

A Competitive Analysis for Balanced Transactional Memory Workloads

Gokarna Sharma and Costas Busch

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

Abstract. We consider transactional memory contention management in the context of *balanced workloads*, where if a transaction is writing, the number of write operations it performs is a constant fraction of its total reads and writes. We explore the theoretical performance boundaries of contention management in balanced workloads from the worst-case perspective by presenting and analyzing two new polynomial time contention management algorithms. The first algorithm *Clairvoyant* is $O(\sqrt{s})$ -competitive, where s is the number of shared resources. This algorithm depends on explicitly knowing the conflict graph. The second algorithm *Non-Clairvoyant* is $O(\sqrt{s} \cdot \log n)$ -competitive, with high probability, which is only a $O(\log n)$ factor worse, but does not require knowledge of the conflict graph, where n is the number of transactions. Both of these algorithms are greedy. We also prove that the performance of *Clairvoyant* is tight, since there is no polynomial time contention management algorithm that is better than $O((\sqrt{s})^{1-\epsilon})$ -competitive for any constant $\epsilon > 0$, unless $\text{NP} \subseteq \text{ZPP}$. To our knowledge, these results are significant improvements over the best previously known $O(s)$ competitive ratio bound.

1 Introduction

The ability of multi-core architectures to increase application performance depends on maximizing the utilization of the computing resources provided by them and using multiple threads within applications. These 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 a non-trivial task because of the complexity of ensuring proper synchronization. Conventional lock based synchronization (i.e., mutual exclusion) suffers from well known limitations, so researchers considered non-blocking transactions as an alternative. Herlihy and Moss [16] proposed Transactional Memory (TM), as an alternative implementation of mutual exclusion, which 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. Shavit and Touitou [24] extended this idea

to Software-only Transactional Memory (STM) by proposing a novel software method for supporting flexible transactional programming of synchronization operations [15,12,13].

A transaction consists of a sequence of read and write operations to a set of shared system resources (e.g. shared memory locations). Transactions may conflict when they access the same shared resources. If a transaction T discovers that it conflicts with another transaction T' (because they share a common resource), it has two choices, it can give T' a chance to commit by aborting itself, or it can proceed and commit by forcing T' to abort; the aborted transaction then retries again until it eventually commits. To solve the transaction scheduling problem efficiently, each transaction consults with the *contention manager* module for which choice to make. Dynamic STM (DSTM) [15], proposed for dynamic-sized data structures, is the first STM implementation that uses a contention manager as an independent module to resolve conflicts between two transactions and ensure progress. Of particular interest are *greedy contention managers* where a transaction restarts immediately after every abort. As TM has been gaining attention, several (greedy) contention managers have been proposed in the literature [2,11,10,5,21,19]. which have been assessed formally and experimentally by specific benchmarks [20].

A major challenge in guaranteeing progress through transactional contention managers is to devise a policy which ensures that all transactions commit in the shortest possible time. The goal is to minimize the *makespan* which is defined as the duration from the start of the schedule, i.e., the time when the first transaction is issued, until all transactions commit. The makespan of the transactional scheduling algorithm can be compared to the makespan of an optimal off-line scheduling algorithm to provide a *competitive ratio*. The makespan and competitive ratio primarily depend on the *workload* – the set of transactions, along with their arrival times, duration, and resources they read and modify [3].

The performance of some of the contention managers has been analyzed formally in [3,2,11,10,21,23] (the detailed description is given in Section 1.2). The best known formal bound is provided in [2] where the authors give an $O(s)$ competitive ratio bound, where s is the number of shared resources. When the number of resources s increases, the performance degrades linearly. A difficulty in obtaining better competitive ratios is that the scheduling problem of n concurrent transactions is directly related to the vertex coloring problem which is a hard problem to approximate [17]. A natural question which we address here is whether it is possible to obtain better competitive ratios. As we show below, it is indeed possible to obtain sub-linear competitive ratios for balanced transaction workloads.

1.1 Contributions

In this paper, we study contention management in the context of *balanced workloads* which have better performance potential for transactional memory. A balanced workload consists of a set of transactions in which each transaction has the

following property: if the transaction performs write operations, then the number of writes it performs is a constant fraction of the total number of operations (read and writes) of the transaction. The *balancing ratio* β expresses the ratio of write operations of a transaction to the overall operations of the transaction. The balancing ratio is bounded as $\frac{1}{s} \leq \beta \leq 1$, since a writing transaction writes to at least one resource. In balanced workloads $\beta = \Theta(1)$ for all the transactions which perform writes. Balanced workloads can also include read-only transactions, but we assume that there is at least one transaction that performs writes, since otherwise the scheduling problem is trivial (no conflicts).

Balanced transaction workloads represent interesting and practical transaction memory scheduling problems. For example balanced workloads represent the case where we have small sized transactions each accessing a small (constant) number of resources, where trivially $\beta = \Theta(1)$. Other interesting scenarios are transaction workloads which are write intensive, where transactions perform many writes, as for example in scientific computing applications where transactions have to update large arrays.

We present two new polynomial time contention management algorithms which are especially tailored for balanced workloads and analyze their theoretical performance boundaries from the worst-case perspective. The first algorithm, called *Clairvoyant*, is $O\left(\ell \cdot \sqrt{\frac{s}{\beta}}\right)$ -competitive where s is the number of shared resources, and ℓ expresses the logarithm ratio of the longest to shortest execution times of the transactions. (The transaction execution time is the time it needs to commit uninterrupted from the moment it starts.) For balanced transaction workloads where $\beta = \Theta(1)$, and when transaction execution times are close to each other, i.e. $\ell = O(1)$, Algorithm *Clairvoyant* is $O(\sqrt{s})$ -competitive. This algorithm is greedy and has the pending commit property (where at least one transaction executes uninterrupted each time). However, it depends on assigning priorities to the transactions based on the explicit knowledge of the transaction conflict graph which evolves while the execution of the transactions progresses. It also assumes that each transaction knows how long is its execution time and how many resources it accesses.

The second algorithm, called *Non-Clairvoyant*, is $O\left(\ell \cdot \sqrt{\frac{s}{\beta}} \cdot \log n\right)$ -competitive, with high probability (at least $1 - \frac{1}{n}$), where n is the number of transactions concurrently executing in n threads. For balanced transaction workloads, where $\beta = \Theta(1)$, and when transaction execution times are close to each other, i.e. $\ell = O(1)$, Algorithm *Non-Clairvoyant* is $O(\sqrt{s} \cdot \log n)$ -competitive. This is only a $O(\log n)$ factor worse than *Clairvoyant*, but does not require explicit knowledge of the conflict graph. The algorithm is also greedy. This algorithm uses as a subroutine a variation of the *RandomizedRounds* scheduling algorithm by Schneider and Wattenhofer [21] which uses randomized priorities and doesn't require knowledge of the conflict graph.

The $O(\sqrt{s})$ bound of Algorithm *Clairvoyant* is actually tight. Through a reduction from the graph coloring problem, we show that it is impossible to approximate in polynomial time any transactional scheduling problem with $\beta = 1$

and $\ell = 1$ with a competitive ratio smaller than $O((\sqrt{s})^{1-\epsilon})$ for any constant $\epsilon > 0$, unless $\text{NP} \subseteq \text{ZPP}$. To our knowledge, these results are significant improvements over the best previously known bound of $O(s)$ for transactional memory contention managers. For general workloads (including non-balanced workloads), where transactions are equi-length ($\ell = O(1)$), our analysis gives $O(s)$ competitive worst case bound, since $\beta \geq 1/s$. This bound matches the best previously known bound of $O(s)$ for general workloads. The parametrization of β that we provide gives more tradeoffs and flexibility for better scheduling performance, as depicted by the performance of our algorithms in balanced workloads.

1.2 Related Work

Almost 10 year after publishing the seminal paper [16] to introduce the new research area of transactional memory, Herlihy *et al.* [15] proposed Dynamic STM (DSTM) for dynamic-sized data structures. Later on, several other STM implementations have been proposed, such as TL2 [4], TinySTM [8], and RSTM [18] to name a few. Among them, DSTM is the first practical obstruction-free¹ implementation that seeks advice from the contention manager module to either wait or abort a transaction at the time of conflict.

Several contention managers have been proposed in STM and the performance of some of them has been analyzed formally in [3,2,11,10,21,23]. The first formal analysis of the performance of a contention manager is given by Guerraoui *et al.* [11] where they present the **Greedy** contention manager which decides in favor of older transactions using timestamps and achieves $O(s^2)$ competitive ratio. This bound holds for any algorithm which ensures the *pending commit* property (see Definition 1). Attiya *et al.* [2] improve the competitive ratio to $O(s)$, and prove a matching lower bound of $\Omega(s)$ for any deterministic *work-conserving* algorithm which schedules as many transactions as possible (by choosing a maximal independent set of transactions). The model in [2] is non-clairvoyant in the sense that it requires no prior knowledge about the transactions while they are executed.

Schneider and Wattenhofer [21] present a deterministic algorithm **Commit-Bounds** with competitive ratio $\Theta(s)$ and a randomized algorithm **RandomizedRounds** with makespan $O(C \log n)$ with high probability, for a set of n transactions, where C denotes the maximum number of conflicts among transactions (assuming unit execution time durations for transactions). Sharma *et al.* [23] study greedy contention managers for $M \times N$ *execution windows of transactions* with M threads and N transactions per thread and present and analyze two new randomized greedy contention management algorithms. Their first algorithm **Offline-Greedy** produces a schedule of length $O(\tau_{\max} \cdot (C + N \log(MN)))$ with high probability, where τ_{\max} is the execution time duration of the longest transaction in the system, and the second algorithm **Online-Greedy** produces a schedule of length $O(\tau_{\max} \cdot (C \log(MN) + N \log^2(MN)))$. The competitiveness of

¹ A synchronization mechanism is obstruction-free if any thread that runs for a long time it eventually makes progress [14].

both of the algorithms is within a poly-log factor of $O(s)$. Another recent work 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 $O(n)$ -competitive and in fact, it ensures that two transactions never conflict more than once.

TM schedulers [3,6,25,1] offer an alternative approach to boost the TM performance. A TM scheduler is a software component which decides when a particular transaction executes. One proposal in this approach is **Adaptive Transaction Scheduling (ATS)** [25] which measures adaptively the contention intensity of a thread, and when the contention intensity increases beyond a threshold it serializes the transactions. The **Restart** and **Shrink** schedulers, proposed by Dragojević *et al.* [6], depend on the prediction of future conflicts and dynamically serialize transactions based on the prediction to avoid conflicts. The ATS, Restart, and Shrink schedulers are $O(n)$ -competitive. **Steal-On-Abort** [1] is 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.* [3] 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 bimodal workloads with equi-length transactions. A bimodal workload contains only early-write and read-only transactions.

Outline of Paper. The rest of the paper is organized as follows. We present our TM model and definitions in Section 2. We present and formally analyze two new randomized algorithms, **Clairvoyant** and **Non-Clairvoyant**, in Sections 3 and 4, respectively. The hardness result of balanced workload scheduling is presented in Section 5. Section 6 concludes the paper.

2 Model and Definitions

Consider a system of $n \geq 1$ threads $\mathcal{P} = \{P_1, \dots, P_n\}$ with a finite set of s shared resources $\mathcal{R} = \{R_1, \dots, R_s\}$. We consider batch execution problems, where the system issues a set of n transactions $\mathcal{T} = \{T_1, \dots, T_n\}$ (*transaction workload*), one transaction T_i per thread P_i . Each transaction is a sequence of actions (operations) each of which is either a read or write to some shared resource. The sequence of operations in a transaction must be *atomic*: all operations of a transaction are guaranteed to either completely occur, or have no effects at all. A transaction that only reads shared resources is called *read-only*; otherwise it is called a *writing* transaction. We consider transaction workloads where at least one transaction is writing.

After a transaction is issued and starts execution it either *commits* or *aborts*. A transaction that has been issued but not committed yet is said to be *pending*. A pending transaction can *restart* multiple times until it eventually commits. Concurrent write-write actions or read-write actions to shared objects by two or

more transactions cause conflicts between transactions. If a transaction conflicts then it either aborts, or it may commit and force to abort all other conflicting transactions. In a *greedy schedule*, if a transaction aborts due to conflicts it then immediately restarts and attempts to commit again. We assume that the execution time advances synchronously for all threads and a preemption and abort require negligible time. We also assume that all transactions in the system are correct, i.e., there are no faulty transactions.²

Definition 1 (Pending Commit Property [11]). *A contention manager obeys the pending commit property if, whenever there are pending transactions, some running transaction T will execute uninterrupted until it commits.*

Let $\mathcal{R}(T_i)$ denote the set of resources used by a 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 2 (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).*

From the definition of transaction conflicts we can define the *conflict graph* for a set of transactions. In the conflict graph, each node corresponds to a transaction and each edge represents a conflict between the adjacent transactions.

Definition 3 (Conflict Graph). *For a set of transactions \mathcal{T} , the conflict graph $G(\mathcal{T}) = (V, E)$ has as nodes the transactions, $V = \mathcal{T}$, and $(T_i, T_j) \in E$ for any two transactions T_i, T_j that conflict.*

Let $\gamma(R_j)$ denote the number of transactions that write resource R_j . Let $\gamma_{\max} = \max_j \gamma(R_j)$. Denote $\lambda_w(T_i) = |\mathcal{R}_w(T_i)|$, $\lambda_r(T_i) = |\mathcal{R}_r(T_i)|$, and $\lambda(T_i) = |\mathcal{R}(T_i)|$, the number of resources which are being accessed by transaction T_i for write, read, and both read and write. Let $\lambda_{\max} = \max_i \lambda(T_i)$. Note that in the conflict graph G the maximum node degree is bounded by $\lambda_{\max} \cdot \gamma_{\max}$, and also there is a node whose degree is at least γ_{\max} .

For any transaction T_i we define the *balancing ratio* $\beta(T_i) = \frac{|\mathcal{R}_w(T_i)|}{|\mathcal{R}(T_i)|}$ as the ratio of number of writes versus the total number of resources it accesses. For a read-only transaction $\beta(T_i) = 0$. For a writing transaction it holds $\frac{1}{s} \leq \beta(T_i) \leq 1$, since there will be at least one write performed by T_i to one of the s resources. We define the *global balancing ratio* as the minimum of the individual writing transaction balancing ratios: $\beta = \min_{(T_i \in \mathcal{T}) \wedge (\lambda_w(T_i) > 0)} \beta(T_i)$. We define *balanced transaction workloads* as follows (recall that we consider workloads with at least one writing transaction):

² 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 [10].

Definition 4 (Balanced Workloads). We say that a workload (set of transactions) \mathcal{T} is balanced if $\beta = \Theta(1)$.

In other words, in balanced transaction workloads the number of writes that each writing transaction performs is a constant fraction of the total number of resource accesses (for read or write) that the transaction performs.

Each transaction T_i has execution time duration $\tau_i > 0$. The execution time is the total number of discrete time steps that the transaction requires to commit uninterrupted from the moment it starts. In our model we assume that the execution time of each transaction is fixed. Let $\tau_{max} = \max_i \tau_i$ be the execution time of the longest transaction, and $\tau_{min} = \min_i \tau_i$ be the execution time of the shortest transaction. We denote $\ell = \lceil \log \left(\frac{\tau_{max}}{\tau_{min}} \right) \rceil + 1$. We finish this section with the basic definitions of *makespan* and *competitive ratio*.

Definition 5 (Makespan and Competitive Ratio). Given a contention manager \mathcal{A} and a workload \mathcal{T} , $\text{makespan}_{\mathcal{A}}(\mathcal{T})$ is the total time \mathcal{A} needs to commit all the transactions in \mathcal{T} . The competitive ratio is $CR_{\mathcal{A}}(\mathcal{T}) = \frac{\text{makespan}_{\mathcal{A}}(\mathcal{T})}{\text{makespan}_{opt}(\mathcal{T})}$, where opt is the optimal off-line scheduler.

3 Clairvoyant Algorithm

We describe and analyze Algorithm Clairvoyant (see Algorithm 1). The writing transactions are divided into ℓ groups $A_0, A_1, \dots, A_{\ell-1}$, where $\ell = \lceil \log \left(\frac{\tau_{max}}{\tau_{min}} \right) \rceil + 1$, in such a way that A_i contains transactions with execution time duration in range $[2^i \cdot \tau_{min}, (2^{i+1} \cdot \tau_{min} - 1)]$, for $0 \leq i \leq \ell - 1$. Each group of transactions A_i is then again divided into κ subgroups $A_i^0, A_i^1, \dots, A_i^{\kappa-1}$, where $\kappa = \lceil \log s \rceil + 1$, such that each transaction $T \in A_i^j$ accesses (for read and write) a number of resources in range $\lambda(T) \in [2^j, 2^{j+1} - 1]$, for $0 \leq j \leq \kappa - 1$. We assign an order to the subgroups in such a way that $A_i^j < A_k^l$ if $i < k$ or $i = k \wedge j < l$. Note that some of the subgroups may be empty. The read-only transactions are placed into a special group B which has the highest order.

At any time t the pending transactions are assigned a priority level which determines which transactions commit or abort. A transaction is assigned a priority which is one of: *high* or *low*. Let Π_t^h and Π_t^l denote the set of transactions which will be assigned high and low priority, respectively, at time t . In conflicts, high priority transactions abort low priority transactions. Conflicts between transactions of the same priority level are resolved arbitrarily. Suppose that \hat{A}_t is the lowest order subgroup that contains pending transactions at time t . Only transactions from \hat{A}_t can be given high priority, that is $\Pi_t^h \subseteq \hat{A}_t$.

The priorities are determined according to the conflict graph for the transactions. Let \mathcal{T}_t denote the set of all transactions which are pending at time t . (Initially, $\mathcal{T}_0 = \mathcal{T}$.) Let $\hat{\mathcal{T}}_t$ denote the pending transactions of \hat{A}_t at time t . (Initially, $\hat{\mathcal{T}}_0 = \hat{A}_0$.) Let \hat{S}_t denote the set of transactions in $\hat{\mathcal{T}}_t$ which are pending and have started executing before t but have not yet committed or aborted. Let \hat{S}_t^c denote the set of transactions in \mathcal{T}_t which conflict with \hat{S}_t . Let \hat{I}_t be a maximal

Algorithm 1: Clairvoyant

Input: A set \mathcal{T} of n transactions with global balancing ratio β ;

Output: A greedy execution schedule;

- Divide writing transactions into $\ell = \lceil \log(\frac{\tau_{\max}}{\tau_{\min}}) \rceil + 1$ groups $A_0, A_1, \dots, A_{\ell-1}$ in such a way that A_i contains transactions with execution time duration in range $[2^i \cdot \tau_{\min}, (2^{i+1} \cdot \tau_{\min} - 1)]$; Read-only transactions are placed in special group B ;
- Divide A_i again into $\kappa = \lceil \log s \rceil + 1$ subgroups $A_i^0, A_i^1, \dots, A_i^{\kappa-1}$ in a way that each subgroup A_i^j contains transactions that access a number of resource in the range $[2^j, 2^{j+1} - 1]$;
- Order the groups and subgroups such that $A_i^j < A_k^l$ if $i < k$ or $i = k \wedge j < l$; special group B has highest order;

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

Set Definitions:

\mathcal{T}_t : set of transactions that are pending; // $\mathcal{T}_0 \leftarrow \mathcal{T}$

\widehat{A}_t : lowest order group that contains pending transactions;

$\widehat{\mathcal{T}}_t$: set of transactions in \widehat{A}_t which are pending; // $\widehat{\mathcal{T}}_0 \leftarrow \widehat{A}_0$

\widehat{S}_t : set of transactions in $\widehat{\mathcal{T}}_t$ which were started before t ;

\widehat{S}'_t : set of conflicting transactions in \mathcal{T}_t which conflict with \widehat{S}_t ;

\widehat{I}_t : maximal independent set in the conflict graph $G(\widehat{\mathcal{T}}_t \setminus \widehat{S}'_t)$;

Priority Assignment:

 High priority transactions: $\Pi_t^h \leftarrow \widehat{I}_t \cup \widehat{S}_t$;

 Low priority transactions: $\Pi_t^l \leftarrow \mathcal{T}_t \setminus \Pi_t^h$;

Conflict Resolution:

 Execute all pending transactions;

On conflict of transaction T_u with transaction T_v :

if $(T_u \in \Pi_t^h) \wedge (T_v \in \Pi_t^l)$ **then** $abort(T_u, T_v)$; **else** $abort(T_v, T_u)$;

 // $abort(T_u, T_v)$ **aborts** transaction T_v

independent set in the conflict graph $G(\widehat{\mathcal{T}}_t \setminus \widehat{S}'_t)$. Then, the set of high priority transactions at time t is set to be $\Pi_t^h = \widehat{I}_t \cup \widehat{S}_t$. The remaining transactions are given low priority, that is, $\Pi_t^l = \mathcal{T}_t \setminus \Pi_t^h$. Note that the transactions in Π_t^h do not conflict with each other. The transactions Π_t^h will remain in high priority in subsequent time steps $t' > t$ until they commit, since the transactions in $\widehat{S}'_{t'}$ are included in $\Pi_{t'}^h$.

This algorithm is clairvoyant in the sense that it requires explicit knowledge of the various conflict relations at each time t . The algorithm is greedy, since at each time step each pending transaction is not idle. The algorithm also satisfies the pending commit property since at any time step t at least one transaction from \widehat{A}_t will execute uninterrupted until it commits. We have assumed above that each transaction knows its execution length and the number of resources it accesses. Clearly, the algorithm computes the schedule in polynomial time.

3.1 Analysis of Clairvoyant Algorithm

We now give a competitive analysis of Algorithm Clairvoyant. Define $\tau_{\min}^j = 2^i \cdot \tau_{\min}$ and $\tau_{\max}^j = (2^{i+1} \cdot \tau_{\min} - 1)$. Note that the duration of each transaction $T \in A_i^j$ is in range $[\tau_{\min}^j, \tau_{\max}^j]$, and also $\tau_{\max}^j \leq 2\tau_{\min}^j$. Define $\lambda_{\min}^j = 2^j$ and $\lambda_{\max}^j = 2^{j+1} - 1$. Note that for each transaction $T \in A_i^j$, $\lambda(T) \in [\lambda_{\min}^j, \lambda_{\max}^j]$, and $\lambda_{\max}^j \leq 2\lambda_{\min}^j$.

In the next results we will first focus on a subgroup A_i^j and we will assume that there are no other transactions in the system. We give bounds for the competitive ratio for A_i^j which will be useful when we later analyze the performance for all the transactions in \mathcal{T} .

Lemma 1. *If we only consider transactions in subgroup A_i^j , then the competitive ratio is bounded by $CR_{\text{Clairvoyant}}(A_i^j) \leq 2 \cdot \lambda_{\max}^j + 2$.*

Proof. Let $\gamma_i^j(R_v)$ denote the number of transactions in a subgroup A_i^j that write R_v , $1 \leq v \leq s$. Let $\gamma' = \max_{v \in [1, s]} \gamma_i^j(R_v)$. Since there is only one subgroup, $\widehat{A}_i = A_i^j$. A transaction $T \in A_i^j$ conflicts with at most $\lambda_{\max}^j \cdot \gamma'$ other transactions in the same subgroup. If transaction T is in low priority it is only because some other conflicting transaction in A_i^j is in high priority. If no conflicting transaction is in high priority then T becomes high priority immediately. Since a high priority transaction executes uninterrupted until it commits, it will take at most $\lambda_{\max}^j \cdot \gamma'$ time steps until all conflicting transactions with T have committed. Thus, it is guaranteed that in at most $\lambda_{\max}^j \cdot \gamma' \cdot \tau_{\max}^j$ time steps T becomes high priority. Therefore, T commits by time $(\lambda_{\max}^j \cdot \gamma' + 1) \cdot \tau_{\max}^j$. Since T is an arbitrary transaction in A_i^j , the makespan of the algorithm is bounded by:

$$\text{makespan}_{\text{Clairvoyant}}(A_i^j) \leq (\lambda_{\max}^j \cdot \gamma' + 1) \cdot \tau_{\max}^j.$$

There is a resource that is accessed by at least γ' transactions of A_i^j for write. All these transactions have to serialize because they all conflict with each other in the common resource. Therefore, the optimal makespan is bounded by:

$$\text{makespan}_{\text{opt}}(A_i^j) \geq \gamma' \cdot \tau_{\min}^j.$$

When we combine the upper and lower bounds we obtain a bound on the competitive ratio of the algorithm:

$$CR_{\text{Clairv.}}(A_i^j) = \frac{\text{makespan}_{\text{Clairv.}}(A_i^j)}{\text{makespan}_{\text{opt}}(A_i^j)} \leq \frac{(\lambda_{\max}^j \cdot \gamma' + 1) \cdot \tau_{\max}^j}{\gamma' \cdot \tau_{\min}^j} \leq 2 \cdot \lambda_{\max}^j + 2.$$

Lemma 2. *If we only consider transactions in subgroup A_i^j , then the competitive ratio is bounded by $CR_{\text{Clairvoyant}}(A_i^j) \leq 4 \cdot \frac{s/\beta}{\lambda_{\max}^j}$.*

Proof. Since the algorithm satisfies the pending-commit property, if a transaction $T \in A_i^j$ does not commit, then some conflicting transaction $T' \in A_i^j$ must commit. Therefore, the makespan of the algorithm is bounded by:

$$\text{makespan}_{\text{Clairvoyant}}(A_i^j) \leq |A_i^j| \cdot \tau_{\max}^j.$$

Each transaction in $T \in A_i^j$ accesses at least $\lambda_w(T)$ resources for write. Since we only consider transactions in A_i^j , $\lambda_w(T) \geq \beta \cdot \lambda_{\min}^j \geq \beta \cdot \lambda_{\max}^j / 2$. Consequently, by the pigeonhole principle, there will be a resource $R \in \mathcal{R}$ which is accessed by at least $\sum_{T \in A_i^j} \lambda_w(T) / s \geq |A_i^j| \cdot \beta \cdot \lambda_{\max}^j / (2s)$ transactions for write. All these transactions accessing R have to serialize because they conflict with each other. Therefore, the optimal makespan is bounded by:

$$\text{makespan}_{\text{opt}}(A_i^j) \geq \frac{|A_i^j| \cdot \beta \cdot \lambda_{\max}^j}{2s} \cdot \tau_{\min}^j.$$

When we combine the above bounds of the makespan we obtain the following bound on the competitive ratio of the algorithm:

$$CR_{\text{Clairvoyant}}(A_i^j) = \frac{\text{makespan}_{\text{Clairvoyant}}(A_i^j)}{\text{makespan}_{\text{opt}}(A_i^j)} \leq \frac{|A_i^j| \cdot \tau_{\max}^j}{\frac{|A_i^j| \cdot \beta \cdot \lambda_{\max}^j}{2s} \cdot \tau_{\min}^j} \leq 4 \cdot \frac{s/\beta}{\lambda_{\max}^j}.$$

From Lemmas 1 and 2, we obtain:

Corollary 1. *If we only consider transactions in subgroup A_i^j , then the competitive ratio of the algorithm is bounded by $CR_{\text{Clairvoyant}}(A_i^j) \leq 4 \cdot \min \left\{ \lambda_{\max}^j, \frac{s/\beta}{\lambda_{\max}^j} \right\}$.*

We now continue to provide a bound for the performance of individual groups. This will help to provide bounds for all the transactions.

Lemma 3. *If we only consider transactions in group A_i , then the competitive ratio of the algorithm is bounded by $CR_{\text{Clairvoyant}}(A_i) \leq 32 \cdot \sqrt{\frac{s}{\beta}}$.*

Proof. Since $\lambda_{\max}^j = (2^{j+1} - 1)$, Corollary 1 gives for each subgroup A_i^j competitive ratio

$$CR_{\text{Clairvoyant}}(A_i^j) \leq 4 \cdot \min \left\{ 2^{j+1} - 1, \frac{s/\beta}{2^{j+1} - 1} \right\} \leq 8 \cdot \min \left\{ 2^j, \frac{s/\beta}{2^j} \right\}.$$

Let $\psi = \frac{\log(s/\beta)}{2}$. Note that $\min \left\{ 2^j, \frac{s/\beta}{2^j} \right\} \leq 2^j$, $\forall j \in [0, \lfloor \psi \rfloor]$; and $\min \left\{ 2^j, \frac{s/\beta}{2^j} \right\} \leq \frac{s/\beta}{2^j} = 2^{2\psi-j}$, $\forall j \in [\lfloor \psi \rfloor + 1, \kappa - 1]$. Group A_i contains κ subgroups of transactions. In the worst case, Algorithm Clairvoyant will commit the transactions in each subgroup according to their order starting from the lowest order subgroup and ending at the highest order subgroup, since that's the order that the transactions are assigned a high priority. Therefore,

$$\begin{aligned} CR_{\text{Clairv.}}(A_i) &\leq \sum_{j=0}^{\kappa-1} CR_{\text{Clairv.}}(A_i^j) \\ &= \sum_{j=0}^{\lfloor \psi \rfloor} CR_{\text{Clairv.}}(A_i^j) + \sum_{j=\lfloor \psi \rfloor+1}^{\kappa-1} CR_{\text{Clairv.}}(A_i^j) \end{aligned}$$

$$\leq 8 \cdot \left(\sum_{j=0}^{\lfloor \psi \rfloor} 2^j + \sum_{j=\lfloor \psi \rfloor+1}^{k-1} 2^{2\psi-j} \right) \leq 8 \cdot (2 \cdot 2^\psi + 2 \cdot 2^\psi) = 32 \cdot \sqrt{\frac{s}{\beta}}.$$

Theorem 1 (Competitive Ratio of Clairvoyant). *For set of transactions \mathcal{T} , Algorithm Clairvoyant has competitive ratio $CR_{\text{Clairvoyant}}(\mathcal{T}) = O\left(\ell \cdot \sqrt{\frac{s}{\beta}}\right)$.*

Proof. As there are ℓ groups of transactions A_i , and one group B , in the worst case, Algorithm Clairvoyant will commit the transactions in each group according to their order starting from the lowest order group and ending at the highest order group. Clearly, the algorithm will execute the read-only transactions in group B in optimal time. Therefore, using Lemma 3, we obtain:

$$\begin{aligned} CR_{\text{Clairvoyant}}(\mathcal{T}) &\leq \sum_{i=0}^{\ell-1} CR_{\text{Clairvoyant}}(A_i) + CR_{\text{Clairvoyant}}(B) \\ &\leq \sum_{i=0}^{\ell-1} 32 \cdot \sqrt{\frac{s}{\beta}} + 1 = 32 \cdot \ell \cdot \sqrt{\frac{s}{\beta}} + 1. \end{aligned}$$

The corollary below follows immediately from Theorem 1.

Corollary 2 (Balanced Workload). *For balanced workload \mathcal{T} ($\beta = \Theta(1)$) and when $\ell = O(1)$, Algorithm Clairvoyant has competitive ratio $CR_{\text{Clairvoyant}}(\mathcal{T}) = O(\sqrt{s})$.*

4 Non-Clairvoyant Algorithm

We present and analyze Algorithm Non-Clairvoyant (see Algorithm 2). This algorithm is similar to Clairvoyant given at Section 3 with the difference that the conflicts are resolved using priorities which are determined without the explicit knowledge of the conflict graph.

Similar to Algorithm Clairvoyant, the transactions are organized into groups and subgroups. Lower order subgroups have always higher priority than higher order subgroups. At each time step t , let \hat{A}_t denote the lowest order subgroup. Clearly, the transactions in \hat{A}_t have higher priority than the transactions in all other subgroups, and in case of conflicts only the transactions in \hat{A}_t win. When transactions in the same subgroup conflict, the conflicts are resolved according to random priority numbers. When a transaction starts execution it chooses uniformly at random a discrete number $r(T) \in [1, n]$. In case of a conflict of transaction T_w with another transaction T_x in the same subgroup with $r(T_x) < r(T_w)$, then T_x aborts T_w , and otherwise T_w aborts T_x . When transaction T_w restarts, it cannot abort T_x until T_x has been committed or aborted. After every abort, the newly started transaction chooses again a new discrete number uniformly at random in the interval $[1, n]$. The idea of randomized priorities has been introduced originally by Schneider and Wattenhofer [21] in their Algorithm RandomizedRounds.

Algorithm 2: Non-Clairvoyant

Input: A set \mathcal{T} of n transactions with global balancing ratio β ;
Output: A greedy execution schedule;

- Divide transactions into $\ell = \lceil \log(\frac{\tau_{\max}}{\tau_{\min}}) \rceil + 1$ groups $A_0, A_1, \dots, A_{\ell-1}$ in such a way that A_i contains transactions with execution time duration in range $[2^i \cdot \tau_{\min}, (2^{i+1} \cdot \tau_{\min} - 1)]$; Read-only transactions are placed in special group B ;
- Divide A_i again into $\kappa = \lceil \log s \rceil + 1$ subgroups $A_i^0, A_i^1, \dots, A_i^{\kappa-1}$ in a way that each subgroup A_i^j contains transactions that access a number of resource in the range $[2^j, 2^{j+1} - 1]$;
- Order the groups and subgroups such that $A_i^j < A_k^l$ if $i < k$ or $i = k \wedge j < l$; special group B has highest order;

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

- Execute all pending transactions; // at $t = 0$ issue all transactions
- On (re)start** of transaction T :
 $r(T) \leftarrow$ random integer in $[1, n]$;
- On conflict** of transaction $T_u \in A_i^j$ with transaction $T_v \in A_k^l$:
if $A_i^j < A_k^l$ **then** $\text{abort}(T_u, T_v)$;
else if $A_i^j > A_k^l$ **then** $\text{abort}(T_v, T_u)$;
else if $r(T_u) < r(T_v)$ **then** $\text{abort}(T_u, T_v)$; // The case $A_i^j = A_k^l$
else $\text{abort}(T_v, T_u)$;
// In case a transaction T_u aborts T_v because $r(T_u) < r(T_v)$,
then when T_v restarts it cannot abort T_u until T_u
commits or aborts

This algorithm is non-clairvoyant in the sense that it does not depend on knowing explicitly the conflict graph to resolve conflicts. The algorithm is greedy but does have the pending commit property. The groups and subgroups can be implemented in the algorithm since we assume that each transaction knows its execution time and the number of resources that it accesses. Clearly, the algorithm computes the schedule in polynomial time.

4.1 Analysis of Non-Clairvoyant Algorithm

In the analysis given below, we study the properties of Algorithm Non-Clairvoyant and give its competitive ratios. We use the following adaptation of the response time analysis of Algorithm RandomizedRounds given in [21]. It uses the following Chernoff bound:

Lemma 4 (Chernoff Bound). *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 5 (Adaptation from Schneider and Wattenhofer [21]). *Given a transaction scheduling problem with n concurrent transactions, where each*

transaction has execution time at most τ , the time span a transaction T needs from the moment it is issued until commit is $16 \cdot e \cdot (d_T + 1) \cdot \tau \cdot \ln n$ 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 problem with the n transaction. Let N_T denote the set of conflicting transactions for T (these are the neighbors of T in G). Let $r(T)$ denote the random priority number choice of T in range $[1, n]$. The probability that for transaction T no transaction $T' \in N_T$ has the same random number is:

$$\Pr(\nexists T' \in N_T | r(T) = r(T')) = \left(1 - \frac{1}{n}\right)^{d_T} \geq \left(1 - \frac{1}{n}\right)^n \geq \frac{1}{e}.$$

The probability that $r(T)$ is at least as small as $r(T')$ for any transaction $T' \in N_T$ is $\frac{1}{d_T + 1}$. Thus, the chance that $r(T)$ is smallest and different among all its neighbors in N_T is at least $\frac{1}{e \cdot (d_T + 1)}$. If we conduct $16 \cdot e \cdot (d_T + 1) \cdot \ln n$ trials, each having success probability $\frac{1}{e \cdot (d_T + 1)}$, then the probability that the number of successes Z is less than $8 \cdot \ln n$ becomes: $\Pr(Z < 8 \cdot \ln n) < e^{-2 \cdot \ln n} = 1/n^2$, using the Chernoff bound of Lemma 4. Since every transaction has execution time at most τ , the total time spent until a transaction commits is at most $16 \cdot e \cdot (d_T + 1) \cdot \tau \cdot \ln n$, with probability at least $1 - 1/n^2$.

We now give competitive bounds for some subgroup A_i^j and later extend the results to all the transactions in \mathcal{T} . The proofs are similar as in the analysis of Algorithm Clairvoyant and can be found in the full version of paper (See [22]).

Lemma 6. *If we only consider transactions in subgroup A_i^j , then the competitive ratio is bounded by $CR_{Non-Clairvoyant}(A_i^j) \leq 64 \cdot e \cdot \lambda_{\max}^j \cdot \ln n$ with probability at least $1 - \frac{|A_i^j|}{n^2}$.*

Lemma 7. *If we only consider transactions in subgroup A_i^j , then the competitive ratio is bounded by $CR_{Non-Clairvoyant}(A_i^j) \leq 64 \cdot e \cdot \frac{s/\beta}{\lambda_{\max}^j} \cdot \ln n$ with probability at least $1 - \frac{|A_i^j|}{n^2}$.*

From Lemmas 6 and 7, we obtain:

Corollary 3. *If we only consider transactions in subgroup A_i^j , then the competitive ratio of the algorithm is bounded by $CR_{Non-Clairvoyant}(A_i^j) \leq 64 \cdot e \cdot \min \left\{ \lambda_{\max}^j, \frac{s/\beta}{\lambda_{\max}^j} \right\} \cdot \ln n$ with probability at least $1 - \frac{|A_i^j|}{n^2}$.*

We now provide a bound for the performance of individual groups which will help to provide bounds for all the transactions.

Lemma 8. *If we only consider transactions in group A_i , then the competitive ratio of the algorithm is bounded by $CR_{Non-Clairvoyant}(A_i) \leq 512 \cdot e \cdot \sqrt{\frac{s}{\beta}} \cdot \ln n$ with probability at least $1 - \frac{|A_i|}{n^2}$.*

Theorem 2 (Competitive Ratio of Non-Clairvoyant). *For a set of transactions \mathcal{T} , Algorithm Non-Clairvoyant has competitive ratio $CR_{Non-Clairvoyant}(\mathcal{T}) = O\left(\ell \cdot \sqrt{\frac{s}{\beta}} \cdot \log n\right)$ with probability at least $1 - \frac{1}{n}$.*

The corollary below follows immediately from Theorem 2.

Corollary 4 (Balanced Workload). *For balanced workload \mathcal{T} ($\beta = \Theta(1)$) and when $\ell = O(1)$, Algorithm Non-Clairvoyant has competitive ratio $CR_{Non-Clairvoyant}(\mathcal{T}) = O(\sqrt{s} \cdot \log n)$ with probability at least $1 - \frac{1}{n}$.*

5 Hardness of Balanced Transaction Scheduling

In this section, we show that the performance of Clairvoyant is tight by reducing the graph coloring problem to the transaction scheduling problem.

A VERTEX COLORING problem instance asks whether a given graph G is k -colorable [9]. A valid k -coloring is an assignment of integers $\{1, 2, \dots, k\}$ (the colors) to the vertices of G so that neighbors receive different integers. The chromatic number, $\chi(G)$ is the smallest k such that G has a valid k -coloring. We say that an algorithm approximates $\chi(G)$ with approximation ratio $q(G)$ if it outputs $u(G)$ such that $\chi(G) \leq u(G)$ and $u(G)/\chi(G) \leq q(G)$. Typically, $q(G)$ is expressed only as a function of n , the number of vertices in G . It is well known that known VERTEX COLORING is NP-complete. It is also shown in [7] that unless $\text{NP} \subseteq \text{ZPP}$, there does not exist a polynomial time algorithm to approximate $\chi(G)$ with approximation ratio $O(n^{1-\epsilon})$ for any constant $\epsilon > 0$, where n denotes the number of vertices in graph G .

A TRANSACTION SCHEDULING problem instance asks whether a set of transactions \mathcal{T} with a set of resources \mathcal{R} has makespan k time steps. We give a polynomial time reduction of the VERTEX COLORING problem to the TRANSACTION SCHEDULING problem. Consider an input graph $G = (V, E)$ of the VERTEX COLORING problem, where $|V| = n$ and $|E| = s$. We construct a set of transactions \mathcal{T} such that for each $v \in V$ there is a respective transaction $T_v \in \mathcal{T}$; clearly, $|\mathcal{T}| = |V| = n$. We also use a set of resources \mathcal{R} such that for each edge $e \in E$ there is a respective resource $R_e \in \mathcal{R}$; clearly, $|\mathcal{R}| = |E| = s$. If $e = (u, v) \in E$, then both the respective transactions T_u and T_v use the resource R_e for write. Since all transaction operations are writes, we have that $\beta = 1$. We take all the transactions to have the same execution length equal to one time step, that is, $\tau_{\max} = \tau_{\min} = 1$, and $\ell = 1$.

Let G' be the conflict graph for the transactions \mathcal{T} . Note that G' is isomorphic to G . Node colors in G correspond to time steps in which transactions in G' are issued. Suppose that G has a valid k -coloring. If a node $v \in G$ has a color x , then the respective transaction $T_v \in G'$ can be issued and commit at time step x , since no conflicting transaction (neighbor in G') has the same time assignment (color) as T_v . Thus, a valid k -coloring in G implies a schedule with makespan k for the transactions in \mathcal{T} . Symmetrically, a schedule with makespan k for \mathcal{T} implies a valid k -coloring in G .

It is easy to see that the problem TRANSACTION SCHEDULING is in NP . From the reduction of the VERTEX COLORING problem, we also obtain that TRANSACTION SCHEDULING is NP -complete.

From the above reduction, we have that an approximation ratio $q(G)$ of the VERTEX COLORING problem implies the existence of a scheduling algorithm \mathcal{A} with competitive ratio $CR_{\mathcal{A}}(\mathcal{T}) = q(G)$ of the respective TRANSACTION SCHEDULING problem instance, and vice-versa. Since $s = |\mathcal{R}| = |E| \leq n^2$, an $(\sqrt{s})^{1-\epsilon}$ competitive ratio of \mathcal{A} implies at most an $n^{1-\epsilon}$ approximation ratio of VERTEX COLORING. Since, we know that unless $NP \subseteq ZPP$, there does not exist a polynomial time algorithm to approximate $\chi(G)$ with approximation ratio $O(n^{1-\epsilon})$ for any constant $\epsilon > 0$, we obtain a symmetric result for the TRANSACTION SCHEDULING problem:

Theorem 3 (Approximation Hardness of Transaction Scheduling). *Unless $NP \subseteq ZPP$, we cannot obtain a polynomial time transaction scheduling algorithm such that for every input instance with $\beta = 1$ and $\ell = 1$ of the TRANSACTION SCHEDULING problem the algorithm achieves competitive ratio smaller than $O((\sqrt{s})^{1-\epsilon})$ for any constant $\epsilon > 0$.*

Theorem 3 implies that the $O(\sqrt{s})$ bound of Algorithm Clairvoyant, given in Corollary 2 for $\beta = \Theta(1)$ and $\ell = O(1)$, is tight.

6 Conclusions

We have studied the competitive ratios achieved by transactional contention managers on balanced workloads. The randomized algorithms presented in this paper allow to achieve best competitive bound on balanced workloads. We also establish hardness results on the competitive ratios in our balanced workload model by reducing the well known NP -complete vertex coloring problem to the transactional scheduling problem.

There are several interesting directions for future work. As advocated in [15], our algorithms are conservative – abort at least one transaction involved in a conflict – as it reduces the cost to track conflicts and dependencies. It is interesting to look whether the other schedulers which are less conservative can give improved competitive ratios by reducing the overall makespan. First, our study can be complemented by studying other performance measures, such as the average response time of transactions under balanced workloads. Second, while we have theoretically analyzed the behavior of balanced workloads, it is interesting to see how our contention managers compare experimentally with prior transactional contention managers, e.g., [5,25,11,1].

References

1. 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. pp. 4–18 (2009)

2. Attiya, H., Epstein, L., Shachnai, H., Tamir, T.: Transactional contention management as a non-clairvoyant scheduling problem. *Algorithmica* 57(1), 44–61 (2010)
3. Attiya, H., Milani, A.: Transactional scheduling for read-dominated workloads. In: OPODIS '09. pp. 3–17 (2009)
4. Dice, D., Shalev, O., Shavit, N.: Transactional locking II. In: DISC '06. pp. 194–208 (2006)
5. Dolev, S., Hendler, D., Suissa, A.: CAR-STM: scheduling-based collision avoidance and resolution for software transactional memory. In: PODC '08. pp. 125–134 (2008)
6. Dragojević, A., Guerraoui, R., Singh, A.V., Singh, V.: Preventing versus curing: avoiding conflicts in transactional memories. In: PODC '09. pp. 7–16 (2009)
7. Feige, U., Kilian, J.: Zero knowledge and the chromatic number. In: CCC '96. pp. 278–287 (1996)
8. Felber, P., Fetzer, C., Riegel, T.: Dynamic performance tuning of word-based software transactional memory. In: PPOPP '08. pp. 237–246 (2008)
9. Garey, M.R., Johnson, D.S.: *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA (1990)
10. Guerraoui, R., Herlihy, M., Kapalka, M., Pochon, B.: Robust Contention Management in Software Transactional Memory. In: SCOOOL '05 (2005)
11. Guerraoui, R., Herlihy, M., Pochon, B.: Toward a theory of transactional contention managers. In: PODC '05. pp. 258–264 (2005)
12. Harris, T., Fraser, K.: Language support for lightweight transactions. In: OOPSLA '03, pp. 388–402. ACM (2003)
13. Harris, T., Marlow, S., Peyton-Jones, S., Herlihy, M.: Composable memory transactions. In: PPOPP '05. pp. 48–60 (2005)
14. Herlihy, M., Luchangco, V., Moir, M.: Obstruction-free synchronization: Double-ended queues as an example. In: ICDCS '03. pp. 522–529 (2003)
15. Herlihy, M., Luchangco, V., Moir, M., Scherer, III, W.N.: Software transactional memory for dynamic-sized data structures. In: PODC '03. pp. 92–101 (2003)
16. Herlihy, M., Moss, J.E.B.: Transactional memory: Architectural support for lock-free data structures. In: ISCA '93. pp. 289–300 (1993)
17. Khot, S.: Improved inapproximability results for maxclique, chromatic number and approximate graph coloring. In: FOCS '01. pp. 600–609 (2001)
18. Marathe, V.J., Spear, M.F., Heriot, C., Acharya, A., Eisenstat, D., Scherer III, W.N., Scott, M.L.: Lowering the overhead of software transactional memory. Tech. Rep. TR 893, Computer Science Department, University of Rochester (2006)
19. 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)
20. Scherer, III, W.N., Scott, M.L.: Advanced contention management for dynamic software transactional memory. In: PODC '05. pp. 240–248 (2005)
21. Schneider, J., Wattenhofer, R.: Bounds on contention management algorithms. In: ISAAC '09. pp. 441–451 (2009)
22. Sharma, G., Busch, C.: A competitive analysis for balanced transactional memory workloads. CoRR abs/1009.0056 (2010)
23. Sharma, G., Estrade, B., Busch, C.: Window-based greedy contention management for transactional memory. In: DISC '10. pp. 64–78 (2010)
24. Shavit, N., Touitou, D.: Software transactional memory. In: PODC '95. pp. 204–213 (1995)
25. Yoo, R.M., Lee, H.H.S.: Adaptive transaction scheduling for transactional memory systems. In: SPAA '08. pp. 169–178 (2008)