**Noname manuscript No.**
(will be inserted by the editor)

# Fast Scheduling in Distributed Transactional Memory

**Costas Busch · Maurice Herlihy ·
Miroslav Popovic · Gokarna Sharma**

Received: date / Accepted: date

**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

A preliminary version of this paper appears in the Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pages 173-182, Washington, DC, USA, July 2017 [4].

Costas Busch
Augusta University, Augusta, GA, USA
E-mail: kbusch@augusta.edu

Maurice Herlihy
Brown University, Providence, RI, USA
E-mail: herlihy@cs.brown.edu

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

Gokarna Sharma
Kent State University, Kent, OH, USA
E-mail: sharma@cs.kent.edu

realistic network models. To our knowledge, this is the first attempt to obtain provably fast schedules for distributed transactional memory.

**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* [16,35] (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 [19] and IBM's Blue Gene/Q [14], zEnterprise EC12 [26], and Power8 [5]. There are proposals for adapting TM to clusters of GPUs [2,12,24]. Here, we consider *distributed* TM systems appropriate for rack-scale or cluster-scale networks of nodes linked by a modern communication network [17,34,37,2,24].

Transactional memory is beneficial in distributed computing platforms where the data is spread across the participating nodes. For example, hierarchical datacenters and GPU clusters can use TM to simplify the burden of distributed synchronization and provide more reliable and efficient concurrent program execution while accessing data in possibly remote compute nodes. The distributed TM designed on top of such systems needs to execute the transactions effectively by taking into consideration the system's infrastructure. Especially, the network structure can play a crucial role in the performance of the distributed TM, since the data that the transactions access has to be reached across the network in a timely manner. Thus, in this work we address the efficiency of the distributed TM execution with emphasis on the underlying network infrastructure.

We consider a *data-flow* of transaction execution [17,34], 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 terminates 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 [8]. Here, we focus on the kinds of specialized networks likely to arise in practice, such as Clique, Line, Grid, Cluster, Hypercube, Butterfly, and Star [6,25,28,11,23]. 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 [8,6,25,28,11,23]. 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 $\beta$ 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 $\beta$ 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) multicore systems. Several scheduling algorithms with provable upper and lower bounds, and impossibility results are given [1,13,32,33], besides several other scheduling algorithms that are evaluated only experimentally [36,18]. Dragojevic *et al.* [10], 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 [2,9,24] 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 previous work [17,34,37,21] 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 [21] provided communication cost bounds for special workloads and problem instances with multiple shared objects. The execution time minimization is considered by Zhang *et al.* [37]. 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 bounds 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 multiversioning [29,24]. In replicated TMs, multiple copies are available for each shared object, whereas multiple versions of each object are available in multiversioning TMs [24]. Kim and Ravindran [20] studied transaction scheduling in replicated distributed TMs. In the *control-flow* model [31], objects are immobile and transactions either move to the network nodes where the required objects reside, or invoke remote procedure calls. Hendler *et al.* [15] 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.* [27] 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 [30] in replicated environments, and transaction scheduling using consistent snapshots [29, 30, 24, 20] 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. Particularly, we discuss in this section (i) the details of the distributed TM model, (ii) properties of a feasible schedule produced by any scheduling algorithm, (iii) a definition of an execution time performance bound for a scheduling algorithm, (iv) formal definitions of Chernoff bounds, and (v) a basic greedy scheduling algorithm based on coloring of transaction dependency graph and its approximation guarantee. Starting from Section 3 until Section 7 we discuss how to apply the basic greedy schedule (Section 2.3) to schedule transactions in the (special) topologies considered in this paper. The separate treatment of scheduling in different topologies is due to the fact that the considered topologies demand different techniques for the basic greedy schedule to provide an efficient approximation guarantee. In fact, our presentation starts with relatively simpler topologies for transaction scheduling such as Cliques, Hypercubes, and Butterflies and ends at more complex Star topology.

We present and analyze a scheduling algorithm to schedule transactions on a complete graph (Cliques) in Section 3, which uses the basic greedy scheduling algorithm developed in Section 2.3 as is. The ideas developed for a complete graph are then extended to schedule transactions in the Hypercube, Butterfly, and other related graphs. In Section 4 we give a scheduling algorithm and its approximation analysis for the Line graph. The idea behind the algorithm for the Line graph departs significantly from the ideas used in the scheduling algorithms for Clique, Hypercube, and Butterfly. In Section 5 we give a scheduling algorithm and its analysis for the Grid graph. The idea is to design a scheduling approach with the basic greedy algorithm as a subroutine. We present the result for the Cluster graph in Section 6, which uses either the basic greedy scheduling algorithm or the more sophisticated algorithm designed depending on whether the objects are used by only a cluster or more clusters. The result for the Star graph in Section 7 extends ideas for the Cluster graph appropriately. We prove our lower bound on execution time based on the TSP object tours in Section 8, showing the difficulty of obtaining execution time proportional to TSP object tours. 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 [32,1]. 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. However, note that in our algorithms the executions are conflict-free.

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 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 1 (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 $h_{\max}$ of any edge in $H$, since some object will require so much time to be transferred from one node to another in $G$. Let $\Delta$ be the maximum node degree in $H$, which is the maximum number of neighbors of any node in $H$. We define the *weighted degree* of $H$ to be $\Gamma = h_{\max} \cdot \Delta$. A greedy coloring scheme assigns colors to the transactions of $H$ one after the other so that $H$ can be colored with at most $\Gamma + 1$ colors in polynomial time. Each node $u$ in $H$ gets a color $k_u h_{\max} + 1 \leq \Gamma + 1$ for some integer $0 \leq k_u \leq \Delta$. In this way, from the pigeonhole principle, for the next node $v$ to be colored by the greedy algorithm there is $k_v$ with $0 \leq k_v \leq \Delta$ such that the respective color $k_v h_{\max} + 1 \leq \Gamma + 1$ has not been used by any of the at most $\Delta$ neighbors of $v$. This $\Gamma + 1$ coloring of $H$ gives a $\Delta + 1$ factor approximation to the optimal schedule in $G$, since the maximum edge weight $h_{\max}$ 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 is as follows.

**Theorem 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 $\ell$ 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 $\ell$, 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 [22] 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 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 $\chi_i$ denote the shortest path length that connects all transactions in $A_i$. Let $\chi = \max_i \chi_i$. Clearly, $\chi$ is a lower bound on execution time. Since $\chi \geq \ell$, we get a $O(kd)$ approximation for execution time. In Butterfly networks [22] and $\log n$-dimensional grids [7], $d = O(\log n)$ which gives a similar bound to the hypercube, $O(k \log n)$. If for a specific scheduling problem $\chi$ is asymptotically larger than $\ell$ 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 $1 \leq i < n - 1$ (see Figure 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 $\ell$ be the longest shortest walk of any object in $O$ (see Figure 1). 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 $\ell$. Let $S_1$ ($S_2$) denote the subset of $S$ consisting of the $L_{x,\ell}$ with even (odd) index $y$ in the definition

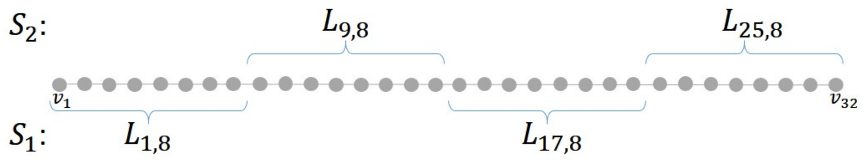

Fig. 1: A line graph with $n = 32$ nodes and $\ell = 8$.

of $S$. Namely, $S_1$ consists of the even subsequence of the line subgraphs of length $\ell$, 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 $\ell$. 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 $v_{i_\ell}$ 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 $\ell$.

*Analysis* The reason for having two phases (when $\ell < n$) is to allow a gap of $\ell$ nodes between subgraphs of length $\ell$. 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 $\ell$ 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 $\ell$. Therefore, transactions in different subgraphs of $S_1$ can execute in parallel. Since each subgraph has $\ell$ nodes, $\ell$ 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 2 (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 $(x, y)$, $1 \leq x, y \leq n$, 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 $1 \leq k \leq w$. Initially, each object is at one of the nodes (if any) that needs it.

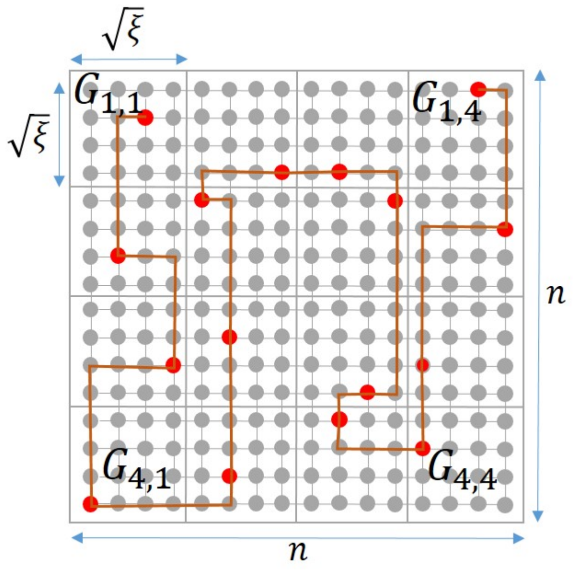

Fig. 2: A grid of size $16 \times 16$ with subgrids of size $4 \times 4$. It depicts the path of an object.

*Algorithm* Let $m = \max(n, w)$ and $\xi = (27w \ln m)/k$. Assume for simplicity that $\xi$ 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 $1 \leq i,j \leq n/\sqrt{\xi}$. Note that $G$ can be divided into $n/\sqrt{\xi}$ columns of subgrids, where each column consists of $n/\sqrt{\xi}$ subgrids. If $\sqrt{\xi} \geq n$, 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 2). In the $j$th 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 $\xi \leq n^2$. Consider a subgrid $G_{i,j} = (V', E')$, with $\xi$ 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 2** *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 $\Pr(X \ge U) = \Pr(X \ge 45 \ln m) \le e^{-4 \ln m} = 1/m^4$. Similarly, using the Chernoff bound in Equation 2 we get $\Pr(X \le L) = \Pr(X \le 9 \ln m) < e^{-4 \ln m} = 1/m^4$. By combining the two bounds, we get the desired result. $\square$

**Lemma 3** *For $\xi \le n^2$, 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 2, 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 $n^2 \le m^2$, 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 $w \le m$, 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 $1 - 2m/m^2 = 1 - 2/m$. $\square$

**Lemma 4** *For $\xi \le n^2/9$, 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 3, each object is requested by more than $L \ge 1$ transactions within each grid. Thus the object has to visit at least one node in all $n^2/\xi$ subgrids. Consider now the *odd* subgrids $G_{i,j}$, where $i$ and $j$ are odd numbers. Recall that $\sqrt{\xi}$ divides $n$, hence, there are at least $n^2/(4\xi)$ odd subgrids. Moreover, since $\xi \le n^2/9$, the number of odd subgrids is at least $9/4 > 2$, which enables the creation of a path of an object between at least two odd subgrids. 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. $\square$

**Lemma 5** *For $\xi \le n^2$, 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 3, 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 $n^2/\xi$ 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 $n^2/\xi$ 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 $\xi \le n^2$, 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 3** *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 $\xi \le n^2/9$, the result follows from Lemma 4 and Lemma 5.

Consider now the case $\xi > n^2/9$. Then, there are no more that 9 subgrids. The bounds with respect to $U$ in Lemma 2 and Lemma 3 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 $\tau$ 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 $\tau$ 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 $\tau$ 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 $\alpha$ subgraphs (clusters) $C_1, \ldots, C_\alpha$ such that each $C_i$ is a complete graph with $\beta$ nodes and edge weights 1 (see Figure 3). 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 $\gamma$. 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 $O = \{o_1, \ldots, o_w\}$.

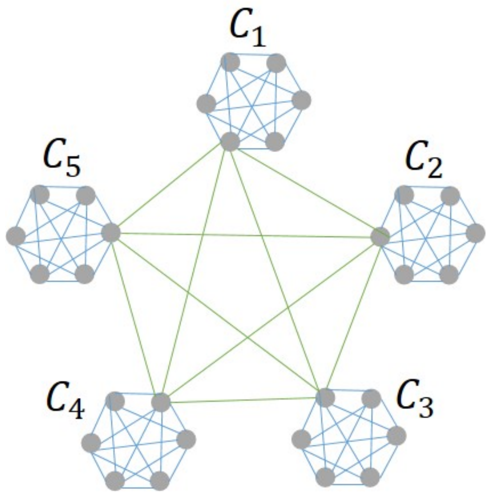

Fig. 3: A graph with 5 clusters where each cluster $C_i$ is a complete graph with 6 nodes; links within clusters have weight 1, while links between clusters have weight $\gamma$.

*Algorithm* Let $m = \max(n, w)$, and let $\sigma$ denote the maximum number of clusters that any object is requested to. In our algorithm, we use two approaches. If each object is used only within one cluster, then we can apply the greedy schedule within each cluster in parallel. In this case, *Approach 1* is suitable, which simply invokes the greedy schedule of Section 2.3. On the other hand, if there are objects used by more than one cluster, then *Approach 2* is more suitable. In the second approach, the clusters are scheduled to execute their transactions in different phases. Time is divided into $O(\sigma/\log m)$ phases, and clusters are randomly assigned to the phases, such that in each phase $O(\log m)$ clusters request an object. Thus, in each phase a small number of clusters request an object, which allows to efficiently execute the transactions within these clusters using a refined randomized schedule. The details are given below.

*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 assign each cluster to a

**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
Let $Z_i$ be the set of clusters that have transactions that use $o_i$;
$m \leftarrow \max(n, w)$; $\sigma \leftarrow \max_i |Z_i|$; $\psi \leftarrow \lceil \sigma/(24 \ln m) \rceil$; $\zeta \leftarrow 2 \cdot 40^k \lceil \ln^{k+1} m \rceil$;
//Assign each cluster to a phase
for $j \leftarrow 1$ to $\alpha$ do
    $x \leftarrow$ random$(1, \psi)$;
    $\Phi_x \leftarrow \Phi_x \cup C_j$; //$\Phi_x$ are the clusters of phase $x$
//Execute the transactions in each phase
for phase $p \leftarrow 1$ to $\psi$ do
    for round $r \leftarrow 1$ to $\zeta$ do
        foreach object $o_i \in O$ do
            //$A_i$ is the cluster in which object $o_i$ activates
            $A_i \leftarrow nil$;
            if $Z_i \cap \Phi_p \neq \emptyset$ then
                $A_i \leftarrow$ a random uniformly chosen cluster from set $Z_i \cap \Phi_p$;
                Move $o_i$ to the bridge node of cluster $A_i$ (if $A_i \neq nil$);
        $E \leftarrow \emptyset$; //set of enabled transactions
        foreach cluster $C_y \in \Phi_p$ do
            foreach transaction $T \in C_y$ do
                if all objects of $T$ have been activated in $C_y$ ($A_i = C_y$ for each
                  $o_i \in T$) then
                    $E \leftarrow E \cup \{T\}$;
        Execute all enabled transactions in $E$ using the greedy schedule of Section
          2.3;
```

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 a 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 dependency graph the maximum weighted degree 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** *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, $|Z_i| \leq \sigma$. Let $\Phi_p$ 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 7** $\Pr(\xi > 40 \ln m) < 1/m$.

*Proof* Since there are $\psi = \lceil \sigma/(24 \ln m) \rceil$ phases, each cluster in $Z_i$ picks phase $\rho$ with probability $1/\psi$. Therefore, the expected number of clusters in $Z_i$ that pick phase $j$ is $|Z_i|/\psi \leq \sigma/\psi \leq 24 \ln m$. 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 $\sigma \leq \alpha \leq n \leq m$, and the number of objects is bounded by $w \leq m$, the result follows by taking the union bound over all phases and objects. □

Consider now some phase $p$, where $1 \leq p \leq \psi$. 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 8** *If $\xi \leq 40 \ln m$, 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 $\xi$ clusters within phase $\phi$ (recall that $|S_{z_j,p}| \leq \xi$). 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 $\xi$ 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 $\zeta$, it holds that $\zeta \geq 2\xi^k \ln m$. Thus, $T_x$ does not get enabled in phase $\phi$ with probability at most

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

□

**Lemma 9** *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 $\xi \leq 40 \ln m$, from Lemma 8 each transaction will execute at the phase its cluster gets assigned with probability at least $1 - 1/m^2$. Therefore, all $n \leq m$ 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 7, we get that with probability at least $1 - 2/m$ all transactions complete execution by the end of the last ($\psi$th) 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 $\beta$ 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 $\zeta$ 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 4 (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 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 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 9 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 4 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 4 gives a poly-log approximation for the optimal execution time.

**Corollary 1** *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 $\alpha$ rays where each ray is a line graph with $\beta$ nodes (see Figure 4). 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

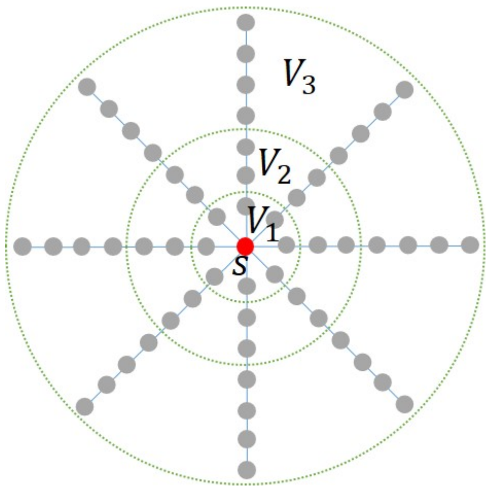

Fig. 4: A star graph with 8 rays, each ray consisting of 7 nodes; the rings depict the set of segments $V_1, V_2, V_3$.

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 $v_\beta$ is the furthest from $s$. The $i$th segment of $r$, where $1 \leq i \leq \eta$, consists of the nodes $v_{2^{i-1}}, \ldots, v_{2^i-1}$. Note that the last segment may be truncated. The $i$th segment, $i < \eta$, has $2^{i-1}$ nodes, while the last segment has no more than $\beta/2 + 1$ nodes. Every node of the $i$th 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 $i$th segment in each ray (see the rings in Figure 4). There are $\eta$ periods where the $i$th period is dedicated to execute the transactions in $V_i$.

Consider now the $i$th 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 $\sigma_i$ 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) 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 9) 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 $\eta$ 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 5 (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 5 gives a schedule which is within a poly-log factor from optimal.

**Corollary 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 $\beta$ 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 Figure 5). 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

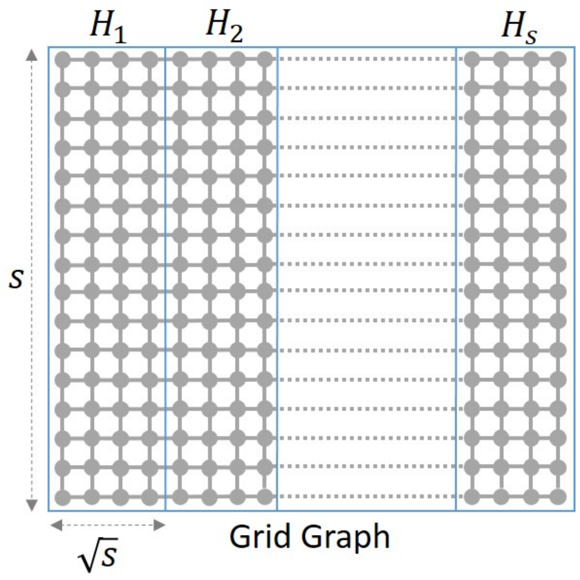

Fig. 5: Grid graph

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 $1 \leq i < s$, 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 $1 \leq i \leq s$. 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 $1 \leq i \leq s$. 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 $1 \leq i \leq s$. 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 $\ell_i$ 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 10** $\Pr(\ell \leq 5s^2) > 1 - s^2 e^{-\sqrt{s}/12}$.

*Proof* Let $\Psi_{i,j}$ denote the set of transactions that pick object $b_i \in B$ in block $H_j$. A node in $H_j$ picks object $b_i$ with probability $1/|B| = 1/s$. Therefore, on expectation, there are $\mu = s\sqrt{s}/s = \sqrt{s}$ nodes that pick object $b_i$ in $H_j$. Hence, from Equation 1, by setting $\delta = 1/2$, we get $\Pr(|\Psi_{i,j}| \leq 3\sqrt{s}/2) > 1 - e^{-\sqrt{s}/12}$.

Within $H_j$ we can form a path $p_{i,j}$ that connects all the nodes of $\Psi_{i,j}$ through horizontal paths of length at most $\sqrt{s} - 1$, to the leftmost column of the block (which has length $s-1$). The path has length at most $(s-1) + |\Psi_{i,j}|(\sqrt{s}-1) \leq (s-1) + (3\sqrt{s}/2)(\sqrt{s}-1) < 5s/2$, with probability greater than $1 - e^{-\sqrt{s}/12}$.

To connect the paths $p_{i,j}$ and $p_{i,j+1}$ in respective consecutive blocks $H_j$ and $H_{j+1}$, we only need to use a joining path of length at most $(\sqrt{s}-1)+s < s+\sqrt{s}$, that connects the top leftmost nodes from each block. Therefore, connecting all the paths together gives a joined path $p_i$ for $b_i$ of length at most $s(5s/2) + (s-1)(s+\sqrt{s}) < 5s^2/2 + s^2 + s\sqrt{s} < 5s^2$, since there are $s$ blocks where $b_i$ is used by at most $5s/2$ transactions in each, and there $s-1$ consecutive joining paths between them of length at most $s+\sqrt{s}$ each. This result holds with probability greater than $1 - se^{-\sqrt{s}/12}$ (union bound for $s$ blocks). The result follows by taking the union bound for all $s$ objects in $B$. □

**Lemma 11** *Any set of $\lambda$ transactions, where $s^{3/8} \leq \lambda \leq s$ and $s \geq e^{7 \cdot 80}$, 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 $1 - s^{-s^{3/8}/161}$.*

*Proof* Consider a period $W$ of $s$ time steps. Note that during $W$ no object in $B$ can be used by transactions in two different blocks $H_i$ and $H_j$, since the distance of any two nodes from the respective blocks is at least $s$. Any set of $\lambda$ transactions during $W$ in $H_i$ execute sequentially since they all share the common object $a_i \in A$. Therefore, $1 \leq \lambda \leq s$.

During period $W$ the common object $a_i$ may follow a path consisting of at most $s$ nodes in $H_i$, out of which $\lambda$ nodes contain transactions which execute in $W$. The number of possible such paths are at most

$$\begin{aligned} & s\sqrt{s} \cdot \binom{s+\lambda}{\lambda} 2^\lambda \cdot \binom{\sqrt{s}+\lambda}{\lambda} 2^\lambda \\ & \leq s^{\frac{3}{2}} \left(\frac{e(s+\lambda)}{\lambda}\right)^\lambda \left(\frac{e(\sqrt{s}+\lambda)}{\lambda}\right)^\lambda 4^\lambda \\ & \leq s^{\frac{3}{2}} \left(\frac{8e^2 s(\sqrt{s}+\lambda)}{\lambda^2}\right)^\lambda . \end{aligned} \tag{3}$$

since there are $s\sqrt{s}$ starting nodes, and the total possible vertical displacements (vertical offsets) between consecutive $\lambda$ nodes is at most $\binom{s+\lambda}{\lambda}$ and the signs of the displacements (positive or negative) is $2^\lambda$. In the horizontal dimension, the number of displacements is at most $\binom{\sqrt{s}+\lambda}{\lambda}$, since the block has at most $\sqrt{s}$ columns.

Consider an arbitrary set $Q$ of $\lambda = |Q|$ transactions in $H_i$. We will give an upper bound on the probability that the transactions in $Q$ use less than $\xi$ different objects of $B$, where $1 \leq \xi \leq s$. This is equivalent to bounding the probability that the transactions in $Q$ do not use at all more than $s-\xi$ objects of $B$.

Each node in $p$ picks one of the $s$ objects in $B$ with probability $1/s$. The probability that in any specific subset $B' \subseteq B$, such that $s-j = |B'|$, no object is picked at all by some specific transaction in $Q$ is $j/s$. Thus, no transaction in $Q$ picks none of the objects in $B'$ with probability $(j/s)^{|Q|} = (j/s)^{\lambda}$, since different transactions pick objects independent of each other.

Hence, in the $\binom{s}{s-j}$ possible subsets of $B$ with $s-j$ objects, the probability that none of the $s-j$ objects are picked at all is $\binom{s}{s-j}(j/s)^{\lambda}$. Considering now all subsets of $B$ with $j < \xi$, we obtain that the probability that $Q$ does not use at all more than $s-\xi$ objects is at most

$$\begin{aligned}\sum_{j=0}^{\xi-1}\binom{s}{s-j}\left(\frac{j}{s}\right)^{\lambda} &\le \xi\binom{s}{s-\xi}\left(\frac{\xi}{s}\right)^{\lambda}\\ &\le \xi\binom{s}{\xi}\left(\frac{\xi}{s}\right)^{\lambda}\\ &\le \xi\left(\frac{es}{\xi}\right)^{\xi}\left(\frac{\xi}{s}\right)^{\lambda}. \end{aligned}\tag{4}$$

Combining Equations 3 and 4 we get that the probability that any set of $\lambda$ nodes uses less than $\xi$ unique objects of $B$ is at most:

$$\begin{aligned}&\xi\left(\frac{es}{\xi}\right)^{\xi}\left(\frac{\xi}{s}\right)^{\lambda}\cdot s^{\frac{3}{2}}\left(\frac{8e^2 s(\sqrt{s}+\lambda)}{\lambda^2}\right)^{\lambda}\\ &\le e^{6\lambda}\xi^{\lambda}s^{\xi+\frac{3}{2}}(\sqrt{s}+\lambda)^{\lambda}\lambda^{-2\lambda}.\end{aligned}$$

By Setting $\xi = \lambda^{3/5}$, this bound becomes:

$$e^{6\lambda}\lambda^{\frac{3\lambda}{5}}s^{\lambda^{3/5}+\frac{3}{2}}(\sqrt{s}+\lambda)^{\lambda}\lambda^{-2\lambda} = e^{6\lambda}s^{\lambda^{3/5}+\frac{3}{2}}(\sqrt{s}+\lambda)^{\lambda}\lambda^{-\frac{7\lambda}{5}}.$$

For the case $\sqrt{s} \le \lambda \le s$, and $s \ge e^{2\cdot 5\cdot 7}$ we obtain that the probability is at most

$$\begin{aligned}e^{6\lambda}\lambda^{2\left(\lambda^{3/5}+\frac{3}{2}\right)}(2\lambda)^{\lambda}\lambda^{-\frac{7\lambda}{5}} &\le e^{7\lambda}\lambda^{2\lambda^{3/5}+3-\frac{2\lambda}{5}}\\ &\le \lambda^{\frac{\lambda}{5}}\lambda^{2\lambda^{3/5}+3-\frac{2\lambda}{5}}\\ &\le \lambda^{-\frac{\lambda}{10}+3}\\ &\le s^{-\frac{\sqrt{s}}{10}+3}.\end{aligned}$$

For the case $s^{3/8} \le \lambda < \sqrt{s}$, and $s \ge e^{7\cdot 80}$ we obtain that the probability is at most

$$\begin{aligned}e^{6\lambda}s^{\lambda^{3/5}+\frac{3}{2}}(2\sqrt{s})^{\lambda}\left(s^{\frac{3}{8}}\right)^{-\frac{7\lambda}{5}} &\le s^{\frac{\lambda}{80}}s^{\frac{\lambda}{2}+\lambda^{3/5}+\frac{3}{2}}s^{-\frac{21\lambda}{40}}\\ &\le s^{\lambda^{3/5}+\frac{3}{2}-\frac{\lambda}{80}}\\ &\le s^{-\frac{\lambda}{160}+\frac{3}{2}}\\ &\le s^{-\frac{s^{\frac{3}{8}}}{160}+\frac{3}{2}}.\end{aligned}$$

Considering now the union bound for all $s$ blocks, this probability becomes at most $ss^{-\frac{s^{\frac{3}{8}}}{160}+\frac{3}{2}} \le s^{-\frac{s^{\frac{3}{8}}}{161}}$. $\square$

The union probability of Lemmas 10 and 11 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 $\lambda$ executed transactions use at least $\lambda^{3/5}$ different objects of $B$. Therefore, we have the following result:

**Corollary 3** *For any $s \geq e^{7 \cdot 80}$, 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 $\lambda$ transactions within any block $H_i$, where $s^{3/8} \leq \lambda \leq s$, 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 3.

**Theorem 6 (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 $O(s^2) = O(n^{4/5})$.*

*Proof* Let $I_s$ be a problem as given by Corollary 3. Consider an arbitrary time window $W$ of $s$ time steps. It suffices to prove that it is impossible to execute $q = 2s^{23/16}(1 + \log s)$ transactions or more during $W$ (where log is base 2).

For the sake of contradiction suppose that at least $q$ transactions execute during $W$. We divide the set of transactions which execute in $W$ into sets $Q_1, \ldots, Q_s$, such that set $Q_i$ consists of all transactions which execute in block $H_i$. Clearly, $\sum_{i=1}^{s} |Q_i| \geq q$.

Any two pairs of sets $Q_i$ and $Q_j$, $i \neq j$, cannot share any object during period $W$, since the minimum distance in $G$ between any two nodes in the respective blocks $H_i$ and $H_j$ is at least $s$, and since the duration of $W$ is $s$ there is not enough time to transfer any object between the two blocks.

Let $R_i$ denote the set consisting of all $Q_j$ such that $|Q_j| \in [2^{i-1}, 2^i)$. Denote $\rho_i = \sum_{Q_j \in R_i} |Q_j|$. We have that $\sum_i \rho_i \geq q$. No more than $s$ transactions can execute in any $Q_j$ within period $W$, since all transactions in $Q_j$ share the internal object $a_j \in A$ in $H_j$; therefore, $|Q_j| \leq s$. Thus, the number of non-empty $R_i$ is at most $1 + \log s$.

There is a $R_y$, $1 \leq y \leq 1 + \log s$, such that $\rho_y \geq q/(1 + \log s)$, since otherwise, $\sum_i \rho_i < (q/(1 + \log s))(1 + \log s) = q$. Since $|R_y| 2^y > \rho_y$, it holds $|R_y| > (q/(1 + \log s))/2^y = q/(2^y(1 + \log s))$.

Note that it has to be that $2^{y-1} \geq s^{3/8}$, since otherwise, $\rho_y < 2|R_y|2^{y-1} < 2ss^{3/8} = 2s^{11/8} < q/(1 + \log s)$. Therefore, for each $Q_j \in R_y$, $|Q_j| \geq 2^{y-1} \geq s^{3/8}$. Hence, from Corollary 3, each set $Q_j \in R_y$ requires at least $|Q_j|^{3/5}$ unique objects of set $B$ to be used in block $H_j$. Since objects from different sets in

$R_y$ are disjoint, the total number of different objects in $B$ used during $W$ is at least

$$\begin{aligned}
\sum_{Q_j \in R_y} |Q_j|^{3/5} &\geq \sum_{Q_j \in R_y} 2^{\frac{3(y-1)}{5}} \\
&= |R_y| 2^{\frac{3(y-1)}{5}} \\
&> \frac{q \cdot 2^{\frac{3(y-1)}{5}}}{2^y(1+\log s)} \\
&= \frac{q}{2^{\frac{2y+3}{5}}(1+\log s)} \\
&\geq \frac{q}{2^{\frac{2(1+\log s)+3}{5}}(1+\log s)} \\
&= \frac{2s^{23/16}(1+\log s)}{2s^{2/5}(1+\log s)} \\
&= s^{83/80} \\
&> s.
\end{aligned}$$

This is a contradiction since $|B| = s$. □

## 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 Figure 6). 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. Particularly, in Lemma 10 the nodes for the objects within each block are assumed to get connected trough a path with the leftmost vertical path, which is still feasible in the tree block, and across blocks the path is formed from the top row which is also still feasible from the top row. Thus, Lemma 10 still applies for the tree. Similarly, the assumptions for the paths in the proof of Lemma 11 remain identical for the tree as well.

Therefore, similar to Theorem 6, 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 $O(n^{4/5})$.

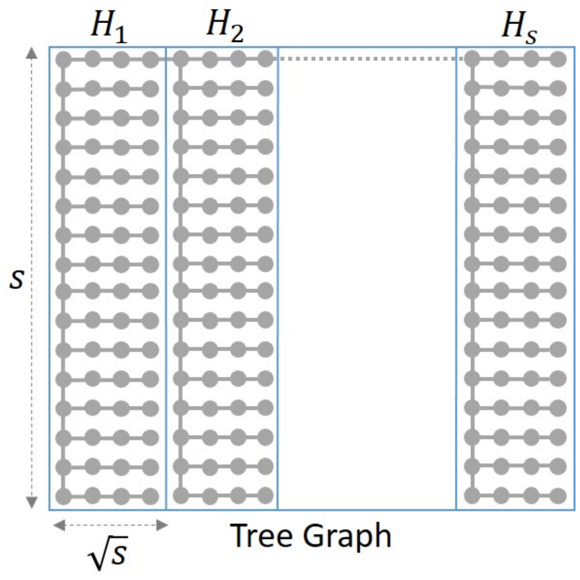

Fig. 6: Tree graph

## 9 Conclusion

We presented a comprehensive set of upper bounds for various specialized network topologies. We also provided a lower bound analysis which shows that it is impossible to optimize the execution time by simply following a schedule in length proportional to the TSP tours for the objects. The lower bound applies to grids and trees and depicts the difficulty of the scheduling problem. Note that the lower bound does not contradict the upper bounds of the specialized graphs and scheduling problems we considered in this work.

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, which we state as follows.

- 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, and are continuously executed over time.
- It would be also interesting to examine the impact of network congestion, where network links have bounded capacity.
- Finally, it would be interesting to establish an execution time lower bound for special topologies beyond trees and grids. Particularly, it would be interesting to start with whether $\Omega(k)$ lower bound can be shown for scheduling in a complete graph. This would show that the schedule provided by Theorem 1 is asymptotically optimal (w.r.t. execution time). The goal will then be to extend to other graphs.

## Acknowledgments

This work is supported by the National Science Foundation grants 1320835, 1420673, and 1936450 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. Transactional contention management as a non-clairvoyant scheduling problem. *Algorithmica*, 57(1):44–61, 2010.
2. Robert L. Bocchino, Vikram S. Adve, and Bradford L. Chamberlain. Software transactional memory for large scale clusters. In *PPoPP*, pages 247–258, 2008.
3. Costas Busch, Maurice Herlihy, Miroslav Popovic, and Gokarna Sharma. Impossibility results for distributed transactional memory. In *PODC*, pages 207–215, 2015.
4. Costas Busch, Maurice Herlihy, Miroslav Popovic, and Gokarna Sharma. Fast scheduling in distributed transactional memory. In *SPAA*, pages 173–182, 2017.
5. Harold W. Cain, Maged M. Michael, Brad Frey, Cathy May, Derek Williams, and Hung Q. Le. Robust architectural support for transactional memory in the power architecture. In *ISCA*, pages 225–236, 2013.
6. Henri Casanova, Arnaud Legrand, and Yves Robert. *Parallel Algorithms.* Chapman & Hall/CRC, 1st edition, 2008.
7. M. Y. Chan. Embedding of d-dimensional grids into optimal hypercubes. In *SPAA*, pages 52–57, 1989.
8. Paolo Costa, Hitesh Ballani, Kaveh Razavi, and Ian Kash. R2c2: A network stack for rack-scale computers. In *SIGCOMM*, pages 551–564, 2015.
9. Maria Couceiro, Paolo Romano, Nuno Carvalho, and Luís Rodrigues. D2stm: Dependable distributed software transactional memory. In *PRDC*, pages 307–313, 2009.
10. Aleksandar Dragojević, Rachid Guerraoui, Anmol V. Singh, and Vasu Singh. Preventing versus curing: Avoiding conflicts in transactional memories. In *PODC*, pages 7–16, 2009.
11. Michel Dubois, Murali Annavaram, and Per Stenstrm. *Parallel Computer Organization and Design.* Cambridge University Press, New York, NY, USA, 2012.
12. Wilson W. L. Fung, Inderpreet Singh, Andrew Brownsword, and Tor M. Aamodt. Hardware transactional memory for gpu architectures. In *MICRO*, pages 296–307, 2011.
13. Rachid Guerraoui, Maurice Herlihy, and Bastian Pochon. Toward a theory of transactional contention managers. In *PODC*, pages 258–264, 2005.
14. 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. The ibm blue gene/q compute chip. *IEEE Micro*, 32(2):48–60, 2012.
15. Danny Hendler, Alex Naiman, Sebastiano Peluso, Francesco Quaglia, Paolo Romano, and Adi Suissa. Exploiting locality in lease-based replicated transactional memory via task migration. In *DISC*, pages 121–133, 2013.
16. Maurice Herlihy and J. Eliot B. Moss. Transactional memory: Architectural support for lock-free data structures. In *ISCA*, pages 289–300, 1993.
17. Maurice Herlihy and Ye Sun. Distributed transactional memory for metric-space networks. *Distributed Computing*, 20(3):195–208, 2007.
18. William N. Scherer III and Michael L. Scott. Advanced contention management for dynamic software transactional memory. In *PODC*, pages 240–248, 2005.
19. Intel. http://software.intel.com/en-us/blogs/2012/02/07/transactional-synchronization-in-haswell, 2012.
20. Junwhan Kim and B. Ravindran. Scheduling transactions in replicated distributed software transactional memory. In *CCGrid*, pages 227–234, 2013.
21. Junwhan Kim and Binoy Ravindran. On transactional scheduling in distributed transactional memory systems. In *SSS*, pages 347–361, 2010.
22. F. Thomson Leighton. *Introduction to Parallel Algorithms and Architectures: Array, Trees, Hypercubes.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992.

23. Dawei Li, Jie Wu, Zhiyong Liu, and Fa Zhang. Towards the tradeoffs in designing data center network architectures. *IEEE Transact. Parallel and Distrib. Syst.*, 28(1):260–273, January 2017.
24. Kaloian Manassiev, Madalin Mihailescu, and Cristiana Amza. Exploiting distributed version concurrency in a transactional memory cluster. In *PPoPP*, pages 198–208, 2006.
25. Marek Michalewicz, Lukasz Orlowski, and Yuefan Deng. Creating interconnect topologies by algorithmic edge removal: Mod and smod graphs. *Supercomput. Front. Innov.: Int. J.*, 2(4):16–47, March 2015.
26. Takuya Nakaike, Rei Odaira, Matthew Gaudet, Maged M. Michael, and Hisanobu Tomari. Quantitative comparison of hardware transactional memory for blue gene/q, zenterprise ec12, intel core, and POWER8. In *ISCA*, pages 144–157, 2015.
27. Roberto Palmieri, Sebastiano Peluso, and Binoy Ravindran. Transaction execution models in partially replicated transactional memory: The case for data-flow and control-flow. In *Transactional Memory*, pages 341–366. Springer, 2015.
28. Sudeep Pasricha and Nikil Dutt. *On-Chip Communication Architectures: System on Chip Interconnect*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.
29. Sebastiano Peluso, Pedro Ruivo, Paolo Romano, Francesco Quaglia, and Luís Rodrigues. When scalability meets consistency: Genuine multiversion update-serializable partial data replication. In *ICDCS*, pages 455–465, 2012.
30. Paolo Romano, Roberto Palmieri, Francesco Quaglia, Nuno Carvalho, and Luís Rodrigues. On speculative replication of transactional systems. *J. Comput. Syst. Sci.*, 80(1):257–276, 2014.
31. Mohamed M. Saad and Binoy Ravindran. Snake: Control flow distributed software transactional memory. In *SSS*, pages 238–252, 2011.
32. Gokarna Sharma and Costas Busch. A competitive analysis for balanced transactional memory workloads. *Algorithmica*, 63(1-2):296–322, 2012.
33. Gokarna Sharma and Costas Busch. Window-based greedy contention management for transactional memory: Theory and practice. *Distrib. Comput.*, 25(3):225–248, 2012.
34. Gokarna Sharma and Costas Busch. Distributed transactional memory for general networks. *Distrib. Comput.*, 27(5):329–362, 2014.
35. Nir Shavit and Dan Touitou. Software transactional memory. *Distrib. Comput.*, 10(2):99–116, 1997.
36. Richard M. Yoo and Hsien-Hsin S. Lee. Adaptive transaction scheduling for transactional memory systems. In *SPAA*, pages 169–178, 2008.
37. Bo Zhang, Binoy Ravindran, and Roberto Palmieri. Distributed transactional contention management as the traveling salesman problem. In *SIROCCO*, pages 54–67, 2014.