Online Balanced Repartitioning*

Chen Avin[1] Andreas Loukas[2] Maciej Pacut[3] Stefan Schmid[4,2]

[1] Ben Gurion University of the Negev, Israel [2] TU Berlin, Germany
[3] University of Wroclaw, Poland [4] Aalborg University, Denmark

Abstract. Distributed cloud applications, including batch processing, streaming, and scale-out databases, generate a significant amount of network traffic and a considerable fraction of their runtime is due to network activity. This paper initiates the study of deterministic algorithms for collocating frequently communicating nodes in a distributed networked systems in an online fashion. In particular, we introduce the *Balanced RePartitioning* (BRP) problem: Given an arbitrary sequence of pairwise communication requests between n nodes, with patterns that may change over time, the objective is to dynamically partition the nodes into ℓ clusters, each of size k, at a minimum cost. Every communication request needs to be served: if the communicating nodes are located in the same cluster, the request is served locally, at cost 0; if the nodes are located in different clusters, the request is served remotely using inter-cluster communication, at cost 1. The partitioning can be updated dynamically (i.e., repartitioned), by *migrating* nodes between clusters at cost α per node migration. The goal is to devise online algorithms which find a good trade-off between the communication and the migration cost, i.e., "rent" or "buy", while maintaining partitions which minimize the number of inter-cluster communications. BRP features interesting connections to other well-known online problems. In particular, we show that scenarios with $\ell = 2$ generalize online paging, and scenarios with $k = 2$ constitute a novel online version of maximum matching. We consider settings both with and without cluster-size augmentation. Somewhat surprisingly (and unlike online paging), we prove that any deterministic online algorithm has a competitive ratio of at least k, *even with augmentation.* Our main technical contribution is an $O(k \log k)$-competitive deterministic algorithm for the setting with (constant) augmentation. This is attractive as, in contrast to ℓ, k is likely to be small in practice. For the case of matching ($k = 2$), we present a constant competitive algorithm that does not rely on augmentation.

Keywords: Dynamic Graphs, Clustering, Graph Partitioning, Algorithms, Competitive Analysis, Cloud Computing

* Research supported by the German-Israeli Foundation for Scientific Research (GIF) Grant I-1245-407.6/2014 and by the Polish National Science Centre grant DEC-2013/09/B/ST6/01538. We thank Marcin Bienkowski for many inputs and feedback on this paper.

1 Introduction

Graph partitioning problems, like minimum graph bisection and maximum matching, are among the most fundamental problems in Theoretical Computer Science. Due to their numerous practical applications (e. g., communication networks, data mining, social networks, etc. [4,15,27]), partitioning problems are also among the most intensively studied problems. Interestingly however, not much is known today about how to dynamically partition nodes which interact or communicate in a time-varying fashion.

This paper initiates the study of a natural model for *online* graph partitioning. We are given a set of n nodes with time-varying pairwise communication patterns, which we have to partition into ℓ clusters of equal size k. Intuitively, we would like to minimize inter-cluster links by mapping frequently communicating nodes to the same cluster. Since communication patterns change over time, partitions should be dynamically readjusted, that is, the nodes should be *repartitioned*, in an online manner, by *migrating* them between clusters. The objective is to jointly minimize inter-cluster communication and reconfiguration costs, defined respectively as the number of communication requests served remotely and the number of times nodes are migrated from one cluster to another.

One practical motivation for our problem arises in the context of server virtualization in datacenters. Distributed cloud applications, including batch processing applications such as MapReduce, streaming applications such as Apache Flink or Apache Spark, and scale-out databases and key-value stores such as Cassandra, generate a significant amount of network traffic and a considerable fraction of their runtime is due to network activity [25]. For example, traces of jobs from a Facebook cluster reveal that network transfers on average account for 33% of the execution time [14]. In such applications, it is desirable that frequently communicating virtual machines are *collocated*, i.e., mapped to the same physical server, since communication across the network (i.e., inter-server communication) induces network load and latency. However, migrating virtual machines between servers also comes at a price: the state transfer is bandwidth intensive, and may even lead to short service interruptions. Therefore the goal is to design online algorithms which find a good trade-off between the inter-server communication cost and the migration cost, similar in spirit to classical ski rental and rent-or-buy problems.

Formally we define the *Balanced RePartitioning* (BRP) problem as follows. The inputs to BRP are:

1. A set V of $n = |V|$ nodes (e.g., the virtual machines), initially distributed arbitrarily across ℓ clusters $\mathcal{C} = \{C_1, \ldots, C_\ell\}$ (e.g., the physical servers, interconnected by a top-of-the-rack switch [2]), each of size k (e.g., the number of cores or slots for virtual machines).
2. An arbitrary and possibly infinite sequence σ of $|\sigma|$ communication requests, $\sigma = \{u_1, v_1\}, \{u_2, v_2\}, \{u_3, v_3\}, \ldots, \{u_{|\sigma|}, v_{|\sigma|}\}$. For any t, $\sigma_t = \{u_t, v_t\}$ denotes a communication request: at time t, nodes $u_t, v_t \in V$ exchange (a fixed amount of) data. Intuitively, every request σ_t can be thought

of as an edge of the communication graph which appears at time t and then disappears at $t+1$.

At any time t, each node $v \in V$ is assigned to a cluster, which we will refer to by $C_t(v) \in \mathcal{C}$. If the time t is clear from the context or irrelevant, we simply write $C(v)$. We call two nodes $u, v \in V$ *collocated* if they are in the same cluster: $C(u) = C(v)$. We consider two settings:

1. *Without augmentation:* The nodes fit perfectly into the clusters, i.e., $n = k \cdot \ell$.
2. *With augmentation:* The online algorithm has access to additional space in each cluster. We say that an algorithm is δ-augmented if the size of each cluster is $k' = \delta \cdot k$, whereas the total number of nodes remains $n = k \cdot \ell < k' \cdot \ell$. As usual, in the competitive analysis, the augmented online algorithm is compared to the optimal offline algorithm without augmentation.

At each time t, the online algorithm needs to serve the communication request $\{u_t, v_t\}$, but can also repartition the nodes into new clusters before serving the request. We assume that a communication request between two nodes located in different clusters costs 1, a communication request between two nodes collocated in the same cluster costs 0, and migrating a node from one cluster to another costs $\alpha \geq 1$. Note that in a setting without augmentation, due to cluster size constraints, a node can never be migrated alone, but it must be swapped with another node at a cost of (at least) 2α.

As it turns out, BRP features some interesting connections to other well-known graph and online problems: (i) The static version (without migration) is the minimum balanced graph partitioning problem (where ℓ is the number of components). (ii) For $\ell = 2$, BRP can be shown to be a generalization of online paging, where the first cluster simulates the cache (the small but fast memory) and the second the slow but large memory. (iii) For $k = 2$, BRP is a novel online version of maximum matching. In the static case, maximum matching is a special case of minimum balanced graph partitioning with $n/2$ components.

The cost of an algorithm ALG for a given sequence of communication requests σ is

$$\mathrm{ALG}(\sigma) = \sum_{t=1}^{|\sigma|} \mathrm{mig}(\sigma_t; \mathrm{ALG}) + \mathrm{com}(\sigma_t; \mathrm{ALG}), \tag{1}$$

where $\mathrm{mig}(\sigma_t; \mathrm{ALG})$ is the migration cost at time t (α or 0) and $\mathrm{com}(\sigma_t; \mathrm{ALG})$ is the communication cost of σ_t (1 or 0). Let $\textsc{On}(\sigma)$ and $\textsc{Off}(\sigma)$ be the cost induced by σ on an online algorithm On and an optimal offline algorithm Off, respectively. In contrast to On, which learns the requests one-by-one as it serves them, Off has a complete knowledge of the entire request sequence σ ahead of time. We are in the realm of online algorithms and competitive analysis: We want to design online repartitioning algorithms which provide conservative (worst-case) guarantees, and minimize the (strict) competitive ratio:

$$\rho(\textsc{On}) = \max_{\sigma} \frac{\textsc{On}(\sigma)}{\textsc{Off}(\sigma)}. \tag{2}$$

To be competitive, an online repartitioning algorithm has to define a strategy for each of the following questions:

A) *Serve remotely or migrate ("rent or buy")?* If a communication pattern is short-lived, it may not be worthwhile to collocate the nodes: the migration cost cannot be amortized.
B) *Where to migrate, and what?* If nodes should be collocated, the question becomes where. Should u_t be migrated to $C(v_t)$, v_t to $C(u_t)$, or should both nodes be migrated to a new cluster? Moreover, an algorithm may be required to pro-actively migrate (resp. swap) additional nodes.
C) *Which nodes to evict?* There may not exist sufficient space in the desired destination cluster. In this case, the algorithm needs to decide which nodes to evict, to free up space.

Our Contributions. This paper introduces the online balanced repartitioning problem. We consider deterministic algorithms and make the following technical contributions:

1. *Online Rematching:* For the special case of online rematching ($k = 2$, but arbitrary ℓ), Theorem 1 presents a greedy online algorithm which is almost optimal: it is 7-competitive and we prove a lower bound of 3.
2. *Lower Bounds:* While in a setting without augmentation, a $k-1$ lower bound for the competitive ratio of any online algorithm follows from a simulation of online paging, in Theorem 2, we show a lower bound which is strictly larger than k, for any $\alpha > 0$. Intriguingly, we show that the online repartitioning problem remains hard even with augmentation. In particular, in Theorem 3 we prove that no augmented online algorithm can achieve a competitive ratio below k, as long as it cannot solve the problem trivially by placing all nodes into a single cluster. In contrast, online paging is known to become constant competitive with constant augmentation [29].
3. *Online Balanced Repartitioning:* Our main technical contribution stated in Theorem 4 is a non-trivial $O(k \log k)$-competitive algorithm for the setting with 4-augmentation.

Observe that none of our bounds depends on ℓ. This is interesting, as for example, in our motivating virtual machine collocation problem, k is likely to be small: a server typically hosts a small number of virtual machines (e.g., related to the constant number of cores on the server).

Paper Organization. The remainder of this paper is organized as follows. After reviewing related work in Section 2, we start by discussing the special case of matchings ($k = 2$) in Section 3. We consider lower bounds for the general setting with and without augmentation in Section 4. Section 5 is then devoted to the presentation and analysis of CREP, an augmented deterministic algorithm. We conclude in Section 6. Due to space constraints, some technical details and proofs only appear in our technical report [6].

2 Related Work

Our work assumes a new perspective on several classic algorithmic problems. The static version of our problem (without migration) is the minimum balanced graph partitioning problem (where ℓ is the number of components). This problem is known to be NP-complete, and cannot even be approximated within any finite factor [4]. The static variant where $k = 2$ corresponds to a maximum matching problem, which is polynomial-time solvable. The static variant where $\ell = 2$ corresponds to the minimum bisection problem [17] and can be approximated within a factor of $O(\log^{1.5} n)$ from the minimum cost [20]. The concept of cluster-size augmentation is inspired by offline *bicriteria* approximations to graph partitioning, in particular the (ℓ, δ)-balanced graph partitioning problem [4], where the graph has to be partitioned in ℓ components of size less than $\delta \cdot \frac{n}{\ell}$, as well as by the concept of c-balanced cuts used by Arora, Rao, and Vazirani [5], where both partitioned components should be of size at least $c \cdot n$.

In terms of online algorithms, the subproblem of finding a good trade-off between serving requests remotely (at a low but repeated communication cost) or migrating nodes together (entailing a potentially high one-time cost α), is essentially a ski rental or rent-or-buy problem [21,22]. A similar tradeoff also arises in the context of online page and server migration problems [9,10], where requests appear in a metric space [11] or graph [7] over time, and need to be served by one [10] or multiple [19] servers. However, in BRP, the number of possible node-cluster configurations is large, rendering it difficult to cast the problem into an online metrical task system. Moreover, in contrast to most online migration problems, which typically optimize the placement of a page or server with respect to the request locations, in our model, *both* end-points of a communication request are subject to optimization. A second difference to the usual models studied in the literature (where requests appear at specific locations in the metric space) is that in our problem a request only reveals partial and binary information about the optimal location (resp. configuration) to serve it: the request can be served at cost zero whenever the communicating are collocated.

Our model can be seen as a generalization of online paging [18,23,24,30], sometimes also referred to as online caching, where requests for data items arrive over time and need to be served from a cache of finite capacity, and where the number of cache misses must be minimized. The online caching and paging problem was first analyzed in the framework of the competitive analysis by Sleator and Tarjan [29], who presented a $kALG/(kALG - kOPT + 1)$-competitive algorithm, where $kALG$ is the cache size of the online and $kOPT$ the cache size of the offline algorithm. The authors also proved that no deterministic online algorithm can beat this ratio. In the classic caching model and its variants [12,13,24], items need to be put into the cache upon each request, and the problem usually boils down to finding a smart eviction strategy, such as Least Recently Used (LRU) or Flush-When-Full (FWF). In contrast, in our setting, requests can be served remotely. In this light, our model is reminiscent of caching models *with bypassing* [1,16]. In fact, it is easy to see that in a scenario with $\ell = 2$ clusters,

online paging can be simulated: in this simulation, one cluster can be used as the cache and the other cluster as the slow memory; by the corresponding problem reduction we also obtain a $k-1$ lower bound for our problem without augmentation. However, in general, in our model, the "cache" is *distributed*: requests occur *between* nodes and *not to* nodes, and costs can be saved by collocation.

BRP also has connections to online packing problems, where items of different sizes arriving over time need to be packed into a minimal number of bins [26,28]. In contrast to these problems, however, in our case the objective is not to minimize the number of bins but rather the number of "links" between bins, given a fixed number of bins.

Finally, our model also connects to recent work on online clique and correlation clustering [3,8,15,27]. In this prior work, nodes and/or links can appear over time, but the underlying communication graph remains invariant.

3 Online Rematching

Let us first consider the special case where clusters are of size two ($k = 2$, arbitrary ℓ). This is essentially an online maximal (re)matching problem: clusters of size two host (resp. "match") exactly one pair of nodes, and maximizing pairwise communication within each cluster is equivalent to minimizing inter-cluster communication. In a $k = 2$ scenario, the question of which node to evict is trivial: there is simply no choice. The problem can also be seen from a ski-rental perspective: one has to identify a good tradeoff between serving requests remotely ("renting") and migrating the communicating nodes together ("buy").

A natural greedy online algorithm Greedy to solve this problem proceeds as follows: For each cluster C_i, hosting nodes u_i, v_i, we count the total number of inter-cluster requests over time for its nodes. After 3α requests occur between nodes inside any cluster C_1 to nodes outside the cluster, we identify the cluster C_2 with which C_1 communicated most frequently in this time period. We then collocate u_1 with the single node in C_2 (v_2 or u_2) with which it communicated the most—ties broken arbitrarily and without involving any other clusters in the repartitioning. Afterwards, we reset all pairwise communication counters involving nodes from (old) clusters C_1 and C_2 (i.e., u_1, u_2, v_1, v_2).

Theorem 1. Greedy *is* 7*-competitive. No deterministic online algorithm achieves a competitive ratio below* 3 *when* $|\sigma| \to \infty$.

4 Lower Bounds for Online Balanced Repartitioning

Our problem is generally hard to approximate online. While it is easy to see that a lower bound of $k-1$ follows by simulating online paging (ufsing only two servers), in the following we prove a strictly larger lower bound (cf. Theorem 2). In fact, we observe that, even with augmentation, our problem is hard to approximate online: as long as the augmentation is less than what would be required to solve the partitioning problem trivially, by putting all nodes into the same

cluster (i. e., $\delta < \ell$), no deterministic online algorithm can achieve a competitive ratio better than k (cf. Theorem 3). This highlights an intriguing difference from online paging, where the competitive ratio becomes constant under augmentation [29]. Our lower bounds are independent of the initial configuration: both Off and On start off having the nodes placed identically in clusters.

Theorem 2. *No deterministic online algorithm can achieve a competitive ratio smaller than* $k + \frac{k-2}{2\alpha}$*, independently of* ℓ*.*

Interestingly, an adversary can outwit any online algorithm, even in the setting with augmentation. In the following, we consider online algorithms which, compared to Off, have δ-times more space in each cluster.

Theorem 3. *No* δ*-augmented deterministic online algorithm can achieve a competitive ratio smaller than* k*, as long as* $\delta < \ell$ *.*

5 CREP Algorithm: An $O(k \log k)$ Upper Bound

The main technical contribution of this paper is an online *Component-based REPartitioning algorithm (*CREP*)* which achieves an almost tight upper bound matching the k lower bound of Theorem 3 with augmentation at least 4. Intuitively, it helps to think of a 4-augmented algorithm as one that can use twice as many clusters, each having twice as much space (though this is a special case of the definition of augmentation). Formally, we claim:

Theorem 4. *With augmentation at least 4,* CREP *is* $O(k \log k)$*-competitive.*

CREP is summarized in Algorithm 1. The algorithm is non-trivial and relies on the following basic ideas:

1. *Communication components.* CREP groups nodes which have recently communicated into *components*. Once the cumulative communication cost of a group of nodes distributed across two or more components exceeds a certain threshold, CREP merges them into a single component, by collocating them in the same cluster. That is, we maintain a logical, time-varying weighted component graph $G_t = (\Phi_t, E_t, w_t)$, where Φ_t is the set of components immediately after request t has been issued, the edges E_t connect components which communicated at least once during this epoch, and w_t is the number of communication requests between the corresponding two nodes in this epoch. In other words, an edge $(i, j) \in E$ between two components $\phi_i, \phi_j \in \Phi_t$ indicates that the two components (resp. the corresponding nodes in ϕ_i and those in ϕ_j) were involved in $w_{ij} > 0$ requests. Although the graph G_t changes over time (when components are merged or split according to CREP), when the time is clear from the context, we drop the time-index and simply write $G = (\Phi, E, w)$. Edges disappear (and their weights are reset) when the components are merged. For a *component set* $X = \{\phi_i, \phi_j, \dots\} \subseteq \Phi$, let $|X|$ denote the number of components in X. We call $\mathrm{vol}(X) = \sum_{\phi \in X} |c|$ the *volume* of the set and $\mathrm{com}(X) = \sum_{\phi_i, \phi_j \in X} w_{ij}$ the *communication cost* among the members of X.

2. *Component epochs.* We analyze CREP in terms of component-wise epochs. A component epoch starts with the first request between two individual nodes (singleton components), and ends when: (i) the size of the component (the number of nodes in the component) exceeds k and (ii) the accumulated communication between the components exceeds a certain threshold. CREP maintains the invariant that components are never split during an epoch, that is, once two nodes of the same component epoch are placed together in a cluster, they will remain in the same cluster in the remainder of the epoch (but they may possibly be migrated together to a new cluster). As such, when a component set X is merged into a new component (Line 7), CREP tries to migrate all the components to the cluster of the largest component (ties broken arbitrarily). If there is not enough reserved space in the cluster, then all components are migrated to a new cluster. If on the other hand $\mathrm{vol}(X)$ exceeds k, the component-epoch ends, and all $\phi \in X$ are reset to singleton components (Line 19). More specifically, according to Algorithm 1, *two* termination criteria have to be fulfilled for a component set Y to end an epoch: $\mathrm{vol}(Y) > k$ *and* $\mathrm{com}(Y) \geq \mathrm{vol}(Y) \cdot \alpha$. This non-trivial criterion is critical, to keep the migration cost competitive. An epoch that ends as a result of a set Y is referred to as a Y-epoch.
3. *Space reservations.* In order to keep the number of migrations low, CREP performs space reservations in clusters. Whenever CREP migrates a component ϕ into a cluster, it reserves additional space $\mathrm{reserve}(\phi) = \min\{k - |\phi|, |\phi|\}$. As we will prove, these proactive space reservations can ensure that a component has to be migrated again only after its size doubles. For a cluster s, let $\mathrm{reserved}(s), \mathrm{occupied}(s)$ and $\mathrm{spare}(s)$ denote the reserved, occupied and spare (unreserved) space in s, where always $\mathrm{reserved}(s) + \mathrm{occupied}(s) + \mathrm{spare}(s) = 2k$. Similarly, for a component ϕ let $\mathrm{reserved}(\phi)$ denote the amount of its reserved space that is still available in its current cluster.

The remainder of this section is devoted to the proof of Theorem 4. The proof unfolds in a number of observations and lemmas. We first observe, in Property 1, that indeed, it is always possible to find a cluster where the to-be-merged components fit. We then derive an upper bound on CREP's cost per component epoch and a lower bound on the optimal offline cost per component epoch. Finally, we show that the competitive ratio is also bounded with respect to incomplete epochs.

We start by observing that there always exists a cluster which can host the entire merged component, including the required reserved space without any evacuation, i.e., its spare space is at least k.

Property 1. At any point in time, a cluster exists having at least k spare space.

So indeed, CREP can always place a merged component greedily into clusters—no global component rearrangement is necessary. On the other hand, augmenting the cluster size allows CREP to reserve additional space for migrated components. As we show in the following, this ensures that each node is

Algorithm 1 CREP with 4 Augmentation

Construct graph $G = (\Psi, E, w)$ with singleton components: one component per node. Set $w_{ij} = 0$ for all $\{v_i, v_j\} \in \binom{V}{2}$. For each component ϕ_i, reserve space reserve$(\phi_i) = 1$.

for each new request $\{u_t, v_t\}$ **do**

▷ Keep track of communication cost.

  Let $\phi_i = \Phi(u_t)$ and $\phi_j = \Phi(v_t)$ be the two components that communicated.
  if $\phi_i \neq \phi_j$ **then**
    $w_{ij} \leftarrow w_{ij} + 1$
  end if

▷ Merge components.

  Let X be the largest cardinality set with $\text{vol}(X) \leq k$ and $\text{com}(X) \geq (|X|-1)\cdot\alpha$
  if $|X| > 1$ **then**
    Let $\phi_0 = \bigcup_{\phi_i \in X} \phi_i$ and for all $\phi_j \in \Phi \backslash X$ set $w_{0j} = \sum_{\phi_i \in X} w_{ij}$.
    Let $\phi \in X$ be the component having the largest reserved space.
    if reserved$(\phi) \geq \text{vol}(X) - |\phi|$ **then**
      Migrate ϕ_0 to the cluster hosting ϕ
      Update reserved$(\phi_0) =$ reserved$(\phi) - (\text{vol}(X) - |\phi|)$
    else
      Migrate ϕ_0 to a cluster s with spare$(s) \geq \min(k, 2|\phi_0|)$
      Set reserved$(\phi_0) = \min(k - |\phi_0|, |\phi_0|)$
    end if
  end if

▷ End of a Y-epoch.

  Let Y be the smallest components set with $\text{vol}(Y) > k$ and $\text{com}(Y) \geq \text{vol}(Y)\cdot\alpha$
  if $Y \neq \emptyset$ **then**
    Split every $\phi_i \in Y$ into ϕ_i singleton components and reset the weights of all edges involving at least one newly created component. Reserve one additional space for each newly created component. If necessary, migrate at most $\text{vol}(Y)/2+1$ singletons to clusters with spare space.
  end if
end for

migrated at most $\log k$ times (rather than k) during the formation of a component.

Upper bound on CREP's costs. The online algorithm's cost during each epoch consists of the *communication cost*, which amounts to the number of communication requests that were served remotely, and the *migration cost*, which is equal to the number of node migrations. The following properties provide upper bounds for both kinds of costs for a single component:

Property 2. At any point in time, consider a component c induced by the communication pattern in this epoch, then:

1. The communication cost between nodes in ϕ is, in this epoch, at most $(|\phi| - 1) \cdot \alpha$.
2. The migration cost of nodes in ϕ is, in this epoch, at most $(|\phi| \log |\phi|) \cdot \alpha$.

Proof (Proof of Property 2). The two properties are proved in turn.

Property 2.1. We prove this property by induction on the *merging sequence*, i. e., the sequence of merges that includes all the nodes in ϕ from the time when they were singletons, ordered by time. To establish the base case, consider the first merge of nodes in ϕ, where X was a set of singletons (Line 7) and $|X|$ singleton components were combined into a new component $\phi_0 = \cup_{\phi_i \in X} \phi_i$. By CREP's merging condition, the cost up to this point is equal to $(|X| - 1) \cdot \alpha = (|\phi_0| - 1) \cdot \alpha$. For the inductive step, consider again that X' is merged into ϕ_0 and suppose that the communication cost paid for each component $\phi' \in X'$ is $(|\phi'| - 1) \cdot \alpha$. After the merge, CREP's total communication cost is equal to $(|X'| - 1) \cdot \alpha + \sum_{\phi_i \in X'} (|\phi_i| - 1) \cdot \alpha = \left(\sum_{\phi_i \in X'} |\phi_i| \right) \cdot \alpha - \alpha = (|\phi_0| - 1) \cdot \alpha$ and the induction holds.

Property 2.2. First observe that any node u which belongs to ϕ is migrated at most $\log |\phi|$ times during an epoch. To see this, suppose that u was just migrated into a cluster and that the size of u's current component is $|\phi'|$. From Property 1, we know that u will not be migrated as a consequence of a merge that does not involve u's component (i.e., it will never be evicted). Furthermore, due to the existence of reserved space, u will stay in the same cluster as long as the size of its current component ϕ' remains smaller or equal to $2|\phi'|$. Since the size of u's component between any consecutive migrations doubles, the total migrations can be at most $\log |\phi|$. This implies the total number of migrations pertaining to all nodes in ϕ is at most $|\phi| \log |\phi|$.

Using Property 2, we can bound the migration and communication cost of a Y-epoch:

Lemma 1. *Consider the end of a Y-epoch (Line 19).* CREP *migrates at most $\sum_{\phi_i \in Y} |\phi_i| \log |\phi_i| \leq vol(Y) \cdot \log k$ nodes and serves at most $2\, vol(Y) \cdot \alpha$ remote requests during this epoch for nodes in Y.*

The proof of Lemma 1 follows directly from Property 2 and is omitted.

Lower bound on Off's cost. Having derived an upper bound on CREP's cost, we next compute a lower bound of Off's cost.

Lemma 2. *By the end of a Y-epoch,* Off *pays at least $vol(Y)/k \cdot \alpha$ communication cost (during this epoch) for nodes in Y.*

To establish the above lower bound, we will need two useful properties.

Property 3. Consider any component ϕ in the current epoch and *any* partition of ϕ into two non-empty disjoint sets B and W, with $B \cup W = \phi$. During the creation of ϕ (by merging), there were at least α communication requests between nodes in B and W.

Proof (Proof of Property 3). Consider the tree T which describes how component ϕ merged from singletons into ϕ during the current epoch. The leafs of the tree are the nodes in ϕ and each internal node corresponds to a component set X that was found in Line 7 of the algorithm, and entails a merge to a new component ϕ_0. Now color the leafs of the tree according to W and B. Since both sets are non-empty there must exist an internal node τ in T, whose descendant leafs in the subtree are not colored identically. Let $B' = \{b_1, b_2, \ldots, b_p\}$ be the child components of τ which are in B, and let $W' = \{w_1, w_2, \ldots, w_q\}$ be the components which are children of τ and which are in W. Let X be the set corresponding to the descendant leafs of τ and note that $X = B' \cup W'$ and $|X| = p + q$. Since neither B' nor W' were merged earlier, the total communication cost among B' (resp. W') is at most $(p-1)\alpha$ (resp. $(q-1)\alpha$), which sums up to at most $(p+q-2)\alpha$. But since X is witnessing a communication cost of at least $(|X|-1)\alpha = (p+q-1)\alpha$, there must have occurred at least $(p+q-1)\alpha - (p+q-2)\alpha = \alpha$ communication cost between the nodes in B and the nodes in W during the current epoch.

Property 4. Consider *any* two non-empty disjoint subsets of components $U, V \subset Y$ s.t. $U \cup V = Y$ and $\mathrm{vol}(U) < k$ (i.e., U's components fit in one cluster). The inter-communication cost between U and V is at least α during the Y-epoch.

Proof (Proof of Property 4). The proof follows from the minimality of Y and because no merge involving components happened in Y. From the Y-epoch termination condition, we have $\mathrm{com}(Y) = \mathrm{com}(U) + \mathrm{com}(V) + \mathrm{inter}(U,V) \geq \mathrm{vol}(Y) \cdot \alpha$, where $\mathrm{inter}(U,V)$ is the inter-communication cost between U and V. Assume for the sake of contradiction that $\mathrm{inter}(U,V) < \alpha$. Recall that $\mathrm{vol}(U) \leq k$ and, since the components in U have not been merged yet, $\mathrm{com}(U) \leq (|U|-1) \cdot \alpha \leq (\mathrm{vol}(U)-1) \cdot \alpha$ (the equality holds when $|U| = 1$). We therefore have

$$\begin{aligned}\mathrm{com}(V) &= \mathrm{com}(Y) - \mathrm{com}(U) - \mathrm{inter}(U,V) \\ &> \mathrm{vol}(Y) \cdot \alpha - (\mathrm{vol}(U)-1) \cdot \alpha - \alpha = \mathrm{vol}(V) \cdot \alpha.\end{aligned}$$

We obtain the desired contradiction by distinguishing between two cases: (i) If $\mathrm{vol}(V) > k$, then $V \subset Y$ meets both termination conditions of a component epoch (Line 19), and thus the minimality of set Y is contradicted. (ii) Next, consider that $\mathrm{vol}(V) \leq k$ and notice that it must hold that $|V| > 1$ (otherwise $\mathrm{com}(V) = 0$). Since the components in V have not been merged yet, $\mathrm{com}(V) \leq (|V|-1) \cdot \alpha \leq (\mathrm{vol}(V)-1) \cdot \alpha \leq \mathrm{vol}(V) \cdot \alpha$, which is again a contradiction.

Proof (Proof of Lemma 2). We now use Properties 3 and 4 to lower bound Off's cost. First, it follows from Property 3 that Off cannot gain by splitting any component $\phi \in Y$ between different clusters (which would only increase its cost). The question is then, how much can Off reduce the inter-communication cost by arranging $\phi \in Y$ more efficiently? Let R_{intra} be the number of inter-communication requests (of CREP) Off did not pay by placing the components in an optimal way. Furthermore, denote by s the number of clusters Off used

and by $b_j \leq k$ the number of nodes Off placed in each cluster $j = 1, \ldots, s$. Consider any cluster j. In the best case for Off, each of the b_j nodes in the cluster is a singleton component which CREP placed in a different cluster. It follows that Off's saved cost cannot be greater than $(b_j - 1)\alpha$: otherwise CREP would have merged the b_j components into a single component. Observe also that, although Off aims to put the components in as few possible clusters, by a simple pigeonhole argument, s must be at least $\mathrm{vol}(Y)/k$. Combining the above, the number of requests R_{inter} Off serves remotely (assuming that no node was migrated during the Y-epoch) is

$$R_{inter} = \mathrm{vol}(Y) \cdot \alpha - R_{intra} \geq \mathrm{vol}(Y) \cdot \alpha - \sum_{j=1}^{s} (b_j - 1) \cdot \alpha$$
$$= \left(\mathrm{vol}(Y) - \sum_{j=1}^{s} b_j + \sum_{j=1}^{s} 1 \right) \cdot \alpha \geq \sum_{j=1}^{\mathrm{vol}(Y)/k} 1 = \frac{\mathrm{vol}(Y)}{k} \cdot \alpha,$$

where the last step follows from the fact that $\sum_{j=1}^{s} b_j = \mathrm{vol}(Y)$.

It remains to show that Off cannot decrease R_{inter} any further by swapping nodes during the Y-epoch. From Property 4, the nodes in each cluster j communicated at least α times with clusters $i \neq j$. Since any swap that Off performs between two clusters costs at least 2α, a swap between the involved clusters can only be beneficial to On. Considering that there are at least $s \geq \mathrm{vol}(Y)/k$ clusters, even with migrations, Off's cost will be at least $\mathrm{vol}(Y)/k \cdot \alpha$.

Incomplete Component Epoch. So far, we have quantified the cost that CREP and Off pay *at the end of each epoch*. It remains to account for the costs that CREP accumulates in incomplete epochs.

First, let us observe that the edge weights w of incomplete epochs in the component graph are naturally bounded: at some point, the edge will cause a merge, or end the epoch. By dividing the edges of the component graph G into *light edges* and *heavy edges*, we can claim the following:

Property 5. For every edge (ϕ_i, ϕ_j) in the component graph, at any given time:

1. If $|\phi_i| + |\phi_j| \leq k$ (we call this a *light edge*), the edge has cost at most α.
2. If $|\phi_i| + |\phi_j| > k$ (we call this a *heavy edge*), the edge cost is at most $(|\phi_i| + |\phi_j|)\alpha \leq (2k)\alpha$

The claim is implied by the definition of CREP. In the first case, if the (ϕ_i, ϕ_j) edge cost was larger than α, CREP would have merged ϕ_i and ϕ_j into a new component. Similarly, in the second case, if the edge (ϕ_i, ϕ_j) cost was larger than $(|\phi_i| + |\phi_j|)\alpha$, CREP would have ended the epoch.

Let us consider the request sequence σ at some time t. Recall that at the end of an epoch, we reset all involved edge weights, and charge Off for them. So at time t, we have not taken into account yet the communication requests that

were not reset. For any two nodes u and v, we consider all their communication requests since the last time they belonged to the same Y, at the end of a Y-epoch. All these requests belong to what we call the *last epoch*. Note that σ may not contain any complete epochs at all. But every request $\{u, v\} \in \sigma$ must belong to some Y-epoch or to the last epoch. Using Property 5 we obtain:

Lemma 3. *The competitive ratio of* CREP *for communication requests which belong to the* last epoch*, is bounded by* $O(k \log k)$.

The competitive ratio of CREP follows from Lemmas 1, 2, and 3.

6 Conclusion

This paper initiated the study of a natural dynamic partitioning problem which finds applications, e.g., in the context of virtualized distributed systems subject to changing communication patterns. We derived different upper and lower bounds, both for the general case as well as for a special case describing a matching problem. While the derived competitive ratios are sometimes linear or even super-linear in k, they do not depend on ℓ: We believe that this is attractive in practice: for example, while the number of servers in a datacenter (i.e., ℓ) can be large, the number of virtual machines hosted per server (e.g., the number of cores) is usually small. The main open question raised by our work regards the optimality of our upper bound: currently, the upper and lower bounds are off by a logarithmic factor. Moreover, it will be interesting to explore randomized settings: While we have some early positive results on the potential of randomization for special problem instances, the feasibility of $o(k)$-competitive randomized algorithms remains an open problem. .

References

1. A. Adamaszek, A. Czumaj, M. Englert, and H. Räcke. An $O(\log\ k)$-competitive algorithm for generalized caching. In *Proc. 23rd SODA*, pages 1681–1689, 2012.
2. M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. *ACM SIGCOMM CCR*, 2008.
3. K. Andreev and H. Räcke. Balanced graph partitioning. In *Proc. 16th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2004.
4. K. Andreev and H. Räcke. Balanced graph partitioning. *Theory of Computing Systems*, 39(6):929–939, 2006.
5. S. Arora, S. Rao, and U. Vazirani. Expander flows, geometric embeddings and graph partitioning. *Journal of the ACM (JACM)*, 56(2):5, 2009.
6. C. Avin, A. Loukas, M. Pacut, and S. Schmid. Online balanced repartitioning. In *arXiv* `https://arxiv.org/abs/1511.02074`, 2016.
7. B. Awerbuch, Y. Bartal, and A. Fiat. Competitive distributed file allocation. *Inf. Comput.*, 185(1):1–40, Aug. 2003.
8. N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Mach. Learn.*, 56(1-3):89–113, June 2004.
9. Y. Bartal, M. Charikar, and P. Indyk. On page migration and other relaxed task systems. *Theoretical Computer Science*, 268(1):43–66, 2001. Also appeared in *Proc. of the 8th SODA*, pages 43–52, 1997.

10. M. Bienkowski, A. Feldmann, J. Grassler, G. Schaffrath, and S. Schmid. The wide-area virtual service migration problem: A competitive analysis approach. *IEEE/ACM Transactions on Networking*, 22(1):165–178, 2014.
11. A. Borodin, N. Linial, and M. E. Saks. An optimal on-line algorithm for metrical task system. *Journal of the ACM*, 39(4):745–763, 1992. Also appeared in *Proc. of the 19th STOC*, pages 373–382, 1987.
12. M. Brehob, R. J. Enbody, E. Torng, and S. Wagner. On-line restricted caching. *Journal of Scheduling*, 6(2):149–166, 2003.
13. N. Buchbinder, S. Chen, and J. Naor. Competitive algorithms for restricted caching and matroid caching. In *Proc. of the 22thEuropean Symp. on Algorithms (ESA)*, pages 209–221, 2014.
14. M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica. Managing data transfers in computer clusters with orchestra. *SIGCOMM CCR*, 2011.
15. C. H. Q. Ding, X. He, H. Zha, M. Gu, and H. D. Simon. A min-max cut algorithm for graph partitioning and data clustering. In *Proc. IEEE International Conference on Data Mining (ICDM)*, pages 107–114, 2001.
16. L. Epstein, C. Imreh, A. Levin, and J. Nagy-György. Online file caching with rejection penalties. *Algorithmica*, 71(2):279–306, 2015.
17. U. Feige and R. Krauthgamer. A polylogarithmic approximation of the minimum bisection. *SIAM Journal on Computing*, 31(4):1090–1118, 2002.
18. A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, 1991.
19. A. Fiat, Y. Rabani, and Y. Ravid. Competitive k-server algorithms. *J. Comput. Syst. Sci.*, 48(3):410–428, 1994.
20. R. Krauthgamer and U. Feige. A polylogarithmic approximation of the minimum bisection. *SIAM Review*, 48(1):99–130, 2006.
21. A. Kumar, A. Gupta, and T. Roughgarden. A constant-factor approximation algorithm for the multicommodity rent-or-buy problem. In *Proc. 43rd Symposium on Foundations of Computer Science (FOCS)*, 2002.
22. Z. Lotker, B. Patt-Shamir, and D. Rawitz. Rent, lease or buy: Randomized algorithms for multislope ski rental. In *Proc. of the 25thSymp. on Theoretical Aspects of Computer Science (STACS)*, pages 503–514, 2008.
23. L. A. McGeoch and D. D. Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(6):816–825, 1991.
24. M. Mendel and S. S. Seiden. Online companion caching. *Theoretical Computer Science*, 324(2–3):183–200, 2004.
25. J. C. Mogul and L. Popa. What we talk about when we talk about cloud network performance. *ACM SIGCOMM CCR*, 42, 2012.
26. P. V. Ramanan, D. J. Brown, C. C. Lee, and D. T. Lee. On-line bin packing in linear time. *Journal of Algorithms*, 10(3):305–326, 1989.
27. S. E. Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27 – 64, 2007.
28. S. S. Seiden. On the online bin packing problem. *J. ACM*, (5), 2002.
29. D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.
30. N. E. Young. On-line caching as cache size varies. In *Proc. of the2ndACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 241–250, 1991.