

# FBLT: A Real-Time TM Contention Manager with Improved Retry Costs and Real-Time Schedulability

Mohammed Elshambakey  
ECE Dept, Virginia Tech  
Blacksburg, VA 24060, USA  
shambake@vt.edu

Binoy Ravindran  
ECE Dept, Virginia Tech  
Blacksburg, VA 24060, USA  
binoy@vt.edu

**Abstract**—We consider software transactional memory (STM) concurrency control for embedded multicore real-time software, and present a novel contention manager for resolving transactional conflicts, called FBLT. We upper bound transactional retries and task response times under FBLT, and identify when FBLT has better real-time schedulability than previous contention managers and lock-free synchronization. Our implementation in the Rochester STM framework/real-time Linux reveals that FBLT yields shorter or comparable retry costs than competitor methods.

## I. INTRODUCTION

Embedded systems sense physical processes and control their behavior, typically through feedback loops. Since physical processes are concurrent, computations that control them must also be concurrent, enabling them to process multiple streams of sensor input and control multiple actuators, all concurrently. Often, such computations need to concurrently read/write shared data objects. They must also process sensor input and react, while satisfying time constraints.

The de facto standard for concurrent programming is the threads abstraction, and the de facto synchronization abstraction is locks. Lock-based concurrency control has significant programmability, scalability, and composability challenges [1]. Transactional memory (TM) is an alternative synchronization model for shared memory objects that promises to alleviate these difficulties. With TM, code that read/write shared objects is organized as *memory transactions*, which execute speculatively, while logging changes made to objects. Two transactions conflict if they access the same object and at least one access is a write. When that happens, a contention manager (CM) [2] resolves the conflict by aborting one and allowing the other to commit, yielding (the illusion of) atomicity. Aborted transactions are re-started, after rolling back the changes. In addition to a simple programming model, TM provides performance comparable to lock-free approach, especially for high contention and read-dominated workloads (see an example TM system's performance in [3]), and is composable [4]. TM has been proposed in hardware, called HTM, and in software, called STM, with the usual tradeoffs: HTM has lesser overhead, but needs transactional support in hardware; STM is available on any hardware.

Given STM's programmability, scalability, and composability advantages, it is a compelling concurrency control

technique also for multicore embedded real-time software. However, this requires bounding transactional retries, as real-time threads, which subsume transactions, must satisfy time constraints. Retry bounds under STM are dependent on the CM policy at hand.

Past real-time CM research (Section V) has proposed resolving transactional contention using dynamic and fixed priorities of parent threads, resulting in Earliest Deadline CM (ECM) and Rate Monotonic CM (RCM) [5]–[7], which are intended to be used with global EDF (G-EDF) and global RMS (G-RMS) multicore real-time schedulers [8], respectively. In particular, [6] shows that ECM and RCM achieve higher schedulability – i.e., greater number of task sets meeting their time constraints – than lock-free synchronization only under some ranges for the maximum atomic section length. That range is significantly expanded with the Length-based CM (LCM) in [7], increasing the coverage of STM's time-liness superiority. ECM, RCM, and LCM suffer from transitive retry (Section IV-C) and cannot handle multiple objects per transaction efficiently. These limitations are overcome with the Priority with Negative value and First access CM (PNF) [9]. However, PNF requires a-priori knowledge of all objects accessed by each transaction. This significantly limits programmability, and is incompatible with dynamic STM implementations [10]. Additionally, PNF is a centralized CM, which increases overheads and retry costs, and has a complex implementation.

We propose the First Bounded, Last Timestamp CM (or FBLT) (Section V). In contrast to PNF, FBLT does not require a-priori knowledge of objects accessed by transactions. Moreover, FBLT allows each transaction to access multiple objects with shorter transitive retry cost than ECM, RCM and LCM. Additionally, FBLT is a decentralized CM and does not use locks in its implementation. Implementation of FBLT is also simpler than PNF. We establish FBLT's retry and response time upper bounds under G-EDF and G-RMA schedulers (Section VI). We also identify the conditions under which FBLT's schedulability is better than ECM, RCM, G-EDF/LCM, G-RMA/LCM, PNF, and lock-free synchronization (Section VII). We implement FBLT and competitor CM techniques in the Rochester STM framework [11] and conduct experimental studies (Section VIII). Our results reveal that FBLT has shorter retry cost than ECM, RCM, LCM and lock-free. FBLT's retry

cost is slighter higher than PNF, but it doesn't require a-priori knowledge of objects accessed by transactions, unlike PNF.

Thus, the paper's contribution is the FBLT contention manager with superior timeliness properties. FBLT, thus allows programmers to reap STM's significant programmability and composability benefits for a broader range of multicore embedded real-time software than what was previously possible.

## II. RELATED WORK

Transactional-like concurrency control without using locks, for real-time systems, has been previously studied in the context of non-blocking data structures (e.g., [12]). Despite their numerous advantages over locks (e.g., deadlock-freedom), their programmability has remained a challenge. Past studies show that they are best suited for simple data structures where their retry cost is competitive to the cost of lock-based synchronization [13]. In contrast, STM is semantically simpler [1], and is often the only viable lock-free solution for complex data structures (e.g., red/black tree) [14] and nested critical sections [3].

STM concurrency control for real-time systems has been previously studied in [6], [7], [9], [14]–[19].

[15] proposes a restricted version of STM for uniprocessors. Uniprocessors do not need contention management. [16] bounds response times in distributed systems with STM synchronization. They consider Pfair scheduling, limit to small atomic regions with fixed size, and limit transaction execution to span at most two quanta. In contrast, we allow transaction lengths with arbitrary duration.

[17] presents real-time scheduling of transactions and serializes transactions based on deadlines. However, the work does not bound retries and response times. In contrast, we establish such bounds. [18] proposes real-time HTM. The work does not describe how transactional conflicts are resolved. Besides, the retry bound assumes that the worst case conflict between atomic sections of different tasks occurs when the sections are released at the same time. However, we show that this is not the worst case. We develop retry and response time upper bounds based on much worse conditions.

[14] upper bounds retries and response times for ECM with G-EDF, and identify the tradeoffs with locking and lock-free protocols. Similar to [18], [14] also assumes that the worst case conflict between atomic sections occurs when the sections are released simultaneously. The ideas in [14] are extended in [19], which presents three real-time CM designs. But no retry bounds or schedulability analysis techniques are presented for those CMs.

[6] presents the ECM and RCM contention managers, and upper bounds transactional retries and task response times under them. The work also identifies the conditions under which ECM and RCM are superior to locking and lock-free techniques. In particular, [6] shows that, STM's superiority holds only under some ranges for the maximum atomic section length. Moreover, [6] restricts transactions to access only one object. [7] presents the LCM contention manager, and upper bounds its transactional retries and task response times under

the G-EDF and G-RMA schedulers. This work also compares (analytically and experimentally) LCM with ECM, RCM, and lock-free synchronization. However, similar to [6], [7] also restricts transactions to access only one object.

[9] presents the PNF contention manager, which allows transactions to access multiple objects and avoids the consequent transitive retry effect. The work also upper bounds transactional retries and task response times under G-EDF and G-RMA. However, PNF requires a-priori knowledge of the objects accessed by each transaction, which is not always possible, limits programmability, and is incompatible with dynamic STM implementations [10]. Additionally, PNF is a centralized CM and uses locks in its implementation, which increases overheads.

Our work builds upon [6], [7], [9]. FBLT allows multiple objects per transaction with no a-priori knowledge needed about those objects. We upper bound transactional retries and task response times under FBLT, and identify the conditions under which FBLT has better schedulability than other synchronization techniques.

## III. PRELIMINARIES

We consider a multiprocessor system with  $m$  identical processors and  $n$  sporadic tasks  $\tau_1, \tau_2, \dots, \tau_n$ . The  $k^{th}$  instance (or job) of a task  $\tau_i$  is denoted  $\tau_i^k$ . Each task  $\tau_i$  is specified by its worst case execution time (WCET)  $c_i$ , its minimum period  $T_i$  between any two consecutive instances, and its relative deadline  $D_i$ , where  $D_i = T_i$ . Job  $\tau_i^j$  is released at time  $r_i^j$  and must finish no later than its absolute deadline  $d_i^j = r_i^j + D_i$ . Under a fixed priority scheduler such as G-RMA,  $p_i$  determines  $\tau_i$ 's (fixed) priority and it is constant for all instances of  $\tau_i$ . Under a dynamic priority scheduler such as G-EDF, a job  $\tau_i^j$ 's priority,  $p_i^j$ , differs from one instance to another. A task  $\tau_j$  may interfere with task  $\tau_i$  for a number of times during an interval  $L$ , and this number is denoted as  $G_{ij}(L)$ .

*Shared objects.* A task may need to read/write shared, in-memory data objects while it is executing any of its atomic sections (transactions), which are synchronized using STM. The set of atomic sections of task  $\tau_i$  is denoted  $s_i$ .  $s_i^k$  is the  $k^{th}$  atomic section of  $\tau_i$ . Each object,  $\theta$ , can be accessed by multiple tasks. The set of distinct objects accessed by  $\tau_i$  is  $\theta_i$  without repeating objects. The set of atomic sections used by  $\tau_i$  to access  $\theta$  is  $s_i(\theta)$ , and the sum of the lengths of those atomic sections is  $len(s_i(\theta))$ .  $s_i^k(\theta)$  is the  $k^{th}$  atomic section of  $\tau_i$  that accesses  $\theta$ .  $s_i^k$  can access one or more objects in  $\theta_i$ . So,  $s_i^k$  refers to the transaction itself, regardless of the objects accessed by the transaction. We denote the set of all accessed objects by  $s_i^k$  as  $\Theta_i^k$ . While  $s_i^k(\theta)$  implies that  $s_i^k$  accesses an object  $\theta \in \Theta_i^k$ ,  $s_i^k(\Theta)$  implies that  $s_i^k$  accesses a set of objects  $\Theta = \{\theta \in \Theta_i^k\}$ .  $\bar{s}_i^k = s_i^k(\Theta)$  refers only once to  $s_i^k$ , regardless of the number of objects in  $\Theta$ . So,  $|s_i^k(\Theta)|_{\forall \theta \in \Theta} = 1$ .  $s_i^k(\theta)$  executes for a duration  $len(s_i^k(\theta))$ .  $len(s_i^k) = len(s_i^k(\theta)) = len(s_i^k(\Theta)) = len(s_i^k(\Theta_i^k))$  The set of tasks sharing  $\theta$  with  $\tau_i$  is denoted  $\gamma_i(\theta)$ .

Atomic sections are non-nested (supporting nested STM is future work). The maximum-length atomic section in  $\tau_i$  that accesses  $\theta$  is denoted  $s_{i_{max}}(\theta)$ , while the maximum one among all tasks is  $s_{max}(\theta)$ , and the maximum one among tasks with priorities lower than that of  $\tau_i$  is  $s_{i_{max}}^i(\theta)$ .

**STM retry cost.** If two or more atomic sections conflict, the CM will commit one section and abort and retry the others, increasing the time to execute the aborted sections. The increased time that an atomic section  $s_i^p(\theta)$  will take to execute due to a conflict with another section  $s_j^k(\theta)$ , is denoted  $W_i^p(s_j^k(\theta))$ . If an atomic section,  $s_i^p$ , is already executing, and another atomic section  $s_j^k$  tries to access a shared object with  $s_i^p$ , then  $s_j^k$  is said to “interfere” or “conflict” with  $s_i^p$ . The job  $s_j^k$  is the “interfering job”, and the job  $s_i^p$  is the “interfered job”.

Due to *transitive retry* (introduced in Section IV-C), an atomic section  $s_i^k(\Theta_i^k)$  may retry due to another atomic section  $s_j^l(\Theta_j^l)$ , where  $\Theta_i^k \cap \Theta_j^l = \emptyset$ .  $\theta_i^*$  denotes the set of objects not accessed directly by atomic sections in  $\tau_i$ , but can cause transactions in  $\tau_i$  to retry due to transitive retry.  $\theta_i^{ex} (= \theta_i + \theta_i^*)$  is the set of all objects that can cause transactions in  $\tau_i$  to retry directly or through transitive retry.  $\gamma_i^*$  is the set of tasks that accesses objects in  $\theta_i^*$ .  $\gamma_i^{ex} (= \gamma_i + \gamma_i^*)$  is the set of all tasks that can directly or indirectly (through transitive retry) cause transactions in  $\tau_i$  to abort and retry.

The total time that a task  $\tau_i$ ’s atomic sections have to retry over  $T_i$  is denoted  $RC(T_i)$ . The additional amount of time by which all interfering jobs of  $\tau_j$  increases the response time of any job of  $\tau_i$  during  $L$ , without considering retries due to atomic sections, is denoted  $W_{ij}(L)$ .

#### IV. MOTIVATION

To understand the need for *First Bounded, Last Timestamp (FBLT) contention manager*, we first give a brief introduction of previous real-time CMs, ECM [6], RCM [6], LCM [7], and PNF [9].

##### A. ECM and RCM

The *Earliest Deadline Contention Manager* (ECM) [6] is used with the G-EDF multicore real-time scheduler. ECM allows the transaction with the shortest absolute deadline to commit first. The other transactions retry and abort.

The *Rate Monotonic Contention Manager* (RCM) [6] is used with the G-RMA scheduler. RCM allows the transaction with the shortest period to commit first. ECM and RCM maintain semantic consistency with the underlying scheduler.

##### B. LCM

For both ECM and RCM,  $s_i^k(\theta)$  can be totally repeated if  $s_j^l(\theta)$  — which belongs to a higher priority job  $\tau_j^b$  than  $\tau_i^a$  — conflicts with  $s_i^k(\theta)$  at the end of its execution, while  $s_i^k(\theta)$  is just about to commit. The *Length-based Contention Manager* (LCM) [7], shown in Algorithm 1, uses the remaining length of  $s_i^k(\theta)$  when it is interfered, as well as  $len(s_j^l(\theta))$ , to decide which transaction must be aborted. If  $p_i^k$  is greater than  $p_j^l$ , then  $s_i^k(\theta)$  is committed, because it belongs to a higher priority

job, and it started before  $s_j^l(\theta)$  (step 2). Otherwise,  $c_{ij}^{kl}$  is calculated (step 4) to determine whether it is worth aborting  $s_i^k(\theta)$  in favor of  $s_j^l(\theta)$ , because  $len(s_j^l(\theta))$  is relatively small compared to the remaining execution length of  $s_i^k(\theta)$ . [7] assumes that:

$$c_{ij}^{kl} = len(s_j^l(\theta)) / len(s_i^k(\theta)) \quad (1)$$

where  $c_{ij}^{kl} \in ]0, \infty[$ , to cover all possible lengths of  $s_j^l(\theta)$ .

Thus, LCM’s key idea is to reduce the opportunity for the abort of  $s_i^k(\theta)$  if it is close to committing when interfered and  $len(s_j^l(\theta))$  is large. This abort opportunity is increasingly reduced as  $s_i^k(\theta)$  gets closer to the end of its execution, or  $len(s_j^l(\theta))$  gets larger. On the other hand, as  $s_i^k(\theta)$  is interfered early, or  $len(s_j^l(\theta))$  is small compared to  $s_i^k(\theta)$ ’s remaining length, the abort opportunity is increased even if  $s_i^k(\theta)$  is close to the end of its execution. To decide whether  $s_i^k(\theta)$  must be aborted or not, a threshold value  $\psi \in [0, 1]$  that determines  $\alpha_{ij}^{kl}$  (step 5) is used, where  $\alpha_{ij}^{kl}$  is the maximum percentage of  $len(s_i^k(\theta))$  below which  $s_j^l(\theta)$  is allowed to abort  $s_i^k(\theta)$ . Thus, if the already executed part of  $s_i^k(\theta)$  — when  $s_j^l(\theta)$  interferes with  $s_i^k(\theta)$  — does not exceed  $\alpha_{ij}^{kl} len(s_i^k(\theta))$ , then  $s_i^k(\theta)$  is aborted (step 8). Otherwise,  $s_j^l(\theta)$  is aborted (step 10).

---

#### ALGORITHM 1: The LCM Algorithm

---

**Data:**  $s_i^k(\theta) \rightarrow$  interfered atomic section.  
 $s_j^l(\theta) \rightarrow$  interfering atomic section.  
 $\psi \rightarrow$  predefined threshold  $\in [0, 1]$ .  
 $\delta_i^k(\theta) \rightarrow$  remaining execution length of  $s_i^k(\theta)$   
**Result:** which atomic section of  $s_i^k(\theta)$  or  $s_j^l(\theta)$  aborts

```

1 if  $p_i^k > p_j^l$  then
2   |  $s_j^l(\theta)$  aborts;
3 else
4   |  $c_{ij}^{kl} = len(s_j^l(\theta)) / len(s_i^k(\theta));$ 
5   |  $\alpha_{ij}^{kl} = \ln(\psi) / (\ln(\psi) - c_{ij}^{kl});$ 
6   |  $\alpha = (len(s_i^k(\theta)) - \delta_i^k(\theta)) / len(s_i^k(\theta));$ 
7   | if  $\alpha \leq \alpha_{ij}^{kl}$  then
8   |   |  $s_i^k(\theta)$  aborts;
9   | else
10  |   |  $s_j^l(\theta)$  aborts;
11  | end
12 end
```

---

LCM reduces the retry cost of a single transaction  $s_i^k(\theta)$  due to another transaction  $s_j^l(\theta)$  from  $2 \cdot s_{max}$  (in case of ECM and RCM) to  $(1 + \alpha_{max}) \cdot s_{max}$ , where  $s_{max}$  is the maximum length transaction among all tasks, and  $\alpha_{max}$  is the maximum *alpha* for any transaction. On the other hand, LCM suffers from bounded priority inversion because a higher priority transaction can be blocked by a lower priority one. Additionally, LCM is not a centralized CM, which means that, upon a conflict, each transaction must decide whether it must commit or abort.

##### C. PNF

ECM, RCM, and LCM suffer from *transitive retry*. Transitive retry is illustrated by the following example:

Consider three atomic sections  $s_1^x$ ,  $s_2^y$ , and  $s_3^z$  belonging to jobs  $\tau_1^x$ ,  $\tau_2^y$ , and  $\tau_3^z$ , with priorities  $p_3^z > p_2^y > p_1^x$ , respectively.

Assume that  $s_1^x$  and  $s_2^y$  share objects, and  $s_2^y$  and  $s_3^z$  share objects.  $s_1^x$  and  $s_3^z$  do not share objects. Now,  $s_3^z$  can cause  $s_2^y$  to retry, which in turn will cause  $s_1^x$  to retry. This means that  $s_1^x$  will retry transitively because of  $s_3^z$ , which will increase the retry cost of  $s_1^x$ .

Now, consider another atomic section  $s_4^f$  with a priority higher than that of  $s_3^z$ . Suppose  $s_4^f$  shares objects only with  $s_3^z$ . Thus,  $s_4^f$  can cause  $s_3^z$  to retry, which in turn will cause  $s_2^y$  to retry, and finally,  $s_1^x$  to retry. Thus, transitive retry will move from  $s_4^f$  to  $s_1^x$ , increasing the retry cost of  $s_1^x$ . The situation gets worse as more higher priority tasks are added, where each task shares objects with its immediate lower priority task.  $\tau_3^z$  may have atomic sections that share objects with  $\tau_1^x$ , but this will not prevent the effect of transitive retry due to  $s_1^x$ .

**Definition 1: Transitive retry.** A transaction  $s_i^k$  suffers from transitive retry when  $s_i^k$  retries due to a higher priority transaction  $s_z^h$ , and  $\Theta_z^h \cap \Theta_i^k = \emptyset$ .

Therefore, the analysis in [6] and [7] extend the set of objects that can cause an atomic section of a lower priority job to retry. This is done by initializing the set of conflicting objects,  $\gamma_i$ , to all objects accessed by all transactions of  $\tau_i$ . We then cycle through all transactions belonging to all other higher priority tasks. Each transaction  $s_j^l$  that accesses at least one of the objects in  $\gamma_i$  adds all other objects accessed by  $s_j^l$  to  $\gamma_i$ . The loop over all higher priority tasks is repeated, each time with the new  $\gamma_i$ , until there are no more transactions accessing any object in  $\gamma_i$ . The final set of objects (tasks) that can cause transactions in  $\tau_i$  to retry is  $\theta_i^{ex}(\gamma_i^{ex})$ , respectively<sup>1</sup>.

ECM, RCM, and LCM suffer from transitive retry. The (PNF) [9] is designed to avoid transitive retry by concurrently executing at most  $m$  non-conflicting transactions together. These executing transactions are non-preemptive. Thus, executing transactions cannot be aborted due to direct or indirect conflict with other transactions. However, with PNF, all objects accessed by each transaction must be known a-priori. Therefore, this is not suitable with dynamic STM implementations [10]. Additionally, PNF is implemented in [9] as a centralized CM that uses locks. This increases overhead.

#### D. Case for FBLT

Thus, it is desirable to have a CM with the following goals:

- 1) reduce the retry cost of each transaction  $s_i^k$  due to another transaction  $s_j^l$ , just as LCM does compared to ECM and RCM.
- 2) avoid or bound the effect of transitive retry, similar to PNF, without prior knowledge of accessed objects by each transaction, enabling dynamic STM.
- 3) decentralized design and avoid the use of locks, thereby reducing overhead.

We propose the *First Bounded, Last Timestamp contention manager* (or FBLT). FBLT achieves these goals by bounding the number of times each transaction  $s_i^k$  is aborted due to other transactions to at most  $\delta_i^k$ .  $\delta_i^k$  includes the number of

aborts due to direct conflict with other transactions, as well as transitive retry (goal 2). If a transaction  $s_i^k$  reaches its  $\delta_i^k$ , it is added to an  $m\_set$  in FIFO order. In the  $m\_set$ ,  $s_i^k$  executes non-preemptively. If transactions in the  $m\_set$  conflict together, they use their FIFO order in the  $m\_set$  to resolve the conflict.  $s_i^k$  can still abort after it becomes a non-preemptive transaction due to other non-preemptive transactions. The number of aborts for any non-preemptive transaction is bounded by  $m - 1$ , where  $m$  is the number of processors, as will be shown in Section VI.

Thus, the key idea behind FBLT is to use a suitable  $\delta_i^k$  for each  $s_i^k$  before it becomes a non-preemptive transaction. The choice of  $\delta_i^k$  should make the total retry cost (and thus, the schedulability) of any job  $\tau_i^x$  under FBLT comparable to the retry cost under ECM, RCM, LCM, and PNF. (In Section VII, we show the suitable  $\delta_i^k$  for each  $s_i^k$  to have equal or better schedulability than other CMs.) Preemptive transactions resolve their conflicts using LCM. Thus, FBLT defaults to LCM if abort bounds have not been violated (goal 1). Each non-preemptive transaction  $s_i^k$  uses the time it joined the  $m\_set$  to resolve conflicts with other non-preemptive transactions. Therefore, FBLT does not have to use locks and is decentralized (goal 3).

#### V. THE FBLT CONTENTION MANAGER

Algorithm 2 illustrates FBLT. Each transaction  $s_i^k$  can be aborted during  $T_i$  for at most  $\delta_i^k$  times.  $\eta_i^k$  records the number of times  $s_i^k$  has already been aborted up to now. If  $s_i^k$  and  $s_j^l$  have not joined the  $m\_set$  yet, then they are preemptive transactions. Preemptive transactions resolve conflicts using Algorithm 1 (step 2). Thus, FBLT defaults to LCM when no transaction reaches its  $\delta$ . If only one of the transactions is in the  $m\_set$ , then the non-preemptive transaction (the one in  $m\_set$ ) aborts the other one (steps 15 to 26).  $\eta_i^k$  is incremented each time  $s_i^k$  is aborted as long as  $\eta_i^k < \delta_i^k$  (steps 5 and 18). Otherwise,  $s_i^k$  is added to the  $m\_set$  and its priority is increased to  $m\_prio$  (steps 7 to 9 and 20 to 22). When the priority of  $s_i^k$  is increased to  $m\_prio$ ,  $s_i^k$  becomes a non-preemptive transaction. Non-preemptive transactions cannot be aborted by other preemptive transactions, nor by any other real-time job. The  $m\_set$  can hold at most  $m$  concurrent transactions because there are  $m$  processors in the system.  $r(s_i^k)$  records the time  $s_i^k$  joined the  $m\_set$  (steps 8 and 21). When non-preemptive transactions conflict together (step 27), the transaction with the smaller  $r()$  commits first (steps 29 and 31). Thus, non-preemptive transactions are executed in FIFO order of the  $m\_set$ .

##### A. Illustrative Example

We now illustrate FBLT's behavior with the following example:

- 1) Transaction  $s_i^k(\theta_1, \theta_2)$  is released while  $m\_set = \emptyset$ .  $\eta_i^k = 0$  and  $\delta_i^k = 3$ .
- 2) Transaction  $s_a^b(\theta_2)$  is released while  $s_i^k(\theta_1, \theta_2)$  is running.  $p_a^b > p_i^k$  and  $\eta_i^k < \delta_i^k$ . Applying LCM,  $s_i^k(\theta_1, \theta_2)$  is aborted in favor of  $s_a^b$  and  $\eta_i^k$  is incremented to 1.

<sup>1</sup>However, note that, this solution may over-extend the set of conflicting objects, and may even contain all objects accessed by all tasks.

---

**ALGORITHM 2: The FBLT Algorithm**


---

**Data:**  $s_i^k$ : interfered transaction;  
 $s_j^l$ : interfering transactions;  
 $\delta_i^k$ : the maximum number of times  $s_i^k$  can be aborted during  $T_i$ ;  
 $\eta_i^k$ : number of times  $s_i^k$  has already been aborted up to now;  
 $m\_set$ : contains at most  $m$  non-preemptive transactions.  $m$  is number of processors;  
 $m\_prio$ : priority of any transaction in  $m\_set$ .  $m\_prio$  is higher than any priority of any real-time task;  
 $r(s_i^k)$ : time point at which  $s_i^k$  joined  $m\_set$ ;  
**Result:** atomic sections that will abort

```

1  if  $s_i^k, s_j^l \notin m\_set$  then
2      Apply Algorithm 1 (default to LCM);
3      if  $s_i^k$  is aborted then
4          if  $\eta_i^k < \delta_i^k$  then
5              Increment  $\eta_i^k$  by 1;
6          else
7              Add  $s_i^k$  to  $m\_set$ ;
8              Record  $r(s_i^k)$ ;
9              Increase priority of  $s_i^k$  to  $m\_prio$ ;
10         end
11     else
12         Swap  $s_i^k$  and  $s_j^l$ ;
13         Go to Step 3;
14     end
15 else if  $s_j^l \in m\_set, s_i^k \notin m\_set$  then
16     Abort  $s_i^k$ ;
17     if  $\eta_i^k < \delta_i^k$  then
18         Increment  $\eta_i^k$  by 1;
19     else
20         Add  $s_i^k$  to  $m\_set$ ;
21         Record  $r(s_i^k)$ ;
22         Increase priority of  $s_i^k$  to  $m\_prio$ ;
23     end
24 else if  $s_i^k \in m\_set, s_j^l \notin m\_set$  then
25     Swap  $s_i^k$  and  $s_j^l$ ;
26     Go to Step 15;
27 else
28     if  $r(s_i^k) < r(s_j^l)$  then
29         Abort  $s_j^l$ ;
30     else
31         Abort  $s_i^k$ ;
32     end
33 end

```

---

- 3)  $s_a^b(\theta_2)$  commits.  $s_i^k(\theta_1, \theta_2)$  runs again. Transaction  $s_c^d(\theta_2)$  is released while  $s_i^k(\theta_1, \theta_2)$  is running.  $p_c^d > p_i^k$ . Applying LCM,  $s_i^k(\theta_1, \theta_2)$  is aborted again in favor of  $s_c^d(\theta_2)$ .  $\eta_i^k$  is incremented to 2.
- 4)  $s_c^d(\theta_2)$  commits.  $s_e^f(\theta_2, \theta_3)$  is released.  $p_e^f > p_i^k$  and  $\eta_e^f = 2$ .  $s_i^k(\theta_1, \theta_2)$  is aborted in favor of  $s_e^f(\theta_2, \theta_3)$  and  $\eta_i^k$  is incremented to 3.
- 5)  $s_j^l(\theta_3)$  is released.  $p_j^l > p_e^f$ .  $s_e^f(\theta_2, \theta_3)$  is aborted in favor of  $s_j^l(\theta_3)$  and  $\eta_e^f$  is incremented to 1.
- 6)  $s_i^k(\theta_1, \theta_2)$  and  $s_e^f(\theta_2, \theta_3)$  are compared again.  $\therefore \eta_i^k = \delta_i^k$ ,  $\therefore s_i^k(\theta_1, \theta_2)$  is added to  $m\_set$ .  $m\_set = \{s_i^k(\theta_1, \theta_2)\}$ .  $s_i^k(\theta_1, \theta_2)$  becomes a non-preemptive transaction. As  $s_e^f(\theta_2, \theta_3)$  is a preemptive transaction,  $\therefore s_e^f(\theta_2, \theta_3)$  is aborted in favor of  $s_i^k(\theta_1, \theta_2)$ , despite  $p_e^f$  being greater than the original priority of  $s_i^k(\theta_1, \theta_2)$ .  $\eta_e^f$  is incremented to 2.
- 7)  $s_j^l(\theta_3)$  commits but  $s_g^h(\theta_3)$  is released.  $p_g^h > p_e^f$  but  $\eta_e^f =$

$\delta_e^f$ . So,  $s_e^f(\theta_2, \theta_3)$  becomes a non-preemptive transaction.  $m\_set = \{s_i^k(\theta_1, \theta_2), s_g^h(\theta_2, \theta_3)\}$ .

- 8)  $s_i^k(\theta_1, \theta_2)$  and  $s_g^h(\theta_2, \theta_3)$  are now non-preemptive transactions.  $s_i^k(\theta_1, \theta_2)$  and  $s_g^h(\theta_2, \theta_3)$  still conflict together. So, they are executed according to their addition order to the  $m\_set$ . So,  $s_i^k(\theta_1, \theta_2)$  commits first, followed  $s_g^h(\theta_2, \theta_3)$ .
- 9)  $s_g^h(\theta_3)$  will continue to abort and retry in favor of  $s_e^f(\theta_2, \theta_3)$  until  $s_e^f(\theta_2, \theta_3)$  commits or  $\eta_g^h = \delta_g^h$ . Even if  $s_g^h(\theta_3)$  joined the  $m\_set$ ,  $s_g^h(\theta_3)$  will still abort and retry in favor of  $s_e^f(\theta_2, \theta_3)$ , because  $s_e^f(\theta_2, \theta_3)$  joined the  $m\_set$  earlier than  $s_g^h(\theta_3)$ .

It is seen from steps 2 to 6 that  $s_i^k(\theta_1, \theta_2)$  can be aborted due to direct conflict with other transactions, or due to transitive retry. Irrespective of the reason for the conflict, once a transaction has reached its maximum allowed  $\delta$ , the transaction becomes a non-preemptive one (steps 6 and 7). Non-preemptive transactions have higher priority than other preemptive transactions (steps 6 and 7). Non-preemptive transactions execute in their arrival order to the  $m\_set$ .

## VI. RETRY COST AND RESPONSE TIME BOUNDS

We now derive an upper bound on the retry cost of any job  $\tau_i^x$  under FBLT during an interval  $L \leq T_i$ . Since all tasks are sporadic (i.e., each task  $\tau_i$  has a minimum period  $T_i$ ),  $T_i$  is the maximum study interval for each task  $\tau_i$ .

*Claim 1:* The total retry cost for any job  $\tau_i^x$  under FBLT due to 1) conflicts between its transactions and transactions of other jobs during an interval  $L \leq T_i$  and 2) release of higher priority jobs is upper bounded by:

$$RC_{to}(L) \leq \sum_{\forall s_i^k \in s_i} \left( \delta_i^k \text{len}(s_i^k) + \sum_{\forall s_{iz}^k \in \chi_i^k} \text{len}(s_{iz}^k) \right) + RC_{re}(L) \quad (2)$$

where  $\chi_i^k$  is the set of at most  $m-1$  maximum length transactions conflicting directly or indirectly (through transitive retry) with  $s_i^k$ . Each transaction  $s_{iz}^k \in \chi_i^k$  belongs to a distinct task  $\tau_j$ .  $RC_{re}(L)$  is the retry cost resulting from the release of higher priority jobs which preempt  $\tau_i^x$ .  $RC_{re}(L)$  is calculated by (6.8) in [9] for G-EDF, and (6.10) in [9] for G-RMA schedulers.

*Proof:* By the definition of FBLT,  $s_i^k \in \tau_i^x$  can be aborted a maximum of  $\delta_i^k$  times before  $s_i^k$  joins the  $m\_set$ . Before joining the  $m\_set$ ,  $s_i^k$  can be aborted due to higher priority transactions, or transactions in the  $m\_set$ . The original priority of transactions in the  $m\_set$  can be higher or lower than  $p_i^x$ . Thus, the maximum time  $s_i^k$  is aborted before joining the  $m\_set$  occurs if  $s_i^k$  is aborted for  $\delta_i^k$  times.

Transactions preceding  $s_i^k$  in the  $m\_set$  can conflict directly with  $s_i^k$ , or indirectly through transitive retry. The worst case scenario for  $s_i^k$  after joining the  $m\_set$  occurs if  $s_i^k$  is preceded by  $m-1$  maximum length conflicting transactions. Hence, in the worst case,  $s_i^k$  has to wait for the previous  $m-1$  transactions to commit first. The priority of  $s_i^k$  after joining the  $m\_set$  is higher than any real-time job. Therefore,  $s_i^k$  is

not aborted by any job. If  $s_i^k$  has not joined the  $m\_set$  yet, and a higher priority job  $\tau_j^y$  is released while  $s_i^k$  is running, then  $s_i^k$  may be aborted if  $\tau_j^y$  has conflicting transactions with  $s_i^k$ .  $\tau_j^y$  causes only one abort in  $\tau_i^x$  because  $\tau_j^y$  preempts  $\tau_i^x$  only once. If  $s_i^k$  has already joined the  $m\_set$ , then  $s_i^k$  cannot be aborted by the release of higher priority jobs. Thus, the maximum number of times transactions in  $\tau_i^x$  can be aborted due to the release of higher priority jobs is less than or equal to the number of interfering higher priority jobs to  $\tau_i^x$ . Claim follows. ■

*Claim 2:* Under FBLT, the blocking time of a job  $\tau_i^x$  due to lower priority jobs is upper bounded by:

$$D(\tau_i^x) = \min \left( \max_1^m (s_{j_{max}, \forall \tau_j^l, p_j^l < p_i^x}) \right) \quad (3)$$

where  $s_{j_{max}}$  is the maximum length transaction in any job  $\tau_j^l$  with original priority lower than  $p_i^x$ . The right hand side of (3) is the minimum of the  $m$  maximum transactional lengths in all jobs with lower priority than  $\tau_i^x$ .

*Proof:*  $\tau_i^x$  is blocked when it is initially released and all processors are busy with lower priority jobs with non-preemptive transactions. Although  $\tau_i^x$  can be preempted by higher priority jobs,  $\tau_i^x$  cannot be blocked after it is released. If  $\tau_i^x$  is preempted by a higher priority job  $\tau_j^y$ , then, when  $\tau_j^y$  finishes execution, the underlying scheduler will not choose a lower priority job than  $\tau_i^x$  before  $\tau_i^x$ . So, after  $\tau_i^x$  is released, there is no chance for any transaction  $s_u^v$  belonging to a lower priority job than  $\tau_i^x$  to run before  $\tau_i^x$ . Thus,  $s_u^v$  cannot join the  $m\_set$  before  $\tau_i^x$  finishes. Consequently, the worst case blocking time for  $\tau_i^x$  occurs when the maximum length  $m$  transactions in lower priority jobs than  $\tau_i^x$  are executing non-preemptively. After the minimum length transaction in the  $m\_set$  finishes, the underlying scheduler will choose  $\tau_i^x$  or a higher priority job to run. Claim follows. ■

*Claim 3:* The response time of any job  $\tau_i^x$  during an interval  $L \leq T_i$  under FBLT is upper bounded by:

$$R_i^{up} = c_i + RC_{to}(L) + D(\tau_i^x) + \left\lceil \frac{1}{m} \sum_{\forall j \neq i} W_{ij}(R_i^{up}) \right\rceil \quad (4)$$

where  $RC_{to}(L)$  is calculated by (2),  $D(\tau_i^x)$  is calculated by (3), and  $W_{ij}(R_i^{up})$  is calculated by (11) in [6] for G-EDF, and (17) in [6] for G-RMA schedulers. (11) and (17) in [6] inflates  $c_j$  of any job  $\tau_j^y \neq \tau_i^x$ ,  $p_j^y > p_i^x$  by the retry cost of transactions in  $\tau_j^y$ .

*Proof:* The response time of a job is calculated directly from FBLT's behavior. The response time of any job  $\tau_i^x$  is the sum of its worst case execution time  $c_i$ , plus the retry cost of transactions in  $\tau_i^x$  ( $RC_{to}(L)$ ), plus the blocking time of  $\tau_i^x$  ( $D(\tau_i^x)$ ), and the workload interference of higher priority jobs. The workload interference of higher priority jobs scheduled by G-EDF is calculated by (11) in [6], and by (17) in [6] for G-RMA. Claim follows. ■

## VII. SCHEDULABILITY COMPARISON

We now (formally) compare the schedulability of G-EDF (G-RMA) with FBLT against ECM, RCM, LCM, PNF, and

lock-free synchronization [6], [7], [9], [20]. Such a comparison will reveal when FBLT outperforms the others. Toward this, we compare the total utilization under G-EDF (G-RMA)/FBLT with that under the other synchronization methods. In this comparison, we use the inflated execution time of the task, which is the sum of the worst-case execution time of the task and its retry cost, in the utilization calculation of the task.

Note that, for a job  $\tau_i^x$ , no processor is available during its blocking time. Since each processor is busy with some job other than  $\tau_i^x$ ,  $D(\tau_i^x)$  is not added to the inflated execution time of  $\tau_i^x$ . Hence,  $D(\tau_i^x)$  is not added to the utilization calculation of  $\tau_i^x$ .

Let  $RC_A(T_i)$  and  $RC_B(T_i)$  denote the retry cost of a job  $\tau_i^x$  during  $T_i$  using the synchronization method  $A$  and synchronization method  $B$ , respectively. Now, schedulability of  $A$  is comparable to  $B$  if:

$$\sum_{\forall \tau_i} \frac{c_i + RC_A(T_i)}{T_i} \leq \sum_{\forall \tau_i} \frac{c_i + RC_B(T_i)}{T_i} \quad (5)$$

$$\sum_{\forall \tau_i} \frac{RC_A(T_i)}{T_i} \leq \sum_{\forall \tau_i} \frac{RC_B(T_i)}{T_i}$$

### A. FBLT vs. ECM

*Claim 4:* The schedulability of FBLT is equal to or better than ECM's when the maximum abort number of any preemptive transaction  $s_i^k$  is less than or equal to the number of transactions directly conflicting with  $s_i^k$  in all other jobs with higher priority than  $\tau_i$ 's current job.

*Proof:*

By substituting  $RC_A(T_i)$  and  $RC_B(T_i)$  in (5) with (2) and (6.7) in [9], respectively, we get:

$$\sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k \in s_i} \left( \delta_i^k \text{len}(s_i^k) + \sum_{s_{iz}^k \in \chi_i^k} \text{len}(s_{iz}^k) \right) + RC_{re}(T_i)}{T_i} \leq \sum_{\forall \tau_i} \frac{\left( \sum_{\forall \tau_j \in \gamma_i^{ex}} \sum_{\theta \in \theta_i^{ex}} \left( \left\lceil \frac{T_i}{T_j} \right\rceil \sum_{\forall s_j^h(\theta)} \text{len}(s_j^h(\theta) + s_{max}^j(\theta)) \right) \right)}{T_i} + \sum_{\forall \tau_i} \frac{RC_{re}(T_i)}{T_i} \quad (6)$$

Let  $\theta_i^{ex} = \theta_i + \theta_i^*$ , where  $\theta_i^*$  is the set of objects not accessed directly by  $\tau_i$  but can cause transactions in  $\tau_i$  to retry due to transitive retry. Let  $\gamma_i^{ex} = \gamma_i + \gamma_i^*$ , where  $\gamma_i^*$  is the set of tasks that access objects in  $\theta_i^*$ .  $s_j^h(\theta)$  can access multiple objects, so  $s_{max}^j(\theta)$  is the maximum length transaction conflicting with  $s_j^h(\theta)$ .  $s_j^h(\theta)$  is included only once for all  $\theta \in \theta_j^h$ . Each  $\theta \in \theta_i^{ex}$  has its own  $s_{max}^j(\theta)$ . But  $s_i^h$  can access multiple objects, denoted as  $\theta_j^h$ . So,  $s_{max}^j(\theta)$  is replaced by  $s_{max}^j(\theta_j^h)$ , where  $s_{max}^j(\theta_j^h) = \max\{s_{max}^j(\theta), \forall \theta \in \theta_j^h\}$ .  $s_{max}^j(\theta_j^h)$  is included once for each  $\theta \in \theta_i$ .

Each job  $\tau_i^x$  has the same interference pattern from higher priority jobs,  $\tau_j^h$ , under FBLT and ECM. Hence,  $RC_{re}(T_i)$  for  $\tau_i^x$  is the same under FBLT and ECM. Consequently, (6) becomes:

$$\sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k \in s_i} \left( \delta_i^k \text{len}(s_i^k) + \sum_{s_{iz}^k \in \chi_i^k} \text{len}(s_{iz}^k) \right)}{T_i} \leq \sum_{\forall \tau_i} \frac{\left( \sum_{\forall \tau_j \in \gamma_i^{ex}} \sum_{\forall s_j^h(\theta_j^h), \theta_j^h \in \theta_i^{ex}} \left( \left\lceil \frac{T_i}{T_j} \right\rceil \text{len}(s_j^h(\theta_j^h) + s_{max}^j(\theta_j^h)) \right) \right)}{T_i} \quad (7)$$

Although different  $s_i^k$ s can have common conflicting transactions  $\bar{s}_j^h$ , no more than one  $s_i^k$  can be preceded by the same  $\bar{s}_j^h$  in the  $m\_set$ . This happens because transactions in the  $m\_set$  are non-preemptive. The original priority of transactions preceding  $s_i^k$  in the  $m\_set$  can be lower or higher than the original priority of  $s_i^k$ . Since under G-EDF,  $\tau_j$  can have at least one job of higher priority than  $\tau_i$ ,  $\lceil \frac{T_i}{T_j} \rceil \geq 1$ . Thus, each one of the  $s_{iz}^k$  term in the left hand side of (7) is included in one of the  $s_j^h(\theta)$  term in the right hand side of (7). Now, (7) holds if:

$$\sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k \in s_i} \delta_i^k \text{len}(s_i^k)}{T_i} \leq \sum_{\forall \tau_i} \frac{\sum_{\forall \tau_j \in \gamma_i^{ex}} \sum_{\forall \bar{s}_j^h(\Theta), \Theta \in (\theta_i \cap \Theta_j^h)} \left( \lceil \frac{T_i}{T_j} \rceil \text{len}(s_{max}^j(\Theta_j^h)) \right)}{T_i} \quad (8)$$

Since FBLT is required to bound the effect of transitive retry, only  $\theta_i$  (not the whole  $\theta_i^{ex}$ ) will be considered in (8). Thus, ECM acts as if there were no transitive retry. Consequently, (8) holds if:

$$\sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k \in s_i} \delta_i^k \text{len}(s_i^k)}{T_i} \leq \sum_{\forall \tau_i} \frac{\sum_{\forall \tau_j \in \gamma_i} \sum_{\forall \bar{s}_j^h(\Theta), \Theta \in (\theta_i \cap \Theta_j^h)} \left( \lceil \frac{T_i}{T_j} \rceil \text{len}(s_{max}^j(\Theta)) \right)}{T_i} \quad (9)$$

where  $s_{max}^j(\Theta) \leq s_{max}^j(\Theta_j^h)$ .

For each  $s_i^k \in s_i$ , there are a set of zero or more  $\bar{s}_j^h(\Theta) \in \tau_j$ ,  $\forall \tau_j \neq \tau_i$  that are conflicting with  $s_i^k$ . Assuming this set of transactions conflicting with  $s_i^k$  is denoted as

$$\nu_i^k = \left\{ \bar{s}_j^h(\Theta) \in \tau_j : (\Theta \in (\theta_i \cap \Theta_j^h)) \wedge (\forall \tau_j \neq \tau_i) \wedge \left( \bar{s}_j^h(\Theta) \notin \nu_i^l, l \neq k \right) \right\}$$

The last condition  $\bar{s}_j^h(\Theta) \notin \nu_i^l, l \neq k$  in the definition of  $\nu_i^k$  ensures that common transactions  $\bar{s}_j^h$  that can conflict with more than one transaction  $s_i^k \in \tau_i$  are split among different  $\nu_i^k, k = 1, \dots, |s_i|$ . This condition is necessary, because in ECM, no two or more transactions of  $\tau_i^x$  can be aborted by the same transaction of  $\tau_j^h$ , where  $p_j^h > p_i^x$ . By substitution of  $\nu_i^k$  in (9), we get:

$$\sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k \in s_i} \delta_i^k \text{len}(s_i^k)}{T_i} \leq \sum_{\forall \tau_i} \frac{\left( \sum_{\forall k=1}^{|s_i|} \sum_{\forall \bar{s}_j^h(\Theta) \in \nu_i^k} \left( \lceil \frac{T_i}{T_j} \rceil \text{len}(s_{max}^j(\Theta)) \right) \right)}{T_i} \quad (10)$$

(10) holds if for each  $s_i^k \in \tau_i$ :

$$\delta_i^k \leq \frac{\sum_{\bar{s}_j^h(\Theta) \in \nu_i^k} \left( \lceil \frac{T_i}{T_j} \rceil \text{len}(s_{max}^j(\Theta)) \right)}{\text{len}(s_i^k)} \quad (11)$$

Since  $\text{len}(s_{max}^j(\Theta)) \geq \text{len}(s_i^k)$ , (11) holds if  $\delta_i^k \leq \sum_{\bar{s}_j^h(\Theta) \in \nu_i^k} \lceil \frac{T_i}{T_j} \rceil$ .  $\sum_{\bar{s}_j^h(\Theta) \in \nu_i^k} \lceil \frac{T_i}{T_j} \rceil$  is the maximum number of transactions directly conflicting with  $s_i^k$  in all jobs with higher priority than  $p_i^x$ . Claim follows. ■

## B. FBLT vs. RCM

*Claim 5:* The schedulability of FBLT is equal to or better than RCM's if

$$\delta_i^k \leq \left( \sum_{\bar{s}_j^h(\Theta) \in \nu_i^k} \left( \lceil \frac{T_i}{T_j} \rceil + 1 \right) \right) - \sum_{u=1, s_{u,max} \in \epsilon}^{\min(n,m)-1} s_{u,max}$$

$\sum_{\bar{s}_j^h(\Theta) \in \nu_i^k} \left( \lceil \frac{T_i}{T_j} \rceil + 1 \right)$  is number of transactions directly conflicting with  $s_i^k$  in all jobs with higher priority than  $\tau_i$ .  $\sum_{u=1, s_{u,max} \in \epsilon}^{\min(n,m)-1} s_{u,max}$  is the sum of the maximum  $m - 1$  transactional lengths in all tasks

*Proof:* By substituting  $RC_A(T_i)$  and  $RC_B(T_i)$  in (5) with (2) and (6.9) in [9], respectively, we get:

$$\sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k \in s_i} \left( \delta_i^k \text{len}(s_i^k) + \sum_{s_{iz}^k \in \chi_i^k} \text{len}(s_{iz}^k) \right) + RC_{re}(T_i)}{T_i} \leq \sum_{\forall \tau_i} \frac{\left( \sum_{\forall \tau_j^* \in \gamma_i^{ex}} \sum_{\forall \theta \in \theta_i^{ex}} \left( \lceil \frac{T_i}{T_j} \rceil + 1 \right) \sum_{\forall \bar{s}_j^h(\theta)} \text{len}(s_j^h(\theta) + s_{max}^j(\theta)) \right)}{T_i} + \frac{RC_{re}(T_i)}{T_i} \quad (12)$$

where  $\tau_j^* = \{\tau_j : (\tau_j \neq \tau_i) \wedge (p_j > p_i)\}$ . Let  $\theta_i^{ex} = \theta_i + \theta_i^*$ , where  $\theta_i^*$  is the set of objects not directly accessed by any job of  $\tau_i$ , but can cause transactions in  $\tau_i$  to retry due to transitive retry. Let  $\gamma_i^{ex} = \gamma_i + \gamma_i^*$ , where  $\gamma_i^*$  is the set of tasks that access objects in  $\theta_i^*$ .  $s_j^h(\theta)$  can access multiple objects, so  $s_{max}^j(\theta)$  is the maximum length transaction conflicting with  $s_j^h(\theta)$ .  $s_j^h(\theta)$  is included only once for all  $\theta \in \Theta_j^h$ . Each  $\theta \in \theta_i^{ex}$  has its own  $s_{max}^j(\theta)$ . But  $s_i^h$  can access multiple objects, denoted as  $\Theta_j^h$ . So,  $s_{max}^j(\theta)$  is replaced by  $s_{max}^j(\Theta_j^h)$ , where  $s_{max}^j(\Theta_j^h) = \max\{s_{max}^j(\theta), \forall \theta \in \Theta_j^h\}$ .  $s_{max}^j(\Theta_j^h)$  is included once for each  $\theta \in \theta_i$ .

Each  $\tau_i^x$  has the same interference pattern from higher priority jobs,  $\tau_j^h$ , under FBLT and RCM. Hence,  $RC_{re}(T_i)$  for  $\tau_i^x$  is the same under FBLT and RCM. Consequently, (12) becomes:

$$\sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k \in s_i} \left( \delta_i^k \text{len}(s_i^k) + \sum_{s_{iz}^k \in \chi_i^k} \text{len}(s_{iz}^k) \right)}{T_i} \leq \sum_{\forall \tau_i} \frac{\sum_{\forall \tau_j^* \in \gamma_i^{ex}} \sum_{\forall \bar{s}_j^h(\Theta_j^h), \Theta_j^h \in \theta_i^{ex}} \left( \lceil \frac{T_i}{T_j} \rceil + 1 \right) \text{len}(s_j^h(\Theta_j^h) + s_{max}^j(\Theta_j^h))}{T_i} \quad (13)$$

Although different  $s_i^k$ s can have common conflicting transactions  $\bar{s}_j^h$ , no more than one  $s_i^k$  can be preceded by the same  $\bar{s}_j^h$  in the  $m\_set$ . This happens because transactions in the  $m\_set$  are non-preemptive. The original priority of transactions preceding  $s_i^k$  in the  $m\_set$  can be of lower or higher priority than the original priority of  $s_i^k$ . Under G-RMA,  $p_j > p_i$ , which means that  $T_j \leq T_i$ . Therefore,  $\lceil \frac{T_i}{T_j} \rceil \geq 1$ . For each  $s_i^k \in s_i$ , there are a set of zero or more  $\bar{s}_j^h(\Theta_j^h) \in \tau_j^*$  that are conflicting with  $s_i^k$ . Assuming this set of transactions conflicting with  $s_i^k$  is denoted as  $\nu_i^k = \left\{ \bar{s}_j^h(\Theta_j^h) \in \tau_j^* : (\Theta_j^h \in \theta_i^{ex}) \wedge \left( \bar{s}_j^h(\Theta_j^h) \notin \nu_i^l, l \neq k \right) \right\}$ .

The last condition  $\bar{s}_j^h(\Theta_j^h) \notin \nu_i^l, l \neq k$  in the definition of  $\nu_i^k$  ensures that common transactions  $\bar{s}_j^h$  that can conflict with

more than one transaction  $s_i^k \in \tau_i$  are split among different  $\nu_i^k$ ,  $k = 1, \dots, |s_i|$ . This condition is necessary, because in RCM, no two or more transactions of  $\tau_i^x$  can be aborted by the same transaction of  $\tau_j^h$ , where  $p_j^h > p_i^x$ . By substitution of  $\nu_i^k$  in (13), we get:

$$\leq \sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k \in s_i} \left( \frac{\delta_i^k \text{len}(s_i^k) + \sum_{s_{iz}^k \in \chi_i^k} \text{len}(s_{iz}^k)}{T_i} \right)}{\left( \sum_{\forall k=1}^{|s_i|} \sum_{s_j^h(\Theta_j^h) \in \nu_i^k} \left( \left( \left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \text{len}(s_j^h(\Theta_j^h) + s_{max}^j(\Theta_j^h)) \right) \right)} \quad (14)$$

$\bar{s}_j^h$  belongs to higher priority jobs than  $\tau_i$ .  $s_{max}^j$  belongs to higher priority jobs than  $\tau_i$  or  $\tau_i$  itself.  $s_{max}^j$  has a lower priority than  $\tau_j$ . Transactions in the  $m\_set$  can belong to jobs with original priority higher or lower than  $\tau_i$ . Thus, (14) holds if for each  $s_i^k \in \tau_i$ :

$$\delta_i^k \text{len}(s_i^k) \leq \left( \sum_{\bar{s}_j^h(\Theta_j^h) \in \nu_i^k} \left( \left( \left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \text{len}(\bar{s}_j^h(\Theta_j^h) + s_{max}^j(\Theta_j^h)) \right) \right) - \sum_{s_{iz}^k \in \chi_i^k} \text{len}(s_{iz}^k)$$

Then,

$$\delta_i^k \leq \left( \sum_{\bar{s}_j^h(\Theta_j^h) \in \nu_i^k} \left( \left( \left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \text{len} \left( \frac{\bar{s}_j^h(\Theta_j^h)}{s_i^k} + \frac{s_{max}^j(\Theta_j^h)}{s_i^k} \right) \right) \right) - \sum_{s_{iz}^k \in \chi_i^k} \text{len} \left( \frac{s_{iz}^k}{s_i^k} \right) \quad (15)$$

Let  $\epsilon = \{s_{u_{max}} : (1 \leq u \leq n) \wedge (s_{u1_{max}} \geq s_{u2_{max}}) \forall u1 < u2\}$ , where  $n$  is the number of tasks, and  $s_{u_{max}}$  is the maximum transactional length in any job of  $\tau_u$ . Thus,  $\epsilon$  is the set of maximum transactional lengths of all tasks in non-increasing order. Each  $s_{u_{max}} \in \epsilon$  belongs to a distinct task. Thus,  $\sum_{s_{iz}^k \in \chi_i^k} \text{len} \left( \frac{s_{iz}^k}{s_i^k} \right) \leq \sum_{u=1, s_{u_{max}} \in \epsilon}^{min(n,m)-1} s_{u_{max}}$ .  $\sum_{u=1, s_{u_{max}} \in \epsilon}^{min(n,m)-1} s_{u_{max}}$  is the sum of at most maximum  $m-1$  transactional lengths of all tasks.  $|\chi_i^k| \leq m-1$  and  $\text{len}(s_{max}^j(\Theta_j^h)) \geq \text{len}(s_i^k)$ . So, (15) holds if:

$$\delta_i^k \leq \left( \sum_{\bar{s}_j^h(\Theta_j^h) \in \nu_i^k} \left( \left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \right) - \sum_{u=1, s_{u_{max}} \in \epsilon}^{min(n,m)-1} s_{u_{max}} \quad (16)$$

To bound the effect of transitive retry, only objects that belong to  $\theta_i$  (not whole  $\theta_i^{ex}$ ) will be considered. Thus, RCM acts as if there were no transitive retry. Thus,  $\nu_i^k$  is modified to  $\nu_i^k = \{s_j^h(\Theta) \in \tau_j^* : (\Theta \in \Theta_j^h \cap \theta_i) \wedge (s_j^h(\Theta) \notin \nu_i^k, l \neq k)\}$ . Since  $\bar{\nu}_i^k \subseteq \nu_i^k$ , (16) still holds if  $\nu_i^k$  is replaced with  $\bar{\nu}_i^k$ .

Consequently, (16) holds if:

$$\delta_i^k \leq \left( \sum_{\bar{s}_j^h(\Theta) \in \bar{\nu}_i^k} \left( \left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \right) - \sum_{u=1, s_{u_{max}} \in \epsilon}^{min(n,m)-1} s_{u_{max}} \quad (17)$$

$\sum_{\bar{s}_j^h(\Theta) \in \bar{\nu}_i^k} \left( \left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right)$  represents the number of transactions directly conflicting with  $s_i^k$  in all jobs with higher priority than  $\tau_i$ . Claim follows. ■

#### C. FBLT vs. G-EDF/LCM

*Claim 6:* The schedulability of FBLT is equal to or better than G-EDF/LCM's when

$$\delta_i^k \leq \left( \sum_{\bar{s}_j^h(\Theta) \in \bar{\nu}_i^k} \left( \left\lceil \frac{T_i}{T_j} \right\rceil \alpha_{max}^{j\bar{h}} \right) \right)$$

$\alpha_{max}^{j\bar{h}}$  is the maximum  $\alpha$  with which  $s_j^h$  can conflict with the maximum length transaction sharing objects with  $s_i^k$  and  $s_j^h$ .

*Proof:* 4. The proof is similar to that of Claim 4 and is therefore skipped for brevity. It can be found in [21]. ■

#### D. FBLT vs. G-RMA/LCM

*Claim 7:* The schedulability of FBLT is equal to or better than G-RMA/LCM's when

$$\delta_i^k \leq \left( \sum_{\bar{s}_j^h(\Theta) \in \bar{\nu}_i^k} \left( \left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \alpha_{max}^{j\bar{h}} \right) - \sum_{u=1, s_{u_{max}} \in \epsilon}^{min(n,m)-1} s_{u_{max}}$$

$\left( \sum_{\bar{s}_j^h(\Theta) \in \bar{\nu}_i^k} \left( \left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \right)$  is the sum of the total number each transaction  $s_j^h$  can directly conflict with  $s_i^k$ .  $\alpha_{max}^{j\bar{h}}$  is the maximum  $\alpha$  with which  $s_j^h$  can conflict with the maximum length transaction sharing objects with  $s_i^k$  and  $s_j^h$ .

*Proof:* The proof is similar to that of Claim 5 and is therefore skipped for brevity. It can be found in [21]. ■

#### E. FBLT vs. PNF

*Claim 8:* Let  $\rho_i^j(k) = \left( \sum_{\forall s_j^h(\Theta) \in \nu_i^k(j)} \text{len}(s_j^h(\Theta)) \right) - s_{i_{max}}$ ,  $\tau_j \in \gamma_i^k$ .  $\rho_i^j(k)$  is the difference between the sum of transactional lengths of all transactions in  $\tau_j$  conflicting with  $s_i^k$ , and the maximum length transaction in  $\tau_i$ . Let  $\sum_{u=1, s_{u_{max}} \in \epsilon}^{min(n,m)-1} s_{u_{max}}$  be sum of at most maximum  $m-1$  transactional lengths in all tasks. Schedulability of FBLT is better or equal to PNF's when

$$\delta_i^k \leq \left( \sum_{\forall \tau_j \in \gamma_i^k} \left( \left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \text{len} \left( \frac{\left( \sum_{\forall s_j^h(\Theta) \in \nu_i^k(j)} \text{len}(s_j^h(\Theta)) \right)}{s_i^k} \right) \right) - \sum_{u=1, s_{u_{max}} \in \epsilon}^{min(n,m)-1} s_{u_{max}}$$



*Proof:* By substituting  $RC_A(T_i)$  and  $RC_B(T_i)$  in (5) with (2) and (6.1) in [9], respectively, we get:

$$\sum_{\forall \tau_i} \frac{\sum_{s_i^k \in s_i} (\delta_i^k \text{len}(s_i^k) + \sum_{s_{iz}^k \in \chi_i^k} \text{len}(s_{iz}^k)) + RC_{re}(T_i)}{T_i} \quad (18)$$

$$\leq \sum_{\forall \tau_i} \frac{\sum_{\tau_j \in \gamma_i} \sum_{\theta \in \theta_i} \left( \left( \left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \sum_{\forall s_j^h(\theta)} \text{len}(s_j^h(\theta)) \right)}{T_i}$$

$\bar{s}_j^h(\theta)$  can access multiple objects.  $\bar{s}_j^h(\theta)$  is included only once for all objects accessed by it.  $RC_{re}(T_i)$  is given by (6.8) in [9] in case of G-EDF, and (6.10) in [9] in case of G-RMA. Substituting  $RC_{re}(T_i)$  given by (6.8) and (6.10) in [9] with  $RC_{re}(T_i) = \sum_{\tau_j \in \gamma_i} \left( \left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) s_{imax}$ , we ensure correctness of (18). If  $\tau_j$  has no shared objects with  $\tau_i$ , then the release of any higher priority job  $\tau_j^y$  will not abort any transaction in any job of  $\tau_i$ . Thus, (18) holds if:

$$\sum_{\forall \tau_i} \frac{\sum_{s_i^k \in s_i} (\delta_i^k \text{len}(s_i^k) + \sum_{s_{iz}^k \in \chi_i^k} \text{len}(s_{iz}^k))}{T_i}$$

$$\leq \sum_{\forall \tau_i} \frac{\sum_{\tau_j \in \gamma_i} \left( \left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \left( \left( \sum_{\forall s_j^h(\theta), \theta \in \theta_i} \text{len}(s_j^h(\theta)) \right) - s_{imax} \right)}{T_i} \quad (19)$$

For each  $s_i^k \in s_i$ , there are a set of zero or more  $\bar{s}_j^h(\theta) \in \tau_j, \forall \tau_j \neq \tau_i$  that are conflicting with  $s_i^k$ . Assuming this set of transactions conflicting with  $s_i^k$  is denoted as  $\nu_i^k(j) = \{ \bar{s}_j^h(\theta) \in \tau_j : (\theta \in \theta_i) \wedge (\tau_j \neq \tau_i) \wedge (\bar{s}_j^h(\theta) \notin \nu_i^l, l \neq k) \}$ . The last condition  $\bar{s}_j^h(\theta) \notin \nu_i^l, l \neq k$  in the definition of  $\nu_i^k$  ensures that common transactions  $\bar{s}_j^h$  that can conflict with more than one transaction  $s_i^k \in \tau_i$  are split among different  $\nu_i^k(j), k = 1, \dots, |s_i|$ . This condition is necessary, because in PNF, no two or more transactions of  $\tau_i^x$  can be aborted by the same transaction of  $\tau_j^h$ .

Let  $\gamma_i^k$  be the subset of  $\gamma_i$  that contains tasks with transactions conflicting directly with  $s_i^k$ . By substitution of  $\nu_i^k$  and  $\gamma_i$  in (19) by  $\nu_i^k(j)$  and  $\gamma_i^k$ , (19) holds if for each  $s_i^k$ :

$$\delta_i^k + \sum_{s_{iz}^k \in \chi_i^k} \text{len}\left(\frac{s_{iz}^k}{s_i^k}\right) \leq \sum_{\tau_j \in \gamma_i^k} \left( \left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right)$$

$$\times \left( \left( \sum_{\forall s_j^h(\theta) \in \nu_i^k(j)} \text{len}\left(\frac{s_j^h(\theta)}{s_i^k}\right) \right) - \text{len}\left(\frac{s_{imax}}{s_i^k}\right) \right) \quad (20)$$

Non-preemptive transactions preceding  $s_i^k$  in the  $m\_set$  can directly or indirectly conflict with  $s_i^k$ . Under PNF, transactions can only directly conflict with  $s_i^k$ . Thus,  $s_{iz}^k$  on the left hand side of (20) is not necessarily included in  $\bar{s}_j^h(\theta)$  on the right hand side of (20). Let  $\epsilon = \{s_{imax} : (1 \leq u \leq n) \wedge (s_{u1max} \geq s_{u2max}, u1 < u2)\}$ , where  $n$  is the number of tasks, and  $s_{imax}$  is the maximum transactional length in any job of  $\tau_u$ . Thus,  $\epsilon$  is the set of maximum transactional lengths of all tasks in

non-increasing order. Each  $s_{imax} \in \epsilon$  belongs to a distinct task. Thus,  $\sum_{s_{iz}^k \in \chi_i^k} \text{len}\left(\frac{s_{iz}^k}{s_i^k}\right) \leq \sum_{u=1, s_{imax} \in \epsilon}^{min(n,m)-1} s_{imax}$ .  $\sum_{u=1, s_{imax} \in \epsilon}^{min(n,m)-1} s_{imax}$  is the sum of at most maximum  $m-1$  transactional lengths of all tasks.  $|\chi_i^k| \leq m-1$ . Then (20) holds if:

$$\delta_i^k \leq \left( \sum_{\tau_j \in \gamma_i^k} \left( \left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \right)$$

$$\times \text{len}\left( \frac{\left( \sum_{\forall s_j^h(\theta) \in \nu_i^k(j)} \text{len}(s_j^h(\theta)) \right) - s_{imax}}{s_i^k} \right)$$

$$- \sum_{u=1, s_{imax} \in \epsilon}^{min(n,m)-1} s_{imax}$$

Since  $\rho_i^j(k) = \left( \sum_{\forall s_j^h(\theta) \in \nu_i^k(j)} \text{len}(s_j^h(\theta)) \right) - s_{imax}, \tau_j \in \gamma_i^k$ , and  $\left( \left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right)$  is the total number of jobs of  $\tau_j$  interfering with  $\tau_i$ , claim follows. ■

#### F. FBLT vs. Lock-free

*Claim 9:* Under G-EDF and G-RMA, the schedulability of FBLT is equal or better than that under lock-free synchronization if  $s_{max} \leq r_{max}$ . If transactions execute in FIFO order (i.e.,  $\delta_i^k = 0, \forall s_i^k$ ) and contention is high,  $s_{max}$  can be much larger than  $r_{max}$ .

*Proof:* Lock-free synchronization [1], [20] allows only one object to be synchronized at a given time (e.g., a lock-free stack). Thus, for comparing FBLT's schedulability with lock-free, we limit the number of accessed objects per transaction under FBLT to one.

By substituting  $RC_A(T_i)$  and  $RC_B(T_i)$  in (5) with (2) and (6.17) in [9], respectively, we get:

$$\sum_{\forall \tau_i} \frac{\sum_{s_i^k \in s_i} (\delta_i^k \text{len}(s_i^k) + \sum_{s_{iz}^k \in \chi_i^k} \text{len}(s_{iz}^k)) + RC_{re}(T_i)}{T_i}$$

$$\leq \sum_{\forall \tau_i} \frac{\left( \sum_{\tau_j \in \gamma_i} \left( \left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \beta_{i,j} r_{max} \right) + RC_{re}(T_i)}{T_i} \quad (21)$$

where  $\beta_{i,j}$  is the number of retry loops of  $\tau_j$  that access the same objects as accessed by any retry loop of  $\tau_i$  [20] and  $r_{max}$  is the maximum execution cost of a single iteration of any retry loop of any task [20].

For G-EDF (G-RMA), any job  $\tau_i^x$  under FBLT has the same pattern of interference from higher priority jobs as ECM (RCM), respectively.  $RC_{re}(T_i)$  for ECM, RCM, and lock-free are given by Claims 25, 26, and 27 in [9], respectively.  $\therefore RC_{re}(T_i) = \left\lceil \frac{T_i}{T_j} \right\rceil s_{imax}, \forall \tau_j \neq \tau_i$  for G-EDF/FBLT and G-RMA/FBLT.  $RC_{re}(T_i) = \left\lceil \frac{T_i}{T_j} \right\rceil r_{imax}, \forall \tau_j \neq \tau_i$  for G-EDF/lock-free and G-RMA/lock-free.  $\therefore$  (21) becomes:

$$\sum_{\forall \tau_i} \frac{\sum_{s_i^k \in s_i} (\delta_i^k \text{len}(s_i^k) + \sum_{s_{iz}^k \in \chi_i^k} \text{len}(s_{iz}^k)) + \sum_{\tau_j \neq \tau_i} \left\lceil \frac{T_i}{T_j} \right\rceil s_{imax}}{T_i}$$

$$\leq \sum_{\forall \tau_i} \frac{\left( \sum_{\tau_j \in \gamma_i} \left( \left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \beta_{i,j} r_{max} \right) + \sum_{\tau_j \neq \tau_i} \left\lceil \frac{T_i}{T_j} \right\rceil r_{imax}}{T_i} \quad (22)$$

Since  $s_{max} \geq s_{i_{max}}$ ,  $len(s_i^k)$ ,  $len(s_{iz}^k)$ ,  $\forall i, z, k$  and  $r_{max} \geq r_{i_{max}}$   $\therefore$  (22) holds if:

$$\sum_{\forall \tau_i} \frac{\left( \left( \sum_{s_i^k \in s_i} (\delta_i^k + |\chi_i^k|) \right) + \sum_{\forall \tau_j \neq \tau_i} \left\lceil \frac{T_i}{T_j} \right\rceil \right) s_{max}}{T_i} \quad (23)$$

$$\leq \sum_{\forall \tau_i} \frac{\left( \left( \sum_{\forall \tau_j \in \gamma_i} \left( \left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \beta_{i,j} \right) + \sum_{\forall \tau_j \neq \tau_i} \left\lceil \frac{T_i}{T_j} \right\rceil \right) r_{max}}{T_i}$$

$\therefore$  (23) holds if for each  $\tau_i$ :

$$\left( \left( \sum_{s_i^k \in s_i} (\delta_i^k + |\chi_i^k|) \right) + \sum_{\forall \tau_j \neq \tau_i} \left\lceil \frac{T_i}{T_j} \right\rceil \right) s_{max} \quad (24)$$

$$\leq \left( \left( \sum_{\forall \tau_j \in \gamma_i} \left( \left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \beta_{i,j} \right) + \sum_{\forall \tau_j \neq \tau_i} \left\lceil \frac{T_i}{T_j} \right\rceil \right) r_{max}$$

$\therefore$

$$\frac{s_{max}}{r_{max}} \leq \frac{\left( \sum_{\forall \tau_j \in \gamma_i} \left( \left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \beta_{i,j} \right) + \sum_{\forall \tau_j \neq \tau_i} \left\lceil \frac{T_i}{T_j} \right\rceil}{\left( \sum_{s_i^k \in s_i} (\delta_i^k + |\chi_i^k|) \right) + \sum_{\forall \tau_j \neq \tau_i} \left\lceil \frac{T_i}{T_j} \right\rceil} \quad (25)$$

It appears from (25) that as  $\delta_i^k$  and  $|\chi_i^k|$  increases,  $s_{max}/r_{max}$  decreases. So, to get the lower bound on  $s_{max}/r_{max}$ , let  $\sum_{s_i^k \in s_i} (\delta_i^k + |\chi_i^k|)$  reach its maximum value. This maximum value is the total number of interfering transactions belonging to any job  $\tau_j$ ,  $j \neq i$ . The priority of  $\tau_j$  can be higher or lower than the current instance of  $\tau_i$ . Beyond this maximum value, there will be no more transactions that conflict with  $s_i^k$ . Thus, higher values for any  $\delta_i^k$  beyond the maximum value will be ineffective.  $\therefore \sum_{s_i^k \in s_i} (\delta_i^k + |\chi_i^k|) \leq \sum_{\forall \tau_j \in \gamma_i} \left( \left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right)$ . Consequently, (25) will be:

$$\frac{s_{max}}{r_{max}} \leq \frac{\left( \sum_{\forall \tau_j \in \gamma_i} \left( \left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \beta_{i,j} \right) + \sum_{\forall \tau_j \neq \tau_i} \left\lceil \frac{T_i}{T_j} \right\rceil}{\left( \sum_{\forall \tau_j \in \gamma_i} \left( \left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \right) + \sum_{\forall \tau_j \neq \tau_i} \left\lceil \frac{T_i}{T_j} \right\rceil} \quad (26)$$

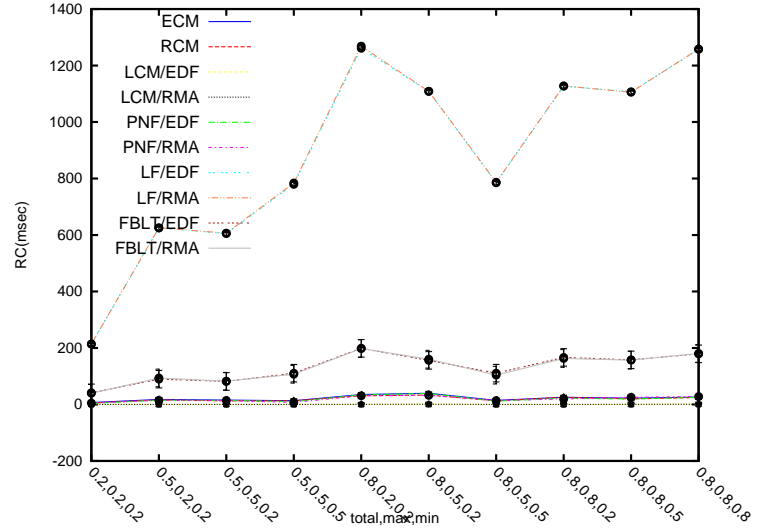
Since we are seeking the lower bound on  $\frac{s_{max}}{r_{max}}$ , let  $\beta_{i,j}$  assume its minimum value. Thus,  $\beta_{i,j} = 1$ .  $\therefore$  (26) holds if  $\frac{s_{max}}{r_{max}} \leq 1$ .

Let  $\delta_i^k(T_i) \rightarrow 0$  in (25). This means that transactions approximately execute according to their arrival order. Let  $\beta_{i,j} \rightarrow \infty$ ,  $\left\lceil \frac{T_i}{T_j} \right\rceil \rightarrow \infty$  in (25). This implies high contention. Consequently,  $\frac{s_{max}}{r_{max}} \rightarrow \infty$ . Hence, if transactions execute in FIFO order and contention is high,  $s_{max}$  can be much larger than  $r_{max}$ . Claim follows. ■

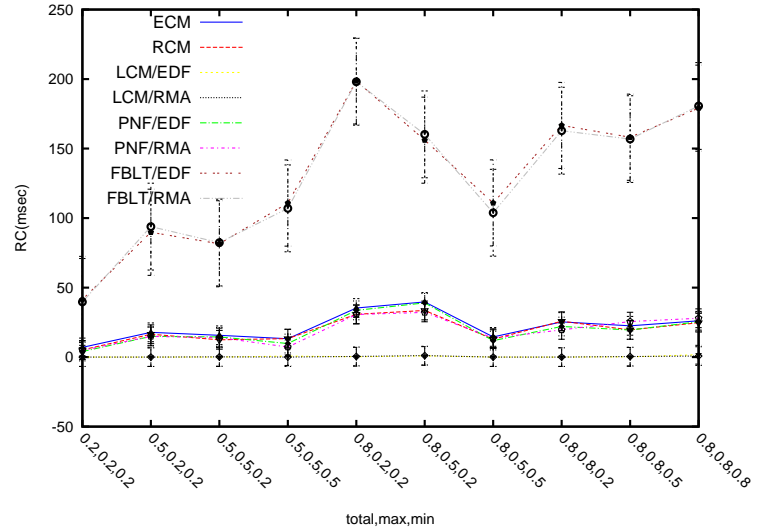
## VIII. EXPERIMENTAL EVALUATION

We now would like to understand how FBLT's retry cost compares with competitors in practice (i.e., on average). Since this can only be understood experimentally, we implement FBLT and the competitors and conduct experiments.

We used the ChronOS real-time Linux kernel [22] and the RSTM library [11] in our implementation. We implemented G-EDF and G-RMA schedulers in ChronOS, and modified RSTM to include implementations of FBLT, ECM, RCM, LCM, and PNF. For the retry-loop lock-free synchronization, we used a loop that reads an object and attempts to write to it using a CAS instruction. The task retries until the CAS



(a) ECM, RCM, LCM, PNF, FBLT, Lock-Free



(b) ECM, RCM, LCM, PNF, FBLT

Fig. 1. Average retry cost (one object/transaction).

succeeds. We used an 8 core, 2GHz AMD Opteron platform. The average time taken for one write operation by RSTM on any core is  $0.0129653375 \mu s$ , and the average time taken by one CAS-loop operation on any core is  $0.0292546250 \mu s$ .

We used four task sets consisting of 4, 5, 8, and 20 periodic tasks. Each task runs in its own thread and has a set of atomic sections. Atomic section properties are probabilistically controlled using three parameters: the maximum and minimum lengths of any atomic section within a task, and the total length of atomic sections within any task. Since lock-free synchronization cannot handle more than one object per atomic section, we first compare FBLT's retry cost with that of lock-free (and other CMs) for one object per transaction. We then compare FBLT's retry cost with that of other CMs for multiple objects per transaction.

Figure 1 shows the average retry cost for the 5 task set

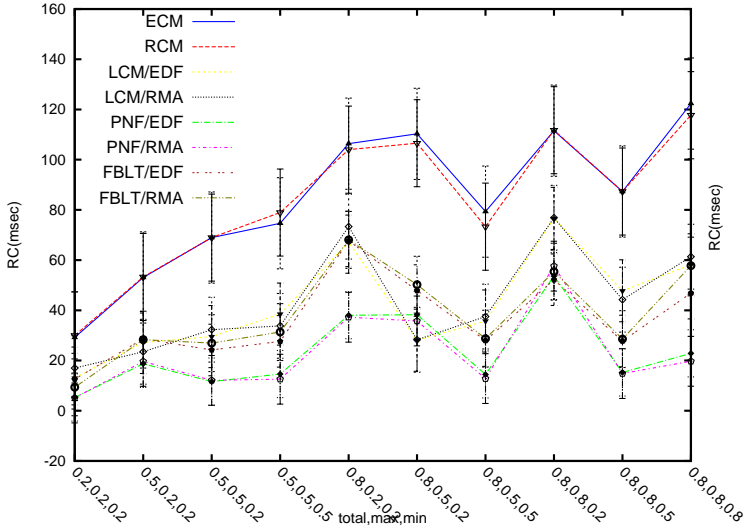


Fig. 2. Average retry cost (20 shared objects, 4 tasks).

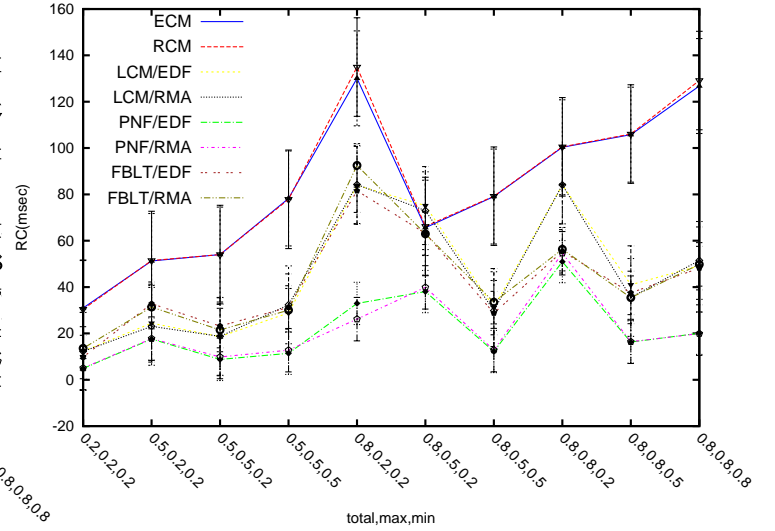


Fig. 3. Average retry cost (40 shared objects, 4 tasks).

sharing one object. On the x-axis of the figures, we record 3 parameters  $x$ ,  $y$ , and  $z$ .  $x$  is the ratio of the total length of all atomic sections of a task to the task WCET.  $y$  is the ratio of the maximum length of any atomic section of a task to the task WCET.  $z$  is the ratio of the minimum length of any atomic section of a task to the task WCET. The confidence level of all data points is 0.95.

While Figure 1(a) includes all synchronization methods, Figure 1(b) excludes lock-free. From these figures, we observe that lock-free has the largest retry cost, as it provides no conflict resolution. FBLT has the largest retry cost among CMs, because transactions share only one object in this case. For multiple objects per transaction, FBLT provides equal or shorter retry cost than LCM, as shown in Figures 2 and 3. PNF has an advantage over FBLT. However, PNF requires a-priori knowledge of all objects accessed by each transaction, whereas FBLT does not. Consequently, retry cost under PNF is shorter than that under FBLT. For 8 and 20 task sets, FBLT's retry cost is comparable to PNF's as shown in Figures 4 to 7. So, experiments show that FBLT's retry cost can be shorter than that under ECM, RCM, and LCM, and can be comparable to that of PNF's.

PNF was designed to avoid transitive retry. Previous experiments compares retry cost of different CMs in case of transitive retry. Figures 8 to 10 compare retry costs of different CMs in case of non-transitive retry. FBLT achieves shorter or comparable retry cost to other CMs including PNF.

## IX. CONCLUSIONS

Transitive retry increases transactional retry costs under ECM, RCM, and LCM. PNF avoids transitive retry by avoiding transactional preemptions. It avoids transitive retry cost by concurrently executing non-conflicting transactions, which are non-preemptive. However, PNF requires a-priori knowledge about objects accessed by each transaction. This is incompatible with dynamic STM implementations. Thus, we introduce

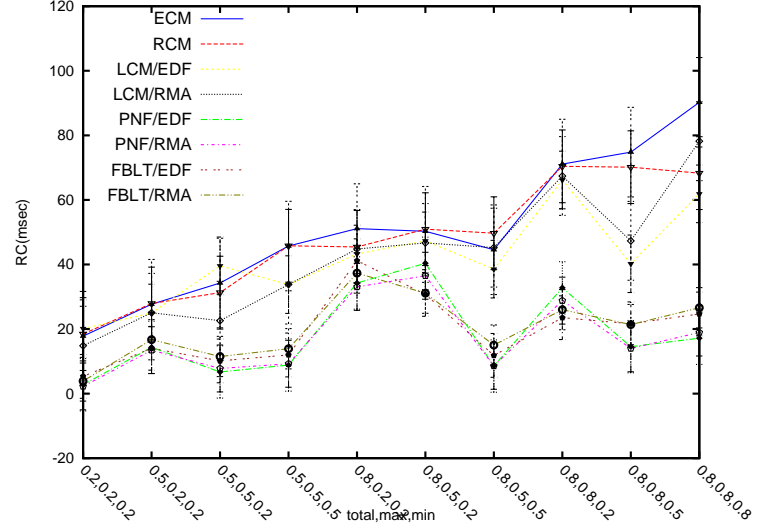


Fig. 4. Average retry cost (20 shared objects, 8 tasks).

the FBLT contention manager. Under FBLT, each transaction is allowed to abort for a no larger than a specified number of times. Afterwards, the transaction becomes non-preemptive. Non-preemptive transactions have higher priorities than other preemptive transactions and real-time jobs. Non-preemptive transactions resolve their conflicts using FIFO order.

By proper adjustment of the maximum abort number of each transaction, we showed that FBLT's schedulability is equal to or better than other synchronization techniques. For FBLT's schedulability to be equal to or better than lock-free synchronization, the upper bound on  $s_{max}/r_{max}$  must be 1. The upper bound on  $s_{max}/r_{max}$  can be higher than 1 if transactions execute in their arrival order and contention is high.

Our experimental results show that FBLT has equal or shorter retry cost than ECM, RCM, and LCM. PNF requires

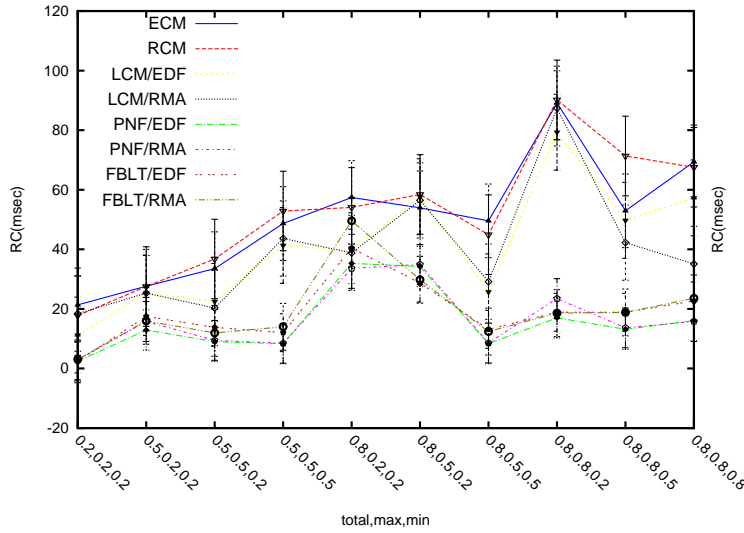


Fig. 5. Average retry cost (40 shared objects, 8 tasks).

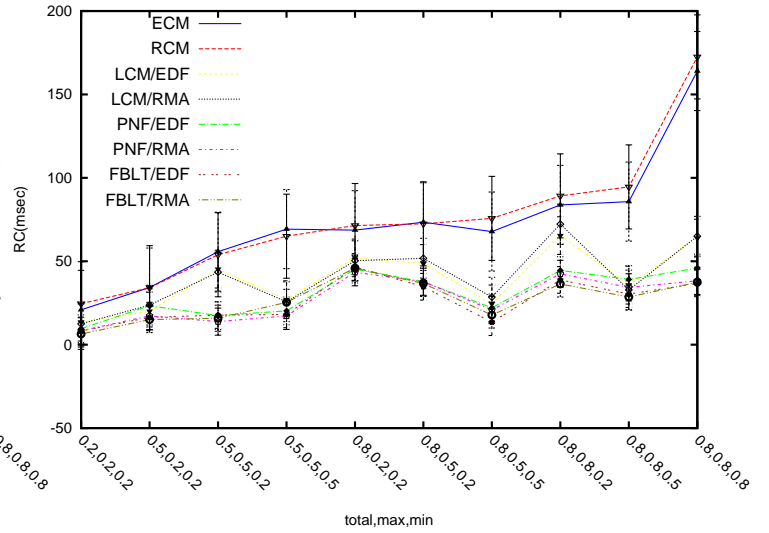


Fig. 7. Average retry cost (40 shared objects, 20 tasks).

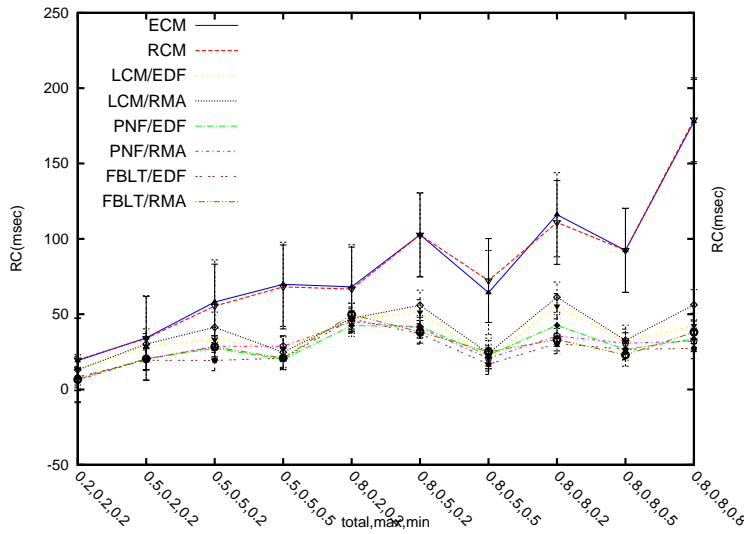


Fig. 6. Average retry cost (20 shared objects, 20 tasks).

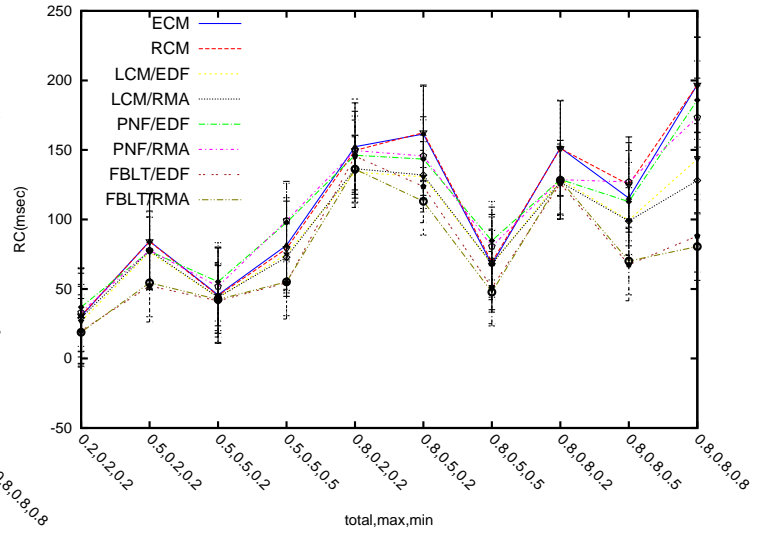


Fig. 8. Average retry cost (20 shared objects, 4 tasks).

a-priori knowledge of all objects accessed by each transaction. This is an advantage for PNF over FBLT. Consequently, retry cost under PNF is shorter than that under FBLT in case of transitive retry. Still, FBLT's retry cost can be comparable to PNF's. In case of no or low transitive retry, FBLT achieves shorter retry cost than other CMs including PNF.

Future work includes choosing another criterion to resolve conflicts of non-preemptive transactions. Also, using feedback from the system to adjust maximum abort number of each transaction. Consequently, retry cost can be reduced over time.

## REFERENCES

- [1] M. Herlihy, "The art of multiprocessor programming," in *PODC*, 2006, pp. 1–2.
- [2] R. Guerraoui, M. Herlihy, and B. Pochon, "Toward a theory of transactional contention managers," in *PODC*, 2005, pp. 258–264.
- [3] B. Saha, A.-R. Adl-Tabatabai *et al.*, "McRT-STM: a high performance software transactional memory system for a multi-core runtime," in *PPoPP*, 2006, pp. 187–197.
- [4] T. Harris, S. Marlow, S. P. Jones, and M. Herlihy, "Composable memory transactions," *Commun. ACM*, vol. 51, pp. 91–100, Aug 2008. [Online]. Available: <http://doi.acm.org/10.1145/1378704.1378725>
- [5] S. Fahmy and B. Ravindran, "On stm concurrency control for multicore embedded real-time software," in *International Conference on Embedded Computer Systems*, ser. SAMOS, July 2011, pp. 1–8.
- [6] M. El-Shambakey and B. Ravindran, "Stm concurrency control for multicore embedded real-time software: time bounds and tradeoffs," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, ser. SAC '12. New York, NY, USA: ACM, 2012, pp. 1602–1609. [Online]. Available: <http://doi.acm.org/10.1145/2231936.2232033>
- [7] —, "Stm concurrency control for embedded real-time software with tighter time bounds," in *Proceedings of the 49th Annual Design Automation Conference*, ser. DAC '12. New York, NY, USA: ACM, 2012, pp. 437–446. [Online]. Available: <http://doi.acm.org/10.1145/2228360.2228437>
- [8] R. I. Davis and A. Burns, "A survey of hard real-time scheduling for multiprocessor systems," *ACM Comput. Surv.*, vol. 43, no. 4, pp. 35:1–

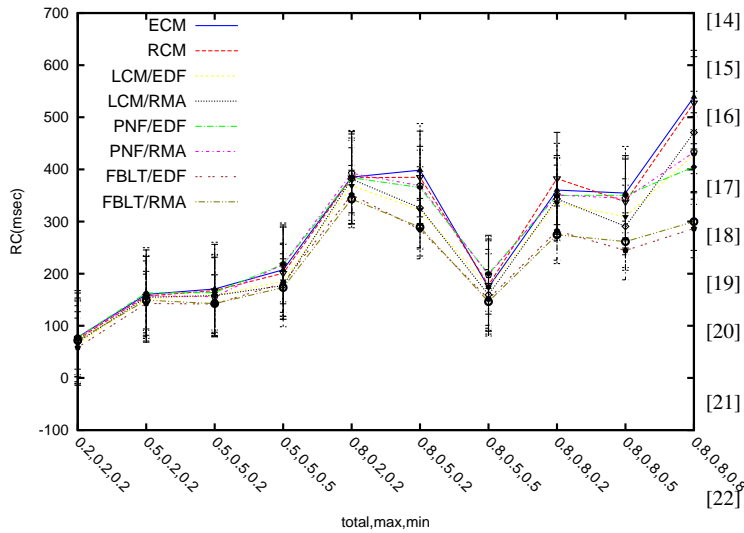


Fig. 9. Average retry cost (20 shared objects, 8 tasks).

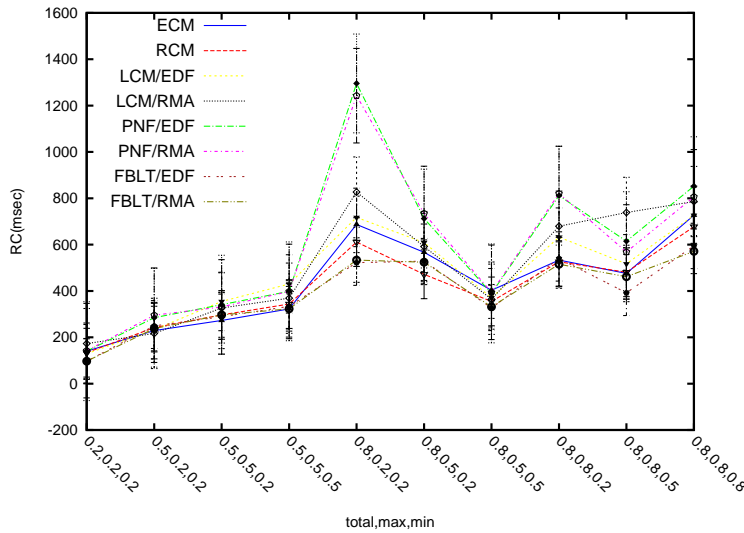


Fig. 10. Average retry cost (20 shared objects, 20 tasks).

- [14] S. F. Fahmy, "Collaborative scheduling and synchronization of distributable real-time threads," Ph.D. dissertation, Virginia Tech, 2010.
- [15] J. Manson, J. Baker *et al.*, "Preemptible atomic regions for real-time Java," in *RTSS*, 2006, pp. 10–71.
- [16] S. Fahmy, B. Ravindran, and E. D. Jensen, "On bounding response times under software transactional memory in distributed multiprocessor real-time systems," in *DATE*, 2009, pp. 688–693.
- [17] T. Sarni, A. Queudet, and P. Valduriez, "Real-time support for software transactional memory," in *RTCSA*, 2009, pp. 477–485.
- [18] M. Schoeberl, F. Brandner, and J. Vitek, "RTTM: Real-time transactional memory," in *ACM SAC*, 2010, pp. 326–333.
- [19] A. Barros and L. Pinho, "Managing contention of software transactional memory in real-time systems," in *IEEE RTSS, Work-In-Progress*, 2011.
- [20] U. Devi, H. Leontyev, and J. Anderson, "Efficient synchronization under global edf scheduling on multiprocessors," in *18th Euromicro Conference on Real-Time Systems*, 0-0 2006, pp. 10 pp. –84.
- [21] M. El-Shambakey and B. Ravindran, "On the design of real-time stm contention managers," ECE Department, Virginia Tech, Tech. Rep., 2011, available as <http://www.real-time.ece.vt.edu/tech-report-rt-stm-cm11.pdf>.
- [22] M. Dellinger, P. Garyali, and B. Ravindran, "Chronos linux: a best-effort real-time multiprocessor linux kernel," in *Proceedings of the 48th Design Automation Conference*, ser. DAC '11. New York, NY, USA: ACM, 2011, pp. 474–479.

35:44, Oct. 2011.

- [9] M. El-Shambakey, "Real-time software transactional memory: Contention managers, time bounds, and implementations," PhD Proposal, Virginia Tech, 2012.
- [10] M. Herlihy, V. Luchangco, M. Moir, and W. N. Scherer, III, "Software transactional memory for dynamic-sized data structures," in *Proceedings of the twenty-second annual symposium on Principles of distributed computing*, ser. PODC '03. New York, NY, USA: ACM, 2003, pp. 92–101. [Online]. Available: <http://doi.acm.org/10.1145/872035.872048>
- [11] V. Marathe, M. Spear, C. Heriot, A. Acharya, D. Eisenstat, W. Scherer III, and M. Scott, "Lowering the overhead of nonblocking software transactional memory," in *Workshop on Languages, Compilers, and Hardware Support for Transactional Computing*, ser. TRANSACT, 2006.
- [12] J. Anderson, S. Ramamurthy, and K. Jeffay, "Real-time computing with lock-free shared objects," in *RTSS*, 1995, pp. 28–37.
- [13] B. Brandenburg, J. Calandrino, A. Block, H. Leontyev, and J. Anderson, "Real-time synchronization on multiprocessors: To block or not to block, to suspend or spin?" in *Real-Time and Embedded Technology and Applications Symposium*, 2008. *RTAS '08. IEEE*, april 2008, pp. 342–353.