Optimal Algorithms for Dynamic Balanced Graph Partitioning

Anonymous author
Anonymous affiliation

Abstract

Distributed applications, including batch processing, streaming, scale-out databases, or machine learning, generate a significant amount of network traffic. By collocating frequently communicating nodes (e.g., virtual machines) 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 in an online manner. This paper revisits the online balanced partitioning problem (introduced by Avin et al. at DISC 2016) that asks for an algorithm that strikes an optimal tradeoff between the benefits of collocation (i.e., lower network load) and its costs (i.e., migrations). Our first contribution is a significantly improved deterministic lower bound of $\Omega(k \cdot \ell)$ on the competitive ratio, where ℓ is the number of clusters and k is the cluster size, even for a scenario in which the communication pattern is static and can be perfectly partitioned; we also provide an asymptotically tight upper bound of $O(k \cdot \ell)$ for this scenario. For $k = 3$, we contribute an asymptotically tight upper bound of $\Theta(\ell)$ for the general model in which the communication pattern can change arbitrarily over time. We improve the result for $k = 2$ by providing a strictly 6-competitive upper bound for the general model. In contrast to most prior work, our algorithms respect all capacity constraints and do not require resource augmentation.

2012 ACM Subject Classification Theory of computation → Online algorithms; Networks → Network algorithms; Computer systems organization → Distributed architectures

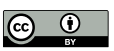
Keywords and phrases online algorithms, competitive analysis, distributed computing, graph partitioning, clustering, self-adjusting networks

Digital Object Identifier 10.4230/LIPIcs.DISC.2020.

1 Introduction

The popularity of data-centric, distributed applications has led to an explosive growth of network traffic, especially in data centers [23, 25]. The performance of these distributed applications often critically depends on the underlying network [21], and efficient operation of these networks is important. At the same time, distributed systems are often highly virtualized today, and provide interesting new opportunities for resource optimization. In particular, it has become possible to operate data centers in a more demand-aware manner: by dynamically migrating nodes (e.g., virtual machines) which communicate frequently topologically closer to each other, network traffic can be reduced significantly. However, migrations entail overhead and should be used moderately.

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. In particular, we are interested in an online variant of the problem: since communication patterns can change over time, an online algorithm needs to react dynamically to new traffic patterns, and migrate nodes accordingly. Ideally, this algorithm should perform close to an optimal offline algorithm, without requiring any information about future traffic demands.



This problem is known as the dynamic balanced graph partitioning problem and was introduced by Avin et al. [8, 6] at DISC 2016. a special variant of the general problem has later been studied by Henzinger et al. [16] at SIGMETRICS 2019. We refer to the latter as the learning model.

1.1 Model

We study two models in this paper: the *general partitioning* model, and its subproblem, the *learning* model. In both models, we assume that communication patterns are not known to our algorithms at the beginning. We measure the quality of presented algorithmic solutions by competitive analysis [9], which is well-suited for problems that are online by their nature. In the competitive analysis, the goal is to optimize *the competitive ratio* of a given online algorithm: the ratio of its cost to the cost of an optimal offline algorithm that knows the entire input sequence in advance.

General partitioning model. In the *dynamic balanced graph partitioning* problem, we are given a set V of n nodes (e.g., virtual machines or processes), initially arbitrarily partitioned into ℓ clusters (e.g., servers or entire racks), each of size k. The nodes interact using 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 certain amount of data. Nodes in $C \subset V$ are *collocated* if they reside in the same cluster.

An algorithm serves a communication request between two nodes either *locally* at cost 0 if they are collocated, or *remotely* at cost 1 if they are located in different clusters. We refer to these two types of requests as *internal* and *external* requests, respectively. Before serving a request, an online algorithm may perform a *repartition*, i.e., it may move ("migrate") some nodes into clusters different from their current clusters, while respecting the capacity of every cluster. Afterward, the algorithm serves the request. The cost of migrating a node from one cluster to another is $\alpha \in \mathbb{Z}^+$. For any algorithm ALG, its cost, denoted by $\mathrm{ALG}(\sigma)$, is the total cost of communications and the cost of migrations performed by ALG while serving the sequence σ.

Learning model. We study a *learning* variant of dynamic balanced graph partitioning, where the communication pattern is *static*: whether a pair of nodes ever communicate or not, is determined a priori and is unknown to algorithms, and such pairs communicate forever. Any algorithm must eventually collocate pairs of communicating nodes, as otherwise it cannot be competitive. As in Henzinger et al. [16], we assume that the communication graph admits a *perfect partition*, i.e., a partition in which no inter-cluster request ever occurs. The algorithm's objective is to *learn* the (static) communication graph while serving all requests, and without executing too many migrations. For the learning model, we assume that the migration cost is $\alpha = 1$.

1.2 Related Work

The two works closest to ours are by Avin et al. (on the general partitioning model) [8, 6] and by Henzinger et al. (on the learning model) [16]. 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. actually showed that their lower bound $\Omega(k)$ holds even for a significant resource 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 not only more realistic but also significantly more challenging, as it is related to hard problems such as integer partitioning [4]. In terms of results without augmentation, so far, it is only known that there exists an $O(k^2 \cdot \ell^2)$-competitive algorithm [8]; the best known lower bound is significantly lower, namely $\Omega(k)$. For $k = 2$, Avin et al. [8] presented a 7-competitive algorithm with a substantial $(\Omega(\ell^2))$ additive constant.

The problem has also been studied in a weaker model where the adversary can only sample requests from a fixed distribution [7].

The static offline version of the partitioning problem, i.e., a problem variant where migration is not allowed, where all requests are known in advance, and where the goal is to find an assignment of n nodes to ℓ physical machines, each of capacity n/ℓ, is known as the *ℓ-balanced graph partitioning problem*. The problem is NP-complete, and cannot even be approximated within any finite factor unless P = NP [3]. The static variant where $\ell = 2$ corresponds to the minimum bisection problem, which is already NP-hard [15], and the currently best approximation ratio is $O(\log n)$ [24, 5, 13, 12, 19, 22].

Our problem is further related to some classic online problems. In particular, it is related to online paging [26, 14, 20, 1], 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) [26]. 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* [10, 11, 17]. 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.

Dynamic graph partitioning problems are generally fundamental in computer science, and arise in many different contexts [27, 2].

1.3 Our Contributions

This paper presents several new results on the dynamic graph partitioning problem without augmentation. For the learning model, we present a lower bound of $\Omega(k \cdot \ell)$ on the competitive ratio of any online deterministic online algorithm (that holds also in the general partitioning model). The best known lower bound so far was $\Omega(k)$ [8] that holds only in the general partitioning model. We complement this result with an asymptotically optimal, $O(k \cdot \ell)$-competitive algorithm for the learning model.

For the general partitioning model, we design an asymptotically optimal, $\Theta(\ell)$-competitive algorithm for $k = 3$, improving the best known upper bound so far $O(\ell^2)$ [8]. 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 (i.e., without an additive term). Table 1 provides an overview of our contributions compared to prior work.

Algorithmic techniques. The variant for general k is still unresolved, hence it is vital to summarize algorithmic techniques used in this paper. A straightforward analysis of the algorithm from Section 3.1 results in the bound $O(\ell^2)$, and the analysis must be improved in two places. One must bound the cost of each of $O(\ell)$ reconfigurations per phase by

Variant	Lower bound	Upper bound
$k = 2$	3 [8]	6 (§3.2)
$k = 3$	$\Omega(\ell)$ (§2.1)	$O(\ell)$ (§3.1)
$k > 3$	$\Omega(k \cdot \ell)$ (§2.1)	$O(k^2 \cdot \ell^2)$ [8]
Learning model	$\Omega(k \cdot \ell)$ (§2.1)	$O(k \cdot \ell)$ (§2.2)

Table 1 Overview of known results and our contributions. The table summarizes the results for the general partitioning model, except for the last row that summarizes the results for the learning model for arbitrary k and ℓ.

a constant, and show that the optimum offline algorithm must pay a significant cost for inter-cluster requests. We bound the cost of the latter by estimating the capabilities of the optimum offline algorithm to *prepare* for an incoming sequence of requests. Furthermore, in the analysis of the algorithm from Section 3.2, we propose a novel charging scheme for edges that share a vertex. We also emphasize the implications of the tight result for the learning model. If one hopes to improve the lower bound for the general model, the ground sets must be dynamically added and *removed*: if ground sets are only added, the instance is solvable by the algorithm from Section 2.2 within the cost $O(k \cdot \ell)$.

2 The Learning Model

In this section, we consider the learning variant of dynamic balanced graph partitioning problem. For this setting, we show a surprisingly high lower bound of $\Omega(k \cdot \ell)$ for $k \geq 3$. The lower bound holds also in the general partitioning model (studied in Section 3). At the end of this section, we discuss an asymptotically optimal upper bound for the learning variant.

2.1 Lower Bound

We provide a lower bound $\Omega(k \cdot \ell)$ for the competitive ratio of any deterministic online algorithm for the learning problem. Later, we elaborate on how to efficiently transform it to a lower bound for the general partitioning problem. 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 3.2).

Throughout this paper, we often refer to groups of communicating nodes. We use this concept slightly differently in the lower bound than the upper bounds. In our algorithms, we group nodes with a communication history into *components*. In this section, we group nodes that may ever communicate, into *ground sets*.

Given a perfect partition, every subset of nodes that belong to the same cluster in this partition is a *ground set*. Any competitive algorithm under the learning model maintains a perfect partition of ground sets into clusters. On each inter-cluster request, a ground set is revealed. An algorithm recovers the (hidden) perfect partition gradually over inter-cluster requests, by merging pairs of ground sets involved in these requests.

The adversary constructs ground sets depending on the choices of a deterministic online algorithm. Once we construct a ground set, it lasts until the end of the input sequence. We say that a ground set is a *singleton* if it contains exactly one node, which is an *isolated* node.

We start by constructing a ground set of size $k - 1$ on an arbitrarily chosen cluster. In any partition, there must exist an isolated node collocated with the ground set of size $k - 1$.

We issue requests between this node and some node that was initially collocated with it. By repeating such requests, almost every node is once collocated with the first ground set. In comparison, we show that there exists an optimal offline algorithm OPT that performs only two node exchanges ("swaps").

▶ **Theorem 1.** *The competitive ratio of any deterministic online algorithm for the learning model of Dynamic Balanced Graph Partitioning is at least* $(k-2)(\ell-1)/2-2$ *for any* $k \geq 3$ *and* $\ell \geq 2$.

Proof. Fix any online algorithm ALG. For a ground set C of nodes that are initially collocated in one cluster, let $I(C)$ denote the cluster. We refer to $I(C)$ as the cluster of *origin*, when C is clear from the context. Initially, all nodes are isolated, i.e., each node is in a singleton ground set. First, we choose a cluster arbitrarily and create a ground set B of $k-1$ nodes in this cluster. Each cluster hosts exactly k nodes, and in any feasible partition, a single isolated node must be collocated with B. At any time, we refer to the isolated node currently collocated with B as the *pivot* node. Let x_0 denote the first pivot node.

Then, we join the pivot node to a larger ground set to force its eviction. Precisely, we create a ground set $\{x_0, y_0\}$, where y_0 is an arbitrary isolated node. Since ALG does not have $\{x_0, y_0\}$ collocated, the adversary issues an external request to this pair so that ALG collocates it. ALG cannot collocate $\{x_0, y_0\}$ with B (as B's size is $k-1$), hence it collocates them in a different cluster. In order to preserve a feasible partition of nodes after collocating $\{x_0, y_0\}$, ALG must replace x_0 with another isolated node that becomes the new pivot.

We proceed in similar steps by joining the current pivot node to a ground set of the same origin residing in a different cluster. Consider the step i, when the isolated node x_i is collocated with B. We issue a request between x_i and some node in C_i, where C_i is the largest ground set s.t. $I(C_i) = I(x_i), C_i \neq \{x_0, y_0\}$. Then ALG must collocate the new ground set $\{x_i\} \cup C_i$ in one cluster. Any feasible partition replaces x_i with some isolated node x_{i+1}, as the new ground set $\{x_i\} \cup C_i$ may not be ever split. We terminate the process once the number of remaining isolated nodes is less than $\ell + 3$. At each step i, the number of isolated nodes decreases either by one or by two if C_i is a singleton. Therefore, once the process terminates, in any case at least $\ell + 1$ isolated nodes are left.

Next, we argue that a feasible partition exists when the process terminates. This implies that a feasible partition exists after any earlier step as well. Since there are at least $\ell + 1$ isolated nodes left, there must be two isolated nodes x^* and y^*, with the same cluster of origin, i.e., $I(\{x^*\}) = I(\{y^*\})$. Consider the partition P^* obtained from the initial partition after swapping x_0 and y_0 with x^* and y^* (respectively). In this partition, the ground set $\{x_0, y_0\}$ is collocated in the cluster $I(\{x^*, y^*\})$. Note that after the first request $\{x_0, y_0\}$, we issue requests only between nodes that have the same cluster of origin and all these nodes are collocated in P^*. Therefore all ground sets constructed so far are collocated in P^*, and it is a feasible partition.

Consider nodes x^* and y^* and the partition P^* obtained previously. OPT moves to P^* by performing only two node swaps. Precisely, OPT collocates $\{x_0, y_0\}$ by swapping them with x^* and y^*. No ground set is split in P^* and OPT pays only for the two swaps.

ALG performs at least one swap at each step i, and some ground set grows. Consider any ground set $C^* \neq B$ after the termination. This ground set has grown exactly $|C^*| - 1$ times until the termination. Let $\mathcal{S}$ be the set of all ground sets after the process terminates. Thus, $\mathcal{S}$ includes ground sets B, $\{x_0, y_0\}$, and (up to) $\ell + 2$ singleton ground sets. Among the remaining ground sets in $\mathcal{S}$, no two ground sets have the same origin. Otherwise, the smaller ground set is either a singleton, which contradicts the bound $\ell + 2$ on the number of singletons, or we have joined nodes to it at some step, contradicting our choice of the largest C_i at step i.

Hence, there are at most $\ell - 1$ such ground sets, one per possible cluster of origin, excluding the cluster containing B. Therefore, $|\mathcal{S}| \leq 1 + 1 + (\ell + 2) + (\ell - 1) = 2\ell + 3$. Note that among all non-singleton ground sets in $\mathcal{S}$, only B does not grow during the process. Thus, the total number of times that a ground set in $\mathcal{S}$ has grown is $\sum_{C^* \in \mathcal{S}} (|C^*| - 1) - (k - 1)$

$$= \sum_{C^* \in \mathcal{S}} |C^*| - \sum_{C^* \in \mathcal{S}} 1 - (k-1) \geq k\ell - (2\ell + 3) - (k-1) = (k-2)(\ell-1) - 4,$$

which bounds the number of swaps performed by ALG. The competitive ratio is then $\mathrm{ALG}/\mathrm{OPT} \geq ((k-2)(\ell-1) - 4)/2$. ◀

Lower bound for the general problem. For a lower bound $\Omega(k \cdot \ell)$ for the general partitioning problem, the adversary continues to issue requests to split nodes of a ground set until the algorithm collocates them. Note that the ground sets constructed in our lower bound can be perfectly partitioned into the clusters. Hence, the optimal algorithm moves to a perfect partition at the beginning (where requests incur the cost 0), and its cost is bounded. This means that the algorithm must eventually collocate all nodes of a ground set to be competitive. We reveal the next ground set only after the collocation, hence we can repeat the analysis of the algorithm for the learning problem. Finally, we note that the construction is oblivious to the choice of the reconfiguration cost α: we compare the number of node exchanges of ALG and OPT.

Resource augmentation. The majority of work on the online balanced partitioning problem so far [8, 16] focuses on the scenario with resource augmentation, where the clusters of an online algorithm are larger than the clusters of the offline optimal algorithm that we compare the performance to. We can adjust our construction to show a lower bound of $\Omega(\ell)$ for a setting with resource augmentation.

Consider a partitioning problem with resource augmentation $1 + 1/3 - \epsilon$. Fix k divisible by 3, and construct 3 ground sets of size $k/3$ in each cluster. Note that no more than 3 such ground sets fit in one cluster. Then, apply the construction from the lower bound for $k = 3$, using these ground sets in the way we used individual nodes. The cost of any algorithm (including OPT) scales up by $k/3$, and the lower bound $\Omega(\ell)$ holds.

Finally, we note the possibility of improvement. The algorithm CREP [8] 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 $1 + 1/3$.

2.2 Upper Bound

We present an asymptotically optimal algorithm for the learning problem. The algorithm collocates a pair as soon as they communicate and it never separates them. In order to preserve collocated pairs, we employ the concept of components, introduced by Avin et al. [8].

We maintain subsets of frequently communicating nodes as *components*. Initially, each node constitutes a single-node component which we refer to as a *singleton* component, and the node in such component is an *isolated* node. We define larger components in terms of smaller components. Concretely, given a (sub)sequence of requests, two components C_1 and C_2 *merge* into one component as soon as for some pair of nodes $v_1 \in C_1$ and $v_2 \in C_2$, the frequency of requests $\{v_1, v_2\}$ reaches a certain threshold through the sequence. We keep all nodes of a component always collocated in the same cluster, i.e., when we move a node, we move the whole component that contains it. A partition that has every component collocated is a *component respecting* partition.

In addition, we maintain a balanced partition of our components as long as such partition exists, a reminiscent of partitioning given integers into sets of equal sum [4]. In contrast, our partition is time-varying: two components are merged into one component once they communicate, and we adjust the partition accordingly.

The algorithm in this section and the algorithm for $k = 3$ (cf. Section 3.1) are modified versions of the algorithm DET from [8]. The difference is in the choice of partition after a component merge. In DET, the partition was arbitrary. In the algorithm for the learning model, we choose a component respecting partition closest to the initial partition. In the algorithm from Section 3.1, we choose the partition closest to the current partition (a repartition of minimum cost).

Perfect Partition Learner algorithm. Now we describe the algorithm PPL. 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 component respecting partition that minimizes the distance to the initial partition P_I. The scheme of the algorithm can be found in the appendix (Algorithm 1).

Fix the initial partition $P_I := \{I_1, \ldots, I_\ell\}$ and OPT's final partition $P_F := \{F_1, \ldots, F_\ell\}$. The *distance* of a partition $P = \{C_1, \ldots, C_\ell\}$ from the initial partition, defined as $\Delta(P) := \sum_{j=1}^{\ell} |C_j \setminus I_j|$, is the number of nodes in P that do not reside in their initial cluster. In other words, at least $\Delta(P)$ node migrations are required in order to reach the partition P from P_I, and thus $\mathrm{OPT} \geq \Delta(P_F)$.

PPL never moves to a partition that is more than $\Delta^* := \Delta(P_F)$ migrations away from P_I. This invariant latter ensures us that PPL does not pay too much while recovering P_F. We emphasize that a *repartitioning* by PPL replaces the current partition P with a perfect partition closest to P_I. This way PPL never moves to a partition beyond the distance Δ^*.

▶ **Property 1.** *Let P be any partition chosen by* PPL *at any time. Then, $\Delta(P) \leq \Delta^*$.*

▶ **Lemma 2.** *The cost of each repartitioning by* PPL *is $2 \cdot \mathrm{OPT}$.*

Proof. Let P_i denote the partition of PPL immediately after serving σ_i. Consider the repartitioning that transforms P_{t-1} to P_t upon the request σ_t. Let $M \subset V$ denote the set of nodes that migrate during this process. Let M^- and M^+ denote the subset of nodes that, respectively, enter or leave their initial cluster during the repartitioning. Then, $M = M^+ \cup M^-$. Since at least $|M^-|$ nodes are not in their initial cluster before the repartitioning (i.e., in P_{t-1}), the distance before the repartitioning is $\Delta(P_{t-1}) \geq |M^-|$. Analogously, the distance afterward 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 $\Delta^* \leq \mathrm{OPT}$, we obtain $|M| \leq 2 \cdot \mathrm{OPT}$. ◀

▶ **Theorem 3.** PPL *reaches the final partition P_F and it is $(2 \cdot (k-1) \cdot \ell)$-competitive.*

Proof. On each inter-cluster request, the algorithm enumerates all 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 $P', \Delta(P') > \Delta^*$, before it enumerates all partitions at distance Δ^*. Therefore, PPL eventually reaches the partition P_F at distance $\Delta^* = \mathrm{OPT}$. With each distinct request, the size of some component increases by one. For any cluster $F_i \in P_F$, we have $\sum_{C \in F_i} |C| = k$. A component $C \in F_i$, initially begins as an isolated node and it grows by gaining $|C| - 1$ more nodes. Hence, the total number of times a component in F_i grows is $\sum_{C \in F_i} (|C| - 1) \leq k - 1$. Therefore, there are at most $(k-1) \cdot \ell$ distinct requests for which PPL performs a repartitioning and PPL performs at most $(k-1) \cdot \ell$ repartitions. By Lemma 2, each repartitioning costs at most $2 \cdot \mathrm{OPT}$. The total cost is thus at most $2 \cdot \mathrm{OPT} \cdot (k-1) \cdot \ell$, which implies the competitive ratio. ◀

3 General Partitioning Model

Now we discuss the general online model where the request sequence can be arbitrary. In Section 3.1, we show an $O(k \cdot \ell)$-competitive algorithm for $k = 3$ using the classic *rent-or-buy* approach [18]. Prior to this section, we showed a lower bound of $\Omega(k \cdot \ell)$ that holds for the general model (cf. Section 2.1), hence the result from this section is asymptotically optimal. Furthermore, in Section 3.2, we show a strictly 6-competitive algorithm for $k = 2$.

3.1 Optimal Algorithm for Clusters of Size 3

The algorithm analyzed in this section is a modified version of the algorithm DET proposed by Avin et al. [8], which for $k = 3$ is $O(\ell^2)$-competitive. In our algorithm, we choose the closest partition after a component merge instead of an arbitrary one. This allows to bound the cost of repartition by a constant (Lemma 5).

This modification alone is insufficient to obtain $O(\ell)$-competitive algorithm, and the analysis must be further improved. In particular, pairs of nodes that did not reach the collocation threshold α (called external requests) incur the cost $O(\ell^2)$ for the algorithm in each phase. The novel part of the analysis lower-bounds the cost of OPT on external requests while considering its savings from migrations and possibly different configuration at the beginning of the phase. This way, we show that OPT paid a significant portion of the algorithm's cost on external requests.

Component-based algorithm. The algorithm ALG_3 partitions nodes into components, and initially, each node is isolated (belongs to its own component). 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. Once $C_{\{x,y\}} = \alpha$, ALG_3 merges the components of u and v, and moves to the closest component respecting partitioning. If no such partitioning exists, ALG_3 resets all components to singleton components, resets all counters to 0, and ends the phase.

▶ **Theorem 4.** *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 among 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 5.** *In a single repartition of nodes (after a merge of components), ALG_3 exchanges at most two pairs of nodes.*

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

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. 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.

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 exchange 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 component respecting partitioning exists. Otherwise, ALG_3 performs one swap — it exchanges the nodes u and v with any two nodes of any cluster of type C_1.

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

Proof of Theorem 4. Fix a completed phase, and consider the state of ALG_3's counters at the end of it (before the reset). We consider the incomplete phase later in this proof.

ALG_3 is component respecting, hence it never increases any counter above α. 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). By σ we denote the input sequence that arrived during the phase. 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. 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 should have ended already. We bound the cost of each repartition after a merge by Lemma 5, 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 $\alpha - 1$. 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 this phase.

Now 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 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 repeat the analysis of a finished phase. Second, consider the case, where there are at least two phases, then we split the cost α charged 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 $\mathsf{ALG}_3(\sigma)/\mathrm{OPT}(\sigma) \leq 60\ell$. ◀

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 5). In the following, we show how to efficiently find a fresh cluster in a distributed manner. To this end, 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.

3.2 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 [8], in which the ratio comes with an additive factor $O(\alpha\ell^2)$. We do not only improve upon their competitive ratio, but also show that our ratio holds *strictly* (i.e., with no additive factor). Our algorithm is slightly simpler than the one in [8], while our analysis is significantly simpler and more concise, thanks to the charging scheme we devise here.

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 pair by swapping x with the node collocated with y.

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

The charging scheme. We charge both OPT and RM whenever RM collocates a pair.RM collocates a pair always with a swap (that 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 1). For this reason, whenever OPT collocates a pair, we charge it only the cost α of moving a single node to the other cluster (in contrast to the cost 2α incurred by RM).

Consider two pairs that share the same node, i.e. *intersecting pairs*, and the set of requests that causes the collocation of both pairs, at some times during $[1, m]$. Observe that OPT must pay a non-zero cost for these requests over the entire σ, since it cannot have both pairs initially collocated. However, we can charge this cost to OPT only the first time RM collocates a pair, and not at any consequent 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 latest separation of the pair by RM.

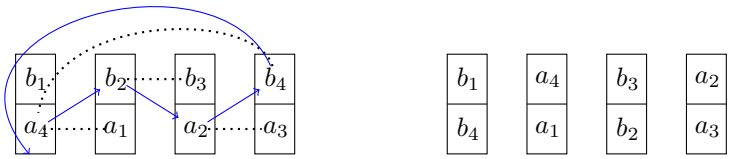


Figure 1 Dashed lines represent requests. An arrow from node x to node y indicates that x replaces y. OPT collocates 4 pairs from the left partition, with 4 migrations, resulting in the right partition.

Proof of Theorem 6. 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 $0 < t'' < t'$

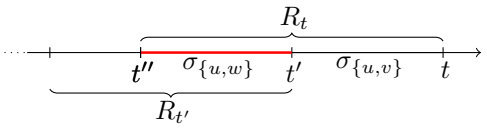


Figure 2 Illustration of the timeline used in the proof of Theorem 6. Requests in R_t are only to $\{u, w\}$ and $\{u, v\}$, which arrive during the interval $(t'', t]$. Similarly, request in $R_{t'}$ are to $\{u, w\}$ and some other pair, irrelevant to the analysis. Hence, requests to $\{u, w\}$ are included in at most two such sets, which are $R_{t'}$ and R_t. This is because their intervals overlap on $(t'', t']$, shown with red thick line.

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 $T := \{i \in [1, m] \mid \exists\{x, y\} : C^i_{\{x,y\}} = \lambda\}$ 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 overall cost of 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 trigger 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 2). Then, $\mathrm{OPT}(R_t) = \mathrm{OPT}(\sigma_{\{u,w\}}) + \mathrm{OPT}(\sigma_{\{u,v\}})$. If OPT has both pairs separated during their respective intervals, then obviously it pays 2λ during those intervals. Note that OPT cannot have both pairs collocated at the same time. Let us assume OPT has one of the pairs, e.g. $\{u, v\}$, collocated already prior its respective interval, $(t', t]$, and keeps it so during the interval. Then it pays zero while serving $\sigma_{\{u,v\}}$. Hence, it must pay α for collocating the other pair, in this case $\{u, w\}$, or (resp., and) it pays (resp., up to) λ for serving (resp., some of) requests in $\sigma_{\{u,w\}}$. Therefore 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 (and) paying for (resp. some of) 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, for all collocations performed by RM, and for final requests at $t = m$, OPT pays

at least $\frac{1}{2}(\sum_{t\in T}\mathrm{OPT}(R_t)+\sum_{\{u,v\}}\mathrm{OPT}(R^*_{\{u,v\}}))$. Then, the total cost to OPT is

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

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

4 Discussion and Future Work

This paper revisited the dynamic 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.

While our bounds are tight, there are several interesting avenues for future research. In particular, we have so far focused on deterministic algorithms, and it would be interesting to study the power of randomization in this context. On the practical side, it would also be interesting to study our algorithms empirically, under realistic workloads.

Our algorithms allow for efficient distributed implementations. The algorithm PPL from Section 2.2 can be distributed similarly to the approach in [16]. The algorithm for $k=2$ from Section 3.2 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 3.1.

References

1. Dimitris Achlioptas, Marek Chrobak, and John Noga. Competitive analysis of randomized paging algorithms. *Theoretical Computer Science*, 234(1–2):203–218, 2000.
2. Dan Alistarh, Jennifer Iglesias, and Milan Vojnovic. Streaming min-max hypergraph partitioning. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 1900–1908. Curran Associates, Inc., 2015.
3. Konstantin Andreev and Harald Räcke. Balanced graph partitioning. *Theory of Computing Systems*, 39(6):929–939, 2006.
4. George Andrews and Kimmo Eriksson. *Integer Partitions*. Cambridge University Press.
5. Sanjeev Arora, David R. Karger, and Marek Karpinski. Polynomial time approximation schemes for dense instances of NP-hard problems. *Journal of Computer and System Sciences*, 58(1):193–210, 1999.
6. Chen Avin, Marcin Bienkowski, Andreas Loukas, Maciej Pacut, and Stefan Schmid. Dynamic Balanced Graph Partitioning. *arXiv e-prints*, page arXiv:1511.02074, November 2015. `arXiv:1511.02074`.
7. Chen Avin, Louis Cohen, Mahmoud Parham, and Stefan Schmid. Competitive clustering of stochastic communication patterns on a ring. *Computing*, 101(9):1369–1390, 2019.
8. Chen Avin, Andreas Loukas, Maciej Pacut, and Stefan Schmid. Online Balanced Repartitioning. *DISC*, pages 243–256, 2016.
9. Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
10. Leah Epstein, Csanád Imreh, Asaf Levin, and Judit Nagy-György. On variants of file caching. In *Proc. 38th Int. Colloq. on Automata, Languages and Programming (ICALP)*, pages 195–206, 2011.
11. Leah Epstein, Csanád Imreh, Asaf Levin, and Judit Nagy-György. Online file caching with rejection penalties. *Algorithmica*, 71(2):279–306, 2015.

558 559 **12** Uriel Feige and Robert Krauthgamer. A polylogarithmic approximation of the minimum bisection. *SIAM Journal on Computing*, 31(4):1090–1118, 2002.

560 561 562 **13** Uriel Feige, Robert Krauthgamer, and Kobbi Nissim. Approximating the minimum bisection size (extended abstract). In *Proc. 32nd ACM Symposium on Theory of Computing (STOC)*, pages 530–536, 2000.

563 564 **14** Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel D. Sleator, and Neal E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, 1991.

565 566 **15** M. R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some Simplified NP-Complete Graph Problems. *Theoretical Computer Science*, 1(3):237–267, 1976.

567 568 **16** Monika Henzinger, Stefan Neumann, and Stefan Schmid. Efficient distributed workload (re-)embedding. In *ACM SIGMETRICS / IFIP Performance 2019*, 2019.

569 570 **17** Sandy Irani. Page replacement with multi-size pages and applications to web caching. *Algorithmica*, 33(3):384–409, 2002.

571 572 **18** Anna Karlin, Mark Manasse, and Lyle McGeoch Karlin. Competitive randomized algorithms for nonuniform problems. *Algorithmica*, 11(6):542—571, 1994.

573 574 **19** Robert Krauthgamer and Uriel Feige. A polylogarithmic approximation of the minimum bisection. *SIAM Review*, 48(1):99–130, 2006.

575 576 **20** Lyle McGeoch and Daniel Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(6):816–825, 1991.

577 578 **21** Jeffrey C Mogul and Lucian Popa. What we talk about when we talk about cloud network performance. *ACM SIGCOMM Computer Communication Review*, 42(5):44–48, 2012.

579 580 **22** Harald Räcke. Optimal hierarchica decompositions for congestion minimization in networks. In *Proc. 40th ACM Symposium on Theory of Computing (STOC)*, pages 255–264, 2008.

581 582 583 **23** Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C Snoeren. Inside the social network's (datacenter) network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 123–137, 2015.

584 585 **24** Huzur Saran and Vijay Vazirani. Finding k cuts within twice the optimal. *SIAM Journal on Computing*, 24(1):101–108, 1995.

586 587 588 589 **25** Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, 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.

590 591 **26** Daniel Sleator and Robert Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

592 593 594 **27** Isabelle Stanton. Streaming balanced graph partitioning algorithms for random graphs. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pages 1287–1301, 2014.

A Pseudocode of Perfect Partition Learner

Algorithm 1 Perfect Partition Learner (PPL)

For each node v create a singleton component $C_v = \{v\}$ and add it to $\mathcal{C}$.
for each request $\sigma_t = \{u, v\}, 1 \leq t \leq N$ **do**
  Let $C_1 \ni u$ and $C_2 \ni v$ be the components containing u and v, respectively.
  if $C_1 \neq C_2$ **then**
    Merge C_1 and C_2 into one component C' and $\mathcal{C} = (\mathcal{C} \setminus \{C_1, C_2\}) \cup \{C'\}$.
    if C_1 and C_2 are not collocated **then**
      Move to a partition closest to P_I and respecting all components in $\mathcal{C}$.
    end if
  end if
end for