Concurrent Distributed Serving with Mobile Servers

Abdolhamid Ghodselahi
Institute of Telematics, Hamburg University of Technology, Germany
abdolhamid.ghodselahi@tuhh.de

Fabian Kuhn
Department of Computer Science, University of Freiburg, Germany
kuhn@cs.uni-freiburg.de

Volker Turau
Institute of Telematics, Hamburg University of Technology, Germany
turau@tuhh.de

Abstract

This paper introduces a new resource allocation problem in distributed computing called *distributed serving with mobile servers (DSMS)*. In DSMS, there are k identical mobile servers residing at the processors of a network. At arbitrary points of time, any subset of processors can invoke one or more requests. To serve a request, one of the servers must move to the processor that invoked the request. Resource allocation is performed in a distributed manner since only the processor that invoked the request initially knows about it. All processors cooperate by passing messages to achieve correct resource allocation. They do this with the goal to minimize the communication cost.

Routing servers in large-scale distributed systems requires a scalable location service. We introduce the distributed protocol Gnn that solves the DSMS problem on *overlay trees*. We prove that Gnn is starvation-free and correctly integrates locating the servers and synchronizing the concurrent access to servers despite asynchrony, even when the requests are invoked over time. Further, we analyze Gnn for "one-shot" executions, i.e., all requests are invoked simultaneously. We prove that when running Gnn on top of a special family of tree topologies – known as *hierarchically well-separated trees (HSTs)* – we obtain a randomized distributed protocol with an expected competitive ratio of $\mathrm{O}\big(\log n\big)$ on general network topologies with n processors. From a technical point of view, our main result is that Gnn optimally solves the DSMS problem on HSTs for one-shot executions, even if communication is asynchronous. Further, we present a lower bound of $\Omega(\max\{k, \log n/\log\log n\})$ on the competitive ratio for DSMS. The lower bound even holds when communication is synchronous and requests are invoked sequentially.

2012 ACM Subject Classification Theory of computation → Online algorithms; Theory of computation → Distributed algorithms; Theory of computation → Graph algorithms analysis; Theory of computation → Discrete optimization

Keywords and phrases Distributed online resource allocation, Distributed directory, Asynchronous communication, Amortized analysis, Tree embeddings

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.53

Related Version A full version of this paper is available at `https://arxiv.org/abs/1902.07354` [14].

Funding This work is supported by the Deutsche Forschungsgemeinschaft (DFG), under grant DFG TU 221/6-3.

1 Introduction

Consider the following family of online resource allocation problems. We are given a metric space with n points. Initially, a set of $k \geq 1$[1] identical mobile servers are residing at different points of the metric space. Requests arrive over time in an online fashion, that is, one or several requests can arrive at any point of time. A request needs to be served by a server at the requesting point sometime after its arrival. The goal is to provide a schedule for serving all requests. This abstract problem lies at the heart of many centralized and distributed online applications in industrial planning, operating systems, content distribution in networks, and scheduling [3, 7, 8, 16, 21]. Each concrete problem of this family is characterized by a cost function. We study this abstract problem in distributed computing and call it the distributed serving with mobile servers (DSMS) problem. A distributed protocol Alg that solves the DSMS problem must compute a schedule for each server consisting of a queue of requests such that consecutive requests are successively served, and all requests are served. The k schedules are distributedly stored at the requesting nodes: each node knows for each of its requests the node which invoked the subsequent request in the schedule so that a server after serving one request can subsequently move to the next node (not necessarily a different node). As long as new requests are invoked the schedule is extended. Therefore, in response to the appearance of a new request at a given processor, Alg must contact a processor that invoked a request but yet has no successor request in the global schedule, to instruct the motion of the corresponding server. This will result in the entry of a server to the requesting processor. Sending a server from a processor to another one is done using an underlying routing scheme that routes most efficiently. The goal is to minimize the ratio between the communication costs of an online and an *optimal offline* protocols that solve DSMS. We assume that an optimal offline DSMS protocol Opt knows the whole sequence of requests in advance. However, Opt still needs to send messages from each request to its predecessor request. The DSMS problem has some interesting applications. We state two of them:

Distributed k-server problem

The k-server problem [5, 18], is arguably one of the most influential research problems in the area of online algorithms and competitive analysis. The distributed k-server was studied in [8] where requests arrive sequentially one by one, but only after the current request is served. The cost function for this problem is defined as the sum of all communication costs and the total movement costs of all servers. A generalization of the k-server problem where requests can arrive over time is called the online service with delay (OSD) problem [4, 9]. The OSD cost function is defined as the sum of the total movement costs of all servers and the total *delay cost*. The delay of a request is the difference between the service and the arrival times.

Distributed queuing problem

This problem is an application of DSMS with $k = 1$, i.e., only one server or shared object [10, 15, 16]. The distributed queuing problem is at the core of many distributed problems that schedule concurrent access requests to a shared object. The goal is to minimize the sum of the total communication cost and the total "waiting time". The waiting time of a request is the difference between the times when the request message reaches the processor of the predecessor request and when the predecessor request is invoked. Note that in this problem,

[1] Table 1 provides an index for the essential notations used throughout the paper.

the processor of a request must only send one message to the processor of the predecessor request in the global schedule. Two well-known applications for this problem are distributed mutual exclusion [19, 21, 22] and distributed transactional memory [24].

Next, we explain why DSMS is also interesting from a theoretical point of view even for one-shot executions, that is, when all requests are simultaneously invoked. Figure 1 shows a rooted tree T, where the lengths of all edges of each level are equal. Further, the length of every edge is shorter than the length of its parent edge by some factor larger than one. A set of six requests arrive at the leaves of T at the same time. Two servers s_0, s_1 are initially located at the points that invoked requests r_0^1 and r_0^2. Serving the requests r_0^1 and r_0^2 does not require communication, and these two requests are the current tails of the queues of s_0 and s_1. The requests r_0^1 and r_0^2 are at the heads of the two queues. An optimal solution for serving the remaining requests is that s_0 consecutively serves the requests r_b, r_c, and r_a after serving r_0^1, while s_1 serves r_d after having served r_0^2. Next, consider an asynchronous network where, in contrast with a synchronous network where there is a global clock, message latencies are arbitrary and protocols have no control over these latencies. A possible schedule, in this case, is shown in Figure 1: Request r_a is scheduled after r_0^1, r_b after r_a, and r_d after r_b, since the message latency of a request further away can be much less than the latency of a closer request. This can lead to complications with regard to improving the *locality* as it is met in the above optimal solution.

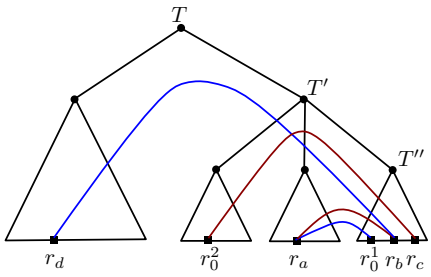


Figure 1 A distributed protocol may lead to complications with regard to improving locality.

GNN protocol

We devise the generalized nearest-neighbor (Gnn) protocol that greedily solves the DSMS problem on **overlay trees**. An overlay tree T is a rooted tree that is constructed on top of the underlying network. The processors of the original network are in a one-to-one correspondence with the leaves of T. Hence, only T's leaves can invoke requests, and the remaining overlay nodes are artificial. The k servers reside at different leaves of T. Initially, all edges of T are oriented such that from each leaf there is a directed path to a leaf, where a server resides. This also implies that every leaf node with a server has a self-loop. Roughly speaking, the main idea of Gnn is to update the directions of edges with respect to future addresses of a server. A leaf invoking a request forwards a message along the directed links, the orientations of all these links are inverted. When a message reaches a node and finds several outgoing (upward/downward) links, it is forwarded via an arbitrary downward link to find the current or a future address of a server. We show that in Gnn a processor holding a request always sends a message through a direct path to the processor of the predecessor request in the global schedule. We refer to Section 3 for a formal description of Gnn.

1.1 Our Contribution

This paper introduces the DSMS problem as a distributed online allocation problem. We devise the greedy protocol Gnn that solves the DSMS problem on overlay trees. We prove that even in an asynchronous system Gnn operates correctly, that is, it does not suffer from starvation, nor livelocks, or deadlocks. To the best of our knowledge, Gnn is the first link-reversal-based protocol that supports navigating more than one server.

▶ **Theorem 1.** *Suppose the overlay tree T is constructed on top of a distributed network. Consider the DSMS problem on T where a set of $k \geq 1$ identical mobile servers are initially located at different leaves of T. Further, a sequence of requests can be invoked at any time by the leaves of T. Then* Gnn *schedules all requests to be served by some server at the requested points in a finite time despite asynchrony.*

While Gnn itself solves any instance of the DSMS problem, we analyze Gnn for the particular case that the requests are simultaneously invoked. We consider general distributed networks with n processors. We model such a network by a graph G. A hierarchically well-separated tree (HST) is an overlay tree with parameter $\alpha > 1$, that is, an α-HST is a rooted tree where every edge weight is shorter by a factor of α from its parent edge weight. A tree is an HST if it is an α-HST for some $\alpha > 1$. There is a randomized embedding of any graph into a distribution over HSTs [6, 11]. We sample an HST T according to the distribution defined by the embedding. We consider an instance I of the DSMS problem where the communication is asynchronous, and the requests are simultaneously invoked by the nodes of G. When running Gnn on T, we get a randomized distributed protocol on G that solves I with an expected competitive ratio of $\mathrm{O}\big(\log n\big)$ against oblivious adversaries[2].

▶ **Theorem 2.** *Let I denote an instance of the DSMS problem consisting of an asynchronous network with n processors and a set of requests that are simultaneously invoked by processors of the network. There is a randomized distributed protocol that solves I with an expected competitive ratio of $O(\log n)$ against an oblivious adversary.*

Consider an instance I of DSMS that consists of an HST T where communication is asynchronous and a set of requests that are simultaneously invoked by the leaves of T. Analyzing Gnn for I turns out to be involved and non-trivial. The fact that the Gnn (as any other protocol) has no control on the message latencies bears a superficial resemblance to the case where the requests are invoked over time. Hence, when analyzing Gnn for I, one faces the following complications: 1) A server may go back to a subtree of T after having left it. 2) A request in a subtree of T that initially hosts at least one server can be served by a server that is initially outside this subtree. 3) Different servers can serve two requests in a subtree of T that does not initially host any server. Theorem 2 is derived from our main technical result for HSTs.

▶ **Theorem 3.** *Consider an instance I of DSMS that consists of an HST T where even the communication is asynchronous and a set of requests that are simultaneously invoked by the leaves of T. The* Gnn *protocol optimally solves I.*

One-shot executions of the distributed queuing problem for synchronous communication were already considered in [16]. The following corollary follows from Theorem 3.

▶ **Corollary 4.** Gnn *optimally solves the distributed queuing problem on HSTs for one-shot executions even when the communication is asynchronous.*

[2] This assumes that the sequence of requests is statistically independent of the randomness used for constructing the given tree.

We provide a simple reduction form the distributed k-server problem to the DSMS problem. Our following lower bound is obtained using this reduction and an existing lower bound [8] on the competitive ratio for the distributed k-server problem.

▶ **Theorem 5.** *There is a network topology with n processors – for all n – such that there is no online distributed protocol solving DSMS with a competitive ratio of $o(\max\{k, \log n/\log\log n\})$ against adaptive online adversaries where k is the number of servers. This result even holds when requests are invoked one by one by processors in a sequential manner and even when the communication is synchronous.*

2 Model, Problem Statement, and Preliminaries

2.1 Communication Model

We consider a point-to-point communication network that is modeled by a graph $G = (V, E)$, where the n nodes in V represent the processors of the network and the edges in E represent bidirectional communication links between the corresponding processors. We suppose that the edge weights are positive and are normalized such that the weight of each edge will be at least 1. If G is unweighted, then we assume that the weight of an edge is 1. We consider the message passing model [20] where neighboring processors can exchange messages with each other. The communication links can have different latencies. These latencies are not even under control of an optimal offline distributed protocol. We consider both synchronous and asynchronous systems. In a synchronous system, the latency for sending a message over an edge equals the weight of the edge. In an asynchronous system, in contrast, the messages arrive at their destinations after a finite but unbounded amount of time. Messages that take a longer path may arrive earlier, and the receiver of a message can never distinguish whether a message is still in transit or whether it has been sent at all. For our analysis, however, we adhere to the conventional approach where the latencies are scaled such that the latency for sending a message over an edge is upper bounded by the edge weight in the "worst case" (for every legal input and in every execution scenario) (see Section 2.2 in [20] for more information).

2.2 Distributed Serving with Mobile Servers (DSMS) Problem

The input for DSMS problem for a graph G consists of $k \geq 1$ identical mobile servers that are initially located at different nodes of G and a set $\mathcal{R}$ of requests that are invoked at the nodes at any time. A request $r_i \in \mathcal{R}$ is represented by (v_i, t_i) where node v_i invoked request r_i at time $t_i \geq 0$. A distributed protocol ALG that solves the DSMS problem needs to serve each request with one of the k servers at the requested node. Hence, ALG must schedule all requests that access a particular server. Consequently, ALG outputs k global schedules. Let π^z_{ALG} where $z \in \{1, \ldots, k\}$ denote the z-th schedule generated by ALG, and s^z be the z-th server. The request sets of these k schedules form a partition of $\mathcal{R}$, and all requests of the schedule π^z_{ALG} consecutively access the server s^z. We assume that at time 0, when an execution starts, the tail of schedule π^z_{ALG} is at a given node $v^z_0 \in V$ that hosts s^z. Formally, this is modeled as a "dummy request" $r^z_0 = (v^z_0, 0)$ that has to be scheduled first in the schedule π^z_{ALG} by ALG. Consider two requests r_i and r_j that are consecutively served by s^z where r_i is scheduled after r_j. To schedule request r_i the protocol needs to inform node v_j, the predecessor request r_j in the constructed schedule. As soon as r_j is served by s^z, node v_j sends the server to v_i for serving r_i using an underlying routing facility that efficiently routes messages. The goal is to minimize the total communication cost, i.e., the sum of the latencies of all messages sent during the execution of ALG.

2.3 Preliminaries

Consider a distributed protocol ALG for the DSMS problem when requests can arrive at any time. Let $\mathcal{R}$ denote the set of requests, including the dummy requests. Assume that ALG partitions $\mathcal{R}$ into k sets $\mathcal{R}^1_{\text{ALG}}, \ldots, \mathcal{R}^k_{\text{ALG}}$, and that it schedules the requests in set $\mathcal{R}^z_{\text{ALG}}$ according to permutation π^z_{ALG}. Denote the request at position i of π^z_{ALG} by $r_{\pi^z_{\text{ALG}}(i)}$. The dummy request r^z_0 of π^z_{ALG} is represented by $r_{\pi^z_{\text{ALG}}(0)}$. Consider a message denoted by μ. Let $\ell_{\text{ALG}}(\mu)$ denote the latency of message μ as routed by ALG. For every $i \in \{1, \ldots, |\mathcal{R}| - 1\}$, if r_i belongs to $\mathcal{R}^z_{\text{ALG}}$, the communication cost $c_{\text{ALG}}\big(r_{\pi^z_{\text{ALG}}(i-1)}, r_{\pi^z_{\text{ALG}}(i)}\big)$ incurred for scheduling $r_{\pi^z_{\text{ALG}}(i)}$ as the successor of $r_{\pi^z_{\text{ALG}}(i-1)}$ is the sum of the latencies of all messages sent by ALG to schedule $r_{\pi^z_{\text{ALG}}(i)}$ immediately after $r_{\pi^z_{\text{ALG}}(i-1)}$. The total communication cost of ALG for scheduling all requests in $\mathcal{R}^z_{\text{ALG}}$ is defined as

$$C_{\text{ALG}}(\pi^z_{\text{ALG}}) := \sum_{i=1}^{|\mathcal{R}^z_{\text{ALG}}|-1} c_{\text{ALG}}\left(r_{\pi^z_{\text{ALG}}(i-1)}, r_{\pi^z_{\text{ALG}}(i)}\right). \tag{1}$$

The total communication cost of ALG for scheduling all requests in $\mathcal{R}$, therefore, is

$$C_{\text{ALG}} := \sum_{z=1}^{k} C_{\text{ALG}}(\pi^z_{\text{ALG}}). \tag{2}$$

2.4 Hierarchically Well-Separated Trees (HSTs)

Embedding of a metric space into probability distributions over tree metrics have found many important applications in both centralized and distributed settings [4, 5, 13]. The notion of a hierarchically well-separated tree was defined by Bartal in [6].

▶ **Definition 6 (α-HST).** *For $\alpha > 1$ an α-HST of depth h is a rooted tree with the following properties: The children of the root are at a distance α^{h-1} from the root and every subtree of the root is an α-HST of depth $h-1$. A tree is an HST if it is an α-HST for some $\alpha > 1$.*

The definition implies that the nodes two hops away from the root are at a distance α^{h-2} from their parents. The probabilistic tree embedding result of [11] shows that for every metric space (X, d) with minimum distance normalized to 1 and for every constant $\alpha > 1$ there is a randomized construction of an α-HST T with a bijection f between the points in X and the leaves of T such that a) the distances on T are dominating the distances in the metric space (X, d), i.e., $\forall x, y \in X : d_T\big(f(x), f(y)\big) \geq d(x, y)$ and such that b) the expected tree distance is $\mathbb{E}\big[d_T\big(f(x), f(y)\big)\big] = O(\alpha \log |X| / \log \alpha) \cdot d(x, y)$ for every $x, y \in X$. The length of the shortest path between any two leaves u and v of T is denoted by $d_T(u, v)$. An efficient distributed construction of the probabilistic tree embedding of [11] has been given in [12].

3 The Distributed GNN Protocol

In this section the GNN protocol is introduced.

3.1 Description of GNN

GNN runs on overlay trees and outputs a feasible solution for the DSMS problem. Consider a rooted tree $T = (V_T, E_T)$ whose leaves correspond to the nodes of the underlying graph $G = (V, E)$, i.e., $V \subseteq V_T$. Let $n = |V|$. The $k \geq 1$ identical mobile servers are initially at different leaves of T. Further, there is a dummy request at every leaf that initially hosts a

server. The leaves of T can invoke requests at any time. A leaf node can invoke a request while it is hosting a server and a leaf can also invoke a request while its previous requests have not been served yet. Initially, a directed version of T is constructed and denoted by H, the directed edges of H are called *links*. During an execution of GNN, GNN changes the directions of the links. Denote by $v.links$ the set of neighbors of v that are pointed by v. After a leaf u has invoked a request it sends a find-predecessor message denoted by $\mu(u)$ along the links to inform the node of the predecessor request in the global schedule. The routing of $\mu(u)$ is explained below. At the beginning before any message is sent and for any server, all the nodes on the direct path from the root of T to the leaf that hosts the server, point to the server. Further, the host points to itself and creates a self-loop. Hence, we have k directed paths with downward links from the root of T to the points of the current tails of the schedules. Any other node points to its parent with an upward link. Therefore, the sets $v.links$ for all $v \in V_T$ are non-empty at the beginning of the executing the protocol. Figure 2a shows the directed HST at the beginning as an example.

Algorithm 1 GNN Protocol.

Input : The rooted tree T, k identical mobile servers that are initially at distinct leaves of T, and a set of requests that are invoked over time

Output : k schedules for serving all requests

Upon requesting a service: Algorithm 2

Upon receiving a find-predecessor message: Algorithm 3

Upon u invoking a new request

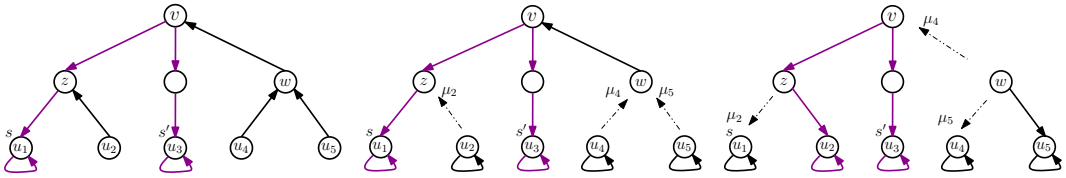
Consider the leaf node u when it invokes a new request r. If u has a self-loop, then r is scheduled immediately after the last request that has been invoked at u. Otherwise, the leaf u atomically sends $\mu(u)$ to its parent through an upward link, u points to itself, and the link from u to its parent is removed. We suppose that messages are reliably delivered. The details of this part of the protocol are given by Algorithm 2. See Figure 2b as an example.

Algorithm 2 Upon u invoking a new request r.

```
1 do atomically
  /* suppose u.links = {v} (u as a leaf always points either to itself or
     to its parent)                                                    */
2     if u = v then
3         r is scheduled immediately after the last request that has been invoked by u
4     else
5         u sends μ(u) to v
6         u.links := {u}
7     end
8 end
```

Upon w receiving $\mu(u)$ from node v

Suppose that node w receives a find-predecessor message $\mu(u)$ from node v. The node w executes the following steps atomically. If w has at least one downward link, then $\mu(u)$ is forwarded to some child of w through a downward link (ties are broken arbitrarily). Then, w removes the downward link and adds a link to v – independently of whether v is the



(a) initial system state. **(b)** step 1. **(c)** step 2.

Figure 2 GNN protocol: (a) The servers s and s' serve requests in schedules π and π', respectively. The dummy requests at u_1 and u_3 are the initial tails of π and π', respectively. (b) Nodes u_2, u_3, u_4, and u_5 respectively issue requests r_2, r_3, r_4, and r_5 at the same time and send the find-predecessor messages μ_2, μ_3, μ_4, and μ_5, respectively, along the arrows. (c) The request r_3 is the current tail of π'. Both μ_4 and μ_5 reach w at the same time. First, the message μ_4 is arbitrarily processed by w and w forwards μ_4 towards v and therefore μ_5 is deflected towards u_4.

parent or a child of w. If w does not have a downward link, it either points to itself, or it has an upward link. In the latter case, $\mu(u)$ is atomically forwarded to the parent of w, the upward link from w to its parent is removed and then w points to v using a downward link. Otherwise, w is a leaf and points to itself. The request r invoked by u is scheduled after the last request that has been invoked by w. Then, w removes the link that points to itself and points to v using an upward link. The details of this part of the protocol are given by Algorithm 3. Also, see Figure 2c and Figure 3.

Algorithm 3 Upon w receiving $\mu(u)$ from node v ($w \neq v$).

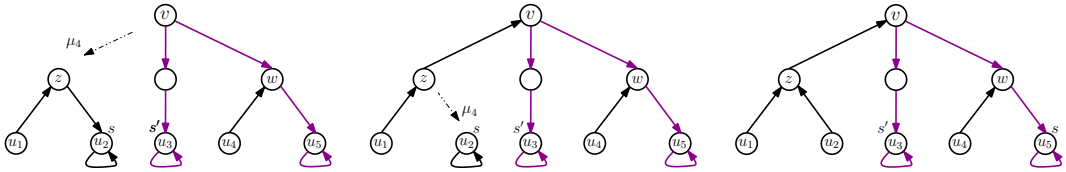
```
1  do atomically
2      if there exists a child node in w.links then
3          z =: an arbitrary child node in w.links
4      else
5          z =: the only node in w.links
6      end
7      w.links := w.links − {z}
8      w.links := w.links ∪ {v}
9      if z ≠ w then
10             w sends μ(u) to z
11     else
12         the corresponding request to μ(u) is scheduled immediately after the last
           request that has been invoked by w
13     end
14 end
```

3.2 Correctness of GNN

Regarding the description of GNN, we need to show two invariants for GNN. The first is that GNN eventually schedules all requests. The second one is that GNN is starvation-free so that a scheduled request is eventually served.

3.2.1 Scheduling Guarantee

▶ **Theorem 7.** GNN *guarantees that the find-predecessor message of any node that invokes a request always reaches a leaf node v in a finite time such that $v.links = \{v\}$.*



(a) step 3. **(b)** step 4. **(c)** step 5.

Figure 3 GNN protocol: (a) The request r_2 is scheduled after the current tail of π and now r_2 is the current tail of π and u_2 obtains the server s. The request r_5 is scheduled after r_4 while μ_4 is still in transit. (b) μ_4 still follows arrows, reversing the directions of arrows along its way. (c) The request r_4 is scheduled after r_2 and s moves to u_4. After r_4 served by s, then s moves from u_4 to u_5 since r_5 has already been scheduled after r_4. Figure 2–Figure 3 illustrates that there is always at least one connected path with purple arrows from the root to some leaf.

We prove the scheduling guarantee stated in Theorem 7 using the following properties of GNN. First, we need to show that any node always has at least one outgoing edge in GNN.

▶ **Lemma 8.** *In* GNN*, $v.links$ is never empty for any node $v \in V_T$.*

Proof. At the beginning of any execution, $v.links$ is not empty for any $v \in V_T$. The set $v.links$ changes only when there is a (find-predecessor) message at v (see Line 6 of Algorithm 2 and Line 7 and Line 8 of Algorithm 3). During an execution, every time v receives a message, a node is removed from $v.links$ while a new node is added to $v.links$. This also covers the case when at least two messages are received by v at the same time. The node v atomically processes all these messages in an arbitrary order. Therefore, $v.links$ never gets empty. ◀

▶ **Lemma 9.** GNN *always guarantees that on each edge of H, there is either exactly one link or exactly one message in transit.*

Proof. Initially, either a node points to its parent with an upward link or a node points to its children with downward links in the GNN protocol. Consider the edge (u, v) where $v \in u.links$. Further, consider the first time in which a message is in transit on (u, v). Immediately before this transition occurs, u must point to v, and there is not any message in transit on the edge. Therefore, w.r.t. the protocol description, the message must be sent by u to v, and the link that points from u to v has been removed. Since there is not any link while the message is in transit, it is not possible to have a second message to be in transit at the same time. When the message arrives at v, the node v points to u, and the message is removed from the edge. The next time, if a message will be transited on the edge, then v must have sent it to u and removed the link that points from v to u. ◀

▶ **Lemma 10.** *The directed tree H always remains acyclic during an execution, hence a path from a node to another node in H is always the direct path.*

Proof. The GNN protocol runs on the directed tree H in which the underlying tree – that is, T – is fixed, and the directions of links on H are only changed. Therefore, H is acyclic because the tree is always fixed, and w.r.t. Lemma 9 that shows that it never occurs a state where on the edge (u, v), u and v point to each other at the same time. ◀

The following lemma implies that a find-predecessor message always reaches the node of its predecessor using a direct path constructed by GNN.

▶ **Lemma 11.** GNN *guarantees that there is always at least one direct path in H from any leaf node u to a leaf node v where $v.links = \{v\}$.*

Proof. If the leaf node u points to itself, we are done. Otherwise, w.r.t. Lemma 8 there must be a path from u to a leaf node v since the tree H is acyclic. This path must be a direct path w.r.t. Lemma 10. The leaf node v must point to itself w.r.t. Lemma 8. ◀

Proof of Theorem 7. Using Lemma 11, it remains to show that any message traverses in finite time a direct path between two leaves. The number of edges on the direct path between any two leaves of T is upper bounded by the diameter of the tree. Further, any message that is in transit at edge (u, v) from u to v is delivered reliably at v in a finite time. Therefore, to show that a request is eventually scheduled in a finite time, it remains to show that a message will never be at a node for the second time. To obtain a contradiction, assume that the message μ is the first message that visits a node twice, and the first node visited twice by μ denoted by $v \in V_T$. With respect to Lemma 10, there is never a cycle in H. Therefore, the edge $e = (u, v)$ must be the first edge that is traversed by μ first from v to u and immediately from u to v for the second time, and μ must be the first message that traverses an edge twice. This implies that immediately before u receives μ, the node u points to v, and μ is in transit on e at the same time. This contradicts Lemma 9. ◀

3.2.2 Serving Guarantee

▶ **Theorem 12.** Gnn *is starvation-free. In other words, any scheduled request is eventually served by some server.*

Consider any of k global schedules that produced by Gnn, say π^w_{Gnn}. Assume that there is more than one request scheduled in π^w_{Gnn}. For any two requests $r_i = (v_i, t_i)$ and $r_j = (v_j, t_j)$ in π^w_{Gnn} where r_i is scheduled immediately before r_j, we see $e = (r_i, r_j)$ as a directed edge where r_j points to r_i. This edge is actually simulated by the direct path – by Lemma 11, a message always finds the node of its predecessor using a direct path on H – between the leaves v_i and v_j that is traversed by the message sent from v_j to v_i. Let F^w_{ALG} denote the graph constructed by the messages of all requests in $\mathcal{R}^w_{\text{ALG}}$.

▶ **Lemma 13.** *F^w_{ALG} is a directed path towards the head of the schedule, that is, $r^w_0 = r_{\pi^w_{\text{Gnn}}(0)}$.*

Proof. The proof has three parts.

1) **Any node of F^w_{Alg}, except the dummy request, has exactly one outgoing edge:** This is obvious since any node that invokes a request sends exactly one message.
2) **Any node in F^w_{Alg} has at most one incoming edge:** For the sake of contradiction, assume that there is a node contained in F^w_{ALG} denoted by $r = (v, t)$ with at least two incoming edges in F^w_{ALG}. This implies that two messages must reach v in H before v invokes any other request after r. However, when the first message reaches v – if any other message does not reach v before these two messages – v removes the link that points to itself and adds a link that points to its parent w.r.t. Line 7 and Line 8 of Algorithm 3. The second message cannot reach v as long as at least one request is invoked by v after invoking r. This contradicts our assumption in which two messages reach v before the time when v invokes another request after invoking r.
3) **F^w_{Alg} is connected:** To obtain a contradiction, assume that the graph F^w_{ALG} is not connected. Hence, w.r.t. the first and second parts, we have at least one connected component with at least two requests in $\mathcal{R}^w_{\text{ALG}}$ that form a cycle, and the connected component does not include the dummy request in r^w_0. Let $\mathcal{R}^{w,c}_{\text{ALG}}$ denote the requests in the connected component $F^{w,c}_{\text{ALG}}$ that forms a cycle. Consider the node z in V_T that is the lowest common ancestor of those leaves of H that invoke the requests in $\mathcal{R}^{w,c}_{\text{ALG}}$. Further, let the subtree $H^{w,c}$ of H denote the tree rooted at z. All messages of requests in $\mathcal{R}^{w,c}_{\text{ALG}}$ must traverse inside $H^{w,c}$ since $F^{w,c}_{\text{ALG}}$ is disconnected with any request in $\mathcal{R}^w_{\text{ALG}} \setminus \mathcal{R}^{w,c}_{\text{ALG}}$.

Assume that at least one message of requests in $\mathcal{R}^{w,c}_{\text{ALG}}$ reaches z. Consider the first message μ by r that reaches z at time t. If there is not any downward link at z at t, then μ is forwarded to the parent of z. This is a contradiction with the fact that $F^{w,c}_{\text{ALG}}$ is disconnected with any request in $\mathcal{R}^{w}_{\text{ALG}} \setminus \mathcal{R}^{w,c}_{\text{ALG}}$. Hence, there must be at least one downward link at z at t. On the other hand, since μ is the first message of requests in $\mathcal{R}^{w,c}_{\text{ALG}}$ that reaches z, all downward links at z at time t must have been created by some messages of requests in $H^{w,c}$ that are not in $\mathcal{R}^{w,c}_{\text{ALG}}$. Note that if a downward link at z is there since the beginning, then we assume that, w.l.o.g., it has been created by a "virtual message" sent by the node of the corresponding dummy request. Suppose μ is forwarded through one of these downward links that was created by the message of r' – as mentioned, r' can be a dummy request – that is in $H^{w,c}$ but not in $\mathcal{R}^{w,c}_{\text{ALG}}$. The original downward path from z to the leaf node of r' can be changed by the message of a request in $H^{w,c}$ – can be a request in $\mathcal{R}^{w,c}_{\text{ALG}}$. Thus, either r is scheduled immediately after some request in $H^{w,c}$ that is not in $\mathcal{R}^{w,c}_{\text{ALG}}$ or some other request in $\mathcal{R}^{w,c}_{\text{ALG}}$. In either case, we get a contradiction with our assumption in which $F^{w,c}_{\text{ALG}}$ is disconnected with any request in $\mathcal{R}^{w}_{\text{ALG}} \setminus \mathcal{R}^{w,c}_{\text{ALG}}$.

If there is not any message of a request in $\mathcal{R}^{w,c}_{\text{ALG}}$ that can reach z, then there must be at least two downward links during the execution at z that have been created by some messages of requests that are not in $\mathcal{R}^{w,c}_{\text{ALG}}$ – this holds because if there is at most one downward link at z, then a message of some request in $\mathcal{R}^{w,c}_{\text{ALG}}$ must reach z w.r.t. the definition of z. However, the existence of at least two downward links at z implies that $F^{w,c}_{\text{ALG}}$ is not connected. This is true because there are at least two downward paths that partition the requests in $\mathcal{R}^{w,c}_{\text{ALG}}$ into two disjoint components in F^{w}_{ALG} w.r.t. the definition of z and our assumption in which there is not any message of request in $\mathcal{R}^{w,c}_{\text{ALG}}$ that can reach z. This is a contradiction with our assumption in which $F^{w,c}_{\text{ALG}}$ is a connected component.

The above three parts all altogether show that F^{w}_{ALG} is indeed a directed path that points towards the dummy request in $\mathcal{R}^{w}_{\text{ALG}}$. ◀

Proof of Theorem 12. Consider any of the k global schedules resulting from GNN, say π^{w}_{GNN}. If there is only one request in π^{w}_{GNN} – there must be at least one request, that is the dummy request r^{w}_{0} – then we are done. Otherwise, w.r.t. Lemma 13 there is a path of directed edges such as $e = (r_i, r_j)$ over the requests in $\mathcal{R}^{w}_{\text{ALG}}$. When v_i obtains a server, and after r_i is served, v_i sends the server to v_j for serving r_j using an underlying routing scheme. Consequently, all requests in $\mathcal{R}^{w}_{\text{ALG}}$ are served. ◀

Proof of Theorem 1. Theorem 7 and Theorem 12 together prove the claim of the theorem. ◀

4 Analysis in a Nutshell

From a technical point of view, we achieve our main result on HSTs. In this section, we provide an analysis of GNN on HSTs in a nutshell. The complete analysis, including all proofs appears in Section 5 of [14]. Our analysis of GNN for general networks appears in Section 5.4 of [14]. The lower bound claimed in Theorem 5 is proved in Section 6 of [14].

Let ALG denote a particular distributed DSMS protocol that sends a unique message from the node of a request to the node of the predecessor request for scheduling the request (the message can be forwarded by many nodes on the path between the two nodes of the predecessor and successor requests). Consider a one-shot execution of ALG where requests are invoked at the same time 0. Let $G = (V, E)$ denote the input graph. Further, let

$B = \big(V_B = \mathcal{R}, E_B = \binom{\mathcal{R}}{2}\big)$ be the complete graph, and consider two requests $r = (v, 0)$ and $r' = (v', 0)$ in $\mathcal{R}$ where $v, v' \in V$. Assume that r' is scheduled as the successor of r by ALG in the global schedule, and w.r.t. the DSMS problem definition ALG informs v by sending the (find-predecessor) message μ' from v' to v. Therefore, the communication cost for scheduling r' equals the latency of μ'. Formally,

$$c_{\text{ALG}}(r, r') = \ell_{\text{ALG}}(\mu'). \tag{3}$$

Let $r_{src}(\mu') = r'$ denote the request corresponding with μ'. Further, let $r_{des}(\mu') = r$ denote the predecessor request r in the global schedule. We see $e = (r, r')$ as an edge in E_B that is constructed by μ'. Let us add $\mu(e)$ and $e(\mu)$ to the notation where $\mu(e)$ is the message that constructs the edge e and $e(\mu)$ is the edge that is constructed by μ. For instance, here, $\mu(e)$ refers to μ' and $e(\mu')$ refers to the edge (r, r').

Representing the solution of ALG as a forest

We observe that any of the k resulting schedules $\pi^1_{\text{ALG}}, \ldots, \pi^k_{\text{ALG}}$ can be seen as a **TSP path** that spans all requests in the corresponding schedule as follows (see Lemma 13). The TSP path F^z_{ALG} starts with the dummy request r^z_0 that is the head of π^z_{ALG}, and a request on the TSP path F^z_{ALG} is connected using an edge to its successor in the schedule π^z_{ALG}. As mentioned, the edge is constructed by the message sent by the requesting node to the node of its predecessor request. Therefore, an edge of any TSP path – that is an edge in E_B – is actually a path on the input graph that is traversed by the corresponding message. For any $F \subseteq \mathcal{F}_{\text{ALG}}$, we define the **total communication cost of F** as follows.

$$L_{\text{ALG}}(F) := \sum_{e \in F} \ell_{\text{ALG}}\big(\mu(e)\big). \tag{4}$$

Therefore, the **total communication cost of a TSP path** equals the sum of latencies of all messages that construct the TSP path. The k TSP paths represent a forest of B. Let $\mathcal{F}_{\text{ALG}}$ be the forest that consists of the k TSP paths $F^1_{\text{ALG}}, F^2_{\text{ALG}}, \ldots, F^k_{\text{ALG}}$ constructed by ALG. We slightly abuse notation and identify a subgraph F of $B = \big(\mathcal{R}, \binom{\mathcal{R}}{2}\big)$ with the set of edges contained in F. The **total communication cost of $\mathcal{F}_{ALG}$** equals the sum of total costs of the k TSP paths $F^1_{\text{ALG}}, F^2_{\text{ALG}}, \ldots, F^k_{\text{ALG}}$. For the input graph $G = (V, E)$, we denote the **weight of edge** $e = (r, r') \in E_B$ by $w_G(e) := d_G(v, v')$ where $v, v' \in V$ (recall $r = (v, t)$ and $r' = (v', t')$). Note that $d_G(v, v')$ is the weight of the shortest path between v and v' on the input graph G. Generally, **the total weight of the subgraph** F of B w.r.t. the input graph G equals the sum of weights of all edges in F. Formally,

$$W_G(F) := \sum_{e \in F} w_G(e). \tag{5}$$

▶ **Definition 14 (S-Respecting m-Forest).** *Let $G = (V, E)$ be a graph and $m \leq |V|$. A forest $\mathcal{F}$ of G is called an m-forest if $\mathcal{F}$ consists of m trees. Further, let $S \subseteq V$, $|S| \leq m$ be a set of at most m nodes. An m-forest $\mathcal{F}$ of G is S-respecting if the nodes in S appear in different trees of $\mathcal{F}$.*

Let $\mathcal{R}_D$ denote the set of k dummy requests in $\mathcal{R}$. W.r.t. the Definition 14, $\mathcal{F}_{\text{ALG}}$ is an $\mathcal{R}_D$-respecting spanning k-forest of $B = \big(\mathcal{R}, \binom{\mathcal{R}}{2}\big)$. *From now on, we consider the HST T as the input graph.*

Locality-based forest

For any subtree T' of T and any subgraph F of B, let $F(T')$ denote the subgraph of F that is induced by those requests contained in F that are also in T'. Further, let $F^1, F^2, \ldots, F^k$ denote the k trees of the spanning k-forest $\mathcal{F}$ of B. Let $\mathcal{F}_{\text{GRD}}$ be any $\mathcal{R}_D$-respecting spanning k-forest of B with the following basic **locality properties**.

I. [**Intra-Component Property**] For any subtree T' of T and for any $w \in \{1, \ldots, k\}$, the component $F^w_{\text{GRD}}(T')$ is a tree.

II. [**Inter-Component Property**] For any subtree T' of T, suppose that there are at least two non-empty components $F^z_{\text{GRD}}(T')$ and $F^w_{\text{GRD}}(T')$ where $w \neq z$ and $w, z \in \{1, \ldots, k\}$. Any of these components includes a dummy request.

We call such a forest a locality-based forest. Any locality-based forest is denoted by $\mathcal{F}_{\text{GRD}}$. The following theorem provides a general version of Theorem 3.

▶ **Theorem 15.** *Let I denote an instance of DSMS that consists of an HST T where the communication is asynchronous and a set $\mathcal{R}$ of requests that are simultaneously invoked at leaves of T. The protocol* Alg *is optimal if the total cost of the resulting forest by* Alg *is upper bounded by the total weight of $\mathcal{F}_{\text{GRD}}$.*

4.1 Optimality of GNN on HSTs

Consider a one-shot execution of Gnn, and suppose that $\mathcal{F}_{\text{GNN}}$ is the resulting forest when running Gnn on the given HST T w.r.t. the input sequence $\mathcal{R}$. With respect to Theorem 15, and the fact that Gnn only sends one unique message for scheduling a request to its predecessor, it is sufficient to show that the forest $\mathcal{F}_{\text{GNN}}$ can be transformed into a locality-based forest such that the total cost of $\mathcal{F}_{\text{GNN}}$ is upper bounded by the total weight of $\mathcal{F}_{\text{GRD}}$. During an execution of Gnn, the Intra or Inter-Component property can be violated (see Figure 1). Consider the following situations:

1. A server goes back to a subtree after the time when it leaves the subtree.
2. A request in a subtree of T that initially hosts at least one server is served by a server that is not initially in the subtree.
3. Two requests in a subtree of T that does not initially host any server, are served by different servers.

The first situation violates the Intra-Component property. Any of the second and the third situation violates the Inter-Component property. In the following, we characterize the Intra-Component and the Inter-Component properties by considering a timeline for the messages that enter and leave a subtree of T. Consider a message μ that enters the subtree T' of T. Another message can enter T' only after some message μ' has left T' after μ entered T' – the arrival times of messages μ and μ' at the root of T' can be the same (cf. Lemma 5.3 and Lemma 5.6 of [14]). Similarly, a message can leave T' after μ' left T' only after some message has entered T' after μ' left T'. We refer to Lemma 5.6 for more details. Consider a message μ that enters T'. The fact that μ enters T' implies that a server will leave T' for serving $r_{src}(\mu)$. Let μ' denote the first message that leaves T' after μ entered T'. Leaving μ' from T' implies that a server will enter T' for serving $r_{src}(\mu')$. If $r_{src}(\mu')$ is in the same TSP path of $\mathcal{F}_{\text{GNN}}$ with $r_{des}(\mu)$, then the server that had served $r_{des}(\mu)$ goes back to T' for serving $r_{src}(\mu')$ after it left T', and therefore the Intra-Component property is violated. Otherwise, the Inter-Component property is violated since two requests in T' are served by two different servers in which at least one of the servers is initially outside of T'. We say Gnn makes an **Inter-Component gap** (μ, μ') on T' in the latter case and an **Intra-Component gap** (μ, μ') on T' in the former case.

Transformation

We transform $\mathcal{F}_{\text{GNN}}$ through *closing the gaps* that are made by GNN on all subtrees of T. A message μ' can leave from several subtrees of T such that different messages enter the subtrees before μ'. Therefore, GNN can make different gaps with the same message μ' on this set of subtrees of T. We especially refer to Lemma 5.9 and Lemma 5.11 of [14] for more details on the gaps of the subtrees of T. We consider the lowest subtree in this set and let (μ, μ') be a gap on that. We **close the gap** (μ, μ') by removing $e(\mu')$ and by adding the new edge $\big(r_{des}(\mu), r_{src}(\mu')\big)$. In the example of Figure 1, for instance, the red edges are removed and the new edges (r_0^1, r_b) and (r_b, r_c) are added. When we close the gap (μ, μ'), all other gaps (μ'', μ') that are on higher subtrees are also closed. Therefore, we transform $\mathcal{F}_{\text{GNN}}$ into a new forest $\mathcal{F}_{mdf}$ by means of closing all gaps. The following lemma shows that $\mathcal{F}_{mdf}$ is indeed the locality-based forest.

▶ **Lemma 16.** *$\mathcal{F}_{mdf}$ is an $\mathcal{R}_D$-respecting spanning k-forest of B that satisfies the Intra-Component and the Inter-Component properties.*

It remains to show that the total cost of $\mathcal{F}_{\text{GNN}}$ is upper bounded by the total weight of the new forest $\mathcal{F}_{mdf}$. Formally, we want to show that $L_{\text{GNN}}(\mathcal{F}_{\text{GNN}}) \leq W_T(\mathcal{F}_{mdf})$. Using Lemma 11, a message always finds the node of its predecessor using a direct path on T in any execution of GNN. Regarding to our communication model described in Section 2.1, therefore, for every edge $e \in \mathcal{F}_{\text{GNN}}$ we have

$$\ell_{\text{GNN}}\big(\mu(e)\big) \leq w_T(e) \tag{6}$$

Let (μ, μ') be the gap on the lowest subtree of T among all subtrees of T with gaps (μ'', μ') for any message μ'' that makes a gap with μ'. By closing the gap (μ, μ'), we remove $e^{old} := \big(r_{src}(\mu'), r_{des}(\mu')\big)$ and add the new edge $e^{new} := \big(r_{src}(\mu'), r_{des}(\mu)\big)$. Using (6), we are immediately done if the latency of μ' is upper bounded by the weight of e^{new}. However, the latency of μ' can be larger than the weight of e^{new}. By contrast, the weight of e^{new} is lower bounded by the latency of μ (cf. Corollary 5.10 and Lemma 5.15 of [14]). This lower bound gives us the go-ahead to show that the weight of e^{new} can be seen as an "amortized" upper bound for $\ell_{\text{GNN}}(\mu')$. In the following, we provide **an overview of our amortized analysis** that appears in Section 5.3.3 of [14]. Let $E^{new} := \mathcal{F}_{mdf} \setminus \mathcal{F}_{\text{GNN}}$ and $E^{old} := \mathcal{F}_{\text{GNN}} \setminus \mathcal{F}_{mdf}$ be the sets of all edges that are added and removed during the transformation of $\mathcal{F}_{\text{GNN}}$, respectively. Further, we consider a set of edges that provides enough "potential" for our amortization.

$$E^{pot} := \big\{e \in \mathcal{F}_{\text{GNN}} : \big(\mu(e), \mu(e')\big) \text{ is a gap for some } e' \in E^{old}\big\}.$$

For every edge $e \in E^{old}$, let $E^{pot}(e) := \big\{e' \in E^{pot} : \big(\mu(e'), \mu(e)\big) \text{ is a gap}\big\}$. Further, for every edge $e \in E^{pot}$, let $E^{old}(e) := \big\{e' \in E^{old} : \big(\mu(e), \mu(e')\big) \text{ is a gap}\big\}$. In this overview, we consider the **simple case** where 1) $|E^{old}(e)| = 1$ for every edge $e \in E^{pot}$ and $|E^{pot}(e)| = 1$ for every edge $e \in E^{old}$. Further, 2) the sets E^{old} and E^{pot} do not share any edge. The execution provided by Figure 1 represents an example of the above simple case. We define the **potential function** $\Phi(F)$ for a subset F of $\mathcal{F}_{\text{GNN}}$ as follows $\Phi(F) := W_T(F) - L_{\text{GNN}}(F)$. W.l.o.g., we assume that the edges in E^{old} are sequentially replaced with the edges in E^{new}. Hence, assume that e_i^{old} is replaced with e_i^{new} during the i-th replacement. Let also e_i^{pot} be the only edge in $E^{pot}(e_i^{old})$.

▶ **Lemma 17.** *If $|E^{old}(e)| = 1$ for every edge $e \in E^{pot}$, $|E^{pot}(e)| = 1$ for every edge $e \in E^{old}$, and $E^{old} \cap E^{pot} = \emptyset$, then*

$$w_T(e_i^{old}) \leq w_T(e_i^{new}) + \Phi\left(E^{pot} \setminus \left\{e_1^{pot}, \ldots, e_{i-1}^{pot}\right\}\right) - \Phi\left(E^{pot} \setminus \left\{e_1^{pot}, \ldots, e_i^{pot}\right\}\right) \tag{7}$$

for every $i \geq 1$.

Proof. Using the definition of the potential function Φ and the definitions of the total weight and the total communication cost of a subset of edges in $\mathcal{F}_{\text{GNN}}$, we have

$$\Phi\left(E^{pot} \setminus \left\{e_1^{pot}, \ldots, e_{i-1}^{pot}\right\}\right) - \Phi\left(E^{pot} \setminus \left\{e_1^{pot}, \ldots, e_i^{pot}\right\}\right) = w_T(e_i^{pot}) - \ell_{\text{GNN}}(e_i^{pot}).$$

Therefore, we need to show that $w_T(e_i^{old}) \leq w_T(e_i^{new}) + w_T(e_i^{pot}) - \ell_{\text{GNN}}(e_i^{pot})$. Let the subtree T' of T be the lowest subtree such that $\left(\mu(e_i^{pot}), \mu(e_i^{old})\right)$ is a gap on T'. This implies that $w_T(e_i^{new}) = \delta(T')$. On the other hand, using Lemma 5.15 of [14] we have $\ell_{\text{GNN}}(e_i^{pot}) \leq \delta(T') = w_T(e_i^{new})$. It remains to show that $w_T(e_i^{old}) \leq w_T(e_i^{pot})$. Let T_j'' be the highest subtree of T such that $\left(\mu(e_i^{pot}), \mu(e_i^{old})\right)$ is a gap on T_j'' and T_j'' is a child subtree of T''. The message $\mu(e_i^{old})$ does not leave T'' since $E^{pot}(e_i^{old}) = \left\{e_i^{pot}\right\}$. Hence, $w_T(e_i^{old}) = \delta(T'')$. On the other hand, the fact that the message $\mu(e_i^{pot})$ enters T_j'' indicates that $w_T(e_i^{pot}) \geq \delta(T'')$. Consequently, $w_T(e_i^{pot}) \geq w_T(e_i^{old})$ and we are done. ◀

When we sum up (7) for all i, we get

$$W_T(E^{old}) \leq W_T(E^{new}) + \Phi\left(E^{pot}\right). \tag{8}$$

Using the definition of the potential function Φ and using $L_{\text{GNN}}(E^{old}) \leq W_T(E^{old})$ w.r.t (6), therefore we get $L_{\text{GNN}}(E^{pot}) + L_{\text{GNN}}(E^{old}) \leq W_T(E^{new}) + W_T(E^{pot})$. Hence, we have $L_{\text{GNN}}(\mathcal{F}_{\text{GNN}}) \leq W_T(\mathcal{F}_{mdf})$ since $\mathcal{F}_{mdf} = \mathcal{F}_{\text{GNN}} \setminus E^{old} \cup E^{new}$ and $L_{\text{GNN}}\left(\mathcal{F}_{\text{GNN}} \setminus (E^{old} \cup E^{pot})\right) \leq W_T\left(\mathcal{F}_{\text{GNN}} \setminus (E^{old} \cup E^{pot})\right)$ w.r.t (6).

▶ **Lemma 18.** *The total cost of $\mathcal{F}_{\text{GNN}}$ is upper bounded by the total weight of $\mathcal{F}_{mdf}$.*

▶ **Theorem 19.** *The forest $\mathcal{F}_{\text{GNN}}$ can be transformed into the locality-based forest $\mathcal{F}_{\text{GRD}}$ such that the total cot of $\mathcal{F}_{\text{GNN}}$ is upper bounded by the total weight of $\mathcal{F}_{\text{GRD}}$.*

5 Further Related Work

Distributed k-server problem

In Section 1, we have seen that the distributed k-server problem is an application of the DSMS problem. In [8], a general translator that transforms any deterministic global-control competitive k-server algorithm into a distributed competitive one is provided. This yields poly(k)-competitive distributed protocols for the line, trees, and the ring synchronous network topologies. In [8], a lower bound of $\Omega(\max\{k, (1/D) \cdot (\log n / \log\log n)\})$ on the competitive ratio for the distributed k-server problem against adaptive online adversaries is also provided where n is the number of processors. D is the ratio between the cost to move a server and the cost to transmit a message over the same distance in synchronous networks. [4] and [9] study OSD on HSTs and lines, respectively. [4] provides an upper bound of $\text{O}\left(\log^3 n\right)$ and [9] provides an upper bound of $\text{O}\left(\log n\right)$ on the competitive ratio for OSD where n is the number of leaves of the input HST as well as the number of nodes of the input line.

Distributed queuing problem and link-reversal-based protocols

A well-known class of protocols has been devised based on link reversals to solve distributed problems in which the distributed queuing problem is at the core of them [2, 17, 19, 21, 22, 23, 24]. In a distributed link-reversal-based protocol nodes keep a link pointing to neighbors in the current or future direction of the server. When sending a message over an edge to request the server, the direction of the link flips. We devise the GNN protocol that is – to the best of our knowledge – the first link-reversal-based protocol that navigates more

than one server. A well-studied link-reversal-based protocol is called Arrow [19, 21, 22]. Several other tree-based distributed queueing protocols that are similar to Arrow have also been proposed. They operate on fixed trees. The Relay protocol has been introduced as a distributed transactional memory protocol [24]. It is run on top of a fixed spanning tree similar to Arrow; however, to more efficiently deal with aborted transactions, it does not always move the shared object to the node requesting it. Further, in [2], a distributed directory protocol called Combine has been proposed. Combine like Gnn runs on a fixed overlay tree, and it is in particular shown in [2] that Combine is starvation-free.

The first paper to study the competitive ratio of concurrent executions of a distributed queueing protocol is [16]. It shows that in synchronous executions of Arrow on a tree T for one-shot executions, the total cost of Arrow is within a factor $\mathrm{O}(\log m)$ compared to the optimal queueing cost on where m is the number of requests. This analysis has later been extended to the general concurrent setting where requests are invoked over time. In [15], it is shown that in this case, the total cost of Arrow is within a factor $O(\log D)$ of the optimal cost on T where D is the diameter of T. Later, the same bounds have also been proven for Relay [24]. Typically, these protocols are run on a spanning tree or an overlay tree on top of an underlying general network topology. In this case, the competitive ratio becomes $O(s \cdot \log D)$, where s is the stretch of the tree. Finally, [13] has shown that when running Arrow on top of HSTs, a randomized distributed online queueing protocol is obtained with expected competitive ratio $O(\log n)$ against an oblivious adversary even on general n-node network topologies. The result holds even if the queueing requests are invoked over time and even if communication is asynchronous. The main technical result of the paper shows that the competitive ratio of Arrow is constant on HSTs.

Online tracking of mobile users

A similar problem to DSMS is the online mobile user tracking problem [3]. In contrast with DSMS where a request r results in moving a server to the requesting point, here the request r can have two types: find request that does not result in moving the mobile user and move request. A request in DSMS that is invoked by v can be seen as a combination of a find request that is invoked at v in the mobile user problem and a move request invoked at the current address of the mobile user. The goal is to minimize the sum of the total communication cost and the total cost incurred for moving the mobile user. [3] provides an upper bound of $\mathrm{O}(\log^2 n)$ on the competitive ratio for the online mobile user problem for one-shot executions. Further, [1] provides a lower bound of $\Omega(\log n / \log\log n)$ on the competitive ratio for this problem against an oblivious adversary.

References

1 N. Alon, G. Kalai, M. Ricklin, and L. Stockmeyer. Lower bounds on the competitive ratio for mobile user tracking and distributed job scheduling. In *FOCS*, 1992.

2 H. Attiya, V. Gramoli, and A. Milani. A provably starvation-free distributed directory protocol. In *SSS*, 2010.

3 B. Awerbuch and D. Peleg. Online tracking of mobile users. *Journal of the ACM*, 1995.

4 Y. Azar, A. Ganesh, R. Ge, and D. Panigrahi. Online Service with Delay. In *STOC*, 2017.

5 N. Bansal, N. Buchbinder, A. Madry, and J. S. Naor. A Polylogarithmic-Competitive Algorithm for the k-Server Problem. In *FOCS*, 2011.

6 Y. Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *FOCS*, 1996.

7 Y. Bartal, A. Fiat, and Y. Rabani. Competitive algorithms for distributed data management. In *STOC*, 1992.

8 Y. Bartal and A. Rosen. The distributed k-server problem-a competitive distributed translator for k-server algorithms. In *FOCS*, 1992.

9 M. Bienkowski, A. Kraska, and P. Schmidt. Online Service with Delay on a Line. In *SIROCCO*, 2018.

10 M. J. Demmer and M. Herlihy. The arrow distributed directory protocol. In *DISC*, 1998.

11 J. Fakcharoenphol, S. Rao, and K. Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *STOC*, 2003.

12 M. Ghaffari and C. Lenzen. Near-optimal distributed tree embedding. In *DISC*, 2014.

13 A. Ghodselahi and F. Kuhn. Dynamic Analysis of the Arrow Distributed Directory Protocol in General Networks. In *DISC*, 2017.

14 Abdolhamid Ghodselahi, Fabian Kuhn, and Volker Turau. Concurrent Distributed Serving with Mobile Servers. *arXiv*, 2019. `arXiv:1902.07354`.

15 M. Herlihy, F. Kuhn, S. Tirthapura, and R. Wattenhofer. Dynamic analysis of the arrow distributed protocol. *Theoretical Computer Science*, 2006.

16 M. Herlihy, S. Tirthapura, and R. Wattenhofer. Competitive concurrent distributed queuing. In *PODC*, 2001.

17 P. Khanchandani and R. Wattenhofer. The Arvy Distributed Directory Protocol. In *SPAA*, 2019.

18 M. Manasse, L. McGeoch, and D. Sleator. Competitive algorithms for on-line problems. In *STOC*, 1988.

19 M. Naimi and M. Trehel. An Improvement of the $\log n$ Distributed Algorithm for Mutual Exclusion. In *ICDCS*, 1987.

20 D. Peleg. *Distributed computing: a locality-sensitive approach*. SIAM, 2000.

21 K. Raymond. A tree-based algorithm for distributed mutual exclusion. *ACM Transactions on Computer Systems*, 1989.

22 J. L. van de Snepscheut. Fair mutual exclusion on a graph of processes. *Distributed Computing*, 1987.

23 J. Welch and J. Walter. Link reversal algorithms. *Synthesis Lectures on Distributed Computing Theory*, 2011.

24 B. Zhang and B. Ravindran. Dynamic analysis of the relay cache-coherence protocol for distributed transactional memory. In *IPDPS*, 2010.

Table 1 The essential notations used throughout the paper.

Notation	Definition	Page
n	number of pints/nodes/processors	2
k	number of servers	2
$\mathcal{R}$	input requests	5
$r_i = (v_i, t_i)$	request r_i that is invoked by node v_i at time t_i	5
π^z_{ALG}	z-th schedule as one of the k resulting schedules by ALG	5
$r^z_0 = (v^z, 0)$	dummy request z as the tail of π^z_{ALG}	5
s^z	z-th server that serves all requests in π^z_{ALG}	5
$\mathcal{R}^z_{\text{ALG}}$	request set of π^z_{ALG}	6
$\pi^z_{\text{ALG}}(i)$	index of the request scheduled at the i-th position of π^z_{ALG}	6
$\ell_{\text{ALG}}(\mu)$	latency of message μ in an execution of ALG	6
$c_{\text{ALG}}(r_i, r_j)$	cost incurred by ALG for scheduling r_j as the successor of r_i	6
$C_{\text{ALG}}(\pi^z_{\text{ALG}})$	total cost incurred by ALG for scheduling requests in z-th schedule	6
C_{ALG}	total cost incurred by ALG	6
$d_G(u, v)$	weight of the shortest path between u and v on the input graph G	6
H	directed version of T that is changing during a GNN execution	7
$\mu(v)$	find-predecessor message sent by v	7
B	complete graph on requests in $\mathcal{R}$	12
$r_{src}(\mu)$	corresponding request with message μ	12
$r_{des}(\mu)$	predecessor request of $r_{src}(\mu)$	12
$e(\mu)$	edge constructed by message μ	12
$\mu(e)$	message that constructs the edge e	12
T	input HST	12
$\mathcal{F}_{\text{ALG}}$	resulting forest by ALG; also, set of edges of the forest	12
F^z_{ALG}	z-th TSP path of $\mathcal{F}_{\text{ALG}}$; also, set of edges of the z-th TSP path	12
$L_{\text{ALG}}(F)$	total cost of F such that $F \subseteq \mathcal{F}_{\text{ALG}}$	12
$w_G\big(e = (r_i, r_j)\big)$	weight of the shortest path between v_i and v_j on the input graph G	12
$W_G(F)$	total weight of F w.r.t. measurements on the input graph G	12
$\mathcal{R}_D$	set of k dummy requests; $\mathcal{R}_D \subseteq \mathcal{R}$	12
$F(T')$	subgraph of F induced by the requests contained in F and T'	13
$\mathcal{F}_{\text{GRD}}$	locality-based forest	13
(μ, μ')	gap	13
$\mathcal{F}_{mdf}$	resulting forest by the transformation of $\mathcal{F}_{\text{GNN}}$	14
E^{old}	set of edges removed throughout the transformation of $\mathcal{F}_{\text{GNN}}$	14
E^{new}	set of edges added throughout the transformation of $\mathcal{F}_{\text{GNN}}$	14
E^{pot}	$\big\{e \in \mathcal{F}_{\text{GNN}} : \big(\mu(e), \mu(e')\big)$ is a gap for some $e' \in E^{old}\big\}$	14
$E^{pot}(e)$	subset of E^{pot} filtered out by $e \in E^{old}$	14
$E^{old}(e)$	subset of E^{old} filtered out by $e \in E^{pot}$	14
$\Phi(F)$	potential of F	14