: An alternative to striping in video servers. In *Proceedings of 5th Multimedia*, pages 219–226, 1997.

[KP98] Bala Kalyanasundaram and Kirk Pruhs. On-line network optimization problems. In A. Fiat and G.J. Woeginger, editors, *Online Algorithms: The State of the Art*, LNCS 1442, pages 268–280. Springer-Verlag, 1998.

[KVV90] Richard M. Karp, Umesh V. Vazirani, and Vijay V. Vazirani. An optimal algorithm for on-line bipartite matching. In *Proceedings of the 22nd Symposium on Theory of Computing*, pages 352–358, 1990.

[LP86] L. Lovász and M.D. Plummer. *Matching Theory*, volume 29 of *Annals of Discrete Mathematics*. North-Holland Mathematics Studies, 1986.

[MBLR97] Burkhard Monien, Petra Berenbrink, Reinhard Lüling, and Marco Riedel. Online scheduling of continuous media streams. In C. Freksa, M. Jantzen, and R. Valk, editors, *Foundations of Computer Science: Potential-Theory-Cognition*, LNCS 1337, pages 313–320. Springer-Verlag, 1997.

[Mit96] Michael Mitzenmacher. Density dependent jump markov processes and applications to load balancing. In *Proceedings of the 36th Symposium on Foundations of Computer Science*, pages 213–222, 1996.

[Mit97] Michael Mitzenmacher. On the analysis of randomized load balancing schemes. In *Proceedings of the 9th Symposium on Parallel Algorithms and Architectures*, pages 292–301, 1997.

[MSF98] R. Muntz, J.R. Santos, and F. Fabbrocino. Design of a fault tolerant realtime storage system for multimedia applications. In *Proceedings of the International Computer Performance and Dependability Symposium*, 1998.

[MV80] Silvio Micali and Vijay V. Vazirani. An $O(\sqrt{|V|} \cdot |E|)$ algorithm for finding the maximum matching in general graphs. In *Proceedings of the 21st Symposium on Foundations of Computer Science*, pages 17–27, 1980.

[Rie99] Marco Riedel. Online matching for scheduling problems. In *Proceedings of the 16th Symposium on Theoretical Aspects in Computer Science*, LNCS 1563, pages 571–580, 1999.

[Sga98] Jiří Sgall. On-line scheduling. In A. Fiat and G.J. Woeginger, editors, *Online Algorithms: The State of the Art*, LNCS 1442, pages 196–231. Springer-Verlag, 1998.

[Ste96] Volker Stemann. Parallel balanced allocations (extended abstract). In *Proceedings of the 8th Symposium on Parallel Algorithms and Architectures*, pages 261–269, 1996.