

Window-Based Greedy Contention Management for Transactional Memory

Gokarna Sharma¹, Brett Estrade², and Costas Busch¹

¹ Department of Computer Science, Louisiana State University,
Baton Rouge, LA 70803, USA. {gokarna,busch}@csc.lsu.edu

² Department of Computer Science, University of Houston,
501 Philip G. Hoffman Hall, Houston, TX 77204, USA. estrabd@cs.uh.edu

Abstract. We consider greedy contention managers for transactional memory for $M \times N$ *execution windows of transactions* with M threads and N transactions per thread. Assuming that each transaction has duration τ and conflicts with at most C other transactions inside the window, a trivial greedy contention manager can schedule them within τCN time. In this paper, we explore the theoretical performance boundaries of this approach from the worst-case perspective. Particularly, we present and analyze two new randomized greedy contention management algorithms. The first algorithm **Offline-Greedy** produces a schedule of length $O(\tau \cdot (C + N \log(MN)))$ with high probability, and gives competitive ratio $O(\log(MN))$ for $C \leq N \log(MN)$. The offline algorithm depends on knowing the conflict graph which evolves while the execution of the transactions progresses. The second algorithm **Online-Greedy** produces a schedule of length $O(\tau \cdot (C \log(MN) + N \log^2(MN)))$, with high probability, which is only a $O(\log(NM))$ factor worse, but does not require knowledge of the conflict graph. Both of the algorithms exhibit competitive ratio very close to $O(s)$, where s is the number of shared resources. Our algorithms provide new tradeoffs for greedy transaction scheduling that parameterize window sizes and transaction conflicts within the window.

1 Introduction

Multi-core architectures present both an opportunity and challenge for multi-threaded software. The opportunity is that threads will be available to an unprecedented degree, and the challenge is that more programmers will be exposed to concurrency related synchronization problems that until now were of concern only to a selected few. Writing concurrent programs is difficult because of the complexity of ensuring proper synchronization. Conventional lock based synchronization suffers from well known limitations, so researchers considered non-blocking transactions as an alternative. Software Transactional Memory (STM) [22,13,14] systems use lightweight and composable in-memory software transactions to address concurrency in multi-threaded systems ensuring safety all the time [10,11].

In transactional memory (TM) systems, a *contention management* strategy is responsible for the system as a whole to make progress [13,18]. If transaction T discovers that it conflicts with another transaction T' (because they share a resource), it has two choices, it can give T' a chance to finish by aborting itself, or it can proceed by forcing T' to abort; the aborted transaction then retries again until it eventually commits. To solve this scheduling problem efficiently, T will consult the *contention manager* module for which choice to make. Of particular interest are *greedy contention managers* where a transaction starts again immediately after every abort. Several (greedy) contention managers have been proposed in the literature [3,9,7,5,21]. However, most contention managers have been assessed only experimentally by specific benchmarks [18]. There is a small amount of work in the literature which analyzes formally the performance of contention managers [3,9,7,21].

The competitive ratio results are not encouraging. For example in [3] the authors give an $O(s)$ competitive ratio bound, where s is the number of resources. When the number of resources s increases, the performance degrades linearly. A difficulty in obtaining tight bounds is that the algorithms studied in [3,9,7,5,21] apply to the *one-shot scheduling problem*, where each thread issues a single transaction. One-shot problems are directly related with vertex coloring, where the problem of determining the chromatic number of a graph is reduced to finding an optimal time schedule for the one-shot problem. Since it is known that computing an optimal coloring given complete knowledge of the graph is a very hard problem to approximate, the one-shot problem is very hard to approximate too [15].

In order to obtain alternative and improved formal bounds, we propose to investigate the performance of program executions in *windows of transactions* (see Fig. 1a), which has the potential to overcome some of the limitations of the coloring reduction in certain circumstances. A $M \times N$ window W consists of M threads with an execution sequence of N different transactions per thread. The execution window W can be viewed as a collection of N one-shot transaction sets with M concurrent transactions in each set. Let C denote the maximum number of conflicting transactions for any transaction in the window. If we assume that all transactions have the same duration τ , then a straightforward upper bound for the makespan of the window is $\tau \cdot \min(CN, MN)$, since τCN follows from the observation that each transaction in a thread may be delayed at most C times by its conflicting transactions, and τMN follows from the serialization of the transactions. The competitive ratio of the makespan using the one-shot analysis results is bounded by $O(sN)$. Using the one-shot Algorithm `RandomizedRounds` provided in [21] N times, the completion time is in the worst case $O(\tau CN \log M)$.

In this paper, we show that we can obtain better results by utilizing the window representation. We present a family of randomized greedy algorithms where transactions are assigned priorities values, such that for some random initial interval in the beginning of the window W each transaction is in low priority mode and then after the random period expires the transactions switch to high priority mode. In high priority mode the transaction can only be aborted by other

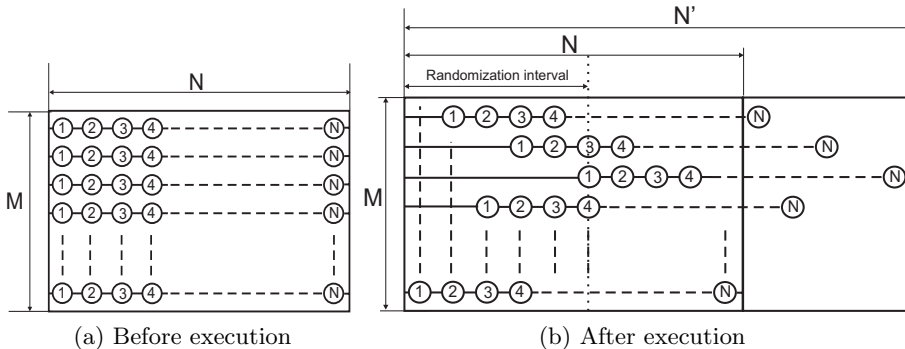


Fig. 1: Execution window model for transactional memory

high priority transactions. The random initial delays have the property that the conflicting transactions are shifted inside their window and their execution times may not coincide (see Fig. 1b). The benefit is that conflicting transactions can execute at different time slots and potentially many conflicts are avoided. The benefits become more apparent in scenarios where the conflicts are more frequent inside the same column transactions and less frequent between different column transactions.

Contributions: We propose the contention measure C within the window to allow more precise statements about the worst-case complexity bound of any contention management algorithm. We give two window-based randomized greedy algorithms for the contention management in any execution window W . For simplicity, we assume that each transaction has the same duration τ (this assumption can be removed). Our first algorithm **Offline-Greedy** gives a schedule of length $O(\tau \cdot (C + N \log(MN)))$ with high probability, and improves on one-shot contention managers from a worst-case perspective. The algorithm is offline in the sense that it uses explicitly the dynamic conflict graph of the transactions at each time step of execution to resolve the conflicts. Our second algorithm **Online-Greedy** produces a schedule of length $O(\tau \cdot (C \log(MN) + N \log^2(MN)))$ with high probability, which is only a factor of $O(\log(MN))$ worse in comparison to **Offline-Greedy**. The benefit of the online algorithm is that it does not need to know the conflict graph of the transactions to resolve the conflicts. The online algorithm uses as a subroutine a variation of algorithm **RandomizedRounds** [21].

We also give a third algorithm **Adaptive-Greedy** which is the adaptive version of the previous algorithms which achieves similar worst-case performance and adaptively guesses the value of the contention measure C . The technique we use for the analysis of these algorithms is similar to the one used by Leighton *et al.* [16] to analyze an online packet scheduling problem.

An advantage of our algorithms is that if the conflicts in the window are bounded by $C \leq N \log(MN)$ then the upper bounds we have obtained are

within poly-logarithmic factors from optimal, since N is a trivial lower bound in execution time. This is an improvement over the trivial approach of using N one-shot executions. We also show that the offline algorithm is $O(s + \log(MN))$ -competitive and the online algorithm is $O(s \cdot \log(MN) + \log^2(MN))$ -competitive (for any choice of C).

Outline of Paper: The rest of the paper is organized as follows: In Section 2, we discuss the related work. We present the transactional memory model in Section 3. We present and formally analyze an offline randomized greedy algorithm in Section 4. The online version of the randomized greedy algorithm is given in Section 5. In Section 6, we describe the adaptive version of the aforementioned algorithms. Section 7 concludes the paper.

2 Related Work

Transactional Memory (TM) has been proposed in the early nineties as an alternative implementation of mutual exclusion that avoids many of the drawbacks of locks e.g., deadlock, reliance on the programmer to associate shared data with locks, priority inversion, and failures of threads while holding locks [14]. In 2003, Dynamic STM (DSTM) [13] was proposed for dynamic-sized data structures which uses a contention manager module as an independent module to resolve conflicts between two transactions and ensure progress. DSTM is a practical obstruction-free³ STM system that seeks advice from the contention manager module to either wait or abort an transaction at the time of conflict.

As TM has been gaining attention, several contention managers are available in the literature [3,9,7,5,21]. Most of them have been assessed by specific benchmarks only and not analytically. A comparison of contention managers based on different benchmarks can be found in [18,19,17,20,7,8]. They observed that the choice of the best contention manager varies with the complexity of the considered benchmark. The more detailed analysis of the performance of different contention managers in complex benchmarks has recently been studied by Ansari *et al.* [1]. From all the aforementioned references, one can notice that the coordination cost and the overhead involved in contention management is very high.

The first formal analysis of the performance of a contention manager was given by Guerraoui *et al.* [9] by presenting the **Greedy** contention manager which decides in favor of old transactions using timestamps and proving that it achieves $O(s^2)$ competitive ratio in comparison to the optimal off-line schedulers for M concurrent transactions that share s objects. They argue that the bound holds for any algorithm which ensure that at any point in time at least one transaction is running uninterrupted until it commits, which is called the *pending commit* property. Later, Guerraoui *et al.* [7] studied the impact of transaction failures on contention management. They presented the algorithm **FTGreedy** and proved

³ A synchronization mechanism is obstruction-free if any thread runs itself for a long time makes progress [12].

the $O(ks^2)$ competitive ratio when some running transaction may fail at most k times and then eventually commits. Attiya *et al.* [3] improved the result of [9] to $O(s)$, and the result of [7] to $O(ks)$, which are significant improvements over the competitive ratio of Greedy. They also proved the matching lower bound of $\Omega(s)$ for the competitive ratio for deterministic *work-conserving* algorithms which schedule as many transactions as possible (by choosing a maximal independent set of transactions).

Recently, Schneider and Wattenhofer [21] presented a deterministic algorithm **CommitBounds** with competitive ratio $\Theta(s)$ for M concurrent transactions using s shared resources and a randomized algorithm **RandomizedRounds** with makespan $O(C \log M)$, for the one-shot problem of a set of M transactions in separate threads with C conflicts (assuming unit delays for transactions). While previous studies showed that contention managers **Polka** [18] and **SizeMatters** [17] exhibit good overall performance for variety of benchmarks, this work showed that they may perform exponentially worse than **RandomizedRounds** from a worst-case perspective. Another recent proposal for the contention management is **Serializer** [5], which resolves a conflict by removing a conflicting transaction T from the processor core where it was running, and scheduling it on the processor core of the other transaction to which it conflicted with. It is $O(M)$ -competitive and in fact, it ensures that two transactions never conflict more than once.

On the other side, *TM schedulers* [6,23,2,4] offer an alternative approach to boost the TM performance. TM scheduler is, basically, a software component which decides when a particular transaction executes. One proposal in this approach is **Adaptive Transaction Scheduling (ATS)**⁴ [23] scheduler. **Restart** [6] scheduler is another recent proposal whose worst case performance is very sensitive to the accuracy of the prediction of the future conflicts. In [6] they also propose a scheduler called **Shrink** which predicts the future accesses of a based on the past accesses, and dynamically serializes transactions based on the prediction to prevent conflicts. The **ATS**, **Restart** and **Shrink** schedulers are $O(M)$ -competitive in the worst-case. **Steal-On-Abort** [2] is the yet another proposal where the aborted transaction is given to the opponent transaction and queued behind it, preventing the two transactions from conflicting again. Recently, Attiya *et al.* [4] proposed the **BIMODAL** scheduler which alternates between *writing epochs* where it gives priority to writing transactions and *reading epochs* where it gives priority to transactions that have issued only reads so far. It achieves $O(s)$ competitive ratio on bimodal⁵ workloads with equi-length transactions.

3 Execution Window Model

Consider M threads $\mathcal{P} = \{P_1, \dots, P_M\}$. We consider a model that is based on a $M \times N$ execution window W consisting of a set of transactions $\mathcal{T}(W) =$

⁴ **ATS** measures adaptively the contention intensity of a thread, when the contention intensity increases beyond a threshold, it serializes the transactions.

⁵ A workload containing only early-write and read-only transactions (see [4] for details).

$\{(T_{11}, \dots, T_{1N}), (T_{21}, \dots, T_{2N}), \dots, (T_{M1}, \dots, T_{MN})\}$, where each thread P_i issues N transactions T_{i1}, \dots, T_{iN} in sequence, so that T_{ij} is issued as soon as $T_{i(j-1)}$ has committed. If $N = 1$ then this is similar to the one-shot TM model, that uses one transaction per thread.

Transactions share a set of s shared resources $\mathcal{R} = \{R_1, \dots, R_s\}$. Each transaction is a sequence of actions that is either a read or write to some shared resource R . A transaction after it is issued it either commits or aborts. A transaction that has been issued but not committed is said to be pending. Concurrent write-write actions or read-write actions to shared objects by two or more transactions cause conflicts between transactions. If a transactions conflicts then it either aborts, or it may commit and force to abort all other conflicting transactions. In a *greedy schedule*, if a transactions aborts it then immediately restarts and attempts to commit again.

The *makespan* of a schedule for the transactions is defined as the duration from the start of the schedule, i.e., the time when the first transaction is issued, until all transactions have committed. The makespan of the transaction scheduling algorithm for the sequences of transactions can be compared to the makespan of an optimal off-line scheduling algorithm, denoted $\text{makespan}_{\text{opt}}$ to provide a competitive ratio. Each transaction T_{ij} has execution time duration τ_{ij} which is greater than 0. When we describe our algorithms we assume that all transactions have the same duration $\tau = \tau_{ij}$. We also assume that the execution time advances synchronously for all threads, where each time step corresponds to a period of duration τ . We also assume that and all transactions inside the execution window are correct, i.e., there are no faulty transactions⁶. Our results can be extended by relaxing these assumptions. In Section 7, we describe the impact that variable time durations for the transactions have to the performance of our algorithms.

3.1 Conflict Graph

Consider a set of k transactions $\mathcal{T} = \{T_1, \dots, T_k\}$. Let $\mathcal{R}(T_i)$ denote the set of resources used by transaction T_i . We can write $\mathcal{R}(T_i) = \mathcal{R}_w(T_i) \cup \mathcal{R}_r(T_i)$, where $\mathcal{R}_w(T_i)$ are the resources which are to be written by T_i and $\mathcal{R}_r(T_i)$ are the resources to be read by T_i .

Definition 1 (Transaction Conflict). *Two transactions T_i and T_j conflict if at least one of them writes on a common resource, that is, there is a resource R such that $R \in (\mathcal{R}_w(T_i) \cap \mathcal{R}(T_j)) \cup (\mathcal{R}(T_i) \cap \mathcal{R}_w(T_j))$ (we also say that R causes the conflict).*

Definition 2 (Conflict Graph). *The conflict graph $G = (\mathcal{T}, E)$ has nodes the transactions, and $(T_i, T_j) \in E$ for any two transactions T_i, T_j that conflict.*

⁶ A transaction is called faulty when it encounters an illegal instruction producing a segmentation fault or experiences a page fault resulting to wait for a long time for the page to be available [7].

Let $\delta(T_i)$ denote the degree of node T_i in G . We denote $C = \max_i \delta(T_i)$. Let $\gamma(R_j)$ denote the number of transactions that write R_j . Let $\gamma_{\max} = \max_j \gamma(R_j)$. Let $\lambda(T_i) = |\{R : R \in \mathcal{R}(T_i) \wedge (\gamma(R) \geq 1)\}|$, denote the number of resources that can be the cause of conflicts to transaction T_i . Let $\lambda_{\max} = \max_i \lambda(T_i)$. Note that $C \leq \lambda_{\max} \cdot \gamma_{\max}$, and, $C \geq \gamma_{\max}$.

Assuming that there is one transaction per thread, the conflict graph can be used to obtain a simple greedy schedule of the transactions as follows. Compute a $C + 1$ vertex coloring of the conflict graph. All transactions of same color can commit simultaneously. The transactions can be scheduled in a greedy manner by giving a different priority to each transaction color. This produces a greedy schedule of length $Makespan = \tau \cdot (C + 1)$. Since $C \leq \lambda_{\max} \cdot \gamma_{\max}$, we have that $makespan \leq \tau \cdot (\lambda_{\max} \cdot \gamma_{\max} + 1)$. Further, since $C \geq \gamma_{\max}$, $makespan_{\text{opt}} \geq \tau \cdot \gamma_{\max}$. Since $\lambda_{\max} \leq s$, the competitive ratio of the schedule is $\lambda_{\max} + 1 = O(s)$.

4 Offline Algorithm

We present the Algorithm Offline-Greedy (Algorithm 1), which is an offline greedy contention resolution algorithm that it uses the conflict graph explicitly to resolve conflicts of transactions. In addition to M and N , we assume that each thread P_i knows C_i , which denotes the maximum number of transactions that any transaction in P_i conflicts with; namely, using the conflict graph $G(\mathcal{T}(W))$, $C_i = \max_j \delta(T_{ij})$. Note that $C = \max_i C_i$.

Time is measured in discrete time steps, where each time step represents the duration τ of the transactions. We divide time into *frames*, which are time periods of duration $\Theta(\tau \cdot \ln(MN))$ (namely, each frame consists of $\Phi = \Theta(\ln(MN))$ time steps). Then, each thread P_i is assigned an initial random time period consisting of q_i frames, where q_i is chosen randomly, independently and uniformly, from the range $[0, \alpha_i - 1]$, where $\alpha_i = C_i / \ln(MN)$. Each transaction has two priorities: *low* or *high*. Transaction T_{ij} is initially in low priority. Transaction T_{ij} switches to high priority in the first time step of frame $F_{ij} = q_i + (j - 1)$ (this is the assigned frame for T_{ij}) and remains in high priority thereafter until it commits. In the analysis, we show that with high probability each transaction commits in its assigned frame.

The priorities are used to resolve conflicts. A high priority transaction may only be aborted by another high priority transaction. A low priority transaction is always aborted if it conflicts with a high priority transaction. Let G_t denote the conflict graph of transactions at time step t which evolves while the execution of the transactions progresses. Note that the maximum degree of G_t is bounded by C , but the effective degree between high priority transactions is lower. At each time step t we select to commit a maximal independent set of transactions in G_t . We first select a maximal independent set I_H of high priority transactions, then remove this set and its neighbors from G_t , and then select a maximal independent set I_L of low priority transactions from the remaining conflict graph. The transactions that commit are $I_H \cup I_L$.

Algorithm 1: Offline-Greedy

Input: A $M \times N$ window W of transactions with M threads each with N transactions; Each thread P_i knows C_i , the maximum number of transactions in W that any transaction in P_i conflicts with; Each transaction has same duration τ ;

Divide time into frames consisting of $\Phi = 1 + (e^2 + 2) \ln(MN)$ time steps;
Each thread P_i chooses a random number $q_i \in [0, \alpha_i - 1]$ for $\alpha_i = C_i / \ln(MN)$;
Each transaction T_{ij} is assigned to frame $F_{ij} = q_i + (j - 1)$;

foreach *time step* $t = 0, 1\tau, 2\tau, 3\tau, \dots$ **do**

Phase 1: Priority Assignment

foreach *transaction* T_{ij} **do**

if $t < F_{ij} \cdot \tau\Phi$ **then** $Priority(T_{ij}) \leftarrow 1$ (*low*); **else**

$Priority(T_{ij}) \leftarrow 0$ (*high*);

Phase 2: Conflict Resolution

begin

 Let G_t be the conflict graph at time t ;

 Compute G_t^H and G_t^L , the subgraphs of G_t induced by high and low priority nodes, respectively;

 Compute $I_H \leftarrow I(G_t^H)$, maximal independent set of nodes in graph G_t^H ;

$Q \leftarrow$ low priority nodes adjacent to nodes in I_H ;

 Compute $I_L = I(G_t^L \setminus Q)$, maximal independent set of nodes in graph G_t^L after removing Q nodes;

 Commit $I_H \cup I_L$;

The intuition behind the algorithm is as follows. Consider a thread i and its first transaction in the window T_{i1} . According to the algorithm, T_{i1} becomes high priority in the beginning of frame F_{i1} . Because q_i is chosen at random among $C_i / \ln(MN)$ positions it is expected that T_{i1} will conflict with at most $O(\ln(MN))$ transactions in its assigned frame F_{i1} which become simultaneously high priority in F_{i1} . Since a time frame contains $\Phi = \Theta(\ln(MN))$ time steps, transaction T_{i1} and all its high priority conflicting transactions will be able to commit by the end of time frame F_{i1} , using the conflict resolution graph. The initial randomization period of $q_i \cdot \Phi$ frames will have the same effect to the remaining transactions of the thread i , which will also commit within their assigned frames.

4.1 Analysis of Offline Algorithm

We study the makespan of Algorithm Offline-Greedy. According to the algorithm, when a transaction T_{ij} is issued, it will be in low priority until the respective frame F_{ij} starts. As soon as F_{ij} starts, the transaction T_{ij} will begin executing in high priority (if it didn't commit already). Let A denote the set of conflicting transactions with T_{ij} in the conflict graph $G(\mathcal{T}(W))$. Let $A' \subseteq A$ denote the subset of conflicting transactions with T_{ij} which become high priority during frame F_{ij} (simultaneously with T_{ij}).

Lemma 1. *If $|A'| \leq \Phi - 1$ then transaction T_{ij} will commit in frame F_{ij} .*

Proof. Due to the use of the high priority independent sets in the conflict graph G_t , if in time t during frame F_{ij} transaction T_{ij} does not commit, then some conflicting transaction in A' must commit. Since there are at most $\Phi - 1$ high priority conflicting transactions, and the length of the frame F_{ij} is at most Φ , T_{ij} will commit by the end of frame F_{ij} .

We show next that it is unlikely that $|A'| > \Phi - 1$. We use the following Chernoff bound:

Lemma 2 (Chernoff Bound 1). *Let X_1, X_2, \dots, X_n be independent Poisson trials such that, for $1 \leq i \leq n$, $\Pr(X_i = 1) = pr_i$, where $0 < pr_i < 1$. Then, for $X = \sum_{i=1}^n X_i$, $\mu = \mathbf{E}[X] = \sum_{i=1}^n pr_i$, and any $\delta > e^2$, $\Pr(X > \delta\mu) < e^{-\delta\mu}$.*

Lemma 3. *$|A'| > \Phi - 1$ with probability at most $(1/MN)^2$.*

Proof. Let $A_k \subseteq A$, where $1 \leq k \leq M$, denote the set of transactions of thread P_k that conflict with transaction T_{ij} . We partition the threads P_1, \dots, P_M into 3 classes Q_0, Q_1 , and Q_2 , such that:

- Q_0 contains every thread P_k which either $|A_k| = 0$, or $|A_k| > 0$ but the positions of the transactions in A_k are such that it is impossible to overlap with F_{ij} for any random intervals q_i and q_k .
- Q_1 contains every thread P_k with $0 < |A_k| < \alpha_i$, and at least one of the transactions in A_k is positioned so that it is possible to overlap with frame F_{ij} for some choices of random intervals q_i and q_k .
- Q_2 contains every thread P_k with $\alpha_i \leq |A_k|$. Note that $|Q_2| \leq C_i/\alpha_i = \ln(MN)$.

Let Y_k be a random binary variable, such that $Y_k = 1$ if in thread P_k any of the transactions in A_k becomes high priority in F_{ij} (same frame with T_{ij}), and $Y_k = 0$ otherwise. Let $Y = \sum_{k=1}^M Y_k$. Note that $|A'| = Y$. Denote $pr_k = \Pr(Y_k = 1)$. We can write $Y = Z_0 + Z_1 + Z_2$, where $Z_\ell = \sum_{P_k \in Q_\ell} Y_k$, for $0 \leq \ell \leq 2$. Clearly, $Z_0 = 0$, and $Z_2 \leq |Q_2| \leq \ln(MN)$.

Recall that for each thread P_k there is a random initial interval with q_k frames, where q_k is chosen uniformly at random in $[0, \alpha_k - 1]$. Given the random choice of P_k , $0 < pr_k \leq |A_k|/\alpha_i < 1$, since there are $|A_k| < \alpha_i$ conflicting transactions in A_i and there are at least α_i random choices for the relative position of transaction T_{ij} . Consequently,

$$\mu = \mathbf{E}[Z_1] = \sum_{P_k \in Z_1} pr_k \leq \sum_{P_k \in Z_1} \frac{|A_k|}{\alpha_i} = \frac{1}{\alpha_i} \cdot \sum_{P_k \in Z_1} |A_k| \leq \frac{C_i}{\alpha_i} \leq \ln(MN).$$

By applying a Chernoff bound we obtain that $\Pr(Z_1 > (e^2 + 1)\mu) < e^{-(e^2 + 1)\mu} < e^{-2 \ln(MN)} = (MN)^{-2}$. Since $Y = Z_0 + Z_1 + Z_2$, and $Z_2 \leq \ln(MN)$, we obtain $\Pr((|A'| = Y) > ((e^2 + 2)\mu = \Phi - 1)) < (MN)^{-2}$, as needed.

Lemma 4. *All transactions commit by the end of their assigned frames with probability at least $1 - (MN)^{-1}$.*

Proof. From Lemmas 1 and 3, Φ time steps do not suffice to commit transaction T_{ij} within its assigned frame F_{ij} with probability at most $(NM)^{-2}$ (we call this a *bad event*). Considering all the MN transactions in the window a bad event for any of them occurs with probability at most $MN \cdot (MN)^{-2} = (MN)^{-1}$. Thus, with probability at least $1 - (MN)^{-1}$, all transactions will commit within their assigned frames.

Since $C = \max_i C_i$, the makespan bound of the algorithm follows immediately from Lemma 4.

Theorem 1 (Makespan of Offline-Greedy). *Algorithm Offline-Greedy produces a schedule of length $O(\tau \cdot (C + N \log(MN)))$ with probability at least $1 - (MN)^{-1}$.*

Since in the conflict graph $G(\mathcal{T}(W))$, $C \leq \lambda_{\max} \cdot \gamma_{\max}$, we have that $\text{makespan} = O(\tau \cdot (\lambda_{\max} \cdot \gamma_{\max} + N \log(MN)))$. Further, since $C \geq \gamma_{\max}$, and $\tau \cdot N$ is a lower bound on the schedule length, $\text{makespan}_{\text{opt}} \geq \tau \cdot \max(\gamma_{\max}, N)$. Therefore, the competitive ratio of the schedule is $O(\lambda_{\max} + \log(MN)) = O(s + \log(MN))$.

Corollary 1 (Competitive Ratio of Offline-Greedy). *The makespan of the schedule produced by Algorithm Offline-Greedy has competitive ratio $O(s + \log(MN))$ with probability at least $1 - (MN)^{-1}$.*

5 Online Algorithm

We present Algorithm Online-Greedy algorithm (Algorithm 2), which is online in the sense that it does not depend on knowing the dependency graph to resolve conflicts. In addition to M and N , we assume that each thread P_i knows C_i . This algorithm is similar to Algorithm 1 with the difference that in the conflict resolution phase we use as a subroutine a variation of Algorithm RandomizedRounds proposed by Schneider and Wattenhofer [21]. The makespan of the online algorithm is slightly worse than the offline algorithm, since the duration of the frame is now $\Phi' = O(\tau \cdot \log^2(MN))$.

There are two different priorities associated with each transaction under this algorithm. The pair of priorities for a transaction is given as a vector $\langle \pi^{(1)}, \pi^{(2)} \rangle$, where $\pi^{(1)}$ represents the Boolean priority value *low* or *high* (with respective values 1 and 0) as described in Algorithm 1, and $\pi^{(2)} \in [1, M]$ represents the random priorities used in Algorithm RandomizedRounds. The conflicts are resolved in lexicographic order based on the priority vectors, so that vectors with lower lexicographic order have higher priority.

When a transaction T is issued, it starts to execute immediately in low priority ($\pi^{(1)} = 1$) until the respective randomly chosen time frame F starts where it switches to high priority ($\pi^{(1)} = 0$). Once in high priority, the field

Algorithm 2: Online-Greedy

Input: A $M \times N$ window W of transactions with M threads each with N transactions; Each thread P_i knows C_i , the maximum number of transactions in W that any transaction in P_i conflicts with; Each transaction has same duration τ ;

Output: A greedy execution schedule for the window of transactions W ;

Divide time into frames of $\Phi' = 16e \cdot \Phi \ln(MN)$ time steps, where

$\Phi = 1 + (e^2 + 2) \ln(MN)$;

Each thread P_i chooses a random number $q_i \in [0, \alpha_i - 1]$ for $\alpha_i = C_i / \ln(NM)$;

Each transaction T_{ij} is assigned to frame $F_{ij} = q_i + (j - 1)$;

Associate pair of priorities $\langle \pi_{ij}^{(1)}, \pi_{ij}^{(2)} \rangle$ to each transaction T_{ij} ;

foreach time step $t = 0, 1\tau, 2\tau, 3\tau, \dots$ **do**

Phase 1: Priority Assignment

foreach transaction T_{ij} **do**

if $t < F_{ij} \cdot \tau\Phi'$ **then** Priority $\pi_{ij}^{(1)} \leftarrow 1$ (*low*); **else** Priority

$\pi_{ij}^{(1)} \leftarrow 0$ (*high*);

Phase 2: Conflict Resolution

if $\pi_{ij}^{(1)} == 0$ (T_{ij} has high priority) **then**

On (re)start of transaction T_{ij} ;

$\pi_{ij}^{(2)} \leftarrow$ random integer in $[1, M]$;

On conflict of transaction T_{ij} with high priority transaction T_{kl} ;

if $\pi_{ij}^{(2)} < \pi_{kl}^{(2)}$ **then** $abort(T_{ij}, T_{kl})$; **else** $abort(T_{kl}, T_{ij})$;

$\pi^{(2)}$ will be used to resolve conflicts with other high priority transactions. A transaction chooses a discrete number $\pi^{(2)}$ uniformly at random in the interval $[1, M]$ on start of the frame F_{ij} , and after every abort. In case of a conflict of T with another high priority transaction K which has higher $\pi^{(2)}$ value than T , then T proceeds and K aborts. The procedure $abort(T, K)$ aborts transaction K .

5.1 Analysis of Online Algorithm

In the analysis given below, we study the makespan and the response time of Algorithm Online-Greedy. The analysis is based on the following adaptation of the response time analysis of a one-shot transaction problem with algorithm RandomizedRounds [21]. It uses the following Chernoff bound:

Lemma 5 (Chernoff Bound 2). *Let X_1, X_2, \dots, X_n be independent Poisson trials such that, for $1 \leq i \leq n$, $\Pr(X_i = 1) = pr_i$, where $0 < pr_i < 1$. Then, for $X = \sum_{i=1}^n X_i$, $\mu = \mathbf{E}[X] = \sum_{i=1}^n pr_i$, and any $0 < \delta \leq 1$, $\Pr(X < (1 - \delta)\mu) < e^{-\delta^2\mu/2}$.*

Lemma 6. (Adaptation from Schneider and Wattenhofer [21]) *Given a one-shot transaction scheduling problem with M transactions, the time span a*

transaction T needs from the moment it is issued until commit is $16e(d_T+1) \log n$ with probability at least $1 - \frac{1}{n^2}$, where d_T is the number of transactions conflicting with T .

Proof. Consider the respective conflict graph G of the one-shot problem. Let N_T denote the set of conflicting transactions for T (these are the neighbors of T in G). Let $d_T = |N_T| \leq M$. Let y_T denote the random priority number choice of T in range $[1, M]$. The probability that for transaction T no transaction $K \in N_T$ has the same random number is:

$$\Pr(\nexists K \in N_T | y_T = y_K) = \left(1 - \frac{1}{M}\right)^{d_T} \geq \left(1 - \frac{1}{M}\right)^M \geq \frac{1}{e}.$$

The probability that y_T is at least as small as y_K for any transaction $K \in N_T$ is $\frac{1}{d_T+1}$. Thus, the chance that y_T is smallest and different among all its neighbors in N_T is at least $\frac{1}{e(d_T+1)}$. If we conduct $16e(d_T+1) \ln n$ trials, each having success probability $\frac{1}{e(d_T+1)}$, then the probability that the number of successes Z is less than $8 \ln n$ becomes: $\Pr(Z < 8 \cdot \ln n) < e^{-2 \cdot \ln n} = \frac{1}{n^2}$, using the Chernoff bound of Lemma 5.

Lemma 7. *In Algorithm Online-Greedy all transactions commit by the end of their assigned frames with probability at least $1 - 2(MN)^{-1}$.*

Proof. According to the algorithm, a transaction T_{ij} becomes high priority ($\pi_{ij}^{(1)} = 0$) in frame F_{ij} . When this occurs the transaction will start to compete with other transactions which became high priority during the same frame. Lemma 3 from the analysis of Algorithm 1, implies that the effective degree of T_{ij} with respect to high priority transactions is $d_T > \Phi - 1$ with probability at most $(MN)^{-2}$ (we call this *bad event-1*). From Lemma 6, if $d_T \leq \Phi - 1$, the transaction will not commit within $16e(d_T + 1) \log n \leq \Phi'$ time slots with probability at most $(MN)^{-2}$ (we call this *bad event-2*). Therefore, T_{ij} does not commit in F_{ij} when either bad event-1 or bad event-2 occurs, which happens with probability at most $(MN)^{-2} + (MN)^{-2} = 2(MN)^{-2}$. Considering now all the MN transactions, the probability of failure is at most $2(MN)^{-1}$. Thus, with probability at least $1 - 2(MN)^{-1}$, every transaction T_{ij} commits during the F_{ij} frame.

The makespan and competitive ratios follow immediately from Lemma 7.

Theorem 2 (Makespan of Online-Greedy). *Algorithm Online-Greedy produces a schedule of length $O(\tau \cdot (C \log(MN) + N \log^2(MN)))$ with probability at least $1 - 2(MN)^{-1}$.*

Corollary 2 (Competitive Ratio of Online-Greedy). *The makespan of the schedule produced by Algorithm Online-Greedy has competitive ratio $O(s \cdot \log(MN) + \log^2(MN))$ with probability at least $1 - 2(MN)^{-1}$.*

Algorithm 3: Adaptive-Greedy

Input: An $M \times N$ execution window W with M threads each with N transactions, where C is unknown;

Output: A greedy execution schedule for the window of transactions;

Code for thread P_i ;

begin

 Initial contention estimate $C_i \leftarrow 1$;

repeat

 Online-Greedy(C_i, W);

if bad event then

$C_i \leftarrow 2 \cdot C_i$;

until all transactions are committed;

6 Adaptive Algorithm

A limitation of Algorithms 1 and 2 is that the values C_i need to be known in advance for each window W . We present the Algorithm Adaptive-Greedy (Algorithm 3) in which each thread can guess the individual values of C_i . The algorithm works based on the exponential back-off strategy used by many contention managers developed in the literature such as Polka.

Each thread P_i starts with assuming $C_i = 1$. Based on the current estimate C_i , the thread attempts to execute Algorithm 2, for each of its transactions assuming the window size $M \times N$. Now, if the choice of C_i is correct then each transaction of the thread in the window W of the thread P_i should commit by the end of the respective assigned frame that it becomes high priority. Thus, all transactions of thread P_i should commit within the time estimate of Algorithm 2 which is $L_i = O(\tau \cdot (C_i \log(MN) + N \log^2(MN)))$. However, if during L_i thread P_i is unable to commit one of its transactions within its assigned frame (we call this a *bad event*), then thread P_i will assume that the choice of C_i is incorrect, and will start over again with the remaining transactions assuming $C'_i = 2C_i$. Eventually thread P_i will guess the right value of C_i for the window W , and all its transactions will commit within their respective time frames. It is easy to that the correct choice of C_i will be reached by a thread P_i within $\log C_i$ iterations. The total makespan is asymptotically the same as with Algorithm 2.

7 Conclusions

We considered greedy contention managers for transactional memory for $M \times N$ windows of transactions with M threads and N transactions per thread and present three new algorithms for contention management in transactional memory from a worst-case perspective. These algorithms are efficient, adaptive, and improve on the worst-case performance of previous results. These are the first such results for the execution of sequences of transactions instead of the one-shot

problem considered in the literature. Our algorithms present new trade-offs in the analysis of greedy contention managers for transactional memory.

When we consider variable time durations for the transactions, in the makespan bounds expressions in Theorems 1 and 2 of our algorithms we can replace the parameter τ with τ^{\max} , which is the maximum duration of any transaction in the window. The impact is that in the competitive ratio in Corollaries 1 and 2 there will appear an additional factor τ^{\max}/τ^{\min} , where τ^{\min} is the minimum duration of any transaction in the window. In the algorithms, the basic time step duration is changed from τ to τ^{\max} . Note that with variable time delays the transactions are not perfectly aligned when they enter a frame. In *Offline-Greedy*, this doesn't cause a problem when we compute the independent sets. On the other hand, we need to modify *Online-Greedy* so that when a high-priority transaction aborts, it always gives the right of way to the transaction that aborted it.

With this work, we are left with some issues for future work. The other aspect is to explore alternative algorithms where the randomization does not occur at the beginning of each window but rather during the execution of the algorithm by inserting random periods of low priority between the subsequent transactions in each thread. One may also consider the dynamic expansion and contraction of the execution window to preserve the contention measure C . This will result to more practical algorithms with good performance guarantees.

References

1. Ansari, M., Kotselidis, C., Lujan, M., Kirkham, C., Watson, I.: On the performance of contention managers for complex transactional memory benchmarks. In: *ISPD'09: Proceedings of the 8th International Symposium on Parallel and Distributed Computing* (2009)
2. Ansari, M., Luján, M., Kotselidis, C., Jarvis, K., Kirkham, C., Watson, I.: Steal-on-abort: Improving transactional memory performance through dynamic transaction reordering. In: *HiPEAC '09: Proceedings of the 4th International Conference on High Performance Embedded Architectures and Compilers*. pp. 4–18 (2009)
3. Attiya, H., Epstein, L., Shachnai, H., Tamir, T.: Transactional contention management as a non-clairvoyant scheduling problem. *Algorithmica* 57(1), 44–61 (2010)
4. Attiya, H., Milani, A.: Transactional scheduling for read-dominated workloads. In: *OPODIS '09: Proceedings of the 13th International Conference on Principles of Distributed Systems*. pp. 3–17. Springer-Verlag, Berlin, Heidelberg (2009)
5. Dolev, S., Hendler, D., Suissa, A.: CAR-STM: scheduling-based collision avoidance and resolution for software transactional memory. In: *PODC '08: Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*. pp. 125–134. ACM, New York, NY, USA (2008)
6. Dragojević, A., Guerraoui, R., Singh, A.V., Singh, V.: Preventing versus curing: avoiding conflicts in transactional memories. In: *PODC '09: Proceedings of the 28th ACM symposium on Principles of distributed computing*. pp. 7–16. ACM, New York, NY, USA (2009)
7. Guerraoui, R., Herlihy, M., Kapalka, M., Pochon, B.: Robust Contention Management in Software Transactional Memory. In: *SCOOOL '05: Proceedings of the*

- OOPSLA 2005 Workshop on Synchronization and Concurrency in Object-Oriented Languages (2005)
8. Guerraoui, R., Herlihy, M., Pochon, B.: Polymorphic contention management. In: DISC '05: Proceedings of the nineteenth International Symposium on Distributed Computing. pp. 303–323. LNCS, Springer (2005)
 9. Guerraoui, R., Herlihy, M., Pochon, B.: Toward a theory of transactional contention managers. In: PODC '01: Proceedings of the Twenty-Fourth Annual Symposium on Principles of Distributed Computing (2005)
 10. Harris, T., Fraser, K.: Language support for lightweight transactions. In: OOPSLA '03: Proceedings of the International Conference on Object-Oriented Programming, Systems, Languages, and Applications. pp. 388–402 (2003)
 11. Harris, T., Marlow, S., Peyton-Jones, S., Herlihy, M.: Composible memory transactions. In: PPOPP '05: Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming. pp. 48–60 (2005)
 12. Herlihy, M., Luchangco, V., Moir, M.: Obstruction-free synchronization: Double-ended queues as an example. In: ICDCS '03: Proceedings of the 23rd International Conference on Distributed Computing Systems. pp. 522–529 (2003)
 13. Herlihy, M., Luchangco, V., Moir, M., Scherer, III, W.N.: Software transactional memory for dynamic-sized data structures. In: PODC '03: Proceedings of the twenty-second annual symposium on Principles of distributed computing. pp. 92–101. ACM, New York, NY, USA (2003)
 14. Herlihy, M., Moss, J.E.B.: Transactional memory: Architectural support for lock-free data structures. In: ISCA '93: Proceedings of the 20th Annual International Symposium on Computer Architecture. pp. 289–300 (1993)
 15. Khot, S.: Improved inapproximability results for maxclique, chromatic number and approximate graph coloring. In: FOCS '01: Proceedings of the 42nd IEEE symposium on Foundations of Computer Science. pp. 600–609 (2001)
 16. Leighton, F.T., Maggs, B.M., Rao, S.B.: Packet routing and job-shop scheduling in $O(\textit{congestion} + \textit{dilation})$ steps. *Combinatorica* 14, 167–186 (1994)
 17. Ramadan, H.E., Rossbach, C.J., Porter, D.E., Hofmann, O.S., Bhandari, A., Witchel, E.: Metatm/txlinux: Transactional memory for an operating system. *IEEE Micro* 28(1), 42–51 (2008)
 18. Scherer, III, W.N., Scott, M.L.: Advanced contention management for dynamic software transactional memory. In: PODC '05: Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing. pp. 240–248 (2005)
 19. Scherer III, W.N., Scott, M.L.: Contention management in dynamic software transactional memory. In: CSJP '04: Proceedings of the ACM PODC Workshop on Concurrency and Synchronization in Java Programs. St. John's, NL, Canada (2004)
 20. Scherer III, W.N., Scott, M.L.: Randomization in STM contention management (POSTER). In: PODC '05: Proceedings of the 24th ACM Symposium on Principles of Distributed Computing. Las Vegas, NV (2005)
 21. Schneider, J., Wattenhofer, R.: Bounds on contention management algorithms. In: ISAAC '09: Proceedings of the 20th International Symposium on Algorithms and Computation (2009)
 22. Shavit, N., Touitou, D.: Software transactional memory. In: PODC '95: Proceedings of the fourteenth annual ACM symposium on Principles of distributed computing. pp. 204–213. ACM, New York, NY, USA (1995)
 23. Yoo, R.M., Lee, H.H.S.: Adaptive transaction scheduling for transactional memory systems. In: SPAA '08: Proceedings of the twentieth annual symposium on Parallelism in algorithms and architectures. pp. 169–178 (2008)