Reconfigurable Resource Scheduling with Variable Delay Bounds

C. Greg Plaxton[1], Yu Sun[2], Mitul Tiwari[2], and Harrick Vin[2]
University of Texas at Austin
Department of Computer Science
Austin, TX 78712-0233
{plaxton, sunyu, mitult, vin}@cs.utexas.edu

Abstract

Certain emerging network applications involve dynamically allocating shared resources to a variety of services to provide QoS guarantees for each service. Motivated by such applications, we address the following online scheduling problem belonging to the recently introduced class of reconfigurable resource scheduling problems: unit jobs of different categories arrive over time and need to be completed within category-specific delay bounds, or else they are dropped at a unit drop cost; processors can be reconfigured to process jobs of a certain category at a fixed reconfiguration cost; the goal is to minimize the total cost. We study this problem in the framework of competitive analysis. Through a novel combination of the EDF and LRU scheduling principles, we obtain an online algorithm that is constant competitive when given a constant factor resource advantage over an optimal offline algorithm.

1. Introduction

Multi-core and multi-processor environments are increasingly used to support a wide range of high-throughput applications, such as web services, network applications, and database servers. These environments host multiple services simultaneously (e.g., a router supporting various packet processing services).

To isolate — with respect to security and performance — services from one another, these environments often configure processors to support only one service at a time. The set of processors configured to support a particular service depends upon the associated workload; fluctuations in workload require changes in processor allocation. For instance, a shared data center dynamically adjusts the allocation of processors to independent services as the composition of the workload changes [4, 5]. Similarly, a multi-service router based on multi-core network processors adjusts the allocation of processors to different packet categories as the traffic load fluctuates [16, 17, 18]. In these systems, reallocating a processor from one category to another tends to incur a nonnegligible overhead. For instance, on Intel's IXP2400 network processor, loading the instruction store of a processor core with the code for a new category incurs a context switch time, which is much (two or three orders of magnitude) greater than the time to process a packet [8]. In certain applications involving QoS guarantees, jobs are required to be processed within a delay tolerance, where the delay tolerance is a function of the job category [9].

Problem Statement. Motivated by the aforementioned applications, we have recently introduced reconfigurable resource scheduling [14], a class of scheduling problems with the following salient features: there are jobs of different categories; resources can be reconfigured to process jobs of a certain category at an overhead, in terms of cost or time.

In this paper, we solve a specific problem in this class. The following is an informal description of this problem; a formal definition is given in Section 2. Each request is a set of unit jobs. Each job has a category, and needs to be executed within a category-specific delay bound from its arrival, or else it is dropped at a unit drop cost. A job of a given category can only be executed on a resource configured for that category. A resource can be reconfigured at any time at a fixed reconfiguration cost. The objective is to minimize the total cost. We refer to this problem as reconfigurable resource scheduling with variable delay bounds.

The high level goal of our work in reconfigurable resource scheduling is to design online algorithms that provide good performance under all possible operating conditions. This motivates us to adopt the framework of competitive analysis, where the performance of an online algorithm is measured by the competitive ratio [15], that is, the

[1] Supported by NSF Grants CNS-0326001, CCF-0310970, and CCF-0635203.
[2] Supported by NSF Grant CNS-0326001 and Texas Advanced Technology Program Grant 003658-0608-2003.

maximum ratio between the cost incurred by the online algorithm and that incurred by an optimal offline algorithm, over all request sequences. (See [1] for a comprehensive introduction to competitive analysis.) In this paper, we adopt a standard technique in competitive analysis, sometimes referred to as *resource augmentation* [7, 13], in which the online algorithm is given extra resources in order to compensate for its lack of future information. We refer to an online algorithm that achieves a constant competitive ratio when given a constant factor resource advantage as a *resource competitive* algorithm. The specific objective of the present work is to provide a resource competitive online algorithm for reconfigurable resource scheduling with variable delay bounds.

Our Contribution. To appreciate some of the difficulties associated with variable delay bounds, consider a scenario in which we are scheduling two categories of jobs on a single resource: "background" jobs and "short-term" jobs. Background jobs have deadlines far in the future, and short-term jobs have smaller delay bounds and arrive intermittently. We need to decide whether to use idle cycles to execute background jobs. If we allow background jobs to use idle cycles whenever available, we may incur a large number of reconfigurations, or drop a lot of short-term jobs; later on, we may regret incurring these costs if we encounter a lengthy interval during which no short-term jobs arrive, and during which all of the background jobs could have been executed using a single reconfiguration. On the other hand, if we do not allow background jobs to use small chunks of idle cycles, and instead wait for a long idle interval, then later on, we may regret doing so if we never encounter a long idle interval. In summary, these two basic approaches lead to either *thrashing* (i.e., excessively high reconfiguration cost) or *underutilization* (i.e., excessively high drop cost).

A natural way to try to overcome these difficulties is to consider algorithms based on the Least Recently Used (LRU) principle. To pursue this approach, we need to define an appropriate notion of an LRU timestamp in the current setting. We have investigated various natural alternatives. (See Section 3.3 for an example.) For all of these alternatives, we encounter the following basic difficulty, even with resource augmentation: If we configure the categories with the most recent LRU timestamps without considering whether these categories have jobs to execute, then we are vulnerable to underutilization; if we configure the categories with the most recent LRU timestamps and with jobs to execute, then we are vulnerable to thrashing.

Another natural approach is to consider algorithms based on the Earliest Deadline First (EDF) principle. As with LRU, there are different ways that we can formulate a specific algorithm based on the EDF principle. (See Section 3.2 for an example.) However, even with resource augmentation, all EDF variants seem to suffer from thrashing, and therefore fail to yield a resource competitive solution. Furthermore, it is not hard to argue that similar scheduling principles, such as Least Slack First, also suffer from thrashing.

Though EDF alone or LRU alone seems insufficient to solve our problem, each maintains a dynamic ordering that addresses a key aspect of the request sequence. EDF addresses the urgency aspect and tends to reduce the drop cost. LRU addresses the recency aspect and tends to reduce the reconfiguration cost. Moreover, each dynamic ordering is efficiently maintainable. It is natural to ask whether we can efficiently combine these two orderings, and thereby address both key aspects of the request sequence. In this paper, we answer the question in the affirmative. We propose a natural and efficient combination of EDF and LRU. The main idea is to keep two sets of categories configured: one set picked by the EDF principle and the other picked by the LRU principle. (See Section 3.4 for a formal definition of this combination.) We prove that this combination yields a resource competitive algorithm for reconfigurable resource scheduling with variable delay bounds. The combining mechanism that we use to combine EDF and LRU is general in nature, and can be used to combine multiple scheduling principles, each of which maintains a dynamic ordering of the jobs. The present work suggests that, for problems which cannot be solved by a single dynamic ordering, it is worthwhile to explore algorithms based on a combination of dynamic orderings.

We use a layered approach to solve reconfigurable resource scheduling with variable delay bounds. First, we use a batching subroutine to reduce the problem to the special case in which jobs of a given category arrive at integral multiples of the category-specific delay bound. Second, we reduce the batched problem to a rate-limited problem in which at most p jobs with delay bound p arrive at each integral multiple of p. Third, we solve the rate-limited problem using the aforementioned combination of EDF and LRU.

Related Work. In recent work, we introduce the class of reconfigurable resource scheduling problems, and use a layered approach to solve a variant with uniform delay bounds and variable drop costs [14]. First, we use a batching subroutine to reduce to the special case in which jobs arrive at integral multiples of a fixed delay bound. Second, we use a reshaping technique to reduce to the special case in which the delay bound is 1. Third, we use a serialization technique to reduce to a file caching problem. Fourth, we solve the file caching problem by modifying Young's Landlord algorithm [19]. There are some high level similarities between the present paper and [14]. The first layer in the present paper is analogous to the first layer in [14], but is more involved. In [14], the Landlord algorithm can be viewed as a generalization of LRU, which handles the recency aspect of the request sequence, but there is no component analogous to EDF, which addresses the urgency aspect. In summary, in order to handle variable delay bounds, the present work introduces substantially different techniques than those presented in [14]. On the other hand, since we do not handle variable drop costs in the present paper, these two works are incomparable. It remains to be seen whether the approach used in the present paper can be extended to handle other problem dimensions such as variable drop costs.

Brucker [2, Chapter 9] surveys a class of offline scheduling problems with changeover time (i.e., context switch time). Results for single and multiple machine problems are summarized. In this class of problems, each job belongs to a certain group, and between the executions of any two jobs in different groups on the same machine, there is a changeover time during which the machine cannot process any job. For a variant with identical machines, equal sized groups, and equal processing and changeover time, Brucker et al. [3] give a polynomial time offline algorithm that decides whether there exists a schedule in which all jobs are executed within a common delay bound.

Srinivasan et al. [17] discuss scheduling problems for multi-core network processors, and consider the application of existing multiprocessor scheduling algorithms in this domain. Various challenges are identified and some initial ideas are presented. Kokku et al. [8] give a scheduling algorithm, called Everest, for multi-core network processors. The parameters considered are per-service delay bounds, per-service execution requirements, and a fixed context switch time. Everest is shown to perform well in experiments in terms of maximizing the number of packets processed within service-specific delay bounds.

The EDF scheduling algorithm is shown [6, 10] to be an optimal preemptive uniprocessor scheduling algorithm for problems that do not involve reconfiguration overhead, in terms of the number of jobs executed. In this paper, we discuss the drawbacks associated with using EDF to solve the problem of reconfigurable resource scheduling with variable delay bounds, and propose a combination of EDF and LRU to address these drawbacks.

The classic disk paging problem studied by Sleator and Tarjan [15] can be viewed as a special case of reconfigurable resource scheduling with unit delay bounds, unit reconfiguration cost, infinite drop cost, and where each request consists of a single job. In this seminal work, the competitive ratio of any deterministic online paging algorithm is shown to be at least the cache size, and certain algorithms, such as LRU, are shown to be resource competitive.

O'Neil et al. [12] consider a variation of LRU called *LRU-K*, which keeps track of the time of each of the last K references to a given page. Megiddo et al. [11] consider a self-tuning cache replacement policy called Adaptive Replacement Cache, which captures the recency and frequency aspects of the request sequence by maintaining a separate ordering for each aspect. As indicated earlier, our combination of EDF and LRU captures the urgency and recency aspects of the request sequence.

Due to space limitations, proof sketches are provided for some of the results claimed in this paper. Complete proof details will be provided in the full version of the paper.

2. Preliminaries

Before we define the reconfigurable resource scheduling problems considered in this paper, we first make some preliminary definitions. We define a *request* as a (possibly empty) set of unit *jobs*, where each job is characterized by a non-black *color*, a nonnegative integer *arrival time*, a positive integer *delay bound*, and a positive integer *drop cost*. The *deadline* of a job is defined as the arrival time plus the delay bound minus one. There is a finite set of *resources* on which jobs are executed. Each resource has an associated color, which is initially black. There is a cost to reconfigure a resource, i.e., to change the color of a resource.

The processing of a given request sequence σ proceeds in rounds numbered from 0 to $|\sigma| - 1$. At the beginning of round i, we have a set of *pending* jobs, each of which has an arrival time smaller than i, and a deadline at least i. Each round i consists of four phases: (1) in the first phase, the *arrival phase*, the next request is received; (2) in the second phase, the *reconfiguration phase*, each resource can be reconfigured to a different color; (3) in the third phase, the *execution phase*, each resource configured with color ℓ can execute up to one pending job of color ℓ; (4) in the fourth phase, the *drop phase*, jobs with deadline i are dropped.

We refer to the sequence of rounds in the processing a given request sequence as a *schedule*. The number of resources used by a schedule is the number of resources that are reconfigured at least once. The cost of a schedule is the sum of all reconfiguration and drop costs incurred.

For the reconfigurable resource scheduling problems considered in this paper, the input is a pair (σ, m), where σ is a request sequence, and m is a positive integer. Given an instance (σ, m), an algorithm produces a schedule for σ. An algorithm is said to be *offline* if it knows all the requests in advance, and it is said to be *online* if does not know the future requests. An algorithm A is *b-feasible* if for any instance (σ, m), A produces a schedule that uses at most bm resources. An algorithm is *feasible* if it is 1-feasible. For any instance (σ, m) and any algorithm A, the cost of A on (σ, m), denoted $\mathit{Cost}(A, \sigma, m)$, is the cost of S, where S is the schedule produced by A on (σ, m). An algorithm A is *(a, b)-competitive* if A is b-feasible and for any instance (σ, m), $\mathit{Cost}(A, \sigma, m)$ is at most $a \cdot \mathit{Cost}(\mathit{OPT}, \sigma, m)$, where *OPT* is an optimal feasible offline algorithm. An algorithm A is *resource competitive* if A is (a, b)-competitive for some positive constant reals a and b.

For the sake of brevity, we use the [*reconfig* | *drop* |

delay | *batch*] notation introduced in [14]. The *reconfig* field describes the details of the reconfiguration cost. In this paper, there is only one possible value for this field, a fixed reconfiguration cost denoted Δ. The *drop* field describes the details of the drop cost. In this paper, there is only one possible value for this field, a unit drop cost denoted 1. The *delay* field contains the details of the delay bound. In this paper, there is only one possible value for this field, per-color delay bounds denoted D_ℓ. The *batch* field constrains that the requests of color ℓ can only arrive at integral multiples of the specified value. In this paper, the possible values for this field are 1 and D_ℓ.

With this notation, our main problem is denoted $[\Delta \mid 1 \mid D_\ell \mid 1]$. The special case in which jobs of color ℓ arrive at integral multiples of D_ℓ is denoted $[\Delta \mid 1 \mid D_\ell \mid D_\ell]$. We use the terminology "rate-limited $[\Delta \mid 1 \mid D_\ell \mid D_\ell]$" to denote the special case of $[\Delta \mid 1 \mid D_\ell \mid D_\ell]$ in which at most D_ℓ color ℓ jobs arrive at each integral multiple of D_ℓ. In this paper, we assume Δ is a positive integer (it is not hard to generalize our results to an arbitrary Δ).

Roadmap. The rest of the paper is organized as follows. Section 3 solves rate-limited $[\Delta \mid 1 \mid D_\ell \mid D_\ell]$, where each D_ℓ is a power of 2. Section 4 solves $[\Delta \mid 1 \mid D_\ell \mid D_\ell]$, where each D_ℓ is a power of 2, by a reduction to rate-limited $[\Delta \mid 1 \mid D_\ell \mid D_\ell]$. Section 5 solves our main problem $[\Delta \mid 1 \mid D_\ell \mid 1]$ by a reduction to $[\Delta \mid 1 \mid D_\ell \mid D_\ell]$.

3. Rate-Limited Batched Arrivals

In this section, we solve rate-limited $[\Delta \mid 1 \mid D_\ell \mid D_\ell]$, where each D_ℓ is a power of 2. This problem is characterized by a fixed reconfiguration cost Δ, a unit drop cost, per-color delay bounds D_ℓ, batched arrivals (jobs of color ℓ arrive at integral multiples of D_ℓ), and rate-limited input (at most D_ℓ jobs of color ℓ arrive at each integral multiple of D_ℓ). As mentioned in Section 1, this problem is a key building block to solve our main problem $[\Delta \mid 1 \mid D_\ell \mid 1]$.

In this section, we introduce three online algorithms: *EDF*, *ΔLRU*, and *ΔLRU-EDF*. In Section 3.1, we first present the common aspects of the three algorithms. For instance, due to the difference between the reconfiguration and drop costs, we do not configure a color until it has enough job arrivals.

Algorithm *EDF* is based on the EDF scheduling principle. The main idea is that, among the colors with enough job arrivals, we configure the colors with the earliest deadlines and with jobs to execute. Algorithm *EDF* addresses the urgency aspect of the request sequence. However, since it favors colors that have jobs to execute, *EDF* suffers from thrashing. See Section 3.2 for a detailed discussion of *EDF*.

Algorithm *ΔLRU* is based on the LRU scheduling principle. The main idea is that, among the colors with enough job arrivals, we configure the colors with the most recent timestamps. (For the formal definition of the timestamp of a color, see Section 3.3.) Algorithm *ΔLRU* addresses the recency aspect of the request sequence. However, since it does not consider whether colors have jobs to execute or not, *ΔLRU* suffers from underutilization. See Section 3.3 for a detailed discussion of *ΔLRU*.

Algorithm *ΔLRU-EDF* is a combination of *EDF* and *ΔLRU*. The *EDF* component ensures that the resources are well utilized. The *ΔLRU* component reduces thrashing by allowing colors with recent timestamps to remain configured. See Section 3.4 for a detailed discussion of *ΔLRU-EDF*, and Section 3.5 for the proof that shows *ΔLRU-EDF* is resource competitive.

3.1. Common Aspects

For convenience of presentation, we consider the set of resources as a cache, where resource k is viewed as location k. We view reconfiguring resource k with color ℓ as caching color ℓ at location k. We use a counting scheme to ensure that only colors with a sufficient number of job arrivals can be brought into the cache.

In the following, we formally present the common aspects of the three algorithms. Given an instance (σ, m) of rate-limited $[\Delta \mid 1 \mid D_\ell \mid D_\ell]$, we allow the online algorithms to use n resources, where $n > m$. Each color is either *eligible* or *ineligible*. Only eligible colors can be brought into the cache. For each color, we maintain a counter and a deadline. Initially, the cache is empty, all colors are ineligible, and the counter and deadline associated with any color are zero. In each round j, the actions performed in the four phases are described as follows.

Arrival phase We receive a request. For any color ℓ, if j is an integral multiple of D_ℓ, we perform the following steps.

1. We increase the counter of ℓ by the number of color ℓ jobs received in this phase.
2. If the counter of ℓ is at least Δ, we set ℓ to eligible and reset the counter of ℓ.
3. We set the deadline of ℓ to $j + D_\ell - 1$.

Reconfiguration phase We update the contents of the cache; the method used depends on the algorithm, see Sections 3.2 through 3.4.

Execution phase For any color ℓ, each resource configured with color ℓ executes one pending job of color ℓ.

Drop phase For any color ℓ, if $j \bmod D_\ell$ is $D_\ell - 1$, we perform the following steps.

1. We drop all pending jobs of color ℓ.
2. If color ℓ is eligible and not in the cache, we set color ℓ to ineligible.

3.2. *EDF*

We say a color ℓ is *idle* if there are no pending jobs of color ℓ, and *nonidle* otherwise. We rank nonidle colors ahead of idle colors. The rank of idle colors is arbitrary. We rank nonidle colors in ascending order of deadlines. Ties are broken according to ascending order of delay bounds. Further ties are broken according to a fixed order of colors. We update the cache as follows. If a nonidle eligible color ℓ in the top n positions of the ranking is not in the cache, we bring ℓ into the cache, evicting the color with the lowest rank if there the cache is full.

Consider a color ℓ with a short delay bound that receives a small number of jobs every D_ℓ rounds. The priority of ℓ changes from high to low, and then low to high, from time to time, which may lead to thrashing. We refer the reader to Appendix A for an example establishing that *EDF* is not resource competitive.

3.3. *ΔLRU*

For each color ℓ, we maintain a *timestamp* as follows. Initially, the timestamp of ℓ is zero. In the arrival phase of any round j, if the counter of ℓ is reset, we set the timestamp of ℓ to j immediately after the counter is reset. In each reconfiguration phase, we cache the n eligible colors with the most recent timestamps, breaking ties as in EDF.

Due to the difference between the reconfiguration and drop costs, we require at least Δ job of color ℓ to arrive in order to update the timestamp of ℓ. Algorithm *ΔLRU* favors idle colors with recent timestamps over nonidle colors that do not have recent timestamps, which may result in low utilization. We refer the reader to Appendix B for an example establishing that *ΔLRU* is not resource competitive.

3.4. *ΔLRU-EDF*

In this section, we formally define algorithm *ΔLRU-EDF*. We give *ΔLRU-EDF* a factor of 8 resource advantage over an optimal feasible offline algorithm, that is, $n = 8m$. We use the first half of the cache capacity to keep distinct colors and the remaining half to replicate the cache contents of the first half. We use the replication to give half of the resources a factor of 2 speedup. Below we describe how we update the first half of the cache.

Let X be the $\frac{n}{4}$ eligible colors with the most recent timestamps, where ties are broken as in *ΔLRU*. We rank eligible colors not in X as in *EDF* (see Section 3.2 for details). Let Y be the set of nonidle eligible colors in the top $\frac{n}{4}$ positions of the ranking. For any color ℓ that is in $X \cup Y$ but not in the cache, we bring ℓ into the cache, replacing an arbitrary color ℓ' that is in the cache but not in $X \cup Y$, if necessary. Since $|X \cup Y| \leq \frac{n}{2}$, such a color ℓ' is guaranteed to exist if the first half of the cache is full.

3.5. Analysis of *ΔLRU-EDF*

In this section, we show that *ΔLRU-EDF* is resource competitive. The analysis is organized as follows. First, Lemmas 3.1 through 3.4 argue that, on any instance such that each color appearing in the request sequence has at least Δ jobs, the cost incurred by *ΔLRU-EDF* is within a constant factor of that incurred by an optimal feasible offline algorithm. For convenience of analysis, we partition the drop costs incurred by *ΔLRU-EDF* into "eligible" and "ineligible" drop costs (the formal definitions are provided later in this section). Lemma 3.1 bounds the eligible drop cost incurred by *ΔLRU-EDF*. Our proof of Lemma 3.1 uses the EDF properties of *ΔLRU-EDF*, and three intermediate algorithms: "parallel" *EDF*, denoted *Par-EDF*, "sequential" *EDF*, denoted *Seq-EDF*, and "double-speed" *Seq-EDF*, denoted *2X-Seq-EDF*. (See the proof of Lemma 3.1 for the formal definitions of the three algorithms.)

To bound the other costs incurred by *ΔLRU-EDF*, for each color ℓ, we partition the sequence of rounds into subsequences, denoted "ℓ-epochs" (the formal definition is given later in this section). Lemma 3.2 gives an upper bound on the ineligible drop cost incurred by *ΔLRU-EDF*, in terms of the total number of epochs, over all colors. The proof of Lemma 3.2 is straightforward. For any problem instance such that each color appearing in the request sequence has at least Δ jobs, Lemma 3.3 upper bounds the reconfiguration cost incurred by *ΔLRU-EDF*, and Lemma 3.4 lower bounds the total cost incurred by an optimal feasible offline algorithm, in terms of the total number of epochs. Our proofs of Lemmas 3.3 and 3.4 make use of amortized analysis; our proof of Lemma 3.4 relies on the LRU properties of *ΔLRU-EDF*.

Second, Theorem 1 establishes the resource competitiveness of *ΔLRU-EDF* by a reduction to a problem instance in which each color appearing in the request sequence has at least Δ jobs, and by using Lemmas 3.1 through 3.4.

Now we give the formal definitions for the analysis. Let (σ, m) be any instance of rate-limited $[\Delta \mid 1 \mid D_\ell \mid D_\ell]$. Let A be any algorithm. Let *OFF* be an optimal feasible offline algorithm for (σ, m). Let *Cost*(A, σ, m) (resp., *ReconfigCost*(A, σ, m), *DropCost*(A, σ, m)) denote the cost (resp., reconfiguration cost, drop cost) incurred by A on (σ, m). A job x of color ℓ is considered to be *ineligible* (resp., *eligible*) if color ℓ is ineligible (resp., eligible) at the end of the arrival phase in which x arrives. We define the ineligible (resp., eligible) drop cost incurred by *ΔLRU-EDF*, denoted *IneligibleDropCost*(*ΔLRU-EDF*$, \sigma, m)$ (resp., *EligibleDropCost*(*ΔLRU-EDF*$, \sigma, m)$), to be the drop cost incurred by *ΔLRU-EDF* on ineligible (resp., eligible) jobs in σ.

For each color ℓ, we partition the sequence of rounds into ℓ-epochs as follows. We define ℓ-epoch 0 to start with round 0 and end with the first round in which ℓ becomes ineligible. For every $i \geq 1$, ℓ-epoch i starts when ℓ-epoch $i-1$ ends, and ends with the first round following ℓ-epoch $i-1$ in which ℓ becomes ineligible. For convenience, we use the term *epoch* to refer to an ℓ-epoch, for some ℓ. We use *numEpochs*(σ) to denote the total number of epochs associated with σ.

Lemma 3.1 *For any instance (σ, m) of rate-limited $[\Delta \mid 1 \mid D_\ell \mid D_\ell]$, EligibleDropCost$(\Delta$LRU-EDF$, \sigma, m)$ is at most DropCost$($OFF$, \sigma, m)$.*

Proof sketch. To show the lemma, we find it convenient to define the following three algorithms: *Par-EDF*, *Seq-EDF*, and *2X-Seq-EDF*. Each of the three algorithms is allowed to use m resources. Algorithm *Par-EDF* is defined as follows. In each reconfiguration phase, we reconfigure the resources in such a way that we can execute m pending jobs with the best ranks in the immediately following execution phase, where jobs are ranked in ascending order of deadlines, and ties are broken as in *EDF*. Algorithm *Seq-EDF* is defined as follows. In each reconfiguration phase, we configure m nonidle colors with the best ranks, where colors are ranked as in *EDF*. We define a *double-speed* schedule to be a schedule in which the reconfiguration and execution phases are performed twice in each round. We use *2X-Seq-EDF* to denote double-speed *Seq-EDF*. Note that the three algorithms defined in this paragraph do not require a color to be eligible to in order to be configured on the resources.

By a standard EDF-type swapping argument, one can easily show the following inequality.

$$\textit{DropCost}(\textit{Par-EDF}, \sigma, m) \leq \textit{DropCost}(\textit{OFF}, \sigma, m) \quad (1)$$

It is more challenging to show the follow two inequalities, which are needed to obtain the lemma.

$$\begin{aligned} & \textit{DropCost}(\textit{2X-Seq-EDF}, \sigma, m) \\ \leq \ & \textit{DropCost}(\textit{Par-EDF}, \sigma, m) \end{aligned} \quad (2)$$

$$\begin{aligned} & \textit{EligibleDropCost}(\Delta\textit{LRU-EDF}, \sigma, m) \\ \leq \ & \textit{DropCost}(\textit{2X-Seq-EDF}, \sigma, m) \end{aligned} \quad (3)$$

We omit the proof for Inequalities (2) and (3) due to space limitations. The lemma follows from Inequalities (1) through (3). ∎

Lemma 3.2 *For any instance (σ, m) of rate-limited $[\Delta \mid 1 \mid D_\ell \mid D_\ell]$, IneligibleDropCost$(\Delta$LRU-EDF$, \sigma, m) <$ numEpochs$(\sigma) \cdot \Delta$.*

Proof. Consider any color ℓ. Let h be any ℓ-epoch. Let C be the ineligible drop cost incurred by *ΔLRU-EDF* on color ℓ jobs in h. It is sufficient to show that C is less than Δ.

Let h' be the longest prefix of h throughout which ℓ is ineligible. Let C' be the drop cost incurred by *ΔLRU-EDF* on color ℓ jobs in h'. Since ℓ does not become eligible in h', the number of color ℓ jobs that arrive in h' is less than Δ. Hence, $C' < \Delta$. By the definition of an epoch, once ℓ becomes eligible in h, it remains eligible until h ends. By the definition of ineligible jobs and ineligible drop cost, $C = C'$. Therefore, $C < \Delta$. ∎

Lemma 3.3 *For any instance (σ, m) of rate-limited $[\Delta \mid 1 \mid D_\ell \mid D_\ell]$ such that each color appearing in σ has at least Δ jobs, ReconfigCost$(\Delta$LRU-EDF$, \sigma, m) \leq O($Cost$($OFF$, \sigma, m) +$ numEpochs$(\sigma) \cdot \Delta)$.*

Proof sketch. In order to establish this result, it is useful to label each eviction as either an "LRU eviction" or an "EDF eviction" in our analysis of *ΔLRU-EDF*. We say that an LRU eviction occurs whenever a color is evicted in a given round and that color was kept by the LRU principle in the preceding round. All other evictions are EDF evictions.

We proceed in three stages. In the first stage, we are able to show the following claim. For any instance (σ, m) of rate-limited $[\Delta \mid 1 \mid D_\ell \mid D_\ell]$ such that each color appearing in σ has at least Δ jobs, the total number of LRU evictions times Δ is $O(\textit{Cost}(\textit{OFF}, \sigma, m))$.

In the second stage, we are able to show the following claim. For any color ℓ, any ℓ-epoch h, and any two rounds i and j in h such that $i < j$ and *ΔLRU-EDF* brings ℓ into the cache in round i and j, the following conditions hold in round j: (1) color ℓ is brought into the cache by the EDF principle, and (2) if bringing ℓ into the cache results in an EDF eviction, then the evicted color is idle. Due to space limitations, we omit the proofs of the claims associated with the first two stages.

In the third stage, we prove the lemma using the above claims and amortized analysis as follows. We associate 4Δ units of credit with each epoch: 2Δ units of "first-time" credit and 2Δ units of "end-of-epoch" credit. We also associate 2Δ units of credit with each LRU eviction. From the claim of the first stage, the total credit is $O(\textit{Cost}(\textit{OFF}, \sigma, m) + \textit{numEpochs}(\sigma) \cdot \Delta)$. It is sufficient to show that the total reconfiguration cost incurred by *ΔLRU-EDF* can be paid for by the total credit.

Consider any color ℓ and any ℓ-epoch h. If *ΔLRU-EDF* does not bring ℓ into the cache in h, then it does not incur any reconfiguration cost in h. Otherwise, let rounds $i_0 < \cdots < i_k$ be the rounds in h in which *ΔLRU-EDF* brings ℓ into the cache. For every j such that $0 \leq j \leq k$, let R_j be the reconfiguration operation performed by *ΔLRU-EDF* to bring in ℓ in round i_j. Since each cached color is replicated in *ΔLRU-EDF*, the cost of operation R_j is 2Δ. We use the 2Δ units of "first-time" credit associated with h to pay for operation R_0. In the following, we show that the remaining R_j's can also be paid for.

Fix j arbitrarily, where $0 < j \leq k$. It is not hard to see that, when color ℓ is brought into the cache in round i_j, some color ℓ' is evicted. If the eviction of color ℓ' is an LRU eviction, operation R_j can be paid for by the 2Δ units of credit associated with the LRU eviction. If the eviction of color ℓ' is an EDF eviction, then the claim of the second stage implies that color ℓ' is evicted idle in round i_j. Since jobs of color ℓ' arrive only at integral multiples of $D_{\ell'}$, ℓ' remains idle until the next integral multiple of $D_{\ell'}$, at which point ℓ' becomes ineligible and its current ℓ-epoch h' ends. Hence, we can use the "end-of-epoch" credit associated with h' to pay for operation R_j. It is not difficult to argue that each unit of credit is used at most once. This completes the proof. ■

Lemma 3.4 *For any instance (σ, m) of rate-limited $[\Delta \mid 1 \mid D_\ell \mid D_\ell]$ such that each color appearing in σ has at least Δ jobs, $Cost(OFF, \sigma, m) = \Omega(numEpochs(\sigma) \cdot \Delta)$.*

Proof sketch. To get a lower bound on the cost of *OFF*, we find it convenient to partition the sequence of rounds into *super-epoch*s. Super-epoch 0 is the minimum sequence of rounds, beginning with round 0, during which at least $2m$ colors have their counters reset. For every $i \geq 1$, super-epoch i is the minimum sequence of rounds following super-epoch $i-1$ during which at least $2m$ colors have their counters reset. Note that the last super-epoch may be incomplete. We say that a color ℓ is *active* in super-epoch i if the counter of ℓ is reset in super-epoch i. We partition the epochs into two sets: *special* epochs, the epochs that are not active in any complete super-epoch, and *regular* epochs, the epochs that are not special. We handle special and regular epochs separately. For special epochs, we show that, $Cost(OFF, \sigma, m)$ is $\Omega(\Delta)$ times the number of special epochs. For regular epochs, we define the amortized cost of *OFF* in such a way that the total amortized cost of *OFF* is within a constant factor of the actual cost of *OFF*, and show that the total amortized cost of *OFF* is $\Omega(\Delta)$ times the number of regular epochs. ■

Theorem 1 *Algorithm ΔLRU-EDF is resource competitive for rate-limited $[\Delta \mid 1 \mid D_\ell \mid D_\ell]$, where each D_ℓ is a power of 2.*

Proof. Let (σ, m) be an arbitrary instance of rate-limited $[\Delta \mid 1 \mid D_\ell \mid D_\ell]$. We say a color ℓ is *heavy* (resp., *light*) if there are at least (resp., less than) Δ jobs of color ℓ in σ. Any job of a heavy (resp., light) color is a heavy (resp., light) job. We break each request into two requests, one consisting of the light jobs and the other consisting of the heavy jobs. Let α (resp., β) denote the resulting sequence of requests involving heavy (resp., light) jobs.

Since there are less than Δ jobs of any light color, *OFF*, as an optimal feasible offline algorithm, drops all light jobs. Hence, $Cost(OFF, \sigma)$ equals $Cost(OFF, \alpha)$ plus the total number of light jobs. Since there are are less than Δ jobs of any light color, no light color ever becomes eligible. Thus, *ΔLRU-EDF* never caches a light color, and drops all light jobs. Hence, $Cost(\Delta LRU\text{-}EDF, \sigma, m)$ equals $Cost(\Delta LRU\text{-}EDF, \alpha, m)$ plus the total number of light jobs. From Lemmas 3.1 through 3.4, $Cost(\Delta LRU\text{-}EDF, \alpha, m) = O(Cost(OFF, \alpha, m))$. Hence, the lemma follows. ■

4. Batched Arrivals

In this section, we solve $[\Delta \mid 1 \mid D_\ell \mid D_\ell]$, where each D_ℓ is a power of 2. This problem is characterized by a fixed reconfiguration cost Δ, a unit drop cost, per-color delay bounds D_ℓ, and batched arrivals (jobs of color ℓ arrive at integral multiples of D_ℓ).

As mentioned in Section 1, $[\Delta \mid 1 \mid D_\ell \mid D_\ell]$ is a building block to solve our main problem $[\Delta \mid 1 \mid D_\ell \mid 1]$. To solve $[\Delta \mid 1 \mid D_\ell \mid D_\ell]$, we use a reduction to rate-limited $[\Delta \mid 1 \mid D_\ell \mid D_\ell]$, which is solved in Section 3. Sections 4.1 and 4.2 give the reduction algorithm and analysis, respectively.

4.1. Algorithm

Given any instance (σ, m) of $[\Delta \mid 1 \mid D_\ell \mid D_\ell]$, where D_ℓ is a power of 2, algorithm *Recolor* proceeds in the following three steps. In the first step, we construct a request sequence σ' for rate-limited $[\Delta \mid 1 \mid D_\ell \mid D_\ell]$ as follows. Let σ_i be request i of σ, where $0 \leq i < |\sigma|$. For any color ℓ, we rank color ℓ jobs in σ_i in an arbitrary order. For any color ℓ and color ℓ job x in σ_i, we construct a job y that is the same as x except that the color of y is given by the pair (ℓ, j), where $j = \left\lfloor \frac{rank(x)}{D_\ell} \right\rfloor$, and $rank(x)$ is the rank of x in σ_i. Let σ'_i be the union of all such y's that are constructed over all colors ℓ. We obtain σ' by concatenating σ'_i's in increasing order of i.

In the second step, we use algorithm *ΔLRU-EDF* on $(\sigma', 3m)$ to obtain a schedule S' for σ'. In the third step, from S' we construct a schedule S for σ as follows. For any color ℓ, any integers j and k, whenever S' configures color (ℓ, j) on resource k, S configures color ℓ on resource k; whenever S' executes a job of color (ℓ, j) on resource k, S executes a job of color ℓ on resource k. Note that *Recolor* is an online algorithm.

4.2. Analysis

In this section, we show that algorithm *Recolor* is resource competitive. The request sequences σ and σ', and the schedules S and S' mentioned in the lemma statements and proofs below are defined in Section 4.1.

Lemma 4.1 *If there exists a schedule T for σ that uses m resources and incurs cost C, then there exists a schedule T' for σ' that uses $3m$ resources and incurs cost $O(C)$.*

The main proof idea of Lemma 4.1 is to construct T' by rearranging and recoloring the jobs executed in T. Due to space limitations, we omit the proof of Lemma 4.1.

Lemma 4.2 *The cost of S is at most that of S'.*

Proof. Since the schedule S replaces color (ℓ, j) with color ℓ, the reconfiguration cost incurred by S is at most that incurred by S'. From the way we construct σ' and S', it is not hard to see that the number of color ℓ jobs executed by S is equal to the total number of color (ℓ, j) jobs executed by S', over all j. Since the number of color ℓ jobs associated with σ is equal to the total number of color (ℓ, j) jobs associated with σ', over all j, the drop cost incurred by S is equal to that incurred by S'. ∎

Theorem 2 *Algorithm Recolor is resource competitive for $[\Delta \mid 1 \mid D_\ell \mid D_\ell]$, where each D_ℓ is a power of 2.*

Proof. Let T be the schedule produced by an arbitrary feasible offline algorithm on (σ, m). By the definition of a feasible algorithm, T uses m resources. Let C be the cost of T. By Lemma 4.1, there exists a schedule T' for σ' that uses $3m$ resources and incurs cost $O(C)$. By Theorem 1, the schedule S' for σ' generated by algorithm *ΔLRU-EDF* uses $O(m)$ resources and incurs cost $O(C)$. By construction, S uses the same number of resources as S'. By Lemma 4.2, the cost of S is at most that of S'. Hence, S uses $O(m)$ resources and incurs $O(C)$ cost. Since S is a schedule for σ, the theorem follows. ∎

5. Main Result

In this section, we solve our main problem $[\Delta \mid 1 \mid D_\ell \mid 1]$, which is characterized by a fixed reconfiguration cost Δ, a unit drop cost, per-color delay bounds D_ℓ, and nonbatched arrivals (requests can arrive at any round).

To simplify the presentation, we focus on the special case where each D_ℓ is a power of 2. The special case is solved by a reduction to $[\Delta \mid 1 \mid D_\ell \mid D_\ell]$, which is solved in Section 4. For any color ℓ such that D_ℓ is equal to 1, jobs of color ℓ are already batched. For convenience, we focus on the case where D_ℓ is greater than 1, for all colors ℓ. Sections 5.1 and 5.2 give the algorithm and analysis for the reduction, respectively. Section 5.3 comments on how to extend our solution to arbitrary delay bounds, that is, the delay bounds are not necessarily powers of 2.

5.1. Algorithm

For any delay bound p and any nonnegative integer i, we define *halfBlock*(p, i) to be the $\frac{p}{2}$ rounds starting from round $i \cdot \frac{p}{2}$. Let σ be an arbitrary request sequence for $[\Delta \mid 1 \mid D_\ell \mid 1]$. We define the *batched version* of σ, denoted σ', as follows. We obtain σ' by moving the arrival of any job x of color ℓ that arrives in *halfBlock*(D_ℓ, i) in σ to the beginning of *halfBlock*$(D_\ell, i+1)$, and changing the delay bound of x to $\frac{D_\ell}{2}$. Thus, the request sequence σ' can be viewed as a request sequence for $[\Delta \mid 1 \mid \frac{D_\ell}{2} \mid \frac{D_\ell}{2}]$.

Algorithm *VarBatch* proceeds in the following three steps. First, given an arbitrary instance (σ, m) of $[\Delta \mid 1 \mid D_\ell \mid 1]$, we construct an instance $(\sigma', 7m)$ of $[\Delta \mid 1 \mid \frac{D_\ell}{2} \mid \frac{D_\ell}{2}]$, where σ' is the batched version of σ. Second, we apply algorithm *Recolor* (defined in Section 4.1) on $(\sigma', 7m)$ to obtain a schedule S' for σ'. Finally, we obtain a schedule S for σ from S'. The schedule S is the same as S' except the request sequence associated with S is σ. Note that algorithm *VarBatch* is an online algorithm.

5.2. Analysis

In this section, we show that algorithm *VarBatch* is resource competitive. The request sequences σ and σ', and the schedules S and S' mentioned in the lemma statements and proofs below are defined in Section 5.1.

Lemma 5.1 *If there exists a schedule T for σ that uses m resources and incurs cost C, then there exists a schedule T' for σ' that uses $7m$ resources and incurs cost $O(C)$.*

Proof sketch. For any color ℓ job x that arrives in *halfBlock*(D_ℓ, i) in σ, we say the execution of x in T is *early* (resp., *punctual*, *late*) if x is executed in *halfBlock*(D_ℓ, i) (resp., *halfBlock*$(D_\ell, i+1)$, *halfBlock*$(D_\ell, i+2)$) in T. We prove the lemma in two stages as follows. First, we construct a schedule T' for σ' that uses $7m$ resources by rearranging the job executions in T. The main idea is to use extra resources to move the early executions in T forward, and the late executions in T backward. Second, we use amortized analysis to show that the cost of T' is $O(C)$. In the following, we give more details about these two stages.

We first describe how to construct T'. For convenience, we number the resources from 0. Consider any integer k such that $0 \le k < m$. Let W_k be the set of jobs that are executed on resource k in T. Let X_k, Y_k, and Z_k be the jobs in W_k such that the corresponding executions are early, punctual, and late in T, respectively. All the jobs in X_k (resp., Y_k, Z_k) are rearranged to execute in T' on resources $7k$ through $7k+2$ (resp., $7k+3$, $7k+4$ through $7k+6$). The jobs in Y_k are arranged to execute in the same round as in T. Below we describe how to rearrange the jobs in X_k. The jobs in Z_k can be rearranged in a similar manner.

A job x in X_k is defined to be k-*special* if, in schedule T, the color of x, call it ℓ, is configured on resource k throughout *halfBlock*(D_ℓ, i) and *halfBlock*$(D_\ell, i+1)$, and x is executed on resource k in *halfBlock*(D_ℓ, i). Any job in X_k that is not k-special is said to be k-*regular*. We use resource $7k$ to execute k-special jobs as follows. For any color ℓ and any k-special job x of color ℓ that is executed in round j in T, we execute x in round $j + \frac{D_\ell}{2}$. We use resources $7k+1$ and $7k+2$ to execute k-regular jobs. To avoid collisions (i.e., different jobs executed on the same resource and in the same round), we proceed in the follow manner. We rearrange k-regular jobs in increasing order of delay bounds. For any delay bound p and any nonnegative integer i, let R be the set of k-regular jobs with delay bound p that are executed on resource k in *halfBlock*(p, i) in T. For any color ℓ with delay bound p, let R_ℓ be the color ℓ jobs in R. To rearrange the jobs in R, we iteratively consider each color ℓ with delay bound p (in arbitrary order), and rearrange the jobs in R_ℓ. In the remaining of the paragraph we describe how to rearrange the jobs in R_ℓ. We define a *slot* to be a round on a resource. We say a slot is *free* if no job is assigned to execute in the slot. We order the slots in increasing order of resource indices, breaking ties by increasing round indices. We arrange the jobs in R_ℓ in the first $|R_\ell|$ free slots in *halfBlock*$(p, i+1)$ on resources $7k+1$ and $7k+2$.

It is not hard to see that T' is a schedule for σ', and that all jobs executed by T are executed by T'. It remains to show that the reconfiguration cost incurred by T' is $O(C)$. Fix k arbitrarily, where $0 \le k < m$. Let C_k be the reconfiguration cost incurred on resource k in T. It is sufficient to show that the reconfiguration cost in T' associated with the jobs in W_k (i.e., the jobs executed on resources $7k$ through $7k+6$) is $O(C_k)$. It is straightforward to show that the reconfiguration cost in T' associated with the jobs in Y_k (i.e., the jobs executed on resource $7k+3$) is at most C_k. Below we argue that the reconfiguration cost in T' associated with the jobs in X_k (i.e., the jobs executed on resources $7k$ through $7k+2$) is $O(C_k)$. Similarly, one can argue that the reconfiguration cost in T' associated with the jobs in Z_k (i.e., the jobs executed on resources $7k+4$ through $7k+6$) is $O(C_k)$. It is straightforward to show that the reconfiguration cost in T' associated with the k-special jobs in X_k is at most C_k. Hence, it is sufficient to account for the reconfiguration cost in T' associated with k-regular jobs in X_k. To do this, we associate $O(\Delta)$ units of credit with each reconfiguration on resource k in T, and show that the total reconfiguration cost incurred by the k-regular jobs can be paid for by the credit. ∎

Theorem 3 *Algorithm VarBatch is resource competitive for* $[\Delta \mid 1 \mid D_\ell \mid 1]$, *where each* D_ℓ *is a power of* 2.

Proof. Let T be the schedule produced by an arbitrary feasible offline algorithm on (σ, m). By the definition of a feasible algorithm, T uses m resources. Let C be the cost of T. By Lemma 5.1, there exists an offline schedule T' for σ' that uses $7m$ resources and incurs cost $O(C)$.

Since σ' can be viewed as a request sequence for $[\Delta \mid 1 \mid \frac{D_\ell}{2} \mid \frac{D_\ell}{2}]$, and by Theorem 2, algorithm *Recolor* is resource competitive for $[\Delta \mid 1 \mid \frac{D_\ell}{2} \mid \frac{D_\ell}{2}]$, S' uses $O(m)$ resources and incurs cost $O(C)$. By definition, S is a schedule for σ, uses $O(m)$ resources, and incurs cost $O(C)$. ∎

5.3. Extension to Arbitrary Delay Bounds

The extension of our solution to arbitrary delay bounds is straightforward. The basic idea is as follows: for any delay bound p such that $2^j \le p < 2^{j+1}$, and any job x with delay bound p that arrives in *halfBlock*$(2^j, i)$, we delay the arrival of x to the beginning of *halfBlock*$(2^j, i+1)$, and change the delay bound of x to 2^{j-1}. The proof that the extended solution is resource competitive is similar to the proof given in Section 5.2.

References

[1] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, Cambridge, 1998.

[2] P. Brucker. *Scheduling Algorithms*. Springer-Verlag, Berlin, 2001.

[3] P. Brucker, M. Y. Kovalyov, Y. M. Shafransky, and F. Werner. Batch scheduling with deadlines on parallel machines. *Annals of Operations Research*, 83:23–40, 1998.

[4] A. Chandra, W. Gong, and P. Shenoy. Dynamic resource allocation for shared data centers using online measurements. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 300–301, June 2003.

[5] J. S. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle. Managing energy and server resources in hosting centers. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, pages 103–116, October 2001.

[6] M. Dertouzos. Control robotics: The procedural control of physical processors. In *Proceedings of the IFIP Congress*, pages 807–813, 1974.

[7] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47:617–643, 2000.

[8] R. Kokku. *ShaRE: Run-time System for High-performance Virtualized Routers*. PhD thesis, Department of Computer Science, University of Texas at Austin, August 2005.

[9] R. Kokku, T. Riché, A. Kunze, J. Mudigonda, J. Jason, and H. Vin. A case for run-time adaptation in packet processing systems. *ACM SIGCOMM Computer Communication Review*, 34:107–112, 2004.

[10] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20:46–61, 1973.
[11] N. Megiddo and D. S. Modha. Arc: A self-tuning, low overhead replacement cache. In *Proceedings of the 2nd USENIX Conference on File and Storage Technologies*, pages 115–130, 2003.
[12] E. J. O'Neil, P. E. O'Neil, and G. Weikum. The LRU-K page replacement algorithm for database disk buffering. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 297–306, May 1993.
[13] C. A. Phillips, C. Stein, E. Torng, and J. Wein. Optimal time-critical scheduling via resource augmentation. *Algorithmica*, pages 163–200, 2002.
[14] C. G. Plaxton, Y. Sun, M. Tiwari, and H. Vin. Reconfigurable resource scheduling. In *Proceedings of 18th ACM Symposium on Parallelism in Algorithms and Architectures*, July 2006.
[15] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985.
[16] T. Spalink, S. Karlin, L. L. Peterson, and Y. Gottlieb. Building a robust software-based router using network processors. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, pages 216–229, October 2001.
[17] A. Srinivasan, P. Holman, J. Anderson, S. K. Baruah, and J. Kaur. Multiprocessor scheduling in processor-based router platforms: Issues and ideas. In *Proceedings of the 2nd Workshop on Network Processors*, February 2003.
[18] H. Vin, J. Mudigonda, J. Jason, E. J. Johnson, R. Ju, A. Kunze, and R. Lian. A programming environment for packet-processing systems: Design considerations. In *Proceedings of the 3rd Workshop on Network Processors and Applications*, February 2004.
[19] N. E. Young. On-line file caching. In *Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms*, pages 82–86, January 1998.

A. Analysis of *EDF*

In this section, we show that *EDF* is not constant competitive, even if *EDF* is given an arbitrary constant factor resource advantage, and an arbitrary constant replication factor r, that is, each color in the cache is replicated in r locations.

Consider an arbitrary instance (σ, m) of rate-limited $[\Delta \mid 1 \mid D_\ell \mid D_\ell]$. Let *OFF* denote an arbitrary feasible offline algorithm. We assume that, n, the number of resources that *EDF* can use, is equal to rsm, where r is the replication factor, and s is an arbitrary positive constant. We consider $(s+1)m$ colors as follows: m colors with a delay bound 2^j, m colors with a delay bound 2^k, m colors with a delay bound 2^{k+1}, ..., and m colors with a delay bound 2^{k+s-1}, where $2^k > 2^j > \Delta$. For convenience, we refer to each color with a delay bound 2^j as a short-term color and any other color as a long-term color. The request sequence proceeds in 2^{k+s-1} rounds as follows. For each short-term color, we receive Δ jobs at each integral multiple of 2^j, in rounds 0 through $2^{k-1}-1$. For each long-term color with a delay bound of 2^{k+i}, for $0 \le i < s$, we receive 2^{k+i-1} jobs at the very beginning.

Consider rounds 0 through $\frac{2^{k-1}}{r}$. Each long-term color always has jobs to execute. Since $2^j > \Delta$, each short-term color is brought into the cache and then evicted $\frac{2^{k-1}}{2^j r}$ times. Hence, the reconfiguration cost incurred by *EDF* is $\Omega(2^{k-j} m\Delta)$.

Suppose that *OFF* caches the short-term colors in rounds 0 through $2^{k-1}-1$, and caches the colors with a delay bound 2^{k+i} in rounds 2^{k+i-1} through $2^{k+i}-1$, where $0 \le i < s$.

Algorithm *OFF* does not incur any drop cost and incurs a reconfiguration cost of $O(m\Delta)$. Hence the competitive ratio of *EDF* is $\Omega(2^{k-j})$, which can be arbitrarily large by setting j and k appropriately.

B. Analysis of *ΔLRU*

In this section, we show that *ΔLRU* is not constant competitive, even if *ΔLRU* is given an arbitrary constant factor resource advantage, and an arbitrary constant replication factor r, that is, each color in the cache is replicated in r locations.

Consider an arbitrary instance (σ, m) of rate-limited $[\Delta \mid 1 \mid D_\ell \mid D_\ell]$. Let *OFF* denote an arbitrary feasible offline algorithm. We assume that n, the number of resources that *ΔLRU* can use, is equal to rsm, where r is the replication factor, and s is an arbitrary positive constant. Consider sm colors with a delay bound 2^j and m colors with a delay bound 2^k, where $2^k > 2^j > \Delta$. For convenience, we refer to each color with a delay bound 2^j as a short-term color and each color with a delay bound 2^k as a long-term color. The request sequence proceeds in 2^k rounds as follows. We receive Δ jobs of each short-term color at each integral multiple of 2^j, and 2^k jobs of each long-term color at the very beginning.

It is not hard to verify that, from the reconfiguration phase of round 2^j, the timestamp of any short-term color is more recent than that of any long-term color. Hence, in the reconfiguration phase of round 2^j, *ΔLRU* caches all short-term colors, and evicts all long-term colors; from onwards, *ΔLRU* does not change the configuration. Thus, the drop cost incurred by *ΔLRU* is at least $(2^k - 2^j)m$. Since $k > j$, the cost incurred by *ΔLRU* is $\Omega(2^k m)$.

Suppose that *OFF* caches the long-term colors throughout. The reconfiguration cost incurred by *OFF* is $m\Delta$. The drop cost incurred by *OFF* is $2^{k-j} sm\Delta$. Hence the total cost incurred by *OFF* is $O(2^{k-j} m\Delta)$. Thus, the competitive ratio of *ΔLRU* is $\Omega(\frac{2^k m}{2^{k-j} m\Delta}) = \Omega(\frac{2^j}{\Delta})$, which can be arbitrarily large by setting j and k appropriately.