The Online Set Aggregation Problem

Rodrigo A. Carrasco[1⋆], Kirk Pruhs[2⋆⋆], Cliff Stein[3⋆⋆⋆], and José Verschae[4†]

[1] Facultad de Ingeniería y Ciencias, Universidad Adolfo Ibáñez, Santiago, Chile, rax@uai.cl
[2] Department of Computer Science, University of Pittsburgh, Pittsburgh, PA, USA, kirk@cs.pitt.edu
[3] Department of Industrial Engineering and Operations Research, Columbia University, New York, NY, USA, cliff@ieor.columbia.edu
[4] Facultad de Matemáticas & Escuela de Ingeniería, Pontificia Universidad Católica de Chile, Santiago, Chile, jverschae@uc.cl

Abstract. We introduce the online *Set Aggregation Problem*, which is a natural generalization of the *Multi-Level Aggregation Problem*, which in turn generalizes the *TCP Acknowledgment Problem* and the *Joint Replenishment Problem*. We give a deterministic online algorithm, and show that its competitive ratio is logarithmic in the number of requests. We also give a matching lower bound on the competitive ratio of any randomized online algorithm.

Keywords: online algorithms, competitive analysis, set aggregation, multilevel aggregation

1 Introduction

Problem Statement: We introduce an online problem, which we call the *Set Aggregation Problem*. In this problem, a sequence R of requests arrives over time. We assume time is continuous. Each request $\rho \in R$ has an associated release time r_ρ when it arrives, and an associated waiting cost function $w_\rho(t)$ that specifies the waiting cost for this request if it is first serviced at time t. We assume that the online algorithm learns $w_\rho(t)$ at time r_ρ. We also assume that each $w_\rho(t)$ is non-decreasing, left continuous, and $\lim_{t\to\infty} w_\rho(t) = \infty$.

At any time t, the online algorithm can decide to service any subset S of the previously released requests. Thus, a schedule for this instance is a sequence $(S_1, t_1), (S_2, t_2), \ldots, (S_k, t_k)$, where the S_i's are sets of requests and the t_i's are the times that these sets were serviced. We will implicitly restrict our attention to feasible schedules, which are those for which every request is serviced, i.e. for all $\rho \in R$, exists an (S_i, t_i) in the resulting sequence where $\rho \in S_i$ and $t_i \geq r_\rho$.

⋆ Supported in part by Fondecyt Project Nr. 1151098.
⋆⋆ Supported in part by NSF grants CCF-1421508 and CCF-1535755, and an IBM Faculty Award.
⋆⋆⋆ Supported in part by NSF grant CCF-1421161.
† Supported in part by Nucleo Milenio Información y Coordinación en Redes ICM/FIC CODIGO RC130003, and Fondecyt Project Nr. 11140579.

We assume that there is a time-invariant service cost function $\mathrm{C}(S)$ that specifies the cost for servicing a set S of the requests. Without any real loss of generality, we will generally assume that $\mathrm{C}(S)$ is monotone in the sense that adding requests to S can not decrease $\mathrm{C}(S)$. We will postpone the discussion on how the online algorithm learns $\mathrm{C}(S)$ until we state our results, but essentially our lower bound holds even if the online algorithm knows $\mathrm{C}(S)$ a priori, and our upper bound holds even if $\mathrm{C}(S)$ is only known for subsets S of released requests.

The online algorithm then incurs in two types of costs, a waiting cost and a service cost. At time t_i the algorithm incurs a waiting cost of $\mathrm{W}_t(S_i) = \sum_{\rho \in S_i} w_\rho(t)$ for servicing set S_i, which is just the aggregate waiting cost of the serviced requests. At the same time t_i the algorithm also incurs in a service cost of $\mathrm{C}(S_i)$, and the total service cost the algorithm incurs is $\sum_{i=1}^{k} \mathrm{C}(S_i)$. The total cost associated with schedule $(S_1, t_1), (S_2, t_2), \ldots, (S_k, t_k)$ is therefore $\sum_{i=1}^{k} \mathrm{C}(S_i) + \sum_{i=1}^{k} \mathrm{W}_t(S_i)$, and the objective is to find the schedule that minimizes the total cost.

In the deadline version of the set aggregation problem, the waiting cost of each request is zero until a specified deadline for that request, after which the waiting cost becomes infinite.

Most Relevant Background: Our main motivation for introducing the Set Aggregation Problem is that it is a natural generalization of the *Multi-Level Aggregation Problem* (MLAP), which was introduced by [1]. In the MLAP, the requests are vertices in an a priori known tree T, and the edges (and/or vertices) of T have associated costs. Further, a set S of requests has service cost equal to the minimum cost subtree of T that contains the root of T and contains all of the requests in S. The motivation for [1] to introduce the MLAP is that it generalizes both the well-known TCP Acknowledgment Problem (TCPAP), and the well-known Joint Replenishment Problem (JRP), both of which correspond to special cases of the MLAP in which the tree T has constant height. In [1] the authors gave an online algorithm for the Multi-Level Aggregation Problem, and showed that the competitive ratio of the algorithm is $O(d^4 2^d)$, where d is the height of T. It is an open question whether constant competitiveness is achievable in the MLAP, and only a constant lower bound is known. In [2] the authors gave an $O(d)$-approximation for the deadline version of the problem.

Our Results: The Set Aggregation Problem allows us to study how critical is the restriction that service costs come from an underlying tree. More specifically it is natural to ask:

- Can constant competitiveness be achieved if there are no restrictions on service costs (other than monotonicity)?
- And if not, what is the best achievable competitive ratio?

In this paper we answer these two questions. We first show, in Section 2, that the competitive ratio of every algorithm (deterministic or randomized) is at least logarithmic in the number of requests for the deadline version of the set aggregation problem. This lower bound holds against an online algorithm that a priori knows the domain $\mathcal{R}$ of all possible requests that might arrive, and the

service cost $C(S)$ for each possible subset S of $\mathcal{R}$. Thus, we can conclude that if a constant competitive algorithm exists for the Multi-Level Aggregation Problem, then the analysis must use the fact that the service costs can not be arbitrary.

Intuitively, the requests in our lower bound instance are each associated with a node in a full binary tree T. There are $n/2^d$ requests associated with a node of depth d in the tree. The lifetime of the requests inherit the same laminar structure of the tree– the lifetime of the n requests associated with the root of T is $[0, n]$, the lifetime of the $n/2$ requests associated with the left child of the root is $[0, n/2]$, and the lifetime of the $n/2$ requests associated with the right child of the root is $[n/2, n]$. The set costs are defined so that there is clearly no benefit to include more than one request associated with each node in any serviced set. Then the requests in the subtree rooted at the left child of the root aggregate with all requests associated with the root, but the only requests associated with the root that aggregate with requests associated with the right child of the root are those that the algorithm serviced during the first half of these request's lifetime. So the algorithm incurs an unnecessary incremental cost for each set serviced during $[n/2, n]$. The instance recursively applies this same idea lower down in T.

To complement this lower bound, in Section 3 we give a deterministic online algorithm RetrospectiveCover for the set aggregation, and show the competitive ratio of this algorithm is logarithmic in the number of requests. The algorithm only needs to know the service cost $C(S)$ for subsets S of requests that are released but unserviced.

Let us give a brief (necessarily simplified) overview of, and intuition for, the RetrospectiveCover algorithm. Define a proactive schedule to be one in which the total waiting cost of every serviced set is at most its service cost. The algorithm maintains a lower bound $\mathrm{LB}(t)$ for the least possible service cost incurred by any proactive schedule up until the current time t. Let u be a time where $\mathrm{LB}(u) = 2^k$ and let v be the future time where $\mathrm{LB}(v) = 2^k$, assuming no more requests are released. Intuitively, at time u the sets in the proactive schedule associated with $\mathrm{LB}(v)$ are serviced, and then a recursive call is made to handle requests that arrive until the time w when $\mathrm{LB}(w) = 2^{k+1}$. Note that due to the release of new requests it may be that w is earlier than v.

Computing the state of the optimum at the current time, and moving to that state, is a classic technique in online algorithms, and is often called the Retrospective algorithm [3]. For example, the Retrospective algorithm is optimally competitive for the online metric matching problem [4, 5]. Intuitively the RetrospectiveCover algorithm is a generalization of the Retrospective algorithm, that computes the state of optimal at many carefully-selected times, for many different carefully-selected sub-instances, and then moves to a state that somehow covers/combines all of these optimal states. This RetrospectiveCover algorithmic design technique seems relatively general, and at least plausibly applicable to other problems.

Within the context of the Multi-Level Aggregation problem, our upper bound on the achievable competitive ratio is incomparable to the upper bound obtained in [1]. The upper bound obtained in [1] is better if $d^r 2^d$ is asymptotically less

than the logarithm of the number of requests, otherwise the upper bound that we obtain is better. As a caveat, computing the lower bound used by our algorithm is definitely NP-hard, and its not clear to us how to even obtain a polynomial-time offline $O(\log |R|)$-approximation algorithm. Techniques used in the prior literature on offline algorithms for TCPAP and JRP do not seem to be applicable.
Further Background: As mentioned above, the Set Aggregation Problem generalizes the TCPAP and the JRP. More generally, we can think of the case of the Set Aggregation Problem where the set relationship forms a tree. More precisely, if we include an edge for every pair of sets S_1 and S_2, where $S_1 \subset S_2$ and there is no set S_3 such that $S_1 \subset S_3 \subset S_2$, the family of sets S_i is laminar and the resulting graph is a tree. In [6], this problem is referred to as the Multi-Level Aggregation Problem. If the tree is of height one (a root and leaves), we obtain the TCPAP. The offline version of the TCPAP on n requests can be solved exactly in $O(n \log n)$ time [7]. The best deterministic approximation has competitive ratio 2 [8], and the best randomized is $e/(e-1)$ [9]. We note that the TCPAP is equivalent to the Lot Sizing Problem that has been studied in the operations research literature since the 1950s.

If the tree has two levels, we obtain the Joint Replenishment Problem. The best offline approximation is 1.791 [10] and the best competitive ratio is 3, via a primal dual-algorithm [11]. There is also a deadline version of the JRP where there is no cost for waiting but each request must be satisfied before its deadline. This problem is a special case of a general cost function and has an approximation ratio of 1.574 [12] and online competitive ratio of 2 [10].

For general trees of height D, we obtain the Multi-Level Aggregation Problem. This more general problem has several applications in computing, including protocols for aggregating control messages [13, 14], energy efficient data aggregation and fusion in sensor networks [15, 16], and message aggregation in organizational hierarchies [17]. There are also applications in lot sizing problems [18–20]. For the deadline version of the MLAP, there is an offline 2-approximation algorithm [21]. In unpublished work, Pedrosa [22] showed how to adapt an algorithm of Levi et al. [23] for the Multistage Assembly Problem to obtain a $2+\epsilon$ approximation algorithm for the MLAP with general waiting cost functions. For the online case, there is no constant competitive algorithm known. In [1], the authors give a $O(d^4 2^d)$ competitive algorithm for trees of height d, and a somewhat better bound of $O(d^2 2^d)$ for the deadline version. Their algorithms use a reduction from general trees to trees of exponentially decreasing weights as one goes down the tree, and therefore relies heavily on the tree structure. Building on this, [2] give an $O(d)$-competitive algorithm for the deadline version.

2 The Lower Bound

We prove a $\Omega(\log |R|)$ lower bound on the competitive ratio of any deterministic online algorithm for the deadline version of the Set Aggregation Problem.

Instance Construction Conceptually the requests are partitioned such that each request is associated with a node in a full n-node binary tree T. The root of T has depth 0 and height $\lg n$. A leaf of T has depth $\lg n$ and height 0. A node

of height h in T will have 2^h requests associated with it. Thus a leaf in T has one associated request, and the root of T has n requests associated with it. Hence there are n requests per level and $n(1 + \lg n)$ in total. The lifetime of a request associated with the k^{th} node at depth d (so $k \in \{1, 2, \ldots, 2^d\}$) in the tree is $[(k-1)n/2^d, kn/2^d)$. So the lifetime of all requests associated with the same node are the same, the lifetimes of the requests inherit the same laminar structure from T, and the lifetimes of the requests associated with nodes at a particular level of the tree cover the time interval $[0, n)$. Let R_x be the requests associated with a node x in T, and T_x be the requests associated with nodes in the subtree rooted at x in T. We say a collection S of requests is *sparse* if it does not contain two requests associated with nodes of the same depth in T.

We now proceed to describe the service costs. To do so we will split each set R_x into two sets of equal cardinality by defining a set $U_x \subseteq R_x$ with $|U_x| = |R_x|/2$. The specific requests that belong to U_x will be decided online by the adversary depending on partial outputs of the algorithm. Let us consider a set of requests S. If S is ever serviced by any algorithm at a given point in time $t \in (0, n)$, the laminar structure of the lifetimes implies that the requests correspond to a subset of nodes in a path P of T with endpoints at the root and some leaf v of T, where the lifetime of v contains t. Otherwise, we can define the cost of S arbitrarily, e.g., as ∞. We first define the cost of a set S that is sparse. The cost of S is defined inductively by walking up the path P. We say that a request r *aggregates* with a set S if the service cost of $S \cup \{r\}$ is the same as the service cost of S. If r does not aggregate with S then the service cost of $S \cup \{r\}$ is the service cost of S plus one. Finally, we say that r is *not compatible* with S if the cost of $S \cup \{r\}$ is $+\infty$. If S is a sparse set of requests in R_x where x is a leaf, then $|S| = 1$ and its service cost is also one. Consider now a sparse set $S \neq \emptyset$ with requests belonging to T_y for some y. Let x be the parent of y and r a request corresponding to x. Note that x is not a node associated to the requests in S. We have three cases:

- If $S \cap R_y = \emptyset$, then r is not compatible with S.
- Otherwise if y is a left-child of x, then r always aggregates with S.
- Otherwise if y is a right-child of x, then r aggregates with S when $r \notin U_x$, and it does not aggregate with S if $r \in U_x$.

Notice that, using this definition inductively, we have that the cost of $S \neq \emptyset$ is either infinity or an integer between 1 and the height of y plus one (inclusive)[5]. Also, the first condition implies that for a sparse set S to have finite cost there must exists a unique path P from a leaf to node x such that exactly one request in S belongs to R_v for each node v of P.

If S is an arbitrary set of requests (always corresponding to a leaf-to-root path P), then S can be decomposed into several sparse sets. We define the cost

[5] We remark that this definition does not yield monotone service costs. However this does not cause any trouble. Indeed, if two sets $S_1 \subseteq S_2$ fulfill $C(S_2) < C(S_1)$, then the algorithm can simply serve S_2 instead of S_1 without increasing its cost and without affecting feasibility. Hence, any instance with service cost C can be turned to an equivalent instance with non-decreasing service cost $C'(S) = \min_{T \supseteq S} C(T)$.

of S as the minimum over all possible decomposition, of the sum of the costs of the sparse sets in the decomposition. In this way, any solution can be converted to a solution of the same cost where each serviced set is sparse. Hence, we can restrict ourselves to consider only sparse sets.

Adversarial Strategy We now explain how to choose the requests in U_x for each $x \in T$. Let (a_x, c_x) be the lifetime of the requests associated with x, and b_x the midpoint of (a_x, c_x). Let the left and right children of x be $\ell(x)$ and $r(x)$, respectively. Let S_x be the requests in R_x that the online algorithm has serviced by time b_x. If $|S_x| \leq |R_x|/2$ then let U_x be an arbitrary subset of $|R_x|/2$ requests from $R_x - S_x$. If $|S_x| > |R_x|/2$ then let U_x contain all the requests in $R_x - S_x$ plus an arbitrary set of $|S_x| - |R_x|/2$ requests from S_x.

To see that this is a valid adversarial strategy, consider a node x in T, and let P be the path in T from the root to x, and $u_1, \ldots, u_k$ be the nodes in P whose right child is also in P. The requests associated with ancestors of x in T that requests in R_x will not aggregate with are affected by the sets $U_{u_1}, \ldots U_{u_k}$, which are all known by time a_x.

Competitiveness Analysis It is not hard to see that the optimum has cost at most n. We can construct a solution where each serviced set is sparse as follows. For each node x in T, any one request in U_x is serviced at each time in (a_x, b_x), and any one request from $R_x - U_x$ is serviced at each time in (b_x, c_x). We can now construct a solution in which each serviced set S corresponds to each different leaf-to-root path, where each node in the path corresponds to exactly one request in S. Each set S can me made to have a service cost of one. Since there are n leaves, thus n leaf-to-root paths, the constructed solution will have cost n.

Now consider the cost to the online algorithm for requests in R_x. We say that the *incremental cost at* x is how much more the online algorithm would pay for servicing the requests associated to nodes in T_x than it would pay for servicing requests in $T_x - R_x$ if requests in R_x were deleted from any (sparse) set S serviced by the online algorithm. First consider the case that $|S_x| \leq |R_x|/2$. In this case at time b_x the online algorithm has $|R_x|/2$ unserviced requests in U_x that do not aggregate with any requests in $T_{r(x)}$. Each such request can be only serviced together with a set $S \subseteq T_{r(x)}$, and thus each request in U_x imply an increase of 1 in the cost. Then the incremental cost at x is at least $|R_x|/2$.

Now let us assume that $|S_x| > |R_x|/2$. Consider requests in S_x, which are serviced during the time period (a_x, b_x). Indeed, a request in S_x can be compatible with at most $|R_{\ell(x)}| = |R_x|/2$ many sparse sets within $R_{\ell(x)}$ (one for each request in $R_{\ell(x)}$). Hence, in this case the online algorithm incurs an incremental cost of at least $|S_x| - |R_x|/2$ for the requests in S_x. Request in $R_x - S_x = U_x$ all do not aggregate (or are incompatible) with sparse sets in $T_{r(x)}$, and hence the incur a cost of $|R_x| - |S_x|$. Thus in both cases the total the incremental cost at x is at least $|R_x|/2$. As there are $n \log n$ requests in total, the online algorithms pays at least $(n \log n)/2$. Thus we have shown a lower bound of $\Omega(\log n)$ on the competitive ratio.

Note that one can apply Yao's technique, where the identity of the requests in U_x are selected uniformly at random from the requests in R_x, to get an $\Omega(\log n)$ lower bound on the competitive ratio of any randomized online algorithm.

Theorem 1. *Any randomized online algorithm for the deadline version of the online set aggregation problem is $\Omega(\log(|R|))$-competitive.*

3 The Upper Bound

In Subsection 3.1 we define a lower bound on the optimum used within the online algorithm, and observe some relatively straightforward properties of this lower bound. In Subsection 3.2 we state the online algorithm. In Subsection 3.3 we show that the online algorithm is $O(\log |R|)$ competitive.

3.1 Lower Bound on the Optimal Solution

We will simplify our lower bound and algorithm by restricting our attention to sets whose waiting cost is at most their service cost. By doing so, we will be able to focus only on service cost.

Definition 1. *We say that a set S of requests is* violated *at time t in a schedule if time t $\mathrm{W}_t(S) > C(S)$. A feasible schedule is* proactive *if it does not contain any violated set.*

We will also use in our algorithm a lower bound on the service cost of schedules for subsets of the input in subintervals of time. As mentioned in Section 1, computing the lower bound is NP-hard, but it is critical to guide our online algorithm.

Definition 2. *The lower bound $LB^-(s,t,d)$ is the minimum cost over all proactive schedules $\mathcal{Z}$ for the requests released in the time interval (s,t), of the total service cost incurred in $\mathcal{Z}$ during the time period (s,d). Let $LB^+(s,t,d)$ be the minimum over all proactive schedules $\mathcal{Z}$ for the requests released in the time interval (s,t), of the total service cost incurred in $\mathcal{Z}$ during the time period $(s,d]$. Polymorphically we will also use $LB^-(s,t,d)$ (resp. $LB^+(s,t,d)$) to denote the sets serviced within (s,d) (resp. $(s,d]$) by the proactive schedules that attains the minimum.*

The difference between $\mathrm{LB}^-(s,t,d)$ and $\mathrm{LB}^+(s,t,d)$ is that service cost incurred at time d are included in $\mathrm{LB}^+(s,t,d)$, but not in $\mathrm{LB}^-(s,t,d)$. Notice that the values of $\mathrm{LB}^-(s,t,d)$ and $\mathrm{LB}^+(s,t,d)$ do not depend on future requests, and thus can be computed at time t by an online algorithm.

Lemma 1. *There exists a proactive schedule whose objective value is at most twice optimal.*

Lemma 2. *The value $LB^-(s,t,d)$ and $LB^+(s,t,d)$ are monotone on the set of requests, that is, adding more requests between times s and t can not decrease either. Moreover, $LB^-(s,t,d)$ and $LB^+(s,t,d)$ are non-decreasing as a function of t and as a function of d, for any $s \le t \le d$.*

The proof for Lemmas 1 and 2 are in Section A.

3.2 Algorithm Design

We now give our algorithm RetrospectiveCover, which is executed at each time t. Although we understand that this is nonstandard, we believe that most intuitive way to conceptualize our algorithm is to think of the algorithm as executing several concurrent processes. The active processes are numbered $1, 2, \ldots, a$. Each process i maintains a start time $s[i]$. A process i reaches a new milestone at time t if the value of $\text{LB}^+(s[i], t, t)$ is at least $2 \cdot \text{LB}^+(s[i], m[i], m[i])$, where $m[i]$ is the time of the last milestone for process i. When a milestone for process i is reached at time t, each higher numbered process ℓ services the sets in $\text{LB}^-(s[\ell], t, t)$, and then terminates. Process i then services the sets in $\text{LB}^-(s[i], t, d[i])$, where $d[i]$ is the earliest time after t where $\text{LB}^+(s[i], t, d[i]) \geq 2 \cdot \text{LB}^+(s[i], t, t)$. Process i then starts a process $i + 1$ with start time of t.

We give pseudocode for our algorithm RetrospectiveCover to explain how various corner cases are handled, and to aide readers who don't want to think about the algorithm in terms of concurrent processes. RetrospectiveCover uses a subroutine CheckMilestone that checks, for a process i, whether it has reached a milestone at the current time t. RetrospectiveCover is initialized by setting $a = 1$, $s[1] = 0$ and $m[1] = s[1]$.

CheckMilestone(i, t)

```
1  if m[i] = s[i] then
2  // Process i has not seen its first milestone yet
3       if LB+(s[i], t, t) > 0 then
4            return true
5       else return false
6  else
7       if LB+(s[i], t, t) ≥ 2 · LB+(s[i], m[i], m[i]) then
8            return true
9       else return false
```

RetrospectiveCover(t)

```
1  for i = 1 to a
2       if CheckMilestone(i, t) then
3            for ℓ = a downto i + 1
4                 if any request has arrived since time m[ℓ] then
5                      service every set in LB−(s[ℓ], t, t)
                       // Intuitively process ℓ now terminates
6            // The analysis will prove in an inductive invariant on the state
             at this point
7            Let d[i] be the earliest time after t where
             LB+(s[i], t, d[i]) ≥ 2 · LB+(s[i], t, t)
               if such a time exists, and d[i] is infinite otherwise.
8            Service the sets in LB−(s[i], t, d[i])
9            m[i] = t
10           a = i + 1; s[a] = t; m[a] = s[a]; return
```

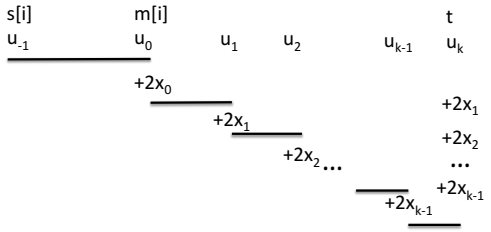

Fig. 1. An illustration of the generic situation, and when costs are incurred by RetrospectiveCover. The terms in the diagonal depict the cost incurred in line 8 and the terms on the right terms in line 5.

3.3 Algorithm Analysis

The first part of this subsection is devoted to proving Lemma 4, which is the key lemma, and states that the service cost incurred by RetrospectiveCover is within a logarithmic factor of optimal. Next, we show how this readily implies that RetrospectiveCover is $O(\log |R|)$-competitive.

We first make some simplifying assumptions, and then prove one simple property of RetrospectiveCover. Without loss of generality, we can assume that no two requests are released at the same time, and that no requests are released exactly at any milestone (one can sequentialize the releases arbitrarily followed by the unique milestone). Similarly, without loss of generality we can assume that each waiting function $w_\rho(t)$ is a continuous function of t. Finally, we can assume without loss of generality that the initial waiting time is 0, that is $w_\rho(r_\rho) = 0$. With this we can state the following Lemma, which is proved in Section A.

Lemma 3. *Assume that* RetrospectiveCover *just computed a new deadline $d[i]$ in line 7. Then process i will either reach a new milestone or be terminated by time $d[i]$.*

The Key Induction Argument

Lemma 4. *Let $c = 6$. Consider a point of time when* RetrospectiveCover *is at line 6. Let Y be the number of requests that arrive during the time interval $(s[i], t)$. Then the service cost incurred by* RetrospectiveCover *up until this point is at most $c(\lg Y)LB^+(s[i], t, t)$.*

The rest of this subsubsection is devoted to proving Lemma 4 by induction on the number of times that line 6 in RetrospectiveCover is invoked. So consider an arbitrary time that RetrospectiveCover is at line 6. We now need to introduce some more notation. (See Figure 1 for an illustration of the various concepts and notation). Let $u_{-1} = s[i]$, $u_0 = m[i]$, y_0 be the number of requests that arrive during (s, u_0), and $x_0 = \mathrm{LB}^+(s[i], m[i], m[i])$. Let $k = a - i$. Then for $j \in \{1, 2, \ldots, k\}$, let $u_{j-1} = s[i+j] = m[i+j-1]$, $u_k = t$, let y_j be the number of requests that arrive during (u_{j-1}, u_j), and $x_j = \mathrm{LB}^+(u_{j-1}, u_j, u_j)$. Also, for notational convenience let $z = \mathrm{LB}^+(s[i], t, t)$.

Before making our inductive argument, we need to detour slightly. Lemma 5 gives an inequality on service costs that will be useful in our analysis, and the proof is given in Section A.

Lemma 5. *It holds that* $\sum_{j=1}^{k-1} x_j \leq x_0$.

We are now ready to make our inductive argument. We start with the base case. The first milestone occurs at the first time t where $\mathrm{LB}^+(0,t,t)$ is positive. When this occurs, RetrospectiveCover has incurred no service cost until this point because $\mathrm{LB}^-(0,t,t) = 0$. Note that if $Y = 1$ or $k = 0$, then again RetrospectiveCover has incurred no service cost until this point because $\mathrm{LB}^-(0,t,t) = 0$. So assume from now on that $Y \geq 2$ and $k \geq 1$.

Now consider the case that $k = 1$. In this case, process i incurred a cost of at most $2x_0$ when line 8 is invoked at time u_0, and as process $i+1$ did not reach its first milestone (which would have caused process $i+2$ to start), no additional cost is incurred at time t. Thus to establish the induction, it is sufficient to show that $2x_0 + cx_0 \lg y_0 \leq cz \lg Y$.

Normalizing costs so that $x_0 = 1$, and using the fact that $z \geq 2x_0$ (since t is the next milestone), it is sufficient to show that $4y_0^c \leq Y^{2c}$. Using the fact that $y_0 \leq Y$, it is sufficient to show $4Y^c \leq Y^{2c}$. This holds as $Y \geq 2$ and $c = 6$.

Now consider the case that $k = 2$. In this case, process i incurred a cost of at most $2x_0$ when line 8 is invoked at time u_0, process $i+1$ incurs a cost of at most $2x_1$ when line 8 is invoked at time u_1 and at most $2x_1$ when line 5 is invoked at time t, and process $i+2$ incurs no cost at time t. Thus to establish the induction, it is sufficient to show that $2x_0 + 4x_1 + cx_0 \lg y_0 + cx_1 \lg y_1 \leq cz \lg Y$.

Normalizing costs so that $x_0 = 1$, and using the fact that $z \geq 2$, $x_1 \leq x_0$, and $y_0 + y_1 \leq Y$, it is sufficient to show that: $64 y_0^c y_1^c \leq (y_0 + y_1)^{2c}$, which holds by the binomial theorem as $64 \leq \binom{2c}{c} = \binom{12}{6}$.

For the remainder of the proof, we assume $k \geq 3$. At time u_j, for $0 \leq j \leq k-1$, process $i+j$ incurs a cost at most $2x_j$ when line 8 is invoked. At time t process $i+j$, for $1 \leq j \leq k-1$, incurs a cost of at most $2x_j$ when line 5 is invoked. Indeed, the algorithm pays $\mathrm{LB}^-(u_{j-1}, t, t)$, which must be less than $2\mathrm{LB}^+(u_{j-1}, u_j, u_j)$, otherwise process $i+j$ would have hit a milestone within (u_j, t). Similarly, at time t point process $i + k = a$ does not incur any cost, as otherwise process a would have hit a milestone before time t. Thus to establish the induction, we need to show that $2x_0 + 4\sum_{j=1}^{k-1} x_j + c\sum_{j=0}^{k-1} x_j \lg y_j \leq cz \lg Y$.

Note that our induction is using the fact that the cost for RetrospectiveCover during a time interval (u_{j-1}, u_j) is identical to the cost of RetrospectiveCover on the subinstance of requests that are released during (u_{j-1}, u_j), essentially because RetrospectiveCover can be viewed as a recursive algorithm. We now normalize costs so that $x_0 = 1$. Note that by Lemma 5, $\sum_{j=1}^{k-1} x_j \leq 1$, and $z \geq 2$ (or t wouldn't be the next milestone). Thus it is sufficient to show that $64 \prod_{j=0}^{k-1} y_j^{cx_j} \leq Y^{2c}$.

We now claim that the left hand side of this inequality is maximized, subject to the constraint that $\sum_{j=0}^{k-1} y_j \leq Y$, when each $y_j = x_j Y / X$, where $X = \sum_{j=0}^{k-1} x_j$ (this can be shown using the method of Lagrangian multipliers). Thus it is sufficient to show that $64 \prod_{j=0}^{k-1} (Y x_j / X)^{cx_j} \leq Y^{2c}$, which is equivalent to $64 Y^{c(X-2)} \prod_{j=0}^{k-1} (x_j)^{cx_j} \leq X^{cX}$.

Since $X \leq 2$ and $Y \geq 2$, the value $Y^{c(X-2)}$ is maximized when $Y = 2$. Thus it suffices to show that $64 \cdot 2^{c(X-2)} \prod_{j=0}^{k-1} (x_j)^{cx_j} \leq X^{cX}$. Because $x_0 = 1$, it is sufficient to show that $64 \cdot 2^{c(X-2)} \prod_{j=1}^{k-1} (x_j)^{cx_j} \leq X^{cX}$.

Using again the method of Lagrangian multipliers, we have that the maximum of the left hand side of this inequality, subject to $\sum_{j=1}^{k-1} x_j = X - 1$, is reached when all x_j are equal. Hence it suffices show that $64 \cdot 2^{c(X-2)} \left(\frac{X-1}{k-1}\right)^{cX} \leq X^{cX}$. Since $((X-1)/X)^{cX}$ is clearly between 0 and 1, it is sufficient to show that $2^{6-2c} \cdot 2^{cX} \leq (k-1)^{cX}$, which, by simple algebra, is true for $c = 6$ and when $k \geq 3$.

The Rest of the Analysis Lemma 6, which is proved in Section A, shows that because the algorithm is trying to mimic proactive schedules, it will be the case that the waiting cost for the algorithm is at most twice its service cost. Finally in Theorem 2, also proved in Section A, we conclude that these lemmas imply that our algorithm is $O(\log |R|)$-competitive.

Lemma 6. *For any set S serviced at time t in the schedule produced by the algorithm* RetrospectiveCover*, it will be the case that* $\mathrm{W}_t(S) \leq 2\,\mathrm{C}(S)$.

Theorem 2. *The* RetrospectiveCover *algorithm is $O(\log |R|)$-competitive.*

4 Conclusion

Another possible way to generalize the multilevel aggregation problem is to assume that the domain $\mathcal{R}$ of possible requests (perhaps it is useful to think of $\mathcal{R}$ as "types" of possible requests), and the service cost $C(S)$ for every subset S of $\mathcal{R}$, is known to the online algorithm a priori, and then consider the competitive ratio as a function of $|\mathcal{R}|$. Our lower bound instance shows that the optimal competitive ratio is $\Omega(\log \log |\mathcal{R}|)$. Its not immediately clear to us how to upper bound the competitiveness of our algorithm RetrospectiveCover in terms of $|\mathcal{R}|$, or how to design an algorithm where such an analysis is natural. So, determining the optimal competitive ratio as a function of $|\mathcal{R}|$ seems like a reasonable interesting open problem.

References

1. Bienkowski, M., Böhm, M., Byrka, J., Chrobak, M., Dürr, C., Folwarcznỳ, L., Jez, L., Sgall, J., Thang, N.K., Veselỳ, P.: Online algorithms for multi-level aggregation. In: European Symposium on Algorithms. (2016) 12:1–12:17
2. Buchbinder, N., Feldman, M., Naor, J.S., Talmon, O.: O(depth)-competitive algorithm for online multi-level aggregation. In: ACM-SIAM Symposium on Discrete Algorithms. (2017) 1235–1244
3. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, New York, NY, USA (1998)
4. Kalyanasundaram, B., Pruhs, K.: Online weighted matching. Journal of Algorithms **14**(3) (1993) 478–488

5. Khuller, S., Mitchell, S.G., Vazirani, V.V.: On-line algorithms for weighted bipartite matching and stable marriages. Theoretical Computer Science **127**(2) (1994) 255–267
6. Bienkowski, M., Böhm, M., Byrka, J., Chrobak, M., Dürr, C., Folwarcznỳ, L., Jeż, L., Sgall, J., Thang, N.K., Veselỳ, P.: Online algorithms for multi-level aggregation. arXiv preprint arXiv:1507.02378 (2015)
7. Aggarwal, A., Park, J.K.: Improved algorithms for economic lot sizing problems. Operations Research **41** (1993) 549–571
8. Dooly, D.R., Goldman, S.A., Scott, S.D.: On-line analysis of the TCP acknowledgment delay problem. Journal of the ACM **48**(2) (2001) 243–273
9. Karlin, A.R., Kenyon, C., Randall, D.: Dynamic TCP acknowledgement and other stories about e/(e - 1). Algorithmica **36**(3) (2003) 209–224
10. Bienkowski, M., Byrka, J., Chrobak, M., Jeż, Ł., Nogneng, D., Sgall, J.: Better approximation bounds for the joint replenishment problem. In: ACM-SIAM Symposium on Discrete Algorithms. (2014) 42–54
11. Buchbinder, N., Kimbrel, T., Levi, R., Makarychev, K., Sviridenko, M.: Online make-to-order joint replenishment model: Primal-dual competitive algorithms. In: ACM-SIAM Symposium on Discrete Algorithms. (2008) 952–961
12. Bienkowski, M., Byrka, J., Chrobak, M., Dobbs, N., Nowicki, T., Sviridenko, M., wirszcz, G., Young, N.E.: Approximation algorithms for the joint replenishment problem with deadlines. In: International Colloquium on Automata, Languages and Programming. (2013) 135–147
13. Badrinath, B., Sudame, P.: Gathercast: the design and implementation of a programmable aggregation mechanism for the internet. In: International Conference on Computer Communications and Networks. (2000) 206–213
14. Bortnikov, E., Cohen, R.: Schemes for scheduling of control messages by hierarchical protocols. In: Joint Conference of the IEEE Computer and Communications Societies. Volume 2. (1998) 865–872
15. Hu, F., Cao, X., May, C.: Optimized scheduling for data aggregation in wireless sensor networks. In: Int. Conference on Information Technology: Coding and Computing (ITCC 2005). Volume 2. (2005) 557–561
16. Yuan, W., Krishnamurthy, S., Tripathi, S.: Synchronization of multiple levels of data fusion in wireless sensor networks. In: Global Telecommunications Conference. Volume 1. (2003) 221–225
17. Papadimitriou, C.: Computational aspects of organization theory. In: European Symposium on Algorithms. (1996) 559–564
18. Crowston, W.B., Wagner, M.H.: Dynamic lot size models for multi-stage assembly systems. Management Science **20**(1) (1973) 14–21
19. Kimms, A.: Multi-level lot sizing and scheduling: Methods for capacitated, dynamic, and deterministic models. Springer-Verlag (1997)
20. Lambert, D.M., Cooper, M.C.: Issues in supply chain management. Industrial Marketing Management **29**(1) (2000) 65–83
21. Becchetti, L., Marchetti-Spaccamela, A., Vitaletti, A., Korteweg, P., Skutella, M., Stougie., L.: Latency-constrained aggregation in sensor networks. ACM Transactions on Algorithms **6**(1) (2009) 13:1–13:20
22. Pedrosa, L.L.C.: (2013) Private communication.
23. Levi, R., Roundy, R., Shmoys, D.B.: Primal-dual algorithms for deterministic inventory problems. Mathematics of Operations Research **31**(2) (2006) 267–284

A Detailed Proofs

In this section we detail some of the proofs from Section 3.

Lemma 1 There exists a proactive schedule whose objective value is at most twice optimal.

Proof. We show how to iteratively transform an arbitrary schedule $\mathcal{Z}$ into a proactive schedule in such a way that the total cost at most doubles. Let t be the next time in $\mathcal{Z}$ when there is an unserved set S, with the property that the waiting time for S, infinitesimally after t, is greater than the service cost for S. We then add the set S at time t to $\mathcal{Z}$. The service cost of this set is at most the total waiting time of the requests that it serves. Thus the total service cost of the final schedule is at most the waiting time in the original $\mathcal{Z}'$. Further the transformation can only decrease the total waiting cost, since the requests in S are being served no later than they were originally.

Lemma 2 The value $\mathrm{LB}^-(s,t,d)$ and $\mathrm{LB}^+(s,t,d)$ are monotone on the set of requests, that is, adding more requests between times s and t can not decrease either. Moreover, $\mathrm{LB}^-(s,t,d)$ and $\mathrm{LB}^+(s,t,d)$ are non-decreasing as a function of t and as a function of d, for any $s \leq t \leq d$.

Proof. Any schedule that is proactive for the larger set of requests is also proactive for the smaller set of requests, because the waiting time of the requests serviced can not be more in the smaller set of requests than in the larger set of requests. The monotonicity on t follows directly from the monotonicity on the set of requests. The monotonicity on d is clear since for any $d \leq d'$, a proactive solution up to time d' is also proactive up to time d.

Lemma 3 Assume that RetrospectiveCover just computed a new deadline $d[i]$ in line 7. Then process i will either reach a new milestone or be terminated by time $d[i]$.

Proof. If $d[i]$ is infinite, then this is obvious, so assume otherwise. If process i is terminated before time $d[i]$ then this is obvious, so assume otherwise. If no requests arrive during the time interval $(m[i], d[i])$ then process i will reach a new milestone exactly at time $d[i]$ by the definition of milestones. If requests arrive before $d[i]$ then the claim follows by the monotonicity of LB^+, see Lemma 2.

Lemma 5 It holds that $\sum_{j=1}^{k-1} x_j \leq x_0$.

Proof. First notice that $\mathrm{LB}^-(u_{-1}, t, t) \leq 2\mathrm{LB}^+(u_{-1}, u_0, u_0) = 2x_0$, since otherwise process i would have hit a milestone within the interval (u_0, t). Then, it suffices to show that $\sum_{j=0}^{k-1} x_j \leq \mathrm{LB}^-(u_{-1}, t, t)$. To show this last bound, fix $j \in \{0, \ldots, k-1\}$ and consider a proactive schedule $\mathcal{Z}$ for requests arriving in (u_{-1}, t). Within each interval (u_{j-1}, u_j), the solution is proactive when considering requests with release date in $(u_{j-1}, u_j]$, and hence the serving cost of the

requests served by $\mathcal{Z}$ within this interval is at most $\mathrm{LB}^+(u_{j-1}, u_{j-1}, u_j)$. We remark that this is also true for $j = k-1$ as $u_{k-1} < t$. Taking $\mathcal{Z}$ minimizing the service cost within (u_{-1}, t), we obtain the required bound, $\sum_{j=0}^{k-1} x_j \leq \mathrm{LB}^-(u_{-1}, t, t)$.

Lemma 6 For any set S serviced at time t in the schedule produced by the algorithm RetrospectiveCover, it will be the case that $\mathrm{W}_t(S) \leq 2\,\mathrm{C}(S)$.

Proof. Assume that a process i hit a milestone at time t. We adopt the notation from subsubsection 3.3 and illustrated in Figure 1. Additionally let U_j be the requests that are released in the time interval (u_{j-1}, u_j) for $j \in \{0, 1, \ldots, k\}$.

We now prove that the sets serviced in line 5 have waiting time at most twice the service cost. Also, we show that after serving such sets, the set $\cup_{h=k-j}^{k} U_h$ has no violated subset at time t, where $j = a - \ell$. We show this by induction on $j = a - \ell$. The base case is when $j = 0$ ($\ell = a$). There is no violated subset of U_k at time t since process $i + k = a$ did not hit a milestone before time t. Thus, no set serviced in $\mathrm{LB}^-(s[a], t, t)$ is violated. Obviously, after servicing such sets, U_k still has no violated subset. Now let us show the two properties for an arbitrary j. By induction hypothesis, the set $\cup_{h=k-(j-1)}^{k} U_h$ has no violated subset at time t. There is no violated subset of U_{k-j} at time t since the servicing of sets in $\mathrm{LB}^-(s[a-j], m[a-j], d[a-j]) = \mathrm{LB}^-(u_{k-j-1}, u_{k-j}, d[a-j])$ at time $m[a-j]$ guaranteed that U_{k-j} would not have any violated subsets until after time $d[a-j]$ and $t \leq d[a-j]$. Thus no set serviced in $\mathrm{LB}^-(s[a-j], t, t)$ in line 5 has waiting time at most twice the service cost. Further since $\mathrm{LB}^-(s[a-j], t, t) = \mathrm{LB}^-(u_{k-j-1}, t, t)$ is a proactive schedule, after serving sets in such solution the set $\cup_{h=k-j}^{k} U_h$ has no violated subset at time t.

Finally the same argument can be applied to the sets serviced in line 8 in RetrospectiveCover.

Theorem 2 The RetrospectiveCover algorithm is $O(\log |R|)$-competitive.

Proof. Applying Lemma 4 to the original process, and noting that the service cost in the final invocation of line 8 is at most twice the previous service costs, one obtains that the service cost for the algorithm is $O(\log |R|)$ times the lower bound. By Lemma 1 the lower bound is at most twice the optimal, and by Lemma 6 the waiting cost for requests serviced by the algorithm is at most the service cost of these requests. Finally the waiting cost of any unserviced requests is at most twice the service cost of the algorithm.