Distrib. Comput. (2018) 31:83–98
https://doi.org/10.1007/s00446-017-0299-x

Distributed backup placement in networks

Magnús M. Halldórsson[1] · Sven Köhler[2] · Boaz Patt-Shamir[3] · Dror Rawitz[4]

Received: 8 August 2016 / Accepted: 30 March 2017 / Published online: 3 June 2017

Abstract We consider the *backup placement* problem, defined as follows. Some nodes (processors) in a given network have objects (e.g., files, tasks) whose backups should be stored in additional nodes for increased fault resilience. To minimize the disturbance in case of a failure, it is required that a backup copy should be located at a neighbor of the primary node. The goal is to find an assignment of backup copies to nodes which minimizes the maximum load (number or total size of backup copies) over all nodes in the network. It is known that a natural selfish local improvement policy has approximation ratio $\Omega(\log n/\log\log n)$; we show that it may take this policy $\Omega(\sqrt{n})$ time to reach equilibrium in the distributed setting. Our main result in this paper is a randomized distributed algorithm which finds a placement in polylogarithmic time and achieves approximation ratio $O\left(\frac{\log n}{\log\log n}\right)$. We obtain this result using a randomized distributed approximation algorithm for f-matching in bipartite graphs that may be of independent interest.

A preliminary version was presented at the 27th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA) 2015.
Magnús M. Halldórsson supported in part by Icelandic Research Fund (Grants Nos. 120032011 and 152679-051).
Sven Köhler supported in part by the Sustainability Center Freiburg, Germany, which is a cooperation of the Fraunhofer Society and the University of Freiburg and is supported by Grants from the Baden-Württemberg Ministry of Economics and the Baden-Württemberg Ministry of Science, Research and the Arts.
Boaz Patt-Shamir supported in part by the Ministry of Science, Technology and Space, Israel (Grant No. 3-10996) and the Israel Science Foundation (Grants No. 1444/14).
Dror Rawitz supported in part by the Ministry of Science, Technology and Space, Israel (Grant No. 3-10996) and the Israel Science Foundation (Grant No. 497/14).

✉ Sven Köhler
koehlers@informatik.uni-freiburg.de

Magnús M. Halldórsson
mmh@ru.is

Boaz Patt-Shamir
boaz@eng.tau.ac.il

Dror Rawitz
dror.rawitz@biu.ac.il

1 Reykjavik University, Reykjavík, Iceland
2 University of Freiburg, Freiburg, Germany
3 Tel Aviv University, Tel Aviv, Israel
4 Bar-Ilan University, Ramat Gan, Israel

1 Introduction

There are many scenarios in networks, e.g., in the context of cloud computing, where an object, such as a file or a task, resides in an unreliable processor. It is common practice in such cases to store a *backup copy* of the object at a nearby location (say, an adjacent node), so that in case of a failure in the primary location, the required service can be quickly switched over to the backup, thus minimizing unavailability time.

Clearly, not all backup placements are equal: consider, as the simplest example, an n-node fully connected network. If all nodes choose to place their backup copies at node 0 (except node 0, of course), then a fault at node 0 will make all backup copies unavailable. A much preferable placement would be for each node i to place its backup at node $(i+1) \bmod n$: this way, each node stores exactly one copy, which is the minimum possible in any case. To quantify this preference, one can measure the quality of a placement by the *maximum load*, i.e., the maximum, over all nodes, of the number (or total size) of copies they store, and the optimization goal is to minimize the maximum load.

Observe that there are cases which do not admit a good solution: if the network topology is a star, then all leaves must store their backup copies at the center node (in this case, all solutions are isomorphic). Therefore it makes sense to measure not the absolute maximum load of a given placement, but rather the *approximation ratio*, i.e., the ratio between the maximum load of a given placement and the smallest possible maximum load in the given topology, i.e., the maximum load of an optimal placement.

In this paper we study distributed solutions to the backup placement problem. To the best of our knowledge, the only known distributed algorithm for this problem is the very simple (and hence attractive) *local improvement* rule, which says that nodes act selfishly to decrease the load their backup copy sees [22]. For example, if the backup copy of node v resides in a neighbor u which stores 7 backup copies, and if v learns that its other neighbor w stores only 3 copies, then v will move its copy from u to w (assuming that moving an object is an atomic operation). It is known that the approximation ratio (called *price of anarchy* in this context) for this rule is $\Theta\left(\frac{\log n}{\log\log n}\right)$, where n is the number of nodes in the system [12]. While the upper bound on the approximation ratio may be acceptable, it turns out that the *dynamics* of the local improvement rule is problematic: as we show in this paper, the number of rounds required to reach equilibrium may be as large as $\Omega(\sqrt{n})$ in the worst case, even under the most optimistic assumptions on concurrency.

The main result in this paper is a randomized polylogarithmic-time distributed algorithm for backup placement, an exponential improvement over the running time over the selfish rule. The algorithm runs in the CONGEST model, i.e., execution proceeds in synchronous rounds in which nodes send small messages (see details in Sect. 2). The algorithm can be used in two ways: in one mode of operation, the algorithm guarantees approximation ratio $O\left(\frac{\log n}{\log\log n}\right)$. In the other mode of operation, the algorithm is fed a load bound L and a parameter $\epsilon > 0$, and it finds a placement with maximum load L which places at least a $(1-\epsilon)$-fraction of the maximum possible number of backups that can be placed under the given constraint of maximum load L (the running time in this case is also polynomial in ϵ^{-1}). In both cases, the algorithm can deal with the following generalizations of the problem:

- The primary locations may be any subset of nodes.
- The backup locations for each primary location may be any subset of nodes, assuming that each primary location can communicate with all its possible backup locations in $O(1)$ rounds.
- Objects may have different sizes, and the load of a node is the sum of the sizes of the copies it stores.
- Each object requires k backup copies in different locations, where $k \in \mathbb{N}$ is a given parameter.

Our algorithm is based on a randomized distributed approximation algorithm for f-matching in bipartite graphs that may be of independent interest. We provide hardness results and both minimum load and maximum coverage versions of the problem, and we also compare maximal solutions (i.e., solution that cannot be augmented) to maximum solutions in the maximum coverage case.

1.1 Paper organization

The remainder of this paper is organized as follows. In Sect. 2 we formalize the problem. Related work is given in Sect. 3. Section 4 discusses the sequential complexity of the problems. In Sect. 5 we analyze a few simple distributed algorithms and present our polylog-time algorithm. Section 6 relates maximal solutions to maximum solutions. We conclude in Sect. 7.

2 Model

Throughout this paper, we consider a simple undirected graph $G = (V, E)$, where V is the set of nodes and E is the set of edges. We use n to denote $|V|$ and Δ to denote the maximum node degree of G. An undirected edge between nodes v and u is denoted by (v, u) and we define (u, v) and (v, u) to denote the same edge. We use $N_G(u) \overset{\text{def}}{=} \{v \mid (u, v) \in E\}$ to denote the set of neighbors of a node u. If G is clear from the context, we simply write $N(u)$. The set of all positive integers is denoted by $\mathbb{N}$ and $\mathbb{N}_0$ is used to denote the set of all non-negative integers.

2.1 Problem statement

In the *backup placement problem* the input is as follows. We are given a set $C \subseteq V$ of *client* nodes and a set $S \subseteq V$ of *server* nodes. The sets may overlap, and their union need not be V. For each client $c \in C$ we are given a *size* (or *length*) $\ell(c) \in \mathbb{N}$ of the resource of which backups copies are to be stored. Let $\ell_{\max} \overset{\text{def}}{=} \max_{c\in C} \ell(c)$ denote the maximum size. We refer to an instance as *uniform* if all sizes are 1. In the non-uniform case, we assume that $\ell_{\max} = n^{O(1)}$. In addition, we are given a parameter $k \in \mathbb{N}$ called the *replication factor*.

The required output is a mapping $\beta : C \to 2^S$ which satisfies, for all $c \in C$, that $\beta(c) \subseteq S \cap N(c)$ and $|\beta(c)| \in \{0, k\}$. Henceforth we assume, without loss of generality, that $|S \cap N(c)| \geq k$ for every client $c \in C$ (otherwise we must set $\beta(c) = \emptyset$.) We say that a *backup copy* (a.k.a. *file*) of c is stored at s if $s \in \beta(c)$. A client $c \in C$ is said to be *satisfied* if $|\beta(c)| = k$, and *unsatisfied* otherwise. Let

$$\beta^{-1}(s) \overset{\text{def}}{=} \{c \in C \mid s \in \beta(c)\} ,$$

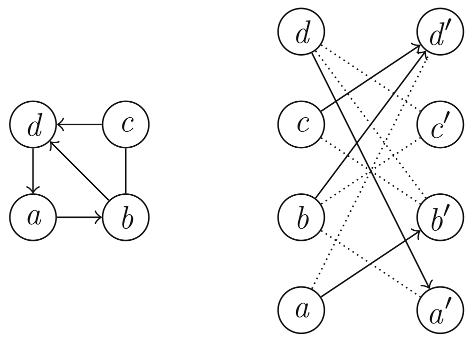


Fig. 1 *Left* A backup placement in a graph. All nodes are servers and clients, and $k = 1$. Placement is indicated by *arrows* from clients to servers. *Right* the corresponding bipartite graph

i.e., $\beta^{-1}(s)$ is the set of clients whose files are stored at s. Define the *load* of a server $s \in S$ by

$$L(s) \stackrel{\text{def}}{=} \sum_{c\in\beta^{-1}(s)} \ell(c) ,$$

i.e., $L(s)$ is the total size of backups stored at s. We call $\max\{L(s) \mid s \in S\}$ the *maximum load* of β.

We consider the following optimization goals for the backup placement problem:

- Minimum Load (MinLoad): Minimize the maximum load under the constraint that all clients must be satisfied.
- Maximum Coverage (MaxCov): Maximize the number of satisfied clients, under the constraint that the maximum load is no more than a threshold L (L is an additional input parameter).
- Maximum Satisfied Requests (MaxSR): Maximize the number of satisfied requests, i.e., maximize the number of placed backups, under the constraint that the maximum load is no more than a threshold L (L is an additional input parameter).

We mainly focus on the first, while the other two are mainly used as intermediate goals.

It is sometimes convenient to formalize the backup problem using a corresponding bipartite graph $G' = (C, S, E')$, where C is the set of clients, S is the set of servers (a node of G may be represented twice in G'), and E' connects a client c to a server s if and only if $(c, s) \in E$. See Fig. 1 for an example. Note that the definitions of backup placement and server load apply to the bipartite graph without change.

2.2 Execution model

We use the CONGEST model [28], which is a network model with small messages. Briefly, in this model nodes are processors with unique IDs, connected by links that can carry $O(\log n)$-bit messages in a time unit. Processors are not restricted computationally (all computations required by our algorithms are polynomial, though). As usual, for our upper bounds, we implicitly assume that the α-synchronizer [2] is employed in the system, so that the algorithms operate in a synchronous manner in the following sense. Execution proceeds in global *rounds*, where in each round each processor: (i) receives messages sent by its neighbors in the previous round, (ii) performs a local computation, and (iii) sends (possibly distinct) messages to its neighbors. Finally, we assume that n or an upper bound $N = O(n)$ is known to all nodes in the network.

3 Related work

MinLoad with a replication factor of 1 is a special case of the problem of *load balancing with restricted assignment*. In the centralized offline setting, the problem is optimally solvable in polynomial time using flow techniques (see Sect. 4). Most work on this problem concerns either online algorithms (see, e.g., [3]), or selfish agents (see, e.g., [30]). It is known that the simple (and centralized) on-line greedy algorithm where each job is assigned to the least loaded server is $(\lceil \log n\rceil + 1)$-competitive [4].

In the centralized setting, a special case of the backup placement problem, called "assigning papers to referees," was studied by Garg et al. [14]. In this problem it is assumed that each client ranks its available servers, and the goal of an algorithm is to find an assignment which maximizes a certain fairness measure defined in terms of the ranking (feasibility of load and coverage constraints is assumed to be trivial).

The "degree constrained subgraph" framework is a generalization of the backup placement problem considered in the centralized setting. In this family of problems the goal is to find a subgraph optimizing a certain measure (e.g., minimize number of edges) while conforming to given degree constraints (e.g., minimum degree at least some d). See [1] for relatively recent results and survey. *Semi-matchings*, introduced in [17], are more closely related to backup placement. Given a bipartite graph $G = (C, S, E)$, a semi-matching is a subset $E' \subseteq E$ of edges such that each client $c \in C$ is the endpoint of exactly one edge in E'. The cost of a semi-matching is defined by the L_p norm of the server load vector (the vector in which each node in S is assigned the number of E' edges it is incident with). In [17], a polynomial time algorithm to find an optimal semi-matching is presented, under any L_p norm, including L_∞ (i.e., minimizing the maximal load). A faster deterministic algorithm is provided in [11], and a randomized algorithm is presented in [13]. Low [25] considers weighted semi-matchings, where each client has a non-negative weight, and the load of a server is the sum of weights of clients assigned to it: under the L_∞ norm, it is shown that the weighted version is NP-hard, and a $\frac{3}{2}$-approximation algorithm is presented. All these algorithms are centralized.

Bokal et al. [6] introduce "(f, g)-quasi-matchings" and "(f, g)-semi-matchings." An (f, g)-quasi-matching of $G = (C, S, E)$ is a set $M \subseteq E$, such that each $c \in C$ is an endpoint of at least $f(c)$ edges and each $s \in S$ is end endpoint of at most $g(s)$ edges of M. In (f, g)-semi-matchings, also called f-matchings in general graphs [8], both functions f and g define an upper bound on the number of edges each node is an endpoint of. In the sequential setting, a maximum (f, g)-semi-matching of a bipartite graph can be found in polynomial time [21].

In the distributed setting, the basic building block used is fast approximate maximum matching algorithms. The best known algorithms are [5,24], both randomized (for deterministic algorithms, see [16,26]). Using a matching algorithm, Patt-Shamir et al. [27] present a polylogarithmic time algorithm for MaxCov with $k = 1$ which guarantees, for any $\epsilon > 0$ and with high probability, a $(1+\epsilon)\frac{2-r}{1-r}$-approximation, where r is the maximum ratio between a client demand and a server capacity. This result was extended by Halldórsson et al. [15] who give a $(1 + \epsilon)\frac{k+1-r}{1-r}$-approximation distributed algorithm for MaxCov with replication factor k. They also provide a distributed algorithm that computes solutions whose expected profit is an $\Omega(k^{-2})$-fraction of the optimum. A 2-approximation of MaxCov with $k = 1$ and uniform file sizes can be computed using the matching algorithm of Koufogiannakis and Young [23]. We note that [15,23,27] consider a more general version of MaxCov in which servers may have different capacities. Regarding MinLoad, we are only aware of distributed algorithms which use large messages (i.e., they run in the LOCAL model [28]): Czygrinow et al. [7] show how to find an approximation of optimal semi-matchings in $O(\Delta^5)$ rounds under the L_2 norm. The approximation ratio is $\min\left(2, \frac{2s+n}{n}\right)$, where s is the number of servers and n is the number of all nodes. They also provide a constant approximation LOCAL algorithm that runs in $O(\min\{\Delta^2, \Delta \log^4 n\})$ rounds.

4 Sequential complexity

In this section we study the complexity of the backup placement problem.

Theorem 1 MaxCov *with uniform file sizes and replication factor $k = 1$ is solvable in polynomial time.*

Proof The proof is by reduction to a maximum flow problem. Given the bipartite representation of the MaxCov instance, add a source s with links of capacity $k = 1$ connecting it to all clients, a sink t with links of capacity L connecting it to

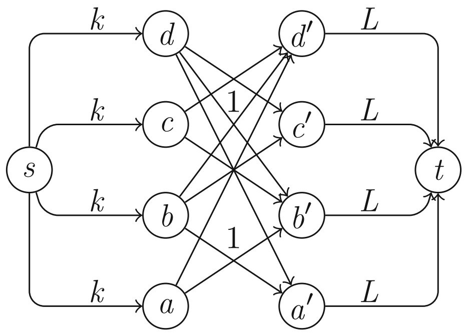


Fig. 2 The flow problem corresponding to the backup placement problem of Fig. 1

all servers, and assign capacity 1 to each edge in the original bipartite graph (see example in Fig. 2). Since capacities in the network are integral, we can find an integral optimal flow in polynomial time (e.g., with an augmenting path algorithm). Furthermore, since the in-degree of each client node is 1 and all capacities of edges incident to a client are 1, there is a one-to-one correspondence between integral flows and MaxCov solutions. It follows that an optimal solution can be found in polynomial time. □

However, if $k > 1$, the above reduction does not work since it maximizes the number of placed backups instead of maximizing the number of satisfied clients. Even worse, in the case of non-uniform sizes, the maximum flow may split a single backup copy between servers, which is not feasible in our setting. The following theorem shows that the failure of the reduction in these cases is no accident.

Theorem 2 MaxCov *with uniform file sizes is APX-complete for $k \geq 3$ and cannot be approximated to within $\Omega\left(\frac{\log k}{k}\right)$, unless* P = NP.

Proof The proof is by reduction from k-dimensional matching (k-DM) to MaxCov with replication factor k and uniform file sizes. In the k-DM problem, given a set of k-tuples $T \subseteq X_1 \times \cdots \times X_k$ where $X_1, \ldots, X_k$ are k disjoint sets, the task is to find the largest possible k-dimensional matching, namely a set $M \subseteq T$ such that if $x, y \in M$ then $x_i \neq y_i$, for every i. It is known that 3DM is APX-Complete [20] and that k-DM cannot be efficiently approximated to within a factor of $\Omega\left(\frac{\log k}{k}\right)$, unless P = NP [18].

Given instance T of k-DM as above, construct an instance of MaxCov with replication factor k as follows. There is a client for each k-tuple in T, and a server for each element of $\bigcup_j X_j$. There is an edge connecting each client $c = (c_1, \ldots, c_k) \in X_1 \times \cdots \times X_k$ to the corresponding k servers. In addition, we set $L := 1$.

Now, we claim that there is a k-dimensional matching M of size t in T if and only if there is a backup placement that satisfies t clients in the constructed instance of MaxCov: Given

M, we define a placement by letting $\beta(c) = \{c_1, \ldots, c_k\}$ for each $c = (c_1, \ldots, c_k) \in M$. This is a valid solution to MAXCOV because no server is assigned more than a single client, due to the fact that M is a k-dimensional matching. Conversely, given a backup placement β, each satisfied client c must be assigned to $\{c_1, \ldots, c_k\}$, and by the max load constraint, no other client can be assigned to these servers, and hence the set of satisfied clients corresponds to a k-dimensional matching. □

MINLOAD can be solved using the following reduction of MINLOAD to MAXCOV.

Lemma 3 *If an optimal solution to* MAXCOV *can be found in* T_{MAXCOV} *time, then an optimal solution to* MINLOAD *can be found in* $O(T_{\text{MAXCOV}} \log n)$ *time.*

Proof Given a MINLOAD instance, do a binary search over the value of L to find the minimal value which allows to cover all clients. To bound the number of times the MAXCOV algorithm is called, note that the largest L to consider is $n \cdot \ell_{\max} = n^{O(1)}$ by our assumption that sizes are polynomial in n. □

Using a reduction to max-flow, wrapped by a binary search as in the proof of Lemma 3, we get that MINLOAD is polynomial for uniform file sizes even for general replication factor.[1] However, for non-uniform file sizes, MINLOAD is strongly NP-hard, as stated next.

Theorem 4 MINLOAD *with non-uniform file sizes is strongly NP-hard, even for replication factor 1.*

Proof The proof is by reduction of the 3-partition problem (3PART) to the decision version of the client coverage problem with $k = 1$ and non-uniform file sizes. 3PART is defined as follows. The input consists of multiset X of $3m$ positive integers, for some $m \in \mathbb{N}$, and the question is whether we can partition X into m subsets such that all subsets have equal sums. 3PART remains hard even if $X \subseteq (Q/4, Q/2)^n$, where $Q = \frac{1}{m}\sum_{x \in X} x$. Note that if $X \in$ 3PART, then all subsets must be triplets.

The reduction is as follows. We construct a bipartite graph $G = (C, S, E)$, where $|C| = 3m$ and $|S| = m$. The client set C is defined as follows. For each element $x_i \in X$ we have a node $c_i \in C$, whose copy size is $\ell(c_i) = x_i$. In addition, all clients are connected to all servers. The load cap is Q. This completes the description of the reduction.

Suppose now that the multiset X can be partitioned into m equal-sum triplets. If we assign the clients corresponding to each triplets to a unique server, the total load on the server would be Q, namely there is a solution to the corresponding MINLOAD instance. Conversely, suppose there is a solution to the MINLOAD instance. Clearly it must be the case that each server has load *exactly* Q. Thus a solution to MAXCOV induces a solution to 3PART and we are done. □

Theorem 4 and Lemma 3 imply that MAXCOV is strongly NP-hard, but Theorem 2 provides a stronger result.

5 Distributed algorithms

In this section we consider distributed algorithms solving MINLOAD. First we show that simple solutions do not work well: simple randomized assignments may produce solutions with $\Omega(n)$ approximation ratio, and a simple selfish rule may yield $\Omega(\sqrt{n})$ running time for approximation ratio $\Omega\left(\frac{\log n}{\log\log n}\right)$. By contrast, we present a polylog-complexity algorithm for MINLOAD, derived by a series of reductions culminating in an algorithm for bipartite f-matching which may be of independent interest.

5.1 Random placements

It may be tempting to consider random placement as a solution to MINLOAD. While this method is extremely fast, we show that its approximation ratio is quite bad, even for replication factor 1. Let R_1 be the algorithm where each client selects a neighbor server uniformly at random.

Theorem 5 *R_1 has expected approximation ratio $\Omega(n)$.*

Proof We show that for every odd n there exists an instance where the expected maximum load generated by R_1 is $(n-1)/2$ whereas the optimal maximum load is 1. Consider a graph with a center node v_0 connected to $n-1$ nodes, such that nodes v_{2i-1} and v_{2i} are connected (see Fig. 3-left). Since each of the $n-1$ nodes chooses the center node with probability 1/2, the expected load on the center is $\frac{n-1}{2}$. However, an optimal solution with a maximum load of 1 exists: nodes v_{2i-1} and v_{2i}, for $i > 1$, select each other, while v_0, v_1, and v_2 select v_1, v_2, and v_0, resp. □

One may be further tempted to fix the problem of R_1 by extending the range which nodes consider. Specifically, let R_2 denote the algorithm where each client selects a neighbor server with probability inversely proportional to the neighbor's degree. While R_2 is not fooled by the simplistic example in the proof of Theorem 5, its approximation ratio is still linear, as we show next.

Theorem 6 *R_2 has expected approximation ratio $\Omega(n)$.*

Proof Assume w.l.o.g. that n is odd, and let $n' = \frac{n-1}{2}$. Define a graph with n nodes $\{a\} \cup \bigcup_{i=1}^{n'} \{b_i, c_i\}$, and edges

[1] The reduction works for general k because the target flow value is known in advance (it is nk). The proof of Theorem 2 uses instances that are not fully covered.

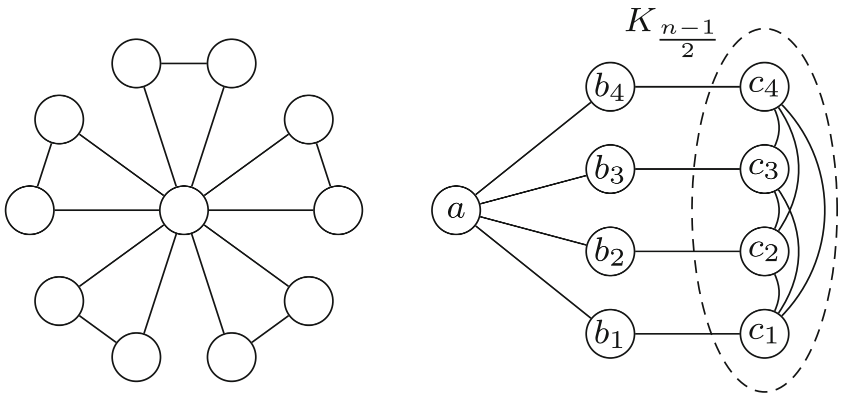


Fig. 3 Graphs used in the proofs of Theorem 5 (*left*) and Theorem 6 (*right*)

$\{(c_i, c_j) \mid i \neq j\} \cup \bigcup_{i=1}^{n'} \{(a, b_i), (b_i, c_i)\}$ (see Fig. 3-right for an example). The c_i nodes induce a complete graph with n' nodes. Since a and each c_i have degree exactly n', in R_2 node b_i selects node a with probability $\frac{1}{2}$, resulting in $n'/2 = \Omega(n)$ expected load on a. However, optimal backup placements on this graph have maximum load 1 if $n' > 1$ using the following assignment of copies: let $x \to y$ denote placing of x's copy in y. To obtain maximal load of 1, we use the assignment $a \to b_1$, $b_1 \to c_1$, $c_1 \to c_2$, $c_2 \to b_2$, $b_2 \to a$, and $b_i \leftrightarrow c_i$ for $i > 2$. (If $n' = 1$ then load 2 is unavoidable and achievable, say by $a \to b_1$, $b_1 \leftrightarrow c_1$.) □

5.2 The local improvement rule

We now turn our attention to the much-studied *local improvement* rule, defined as follows. Starting with an arbitrary placement, clients move their copy at will: a client moves one of its copies to another neighbor server if the resulting load at the target server is smaller than the load at the current server. Such a move is called a *local improvement step*. An assignment in which no such move is possible is called (Nash) equilibrium. A self-stabilizing distributed implementation of the local improvement rule is given in [22].

It is easy to see that the local improvement rule converges to a Nash equilibrium, in which no client wants to move its copy: the vector of sorted loads decreases lexicographically in each move. However, we now prove the following stronger result.

Theorem 7 *Starting at any initial backup placement, Nash equilibrium is reached in* $O(\ell_{\max}^2 kn\Delta_S)$ *local improvement steps, where* Δ_S *is the maximum server degree.*

Proof We generalize the proof of [22] to backups of arbitrary size. Define a potential function $\sum_{s\in S} L(s)^2$. A client that moves a backup of size ℓ from server s to t decreases the potential by $L(s)^2 + L(t)^2 - (L(s) - \ell)^2 - (L(t) + \ell)^2 = 2\ell(L(s) - L(t) - \ell)$. Since this is a local improvement step, it must hold that $L(s) > L(t) + \ell$ beforehand. Moreover, all file sizes and thus the server loads are integers. Thus the potential function decreases by at least $2\ell \geq 2$ with each local improvement step. The potential function attains its worst-case value if there are n clients and the backups are concentrated on as few servers as possible. In our case, a server can store at most Δ_S backups. So assume that $\lfloor kn/\Delta_S \rfloor$ servers store Δ_S backups each. Since backup sizes are at most $\ell_{\max}$, each of the $\lfloor kn/\Delta_S \rfloor$ servers has a load of at most $\ell_{\max}\Delta_S$. The $O(\ell_{\max}^2 kn\Delta_S)$ upper bound follows. □

Moreover, the following result is known regarding the price of anarchy for the local improvement rule.

Theorem 8 ([12]) *A Nash-equilibrium of the local improvement rule is a* $\Theta\left(\frac{\log n}{\log\log n}\right)$*-approximate backup placement for the case of replication factor* 1.

We complement the above results with some bad news, namely that there are scenarios in which a large number of steps is required to reach equilibrium.

Theorem 9 *Fix a replication factor* k. *For any given integers* n *and* Δ, *where* $8\max\{8, k\} \leq \Delta + 1 \leq n$, *there exists a graph with* n *nodes, maximum degree* Δ *and an initial backup placement such that* $\Omega(kn\sqrt{\Delta})$ *local improvement steps can be performed before an equilibrium is reached.*

Proof We first show an $\Omega(kn^{3/2})$ lower bound for a complete graph with n nodes. This is used as a building block to obtain the desired $\Omega(kn\sqrt{\Delta})$ lower bound.

Let G be a complete graph with $n \geq 8\max\{8, k\}$ nodes, where all nodes are both clients and servers. Then for the corresponding bipartite graph $G' = (C, S, E')$ we have $|C| = |S| = n$, where C and S are the client and server sets. Let all backups have size 1. The initial placement is as follows. Partition the servers into three sets S_h, S_c, and S_p with $|S_h| = k$, $|S_c| = c \overset{\text{def}}{=} \lceil\sqrt{n}\rceil$, and $|S_p| = n - k - c$. Let $s_1, s_2, \ldots, s_c$ denote servers in S_c. We place backups on these servers until the load of each $s_i \in S_c$ is equal to $k+c-i$. To do so, we repeatedly visit servers s_1 to s_c, placing a single backup on each server unless the desired load has been

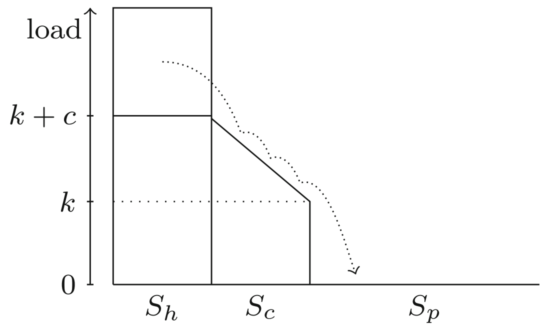


Fig. 4 Initial server load distribution used in the proof of Theorem 9

reached. We use backups of the same client as long as possible and pick a new client when needed. It can happen at most $k+c-1$ times, that not all backups of a client can be placed on servers of S_c. The remaining backups of such a client (at most $k-1$) are placed on the servers of S_h, giving priority to the servers having the minimum load. This way, at most $(k+c-1)(k-1) \leq kc$ backups are placed on servers of S_h. As the load is evenly distributed among these servers, no server of S_h has a load larger than $k+c-1$. More backups are placed on the servers of S_h until all of them have a load equal to $k+c$. None of the backups that have been placed so far will be moved by a local improvement step in our example.

The number of unplaced backups we are now left with is

$$\begin{aligned} h &\overset{\text{def}}{=} kn - \left(k(k+c) + kc + \frac{c(c-1)}{2}\right) \\ &= kn - k^2 - 2kc - \frac{c(c-1)}{2} \\ &\geq kn - k^2 - 2k(\sqrt{n}+1) - \frac{(\sqrt{n}+1)\sqrt{n}}{2} \\ &= kn - k^2 - 2k\sqrt{n} - 2k - \frac{n}{2} - \frac{\sqrt{n}}{2} \\ &\geq kn - \frac{kn}{8} - \frac{kn}{4} - \frac{kn}{32} - \frac{kn}{2} - \frac{kn}{16} \\ &= \frac{kn}{32}\,. \end{aligned}$$

We distribute these backups evenly among the servers of S_h. No backup is placed on any server in S_p or S_c. The resulting server load distribution looks similar to what is shown in Fig. 4. Starting at this initial placement, as many of the S_h backups as possible are moved to servers of S_p. For each backup we use a sequence of local improvement steps such that the backup visits the servers s_1 to s_c exactly once. Note that the servers of S_p can hold all the backups of a client as

$$\begin{aligned} |S_p| &= n - k - c \\ &\geq n - 2k - \sqrt{n} - 1 + k \\ &\geq n - \frac{n}{4} - \frac{n}{8} - 1 + k \\ &= \frac{5}{8}n - 1 + k \\ &\geq k\,. \end{aligned}$$

Since s_c has a load of k, at most $k(n-k-c)$ backups can be moved from s_c to servers in S_p as their load grows over time. Summing up, exactly $b \overset{\text{def}}{=} \min(h, k(n-k-c)) = h$ backups can be moved from the servers in S_h to servers in S_p. Since each backup visits each server of S_c once, we have c local improvement steps per backup yielding $b \cdot c \in \Omega(kn\sqrt{n})$ improvement steps in total.

To create a graph with n' nodes and degree Δ, for $8\min\{8,k\} \leq \Delta+1 \leq n'$, we use the complete graph described above with $\Delta+1$ nodes as a building block and create $\lfloor n'/(\Delta+1)\rfloor$ copies of it. Summing up the number of local improvement steps over all building blocks, we then obtain the claimed lower bound of $\Omega(kn'\sqrt{\Delta})$. □

We note that Theorem 9 is an improvement over the previously known lower bound, which cannot exceed $\Omega\left(n\frac{\log^2 n}{\log\log n}\right)$ in an n-node graph [10]. A construction similar to Fig. 4 is used in [9] to prove an $\Omega(n\sqrt{n})$ lower bound for the problem inverse to ours with $k=1$.

The local improvement rule is sequential: one improvement at a time. For a distributed implementation, we assume that the local improvement steps are done *atomically* and in *adversarial* order. By atomically we mean that a server cannot participate in two concurrent improvement steps. This is required to make sure that each step is indeed an improvement (otherwise non-termination may occur due to oscillations [22]). The adversarial order is for the sake of studying the worst case.

Note that to implement local improvement, one has to somehow synchronize the clients so that at most one copy is removed or added to a server in a step. Otherwise, for example, it may be the case that a server node v had 0 load in round t, driving all its neighbor clients to move their copies to it, resulting in load $\deg(v)$ in round $t+1$. But even ignoring this difficulty, and assuming the most optimistic assumption about concurrency, namely that each server is able to perform one local improvement step per parallel round, only $O(n)$ local improvement steps would be performed in a single round. Thus Theorem 9 yields the following immediate corollary.

Corollary 10 *For any given k and for any given integers n and Δ, where $8\max\{8,k\} \leq \Delta+1 \leq n$, there are instances of graphs with n nodes, maximum degree Δ, and replication factor k, such that for some sequences of the local rule the worst-case distributed running time is $\Omega(k\sqrt{\Delta})$.*

We note that the self-stabilizing algorithm of [22] meets the lower bound of Corollary 10.

5.3 Polylog-time approximation algorithm

We now present our main result, namely an algorithm for the backup placement problem in the CONGEST model.

We start with a standard classify-and-select reduction that allows us to reduce non-uniform file sizes to the case of uniform file sizes, and whose cost is an additional $O(\log n)$ factor in both the running time and approximation ratio. The reduction works for general k. Recall that we assume that $\ell_{\max} = n^{O(1)}$.

Lemma 11 *Let Δ_S denote the maximum server degree in an instance. If* MinLoad *can be solved for uniform file size and replication factor k in time T_k and approximation ratio α_k, then* MinLoad *can be solved for non-uniform file size and replication factor k in time $O(T_k \log \Delta_S)$ and approximation ratio $O(\alpha_k \log \Delta_S)$.*

Proof As a first intuitive step, assume that $\ell_{\max}$ and Δ_S are known to each node. Let OPT denote the optimal load for the non-uniform MinLoad instance. We classify all clients according to their file size: class i contains all clients with file sizes in $(2^{i-1}, 2^i]$. We consider the top $\lceil \log \Delta_S \rceil$ classes and solve MinLoad for each class independently, assume that all file sizes are equal to 2^i. Let ALG$_i$ denote the maximum load of the solution for class i and let OPT$_i$ denote the optimum load. Observe that OPT$_i \leq 2$OPT as we assume at most twice the original file size. Thus it follows that ALG$_i \leq \alpha$OPT$_i \leq 2\alpha$OPT. Merging the solutions for the top Δ_S classes, the load on each server is at most $2\alpha_k \lceil \log \Delta_S \rceil \cdot$ OPT. The remaining files have small sizes of less than $\frac{2}{\Delta_S}\ell_{\max}$ and are called *globally small*. These files are placed arbitrarily. This adds at most load $2\ell_{\max}$ per server as a single server has at most Δ_S adjacent clients. The result follows.

We now show how to avoid the assumption that nodes know $\ell_{\max}$ or Δ_S. The idea is that each client determines whether its file is *locally small* by comparing it to the maximum file size within its $2T_k$-hop neighborhood instead of to $\ell_{\max}$. Since $\ell_{\max}$ is an upper bound, locally small files are also globally small and can be placed on an arbitrary adjacent server. Otherwise, the client places its file according to its class.

Consider the bipartite graph $G = (C, S, E)$ where C is the set of clients and S is the set of servers. W.l.o.g. let C and S be disjoint and we define $V \overset{\text{def}}{=} C \cup S$.

We classify all clients according to their file size: class $i \in \mathbb{N}_0$ contains all clients with file sizes in $(2^{i-1}, 2^i]$. For a client $c \in C$, let *class*(c) denote the index of its class. For convenience we define *class*$(s) = 0$ for every server $s \in S$. Since $\ell_{\max}$ is assumed to be in $n^{O(1)}$, there are $O(\log n)$ classes.

Let dist(v, u) denote the length of the shortest path between nodes v and u and deg(v) the degree of node v. For a client $c \in C$ we define

$$\Delta_S(c) \overset{\text{def}}{=} \max\{\deg(s) \mid s \in S \wedge \operatorname{dist}(c, s) = 1\}$$

and for every node $v \in V$ and $r \in \mathbb{N}_0$ we define

$$cl_{\max}(v, r) \overset{\text{def}}{=} \max\{class(u) \mid \operatorname{dist}(v, u) \leq r\}\,.$$

We call a client $c \in C$ an *initiator* if and only if

$$class(c) > cl_{\max}(c, 2T_k) - \lceil \log \Delta_S(c) \rceil\;.$$

Let $I \subseteq C$ be the set of all initiators. For a node $v \in V$ and $r \in \mathbb{N}_0$ we define

$$Classes(v, r) \overset{\text{def}}{=} \{i \mid \exists c \in I : \operatorname{dist}(v, c) \leq r \wedge class(c) = i\}\,.$$

We claim that $|Classes(v, r)| \leq \lceil \log \Delta_S \rceil$ for all nodes $v \in V$ and any $r \leq T_k$. Suppose that there is a node $v \in V$ and a non-negative integer $r \leq T_k$ such that $|Classes(v, r)|$ exceeds the given bound. Then there are two initiators u and w within distance r of v such that

$$class(u) \leq class(w) - \lceil \log \Delta_S \rceil\;.$$

It holds that dist$(u, w) \leq 2r \leq 2T_k$ due to the triangle inequality. Therefore $cl_{\max}(u, 2T_k) \geq class(w)$ and, since u is an initiator, it holds that

$$\begin{aligned} class(u) &> cl_{\max}(u, 2T_k) - \lceil \log \Delta_S(u) \rceil \\ &\geq class(w) - \lceil \log \Delta_S \rceil\,. \end{aligned}$$

That is a contradiction to the above upper bound on *class*(u).

Now assume that every node v knows the value of *Classes*(v, T_k) and all clients know whether they are initiators. We postpone the discussion on how to compute that efficiently to the end of the proof. Let A be an algorithm that, for replication factor k and uniform files sizes, computes an α_k-approximate backup placement in at most T_k rounds. We define

$$V_i \overset{\text{def}}{=} \{v \in V \mid i \in Classes(v, T_k)\}\,.$$

Fix a class index $i \in \mathbb{N}_0$. Let β_i denote a placement that only considers the files of initiators of V_i. We compute β_i using algorithm A, pretending that files sizes are uniform. We also exploit the fact that $\beta_i(c)$ for an initiator $c \in V_i$ merely depends on inputs within distance T_k of c. By construction, V_i contains all nodes within distance T_k of c. Thus it suffices that only the nodes in V_i participate in the simulation of A.

We compute the placements β_i for all classes i in parallel. A single node v participates in the simulation of A for at most $|Classes(v, T_k)|$ classes. As shown above, we have that $|Classes(v, T_k)| \in O(\log \Delta_S)$. Therefore, it follows that the parallel simulation transmits at most $O(\log \Delta_S)$-times the amount of data as a single execution of A. Thus, all β_i can be computed in parallel in time $O(T_k \log \Delta_S)$ in the CONGEST model.

Each β_i is a $2\alpha_k$ approximation. A server $s \in S$ receives files by at most $O(\log \Delta_S)$ distinct β_i since $|Classes(s, T_k)| \in O(\log \Delta_S)$. Thus, the union of all β_i has an approximation ratio of $O(\alpha_k \log \Delta_S)$. Clients that are not initiators place their files arbitrarily. Let c be such a client and let $s \in S$ be a server adjacent to c. Then we have

$$\begin{aligned}\ell(c) \le 2^{class(c)} &\le 2^{cl_{\max}(c,2T_k)-\lceil \log \Delta_S(c)\rceil} \\ &\le 2^{\lceil \log \ell_{\max}\rceil - \log \deg(s)} \\ &\le 2\ell_{\max}/\deg(s)\ .\end{aligned}$$

Since a server s cannot be assigned more than $\deg(s)$ files, it follows that this adds at most a load of $2\ell_{\max}$ per server. The claimed approximation ratio follows.

We now turn to computing the value of $Classes(v, T_k)$ and whether a client is an initiator or not. The algorithm for that is composed of three phases. Phase 1 is as follows:

- Every server $s \in S$ sends $\deg(s)$ to all adjacent clients.
- Each client remembers the maximum value it received.

Clearly, all clients $c \in C$ know the value of $\Delta_S(c)$ at the end of Phase 1. Phase 1 takes $O(1)$ rounds to complete. Phase 2 computes $cl_{\max}(c, 2T_k)$ for all clients $c \in C$. Assume that nodes have an integer variable $cl_{\max}$.

```
– Every node $v \in V$ sets $cl_{\max} := class(v)$.
– For $i = 1, 2, \dots, 2T_k$ do
    1. All nodes broadcast $cl_{\max}$ to their neighbors.
    2. All nodes update $cl_{\max}$ with the largest value received
       if a value larger than $cl_{\max}$ was received.
```

Observe that $cl_{\max} = class(v) = cl_{\max}(v, 0)$ before the loop. By induction on i, it follows that $cl_{\max}$ of node v is equal to $cl_{\max}(v, i)$ after i iterations of the loop. Phase 2 takes at most $O(T_k)$ rounds since $O(\log \log n)$ bits are transmitted per edge in each iteration. After phases 1 and 2, a client c knows the values of $\Delta_S(c)$ and $cl_{\max}(c, 2T_k)$ and thus can determine whether it is an initiator. The goal of phase 3 is to compute the value of $Classes(v, T_k)$ for all nodes $v \in V$. Assume that each node has a variable *Classes* which is a set of class indexes.

```
– Every initiator $c \in I$ sets $Classes := \{class(c)\}$ and all
  other nodes set $Classes := \emptyset$.
– For $i = 1, 2, \dots, T_k$ do
    1. All nodes broadcast $Classes$ to their neighbors.
    2. Every node adds the elements of all sets received to
       its $Classes$ variable.
```

Again, by induction on i, it follows that the variable *Classes* of every node v is equal to $Classes(v, i)$ after i iterations. Note that before the first iteration of the loop we have $Classes = \{class(c)\} = Classes(v, 0)$. As shown above, $|Classes(v, i)|$ cannot exceed $\lceil \log \Delta_S \rceil$ for $i \le T_k$. Hence $O(\log \Delta_S \log \log n)$ bits are transmitted over an edge per iteration. Therefore, the running time of phase 3 is $O(T_k \log \Delta_S)$ rounds. □

Let us consider now the problem of minimizing the maximum load with uniform sizes and replication factor k. Fix the instance graph. Given a value L, let MAXCOV(L) denote the number of clients that can be satisfied when the instance is viewed as an instance of MAXCOV with server load cap L. We say that an algorithm A *α-solves* MAXCOV, for some parameter $\alpha \ge 1$, if for any given $L \ge 0$, A finds a placement which places at least $k \cdot MaxCov(L)$ files, but with relaxed maximum load αL.

Lemma 12 *If* MAXCOV *with uniform file size and replication factor k can be α-solved in time T_c for any L, then* MINLOAD *with uniform file size and replication factor k can be solved in time $O(T_c \log n)$ and approximation ratio 4α.*

Proof Let A be an algorithm which α-solves the instance in time T_c. We use the following algorithm:

```
– For $i = 0$ to $\log n$ do
    1. Run $A$ with parameter $L = 2^i$. Obtain solution $\beta_i$
       which places at least $k \cdot$ MAXCOV$(2^i)$ files.
    2. Add $\beta_i$ to the output, and remove all edges used by
       $\beta_i$ from further consideration.
```

Let OPT be the optimal load for the given MINLOAD instance. Clearly, when $L \ge$ OPT, the algorithms can place all remaining files. Therefore all files are placed by some β_i for some $i \le \lceil \log \mathrm{OPT} \rceil$. Hence the algorithm produces all its output by time $O(T_c \log \mathrm{OPT}) = O(T_c \log n)$. Since each β_i with $i \le \lceil \log \mathrm{OPT} \rceil$ is a backup placement with maximum server load at most $\alpha 2^i$, the load of the union of all placements is at most $\sum_{i=0}^{\lceil \log \mathrm{OPT} \rceil} \alpha 2^i \le 4\alpha \mathrm{OPT}$. □

Finally, we reduce the problem of α-solving MAXCOV to finding approximate solutions of $MaxSR$. We use a reduction of $MaxSR$ to the f-matching problem. Given a graph $G = (V, E)$ and a mapping $f : V \to \mathbb{N}_0$, an *f-matching* is a subset $M \subseteq E$ such that each node $v \in V$ is incident to at most $f(v)$ edges of M. The goal is to find such a set M of maximum size. In the remainder of this section, ordinary matchings, where $f(v) = 1$ for all nodes $v \in V$, will be called *1-matchings*.

Lemma 13 *If f-matching can be approximated to within $(1-\frac{1}{x})$ on bipartite graphs in time T_m for some $x > 1$, then* MaxCov *with uniform file size, replication factor k, and maximum load L can be $O(\log_x n)$-solved in time $O(T_m \log_x n)$.*

Proof Let A be an algorithm obtaining, in T_m time, $(1-\frac{1}{x})$-approximate f-matchings. For each client c we set $f(c) := k$ and for each server s we set $f(s) = L$. We set $t = \lfloor \log_x kn \rfloor + 1$ and use the following algorithm:

- For $i = 1$ to t do
  1. Run A and obtain f-matching M_i.
  2. Add M_i to the output, remove all edges used by M_i, and for each client c, decrease $f(c)$ by the number of edges of M_i incident to c.

Let OPT denote the set of edges used by an optimal solution of MaxCov, and let ALG(i) denote the set of edges used by the algorithm after i iterations, that is $\mathrm{ALG}(i) \overset{\text{def}}{=} \bigcup_{j=1}^{i} M_j$. A maximum f-matching computed in iteration $i+1$ would contain at least $|\mathrm{OPT} \setminus \mathrm{ALG}(i)|$ edges. However, since M_i is $(1-\frac{1}{x})$-approximate, we have

$$\begin{aligned}|\mathrm{ALG}(i+1)| &\geq |\mathrm{ALG}(i)| + \left(1-\frac{1}{x}\right)|\mathrm{OPT} \setminus \mathrm{ALG}(i)| \\ &\geq |\mathrm{ALG}(i)| + \left(1-\frac{1}{x}\right)(|\mathrm{OPT}| - |\mathrm{ALG}(i)|) \\ &= |\mathrm{OPT}| - \frac{1}{x}(|\mathrm{OPT}| - |\mathrm{ALG}(i)|)\end{aligned}$$

which yields

$$|OPT| - |\mathrm{ALG}(i+1)| \leq \frac{1}{x}(|OPT| - |\mathrm{ALG}(i)|)\,.$$

The value of t is chosen such that $|\mathrm{OPT}| - |\mathrm{ALG}(t)| < 1$. This is true for any $t > \log_x kn$ since $|\mathrm{OPT}| \leq kn$. This immediately proves the claimed time bound. Regarding the approximation ratio, note that each M_i increases the max load by at most L since it matches a server with at most L clients. □

By the chain of reductions above, it suffices to focus on f-matching alone. We present an approach for computing approximate f-matchings based on the algorithm of [24] and a reduction of f- to 1-matchings by Shiloach [29].

The basic idea in the algorithm of [24] is to eliminate all augmenting paths up to a certain length: Hopcroft and Karp [19] show how to translate a lower bound on the length of augmenting paths to a lower bound on the approximation ratio in 1-matching. Using Shiloach's reduction, we generalize the result of [19] to f-matchings.

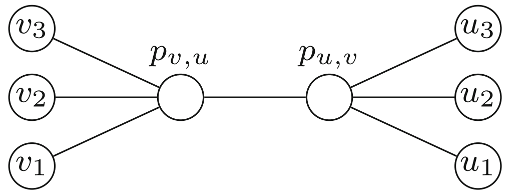


Fig. 5 Subgraph of the f-replicated graph corresponding to an edge (v, u), where $f(v) = f(u) = 3$

We start by generalizing the notions of augmenting paths and of independent set of augmenting paths to the context of f-matchings. Let M be an f-matching in a graph $G = (V, E)$. Let $\deg_M(v)$ denote the number of edges of M incident with $v \in V$. A node v is called *saturated* if $\deg_M(v) = f(v)$ and *unsaturated* if $\deg_M(v) < f(v)$. An *augmenting path* of M is a trail of odd length that starts and ends at unsaturated nodes and alternates between edges of $E \setminus M$ and M. (A *trail* may visit nodes multiple times but its edges are distinct.) An augmenting path of M may start and end in the same node v, but only if $\deg_M(v) \leq f(v) - 2$.

When augmenting an f-matching M with an augmenting path p, the edges of p with an odd index are added to M and the edges with an even index are removed from M. Observe that $\deg_M(v)$ of a node v increases by exactly *gain*(v, p) defined as follows:

$$gain(v,p) \overset{\text{def}}{=} \begin{cases} 2 & \text{if } v \text{ is start- and end-node of } p \\ 1 & \text{if } v \text{ is either start or end-node of } p \\ 0 & \text{otherwise} \end{cases}$$

Using this function, we now give a definition of independent sets of augmenting paths.

Definition 1 A set P of augmenting paths of an f-matching M is called *independent* if the augmenting paths in P are pairwise edge-disjoint and for all nodes v,

$$\sum_{p \in P} gain(v,p) \leq f(v) - \deg_M(v)\,.$$

If P is an independent set of augmenting paths, then a node v can occur as an internal node of the trails of P at most $\deg_M(v) \leq f(v)$ times. This is due to the edge-disjointness requirement, since there are exactly $\deg_M(v)$ edges of M incident to a node v. This means, for example, that in the case of 1-matchings (but not in general), all paths in an independent set are node-disjoint.

Shiloach [29] showed that every maximum f-matching in a graph $G = (V_G, E_G)$ has a corresponding maximum 1-matching in a graph $H = (V_H, E_H)$ which is obtained by replacing every edge of G with the subgraph shown in Fig. 5. We call H the *f-replicated graph*. It is formally defined as follows:

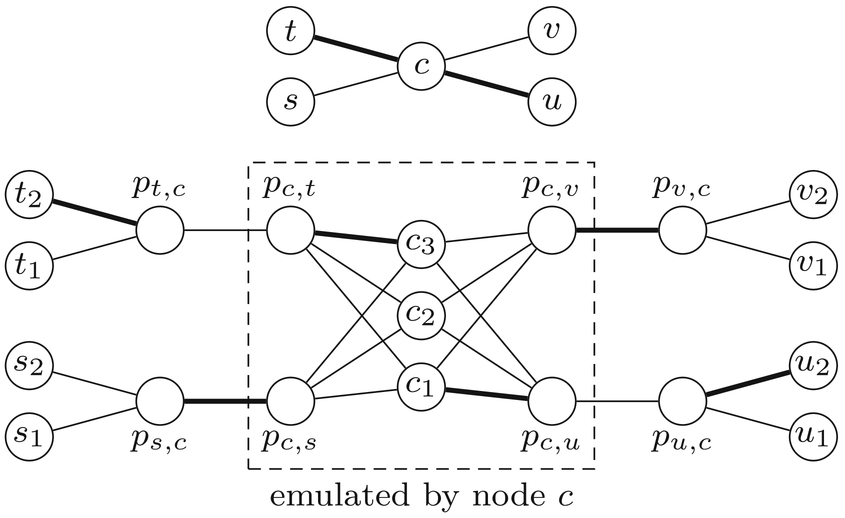


Fig. 6 An f-matching (*top*) and a corresponding matching in the replicated graph (*bottom*) with $f(c) = 3$ and $f(s) = f(t) = f(u) = f(v) = 2$

$$V_H \overset{\text{def}}{=} \{v_i \mid v \in V_G \wedge i \in [1, f(v)]\} \\ \cup \{p_{v,u} \mid v \in V_G \wedge u \in N_G(v)\}$$

$$E_H \overset{\text{def}}{=} \{(v_i, p_{v,u}) \mid v \in V_G \wedge u \in N_G(v) \wedge i \in [1, f(v)]\} \\ \cup \{(p_{v,u}, p_{u,v}) \mid (v, u) \in E_G\}$$

The nodes v_i with $v \in V_G$ and $i \in [1, f(v)]$ are called *virtual copies* of v. A node $p_{v,u}$ with $(v, u) \in E_G$ is called a *port node* of v. An edge between two port nodes is called *physical edge*. If f is clear from the context, then f is omitted and H is just called the replicated graph of G.

As we will show, any f-matching M_G has a corresponding 1-matching M_H in the replicated graph such that if M_H has an independent set of augmenting paths, then M_G has an independent set of augmenting paths of the same size. This holds true even if we require M_H to be a *normalized* 1-matching, where a 1-matching is called normalized if it saturates all port nodes. An f-matching M_G of a graph $G = (V_G, E_G)$ and a normalized 1-matching M_H in the replicated graph H are called *corresponding* if $(c, s) \in M_G$ if and only if $(p_{c,s}, p_{s,c}) \notin M_H$. An example of an f-matching and the corresponding 1-matching is shown in Fig. 6.

Lemma 14 *Let M_G be an f-matching of a graph $G = (V_G, E_G)$. Then M_G has a corresponding 1-matching M_H in the replicated graph H with $|M_H| = |E_G| + |M_G|$.*

Proof Start with the normalized 1-matching

$$M_H = \{(p_{v,u}, p_{u,v}) \mid (v, u) \in E_G\}\ .$$

For each edge $(v, u) \in M_G$, augment M_H with a path $v_i, p_{v,u}, p_{u,v}, u_j$ where i and j are chosen such that v_i and u_j are unsaturated. Clearly, the result is a normalized matching and it contains exactly $|E_G| + |M_G|$ edges. Note that H contains enough unsaturated virtual copies of each node of G for this procedure to complete. □

Lemma 15 *Every augmenting path of a normalized 1-matching M in a replicated graph $H = (V_H, E_H)$ contains an odd number of physical edges and these edges alternate between M and $E_H \setminus M$.*

Proof Fix an augmenting path p of M. Recall that p must start at an unsaturated node. Since H is normalized, all unsaturated nodes are virtual copies. Also, any trail of H of odd length that starts at a virtual copy and does not include a physical edge ends at a port node. Hence, p must contain a physical edge.

Also note that any trail of H that starts at a virtual copy and ends with its first physical edge is of even length. Thus, the first physical edge of p must be in M. By symmetry, the last edge of p must also be in M. Furthermore, any trail of H that starts with a physical edge and ends with its second physical edge is of even length. Thus, the physical edges of p must alternate between M and $E_H \setminus M$. □

Lemma 16 *Let M_G be an f-matching in a graph G and let M_H be a corresponding 1-matching in the replicated graph H. If there is an independent set P_H of augmenting paths of M_H, then there is an independent set P_G of augmenting paths of M_G such that $|P_G| = |P_H|$.*

Proof The set P_G contains one augmenting path p_G for each $p_H \in P_H$. In the remainder of the proof, we describe how to construct p_G from p_H and show that P_G is indeed independent.

Let p'_H be the subsequence of p_H that contains all physical edges of p_H. Each physical edges $(p_{v,u}, p_{u,v})$ of p'_H corresponds to the edge $(v, u) \in E_G$. We set p_G to the sequence obtained by replacing each edge of p'_H with the corresponding edge of E_G.

By Lemma 15, the subsequence of physical edges of p_H alternates between M_H and $E_H \setminus M_H$ and is of odd length. Thus, p_G is of odd length and alternates between $E_G \setminus M_G$

and M_G since M_G and M_H are corresponding. Also, as p_H contains each physical edge at most once, p_G contains each edge at most once. Thus p_G is a valid augmenting path.

The paths in P_G are edge-disjoint as the paths in P_H are edge-disjoint and each physical edge of H corresponds to a different edge of G. It remains to show that for all $v \in V_G$, $\sum_{p \in P_G} gain(v, p) \leq f(v) - \deg_{M_G}(v)$. Fix a node $v \in V_G$. The matching M_H saturates exactly $\deg_{M_G}(v)$ virtual copies of v. The remaining $f(v) - \deg_{M_G}(v)$ virtual copies are unsaturated. We proceed by mapping $gain(v, p_G)$ unsaturated virtual copies of v to each $p_G \in P_G$ which proves the claim.

By construction, p_G with $gain(v, p_G) = 2$ was derived from an augmenting path p_H that starts at v_i and ends at v_j, both distinct unsaturated virtual copies of v. We map v_i and v_j to p_G. If $gain(v, p_G) = 1$, then p_G was derived from an augmenting path p_H that either starts or ends at a virtual copy v_i. We map v_i to p_G. Note that since M_H is a 1-matching, the paths in P_H are vertex-disjoint and thus no virtual copy of v is mapped twice. □

With the above results in place, we now proceed to prove the following generalizations of results in [19].

Lemma 17 *Let M_G and N_G be f-matchings of a graph G. If $|M_G| = r$, $|N_G| = s$, and $s > r$, then there is an independent set of at least $s - r$ augmenting paths of M.*

Proof Let H be the replicated graph of G. Lemma 14 implies that there are two 1-matchings M_H and N_H that correspond to M_G and N_G, resp., and it holds $|N_H| - |M_H| = s - r$. As this lemma is known to hold for 1-matchings [19], we have that M_H has an independent set of at least $s - r$ augmenting paths. Hence, M_G has an independent set of at least $s - r$ augmenting paths by Lemma 16. □

Theorem 18 *Let M be an f-matching of a graph G and let s be the cardinality of a maximum f-matching of G. Then M has an augmenting path of length at most $2\lfloor r/(s-r) \rfloor + 1$ with $r = |M|$.*

Proof By Lemma 17 there is an independent set P of at least $s - r$ augmenting paths of M. Since the augmenting paths of P are pairwise edge-disjoint, P contains at most r edges from M. On average, an augmenting path in P contains at most $r/(s-r)$ edges of M. Thus there is at least one augmenting path p of P that contains at most $\lfloor r/(s-r) \rfloor$ edges of M. As the edges of p alternate between $V \setminus M$ and M, the length of p is at most $2\lfloor r/(s-r) \rfloor + 1$. □

Theorem 18 yields the following relation between the approximation ratio and the length of the shortest augmenting path of an f-matching.

Corollary 19 *If the shortest augmenting path of an f-matching M has length $2x - 1$ for some $x \in \mathbb{N}$, then the approximation ratio of M is at least $(1 - \frac{1}{x})$.*

Proof Let $r = |M|$ and let s be the size of a maximum f-matching. By Theorem 18 we have $2x - 1 \leq 2\lfloor r/(s-r) \rfloor + 1$. It follows that $x - 1 \leq r/(s-r)$ and that $r \geq \left(1 - \frac{1}{x}\right) s$ which proves the claim. □

So as in the case of 1-matchings, the approximation ratio of f-matchings is related to the length of the shortest augmenting path. We now show that this relation carries over to the length of the shortest augmenting path of the corresponding 1-matching in the replicated graph.

Lemma 20 *If an f-matching M_G of a graph G has an augmenting path of length x, $x \in \mathbb{N}$, then any corresponding matching M_H of the replicated graph H has an augmenting path of length $3x$.*

Proof Let p_G be the augmenting path of M_G. We construct the augmenting path p_H of M_H as follows. The augmenting path p_H consists of x segments of length 3, one per each edge of p_G. Let s_i denote the i-th segment of p_H and e_i denote the i-the edge of p_G. The second edge of segment s_i is the physical edge $(p_{v,u}, p_{u,v})$ where $e_i = (v, u)$. We continue by specifying the first and last edge of segments s_i with even index (i.e., $e_i \in M_G$) as well as the start-node and end-node of p_H. The first and last edges of all segments s_i with odd index (i.e., $e \notin M_G$) are implied as the $(i + 1)$-th segment of p_H starts with the node that the i-th segment ends with.

Let i be even and $e_i = (v, u)$. The first edge of s_i is the edge of M_H that connects a virtual copy of v to $p_{v,u}$. Similarly, the last edge of s_i is the edge of M_H that connects $p_{u,v}$ to a virtual copy of u. It remains to specify the start-node and end-node of p_H.

Let v be the start-node and u the end-node of p_G. Since M_H is corresponding to M_G, there must be at least one unsaturated virtual copy of each v and u. They are chosen as start-node and end-node of p_H. Note that if $v = u$, then at least two virtual copies of v are unsaturated and p_H starts and ends at distinct virtual copies of v. □

Corollary 21 *Let M_G be an f-matching. If the corresponding 1-matching M_H in the replicated graph does not have augmenting paths of length less than $3(2x - 1)$, $x \in \mathbb{N}$, then the approximation ratio of M_G is at least $(1 - \frac{1}{x})$.*

Proof Assume that M_H has no augmenting path of length less than $3(2x - 1)$. Also assume that M_G has an augmenting path of length less than $2x - 1$. Then M_H has an augmenting path of length less than $3(2x - 1)$ by Lemma 20 which is a contradiction to the first assumption. Thus the shortest augmenting path of M_G has length at least $2x - 1$ and thus M_G is a $(1 - \frac{1}{x})$-approximation by Corollary 19. □

In addition, computations on the replicated graph can be efficiently emulated on the original graph as the following result shows.

Lemma 22 *Let $G = (V, E)$ be a graph, and let $f : V \to \mathbb{N}_0$ be such that $f(v) \leq |V|$ for every $v \in V$. Let A be an algorithm in the* CONGEST *model. An execution of x rounds of A on the f-replicated graph of G can be emulated by an algorithm running on G in $O(x)$ rounds in the* CONGEST *model.*

Proof Let H be the f-replicated graph of G and $n = |V|$. Then H has $n' = 2m + \sum_{v \in V} f(v) \leq 3n^2$ nodes, namely $f(v)$ virtual copies per node $v \in V$ and 2 port nodes per edge of G.

The emulation of a computation of A on H sends all messages exchanged over a physical edge $(p_{v,u}, p_{u,v})$ over the corresponding edge (v, u) of G. All other communication of A is over an edge $(v_i, p_{v,u})$ which is emulated internally by node v. This is depicted in Fig. 6.

Since A is designed for the CONGEST model, each node of H transmits at most $O(\log n')$ bits over a physical edge of H per round. Thus the emulation transmits at most $O(\log n') = O(\log n)$ bits per emulated round and edge of G. Hence, it takes $O(1)$ rounds to emulate one round of A running on H. □

The above results on f-matchings hold for general graphs. However, we now shift the focus back to bipartite graphs $G = (C, S, E)$, where C is the set of clients and S the set of servers. Let H be an f-replicated graph of G for some mapping f. Recall the definition of H. We show that H is also bipartite by giving a partition of V_H into two independent sets L_H and R_H. Part nodes of clients and virtual copies of server are added to L_H. Similarly, port nodes of servers and virtual copies of clients are added to R_H. We show that L_H is an independent set. The proof for R_H is analogous. Consider two nodes u and v of L_H. Consider the following three cases:

1. u and v are virtual copies: The construction of H does not include edges between virtual copies. Thus u and v are independent.
2. u is v are port nodes: Then u and v are port nodes of clients c and c' respectively. H only includes an edge between u and v if c and c' are connected in G. Thus v and u are independent in H as c and c' are independent in G.
3. u is a port node and v is a virtual copy. As u is a port node of a client c, it is only connected to virtual copies of c. However, v is a virtual copy of a server by definition of L_H. Thus u and v are independent.

The authors of [24] provide an algorithm for bipartite graphs that computes 1-matchings such that the shortest augmenting path has a length of at least $2x - 1$, where $x \in \mathbb{N}$ is an input parameter of the algorithm. Note that the matching algorithm for general graphs does not have this property. Starting at a matching $M_0 = \emptyset$, the algorithm for bipartite graphs operates in phases. In phase i, the algorithm finds a maximal independent set P_i of augmenting paths of M_{i-1}, where all augmenting paths of P_i are of length $2i - 1$. The matching M_i is then obtained by augmenting M_{i-1} with P_i. In [24] it is shown by induction on i that M_i does not have augmenting paths shorter than length $2i - 1$. Hence, M_x is the desired output of the algorithm. The construction of a maximal independent set of augmenting paths in each phase is randomized. However, with high probability, the independent sets found in all phases are maximal. As shown in [24], the algorithm completes x phases within $O(x^3 \log n)$ rounds in the CONGEST model.

We modify the algorithm to find normalized 1-matchings of the replicated graph. The modified algorithm skips the first phase and starts with the 1-matching M_1 which consists of all physical edges of the replicated graph. Note that M_1 does not have augmenting paths of length 1. Thus, the proofs of [24] apply and by induction on i, M_i does not have augmenting paths shorter than length $2i - 1$. Furthermore, we show that M_i is normalized if M_{i-1} is normalized. Thus, by induction on i, the algorithm computes a normalized matching.

Lemma 23 *If the 1-matching M' is the result of augmenting a normalized 1-matching M of a replicated graph, then M' is also normalized.*

Proof Since M' is the result of augmenting M, it holds that $\deg_{M'}(v) \geq \deg_M(v)$ for all nodes v. In particular, this holds for all port nodes. Thus, since M saturates all port nodes, so does M'. □

We summarize our results on the modified bipartite 1-matching algorithm as follows:

Lemma 24 *A normalized 1-matching M of a bipartite replicated graph such that its shortest augmenting path has a length of at least $2x - 1$, $x \in \mathbb{N}$, can be computed w.h.p. within $O(x^3 \log n)$ rounds with messages of $O(\log n)$ bits, where n denotes the number of nodes in the replicated graph.*

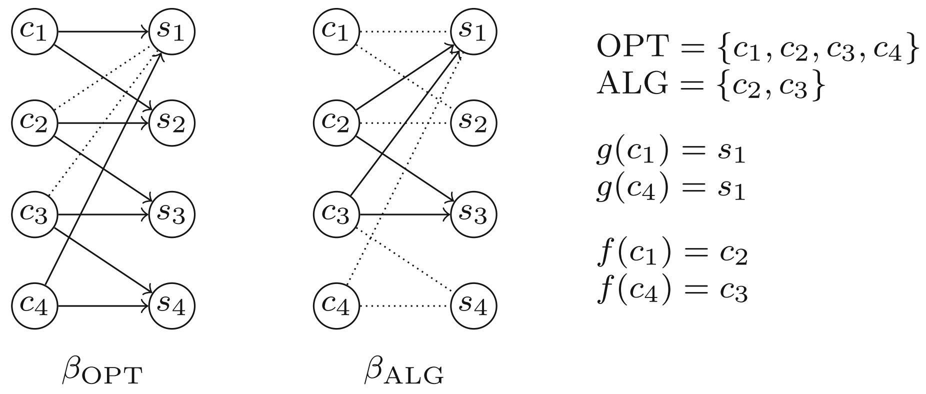
We conclude our results on bipartite f-matching with the following result:

Theorem 25 *For any $x > 1$, a $(1 - \frac{1}{x})$-approximate f-matching of a bipartite graph G can be computed w.h.p. in time $O(x^3 \log n)$ in the* CONGEST *model, where n is the number of nodes of G.*

Proof By Lemma 24, a normalized 1-matching without any augmenting paths shorter than length $6x - 3$ can be computed in time $O(x^3 \log n')$, where n' is the number of nodes of the replicated graph. Since $n' \leq 3n^2$, we have that $O(x^3 \log n') = O(x^3 \log n)$. By Corollary 21, the corresponding f-matching is a $(1 - \frac{1}{x})$-approximation. □

Combining the above result with the reductions of Lemmas 12 and 13, we obtain

Fig. 7 Example of the construction of the mapping f used in the proof of Theorem 28



Corollary 26 *Given a* MINLOAD *instance with uniform sizes and replication factor k and some $x > 1$, an $O\left(\frac{\log n}{\log x}\right)$-approximate solution can be found w.h.p. in the* CONGEST *model within $O\left(\frac{x^3 \log^3 n}{\log x}\right)$ rounds.*

If we set $x = 2$, our algorithm computes an $O(\log n)$-approximate solution of MINLOAD in $O(\log^3 n)$ rounds. We optimize slightly by choosing $x = \frac{\log n}{\log \log n}$ to obtain an $O\left(\frac{\log n}{\log \log n}\right)$-approximate solution of MINLOAD in $O\left(\frac{\log^6 n}{(\log \log n)^4}\right)$ rounds. Another possible choice for our algorithm is $x = n^{1/c}$, for any positive constant c. It then computes an $O(1)$-approximation in $O(n^{3/c} \log^2 n)$ rounds.

By combining Corollary 26 with Lemma 11, we obtain:

Corollary 27 *Given a* MINLOAD *instance with non-uniform sizes and replication factor k and some $x > 1$, an $O\left(\frac{\log^2 n}{\log x}\right)$-approximation can be found w.h.p. in the* CONGEST *model within $O\left(\frac{x^3 \log^4 n}{\log x}\right)$ rounds.*

6 Maximal versus maximum coverage

An alternative approach to approximate MINLOAD is via reduction to maximal MAXCOV. If the replication factor is k and file sizes are uniform, then this approach leads to an $O(k \log n)$-approximation of MINLOAD. A central ingredient in the proof is the fact that a maximal solution to MAXCOV is a $\frac{1}{k+1}$-approximation to the optimal solution. In the remainder of this section we prove this algorithm-independent bound and show that it is tight.

Let us first define maximal and maximum solutions to MAXCOV. Given an instance I of MAXCOV, an assignment β which satisfies a client set C is called *maximum* if for every other solution β' for I, the set of satisfied clients C' is such that $|C'| \leq |C|$. The solution β is called *maximal* if there is no solution β' for I that strictly extends β, i.e., there is no β' such that $\beta(c) \subseteq \beta'(c)$ for all clients c and $\beta(c) \neq \beta'(c)$ for some client c.

Note that all maximum solutions satisfy the same number of clients, which is the optimum. Regarding maximal solutions, we have the following results. First, we show that the number of satisfied clients in a maximal solution to MAXCOV is at least a $(\frac{1}{k+1})$-fraction of the number of satisfied clients in any maximum solution.

Theorem 28 *Let $I = (G, k, L)$ be a* MAXCOV *instance with uniform file sizes. Let* OPT *be the set of clients satisfied by some optimal solution and* ALG *be the set of clients satisfied by some maximal solution to I. Then $|\mathrm{ALG}| \geq \frac{|\mathrm{OPT}|}{k+1}$.*

Proof We shall define a mapping $f : \mathrm{OPT} \to \mathrm{ALG}$, and the claim will follow from showing that $|f^{-1}(c)| \leq k+1$ for any $c \in \mathrm{ALG}$. We first define an auxiliary function $g : (\mathrm{OPT} \setminus \mathrm{ALG}) \to S_{\mathrm{ALG}}$, where S_{ALG} is the set of servers with load L in ALG. For a client c, we define $g(c)$ to be an arbitrary server which is used for c under OPT and is fully loaded under ALG. Such a server always exists because otherwise c could be added to ALG, contradicting its maximality. Note that $|g^{-1}(s)| \leq |\beta^{-1}_{\mathrm{OPT}}(s)| \leq L$ for all $s \in S_{\mathrm{ALG}}$, because under OPT, at most L clients are mapped to s.

We now define f. Let $c \in \mathrm{OPT}$. If $c \in \mathrm{ALG}$, define $f(c) := c$. Suppose now that $c \in \mathrm{OPT} \setminus \mathrm{ALG}$. For this part, we construct f using the following procedure. For any node c', we record whether c' was assigned to c *via* $g(c)$. Initially no node is assigned this way. To define $f(c)$, we choose any node $c' \in \beta^{-1}_{\mathrm{ALG}}(g(c))$ that was not assigned yet by f via $g(c)$. We set $f(c) := c'$ and we say that c' is assigned to c via $g(c)$. This is possible, because $|\beta^{-1}_{\mathrm{ALG}}(g(c))| = L$, and the number of times we assign f via $g(c)$ is at most $|g^{-1}(g(c))| \leq L$. This concludes the construction of f. An example is depicted in Fig. 7.

To complete the proof, observe that for all $c \in \mathrm{ALG}$, $|f^{-1}(c)| \leq k+1$: it could be that $c \in f^{-1}(c)$, and on top of that, there are k servers s such that $c \in \beta^{-1}_{\mathrm{ALG}}(s)$, and by construction, f assigns a client to c at most once via each of these servers. □

The $\frac{1}{k+1}$ ratio from Theorem 28 cannot be improved in general, as made precise in the following result.

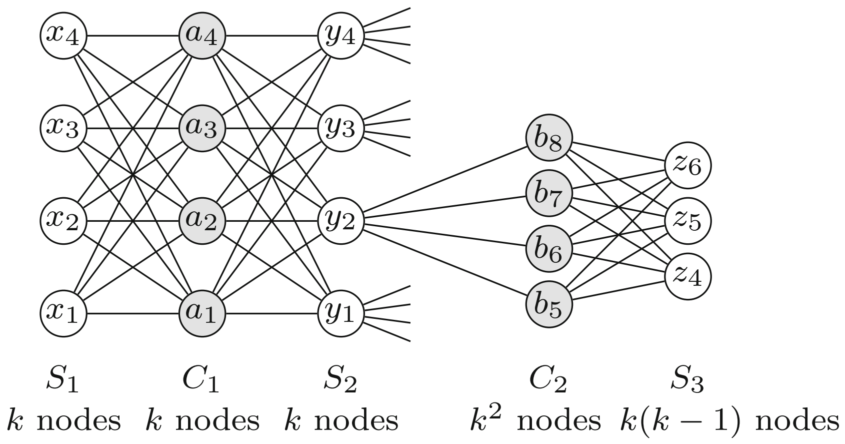


Fig. 8 The graph used in the proof of Theorem 29 for $k = 4$. Each node of S_2 is attached to a distinct complete bipartite graph (only one shown) with k clients of C_2 and $k-1$ servers of S_3

Theorem 29 *For any replication factor k and uniform file sizes, there exists a* MaxCov *instance I_k and a maximal solution* ALG_k *such that* $|\text{ALG}_k| = \frac{|\text{OPT}_k|}{k+1}$, *where OPT_k is the set of clients satisfied by an optimal solutions of I_k.*

Proof We construct I_k as follows (see Fig. 8). There are two sets of clients: $C_1 = \{a_i\}_{i=1}^{k}$ and $C_2 = \{b_i\}_{i=1}^{k^2}$. There are three sets of servers: $S_1 = \{x_i\}_{i=1}^{k}$, $S_2 = \{y_i\}_{i=1}^{k}$, and $S_3 = \{z_i\}_{i=1}^{k(k-1)}$. The edges are defined as follows.

- Each $a_i \in C_1$ is connected to every server from $S_1 \cup S_2$,
- Each $b_i \in C_2$ is connected to $y_{\lceil i/k \rceil} \in S_2$, and
- Each $b_i \in C_2$ is also connected to every $z_j \in S_3$ with $\lceil j/(k-1) \rceil = \lceil i/k \rceil$.

Let all servers have capacity k, i.e., $L := k$. This concludes the description of the instance I_k. (This graph of $2(k^2+k)$ nodes can be replicated any desired number of times.)

Now, in an optimal solution to I_k, each client in C_1 is assigned all servers in S_1, and each client in C_2 is assigned $k-1$ servers from S_3 and a single server from S_2. Thus, all k^2+k clients can be satisfied. On the other hand, consider the solution in which each client in C_1 is assigned all servers in S_2. This is a maximal solution since clearly no other client (namely, from C_2) can be added to this assignment, since all servers in S_2 are fully loaded. Hence, there is a maximal solution with k satisfied clients. The claim follows. □

We proceed to describe a reduction of α-solving MaxCov to computing maximal solution to MaxCov for the uniform case.

Lemma 30 *Given an algorithm that computes maximal solutions of* MaxCov *with uniform file sizes and replication factor k in time T,* MaxCov *can be α-solved in time αT with $\alpha = \lceil (k+1)\ln n \rceil$.*

Proof Let A be an algorithm that computes maximal solutions to MaxCov for uniform file sizes and replication factor k. Let L denote the bound on the server load. The following algorithm proves the claim:

– For $i = 0$ to α do
  1. Run algorithm A. Obtain solution β_i which satisfies at least $\frac{1}{k+1}\text{MaxCov}(L)$ clients.
  2. Add β_i to the output, and remove clients satisfied by β_i from further consideration.

We proceed to show that after α iterations, all clients are satisfied. In each iteration, at most a $(1-\frac{1}{k+1})$-fraction of the unsatisfied clients remains unsatisfied by Theorem 28. Hence, after α iterations, the number of unsatisfied clients is

$$\begin{aligned} n \cdot \left(1 - \frac{1}{k+1}\right)^{\alpha} &= n \cdot \left(\left(1 - \frac{1}{k+1}\right)^{k+1}\right)^{\frac{\alpha}{k+1}} \\ &< n \cdot (1/e)^{\frac{\alpha}{k+1}} \\ &\le 1 \,. \end{aligned}$$

This concludes the proof. □

Corollary 31 *An $O(k \log n)$-approximate solution to* MinLoad *with uniform file sizes and replication factor k can be computed in time $O(T \cdot k \log^2 n)$ where T is the time to compute maximal solutions to* MaxCov.

Proof Follows directly from Lemmas 30 and 12. □

We emphasize that this result is inferior to Corollary 26 regarding the approximation ratio.

7 Conclusion

This paper considered the backup placement problem which we view as a central problem in network algorithms. We showed that simple random placements perform very badly in terms of placement quality, and that selfish local improvement may run a very long time until stabilization is reached. Our algorithm uses at its core a distributed matching procedure, and thus it can guarantee, with high probability, both polylog running time and a good approximation ratio. Several open problems remain, including

- The sequential complexity of MaxCov with replication factor $k = 2$ remains open: it is solvable in polynomial time with $k = 1$ while it is NP-hard with $k \geq 3$.
- Can MinLoad be approximated distributively to within $(1 + \epsilon)$ in polylogarithmic time?
- Is there an improvement over the classify-and-select technique used in the non-uniform case?

A collateral result of our paper is a distributed approximation algorithm for f-matching in bipartite graphs. It remains open whether it can be extended to general graphs.

References

1. Amini, O., Peleg, D., Pérennes, S., Sau, I., Saurabh, S.: On the approximability of some degree-constrained subgraph problems. Discret. Appl. Math. **160**(12), 1661–1679 (2012). doi:10.1016/j.dam.2012.03.025
2. Awerbuch, B.: Complexity of network synchronization. J. ACM **32**(4), 804–823 (1985). doi:10.1145/4221.4227
3. Azar, Y.: On-line load balancing. In: Fiat, A., Woeginger, G.J. (eds.) Online Algorithms. Lecture Notes in Computer Science, vol. 1442, pp. 178–195. Springer, Berlin (1998). doi:10.1007/BFb0029569
4. Azar, Y., Naor, J., Rom, R.: The competitiveness of on-line assignments. J. Algorithms **18**(2), 221–237 (1995). doi:10.1006/jagm.1995.1008
5. Barenboim, L., Elkin, M., Pettie, S., Schneider, J.: The locality of distributed symmetry breaking. J. ACM **63**(3), 20:1–20:45 (2016). doi:10.1145/2903137
6. Bokal, D., Brešar, B., Jerebic, J.: A generalization of hungarian method and Hall's theorem with applications in wireless sensor networks. Discret. Appl. Math. **160**(4–5), 460–470 (2012). doi:10.1016/j.dam.2011.11.007
7. Czygrinow, A., Hanćkowiak, M., Szymańska, E., Wawrzyniak, W.: Distributed 2-approximation algorithm for the semi-matching problem. In: 26th DISC, pp. 210–222 (2012). doi:10.1007/978-3-642-33651-5_15
8. Dessmark, A., Garrido, O., Lingas, A.: A note on parallel complexity of maximum f-matching. Inform. Process. Lett. **65**(2), 107–109 (1998). doi:10.1016/S0020-0190(97)00196-8
9. Dósa, G., Epstein, L.: The convergence time for selfish bin packing. In: 7th SAGT, pp. 37–48 (2014). doi:10.1007/978-3-662-44803-8_4
10. Even-Dar, E., Kesselman, A., Mansour, Y.: Convergence time to Nash equilibrium in load balancing. ACM T. Algorithms **3**(3), 32 (2007). doi:10.1145/1273340.1273348
11. Fakcharoenphol, J., Laekhanukit, B., Nanongkai, D.: Faster algorithms for semi-matching problems. ACM Trans. Algorithms **10**(3), 14:1–14:3 (2014). doi:10.1145/2601071
12. Gairing, M., Lücking, T., Mavronicolas, M., Monien, B.: The price of anarchy for restricted parallel links. Parallel Process. Lett. **16**(1), 117–131 (2006). doi:10.1142/S0129626406002514
13. Galčík, F., Katrenič, J., Semanišin, G.: On computing an optimal semi-matching. In: 37th WG, pp. 250–261 (2011). doi:10.1007/978-3-642-25870-1_23
14. Garg, N., Kavitha, T., Kumar, A., Mehlhorn, K., Mestre, J.: Assigning papers to referees. Algorithmica **58**(1), 119–136 (2010). doi:10.1007/s00453-009-9386-0
15. Halldórsson, M.M., Köhler, S., Rawitz, D.: Distributed approximation of k-service assignment. In: 19th International Conference on Principles of Distributed Systems, pp. 122–137 (2015)
16. Hańćkowiak, M., Karoński, M., Panconesi, A.: A faster distributed algorithm for computing maximal matchings deterministically. In: 18th PODC, pp. 219–228 (1999). doi:10.1145/301308.301360
17. Harvey, N.J.A., Ladner, R.E., Lovász, L., Tamir, T.: Semi-matchings for bipartite graphs and load balancing. J. Algorithms **59**(1), 53–78 (2006). doi:10.1016/j.jalgor.2005.01.003
18. Hazan, E., Safra, S., Schwartz, O.: On the complexity of approximating k-set packing. Comput. Complex. **15**(1), 20–39 (2006). doi:10.1007/s00037-006-0205-6
19. Hopcroft, J.E., Karp, R.M.: An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. SIAM J. Comput. **2**(4), 225–231 (1973). doi:10.1137/0202019
20. Kann, V.: Maximum bounded 3-dimensional matching is MAX SNP-complete. Inform. Process. Lett. **37**(1), 27–35 (1991). doi:10.1016/0020-0190(91)90246-E
21. Katrenič, J., Semanišin, G.: Maximum semi-matching problem in bipartite graphs. Discuss. Math. Graph Theory **33**(3), 559–569 (2013). doi:10.7151/dmgt.1694
22. Köhler, S., Turau, V., Mentges, G.: Self-stabilizing local k-placement of replicas with minimal variance. In: 14th SSS, pp. 16–30 (2012). doi:10.1007/978-3-642-33536-5_2
23. Koufogiannakis, C., Young, N.E.: Distributed algorithms for covering, packing and maximum weighted matching. Distrib. Comput. **24**(1), 45–63 (2011). doi:10.1007/s00446-011-0127-7
24. Lotker, Z., Patt-Shamir, B., Pettie, S.: Improved distributed approximate matching. J. ACM **62**(5), 38:1–38:17 (2015). doi:10.1145/2786753
25. Low, C.P.: An approximation algorithm for the load-balanced semi-matching problem in weighted bipartite graphs. Inform. Process. Lett. **100**(4), 154–161 (2006). doi:10.1016/j.ipl.2006.06.004
26. Panconesi, A., Rizzi, R.: Some simple distributed algorithms for sparse networks. Distrib. Comput. **14**(2), 97–100 (2001). doi:10.1007/PL00008932
27. Patt-Shamir, B., Rawitz, D., Scalosub, G.: Distributed approximation of cellular coverage. J. Parallel Distrib. Comput. **72**(3), 402–408 (2012). doi:10.1016/j.jpdc.2011.12.003
28. Peleg, D.: Distributed Computing: A Locality-Sensitive Approach. Society for Industrial and Applied Mathematics, Philadelphia (2000)
29. Shiloach, Y.: Another look at the degree constrained subgraph problem. Inform. Process. Lett. **12**(2), 89–92 (1981). doi:10.1016/0020-0190(81)90009-0
30. Vöcking, B.: Selfish load balancing. In: Nisan, N., Roughgarden, T., Tardos, E., Vazirani, V. (eds.) Algorithmic Game Theory, pp. 699–716. Cambridge University Press, Cambridge (2007)