Optimal Online Balanced Graph Partitioning

Maciej Pacut[1], Mahmoud Parham[2*] and Stefan Schmid[1,2,3]

[1] TU Berlin, Germany.
[2] University of Vienna, Austria.
[3] Fraunhofer SIT, Germany.

*Corresponding author(s). E-mail(s): mahmoud.parham@univie.ac.at;
Contributing authors: maciej.pacut@inet.tu-berlin.de; stefan_schmid@univie.ac.at;

Abstract

Distributed applications generate a significant amount of network traffic in datacenters. By collocating nodes (e.g., virtual machines) that communicate frequently so that they reside on the same clusters (e.g., server or rack), we can reduce the network load and improve application performance. However, the communication pattern of different applications is often unknown a priori and may change over time; hence it needs to be learned online. This paper revisits the online balanced partitioning problem, introduced by Avin et al. at DISC 2016, that asks for an algorithm that strikes a trade-off between the benefits of collocation (i.e., reduced network traffic) and its costs (i.e., migrations). Our first contribution is a significantly improved deterministic lower bound of $\boldsymbol{\Omega(k \cdot \ell)}$ on the competitive ratio, where $\boldsymbol{\ell}$ is the number of clusters and $\boldsymbol{k}$ is the cluster size. The bound holds even for scenarios where the communication pattern can be perfectly partitioned so that all communications are internal to the clusters. We match this result with an asymptotically tight upper bound of $\boldsymbol{O(k \cdot \ell)}$ for this scenario. For $\boldsymbol{k = 3}$, we contribute an asymptotically tight upper bound of $\boldsymbol{\Theta(\ell)}$ for the case where the communication pattern can change arbitrarily over time. We improve the result for $\boldsymbol{k = 2}$ by providing a strictly $\boldsymbol{6}$-competitive upper bound.

Keywords: Distributed applications, cloud computing, online algorithms, competitive analysis

1 Introduction

Data-centric applications, from distributed machine learning to distributed databases, generate a significant amount of network traffic in datacenters [1, 2]. The performance of these distributed applications often depends on the performance of the underlying communication networks [3, 4].

The virtualization of resources in datacenters introduces an intriguing opportunity to reduce the network traffic and improve performance. In particular, it becomes possible to adaptively *migrate* frequently communicating virtual machines (or containers) closer to each other, in a demand-aware manner. Such adaptive migrations, however, come at a cost (e.g., resource and time overhead) and introduce a tradeoff.

This paper studies the algorithmic problem underlying such demand-aware optimizations, aiming to strike a balance between the benefits of migrations (e.g., reduced network load) and their costs. This is particularly challenging in a setting where the traffic can be bursty and change over time, in a hard to predict manner, as it is often the case in practice [5]. We are in the realm of *online algorithms and competitive analysis*, and ideally, the algorithm should perform closely to an optimal offline algorithm without requiring any information about future traffic demands.

1.1 Online Algorithms and Competitive Analysis

We measure the quality of the proposed solutions with competitive analysis [6], which suits well the networking problems that are online by their nature. The sequence of requests σ is revealed one-by-one, in an online fashion. Upon seeing a request, the algorithm must serve it without the knowledge of future requests.

We measure the performance of an online algorithm by comparing to the performance of an optimal offline algorithm. For a given sequence of requests σ, let $\mathrm{ALG}(\sigma)$ be the cost incurred by a deterministic online algorithm ALG, and let $\mathrm{OPT}(\sigma)$ be the cost incurred by an optimal offline algorithm OPT. In contrast to ALG that learns the requests one-by-one as it serves them, OPT has complete knowledge of the entire request sequence σ *ahead of time*. The goal is to design online algorithms that provide worst-case guarantees. In particular, ALG is said to be *α-competitive* if there is a constant β, such that for any input sequence σ it holds that

$$\mathrm{ALG}(\sigma) \leq \alpha \cdot \mathrm{OPT}(\sigma) + \beta.$$

Note that β cannot depend on input σ but can depend on other parameters of the problem, such as the number of nodes. The minimum α for which ALG is α-competitive is called the *competitive ratio* of ALG. We say that ALG is *strictly α-competitive* if additionally $\beta = 0$.

1.2 Model

The problem known as *online balanced graph repartitioning* was introduced by Avin et al. [7] at DISC 2016. A special variant of the general problem, called the *learning* model, was later introduced by Henzinger et al. [8]. We define our terminology and the problems considered in this paper as follows.

Preliminaries

We say that an assignment of nodes to clusters is a *partitioning* of nodes, and it represents the *configuration* of the algorithm. The reconfiguration or migration of nodes is a *repartitioning* operation. We say that a subset of nodes are *collocated* if they reside in the same cluster. A communication request between two nodes is *internal* if the nodes are collocated in the same cluster. Otherwise, they are in different clusters, and the request is *external*. Algorithms serve internal requests *locally* at cost 0 and external requests *remotely* at cost 1.

We refer to the graph structure of a request sequence as its *communication graph*, which contains an edge between pairs of nodes with at least one request in the sequence. A set of nodes belong to a *communicating (connected) component* if a path exists between them in the communication graph. A component is a *singleton* if it contains exactly one node, and we refer to it as an *isolated* node. The *origin* of a node is the cluster in which the node resides in the initial partitioning. For any subset of nodes C collocated in the initial partitioning, we let $I(C)$ denote their cluster of origin.

The General Partitioning Problem

We are given a set of n nodes, initially arbitrarily partitioned into $\ell \in \mathbb{N}^+$ clusters each of capacity $k = n/\ell$ nodes. The nodes interact in an online manner: we are given a sequence of pairwise communication requests $\sigma = (u_1, v_1), (u_2, v_2), (u_3, v_3), \ldots$, where a pair (u_t, v_t) indicates that nodes u_t and v_t exchange a fixed amount of data at time t. We sometimes refer to the nodes as virtual machines or processes, and we refer to the clusters as servers.

The cost of serving a request (u, v) depends on the relative positions of u and v: if they reside in the same cluster, the request costs 0, and it costs 1 otherwise. Before serving a request, an online algorithm may perform a *repartition* of nodes. That is, it may migrate any number of nodes to different clusters while ensuring the number of nodes in each cluster does not exceed k by more than ϵk for constant $\epsilon \geq 0$. We refer to the extra capacity as *resource augmentation*. We consider the problem without any augmentation for most of this paper, i.e., $\epsilon = 0$. The cost of migrating a single node is $\alpha \in \mathbb{N}^+$. The goal is to minimize the total cost of repartitions and to serve the requests. We use the terms "partition", "partitioning", and "configuration" interchangeably.

The Learning Variant

Consider a variant of the general partitioning problem, where each pair of nodes either never communicates, or they communicate indefinitely. Once a communication request (u, v) arrives, any algorithm must keep u and v collocated for the rest of the input sequence. We assume that requests of σ constitute a communication graph that admits a *perfect partitioning*: a partitioning that assigns exactly k nodes to each cluster, and no inter-cluster request ever occurs in this partitioning. Moreover, an optimal offline algorithm moves to this partitioning before serving the sequence, and it stays there permanently. The goal of the algorithm is to *learn* the communication graph (as it is revealed one edge at a time) while serving requests without performing too many node migrations. In the learning model, any two communicating nodes must be collocated, and only the migration cost is relevant; for simplicity, we may scale it down to $\alpha = 1$ (our bounds hold for any $\alpha > 1$ as well).

1.3 Related Work

Closest to our work are those of Avin et al. [7] at DISC 2016 (on the general partitioning model) and Henzinger et al. (on the learning model) [8] at SIGMETRICS 2019 and SODA 2021 [9]. Recently, a polynomial-time online algorithm achieving the same competitive ratio as in [7] has been proposed by Forner et al. [10]. However, the focus of these papers is primarily on models with resource augmentation: the online algorithm can use slightly larger clusters than the offline algorithm. Avin et al. showed that their lower bound $\Omega(k)$ holds even in a scenario with significant capacity augmentation, and they provided an algorithm with the competitive ratio $O(k \log k)$ using the $(2+\epsilon)$-augmented cluster capacity. Their ratio is independent of ℓ, which is impossible without significant resource augmentation.

In contrast, we study the non-augmented setting, where the nodes need to be perfectly balanced among the clusters. This assumption is more realistic, as it utilizes all the processing capacity of a datacenter instead of leaving some CPUs idle. This variant is significantly more challenging, as it is related to hard problems such as integer partitioning [11]. Not much is known about the setting without augmentation. For $k = 2$, Avin et al. [7] presented a 7-competitive algorithm with a substantial $(\Omega(\ell^2))$ additive constant. For $k > 3$, a $O(k^2 \cdot \ell^2)$-competitive (phase-based) algorithm was given by [7]. Later, a more sophisticated analysis by Bienkowski et al. [12] improved the ratio of the same algorithm to $O(2^{O(k)} \cdot \ell)$, which is significant for its linear dependency on ℓ. The best known lower bound for the problem without augmentation is $\Omega(k)$ [7]. Given our lower bound of $\Omega(k \cdot \ell)$ in this paper, the quest for closing the gap remains open. See Table 1 for known results.

The problem has also been studied in a weaker model where the adversary can only sample requests from a fixed distribution [13] over the edges of a ring communication graph.

The *static* offline version of the partitioning problem is known as the *ℓ-balanced graph partitioning problem*, where the entire communication graph is known in advance, and the task is to partition n nodes into ℓ clusters of capacity n/ℓ each, minimizing the number of inter-cluster edges, The problem is NP-complete, and cannot even be approximated within any finite factor

Variant	Lower bound	Upper bound
Learning, $k \geq 3$	$\Omega(k\ell), \epsilon = 0$ (Thm. 1)	$O(k\ell), \epsilon = 0$ (Thm. 3)
Learning, $k \geq 3$	$\Omega(\ell \log k), \epsilon \leq \frac{1}{32}$ [9]	$O(\ell \log k), \epsilon \in \Omega(1)$ [9]
Learning, $k \geq 3$	$\Omega(\ell), \epsilon < 1/3$ (Thm. 2)	$O(\log k), \epsilon > 1$ [9]
General, $k = 3$	$\Omega(\ell), \epsilon = 0$ (Thm. 4)	$O(\ell), \epsilon = 0$ (Thm. 6)
General, $k = 2$	$3, \epsilon = 0$ [7]	$6, \epsilon = 0$ (Thm. 7)
General, $k > 3$	$\Omega(k\ell), \epsilon = 0$ (Thm. 4)	$O(2^{O(k)}\ell), \epsilon = 0$ [12]

Table 1: Overview of our contributions and known results on the deterministic online partitioning problem.

unless P=NP [14]. The static variant where $\ell = 2$ corresponds to the minimum bisection problem, which is already NP-hard [15], and currently the best approximation ratio is $O(\log n)$ [16–21].

The studied problem is further related to some classic online problems. In particular, it is related to online paging [22–25], sometimes also referred to as online caching, where requests for data items (nodes) arrive over time and need to be served from a cache of finite capacity, and where the number of cache misses must be minimized. Classic problem variants usually boil down to finding a smart eviction strategy, such as Least Recently Used (LRU) [22]. In our setting, requests can be served remotely (i.e., without fetching the corresponding nodes to a single physical machine). In this light, our model is more reminiscent of caching models *with bypassing* [26–28]. A major difference between these problems is that in the caching problems, each request involves a single element of the universe, while in our model, *both* endpoints of a communication request are subject to optimization. In this light, we can see our model as a "symmetric" version of online paging. The general problem additionally generalizes symmetric ski rental [29].

Graph partitioning and clustering problems are fundamental in computer science and arise in multiple contexts, see [30, 31].

1.4 Contributions

This paper presents several new results for the online graph partitioning problem. Table 1 provides an overview of our contributions compared to prior work.

For both the learning model and the general model, we obtain a lower bound of $\Omega(k \cdot \ell)$ on the competitive ratio of any online deterministic online algorithm (that also holds in the general partitioning model). This improves over the best known lower bound $\Omega(k)$ [7] that holds only in the general partitioning model. The generalized lower bound $\Omega(k \cdot \ell)$ for the learning model holds in the general model as well. We further adjust the lower bounds to show that the factor of $\Omega(\ell)$ is unavoidable even with a significant augmentation.

We complement these result with an asymptotically optimal, $O(k \cdot \ell)$-competitive algorithm for the learning model. For the general partitioning model and $k = 3$, we design an asymptotically optimal $O(\ell)$-competitive algorithm (after the conference version of this paper, the upper bound was generalized to arbitrary ℓ by Bienkowski et. al. [12], yielding an $O(2^{O(k)} \cdot \ell)$-competitive algorithm). We further present a strictly 6-competitive algorithm for $k = 2$ that improves upon the previous 7-competitive algorithm with $O(\alpha\ell^2)$ additive constant. All algorithms in this paper have a strict competitive ratio (cf. Section 1.1), which improves over previous results with substantial additive in terms of $(\alpha \cdot k \cdot \ell)^2$.

2 The Learning Model

We begin with the learning variant of online balanced graph partitioning problem. First, we show a surprisingly high lower bound of $\Omega(k \cdot \ell)$ against deterministic algorithms for $k \geq 3$. The lower bound holds also in the general partitioning model (see Theorem 4 for details). Second, we provide a deterministic algorithm that asymptotically matches this lower bound for the learning model.

2.1 Lower Bound

Overview of the construction. At each time step, we issue a new request, depending on the configuration of the online algorithm. First, we issue requests between $k-1$ nodes of some arbitrarily chosen cluster, and we refer to this communicating component as B. In any partitioning, the communicating component B is collocated with exactly one isolated node, called the *pivot*. Second, we issue a request between the pivot and an arbitrarily chosen node from a different cluster. Any algorithm must collocate these nodes (recall that we consider a learning variant), and it must place them in a different cluster than B's (otherwise, its capacity would be violated). Note that after collocating them, another isolated node must the take place of the pivot. Third, we issue the request between the new pivot and an arbitrarily chosen node from the same cluster of origin as the pivot. We repeat the last step of the construction $\Theta(k \cdot \ell)$ times (exact condition to be determined), using the node isolated node collocated with B as a new pivot. The algorithm pays 1 per each such request.

We claim that this sequence is cheap to serve offline. Roughly, if an offline algorithm would reside in the initial configuration, only the requests between x_0 and y_0 would be external, and all the subsequent requests would be free. Consider an offline strategy that collocates x_0, y_0 by swapping them with some initially collocated nodes x^*, y^* that did not participate in any request. To make sure that such a pair always exists, we stop repeating the requests concerning the pivot while there are still two isolated nodes collocated on some server. We illustrate the construction in Figure 1.

Theorem 1 *Any deterministic online algorithm for the learning model of online balanced graph partitioning and $k \geq 3$ has the competitive ratio of at least $(k-2)(\ell-1)/2 - 2$.*

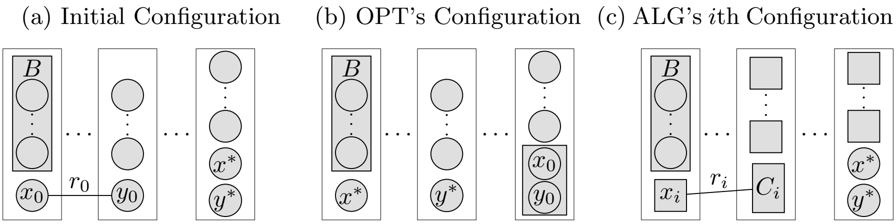

Fig. 1: Large rectangles represent clusters. Nodes are shown in gray circles, and gray rectangles represent components. Both ALG and OPT start in the configuration (a). OPT performs only two swaps and ends up at the configuration (b). At the beginning of the ith iteration, ALG is in the configuration (c) before evicting the ith pivot node x_i from the cluster of component B.

Proof Fix any online algorithm ALG. Initially, all nodes are isolated, i.e., each node is in a singleton communicating component. We issue requests one-by-one, in reaction to ALG's choices.

1. **Component B.** We issue requests among $k-1$ nodes in an arbitrary cluster, and we refer to these nodes as a communicating component B. In any feasible partition, a single isolated node must be collocated with B (each cluster hosts exactly k nodes). We refer to the isolated node collocated with B at any time as the *pivot* node. Let x_0 denote the first pivot node.
2. **Request between nodes originating from different clusters.** We issue a request between x_0 and an arbitrarily chosen isolated node y_0. This leads to the eviction of x_0 (otherwise, the algorithm incurs an arbitrarily large cost, while the optimal strategy is to collocate all communicating pairs). Hence, ALG must collocate this pair in a different cluster (cannot collocate it with B). In order to maintain a feasible partitioning of nodes after collocating $\{x_0, y_0\}$, ALG must replace x_0 with another isolated node, the new pivot node.
3. **Requests between nodes originating from the same clusters.** We continue to issue requests between the current pivot node and any node with the same origin as the pivot. Consider the i-th such request, and the isolated node x_i, collocated with B. Precisely, we issue a request between x_i and some node in C_i, where C_i is the largest communicating component s.t. $I(C_i) = I(x_i), C_i \neq \{x_0, y_0\}$. Then, ALG must collocate the communicating component $\{x_i\} \cup C_i$ in one cluster, and again, the algorithm replaces x_i with some isolated node x_{i+1}. We terminate the process once the number of remaining isolated nodes is smaller than $\ell + 3$. At each step i, the number of isolated nodes decreases either by one, or it decreases by two if C_i is a singleton. Therefore, once the process terminates, at least $\ell + 1$ nodes remain isolated.

To assure the correctness of this input sequence, we claim that the communicating components admit a feasible partition. Once we terminate, there are at least $\ell + 1$ isolated nodes left (the number of isolated nodes decreases at most by 2 at each step). Therefore, two isolated nodes x^* and y^* exist with the same cluster of origin, $I(\{x^*\}) = I(\{y^*\})$. Consider a partition P^*, obtained from the initial partition after swapping x_0 and y_0 with x^* and y^*, respectively. In P^*, the pair $\{x_0, y_0\}$ resides in the cluster $I(\{x^*, y^*\})$. After the first request $\{x_0, y_0\}$, the requests are issued only between nodes originating from the same cluster, and all of these are collocated in P^*. This implies that no request is external in P^*, and we conclude that it is a feasible partition.

We bound the cost of ALG on the produced request sequence. At each request issued at step (3) of our construction, some communicating component grows, and ALG performs at least one swap. Let $\mathcal{S}$ be the set of all communicating components created by issuing requests at step (3) of the construction. Each component $S \in \mathcal{S}$ grew exactly $|S|-1$ times, each time joining an isolated node, and hence the number of nodes ALG swaps is at least

$$\mathrm{ALG} \geq \sum_{S\in\mathcal{S}} (|S| - 1) = |\bigcup \mathcal{S}| - |\mathcal{S}|.$$

In total, $\bigcup \mathcal{S}$ contains all the $k \cdot \ell$ nodes excluding $k-1$ nodes of B, the 2 nodes in $\{x_0, y_0\}$ and at most $\ell + 2$ singletons, which amounts to at least $k \cdot \ell - k - \ell - 3$ nodes. The set $\mathcal{S}$ consists of at most $\ell - 1$ components, one per possible cluster of origin excluding B's cluster of origin. Hence, the total number of swaps performed by ALG is

$$\mathrm{ALG} \geq |\bigcup \mathcal{S}| - |\mathcal{S}| = k \cdot \ell - k - 2\ell - 2 = (k-2)(\ell - 1) - 4.$$

Finally, we bound offline algorithm's cost for the constructed sequence of requests. Consider an offline algorithm OPT that moves to P^* (described earlier in this proof) by performing only two node swaps. As argued earlier, no communicating component is split in P^* and OPT pays only for the two swaps. We combine the above arguments to conclude that the competitive ratio is bounded by $\mathrm{ALG}/\mathrm{OPT} \geq (k-2)(\ell-1)/2-2$. □

We note that the lower bound requires $k \geq 3$. In contrast, for $k = 2$ the learning problem is trivial: immediate collocation of communicating pairs is 1-competitive. In contrast, the general partitioning problem for $k = 2$ is non-trivial (see Section 3.4), a lower bound of 3 is known [7], and we provide a 6-competitive algorithm, see Section 3.4.

Later in Section 3.1, we elaborate on how to transform this construction to a lower bound for the general partitioning problem.

2.2 Lower Bound under Resource Augmentation

The majority of work on the online balanced partitioning problem so far [7–9] focuses on the scenario with resource augmentation, where the cluster capacity of an online algorithm is larger than that of the optimal offline algorithm to which we compare the performance. We say that an online algorithm uses augmentation $\epsilon > 1$ if each of its clusters has the capacity of $\epsilon \cdot k$ nodes. The number of all nodes remains $k \cdot \ell$.

By using a variant of the lower bound construction from Theorem 1, we show a lower bound of $\Omega(\ell)$ that holds even with significant resource augmentation.

Theorem 2 *With resource augmentation strictly smaller than $k/3$ (i.e., $\epsilon < 1/3$), the competitive ratio of any deterministic online algorithm for the learning model of online balanced graph partitioning is in $\Omega(\ell)$.*

Proof Fix k divisible by 3, and construct 3 communication components of size $k/3$ in each cluster. Consider any deterministic online algorithm with resource augmentation $\epsilon < 4/3$. Note that no more than 3 such communication components fit in one cluster. Then, apply the construction from the lower bound for $k = 3$ (Theorem 1), treating these communication components as individual nodes. The cost of any algorithm (including the optimal offline algorithm) scales up by $k/3$ due to the increased cost of moving entire components instead of individual nodes, and we conclude that the lower bound $\Omega(\ell)$ holds. □

2.3 Upper Bound

We present an asymptotically optimal algorithm PPL (*Perfect Partition Learner*) for the learning model. The algorithm is a modification of the algorithm DET from [7]. The difference is in the choice of partition after a component merge. In DET, the choice of the partition was arbitrary, whereas our algorithm chooses an arbitrary partition closest to the initial partition that keeps all communicating nodes collocated. The algorithm PPL is listed in Algorithm 1.

To maintain the feasibility of the solution, the algorithm maintains *components* of communicating nodes. Initially, all nodes are in their own component, and upon a request between two nodes from different components, we *merge* the components. We maintain an invariant that the nodes of each communicating component are collocated. We say that a partition that collocates all nodes of all communicating components is a *communicating component respecting* partition.

Perfect Partition Learner

On each inter-cluster request $\{u, v\}$, PPL creates new components by merging the two components that contain nodes u and v. In order to collocate nodes of the new component, PPL moves to a communication component-respecting partition that minimizes the distance to the initial partition P_I. We measure the distance between two configurations in the number of swaps to transform one configuration to the other. The distance to the initial partition is equivalent to the number of nodes that migrated from their cluster origin. Algorithm 1 describes the scheme of the algorithm.

Analysis

Fix a request sequence σ, the initial partition $P_I := \{I_1, \ldots, I_\ell\}$ and an optimal offline strategy OPT with the final partition P_{OPT}. For each partition $P = \{C_1, \ldots, C_\ell\}$ we define its *distance* from the initial partition as the number of nodes in P that do not reside in their initial cluster. Observe that at least

Algorithm 1 Perfect Partition Learner (PPL)

```
For each node v create a singleton component C_v = {v} and add it to 𝒞.
for each request σ_t = {u, v}, 1 ≤ t ≤ N do
    Let C_1 ∋ u and C_2 ∋ v be the components containing u and v,
      respectively.
    if C_1 ≠ C_2 then
        Merge C_1 and C_2 into one component C'
              and set 𝒞 = (𝒞 \ {C_1, C_2}) ∪ {C'}.
        if C_1 and C_2 are not collocated then
            Move to a component respecting partitioning, closest to P_I.
        end if
    end if
end for
```

$\Delta(P)$ node migrations are required in order to reach the partition P from P_I, and thus $\mathrm{OPT}(\sigma) \geq \Delta(P_{\mathrm{OPT}})$.

PPL replaces the current partition P with a perfect partition closest to P_I, thus it never moves to a partition that is more than $\Delta^* := \Delta(P_{\mathrm{OPT}})$ migrations away from P_I. Consequently, PPL never moves to a partition beyond the distance Δ^*. We use this property to bound the cost of each repartitioning of PPL.

Property 1 Let P be any partitioning chosen by PPL at any time. Then, its distance from the initial partitioning is $\Delta(P) \leq \Delta^*$.

Lemma 1 The cost of each repartitioning by PPL is at most $2 \cdot \mathrm{OPT}(\sigma)$, where $\mathrm{OPT}(\sigma)$ is the cost of an optimal offline algorithm for the request sequence σ.

Proof Let P_i denote the partition of PPL immediately after serving σ_i, the request that arrives at time t. Consider the repartitioning that transforms P_{t-1} to P_t upon the request σ_t. Let $M \subseteq V$ denote the set of nodes that migrate at t. Let M^- and M^+ denote the subset of nodes that, respectively, enter or leave their initial cluster during the repartitioning. In total, M^+ and M^- account for all migrations, $M = M^+ \cup M^-$.

Since at least $|M^-|$ nodes are not in their initial cluster before the repartitioning (i.e., in P_{t-1}), the distance from P_I before the repartitioning is $\Delta(P_{t-1}) \geq |M^-|$. Analogously, the distance after the repartitioning is $\Delta(P_t) \geq |M^+|$. Thus, $|M| \leq \Delta(P_{t-1}) + \Delta(P_t)$. By Property 1, $\Delta(P_{t-1}) \leq \Delta^*$ and $\Delta(P_t) \leq \Delta^*$. Since $\Delta^* \leq \mathrm{OPT}(\sigma)$, we conclude that the total cost of the algorithm is $|M| \leq 2 \cdot \mathrm{OPT}(\sigma)$. □

Theorem 3 PPL *is* $(2(k-1) \cdot \ell)$*-competitive.*

Proof Let $P_F := \{F_1, \ldots, F_\ell\}$ be the partition of PPL after serving the sequence σ. At each request, the algorithm enumerates all communicating component-respecting ℓ-way partitions of components that are in the same (closest) distance to P_I. That is, once it reaches a partition P at distance $\Delta^* = \Delta(P)$, it does not move to a partition P' where $\Delta(P') > \Delta^*$ before it enumerates all partitions at distance Δ^*. Hence, P_F is at distance at most $\Delta^* = \mathrm{OPT}(\sigma)$ from the initial partition.

We claim that PPL performs at most $(k-1)\cdot\ell$ repartitions while serving σ. Each component begins as a singleton, and with each request, the size of some component increases by one. Consequently, the number of repartitions of PPL is bounded by the number of times the components grow. Consider any cluster $F_i \in P_F$. Each cluster has the capacity k, thus the total number of times a component in F_i grows is at most $\sum_{C\in F_i}(|C|-1) \leq k-1$. Summing this bound over all ℓ clusters gives us at most $(k-1)\cdot\ell$ repartitions. By Lemma 1, each repartitioning costs at most $2\cdot\mathrm{OPT}(\sigma)$. The total cost of PPL is thus at most $2\cdot(k-1)\cdot\ell\cdot\mathrm{OPT}(\sigma)$, which implies the claim. □

By Theorem 1, the lower bound for the competitive ratio of any deterministic algorithm is $\Omega(k\cdot\ell)$, and we conclude that PPL is asymptotically optimal.

Note on running time. Since the component sizes are in $O(n)$, computing a component-respecting partition for $\ell = 2$ is feasible in polynomial time using dynamic programming [13], but is strongly NP-hard for $\ell > 2$ [32]. However, we assume unlimited computational power and focus on competitiveness instead.

3 General Partitioning Model

In this section, we discuss the general online model where the request sequence can be arbitrary. First, in Section 3.1, we show a lower bound of $\Omega(k\cdot\ell)$ by generalizing the construction for the learning model from Section 2.1. Second, we generalize the lower bound for resource augmentation from Theorem 2. Third, in Section 3.3, we show an $O(k\cdot\ell)$-competitive algorithm for $k = 3$. Finally, in Section 3.4, we show a strictly 6-competitive algorithm for $k = 2$.

3.1 Lower Bound

We generalize the lower bound construction of for the learning model in two dimensions. Recall that in the learning model, algorithms by assumption collocate any pair as soon as they communicate. Hence, we cannot apply the lower bound construction of Theorem 1 directly since algorithms may resist collocating such nodes. For the first dimension of our generalization, we impose a collocation by defining *ground sets* of nodes. Nodes that are assigned to the same ground set communicate as long as they are separated. For the second dimension of our generalization, we iterate the construction in batches. Repeating batches ensures that the input sequence with the claimed ratio can be arbitrarily long, and the algorithm cannot have a small competitive ratio with a large additive. After each batch, we ensure that the online algorithm resides in the same configuration as the optimal offline solution. A single batch resembles the construction from Theorem 1, but in place of each request, we *reveal* a ground set: once the online algorithm splits a revealed set, we issue as many requests as it takes for it to collocate the split parts.

We start by revealing a ground set B of size $k-1$ in an arbitrary cluster. Then, we reveal a cross-origin ground set with the pivot (the node collocated with B). Then, we repeatedly reveal ground sets of the current pivot and a node originating from the same cluster as the pivot. We repeat the last step of the construction $\Theta(k \cdot \ell)$ times (exact condition to be determined), using the isolated node collocated with B as a new pivot. The algorithm swaps at least one node at each such step.

We claim that this sequence is cheap to serve offline. Roughly, if an offline algorithm would reside in the initial configuration, only the requests between x_0 and y_0 would be external, and all the subsequent requests would be free. Consider an offline strategy that collocates x_0, y_0 by swapping them with some initially collocated nodes x^*, y^* that do not participate in any request. To make sure that such a pair always exists, we stop repeating the requests concerning the pivot while there are still two isolated nodes collocated on some server.

Ground sets. We construct our lower bound using *ground sets*. Instead of directly constructing the request sequence, we construct ground sets of nodes that start communicating if split by the online algorithm. This is possible since the algorithm is deterministic, and the adversary knows its configuration at any time. If the algorithm insists on keeping a ground set split, we continue to issue requests between non-collocated nodes of the ground set until the algorithm collocates them. Under such a request sequence, the algorithm must maintain a perfect partition of ground sets, as otherwise, it is not competitive. The ground sets are initially unknown to the algorithm, and the perfect partition is hidden from it. In contrast, the optimal offline algorithm knows the entire sequence in advance and may move to the perfect partition at the beginning.

Theorem 4 *Any deterministic online algorithm for the general model of online balanced graph partitioning has the competitive ratio at least* $(k-2)(\ell-1)/2-2$.

Proof Fix any deterministic online algorithm ALG and any optimal offline algorithm OPT. We construct a sequence for the general model in *batches* of requests that can be repeated arbitrarily many times.

We construct the first batch by repeating the construction from Theorem 1, where in place of a request (u, v), we merge the ground sets of u and v. This ensures that the algorithm collocates all nodes from each communicating component. If the algorithm does not collocate the nodes, it is not competitive as it pays an arbitrarily large cost for split ground sets. By the construction from Theorem 1, there exists a partitioning that collocates all the ground sets, and therefore any competitive algorithm eventually moves to such configuration.

After the batch finishes, we force the algorithm to move into the partitioning that is identical to OPT's partitioning. Let $\{C_1, C_2, \ldots, C_\ell\}$ be OPT's configuration at this point. We reveal additional ground sets C_i for $i \in [1, \ell]$ (i.e., containing all nodes that OPT has collocated). Note that these requests are free for OPT, and therefore OPT does not change its configuration. The batch ends once the algorithm moves to OPT's configuration.

Once the algorithm reaches OPT's configuration, we issue the next batch by repeating the construction. We may repeat issuing batches this way an arbitrary number of times. By applying similar reasoning to the proof of Theorem 1, in each batch the algorithm performs at least $(k \cdot \ell - k - 2\ell - 2)$ swaps, each for cost α. OPT performs at most two swaps in each batch. The competitive ratio from Theorem 1 holds for each batch separately, and therefore it holds for the entire sequence. □

3.2 Lower Bound for Algorithms with Resource Augmentation

We generalize the lower bound for the learning model (Theorem 2) to show that the factor of $\Omega(\ell)$ is unavoidable even with significant augmentation.

Theorem 5 *With resource augmentation strictly smaller than $k/3$ (i.e., $\epsilon < 1/3$), the competitive ratio of any deterministic online algorithm for the general model of online balanced graph partitioning is in $\Omega(\ell)$.*

Proof Fix k divisible by 3, and construct 3 ground sets of size $k/3$ in each cluster. Consider any deterministic online algorithm with resource augmentation $1 + 1/3 - \epsilon$. Note that no more than 3 such ground sets fit in one cluster. Then, apply the construction from the lower bound (Theorem 4) for $k = 3$ using these communication components treating them as individual nodes. The cost of any algorithm (including the optimal offline algorithm) scales up by $k/3$ due to increased cost of moving entire ground sets instead of individual nodes, and we conclude that the lower bound $\Omega(\ell)$ holds. □

3.3 Optimal Algorithm for Clusters of Size 3

For the setting with $k = 3$, Avin et al. [7] obtained a $O(\ell^2)$-competitive algorithm. Their algorithm keeps track of external communication between nodes, and upon reaching a threshold α (the cost of migrating a node), we call the edge between them *saturated*. The algorithm keeps the invariant that the endpoint nodes of each saturated edge are collocated, and to this end, the algorithm tracks connected components consisting of nodes reachable via saturated edges. To collocate the nodes, the algorithm moves to an arbitrary partition where all nodes of all connected components are collocated. The algorithm operates in *phases*, and if no partition satisfying the invariant exists, it resets the counters for all pairs of nodes.

The algorithm ALG_3 presented in this section is a modified version of this algorithm. The difference is in the choice of partition after a component merge. Their algorithm chooses the partitioning arbitrarily, and our algoritm chooses the partition closest to the current partition (a repartition of minimum cost).

Our modification of the algorithm is straightforward, and our main contribution is an improved analysis of the algorithm. A straightforward analysis of our algorithm results in the bound $O(\ell^2)$, and we improve the analysis two aspects. First, we observe that the cost of each of $O(\ell)$ reconfigurations per phase is constant. Second, we deal with pairs of nodes that do not reach the threshold α (*unsaturated edges*). In the analysis of the algorithm from [7], these caused an additive $O(\alpha \cdot k^2 \cdot \ell^2)$. In our analysis, we observe that either the number of unsaturated edges is small, or any algorithm pays for a significant fraction of them. We bound the cost of the latter by estimating the capabilities of the optimal offline algorithm to *prepare* for an incoming sequence of requests.

Saturated components. We say that the pair (u, v) is *saturated* if the counter's value is α, and *unsaturated* otherwise (saturation of a pair leads to a merge action). We say that a partition that collocates all nodes of all saturated components is a *saturated component respecting* partition.

The algorithm ALG_3. For each pair of nodes $\{x, y\}$, ALG_3 maintains a counter $C_{\{x,y\}}$ and increments it on every external request between x and y. Initially, each node is isolated (belongs to its own component). Once $C_{\{x,y\}} = \alpha$, ALG_3 merges the components of u and v, and moves to the closest saturated component respecting partition. If no such partitioning exists, ALG_3 resets all components to singleton components, resets all counters to 0, and ends the phase.

Theorem 6 *ALG_3 is 60ℓ-competitive for $k = 3$.*

Before bounding the competitive ratio of ALG_3, we upper-bound the cost of a single repartition of ALG_3. In our analysis, we distinguish between three types of clusters: C_1, C_2 and C_3. In a cluster of type C_i, the size of the largest component contained in this cluster is i.

Lemma 2 In a single repartition of ALG_3, it swaps at most two pairs of nodes.

Proof If no saturated component respecting partition exists after the merge of components, then ALG_3 resets all components, ends the phase, and performs no repartition. Thus it suffices to show that the merged component has a size at least 4 to conclude that ALG_3 incurs no migration cost.

Consider a request between u and v that triggered the repartition and let U and V be their respective clusters. The request triggered the repartition, hence it was external and $U \neq V$. We consider cases based on the types of clusters U and V.

1. If either U or V is of type C_1, then this cluster can fit the merged component, and the repartition is local within U and V, for the cost of at most 2 swaps.

2. If either U or V is of type C_3, a component of size 3 participates in a merge, and we have a component of size at least 4, and ALG_3 ends the phase with no repartition.
3. It remains to consider the case where both U and V are of type C_2. If (u, v) both belong to components of size 2, then the merged component has size 4, and ALG_3 incurs no cost. Otherwise, if one of u, v belongs to a component of size 2, then it suffices to swap components of size 1 between U and V. Finally, if u and v belong to components of size 1, then we must place them in a cluster different from U and V. Note that if C_1-type cluster does not exist, then no saturated component respecting partitioning exists. Otherwise, ALG_3 performs two swaps — it swaps the nodes u and v with any two nodes of any cluster of type C_1.

In each case, we showed that a saturated component respecting partition is reachable in at most two swaps. □

Next, we observe that ALG_3 keeps saturated edges internal, and it increases counters only upon external communication, thus we upper-bound the ALG_3's counter value for each unsaturated edge in any phase.

Observation 1 The external request counter for each unsaturated edge has a value at most $\alpha - 1$.

Now we are ready to bound the competitive ratio of ALG_3.

Proof of Theorem 6 Fix a completed phase, and consider the state of ALG_3's counters at the end of it (before the reset). By σ we denote the input sequence that arrived during the phase. We consider the incomplete phase later in this proof.

In our analysis, we focus on the requests that were external to ALG_3 at the moment of their arrival; these are the only requests that incur a cost for ALG_3. We denote these external requests by σ_{cost}. We partition the sequence σ_{cost} into subsequences σ_I and σ_E. The sequence σ_I (inter-component requests) denotes the requests from σ_{cost} issued to pairs that belong to the same component of ALG_3 at the end of the phase. The sequence σ_E (extra-component requests) denotes the requests from σ_{cost} that do not appear in σ_I. Let $\mathsf{ALG}_3(M)$ denote the cost of migrations performed by ALG_3 in this phase.

First, we bound the cost of ALG_3 in the phase. During the phase, ALG_3 performs at most 2ℓ component merge operations — exceeding this number would mean that a component of size 4 exists, and the phase would have ended already. We bound the cost of each repartition after a merge by Lemma 2, obtaining $\mathsf{ALG}_3(M) \leq 8\alpha \cdot \ell$. We bound $\mathsf{ALG}_3(\sigma_I)$ by summing the intra-component counters of each cluster at the end of the phase. The sum of intra-component counters in a cluster of type C_3 is at most $3\alpha - 1$: two pairs of nodes from the component are saturated and its counter is α each, and the counter of the third, unsaturated pair is at most $\alpha - 1$ by Observation 1. The sum of counters inside C_1 is 0, and inside C_2 it is α. Summing over all ℓ clusters gives us $\mathsf{ALG}_3(\sigma_I) \leq (3\alpha - 1) \cdot \ell \leq 3\alpha \cdot \ell$. Furthermore, ALG_3 paid for all requests from σ_E, and thus $\mathsf{ALG}_3(\sigma_E) = |\sigma_E|$. In total, the cost of ALG_3 is at most $\mathsf{ALG}_3(\sigma_I) + \mathsf{ALG}_3(\sigma_E) + \mathsf{ALG}_3(M) \leq 11\alpha \cdot \ell + |\sigma_E|$ during the phase.

Second, we lower-bound the cost of the optimal offline solution. To this end, we fix any optimal offline algorithm OPT. By $\mathrm{OPT}(\sigma_I)$ and $\mathrm{OPT}(\sigma_E)$ we denote the cost of OPT on requests from sequences σ_I and σ_E, respectively. Note that these costs are defined with respect to components of ALG_3 in this phase. By $\mathrm{OPT}(M)$ we denote the cost of migrations performed by OPT in this phase.

The cost of OPT is lower-bounded by the cost of serving σ_I and the cost of serving σ_E. While serving these requests, OPT may perform migrations, and we account for them in both parts: we separately bound OPT by $\mathrm{OPT}(\sigma_I)+\mathrm{OPT}(M)$ and $\mathrm{OPT}(\sigma_E)+\mathrm{OPT}(M)$. Combining those bounds and using the relation between the maximum and the average, we obtain the bound

$$\begin{aligned}\mathrm{OPT} &\geq \max\{\mathrm{OPT}(\sigma_I)+\mathrm{OPT}(M), \mathrm{OPT}(\sigma_E)+\mathrm{OPT}(M)\}\\ &\geq (\mathrm{OPT}(\sigma_I)+\mathrm{OPT}(M))/2 + (\mathrm{OPT}(\sigma_E)+\mathrm{OPT}(M))/2.\end{aligned}$$

First, we show $\mathrm{OPT}(M)+\mathrm{OPT}(\sigma_I) \geq \alpha$. Assume that OPT's partition is fixed throughout the phase (as otherwise OPT pays α for a migration). The phase ended when the components of ALG_3 could not be partitioned without splitting them. Hence, for every possible partition of OPT, there exists a non-collocated saturated pair, and OPT paid for α requests that saturated the pair.

Next, we bound $\mathrm{OPT}(\sigma_E)+\mathrm{OPT}(M)$. The sequence σ_E accounts only for unsaturated edges, thus by Observation 1, there are at most $\alpha-1$ requests to each pair in σ_E. OPT may have at most 3ℓ pairs of nodes collocated in its clusters, and thus avoid paying for $3\ell\cdot(\alpha-1)$ requests from σ_E. Hence, at least $\chi := |\sigma_E| - 3\ell\cdot(\alpha-1)$ requests from σ_E are external requests with respect to OPT's configuration at the beginning of the phase. Faced with these requests, OPT may serve them remotely or perform migrations to decrease its cost. By swapping a pair of nodes (u,v), OPT collocates u with two nodes u', u'', and v with two nodes v', v''. This may allow serving requests between (u,u'), (u,u''), (v,v') and (v,v'') for free afterward. Hence, by performing a single swap that costs 2α, OPT may avoid paying the remote serving costs for at most $4(\alpha-1)$ requests from σ_E. The total cost of OPT is then at least

$$\mathrm{OPT}(\sigma_E)+\mathrm{OPT}(M) \geq \chi\cdot\frac{2\alpha}{4(\alpha-1)} \geq \frac{|\sigma_E|}{2} - 2\alpha\cdot\ell.$$

Finally, to bound the competitive ratio, we transform the above inequality in the following way: $|\sigma_E| \leq 2(\mathrm{OPT}(\sigma_E)+\mathrm{OPT}(M)) + 4\alpha\cdot\ell$. For succinctness, let $\xi := \mathrm{OPT}(\sigma_E)+\mathrm{OPT}(M)$. Combining the bounds on the cost of ALG_3 and OPT during each finished phase, the competitive ratio is

$$\frac{\mathsf{ALG}_3(\sigma)}{\mathrm{OPT}(\sigma)} \leq \frac{11\alpha\cdot\ell+|\sigma_E|}{\alpha/2+\xi/2} \leq \frac{30\alpha\cdot\ell+4\cdot\xi}{\alpha+\xi} \leq 30\ell.$$

It remains to consider the last, unfinished phase. First, consider the case where the unfinished phase is also the first one. Then, we cannot charge OPT due to the inability to partition the components. Instead, we use the fact that ALG_3 and OPT started with the same initial partition. If the input finished before the first α external requests, then ALG_3 is 1-competitive. If at least α external requests were issued, then OPT either paid α for serving them remotely or paid α for a migration. Charging this cost to OPT serves the purpose of charging α at the end of a finished phase, and thus we can apply the reasoning as for a finished phase. Second, consider the case, where there are at least two phases. Then we split the cost α of OPT accounted in the penultimate phase into the last two phases, and we repeat the analysis of a finished phase. This way, the competitive ratio increases at most twofold in comparison to a finished phase, and the competitive ratio is $\mathsf{ALG}_3(\sigma)/\mathrm{OPT}(\sigma) \leq 60\ell$. □

Note on Arbitrary Capacity

The presented algorithm assumes the servers have capacity 3. The challenge in generalizing beyond this fixed capacity lies in bounding the cost of finding the closest saturated component respecting partition. For the capacity $k = 3$, we bound the cost of each reconfiguration by enumerating all cluster possibilities. We argue that the reconfiguration for $k = 3$ impacts only a constant number of clusters, and its cost is bounded only in terms of k, independently of ℓ. The major challenge at the time of our work was bounding the reconfiguration cost independently of ℓ, which was later addressed by Bienkowski et al. [12].

Distributed Implementation

While we have described the algorithm globally so far, we note that it allows for efficient distributed implementations. The algorithm performs two types of operations that require communication with other clusters: a component merge, and a broadcast of the end of the phase. We say that a cluster containing 3 isolated nodes is *fresh*. A merge of two components may require finding a fresh cluster (for details see the proof of Lemma 2). In the following, we show how to efficiently find a fresh cluster in a distributed manner. We organize the clusters into an arbitrary rooted balanced binary tree, and we broadcast the root to each cluster. Each cluster maintains the counter of fresh clusters in its subtree. To find a fresh cluster, we traverse an arbitrary path of non-zero counters from the root. Upon encountering a fresh cluster, we end the traversal and decrease the counters on the followed path by 1. Summarizing, ending the phase requires a single broadcast, and merging components has $O(\log \ell)$ communication complexity.

3.4 Improved Algorithm for Online Rematching

In this section, we present RM, an algorithm for clusters of capacity $k = 2$. We interpret a pair of nodes collocated in one cluster as a "matched" pair. Hence, the problem is an online variant of the maximal matching problem where a matched pair can separate in order to "rematch" with two other nodes. Rematching is necessary for maximizing intra-cluster communications, which is equivalent to minimizing inter-cluster communications. This is known as the *online rematching* problem, and a non-strict 7-competitive algorithm is already given by [7], in which the ratio comes with an additive factor $O(\alpha\ell^2)$. We do not only improve upon their competitive ratio, but also we show that our ratio holds *strictly* (i.e., with no additive factor).

Our algorithm is slightly simpler than the one in [7], while our analysis is simpler and more concise. In the analysis of the algorithm, we propose a novel charging scheme for edges that share a vertex.

Algorithm ReMatch

The algorithm ReMatch (RM) maintains a counter $C_{\{x,y\}}$ for each pair of nodes $\{x, y\}$ and increments it on every remote request between x and y. Once

$C_{\{x,y\}} = \lambda$, it resets the counter $C_{\{x,y\}} := 0$ and collocates the two nodes by swapping one of them, say x, with the node collocated with y.

Theorem 7 *For $\lambda = \alpha$, the algorithm* RM *is strictly 6-competitive.*

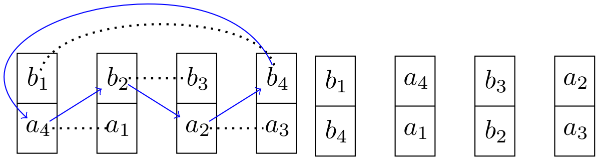

Fig. 2: Dashed lines represent external requests. An arrow from node x to node y indicates that x replaces y. In the left configuration, OPT collocates 4 pairs by performing 4 migrations, which results in the right configuration.

The Charging Scheme

We charge both OPT and RM whenever RM collocates a pair. RM collocates a pair always with a swap which costs 2α, while OPT may save some costs by collocating multiple pairs at once. Thus it pays the price of only one migration per pair (see Figure 2). Therefore, OPT possibly collocates a pair by moving one node to the cluster of the other node paying only α, in contrast to the swapping cost 2α incurred by RM.

Consider two pairs that share the same node, i.e., *intersecting pairs*, and the set of requests that cause (first) collocations of these pairs. This set contains at least one request to each pair, and OPT must pay a non-zero cost over requests in this set, as it cannot have both pairs collocated at the same time. However, we can charge this cost to OPT only the first time RM collocates a pair and not at any later time when RM collocates it a second time. Otherwise, OPT is possibly charged for the same cost repeatedly. For this reason, we charge OPT a cost inflicted by a pair if and only if OPT incurs that cost after the last time RM separates the pair.

Proof of Theorem 7 Fix an input sequence of requests $\sigma := \{\sigma_1, \ldots, \sigma_m\}$. Assume that RM collocates a pair $\{u, v\}$ at time t. The value of $C_{\{u,v\}}$ at t, denoted $C^t_{\{u,v\}}$, reaches λ immediately before RM resets the counter. For any interval $[t_1, t_2]$, by $\sigma_{\{x,y\}}[t_1, t_2]$ we denote the set of all requests to a pair $\{x, y\}$ that arrive during $[t_1, t_2]$. We may use $\sigma_{\{x,y\}}$ whenever the interval $[t_1, t_2]$ is clear from the context.

If t is not the first time that RM collocates $\{u, v\}$ then let $0 < t' < t$ be the latest time when RM separates $\{u, v\}$ in order to collocate some intersecting pair $\{x, y\} \neq \{u, v\}$, $\{x, y\} \cap \{u, v\} \neq \emptyset$, e.g., $\{x, y\} = \{u, w\}$. Else, t is the first time that RM collocates $\{u, v\}$ and let $t' := 0$. Similarly, if $t' > 0$ is not the first time that RM collocates $\{u, w\}$ then let $0 < t'' < t'$ be the latest time before t' when RM separates $\{u, w\}$. Else, t' is the first time that RM collocates $\{u, w\}$ and we let $t'' = 0$.

First, we bound costs incurred by RM for requests that lead to the collocation of $\{u,v\}$ at time $t \in T$, where $T := \{\tau \in [1,m] \mid \exists\{x,y\} : C^{\tau}_{\{x,y\}} = \lambda\}$ is the set of times when RM performs a collocation. By definitions of t and t', the overall cost of requests in $\sigma_{\{u,v\}}$ incurred by RM, i.e., the total cost of remote serving and the moving cost is $\lambda + 2\alpha$.

Next, we bound costs incurred by RM for requests that do not lead to collocations until the end of the sequence at $t = m$. Assume $\{u,v\}$ is not collocated at $t = m$ and $0 < C^{m}_{\{u,v\}} < \lambda$, which means RM pays $C^{m}_{\{u,v\}}$ for requests in $\sigma_{\{u,v\}}(t', m]$. Then the total cost incurred by RM is

$$\mathrm{RM}(\sigma) = \sum_{t \in T} (\lambda + 2\alpha) + \sum_{\{u,v\}} C^{m}_{\{u,v\}}.$$

Next, we bound costs incurred by OPT for requests that lead to the collocation of $\{u,v\}$ at $t \in T$. If t is the first time that RM collocates $\{u,v\}$, then OPT pays λ for serving requests in $\sigma_{\{u,v\}}[0,t]$ (remotely), or α for collocating the pair and serving (some of) the requests with cost zero. Therefore, in this case, $\mathrm{OPT}(\sigma_{\{u,v\}}(0,t]) \geq \min\{\lambda, \alpha\}$. Otherwise, it is not the first collocation and consider times t' and t'' as defined previously, and let $R_t := \sigma_{\{u,w\}}(t'', t'] \cup \sigma_{\{u,v\}}(t', t]$. We define $R_{t'}$ for the collocation at t' analogously (see Figure 3). Then, $\mathrm{OPT}(R_t) = \mathrm{OPT}(\sigma_{\{u,w\}}) + \mathrm{OPT}(\sigma_{\{u,v\}})$.

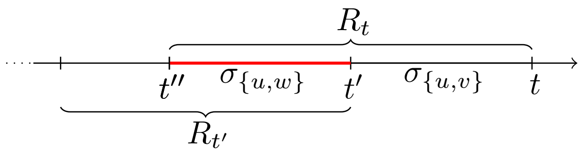

Fig. 3: Illustration of the timeline used in the proof of Theorem 7. The set R_t consists of requests to the pairs $\{u,w\}$ and $\{u,v\}$, which arrive in the interval $(t'', t]$. Similarly, $R_{t'}$ consists of requests to $\{u,w\}$ and some other pair irrelevant to the analysis. Hence, requests to $\{u,w\}$ are contained in both sets, and they arrive in $(t'', t']$ highlighted in red thick line.

If OPT has both pairs separated during their respective intervals, then clearly it pays 2λ in those intervals. Note that (trivially) OPT cannot have both pairs collocated at the same time. Else, OPT has one of the pairs, say $\{u,v\}$, already collocated prior its respective interval, $(t', t]$, and keeps it so during this interval. Then it pays zero for requests to this pair. Hence, it pays α for collocating the other pair, in this case $\{u,w\}$, or it pays λ for serving requests in $\sigma_{\{u,w\}}$. Note that OPT may deviate from the two cases by collocating the pair after serving some of its external requests. In any case, $\mathrm{OPT}(R_t) \geq \min\{\lambda, \alpha\} = \alpha$.

It remains to bound the cost incurred by OPT due to requests to $\{u,v\}$ that do not lead to its collocation until the end of the sequence at $t = m$. We bound the cost analogously to the case where RM collocates $\{u,v\}$. If $\{u,v\}$ is not collocated in the initial matching and RM never collocates it, then $C^{m}_{\{u,v\}} = |\sigma_{\{u,v\}}[1,m]|$. OPT pays $\mathrm{OPT}(\sigma_{\{u,v\}}[1,m]) \geq \min\{\alpha, C^{m}_{\{u,v\}}\}$, for collocating this pair, or it pays for requests in $\sigma_{\{u,v\}}[1,m]$. Else, either $\{u,v\}$ is collocated in the initial matching or

RM collocates it at some point. Then there exists an intersecting pair $\{u, w\}$ that is collocated by RM at $t' < m$, separating $\{u, v\}$. We define times $t'' < t' < m$ analogously to the former case. Let $R^*_{\{u,v\}} := \sigma_{\{u,w\}}(t'', t'] \cup \sigma_{\{u,v\}}(t', m]$. Then, OPT must pay for collocating at least one pair or (and) serving requests to the other pair remotely. Thus, $\mathrm{OPT}(R^*_{\{u,v\}}) \geq \min\{C^m_{\{u,v\}}, \alpha\}$.

Next, we sum up all costs incurred by OPT. By definitions of R_t and $R^*_{\{u,v\}}$, we have either $R_{t'} \cap R_t = \sigma_{\{u,w\}}$ or $R_{t'} \cap R^*_{\{u,v\}} = \sigma_{\{u,w\}}$. This means, $\mathrm{OPT}(\sigma_{\{u,w\}})$ is counted at most twice in each of the expressions $\mathrm{OPT}(R_{t'}) + \mathrm{OPT}(R_t)$ and $\mathrm{OPT}(R_{t'}) + \mathrm{OPT}(R^*_{\{u,v\}})$. Hence, the total cost to OPT is

$$\mathrm{OPT}(\sigma) = \frac{1}{2}\Big(\sum_{t\in T} \mathrm{OPT}(R_t) + \sum_{\{u,v\}} \mathrm{OPT}(R^*_{\{u,v\}})\Big) \geq \frac{1}{2}\Big(\sum_{t\in T} \alpha + \sum_{\{u,v\}} C^m_{\{u,v\}}\Big).$$

Finally, we bound the competitive ratio by aggregating the above bounds, obtaining

$$\mathrm{RM}(\sigma)/\mathrm{OPT}(\sigma) \leq 2\Big(\sum_{t\in T} 3\alpha + \sum_{\{u,v\}} C^m_{\{u,v\}}\Big)/\Big(\sum_{t\in T} \alpha + \sum_{\{u,v\}} C^m_{\{u,v\}}\Big) \leq 6.$$

□

4 Discussion and Future Work

This paper revisited the online graph partitioning problem and presented several tight bounds for the important model where capacities cannot be exceeded, both for a general partitioning model and for a special learning model.

Our algorithms allow for efficient distributed implementations. The algorithm PPL from Section 2.3 can be distributed similarly to the approach in [8]. The algorithm for $k = 2$ from Section 3.4 performs only local communication for each request: counters are kept on the clusters and updated locally, and each migration is local within two clusters that reached the collocation threshold λ. Furthermore, we proposed an efficient distributed implementation of the algorithm for $k = 3$ in Section 3.3.

There remain several interesting avenues for future research. A general open direction concerns the study of the power of randomization in our setting. It would generally also be interesting to study the performance of our algorithms empirically, under realistic workloads, and engineer algorithms to speed up computations. But there are also more specific open questions. First open question regards the dependency on ℓ with resource augmentation. Our lower bounds from Theorems 2 and 5 shed light on the dependency on ℓ in the competitive ratio for the setting with augmentation. The algorithm CREP from [7] requires $(2+\epsilon)$-augmentation to guarantee the competitive ratio independent of ℓ. In contrast, our construction shows that the linear term ℓ is inevitable if the augmentation is smaller than $4/3$. This raises a question about the tradeoffs between augmentation and the dependency on ℓ for the competitive ratio. Another open question concerns the runtime. The algorithm Perfect Partition Learner runs in superpolynomial time, and the dominating term in the runtime is due to finding the communicating component-respecting partition. We hence wonder if there exists a polynomial-time algorithm achieving an (asymptotically) optimal competitive ratio.

5 Acknowledgments

This project has received funding from the European Union's Horizon 2020 European Research Council (ERC), grant agreement No. 864228, 2020-2025.

References

[1] Roy, A., Zeng, H., Bagga, J., Porter, G., Snoeren, A.C.: Inside the social network's (datacenter) network. In: Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, pp. 123–137 (2015)

[2] Singh, A., Ong, J., Agarwal, A., Anderson, G., Armistead, A., Bannon, R., Boving, S., Desai, G., Felderman, B., Germano, P., *et al.*: Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. ACM SIGCOMM Computer Communication review **45**(4), 183–197 (2015)

[3] Mogul, J.C., Popa, L.: What we talk about when we talk about cloud network performance. ACM SIGCOMM Computer Communication Review **42**(5), 44–48 (2012)

[4] Chowdhury, M., Zaharia, M., Ma, J., Jordan, M.I., Stoica, I.: Managing Data Transfers in Computer Clusters with Orchestra. ACM SIGCOMM **41**(4), 98–109 (2011)

[5] Avin, C., Ghobadi, M., Griner, C., Schmid, S.: On the complexity of traffic traces and implications. In: Proc. ACM SIGMETRICS (2020)

[6] Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis, (1998)

[7] Avin, C., Bienkowski, M., Loukas, A., Pacut, M., Schmid, S.: Dynamic balanced graph partitioning. SIAM J. Discret. Math. **34**(3), 1791–1812 (2020)

[8] Henzinger, M., Neumann, S., Schmid, S.: Efficient distributed workload (re-)embedding. In: Proceedings of ACM SIGMETRICS / IFIP Performance 2019 (2019)

[9] Henzinger, M., Neumann, S., Raecke, H., Schmid, S.: Tight bounds for online graph partitioning. In: Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA) (2021)

[10] Forner, T., Raecke, H., Schmid, S.: Online balanced repartitioning of dynamic communication patterns in polynomial time. In: Proc. SIAM Symposium on Algorithmic Principles of Computer Systems (APOCS) (2021)

[11] Andrews, G., Eriksson, K.: Integer Partitions. Cambridge University Press

[12] Bienkowski, M., Böhm, M., Koutecký, M., Rothvoß, T., Sgall, J., Veselý, P.: Improved analysis of online balanced clustering. Approximation and Online Algorithms - International Workshop (WAOA) (2021)

[13] Avin, C., Cohen, L., Parham, M., Schmid, S.: Competitive clustering of stochastic communication patterns on a ring. Computing **101**(9), 1369–1390 (2019)

[14] Andreev, K., Räcke, H.: Balanced graph partitioning. Theory of Computing Systems **39**(6), 929–939 (2006)

[15] Garey, M.R., Johnson, D.S., Stockmeyer, L.J.: Some Simplified NP-Complete Graph Problems **1**(3), 237–267 (1976)

[16] Saran, H., Vazirani, V.: Finding k cuts within twice the optimal. SIAM Journal on Computing **24**(1), 101–108 (1995)

[17] Arora, S., Karger, D.R., Karpinski, M.: Polynomial time approximation schemes for dense instances of NP-hard problems. Journal of Computer and System Sciences **58**(1), 193–210 (1999)

[18] Feige, U., Krauthgamer, R., Nissim, K.: Approximating the minimum bisection size (extended abstract). In: Proc. 32nd ACM Symposium on Theory of Computing (STOC), pp. 530–536 (2000)

[19] Feige, U., Krauthgamer, R.: A polylogarithmic approximation of the minimum bisection. SIAM Journal on Computing **31**(4), 1090–1118 (2002)

[20] Krauthgamer, R., Feige, U.: A polylogarithmic approximation of the minimum bisection. SIAM Review **48**(1), 99–130 (2006)

[21] Räcke, H.: Optimal hierarchica decompositions for congestion minimization in networks. In: Proc. 40th ACM Symposium on Theory of Computing (STOC), pp. 255–264 (2008)

[22] Sleator, D., Tarjan, R.: Amortized efficiency of list update and paging rules. Communications of the ACM **28**(2), 202–208 (1985)

[23] Fiat, A., Karp, R.M., Luby, M., McGeoch, L.A., Sleator, D.D., Young,

N.E.: Competitive paging algorithms. Journal of Algorithms **12**(4), 685–699 (1991)

[24] McGeoch, L., Sleator, D.: A strongly competitive randomized paging algorithm. Algorithmica **6**(6), 816–825 (1991)

[25] Achlioptas, D., Chrobak, M., Noga, J.: Competitive analysis of randomized paging algorithms. Theoretical Computer Science **234**(1–2), 203–218 (2000)

[26] Epstein, L., Imreh, C., Levin, A., Nagy-György, J.: On variants of file caching. In: Proc. 38th Int. Colloq. on Automata, Languages and Programming (ICALP), pp. 195–206 (2011)

[27] Epstein, L., Imreh, C., Levin, A., Nagy-György, J.: Online file caching with rejection penalties. Algorithmica **71**(2), 279–306 (2015)

[28] Irani, S.: Page replacement with multi-size pages and applications to web caching. Algorithmica **33**(3), 384–409 (2002)

[29] Karlin, A.R., Manasse, M.S., McGeoch, L.A., Owicki, S.S.: Competitive randomized algorithms for nonuniform problems. Algorithmica **11**(6), 542–571 (1994)

[30] Stanton, I.: Streaming balanced graph partitioning algorithms for random graphs. In: Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms. SODA '14, pp. 1287–1301 (2014)

[31] Alistarh, D., Iglesias, J., Vojnovic, M.: Streaming min-max hypergraph partitioning. In: Advances in Neural Information Processing Systems 28, pp. 1900–1908 (2015)

[32] Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness, (1990)