Length-based Contention Management: Increasing Schedulability of Real-Time Software with STM Concurrency Control

***Abstract*—We consider software transactional memory (STM) concurrency control for multicore real-time software, and present a novel contention manager (CM) for resolving transactional conflicts, called length-based CM (or LCM). We upper bound transactional retries and response times under LCM, when used with G-EDF and G-RMA schedulers. We identify the conditions under which LCM is superior to previous real-time CMs including those based on dynamic and fixed priorities. Presented work is analytical as the questions we try to answer is analytical, so our results. Experimental evaluation will be done in future work.**

I. Introduction

Lock-based concurrency control suffers from programmability, scalability, and composability challenges [1]. 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 data 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 proceed to commit, yielding (the illusion of) atomicity. Aborted transactions are re-started. 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 [2] 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 (analogous to the way thread response time bounds are scheduler-dependent). 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), respectively [3]–[5]. These works show that, ECM and RCM, when used with Global EDF (G-EDF) and Global RMA (G-RMA) schedulers, respectively, achieve higher schedulability than locking and 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 IV-A). LCM resolves conflicts based on the priority of conflicting jobs, besides the length of the interfering atomic section, and the normalized length of the interfered atomic section. We establish LCM's retry and response time upper bounds, when used with G-EDF (Section IV-B) and with G-RMA (Section IV-D) schedulers. We identify the conditions under which G-EDF/LCM outperforms ECM (Section IV-C) and G-RMA/LCM outperforms RCM (Section IV-E).

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., [6]). 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 [7]. 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) [8] and nested critical sections [9].

STM concurrency control for real-time systems has been previously studied in [3], [5], [8], [10]–[13].

[10] proposes a restricted version of STM for uniprocessors. Uniprocessors do not need contention management.

[3] bounds response times in distributed multiprocessor 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.

[11] 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.

[12] 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.

[8] upper bounds retries and response times for ECM with G-EDF, and identify the tradeoffs against locking and lock-free protocols. Similar to [12], [8] also assumes that the worst case conflict between atomic sections occurs when the sections are released simultaneously.

The ideas in [8] are extended in [13], which presents three real time CM designs. But no retry bounds nor schedulability analysis techniques are presented for those CMs.

[5] presents the ECM and RCM contention managers, and upper bounds transactional retries and 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, they show that, the STM superiority holds only under some ranges for the maximum atomic section length. Our work builds upon this result.

III. 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 τ_i^j's priority, p_i^j, is determined by its absolute deadline. 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 $G_{ij}(L)$. τ_j's workload that interferes with τ_i during L is denoted $W_{ij}(L)$.

Shared objects. A task may need to access (i.e., read, write) shared, in-memory objects while it is executing any of its atomic sections, 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 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. $s_i^k(\theta)$ executes for a duration $len(s_i^k(\theta))$. If θ is shared by multiple tasks, then $s(\theta)$ is the set of atomic sections of all tasks accessing θ, and the set of tasks sharing θ with τ_i is denoted $\gamma_i(\theta)$. Atomic sections are non-nested, and each atomic section is assumed to access only one object to be consistent with the assumptions made in [5] that enabled comparison with retry-loop lock-free approach [14].

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 $s^i_{max}(\theta)$.

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 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, and s_j^k is the "interfering job", while s_i^p is the "interfered job".

The total time that a task τ_i's atomic sections have to retry is denoted $RC(\tau_i)$. When this retry cost is calculated over the task period T_i or an interval L, it is denoted, respectively, as $RC(T_i)$ and $RC_i(L)$.

IV. Length-based Contention Manager (LCM)

LCM resolves conflicts based on the priority of conflicting jobs, besides the length of the interfering atomic section, and the normalized length of the interfered atomic section. So, its design is less straight forward than ECM and RCM [5] as they depend only on the priority of the conflicting jobs. This modification in design allows lower priority jobs, under LCM, to retry for less time than ECM and RCM, but higher priority jobs, sometimes, wait for lower priority ones, but this will not result in priority inversion as will be illustrated.

A. LCM: Design and Rationale

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, LCM 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 $s_i^k(\theta)$ is the one which interferes with $s_j^l(\theta)$, then $s_j^l(\theta)$ commits because it belongs to the higher priority job and it because it started before $s_i^k(\theta)$.

We assume that

$$len(s_j^l(\theta)) = c_m len(s_i^k(\theta)) \tag{1}$$

where $c_m \in]0, \infty[$, to cover all possible lengths of $s_j^l(\theta)$. Our idea is to reduce the opportunity for the abortion of $s_i^k(\theta)$ if it is close to committing when interfered, and $len(s_j^l(\theta))$ is large. This abortion opportunity is reduced more and more as $s_i^k(\theta)$ gets closer to its end of execution, or $len(s_j^l(\theta))$ gets larger.

On the other side, 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 abortion opportunity is increased even if $s_i^k(\theta)$ is close to its end of execution. To decide whether $s_i^k(\theta)$ should be aborted or not, we use a threshold value $\psi \in [0, 1]$, that determines the length percentage of $s_i^k(\theta)$ below which $s_i^k(\theta)$ will abort due to $s_j^l(\theta)$. This percentage value is denoted α^{jl} as it differs according to s_j^l. If the abort percent is 0, it means not to abort, and 1 means to abort.

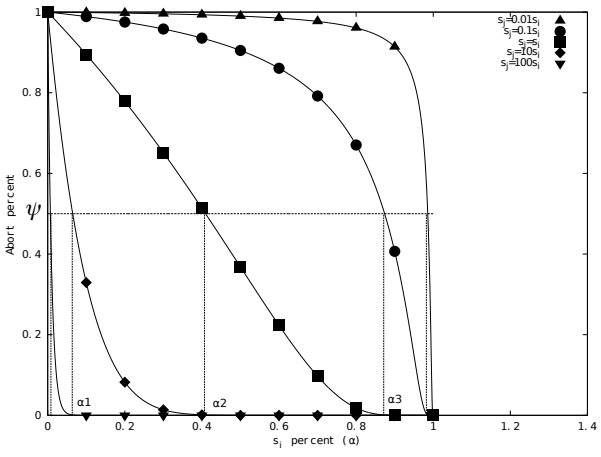


Fig. 1. Interference of $s_i^k(\theta)$ by various lengths of $s_j^l(\theta)$

The behavior of LCM is illustrated in Figure 1. The figure represents five different lengths of $s_j^l(\theta)$ interfering with $s_i^k(\theta)$ at all points of $s_i^k(\theta)$. For a specific curve (which means a specific length for $s_j^l(\theta)$), ψ determines the percentage of $len(s_i^k(\theta))$ below which $s_i^k(\theta)$ will be aborted. For example, for $len(s_j^l(\theta)) = 0.1 \times len(s_i^k(\theta))$, $s_i^k(\theta)$ will be aborted by $s_j^l(\theta)$ if the latter interferes with $s_i^k(\theta)$ no later than $s_i^k(\theta)$ reaches $\alpha 3$ percentage of its length ($\alpha 3$ is shown in Figure 1). After that, $s_j^l(\theta)$ will have to retry. As $len(s_j^k(\theta))$ decreases, the opportunity that it will abort $s_i^k(\theta)$ at a higher percentage α_{max} increases (as illustrated in Figure 1, $\alpha 3 > \alpha 2 > \alpha 1$ for reduced length of $s_j^l(\theta)$). The function that is used to represent the different curves in Figure 1 is:

$$f(c_m, \alpha) = e^{\frac{-c_m \alpha}{1-\alpha}} \tag{2}$$

where c_m is fixed for a specific curve and is calculated by (1), but α changes along each curve, with a specific value of α corresponds to ψ. This function achieves the desired requirement that the abortion opportunity is reduced as $s_i^k(\theta)$ gets closer to its end of execution (as $\alpha \to 1$, $f(c_m, 1) \to 0$), or as the length of the conflicting transaction is large (as $c_m \to \infty$, $f(\infty, \alpha) \to 0$). Meanwhile, this abortion opportunity is increased as $s_i^k(\theta)$ is interfered closer to its release (as $\alpha \to 0$, $f(c_m, 0) \to 1$), or as the length of the conflicting transaction decreases (as $c_m \to 0$, $f(0, \alpha) \to 1$). Note that all lengths of $s_i^k(\theta)$ are normalized to the same unit length. This way, different values of $s_j^l(\theta)$ interfere with different lengths of $s_i^k(\theta)$ at the same percentage α^{jl}, but the actual length of the interference for different lengths of $s_i^k(\theta)$ differ according to $len(s_i^k(\theta))$ i.e., let $len(s_i^k(\theta)) \neq len(s_i^{k+1}(\theta))$. Then, for one $s_j^l(\theta)$, both $s_i^k(\theta)$ and $s_i^{k+1}(\theta)$ will be interfered at the same value α^{jl}, but $\alpha^{jl} len(s_i^k(\theta))$ differs from $\alpha^{jl} len(s_i^{k+1}(\theta))$. The normalization of different lengths of $s_i^k(\theta)$ is done to simplify calculations.

As $s_j^l(\theta)$ belongs to a higher priority job than $s_i^k(\theta)$, if $s_j^l(\theta)$ starts before or at the same start time of $s_i^k(\theta)$, then $s_i^k(\theta)$ will have to abort and retry until $s_j^l(\theta)$ finishes execution. But if $s_j^l(\theta)$ starts after $s_i^k(\theta)$, then the comparison illustrated previously will be applied. LCM, ECM and RCM are not central CMs, which means each two transactions have to decide which one of them is to commit.

Claim 1: Let $s_j^l(\theta)$ interfere with $s_i^k(\theta)$ at α^{jl} percentage which corresponds to the threshold value ψ. Then, the maximum contribution of $s_j^l(\theta)$ to the retry cost of $s_i^k(\theta)$ is:

$$W_i^k(s_j^l(\theta)) \leq \alpha^{jl} len\Big(s_i^k(\theta)\Big) + len\Big(s_j^l(\theta)\Big) \tag{3}$$

Proof: If $s_j^l(\theta)$ interferes with $s_i^k(\theta)$ at a Υ percentage, where $\Upsilon < \alpha^{jl}$, then the retry cost of $s_i^k(\theta)$ will be $\Upsilon len(s_i^k(\theta)) + len(s_j^l(\theta))$, which is lower than that calculated in (3). Besides, if $s_j^l(\theta)$ interferes with $s_i^k(\theta)$ after α^{jl} percentage, then $s_i^k(\theta)$ will not abort. ■

Claim 2: An atomic section of a higher priority job, τ_j^b, may have to abort and retry due to a lower priority job , τ_i^a, if $s_j^l(\theta)$ interferes with $s_i^k(\theta)$ after the α^{jl} percentage. This retrial time of τ_j, due to $s_i^k(\theta)$ and $s_j^l(\theta)$, is upper bounded by:

$$W_j^l(s_i^k(\theta)) \leq \Big(1 - \alpha^{jl}\Big) len\Big(s_i^k(\theta)\Big) \tag{4}$$

Proof: It is derived directly from Claim 1, as $s_j^l(\theta)$ will have to retry for the remaining length of $s_i^k(\theta)$. ■

Claim 3: A higher priority job, τ_i^k, suffers from priority inversion for at most number of atomic sections in τ_i^k.

Proof: Assuming three atomic sections, $s_i^k(\theta)$, $s_j^l(\theta)$ and $s_a^b(\theta)$, where $p_j > p_i$ and $s_j^l(\theta)$ interferes with $s_i^k(\theta)$ after α^{jl} that corresponds to ψ. Then $s_j^l(\theta)$ will have to abort and retry. At this time, if $s_a^b(\theta)$ interferes with the other two atomic sections, and the LCM decides which transaction to commit based on comparison between each two transactions. So, we have the following cases:-

- $p_a < p_i < p_j$, then $s_a^b(\theta)$ will not abort any one because it is still in its beginning and it is of the lowest priority. So. τ_j is not indirectly blocked by τ_a.
- $p_i < p_a < p_j$ and even if $s_a^b(\theta)$ interferes with $s_i^k(\theta)$ before α^{ab} that corresponds to the threshold value ψ. So, $s_a^b(\theta)$ is allowed abort $s_i^k(\theta)$. But comparison between $s_j^l(\theta)$ and $s_a^b(\theta)$ will result in LCM choosing $s_j^l(\theta)$ to commit and abort $s_a^b(\theta)$ because the latter is still beginning, and τ_j is of higher priority. If $s_a^b(\theta)$ is not allowed to abort $s_i^k(\theta)$, the situation is still the same, because $s_j^l(\theta)$ was already retrying until $s_i^k(\theta)$ finishes, and when it is time to compare between $s_j^l(\theta)$ and $s_a^b(\theta)$, $s_j^l(\theta)$ will be chosen because it is of higher priority.
- $p_a > p_j > p_i$, then if $s_a^b(\theta)$ is chosen to commit, this is not priority inversion for τ_j because τ_a is of higher priority.
- if τ_a preempts τ_i, then LCM will compare only between $s_j^l(\theta)$ and $s_a^b(\theta)$. If $p_a < p_j$, then $s_j^l(\theta)$ will commit because of its task's higher priority and $s_a^b(\theta)$ is still at its beginning, otherwise, $s_j^l(\theta)$ will retry, but this will not be priority inversion because τ_a is already of higher priority than τ_j.

So, by generalizing these cases to any number of conflicting jobs, it is seen that when an atomic section, $s_j^l(\theta)$, of a higher

priority job is in conflict with a number of atomic sections belonging to lower priority jobs, $s_j^l(\theta)$ can suffer from priority inversion by only one of them, so if each atomic section belonging to the higher priority job suffers from priority inversion, Claim follows. ■

B. Response time of G-EDF/LCM

It is desired to determine the response time when LCM is used with G-EDF. So, the following claims are introduced.

Claim 4: When all instances of τ_h interfering with one instance of τ_i, τ_i^x, are of higher priority than τ_i^x (as shown in Figure 2(a)), then the retry cost of τ_i over T_i due to these instances of τ_h is upper bounded by

$$\Phi_i(h) = \sum_{\theta \in \theta_i \wedge \theta_h} \left(\left\lceil \frac{T_i}{T_h} \right\rceil \sum_{\forall s_h^l(\theta)} len\Big(s_h^l(\theta)\Big) + \alpha_{max}^{hl} len\Big(s_{max}^*(\theta)\Big) \right) \quad (5)$$

where $s_{max}^*(\theta)$ is the maximum length atomic section, not associated with τ_h, that accesses θ. And α_{max}^{hl} is α value that corresponds to ψ due to interference of $s_{max}^*(\theta)$ by $s_h^l(\theta)$.

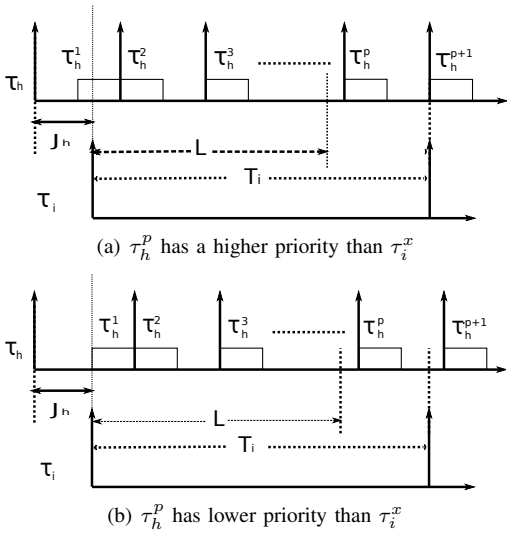
Proof: If the absolute deadline of one instance of τ_h, τ_h^p as shown in Figure 2(a), coincides with the absolute deadline of τ_i^x, then all interfering instances of τ_h with τ_i^x will have a higher priority than τ_i^x, and Claim 1 will be used to determine the retry cost of each atomic section in τ_i^x due to atomic sections in τ_h. By combining Claim 2 in [5] (noting that (5) is an upper bound for both Φ_1 and Φ_2 in Claim 2 in [5]), then Claim follows. ■

Claim 5: For an instance τ_i^x interfered by multiple instances of τ_h, and the last interfering instance, τ_h^p, has a larger absolute deadline than τ_i^x as shown in Figure 2(b). Then the retry cost of τ_i due to interfering instances of τ_h over T_i is calculated as

$$\Phi_i^*(h) = \sum_{\theta \in \theta_i \wedge \theta_h} \left(\left\lfloor \frac{T_i}{T_h} \right\rfloor \sum_{\forall s_h^l(\theta)} len\Big(s_h^l(\theta)\Big) + \alpha_{max}^{hl} len\Big(s_{max}^*(\theta)\Big) \right) + \sum_{\forall s_i^y(\theta)} \Big(1 - \alpha^{iy}\Big) len\Big(s_{h_{max}}(\theta)\Big) \quad (6)$$

where the first and second sums are the same as that calculated by (5) except it only includes the contribution of instance τ_h^1 to τ_h^{p-1} instead of instances τ_h^1 to τ_h^p, the third sum is the upper bound on retry cost of τ_i^x due to atomic section in τ_h^p, and α^{iy} is the α value that corresponds to ψ for atomic section $s_i^y(\theta)$.

Proof: Under G-EDF, there can only be at most one instance of τ_h, τ_h^p, interfering with τ_i^x, that can have a lower priority (i.e., larger absolute deadline) than τ_i^x. This is obtained by shifting Figure 2(a) to the right to give Figure 2(b). While τ_h^p cannot affect the non-atomic operation of τ_i^x, because of its lower priority, τ_h^p can abort and retry atomic sections of τ_i^x.



(a) τ_h^p has a higher priority than τ_i^x

(b) τ_h^p has lower priority than τ_i^x

Fig. 2. Interference to job τ_i by higher and lower priority jobs

So, Claim 1 is used to calculate retry cost of τ_i^x due to instances τ_h^1 to τ_h^{p-1}, and Claim 2 is used to calculate retry cost of τ_i^x due to τ_h^p. Since τ_i^x's priority is higher than that of τ_h^p, any atomic section $s_i^y(\theta)$ in τ_i^x can be aborted by only one conflicting atomic section, $s_h^z(\theta)$, of τ_h^p. This is because, after $s_h^z(\theta)$, $s_i^y(\theta)$ will start at most at the same time as any further conflicting atomic section in τ_h^z, and since $s_i^y(\theta)$ belongs to a higher priority job, LCM will commit it first. This means that $s_i^y(\theta)$ cannot be blocked by two or more atomic sections of τ_h^p. On the other hand, one atomic section in τ_h^p, $s_h^z(\theta)$, can block multiple atomic sections in τ_i^x, because any atomic section with a suitable length in any other task can cause $s_h^z(\theta)$ to retry multiple times, causing multiple atomic sections in τ_i^x to interfere with $s_h^z(\theta)$. Claim follows. ■

Claim 6: The total retry cost of τ_i due to all other tasks $\tau_h \in \gamma_i$ over T_i is upper bounded by:

$$RC(T_i) \leq \sum_{\forall \tau_h \in \gamma_i} max\{\Phi_i(h), \Phi_i^*(h)\} \quad (7)$$

Proof: The maximum contribution of each conflicting task τ_h with τ_i during T_i (which is the maximum of (5) and (6)) are summed together to give the total retry cost of τ_i. Claim follows. ■

If $\Phi_i^*(h)$ in (7) is inflated using (6), then τ_i suffers priority inversion using the sum of all other tasks, and this priority inversion is an upper bound for that in Claim 3. This is because effect of priority inversion in (6) is calculated for a single lower priority instance of τ_h, but Claim 3 considers all lower priority tasks at once. (7) has to do it this way to determine the maximum conflict effect of each task to τ_i by getting the maximum of $\Phi_i(h)$ and $\Phi_i^*(h)$. This is not the case for G-RMA/LCM as will be shown in Claim 8.

Response time of τ_i is calculated by (11) in [5].

C. Schedulability comparison of G-EDF/LCM and ECM

We now compare the schedulability of G-EDF/LCM with ECM [5] to understand when G-EDF/LCM will perform better. Toward this, we compare the total utilization of ECM against that of G-EDF/LCM. In each method (including G-EDF/LCM), we inflate the c_i for each τ_i by adding the retry cost suffered by τ_i. Thus, if method A adds retry cost $RC_A(T_i)$ to c_i, and method B adds retry cost $RC_B(T_i)$ to c_i, then the schedulability of A and B are compared as follows:

$$\begin{aligned} \sum_{\forall \tau_i} \frac{c_i + RC_A(T_i)}{T_i} &\leq \sum_{\forall \tau_i} \frac{c_i + RC_B(T_i)}{T_i} \\ \sum_{\forall \tau_i} \frac{RC_A(T_i)}{T_i} &\leq \sum_{\forall \tau_i} \frac{RC_B(T_i)}{T_i} \end{aligned} \tag{8}$$

Thus, schedulability is compared by substituting the retry cost added by synchronization methods in (8).

Claim 7: Let s_{max} be the maximum length atomic section accessing any object θ. Let α_{max} and α_{min} be the maximum and minimum percentages of normalized atomic section s_i during which s_i will abort and retry by the minimum and maximum length atomic sections, respectively. Schedulability of G-EDF/LCM is equal or better than that of ECM if for any task τ_i and interfering one τ_h:

$$\frac{1-\alpha_{min}}{1-\alpha_{max}} \leq \left\lceil \frac{T_i}{T_h} \right\rceil \tag{9}$$

Proof: Under ECM, $RC(T_i)$ is upper bounded by:

$$RC(T_i) \leq \sum_{\forall \tau_h \in \gamma_i} \sum_{\forall \theta \in (\theta_i \wedge \theta_h)} \left(\left\lceil \frac{T_i}{T_h} \right\rceil \sum_{\forall s_h^z(\theta)} 2len(s_{max}) \right) \tag{10}$$

with the assumption that all lengths of atomic sections of (4) and (8) in [5] are replaced by s_{max}.

If α_{max}^{hl} in (5) and (6) is replaced with α_{max} which results from interference of the minimum length atomic section to s_i, and α_{max}^{iy} in (6) is replaced with α_{min} which results from interference of the maximum length atomic section to s_i. As α_{max}, α_{min}, and $len(s_{max})$ are all constants, then (5) is upper bounded by:

$$\Phi_i(h) \leq \sum_{\forall \theta \in (\theta_i \wedge \theta_h)} \left(\left\lceil \frac{T_i}{T_h} \right\rceil \sum_{\forall s_h^z(\theta)} (1+\alpha_{max}) len(s_{max}) \right) \tag{11}$$

and (6) is upper bounded by:

$$\begin{aligned} \Phi_h^*(T_i) \leq & \sum_{\forall \theta \in (\theta_i \wedge \theta_h)} \left(\sum_{\forall s_i^y(\theta)} \left((1-\alpha_{min}) len(s_{max}) \right) \right. \\ & \left. + \left\lfloor \frac{T_i}{T_h} \right\rfloor \sum_{\forall s_h^z(\theta)} \left((1+\alpha_{max}) len(s_{max}) \right) \right) \end{aligned} \tag{12}$$

$RC(T_i)$ is calculated by (7). If β_1 is the total number of times any instance of τ_h accesses shared objects with τ_i, then $\beta_1 = \sum_{\theta \in (\theta_i \wedge \theta_h)} \sum_{\forall s_h^z(\theta)}$. And if β_2 is the total number of times any instance of τ_i accesses shared objects with τ_h, $\beta_2 = \sum_{\theta \in (\theta_i \wedge \theta_h)} \sum_{\forall s_i^y(\theta)}$. Then $\beta_{i,h} = max(\beta_1, \beta_2)$ is the maximum number of accesses to all shared objects by any instance of τ_i or τ_h. Thus, (10) becomes:

$$RC(T_i) \leq \sum_{\tau_h \in \gamma_i} 2 \left\lceil \frac{T_i}{T_h} \right\rceil \beta_{i,h} len(s_{max}) \tag{13}$$

and (11) becomes:

$$\Phi_i(h) \leq \left\lceil \frac{T_i}{T_h} \right\rceil \beta_{i,h} (1+\alpha_{max}) len(s_{max}) \tag{14}$$

and (12) becomes:

$$\Phi_h^*(T_i) \leq \beta_{i,h} len(s_{max}) \left((1-\alpha_{min}) + \left\lfloor \frac{T_i}{T_h} \right\rfloor (1+\alpha_{max}) \right) \tag{15}$$

The retry cost of τ_i in (14) and (15) can be combined into one equation that represents an upper bound for both of them. This upper bound for the retry cost of τ_i due to τ_h is:

$$RC_h(T_i) = \left((1-\alpha_{min}) + \left\lceil \frac{T_i}{T_h} \right\rceil (1+\alpha_{max}) \right) \beta_{i,h} s_{max} \tag{16}$$

We can now compare the total utilization of G-EDF/LCM with that of ECM:

$$\begin{aligned} & \forall \tau_i \frac{\sum_{\forall \tau_h \in \gamma_i} \left((1-\alpha_{min}) + \left\lceil \frac{T_i}{T_h} \right\rceil (1+\alpha_{max}) \right) \beta_{i,h}}{T_i} \\ \leq \quad & \forall \tau_i \frac{\sum_{\forall \tau_h \in \gamma_i} 2 \left\lceil \frac{T_i}{T_h} \right\rceil \beta_{i,h}}{T_i} \end{aligned} \tag{17}$$

(17) is satisfied if for each τ_i and τ_h, the following condition is satisfied:

$$(1-\alpha_{min}) + \left\lceil \frac{T_i}{T_h} \right\rceil (1+\alpha_{max}) \leq 2 \left\lceil \frac{T_i}{T_h} \right\rceil$$

$$\therefore \frac{1-\alpha_{min}}{1-\alpha_{max}} \leq \left\lceil \frac{T_i}{T_h} \right\rceil$$

Claim follows. ■

D. Response time of G-RMA/LCM

Claim 8: Let $\lambda_2(j,\theta) = \sum_{\forall s_j^l(\theta)} len(s_j^l(\theta)) + \alpha_{max}^{jl} len(s_{max}^j(\theta))$, and $\chi_2(i,h,\theta) = \sum_{\forall s_i^y(\theta)} (1 - \alpha_{max}^{iy}) len(s_{h_{max}}(\theta))$. Now, the retry cost of any task τ_i under G-RMA/LCM for any duration L is given by:

$$\begin{aligned} RC_i(L) = & \sum_{\forall \tau_j^*} \left(\sum_{\theta \in (\theta_i \wedge \theta_j)} \left(\left(\left\lceil \frac{L-c_j}{T_j} \right\rceil + 1 \right) \lambda_2(j,\theta) \right) \right) \\ & + \sum_{\forall \bar{\tau}_h} \left(\sum_{\theta \in (\theta_i \wedge \theta_h)} \left(\left(\left\lceil \frac{L-c_h}{T_h} \right\rceil + 1 \right) \chi_2(i,h,\theta) \right) \right) \end{aligned} \tag{18}$$

where L can extend to T_i, $\tau_j^* = \{\tau_j | (\tau_j \in \gamma_i) \wedge (p_j > p_i)\}$, and $\bar{\tau}_h = \{\tau_h | (\tau_h \in \gamma_i) \wedge (p_h < p_i)\}$.

Proof: Under G-RMA, all instances of a higher priority task, τ_j, can conflict with a lower priority task, τ_i, during

T_i. (3) will be used to determine the contribution of each conflicting atomic section in τ_j to τ_i. Meanwhile, all instances of any task, τ_h, with lower priority than τ_i, can conflict with τ_i during T_i, and (4) will be used to determine the contribution of each conflicting atomic section in τ_h to τ_i. Besides, due to the fixed priority of all instances of τ_j^* and $\bar{\tau}_h$, the equations used to calculate the retry cost of τ_i over an interval L, can be directly extended to the interval T_i. Using the previous notations and Claim 3 in [5], Claim follows. ■

The response time is calculated by (17) in [5] with replacing $RC(R_i^{up})$ with $RC_i(T_i)$.

E. Schedulability Comparison of G-RMA/LCM with RCM

Claim 9: Under the same assumptions of Claims 7 and 8, G-RMA/LCM's schedulability is equal or better than that of RCM if:

$$\frac{1-\alpha_{min}}{1-\alpha_{max}} \leq \frac{\sum_{\forall \tau_i} \frac{\sum_{\forall \tau_j^*} \left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1\right)\beta_{ij}}{T_i}}{2\sum_{\forall \tau_i} \frac{\sum_{\forall \bar{\tau}_h} \beta_{ih}}{T_i}} \tag{19}$$

Proof: Under the same assumptions as that of Claims 7 and 8, (18) can be upper bounded as:

$$\begin{aligned} RC(T_i) \quad \leq \quad & \sum_{\forall \tau_j^*} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) (1+\alpha_{max}) len(s_{max}) \beta_{ij} \right) \\ + \quad & \sum_{\forall \bar{\tau}_h} \left(\left(\left\lceil \frac{T_i}{T_h} \right\rceil + 1 \right) (1-\alpha_{min}) len(s_{max}) \beta_{ih} \right) \end{aligned} \tag{20}$$

For RCM, (16) in [5] for $RC(T_i)$ is upper bounded by:

$$RC(T_i) \leq \sum_{\forall T_j^*} \left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) 2\beta_{ij} s_{max}$$

By comparing the total utilization of G-RMA/LCM with that of RCM, we get:

$$\begin{aligned} & \sum_{\forall \tau_i} \frac{\sum_{\forall \bar{\tau}_h} \left(\left\lceil \frac{T_i}{T_h} \right\rceil + 1 \right) (1-\alpha_{min}) \beta_{ih}}{T_i} \\ \leq \quad & \sum_{\forall \tau_i} \frac{\sum_{\forall \tau_j^*} \left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) (1-\alpha_{max}) \beta_{ij}}{T_i} \end{aligned}$$

$$\therefore \frac{1-\alpha_{min}}{1-\alpha_{max}} \leq \frac{\sum_{\forall \tau_i} \frac{\sum_{\forall \tau_j^*} \left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1\right)\beta_{ij}}{T_i}}{\sum_{\forall \tau_i} \frac{\sum_{\forall \bar{\tau}_h} \left(\left\lceil \frac{T_i}{T_h} \right\rceil + 1\right)\beta_{ih}}{T_i}}$$

Since task relative deadline equals task period, and τ_h is of lower priority than τ_i, $T_i \leq T_h$. Therefore, $\left\lceil \frac{T_i}{T_h} \right\rceil = 1$ for any τ_i and τ_h. Claim follows. ■

Thus, if retry time for each task τ_i, caused by lower priority tasks, is reduced, then the right hand side of (19) is increased, giving a larger upper bound for $\frac{1-\alpha_{min}}{1-\alpha_{max}}$, and giving a wider range for α_{min} and α_{max} to choose from. Hence, by the proper choice of α_{max} and α_{min} in (19), schedulability of G-RMA/LCM can be better or equal to that of RCM.

V. CONCLUSIONS

In ECM and RCM, a task incurs at most $2s_{max}$ retry cost for each one of its atomic sections due to conflict with another task's atomic section. With LCM, this retry cost is reduced to $(1+\alpha_{max})s_{max}$ for each aborted atomic section. In ECM and RCM, tasks do not retry due to lower priority tasks, while in LCM, this happens. In G-EDF/LCM, retrial due to lower priority job is encountered only from a task τ_j's last instance during τ_i's period. This is not the case with G-RMA/LCM, because, each higher priority task can be aborted and retried by all lower priority tasks.

Schedulability of G-EDF/LCM and G-RMA/LCM can be better or equal to ECM and RCM, respectively, by proper choices for α_{min} and α_{max}.

Our work has only further scratched the surface of real-time STM. Our work is analytical, because our question is analytical—i.e., how to increase STM's schedulability advantage through a novel CM design? That said, significant insights can be gained by experimental work, which is outside this work's scope. For e.g., what are the typical range of values for the different parameters that affect the retry and blocking costs (and hence response time)? How tight is our derived upper bounds in practice? What is the most practically suitable value for ψ, α_{min}, and α_{max}? Is it more suitable to have different αs (hence different ψs) for different atomic section lengths, instead of using the normalized atomic section length?

REFERENCES

[1] M. Herlihy, "The art of multiprocessor programming," in *PODC*, 2006, pp. 1–2.

[2] T. Harris, J. Larus, and R. Rajwar, *Transactional Memory*, 2nd ed. Morgan & Claypool Publishers, December 2010.

[3] 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.

[4] S. Fahmy, B. Ravindran, and E. Jensen, "Response time analysis of software transactional memory-based distributed real-time systems," in *ACM SAC*, 2009, pp. 334–338.

[5] M. Elshambakey and B. Ravindran, "Stm concurrency control for multicore embedded real-time software: Time bounds and tradeoffs," ECE Department, Virginia Polytechnic Institute and State University, Tech. Rep., 2011. [Online]. Available: http://portal.acm.org/citation.cfm?id=956418.956614

[6] J. Anderson, S. Ramamurthy, and K. Jeffay, "Real-time computing with lock-free shared objects," in *RTSS*, 1995, pp. 28–37.

[7] B. B. Brandenburg *et al.*, "Real-time synchronization on multiprocessors: To block or not to block, to suspend or spin?" in *RTAS*, 2008, pp. 342–353.

[8] S. F. Fahmy, "Collaborative scheduling and synchronization of distributable real-time threads," Ph.D. dissertation, Virginia Tech, 2010.

[9] 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.

[10] J. Manson, J. Baker *et al.*, "Preemptible atomic regions for real-time Java," in *RTSS*, 2006, pp. 10–71.

[11] T. Sarni, A. Queudet, and P. Valduriez, "Real-time support for software transactional memory," in *RTCSA*, 2009, pp. 477–485.

[12] M. Schoeberl, F. Brandner, and J. Vitek, "RTTM: Real-time transactional memory," in *ACM SAC*, 2010, pp. 326–333.

[13] A. Barros and L. Pinho, "Managing contention of software transactional memory in real-time systems," in *IEEE RTSS, Work-In-Progress*, 2011.

[14] U. C. Devi, H. Leontyev, and J. H. Anderson, "Efficient synchronization under global EDF scheduling on multiprocessors," in *ECRTS*, 2006, pp. 75–84.