Online Algorithms to Minimize Resource Reallocations and Network Communication

Sashka Davis[1], Jeff Edmonds[2], and Russell Impagliazzo[1]

[1] Dept. of Computer Science, Univ. of California, San Diego
[2] Dept. of Computer Science, York University, Canada

Abstract. In this paper, we consider two new online optimization problems (each with several variants), present similar online algorithms for both, and show that one reduces to the other. Both problems involve a control trying to minimize the number of changes that need to be made in maintaining a state that satisfies each of many users' requirements. Our algorithms have the property that the control only needs to be informed of a change in users needs when the current state no longer satisfies the user. This is particularly important when the application is one of trying to minimize communication between the users and the control.

The Resource Allocation Problem (RAP) is an abstraction of scheduling malleable and evolving jobs on multiprocessor machines. A scheduler has a fixed pool of resources of total size T. There are n users, and each user j has a resource requirement for $r_{j,t}$ resources. The scheduler must allocate resources $\ell_{j,t}$ to user j at time t such that each allocation satisfies the requirement ($r_{j,t} \leq \ell_{j,t}$) and the combined allocations do not exceed T ($\sum_j \ell_{j,t} \leq T$). The objective is to minimize the total number of changes to allocated resources (the number of pairs j, t where $\ell_{j,t} \neq \ell_{j,t+1}$).

We consider online algorithms for RAP whose resource pool is increased to sT and obtain an online algorithm which is $O(\log_s n)$ competitive. Further we show that the increased resource pool is crucial to the performance of the algorithm by proving that there is no online algorithm using T resources which is $f(n)$-competitive for any $f(n)$. Note that our upper bounds all have the property that the algorithms only know the list of users whose requirements are currently unsatisfied and never learn the precise requirements of users. We feel this is important for many applications, since users rarely report underutilized resources as readily as they do unmet requirements. On the other hand, our lower bounds apply to online algorithms that have complete knowledge about past requirements.

The Transmission-Minimizing Approximate Value problem is a generalization of one defined in [OLW01], in which low-power sensors monitor real-time events in a distributed wireless network and report their results to a centralized cache. In order to minimize network traffic, the cache is allowed to maintain approximations to the values at the sensors, in the form of intervals containing the values, and to vary the lengths of intervals for the different sensors so that sensors with fluctuating values are measured less precisely than more stable ones. A constraint for the cache is that the sum of the lengths of the intervals must be within some precision parameter T. Similar models are described in [CYV04,cKA02].

We adapt the online randomized algorithm for the RAP problem to solve TMAV problem with similar competitive ratio: an algorithm can maintain sT precision and be $O(\log_s n)$-competitive in transmissions against an adversary maintaining precision T.
Further we show that solving TMAV is as hard as solving RAP, by reducing RAP to TMAV. This proves similar lower bounds for TMAV as we had for RAP, when s is near 1.

1 Introduction

Many applications have the following form: a central control is allocating resources among several users. Users have requirements and complain to the control when requirements are not met. The control may then reallocate resources to satisfy the complaints. However, such reallocation is expensive and should be minimized.

1.1 Resource Allocation Problems

For example, consider jobs competing for processor time on a parallel machine. In such time-shared multiprocessor machines, jobs with *rigid* requirements, which cannot run unless they obtain at least a fixed number of processors, can be a bottleneck, reducing system throughput and processor utilization. This overhead might be reduced by treating jobs as having *malleable, evolving* requirements rather than having a hard, fixed requirement for a specific number of processors. Malleable jobs are parallel tasks that can be performed with different numbers of processors, depending on how many they are allotted by the control. Evolving jobs have requirements that vary over time, and can request changes in their allotments when their needs change. Scheduling for such jobs is studied in [KKD02,PL95,IRSD99,Edm00]. (The exact usage of these terms does not seem to be completely fixed. [KKD02] for example, uses the term adaptive job to denote both concepts.)

We consider the problem of scheduling evolving parallel tasks that can request more processors from a scheduler of a multi-processor system with T identical processors while they are running. Preemption is possible, but expensive, since many parallel tasks have a high context switch cost (e.g. rendering applications). The goal of the scheduler is thus to minimize preemption while satisfying the processor requirements. We make this precise below. In the following problem, a *user* will represent a task/job and *resources* represent processors.

Definition 1. Resource Allocation Problem (RAP) *There are n users. The input specifies the amount of resources $r_{j,t}$ required by user j at time t. At each point in time, a scheduler A must allocate an amount $\ell^A_{j,t}$ to user j that is at least this required amount ($\ell^A_{j,t} \geq r_{j,t}$). The scheduler has only T resources to allocate and hence $\sum_j \ell^A_{j,t} \leq T$. The objective function is to minimize the number of times that the schedule changes the amount $\ell^A_{j,t}$ allocated to each user.*

We assume that at each point in time, the total resources requested $\sum_j r_{j,t}$ does not exceed the amount T available. Otherwise, all schedulers will fail. If these amounts are equal, then the scheduler has no choice. However, if it is less, then the scheduler must decide where to allocated the extra resources. If the scheduler knows the future then it will give the extra resources to users that will need more later.

An online scheduler lacks knowledge about future requirements of the users and so it must guess where to allocate its extra resources. Making it harder, in many situations, the online scheduler will not learn of changes to user's requirements until the user complains because it does not have enough. To formalize the above limitation on information available to a scheduler, we consider a restricted class of schedulers. A *restricted scheduler* only has access to input in the following way. At time t, it learns the set of currently unsatisfied users. It then repeatedly reallocates resources, paying a cost for each change. After each reallocation, it learns the set of unsatisfied users (some users given more might still not have enough resources, whereas others might become unsatisfied after part of their allocations were given to complaining users.) This repeats until all users are satisfied. (Note that the scheduler in this model may be charged repeatedly for reallocations involving the same processor in the same time step.) We denote the above **Restricted-scheduler Resource Allocation Problem as RRAP**. Note that unlike RAP, in RRAP the scheduler never learns the resource requirements exactly, only an upper bound for each.

In RAP, it is always possible to fulfill all the requirements. However, schedulers often have to deny some requests in order to preserve resources for others. Our techniques also apply to the following variant, where the scheduler can deny requests by paying a penalty.

Definition 2. Resource Allocation Problem with Penalties (RAPP) *There are n users and a pool of T resources. At time t some user $j \in [n]$ produces a request for resource $r_{j,t}$ with a* **penalty** *$p_{j,t} \geq 1$. At each time, the scheduler allocates $\ell_{j,t}$ resources to user j, where the sum of the allocations is at most T. The scheduler's cost is the total number of changes made to allocations, plus the total penalties $p_{j,t}$ over all times when the scheduler fails to satisfy the t'th request.*

Note that, for simplicity, we define user requirements for this problem as instantaneous rather than continuing. However, we could model users with continuing requirements by having users issue requests whenever their current requirements are unmet (due to increase in their requirements or reallocation of their resources).

A restricted scheduler for the RAPP problem only learns of requests that are currently unsatisfied, but learns the required amount and penalty for such requests.

RAPP could model resource allocation for distributed grid computing systems, where the penalty represents the priority of the job requesting the resource. The higher the priority the higher the penalty if the algorithm chooses to not satisfy the request. It could also model malleable jobs, which would make requests for each possible number of processors from largest to smallest until a request is satisfied, with increasing penalties.

We will show that the RAP problem reduces to RAPP, by repeating requests until they must be satisfied. RAP could represent a situation where the jobs are rigid but evolving and the scheduler is not allowed to starve any job.

For our algorithms, we consider memoryless online algorithms that utilize sT resources, and compare them to all knowing all powerful adversaries with a total budget of T. The reason is that when given no extra resources, the online/off-line competitive ratio is infinite (See Sect. 5). Let $\mathrm{E}(\mathcal{A}_s(\sigma))$ denote the expected cost of algorithm $\mathcal{A}$ with sT resource on input σ. Let $OPT_1(\sigma)$ denote the minimum cost of any solution with T resources on the same sequence σ. We call the algorithm $\mathcal{A}$ (s, c)-competitive if there exists a constant d, possibly depending on n and T, such that for all inputs σ, $\mathrm{E}(\mathcal{A}_s(\sigma)) \leq c\ OPT_1(\sigma) + d$. Our goal is to find algorithms that are (s, c)-competitive for as small values of s and c as possible. Our results are $(s, O(\log_s n)$-competitive algorithms for RAP and RAPP problems.

1.2 Transmission-Minimizing Approximate Value Problem

Small power wireless sensor networks are used for variety of applications from monitoring seismological data to tracking wildlife [LSZM04]. Communication in such networks is expensive, especially in terms of power usage. Consider such a network in which there is a set of sensors each capable of transmitting one value to a central cache (base station). The cache needs to know the values read by each sensor, allowing for some imprecision. If the cache needs perfect precision then the cache must be updated each time a sensor's value changes. To minimize communication, the central station might relax the precision for the values read by the sensor. The cache would still need to be notified when the value changed by more than the allowable precision. Thus, some sensors (with fluctuating values) might be given a more relaxed standard of precision than sensors with more stable values.

The problem of setting the precision of approximated cached values is defined in [OLW01] as follows. There are data sources $S_1, \ldots, S_n$. Each data source S_i hosts a value V_i. There is a cache C, which holds an interval approximation to the exact values $V_1, \ldots, V_n$. An interval $[L, H]$ is a valid approximation of a numeric value V if $V \in [L, H]$. If an interval becomes invalid, then the cache must be updated. We add a hard requirement that the sum of the lengths of intervals assigned to the data sources be at most a parameter T; in [OLW01], this was a soft requirement. The goal is to maintain valid intervals of total size T in a way that minimizes the overall network traffic, assuming that each report of an invalid interval and each reassignment of intervals takes one message. We call our version of this problem **Transmission-Minimizing Approximate Value Problem (TMAV)**. [OLW01] give an optimal solution to a related problem under the assumption that the sources' values arise from certain types of probability distributions. Similar problems were studied in [CYV04,cKA02], who present heuristic approaches and experimentally evaluate their performance. However, we give the first analysis of algorithms when we allow the values read by the data sources to be adversarial.

Similarly to the RAP problem, we compare online algorithms whose sum of all intervals adds up to sT to an adversary bounded by the total precision T. Our algorithm is almost identical to that for RAP, and achieves a similar competitive ratio of $O(\log_s n)$. For scheduling, it would seem more compelling to take s as small as possible, but for TMAV, a larger value of s might still be reasonable, since a factor of s corresponds to a loss of only $\log s$ bits of precision. Again, our algorithm only uses very restricted access to the input: it only sees the current value at a sensor when it becomes invalid or when it is changes the required precision at the sensor. This seems essential to any meaningful solution to the problem, since we are trying to minimize communication. Nevertheless, when we prove lower bounds, they apply to algorithms that have a complete history of all values up to the current time.

Our results and outline of paper:

1. In Sect. 2 we present a randomized, memoryless, online algorithm for RRAP problem that is $(s, O(\log_s n)$-competitive for $s > 3$ and show how this algorithm can be used to solve RAP and RAPP.

2. In Sect. 3 we show how the algorithm above is modified to give a similar result for the TMAV problem, for $s > 6$.

3. In Sect. 4 we also show how to reduce RAP to TMAV and formally define reduction between online problems with respect to adaptive online adversary.

4. In Sect. 5 we prove lower bounds on the competitive ratio achieved by any online algorithm.

2 The Steal from the Rich Algorithm

Our algorithm, *Steal-From-the-Rich* (SFR) for the RRAP problem, is more or less as follows. When a user j requests more resources (but does not specify how much he needs), the scheduler chooses a random user k with probability proportional to the amount $\ell^A_{k,t}$ currently allocated to it. Then he moves resources from j to k so that neither changes by more than a factor of $r = \Theta(\sqrt{s})$.

The *Steal-From-the-Rich* (SFR) algorithm is defined more precisely as follows. Initially, the resources are partitioned evenly, i.e. $\ell^{\mathrm{SFR}}_{j,0} = \frac{sT}{n}$. Let $r = \Theta(\sqrt{s})$ and $\mu > 0$ be parameters defined more precisely later. If at time t user j requests more resources, the algorithm repeats the following until all demands are satisfied; setting j to be the user whose demands are not satisfied:

– It selects another user k at random with probability proportional to its resource allocation, i.e. $\Pr[\ k \text{ is selected }] = \frac{\ell^{\mathrm{SFR}}_{k,t-1}}{sT - \ell^{\mathrm{SFR}}_{j,t-1}}$.

– $\delta \leftarrow \min\left\{\ \ell^{\mathrm{SFR}}_{k,t-1} - \mu\frac{T}{n}, \frac{r-1}{r}\ell^{\mathrm{SFR}}_{k,t-1}, (r-1)\ell^{\mathrm{SFR}}_{j,t-1}\ \right\}$, $\ell^{\mathrm{SFR}}_{k,t} \leftarrow \ell^{\mathrm{SFR}}_{k,t-1} - \delta$, $\ell^{\mathrm{SFR}}_{j,t} \leftarrow \ell^{\mathrm{SFR}}_{j,t-1} + \delta$.
The other allocations are left unchanged.

Note that the choice of δ is the maximum so that ℓ_k^{SFR} does not decrease below $\mu\frac{T}{n}$ nor by more than a factor of r and that ℓ_j^{SFR} does not increase by more than a factor of r. In addition SFR maintains that $\sum_j \ell_{j,t}^{\mathrm{SFR}} = sT$ and $\ell_{j,t}^{\mathrm{SFR}} \geq \mu\frac{T}{n}$ for all intervals, hence it is a valid schedule using sT resources.

Theorem 1. *For $s > 3$, the algorithm* SFR *for the Resource Allocation Problem is $(s, O(\log_s n))$-competitive against an adaptive online adversary.*

Proof. Let OPT be any adaptive online adversary strategy with a total of T resources. We prove the $O(\log_s n)$ competitive ratio using the potential function $\Phi\colon R^n \times R^n \to R^+$, where the first input describes the algorithm's configuration and the second the adversary's. More precisely,

$$\Phi_{j,t} = \frac{14}{\log r}\left|\log\left(\frac{\ell_{j,t}^{\mathrm{SFR}}}{r\ell_{j,t}^{\mathrm{OPT}} + \mu T/n}\right)\right| \qquad \text{and} \qquad \Phi_t = \sum_{j=1}^{j=n} \Phi_{j,t}.$$

The intuition is that this potential function is small when all allocations assigned by the SFR are proportional to those assigned by the adversary. In this case, SFR allocates more to each user and hence any cost incurred by the algorithm will also be incurred by the adversary. At any point when SFR has a cost and the adversary does not, SFR will grow an allocation that is short relative to the adversary's and will probably shrink an allocation that is long relative to the adversary's, thus reducing the potential.

Observe that $\Phi_{j,t} \leq O(\log_s n)$. There are two cases in bounding $\Phi_{j,t}$.

1. If $\ell_{j,t}^{\mathrm{SFR}} \gg \ell_{j,t}^{\mathrm{OPT}}$ then
$\Phi_{j,t} = \frac{14}{\log r}\log\left(\frac{\ell_{j,t}^{\mathrm{SFR}}}{r\ell_{j,t}^{\mathrm{OPT}}+\mu T/n}\right) \leq \frac{14}{\log r}\log\left(\frac{sT}{\mu T/n}\right) \leq O(\frac{\log n}{\log s}) = O(\log_s n)$.
2. If $\ell_{j,t}^{\mathrm{SFR}} \ll \ell_{j,t}^{\mathrm{OPT}}$, then
$\Phi_{j,t} = \frac{14}{\log r}\log\left(\frac{r\ell_{j,t}^{\mathrm{OPT}}+\mu T/n}{\ell_{j,t}^{\mathrm{SFR}}}\right) \leq \frac{14}{\log r}\log\left(\frac{rT+\mu T/n}{\mu T/n}\right) \leq O(\frac{\log n}{\log s}) = O(\log_s n)$.

This implies that $\Phi_t \leq O(n \log n)$, for all t.

Let SFR_t and OPT_t be the costs incurred by the algorithm and the adversary during the t^{th} change in allocations and define $a_t = \mathrm{SFR}_t + (\Phi_t - \Phi_{t-1})$ to be the *amortized update cost* to the algorithm. We will show that for every t, $\mathrm{E}(a_t) \leq O(\log_s n)OPT_t$, where the expectation is over the SFR's random choice for k conditioned on the configurations at time t–1. This establishes the claimed competitive ratio because:

$$\begin{aligned}
\mathrm{E}(\mathrm{SFR}_s(\sigma)) &= \mathrm{E}\Big(\sum_t \mathrm{SFR}_t\Big) = \mathrm{E}\Big(\sum_t (a_t - \Phi_t + \Phi_{t-1})\Big) \\
&= \sum_t \mathrm{E}(a_t) - \Phi_{end} + \Phi_0 \leq \sum_t \mathrm{E}(a_t) + \Phi_0 \\
&\leq \sum_t O(\log_s n)OPT_t + O(n \log n) = O(\log_s n)OPT(\sigma) + d,
\end{aligned}$$

for some $d \in O(n \log n)$ (Recall that for all j, t we bounded $\Phi_{j,t} \leq O(\log_s n)$ and s is a constant, hence $\Phi_0 \leq O(n \log n)$.

Our goal is to establish $\mathrm{E}(a_t) \leq O(\log_s n) OPT_t$. We assume without loss of generality that the adversary reallocates resources before issuing an increased request for resources, since moving such a reallocation to before the request changes neither the adversary's nor the algorithm's cost. Thus, we can break the analysis into two cases where one type of two events described below happen.

- **Case 1:** The adversary reallocates resources to users, and the algorithms does nothing (since no user's demands have changed).
- **Case 2:** One iteration of SFR's main loop occurs. The algorithm moves resources from one user to another, the adversary does nothing.

Note that if $\mathrm{SFR}_t = OPT_t = 0$, then $a_t = 0$ and the claim, $\mathrm{E}(a_t) \leq O(\log_s n) OPT_t$, holds trivially.

- **Analysis Case 1:** The adversary, anticipating that the user's needs will change, adjusts the allocations of some of the users, while the the algorithm not aware of them makes no changes to its configuration: $\mathrm{SFR}_t = 0$ and $OPT_t > 0$. For each such user j, Φ_j increases by at most its maximum value, which we saw is $O(\log_s n)$. Hence, $a_t \leq 0 + O(\log_s n) OPT_t$, giving the competitive ratio as stated.
- **Analysis Case 2:** One iteration of SFR has occurred. $\mathrm{SFR}_t = 2$, because SFR has changed the allocations of two users j and k, and OPT does nothing, hence $OPT_t = 0$. Because user j is requesting more from SFR and not from OPT, we have that $\ell^{\mathrm{SFR}}_{j,t-1} \leq \ell^{\mathrm{OPT}}_{j,t-1}$. Having changed only the allocations of user j and the randomly chosen user k, $\Delta\Phi = \Phi_t - \Phi_{t-1} = \Delta\Phi_j + \Delta\Phi_k$. We bound the expectation of this change to be at most -2. This gives the required bound as $\mathrm{E}(a_t) = 2 + \mathrm{E}(\Phi_t - \Phi_{t-1}) \leq 0$.
  We consider two cases.
  Event B**:** Let B be the unlikely and unfortunate event that user k which is randomly selected by SFR has a relatively small allocation, namely $\ell^{\mathrm{SFR}}_{k,t-1} < r^2 \ell^{\mathrm{OPT}}_{k,t-1} + \mu r \frac{T}{n}$. This event is unlikely because on average the users are allocated $s = \Theta(r^2)$ times more under SFR and because the probability k is selected is proportionally to its allocation. More formally, let K denote the set of users k for which if selected event B occurs. We choose each such k with probability $\frac{\ell^{\mathrm{SFR}}_{k,t-1}}{sT - \ell^{\mathrm{SFR}}_{j,t-1}}$. Because j is requesting, we have that $\ell^{\mathrm{SFR}}_{j,t-1} \leq \ell^{\mathrm{OPT}}_{j,t-1} \leq T$. Because k causes event B, we have that $\ell^{\mathrm{SFR}}_{k,t-1} < r^2 \ell^{\mathrm{OPT}}_{k,t-1} + \mu r \frac{T}{n}$. This gives
$$\begin{aligned} \Pr[B] &= \sum_{k \in K} \Pr[\text{SFR chooses } k\ ] = \sum_{k \in K} \frac{\ell^{\mathrm{SFR}}_{k,t-1}}{sT - \ell^{\mathrm{SFR}}_{j,t-1}} \\ &< \frac{\sum_{k \in K} (r^2 \ell^{\mathrm{OPT}}_{k,t-1} + \mu r \frac{T}{n})}{sT - T} \leq \frac{r^2 \cdot T + \mu r \cdot T}{(s-1)T} = \frac{r^2 + \mu r}{s-1} \leq \frac{3}{7}. \end{aligned}$$
  We set $r = \Theta(\sqrt{s})$ so that this is true. Note that for $r > 1$ and $\mu > 0$, we need $s > 3.34$ for this to be true. (We could decrease s to $3{+}\epsilon$ by setting $r = 1{+}\frac{\epsilon}{11}$,

$\mu = \frac{\epsilon}{11}$, $\Pr[B] = \frac{1}{2} - \frac{\epsilon}{11}$, the multiplicative constant 14 in the formula for $\Phi_{t,j}$ to $\frac{11}{\epsilon}$, and the competitive ratio[3] to $O(\frac{11}{\epsilon \log r} \log(\frac{rn}{\mu})) = O(\frac{\log n}{\epsilon^2})$.)

Event $\overline{B}$: Suppose that the unlikely and unfortunate event B does not occur and the user selected has relatively big allocation, namely k has $\ell^{\mathrm{SFR}}_{k,t-1} \geq r^2 \ell^{\mathrm{OPT}}_{k,t-1} + \mu r \frac{T}{n}$. Recall that $\ell^{\mathrm{SFR}}_{j,t-1}$ is increased and $\ell^{\mathrm{SFR}}_{k,t-1}$ is decreased by $\delta = \min\{\ell^{\mathrm{SFR}}_{k,t-1} - \mu\frac{T}{n},\ \frac{r-1}{r}\ell^{\mathrm{SFR}}_{k,t-1},\ (r-1)\ell^{\mathrm{SFR}}_{j,t-1}\}$. The first of these possible values for δ does not occur when $\overline{B}$ happens, because $\ell^{\mathrm{SFR}}_{k,t-1} \geq \mu r \frac{T}{n}$ and hence $\ell^{\mathrm{SFR}}_{k,t-1} - \mu\frac{T}{n} \geq \frac{r-1}{r}\ell^{\mathrm{SFR}}_{k,t-1}$.

Now lets look at the change of the potential function for user j when we are in case 2. Here we have $\ell^{\mathrm{SFR}}_{j,t-1} < \ell^{\mathrm{OPT}}_{j,t-1}$, then $\ell^{\mathrm{SFR}}_{j,t} \leq r\ell^{\mathrm{SFR}}_{j,t-1} \leq r\ell^{\mathrm{OPT}}_{j,t-1} = r\ell^{\mathrm{OPT}}_{j,t} < r\ell^{\mathrm{OPT}}_{j,t} + \mu\frac{T}{n}$. Recall that $\Phi_j = \frac{14}{\log r}\left|\log\left(\frac{\ell^{\mathrm{SFR}}_j}{r\ell^{\mathrm{OPT}}_j + \mu T/n}\right)\right|$ and because of the absolute value operator, this function decreases as ℓ^{SFR}_j increases when ℓ^{SFR}_j is small and increases with it when it is large.

We now consider the effect of these changes on Φ_j and Φ_k in the two remaining cases: when j increases by a factor of r (event C) or when k decreases by a factor of r (event D).

- Let $C \cap \overline{B}$ be the event that $\delta = (r-1)\ell^{\mathrm{SFR}}_{j,t-1}$, and hence $\ell^{\mathrm{SFR}}_{j,t-1}$ increases by a factor of r. Then $\mathrm{E}(\Delta\Phi^{\mathrm{SFR}}_j | C \cap \overline{B}) = \frac{14}{\log r}\log\left(\frac{\ell^{\mathrm{SFR}}_{j,t-1}}{\ell^{\mathrm{SFR}}_{j,t}}\right) = -14$. Further because ℓ^{SFR} decreases then $\mathrm{E}(\Delta\Phi^{\mathrm{SFR}}_k | C \cap \overline{B}) = \frac{14}{\log r}\log\left(\frac{\ell^{\mathrm{SFR}}_{k,t}}{\ell^{\mathrm{SFR}}_{k,t-1}}\right) < 0$. Combined we have that
$$\mathrm{E}(\Delta\Phi^{\mathrm{SFR}} | C \cap \overline{B}) = \mathrm{E}(\Delta\Phi^{\mathrm{SFR}}_k | C \cap \overline{B}) + \mathrm{E}(\Delta\Phi^{\mathrm{SFR}}_j | C \cap \overline{B}) < -14.$$
- Let $D \cap \overline{B}$ be the event that $\delta = \frac{r-1}{r}\ell^{\mathrm{SFR}}_{k,t-1}$, and hence $\ell^{\mathrm{SFR}}_{k,t-1}$ decreases by a factor of r. Given event $\overline{B}$ happens, we have $\ell^{\mathrm{SFR}}_{k,t} \geq \frac{1}{r}\ell^{\mathrm{SFR}}_{k,t-1} \geq \frac{1}{r}\left(r^2\ell^{\mathrm{OPT}}_{k,t-1} + \mu r\frac{T}{n}\right) = r\ell^{\mathrm{OPT}}_{k,t} + \mu\frac{T}{n}$, then $\mathrm{E}(\Delta\Phi^{\mathrm{SFR}}_k | D \cap \overline{B})$ $= \frac{14}{\log r}\log\left(\frac{\ell^{\mathrm{SFR}}_{k,t}}{\ell^{\mathrm{SFR}}_{j,t-1}}\right) = -14$. As in the previous case since $\ell^{\mathrm{SFR}}_{j,t-1}$ increases we have $\mathrm{E}(\Delta\Phi^{\mathrm{SFR}}_j | D \cap \overline{B}) = \frac{14}{\log r}\log\left(\frac{\ell^{\mathrm{SFR}}_{j,t-1}}{\ell^{\mathrm{SFR}}_{j,t}}\right) < 0$. Combined we have
$$\mathrm{E}(\Delta\Phi^{\mathrm{SFR}} | D \cap \overline{B}) = \mathrm{E}(\Delta\Phi^{\mathrm{SFR}}_k | D \cap \overline{B}) + \mathrm{E}(\Delta\Phi^{\mathrm{SFR}}_j | D \cap \overline{B}) = -14.$$

Now we bound the expectation of the change of the potential function when we are in Case 2, and event B has not occurred:
$$\begin{aligned}\mathrm{E}(\Delta\Phi^{\mathrm{SFR}} | \overline{B}) &= \Pr[C \cap \overline{B}] \cdot \mathrm{E}(\Delta\Phi^{\mathrm{SFR}} | C \cap \overline{B}) + \Pr[D \cap \overline{B}] \cdot \mathrm{E}(\Delta\Phi^{\mathrm{SFR}} | D \cap \overline{B}) \\ &\leq -14(\Pr[C \cap \overline{B}] + \Pr[D \cap \overline{B}]) \leq -14\Pr[\overline{B}].\end{aligned}$$

Event B: We now bound $\Delta\Phi$ when the unlikely and unfortunate event B does occur. Because we are in Case 2, the above argument that $\mathrm{E}(\Delta\Phi^{\mathrm{SFR}}_j | B) \leq$

[3] If OPT was restricted so that it could never give more than αT to a single user, then we could decrease s to $2+\alpha+\epsilon$, because $\ell^{\mathrm{SFR}}_{j,t-1} \leq \ell^{\mathrm{OPT}}_{j,t-1} \leq \alpha T$ allows us to change the $s-1$ to $s-\alpha$.

0 does not change. Since SFR does not change k by more than a factor of r and because OPT does not change the allocations at all, then $\mathrm{E}(\Delta\Phi_k^{\mathrm{SFR}}|B) = \frac{14}{\log r}\log(\frac{\ell^{\mathrm{SFR}}_{k,t-1}}{\ell^{\mathrm{SFR}}_{k,t}}) \le \frac{14}{\log r}\log r$.

$$\mathrm{E}(\Delta\Phi|B) \le \mathrm{E}(\Delta\Phi_j|B) + \mathrm{E}(\Delta\Phi_k|B) \le 14.$$

Conclude Case 2: We bound the change of the potential function as:

$$\mathrm{E}(\Delta\Phi) = \Pr[B]\cdot\mathrm{E}(\Delta\Phi \mid B)+(1-\Pr[B])\cdot\mathrm{E}(\Delta\Phi \mid \overline{B}) \le \frac{3}{7}\cdot(14)+\frac{4}{7}\cdot(-14) = -2.$$

We have established that $\mathrm{E}(a_t) = \mathrm{SFR}_t + \mathrm{E}(\Delta\Phi) \le 0 = O(\log_s n)OPT_t$.

Hence $\mathrm{E}(a_t) \le O(\log_s n)OPT_t$, thus concluding the proof. □

2.1 Using SFR to Solve RAPP

We can also use a variant of SFR to solve RAPP as follows. When at time t user j requests $r_{j,t}$ resources with penalty $p_{j,t}$ then we call the main loop for SFR(j) $\lceil p_{j,t} \rceil$ times or until $l_{j,t} \ge r_{j,t}$.

We argue that this is $O(\log_s n)$ competitive. First, note that the total penalty costs for SFR are at most half that of its communication costs, since we only suffer a penalty $p_{j,t}$ after calling the main loop SFR(j) at least $p_{j,t}$ times, and each time has communication cost 2. Thus, if we can bound the communication costs of SFR in terms of the total costs of OPT, the same bound, times 1.5, holds for the total costs for SFR.

We use the same potential function as for the analysis of SFR for RAPP. Note that the same bounds in the proof hold for changes in the adversary's allotments. Moreover, the same bound on the expected amortized cost of a loop of SFR holds when $\ell^{\mathrm{SFR}}_{j,t-1} \le \ell^{\mathrm{OPT}}_{j,t-1}$. If $\ell^{\mathrm{SFR}}_{j,t-1} \ge r_{j,t}$, there is no complaint and the algorithm has no costs. In the final case, $r_{j,t} \ge \ell^{\mathrm{SFR}}_{j,t-1} \ge \ell^{\mathrm{OPT}}_{j,t-1}$, so the adversary fails to satisfy the request and pays penalty $p_{j,t}$. Since SFR performs at most $2p_{j,t}$ iterations of its main loop, and each iteration has communication cost 2, and changes the potential function by at most 28, the total amortized communication costs are at most 60 times the costs for OPT.

3 The Transmission Minimizing Approximate Value Problem

Recall that TMAV problem has n sensors reading values and a central cache must maintain an estimate of each value by knowing an interval $I_{j,t} = [a_{j,t}, b_{j,t}]$ containing this value. The constraint is that the sum of the intervals (or allowable errors) always be bounded by T, namely $\sum_j (b_{j,t} - a_{j,t}) \le T$. Among other things, this assures that the cache knows the sum $\sum_j v_{j,t} \in \left[\sum_j a_{j,t}, \sum_j b_{j,t}\right]$ within an accuracy of T. In an online algorithm, the cache only learns the new value $v_{j,t}$ if it moves outside of its current interval. At such times, the cache sends a message to the node telling it its new interval. The objective is to minimize the the number of messages sent between the cache and the nodes.

To solve the TMAV problem we will modify our algorithm for the RAP. The input to the TMAV problem specifies the value $v_{j,t}$ of node $j \in [n]$ at time $t \geq 0$. The algorithm for RAP on the other side computes allocations, based on resource requests, so we need to map 1) the allocations computed to intervals and 2) resource pool sT to appropriate precision parameter.

The Steal-From-the-Rich algorithm for this problem mimics that for RAP but with a total precision of $sT/2$. The algorithm partitions the precision $sT/2$ amongst the n nodes. When the cache learns a new value $v_{j,t}$, it sets $I_{j,t} = [v_{i,t} - \ell^{\mathrm{SFR}}_{j,t}, v_{j,t} + \ell^{\mathrm{SFR}}_{j,t}]$ to have width $2\ell^{\mathrm{SFR}}_{j,t}$ centered around this new value. We assume the algorithms learns a new value $v_{j,t}$, when the value moves out of the current interval or when the interval is changed.

When a node reports that its value is outside its current interval, then we mimic one iteration of SFR to increase its allocation $\ell^{\mathrm{SFR}}_{j,t}$, decrease one other nodes interval. We assign both nodes intervals of width their new allocations, centered around their current values. Note that since $\sum_i \ell^{\mathrm{SFR}}_{j,t} = \frac{sT}{2}$ then the total length of intervals is sT.

Theorem 2. *For $s > 6$, the algorithm* SFR *for TMAV Problem is* $(s, O(\log_s n))$*-competitive against an adaptive online adversary.*

Proof. (sketch) Let $I^{\mathrm{OPT}}_{j,t}$ and $I^{\mathrm{SFR}}_{j,t}$ be the current intervals assigned by OPT and SFR to node j. We'll say that node j is *separated* if the center $c_{j,t}$ of $I^{\mathrm{SFR}}_{j,t}$ is not contained in $I^{\mathrm{OPT}}_{j,t}$. This can occur when OPT incurs a cost in order to move its interval. But as soon as SFR incurs its next cost for this node, this node is no longer separated because SFR moves the center of its interval to the new value, which is contained in OPT's interval. Then no matter how the value changes, this node remains unseparated until OPT is required to move its interval again.

We use the same potential function as in Sect. 2, setting $\ell^{\mathrm{OPT}}_{j,t} = |I^{\mathrm{OPT}}_{j,t}|$, but with one additional item. We add to the potential $O(\frac{\log n}{\log s})$ times the number of separated intervals at time t, $N_s(t)$.

$$\Phi_{j,t} = \frac{14}{\log r} \left| \log \left(\frac{|I^{\mathrm{SFR}}_{j,t}|}{r|I^{\mathrm{OPT}}_{j,t}| + \mu T/n} \right) \right| \quad \text{and} \quad \Phi_t = \sum_{j=1}^{j=n} \Phi_{j,t} + O(\log_s n) \cdot N_s(t).$$

A key observation is that if an update to an unseparated sensor incurs a cost for SFR but not for OPT, then $v_{j,t}, c_{j,t} \in I^{\mathrm{OPT}}_{j,t-1}$ and $v_{j,t} \notin I^{\mathrm{SFR}}_{j,t-1}$, hence $\frac{1}{2}\ell^{\mathrm{SFR}}_{j,t-1} = \frac{1}{2}|I^{\mathrm{SFR}}_{j,t-1}| \leq |v_{j,t} - c_{j,t}| \leq |I^{\mathrm{OPT}}_{j,t-1}| = \ell^{\mathrm{OPT}}_{j,t-1}$. Except for an extra factor of two, which we compensate for by having SFR for this problem have an s that is twice as big, this conclusion $\ell^{\mathrm{SFR}}_{j,t-1} \leq \ell^{\mathrm{OPT}}_{j,t-1}$ is the same as it had been in case 2 for the RAP. On the other hand, an update to a separated interval reduces $N_s(t)$ by 1, and hence has negative amortized cost. Thus, the amortized costs for SFR for each operation are at most $O(\log_s n)$ times the costs of OPT. □

4 Reductions Between Online Problems

In general reductions between problems are used in at least two different ways. The first usage is: given an algorithm for problem A we can obtain an algorithm for problem B by combining an appropriate reduction function which maps instances of B to instances of A, then use a good algorithm for A. On the other hand, if we have a lower bound for problem B, we inherit the same lower bound for problem A from a reduction from B to A. Online reductions have been used to design algorithms in, for example, [AL04], which solves a fractional version of a Maximizing Switch Throughput problem and then reduces the more interesting discrete version to the fractional version. Unlike in complexity, online reductions have rarely been used to compare the likely hardness of online algorithms, probably because researchers have been successful at proving lower bounds directly. Since there is a large gap between our lower and upper bounds for these problems, however, we are interested in the relationship between them.

In this section, we show that the RAP problem is at least as hard as the TMAV problem by reducing RAP to TMAV, in a way that preserves the competitive ratio. (Although intuitively, our algorithm for TMAV is based on one for RAP, we do not know of any general reduction in this direction.) We later give a general discussion of what constitutes an online reduction, especially against adaptive online adversaries. We also give a number of other reductions, from RAP to RAPP, and from paging to RAP.

Theorem 3. *[RAP $\leq_{\mathrm{AD\_ON}}$ TMAV] Let A be any (s, k)-competitive algorithm against adaptive online adversaries for the TMAV problem. Then there exists an (s, k)-competitive algorithm against adaptive online adversaries for the RAP problem.*

Proof. Given an algorithm A for TMAV we construct an algorithm B for RAP, which has the same competitive ratio as A. B uses A as a subroutine to compute the allocations of resources to the users. Let $\ell_{j,t}$ (length) be the high end point of the interval assigned to sensor j at time t by A. The input to B is a request $r_{j,t}$ from user j for more resources. B maintains the following set of invariants:

1. The current interval A has assigned sensor j contains 0 and has high endpoint $\ell_{j,t}$, the amount of resources B has assigned to user j.
2. The resource allocated $\ell_{i,t-1}$ to each user prior to serving the t-th request is at least as big as the last requested resource by the user, i.e, $r_{i,t-1} \in [0, \ell_{i,t-1}]$, for all $i \in [n]$.

Suppose the set of invariants holds after $(t-1)$-st request. And suppose B receives a request $r_{j,t}$ for more resource from user j. Then B makes a call to A, specifying that sensor j, has changed its value to $v = r_{j,t}$, $A(j, v)$. Then B makes n calls to A, changing all sensors current values to 0 (in any order.) At this point, A must have assigned each sensor an interval containing 0. B gives each user an allocation equal to the high endpoint of the corresponding interval assigned by A. Note that only intervals changed by A need a reallocation, so B's costs are at most those of A. If all users demands are satisfied by this allocation, the algorithm halts. Otherwise, it sets j to be some complaining user, and v to be the users demand, and repeats the above process. The above process repeats until all users are satisfied. (This must happen eventually, since in each iteration A has cost at least 1, whereas an adversary that assigns each sensor i the interval $[0, r_{i,t}]$ will have cost at most n no matter how many iterations occur. If A's costs tends towards infinity, it would contradict the assumption that A is competitive.) By the invariant, B's total allocation is bounded above by the total length of intervals that A uses, and so B also stays within budget.

Let OPT_B be an adaptive online adversary for B. Let OPT_A be the following online adversary for A, maintaining the invariant that the interval assigned by OPT_A to sensor i is $[0, l_{i,t}^{OPT_B}]$ at all times. If OPT_B changes an allocation to a user, OPT_A changes the high endpoint of that sensor's interval. If OPT_B generates a request for r from user j, OPT_A generates the series of requests that B would to A. (Note that this requires OPT_A to be adaptive, since it must know A's state in order to generate these requests.) Note that OPT_A has no costs for this series of requests, since its intervals contain $[0, p_{i,t}]$ as sub-intervals during this time period. Thus, the total costs for OPT_A are at most those for OPT_B.

We then have: $Cost(B) \leq Cost(A) \leq k \cdot Cost(OPT_A) + c \leq k \cdot Cost(OPT_B) + c$. Thus, B is also k-competitive against any adaptive online adversary. □

Next we formalize the notion of competitive ratio preserving online reduction with respect to adaptive online adversary, denoted as $\leq_{\mathrm{AD\_ON}}$.

Let Π_1, Π_2 be two online problems whose hardness we want to relate. Let the quality of any solution σ_i for Π_i be measured by $Cost_i(\sigma_i)$ functions, for $i = 1, 2$. We want to show that Π_1 reduces to Π_2 via competitive ratio preserving reduction, $\Pi_1 \leq_{AD\_ON} \Pi_2$, such that, if there exist a c-competitive algorithm for Π_2 then there exists $F(c)$-competitive algorithm for Π_1.

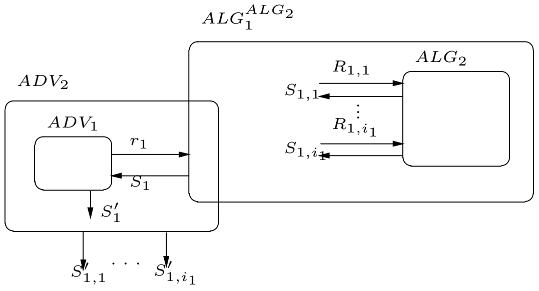

Fig. 1. One round of interaction between online algorithms for Π_1, Π_2 ALG_1, ALG_2 and the respective adversaries ADV_1, ADV_2.

Let ALG_2 be a c-competitive algorithm for Π_2 and ADV_1, ADV_2 be adaptive online adversaries for Π_1, Π_2, respectively. We use the following notation. We let ADV_1 generate the sequence $r_1, r_2, \ldots, r_t$. When the algorithm for Π_1 ALG_1

receives a request $r_m \in r_1, \ldots, r_t$ it starts a loop for say i_m iterations and calls the algorithm for Π_2 ALG_2. In each iteration ALG_1 sends a request $R_{m,j}$ to ALG_2 and gets back a response $S_{m,j}$ for $j = 1, \ldots, i_m$. Then ALG_1 makes a decision S_m. ADV_1 observes S_m and commits to decision S'_m. Then the adversary for Π_2 observes everything ALG_1 has observed and in addition the response of ADV_1, namely S'_m and has to commit itself to decisions $S'_{m,1}, \ldots, S'_{m,i_m}$. Let $H^{(t)}_{ADV_1}$ be the history known to ADV_1 until time t. ADV_1 does not need to remember $r_1, \ldots, r_t$ since it has generated the sequence, but it does remember the responses of ALG_1 to the sequence $r_1, \ldots, r_t$, hence $H^{(t)}_{ADV_1} = (S_1, S_2, \ldots, S_t)$. Let $H^{(t,j)}_{ALG_1} = (r_1, S_{1,1}, \ldots, S_{1,i_1}, \ldots, r_t, S_{t,1}, \ldots, S_{t,j})$ be the history known to ALG_1, namely the requests given to it by ADV_1 and the responses from ALG_2 up until the j-th response for the t-th request. The history for ALG_2 be $H^{(t,j)}_{ALG_2} = (R_{1,1}, \ldots, R_{1,i_1}, \ldots, R_{t,1}, \ldots, R_{t,j})$ and the history for the adversary ADV_2 be the history observed by ALG_1 along with the responses of ADV_1 to the requests up until time t, namely $H^{(t)}_{ADV_2} = (H^{(t)}_{ALG_1} \circ \{S'_1, S'_2, \ldots, S'_t\})$. An online reduction will have to specify the following online computable functions:

1. A request mapping function g, which ALG_1 uses to generates the next request for ALG_2 based on its current history, namely $g(H^{(t,j)}_{ALG_1}) = R_{t,j+1}$ or $\emptyset$ if the sequence of requests needed to be generated to satisfy the t-th request r_t is complete. Let $S_{t,j+1} = ALG_2(R_{t,j+1})$. Then the two histories are updated accordingly $H^{(t,j+1)}_{ALG_2} = (H^{(t,j)}_{ALG_2} \circ R_{t,j+1})$, $H^{(t,j+1)}_{ALG_1} = (H^{(t,j)}_{ALG_1} \circ S_{t,j+1})$.
2. Once ALG_1 has generated all requests $R_{t,1}, \ldots, R_{t,i_t}$ and has received the corresponding responses from ALG_2 we need a response mapping function h to generate the respond ALG_1 to request r_t: $h(H^{(t,i_t)}_{ALG_1}) = S_t$. Once ALG_1 has computed S_t, then the adversary ADV_1 updates its history: $H^{(t)}_{ADV_1} = (H_{ADV_1} \circ S_t)$ and generates $S'_t = \rho_1(H^{(t)}_{ADV_1})$ in response.
3. We also need to specify the actions of the adversary ADV_2. Given a history $H^{(t,j)}_{ADV_2}$, ADV_2 generates a response $\rho_2(H^{(t,j)}_{ADV_2}) = S'_{t,j}$, for $j = 1, \ldots, i_t$.

Suppose that the following holds:

1. $Cost_1(S_1 \ldots, S_t) \leq \alpha \cdot Cost_2(S_{1,1}, \ldots, S_{1,i_1}, \ldots, S_{t,1}, \ldots, S_{t,i_t})$.
2. $Cost_2(S'_{1,1}, \ldots, S'_{1,i_1}, \ldots, S'_{t,1}, \ldots, S'_{t,i_t}) \leq \beta \cdot Cost_1(S'_1 \ldots, S'_t)$.
3. ALG_2 is c-competitive, hence $Cost_2(S_{1,1}, \ldots, S_{t,i_t}) \leq c \cdot Cost_2(S'_{1,1}, \ldots, S'_{t,i_t})$. then ALG_1 is $F(c) = (\alpha \cdot c \cdot \beta)$-competitive against an adaptive online adversary.

The proof of Theorem 3 is an example of an (α, β) online reduction with $\alpha = \beta = 1$. We can also reduce RAP to RAPP.

Theorem 4. *(RAP $\leq_{\mathrm{AD\_ON}}$ RAPP) There is an online adaptive reduction from RAP with resource factor s to RAPP with resource factor s, with $\alpha = \beta = 1$.*

Proof. Let A be an algorithm for RAPP. We define an algorithm B for RAP as follows. At any time B will have the same allocation as A does. If at any time t, a user j becomes unsatisfied, B simulates a request for $r_{j,t}$ resources from user j to A, assigning it penalty 1, and changes allocations as A does. This continues while there is an unsatisfied user. B maintains the same budget as A, and has at most the costs of A.

Let Opt_B be an online adaptive adversary for RAP. When Opt_B reallocates resources, Opt_A reallocates resources accordingly. When Opt_B generates a request $r_{j,t}$ for resources, let Opt_A simulate B, generating the same sequence of requests to A. Note that, since Opt_B's allocation must satisfy all users' requirements, there is no penalty cost for Opt_A for this sequence of requests. Thus, Opt_A's total costs are the same as its reallocation costs, which are the same as Opt_B's. Opt_A maintains the same budget as Opt_B. $\square$

5 Lower Bounds

Next we show that the factor of s extra resources is crucial to algorithm's performance. We consider the case when both the algorithm and the adversary have the same resources.

Theorem 5. *There is no online algorithm that is (1, O(f(n))) competitive for the RAP problem or the TMAV, for any function f.*

Proof. We give a strategy for the adversary. We consider RAP with $n = 2$ and $T = 1$. The adversary picks a random $r \in [0, 1]$ and places r resources with user 1 and $1 - r$ resources with user 2. The request sequence is then generated as follows: To generate the t-th request, $s_t \in [0, 1]$ is chosen uniformly at random. If $s_t \leq r$, a request for s_t resources is made by user 1. If $s_t > r$, a request for $1 - s_t$ resources is made by user 2. The adversary can always meet these requests without transferring resources, so the adversaries costs are constant for the entire sequence. We show that, for any online algorithm, the expected costs diverge as t goes to infinity.

Let L_t be the largest $s_i \leq r$ with $i \leq t$ (defining $L_t = 0$ if no such s_i exists), and let R_t be the smallest $s_i > r$ with $i \leq t$ (defining R_t to be 1 if no such i exists). For a fixed sequence $s_1, \ldots, s_t$, any value of r with $L_t \leq r \leq R_t$ will generate the same request sequence. Thus, given the request sequence, the value of r is uniformly distributed in the interval $[L_t, R_t]$. Let ℓ_t be the amount of resources assigned to user 1 by the algorithm after the first t requests. Note that ℓ_t is determined by the request sequence $s_1, \ldots, s_t$, so conditioning on ℓ_t does not change the conditional distribution on r. User 2 is then assigned at most $1 - \ell_t$ resources by the algorithm.

If $\ell_t < r$, then any $s_{t+1} \in (\ell_t, r)$ will force the algorithm to reallocate resources, and if $\ell_t > r$, then any $s_{t+1} \in (r, \ell_t)$ will similarly force a reallocation. Thus, the probability that the algorithm has a cost of at least 1 for request $t + 1$ is at least $|r - \ell_t|$. Let $Cost(Alg(s_{t+1}))$ denote the cost of the algorithm for servicing the $t + 1$-st request.

$$\Pr\left[Cost(Alg(s_{t+1})) = 1 \mid s_1, \ldots, s_t\right] \geq |r - \ell_t|.$$

Since $r \in_U [L_t, R_t]$, the expected value of this quantity (given the request sequence so far) is at least $(\ell_t - L_t)/2 + (R_t - \ell_t)/2 = 1/2(R_t - L_t)$.

$$\mathrm{E}\left[Cost(Alg(s_{t+1})) \mid s_1, \ldots, s_t\right] \geq \frac{R_t - L_t}{2}.$$

Fix r. Then the probability that $R_t \geq L_t + 1/(2t)$ is at least the probability that no $s_i \in (r - 1/(4t), r + 1/(4t))$.

$$\Pr\left[R_t \geq L_t + \frac{1}{2t}\right] \geq \Pr\left[\nexists\, s_i \in (r - \frac{1}{4t}, r + \frac{1}{4t})\right] = 1 - \Pr\left[\exists s_i \in (r - \frac{1}{4t}, r + \frac{1}{4t})\right]$$
$$\geq 1 - \sum_{i=1}^{t} \Pr\left[s_i \in (r - \frac{1}{4t}, r + \frac{1}{4t})\right] \geq 1 - \frac{t}{2t} \geq \frac{1}{2}.$$

Thus, the probability that $R_t = L_t \geq 1/(2t)$ is at least $1/2$, and conditioned on this, the probability that the algorithm has a cost of 1 is at least $1/(4t)$, $\Pr\left[Cost(Alg(s_{t+1})) = 1 \mid R_t \geq L_t + \frac{1}{2t}\right] \geq \frac{R_t - L_t}{2} \geq \frac{1}{4t}$.

$$\Pr\ \left[Cost(Alg(s_{t+1})) = 1\right] \geq \Pr\left[(Cost(Alg(s_{t+1})) = 1)\ \cap\ (R_t \geq L_t + \frac{1}{2t})\right]$$
$$\geq \Pr\left[Cost(Alg(s_{t+1})) = 1 \mid R_t \geq L_t + \frac{1}{2t}\right] \cdot \Pr\left[R_t \geq L_t + \frac{1}{2t}\right] \geq \frac{1}{8t}.$$

Thus, the expected cost of the algorithm for the $t+1$'st request is at least $1/(8t)$. Since the sum of the expected costs diverges as t goes to infinity, the expected cost of the algorithm is unbounded, compared to a fixed cost for the adversary.

Similarly, for the TMAV, the adversary assigns node one an interval $(0, r)$ and node 2 an interval $(r, 1)$ for a random $r \in [0, 1]$. Then the adversary generates values for the nodes by picking random s_i in $[0, 1]$. If $s_t \leq r$, node one's new value is s_t, otherwise node two's new value is s_t. The analysis is identical to the proof for RAP. □

In our upper bounds we compared the performance of an online algorithms using sT resources against adversary using T resources. We want to obtain lower bounds on the competitive ratio achievable by online algorithms for RAP using $(1 + \epsilon)$ resources against adversary using 1 resource. We use the following lower bound for Paging.

Theorem 6 ([You91,You94]). *Any randomized online algorithm for (h, k)-paging problem has a competitive ratio of at least $\ln \frac{k}{k-h} - \ln\ln \frac{k}{k-h} + \frac{1}{2}$ against oblivious adversary.*

Lemma 1 ((h,k)-Paging $\leq_{\mathrm{AD\_ON}}$ *RAP*). *There is (α, β) reduction with $\alpha = \beta = 1$ from the (h,k)-Paging problem with algorithm using cache of size $k = \frac{1}{\epsilon}$, adversary using cache of size $h = k - 1$, and total number of pages needed to be served $k + 1$ to RAP.*

The reduction is trivial. The $k+1$ pages are mapped each to one user of RAP instance. The RAP algorithm is given k resources while the RAP adversary will use h resources. Each request generated for page i at time t is mapped to $r_{i,t} = 1$ request to the RAP algorithm. If the interval assigned to user j by the RAP algorithm is at least 1 then the j-th page will be in the cache, else it will be left out.

Lemma 1, Theorem 6, and the the standard paging lower bound imply the following results.

Theorem 7. *Any online algorithm using* $(1+\epsilon)$ *resources for the RAP problem has competitive ratio* $\Omega(\log(\frac{1}{\epsilon}))$ *against an oblivious adversary using resource pool of size* 1.

Theorem 8. *Any online algorithm for the RAP problem using* $(1+\epsilon)$ *resources has competitive ratio* $\Omega(\frac{1}{\epsilon})$ *against an adaptive online adversary using resource pool of size* 1.

6 Future work

We obtained $(s, O(\log_s n))$-competitive algorithms for the RAP, RRAP, and TMAV problems and proved that the extra resource sT granted to the algorithm is vital. Our intuition is that the upper bound proved in Sect. 2 is tight, however we have not been able to prove a matching lower bound.

There are two new issues that we have raised. Although online reductions were used before our work (See [AL04]) as a general technique for obtaining new online algorithms from algorithms for other problems and we used it in similar fashion in Sect. 3, to our knowledge our work is the first that uses the notion of reduction between online algorithms to prove lower bounds on competitive ratio and relate hardness of one problem to that of another (Sect. 4, 5). Hence we believe it will be interesting to further study the relations among other online problems and potentially derive new algorithms and lower bounds using reductionist approach.

The second issue is that traditionally online algorithms have known the past history and were oblivious to the future. In this paper we study memoryless algorithms that not only were unaware of the past when making current decision, but also *did not know the current demands exactly.* Their knowledge of the current request is limited and in the process they only learn an upper bound approximation of the request. It will be interesting to know whether other online problems can have similar online solutions.

References

[AL04] Yossi Azar and Arik Litichevskey. Maximizing throughput in multi-queue switches. In *ESA*, pages 53–64, September 14-17 2004.

[BEY98] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

[cKA02] Ugur Çetintemel, Peter J. Keleher, and Yanif Ahmad. Exploiting precision vs. efficiency tradeoffs in symmetric replication environments. In *PODC*, page 128, 2002.

[CYV04] Badrish Chandramouli, Jun Yang, and Amin Vahdat. Distributed network querying with bounded approximate caching. *Technical Report, Department of Computer Science, Duke University*, June 2004.

[Edm00] Jeff Edmonds. Scheduling in the dark. *Theor. Comput. Sci.*, 235(1):109–141, 2000.

[IRSD99] Sotiris Ioannidis, Umit Rencuzogullari, Robert Stets, and Sandhya Dwarkadas. Craul: Compiler and run-time integration for adaptation under load. *Scientific Programming*, 7(3-4):261–273, 1999.

[KKD02] Lzimikant V. Kalé, Sameer Kumar, and Jayant DeSouza. A malleable-job system for timeshared parallel machines. In *CCGRID*. IEEE Computer Society, 2002.

[LSZM04] Ting Liu, Christopher M. Sadler, Pei Zhang, and Margaret Martonosi. Implementing software on resource-constrained mobile sensors: Experiences with impala and zebranet. In *MobiSys*, 2004.

[OLW01] Chris Olston, Boon Thau Loo, and Jennifer Widom. Adaptive precision setting for cached approximate values. In *SIGMOD Conference*, pages 355–366, 2001.

[PL95] Jim Pruyne and Miron Livny. Parallel processing on dynamic resources with carmi. In *JSSPP*, pages 259–278, 1995.

[RS89] Prabhakar Raghavan and Marc Snir. Memory versus randomization in on-line algorithms (extended abstract). In *ICALP*, pages 687–703, 1989.

[You91] Neal E. Young. On-line caching as cache size varies. In *SODA*, pages 241–250, 1991.

[You94] Neal E. Young. The k-server dual and loose competitiveness for paging. *Algorithmica*, 11(6):525–541, 1994.