STM Concurrency Control for Embedded Real-Time Software with Tighter Time Bounds

Mohammed El-Shambakey
ECE Dept., Virginia Tech
Blacksburg, VA 24060, USA
shambake@vt.edu

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

ABSTRACT

We consider software transactional memory (STM) concurrency control for 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 outperforms previous real-time STM CMs and lock-free synchronization. Our implementation and experimental studies reveal that G-EDF/LCM and G-RMA/LCM have shorter or comparable retry costs and response times than other synchronization techniques.

Categories and Subject Descriptors

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

General Terms

Design, Experimentation, Measurement

Keywords

Software transactional memory (STM), real-time contention manager

1. INTRODUCTION

Lock-based concurrency control suffers from programmability, scalability, and composability challenges [12]. 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 [11] 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 [7–9]. 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 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) and lock-free synchronization (Section 4.4), and G-RMA/LCM outperforms RCM (Section 4.6). We implement LCM and competitor CM techniques in the Rochester STM framework [14] and conduct experimental studies (Section 5). 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

DAC 2012, June 3-7, 2012, San Francisco, California, USA.

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., [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 [12], and is often the only viable lock-free solution for complex data structures (e.g., red/black tree) [10] and nested critical sections [15].

STM concurrency control for real-time systems has been previously studied in [2, 7, 9, 10, 13, 16, 17].

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

[9] 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.

[16] 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.

[17] 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.

[10] upper bounds retries and response times for ECM with G-EDF, and identify the tradeoffs with locking and lock-free protocols. Similar to [17], [10] also assumes that the worst case conflict between atomic sections occurs when the sections are released simultaneously. The ideas in [10] are extended in [2], which presents three real-time CM designs. But no retry bounds or schedulability analysis techniques are presented for those CMs.

[7] 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 synchronization. In particular, they show that, STM's 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 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, 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. $s_i^k(\theta)$ 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 (supporting nested STM is future work). Each section is assumed to access only one object; this allows us to be consistent with the assumptions in [7], enabling a comparison with retry-loop lock-free synchronization [5], which is an important goal of this paper. 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." 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 a task τ_j causes to response time of τ_i when interfering with τ_i during L, without considering retries due to atomic sections, is denoted $W_{ij}(L)$.

4. LENGTH-BASED CM

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. This is in contrast to ECM and RCM [7], where conflicts are resolved using the priority of the conflicting jobs. This strategy allows lower priority jobs, under LCM, to retry for lesser time than that under ECM and RCM, but higher priority jobs, sometimes, wait for lower priority ones with bounded priority-inversion.

4.1 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, 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

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 | $\alpha = \big(len(s_i^k(\theta)) - \delta_i^k(\theta)\big)\,/len(s_i^k(\theta))$;
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**

$s_i^k(\theta)$ (explained further).

We assume 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 $s_j^l(\theta)$. Our 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, 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 $s_i^k(\theta)$. 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).

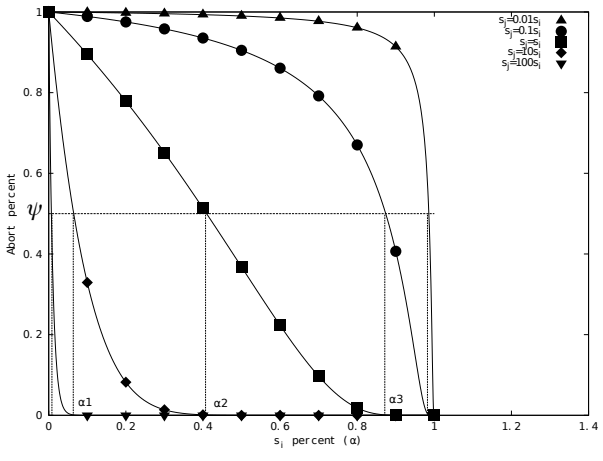


Figure 1: Interference of $s_i^k(\theta)$ by various lengths of $s_j^l(\theta)$

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

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

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 $f(c_{ij}^{kl}, \alpha)$ (denoted 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 abort opportunity is reduced as $s_i^k(\theta)$ gets closer to the end of its execution (as $\alpha \rightarrow 1$, $f(c_{ij}^{kl}, 1) \rightarrow 0$), or as the length of the conflicting transaction increases (as $c_{ij}^{kl} \rightarrow \infty$, $f(\infty, \alpha) \rightarrow 0$). Meanwhile, this abort opportunity is increased as $s_i^k(\theta)$ is interfered closer to its release (as $\alpha \rightarrow 0$, $f(c_{ij}^{kl}, 0) \rightarrow 1$), or as the length of the conflicting transaction decreases (as $c_{ij}^{kl} \rightarrow 0$, $f(0, \alpha) \rightarrow 1$).

LCM is not a centralized CM, which means that, upon a conflict, each transactions has to decide whether it must commit or abort.

Claim 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 $s_i^k(\theta)$'s retry cost is:*

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

(Proofs of all claims are provided in the Supplementary Material section at the end of the paper.)

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_{ij}^{kl} percentage. τ_j's retry time, due to $s_i^k(\theta)$ and $s_j^l(\theta)$, is upper bounded by:*

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

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

4.2 Response Time of G-EDF/LCM

Claim 5. *$RC(T_i)$ for a task τ_i under G-EDF/LCM is*

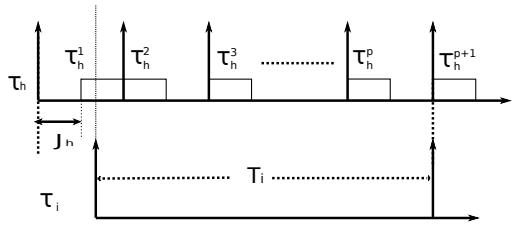


Figure 2: τ_h^p has a higher priority than τ_i^x

upper bounded by:

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

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

Response time of τ_i is calculated by (11) in [7].

4.3 Schedulability of G-EDF/LCM and ECM

We now compare the schedulability of G-EDF/LCM with ECM [7] to understand when G-EDF/LCM will perform better. Toward this, we compare the total utilization of ECM with that of G-EDF/LCM. For each method, we inflate the c_i of each task 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:

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

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

CLAIM 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 $s_j^l(\theta)$. Given a threshold ψ, schedulability of G-EDF/LCM is equal or better than 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

We consider the retry-loop lock-free synchronization for G-EDF given in [5]. This lock-free approach is the most relevant to our work.

CLAIM 7. *Let s_{max} denote $len(s_{max})$ and r_{max} denote the maximum execution cost of a single iteration of any retry loop of any task in the retry-loop lock-free algorithm in [5]. Now, G-EDF/LCM achieves higher schedulability than the retry-loop lock-free approach if the upper bound on s_{max}/r_{max} under G-EDF/LCM ranges between 0.5 and 2 (which is higher than that under ECM).*

4.5 Response Time of G-RMA/LCM

CLAIM 8. *Let*

$$\lambda_2(j, \theta) = \sum_{\forall s_j^l(\theta)} len(s_j^l(\theta)) + \alpha_{max}^{jl} len(s_{max}^j(\theta))$$

where α_{max}^{jl} is the α value corresponding to ψ due to the interference of $s_{max}^j(\theta)$ by $s_j^l(\theta)$. The retry cost of any task τ_i under G-RMA/LCM during T_i is given by:

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

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

The response time is calculated by (17) in [7] with replacing $RC(R_i^{up})$ with $RC(T_i)$.

4.6 Schedulability of G-RMA/LCM and RCM

CLAIM 9. *Under the same assumptions of Claims 6 and 8, G-RMA/LCM's schedulability is equal or better than 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

Having established LCM's retry and response time upper bounds, and the conditions under which it outperforms ECM, RCM, and lock-free synchronization, we now would like to understand how LCM's retry and response times in practice (i.e., on average) compare with that of competitor methods. Since this can only be understood experimentally, we implement LCM and the competitor methods and conduct experimental studies.

5.1 Experimental Setup

We used the ChronOS real-time Linux kernel [4] and the RSTM library [14]. We modified RSTM to include implementations of ECM, RCM, G-EDF/LCM, and G-RMA/LCM contention managers, and modified ChronOS to include implementations of G-EDF and G-RMA schedulers.

For the retry-loop lock-free implementation, we used a loop that reads an object and attempts to write to the object using a compare-and-swap (CAS) instruction. The task retries until the CAS succeeds.

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

We used the periodic task set shown in Table 1. Each task runs in its own thread and has an atomic section. Atomic section properties are probabilistically controlled (for experimental evaluation) using three parameters: the maximum

Table 1: Task sets. (a) Task set 1: 5-task set; (b) Task set 2: 10-task set; (c) Task set 3: 12-task set

(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)

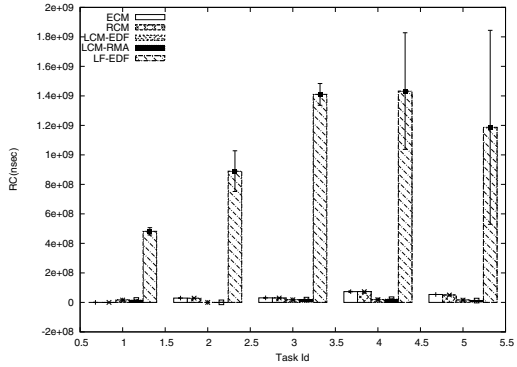
	$T_i(\mu s)$	$c_i(\mu s)$
τ_1	400000	58195
τ_2	750000	53963
τ_3	1000000	206330
τ_4	1200000	53968
τ_5	1500000	117449
τ_6	2400000	221143
τ_7	3000000	290428
τ_8	4000000	83420
τ_9	7500000	380917
τ_{10}	10000000	613700
τ_{11}	15000000	936422
τ_{12}	20000000	1371302

and minimum lengths of any atomic section within the task, and the total length of atomic sections within any task. All task atomic sections access the same object, and do write operations on the object (thus, contention is the highest).

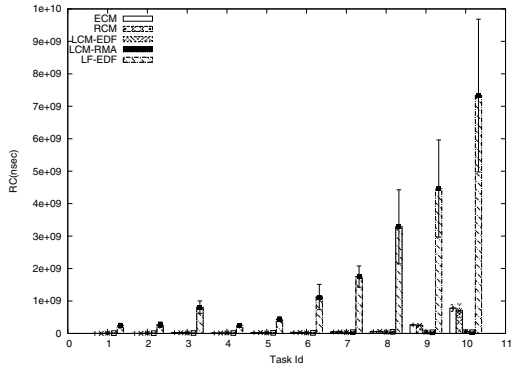
5.2 Results

Figure 3 shows the retry cost (RC) for each task in the three task sets in Table 1, where each task's atomic section length is equal to half of the task length. Each data point in the figure has a confidence level of 0.95. We observe that G-EDF/LCM and G-RMA/LCM achieve shorter or comparable retry cost than ECM and RCM. Since all tasks are initially released at the same time, and due to the specific nature of task properties, tasks with lower IDs somehow have higher priorities under the G-EDF scheduler. Note that tasks with lower IDs have higher priorities under G-RMA, since tasks are ordered in non-decreasing order of their periods.

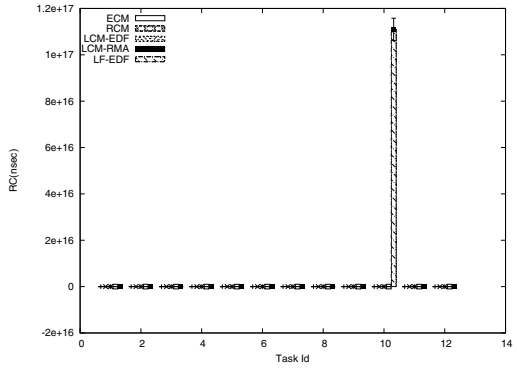
Thus, we observe that G-EDF/LCM and G-RMA/LCM achieve comparable retry costs to ECM and RCM for some tasks with lower IDs. But when task ID increases, LCM — for both schedulers — achieves much shorter retry costs than ECM and RCM. This is because, higher priority tasks in LCM suffers blocking by lower priority tasks, which is not the case for ECM and RCM. However, as task priority decreases, LCM, by definition, prevents higher priority tasks from aborting lower priority ones if a higher priority task interferes with a lower priority one after a specified threshold. In contrast, under ECM and RCM, lower priority tasks abort in favor of higher priority ones. G-EDF/LCM and G-RMA/LCM also achieve shorter retry costs than the retry-loop lock-free algorithm.



(a) Task set 1



(b) Task set 2



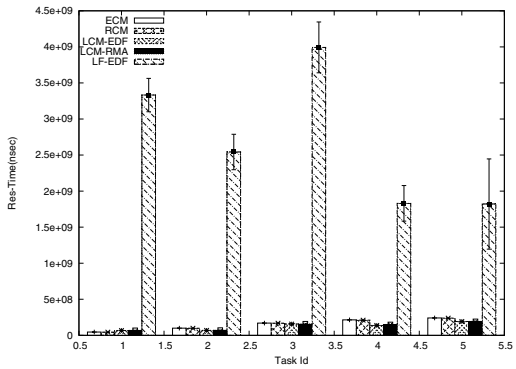
(c) Task set 3

Figure 3: Task retry costs under LCM and competitor synchronization methods

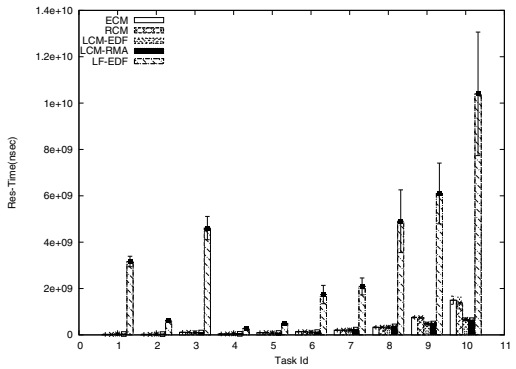
Figure 4 shows the response time of each task in the Table 1 task sets with a confidence level of 0.95. (Again, each task's atomic section length is equal to half of the task length.) We observe that G-EDF/LCM and G-RMA/LCM achieve shorter response time than the retry-loop lock-free algorithm, and shorter or comparable response time than ECM and RCM.

We repeated the experiments by varying three parameters: the relative total length of all atomic sections to the length of the task, the maximum relative length of any atomic section to the length of the task, and the minimum relative length of any atomic section to the length of the task. Full set of

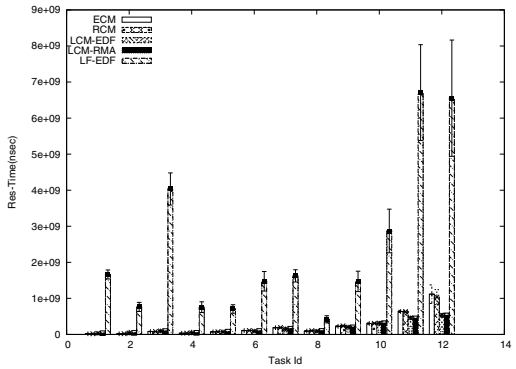
results are omitted here due to space constraints; however additional results are included in the Supplementary Material section. Full set of results are given in Appendix B in [6].



(a) Task set 1



(b) Task set 2



(c) Task set 3

Figure 4: Task response times under LCM and competitor synchronization methods

6. CONCLUSIONS

In ECM and RCM, a task incurs at most $2s_{max}$ retry cost for each of its atomic section 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, whereas in LCM, they do so. In G-EDF/LCM, retry due to a lower priority job is encountered only from a task τ_j's last job 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 job instance of lower priority tasks. Schedulability of G-EDF/LCM and G-RMA/LCM is 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 synchronization for G-EDF if the upper bound on s_{max}/r_{max} is between 0.5 and 2, which is higher than that achieved by ECM.

Acknowledgments

This work is supported in part by NSF CNS 0915895, NSF CNS 1116190, and NSF CNS 1130180.

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. El-Shambakey and B. Ravindran. On the design of real-time stm contention managers. Technical report, ECE Department, Virginia Tech, 2011. Available as `http://www.real-time.ece.vt.edu/tech-report-rt-stm-cm11.pdf`.

[7] M. Elshambakey and B. Ravindran. Stm concurrency control for multicore embedded real-time software: Time bounds and tradeoffs. In *SAC*, 2012.

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

[9] 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.

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

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

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

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

[14] 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.

[15] 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.

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

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

Supplementary Material

This section includes proofs of all Claims.

S.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 $s_i^k(\theta)$ is $\Upsilon len(s_i^k(\theta)) + len(s_j^l(\theta))$, 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. □

S.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 $s_i^k(\theta)$. □

S.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 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 section. Claim follows. □

S.4 Proof of Claim 4

PROOF. Assume 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 $len(s_i^k(\theta)) > len(s_h^z(\theta))$. Now, $\alpha_{ij}^{kl} > \alpha_{hj}^{zl}$ and $c_{ij}^{kl} < c_{hj}^{zl}$. By applying (4) to obtain 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. □

S.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 and τ_h^p coincides with the absolute deadline of τ_i^x.

By using Claims 1, 2, 3, and 4, and Claim 1 in [7] to determine the effect of atomic sections belonging to higher and lower priority instances of interfering tasks to τ_i^x, claim follows. □

S.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 [7] and (5) are replaced by s_{max}. Let α_{max}^{hl} in (5) be replaced with α_{max}, and α_{max}^{iy} in (5) be replaced with α_{min}. As α_{max}, α_{min}, and $len(s_{max})$ are all constants, (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 $\beta_1^{ih} = \sum_{\forall \theta \in (\theta_i \wedge \theta_h)} \sum_{\forall s_h^z(\theta)}$. Furthermore, 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:

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

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

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

(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. □

S.7 Proof of Claim 7

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

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

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:

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

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

Let the number of tasks that have shared objects with τ_i be ω (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). 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 maximum values for the upper bound on s_{max}/r_{max}, we consider the following cases:

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

$\therefore$ (16) will be:

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

By substituting the edge values for ω and $\left\lceil \frac{T_i}{T_h} \right\rceil$ in (17), we derive that the upper bound on s_{max}/r_{max} lies between 1 and 2.

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

(16) becomes

$$\frac{s_{max}}{r_{max}} \leq 0.5 + \frac{\sum_{\forall \tau_i} \frac{\omega - 0.5}{T_i}}{\sum_{\forall \tau_i} \frac{1 + 2 \sum_{\forall \tau_h \in \gamma_i} \left\lceil \frac{T_i}{T_h} \right\rceil}{T_i}} \quad (18)$$

By applying the edge values for ω and $\left\lceil \frac{T_i}{T_h} \right\rceil$ in (18), we derive that the upper bound on s_{max}/r_{max} lies between 0.5 and 1.

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

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

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

$\therefore$ (16) becomes:

$$\frac{s_{max}}{r_{max}} \leq 0.5 + \frac{\sum_{\tau_i} \frac{\omega}{T_i}}{2 \sum_{\tau_i} \frac{\sum_{\forall \tau_h \in \gamma_i} \left\lceil \frac{T_i}{T_h} \right\rceil}{T_i}} \quad (19)$$

By applying the edge values for ω and $\left\lceil \frac{T_i}{T_h} \right\rceil$ in (19), we derive that the upper bound on s_{max}/r_{max} lies between 0.5 and 1, which is similar to that achieved by ECM.

Summarizing from the previous cases, the upper bound on s_{max}/r_{max} lies between 0.5 and 2, whereas for ECM [7], it lies between 0.5 and 1. Claim follows.

□

S.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) can 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. Claims 2 and 3 can 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 [7], the claim follows. □

S.9 Proof of Claim 9

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

$$RC(T_i) \leq \sum_{\forall \tau_j^*} \Bigg(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) (1+\alpha_{max}) len(s_{max}) \beta_i \Bigg) + (1-\alpha_{min}) len(s_{max}) \beta_i \quad (20)$$

For RCM, (16) in [7] for $RC(T_i)$ is upper bounded by:

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

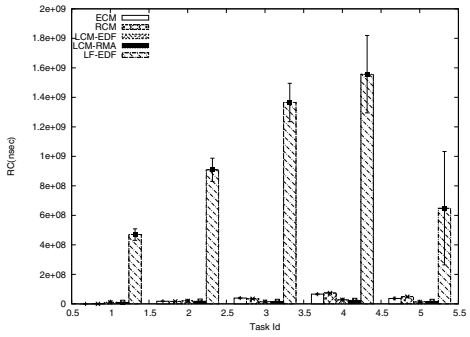
By comparing the total utilization of G-RMA/LCM with that of RCM, we get:

$$\sum_{\forall \tau_i} \frac{len(s_{max}) \beta_i \left((1-\alpha_{min}) + \sum_{\forall \tau_j^*} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) (1+\alpha_{max}) \right) \right)}{T_i} \leq \sum_{\forall \tau_i} \frac{2 len(s_{max}) \beta_i \sum_{\forall \tau_j^*} \left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right)}{T_i} \quad (21)$$

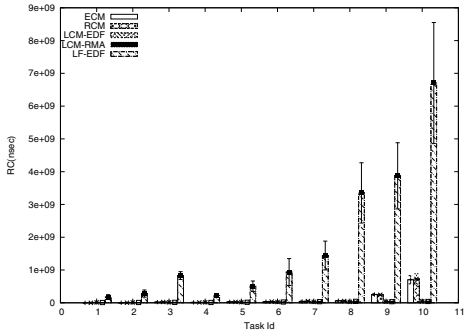
(21) is satisfied if $\forall \tau_i$ (9) is satisfied. Claim follows. □

S.10 EXTENDED RESULTS

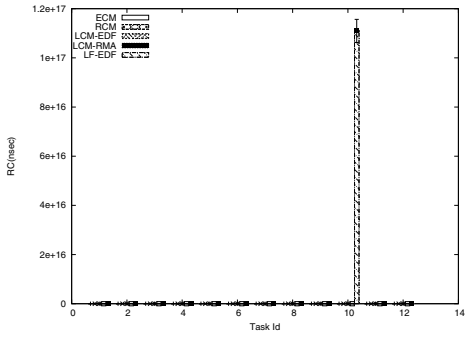
The three parameters x, y, z for each figure specify respectively the relative total length of all atomic sections to the length of the task, the maximum relative length of any atomic section to the length of the task, and the minimum relative length of any atomic section to the length of the task.



(a) Task set 1

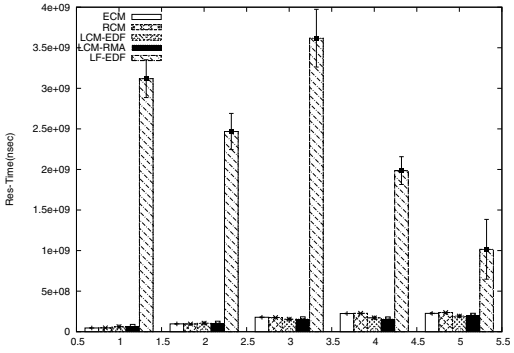


(b) Task set 2

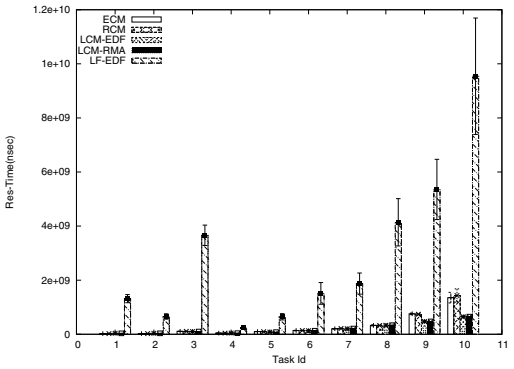


(c) Task set 3

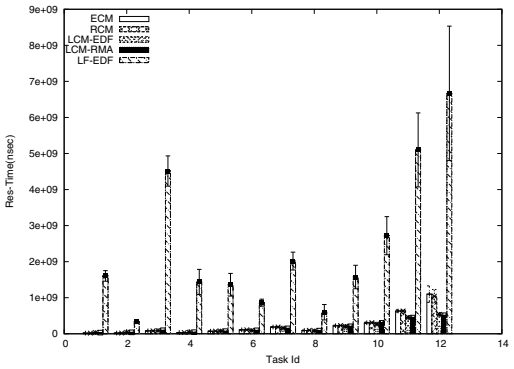
Figure 5: Task retry costs under LCM and competitor synchronization methods (0.5,0.2,0.2)



(a) Task set 1

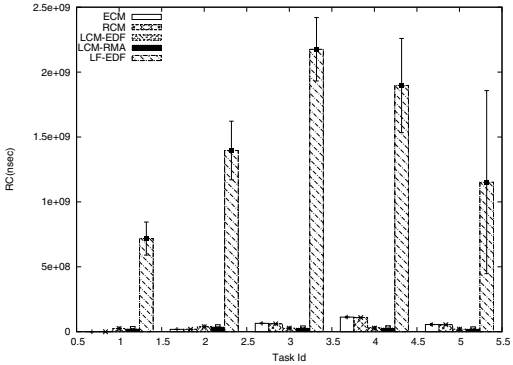


(b) Task set 2

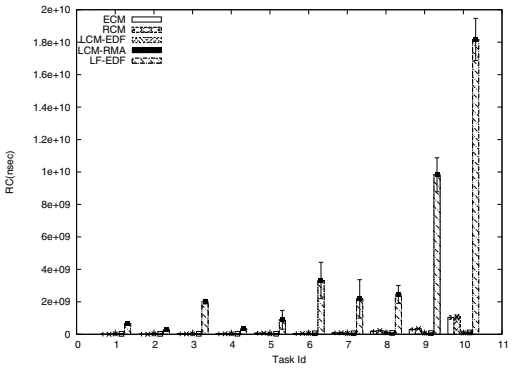


(c) Task set 3

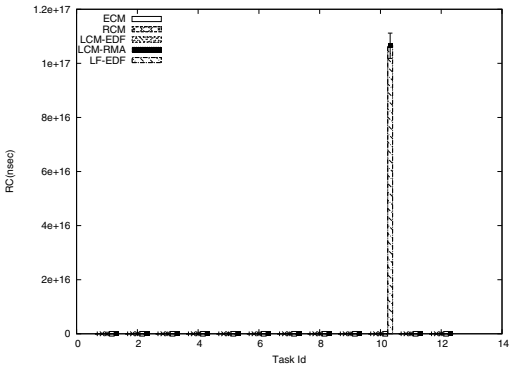
Figure 6: Task response times under LCM and competitor synchronization methods (0.5,0.2,0.2)



(a) Task set 1

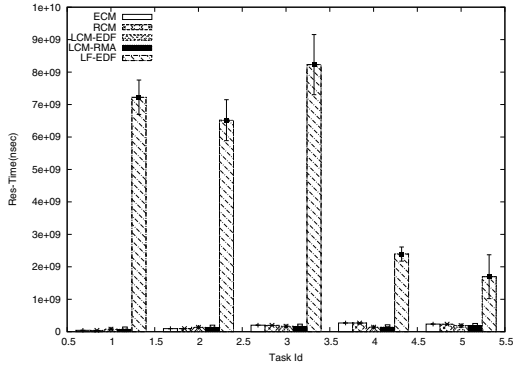


(b) Task set 2

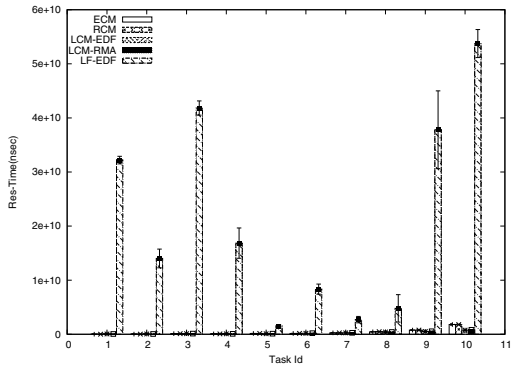


(c) Task set 3

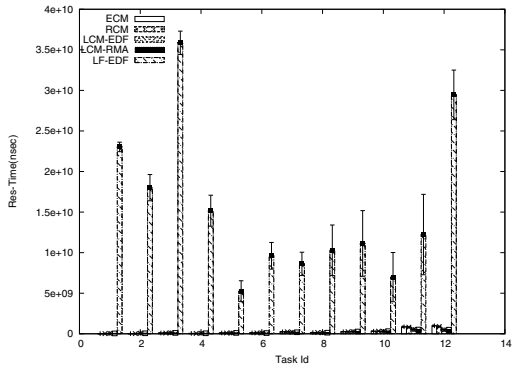
Figure 7: Task retry costs under LCM and competitor synchronization methods (0.8,0.5,0.2)



(a) Task set 1



(b) Task set 2



(c) Task set 3

Figure 8: Task response times under LCM and competitor synchronization methods (0.8,0.5,0.2)