FBLT

Mohammed Elshambakey, Virginia Tech University
Binoy Ravindran, Virginia Tech University

NEEDS TO BE WRITTEN

Categories and Subject Descriptors: C.3 [**Special-Purpose and Application-based Systems**]: Real-time and embedded systems

General Terms: Design, Experimentation, Measurement

Additional Key Words and Phrases: Software transactional memory (STM), real-time contention manager

ACM Reference Format:
Elshambakey, M., Binoy, R. 2012. FBLT. ACM Trans. Embedd. Comput. Syst. , , Article ( ), 25 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

Lock-based concurrency control suffers from programmability, scalability, and composability challenges [Herlihy 2006]. These challenges are exacerbated in emerging multicore architectures, on which improved software performance must be achieved by exposing greater concurrency. Transactional memory (TM) is an alternative synchronization model for shared memory objects that promises to alleviate these difficulties. With TM, programmers organize code that read/write shared objects as transactions, which appear to execute atomically. Two transactions conflict if they access the same object and one access is a write. When that happens, a contention manager (or CM) resolves the conflict by aborting one and allowing the other to commit, yielding (the illusion of) atomicity. In addition to a simple programming model, TM provides performance comparable to highly concurrent fine-grained locking and lock-free approaches, and is composable. 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. See [Harris et al. 2010] for an excellent overview on TM.

Given STM's programmability, scalability, and composability advantages, we consider it for concurrency control in multicore real-time software. Doing so requires bounding transactional retries, as real-time threads, which subsume transactions, must satisfy time constraints. Retry bounds in STM are dependent on the CM policy at hand. Thus, real-time CM is logical.

Past research on real-time CM have proposed resolving transactional contention using dynamic and fixed priorities of parent threads, resulting in Earliest-Deadline-First-based CM (ECM) and Rate Monotonic Assignment-based CM (RCM), respec-

New Paper, Not an Extension of a Conference Paper.
Author's addresses: M. Elshambakey and B. Ravindran, ECE Department, Virginia Tech University, Blacksburg, VA 24060, USA.

DOI 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

tively [Fahmy et al. 2009b; 2009a; El-Shambakey and Ravindran 2012b]. These works show that, ECM and RCM, when used with the Global EDF (G-EDF) and Global RMA (G-RMA) multicore schedulers, respectively, achieve higher schedulability than lock-free synchronization techniques only under some ranges for the maximum atomic section length. This raises a fundamental question: is it possible to increase the atomic section length by an alternative CM design, so that STM's schedulability advantage has a larger coverage?

We answer this question by designing a novel CM that can be used with both dynamic and fixed priority (global) multicore real-time schedulers: length-based CM or LCM (Section **??**). LCM resolves conflicts based on the priority of conflicting jobs, besides the length of the interfering atomic section, and the length of the interfered atomic section. We establish LCM's retry and response time upper bounds, when used with G-EDF (Section **??**) and with G-RMA (Section **??**) schedulers. We identify the conditions under which G-EDF/LCM outperforms ECM (Section **??**) and lock-free synchronization (Section **??**), and G-RMA/LCM outperforms RCM (Section **??**). We implement LCM and competitor CM techniques in the Rochester STM framework [Marathe et al.] and conduct experimental studies (Section 8). Our study reveals that G-EDF/LCM and G-RMA/LCM have shorter or comparable retry costs and response times than competitors.

Thus, the paper's contribution is LCM with superior timeliness properties. This result 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.

2. 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., [Anderson et al. 1995]). 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 [Brandenburg et al. 2008]. In contrast, STM is semantically simpler [Herlihy 2006], and is often the only viable lock-free solution for complex data structures (e.g., red/black tree) [Fahmy 2010] and nested critical sections [Saha et al. 2006].

STM concurrency control for real-time systems has been previously studied in [Manson et al. 2006; Fahmy et al. 2009b; Sarni et al. 2009; Schoeberl et al. 2010; Fahmy 2010; Barros and Pinho 2011; El-Shambakey and Ravindran 2012b; 2012a; El-Shambakey 2012].

[Manson et al. 2006] proposes a restricted version of STM for uniprocessors. Uniprocessors do not need contention management. [Fahmy et al. 2009b] 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.

[Sarni et al. 2009] 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. [Schoeberl et al. 2010] 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.

[Fahmy 2010] upper bounds retries and response times for ECM with G-EDF, and identify the tradeoffs with locking and lock-free protocols. Similar to [Schoeberl et al. 2010], [Fahmy 2010] also assumes that the worst case conflict between atomic sections occurs when the sections are released simultaneously. The ideas in [Fahmy 2010] are extended in [Barros and Pinho 2011], which presents three real-time CM designs. But no retry bounds or schedulability analysis techniques are presented for those CMs.

[El-Shambakey and Ravindran 2012b] 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, [El-Shambakey and Ravindran 2012b] shows that, STM's superiority holds only under some ranges for the maximum atomic section length. Moreover, [El-Shambakey and Ravindran 2012b] restricts transactions to access only one object. [El-Shambakey and Ravindran 2012a] presents length-based CM (LCM), and upper bounds transactional retries and response time for G-EDF/LCM and G-RMA/LCM. [El-Shambakey and Ravindran 2012a] compares (analytically and experimentally) between ECM, RCM and LCM, as well as lock-free and LCM. [El-Shambakey and Ravindran 2012a], as [El-Shambakey and Ravindran 2012b], restricts transactions to access only one object.

[El-Shambakey 2012] presents Priority CM with Negative value and First Access (PNF). PNF was designed to avoid transitive retry effect when each transaction accesses multiple objects. PNF also optimizes processor usage by reducing priority of retrying transactions below priorities of any real-time task. PNF requires prior knowledge of accessed objects by each transaction which is not always available. Besides, PNF is a centralized CM that uses locks in its implementation. Accordingly, reduction in retry cost is wasted by high overhead. [El-Shambakey 2012] upper bounds transactional retries and response time for G-EDF and G-RMA. [El-Shambakey 2012] compares (analytically and experimentally) between PNF and ECM, RCM, LCM and lock-free.

Our work builds upon [El-Shambakey and Ravindran 2012b; 2012a; El-Shambakey 2012], allows multiple objects per transaction with no prior knowledge about these objects. We upper bound transactional retries and task response times. We identify the conditinos for better schedulability for FBLT than other synchronization techniques.

3. PRELIMINARIES

We consider a multiprocessor system with m identical processors and n sporadic tasks $\tau_1, \tau_2, \ldots, \tau_n$. The k^{th} instance (or job) of a task τ_i is denoted τ_i^k. Each task τ_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 τ_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 τ_i's (fixed) priority and it is constant for all instances of τ_i. Under a dynamic priority scheduler such as G-EDF, a job τ_i^j's priority, p_i^j, differs from one instance to another. A task τ_j may interfere with task τ_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 τ_i is denoted s_i. s_i^k is the k^{th} atomic section of τ_i. Each object, θ, can be accessed by multiple tasks. The set of distinct objects accessed by τ_i is θ_i without repeating objects. The set of atomic sections used by τ_i to access θ 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 τ_i that accesses θ. s_i^k can access one or more objects in θ_i. So, s_i^k refers to the transaction itself regardless of accessed objects by this transaction. We denote the set of all accessed objects by s_i^k as Θ_i^k. While $s_i^k(\theta)$ means that s_i^k accesses a specific object $\theta \in \Theta_i^k$, $s_i^k(\Theta)$ means that s_i^k accesses a set of objects $\Theta = \{\theta \in \Theta_i^k\}$. $\bar{s_i^k} = \bar{s_i^k}(\Theta)$ refers only once to s_i^k regardless of number of objects in Θ. So, $|\bar{s_i^k}(\Theta)|_{\forall \theta \in \Theta} = 1$. $s_i^k(\theta)$ 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 θ with τ_i is denoted $\gamma_i(\theta)$.

Atomic sections are non-nested (supporting nested STM is future work). The maximum-length atomic section in τ_i that accesses θ 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 τ_i is $s_{max}^i(\theta)$.

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 4.3), an atomic section $s_i^k(\Theta_i^k)$ may retry due to another atomic section $s_j^l(\Theta_j^l)$ where $\Theta_i^k \cap \Theta_j^l = \emptyset$. θ_i^* denotes the set of objects not accessed directly by atomic sections in τ_i but can cause transactions in τ_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 τ_i to retry directly or through transitive retry. γ_i^* is the set of tasks that access objects in θ_i^*. $\gamma_i^{ex}(= \gamma_i + \gamma_i^*)$ is the set of all tasks that can can directly or indirectly (through transitive retry) cause transactions in τ_i to abort and retry. The total time that a task τ_i's atomic sections have to retry over T_i is denoted $RC(T_i)$. The additional amount of time that all interfering jobs of τ_j causes to response time of any job of τ_i during L, without considering retries due to atomic sections, is denoted $W_{ij}(L)$.

4. MOTIVATION

To understand the need for FBLT, we give a brief introduction about previous CMs.

4.1. ECM and RCM

Earliest Deadline Contention Manager (ECM) [El-Shambakey and Ravindran 2012b] is used with G-EDF. ECM allows the transaction with the shortest absolute deadline to commit first. The others retry and abort. RCM [El-Shambakey and Ravindran 2012b] is used with G-RMA. *Rate Monotonic Contention Manager (RCM)* allows the transaction with the shortest period to commit first. ECM and RCM maintains consistency with the underlying scheduler.

4.2. 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 τ_j^b than τ_i^a — conflicts with $s_i^k(\theta)$ at the end of its execution, while $s_i^k(\theta)$ is just about to commit. Thus, *Length-based Contention Manager (LCM)* [El-Shambakey and Ravindran 2012a], 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 was greater than p_j^l, then $s_i^k(\theta)$ would be the one that commits, 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

$s_i^k(\theta)$. [El-Shambakey and Ravindran 2012a] assumes that:

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

where $c_{ij}^{kl} \in]0, \infty[$, to cover all possible lengths of $s_j^l(\theta)$. The 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 α_{ij}^{kl} (step 5), where α_{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: LCM

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

4.3. PNF

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

Example 1. Consider three atomic sections s_1^x, s_2^y, and s_3^z belonging to jobs τ_1^x,τ_2^y, and τ_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, s_2^y and s_3^z share objects. s_1^x and s_3^z do not share objects. 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 may retry transitively because of s_3^z, which will increase the retry cost of s_1^x.

Assume another atomic section s_4^f is introduced. Priority of s_4^f is higher than priority of s_3^z. s_4^f shares objects only with s_3^z. Thus, s_4^f can make s_3^z to retry, which in turn will make 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 tasks of higher priorities are added, where each task shares objects with its immediate lower priority task. τ_3^z may have atomic sections that share objects with τ_1^x, but this will not prevent the effect of transitive retry due to s_1^x.

Therefore, the analysis in [El-Shambakey and Ravindran 2012b] and [El-Shambakey and Ravindran 2012a] must extend the set of objects that can cause an atomic section of a lower priority job to retry. This can be done by initializing the set of conflicting objects, γ_i, to all objects accessed by all transactions of τ_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 γ_i adds all other objects accessed by s_j^l to γ_i. The loop over all higher priority tasks is repeated, each time with the new γ_i, until there are no more transactions accessing any object in γ_i. The final set of objects(tasks) that can cause transactions in τ_i to retry is $\theta_i^{ex}(\gamma_i^{ex})$ respectively[1]. *Priority contention manager with Negative value and First access (PNF)* [El-Shambakey 2012] is designed to avoid transitive retry. ECM, RCM and LCM suffer from transitive retry. PNF avoids 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.

The problem with PNF is to know a priori each object accessed by each transaction. This is not suitable with dynamic STM [Herlihy et al. 2003]. Besides, current implementation of PNF is a centralized CM that uses locks. This increases overhead and wastes reduction in retry cost. PNF is most suitable in case of heavily transitive retry among transactions. In other cases, LCM is preferred.

4.4. The need for FBLT

It is required to have a CM that combines benefits of previous CMs. This CM should:

(1) reduce 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 effect of transitive retry just as PNF without prior knowledge of accessed objects by each transactions. Thus, it maintains semantics of dynamic STM.
(3) be decentralized and do not use locks. So, overhead is reduced.

FBLT acheives these goals by bounding abortion number of each transaction s_i^k due to other transactions to at most δ_i^k. ${}_i^k$ includes abort numbers due to direct conflict with other transactions, as well as transitive retry (goal 2). If a transaction s_i^k reaches its δ_i^k, it is added to an m_set in FIFO order. In m_set, s_i^k executes non-preemptively. If transactions in m_set conflict together, they use their FIFO order in the m_set to resolve conflict. s_i^k can still abort after it becomes a non-preemptive transaction due to other non-preemptive transactions. The number of abort times for any non-preemptive transactions is bounded by $m-1$, where m is number of processors, as will be shown in Section 6. So, the key idea is to use the suitable δ_i^k for each s_i^k before it becomes a

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

non-preemptive transaction. The choice of δ_i^k should make the total retry cost (thus, schedulability) of any job τ_i^x under FBLT comparable to the retry cost under ECM, RCM, LCM and PNF. Section 7 shows the suitable δ_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 crossed (goal 1). Each non-preemptive transaction, s_i^k, uses the time it joined m_set to resolve conflicts with other non-preemptive transactions. So, FBLT does not have to use locks and it is decentralized (goal 3).

5. FBLT

ALGORITHM 2: FBLT

Data: s_i^k: interfered transaction;
s_j^l: interfering transactions;
δ_i^k: the maximum number of times s_i^k can be aborted during T_i;
η_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 η_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 η_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**

Algorithm 2 shows behaviour of FBLT. Each transaction s_i^k can be aborted during T_i for at most δ_i^k times. η_i^k records number of times s_i^k has already been aborted up to now. If s_i^k and s_j^l have not joined $m_$set yet, then they are preemptive transactions. Preemptive transactions resolve conflicts using Algorithm 1 (step 2). So, FBLT defaults to LCM when no transaction reaches its δ. If only one of the transactions is in $m_$set, then non-preemptive transaction (the one in $m_$set) aborts the other one (steps 15 to 26). η_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 $m_$set and its priority is increased to m_prio (steps 7 to 9 and 20 to 22). When 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. $m_$set can hold at most m concurrent transactions because there are m processors in the system. $r(s_i^k)$ records time s_i^k joined $m_$set (steps 8 and 21). When non-preemptive transactions conflict together (step 27), the transaction with smaller $r()$ commits first (steps 29 and 31). Thus, non-preemptive transactions are executed in FIFO order of $m_$set.

5.1. Illustrative Example

FBLT's behaviour is explained by 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 favour of s_a^b and η_i^k is incremented to 1.
(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 favour of $s_c^d(\theta_2)$. η_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 favour of $s_e^f(\theta_2, \theta_3)$ and η_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 favour of $s_j^l(\theta_3)$ and η_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. $\eta_i^k = \delta_i^k$, $s_i^k(\theta_1, \theta_2)$ 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, $s_e^f(\theta_2, \theta_3)$ is aborted in favour of $s_i^k(\theta_1, \theta_2)$ despite p_e^f is greater than original priority of $s_i^k(\theta_1, \theta_2)$. η_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 $m_$set. So, $s_i^k(\theta_1, \theta_2)$ commits first, then $s_g^h(\theta_2, \theta_3)$.
(9) $s_g^h(\theta_3)$ will continue to abort and retry in favour 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 $m_$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 $m_$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. Whatever the reason of conflict, once a transaction has reached its maximum allowed δ, the transaction becomes a non-preemptive one (steps 6 and 7). Non-preemptive transactions has higher priority than other preemptive transactions (steps 6 and 7). Non-preemptive transactions execute in their arrival order to $m_$set.

6. RETRY COST AND RESPONSE TIME UNDER FBLT

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

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

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

where χ_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 τ_j. $RC_{re}(L)$ is the retry cost resulting from release of higher piority jobs which preempt τ_i^x. $RC_{re}(L)$ is calculated by (6.8) in [El-Shambakey 2012] for G-EDF, and (6.10) in [El-Shambakey 2012] for G-RMA.

PROOF.
By definition of FBLT, $s_i^k \in \tau_i^x$ can be aborted at maximum δ_i^k times before s_i^k joins m_set. Before joining m_set, s_i^k can be aborted due to higher priority transactions, or transactions in the m_set. Original priority of transactions in m_set can be of higher or lower priority than p_i^x. Thus, the maximum time s_i^k is aborted before joining m_set occurs if s_i^k is aborted for δ_i^k times. Transactions preceding s_i^k in m_set can conflict directly with s_i^k, or indirectly through transitive retry. The worst case scenario for s_i^k after joining m_set occurs if s_i^k is preceded by $m-1$ maximum length conflicting transactions. Hence, in worst case, s_i^k has to wait for the previous $m-1$ transactions to commit first. Priority of s_i^k after joining m_set is higher than any real-time job. So, s_i^k is not aborted by any job. If s_i^k has not joined m_set yet, and a higher priority job τ_j^y is released while s_i^k is running, then s_i^k may be aborted if τ_j^y has conflicting transactions with s_i^k. τ_j^y causes only one abort in τ_i^x because τ_j^y preempts τ_i^x only once. If s_i^k has already joined m_set, then s_i^k cannot be aborted by release of higher priority jobs. So, the maximum number of abort times to transactions in τ_i^x due to release of higher priority jobs is less or equal to number of interfering higher priority jobs to τ_i^x. Claim follows.

□

CLAIM 2.
The blocking time for a job τ_i^x due to lower priority jobs is upper bounded by:

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

where $s_{j_{max}}$ is the maximum length transaction in any job τ_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 τ_i^x.

PROOF.
τ_i^x is blocked when it is initially released and all processors are busy with lower priority jobs with non-preemptive transactions. Although τ_i^x can be preempted by higher

priority jobs, τ_i^x cannot be blocked after it is released. If τ_i^x is preempted by a higher priority job τ_j^y, then τ_j^y finishes execution, the underlying scheduler will not choose a lower priority job than τ_i^x before τ_i^x. So, after τ_i^x is released, there is no chance for any transaction s_u^v belonging to a lower priority job than τ_i^x to run before τ_i^x. Thus, s_u^v cannot join m_set before τ_i^x finishes. Consequently, the worst case blocking time for τ_i^x occurs when the maximum length m transactions in lower priority jobs than τ_i^x are executing non-preemptively. After the minimum length transaction in the m_set finishes, the underlying scheduler will choose τ_i^x or a higher priority job to run. Claim follows.

□

CLAIM 3.
Response time of any job τ_i^x during an interval $L \leq T_i$ under FBLT is upper bounded by

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

where $RC_{to}(L)$ is calculated by (2), $D(\tau_i^x)$ is calculated by by (3), and $W_{ij}(R_i^{up})$ is calculated by (11) in [El-Shambakey and Ravindran 2012b] for G-EDF, and (17) in [El-Shambakey and Ravindran 2012b] for G-RMA. (11) and (17) in [El-Shambakey and Ravindran 2012b] inflates c_j of any job $\tau_j^y \neq \tau_i^x$, $p_j^y > p_i^x$ by retry cost of transactions in τ_j^y.

PROOF.
Response time of any job τ_i^x is calculated directly from FBLT's behaviour. Response time of any job τ_i^x is the sum of its worst case execution time c_i, plus retry cost of transactions in τ_i^x ($RC_{to}(L)$), plus blocking time of τ_i^x ($D(\tau_i^x)$), and the workload interference of higher priority jobs. Workload interfernence of higher priority jobs scheduled by G-EDF is calculated by (11) in [El-Shambakey and Ravindran 2012b], and by (17) in [El-Shambakey and Ravindran 2012b] for G-RMA. Claim follows.

□

7. FBLT VS. OTHER SYNCHRONIZATION TECHNIQUES

We now (formally) compare the schedulability of G-EDF (G-RMA) with FBLT against ECM, RCM, LCM, PNF and lock-free synchronization [El-Shambakey and Ravindran 2012b; 2012a; Devi et al. 2006; El-Shambakey 2012]. Such a comparison will reveal when FBLT outperforms others. Toward this, we compare the total utilization under G-EDF (G-RMA)/FBLT, with that under the other synchronization methods. Inflated execution time of each method, which is the sum of the worst-case execution time of the task and its retry cost, is used in the utilization calculation of each task.

No processor is available for τ_i^x during the blocking time. As each processor is busy with some job other than τ_i^x, $D(\tau_i^x)$ is not added to the inflated execution time of τ_i^x. Hence, $D(\tau_i^x)$ is not added to the utilization calculation of τ_i^x.

Let $RC_A(T_i)$ denote the retry cost of any τ_i^x using the synchronization method A during T_i. Let $RC_B(T_i)$ denote the retry cost of any τ_i^x using synchronization method B during T_i. Then, schedulability of A is comparable to B if

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

$$\sum_{\forall \tau_i} \frac{RC_A(T_i)}{T_i} \leq \sum_{\forall \tau_i} \frac{RC_B(T_i)}{T_i} \tag{5}$$

7.1. FBLT vs. ECM

CLAIM 4.
Schedulability of FBLT is equal or better to ECM's when maximum abort number of any preemptive transaction s_i^k is less or equal to number of direct conflicting transactions with s_i^k in all other jobs with higher priority than priority of current job of τ_i.

PROOF.
By substituing $RC_A(T_i)$ and $RC_B(T_i)$ in (5) with (2) and (6.7) in [El-Shambakey 2012] respectively.

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

Let $\theta_i^{ex} = \theta_i + \theta_i^*$ where θ_i^* is the set of objects not accessed directly by τ_i but can enforce transactions in τ_i to retry due to transitive retry. Let $\gamma_i^{ex} = \gamma_i + \gamma_i^*$ where γ_i^* is the set of tasks that access objects in θ_i^*. $\bar{s_j^h}(\theta)$ can access multiple objects, so $s_{max}^j(\theta)$ is the maximum length transaction conflicting with $\bar{s_j^h}(\theta)$. $\bar{s_j^h}(\theta)$ 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 Θ_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 τ_i^x has the same interference pattern from higher priority jobs, τ_j^h, under FBLT and ECM. Hence, $RC_{re}(T_i)$ for τ_i^x is the same under FBLT and ECM. Consequently, (6) becomes

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

Although different s_i^k can have common conflicting transactions $\bar{s_j^h}$, but 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. Original priority of transactions preceding s_i^k in the m_set can be of lower or higher priority than original priority of s_i^k. Under G-EDF, τ_j can have at least one job of higher priority than τ_i^x, then $\left\lceil \frac{T_i}{T_j} \right\rceil \geq 1$. So, each one of the s_{iz}^k in the left hand side of (7) is included in one of the $\bar{s_j^h}(\theta)$ in the right hand side of (7). Then (7) holds if

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

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

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

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 conflicting transactions with s_i^k is denoted as $\eta_i^k = \left\{ \bar{s_j^h}(\Theta) \in \tau_j : \left(\Theta \in \theta_i \cap \Theta_j^h\right)\right) \wedge (\forall \tau_j \neq \tau_i) \wedge \left(\bar{s_j^h}(\Theta) \notin \eta_i^l,\, l \neq k \right) \right\}$. The last condition $\bar{s_j^h}(\Theta) \notin \eta_i^l,\, l \neq k$ in definition of η_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 η_i^k, $k = 1, .., |s_i|$. This condition is necessary because in ECM, no two or more transactions of τ_i^x can be aborted by the same transaction of τ_j^h where $p_j^h > p_i^x$. By substitution of η_i^k in (9), we get

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

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

$$\delta_i^k \leq \frac{\sum_{\bar{s_j^h}(\Theta) \in \eta_i^k} \left(\left\lceil \frac{T_i}{T_j} \right\rceil len\left(s_{max}^j(\Theta)\right) \right)}{len(s_i^k)} \tag{11}$$

As $len\left(s_{max}^j(\Theta)\right) \geq len(s_i^k)$, then (11) holds if $\delta_i^k \leq \sum_{\bar{s_j^h}(\Theta) \in \eta_i^k} \left\lceil \frac{T_i}{T_j} \right\rceil$

$\sum_{\bar{s_j^h}(\Theta) \in \eta_i^k} \left\lceil \frac{T_i}{T_j} \right\rceil$ is the maximum number of directly conflicting transactions with s_i^k in all jobs with higher priority than ^x_i. Claim follows.

□

7.2. FBLT vs. RCM

CLAIM 5.
Schedulability of FBLT is equal or better to RCM's when maximum abort number of any preemptive transaction s_i^k is less or equal to number of directly conflicting transactions with s_i^k in all jobs with higher priority than τ_i minus sum of maximum $m-1$ transactional lengths in all tasks.

PROOF.
By substituing $RC_A(T_i)$ and $RC_B(T_i)$ in (5) with (2) and (6.9) in [El-Shambakey 2012] respectively.

$$\sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k \in s_i} \left(\delta_i^k len(s_i^k) + \sum_{s_{iz}^k \in \chi_i^k} len(s_{iz}^k) \right) + RC_{re}(T_i)}{T_i} \tag{12}$$

$$\leq \sum_{\forall \tau_i} \frac{\left(\sum_{\forall \tau_j^* \in \gamma_i^{ex}} \sum_{\forall \theta \in \theta_i^{ex}} \left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1\right) \sum_{\forall s_j^{\bar{h}}(\theta)} len\left(s_j^{\bar{h}}(\theta) + s_{max}^j(\theta)\right)\right) + RC_{re}(T_i)}{T_i}$$

where $\tau_j^* = \{\tau_j : (\tau_j \neq \tau_i) \wedge (p_j > p_i)\}$. Let $\theta_i^{ex} = \theta_i + \theta_i^*$ where θ_i^* is the set of objects not accessed directly by τ_i but can enforce transactions in τ_i to retry due to transitive retry. Let $\gamma_i^{ex} = \gamma_i + \gamma_i^*$ where γ_i^* is the set of tasks that access objects in θ_i^*. $s_j^{\bar{h}}(\theta)$ can access multiple objects, so $s_{max}^j(\theta)$ is the maximum length transaction conflicting with $s_j^{\bar{h}}(\theta)$. $s_j^{\bar{h}}(\theta)$ 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 Θ_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 τ_i^x has the same interference pattern from higher priority jobs, τ_j^h, under FBLT and RCM. Hence, $RC_{re}(T_i)$ for τ_i^x is the same under FBLT and RCM. Consequently, (12) becomes

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

Although different s_i^k can have common conflicting transactions $s_j^{\bar{h}}$, but no more than one s_i^k can be preceded by the same $s_j^{\bar{h}}$ in the $m_$set. This happens because transactions in the $m_$set are non-preemptive. Original priority of transactions preceding s_i^k in the $m_$set can be of lower or higher priority than original priority of s_i^k. Under G-RMA, $p_j > p_i$ means that $T_j \leq T_i$, then $\left\lceil \frac{T_i}{T_j} \right\rceil \geq 1$. For each $s_i^k \in s_i$, there are a set of zero or more $s_j^{\bar{h}}(\Theta_j^h) \in \tau_j^*$ that are conflicting with s_i^k. Assuming this set of conflicting transactions with s_i^k is denoted as $\eta_i^k = \left\{s_j^{\bar{h}}(\Theta_j^h) \in \tau_j^* : (\Theta_j^h \in \theta_i^{ex}) \wedge \left(s_j^{\bar{h}}(\Theta_j^h) \notin \eta_i^l,\, l \neq k\right)\right\}$. The last condition $s_j^{\bar{h}}(\theta) \notin \eta_i^l,\, l \neq k$ in definition of η_i^k ensures that common transactions $s_j^{\bar{h}}$ that can conflict with more than one transaction $s_i^k \in \tau_i$ are split among different η_i^k, $k = 1, .., |s_i|$. This condition is necessary because in RCM, no two or more transactions of τ_i^x can be aborted by the same transaction of τ_j^h where $p_j^h > p_i^x$. By substitution of η_i^k in (13), we get

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

$s_j^{\bar{h}}$ belongs to higher priority jobs than τ_i. s_{max}^j belongs to higher priority jobs than τ_i or τ_i itself. s_{max}^j has a lower priority than τ_j. Transactions in $m_$set can belong to jobs with original priority higher or lower than τ_i. So, (14) holds if for each $s_i^k \in \tau_i$

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

Then

$$\delta_i^k \le \left(\sum_{s_j^{\bar{h}}(\Theta_j^h) \in \eta_i^k} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) len \left(\frac{s_j^{\bar{h}}(\Theta_j^h) + s_{max}^j(\Theta_j^h)}{s_i^k} \right) \right) \right) - \sum_{s_{iz}^k \in \chi_i^k} len \left(\frac{s_{iz}^k}{s_i^k} \right) \tag{16}$$

Let $\epsilon = \{s_{u_{max}} : (1 \le u \le n) \wedge (s_{u1_{max}} \ge s_{u2_{max}},\, u1 < u2)\}$, where n is number of tasks, and $s_{u_{max}}$ is maximum transactional length in any job of τ_u. Thus, ϵ is the set of maximum transactional lengths of all task in non-increasing order. Each $s_{u_{max}} \in \epsilon$ belongs to a distinct task. Thus, $\sum_{s_{iz}^k \in \chi_i^k} len\left(\frac{s_{iz}^k}{s_i^k}\right) \le \sum_{u=1,\, s_{u_{max}} \in \epsilon}^{min(n,m)-1} s_{u_{max}}$. $\sum_{u=1,\, s_{u_{max}} \in \epsilon}^{min(n,m)-1} s_{u_{max}}$ is the sum of at most maximum $m-1$ transactional lengths of all tasks. $|\chi_i^k| \le m-1$ and $len(s_{max}^j(\Theta_j^h)) \ge len(s_i^k)$. So, (16) holds if

$$\delta_i^k \le \left(\sum_{s_j^{\bar{h}}(\Theta_j^h) \in \eta_i^k} \left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \right) - \sum_{u=1,\, s_{u_{max}} \in \epsilon}^{min(n,m)-1} s_{u_{max}} \tag{17}$$

To bound effect of transitive retry, only objects that belong to θ_i (not whole θ_i^{ex}) will be considered. So, RCM acts as if there were no transitive retry. Thus, η_i^k is modified to $\bar{\eta_i^k} = \left\{ s_j^{\bar{h}}(\Theta) \in \tau_j^* : (\Theta \in \Theta_j^h \cap \theta_i) \wedge \left(s_j^{\bar{h}}(\Theta) \notin \eta_i^l,\, l \ne k \right) \right\}$. As $\bar{\eta_i^k} \subseteq \eta_i^k$, then (17) still holds if η_i^k is replaced with $\bar{\eta_i^k}$. Consequently, (17) holds if

$$\delta_i^k \le \left(\sum_{s_j^{\bar{h}}(\Theta) \in \bar{\eta_i^k}} \left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \right) - \sum_{u=1,\, s_{u_{max}} \in \epsilon}^{min(n,m)-1} s_{u_{max}} \tag{18}$$

$\sum_{s_j^{\bar{h}}(\Theta) \in \bar{\eta_i^k}} \left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right)$ represents number of directly conflicting transactions with s_i^k in all jobs with higher priority than τ_i. Claim follows.

□

7.3. FBLT vs. G-EDF/LCM

CLAIM 6.
Schedulability of FBLT is equal or better to G-EDF/LCM's when maximum abort number of each preemptive transaction s_i^k is less or equal to sum of total number each transaction s_j^h can directly conflict with s_i^k multiplied by maximum α with which s_j^h can conflict with maximum length transaction sharing objects with s_i^k and s_j^h.

PROOF.
By substituting $RC_A(T_i)$ and $RC_B(T_i)$ in (5) with (2) and (6.7) in [El-Shambakey 2012] respectively.

$$\begin{aligned} & \sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k \in s_i} \left(\delta_i^k len(s_i^k) + \sum_{\forall s_{iz}^k \in \chi_i^k} len(s_{iz}^k) \right) + RC_{re}(T_i)}{T_i} \\ \le \; & \sum_{\forall \tau_i} \frac{\left(\sum_{\forall \tau_j \in \gamma_i^{ex}} \sum_{\theta \in \theta_i^{ex}} \left(\left\lceil \frac{T_i}{T_j} \right\rceil \sum_{\forall s_j^{\bar{h}}(\theta)} len\left(s_j^{\bar{h}}(\theta) + \alpha_{max}^{j\bar{h}} s_{max}^j(\theta) \right) \right) \right)}{T_i} \\ + \; & \sum_{\forall \tau_i} \frac{\left(\sum_{\forall s_i^k} \left(1 - \alpha_{max}^{ik}\right) len\left(s_{max}^i\right) \right) + RC_{re}(T_i)}{T_i} \end{aligned} \tag{19}$$

Let $\theta_i^{ex} = \theta_i + \theta_i^*$ where θ_i^* is the set of objects not accessed directly by τ_i but can enforce transactions in τ_i to retry due to transitive retry. Let $\gamma_i^{ex} = \gamma_i + \gamma_i^*$ where γ_i^* is the set of tasks that access objects in θ_i^*. $s_j^{\bar{h}}(\theta)$ can access multiple objects, so $s_{max}^j(\theta)$ is the maximum length transaction conflicting with $s_j^{\bar{h}}(\theta)$. $s_j^{\bar{h}}(\theta)$ 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 Θ_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 τ_i^x has the same interference pattern from higher priority jobs, τ_j^h, under FBLT and G-EDF/LCM. Hence, $RC_{re}(T_i)$ for τ_i^x is the same under FBLT and G-EDF/LCM. Consequently, (19) holds if

$$\sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k \in s_i} \left(\delta_i^k len(s_i^k) + \sum_{\forall s_{iz}^k \in \chi_i^k} len(s_{iz}^k) \right)}{T_i} \tag{20}$$
$$\leq \sum_{\forall \tau_i} \frac{\left(\sum_{\forall \tau_j \in \gamma_i^{ex}} \sum_{\forall s_i^h(\Theta_j^h),\, \Theta_j^h \in \theta_i^{ex}} \left(\left\lceil \frac{T_i}{T_j} \right\rceil len\left(s_j^{\bar{h}}(\Theta_j^h) + \alpha_{max}^{j\bar{h}} s_{max}^j(\Theta_j^h) \right) \right) \right)}{T_i}$$
$$+ \quad \sum_{\forall \tau_i} \frac{\left(\sum_{\forall s_i^k} (1 - \alpha_{max}^{ik}) len(s_{max}^i) \right)}{T_i}$$

Although different s_i^k can have common conflicting transactions $s_j^{\bar{h}}$, but no more than one s_i^k can be preceded by the same $s_j^{\bar{h}}$ in the m_set. This happens because transactions in the m_set are non-preemptive. Original priority of transactions preceding s_i^k in the m_set can be of lower or higher priority than original priority of s_i^k. Under G-EDF/LCM, $\tau_j \neq \tau_i$ can have at least one job of higher priority than current job of τ_i, then $\left\lceil \frac{T_i}{T_j} \right\rceil \geq 1$. So, each one of the s_{iz}^k in the left hand side of (20) is included in one of the $s_j^{\bar{h}}(\Theta_j^h)$ in the right hand side of (20). Then, (20) holds if

$$\sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k \in s_i} \delta_i^k len(s_i^k)}{T_i} \tag{21}$$
$$\leq \sum_{\forall \tau_i} \frac{\left(\sum_{\forall \tau_j \in \gamma_i^{ex}} \sum_{\forall s_j^{\bar{h}}(\Theta_j^h), \Theta_j^h \in \theta_i^{ex}} \left(\left\lceil \frac{T_i}{T_j} \right\rceil len\left(\alpha_{max}^{j\bar{h}} s_{max}^j(\Theta_j^h) \right) \right) \right)}{T_i}$$
$$+ \quad \sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k} (1 - \alpha_{max}^{ik}) len(s_{max}^i)}{T_i}$$

To bound effect of transitive retry, only θ_i (not the whole θ_i^{ex}) will be considered in (21). So, G-EDF/LCM acts as if there is no transitive retry. Consequently, (21) holds if

$$\sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k \in s_i} \delta_i^k len(s_i^k)}{T_i} \tag{22}$$
$$\leq \sum_{\forall \tau_i} \frac{\left(\sum_{\forall \tau_j \in \gamma_i} \sum_{\forall s_j^{\bar{h}}(\Theta), \Theta \in \Theta_j^h \cap \theta_i} \left(\left\lceil \frac{T_i}{T_j} \right\rceil len\left(\alpha_{max}^{j\bar{h}} s_{max}^j(\Theta) \right) \right) \right)}{T_i}$$
$$+ \quad \sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k} (1 - \alpha_{max}^{ik}) len(s_{max}^i)}{T_i}$$

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 $s_j^{\bar{h}}(\Theta_j^h) \in \tau_j, \forall \tau_j \neq \tau_i$ that are conflicting with s_i^k. Assuming this set of conflicting transactions with s_i^k is denoted as $\eta_i^k =$

$\left\{ \bar{s_j^h}(\Theta) \in \tau_j : \left(\Theta \in \theta_i \cap \Theta_j^h\right) \wedge (\forall \tau_j \neq \tau_i) \wedge \left(\bar{s_j^h}(\theta) \notin \eta_i^l,\, l \neq k\right)\right\}$. The last condition $\bar{s_j^h}(\theta) \notin \eta_i^l,\, l \neq k$ in definition of η_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 η_i^k, $k = 1, .., |s_i|$. This condition is necessary because in G-EDF/LCM, no two or more transactions of τ_i^x can be aborted by the same transaction of τ_j^h where $p_j^h > p_i^x$. By substitution of η_i^k in (21), we get

$$\sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k \in s_i} \delta_i^k len(s_i^k)}{T_i} \tag{23}$$
$$\leq \sum_{\forall \tau_i} \frac{\sum_{k=1}^{|s_i|} \sum_{\forall \bar{s_j^h}(\Theta) \in \eta_i^k} \left(\left\lceil \frac{T_i}{T_j} \right\rceil len\left(\alpha_{max}^{j\bar{h}} s_{max}^j(\Theta)\right)\right)}{T_i}$$
$$+ \sum_{\forall \tau_i} \frac{\left(\sum_{\forall s_i^k} (1-\alpha_{max}^{ik}) len\left(s_{max}^i\right)\right)}{T_i}$$

$\bar{s_j^h}$ belongs to higher priority jobs than τ_i and s_{max}^j belongs to higher priority jobs than τ_i or τ_i itself. Transactions in m_set can belong to jobs with original priority higher or lower than τ_i. So, (23) holds if for each $s_i^k \in \tau_i$

$$\delta_i^k len(s_i^k) \leq \left(\sum_{\forall \bar{s_j^h}(\Theta) \in \eta_i^k} \left(\left\lceil \frac{T_i}{T_j} \right\rceil \right) len\left(\alpha_{max}^{j\bar{h}} s_{max}^j(\Theta)\right)\right) + \left(1 - \alpha_{max}^{ik}\right) len\left(s_{max}^i\right)$$

Then

$$\delta_i^k \leq \left(\sum_{\forall \bar{s_j^h}(\Theta) \in \eta_i^k} \left(\left\lceil \frac{T_i}{T_j} \right\rceil \right) len\left(\frac{\alpha_{max}^{j\bar{h}} s_{max}^j(\Theta)}{s_i^k}\right)\right) + \left(1 - \alpha_{max}^{ik}\right) len\left(\frac{s_{max}^i}{s_i^k}\right) \tag{24}$$

As $len\left(\frac{s_{max}^j(\Theta)}{s_i^k}\right) \geq 1$, then (24) holds if $\delta_i^k \leq \left(\sum_{\bar{s_j^h}(\Theta) \in \eta_i^k} \left(\left\lceil \frac{T_i}{T_j} \right\rceil \alpha_{max}^{j\bar{h}}\right)\right)$. Claim follows.

□

7.4. FBLT vs. G-RMA/LCM

CLAIM 7.
Schedulability of FBLT is equal or better to G-RMA/LCM's when maximum abort number of each preemptive transaction s_i^k is less or equal to sum of maximum $m-1$ transactional lengths in all tasks subtracted from sum of total number each higher priority transaction s_j^h can directly conflict with s_i^k times maximum α with which s_j^h can conflict with maximum length transaction sharing objects with s_i^k and s_j^h.

PROOF.
By substituing $RC_A(T_i)$ and $RC_B(T_i)$ in (5) with (2) and (6.9) in [El-Shambakey 2012] respectively.

$$\sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k \in s_i} \left(\delta_i^k len(s_i^k) + \sum_{s_{iz}^k \in \chi_i^k} len(s_{iz}^k)\right) + RC_{re}(T_i)}{T_i} \tag{25}$$
$$\leq \sum_{\forall \tau_i} \frac{\sum_{\forall \tau_j^* \in \gamma_i^{ex}} \sum_{\forall \theta \in \theta_i^{ex}} \left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1\right) \sum_{\forall s_j^{\bar{h}}(\theta)} len\left(s_j^{\bar{h}}(\theta) + \alpha_{max}^{j\bar{h}} s_{max}^j(\theta)\right)}{T_i}$$

$$+ \quad \sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k} \left(1-\alpha_{max}^{ik}\right) len(s_{max}^i) + RC_{re}(T_i)}{T_i}$$

where $\tau_j^* = \{\tau_j : (\tau_j \neq \tau_i) \wedge (p_j > p_i)\}$. Let $\theta_i^{ex} = \theta_i + \theta_i^*$ where θ_i^* is the set of objects not accessed directly by τ_i but can enforce transactions in τ_i to retry due to transitive retry. Let $\gamma_i^{ex} = \gamma_i + \gamma_i^*$ where γ_i^* is the set of tasks that access objects in θ_i^*. $\bar{s_j^h}(\theta)$ can access multiple objects, so $s_{max}^j(\theta)$ is the maximum length transaction conflicting with $\bar{s_j^h}(\theta)$. $\bar{s_j^h}(\theta)$ 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 Θ_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 τ_i^x has the same interference pattern from higher priority jobs, τ_j^h, under FBLT and G-RMA/LCM. Hence, $RC_{re}(T_i)$ for τ_i^x is the same under FBLT and G-RMA/LCM. Consequently, (25) will be

$$\begin{aligned} &\sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k \in s_i} \left(\delta_i^k len(s_i^k) + \sum_{s_{iz}^k \in \chi_i^k} len(s_{iz}^k)\right)}{T_i} \\ \leq &\sum_{\forall \tau_i} \frac{\sum_{\forall \tau_j^* \in \gamma_i^{ex}} \sum_{\forall \bar{s_j^h}(\Theta_j^h),\, \Theta_j^h \in \theta_i^{ex}} \left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1\right) len\left(\bar{s_j^h}(\Theta_j^h) + \alpha_{max}^{j\bar{h}} s_{max}^j(\Theta_j^h)\right)}{T_i} \\ + &\sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k} \left(1-\alpha_{max}^{ik}\right) len(s_{max}^i)}{T_i} \end{aligned} \quad (26)$$

Although different s_i^k can have common conflicting transactions $\bar{s_j^h}$, but 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. Original priority of transactions preceding s_i^k in the m_set can be of lower or higher priority than original priority of s_i^k. Under G-RMA, $p_j > p_i$ means that $T_j \leq T_i$, then $\left\lceil \frac{T_i}{T_j} \right\rceil \geq 1$. 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 conflicting transactions with s_i^k is denoted as $\eta_i^k = \left\{\bar{s_j^h}(\Theta_j^h) \in \tau_j^* : \left(\Theta_j^h \in \theta_i^{ex}\right) \wedge \left(\bar{s_j^h}(\Theta_j^h) \notin \eta_i^l,\, l \neq k\right)\right\}$. The last condition $\bar{s_j^h}(\Theta_j^h) \notin \eta_i^l,\, l \neq k$ in definition of η_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 η_i^k, $k = 1, .., |s_i|$. This condition is necessary because in G-RMA/LCM, no two or more transactions of τ_i^x can be aborted by the same transaction of τ_j^h where $p_j^h > p_i^x$. By substitution of η_i^k in (26), we get

$$\begin{aligned} &\sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k \in s_i} \left(\delta_i^k len(s_i^k) + \sum_{s_{iz}^k \in \chi_i^k} len(s_{iz}^k)\right)}{T_i} \\ \leq &\sum_{\forall \tau_i} \frac{\left(\sum_{\forall k=1}^{|s_i|} \sum_{\bar{s_j^h}(\Theta_j^h) \in \eta_i^k} \left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1\right) len\left(\bar{s_j^h}(\Theta_j^h) + \alpha_{max}^{j\bar{h}} s_{max}^j(\Theta_j^h)\right)\right)}{T_i} \\ + &\sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k} \left(1-\alpha_{max}^{ik}\right) len(s_{max}^i)}{T_i} \end{aligned} \quad (27)$$

$\bar{s_j^h}$ belongs to higher priority jobs than τ_i. s_{max}^j belongs to higher priority jobs than τ_i or τ_i itself. s_{max}^j has a lower priority than τ_j. Transactions in m_set can belong to jobs with original priority higher or lower than τ_i. So, (27) holds if for each $s_i^k \in \tau_i$

$$\delta_i^k len(s_i^k) \le \left(\sum_{s_j^{\bar{h}}(\Theta_j^h) \in \eta_i^k} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) len\left(s_j^{\bar{h}}(\Theta_j^h) + \alpha_{max}^{j\bar{h}} s_{max}^j(\Theta_j^h) \right) \right) \right) + \left(1 - \alpha_{max}^{ik}\right) len(s_{max}^i) - \sum_{s_{iz}^k \in \chi_i^k} len(s_{iz}^k)$$

Then

$$\delta_i^k \le \left(\sum_{s_j^{\bar{h}}(\Theta_j^h) \in \eta_i^k} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) len\left(\frac{s_j^{\bar{h}}(\Theta_j^h) + \alpha_{max}^{j\bar{h}} s_{max}^j(\Theta_j^h)}{s_i^k} \right) \right) \right) + \left(1 - \alpha_{max}^{ik}\right) len\left(\frac{s_{max}^i}{s_i^k}\right) - \sum_{s_{iz}^k \in \chi_i^k} len\left(\frac{s_{iz}^k}{s_i^k}\right) \tag{28}$$

Let $\epsilon = \{s_{u_{max}} : (1 \le u \le n) \wedge (s_{u1_{max}} \ge s_{u2_{max}},\, u1 < u2)\}$, where n is number of tasks, and $s_{u_{max}}$ is maximum transactional length in any job of τ_u. Thus, ϵ is the set of maximum transactional lengths of all task in non-increasing order. Each $s_{u_{max}} \in \epsilon$ belongs to a distinct task. Thus, $\sum_{s_{iz}^k \in \chi_i^k} len\left(\frac{s_{iz}^k}{s_i^k}\right) \le \sum_{u=1,\, s_{u_{max}} \in \epsilon}^{min(n,m)-1} s_{u_{max}}$. $\sum_{u=1,\, s_{u_{max}} \in \epsilon}^{min(n,m)-1} s_{u_{max}}$ is the sum of at most maximum $m-1$ transactional lengths of all tasks. $|\chi_i^k| \le m-1$ and $len(s_{max}^j(\Theta_j^h)) \ge len(s_i^k)$. So, (28) holds if

$$\delta_i^k \le \left(\sum_{s_j^{\bar{h}}(\Theta_j^h) \in \eta_i^k} \left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \alpha_{max}^{j\bar{h}} \right) - \sum_{u=1,\, s_{u_{max}} \in \epsilon}^{min(n,m)-1} s_{u_{max}} \tag{29}$$

To bound effect of transitive retry, only objects that belong to θ_i (not whole θ_i^{ex}) will be considered. So, G-RMA/LCM acts as if there were no transitive retry. Thus, η_i^k is modified to $\bar{\eta}_i^k = \left\{ s_j^{\bar{h}}(\Theta) \in \tau_j^* : (\Theta \in \Theta_j^h \cap \theta_i) \wedge \left(s_j^{\bar{h}}(\Theta) \notin \eta_i^l,\, l \ne k \right) \right\}$. As $\bar{\eta}_i^k \subseteq \eta_i^k$, then (29) still holds if η_i^k is replaced with $\bar{\eta}_i^k$. Consequently, (29) holds if

$$\delta_i^k \le \left(\sum_{s_j^{\bar{h}}(\Theta) \in \bar{\eta}_i^k} \left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \alpha_{max}^{j\bar{h}} \right) - \sum_{u=1,\, s_{u_{max}} \in \epsilon}^{min(n,m)-1} s_{u_{max}} \tag{30}$$

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

□

7.5. FBLT vs. PNF

CLAIM 8.

Schedulability of FBLT is equal or better to PNF's with G-EDF and G-RMA if for any τ_i^x the following conditions are satisfied:

(1) *Total sum of transactional lengths of all transactions conflicting with the maximum length transaction in τ_i^x, $s_{i_{max}}$, is no less than maximum retry cost of τ_i^x due to release of higher priority jobs.*
(2) *Maximum abort number of $s_{i_{max}}$ is less or equal to difference between total sum of transactional lengths of all transactions conflicting with $s_{i_{max}}$ and maximum retry cost of τ_i^x due to release of higher priority jobs divided by length of $s_{i_{max}}$.*
(3) *Maximum abort number of any s_i^k, other than $s_{i_{max}}$, should be less or equal to total sum of transactional lengths of all conflicting transactions with s_i^k divided by length of s_i^k.*

PROOF.
By substituting $RC_A(T_i)$ and $RC_B(T_i)$ in (5) with (2) and (6.1) in [El-Shambakey 2012] respectively.

$$\begin{aligned} &\sum_{\forall\tau_i}\frac{\sum_{\forall s_i^k\in s_i}\left(\delta_i^k len(s_i^k)+\sum_{s_{iz}^k\in\chi_i^k} len(s_{iz}^k)\right)+RC_{re}(T_i)}{T_i} \\ \le\ &\sum_{\forall\tau_i}\frac{\sum_{\forall\tau_j\in\gamma_i}\sum_{\theta\in\theta_i}\left(\left(\left\lceil\frac{T_i}{T_j}\right\rceil+1\right)\sum_{\forall \bar{s}_j^h(\theta)} len\left(\bar{s}_j^h(\theta)\right)\right)}{T_i} \end{aligned} \quad (31)$$

$\bar{s}_j^h(\theta)$ can access multiple objects, so $s_{max}^j(\theta)$ is the maximum length transaction conflicting with $\bar{s}_j^h(\theta)$. As $\bar{s}_j^h(\theta)$ is included only once for all objects accessed by it. $s_{max}^j(\theta)$ is also included once for each $\bar{s}_j^h(\theta)$. Consequently, 31 becomes

$$\begin{aligned} &\sum_{\forall\tau_i}\frac{\sum_{\forall s_i^k\in s_i}\left(\delta_i^k len(s_i^k)+\sum_{s_{iz}^k\in\chi_i^k} len(s_{iz}^k)\right)+RC_{re}(T_i)}{T_i} \\ \le\ &\sum_{\forall\tau_i}\frac{\sum_{\forall\tau_j\in\gamma_i}\sum_{\bar{s}_j^h(\theta),\,\theta\in\theta_i}\left(\left(\left\lceil\frac{T_i}{T_j}\right\rceil+1\right) len\left(\bar{s}_j^h(\theta)\right)\right)}{T_i} \end{aligned} \quad (32)$$

$RC_{re}(T_i)$ is given by (6.8) in [El-Shambakey 2012] in case of G-EDF, and (6.10) in [El-Shambakey 2012] in case of G-RMA. Substituting $RC_{re}(T_i)=\sum_{\forall\tau_j\in\gamma_i}\left\lceil\frac{T_i}{T_j}\right\rceil s_{i_{max}}$, covers $RC_{re}(T_i)$ given by (6.8) and (6.10) in [El-Shambakey 2012] and maintains validity of 32. If τ_j has no shared objects with τ_i, then release of any higher priority job τ_j^y will not abort any transaction in any job of τ_i. 32 holds if

$$\begin{aligned} &\sum_{\forall\tau_i}\frac{\sum_{\forall s_i^k\in s_i}\left(\delta_i^k len(s_i^k)+\sum_{s_{iz}^k\in\chi_i^k} len(s_{iz}^k)\right)+\left(\sum_{\forall\tau_j\in\gamma_i}\left\lceil\frac{T_i}{T_j}\right\rceil s_{i_{max}}\right)}{T_i} \\ \le\ &\sum_{\forall\tau_i}\frac{\left(\sum_{\forall\tau_j\in\gamma_i}\sum_{\bar{s}_j^h(\theta),\,\theta\in\theta_i}\left(\left(\left\lceil\frac{T_i}{T_j}\right\rceil+1\right) len\left(\bar{s}_j^h(\theta)\right)\right)\right)}{T_i} \end{aligned} \quad (33)$$

Although different s_i^k can have common conflicting transactions $\bar{s}_j^h$, but 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. Original priority of transactions preceding s_i^k in the m_set can be of lower or higher priority than original priority of s_i^k. Under PNF, τ_j^y can have a priority higher or lower priority than τ_i^x, still transactions in τ_j^y can abort transactions in τ_i^x. So, each one of the s_{iz}^k in the left hand side of 33 is included in one of the $\bar{s_j^h}(\theta)$ in the right hand side of 33. 33 holds if

$$\sum_{\forall\tau_i} \frac{\sum_{\forall s_i^k \in s_i}\left(\delta_i^k len(s_i^k)\right) + \left(\sum_{\forall \tau_j \in \gamma_i}\left\lceil \frac{T_i}{T_j}\right\rceil s_{i_{max}}\right)}{T_i} \leq \sum_{\forall\tau_i} \frac{\left(\sum_{\forall \tau_j \in \gamma_i}\sum_{\bar{s_j^h}(\theta),\, \theta\in\theta_i}\left(\left\lceil \frac{T_i}{T_j}\right\rceil len\left(\bar{s_j^h}(\theta)\right)\right)\right)}{T_i} \tag{34}$$

One of the s_i^k is $s_{i_{max}}$, so 34 becomes

$$\sum_{\forall\tau_i} \frac{\sum_{\forall s_i^k \in s_i,\, s_i^k \neq s_{i_{max}}}\left(\delta_i^k len(s_i^k)\right) + \left(\left(\delta_{i_{max}} + \sum_{\forall \tau_j \in \gamma_i}\left\lceil \frac{T_i}{T_j}\right\rceil\right) s_{i_{max}}\right)}{T_i} \leq \sum_{\forall\tau_i} \frac{\left(\sum_{\forall \tau_j \in \gamma_i}\sum_{\bar{s_j^h}(\theta),\, \theta\in\theta_i}\left(\left\lceil \frac{T_i}{T_j}\right\rceil len\left(\bar{s_j^h}(\theta)\right)\right)\right)}{T_i} \tag{35}$$

For each $s_i^k \in s_i$ including $s_{i_{max}}$, there are a set of one or more $\bar{s_j^h}(\theta) \in \tau_j,\, \forall\tau_j \in \gamma_i$ that are conflicting with s_i^k. Assuming this set of conflicting transactions with s_i^k is denoted as $\eta_i^k = \left\{\bar{s_j^h}(\theta) \in \tau_j : (\theta \in \theta_i) \wedge (\forall\tau_j \in \gamma_i) \wedge \left(\bar{s_j^h}(\theta) \notin \eta_i^l,\, l \neq k\right)\right\}$. The last condition $\bar{s_j^h}(\theta) \notin \eta_i^l,\, l \neq k$ in definition of η_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 η_i^k, $k = 1, .., |s_i|$. This condition is necessary because in PNF, no two or more transactions of τ_i^x can be aborted by the same transaction of τ_j^h whether $p_j^h > p_i^x$ or not. By substitution of η_i^k in 34, we get

$$\sum_{\forall\tau_i} \frac{\left(\sum_{\forall s_i^k \in s_i,\, s_i^k \neq s_{i_{max}}}\left(\delta_i^k len(s_i^k)\right)\right) + \left(\left(\delta_{i_{max}}(T_i) + \sum_{\forall \tau_j \in \gamma_i}\left\lceil \frac{T_i}{T_j}\right\rceil\right) s_{i_{max}}\right)}{T_i} \leq \sum_{\forall\tau_i} \frac{\left(\sum_{\forall k=1}^{|s_i|}\sum_{\bar{s_j^h}(\theta)\in\eta_i^k}\left(\left\lceil \frac{T_i}{T_j}\right\rceil len\left(\bar{s_j^h}(\theta)\right)\right)\right)}{T_i} \tag{36}$$

Since $s_{i_{max}} \in s_i$, (36) holds if the following two conditions hold for each τ_i :

(1) $\delta_{i_{max}} \leq \frac{\left(\sum_{\bar{s_j^h}(\theta)\in\eta_{i_{max}}}\left\lceil \frac{T_i}{T_j}\right\rceil len\left(\bar{s_j^h}(\theta)\right)\right) - \left(\sum_{\forall\tau_j\in\gamma_i}\left\lceil \frac{T_i}{T_j}\right\rceil s_{i_{max}}\right)}{s_{i_{max}}}$, where $\eta_{i_{max}}$ is one of the η_i^ks that corresponds to $s_{i_{max}}$. $\delta_{i_{max}} \geq 0$, $\left(\sum_{\bar{s_j^h}(\theta)\in\eta_{i_{max}}}\left\lceil \frac{T_i}{T_j}\right\rceil len\left(\bar{s_j^h}(\theta)\right)\right) \geq \left(\sum_{\forall\tau_j\in\gamma_i}\left\lceil \frac{T_i}{T_j}\right\rceil s_{i_{max}}\right)$.

(2) For each s_i^k other than $s_{i_{max}}$, $\delta_i^k \leq \sum_{\bar{s_j^h}(\theta)\in\eta_i^k}\left(\left\lceil \frac{T_i}{T_j}\right\rceil len\left(\frac{\bar{s_j^h}(\theta)}{s_i^k}\right)\right)$.

Claim follows.

□

7.6. FBLT vs. Lock-free

CLAIM 9.

Under G-EDF and G-RMA, schedulability of FBLT is equal or better than lock-free's 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 [Devi et al. 2006; El-Shambakey and Ravindran 2012b] accesses only one object. Thus, the number of accessed objects per transaction in FBLT is limited to one. This allows us to compare the schedulability of FBLT with the lock-free algorithm.

By substituting $RC_A(T_i)$ and $RC_B(T_i)$ in (5) with (2) and (6.17) in [El-Shambakey 2012] respectively.

$$\begin{aligned} &\sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k \in s_i} \left(\delta_i^k len(s_i^k) + \sum_{s_{iz}^k \in \chi_i^k} len(s_{iz}^k) \right) + RC_{re}(T_i)}{T_i} \\ \leq \quad &\sum_{\forall \tau_i} \frac{\left(\sum_{\forall \tau_j \in \gamma_i} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \beta_{i,j} r_{max} \right) \right) + RC_{re}(T_i)}{T_i} \end{aligned} \tag{37}$$

where $\beta_{i,j}$ is the number of retry loops of τ_j that access the same objects as accessed by any retry loop of τ_i [Devi et al. 2006]. r_{max} is the maximum execution cost of a single iteration of any retry loop of any task [Devi et al. 2006]. For G-EDF(G-RMA), any job τ_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 [El-Shambakey 2012] respectively. $RC_{re}(T_i) = \left\lceil \frac{T_i}{T_j} \right\rceil s_{i_{max}}, \forall \tau_j \in \gamma_i$ covers $RC_{re}(T_i)$ for G-EDF/FBLT and G-RMA/FBLT. $RC_{re}(T_i) = \left\lceil \frac{T_i}{T_j} \right\rceil r_{i_{max}}, \forall \tau_j \in \gamma_i$ covers retry cost for G-EDF/lock-free and G-RMA/lock-free. (37) becomes

$$\begin{aligned} &\sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k \in s_i} \left(\delta_i^k len(s_i^k) + \sum_{s_{iz}^k \in \chi_i^k} len(s_{iz}^k) \right) + \sum_{\tau_j \in \gamma_i} \left\lceil \frac{T_i}{T_j} \right\rceil s_{i_{max}}}{T_i} \\ \leq \quad &\sum_{\forall \tau_i} \frac{\left(\sum_{\forall \tau_j \in \gamma_i} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \beta_{i,j} r_{max} \right) \right) + \sum_{\tau_j \in \gamma_i} \left\lceil \frac{T_i}{T_j} \right\rceil r_{i_{max}}}{T_i} \end{aligned} \tag{38}$$

Since $s_{max} \geq s_{i_{max}}$, $len(s_i^k)$, $len(s_{iz}^k)$, $\forall i, z, k$ and $r_{max} \geq r_{i_{max}}$ (38) holds if

$$\begin{aligned} &\sum_{\forall \tau_i} \frac{\left(\left(\sum_{\forall s_i^k \in s_i} \left(\delta_i^k + \sum_{s_{iz}^k \in \chi_i^k} 1 \right) \right) + \sum_{\tau_j \in \gamma_i} \left\lceil \frac{T_i}{T_j} \right\rceil \right) s_{max}}{T_i} \\ \leq \ &\sum_{\forall \tau_i} \frac{\left(\left(\sum_{\forall \tau_j \in \gamma_i} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \beta_{i,j} \right) \right) + \sum_{\tau_j \in \gamma_i} \left\lceil \frac{T_i}{T_j} \right\rceil \right) r_{max}}{T_i} \end{aligned} \tag{39}$$

(39) holds if for each τ_i

$$\begin{aligned} &\left(\left(\sum_{\forall s_i^k \in s_i} \left(\delta_i^k + \sum_{s_{iz}^k \in \chi_i^k} 1 \right) \right) + \sum_{\tau_j \in \gamma_i} \left\lceil \frac{T_i}{T_j} \right\rceil \right) s_{max} \\ \leq \ &\left(\left(\sum_{\forall \tau_j \in \gamma_i} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \beta_{i,j} \right) \right) + \sum_{\tau_j \in \gamma_i} \left\lceil \frac{T_i}{T_j} \right\rceil \right) r_{max} \end{aligned} \tag{40}$$

$$\frac{s_{max}}{r_{max}} \leq \frac{\left(\sum_{\forall \tau_j \in \gamma_i} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \beta_{i,j} \right) \right) + \sum_{\tau_j \in \gamma_i} \left\lceil \frac{T_i}{T_j} \right\rceil}{\left(\sum_{\forall s_i^k \in s_i} \left(\delta_i^k + \sum_{s_{iz}^k \in \chi_i^k} 1 \right) \right) + \sum_{\tau_j \in \gamma_i} \left\lceil \frac{T_i}{T_j} \right\rceil} \tag{41}$$

It appears from (41) that as δ_i^k, as well as $|\chi_i^k|$, increases, then s_{max}/r_{max} decreases. So, to get the lower bound on s_{max}/r_{max}, let $\sum_{\forall s_i^k \in s_i} \left(\delta_i^k + \sum_{s_{iz}^k \in \chi_i^k} 1 \right)$ reaches its maximum value. This maximum value is the total number of interfering transactions belonging to any job τ_j^l, $j \neq i$. Priority of τ_j^l can be higher or lower than current instance of τ_i. Beyond this maximum value, higher values for any δ_i^k are ineffective as there will be no more transactions to conflict with s_i^k. $\sum_{\forall s_i^k \in s_i}\left(\delta_i^k + \sum_{s_{iz}^k \in \chi_i^k} 1\right) \leq \sum_{\forall \tau_j \in \gamma_i}\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1\right)$. Consequently, (41) will be

$$\frac{s_{max}}{r_{max}} \leq \frac{\left(\sum_{\forall \tau_j \in \gamma_i}\left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1\right)\beta_{i,j}\right)\right) + \sum_{\tau_j \in \zeta_i}\left\lfloor \frac{T_i}{T_j} \right\rfloor}{\left(\sum_{\forall \tau_j \in \gamma_i}\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1\right)\right) + \sum_{\tau_j \in \zeta_i}\left\lfloor \frac{T_i}{T_j} \right\rfloor} \tag{42}$$

As we look for the lower bound on $\frac{s_{max}}{r_{max}}$, let $\beta_{i,j}$ assumes its minimum value. So, $\beta_{i,j} = 1$. (42) holds if $\frac{s_{max}}{r_{max}} \leq 1$.

Let $\delta_i^k \rightarrow 0$ in (41). This means transactions approximately execute in their arrival order. Let $\beta_{i,j} \rightarrow \infty$, $\left\lceil \frac{T_i}{T_j} \right\rceil \rightarrow \infty$ in (41) . This means contention is high. Consequently, $\frac{s_{max}}{r_{max}} \rightarrow \infty$. So, if transactions execute in FIFO order and contention is high, s_{max} can be much larger than r_{max}. Claim follows.

□

8. EXPERIMENTAL EVALUATION

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

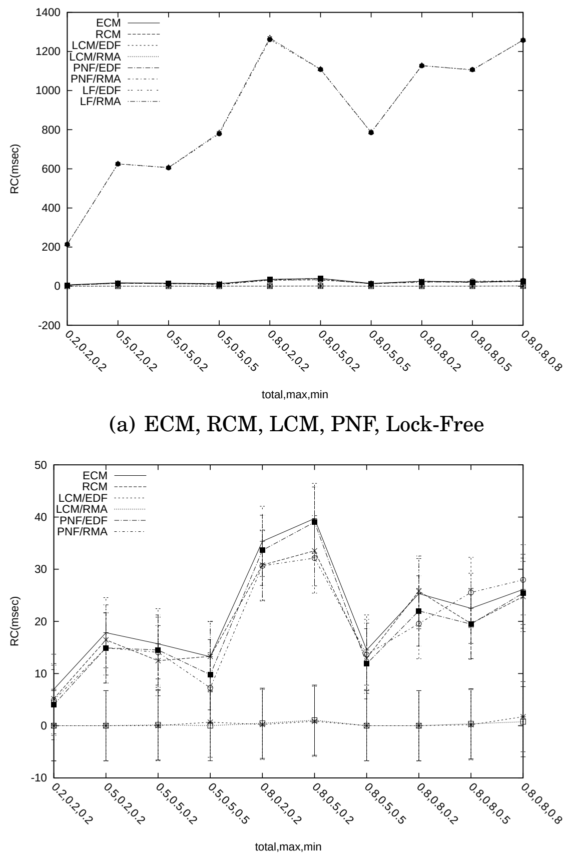
We used the ChronOS real-time Linux kernel [Dellinger et al. 2011] and the RSTM library [Marathe et al.] in our implementation. We implemented G-EDF and G-RMA schedulers in ChronOS, and modified RSTM to include implementations of ECM, RCM, LCM, and PNF. For the retry-loop lock-free synchronization, we used a loop that reads an object and attempts to write to it using a CAS instruction. The task retries until the CAS succeeds. We used an 8 core, 2GHz AMD Opteron platform. The average time taken for one write operation by RSTM on any core is 0.0129653375μs, and the average time taken by one CAS-loop operation on any core is 0.0292546250 μs.

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 PNF's retry cost with that of lock-free (and other CMs) for one object per transaction. We then compare PNF's retry cost with that of other CMs for multiple objects per transaction.

Figures 1 and 2 show the average retry cost for the 5 task and 4 task case, respectively, under 1 and 5 shared objects, respectively. 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. Confidence level of all data points is 0.95.

While Figure 1(a) includes all 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. LCM is better than the others. PNF's retry cost closely approximates ECM's and RCMs, as there is no transitive retry. From Figure 2, we observe that PNF has shorter or comparable retry cost than ECM, RCM, and LCM.

Similar trends were observed for the other task sets.



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

(b) ECM, RCM, LCM, PNF

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

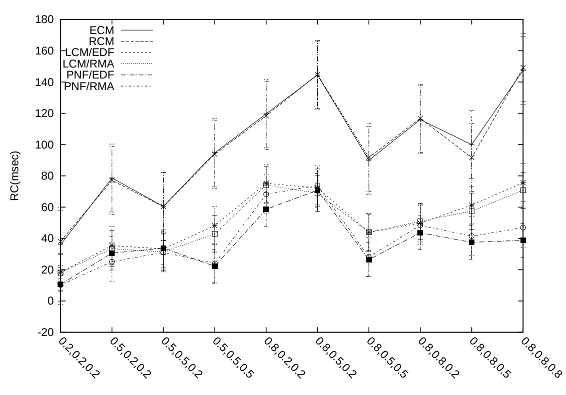


Fig. 2. Avg. retry cost (5 shared objects, 4 tasks).

9. CONCLUSIONS

Transitive retry increases transactional retry cost under ECM, RCM, and LCM. PNF avoids transitive retry by avoiding transactional preemptions. PNF reduces the priority of aborted transactions to enable other tasks to execute, increasing processor usage. Executing transactions are not preempted due to the release of higher priority jobs. On the negative side of PNF, higher priority jobs can be blocked by executing transactions of lower priority jobs.

EDF/PNF's schedulability is equal or better than ECM's when atomic section lengths are almost equal. RMA/PNF's schedulability is equal or better than RCM's when lower priority jobs suffer greater conflicts from higher priority ones. Similar conditions hold for the schedulability comparison between PNF and LCM, in addition to the increase of α terms to 1. This is logical as LCM with G-EDF (G-RMA) defaults to ECM (RCM) with $\alpha \rightarrow 1$. For PNF's schedulability to be equal or better than lock-free, the upper bound on s_{max}/r_{max} must be 1, instead of 0.5 under ECM and RCM.

REFERENCES

ANDERSON, J., RAMAMURTHY, S., AND JEFFAY, K. 1995. Real-time computing with lock-free shared objects. In *RTSS*. 28–37.

BARROS, A. AND PINHO, L. 2011. Managing contention of software transactional memory in real-time systems. In *IEEE RTSS, Work-In-Progress*.

BRANDENBURG, B., CALANDRINO, J., BLOCK, A., LEONTYEV, H., AND ANDERSON, J. 2008. 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*. 342 –353.

DELLINGER, M., GARYALI, P., AND RAVINDRAN, B. 2011. Chronos linux: a best-effort real-time multiprocessor linux kernel. In *Proceedings of the 48th Design Automation Conference*. DAC '11. ACM, New York, NY, USA, 474–479.

DEVI, U., LEONTYEV, H., AND ANDERSON, J. 2006. Efficient synchronization under global edf scheduling on multiprocessors. In *18th Euromicro Conference on Real-Time Systems*. 10 pp. –84.

EL-SHAMBAKEY, M. 2012. Phd proposal. Ph.D. thesis, Virginia Tech.

EL-SHAMBAKEY, M. AND RAVINDRAN, B. 2012a. Stm concurrency control for embedded real-time software with tighter time bounds. In *Proceedings of the 49th Annual Design Automation Conference*. DAC '12. ACM, New York, NY, USA, 437–446.

EL-SHAMBAKEY, M. AND RAVINDRAN, B. 2012b. Stm concurrency control for multicore embedded real-time software: time bounds and tradeoffs. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. SAC '12. ACM, New York, NY, USA, 1602–1609.

FAHMY, S., RAVINDRAN, B., AND JENSEN, E. 2009a. Response time analysis of software transactional memory-based distributed real-time systems. In *ACM SAC*. 334–338.

FAHMY, S., RAVINDRAN, B., AND JENSEN, E. D. 2009b. On bounding response times under software transactional memory in distributed multiprocessor real-time systems. In *DATE*. 688–693.

FAHMY, S. F. 2010. Collaborative scheduling and synchronization of distributable real-time threads. Ph.D. thesis, Virginia Tech.

HARRIS, T., LARUS, J., AND RAJWAR, R. 2010. *Transactional Memory* 2nd. Ed. Morgan & Claypool Publishers.

HERLIHY, M. 2006. The art of multiprocessor programming. In *PODC*. 1–2.

HERLIHY, M., LUCHANGCO, V., MOIR, M., AND SCHERER, III, W. N. 2003. Software transactional memory for dynamic-sized data structures. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing*. PODC '03. ACM, New York, NY, USA, 92–101.

MANSON, J., BAKER, J., ET AL. 2006. Preemptible atomic regions for real-time Java. In *RTSS*. 10–71.

MARATHE, V., SPEAR, M., HERIOT, C., ACHARYA, A., EISENSTAT, D., SCHERER III, W., AND SCOTT, M. Lowering the overhead of nonblocking software transactional memory. In *Workshop on Languages, Compilers, and Hardware Support for Transactional Computing*. TRANSACT.

SAHA, B., ADL-TABATABAI, A.-R., ET AL. 2006. McRT-STM: a high performance software transactional memory system for a multi-core runtime. In *PPoPP*. 187–197.

SARNI, T., QUEUDET, A., AND VALDURIEZ, P. 2009. Real-time support for software transactional memory. In *RTCSA*. 477–485.

SCHOEBERL, M., BRANDNER, F., AND VITEK, J. 2010. RTTM: Real-time transactional memory. In *ACM SAC*. 326–333.

Received February 2007; revised March 2009; accepted June 2009