Real-Time Length-based Contention Management for STM

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, and show how LCM can achieve higher schedulability than retry-loop lock-free synchronization for G-EDF systems. Experiments compare between the retry cost and response time of G-EDF/LCM and G-RMA/LCM on one side, and ECM, RCM and lock-free concurrency control on the other, and it reveals that G-EDF/LCM and G-RMA/LCM are better or comparable to other synchronization techniques.

1. INTRODUCTION

Lock-based concurrency control suffers from programmability, scalability, and composability challenges [11]. 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 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 [10] 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), respectively [6–8]. 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 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 4.1). 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 4.2) and with G-RMA (Section 4.5) schedulers. We identify the conditions under which G-EDF/LCM outperforms ECM (Section 4.3), lock-free approach (Section 4.4) and G-RMA/LCM outperforms RCM (Section 4.6). We experimentally compare retry cost and response time for a set of tasks using G-EDF/LCM and G-RMA/LCM against ECM, RCM and retry-loop lock-free concurrency control (Section 5).

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., [1]). 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 [3]. In contrast, STM is semantically simpler [11], and is often the only viable lock-free solution for complex data structures (e.g., red/black tree) [9] and nested critical sections [14].

STM concurrency control for real-time systems has been previously studied in [2, 6, 8, 9, 12, 15, 16].

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

[8] 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.

[15] 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.

[16] 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.

[9] upper bounds retries and response times for ECM with G-EDF, and identify the tradeoffs against locking and lock-free protocols. Similar to [16], [9] also assumes that the worst case conflict between atomic sections occurs when the sections are released simultaneously.

The ideas in [9] are extended in [2], which presents three real time CM designs. But no retry bounds nor schedulability analysis techniques are presented for those CMs.

[6] 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 lock-free technique. 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.

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 τ_i^j's priority, p_i^j, differs from 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 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 $len(s_i^k(\theta))$. 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 [6].

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 job of s_j^k is the "interfering job", while job of s_i^p is the "interfered job". 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 τ_j provides when interfering with τ_i during L, without considering retrial due to atomic sections, is denoted $W_{ij}(L)$.

LCM notations. ψ is a chosen threshold value. α is a percentage value. α_{ij}^{kl} is the α value corresponding to ψ for interference of $s_i^k(\theta)$ by $s_j^l(\theta)$. c_{ij}^{kl} is the result of $len(s_j^l(\theta))/len(s_i^k(\theta))$.

4. 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 length of the interfered atomic section. So, its design is less straight forward than ECM and RCM [6] 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 with bounded priority-inversion.

4.1 LCM: Design and Rationale

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

```
if p_i^k > p_j^l then
    s_j^l(θ) aborts;
else
    c_ij^kl = len(s_j^l(θ))/len(s_i^k(θ));
    α_ij^kl = ln(ψ)/(ln(ψ) − c_ij^kl);
    α = (len(s_i^k(θ)) − δ_i^k(θ)) /len(s_i^k(θ));
    if α ≤ α_ij^kl then
        s_i^k(θ) aborts;
    else
        s_j^l(θ) aborts;
    end
end
```

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 (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 to abort $s_i^k(\theta)$ in favor of $s_j^l(\theta)$ because $len(s_j^l(\theta))$ is relatively small compared to the remaining execution length of $s_i^k(\theta)$ as explained below.

So, it is assumed that

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

where $c_{ij}^{kl} \in]0, \infty[$, to cover all possible lengths of $s_j^l(\theta)$. 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 abort or not, we use 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 $s_i^k(\theta)$. So, if the already executed part of $s_i^k(\theta)$- when $s_j^l(\theta)$ interferes with $s_i^k(\theta)$- does not exceed $\alpha_{ij}^{kl} len(s_i^k(\theta))$, $s_i^k(\theta)$ aborts (step 8), otherwise, $s_j^l(\theta)$ aborts (step 10).

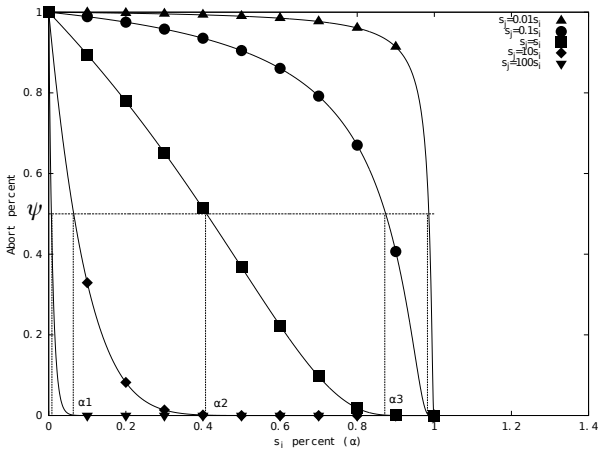


Figure 1: Interference of $s_i^k(\theta)$ by various lengths of $s_j^l(\theta)$

The behavior of LCM is illustrated in Figure 1 where the horizontal axis corresponds to different values of α ranging from 0 to 1, and the vertical axis corresponds to different values of abortion opportunities, $f(c_{ij}^{kl}, \alpha)$, ranging from 0 to 1 and calculated by (2).

$$f(c_{ij}^{kl}, \alpha) = e^{\frac{-c_{ij}^{kl}\alpha}{1-\alpha}} \quad (2)$$

where c_{ij}^{kl} is calculated by (1).

Figure 1 shows one atomic section $s_i^k(\theta)$ (whose α changes along the horizontal axis) interfered by five different lengths of $s_j^l(\theta)$. For a predefined value of $f(c_{ij}^{kl}, \alpha)$ (known as ψ in Algorithm 1), there corresponds a specific value of α (which is α_{ij}^{kl} in Algorithm 1) for each curve. For example, when $len(s_j^l(\theta)) = 0.1 \times len(s_i^k(\theta))$, $s_j^l(\theta)$ aborts $s_i^k(\theta)$ if the latter has not executed more than $\alpha 3$ percentage (shown in Figure 1) of its execution length. As $len(s_j^l(\theta))$ decreases, the corresponding α_{ij}^{kl} increases (as shown in Figure 1, $\alpha 3 > \alpha 2 > \alpha 1$).

Equation (2) achieves the desired requirement that the abortion opportunity is reduced as $s_i^k(\theta)$ gets closer to its end of execution (as $\alpha \to 1$, $f(c_{ij}^{kl}, 1) \to 0$), or as the length of the conflicting transaction is large (as $c_{ij}^{kl} \to \infty$, $f(\infty, \alpha) \to 0$). Meanwhile, this abortion opportunity is increased as $s_i^k(\theta)$ is interfered closer to its release (as $\alpha \to 0$, $f(c_{ij}^{kl}, 0) \to 1$), or as the length of the conflicting transaction decreases (as $c_{ij}^{kl} \to 0$, $f(0, \alpha) \to 1$).

LCM is not a central CM, which means each two transactions have to decide which one of them is to commit.

Claim 1. *(Proved in Section A.1) Let $s_j^l(\theta)$ interfere once with $s_i^k(\theta)$ at α_{ij}^{kl}. Then, the maximum contribution of $s_j^l(\theta)$ to the retry cost of $s_i^k(\theta)$ is:*

$$W_i^k(s_j^l(\theta)) \leq \alpha_{ij}^{kl} len\Big(s_i^k(\theta)\Big) + len\Big(s_j^l(\theta)\Big) \quad (3)$$

Claim 2. *(Proved in Section A.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_{ij}^{kl} percentage. The retrial time of τ_j, due to $s_i^k(\theta)$ and $s_j^l(\theta)$, is upper bounded by:*

$$W_j^l(s_i^k(\theta)) \leq \Big(1 - \alpha_{ij}^{kl}\Big) len\Big(s_i^k(\theta)\Big) \quad (4)$$

Claim 3. *(Proved in Section A.3) A higher priority job, τ_i^z, suffers from priority inversion for at most number of atomic sections in τ_i^z.*

Claim 4. *(Proved in Section A.4) The maximum delay suffered by $s_j^l(\theta)$ due to priority inversion is caused by the maximum length atomic section accessing object θ that belongs to a lower priority job than τ_j^b that owns $s_j^l(\theta)$.*

4.2 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 5. *(Proved in Section A.5) $RC(T_i)$ for τ_i under G-EDF/LCM is upper bounded by*

$$\begin{aligned} RC(T_i) &= \Bigg(\sum_{\forall \tau_h \in \gamma_i} \sum_{\forall \theta \in \theta_i \wedge \theta_h} \Bigg(\left\lceil \frac{T_i}{T_h} \right\rceil \sum_{\forall s_h^l(\theta)} len\Big(s_h^l(\theta)\Big) \\ &+ \alpha_{max}^{hl} len\Big(s_{max}^h(\theta)\Big) \Bigg) \Bigg) \\ &+ \sum_{\forall s_i^y(\theta)} \Big(1 - \alpha_{max}^{iy}\Big) len\Big(s_{max}^i(\theta)\Big) \end{aligned} \quad (5)$$

where α_{max}^{hl} is α value that corresponds to ψ due to interference of $s_{max}^h(\theta)$ by $s_h^l(\theta)$. And α_{max}^{iy} is α value that corresponds to ψ due to interference of $s_{max}^i(\theta)$ by $s_i^y(\theta)$.

Response time of τ_i is calculated by (11) in [6].

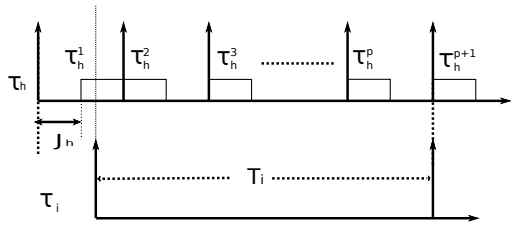


Figure 2: τ_h^p has a higher priority than τ_i^x

4.3 Schedulability comparison of G-EDF/LCM and ECM

We now compare the schedulability of G-EDF/LCM with ECM [6] 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, 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:

$$\begin{aligned} \sum_{\forall \tau_i} \frac{c_i + RC_A(T_i)}{T_i} &\leq \sum_{\forall \tau_i} \frac{c_i + RC_B(T_i)}{T_i} \\ \sum_{\forall \tau_i} \frac{RC_A(T_i)}{T_i} &\leq \sum_{\forall \tau_i} \frac{RC_B(T_i)}{T_i} \end{aligned} \quad (6)$$

Thus, schedulability is compared by substituting the retry cost added by synchronization methods in (6).

Claim 6. *(Proved in Section A.6) Let s_{max} be the maximum length atomic section accessing any object θ. Let α_{max} and α_{min} be the maximum and minimum values of α for any two atomic sections $s_i^k(\theta)$ and $s_j^l(\theta)$, and a predefined threshold ψ. Schedulability of G-EDF/LCM is equal or better than that of ECM if for any task τ_i:*

$$\frac{1-\alpha_{min}}{1-\alpha_{max}} \leq \sum_{\forall \tau_h \in \gamma_i} \left\lceil \frac{T_i}{T_h} \right\rceil \quad (7)$$

4.4 G-EDF/LCM versus lock-free synchronization

We consider the retry-loop lock-free synchronization for G-EDF system in [5]. This lock-free approach is the most relevant to our work. In the following Claim, s_{max} is used to denote $len(s_{max})$.

Claim 7. *(Proved in Section A.7) If r_{max} is the maximum execution cost of a single iteration of any retry loop of any task in the retry-loop lock-free algorithm in [5], and if the upper bound on s_{max}/r_{max} provided by G-EDF/LCM ranges between 0.5 and 2 (which is higher than that provided by ECM), then G-EDF/LCM achieves higher schedulability than retry-loop lock-free approach.*

4.5 Response time of G-RMA/LCM

Claim 8. *(Proved in Section A.8)*
Let $\lambda_2(j,\theta) = \sum_{\forall s_j^l(\theta)} len(s_j^l(\theta)) + \alpha_{max}^{jl} len(s_{max}^j(\theta))$ where α_{max}^{jl} is the α value corresponding to ψ due to interference of $s_{max}^j(\theta)$ by $s_j^l(\theta)$. The retry cost of any task τ_i under G-RMA/LCM for T_i is given by:

$$\begin{aligned} RC(T_i) &= \sum_{\forall \tau_j^*} \left(\sum_{\theta \in (\theta_i \wedge \theta_j)} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \lambda_2(j,\theta) \right) \right) \\ &+ \sum_{\forall s_i^y(\theta)} \left(1 - \alpha_{max}^{iy}\right) len\left(s_{max}^i(\theta)\right) \end{aligned} \quad (8)$$

where $\tau_j^* = \{\tau_j | (\tau_j \in \gamma_i) \wedge (p_j > p_i)\}$.

The response time is calculated by (17) in [6] with replacing $RC(R_i^{up})$ with $RC(T_i)$.

4.6 Schedulability Comparison of G-RMA/LCM with RCM

Claim 9. *(Proved in Section A.9) Under the same assumptions of Claims 6 and 8, G-RMA/LCM's schedulability is equal or better than that of RCM if:*

$$\frac{1-\alpha_{min}}{1-\alpha_{max}} \leq \sum_{\forall \tau_j^*} \left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \quad (9)$$

5. EXPERIMENTAL EVALUATION

The following experiments compare between the retry cost of G-EDF/LCM and G-RMA/LCM on one side, and ECM, RCM and retry-loop lock-free on the other side. They also compare between the response time for each task in the experiment using the previous concurrency control methods.

5.1 Experiment setup

We used ChronOS real-time operating system [4] and Roc–hester STM implementation [13], known as RSTM . We modified the RSTM to include the new contention managers ECM, RCM, G-EDF/LCM and G-RMA/LCM. For retry-loop lock-free implementation, we use a loop that reads the value of a varible, then tries to write a specified value in that variable using a CAS API provided by Rochester University; if the CAS does not succeed, task loops until it can modify the variable. We refer to this retry-loop lock-free operation as CAS-loop operation.

We use 8 core, 2GHz AMD Opeteron processor server. The average time taken by one "write" operation by RSTM implementation on any core is 0.0129653375 μs, and the average time taken by one "CAS-loop" operation on any core is 0.0292546250 μs.

Three periodic task sets are used with time properties shown in Table 1. Each task runs in its own thread is modified to include a random number of atomic sections by specifying three parameters:- the maximum and minimum lengths of any atomic section within the task, and the total length of atomic sections within any task. In the current set of experiments, each task have atomic sections, and all atomic sections access the same object. All atomic sections try to make "write" operations, so contention between tasks is at its highest level. To compare STM against retry-loop lock-free method, the same length of each atomic section within any task is implemented with the CAS-loop operation. Due to the longer time taken by one CAS-loop, the number of CAS-loops within the same length of atomic sections is smaller than the number of "STM write" operations within the same length of atomic section, but using

Table 1: Task sets (a) 5 tasks (b) 10 tasks (c)12 tasks

(a)

	$T_i(\mu s)$	$c_i(\mu s)$
τ_1	500000	150000
τ_2	1000000	227000
τ_3	1500000	410000
τ_4	3000000	299000
τ_5	5000000	500000

(b)

	$T_i(\mu s)$	$c_i(\mu s)$
τ_1	400000	75241
τ_2	750000	69762
τ_3	1200000	267122
τ_4	1500000	69863
τ_5	2400000	152014
τ_6	4000000	286301
τ_7	7500000	493150
τ_8	10000000	794520
τ_9	15000000	1212328
τ_{10}	20000000	1775342

(c)

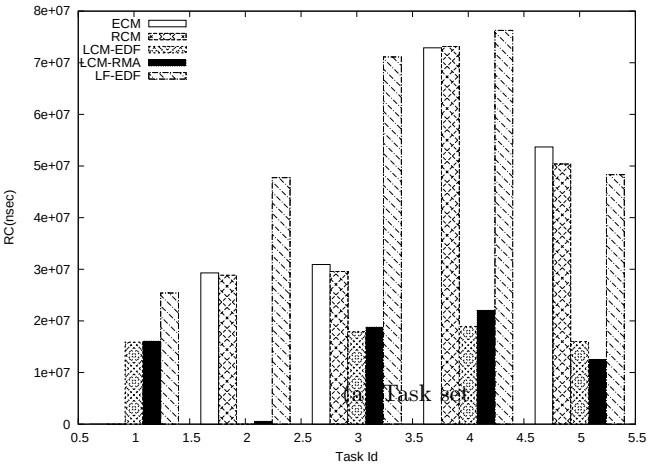
	$T_i(\mu s)$	$c_i(\mu s)$
τ_1	400000	58195
τ_2	750000	53963
τ_3	1000000	206330
τ_4	1200000	53968
τ_5	1500000	117449
τ_6	2400000	221143
τ_7	3000000	290428
τ_8	4000000	83420
τ_9	7500000	380917
τ_{10}	10000000	613700
τ_{11}	15000000	936422
τ_{12}	20000000	1371302

the same length of each atomic section in comparing STM against retry-loop lock-free allowes us to compare the effect of contention resolution between both methods, ignoring the overhead of their implementations.

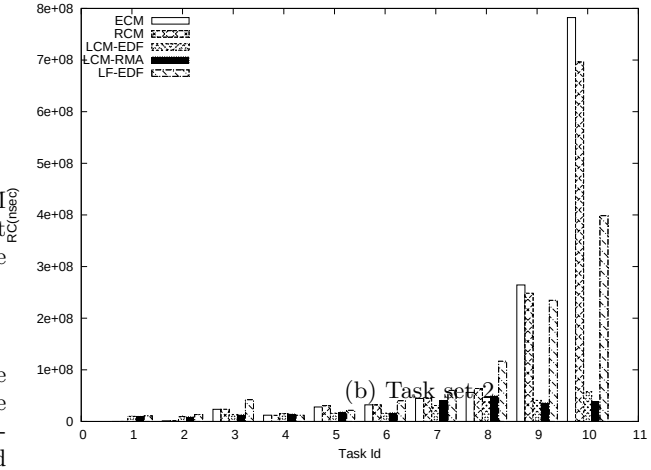
5.2 Results

Figure 3 represents retry cost (RC) for each task in the three task sets given in Table 1 where each task has one atomic section of length equal to half of its corresponding task length. It can be seen that G-EDF/LCM and G-RMA/LCM acheive better or comparative retry cost to ECM and RCM. As all tasks are initially released at the same time, and due to time propereties of tasks as shown in Table 1, this makes tasks with lower ID somehow have higher priority when using G-EDF scheduler, meanwhile, these tasks with lower ID are definitly of higher priority when using G-RMA scheduling as tasks are ordered in non-decreasing periods. Thus, it is seen that G-EDF/LCM and G-RMA/LCM acheive comparative retry cost to ECM and RCM for some tasks with lower ID, but as task ID increases, LCM - for both schedulers- acheive much better results than ECM and RCM. This is because higher priority tasks in LCM suffers blocking by lower priority tasks, which is not the case for ECM and RCM, but as task priority decreases, LCM, by definition, pervents higher priority tasks from aborting lower priority ones if the higher priority task interfers with a lower priority one after a specified threshold, but for ECM and RCM, lower priority tasks must abort in favor of higher priority ones. G-EDF/LCM and G-RMA/LCM also acheive comparative or better results than retry-loop lock-free algorithm.

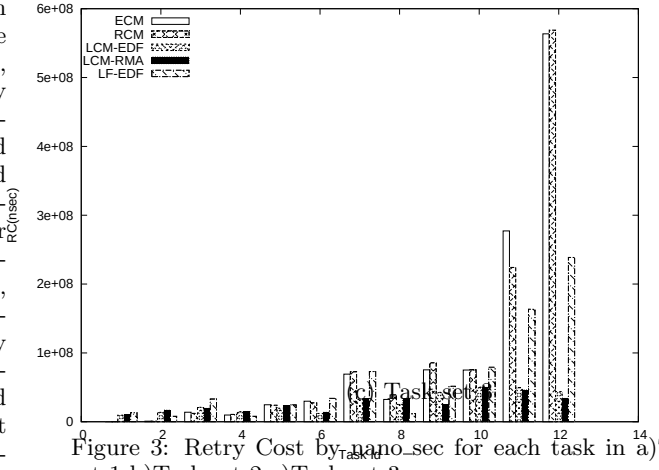
Figure 4 shows response time of each task in the three task



(a) Task set 1



(b) Task set 2



(c) Task set 3

Figure 3: Retry Cost by nano_sec for each task in a)Task set 1 b)Task set 2 c)Task set 3

sets given in Table 1 where each task has one atomic section of length equals to half of its corresponding task. It appears from Figure 4 that G-EDF/LCM and G-RMA/LCM acheive better response time than retry-loop lock-free algorithm, and comparative or better response time than ECM and RCM.

6. 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 any instance of 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}. Schedulability of G-EDF/LCM is better than retry-loop lock-free approach for G-EDF system if the upper bound on s_{max}/r_{max} lies between 0.5 and 2, which is higher than that achieved by ECM.

Our work has only further scratched the surface of real-time STM. Experimental work reveals the superiority of G-EDF/LCM and G-RMA/LCM to other real time contention managers and retry-loop lock-free concurrency control. Further experiments are needed to answer questions like: 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 ψs for different atomic section lengths, instead of using a common one? Future work is expected to include multiple objects per atomic section and nested atomic sections.

7. REFERENCES

[1] J. Anderson, S. Ramamurthy, and K. Jeffay. Real-time computing with lock-free shared objects. In *RTSS*, pages 28–37, 1995.

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

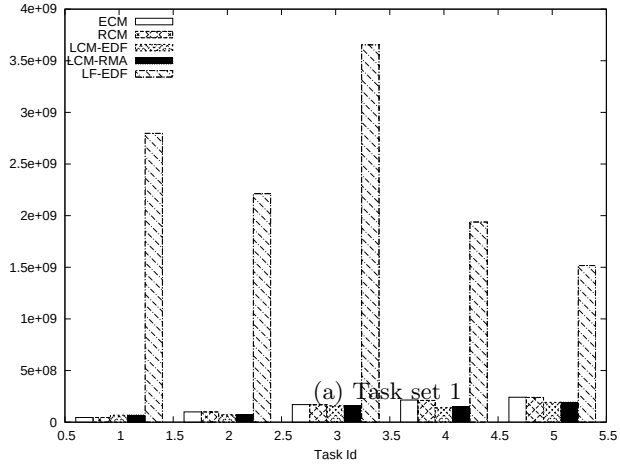
[3] B. B. Brandenburg et al. Real-time synchronization on multiprocessors: To block or not to block, to suspend or spin? In *RTAS*, pages 342–353, 2008.

[4] M. Dellinger, P. Garyali, and B. Ravindran. Chronos linux: a best-effort real-time multiprocessor linux kernel. In *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, pages 474–479. IEEE, 2011.

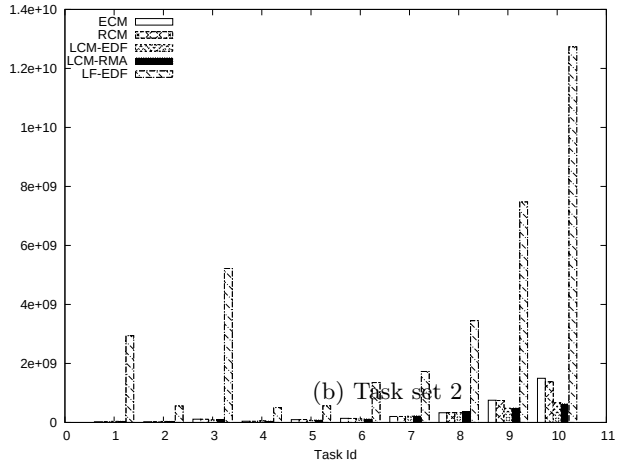
[5] U. C. Devi, H. Leontyev, and J. H. Anderson. Efficient synchronization under global EDF scheduling on multiprocessors. In *ECRTS*, pages 75–84, 2006.

[6] M. Elshambakey and B. Ravindran. Stm concurrency control for multicore embedded real-time software: Time bounds and tradeoffs. In *27th Symposium on Applied Computing*, Accepted.

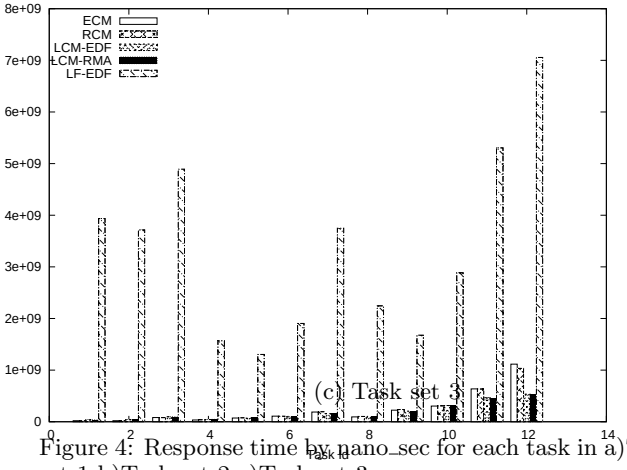
[7] S. Fahmy, B. Ravindran, and E. Jensen. Response time analysis of software transactional memory-based



(a) Task set 1



(b) Task set 2



(c) Task set 3

Figure 4: Response time by nano_sec for each task in a)Task set 1 b)Task set 2 c)Task set 3

distributed real-time systems. In *ACM SAC*, pages 334–338, 2009.

[8] S. Fahmy, B. Ravindran, and E. D. Jensen. On bounding response times under software transactional memory in distributed multiprocessor real-time systems. In *DATE*, pages 688–693, 2009.

[9] S. F. Fahmy. *Collaborative Scheduling and Synchronization of Distributable Real-Time Threads*. PhD thesis, Virginia Tech, 2010.

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

[11] M. Herlihy. The art of multiprocessor programming. In *PODC*, pages 1–2, 2006.

[12] J. Manson, J. Baker, et al. Preemptible atomic regions for real-time Java. In *RTSS*, pages 10–71, 2006.

[13] V. Marathe, M. Spear, C. Heriot, A. Acharya, D. Eisenstat, W. Scherer III, and M. Scott. Lowering the overhead of nonblocking software transactional memory. In *Workshop on Languages, Compilers, and Hardware Support for Transactional Computing (TRANSACT)*, 2006.

[14] B. Saha, A.-R. Adl-Tabatabai, et al. McRT-STM: a high performance software transactional memory system for a multi-core runtime. In *PPoPP*, pages 187–197, 2006.

[15] T. Sarni, A. Queudet, and P. Valduriez. Real-time support for software transactional memory. In *RTCSA*, pages 477–485, 2009.

[16] M. Schoeberl, F. Brandner, and J. Vitek. RTTM: Real-time transactional memory. In *ACM SAC*, pages 326–333, 2010.

APPENDIX

A. PROOFS OF CLAIMS

This section includes proofs of presented Claims in the paper.

A.1 Proof of Claim 1

PROOF. If $s_j^l(\theta)$ interferes with $s_i^k(\theta)$ at a Υ percentage, where $\Upsilon < \alpha_{ij}^{kl}$, 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_{ij}^{kl} percentage, then $s_i^k(\theta)$ will not abort. □

A.2 Proof of Claim 2

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

A.3 Proof of Claim 3

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_{ij}^{kl}. 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_{ia}^{kb}, so, $s_a^b(\theta)$ is allowed to abort $s_i^k(\theta)$. 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.
- $p_a > p_j > p_i$, 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. If τ_a does not access any object but it preempts τ_i, then CM will choose $s_j^l(\theta)$ to commit as only already running transactions are competing together.

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, each higher priority job can suffer priority inversion at most its number of atomic sections. Claim follows. □

A.4 Proof of Claim 4

PROOF. For three atomic sections, $s_i^k(\theta)$, $s_j^l(\theta)$ and $s_h^z(\theta)$, where $p_j > p_i$, $p_j > p_h$ and $len(s_i^k(\theta)) > len(s_h^z(\theta))$, then $\alpha_{ij}^{kl} > \alpha_{hj}^{zl}$ and $c_{ij}^{kl} < c_{hj}^{zl}$. By applying (4) to get the contribution of $s_i^k(\theta)$ and $s_h^z(\theta)$ to the priority inversion of $s_j^l(\theta)$ and dividing them, we get

$$\frac{W_j^l(s_i^k(\theta))}{W_j^l(s_h^z(\theta))} = \frac{\left(1-\alpha_{ij}^{kl}\right) len(s_i^k(\theta))}{\left(1-\alpha_{hj}^{zl}\right) len(s_h^z(\theta))}$$

By substitution for αs from (2)

$$= \frac{(1-\frac{ln\psi}{ln\psi-c_{ij}^{kl}})len(s_i^k(\theta))}{(1-\frac{ln\psi}{ln\psi-c_{hj}^{zl}})len(s_h^z(\theta))} = \frac{(\frac{-c_{ij}^{kl}}{ln\psi-c_{ij}^{kl}})len(s_i^k(\theta))}{(\frac{-c_{hj}^{zl}}{ln\psi-c_{hj}^{zl}})len(s_h^z(\theta))}$$

$\because ln\psi \leq 0$ and $c_{ij}^{kl}, c_{hj}^{kl} > 0, \therefore$ by substitution from (1)

$$= \frac{len(s_j^l(\theta))/(ln\psi - c_{ij}^{kl})}{len(s_j^l(\theta))/(ln\psi - c_{hj}^{zl})} = \frac{ln\psi - c_{hj}^{zl}}{ln\psi - c_{ij}^{kl}} > 1$$

Thus, as the length of the interfered atomic section increases, the effect of priority inversion on the interfering atomic section increases. Claim follows. □

A.5 Proof of Claim 5

PROOF. The maximum number of higher priority instances of τ_h that can interfere with τ_i^x is $\left\lceil \frac{T_i}{T_h} \right\rceil$ as shown in Figure 2 where one instance of τ_h, τ_h^p, coincides with the absolute deadline of τ_i^x.

By using Claims 1, 2, 3, 4 and Claim 1 in [6] to determine the effect of atomic sections belonging to higher and lower priority instances of interfering tasks to τ_i^x, Claim follows. □

A.6 Proof of Claim 6

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 [6] and (5) are replaced by s_{max}. If α_{max}^{hl} in (5) is replaced with α_{max}, and α_{max}^{iy} in (5) is replaced with α_{min}. As α_{max}, α_{min}, and $len(s_{max})$ are all constants, then (5) is upper bounded by:

$$RC(T_i) \leq \left(\sum_{\forall \tau_h \in \gamma_i} \sum_{\forall \theta \in \theta_i \wedge \theta_h} \left(\left\lceil \frac{T_i}{T_h} \right\rceil \sum_{\forall s_h^l(\theta)} (1+\alpha_{max}) len\Big(s_{max}\Big) \right) \right) + \sum_{\forall s_i^y(\theta)} \Big(1-\alpha_{min}\Big) len\Big(s_{max}\Big) \tag{11}$$

If β_1^{ih} is the total number of times any instance of τ_h accesses shared objects with τ_i, then $\beta_1^{ih} = \sum_{\forall \theta \in (\theta_i \wedge \theta_h)} \sum_{\forall s_h^z(\theta)}$. And if β_2^i is the total number of times any instance of τ_i accesses shared objects with any other instance, $\beta_2^i = \sum_{\forall s_i^y(\theta)}$, *where θ is shared with another task.* Then $\beta_i = max\{max_{\forall \tau_h \in \gamma_i}\{\beta_1^{ih}\}, \beta_2^i\}$ is the maximum number of accesses to all shared objects by any instance of τ_i or τ_h. Thus, (10) becomes:

$$RC(T_i) \leq \sum_{\tau_h \in \gamma_i} 2 \left\lceil \frac{T_i}{T_h} \right\rceil \beta_i len(s_{max}) \tag{12}$$

and (11) becomes:

$$RC(T_i) \leq \beta_i len(s_{max}) \left((1-\alpha_{min}) + \sum_{\forall \tau_h \in \gamma_i} \left\lceil \frac{T_i}{T_h} \right\rceil (1+\alpha_{max}) \right) \tag{13}$$

We can now compare the total utilization of G-EDF/LCM with that of ECM by comparing (11) and (13) for all τ_i:

$$\sum_{\forall \tau_i} \frac{(1-\alpha_{min}) + \sum_{\forall \tau_h \in \gamma_i} \left(\left\lceil \frac{T_i}{T_h} \right\rceil (1+\alpha_{max}) \right)}{T_i} \leq \sum_{\forall \tau_i} \frac{\sum_{\forall \tau_h \in \gamma_i} 2 \left\lceil \frac{T_i}{T_h} \right\rceil}{T_i} \tag{14}$$

(14) is satisfied if for each τ_i the following condition is satisfied:

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

$$\therefore \frac{1-\alpha_{min}}{1-\alpha_{max}} \leq \sum_{\forall \tau_h \in \gamma_i} \left\lceil \frac{T_i}{T_h} \right\rceil$$

Claim follows. □

A.7 Proof of Claim 7

PROOF. From [5], the retry-loop lock-free algorithm is upper bounded by:

$$RL(T_i) = \sum_{\tau_h \in \gamma_i} \left(\left\lceil \frac{T_i}{T_h} \right\rceil + 1 \right) \beta_i r_{max} \tag{15}$$

where β_i is as defined in Claim 6. The retry cost of τ_i in G-EDF/LCM is upper bounded by (13). By comparing G-EDF/LCM's total utilization with that of the retry-loop lock-free algorithm, we get:

$$\sum_{\forall \tau_i} \frac{\Big((1-\alpha_{min}) + \sum_{\forall \tau_h \in \gamma_i} \Big(\left\lceil \frac{T_i}{T_h} \right\rceil (1+\alpha_{max}) \Big)\Big) \beta_i s_{max}}{T_i} \leq \sum_{\forall \tau_i} \frac{\sum_{\forall \tau_h \in \gamma_i} \Big(\left\lceil \frac{T_i}{T_h} \right\rceil + 1 \Big) \beta_i r_{max}}{T_i}$$

$$\therefore \frac{s_{max}}{r_{max}} \leq \frac{\sum_{\forall \tau_i} \frac{\sum_{\forall \tau_h \in \gamma_i} \Big(\left\lceil \frac{T_i}{T_h} \right\rceil + 1 \Big) \beta_i}{T_i}}{\sum_{\forall \tau_i} \frac{\Big((1-\alpha_{min}) + \sum_{\forall \tau_h \in \gamma_i} \Big(\left\lceil \frac{T_i}{T_h} \right\rceil (1+\alpha_{max}) \Big)\Big) \beta_i}{T_i}} \tag{16}$$

If

$$\frac{s_{max}}{r_{max}} \leq \frac{\sum_{\forall \tau_h \in \gamma_i} \left(\left\lceil \frac{T_i}{T_h} \right\rceil + 1 \right)}{(1-\alpha_{min}) + \sum_{\forall \tau_h \in \gamma_i} \left(\left\lceil \frac{T_i}{T_h} \right\rceil (1+\alpha_{max}) \right)} \tag{17}$$

for each τ_i, then (16) holds.

Let the number of tasks that have shared objects with τ_i (i.e., $\sum_{\tau_h \in \gamma_i} = \omega \geq 1$ since at least one task has a shared object with τ_i, otherwise, there is no conflict between tasks), and let the total number of tasks be n, so $1 \leq \omega \leq n-1$, and $\left\lceil \frac{T_i}{T_h} \right\rceil \in [1, \infty[$. To find the minimum and upper values for the upper bound on s_{max}/r_{max}, we consider the following cases:-

- $\alpha_{min} \to 0, \alpha_{max} \to 0$

$\therefore$ (17) will be

$$\frac{s_{max}}{r_{max}} \leq \frac{\sum_{\forall \tau_h \in \gamma_i} \left(\left\lceil \frac{T_i}{T_h} \right\rceil + 1 \right)}{1 + \sum_{\forall \tau_h \in \gamma_i} \left\lceil \frac{T_i}{T_h} \right\rceil} = 1 + \frac{\omega - 1}{1 + \sum_{\forall \tau_h \in \gamma_i} \left\lceil \frac{T_i}{T_h} \right\rceil} \tag{18}$$

By substituting edge values for ω and $\left\lceil \frac{T_i}{T_h} \right\rceil$ into (18), then the upper bound on s_{max}/r_{max} lies between 1 and 2.

- $\alpha_{min} \to 0, \alpha_{max} \to 1$

(17) becomes

$$\frac{s_{max}}{r_{max}} \leq 0.5 + \frac{\omega - 0.5}{1 + \sum_{\forall \tau_h \in \gamma_i} \left\lceil \frac{T_i}{T_h} \right\rceil} \tag{19}$$

By applying edge values for ω and $\left\lceil \frac{T_i}{T_h} \right\rceil$ in (19), then the upper bound on s_{max}/r_{max} lies between 0.5 and 1.5.

- $\alpha_{min} \to 1, \alpha_{max} \to 0$

This case is rejected since $\alpha_{min} \leq \alpha_{max}$

- $\alpha_{min} \to 1, \alpha_{max} \to 1$

$\therefore$ (17) becomes

$$\frac{s_{max}}{r_{max}} \leq \frac{\omega + \sum_{\forall \tau_h \in \gamma_i} \left\lceil \frac{T_i}{T_h} \right\rceil}{2 \sum_{\forall \tau_h \in \gamma_i} \left\lceil \frac{T_i}{T_h} \right\rceil} = \frac{\omega}{2 \sum_{\forall \tau_h \in \gamma_i} \left\lceil \frac{T_i}{T_h} \right\rceil} + 0.5 \tag{20}$$

By applying edge values for ω and $\left\lceil \frac{T_i}{T_h} \right\rceil$ in (20), then the upper bound on s_{max}/r_{max} lies between 0.5 and 1, which is similar to that achieved by ECM.

So, from the previous cases, the upper bound on s_{max}/r_{max} lies between 0.5 and 2, while in case of ECM [6], it lies between 0.5 and 1. Claim follows. □

A.8 Proof of Claim 8

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 with lower priority than τ_i, can conflict with τ_i during T_i, and Claims 2 and 3 will be used to determine the contribution of conflicting atomic sections in lower priority tasks to τ_i. Using the previous notations and Claim 3 in [6], Claim follows. □

A.9 Proof of Claim 9

PROOF. Under the same assumptions as that of Claims 6 and 8, (8) can be upper bounded as:

$$\begin{aligned} RC(T_i) &\leq \sum_{\forall \tau_j^*} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) (1 + \alpha_{max}) len(s_{max}) \beta_i \right) \\ &+ (1 - \alpha_{min}) len(s_{max}) \beta_i \end{aligned} \tag{21}$$

For RCM, (16) in [6] for $RC(T_i)$ is upper bounded by:

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

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

$$\begin{aligned} &\sum_{\forall \tau_i} \frac{len(s_{max})\beta_i \left((1-\alpha_{min}) + \sum_{\forall \tau_j^*} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) (1+\alpha_{max}) \right) \right)}{T_i} \\ \leq &\sum_{\forall \tau_i} \frac{2 len(s_{max}) \beta_i \sum_{\forall \tau_j^*} \left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right)}{T_i} \end{aligned} \tag{22}$$

(22) is satisfied if $\forall \tau_i$ (9) is satisfied. Claim follows. □