Fast Scheduling in Distributed Transactional Memory

Costas Busch
Louisiana State University
Baton Rouge, LA
busch@csc.lsu.edu

Maurice Herlihy
Brown University
Providence, RI
herlihy@cs.brown.edu

Miroslav Popovic
University of Novi Sad
Novi Sad, Serbia
miroslav.popovic@rt-rk.uns.ac.rs

Gokarna Sharma
Kent State University
Kent, OH
sharma@cs.kent.edu

ABSTRACT

We investigate scheduling algorithms for distributed transactional memory systems where transactions residing at nodes of a communication graph operate on shared, mobile objects. A transaction requests the objects it needs, executes once those objects have been assembled, and then possibly forwards those objects to other waiting transactions. Minimizing execution time in this model is known to be NP-hard for arbitrary communication graphs, and also hard to approximate within any factor smaller than the size of the graph. Nevertheless, networks on chips, multi-core systems, and clusters are not arbitrary. Here, we explore efficient execution schedules in specialized graphs likely to arise in practice: Clique, Line, Grid, Cluster, Hypercube, Butterfly, and Star. In most cases, when individual transactions request k objects, we obtain solutions close to a factor $O(k)$ from optimal, yielding near-optimal solutions for constant k. These execution times approximate the TSP tour lengths of the objects in the graph. We show that for general networks, even for two objects ($k = 2$), it is impossible to obtain execution time close to the objects' optimal TSP tour lengths, which is why it is useful to consider more realistic network models. To our knowledge, this is the first attempt to obtain provably fast schedules for distributed transactional memory.

CCS CONCEPTS

•Theory of computation → Design and analysis of algorithms; Distributed algorithms; •Computer communication networks → Network architecture and design; Distributed networks;

KEYWORDS

Transactional memory; distributed systems; execution time; approximation; data-flow model; scheduling; contention

1 INTRODUCTION

Concurrent processes (threads) need to synchronize to avoid introducing inconsistencies in shared data objects. Traditional synchronization mechanisms such as locks and barriers have well-known limitations and pitfalls, including deadlock, priority inversion, reliance on programmer conventions, and vulnerability to failure or delay. *Transactional memory* [15, 34] (TM) has emerged as an alternative. Using TM, code is split into *transactions*, blocks of code that appear to execute atomically with respect to one another. Transactions are executed *speculatively*: synchronization conflicts or failures may cause an executing transaction to *abort*: its effects are rolled back and the transaction is restarted. In the absence of conflicts or failures, a transaction typically *commits*, causing its effects to become visible.

Several commercial processors provide direct hardware support for TM, including Intel's Haswell [18] and IBM's Blue Gene/Q [13], zEnterprise EC12 [25], and Power8 [4]. There are proposals for adapting TM to clusters of GPUs [2, 11, 23]. Here, we consider *distributed* TM systems appropriate for rack-scale or cluster-scale networks of nodes linked by a modern communication network [2, 16, 23, 33, 36].

We consider a *data-flow* of transaction execution [16, 33], in which each transaction executes at a single node, but data objects are mobile. A transaction initially requests the objects it needs, and executes only after it has assembled them. When the transaction commits, it releases its objects, possibly forwarding them to other waiting transactions.

In a distributed TM, execution time is dominated by the costs of moving objects from one transaction to another. The goal of a *transaction scheduling algorithm* (sometimes called *contention management*) is to minimize delays caused by data conflicts and data movement.

Here, we consider scheduling algorithms in a synchronous data-flow model where time is divided into discrete steps [3]. The network is modeled as a weighted graph G, where transactions reside at nodes, edges are communication links, and edge weights are communication delays. At any time step, a node may perform three actions: (1) it may receive objects from adjacent nodes, (2) it executes any transaction that has assembled its required objects, and (3) it may forward objects to adjacent nodes. A transaction's execution step models when it commits. That transaction may have started earlier, but may have been blocked while assembing the objects it needed.

We provide offline algorithms to compute conflict-free schedules. We consider batch problem instances where transactions and objects initially reside at different nodes of G. Each node has a single transaction and each object has a single copy. The objective is to minimize the total execution time (makespan) until all transactions complete. The schedule determines the time step when each transaction executes and commits. After a transaction commits, it forwards its objects to any next requesting transactions in the execution order. Typically, an object is sent along a shortest path, implying that the transfer time depends on the distance in G between the sender and receiver nodes. Execution time depends on both the objects' traversal times and on inter-transaction data dependencies.

It is known [3] through a reduction from vertex coloring that determining the shortest execution time in arbitrary graphs is NP-hard, and even hard to approximate within a sub-linear factor of n, the number of nodes in G. Fortunately, however, networks for rack-scale and cluster-scale computing centers are not likely to be arbitrary [7]. Here, we focus on the kinds of specialized networks likely to arise in practice, such as Clique, Line, Grid, Cluster, Hypercube, Butterfly, and Star [5, 10, 22, 24, 27]. For example, the clique graph has implications on the hypercube and butterfly which are typical supercomputer topologies. The line graph represents bus system architectures, for example connecting boards in a rack. The grid graph represents systems on chips or multi-cores (e.g. XMOS architecture, Intel Xeon Phi). The cluster graph is an abstraction of clusters of computers found in data centers. Star graphs correspond to hubs, multiplexer, concentrators, and switches, which are normally used on supercomputers, clusters, and data centers.

1.1 Contributions

Suppose we have a set of w shared objects $O = \{o_1, \ldots, o_w\}$. We consider scheduling problems where each node holds a single transaction, and each transaction requests k objects (out of w). In most of the problems that we study, a transaction's set of k objects is chosen arbitrarily. We make two kinds of contributions.

Lower Bounds. A trivial lower bound for the execution time of the transactions is the longest shortest path that any object has to follow. This path length is within a constant factor from the optimal TSP tour length of the object. Using the probabilistic method, we demonstrate the existence of a scheduling problem on the grid, with 2 objects per transaction, where every schedule must have execution time $\Omega(n^{1/40}/\log n)$ factor away from the optimal TSP tour length of any object. The same lower bound holds also for trees. Therefore, we cannot expect in general to compute schedules that respect the optimal TSP length of any object. Nevertheless, the specialized graphs and scheduling problems considered here admit approximations that beat this lower bound.

Upper Bounds. We give polynomial time algorithms that compute execution schedules for a variety of graphs: Clique, Line, Grid, Cluster, Hypercube, Butterfly, and Star. These are the kinds of graphs one would expect to find in multiprocessor systems, Networks-on-chip (NoCs), or rack-scale or cluster-scale distributed systems [5, 7, 10, 22, 24, 27]. For one transaction per node requesting k objects out of w, we obtain the following results:

Clique: In any clique (complete graph) of n nodes there is a schedule which is within a factor $O(k)$ from optimal.

Line: In any line graph of n nodes there is a schedule which is within a factor $O(1)$ from optimal (asymptotically optimal).

Grid: In a $n \times n$ grid where each transaction requests a random set of k objects, we show that with high probability there is a schedule which is within a factor $O(k \log m)$ from optimal, where $m = \max(n, w)$.

Cluster: We consider a cluster graph which consists of cliques with β nodes connected to each other through bridge edges of weight $\gamma \geq \beta$. We show that there is a schedule which is a factor $O(\min(k\beta, \log_c^k m))$, for some constant c.

In the Hypercube and Butterfly graphs the results are extensions of the results in cliques scaled by a $\log n$ factor, since a shortest path connecting two nodes has length $O(\log n)$, instead of 1 in cliques. We also consider the Star graph topology where there is a central node that connects rays each consisting of β nodes. We observe that the analysis of the Star graph has many similarities with the Cluster and Line graphs and we show that there is a schedule which is a factor $O(\log \beta \cdot \min(k\beta, \log_c^k m))$ from optimal, for a constant c.

When k is a constant, in all graph cases we either obtain asymptotically optimal schedules or we obtain schedules within a poly-log factor from optimal. In most cases, with the only exception of the Grid, for the input problem we assume that each transaction holds an arbitrary set of k objects. In the Grid, a transaction holds a randomly-chosen set of k objects, and the reason for doing this is that the TSP lower bound on the Grid prohibits good schedules for arbitrary input problems, even when $k = 2$.

The main approach for computing the schedules is to appropriately apply a greedy schedule which colors the dependency graph of the transactions, where each color represents a different time step. The result in the Clique is a direct application of the greedy schedule. The result in the Grid is a repeated application of the greedy schedule in subgrid graphs carefully chosen such that the greedy schedule within each of them is efficient. The result in the Cluster graph is an application of the greedy schedule within the constituent cliques and also carefully synchronizing the object movements between the cliques. In all cases the resulting approximation factor compares the execution time to the TSP object tour lengths.

Our results for the data-flow model also apply to restricted versions of other models where objects may be replicated or versioned (Section 1.2).

1.2 Related Work

Transaction scheduling problems are widely studied in (tightly-coupled) multi-core systems. Several scheduling algorithms with provable upper and lower bounds, and impossibility results are given [1, 12, 31, 32], besides several other scheduling algorithms that are evaluated only experimentally [17, 35]. Dragojevic *et al.* [9], provide some theoretical evaluation of conflict prediction for online schedulers and also an experimental algorithm. These scheduling algorithms, however, are not suitable for scheduling in distributed TMs as they do not typically deal with communication cost in accessing shared resources.

Several researchers [2, 8, 23] present techniques to implement distributed TMs. However, they either use global lock, serialization lease, or commit-time broadcasting technique which may not scale well with the size of the network. Moreover, they do not provide formal analysis of either the execution time or the communication cost.

Most of the previous work [16, 19, 33, 36] on the data-flow model of distributed TMs focused on minimizing only the *communication cost* – the total distance traversed by all the objects in G. However, these works studied communication cost for scheduling problem instances with only a single shared object. Kim and Ravindran [19] provided communication cost bounds for special workloads and problem instances with multiple shared objects. The execution time minimization is considered by Zhang *et al.* [36]. However, the graph topology G they considered is arbitrary, except for the assumption of the known diameter D. Therefore, their result is not suitable for the specialized networks we study in this paper. Moreover, they do not study lower bound on execution time whereas we provide for the first time an execution time lower bound, improving significantly on the known TSP lower bound, even for the instances with only two shared objects. Busch *et al.* [3] considered minimizing both the execution time and communication cost. They showed that it is impossible to simultaneously minimize execution time and communication cost, that is, minimizing execution time implies high communication cost (and vice-versa).

There are distributed TM proposals that employ replication and multi-versioning [23, 28]. In replicated TMs, multiple copies are available for each shared object, whereas multiple versions of each object are available in multi-versioning TMs [23]. Kim and Ravindran [20] studied transaction scheduling in replicated distributed TMs. In the *control-flow* model [30], objects are immobile and transactions either move to the network nodes where the required objects reside, or invoke remote procedure calls. Hendler *et al.* [14] studied a lease based hybrid (combining data-flow with control-flow) distributed TM 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. Palmieri *et al.* [26] present a comparative study of data-flow versus control-flow models for distributed TMs in partially-replicated environments. Others have studied the speculative transaction execution [29] in replicated environments, and transaction scheduling using consistent snapshots [20, 23, 28, 29] for replicated and multiversioning environments. These works provide no theoretical analysis of execution time or communication cost.

1.3 Roadmap

In Section 2 we give the model and preliminaries, including the basic greedy schedule. We present result for Cliques in Section 3, where we also explain the implications for the Hypercube, Butterfly, and other related graphs. In Section 4 we give the scheduling algorithm for the Line graph, and in Section 5 we give the scheduling algorithm for the Grid graph. We present the result for the Cluster graph in Section 6, and the related result for the Star graph in Section 7. We prove our lower bound on execution time based on the TSP object tours in Section 8. Finally, we give our conclusions in Section 9.

2 MODEL AND PRELIMINARIES

2.1 Distributed TM Model

We assume a synchronous communication model: at each time step a node receives messages, performs a local computation, and then transmits messages to adjacent nodes. The message size is sufficient to convey the information about an object over the network, and there is no limit on the number of messages that may concurrently traverse an edge.

Let $O = \{o_1, \ldots, o_w\}$ denote w shared memory objects. Each object has a value which can be read or modified (written). The shared objects reside at nodes and are mobile, that is, an object can move from node to node in a message. There is a single copy of each object. A transaction T_i is an atomic code sequence executed at a node v_i which requires a set of objects $O(T_i) \subseteq O$. Transaction T_i can finish execution and commit at a specific time step once all the objects it needs have been gathered at v_i. A transaction T_i may modify some of its objects while others may remain unchanged. Initially, an object is at an arbitrary node of G.

Consider a set of m transactions $\mathcal{T} = \{T_1, T_2, \ldots, T_m\}$, where $m \leq n$, with at most one transaction per node. This batch problem setting is similar to the one usually considered for scheduling in multi-core systems [1, 31]. The transactions are distributed across network nodes and at most n transactions execute concurrently. Conflicts among transactions in accessing shared objects are defined in the usual way, where an aborting transaction restarts immediately.

A *scheduling algorithm* $\mathcal{A}$ determines the time step $t(T_i)$ when a transaction executes. The schedule is feasible if the objects that each transaction requests have moved to the transaction's node at time $t(T_i)$. Let $\mathcal{E}$ be an execution schedule based on $\mathcal{A}$.

Definition 2.1 (Execution Time). For a set of transactions $\mathcal{T}$ in graph G, the time of an execution $\mathcal{E}$ is the time elapsed until the last transaction finishes its execution in $\mathcal{E}$. The execution time of scheduling algorithm $\mathcal{A}$ is the maximum time over all possible executions for $\mathcal{T}$.

2.2 Chernoff Bounds

In our analysis, we use the following Chernoff bounds:

LEMMA 2.2 (CHERNOFF BOUNDS). *Let $X_1, \ldots, X_n$ be independent random variables such that $X_i \in \{0, 1\}$, for $1 \leq i \leq n$. Let $X = \sum_{i=1}^{n} X_i$ and let $\mu = E[X]$. Then,*

$$\Pr(X \geq (1+\delta)\mu) \leq e^{-\frac{\delta^2\mu}{3}}, \qquad 0 < \delta < 1, \tag{1}$$

$$\Pr(X \leq (1-\delta)\mu) \leq e^{-\frac{\delta^2\mu}{2}}, \qquad 0 < \delta < 1. \tag{2}$$

2.3 Greedy Schedule

Consider a set of transactions in a graph G. In the *dependency graph* H each node corresponds to a transaction, and an edge between two nodes corresponds a dependency (conflict) that arises when the respective transactions share one or more objects. The weight of an edge in H represents the distance between the respective transactions. We can schedule the transactions in G using a *greedy schedule* which assigns execution times to transactions based on a coloring of H. A valid coloring of H assigns a unique positive integer to each transaction such that two adjacent transactions receive colors which differ by at least the weight of the incident edge that connects them. The colors correspond to the distinct time steps where respective transactions execute.

A lower bound for the execution time schedule is the maximum weight of any edge in H, since some object will require so much time to be transferred from one node to another in G. The weighted degree of a transaction in H is the sum of the weights of the edges adjacent to the transaction. Let Γ denote the maximum weighted degree of all transactions in H. A greedy coloring scheme assigns colors to the transactions of H one after the other, giving the smallest available color. It can be shown that H can be colored with $\Gamma + 1$ colors in polynomial time [3]. Let Δ be the maximum (non weighted) degree of H. Then a $\Gamma + 1$ coloring gives a $\Delta + 1$ factor approximation to the optimal schedule in G, since the maximum edge weight is a lower bound. The $O(\Delta)$ approximation assumes that the objects are initially positioned in the first transaction in the greedy schedule.

3 COMPLETE GRAPH

Scheduling Problem. Consider a unweighted complete graph (clique) G with n nodes where every node is connected to every other node with an edge of weight 1. Every node holds one transaction. There are w objects $O = \{o_1, \ldots, o_w\}$. Each transaction uses an arbitrary subset of k objects, where $1 \leq k \leq w$.

Algorithm and Analysis. For the algorithm, we use the greedy schedule of Section 2.3. The analysis if as follows.

Theorem 3.1 (complete graph). *In the complete graph the greedy schedule gives a $O(k)$ approximation to the optimal schedule.*

Proof. Let A_i denote the set of transactions that use object o_i. Denote $\ell_i = |A_i|$ and $\ell = \max_i \ell_i$. Every transaction uses k objects, and every object is used by at most ℓ transactions. Therefore, the maximum weighted degree in the dependency graph is $k\ell$, and hence the dependency graph can be colored with at most $k\ell + 1$ colors. At the same time, the length of the schedule is at least ℓ, since an object has to visit every transaction that requests it. Consequently, the schedule is an $O(k)$ factor approximation of the optimal. □

3.1 Hypercube and Other Graphs

The result on complete graphs has implications to other graphs as well. In a hypercube graph [21] with n nodes there is a path of length $\log n$ connecting any pair of nodes. Therefore, the hypercube graph can be represented as a complete graph with n nodes where the weight of any edge is between 1 and $\log n$. Thus, similar to the proof of Theorem 3.1, the maximum weighted degree in the dependency graph is $k\ell \log n$, and hence it can be colored with with $O(k\ell \log n)$ colors. Since $\Omega(\ell)$ is a lower bound, this gives a $O(k \log n)$ approximation.

Generalizing, in any graph where the maximum distance between any pair of nodes is d (diameter), the greedy algorithm gives a $O(k\ell d)$ schedule. For each object o_i let χ_i denote the shortest path length that connects all transactions in A_i. Let $\chi = \max_i \chi_i$. Clearly, χ is a lower bound on execution time. Since $\chi \geq \ell$, we get a $O(kd)$ approximation for execution time. In Butterfly networks [21] and $\log n$-dimensional grids [6], $d = O(\log n)$ which gives a similar bound to the hypercube, $O(k \log n)$. If for a specific scheduling problem χ is asymptotically larger than ℓ then we can get a tighter approximation.

4 LINE GRAPH

Scheduling Problem. Consider a line graph $L = (V, E)$ which is a sequence of n nodes, $V = \{v_1, \ldots, v_n\}$, where there is an edge (v_i, v_{i+1}) of weight 1 for any $1 \leq i < n - 1$. Assume an orientation of the nodes in L from left to right, so that v_1 is the leftmost while v_n is the rightmost. Let O be a set of shared objects and suppose that each node executes a transaction which uses an arbitrary subset of O. Assume that each object is initially in a node with a transaction that requests it.

Algorithm. Let ℓ be the longest shortest walk of any object in O. Let $L_{i,z} = (V_{i,z}, E_{i,z})$ denote the subgraph with up to z nodes $V_{i,z} = \{v_i, v_{i+1}, \ldots, v_{i+z-1}\}$, where $E_{i,z} \subseteq E$ consists only of the edges connecting nodes in $V_{i,z}$. If $i+z-1 > n$, then we ignore all the nodes with subscript higher than n. Let $S = \{L_{x,\ell} : x = y\ell + 1,\ y \geq 0\}$, denote a decomposition of L into consecutive line subgraphs each of size ℓ. Let S_1 (S_2) denote the subset of S consisting of the $L_{x,\ell}$ with even (odd) index y in the definition of S. Namely, S_1 consists of the even subsequence of the line subgraphs of length ℓ, while S_2 consists of the odd subsequence of the line subgraphs. We schedule the transactions in two phases. (If $\ell = n$ then we only have Phase 1.) In the first phase we execute the transactions in S_1 while in the second phase we execute the transactions in S_2.

Phase 1: The first phase consists of two periods:
- *Period 1:* In the first period, each object is positioned to the leftmost node of a node in S_1 that needs it. This period has duration $\ell - 1$.
- *Period 2:* In the second period we execute the transactions in S_1. Within each subgraph $L' \in S_1$, the transactions execute from left to right. This period has duration ℓ. Suppose that $L' = (V', E') \in S_1$ where $V' = \{v_{i_1}, v_{i_2}, \ldots, v_{i_\ell}\}$. In the first time step of the period we execute the transaction (if any) in the first node v_{i_1} of L' and then immediately all the objects in v_{i_1} which are needed by transactions on the right move to the second node v_{i_2}. In the second time step we execute the transaction in v_{i_2} (if any) and then the objects in v_{i_2} move to v_{i_3} (if needed by transactions on the right). This repeats until the transaction (if any) in v_{i_ℓ} executes.

Phase 2: The second phase consists of two periods:
- *Period 1:* In the first period, each remaining object (still needed by an non-executed transaction) is positioned to the leftmost node that needs it. This period has duration $\ell - 1$.
- *Period 2:* In the second period we execute the transactions in S_2. Within each subgraph $L' \in S_2$, the transactions execute from left to right, similar to phase 1, period 2. This period has duration ℓ.

Analysis. The reason for having two phases (when $\ell < n$) is to allow a gap of ℓ nodes between subgraphs of length ℓ. The gap allows to execute in parallel the transactions in the subgraphs of each phase, since there is no object that will be needed concurrently by two subgraphs of a phase, since ℓ is a bound on the shortest walk of any object. Phase 1, period 1, finishes within $\ell - 1$ steps since each object moves to a node in S_1 at distance at most $\ell - 1$ from its original position. In phase 1, period 2, no object can be requested simultaneously by different subgraphs of S_1, since the maximum walk length is ℓ. Therefore, transactions in different subgraphs of S_1 can execute in parallel. Since each subgraph has ℓ nodes, ℓ steps suffice to execute all transactions in S_1. A similar analysis holds for the second phase. This execution has total duration $4\ell - 2$ steps which is asymptotically optimal (within a factor 4).

Theorem 4.1 (Line Graph). *For the line graph, for any arbitrary input instance there is an execution schedule with asymptotically optimal time.*

5 GRID GRAPH

Scheduling Problem. Consider a $n \times n$ grid graph $G = (V, E)$ where $|V| = n^2$. Every node has a coordinate (x, y), $1 \le x, y \le n$, and connects with its four neighbors (up, down, left, right) with an edge of weight 1. Assume an orientation of the grid on the plane such that node $(1, 1)$ appears at the top left. Nodes at the border of the grid connect with three neighbors and the corner nodes connect to two neighbors. There are w objects $O = \{o_1, \ldots, o_w\}$. Each node holds a transaction that uses a random subset of k objects, where $1 \le k \le w$. Initially, each object is at one of the nodes (if any) that needs it.

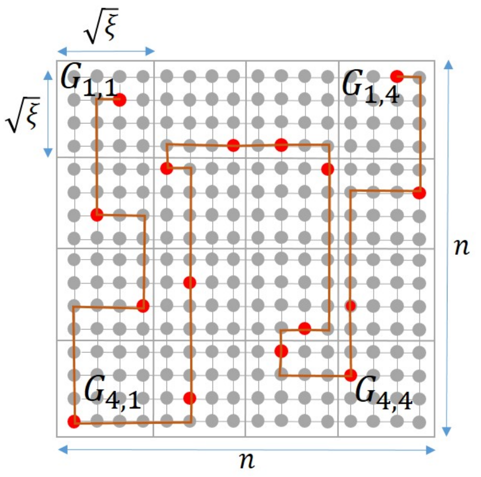


Figure 1: A grid of size 16×16 with subgrids of size 4×4. It depicts the path of an object.

Algorithm. Let $m = \max(n, w)$ and $\xi = (27w \ln m)/k$. Assume for simplicity that ξ and $\sqrt{\xi}$ are integers (otherwise, we may simply use ceilings which do not affect the proven asymptotic bounds). If $\sqrt{\xi} < n$, decompose the grid into subgrids $G_{i,j}$ of size $\sqrt{\xi} \times \sqrt{\xi}$. Assume also for simplicity that $\sqrt{\xi}$ divides n, since otherwise we obtain partial subgrids at the borders of G that have size smaller than $\xi \times \xi$, but those can be treated similarly to complete subgrids without affecting the analysis. The top-left corner of subgrid $G_{i,j}$ is positioned at node $((i-1)\sqrt{\xi}+1, (j-1)\sqrt{\xi}+1)$, where $1 \le i, j \le n/\sqrt{\xi}$. Note that G can be divided into $n/\sqrt{\xi}$ columns of subgrids, where each column consists of $n/\sqrt{\xi}$ subgrids. If $\sqrt{\xi} \ge n$, then there is only one subgrid, which is the whole of G.

The schedule executes the transactions in each subgrid separately, one subgrid at a time. The schedule follows a column major order of subgrids, starting from the leftmost column and ending at the rightmost column of subgrids (see Figure 1). In the jth column of subgrids the order of subgrids is top to bottom if j is odd, while the order is bottom to top if j is even. The first subgrid is $G_{1,1}$ at the top of the leftmost column. All transactions within $G_{1,1}$ execute and once they finish then execution continues with the transactions in subgrid $G_{2,1}$ immediately below. When all transactions within $G_{2,1}$ finish execution, then the execution continues in subgrid $G_{3,1}$ immediately below, and so on, until the last subgrid $G_{n/\sqrt{\xi},1}$ (at the bottom) of the first column. Then the execution continues in the second column starting with the bottom subgrid $G_{n/\sqrt{\xi},2}$ at the bottom, and then it executes the subgrids above in the second column in a similar way until the topmost subgrid. The third column of subgrids is processed top to bottom. The execution continues in a similar way, alternating the direction in each column of subgrids, and it ends when all transactions complete at the last subgrid of the rightmost column.

Within a subgrid $G_{i,j}$ an object may be requested by multiple nodes. We use the greedy schedule described in Section 2.3 to execute the transactions within each subgrid. We will refer to this as the *internal schedule* of each subgrid. Between the internal schedule of two consecutive subgrids there is a *transition period* where objects move from one subgrid to the next. Initially, before execution in $G_{1,1}$ starts, all required objects move to $G_{1,1}$ and position themselves to the respective first node that needs them in the internal schedule of $G_{1,1}$. Once execution in a subgrid finishes then the objects move from the current subgrid to the next subgrid in the order. Once the next subgrid has all the objects positioned in the first node of its internal schedule, then execution begins according to the internal schedule. Whenever objects move from one node to another they follow a shortest path in G.

Note that there may be the case that some object may not be requested by the current subgrid in the column order. In this case, the object moves directly to the immediately next subgrid in the order that contains a transaction that requests it. The object waits there until the respective subgrid becomes the current one to execute.

Analysis. Suppose for now that $\xi \le n^2$. Consider a subgrid $G_{i,j} = (V', E')$, with ξ nodes $v_{i_1}, \ldots, v_{i_\xi}$. Consider an object $o_z \in O$. Let $X_y \in \{0, 1\}$ denote an event such that $X_y = 1$ if object o_z is used by a transaction in node n_y in $G_{i,j}$, and otherwise $X_y = 0$. Let $X = \sum_{v_y \in V'} X_y$. Denote $L = 9 \ln m$ and $U = 45 \ln m$.

Lemma 5.1. *For $\xi \le n^2$, $\Pr(L < X < U) \ge 1 - 2/m^4$.*

Proof. There are $\binom{w}{k}$ subsets of k objects. The number of these subsets containing object o_z is $\binom{w-1}{k-1}$. Therefore, the probability that node n_y picks object o_z is $\binom{w-1}{k-1}/\binom{w}{k} = k/w$, or equivalently,

$\Pr(X_y = 1) = k/w$. We have that $E[X_y] = k/w$, which implies that

$$\mu = E[X] = E\left[\sum_{v_y \in V'} X_y\right] = \sum_{v_y \in V'} E[X_y] = \frac{\xi k}{w} = 27 \ln m.$$

Let $\delta = 2/3$. Using the Chernoff bound in Equation 1 we get $\Pr(X \geq U) = \Pr(X \geq 45 \ln m) \leq e^{-4 \ln m} = 1/m^4$. Similarly, using the Chernoff bound in Equation 2 we get $\Pr(X \leq L) = \Pr(X \leq 9 \ln m) < e^{-4 \ln m} = 1/m^4$. By combining the two bounds, we get the desired result. □

Lemma 5.2. *For $\xi \leq n^2$, with probability at least $1 - 2/m$, each of the w objects is used by more than L and less than U transactions in each subgrid of G.*

Proof. From Lemma 5.1, any specific object o_z within a specific subgrid is used by more than L and less than U transactions with probability at least $1 - 2/m^4$. Considering now all subgrids, which do not exceed $n^2 \leq m^2$, we get that object o_z is used by more than L and less than U transactions with probability at least $1 - 2m^2/m^4 = 1 - 2/m^2$. Considering now all w objects, where $w \leq m$, we get that each of the w objects is used by more than L and less than U transactions in each subgrid of G, with probability at least $1 - 2m/m^2 = 1 - 2/m$. □

Lemma 5.3. *For $\xi \leq n^2/9$, any schedule requires at least $\Omega(n^2/\sqrt{\xi})$ time steps to execute all the transactions in G, with probability at least $1 - 2/m$.*

Proof. From Lemma 5.2, each object is requested by more than $L \geq 1$ transactions within each grid. Thus the object has to visit at least one node in all n^2/ξ subgrids. Consider now the *odd* subgrids $G_{i,j}$, where i and j are odd numbers. Note that there are at least $n^2/(4\xi)$ odd subgrids, since $\xi \leq n^2/9$. The shortest walk to connect the respective nodes within the odd subgrids is at least $(n^2/(4\xi) - 1)\sqrt{\xi} = \Omega(n^2/\sqrt{\xi})$ since the shortest walk has to cross *even* subgrids between any two odd subgrids with a path of length at least $\sqrt{\xi}$, and no odd subgrid is repeated in the best case. □

Lemma 5.4. *For $\xi \leq n^2$, our algorithm requires $O(kn^2 \log m/\sqrt{\xi})$ time steps to execute all the transactions in G, with probability at least $1 - 2/m$.*

Proof. The execution time is divided into phases of internal grid execution and phases of transferring the objects from one subgrid to the next.

From Lemma 5.2, each object is requested by less than U transactions within each grid with probability at least $1 - 2/m$ (and by more than L transactions). The diameter of a subgrid is less than $2\sqrt{\xi}$, and each transaction uses exactly k objects. Therefore, the weighted degree of the dependency graph is bounded by $2\sqrt{\xi}Uk = O(k\sqrt{\xi}\log m)$ which is also the time spent in each subgrid for the internal schedule. Since there are n^2/ξ subgrids, the total time spent in internal executions is $O((n^2/\xi) \cdot k\sqrt{\xi}\log m) = O(kn^2 \log m/\sqrt{\xi})$.

At most $2n = O(n)$ steps are needed to move the objects from their original positions to the transactions that request them in the first subgrid $G_{1,1}$. Once execution in the column order starts, the time to move the objects from one subgrid to the immediately next in the same column, is no more than $3\sqrt{\xi}$. This is also the same time to move the objects from the last subgrid of the current column to the first subgrid of the next column (since these subgrids are adjacent). Since there are in total n^2/ξ subgrids, it takes total time $3\sqrt{\xi}(n^2/\xi - 1) = O(n^2/\sqrt{\xi})$ time steps to transfer the objects between subgrids. Since $\xi \leq n^2$, this bound remains the same even if we consider the additional $O(n)$ time to initially position the objects in the first subgrid.

Combining the above bounds, we obtain that the total time spent is $O(kn^2 \log m/\sqrt{\xi})$. □

Theorem 5.5. *The grid scheduling algorithm provides an $O(k \log m)$ approximation to the optimal schedule with probability at least $1 - \Theta(1/m)$.*

Proof. For $\xi \leq n^2/9$, the result follows from Lemma 5.3 and Lemma 5.4.

Consider now the case $\xi > n^2/9$. Then, there are no more that 9 subgrids. The bounds with respect to U in Lemma 5.1 and Lemma 5.2 still hold. Therefore, each object is used by less than U transactions in each subgrid with probability at least $1 - 2/m$. Consequently, an object is used by less than $9U$ transactions in total in G. Thus, the maximum number of other transactions that a transaction conflicts with is less than $9Uk$ (with high probability).

We apply the greedy schedule in the whole G. If each object is requested by at most one transaction, then the execution takes trivially 1 time step, which is optimal. Suppose now that some object is requested by at least two transactions. Let τ be the maximum distance between any pair of transactions that request the same object. Since objects are originally positioned at some transaction that needs them, it takes at most τ time steps to position the objects in the first transaction according to the greedy schedule. In the dependency graph the maximum weighted degree is bounded by $9Uk\tau$. Therefore, the total time to execute the transactions is bounded by $\tau + 9Uk\tau + 1 = O(Uk\tau) = O(\tau k \log m)$. Since τ is a lower bound for the execution time, we obtain an $O(k \log m)$ approximation. □

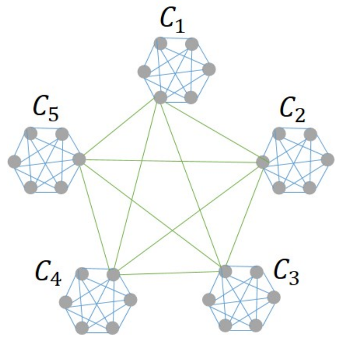
6 CLUSTER GRAPH

Scheduling Problem. The communication graph $G = (V, E)$ consists of α sub-graphs (clusters) $C_1, \ldots, C_\alpha$ such that each C_i is a complete graph with β nodes and edge weights 1 (see left part of Figure 2). In each cluster C_i there is a designated bridge node such that between any pair of clusters there is a bridge edge connecting the respective bridge nodes. Each bridge edge has weight γ. We assume that $\gamma \geq \beta$, namely, the clusters are far apart from each other. Each node of G holds one transaction with k arbitrary objects from the set of transactions $O = \{o_1, \ldots, o_w\}$.

Algorithm. If each object is used only within one cluster, then we apply the greedy schedule within each cluster in parallel. Otherwise, if there are objects used by more than one cluster, the algorithm uses one of the two following approaches, whichever gives smaller execution time. Let $m = \max(n, w)$. Let σ denote the maximum number of clusters that any object is requested to.

Approach 1: We execute the transactions in G using the greedy schedule of Section 2.3.

Approach 2: The details of this approach appear in Algorithm 1. The algorithm consists of $\psi = \lceil \sigma/(24 \ln m) \rceil$ phases. We



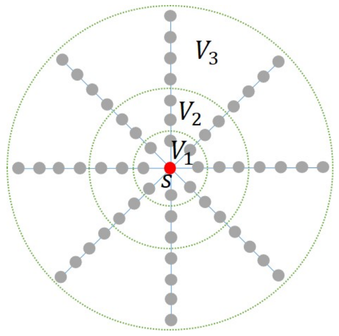


Figure 2: Left: A graph with 5 clusters where each cluster C_i is a complete graph with 6 nodes; links within clusters have weight 1, while links between clusters have weight γ. Right: A star graph with 8 rays, each ray consisting of 7 nodes; the rings depict the set of segments V_1, V_2, V_3.

assign each cluster to a uniform randomly chosen phase, and execute all the transactions of the cluster in that phase. A phase consists of $\zeta = 2 \cdot 40^k \lceil \ln^{k+1} m \rceil$ rounds, where each round has duration $\beta + \gamma + 2$ time steps. Within each round an object *gets active in some cluster*, namely, the object picks uniformly at random one of the clusters (if any) that has a transaction that needs it at that phase and the object moves to that cluster. A transaction is enabled when all its objects are activated (in the analysis we show that a transaction is enabled with certain probability). In a round, within each cluster, the enabled transactions execute using the greedy schedule of Section 2.3. The duration of a round guarantees that there is enough time to execute the enabled transactions within each cluster.

Analysis. If we use Approach 1, then in the transaction dependency graph the weighted degree of a transaction is bounded by $k\sigma\beta(\gamma + 2)$, since any pair of transactions in different clusters are at distance $\gamma + 2$ from each other (through the respective bridges), and a transaction requests k objects and each object visits at most $\sigma\beta$ transactions.

Lemma 6.1. *Using Approach 1, the algorithm executes all transaction within time $O(k\sigma\beta\gamma)$ time steps.*

Now, consider Approach 2. For any object $o_i \in O$, let Z_i denote the set of clusters that have at least one transaction that uses o_i. We have that $\sigma = \max_i |Z_i|$, and hence, $|Z_i| \le \sigma$. Let Φ_p be the set of clusters which are randomly assigned in phase p. Let $S_{i,p} = Z_i \cap \Phi_p$ denote the clusters which use object o_i and are assigned in phase p. Let $\xi = \max_{i,p} |S_{i,p}|$ denote the maximum number of clusters assigned in any phase for any object.

Lemma 6.2. $\Pr(\xi > 40 \ln m) < 1/m$.

Proof. Since there are $\psi = \lceil \sigma/(24 \ln m) \rceil$ phases, each cluster in Z_i picks phase ρ with probability $1/\psi$. Therefore, the expected number of clusters in Z_i that pick phase j is $|Z_i|/\psi \le \sigma/\psi \le 24 \ln m$. From the Chernoff bound in Equation 1, by setting $\delta = 2/3$ we obtain $\Pr(|S_{i,p}| > 40 \ln m) < 1/m^3$. Since the number of phases is bounded by $\sigma \le \alpha \le n \le m$, and the number of objects is bounded by $w \le m$, the result follows by taking the union bound over all phases and objects. □

Algorithm 1: Cluster Schedule (Approach 2)

```
Input: Graph $G$ with $n$ nodes and clusters $C_1, \ldots, C_\alpha$, where
       each transaction in $G$ uses $k$ objects in set
       $O = \{o_1, \ldots, o_w\}$
Output: An execution schedule for the transactions
1  Let $Z_i$ be the set of clusters that have transactions that use $o_i$;
2  $m \leftarrow \max(n, w)$; $\sigma \leftarrow \max_i |Z_i|$; $\psi \leftarrow \lceil \sigma/(24 \ln m) \rceil$;
   $\zeta \leftarrow 2 \cdot 40^k \lceil \ln^{k+1} m \rceil$;
3  //Assign each cluster to a phase
4  for $j \leftarrow 1$ to $\alpha$ do
5      $x \leftarrow$ random$(1, \psi)$;
6      $\Phi_x \leftarrow \Phi_x \cup C_j$; //$\Phi_x$ are the clusters of phase $x$
7  //Execute the transactions in each phase
8  for phase $p \leftarrow 1$ to $\psi$ do
9      for round $r \leftarrow 1$ to $\zeta$ do
10         foreach object $o_i \in O$ do
11             //$A_i$ is the cluster in which object $o_i$ activates
12             $A_i \leftarrow nil$;
13             if $Z_i \cap \Phi_p \neq \emptyset$ then
14                 $A_i \leftarrow$ a random uniformly chosen cluster from
                   set $Z_i \cap \Phi_p$;
15                 Move $o_i$ to the bridge node of cluster $A_i$ (if
                   $A_i \neq nil$);
16         $E \leftarrow \emptyset$; //set of enabled transactions
17         foreach cluster $C_y \in \Phi_p$ do
18             foreach transaction $T \in C_y$ do
19                 if all objects of $T$ have been activated in $C_y$
                   ($A_i = C_y$ for each $o_i \in T$) then
20                     $E \leftarrow E \cup \{T\}$;
21         Execute all enabled transactions in $E$ using the greedy
           schedule of Section 2.3;
```

Consider now some phase p, where $1 \le p \le \psi$. Let T_x denote a transaction that has not executed yet (in a previous phase) and is in one of the clusters, say C_y, assigned to phase p (namely, $C_y \in \Phi_p$).

Lemma 6.3. *If $\xi \le 40 \ln m$, then with probability at least $1 - 1/m^2$, transaction T_x will execute in phase p.*

Proof. Within phase p, transaction T_x will execute in a round that it gets enabled. Let $o_{z_1}, \ldots, o_{z_k}$ denote the objects of transaction T_x. Each object o_{z_j} may be needed by up to ξ clusters within phase ϕ (recall that $|S_{z_j,p}| \le \xi$). In a round, object o_{z_j} becomes active in cluster C_y with probability at least $1/\xi$, since the object picks randomly one of the at most ξ available clusters for it (in $S_{z_j,p}$). Thus, transaction T_x becomes enabled with probability at least $1/\xi^k$. For the number of rounds ζ, it holds that $\zeta \ge 2\xi^k \ln m$. Thus, T_x does not get enabled in phase ϕ with probability at most

$$\left(1 - \frac{1}{\xi^k}\right)^{\zeta} \le \left(1 - \frac{1}{\xi^k}\right)^{2\xi^k \ln m} \le \left(\frac{1}{e}\right)^{2 \ln m} = \frac{1}{m^2}.$$

□

Lemma 6.4. *Using Approach 2, with probability at least $1 - 2/m$, all transactions execute within time $O(\sigma\gamma 40^k \ln^k m)$.*

Proof. Each transaction belongs to some assigned cluster of some phase. Assuming that $\xi \le 40 \ln m$, from Lemma 6.3 each transaction will execute at the phase its cluster gets assigned with probability at least $1 - 1/m^2$. Therefore, all $n \le m$ transactions will finish by the end of the last phase, with probability at least $1 - m/m^2 = 1 - 1/m$. Combining this with Lemma 6.2, we get that with probability at least $1 - 2/m$ all transactions complete execution by the end of the last (ψth) phase.

It only remains to establish that a round has enough time steps to execute the transactions. Clearly, in a round there can be no more than β enabled transactions within a cluster. It takes $\gamma + 2$ steps to transfer objects from one cluster to another through the respective bridge nodes. Therefore, the duration of a round, $\beta + \gamma + 2$, suffices to execute all enabled transactions and then transfer them to the respective cluster that will be used in the next round. There are ζ rounds, and $\lceil \sigma/(24 \ln m) \rceil$ phases. Consequently, since $\beta \le \gamma$, the total number of time steps is:

$$\lceil \sigma/(24 \ln m) \rceil \cdot \zeta \cdot (\beta + \gamma + 2) = O(\sigma\gamma 40^k \ln^k m).$$

□

Theorem 6.5 (cluster graph). *With probability at least $1 - \Theta(1/m)$, the cluster graph algorithm will execute all the transactions in time within $O(\min(k\beta, 40^k \ln^k m))$ factor from optimal.*

Proof. If each transaction is used in only one cluster, then with the same analysis as in Theorem 3.1, we obtain an $O(k)$ approximation to the optimal solution. In any other case, $\Omega(\sigma\gamma)$ is a lower bound, since each object will have to move to at least $\sigma - 1$ different clusters after the initial one, and it takes $\gamma + 2$ steps to reach a node from one cluster to another through the respective bridge nodes.

If Approach 1 is used, then from Lemma 6.1 the algorithm executes all transaction within time $O(k\sigma\beta\gamma)$ time steps, which is a $O(k\beta)$ factor from optimal.

If Approach 2 is used, then from Lemma 6.4 with probability at least $1 - 2/m$, all transactions execute in time $O(\sigma\gamma 40^k \ln^k m)$, which is a $O(40^k \ln^k m)$ factor from optimal.

Combining all the approximation factors from the three different cases we obtain the desired result, as needed. □

Note that the time bound of Theorem 6.5 may also be written as $O(\min(k\beta, \log_c^k m))$, where c is a constant such that $40 = 1/\ln c$. For constant k, Theorem 6.5 gives a poly-log approximation for the optimal execution time.

Corollary 6.6. *For constant k, with high probability the cluster graph algorithm will execute the transactions in time within a poly-log factor of m from optimal.*

7 STAR GRAPH

A star graph G consists of α rays where each ray is a line graph with β nodes (see right part of Figure 2). There is a *center node s* which is adjacent to the tip of each ray. Every edge in the star graph G has weight 1. We consider scheduling problems where each node holds a transaction that uses k arbitrary objects out of the object set $O = \{o_1, \ldots, o_w\}$.

A star graph can be analyzed similarly to the cluster graph (Section 6) in combination with the analysis of the line graph (Section 4). We divide each ray into $\eta = \lceil \log \beta \rceil$ segments (log is base 2) of exponentially increasing lengths. In particular, consider a ray r and assume that it contains nodes $v_1, \ldots, v_\beta$, such that v_1 is adjacent to the center s, while v_β is the furthest from s. The ith segment of r, where $1 \le i \le \eta$, consists of the nodes $v_{2^{i-1}}, \ldots, v_{2^i - 1}$. Note that the last segment may be truncated. The ith segment, $i < \eta$, has 2^{i-1} nodes, while the last segment has no more than $\beta/2 + 1$ nodes. Every node of the ith segment is at distance at least 2^{i-1} from the center s.

We start the schedule by executing the transaction in the center node s. We then continue execution with the transactions in the rays. The execution in the rays is performed in different time periods, one period dedicated for each segment length. Let V_i denote the set of nodes of G which are in the ith segment in each ray (see the rings in the right part of Figure 2). There are η periods where the ith period is dedicated to execute the transactions in V_i.

Consider now the ith period. Each ray segment of V_i can be treated as if it is a cluster. The clusters (segments) communicate through s, with paths of length $2 \cdot 2^{i-1} = 2^i$ (from one tip of the segment to the next). This is similar to directly connecting two nodes of the clusters with a link (bridge edge) of weight $\gamma = 2^i$. Since each segment is a line graph, the transactions in it can execute sequentially in time no more that the length of each segment (see also Section 4). Let σ_i denote the maximum number of segments in V_i that an object has to visit due to transactions requesting it. If $\sigma_i = 1$ (i.e. an object visits one segment only), then we can execute the transactions in the segments in parallel in $O(2^i)$ time. So assume that $\sigma_i > 1$.

Using a greedy schedule similar to Approach 1 of the Cluster graph, we can prove (similar to Lemma 6.1) that the transactions in V_i can execute in time $O(k\sigma_i 2^{2i})$ (for size of cluster we have $O(2^i)$). When using the schedule similar to Approach 2 of the Cluster graph, we can prove (similar to Lemma 6.4) that with high probability, the transactions in V_i can execute in time $O(\sigma_i 2^i c^k \ln^k m)$, for some constant c, where $m = \max\{n, w\}$. Noting that $\Omega(\sigma_i 2^i)$ is a lower bound for the execution, we obtain that the transactions in V_i can execute in time which is within factor $O(\min(k2^i, c^k \ln^k m))$ from optimal. Since we have η periods, we get that the approximation factor is $O(\eta \max_i \min(k2^i, c^k \ln^k m)) = O(\log\beta \cdot \min(k\beta, c^k \ln^k m))$.

Theorem 7.1 (star graph). *There is an execution schedule in the star graph, which with high probability all transactions execute within time $O(\log\beta \cdot \min(k\beta, c^k \ln^k m))$ factor from the optimal schedule, for some constant c.*

For constant k, Theorem 7.1 gives a schedule which is within a poly-log factor from optimal.

Corollary 7.2. *For constant k, there is a schedule in the star graph which with high probability the transactions execute within time a poly-log factor of β and m from optimal.*

8 A LOWER BOUND FOR EXECUTION TIME

The *shortest walk* of an object minimizes the total distance to visit all the transactions that require the object. The maximum shortest walk of any object is a lower bound for the execution time. Note that an optimal TSP tour length of an object is no more than twice its shortest walk length. In other words, half of the maximum optimal TSP tour length of any object is a lower bound on the execution time as well.

We use the probabilistic method to prove the existence of scheduling problem instances on graphs with n nodes, such that the optimal TSP tour length of any object is $O(n^{4/5})$ and yet, any possible schedule has execution time $\Omega(n^{4/5+1/40}/\log n)$. The problem instances use two objects per transaction. We consider two kinds of graphs, grid graphs and tree graphs. We first give the grid description and then provide the tree description which is a straightforward modification based on the grid.

8.1 Lower Bound on Grids

Consider a graph G which is a $s \times s\sqrt{s}$ grid of nodes, for a total of $n = s^{5/2}$ nodes (see left part of Figure 3). Divide the grid into s subgraphs (blocks) $H_1, \ldots, H_s$, each having s rows and $\sqrt{s}$ columns. (For simplicity assume that $\sqrt{s}$ is an integer.) Within each block, a node is connected to four neighbor nodes (up, down, left, right) by an edge of weight 1, except for the corner nodes or side nodes of the block which are connected to two or three neighbors within the same block, respectively. Adjacent blocks H_i and H_{i+1}, where $1 \le i < s$, are connected to each other through horizontal edges of weight s between two neighbor nodes.

Each node in H_i holds a transaction, for all $1 \le i \le s$. The set of objects is $O = A \cup B$, such that $A = \{a_1, \ldots, a_s\}$ and $B = \{b_1, \ldots, b_s\}$. Object $a_i \in A$ is used by all the transactions in block H_i, for any $1 \le i \le s$. Initially, each object a_i resides in the top left corner node in H_1. In each block H_i, each transaction picks randomly and uniformly one of the objects in B, for any $1 \le i \le s$. Initially, each object $b_i \in B$ resides in some node of H_1 that uses it (if any, otherwise in an arbitrary node of H_1).

We first bound the shortest walks (and hence, TSP tours) for the objects. Let ℓ_i denote the shortest path length (number of edges) for object $b_i \in B$. Let $\ell = \max_i \ell_i$. We can prove the following results:

Lemma 8.1. $\Pr(\ell \le 5s^2) > 1 - s^2 e^{-\sqrt{s}/12}$.

Lemma 8.2. *Any set of λ transactions, where $s^{3/8} \le \lambda \le s$ and $s \ge e^{7\cdot 80}$, that execute in any block H_i during a period of s time steps, requires at least $\lambda^{3/5}$ unique objects of B, with probability at least $1 - s^{-s^{3/8}/161}$.*

The union probability of Lemmas 8.1 and 8.2 is fast reaching 1. Therefore, these two lemmas imply that for any large enough s there is a problem instance I_s such that the maximum path length of any object in B is bounded by $5s^2$. Since objects in A have shortest path length in their blocks of length at most $s\sqrt{s}$, the bound of $5s^2$ applies for all objects in O. Moreover, in any period of s time steps, sets of λ executed transactions use at least $\lambda^{3/5}$ different objects of B. Therefore, we have the following result:

Corollary 8.3. *For any $s \ge e^{7\cdot 80}$, there is a problem instance I_s on the grid graph such that:*

- *the shortest path length of any object is at most $5s^2$, and*
- *any set of λ transactions within any block H_i, where $s^{3/8} \le \lambda \le s$, which execute during a period of s time steps, use at least $\lambda^{3/5}$ distinct objects of B.*

The following theorem applies to problem instances I_s as given by Corollary 8.3.

Theorem 8.4 (Execution Time Lower Bound). *There are problem instances on the grid graph with two objects per transaction, such that every execution schedule has duration $\Omega(s^{33/16}/\log s) = \Omega(n^{4/5+1/40}/\log n)$, and every object has TSP tour length at most $O(s^2) = O(n^{4/5})$.*

8.2 Lower Bound on Trees

The lower bound construction of graph G for trees is very similar to the lower bound construction on grids. The main difference is in the structure of the blocks $H_1, \ldots, H_s$ (see right part of Figure 3). Each block is a tree such that the leftmost column is connected, and each row is connected and attached to the leftmost column. The weights of the edges within the block are equal to 1. The trees of the adjacent blocks are connected through the topmost row, where the edge weight between two blocks is s.

All the results for the grid graph hold also verbatim for the tree graph. Therefore, similar to Theorem 8.4, we obtain that there are problem instances on the tree graph with two objects per transaction such that every execution schedule has duration $\Omega(n^{4/5+1/40}/\log n)$ and every object has TSP tour length $O(n^{4/5})$.

9 CONCLUSION

We presented a comprehensive set of bounds for various specialized network topologies. If the system is not completely synchronous, then our bounds are affected by the synchronicity factor (maximum delay divided by minimum delay). There are some open questions. It would be interesting to extend the results to the online setting, where the set of transactions to be executed are not known ahead of time. It would also be interesting to examine the impact of congestion, where network links have bounded capacity.

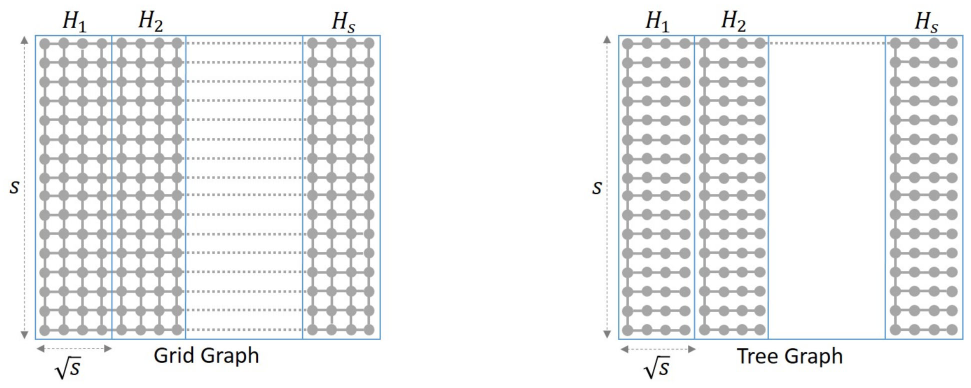


Figure 3: Left: grid graph; Right: tree graph.

ACKNOWLEDGMENTS

This work is supported by the National Science Foundation grants 1320835 and 1420673, and partly supported by the Serbian Ministry of Education & Science, through grants No. III 44009, and TR 32031.

REFERENCES

[1] Hagit Attiya, Leah Epstein, Hadas Shachnai, and Tami Tamir. 2010. Transactional Contention Management as a Non-Clairvoyant Scheduling Problem. *Algorithmica* 57, 1 (2010), 44–61.

[2] Robert L. Bocchino, Vikram S. Adve, and Bradford L. Chamberlain. 2008. Software transactional memory for large scale clusters. In *PPoPP*. 247–258.

[3] Costas Busch, Maurice Herlihy, Miroslav Popovic, and Gokarna Sharma. 2015. Impossibility Results for Distributed Transactional Memory. In *PODC*. 207–215. DOI:http://dx.doi.org/10.1145/2767386.2767433

[4] Harold W. Cain, Maged M. Michael, Brad Frey, Cathy May, Derek Williams, and Hung Q. Le. 2013. Robust architectural support for transactional memory in the power architecture. In *ISCA*. 225–236. DOI:http://dx.doi.org/10.1145/2485922.2485942

[5] Henri Casanova, Arnaud Legrand, and Yves Robert. 2008. *Parallel Algorithms* (1st ed.). Chapman & Hall/CRC.

[6] M. Y. Chan. 1989. Embedding of D-dimensional Grids into Optimal Hypercubes. In *SPAA*. 52–57. DOI:http://dx.doi.org/10.1145/72935.72941

[7] Paolo Costa, Hitesh Ballani, Kaveh Razavi, and Ian Kash. 2015. R2C2: A Network Stack for Rack-scale Computers. In *SIGCOMM*. 551–564. DOI:http://dx.doi.org/10.1145/2785956.2787492

[8] Maria Couceiro, Paolo Romano, Nuno Carvalho, and Luís Rodrigues. 2009. D2STM: Dependable Distributed Software Transactional Memory. In *PRDC*. 307–313.

[9] Aleksandar Dragojević, Rachid Guerraoui, Anmol V. Singh, and Vasu Singh. 2009. Preventing Versus Curing: Avoiding Conflicts in Transactional Memories. In *PODC*. 7–16. DOI:http://dx.doi.org/10.1145/1582716.1582725

[10] Michel Dubois, Murali Annavaram, and Per Stenstrm. 2012. *Parallel Computer Organization and Design.* Cambridge University Press, New York, NY, USA.

[11] Wilson W. L. Fung, Inderpreet Singh, Andrew Brownsword, and Tor M. Aamodt. 2011. Hardware transactional memory for GPU architectures. In *MICRO*. 296–307.

[12] Rachid Guerraoui, Maurice Herlihy, and Bastian Pochon. 2005. Toward a Theory of Transactional Contention Managers. In *PODC*. 258–264.

[13] Ruud Haring, Martin Ohmacht, Thomas Fox, Michael Gschwind, David Satterfield, Krishnan Sugavanam, Paul Coteus, Philip Heidelberger, Matthias Blumrich, Robert Wisniewski, Alan Gara, George Chiu, Peter Boyle, Norman Chist, and Changhoan Kim. 2012. The IBM Blue Gene/Q Compute Chip. *IEEE Micro* 32, 2 (2012), 48–60.

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

[15] Maurice Herlihy and J. Eliot B. Moss. 1993. Transactional Memory: Architectural Support for Lock-free Data Structures. In *ISCA*. 289–300.

[16] Maurice Herlihy and Ye Sun. 2007. Distributed transactional memory for metric-space networks. *Distributed Computing* 20, 3 (2007), 195–208.

[17] William N. Scherer III and Michael L. Scott. 2005. Advanced contention management for dynamic software transactional memory. In *PODC*. 240–248.

[18] Intel. 2012. http://software.intel.com/en-us/blogs/2012/02/07/transactional-synchronization-in-haswell. (2012).

[19] Junwhan Kim and Binoy Ravindran. 2010. On Transactional Scheduling in Distributed Transactional Memory Systems. In *SSS*. 347–361.

[20] Junwhan Kim and B. Ravindran. 2013. Scheduling Transactions in Replicated Distributed Software Transactional Memory. In *CCGrid*. 227–234.

[21] F. Thomson Leighton. 1992. *Introduction to Parallel Algorithms and Architectures: Array, Trees, Hypercubes.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[22] Dawei Li, Jie Wu, Zhiyong Liu, and Fa Zhang. 2017. Towards the Tradeoffs in Designing Data Center Network Architectures. *IEEE Transactions on Parallel and Distributed Systems* 28, 1 (Jan. 2017), 260–273. DOI:http://dx.doi.org/10.1109/TPDS.2016.2610970

[23] Kaloian Manassiev, Madalin Mihailescu, and Cristiana Amza. 2006. Exploiting Distributed Version Concurrency in a Transactional Memory Cluster. In *PPoPP*. 198–208.

[24] Marek Michalewicz, Lukasz Orlowski, and Yuefan Deng. 2015. Creating Interconnect Topologies by Algorithmic Edge Removal: MOD and SMOD Graphs. *Supercomput. Front. Innov.: Int. J.* 2, 4 (March 2015), 16–47.

[25] Takuya Nakaike, Rei Odaira, Matthew Gaudet, Maged M. Michael, and Hisanobu Tomari. 2015. Quantitative comparison of hardware transactional memory for Blue Gene/Q, zEnterprise EC12, Intel Core, and POWER8. In *ISCA*. 144–157. DOI:http://dx.doi.org/10.1145/2749469.2750403

[26] Roberto Palmieri, Sebastiano Peluso, and Binoy Ravindran. 2015. Transaction Execution Models in Partially Replicated Transactional Memory: The Case for Data-Flow and Control-Flow. In *Transactional Memory*. 341–366. DOI:http://dx.doi.org/10.1007/978-3-319-14720-8_16

[27] Sudeep Pasricha and Nikil Dutt. 2008. *On-Chip Communication Architectures: System on Chip Interconnect.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[28] Sebastiano Peluso, Pedro Ruivo, Paolo Romano, Francesco Quaglia, and Luís Rodrigues. 2012. When Scalability Meets Consistency: Genuine Multiversion Update-Serializable Partial Data Replication. In *ICDCS*. 455–465. DOI:http://dx.doi.org/10.1109/ICDCS.2012.55

[29] Paolo Romano, Roberto Palmieri, Francesco Quaglia, Nuno Carvalho, and Luís Rodrigues. 2014. On speculative replication of transactional systems. *J. Comput. Syst. Sci.* 80, 1 (2014), 257–276. DOI:http://dx.doi.org/10.1016/j.jcss.2013.07.006

[30] Mohamed M. Saad and Binoy Ravindran. 2011. Snake: Control Flow Distributed Software Transactional Memory. In *SSS*. 238–252. DOI:http://dx.doi.org/10.1007/978-3-642-24550-3_19

[31] Gokarna Sharma and Costas Busch. 2012. A Competitive Analysis for Balanced Transactional Memory Workloads. *Algorithmica* 63, 1-2 (2012), 296–322.

[32] Gokarna Sharma and Costas Busch. 2012. Window-Based Greedy Contention Management for Transactional Memory: Theory and Practice. *Distrib. Comput.* 25, 3 (2012), 225–248.

[33] Gokarna Sharma and Costas Busch. 2014. Distributed Transactional Memory for General Networks. *Distrib. Comput.* 27, 5 (2014), 329–362.

[34] Nir Shavit and Dan Touitou. 1997. Software Transactional Memory. *Distrib. Comput.* 10, 2 (1997), 99–116.

[35] Richard M. Yoo and Hsien-Hsin S. Lee. 2008. Adaptive transaction scheduling for transactional memory systems. In *SPAA*. 169–178.

[36] Bo Zhang, Binoy Ravindran, and Roberto Palmieri. 2014. Distributed Transactional Contention Management as the Traveling Salesman Problem. In *SIROCCO*. 54–67. DOI:http://dx.doi.org/10.1007/978-3-319-09620-9_6