A Poly-Log Approximation for Transaction Scheduling in Fog-Cloud Computing and Beyond

RAMESH ADHIKARI, School of Computer & Cyber Sciences Augusta University, USA
COSTAS BUSCH, School of Computer & Cyber Sciences Augusta University, USA
PAVAN POUDEL, University of Houston-Clear Lake, USA

Transaction scheduling is crucial to efficiently allocate shared resources in a conflict-free manner in distributed systems. We investigate the efficient scheduling of transactions in a network of fog-cloud computing model, where transactions and their associated shared objects can move within the network. The schedule may require objects to move to transaction nodes, or the transactions to move to the object nodes. Moreover, the schedule may determine intermediate nodes where both objects and transactions meet. Our goal is to minimize the total combined cost of the schedule. We focus on networks of constant doubling dimension, which appear frequently in practice. We consider a batch problem where an arbitrary set of nodes has transactions that need to be scheduled. First, we consider a single shared object required by all the transactions and present a scheduling algorithm that gives an $O(\log n \cdot \log D)$ approximation of the optimal schedule, where n is the number of nodes and D is the diameter of the network. Later, we consider transactions accessing multiple shared objects (at most k objects per transaction) and provide a scheduling algorithm that gives an $O(k \cdot \log n \cdot \log D)$ approximation. We also provide a fully distributed version of the scheduling algorithms where the nodes do not need global knowledge of transactions.

CCS Concepts: • **Computing methodologies** → **Distributed algorithms**; • **Theory of computation** → **Scheduling algorithms**.

Additional Key Words and Phrases: Distributed systems, shared object, fog-cloud computing, transaction scheduling, communication cost, doubling dimension graph.

1 Introduction

There are distributed systems that process a large number of concurrent transactions in industry sectors like FinTech, e-commerce, social media, telecommunications, fog-cloud computing, etc. [9, 42, 44]. A coordination problem arises when multiple transactions attempt to simultaneously read from and write to the same shared objects, such as common accounts. To prevent inconsistencies while accessing shared objects, each transaction should be executed in an atomic way. Traditionally, locks are used to coordinate the actions of transactions and prevent inconsistencies while accessing shared objects [43]; however, if locks are not handled properly, that leads to a deadlock and priority inversion. To address these issues, we need to efficiently schedule the execution of transactions ensuring that each shared object is accessed by only one transaction at a time.

Consider a distributed system consisting of n processing nodes interconnected in a network represented as graph G. A set of transactions $\mathcal{T}$ and a set of shared objects O that are required to be accessed by the transactions are initially located at different nodes of the graph G. We consider the case where multiple transactions require to access the shared objects concurrently. In order for a transaction to execute, it needs to have an exclusive access to all the required objects. This can be achieved by either moving the objects to the transaction node (data-flow model [18, 37]), or moving the transaction to the object nodes (control-flow model [33, 36]). Recently, Busch *et al.* [4] considered a more flexible transaction execution model called *dual-flow* model where objects and transactions meet at arbitrary nodes of G during the execution.

Busch *et al.* [4] studied transaction scheduling problem in *Trees* with the objective of minimizing total communication cost. However, trees are inherently fragile (i.e., they may easily get disconnected on edge removals), lack redundancy, and are inefficient for modeling general networks (e.g., distances may not be preserved and

Authors' Contact Information: Ramesh Adhikari, School of Computer & Cyber Sciences Augusta University, Augusta, Georgia, USA, radhikari@augusta.edu; Costas Busch, School of Computer & Cyber Sciences Augusta University, Augusta, Georgia, USA, kbusch@augusta.edu; Pavan Poudel, University of Houston-Clear Lake, Houston, Texas, USA, poudel@uhcl.edu.

Source	Scheduling Model	Metric	Communication cost approximation	
			Single object	Multiple objects (at most k)
Busch *et al.* [4]	Centralized	Tree	$O(1)$	O(k)
This paper	Centralized and distributed	Constant doubling dimension graph	$O(\log n \cdot \log D)$	$O(k \cdot \log n \cdot \log D)$

Table 1. Comparison of our proposed scheduler with the most related work [4], where n is the total number of nodes, D is the diameter of graph G, and k is the maximum number of objects accessed by each transaction.

congestion may increase). In contrast, graphs with a constant doubling dimension [15] provide a compelling alternative, particularly in the context of scalable, fault-tolerant, and real-time systems, by introducing redundancy, robustness, and flexibility needed for real-world high-performance distributed systems. Typical examples of graphs with constant doubling dimension are 2-dimensional grids [1, 12] and randomly distributed unit disk graphs [28]. Such graphs are used in various distributed and routing systems, such as solving communication and graph problems [24, 26], resource management [10, 12, 40], vehicular routing [19], routing and location services in networks [1, 11, 25].

In this paper, we studied transaction scheduling in a distributed system modeled as a graph G with a constant doubling dimension, aiming to minimize the total communication cost for executing transactions. Similar to [4], we also adopted the dual-flow model for transaction execution. We assume that moving an object along a unit-length edge in G costs α, while moving a transaction along the unit-length edge costs $\beta < \alpha$. (The case $\alpha \leq \beta$ corresponds to the well-studied data-flow model [6–8, 37, 38].) We consider a synchronous communication model, where time is divided into discrete steps [18]. At each time step, a node may perform one of the following actions: receive objects or transactions from neighboring nodes; execute transactions that have gathered all required objects; or forward objects or transactions to neighboring nodes.

Contributions. The primary goal of this paper is to provide an efficient schedule for the execution of transactions in a network G accessing shared objects. We consider G as a graph of constant doubling dimension. Table 1 compares our results with the most related work [4]. We provide the following contributions:

- Assuming each node has global knowledge of all transactions and each transaction accesses a single shared object, we propose a global-aware distributed scheduling algorithm with an $O(\log n \cdot \log D)$ approximation in communication cost, where n is total number of nodes and D is the network diameter.
- For transactions accessing up to k shared objects, we present a global-aware distributed scheduling algorithm with an $O(k \cdot \log n \cdot \log D)$ approximation in communication cost.
- When nodes do not have global knowledge of transactions, we introduce a fully distributed scheduling algorithm for the single-object case and explain how it can be extended to handle multiple shared objects.

Techniques. When there is a single shared object o to be accessed by all the transactions in $\mathcal{T}$, our scheduling algorithm consists of two main steps: (1) Find a set of nodes S_f in G where transactions and the object o can meet together with minimal cost, and (2) Make the object o visit each node in S_f and execute the transactions in order. Each node in S_f contains at least one transaction. When the object visits a node in S_f, respective transaction(s) in the node will execute sequentially. The overall schedule provides a global order of all transactions in $\mathcal{T}$. We show that the schedule of our algorithm has $O(A \log D)$ approximation, where A be the approximation of the optimal TSP tour for the nodes in S_f.

To calculate the set S_f, we use a hierarchical sparse partition H of G, as in [20]. Then the object follows an approximate TSP tour for visiting the nodes of S_f. We can use any of the two ways to calculate the tour.

- *Universal TSP*: As outlined in [20], the hierarchy H can be used to provide a global order of all the nodes in G which is called a *universal TSP tour*. For any subset of G, the respective order of nodes in H gives an

$A = O(\log n)$ approximation of the optimal TSP tour. When considering S_f, using the respective TSP tour, we obtain the $O(\log n \cdot \log D)$ approximation.

- *MST*: Alternatively, we can first calculate a minimum weight spanning tree (MST) with the nodes of S_f, which can be used to approximate the tour with $A = 2$. Thus, this approach gives an overall approximation of $O(\log D)$ for our proposed algorithm.

We would like to note that although the MST approach gives a better approximation, the distributed version of the approach could require more messages. In particular, assuming that the hierarchical partition H is given, the number of messages to compute the tour with H is $O(n \log D)$. While using the MST approach for the tour, it involves $O(n^2)$ messages.

When transactions are required to access multiple shared objects, we extend the algorithm for a single shared object. First, a set of nodes S in G is calculated with respect to each individual object and the transactions requiring that object where the transactions and the object can meet together with minimal cost. Next, for each transaction T, a common node s' is found where all the required objects for T will gather with minimal additional cost. s' is added to the final set of nodes S_f and T will also move to s' for the execution. Following a TSP tour, objects will move to their respective nodes in S_f, and the transactions in each node execute sequentially when all the required objects gather at that node.

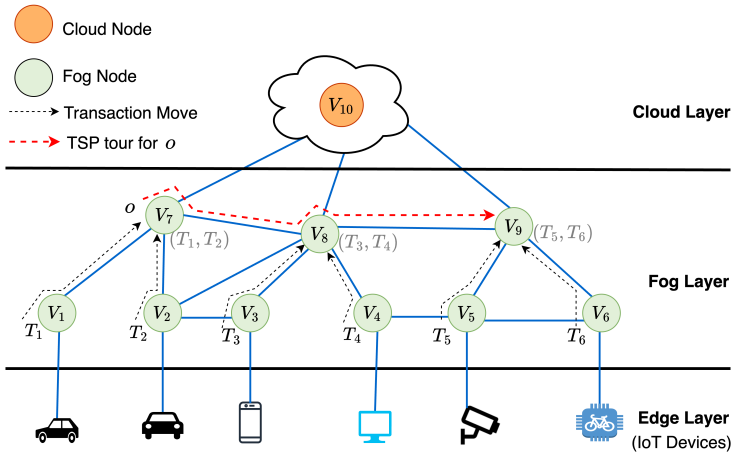

Fig. 1. Illustration of fog-cloud computing model.

Applications. Our scheduling technique is applicable in many distributed applications such as transactional memory (TM) [17, 39], IoT and fog-cloud computing [13, 29, 30, 42], financial systems [14], and Big Data [21]. In the following, we discuss the applicability in fog-cloud computing in detail.

Fog-cloud computing: IoT applications often rely on centralized cloud computing nodes. However, there exists a high communication delay between cloud and IoT devices [22, 30]. Distributed fog computing addresses this challenge by providing resources at the edge of the network, closer to end devices that reduce network delay [27, 30]. In fog-cloud computing model, transactions are generated from user IoT devices and sent to the fog layer. If the fog node has sufficient resources and the object it requires to execute, then the transaction is scheduled and executed in the fog layer; otherwise, it is routed to a cloud server for execution [22, 27, 30]. Figure 1 shows a simple example of the fog-cloud computing.

As a practical example, consider a vehicle-tracking scenario where IoT devices, like traffic cameras, transmit vehicle data as transactions to the fog layer. Each transaction creates, reads, or updates a vehicle-related object. Due to vehicle mobility, the object may be distant from the transaction node, leading to communication overhead. To reduce this cost, the object, the transaction, or both can be moved. Our proposed algorithm efficiently enables such movements for better transaction scheduling and execution.

Paper Organization: The rest of the paper is organized as follows: We discuss related work in Section 2 and model and preliminaries in Section 3. We present global-aware distributed scheduling algorithms in Section 4, followed by fully-distributed algorithms in Section 5. We conclude our paper in Section 6. Some of the pseudocodes, proofs, and other details are omitted due to space constraints.

2 Related Work

Several studies have explored the scheduling of transactions within distributed systems considering fog-cloud computing [30, 31]. Nikoui et al. [31] employed a genetic algorithm (GA) to minimize energy consumption in task scheduling for green cloud computing systems. However, their approach assumes a central cloud broker utilizing the GA to allocate tasks across a set of virtual machines. Later, Nikoui et al. [30] introduced a cost-aware genetic-based task scheduling algorithm for fog-cloud environments, also relying on a centralized fog node, termed a fog broker, to handle transaction scheduling. Peixoto et al. [32] proposed a multilevel fog-cloud architecture for transaction scheduling. All of these algorithms depend on a single fog broker, and none of these works considers efficient communication analysis, such as determining when to move the object, the transaction, or both to intermediate nodes, to optimize communication costs and improve the overall efficiency of transaction scheduling.

Extensive research [5, 7, 8, 23, 33–35] has been done on scheduling transactions in distributed transactional memory systems. In transactional memory, each transaction requires access to specific objects for execution. Coordination between objects and transactions is often necessary for accessing objects and executing transactions. The majority of TM research is based on the data-flow model [18, 41], in which transactions remain static while objects move between nodes to reach the locations where the transactions are held. In contrast, several studies adopt the control-flow model [33, 36], where transactions move across nodes to access static objects. Lately, the dual-flow model [4, 16] has been introduced in which both objects and transactions are moved to some intermediate node to minimize total communication cost. Additionally, there are transaction scheduling techniques in the context of blockchain sharding [2, 3]. However, in blockchain sharding, objects are static and only transactions send messages, without a focus on reducing communication costs.

The most closely related work is [4], where the authors provide two variants of transaction scheduling algorithms for minimizing communication cost in transactional memory. However, their results are restricted to trees and do not apply to graphs with constant doubling dimension that we consider here. Moreover, the algorithms proposed in [4] rely on a centralized model where nodes have access to global information about transactions and objects. In contrast, this paper also presents a fully distributed version of the algorithms that operate without requiring global knowledge of transactions. The novelty of our algorithms lies in the use of hierarchical sparse partition H to compute the schedule.

3 Technical Preliminaries and Model

We model the network as a connected weighted graph $G = (V, E, w)$. The n vertices in the set V represent the processing nodes that may hold object(s) and transaction(s). Communication links between nodes are represented by edges in the set $E \subseteq V \times V$, and each edge is associated with a weight assigned by the function $w : E \to \mathbb{R}^+$ to denote the distance between the two nodes. The minimum distance between a pair of nodes is 1. A *path* p in G is a sequence of nodes with respective edges between adjacent nodes. There is a path between each pair of nodes and the distance between two nodes varies from 1 to D, where D is the diameter of G.

For a node v, the y neighborhood $N_y(v)$, where $y \geq 0$, is the set of nodes that are at a distance at most y from v (we also include v in $N_y(v)$). For a set of nodes S, the y-neighborhood of S is $N_y(S) = \bigcup_{v \in S} N_y(v)$, which is all the nodes that are at a distance at most y from some node in S. The length of a path p in G, denoted as $|p|$, is the sum of the weights of its edges; if the path is just a single node, then its length is trivially 0.

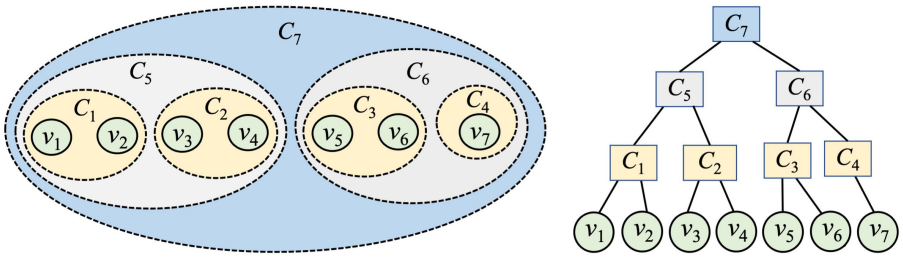

Fig. 2. Hierarchical partitioning of graph G with constant doubling dimension.

Similar to the previous work in [4], we adopt dual-flow model, allowing both shared objects and transactions the flexibility to move between the network nodes. The cost of moving an object of size α across a unit-weight edge is represented by α. Similarly, the cost of moving a transaction across a unit-weight edge is denoted by β where $\alpha > \beta$. The scheduling algorithm is responsible for determining the execution schedule $\mathcal{E}$ of the transactions, considering the movements of both objects and transactions in G.

We consider graphs with constant doubling dimension ($2^{\delta} = \Theta(1)$) as described in [40]. In the following definition, a *ball of radius* r refers to the $N_r(v)$-neighborhood of some node v.

Definition 3.1 (doubling-dimension of graph [40]). The doubling dimension of a graph G is the smallest value of δ such that every ball of radius r in G can be covered by the union of at most 2^{δ} balls of radius $r/2$. If δ remains constant, we say G has a constant doubling dimension.

3.1 Partition Hierarchy

Given a weighted graph $G = (V, E, w)$ with diameter $D \leq n$, we can build a partition hierarchy H with $O(\log D)$ levels (or layers), similar to the construction in [20]. A (r, σ, I)-partition divides V into a group of sets $\mathcal{X} = \{X_1, X_2, \dots\}$ such that each set $X_i \in \mathcal{X}$ has diameter at most $r \cdot \sigma$ and for each node $v \in V$, the neighbourhood $N_r(v)$ intersects with at most I sets in the partition; namely, $|\{X_i : X_i \in \mathcal{X} \wedge X_i \cap N_r(v) \neq \emptyset\}| \leq I$.

H consists of $h + 2$ levels, where $h = \lceil \log_{\rho} D \rceil$, where $\rho = 4\sigma$. Each level $l \geq 0$ of H, is a (r_l, σ, I)-partition $\mathcal{P}_l$ computed with $r_l = \min(D, \rho^l)$. In each set X_i in $\mathcal{P}_l$, an arbitrary node is designated as leader node $leader(X_i)$. Let $\mathcal{P}_{-1}$ be the trivial partition where each node of G is a cluster by itself. The maximum level is h, and at that level, the whole graph is a single cluster. For a leader ℓ at level $l < h$, the parent cluster (and respective parent leader) is the cluster at layer $\mathcal{P}_{l+1}$ that contains ℓ.

For graphs with a constant doubling dimension, the parameters ρ, σ, and I are all constant values [20]. For any subset S', let G' be the complete graph consisting of only the nodes in S', such that the edge weight between two nodes in G' is the same as the distance between them in G. Then, the order of the nodes for S' in the TSP tour of G gives a κ-factor approximation for the TSP tour of S' in G' (by connecting the last to the first node in S'), where $\kappa = O(\log n)$. In [20], the authors describe a method to transform the hierarchy H into to a κ-universal TSP tour for G. Figure 2 illustrates a partition hierarchy on a constant doubling dimension of graph G with $n = 7$ nodes where the left figure shows a partitioning scheme and the right figure shows leader nodes of each cluster and their parents (connected through a virtual link) in the graph G.

4 Global-Aware Distributed Scheduling

In this section, we provide two basic transaction scheduling algorithms, GlobalAware_SingleObj and GlobalAware_MultipleObjs, for graphs with a constant doubling dimension, assuming that the nodes have global knowledge of transactions and objects they access.

Algorithm 1: GlobalAware_SingleObj

Input : Graph G with (r, σ, I)-partition H (with $h+2$ levels), and a set of transactions $\mathcal{T}$

$v' \leftarrow$ initial home node for object o;
$\alpha, \beta \leftarrow$ cost of moving object o and a transaction over a unit weight edge of G, respectively;
Initialize $S \leftarrow \emptyset$ and $\gamma \leftarrow \lceil \frac{\alpha}{\beta} \rceil$;
for each *level* l *from* 0 *to* $h+1$ *in* H *with partition* $\mathcal{P}_l$ **do**
  for each *cluster* $X \in \mathcal{P}_l$ **do**
    $T(X) \leftarrow$ set of transactions contained by the nodes in X that have not been assigned a dedicated super-leader yet;
    if $|T(X)| \geq 2\gamma$ **then**
      Add $leader(X)$ to the set of super-leaders S;
      Assign $leader(X)$ as the dedicated super-leader for all the transactions in $T(X)$;
for each $T_i \in \mathcal{T}$, **if** T_i doesn't have a dedicated super-leader, **then** move T_i to v';
$S_f \subseteq S \leftarrow$ final set of dedicated super-leaders;
for each *level* $l \geq 0$ *in* H *with partition* $\mathcal{P}_l$ **do**
  Let $\widehat{L}_l$ be the set of dedicated super-leaders at level l;
  $\widehat{T}_l \leftarrow$ set of transactions contained by the dedicated super-leaders at level l;
  if $|\widehat{T}_l| < 8I\alpha$ **then**
    Move all the transactions in $\widehat{T}_l$ to v';
    Update $S_f \leftarrow S_f \setminus \widehat{L}_l$;
Move each transaction $T_i \in \mathcal{T}$ to its corresponding dedicated super-leader;
Calculate TSP tour on the set of dedicated super-leaders $S_f \cup \{v'\}$;
Object o traverses the ordered nodes of the TSP tour with transactions at the respective node being executed;

4.1 Single Object

We consider a single shared object o of size $\alpha > 1$ and a set of transactions $\mathcal{T} = \{T_1, T_2, \dots\}$ initially positioned at the nodes of G. All the transactions in $\mathcal{T}$ require object o for execution. We provide a basic scheduling algorithm denoted as GlobalAware_SingleObj for the execution of transactions accessing object o and its pseudocode is given as Algorithm 1.

A hierarchical partition H of a given graph G can be computed as described in Section 3.1. Now, our task is to find a set of special nodes in H (which we call *super-leaders*) to move object o for providing access to the transactions. The concept of super-leaders introduced here is derived from the notion of supernodes utilized in trees, as outlined in [4]. A leader node v of a cluster (set) X of a partition at some level $l \geq 0$ becomes a super-leader if the cluster contains sufficiently large number of transactions such that the cost of moving object o from v to any of the transaction nodes contained in X is less than the total cost of moving all the transactions contained in X to v. The idea is that the movement of an object incurs high communication cost compared to the movement of a transaction, thus if some cluster contains trivially less number of transactions, then moving an object to the leader of that cluster results in greater communication cost than transferring the transactions from that cluster to the next-level cluster. Overall, we attempt to reduce the length of object movement path at each level by computing the super-leaders. For each transaction $T_i \in \mathcal{T}$, we find a super-leader at which T_i is executed and we call it a *dedicated super-leader* for T_i. If a transaction does not have a dedicated super-leader, then it is moved and executed at the node where object o is initially located.

Let v' be the initial home node for object o in H. At each level $l \geq 0$ in H, if a cluster (set) X of the partition $\mathcal{P}_l$ at level l contains at least 2γ (where $\gamma = \lceil \frac{\alpha}{\beta} \rceil$) transactions that are not assigned a dedicated super-leader yet, then $leader(X)$ becomes a super-leader and each transaction $T_i \in X$ is assigned $leader(X)$ as a dedicated super-leader. Let S be the set of such super-leaders. Each super-leader $s \in S$ also maintains the set of transactions (nodes) $\widehat{T}(s)$ for which s is a dedicated super-leader. If some transactions do not have a dedicated super-leader yet, then those transactions are moved directly to v' for the execution.

Next, we prune the dedicated super-leaders. Let $S_f \subseteq S$ be the final set of dedicated super-leaders. Initially, each dedicated super-leader $s \in S$, where $|\widehat{T}(s)| \geq 2\gamma$, is added to S_f. At each level $l \geq 0$ in H, if the sum of total number of transactions contained by the dedicated super-leaders at level l is less than $8I\alpha$, then those transactions will be relocated to the object node for the execution and all the dedicated super-leaders at level l are removed from S_f. This threshold is needed in order to minimize the total communication cost as shown later in the proof of Lemma 4.3 that if the number of transactions is less than $8I\alpha$, then moving transactions to the node where the object is located is cheaper than moving the object to the transactions node.

In the final stage, each transaction is sent to its assigned dedicated super-leader in the set S_f. The goal is to move object o to each super-leader in S_f and execute the corresponding transactions there. This is achieved by computing a TSP tour starting from v' that visits all nodes in S_f. To compute the tour, we can use the Universal TSP approach based on the hierarchy H as in [20].

Figure 3 illustrates an execution of Algorithm 1. The figure in the left shows the set of dedicated super-leaders highlighted in orange color. Assuming that the object o is initially positioned at node v_1, the dashed line on the right figure traces the TSP tour for o moving from v_1 to the dedicated super-leaders executing the transactions in each node.

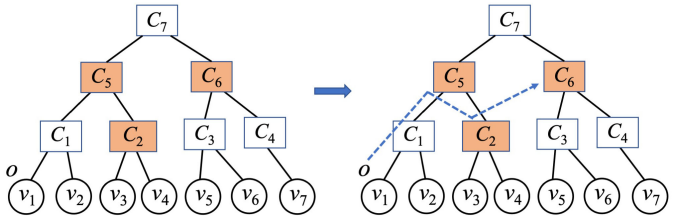

Fig. 3. Illustration of dedicated super-leaders and the TSP tour for moving object along the dedicated super-leaders in H by Algorithm 1.

Analysis. Without loss of generality, assume that $\alpha > 1$, and $\beta = 1$. Hence, $\gamma = \alpha/\beta = \alpha$, and the criterion of a leader becoming a super-leader is $2\gamma = 2\alpha$.

Each involved node in V holds at most one transaction. Let $\widehat{T}_i \subseteq \mathcal{T}$ denotes the set of transactions whose dedicated super-leader is at level i. Let $\widehat{L}_i$ denotes the set of dedicated leaders at level i (i.e., each $v \in \widehat{L}_i$ has a transaction that picked it as dedicated). Let $Tour^*(\widehat{L}_i)$ denotes the optimal length TSP tour for the nodes $\widehat{L}_i$, which also includes the original position of the object o. Let C^* be the optimal cost of executing all the transactions.

For a set of Z transactions, let $C_{\text{direct}}(Z)$ be the cost of a schedule that sends all Z transactions to the object's node (object does not move in this schedule).

Lemma 4.1. *For $Z > 0$ transactions, $C_{direct}(Z) \leq 4(|Z|/\alpha + \log_2 D)C^*$.*

Proof. Let u be the node with the object o. Let $Z_i \subseteq Z$, $i \geq 0$, be the set of transactions whose distance from u is in the range $[2^i, 2^{i+1})$. Let $C(Z_i)$ be the cost of moving the transactions Z_i to u. We have $C(Z_i) < |Z_i|2^{i+1}$.

In the best case scenario, all Z_i transactions are gathered at a single node v at distance 2^i from u. Ideally, the optimal schedule would move o in an intermediate node x between u and v (or x may coincide with u or v). Let d_1 be the distance between u and x and d_2 be the distance between x and v, where $d_1 + d_2 = 2^i$, and $d_1, d_2 \geq 0$. The cost of moving object o to x is $d_1\alpha$. The cost of moving the Z_i transactions to x is $d_2|Z_i|$. Thus, $C^* = d_1\alpha + d_2|Z_i|$. We examine two cases:

- $|Z_i| < \alpha$: then $C^* > d_1|Z_i| + d_2|Z_i| = 2^i|Z_i|$. Hence, $C(Z_i) < |Z_i|2^{i+1} < 2C^* < 2(|Z_i|/\alpha + 1)C^*$.
- $|Z_i| \geq \alpha$: then $C^* \geq d_1\alpha + d_2\alpha = 2^i\alpha$. Hence, $C(Z_i) < |Z_i|2^{i+1} = |Z_i|2^{i+1}\alpha/\alpha \leq 2|Z_i|C^*/\alpha < 2(|Z_i|/\alpha + 1)C^*$.

Hence, $C(Z_i) \leq 2(|Z_i|/\alpha + 1)C^*$. Since $0 \leq i \leq \lceil \log_2 D \rceil$, and $\sum_{i=0}^{\lceil \log_2 D \rceil} |Z_i| = |Z|$, we have:

$$C_{\text{direct}}(Z) \leq \sum_{i=0}^{\lceil \log_2 D \rceil} C(Z_i) \leq \sum_{i=0}^{\lceil \log_2 D \rceil} 2(|Z_i|/\alpha + 1)C^* = 2C^* \sum_{i=0}^{\lceil \log_2 D \rceil} (|Z_i|/\alpha + 1)$$

$$= 2C^*(|Z|/\alpha + 1 + \lceil \log_2 D \rceil) \leq 4(|Z|/\alpha + \log_2 D)C^*. \qquad \square \qquad \square$$

Let q be the path of the object in an optimal schedule for all the transactions in $\mathcal{T}$ which results in the optimal schedule with cost C^*. We can recursively decompose q to a sequence of edge-disjoint subpaths $q_1, q_2, \ldots, q_k$, $k \geq 1$, with parameter ξ as follows. Let q_1 be the shortest prefix subpath of q such that its length $|q_1|$ is at least $|q_1| \geq \xi$. If such q_1 does not exist, then $q_1 = q$. Let q' be the remaining subpath of q where the first node of q' is the same with the last node of q_1. If q' consists of at least two nodes, repeat the same process recursively on q'. We have the following properties for optimal path q:

- The first and last node of q is the first of q_1, and the last of q_k, respectively.
- The first node of q_j is the same with the last node of q_{j-1}, where $j > 1$.
- For $1 \leq j < k$, each q_j has a length at least $|q_j| \geq \xi$ such that the removal of the last edge splits q_j into prefix q'_j and suffix q''_j with $|q'_j| < \xi$.
- For the last subpath q_k, if it consists of three or more nodes, then for its prefix q'_j, it holds $|q'_j| < \xi$; otherwise, $|q_k| > 0$.
- $k \leq |q|/\xi + 1$: Since each of the first $k-1$ segments has length at least ξ, we have $|q| \geq (k-1)\xi$. Therefore, $k - 1 \leq |q|/\xi$, which implies $k \leq |q|/\xi + 1$.

Lemma 4.2. *For a path q_j, $1 \leq j \leq k$, with v_1 and v_2 being the first and last nodes of q_j, any $c \geq 0$, $N_c(q_j) \subseteq N_{2c}(\{v_1, v_2\})$.*

Proof. This holds immediately if q_j consists only of v_1 and v_2, since $N_c(q_j) = N_c(\{v_1 \cup v_2\}) \subseteq N_{2c}(\{v_1, v_2\})$. Suppose now that q_j consists of three or more nodes. Let v' be the node of q_j which is adjacent to v_2. We have that $dist_G(v_1, v') \leq c$. Thus, all the nodes of q_j between v_1 and v' are at distance less than ξ from v_1; hence, the ξ-neighbors of all these nodes are at a distance less than 2ξ from v_1. Thus, $N_c(q_j) \subseteq N_{2c}(v_1) \cup N_c(v_2) \subseteq N_{2c}(\{v_1, v_2\})$. $\square$ $\square$

Lemma 4.3. *For $|\widehat{T}_i| \geq 8I\alpha$, $C^* \geq |\widehat{T}_i|\rho^{i-1}/(16I)$.*

The proof of Lemma 4.3 is omitted due to space constraints. $\square$

Lemma 4.4. *The number of ρ^i-neighborhoods that are required to cover a $8\sigma\rho^i$-neighborhood of G is at most $\zeta := 2^{\delta \log(8\sigma)}$.*

Proof. Let G be a graph with doubling dimension δ. Consider a ball of radius $8\sigma\rho^i$ centered at a vertex v in G, denoted as $B(v, 8\sigma\rho^i)$. We want to cover this ball with ρ^i-neighborhoods. From the definition, any ball of radius $2r$ can be covered by at most 2^δ balls of radius r. Here, $r = 4\sigma\rho^i$. Therefore, the ball $B(v, 8\sigma\rho^i)$ can be covered by at most 2^δ balls of radius $4\sigma\rho^i$. Similarly, to cover $4\sigma\rho^i$, we need 2^δ balls of radius $2\sigma\rho^i$ and so on. We can estimate the total number of balls as $\prod_{i=1}^{k} 2^\delta = (2^\delta)^k$, where $\frac{8\sigma\rho^i}{2^k} = \rho^i$. Thus, $k = \log(8\sigma)$. Hence, $2^{\delta k} = 2^{\delta \log(8\sigma)}$. $\square$ $\square$

Lemma 4.5. $\widehat{L}_i \subseteq N_{4\sigma\rho^i}(q)$.

Proof. Suppose that there is $v \in \widehat{L}_i$, such that $v \notin N_{4\sigma\rho^i}(q)$. Let X be the cluster of v at partition $\mathcal{P}_i$. Let d be the distance of v to the closest node in q; note that $d > 4\sigma\rho^i$. The distance between any node in X to its closest node in q is at least $d - \sigma\rho^i$. Let $T(X)$ be the transactions with home node in X. Since v is a super-leader, $|T(X)| \geq 2\alpha$. The cost c_1 of moving these transactions to q is

$$c_1 \geq (d - \sigma\rho^i)2\alpha = (2d - 2\sigma\rho^i)\alpha > (d + 4\sigma\rho^i - 2\sigma\rho^i)\alpha = (d + 2\sigma\rho^i)\alpha\ .$$

On the other hand, the cost c_2 of gathering the $T(X)$ transactions to v and then having the object move from q to v is $c_2 \leq 2\alpha\sigma\rho^i + d\alpha < c_1$. Therefore, the path q is not optimal; a contradiction. Thus, $\widehat{L}_i \subseteq N_{4\sigma\rho^i}(q)$. $\square$ $\square$

Using Lemmas 4.2, 4.4, and 4.5, we obtain the following two results:

Lemma 4.6. *For $|\widehat{T}_i| \geq 4I\alpha$ and $|q| \geq \rho^{i-1}/2$, $Tour^*(\widehat{L}_i) \leq 74\zeta I\sigma\rho|q|$.* □

Lemma 4.7. *For $|\widehat{T}_i| \geq 4I\alpha$ and $|q| < \rho^{i-1}/2$, $Tour^*(\widehat{L}_i) \leq 36C^*\rho\zeta\sigma/\alpha$.* □

Let R be the set of indices in range $0, \ldots, h$ such that for each $i \in R$, $|\widehat{T}_i| \geq 8I\alpha$ and $\widehat{T} = \bigcup_{i\in R} \widehat{T}_i$. We have a set of dedicated super-leaders $S_f = \bigcup_{i\in R} \widehat{L}_i$. Let $Tour^*(S_f)$ denote the cost of the optimal TSP tour covering all the nodes in S_f, including the object's initial position, and $Tour(S_f)$ be the actual cost of the object tour incurred by Algorithm 1. (Without loss of generality, assume that $Tour^*(S_f) > 0$, since otherwise we would only have the direct move cost for the transactions.) We establish the following three Lemmas (proofs are omitted for brevity).

Lemma 4.8. $\alpha \cdot Tour^*(S_f) \leq 74(h+1)\zeta I\sigma\rho C^*$. □

Lemma 4.9. *For transactions $\widehat{T}$, cost of Algorithm 1 is $C(\widehat{T}) \leq 74\frac{Tour(S_f)}{Tour^*(S_f)}(h+1)\zeta I\sigma\rho C^*$.* □

Lemma 4.10. *For transactions $\mathcal{T} \setminus \widehat{T}$, cost of Algorithm 1 is $C_{direct}(\mathcal{T} \setminus \widehat{T}) \leq 36(h+2)IC^* + 4C^* \log_2 D$.* □

Theorem 4.11. *For executing all transactions in $\mathcal{T}$, the total communication cost of Algorithm 1 is $C = O\left(C^* \cdot \frac{Tour(S_f)}{Tour^*(S_f)} \cdot \log D\right)$.*

Proof. For the cost C of Algorithm 1 we have, $C = C(\widehat{T}) + C_{\text{direct}}(\mathcal{T} \setminus \widehat{T})$.
Thus, from Lemmas 4.9 and 4.10, we get:

$$C \leq 74\frac{Tour(S_f)}{Tour^*(S_f)}(h+1)\zeta I\sigma\rho C^* + 36(h+2)IC^* + 4C^* \log_2 D.$$

The result follows since $h = O(\log D)$, and σ, I, ρ, ζ are constants (see Lemma 4.4 for ζ being a constant). □ □

When using a TSP tour based on the minimum weight spanning tree, we get $Tour(S_f)/Tour^*(S_f) \leq 2$. Thus, from Theorem 4.11, we get the following result.

Corollary 4.12. *Using a MST based TSP to implement the tour of the object, the approximation of Algorithm 1 is $O(\log D)$.* □

Using a universal TSP tour based on H, we get $Tour(S_f)/Tour^*(S_f) = O(\log n)$. Thus, from Theorem 4.11, we get the following result.

Corollary 4.13. *Using a universal TSP to implement the tour of the object, the approximation of Algorithm 1 is $O(\log n \cdot \log D)$.* □

4.2 Multiple Objects

We consider a set of shared objects $\mathcal{O} = \{o_1, o_2, \ldots\}$, all initially positioned at some node v' of G. Each transaction $T_i \in \mathcal{T}$ accesses at most k objects from $\mathcal{O}$. The set of objects accessed by transaction T_i is denoted as $objs(T_i) \subseteq \mathcal{O}$. We assume that nodes have global knowledge of transactions and provide a scheduling algorithm GlobalAware_MultipleObjs for executing the transactions in $\mathcal{T}$. The pseudocode is provided in Algorithm 2.

In Algorithm 2, the objective is to ensure synchronized access to the objects at minimal cost for executing the transactions. We accomplish this by first using Algorithm 1 to calculate the dedicated super-leaders for each object individually. We then combine some of the dedicated super-leaders of the objects in a set of nodes S_f. Finally, we use a single TSP tour over $S_f \cup \{v'\}$, where the objects move synchronously from node to node along the tour, executing the corresponding transactions at each visited node. Note that an object visits only the nodes in the TSP tour where it is needed for a transaction, skipping all others.

Algorithm 2: GlobalAware_MultipleObjs

Input : Graph $G = (V, E, w)$ with (r, σ, I)-partition H, and a set of transactions $\mathcal{T}$

$v' \leftarrow$ initial home node for all objects in O; $S_f \leftarrow \emptyset$;
// Find super-leaders w.r.t. each object
for each *object* $o_j \in O$ **do**
    Use Algorithm 1 to find the dedicated super-leaders for object o_j;
// Find dedicated super-leader for T_i
for each *transaction* $T_i \in \mathcal{T}$ **do**
    if T_i *has at least one dedicated super-leader assigned above for any of the objects in* $objs(T_i)$ **then**
        Find the closest super-leader s in H among all the dedicated super-leaders assigned to T_i and mark it as the dedicated super-leader for T_i;
        Move T_i to s and add s to S_f;
    else move T_i to the initial home node of objects, v';
// Build and traverse TSP tour
Calculate TSP tour for $S_f \cup \{v'\}$;
Each object $o_j \in O$ follows the order of the TSP tour for visiting the nodes where o_j is required to execute respective transactions;

The set of dedicated super-leaders S_f is determined as follows. Each transaction $T_i \in \mathcal{T}$ accesses up to k objects, potentially with different dedicated super-leaders for each such object in $objs(T_i)$. For each T_i, we select the nearest super-leader among those assigned to its objects, and designate it as the T_i's dedicated super-leader s. Transaction T_i moves to s, and s is added to S_f. During the TSP traversal of S_f, all objects in $objs(T_i)$ will be brought to the leader node s, and T_i is executed. If T_i has no dedicated super-leader (i.e., none of its objects do), then it moves to v' and is executed when v' is visited in the tour.

Analysis. We continue with the analysis of Algorithm 2. Consider an object $o_j \in O$. If a super-leader s' of o_j, as returned by Algorithm 1, is not used in the set S_f calculated by Algorithm 2, then another super-leader is being used in S_f which is closer to the transactions that were going to move to s'. Thus, since $\alpha > \beta$, the cost of such a change of super-leader does not incur an additional cost with respect to object o_j (where cost of o_j is as calculated in Theorem 4.11).

Now consider the case where object o_j is accessed by a transaction T_i, originally located at some node u, and T_i does not have a dedicated super-leader for o_j under Algorithm 1, but does have a dedicated super-leader s under Algorithm 2. If the distance from u to s is no greater than the distance from u to v', then the cost of accessing o_j remains unaffected (as calculated in Theorem 4.11).

If the distance from u to s is more than the distance from u to v', then there is additional cost due to the transaction T_i traveling a longer distance in Algorithm 2. However, since T_i has s as a dedicated super-leader, there must exist some object $o_z \in O$ which is accessed by T_i for which s is a dedicated super-leader for T_i. The additional cost with respect to object o_j is no more than the cost C' of moving T_i to s, as counted in the analysis of object o_z (in Theorem 4.11). Since T_i can access at most $k-1$ other objects similar to o_j, the additional cost for moving these objects to s is at most $(k-1)C'$.

Hence, using S_f increases the total cost by at most a factor of k compared to the sum of the costs of individual objects. As a result, the approximation of Theorem 4.11 scales by a factor of k, giving the following theorem and corollaries:

Theorem 4.14. *For executing all transactions in $\mathcal{T}$, the total communication cost of Algorithm 2 is* $C = O\left(k \cdot C^* \cdot \frac{Tour(S_f)}{Tour^*(S_f)} \cdot \log D\right)$. □

Corollary 4.15. *Using a MST based TSP to implement the tour of the object, the approximation of Algorithm 2 is* $O(k \cdot \log D)$. □

Corollary 4.16. *Using a universal TSP to implement the tour of the object, the approximation of Algorithm 2 is* $O(k \cdot \log n \cdot \log D)$. □

5 Fully Distributed Scheduling

In this Section, we provide fully distributed transaction scheduling algorithms assuming that the nodes in G do not have the global knowledge of transactions. First, we discuss the algorithm in detail for the case where transactions access a single shared object. Later, we provide a high-level idea for extending the algorithm for multiple shared objects.

5.1 Single Object

As in Section 4.1, we consider a single shared object o of size $\alpha > 1$ and a set of transactions $\mathcal{T} = \{T_1, T_2, \dots\}$ initially positioned at arbitrary nodes of G. Each transaction $T_i \in \mathcal{T}$ requires access to object o for the execution. We provide a fully distributed scheduling algorithm FullyDistributed_SingleObj for the execution of transactions in $\mathcal{T}$ accessing the object o. The algorithm operates in three phases: (1) Cluster leaders share transaction information with their parent leaders and identify super-leaders; (2) Each transaction selects its dedicated super-leader and moves to that node for execution; (3) A TSP tour is computed to move the object to each dedicated super-leader for transaction execution. We now describe each phase in detail.

Phase 1: In this phase, transaction information is propagated through the levels of the hierarchy H to identify the super-leaders. The process begins with each node simultaneously sharing its transaction data with the leader of the corresponding cluster at level $l = 0$ in H. At each level $l \geq 0$, the leader of each cluster (set) X ($leader(X)$) in the partition $\mathcal{P}_l$ counts the transactions within its cluster. If the count reaches at least 2γ, the cluster leader becomes a super-leader and notifies all child leaders recursively down to the level 0. It also forwards the transaction count to the parent cluster at level $l + 1$, continuing this process up to the root of the hierarchy H. At the end of Phase 1, a set S of all possible super-leaders is computed, and each cluster leader has the knowledge of S.

Phase 2: In this phase, each node independently determines the dedicated super-leader for its associated transaction $T_i \in \mathcal{T}$. Let X be the lowest-level cluster in H that contains T_i, and S be the set of super-leaders whose information was received by $leader(X)$ from its parent clusters. Then, T_i selects a dedicated super-leader $s \in S$ such that s is the closest super-leader to T_i among all in S. If S is empty (i.e., $leader(X)$ didn't receive any super-leader information from its parents), then T_i is moved to the node where the object o is initially located.

The set of dedicated super-leaders is further pruned as follows. At each level $l \geq 0$ in H, the algorithm checks the total number of transactions across all dedicated super-leaders. If this total is less than $8I\alpha$, then the associated transactions are moved to the initial home node of object o and the corresponding dedicated super-leaders are unmarked. To compute the total at level l, a reference leader node s_z is chosen at the level l, and all dedicated super-leaders at that level send their transaction counts to s_z. Consequently, s_z sums all the counts of the number of transactions it received and sends the calculated sum to all the dedicated super-leaders of that level.

Phase 3: In this phase, the algorithm determines the schedule to move the object o to each dedicated super-leader and executes the transactions. This can be done in two ways: by using a universal TSP tour or an MST-based tour. Pseudocode and descriptions of the approaches are omitted due to space constraints.

Analysis. In the fully distributed algorithm FullyDistributed_SingleObj, each node initially knows only its own transactions. The key challenge is to compute the set of super-leaders, which is handled in Phase 1 of the algorithm. After that, the algorithm proceeds similarly to Algorithm 1. Thus, any additional communication cost comes only from Phase 1 of the distributed algorithm. Nevertheless, we show that FullyDistributed_SingleObj also achieves the same approximation as of GlobalAware_SingleObj (i.e., Algorithm 1).

Let C be the total communication cost of Algorithm 1 that involves the movement of transactions from current node to the respective dedicated super-leaders (say C_1) and the movement of object between the dedicated super-leaders in H (say C_2). That means, $C = C_1 + C_2$.

Let C' be the total communication cost of FullyDistributed_SingleObj, which involves three phases. Let C'_{p_1}, C'_{p_2}, and C'_{p_3} be the communication cost of Phase 1, Phase 2, and Phase 3, respectively. Then, $C' = C'_{p_1} + C'_{p_2} + C'_{p_3}$. Since the execution of Phase 2 and Phase 3 of the algorithm is equivalent to Algorithm 1, we have, $C'_{p_2} + C'_{p_3} = C$. That means, $C' = C'_{p_1} + C$.

Phase 1 runs only once throughout the execution. Three types of messages are exchanged between different nodes of H during the execution of Phase 1.

i. Each node sends its transaction information to its cluster leader at level $l = 0$ in H. This cost is at most C_1.
ii. Each cluster leader at level $l \geq 0$ sends the count of total transactions in its cluster to the leader of parent cluster at level $l + 1$. This cost is at most C_2.
iii. When a cluster leader becomes a super-leader, it recursively notifies all child leaders down to the level 0. This cost is also at most C_2.

Hence, $C'_{p_1} \leq C_1 + 2C_2 < 2C$, which implies, $C' < 2C + C < 3C$. Therefore, the following theorem is immediate:

Theorem 5.1. *Algorithm 2 achieves $O(\log n \cdot \log D)$–approximation in communication cost when using universal TSP tour, and $O(\log D)$–approximation when using universal TSP tour of the object.*

5.2 Multiple Objects

In this section, we briefly describe how the fully distributed algorithm for single object can be extended to handle the transactions accessing multiple shared objects. Consider a set of shared objects $\mathcal{O} = \{o_1, o_2, \ldots\}$, and each transaction $T_i \in \mathcal{T}$ accesses at most k objects from $\mathcal{O}$. As in FullyDistributed_SingleObj, transaction information is first shared through the levels of the hierarchy H. Then, we calculate the dedicated super-leaders with respect to each object by running FullyDistributed_SingleObj. After that, the set of dedicated super-leaders for each transaction $T_i \in \mathcal{T}$ are determined by combining the object-specific super-leaders as in Algorithm 2. Finally, we calculate the TSP tour on the dedicated super-leaders. Each object $o_j \in \mathcal{O}$ visits the respective dedicated super-leader following the TSP tour and transactions are executed at each dedicated super-leader when the required objects gather.

6 Conclusions

In this paper, we studied transaction scheduling problem in distributed systems modeled as constant doubling dimension graphs. We used a dual-flow transaction execution model in which both objects and transactions can move within the network to minimize communication cost. We proposed centralized and fully distributed versions of the algorithm for transaction scheduling that achieve polylogarithmic approximations in communication cost. In the future, it would be interesting to extend the algorithms and analysis for general and arbitrary graphs. Experimental evaluation of the designed algorithms against different application benchmarks in a practical setting would also be interesting.

Acknowledgments

This paper is supported by NSF grant CNS-2131538.

References

[1] Ittai Abraham, Cyril Gavoille, Andrew V Goldberg, and Dahlia Malkhi. 2006. Routing in networks with low doubling dimension. In *26th IEEE International Conference on Distributed Computing Systems (ICDCS'06)*. IEEE, 75–75.

[2] Ramesh Adhikari and Costas Busch. 2023. Lockless blockchain sharding with multiversion control. In *International Colloquium on Structural Information and Communication Complexity*. Springer, 112–131.

[3] Ramesh Adhikari, Costas Busch, and Dariusz R Kowalski. 2024. Stable Blockchain Sharding under Adversarial Transaction Generation. In *Proceedings of the 36th ACM Symposium on Parallelism in Algorithms and Architectures*. 451–461.

[4] Costas Busch, Bogdan S Chlebus, Maurice Herlihy, Miroslav Popovic, Pavan Poudel, and Gokarna Sharma. 2023. Flexible scheduling of transactional memory on trees. *Theoretical Computer Science* 978 (2023), 114184.

[5] Costas Busch, Bogdan S Chlebus, Dariusz R Kowalski, and Pavan Poudel. 2023. Stable Scheduling in Transactional Memory. In *International Conference on Algorithms and Complexity*. Springer, 172–186.

[6] Costas Busch, Maurice Herlihy, Miroslav Popovic, and Gokarna Sharma. 2015. Impossibility results for distributed transactional memory. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*. 207–215.

[7] Costas Busch, Maurice Herlihy, Miroslav Popovic, and Gokarna Sharma. 2017. Fast scheduling in distributed transactional memory. In *Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures*. 173–182.

[8] Costas Busch, Maurice Herlihy, Miroslav Popovic, and Gokarna Sharma. 2022. Dynamic scheduling in distributed transactional memory. *Distributed Computing* 35, 1 (2022), 19–36.

[9] Yang Cao, Wenfei Fan, Weijie Ou, Rui Xie, and Wenyue Zhao. 2023. Transaction Scheduling: From Conflicts to Runtime Conflicts. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–26.

[10] Matteo Ceccarello, Andrea Pietracaprina, Geppino Pucci, and Eli Upfal. 2017. MapReduce and streaming algorithms for diversity maximization in metric spaces of bounded doubling dimension. *Proceedings of the VLDB Endowment* 10, 5 (2017), 469–480.

[11] T-H Hubert Chan, Anupam Gupta, Bruce M Maggs, and Shuheng Zhou. 2016. On hierarchical routing in doubling metrics. *ACM Transactions on Algorithms (TALG)* 12, 4 (2016), 1–22.

[12] Jie Gao, Leonidas Guibas, Nikola Milosavljevic, and Dengpan Zhou. 2009. Distributed resource management and matching in sensor networks. In *2009 International Conference on Information Processing in Sensor Networks*. 97–108.

[13] Mohammad Goudarzi, Marimuthu Palaniswami, and Rajkumar Buyya. 2022. Scheduling IoT applications in edge and fog computing environments: A taxonomy and future directions. *Comput. Surveys* 55, 7 (2022), 1–41.

[14] Vincent Gramoli, Zhenliang Lu, Qiang Tang, and Pouriya Zarbafian. 2024. AOAB: optimal and fair ordering of financial transactions. In *2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 377–388.

[15] Anupam Gupta, Robert Krauthgamer, and James R. Lee. 2003. Bounded Geometries, Fractals, and Low-Distortion Embeddings. In *44th Symposium on Foundations of Computer Science, FOCS 2003, Cambridge, MA, USA, October 11-14, 2003, Proceedings*. IEEE Computer Society, 534–543.

[16] Danny Hendler, Alex Naiman, Sebastiano Peluso, Francesco Quaglia, Paolo Romano, and Adi Suissa. 2013. Exploiting Locality in Lease-Based Replicated Transactional Memory via Task Migration. In *DISC*. 121–133.

[17] Maurice Herlihy and J. Eliot B. Moss. 1993. Transactional Memory: Architectural Support for Lock-free Data Structures. In *ISCA* (San Diego, California, USA). 289–300.

[18] Maurice Herlihy and Ye Sun. 2007. Distributed transactional memory for metric-space networks. *Distributed Computing* 20 (2007), 195–208.

[19] Aditya Jayaprakash and Mohammad R Salavatipour. 2023. Approximation schemes for capacitated vehicle routing on graphs of bounded treewidth, bounded doubling, or highway dimension. *ACM Transactions on Algorithms* 19, 2 (2023), 1–36.

[20] Lujun Jia, Guolong Lin, Guevara Noubir, Rajmohan Rajaraman, and Ravi Sundaram. 2005. Universal approximations for TSP, Steiner tree, and set cover. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*. 386–395.

[21] Youyou Kang, Li Pan, and Shijun Liu. 2022. Job scheduling for big data analytical applications in clouds: A taxonomy study. *Future Generation Computer Systems* 135 (2022), 129–145.

[22] Abdelhamid Khiat, Mohamed Haddadi, and Nacera Bahnes. 2024. Genetic-Based Algorithm for Task Scheduling in Fog–Cloud Environment. *Journal of Network and Systems Management* 32, 1 (2024), 3.

[23] Junwhan Kim and Binoy Ravindran. 2010. On transactional scheduling in distributed transactional memory systems. In *Symposium on Self-Stabilizing Systems*. Springer, 347–361.

[24] Naoki Kitamura, Hirotaka Kitagawa, Yota Otachi, and Taisuke Izumi. 2021. Low-congestion shortcut and graph parameters. *Distributed Computing* 34, 5 (2021), 349–365.

[25] Goran Konjevod, Andréa W Richa, and Donglin Xia. 2008. Dynamic routing and location services in metrics of low doubling dimension. In *Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*. 417–417.

[26] Fabian Kuhn, Thomas Moscibroda, and Rogert Wattenhofer. 2005. On the locality of bounded growth. In *Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*. 60–68.

[27] Marwa Mokni, Sonia Yassa, Jalel Eddine Hajlaoui, Mohamed Nazih Omri, and Rachid Chelouah. 2023. Multi-objective fuzzy approach to scheduling and offloading workflow tasks in Fog–Cloud computing. *Simulation Modelling Practice and Theory* 123 (2023), 102687.

[28] Wolfgang Mulzer and Max Willert. 2020. Compact routing in unit disk graphs. In *31st International Symposium on Algorithms and Computation (ISAAC 2020)*. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 16–1.

[29] Husnu S Narman, Md Shohrab Hossain, Mohammed Atiquzzaman, and Haiying Shen. 2017. Scheduling internet of things applications in cloud computing. *Annals of Telecommunications* 72, 1 (2017), 79–93.

[30] Tina Samizadeh Nikoui, Ali Balador, Amir masoud Rahmani, and Zeinab Bakhshi. 2020. Cost-Aware Task Scheduling in Fog-Cloud Environment. *2020 CSI/CPSSI International Symposium on Real-Time and Embedded Systems and Technologies (RTEST)* (2020), 1–8.

[31] Tina Samizadeh Nikoui, Sam Jabbehdari, and Alireza Bagheri. 2016. Providing a cloud broker-based approach to improve the energy consumption and achieve a green cloud computing. *International Journal of Computer Applications* 138, 1 (2016), 42–49.

[32] Maycon Leone Maciel Peixoto, Thiago AL Genez, and Luiz F Bittencourt. 2021. Hierarchical scheduling mechanisms in multi-level fog computing. *IEEE Transactions on services computing* 15, 5 (2021), 2824–2837.

[33] Pavan Poudel, Shishir Rai, and Swapnil Guragain. 2024. Ordered scheduling in control-flow distributed transactional memory. *Theor. Comput. Sci.* 993 (2024), 114463. doi:10.1016/J.TCS.2024.114463

[34] Pavan Poudel, Shishir Rai, and Gokarna Sharma. 2021. Processing Distributed Transactions in a Predefined Order. In *ICDCN '21: International Conference on Distributed Computing and Networking, Virtual Event, Nara, Japan, January 5-8, 2021*. ACM, 215–224. doi:10.1145/3427796.3427819

[35] Pavan Poudel and Gokarna Sharma. 2020. GraphTM: An Efficient Framework for Supporting Transactional Memory in a Distributed Environment. In *ICDCN*. 11:1–11:10.

[36] Mohamed M. Saad and Binoy Ravindran. 2011. Snake: Control Flow Distributed Software Transactional Memory. In *Stabilization, Safety, and Security of Distributed Systems*, Xavier Défago, Franck Petit, and Vincent Villain (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 238–252.

[37] Gokarna Sharma and Costas Busch. 2014. Distributed transactional memory for general networks. *Distributed Computing* 27, 5 (01 Oct 2014), 329–362. doi:10.1007/s00446-014-0214-7

[38] Gokarna Sharma and Costas Busch. 2015. A load balanced directory for distributed shared memory objects. *J. Parallel and Distrib. Comput.* 78 (2015), 6–24.

[39] Nir Shavit and Dan Touitou. 1995. Software transactional memory. In *Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing*. 204–213.

[40] Srivathsan Srinivasagopalan, Costas Busch, and SS Iyengar. 2011. An oblivious spanning tree for single-sink buy-at-bulk in low doubling-dimension graphs. *IEEE Trans. Comput.* 61, 5 (2011), 700–712.

[41] Eli Tilevich and Yannis Smaragdakis. 2002. J-Orchestra: Automatic Java Application Partitioning. In *ECOOP 2002 — Object-Oriented Programming*, Boris Magnusson (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 178–204.

[42] Hoa Tran-Dang and Dong-Seong Kim. 2023. DISCO: Distributed computation offloading framework for fog computing networks. *Journal of Communications and Networks* 25, 1 (2023), 121–131.

[43] Takayuki Usui, Reimer Behrends, Jacob Evans, and Yannis Smaragdakis. 2010. Adaptive locks: Combining transactions and locks for efficient concurrency. *J. Parallel and Distrib. Comput.* 70, 10 (2010), 1009–1023.

[44] Qian Zhang, Jingyao Li, Hongyao Zhao, Quanqing Xu, Wei Lu, Jinliang Xiao, Fusheng Han, Chuanhui Yang, and Xiaoyong Du. 2023. Efficient Distributed Transaction Processing in Heterogeneous Networks. *Proceedings of the VLDB Endowment* 16, 6 (2023), 1372–1385.