Flexible Scheduling of Transactional Memory on Trees

Costas Busch[1], Bogdan S. Chlebus[1], Maurice Herlihy[2], Miroslav Popovic[3], Pavan Poudel[1], and Gokarna Sharma[4][0000−0002−4930−4609]

[1] Augusta University, Augusta, Georgia, USA
{kbusch,bchlebus,ppoudel}@augusta.edu
[2] Brown University, Providence, Rhode Island, USA
herlihy@cs.brown.edu
[3] University of Novi Sad, Novi Sad, Serbia
miroslav.popovic@rt-rk.uns.ac.rs
[4] Kent State University, Kent, OH, USA
gsharma2@kent.edu

Abstract. We study the efficiency of executing transactions in a distributed transactional memory system. The system is modeled as a wired 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 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 they execute. When all the transactions use a single object, we give an offline 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.

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 [10] and Shavit and Touitou [25]. The idea is to designate blocks of program code as *transactions* to be executed atomically. 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 effects 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 [11], Sharma and Busch [23], and Siek and Wojciechowski [26]. The efficiency 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 efficiency 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 [1]. A job management system for a computer cluster may migrate a job to a different 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. [13]. A related system that uses live virtual machine migration to support autonomic adaptation of virtual computation environments is described by Ruth et al. [20].

Coordinating accessing objects to execute transactions may involve relocation of objects or transactions. Efficiency 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 [27] and Herlihy and Sun [11]. 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. [2] and Saad and Ravindran [22].

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. We consider distributed systems whose networks interpreted as graphs have 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 [8].

We study the efficiency of executing transactions by a distributed system represented as a tree in the dual-flow model. The efficiency 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 offline 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.

Related work. Attiya et al. [3], Busch et al. [5–7], and Sharma and Busch [23, 24] considered transaction scheduling with provable performance bounds in the data-flow model. Saad and Ravindran [22], Palmieri et al. [17], Siek and Wojciechowski [26] studied scheduling transactions in the control-flow model. Palmieri et al. [17] 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 [21] supports experimentation for both data-flow and control-flow models. Bocchino et al. [4] 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. [9] 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. [5] 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 offline setting. Specifically, Busch et al. [5] demonstrated a tradeoff between minimizing execution time and communication cost and provided offline algorithms optimizing execution time and communication cost separately. Busch et al. [7] considered transaction scheduling tailored to specific popular topologies and provided offline algorithms that minimize simultaneously execution time and communication cost. In a follow-up work, Poudel and Sharma [19] provided an evaluation framework for processing transactions in distributed systems. Busch et al. [6] studied online algorithms to schedule transactions arriving continuously. Distributed directory protocols have been designed by Herlihy and Sun [11], Sharma and Busch [23], and Zhang et al. [28], with the goal to optimize communication cost in scheduling transactions.

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 Hirve et al. [12], Kim and Ravindran [14], Kobus et al. [15], Manassiev et al. [16], and Peluso et al. [18]. 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.

2 Technical Preliminaries

A distributed system can be modeled as weighted graph $G = (V, E, \mathrm{w})$ which in our case is a tree. 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 $\mathrm{w} : E \to \mathbb{Z}^+$ assigns a weight to each edge representing a communication delay. We let $\mathrm{dist}(u, v)$ 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 different 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 shared objects, transaction code and control messages.

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 α over a unit weight edge is denoted by α. We denote the cost of sending a control message over a unit weight edge by β. The cost of moving a transaction over a unit weight edge is denoted by γ.

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. Formally, a *schedule* of executing transactions is a sequence of actions $s_1, s_2, \ldots$ to be performed by the nodes. An *action* s_i is a set of instructions to be performed by a node to facilitate processing transaction 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 α, β, and γ.

3 A Single Object

We assume a single shared object o of size $\alpha > 1$ positioned at the root node of a tree G. 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.

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 $\mathcal{E}$ as soon as o reaches the respective node. When o reaches some supernode, the transactions from its local execution schedule get added to $\mathcal{E}$ in the respective order. The execution ends when all the transactions have been added to $\mathcal{E}$. The algorithm can be modified as follows if performed in the transaction-migration technique. Determine supernodes with respect to transaction migration cost rather than control messages cost. 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.

We elaborate on the details of the algorithm next. The cost of moving o over an edge of unit length is α. Let β represents the control message cost for a transaction to access object o at one unit away and $\alpha > \beta$. Let $\mathcal{T} = \{T_1, T_2, \ldots T_n\}$ 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 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 send control messages to the supernode to access the object. Then the object moves to the next supernode and 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 loads and transaction counts at all nodes, which are defined as follows. 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. 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. 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.

To identify supernodes, we start from the leaves of G and work through the ancestors towards the root. 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$(v_{cur}) + \alpha \cdot \mathrm{dist}(v_{cur}, v_{next})$. Here, $\alpha \cdot \mathrm{dist}(v_{cur}, v_{next})$ 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$(v_{cur}) \cdot \beta \cdot \mathrm{dist}(v_{cur}, v_{next})$. 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}. 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. If the following inequality holds

$$\text{txload}(v_{cur}) + 2\alpha \cdot \mathrm{dist}(v_{cur}, v_{next}) \leq \text{txload}(v_{cur}) + \text{txnum}(v_{cur}) \cdot \beta \cdot \mathrm{dist}(v_{cur}, v_{next}) \,,$$

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.

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 the root of G. Figure 1 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 the nodes beyond the pruned tree P, marked by color orange in Figure 1, 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.

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 $\mathcal{E}$ is computed as follows. First add transaction at the root to $\mathcal{E}$. During the pre-order tree traversal to visit the supernodes, if $\mathcal{E}$ does not contain the transaction at a visited node v, then add it to $\mathcal{E}$. If the visited node v is a supernode, add to $\mathcal{E}$ the transactions that sent control messages to v from the subtree rooted at v.

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. 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

$$2\alpha \cdot \mathrm{dist}(v_{cur}, v_{next}) > \text{txnum}(v_{cur}) \cdot \beta \cdot \mathrm{dist}(v_{cur}, v_{next}) \tag{1}$$

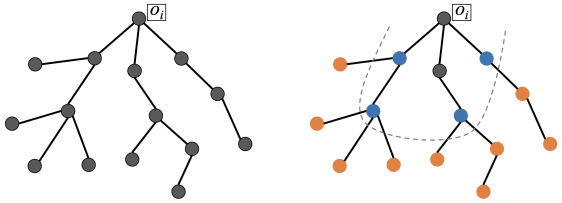

Fig. 1. Identification of supernodes by algorithm SINGLE-OBJECT. The tree on the left is G. The tree on the right is the same G after determining the status of nodes. Supernodes are colored blue. Nodes on the path from the root to a blue node are colored black. The dashed line delineates P obtained from G by pruning G of vertices beyond the supernodes, which are colored orange.

so it 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

$$\alpha \cdot \text{dist}(v_{cur}, v_{next}) > \text{txnum}(v_{cur}) \cdot \beta \cdot \text{dist}(v_{cur}, v_{next}) \tag{2}$$

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 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 $v \in S$, 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 $v \in S$, we find the set of nodes $D(v)$ that are candidates for v_{last} using the condition in Inequality (2). Then the difference between the cost of selecting v as a supernode and $v_j \in D(v)$ as a supernode is computed. Among these differences for every $v \in S$, the one with the highest difference is chosen as the last supernode. Let $v_{ref} \in S$ and $v_k \in D(v_{ref})$ be the set of two nodes that provided the highest difference. 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.

We state following three lemmas whose proofs are immediate from the discussion:

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 2α.*

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 α.*

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.*

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, $\beta = 1$ and $\alpha > \beta$. Let P be the pruned tree containing nodes only up to the supernodes starting from the root of G. Let $v_{last} \in S$ be the last supernode for object o to visit. Let C be the total communication cost of Algorithm SINGLE-OBJECT. Let $v \in S$ be a supernode in G, v_p be an ancestor of v with distance $\mathrm{dist}(v_p, v) \geq 1$, and v_q be a descendant of v with $\mathrm{dist}(v, v_q) \geq 1$. 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

$$\text{txnum}(v_p) \geq \text{txnum}(v) \geq \alpha > \text{txnum}(v_q) \tag{3}$$

Case (b): $v \neq v_{last}$, $v_q \notin P$, 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

$$\text{txnum}(v_p) \geq \text{txnum}(v) \geq 2\alpha > \text{txnum}(v_q) \tag{4}$$

Case (c): Either $v_q \in P$ or $v_q \notin P$ 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:

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

In Case (a), from Inequality (3), since $\text{txnum}(v) \geq \alpha$, $C_{v_p} > C$. In Case (b), from Inequality (4), since $\text{txnum}(v) \geq 2\alpha$, $C_{v_p} > C$. 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:

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

Let $\text{dist}(v, v_q) = k$ where $k \geq 1$. In Case (a), from Inequality (3), $\text{txnum}(v_q) < \alpha$. Let $\text{txnum}(v_q) = \alpha - j$, $1 \leq j < \alpha$. 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: $\sum_{v_k \in S_q} (\text{txnum}(v_k) - \text{txnum}(v_q)) < j \cdot k$. Thus,

$$C_{v_q} > C + \alpha \cdot k - (\alpha - j) \cdot k - j \cdot k > C \ .$$

In Case (b), $\text{txnum}(v_q) < 2\alpha$ by the Inequality (4). Let $\text{txnum}(v_q) = 2\alpha - l$, for $1 \leq l < 2\alpha$. 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: $\sum_{v_k \in S_q} (\text{txnum}(v_k) - \text{txnum}(v_q)) < l \cdot k$. Thus

$$C_{v_q} > C + 2\alpha \cdot k - (2\alpha - l) \cdot k - l \cdot k > C \ .$$

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

Case (c.1): $v_q \in P$. There is no extra movement of o and the $z \geq 1$ 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: $C_{v_q} = C + z \cdot \text{dist}(v, v_q) > C$.

Case (c.2): $v_q \notin P$ 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 $v_{bot} \in S$ 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, $M > \alpha \cdot \text{dist}(v_{bot}, v_q)$. Thus, the total communication cost C_{v_q} compared to C in this case becomes: $C_{v_q} = C + z \cdot \text{dist}(v, v_q) + M > C$.

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

Consider two supernodes $v_r, v_s \in S$ 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 cost:

Consider a supernode $v_j \in S$. Let v_x, v_z be two descendant nodes of v_j at two different sub-branches. Let v_x and v_z are chosen as two different 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. This means that scheduling by algorithm SINGLE-OBJECT minimizes the communication cost. □

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

Theorem 2. *Algorithm* SINGLE-OBJECT *provides 2-approximation in communication cost without optimization.*

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.

We consider a set of shared objects $\mathcal{O} = \{o_1, o_2, \ldots, o_\delta\}$ initially positioned at arbitrary nodes of G. We assume that each object has size α. Each transaction in $\mathcal{T}$ accesses a subset of objects in $\mathcal{O}$. Let objs$(T_i) \subseteq \mathcal{O}$ 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 different 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,

$$v'_{root} = v_i : W(v_i) = \min_{v \in V} W(v), \tag{5}$$

where $$W(v) = \sum_{j=1}^{n} \big(\text{dist}(v, \text{home}(T_j)) + \sum_{o \in \text{objs}(T_j)} \text{dist}(v, \text{home}(o))\big).$$

Multiple Objects with Control Messages. The algorithm for the control-message technique is named MULTIPLEOBJECTS-CTRLMSG. The algorithm runs in two phases.

Phase 1: We compute sets of supernodes $S(o_i)$ w.r.t. each object $o_i \in \mathcal{O}$ 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 $\mathcal{E}$ and paths of movement for each object $o_i \in \mathcal{O}$ along their respective supernodes. For this, let a random node in G be selected as the virtual root v'_{root} of G. We perform an iterative pre-order tree traversal in G starting from v'_{root}. During the traversal, if there is a transaction T_j at current node v_{cur}, T_j is added to the schedule $\mathcal{E}$ and 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'_{root} at most three times.*

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 $o_i \in \mathcal{O}$ 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 $\mathcal{T}$, then,

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

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

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 $\mathcal{T}$, then,

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

Equations (6) and (7) imply

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

This gives the estimate

$$C_{ALG} = \sum_{o_i \in \mathcal{O}} C_{ALG}(o_i) \leq \sum_{o_i \in \mathcal{O}} (3 \cdot C_{OPT}(o_i)) \leq 3 \cdot C_{OPT} \ ,$$

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. □

Multiple Objects with Migration of Transactions. The algorithm for multiple objects implemented in the transaction-migration technique is named MULTIPLEOBJECTS-TXMIGR. First, we discuss the algorithm assuming all the objects are initially positioned at the same node, the virtual root v'_{root}, of G. Later, we relax the algorithm where objects can be positioned initially at arbitrary nodes in G.

The algorithm works in four phases. In Phase 1, we compute sets of supernodes with respect to individual object $o_i \in \mathcal{O}$. In Phase 2, we find a common supernode for each transaction $T \in \mathcal{T}$ 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 common supernodes. We describe each phase below.

Phase 1: In this phase, we compute supernodes with respect to each object $o_i \in \mathcal{O}$ using algorithm SINGLE-OBJECT without optimization where control message cost β over an edge is replaced with the transaction migration cost γ. Let $S(o_i)$ be the set of supernodes with respect to object $o_i \in \mathcal{O}$ and $sv(T(o_i)) \in S(o_i)$ represents the supernode for transaction T at which T accesses o_i. After this, each transaction $T_j \in \mathcal{T}$ has a set of respective supernodes $sv(T_j(o_i))$ to access each required object $o_i \in$ 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.

Phase 2: In this phase, we find a common supernode of objects $sv(T)$ for each transaction $T \in \mathcal{T}$. 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 $sv(T(o_i))$, $o_i \in$ objs(T), computed in Phase 1 are the same, it automatically becomes the common supernode for T. If they are different, then we select the one among $sv(T(o_i)), o_i \in$ objs(T), which is the closest from v'_{root}.

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 $T \in \mathcal{T}$. For each common supernode $v \in sv(*)$, following information is maintained separately:

- numtxs(v): total number of transactions that selected v as a common supernode.
- objs(v): set of objects with respect to which the node v is a supernode.
- txs($v(o_i)$), $o_i \in$ 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}. Starting from every leaf node of P towards v'_{root}, we check at each node how many transactions have selected it as a common supernode. Particularly, if $v \in P$ 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) $\cdot\, \gamma \geq 2\alpha \cdot |$objs($v$)$|$. If the condition is satisfied, v belongs to FinalSV with respect to all objects in objs(v). Otherwise, for each object $o_i \in$ 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 $|\text{txs}(v(o_i))| \cdot \gamma \geq 2\alpha$, v belongs to FinalSV. But if $|\text{txs}(v(o_i))| \cdot \gamma < 2\alpha$, 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 $(|\text{txs}(v(o_i))| + |\text{txs}(parent(v)(o_i))|) \cdot \gamma \geq 2\alpha$. 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

$$(|\text{txs}(v(o_i))| + |\text{txs}(parent(v)(o_i))| + \dots) \cdot \gamma \geq 2\alpha$$

is satisfied or reach at v'_{root}. We apply this approach recursively until at each leaf node $v \in P$, numtxs(v) $\cdot\, \gamma \geq 2\alpha$ where P is the pruned tree containing nodes only up to final set of common supernodes FinalSV starting from v'_{root}.

Phase 4: In this phase, we find the transaction execution schedule $\mathcal{E}$ and the paths of movement for each object $o_i \in \mathcal{O}$ 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 $\mathcal{E}$. Additionally, the objects in $\mathcal{O}$ 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 4. *Algorithm* MULTIPLEOBJECTS-TXMIGR *provides k-approximation in communication cost, where k is the maximum number of objects a transaction accesses.*

Proof. After computing the final set of common supernodes FinalSV, at the bottommost common super $v \in$ FinalSV in each branch of G, the number of transactions that require object o are at least 2α. These 2α number of transactions in the sub-tree of v may require

$k \leq \delta$ number of objects in $\mathcal{O}$. Thus node v can be a common supernode for all those 2α transactions with respect to $k \leq \delta$ objects. During the execution, these k objects are moved from v'_{root} to v and the cost is $k \cdot 2\alpha \cdot \text{dist}(v'_{root}, v)$. Instead, if we move those 2α transactions up towards some closest common supernode v_j that contains at least $k \cdot 2\alpha$ number of transactions, then the cost due to transaction migration increases by $2\alpha \cdot \text{dist}(v_j, v)$ reducing the object movement cost by $k \cdot 2\alpha \cdot \text{dist}(v_j, v)$. That means the total cost may increase by at most a k factor from optimal. □

Arbitrary Initial Positions of Objects. We discuss algorithm MULTIPLEOBJECTS-TXMIGR with the relaxed setting where objects are located 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 $\mathcal{O}$ 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 cost due to the movements of objects from their home nodes to v_{root} which is:

$$C_{extra} = \sum_{o_i \in \mathcal{O}} \alpha \cdot \text{dist}(\text{home}(o_i), v'_{root}) \tag{9}$$

Let FinalSV be the finalized set of common supernodes computed in Phase 3 of algorithm MULTIPLEOBJECTS-TXMIGR after moving all objects in $\mathcal{O}$ to v'_{root}. Let C_{mov} be the total cost due to the movements of objects from v'_{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 $o_i \in \mathcal{O}$ 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 $C_{extra} + C_{mov} \leq k \cdot C_{opt-mov}$, then algorithm MULTIPLEOBJECTS-TXMIGR has performance as in Theorem 4 in the relaxed setting as well. Otherwise, by Equation 5, it provides $O(\alpha \cdot k \cdot D)$-approximation in the relaxed setting because of the bound $\text{dist}(\text{home}(o), v'_{root}) \leq D$, where D is the diameter of tree G.

Acknowledgements G. Sharma was supported by National Science Foundation under Grant No. CAREER CNS-2045597.

References

1. Armstrong, J.: Programming Erlang: Software for a Concurrent World. Pragmatic Bookshelf (2007)
2. Arnold, K., Scheifler, R., Waldo, J., O'Sullivan, B., Wollrath, A.: Jini Specification. Addison-Wesley Longman Publishing (1999)
3. Attiya, H., Gramoli, V., Milani, A.: Directory protocols for distributed transactional memory. In: Transactional Memory. Foundations, Algorithms, Tools, and Applications, Lecture Notes in Computer Science, vol. 8913, pp. 367–391. Springer (2015)
4. Bocchino Jr., R.L., Adve, V.S., Chamberlain, B.L.: Software transactional memory for large scale clusters. In: PPOPP. pp. 247–258. ACM (2008)
5. Busch, C., Herlihy, M., Popovic, M., Sharma, G.: Time-communication impossibility results for distributed transactional memory. Distributed Computing **31**(6), 471–487 (2018)

6. Busch, C., Herlihy, M., Popovic, M., Sharma, G.: Dynamic scheduling in distributed transactional memory. In: IPDPS. pp. 874–883. IEEE (2020)
7. Busch, C., Herlihy, M., Popovic, M., Sharma, G.: Fast scheduling in distributed transactional memory. Theory of Computing Systems **65**(2), 296–322 (2021)
8. Comer, D.E.: The Cloud Computing Book: The Future of Computing Explained. Chapman and Hall/CRC (2021)
9. Hendler, D., Naiman, A., Peluso, S., Quaglia, F., Romano, P., Suissa, A.: Exploiting locality in lease-based replicated transactional memory via task migration. In: DISC. Lecture Notes in Computer Science, vol. 8205, pp. 121–133. Springer (2013)
10. Herlihy, M., Moss, J.E.B.: Transactional memory: Architectural support for lock-free data structures. In: ISCA. pp. 289–300. ACM (1993)
11. Herlihy, M., Sun, Y.: Distributed transactional memory for metric-space networks. Distributed Computing **20**(3), 195–208 (2007)
12. Hirve, S., Palmieri, R., Ravindran, B.: Hipertm: High performance, fault-tolerant transactional memory. Theoretical Computer Science **688**, 86–102 (2017)
13. Hwang, K., Dongarra, J., Fox, G.C.: Distributed and Cloud Computing: From Parallel Processing to the Internet of Things. Morgan Kaufmann Publishers (2011)
14. Kim, J., Ravindran, B.: Scheduling transactions in replicated distributed software transactional memory. In: CCGrid. pp. 227–234. IEEE Computer Society (2013)
15. Kobus, T., Kokocinski, M., Wojciechowski, P.T.: Hybrid replication: State-machine-based and deferred-update replication schemes combined. In: ICDCS. pp. 286–296. IEEE Computer Society (2013)
16. Manassiev, K., Mihailescu, M., Amza, C.: Exploiting distributed version concurrency in a transactional memory cluster. In: PPOPP. pp. 198–208. ACM (2006)
17. Palmieri, R., Peluso, S., Ravindran, B.: 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, pp. 341–366. Springer (2015)
18. Peluso, S., Ruivo, P., Romano, P., Quaglia, F., Rodrigues, L.E.T.: When scalability meets consistency: Genuine multiversion update-serializable partial data replication. In: ICDCS. pp. 455–465. IEEE Computer Society (2012)
19. Poudel, P., Sharma, G.: GraphTM: An efficient framework for supporting transactional memory in a distributed environment. In: ICDCN. pp. 11:1–11:10. ACM (2020)
20. Ruth, P., Rhee, J., Xu, D., Kennell, R., Goasguen, S.: Autonomic live adaptation of virtual computational environments in a multi-domain infrastructure. In: ICAC. pp. 5–14. IEEE Computer Society (2006)
21. Saad, M.M., Ravindran, B.: HyFlow: a high performance distributed software transactional memory framework. In: HPDC. pp. 265–266. ACM (2011)
22. Saad, M.M., Ravindran, B.: Snake: Control flow distributed software transactional memory. In: SSS. Lecture Notes in Computer Science, vol. 6976, pp. 238–252. Springer (2011)
23. Sharma, G., Busch, C.: Distributed transactional memory for general networks. Distributed Computing **27**(5), 329–362 (2014)
24. Sharma, G., Busch, C.: A load balanced directory for distributed shared memory objects. Journal of Parallel and Distributed Computing **78**, 6–24 (2015)
25. Shavit, N., Touitou, D.: Software transactional memory. Distributed Computing **10**(2), 99–116 (1997)
26. Siek, K., Wojciechowski, P.T.: Atomic RMI: A distributed transactional memory framework. International Journal of Parallel Programming **44**(3), 598–619 (2016)
27. Tilevich, E., Smaragdakis, Y.: J-Orchestra: Automatic java application partitioning. In: ECOOP. Lecture Notes in Computer Science, vol. 2374, pp. 178–204. Springer (2002)
28. Zhang, B., Ravindran, B., Palmieri, R.: Distributed transactional contention management as the traveling salesman problem. In: SIROCCO. vol. 8576, pp. 54–67. Springer (2014)