Flexible Scheduling of Transactional Memory on Trees[l]

Costas Busch[a], Bogdan S. Chlebus[a], Maurice Herlihy[b], Miroslav Popovic[c], Pavan Poudel[d,], Gokarna Sharma[e]

[a]Augusta University, Augusta, Georgia, USA
[b]Brown University, Providence, Rhode Island, USA
[c]University of Novi Sad, Novi Sad, Serbia
[d]ATGWORK, Norcross, Georgia, USA
[e]Kent State University, Kent, Ohio, USA

Abstract

We study the eciency of executing transactions in a distributed transactional mem-ory system. The system is modeled as a static network with the topology of a tree. Contrary to previous approaches, we allow the flexibility for both transactions and their requested objects to move simultaneously among the nodes in the tree. Given a batch of transactions and shared objects, the goal is to produce a schedule of executing the transactions that minimizes the cost of moving the transactions and the objects in the tree. We consider both techniques for accessing a remote object with respect to a transaction movement. In the first technique, instead of moving, transactions send control messages to remote nodes where the requested objects are gathered. In the second technique, the transactions migrate to the remote nodes where the objects are gathered to access them. When all the transactions use a single object, we give an oine algorithm that produces optimal schedules for both techniques. For the general case of multiple objects per transaction, in the first technique, we obtain a schedule with a constant-factor approximation of optimal. In the second technique, with transactions migrating, we give a k factor approximation where k is the maximum number of objects per transaction.

Keywords: Distributed system, transactional memory, shared object, network, communication cost

[l] A preliminary version of this article appears in the Proceedings of SSS'22 [1].

Corresponding author. Tel.: +1 800 551 7943

Email addresses: kbusch@augusta.edu (Costas Busch), bchlebus@augusta.edu (Bogdan S. Chlebus), herlihy@cs.brown.edu (Maurice Herlihy), miroslav.popovic@rt-rk.uns.ac.rs (Miroslav Popovic), poudelpavan@gmail.com (Pavan Poudel), gsharma2@kent.edu (Gokarna Sharma)

1. Introduction

Threads executed concurrently require synchronization to prevent inconsistencies while accessing shared objects. Traditional low-level thread synchronization mechanisms such as locks and barriers are prone to deadlock and priority inversion, among multiple vulnerabilities. The concept of transactional memory has emerged as a high-level abstraction of the functionality of distributed systems; see Herlihy and Moss [2] and Shavit and Touitou [3]. The idea is to designate blocks of program code as transactions to be executed atomically. Several commercial processors support transactional memory in hardware, for example, Haswell of Intel [4], Blue Gene/Q of IBM [5], zEnterprise EC12 of IBM [6], and Power8 of IBM [7].

Transactions are executed speculatively, in the sense that if a transaction aborts due to synchronization conflicts or failures then the transaction's execution is rolled back to be restarted later. A transaction commits if there are no conflicts or failures, and its eects become visible to all processes. If multiple transactions concurrently attempt to access the same object, then this creates a conflict for access and could trigger aborting some of the involved transactions. Scheduling transactions to minimize conflicts for access to shared objects improves the system's performance.

The processing units of a distributed transactional memory system are the nodes of a communication network, which is an integral part of the system. A transaction executing at a node may want to access shared memory objects residing in other nodes. This could be implemented such that the transaction coordinates access to the needed shared objects with the nodes hosting the objects. Such systems were studied by Herlihy and Sun [8], Sharma and Busch [9], and Siek and Wojciechowski [10]. The eciency of executing a specific transaction may reflect the topology of the communication network that is part of a distributed system. For example, the amount of communication needed to execute a transaction interacting with some objects could be proportional to the distances in the network between all the nodes hosting the transaction and the objects.

To improve eciency of processing transactions on shared objects, we may preemptively move objects and transactions among the nodes to schedule their presence at specific nodes at specific times. Moving transactions or program code among network nodes is currently used in several real-world applications. For example, Erlang Open Telecom Platform aids dynamic code upgrade by supporting transactional servers with hot code swapping whose call-back modules may be changed on the fly [11]. A job management system for a computer cluster may migrate a job to a dierent node, if the target node's load is below the migration threshold and the migration overhead is acceptable, in order to achieve better load balancing among the nodes, see Hwang et al. [12]. A related system that uses live virtual machine migration to support autonomic adaptation of virtual computation environments is described by Ruth et al. [13].

Coordinating accessing objects to execute transactions may involve relocation of objects or transactions. Eciency of such coordination may depend on additional model's specification which determines the very feasibility of moving transactions and objects across the network. In the data-flow model, transactions are static and objects move from one node to another to reach the nodes hosting transactions that require interacting with them; see Tilevich and Smaragdakis [14] and Herlihy and Sun [8]. In that model, a transaction initially requests the objects it needs, and executes after assembling them. After a transaction commits, it releases its objects, possibly forwarding them to pending transactions. In the control-flow model, objects are static and transactions move from one node to another to access the objects. Control-flow allows transactions to send control requests, in a manner similar to remote procedure calls, to the nodes where the required objects are located; see Arnold et al. [15] and Saad and Ravindran [16].

1.1. Contributions

We consider a flexible scheduling approach that combines the benefits of the data-flow and control-flow models. We study the dual-flow model that allows for both transactions and objects to move among the nodes to synchronize transactions and objects. This model combines the functionality of the data-flow and control-flow models. Assessing the cost of synchronizing transactions with objects in the dual-flow model takes into account the communication cost of relocating objects, and the communication cost of relocating transactions or the cost of sending control messages from nodes hosting transactions to nodes hosting objects.

We consider distributed systems whose networks interpreted as graphs have static tree topologies. This represents many real-world networks. For example, the internet cloud consists of the cloud network, representing a root, the fog network gateways and/or the edge network gateways, as internal nodes, and the IoT devices as leaves, see Comer [17].

We study the eciency of executing transactions by a distributed system represented as a tree in the dual-flow model. The eciency is measured by the cost of communication. Scheduling transactions is considered in a batch setting, in which all the transactions are given at the outset, subject to the constraint that each node is assigned at most one original transaction. The initial position of shared objects are distributed arbitrarily among the nodes. We consider scheduling transactions in the general case of arbitrarily many shared objects, and also in a special case of a single shared object that needs to be accessed by all the transactions. Given a batch of transactions and objects residing at nodes of the system, the goal is to produce a schedule of executing transactions that minimizes the cost of moving transactions and objects among the nodes and sending control messages to facilitate executing the transactions. Such a schedule is computed by a centralized oine algorithm to be executed by the distributed system. We develop a centralized algorithm finding an optimal schedule in the case when all the transactions use a single object. The general case of multiple objects is studied in two models that determine if executing a transaction may involve sending control messages. For multiple shared objects and with transactions sending control messages, we give a centralized algorithm that finds a schedule with a constant-factor approximation of communication cost with respect to an optimal schedule. For multiple shared objects and with transactions migrating and not using control messages, we give a centralized algorithm that finds a schedule approximating an optimal one by a factor k that equals the maximum number of shared objects requested by a transaction.

1.2. Related work

We discuss related work on data-flow, control-flow, and the dual-flow models. Attiya et al. [18], Busch et al. [19, 20, 21], and Sharma and Busch [9, 22] considered transaction scheduling with provable performance bounds in the data-flow model. Saad and Ravindran [16], Palmieri et al. [23], Siek and Wojciechowski [24, 10] studied scheduling transactions in the control-flow model. Palmieri et al. [23] also gave a comparative study of data-flow versus control-flow models for distributed transactional memory. A prototype distributed transactional memory system described by Saad and Ravindran [25] supports experimentation for both data-flow and control-flow models. Bocchino et al. [26] considered the dual-flow model by allowing programmers to either bring the data to the code of computation (transaction) or send the code of computation to the data. Hendler et al. [27] studied a lease based dual-flow model which dynamically determines whether to migrate transactions to the nodes that own the leases or to demand the acquisition of these leases by the node that originated the transaction.

Transaction scheduling in a distributed system with the goal of minimizing execution time was first considered by Zhang et al. [28]. Busch et al. [19] considered minimizing both the execution time and communication cost simultaneously. They showed that it is impossible to simultaneously minimize execution time and communication cost for all the scheduling problem instances in arbitrary graphs even in the oine setting. Specifically, Busch et al. [19] demonstrated a tradeo between minimizing execution time and communication cost and provided oine algorithms optimizing execution time and communication cost separately. Busch et al. [21] considered transaction scheduling tailored to specific popular topologies and provided oine algorithms that minimize simultaneously execution time and communication cost. In a follow-up work, Poudel and Sharma [29] provided an evaluation framework for processing transactions in distributed systems. Busch et al. [20] studied online algorithms to schedule transactions arriving continuously.

Distributed directory protocols have been designed by Herlihy and Sun [8], Sharma and Busch [9], and Zhang et al. [28], with the goal to optimize communication cost in scheduling transactions. A distributed directory protocol has been designed by Rai et al. [30] in the data-flow model that reduces processing load of network nodes in addition to communication cost.

Alternative approaches to distributed transactional memory systems have been proposed in the literature by way of replicating transactional memory on multiple nodes and providing means to guarantee consistency of replicas. This includes work by Couceiro et al. [31], Hirve et al. [32], Kobus et al. [33], Manassiev et al. [34], Peluso et al. [35], and Peluso et al. [36]. In this work, we use a single copy of each object. Replicas of objects help to improve reliability of the systems rather than decrease the communication overhead. Other systems extend non-distributed transactional memory with a communication layer, for example, the system presented by Kotselidis et al. [37] extends the system described by Herlihy et al. [38] with distributed coherence protocols.

Transaction scheduling has been widely-studied in shared memory multi-core systems. Scheduling algorithms with provable upper bounds, along with lower bounds and impossibility results were given by Attiya et al. [39], Dragojevic et al. [40], Guerraoui et al. [41], Sharma and Busch [42, 43]. Several other scheduling algorithms were evaluated only experimentally, like the ones given in Yoo and Lee [44], Baldassin et al. [45], Manassiev et al. [46], and Kolli et al. [47].

2. Technical Preliminaries

A distributed system consists of processing nodes with some pairs of nodes connected by links. It is represented as a graph $G = (V, E)$. There are n vertices in the set V, each representing a processing node. Edges in the set $E \subseteq V \times V$ represent communication links between nodes. The function $w : E \to \mathbb{Z}^+$ assigns a weight to each edge representing a communication delay. We let $dist(u, v)$ denote the shortest path distance between two vertices u and v.

The initial configuration of the distributed system consists of a set of transactions and shared objects distributed among the nodes. Each node hosts at most one transaction. During executing transactions, both shared objects and transactions can move among the nodes of a network, which we call the dual-flow model. If a transaction requests access to an object, that object may move to a dierent node, possibly closer to the requesting transaction. At the same time, the transaction can also migrate to the object's new location, or send a control message to that new location to access the object. The combined cost of executing a transaction is measured with relation to the distances traversed by the shared objects, transaction and control messages.

In the dual-flow model, objects may not move to every transaction's node. Instead some transactions may need to access some objects at remote nodes. A transaction that performs multiple updates to a remote object iterates communication exchanges between the transaction's node and the object's node. In this case, migrating the transaction closer to the object's node decreases communication costs. To provide a formal framework for such situations, we consider the following two specializations of the dual-flow model for remote object access: (i) Control-message technique, where a transaction sends a control message to access the remote object. The control-message technique is motivated by a scenario in which each transaction performs a number of updates to an object bounded by a constant, with each update requiring a control message, for a total of a constant number of such messages. (ii) Transaction-migration technique, in which a transaction moves to the node where objects are located and no control messages are sent. This technique is motivated by the scenarios in which a transaction may issue a variable number of requests to an object, in which case it is advantageous to migrate the transaction to the object location to avoid potentially unbounded communication overhead.

We parameterize the costs of transmitting messages that carry transactions, objects, or control instructions. The cost of moving an object of size over a unit weight edge is denoted by . We denote the cost of sending a control message over a unit weight edge by . The cost of moving a transaction over a unit weight edge is denoted by .

A scheduling algorithm determines a schedule to execute transactions, including movements of objects and transactions. A centralized algorithm takes as input a configuration of transactions and objects in the system as arranged at the outset. We assume that each node has this input available so that it can execute it locally. A schedule E of executing transactions is a sequence of transactions $T_1; T_2; :::$ that specifies for each transaction when to execute. Each transaction T_i is processed at its turn from the schedule E by accessing the required objects for T_i.

The communication cost of executing such a schedule is the sum of distances traversed by the shared objects, control messages, and transactions according to the schedule, weighted by the corresponding parameters , , and . The following example illustrates the cost of executing transactions in each of the three models and how dual-flow model minimizes total communication cost compared to the data-flow and control-flow models.

Consider a tree G with six nodes $fv_1; v_2; :::; v_6g$ with the topology as shown in Figure 1. Let each edge of G has weight 1 and each node is assigned with one transaction requiring a single shared object o initially positioned at v_1. Let the size of object o be 4 and size of a control message be 2, which means $= 4$ and $= 2$. In the data-flow model, object o needs to visit each node of G to provide access to the transactions. The object may traverse the route $(v_1; v_2; v_3; v_4; v_5; v_6)$

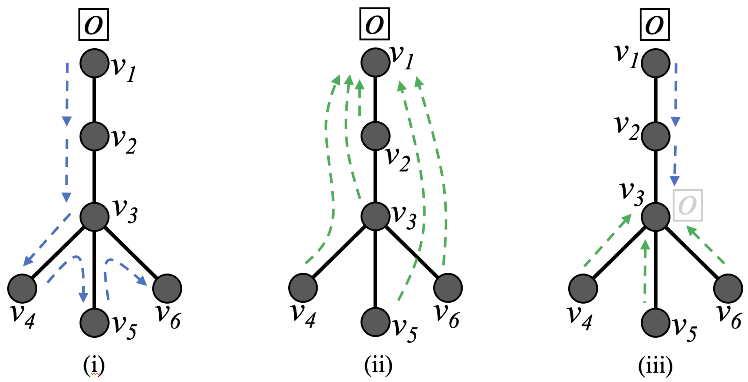

Figure 1: Illustration of communication overhead in (i) data-flow, (ii) control-flow, and (iii) dual-flow model.

to allow the transactions at each node to execute; see part (i) in Figure 1. The total communication cost becomes $4 + 4 + 8 + 8 + 4 = 28$, as the object moves twice in the edges between $v_3; v_4$ and $v_3; v_5$. In the control-flow model, the object resides at v_1 and all the transactions send control messages to v_1 to access o; see part (ii) in Figure 1. The total communication cost for executing all six transactions becomes $0 + 2 + 4 + 6 + 6 + 6 = 24$. In the dual-flow model, object o may move up to node v_3 following the route $(v_1; v_2; v_3)$. The transactions at v_1, v_2 and v_3 execute as soon as o arrives at each node. The remaining transactions at v_4, v_5 and v_6 send control messages to v_3 to access object o; see part (iii) in Figure 1, where blue dashed arrows denote the movements of object and green dashed arrow denote the control messages sent by the transactions. The total communication cost for executing all six transactions in the dual-flow model becomes $4 + 4 + 2 + 2 + 2 = 14$ which is less compared to that in both the data-flow and control-flow models.

3. A Single Object

We assume a single shared object o of size > 1 positioned at the root node of a tree G. Note that G is a static tree with a fixed set of nodes and links, and each node has a global view of the system. We develop an optimal scheduling algorithm denoted as Single-Object in the dual-flow model considering both techniques for accessing a remote object: control-message and transaction-migration. The algorithm is called Single-Object, and its pseudocode is given as Algorithm 1.

3.1. Control-Message Technique

A general idea of the algorithm in the control-message technique is as follows. First we find a set of intermediate nodes in G to move the object o to. These nodes are referred to as supernodes. An intermediate node v becomes a supernode if the cost of moving o from v to one of its children is greater than the cost of sending control messages from the transactions contained by the sub-tree of that child to v. Each supernode contains a set of transactions in its sub-tree which send control messages to that supernode to access object o. These transactions are added to the local execution schedule of the supernode following an iterative pre-order tree traversal in the sub-tree. We determine a subtree P containing paths in G that reach the supernodes from the root of G. Starting from the root, object o travels all the supernodes following the iterative pre-order tree traversal of P. Any transaction that lies along the path is added to the execution schedule E as soon as o reaches the respective node. When o reaches some supernode, the transactions from its local execution schedule get added to E in the respective order. The execution ends when all the transactions have been added to E. The algorithm can be modified as follows if performed in the transaction-migration technique: (i) Determine supernodes with respect to transaction migration cost rather than control messages cost; and (ii) Migrate transactions to the corresponding supernodes instead of sending control messages to access the object. These modifications result in creating an algorithm of a comparable communication performance.

Remember that each node has a global view of the system which helps to determine whether a given node is a supernode or not by comparing the possible cost of control messages vs. object movement. During the computation of supernodes, no object or transaction is moved or a control message is sent, thus there is no communication cost involved. The actual communication cost will occur during the execution of transactions following the respective algorithm. We elaborate on the details of the algorithm next.

The cost of moving o over an edge of unit length is . Let represents the control message cost for a transaction to access object o at one unit away and > . Let $T = \{T_1; T_2; \ldots T_n\}$ be the set of n transactions issued to the nodes of G, one at each node. The first objectives are to determine the walk the object traverses and to find transaction execution schedule. Intuitively, since it costs more to move the object across a link than to send a control message through the link, we strive to move the object minimally, only when this pays, and this approach is captured by the concept of supernodes. The object o first travels from the root up to a supernode. Transactions that lie along the path the object traverses execute as soon as the object reaches the respective nodes. The remaining transactions beyond that supernode and towards the leaves (i.e., in the sub-tree of the supernode) send control messages to the supernode to access the object. Observe that the control messages are sent only up to the supernode and the transactions will execute at that supernode after the object reaches there. When all the transactions that have sent control messages to the current supernode finish their executions, object o moves to the next supernode and the remaining transactions get executed following a similar approach.

The communication cost of an execution of the algorithm is determined by the location of supernodes. The set of supernodes is selected by referring to transaction counts and transaction loads at all nodes, which are defined as follows. A transaction count at node v, denoted txnum(v), is the total number of transactions contained in the sub-tree of node v, including v (Line 4 of Algorithm 1). A transaction load of a node v, denoted txload(v), is the sum of distances from v to the positions of transactions contained in the sub-tree of v, including v (Lines 9–15 of Algorithm 1). The transaction load of v represents the cost of sending control messages due to the transactions contained in its sub-tree, assuming o is moved to v.

To identify supernodes, we start from the leaves of G and work through the ancestors towards the root (Lines 16–24 of Algorithm 1). Let v_{cur} be a leaf node and v_{next} be the parent of v_{cur}. During the computation of supernodes, we can assume that the object is at the parent node v_{next} and check if it pays to move the object down to v_{cur}, since object moves away from the root. Let txload(v_{cur}) denote the control message cost incurred by the txnum(v_{cur}) number of transactions contained in the sub-tree of v_{cur}, including v_{cur}. If the object o moves to v_{cur}, the transactions contained in the sub-tree of v_{cur} can access o at v_{cur} and the cost becomes txload(v_{cur}) + dist($v_{cur}; v_{next}$). Here, dist($v_{cur}; v_{next}$) is the cost incurred by the movement of object o from v_{next} to v_{cur}. Otherwise, these transactions send control messages to v_{next} to access o and the cost becomes txload(v_{cur}) + txnum(v_{cur}) dist($v_{cur}; v_{next}$). Object o will move to v_{cur} from v_{next} only if the control message cost from v_{cur} to v_{next}, due to the transactions contained in the sub-tree of v_{cur}, is more than or equal to the object movement cost from v_{next} to v_{cur}. This has been checked in Lines 19–23 of Algorithm 1.

After reaching a supernode, object o may need to move back to the root or intermediate nodes to visit other supernodes. To account for this and simplify the argument, we assume that the object moves over each edge twice, but this assumption will be revisited when we optimize the algorithm (Line 25 of Algorithm 1). If the following inequality holds:

$$\mathrm{txload}(v_{cur}) + 2\ \mathrm{dist}(v_{cur}; v_{next})\ \ \mathrm{txload}(v_{cur}) + \mathrm{txnum}(v_{cur})\ \ \mathrm{dist}(v_{cur}; v_{next})\ ;$$

then we choose v_{cur} as a supernode. Otherwise, if v_{cur} is not the root, a new pair of v_{cur} and v_{next} is checked such that current v_{next} becomes new v_{cur} and the parent of current v_{next} becomes a new node v_{next}. If v_{cur} is the root, then it becomes a supernode (Line 24 of Algorithm 1).

Let P denote the pruned tree, which contains only the supernodes and nodes that need to be traversed on the way from the root to a supernode. Tree P is rooted at the root of G. Figure 2 illustrates such a tree P. The object o is originally located at the root, from which it moves to the supernodes in a pre-order traversal manner. The transactions are executed along the way of the object's movement. Transactions at

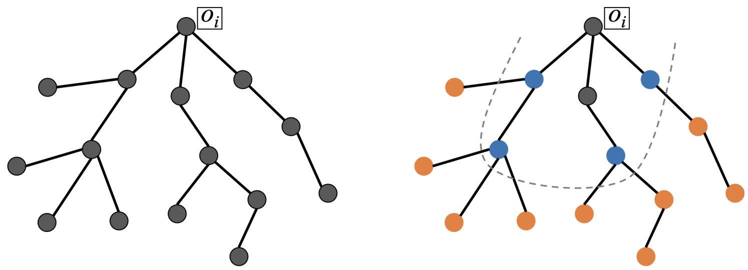

Figure 2: Identification of supernodes by algorithm Single-Object. The tree on the left is G. The tree on the right is the same G after determining the status of nodes. Supernodes are colored blue. Nodes on the path from the root to a blue node are colored black. The dashed line delineates P obtained from G by pruning G of vertices beyond the supernodes, which are colored orange.

the nodes beyond the pruned tree P, marked by color orange in Figure 2, either send control messages or move to access o to their closest supernodes. When object o reaches the respective supernode, these transactions are executed in order.

Lemma 1. If a node v does not belong to the pruned tree P, then the total number of transactions contained in the sub-tree of v is less than 2.

Proof. This follows from the specification of the nodes that make P and how supernodes are determined, while we assume that the object moves over each edge twice.

After computing the set of supernodes, the object performs a pre-order tree traversal starting from the root to visit all the supernodes. The transaction execution schedule E is computed as follows. First add transaction at the root to E. During the pre-order tree traversal to visit the supernodes, if E does not contain the transaction at a visited node v, then add it to E. If the visited node v is a supernode, add to E the transactions that sent control messages to v from the subtree rooted at v. Lines 27–31 of Algorithm 1 represent how each transaction is added into the schedule E following the pre-order tree traversal.

Next we show how to refine this approach, which is based on the assumption that during the computation of supernodes if the object moves from some parent node to the child node, then it will ultimately move back from that child node to the parent. When the object reaches the last supernode, it does not move back because there is no any other supernode remained to visit. A pseudocode to accomplish this is given as Algorithm 2.

We define a one-way path to be such a path from v_{root} to the last supernode v_{last}, all the edges of which the object traverses only once. This v_{last} must be chosen in such a way that the total communication cost is minimized. A condition for computing a supernode is:

$$2 \quad dist(v_{cur}; v_{next}) > txnum(v_{cur}) \quad dist(v_{cur}; v_{next}) \tag{1}$$

Algorithm 1: Single-Object

Input : Tree graph G of n nodes containing T transactions
Output: Transaction execution schedule E

v_{root} root of G at which o lies initially;
L set of leaves of G;
; cost of moving o and a control message over a unit weight edge of G, respectively;
txnum(v_i) transactions contained in the sub-tree of v_i;
txload(v_i) sum of distances from v_i to transactions in sub-tree of v_i ; // initialize 0
S set of supernodes ; // initialize fg
C(v_i) child nodes of v_i 2 S towards which object o does not move further;
D(v_i) set of candidates for the last supernode that are descendants of v_i 2 S ;
/* Compute txload(v_i) iteratively */
for each transaction T_i 2 T do
    v_i current node of G at which T_i is positioned;
    v_{cur} v_i; txnum(v_i) + +;
    while v_{cur} , v_{root} do
        v_{next} parent node of v_{cur} in G; txnum(v_{next}) + +;
        txload(v_{next}) txload(v_{next}) + dist(v_i; v_{next});
        v_{cur} v_{next};
/* Compute set of supernodes */
for each node v 2 L do
    v_{cur} v; v_{prev} null; v_{cand} null;
    v_{next} parent node of v_{cur} in G;
    while (v_{cur} , v_{root}) ^ (txload(v_{cur}) + 2 dist(v_{cur}; v_{next}) > txload(v_{cur}) + txnum(v_{cur}) dist(v_{cur}; v_{next})) do
        if > txnum(v_{cur}) then
            v_{cand} v_{cur}; add v_{prev} to C(v_{cur});
        v_{prev} v_{cur}; v_{cur} v_{next};
        v_{next} parent node of v_{next} in G;
    add v_{cur} to S ; add v_{prev} to C(v_{cur}); add v_{cand} to D(v_{cur})
/* Find last supernode to visit optimizing total cost */
v_{last} FindLastSuperNode(S; D; C);
reorder G so that each node on path from v_{root} to v_{last} becomes the last child of parent;
perform a pre-order tree traversal on G starting at v_{root};
    if a visited node v_{cur} is a supernode then
        move the object o to v_{cur};
        add transactions in the sub-tree of each node v 2 C(v_{cur}) to E;
    add transaction at v_{cur} to E, unless added already;

Inequality 1 accounts for the object traversing each edge twice, which is not required for v_{last}. The object can move further down until the following holds:

$$\mathrm{dist}(v_{cur}; v_{next}) > \mathrm{txnum}(v_{cur}) \quad \mathrm{dist}(v_{cur}; v_{next}) \qquad (2)$$

We find the last supernode v_{last} and the one-way path as follows. Let S be the initial set of supernodes computed considering that the object moves twice on each edge

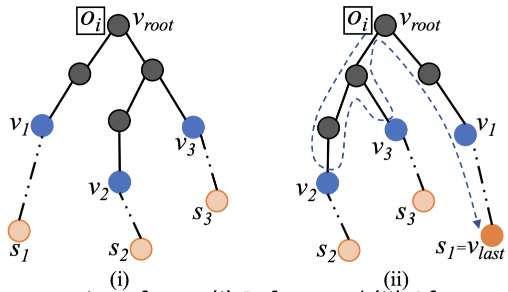

Figure 3: Illustration of computation of v_{last}. (i) Before; and (ii) After computing v_{last} and rearranging G.

up to the supernode. In a one-way path, the object may move further down towards the leaf node satisfying the condition in Inequality (2). For each node v 2 S, if the sub-tree of v contains multiple branches, there could be a number of possible paths for the object to move. There will always be a unique one-way path that minimizes the total cost. In each sub-tree of v 2 S, we find the set of nodes D(v) that are candidates for v_{last} using the condition in Inequality (2) (Lines 20, 21 and 24 of Algorithm 1). Then the dierence between the cost of selecting v as a supernode and v_j 2 D(v) as a supernode is computed (Lines 2–9 of Algorithm 2). Among these dierences for every v 2 S, the one with the highest dierence is chosen as the last supernode v_{last} (Line 10 of Algorithm 2). Let v_{ref} 2 S and v_k 2 D(v_{ref}) be the set of two nodes that provided the highest dierence. Then v_k becomes v_{last} and is added to S. The path from v_{root} to v_{last} becomes the one-way-path and is visited at last following the pre-order tree traversal. Moreover, if a node between v_{ref} and v_{last} (including v_{ref}) in the one-way-path contains transactions in its sub-tree other than the one-way-path branch, it becomes a supernode to serve control requests to the transactions in those branches and is added to S (Lines 11–18 of Algorithm 2).

Figure 3 illustrates the computation of v_{last} and rearrangement of G for the pre-order tree traversal.

Lemma 2. If v is a descendant of v_{last}, then the total number of transactions contained in the sub-tree of v is always less than .

Proof. This follows from the specification of the nodes that make P and how the last supernode v_{last} is determined. We assume that the object traverses only once on each link in the path from the root to v_{last} that is not needed for backtracking.

Lemma 3. For any transaction, the corresponding supernode for accessing the object always lies at or above its position along the path towards the root of G.

Proof. To compute a supernode in G, we start from a leaf node and proceed towards the root node following the shortest path.

Algorithm 2: FindLastSuperNode(S; D; C)

v_{last} null; v_{ref} null; di 0;
for each node v 2 S do
  ctrl(v) cost of control messages sent to v;
  twowaycost 2 dist(v_{root}; v) + ctrl(v);
  for each node v_j 2 D(v) do
    onewaycost dist(v_{root}; v_j) + ctrl(v_j);
    if twowaycost onewaycost > di then
      di twowaycost onewaycost;
      v_{last} v_j; v_{ref} v;
add v_{last} to S; v_{cur} v_{last}; set v_{next} to the parent of v_{last};
while v_{cur} , v_{ref} do
  if v_{next} has more children than v_{cur} then
    add v_{next} to S;
    for each child node v_k , v_{cur} of v_{next} do
      add v_k to C(v_{next});
  if v_{next} == v_{ref} then
    remove v_{cur} from C(v_{next});
  v_{cur} v_{next}; set v_{next} to the parent of v_{next};
return v_{last};

Theorem 1. Algorithm Single-Object schedules transactions with the optimal communication cost.

Proof. Let S be the set of supernodes found for a tree G with respect to object o. We will show that any other selection of supernodes gives strictly higher communication cost and hence, S provides optimal communication cost.

To simplify the problem, without loss of generality, we assume that each edge of G has weight 1, = 1 and > . Let P be the pruned tree containing nodes only up to the supernodes starting from the root of G. Let v_{last} 2 S be the last supernode for object o to visit. Let C be the total communication cost of Algorithm Single-Object. Let v 2 S be a supernode in G, v_p be an ancestor of v with distance dist(v_p; v) 1, and v_q be a descendant of v with dist(v; v_q) 1. Based on the positions of v and v_q, it can have one of the following three cases:

Case (a): v = v_{last}. Then, by Lemma 2, we have that

$$\text{txnum}(v_p)\ \ \text{txnum}(v)\ \ > \text{txnum}(v_q) \quad (3)$$

Case (b): v , v_{last}, v_q < P, and the path from v to v_q contains no other supernode, in that v is the bottommost supernode in the current branch. Then, by Lemma 1, we have

$$\text{txnum}(v_p)\ \ \text{txnum}(v)\ \ 2 > \text{txnum}(v_q) \quad (4)$$

Case (c): Either v_q 2 P or v_q < P and the path from v to v_q contains at least one other supernode. Let $z \geq 1$ be the transactions that send control messages to v to access o.

We have following four subcases with respect to each supernode $v \in S$:

(i) Choosing an ancestor of v as a supernode instead of v increases communication:

Let S_p be the set of nodes contained between v and v_p (excluding both). Suppose v_p be selected as a supernode instead of v. Then in Case (a) and Case (b), o moves only up to v_p, and in addition to the transactions issued to the sub-tree of v, all the transactions between v and v_p send control messages to v_p. But, in Case (c), since the sub-tree of v (excluding v) still contains another supernode $v_k \in S$, o still moves to v_k passing through v. When v was the supernode, $z \geq 1$ transactions could access o at v. Now, since v_p is selected as the supernode instead of v, all those z transactions send control messages to v_p to access o. So, the total communication cost C_{v_p} of selecting v_p as a supernode compared to that of selecting v in each case becomes:

$$C_{v_p} = \begin{cases} C - \mathrm{dist}(v_p; p) + \mathrm{txnum}(v) \cdot \mathrm{dist}(v_p; v) \\ \quad + \sum_{v_k \in S_p} (\mathrm{txnum}(v_k) - \mathrm{txnum}(v)); & \text{Case (a)} \\ C - 2 \cdot \mathrm{dist}(v_p; p) + \mathrm{txnum}(v) \cdot \mathrm{dist}(v_p; v) \\ \quad + \sum_{v_k \in S_p} (\mathrm{txnum}(v_k) - \mathrm{txnum}(v)); & \text{Case (b)} \\ C + z \cdot \mathrm{dist}(v_p; v); & \text{Case (c)} \end{cases}$$

In Case (a), from Inequality (3), since $\mathrm{txnum}(v)$, $C_{v_p} > C$. In Case (b), from Inequality (4), since $\mathrm{txnum}(v)$ 2, $C_{v_p} > C$. Also, in case (c), $C_{v_p} > C$.

(ii) Choosing a descendant of v as a supernode instead of v increases communication:

Now, we analyze the communication cost of selecting a descendant node v_q as a supernode instead of $v \in S$. Let S_q be the set of nodes contained between v and v_q (excluding both). As v_q is a new supernode, object moves up to it. So, in Case (a) and Case (b), to get the change in total communication cost compared to C, we have to add object movement cost of o from v to v_q and subtract the control message cost for the transactions between v and v_q. Moreover, the transactions in the sub-tree of v_q will also send control messages only up to v_q. Thus, the total communication cost C_{v_q} of selecting node v_q as a supernode compared to C in Case (a) and Case (b) becomes:

$$C_{v_q} = \begin{cases} C + \mathrm{dist}(v; v_q) - \mathrm{txnum}(v_q) \cdot \mathrm{dist}(v; v_q) \\ \quad - \sum_{v_k \in S_q} (\mathrm{txnum}(v_k) - \mathrm{txnum}(v_q)); & \text{Case (a)} \\ C + 2 \cdot \mathrm{dist}(v; v_q) - \mathrm{txnum}(v_q) \cdot \mathrm{dist}(v; v_q) \\ \quad - \sum_{v_k \in S_q} (\mathrm{txnum}(v_k) - \mathrm{txnum}(v_q)); & \text{Case (b)} \end{cases}$$

Let $dist(v; v_q) = k$ where k 1. In Case (a), from Inequality (3), $txnum(v_q) <$. Let $txnum(v_q) =$ j, 1 j < . Following Lemma 2, the nodes between v and v_q (i.e., S_q) contain at most j number of transactions. The control message cost sent to v due to these transactions is: $\sum_{v_k 2 S_q}(txnum(v_k) \quad txnum(v_q)) <$ j k. Thus,

$$C_{v_q} > C + \quad k \quad (\quad j) \; k \quad j \; k > C :$$

In Case (b), $txnum(v_q) <$ 2 by the Inequality (4). Let $txnum(v_q) = 2$ l, for 1 l < 2. By Lemma 1, there are at most l transactions between v and v_q, and control message cost sent to v due to them is: $\sum_{v_k 2 S_q}(txnum(v_k) \quad txnum(v_q)) <$ l k. Thus

$$C_{v_q} > C + 2 \; k \quad (2 \quad l) \; k \quad l \; k > C :$$

Now, we analyze Case (c). Based on the position of v_q, it has two sub-cases:

Case (c.1): v_q 2 P. There is no extra movement of o and the z 1 number of transactions that previously depend on v now send control messages to v_q to access o. So, the total communication cost C_{v_q} compared to C becomes: $C_{v_q} = C + z \; dist(v; v_q) > C$.

Case (c.2): v_q < P but the path from v to v_q contains at least one other supernode in S. The node v_q lies below the bottommost supernode of current branch. Let v_{bot} 2 S be the bottommost supernode in the path between v and v_q. When v_q is selected as a supernode, there will be extra movement of object o from v_{bot} up to v_q. If $v_{bot} = v_{last}$, and o moves up to v_q. Otherwise, object o also needs to return back at v_{bot}. Let M represents the cost due to the movement of object o between v_{bot} and v_q, then, M > $dist(v_{bot}; v_q)$. Thus, the total communication cost C_{v_q} compared to C in this case becomes: $C_{v_q} = C + z \; dist(v; v_q) + M > C$.

(iii) Merging multiple supernodes at some ancestor node increases communication:

Consider two supernodes $v_r; v_s$ 2 S have a common ancestor v_y. Instead of v_r and v_s, let v_y be chosen as a supernode. Since v_y is ancestor of both v_r and v_s, following argument (i), total communication cost C_{v_y} of selecting v_y as a supernode instead of v_r and v_s is more compared to C.

(iv) Splitting any supernode into multiple supernodes increases communication:

Consider a supernode v_j 2 S. Let $v_x; v_z$ be two descendant nodes of v_j at two dierent sub-branches. Let v_x and v_z are chosen as two dierent supernodes instead of v_j. Since both v_x and v_z are descendants of v_j, following argument (ii), total cost $C_{v_{xz}}$ of selecting $v_x; v_z$ as supernodes instead of v_j is more compared to C.

The set of supernodes S computed in algorithm Single-Object is unique. If any new node is added to S or any node in S is removed or replaced by another node, the total communication cost increases. It implies that scheduling by algorithm Single-Object provides the optimal communication cost.

Theorem 2. Algorithm Single-Object provides 2-approximation in communication cost without optimization.

Proof. The algorithm in its optimized version has object o traverse each edge along the path from v_{root} to v_{last} only once. The algorithm without optimization does not identify and leverage the last supernode v_{last} for object o to visit and reorder G accordingly. Without the optimization, object o still visits each edge along the path from v_{root} to the supernode(s) at most twice.

3.2. Transaction-Migration Technique

Next we consider the transaction-migration technique. Let be the cost of moving a transaction over a unit weight edge of G. Consider algorithm Single-Object modified such that transactions are moved instead of sending control messages to the supernodes and the cost of moving transaction replaces the cost of sending control messages, in that we use the parameter instead of . After these modifications in algorithm Single-Object and its analysis, we obtain optimality similarly as stated in Theorem 1.

4. Multiple Objects

We provide two scheduling algorithms for multiple shared objects, which extend the single object algorithm above. For the control-message technique, we present the algorithm denoted as MultipleObjects-CtrlMsg, which provides an O(1)-approximation. For the transaction-migration technique, our algorithm is denoted as MultipleObjects-TxMigr, which provides O(k)-approximation, where k is the maximum number of shared objects accessed by a transaction.

The idea in the version MultipleObjects-CtrlMsg is as follows: first, we compute separate sets of supernodes with respect to each object in O and the transactions accessing that object. Each transaction will have a list of supernodes at which required objects can be accessed with the minimum cost. We select a random node of G as the virtual root (v_{root}) of G. After that, transactions are added to the execution schedule E by following the iterative pre-order tree traversal algorithm in G rooted at v_{root}. Objects will then move to the respective supernodes following the iterative pre-order tree traversal in G. Particularly, a transaction T_i is scheduled to execute at time step $t(T_i)$ in such a way that the objects required by T_i are also reached at the respective supernodes where T_i accesses them at time step $t(T_i)$. When an object o_i 2 O reaches a supernode v_s, it stays there until all the transactions that access object o at v_s finish their executions. Object o then moves to the next supernode in order where other transactions are waiting for it. The algorithm ends when all the transactions are added to the schedule E.

In algorithm MultipleObjects-TxMigr), if all the objects are positioned at the same node initially, we assume that node as the virtual root (v_{root}) of G. Otherwise, if objects are at arbitrary nodes of G initially, we find the virtual root of G with respect to the initial positions of objects in O and the positions of transactions accessing those objects. We then move the objects to v_{root}. We compute separate sets of supernodes with respect to each object $o_i \in O$ positioned at v_{root} of G. Then, if a transaction requires multiple objects, it may have a set of dierent supernodes to access each of them. Among them, we select a single supernode (called common supernode) at which all the required objects for that transaction are gathered together and the transaction is also migrated during the execution. We find a common supernode for each transaction $T \in \mathcal{T}$. At the common supernode, objects required by T (i.e., $objs(T)$) wait for each other like threads wait for each other at a barrier. Then, we find a pruned tree P containing the nodes only up to the common supernodes starting from the virtual root in G. We perform an iterative pre-order tree traversal in P to move objects from one common supernode to the next. At each common supernode $v_s \in P$, the transactions which have chosen v_s as the common supernode are scheduled for execution following the iterative pre-order tree traversal in the sub-tree of v_s. As previously, at each common supernode v_s, object o stays until all the transactions requiring object o at v_s finish their executions. Object o then moves to the next common supernode. The algorithm terminates once all the transactions get scheduled.

We consider a set of shared objects $O = \{o_1, o_2, \ldots, o\}$ initially positioned at arbitrary nodes of G. We assume that each object has size . Each transaction in $\mathcal{T}$ accesses a subset of objects in O. Let $objs(T_i) \subseteq O$ be the set of objects accessed by transaction T_i. We assume that each object has a single copy and $home(o_i) \in V$ represents the home node at which object o_i is originally positioned. The ownership of an object is also transferred with the movement of that object. Similarly, $home(T_i) \in V$ represents the node at which transaction T_i is positioned.

The idea in the algorithms is to provide synchronized accesses to the objects with minimum cost while executing the transactions in order. We achieve this extending the techniques used in algorithm Single-Object. In particular, we compute supernodes w.r.t. each object and the transactions requiring those objects. We then perform iterative pre-order tree traversal to move each object to the respective supernodes and execute transactions in order.

For brevity, let T_i be a transaction that requires objects in $objs(T_i) = \{o_x, \ldots, o_z\}$. Let $sv_i(o_x), \ldots, sv_i(o_z)$ be the respective supernodes (computed using algorithm Single-Object w.r.t. each object) at which T_i can access $o_x, \ldots, o_z$, respectively. Then, one way of providing synchronised access to the required objects by T_i is to bring each object in $objs(T_i)$ at the respective supernode (i.e., $sv_i(o_x), \ldots, sv_i(o_z)$) at the same time so that T_i can access them by sending control messages. This approach is used in the control-message technique. The other way is to gather all the objects in $objs(T_i)$ at a single node $sv(T_i)$ (i.e., common supernode for T_i) and access them at that node by migrating T_i. This approach is used in the transaction-migration technique.

We now describe how transactions are executed in order and the objects are moved from one supernode to the next minimizing the communication cost. As in algorithm Single-Object, this can be achieved using iterative pre-order tree traversal algorithm in G, provided that there is a single reference point, i.e., root node. We find a virtual root (v_{root}) of tree G as a single reference point.

In the control-message technique, any node of G can be selected as the virtual root (v_{root}). In the transaction-migration technique, if all the objects are initially positioned at the same node, that node is selected as the virtual root of G. If objects are positioned at dierent nodes initially, we compute the virtual root with respect to the initial positions (home nodes) of transactions and the objects they access. The virtual root of tree G is the node in G from which the sum of distances to home nodes of all the transactions and the objects they access is the minimum, that is,

$$v^C_{root} = v_i : W(v_i) = \min_{v \in V} W(v); \tag{5}$$

where,

$$W(v) = \sum_{j=1}^{n} dist(v; home(T_j)) + \sum_{o \in objs(T_j)} dist(v; home(o)):$$

To compute the virtual root of G, we take into account both the initial positions of transactions and the initial positions of objects. This is needed since we want to minimize the distances from initial positions of objects to the virtual root as well as the distances from virtual root to the transactions.

4.1. Multiple Objects with Control Messages

The algorithm for multiple objects in the control-message technique is named MultipleObjects-CtrlMsg, and the pseudocode is given in Algorithm 3. The algorithm runs in two phases.

Phase 1: We compute sets of supernodes $S(o_i)$ with respect to each object $o_i \in O$ individually following algorithm Single-Object without optimization. For each o_i, $home(o_i)$ is assumed as the root of G during the computation of respective supernodes $S(o_i)$. If a transaction T_i requires an object o_j, T_i accesses o_j at supernode $sv(T_i(o_j)) \in S(o_j)$.

Phase 2: We find transaction execution schedule E and paths of movement for each object $o_i \in O$ along their respective supernodes. For this, let a random node in G be

Algorithm 3: MultipleObjects-CtrlMsg

Input : Tree graph G of n nodes containing T transactions and O objects positioned at arbitrary nodes

Output: Transaction execution schedule E

$v^0_{root} \leftarrow$ virtual root of G;

/* Phase 1: Find set of supernodes for each object in O following algorithm Single-Object without optimization */

for each object $o_i \in O$ **do**

  $S(o_i) \leftarrow$ set of supernodes with respect to o computed using algorithm Single-Object without optimization and assuming home(o_i) as the root of G;

  $S \leftarrow S \cup S(o_i)$;

/* Phase 2: Pre-order tree traversal on G */

perform pre-order tree traversal on G rooted at v^0_{root}

  $v_{cur} \leftarrow$ current visited node on G;

  if v_{cur} is a supernode with respect to o **then**

    move object o to v_{cur};

  add transaction at v_{cur} to E, if not added previously;

selected as the virtual root v^c_{root} of G. We perform an iterative pre-order tree traversal in G starting from v^c_{root}. During the traversal, if there is a transaction T_j at current node v_{cur}, T_j is added to the schedule E. Each object o_k required by T_j (in notation, $o_k \in objs(T_j)$) is scheduled to move to the respective supernode $sv_j(o_k)$. When the traversal of G completes, all the transactions get scheduled and the execution ends.

Lemma 4. *An object o may traverse an edge along the path from home(o_i) to v^c_{root} at most three times.*

Proof. Let G be a tree and v^c_{root} be the virtual root of G. Let the shared objects $O = \{o_1, \ldots, o\}$ be positioned at arbitrary nodes of G. Then, with respect to each object $o_i \in O$, a set of supernodes $S(o_i)$ is computed using algorithm Single-Object. Let $S = \bigcup_{i=1} S(o_i)$ be the union of all supernodes and P be the pruned tree containing all the nodes in S starting from v^c_{root}. Transactions are scheduled following an iterative pre-order tree traversal in G rooted at v^c_{root} and objects are moved from one supernode to the next in S following the iterative pre-order tree traversal in P rooted at v^c_{root}. We start from v^c_{root}. For the transaction T assigned at v^c_{root}, if v^c_{root} is a supernode with respect to some object $o_i \in objs(T)$, then object o moves from home(o_i) to v^c_{root}. Otherwise, $o_i \in objs(T)$ moves to the respective supernode $sv(T(o_i))$ below v^c_{root}. By Lemma 3, the supernode for transaction T at v^c_{root} for accessing object o is always at v^c_{root} or below v^c_{root} towards home(o_i). After this, thanks to the properties of the pre-order tree traversal that each edge of P is visited no more than twice and hence each object $o_i \in O$ also traverses each edge along the path from v^c_{root} to $sv(T(o_i))$ in P at most two times. home(o_i) itself can be one of the supernodes $sv(T(o_i))$ for T with respect to object o in P. Thus, in total, object o can traverse at most 3 times along the path from home(o_i) to v^c_{root}.

Theorem 3. Algorithm MultipleObjects-CtrlMsg provides a 3-approximation of communication cost.

Proof. Let $S(o_i)$ be the set of supernodes computed with respect to object $o_i \in O$ following algorithm Single-Object without optimization. Let P_i be the pruned tree containing nodes only up to the supernodes $S(o_i)$ starting from home(o_i) in G. Let C_{obj} denotes the cost of moving object o_i at each edge inside P_i only once and C_{ctrl} denotes the communication cost incurred due to the control messages sent from transactions beyond P_i in G. By the analysis of algorithm Single-Object , o_i visits each edge of P_i at most twice during the execution. Theorem 1 shows that the set of supernodes computed in algorithm Single-Object provides the minimum communication cost and Theorem 2 shows that algorithm Single-Object without optimization provides 2-approximation. Thus, if $C_{OPT}(o_i)$ be the optimal communication cost for accessing o_i by a set of transactions T , then,

$$C_{obj} + C_{ctrl} \le C_{OPT}(o_i) \le 2(C_{obj} + C_{ctrl}) \tag{6}$$

and $C_{OPT} = \sum_{o_i \in O} C_{OPT}(o_i)$.

The algorithm in MultipleObjects-CtrlMsg uses the same set of supernodes $S(o_i)$ computed in algorithm Single-Object without optimization and object o_i does not move beyond the pruned tree P_i. So, C_{ctrl} for MultipleObjects-CtrlMsg remains the same. From Lemma 4, object o_i may traverse an edge inside P_i at most 3 times. Thus, if $C_{ALG}(o_i)$ represents the total communication cost for accessing o_i by a set of transactions T , then,

$$C_{ALG}(o_i) \le 3C_{obj} + C_{ctrl} \tag{7}$$

Inequalities (6) and (7) imply

$$C_{ALG}(o_i) \le 3 \cdot C_{OPT}(o_i) \tag{8}$$

This gives the estimate

$$C_{ALG} = \sum_{o_i \in O} C_{ALG}(o_i) \le \sum_{o_i \in O} (3 \cdot C_{OPT}(o_i)) \le 3 \cdot C_{OPT} ;$$

where C_{ALG} represents the total communication cost in MultipleObjects-CtrlMsg for executing all the transactions accessing multiple objects and C_{OPT} represents that of any optimal algorithm.

4.2. Multiple Objects with Migration of Transactions

The algorithm for multiple objects implemented in the transaction-migration model is named MultipleObjects-TxMigr. First, we discuss the algorithm assuming all the objects are initially positioned at the same node (i.e., home(o_i) for each

Algorithm 4: MultipleObjects-TxMigr

```
Input : Tree graph G of n nodes containing T transactions and O objects
Output: Transaction execution schedule E
v0root     virtual root of G;
/* Phase 1:   Compute supernodes */
for each object oi 2 O do
    if home(oi) ,  v0root then move oi to v0root; S(oi)     set of supernodes of oi;
/* Phase 2:   Find common supernode */
for each transaction T 2 T do
    for each object oi 2 objs(T) do
        sv(T(oi))     node in S(oi) where T accesses oi;
    v     the node in sv(T(oj)); oj 2 objs(T); which has the minimum dist(sv(T(oj)); v0root);
    sv(T)     v; numtxs(sv(T)) + +;
    for each object oi 2 objs(T) do
        sv(T(oi))     v; objs(sv(T))     objs(sv(T)) [ oi;
        txs(sv(T)(oi))     txs(sv(T)(oi)) [ T;
/* Phase 3:   Finalize supernodes */
L     set of leaf nodes in G;
repeat
    for each node v 2 L do
        while numtxs == 0 do v     parent of v in G;
        if numtxs(v) < 2 jobjs(v)j then
            for each object oi 2 objs(v) do
                while v ,  v0root ^ jtxs(v(oi))j < 2 do
p               parent node of v in G;
                    txs(v(oi))     txs(v(oi)) [ txs(p(oi)); v     p;
                for each transaction T in txs(v(oi)) do sv(T)     v;
    FinalSV     fg;
    for each transaction T 2 T do FinalSV     FinalSV [ sv(T);
    P     pruned tree containing nodes up to FinalSV starting from v0root in G;
    L     leaf nodes of P;
    for each node v 2 L do  if jnumtxs(v)j    2 then exit repeat; /*
Phase 4:      Find execution schedule */
perform iterative pre-order tree traversal on P;
    if current node vcur 2 FinalSV then
        move each object o 2 objs(vcur) to vcur;
        add each T 2 T , where sv(T) = vcur, to E;
```

o_i 2 O is the same) which is the virtual root v^c_{root} of G. Later, we relax the algorithm where objects can be positioned initially at arbitrary nodes in G. A pseudocode is given as Algorithm 4.

The algorithm works in four phases. In Phase 1, we compute sets of supernodes with respect to individual object o_i 2 O. In Phase 2, we find a common supernode for each transaction T 2 T where all the required objects for T can be gathered together. In Phase 3, we finalize the set of common supernodes. Finally, in Phase 4, we perform iterative pre-order tree traversal on G to create transaction execution schedule and object movement paths along the respective common supernodes. We describe each phase in detail next.

Phase 1: In this phase, we compute supernodes with respect to each object o_i 2 O using algorithm Single-Object without optimization where control message cost over an edge is replaced with the transaction migration cost . Let $S(o_i)$ be the set of supernodes with respect to object o_i 2 O and $sv(T(o_i))$ 2 $S(o_i)$ represents the supernode for transaction T at which T accesses o_i. After this, each transaction T_j 2 T has a set of respective supernodes $sv(T_j(o_i))$ to access each required object o_i 2 $objs(T_j)$. Since all the objects in $objs(T_j)$ need to gather at a single node, a common supernode $sv(T_j)$ for transaction T_j is selected out of all $sv(T_j(o_i))$ in the next phase.

Observation 1. For a transaction T, all the supernodes $sv(T(o_i))$, for o_i 2 $objs(T)$; lie on the same path towards the root.

Proof. All the objects are initially positioned at the same node v_{root}, so it suces to refer to Lemma 3.

Phase 2: In this phase, we find a common supernode of objects $sv(T)$ for each transaction T 2 T . The objective of selecting a common supernode for a transaction is to allow all the required objects for that transaction to gather together at the common supernode. After that, the transaction is also migrated at the common supernode and all the required objects are accessed locally. For a transaction T, if all the supernodes $sv(T(o_i))$, o_i 2 $objs(T)$; computed in Phase 1 are the same, it automatically becomes the common supernode for T. If they are dierent, then we select the one among $sv(T(o_i))$; o_i 2 $objs(T)$; which is the closest from v_{root}.

Theorem 4. For a transaction T 2 T , selecting the topmost node among $sv(T(o_i))$, o_i 2 $objs(T)$, as a common supernode provides the minimum communication cost during the execution of T.

Proof. Consider a tree graph G rooted at v_{root}. Let T be a transaction requiring a set of objects f$o_1; o_2; : : : ; o_k$g. Let $sv_1; sv_2; : : : ; sv_k$ be the supernodes for T with respect to objects $o_1; o_2; : : : ; o_k$; respectively, computed in Phase 2. Among them, let sv_1 be the topmost, sv_k be the bottommost and the remaining nodes lie between sv_1 and sv_k. By Observation 1, all the supernodes $sv_1; sv_2; : : : ; sv_k$ lie on the same path towards v_{root}. First, let sv_1 be selected as the common supernode for T. If C_{obj} be the cost of moving objects $o_1; o_2; : : : ; o_k$ from v_{root} to sv_1, then the cost of executing T at sv_1 becomes C_{obj} + dist(home(T); sv_1). Now, instead of sv_1, let sv_2 be selected as the common supernode for T. Then, all the objects $o_1; o_2; : : : ; o_k$

move up to node sv_2 and T also moves to sv_2 to execute. Thus, total cost for executing T becomes $C_{obj} + 2\ |objs(T)|\ dist(sv_1; sv_2) + \ dist(home(T); sv_2)$. On the one hand, since o_1 moves below sv_1, from the argument (ii) of Theorem 1, the communication cost increases. On the other hand, object movement cost increases by $2|objs(T)|\, dist(sv_1; sv_2)$ which is more than the decrease in transaction movement (i.e., $dist(sv_1; sv_2)$). Thus, in any case, communication cost for executing T at sv_2 by selecting it as a common supernode is more compared to that by selecting sv_1 as a common supernode. Arguing similarly, selecting any descendant of sv_1 as a common supernode for executing T increases the communication cost than that with sv_1. Let us also see what happens if we select an ancestor sv_1 as a common supernode. Note here that, during the merging of supernodes up to sv_1, there is no extra cost due to the movement of transaction since sv_1 already accounts for the transaction movement cost up to it. If some ancestor of sv_1 is selected as a common super for executing T instead of sv_1, T needs to move further upward up to that ancestor node, and from the argument (i) of Theorem 1, the communication cost increases. Hence, the theorem follows.

Optimization. By the end of Phase 2, each transaction $T \in \mathcal{T}$ has a corresponding common supernode $sv(T)$ at which T can migrate for accessing all the required objects and execute. Since some of the initially computed supernodes with respect to individual objects are now merged together at some ancestor node, some common supernodes may contain fewer transactions and instead of moving objects to those common supernodes, moving the transactions upward may further decrease the total communication cost.

For example, let u and v be the two inner nodes of a tree rooted at v_{root} such that $u = parent(v)$. Let $= 1$ and $>$. Let sub-tree of node v contains $2 + 1$ transactions where $+ 1$ transactions access two objects $a; b$ and remaining transactions access only object a. Let the sub-tree of node u contains $2 + 1$ number of transactions requiring both objects $a; b$, that is, extra transactions requiring the object b. From Phase 1, since the sub-tree of v contains $2 + 1$ number of transactions accessing object a, it becomes a supernode with respect to a. Since the sub-tree of v contains only number of transactions requiring object b, it cannot be a supernode with respect to b. The sub-tree of node u contains $2 + 1$ number of transactions requiring object b and number of transactions not included in the sub-tree of v requiring object a. Thus u becomes the supernode with respect to both a and b. In Phase 2, the $2 + 1$ number of transactions requiring both objects $a; b$ select u as a common supernode and remaining number of transactions requiring only object a select v as a common supernode. When object a is moved from node u to v and again back to u, the object movement cost is 2. Instead, if we move those number of transactions from v to u and execute at u, transaction movement cost will increase by reducing object movement cost by 2. That means it minimizes the total cost by .

We provide such optimization in Phase 3 of the algorithm.

Phase 3: In this phase, we compute the final set of supernodes FinalSV in G where respective transactions and the required objects are gathered together. From Phase 2, we have a set of common supernodes sv(T) for each transaction T 2 T . For each common supernode v 2 sv(), the following information is maintained separately:

- numtxs(v): total number of transactions that have selected v as a common supernode.
- objs(v): set of objects with respect to which the node v is a supernode.
- txs(v(o_i)); o_i 2 objs(v): set of transactions requiring object o_i that have selected v as the common supernode.

Let P be the pruned tree containing the nodes of G only up to the common supernodes moving down from v_{root}^0. Starting from every leaf node of P towards v_{root}^0, we check at each node how many transactions have selected it as a common supernode. Particularly, if v 2 P is a leaf node in P and is selected as a common supernode with respect to the set of objects objs(v), then, we check if numtxs(v) 2jobjs(v)j. If the condition is satisfied, v belongs to FinalSV with respect to all objects in objs(v). Otherwise, for each object o_i 2 objs(v), we check how many transactions requiring the object o_i have selected v as the common supernode in Phase 2. Let txs(v(o_i)) be the set of transactions requiring object o_i that have selected v as a common supernode. If jtxs(v(o_i))j 2, v belongs to FinalSV. But if jtxs(v(o_i))j < 2, we visit its parent node parent(v), find the set of transactions txs(parent(v)(o_i)) requiring object o_i that have selected parent(v) as the common supernode. At the parent node parent(v), we again check if (jtxs(v(o_i))j + jtxs(parent(v)(o_i))j) 2. If the condition is met, parent(v) belongs to FinalSV and all the transactions in txs(v(o_i)) that previously selected node v as the common supernode now select parent(v) as the common supernode. Otherwise, if the condition is not met, we repeat the same procedure by selecting the parent of parent(v) and so on until the inequality

$$(\text{j}txs(v(o_i))\text{j} + \text{j}txs(parent(v)(o_i))\text{j} + :::) \quad 2$$

is satisfied or reach at v_{root}^c. We apply this approach recursively until at each leaf node v 2 P, numtxs(v) 2 where P is the pruned tree containing nodes only up to final set of common supernodes FinalSV starting from v_{root}^c.

Phase 4: In this phase, we find the transaction execution schedule E and the paths of movement for each object o_i 2 O along their respective supernodes. We find the pruned tree P containing the nodes up to the common supernodes in FinalSV starting from v'_{root}. Then we perform iterative pre-order traversal on P starting from v'_{root}. At each current visited node v, if $v \in$ FinalSV, then all the transactions which have selected v as their common supernode (i.e., $sv(T) = v$) are added to the execution schedule E. Additionally, the objects in O for which v is a common supernode (i.e., objs(v)) are scheduled to move at v. An object $o_k \in objs(v)$ remains at v until all the transactions that require o_k finish their executions. After all the transactions that require object $o_k \in objs(v)$ finish their executions, o_k can move to the next common supernode in the order where other transactions are waiting for it. When the traversal of P completes, all the transactions get scheduled and the algorithm ends.

Theorem 5. Algorithm MultipleObjects-TxMigr provides k-approximation in communication cost, where k is the maximum number of objects accessed by a transaction.

Proof. After computing the final set of common supernodes FinalSV, at the bottom-most common super $v \in$ FinalSV in each branch of G, the number of transactions that require object o are at least 2. These 2 number of transactions in the subtree of v may require k number of objects in O. Thus node v can be a common supernode for all those 2 transactions with respect to k objects. During the execution, these k objects are moved from v_{root} to v and the cost is $k \cdot 2dist(v'_{root}; v)$. Instead, if we move those 2 transactions up towards some closest common supernode v_j that contains at least $k \cdot 2$ number of transactions, then the cost due to transaction migration increases by $2dist(v_j; v)$ reducing the object movement cost by $k \cdot 2 \cdot dist(v_j; v)$. That means the total cost may increase by at most a k factor from optimal.

Arbitrary Initial Positions of Objects. Here, we discuss algorithm MultipleObjects-TxMigr with the relaxed setting where objects are positioned at arbitrary nodes of G initially. In this case, before Phase 1, we compute the virtual root v'_{root} of G using Equation 5. All the objects in O are then moved to v'_{root}. After this, algorithm continues with Phase 1 to Phase 4 as it is. There is an extra cost incurred before Phase 1 due to the movements of objects from their home nodes to the virtual root. Let C_{extra} represents this extra cost due to the movements of objects from their home nodes to v_{root} which is:

$$C_{extra} = \sum_{o_i \in O} dist(home(o_i); v'_{root}) \tag{9}$$

Let FinalSV be the finalized set of common supernodes computed in Phase 3 of algorithm MultipleObjects-TxMigr after moving all objects in O to v'_{root}. Let C_{mov}

be the total cost due to the movements of objects from v^{c}_{root} to their respective common supernodes in FinalSV following the iterative pre-order tree traversal. Now, let $S(o_i)$ be the sets of supernodes computed with respect to each object o_i 2 O positioned at the respective home node and using algorithm Single-Object without optimization. Let $C_{opt \ mov}$ denotes the total cost due to the movements of objects in their respective supernodes in $S(o)$ following iterative pre-order tree traversal. By Theorem 2, we have that $C_{opt \ mov}$ is asymptotically optimal with respect to the objects movement cost. If $C_{extra} + C_{mov} \quad k\, C_{opt \ mov}$ then algorithm MultipleObjects-TxMigr has performance as in Theorem 5 in the relaxed setting as well.

On the other hand, consider a case where objects are already on the respective common supernodes during the initial configuration. Then since there are total objects, from Equation 5, algorithm MultipleObjects-TxMigr provides O(D)-approximation in the relaxed setting where D is the diameter of tree G and $dist(home(o); v_{root}) \quad D.^{c}$

5. Conclusion

In this paper, we studied transaction scheduling problem on trees to minimize communication cost in the dual-flow model where both data and transactions are mobile. When transactions access the same single shared object, we provided an optimal schedule for executing the transactions. Extending it for multiple shared objects, we provided algorithms that are within k factor away from the optimal, where k is the maximum number of shared objects requested by a transaction.

In the future work, it will be interesting to study the dynamic online setting of the scheduling problem in the dual-flow model. It will also be interesting to consider other kinds of graph or extend the algorithms towards the general networks. Consideration of other performance metrics such as execution time and congestion is also a probable future direction.

References

[1] C. Busch, B. S. Chlebus, M. Herlihy, M. Popovic, P. Poudel, G. Sharma, Flexible scheduling of transactional memory on trees, in: Stabilization, Safety, and Security of Distributed Systems - 24th International Symposium, SSS 2022, Clermont-Ferrand, France, November 15-17, 2022, Vol. 13751 of LNCS, Springer, 2022, pp. 146–163.

[2] M. Herlihy, J. E. B. Moss, Transactional memory: Architectural support for lock-free data structures, in: Proceedings of the 20th Annual International Symposium on Computer Architecture, ACM, 1993, pp. 289–300.

[3] N. Shavit, D. Touitou, Software transactional memory, Distributed Computing 10 (2) (1997) 99–116.

[4] P. Hammarlund, A. J. Martinez, A. A. Bajwa, D. L. Hill, E. G. Hallnor, H. Jiang, M. G. Dixon, M. Derr, M. Hunsaker, R. Kumar, R. B. Osborne, R. Rajwar, R. Singhal, R. D'Sa, R. Chappell, S. Kaushik, S. Chennupaty, S. Jourdan, S. Gunther, T. Piazza, T. Burton, Haswell: The fourth-generation Intel core processor, IEEE Micro 34 (2) (2014) 6–20.

[5] R. A. Haring, M. Ohmacht, T. W. Fox, M. Gschwind, D. L. Satterfield, K. Sugavanam, P. Coteus, P. Heidelberger, M. A. Blumrich, R. W. Wisniewski, A. Gara, G. L. Chiu, P. A. Boyle, N. H. Christ, C. Kim, The IBM blue gene/q compute chip, IEEE Micro 32 (2) (2012) 48–60.

[6] T. Nakaike, R. Odaira, M. Gaudet, M. M. Michael, H. Tomari, Quantitative comparison of hardware transactional memory for Blue Gene/Q, zEnterprise EC12, Intel Core, and POWER8, in: Proceedings of the 42nd Annual International Symposium on Computer Architecture, ACM, 2015, pp. 144–157.

[7] H. W. Cain, M. M. Michael, B. Frey, C. May, D. Williams, H. Q. Le, Robust architectural support for transactional memory in the power architecture, in: Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA 2013), ACM, 2013, pp. 225–236.

[8] M. Herlihy, Y. Sun, Distributed transactional memory for metric-space networks, Distributed Computing 20 (3) (2007) 195–208.

[9] G. Sharma, C. Busch, Distributed transactional memory for general networks, Distributed Computing 27 (5) (2014) 329–362.

[10] K. Siek, P. T. Wojciechowski, Atomic RMI: A distributed transactional memory framework, International Journal of Parallel Programming 44 (3) (2016) 598–619.

[11] J. Armstrong, Programming Erlang: Software for a Concurrent World, Pragmatic Bookshelf, 2007.

[12] K. Hwang, J. Dongarra, G. C. Fox, Distributed and Cloud Computing: From Parallel Processing to the Internet of Things, Morgan Kaufmann Publishers, 2011.

[13] P. Ruth, J. Rhee, D. Xu, R. Kennell, S. Goasguen, Autonomic live adaptation of virtual computational environments in a multi-domain infrastructure, in: Proceedings of the 3rd International Conference on Autonomic Computing (ICAC 2006), IEEE Computer Society, 2006, pp. 5–14.

[14] E. Tilevich, Y. Smaragdakis, J-Orchestra: Automatic java application partitioning, in: Proceedings of the 16th European Conference on Object-Oriented Programming (ECOOP 2002), Vol. 2374 of LNCS, Springer, 2002, pp. 178–204.

[15] K. Arnold, R. Scheifler, J. Waldo, B. O'Sullivan, A. Wollrath, Jini Specification, Addison-Wesley Longman Publishing, 1999.

[16] M. M. Saad, B. Ravindran, Snake: Control flow distributed software transactional memory, in: Proceedings of the 13th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2011), Vol. 6976 of LNCS, Springer, 2011, pp. 238–252.

[17] D. E. Comer, The Cloud Computing Book: The Future of Computing Explained, Chapman and Hall/CRC, 2021.

[18] H. Attiya, V. Gramoli, A. Milani, Directory protocols for distributed transactional memory, in: Transactional Memory. Foundations, Algorithms, Tools, and Applications, Vol. 8913 of LNCS, Springer, 2015, pp. 367–391.

[19] C. Busch, M. Herlihy, M. Popovic, G. Sharma, Time-communication impossibility results for distributed transactional memory, Distributed Computing 31 (6) (2018) 471–487.

[20] C. Busch, M. Herlihy, M. Popovic, G. Sharma, Dynamic scheduling in distributed transactional memory, in: Proceedings of the 2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS 2020), IEEE, 2020, pp. 874–883.

[21] C. Busch, M. Herlihy, M. Popovic, G. Sharma, Fast scheduling in distributed transactional memory, Theory of Computing Systems 65 (2) (2021) 296–322.

[22] G. Sharma, C. Busch, A load balanced directory for distributed shared memory objects, Journal of Parallel and Distributed Computing 78 (2015) 6–24.

[23] R. Palmieri, S. Peluso, B. Ravindran, Transaction execution models in partially replicated transactional memory: The case for data-flow and control-flow, in: Transactional Memory. Foundations, Algorithms, Tools, and Applications, Vol. 8913 of LNCS, Springer, 2015, pp. 341–366.

[24] K. Siek, P. T. Wojciechowski, Atomic RMI 2: Highly parallel pessimistic distributed transactional memory, CoRR abs/1606.03928.

[25] M. M. Saad, B. Ravindran, HyFlow: a high performance distributed software transactional memory framework, in: Proceedings of the 20th ACM International Symposium on High Performance Distributed Computing (HPDC 2011), ACM, 2011, pp. 265–266.

[26] R. L. Bocchino Jr., V. S. Adve, B. L. Chamberlain, Software transactional memory for large scale clusters, in: Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP 2008), ACM, 2008, pp. 247–258.

[27] D. Hendler, A. Naiman, S. Peluso, F. Quaglia, P. Romano, A. Suissa, Exploiting locality in lease-based replicated transactional memory via task migration, in: Proceedings of the 27th International Symposium on Distributed Computing (DISC 2013), Vol. 8205 of LNCS, Springer, 2013, pp. 121–133.

[28] B. Zhang, B. Ravindran, R. Palmieri, Distributed transactional contention management as the traveling salesman problem, in: Proceedings of the 21st International Colloquium on Structural Information and Communication Complexity (SIROCCO 2014), Vol. 8576 of LNCS, Springer, 2014, pp. 54–67.

[29] P. Poudel, G. Sharma, GraphTM: An ecient framework for supporting transactional memory in a distributed environment, in: Proceedings of the 21st International Conference on Distributed Computing and Networking (ICDCN 2020), ACM, 2020, pp. 11:1–11:10.

[30] S. Rai, G. Sharma, C. Busch, M. Herlihy, Load balanced distributed directories, in: Proceedings of the 20th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 2018), LNCS, Springer, 2018, pp. 221–238.

[31] M. Couceiro, P. Romano, N. Carvalho, L. E. T. Rodrigues, D2STM: dependable distributed software transactional memory, in: Proceedings of the 15th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2009), IEEE Computer Society, 2009, pp. 307–313.

[32] S. Hirve, R. Palmieri, B. Ravindran, Hipertm: High performance, fault-tolerant transactional memory, Theoretical Computer Science 688 (2017) 86–102.

[33] T. Kobus, M. Kokocinski, P. T. Wojciechowski, Hybrid replication: State-machine-based and deferred-update replication schemes combined, in: Proceedings of the IEEE 33rd International Conference on Distributed Computing Systems (ICDCS 2013), IEEE Computer Society, 2013, pp. 286–296.

[34] K. Manassiev, M. Mihailescu, C. Amza, Exploiting distributed version concurrency in a transactional memory cluster, in: Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP 2006), ACM, 2006, pp. 198–208.

[35] S. Peluso, P. Romano, F. Quaglia, SCORe: A scalable one-copy serializable partial replication protocol, in: Proceedings of ACM/IFIP/USENIX 13th International Middleware Conference (Middleware 2012), Vol. 7662 of LNCS, Springer, 2012, pp. 456–475.

[36] S. Peluso, P. Ruivo, P. Romano, F. Quaglia, L. E. T. Rodrigues, When scalability meets consistency: Genuine multiversion update-serializable partial data replication, in: Proceedings of the 2012 IEEE 32nd International Conference on Distributed Computing Systems, IEEE Computer Society, 2012, pp. 455–465.

[37] C. Kotselidis, M. Ansari, K. Jarvis, M. Luján, C. C. Kirkham, I. Watson, DiSTM: A software transactional memory framework for clusters, in: Proceedings of the 2008 International Conference on Parallel Processing (ICPP 2008), IEEE Computer Society, 2008, pp. 51–58.

[38] M. Herlihy, V. Luchangco, M. Moir, A flexible framework for implementing software transactional memory, in: Proceedings of the 21th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2006), ACM, 2006, pp. 253–262.

[39] H. Attiya, L. Epstein, H. Shachnai, T. Tamir, Transactional contention management as a non-clairvoyant scheduling problem, Algorithmica 57 (1) (2010) 44–61.

[40] A. Dragojevic, R. Guerraoui, A. V. Singh, V. Singh, Preventing versus curing: avoiding conflicts in transactional memories, in: Proceedings of the 28th Annual ACM Symposium on Principles of Distributed Computing, (PODC 2009), ACM, 2009, pp. 7–16.

[41] R. Guerraoui, M. Herlihy, B. Pochon, Toward a theory of transactional contention managers, in: Proceedings of the Twenty-Fourth Annual ACM Symposium on Principles of Distributed Computing (PODC 2005), ACM, 2005, pp. 258–264.

[42] G. Sharma, C. Busch, A competitive analysis for balanced transactional memory workloads, Algorithmica 63 (1-2) (2012) 296–322.

[43] G. Sharma, C. Busch, Window-based greedy contention management for transactional memory: theory and practice, Distributed Computing 25 (3) (2012) 225–248.

[44] R. M. Yoo, H. S. Lee, Adaptive transaction scheduling for transactional memory systems, in: Proceedings of the 20th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA 2008), ACM, 2008, pp. 169–178.

[45] A. Baldassin, R. Murari, J. P. L. de Carvalho, G. Araujo, D. Castro, J. Barreto, P. Romano, NV-PhTM: an ecient phase-based transactional system for non-volatile memory, in: Proceedings of the 26th International Conference on Parallel and Distributed Computing (Euro-Par 2020), Vol. 12247 of LNCS, Springer, 2020, pp. 477–492.

[46] M. Liu, M. Zhang, K. Chen, X. Qian, Y. Wu, W. Zheng, J. Ren, DudeTM: building durable transactions with decoupling for persistent memory, in: Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2017), ACM, 2017, pp. 329–343.

[47] A. Kolli, S. Pelley, A. G. Saidi, P. M. Chen, T. F. Wenisch, High-performance transactions for persistent memories, in: Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2016), ACM, 2016, pp. 399–411.