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.

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 time-liness properties. This result thus allows programmers to reap STM’s significant programmability and composability benefits for a broader range of multicore embedded real-time software than what was previously possible.

2. RELATED WORK

Transactional-like concurrency control without using locks, for real-time systems, has been previously studied in the context of non-blocking data structures (e.g., [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.
3. PRELIMINARIES

We consider a multiprocessor system with $m$ identical processors and $n$ sporadic tasks $t_1, t_2, \ldots, t_n$. The $k^{th}$ instance (or job) of a task $t_i$ is denoted $t_i^k$. Each task $t_i$ 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 $t_i^j$ is released at time $d_i^j$ 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 $t_i$’s (fixed) priority and it is constant for all instances of $t_i$. Under a dynamic priority scheduler such as G-EDF, a job $t_i$’s priority, $p_i$, differs from one instance to another. A task $t_i$ may interfere with task $t_j$ 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 $t_i$ is denoted $s_i$. $s_i^k$ is the $k^{th}$ atomic section of $t_i$. Each object, $\theta$, can be accessed by multiple tasks. The set of distinct objects accessed by $t_i$, to access $\theta$ is $s_i(\theta)$, and the sum of the lengths of those atomic sections is $\text{len}(s_i(\theta))$. $s_i^k(\theta)$ is the $k^{th}$ atomic section of $t_i$ that accesses $\theta$. $s_i^k(\theta)$ executes for a duration $\text{len}(s_i^k(\theta))$. The set of tasks sharing $\theta$ with $t_i$ 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 $\gamma_i$, that accesses $\theta$ is denoted $s_{\max}(\theta)$, while the maximum one among all tasks is $s_{\max}(\theta)$, and the maximum one among tasks with priorities lower than that of $t_i$ is $s_{\max}(\theta)$.

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 $s_i^k(\theta)$ will take to execute due to a conflict with another section $s_j^l(\theta)$, is denoted $W_i^j(s_i^k(\theta))$. If an atomic section, $s_i^k$, is already executing, and another atomic section $s_i^l$ tries to access a shared object with $s_i^k$, then $s_j^l$ is said to “interfere” or “conflict” with $s_i^k$. The job $s_i^l$ is the “interfering job”, and the job $s_i^k$ is the “interfered job.” The total time that a task $t_i$’s atomic sections have to retry over $T_i$ is denoted $RC(T_i)$. The additional amount of time that a task $t_i$ causes to response time of $t_i$ when interfering with $t_j$ 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

Algorithm 1: LCM

Data: $s_i^k(\theta) \rightarrow$ interfered atomic section.
$s_i^j(\theta) \rightarrow$ 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 $s_i^l(\theta)$ aborts

1. if $p_i^k > p_j^l$ then
2. \quad $s_i^j(\theta)$ aborts;
3. else
4. \quad $\alpha_{ij}^k = \text{len}(s_i^j(\theta))/\text{len}(s_i^k(\theta))$;
5. \quad $\alpha_{ij}^i = \text{len}(\psi)/\text{len}(\psi - \epsilon_i^j)$;
6. \quad $\alpha = (\text{len}(s_i^k(\theta)) - \delta_i^k(\theta))/\text{len}(s_i^k(\theta))$;
7. if $\alpha \leq \alpha_{ij}^k$ then
8. \quad $s_i^k(\theta)$ aborts;
9. else
10. \quad $s_i^l(\theta)$ aborts;
11. end
12. end

For both ECM and RCM, $s_i^k(\theta)$ can be totally repeated if $s_i^j(\theta)$ — which belongs to a higher priority job $r_j^l$ than
\( \tau^a_i \) — conflicts with \( s_j^k(\theta) \) at the end of its execution, while \( s_j^k(\theta) \) is just about to commit. Thus, LCM, shown in Algorithm 1, uses the remaining length of \( s_j^k(\theta) \) when it is interfered, as well as \( \text{len}(s_j^k(\theta)) \), to decide which transaction must be aborted. If \( p_k^i \) was greater than \( p_j^i \), then \( s_j^k(\theta) \) would be the one that commits, because it belongs to a higher priority job, and it started before \( s_j^k(\theta) \) (step 2). Otherwise, \( c_{ij}^{kl} \) is calculated (step 4) to determine whether it is worth aborting \( s_j^k(\theta) \) in favor of \( s_j^k(\theta) \), because \( \text{len}(s_j^k(\theta)) \) is relatively small compared to the remaining execution length of \( s_j^k(\theta) \) (explained further).

We assume that:

\[
\hat{c}_{ij}^{kl} = \frac{\text{len}(s_j^k(\theta))}{\text{len}(s_j^k(\theta))} \tag{1}
\]

where \( \hat{c}_{ij}^{kl} \in [0, \infty], \) to cover all possible lengths of \( s_j^k(\theta) \). Our idea is to reduce the opportunity for the abort of \( s_j^k(\theta) \) if it is close to committing when interfered and \( \text{len}(s_j^k(\theta)) \) is large. This abort opportunity is increasingly reduced as \( s_j^k(\theta) \) gets closer to the end of its execution, or \( \text{len}(s_j^k(\theta)) \) gets larger.

On the other hand, as \( s_j^k(\theta) \) is interfered early, or \( \text{len}(s_j^k(\theta)) \) is small compared to \( s_j^k(\theta) \)’s remaining length, the abort opportunity is increased even if \( s_j^k(\theta) \) is close to the end of its execution. To decide whether \( s_j^k(\theta) \) must be aborted or not, we use a threshold value \( \psi \in [0, 1] \) that determines \( \alpha_{ij}^{kl} \) (step 5), where \( \alpha_{ij}^{kl} \) is the maximum percentage of \( \text{len}(s_j^k(\theta)) \) below which \( s_j^k(\theta) \) is allowed to abort \( s_j^k(\theta) \). Thus, if the already executed part of \( s_j^k(\theta) \) — when \( s_j^k(\theta) \) interferes with \( s_j^k(\theta) \) — does not exceed \( \alpha_{ij}^{kl} \text{len}(s_j^k(\theta)) \), then \( s_j^k(\theta) \) is aborted (step 8). Otherwise, \( s_j^k(\theta) \) is aborted (step 10).

\[
f(c_{ij}^{kl}, \alpha) = e^{-\frac{\hat{c}_{ij}^{kl} \alpha}{1 - \alpha}} \tag{2}
\]

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

Figure 1 shows one atomic section \( s_j^k(\theta) \) (whose \( \alpha \) changes along the horizontal axis) interfered by five different lengths of \( s_j^k(\theta) \). For a predefined value of \( f(c_{ij}^{kl}, \alpha) \) (denoted as \( \psi \) in Algorithm 1), there corresponds a specific value of \( \alpha \) (which is \( \alpha_{ij}^{kl} \) in Algorithm 1) for each curve. For example, when \( \text{len}(s_j^k(\theta)) = 0.1 \times \text{len}(s_j^k(\alpha)) \), \( s_j^k(\theta) \) aborts \( s_j^k(\theta) \) if the latter has not executed more than \( \alpha_3 \) percentage (shown in Figure 1) of its execution length. As \( \text{len}(s_j^k(\theta)) \) decreases, the corresponding \( \alpha_{ij}^{kl} \) 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 \( s_j^k(\theta) \) gets closer to the end of its execution (as \( \alpha \to 1, f(c_{ij}^{kl}, 1) \to 0 \), or as the length of the conflicting transaction increases (as \( c_{ij}^{kl} \to \infty, f(\infty, \alpha) \to 0 \)). Meanwhile, this abort opportunity is increased as \( s_j^k(\theta) \) is interfered closer to its release (as \( \alpha \to 0, f(0, \alpha) \to 1 \)), or as the length of the conflicting transaction decreases (as \( c_{ij}^{kl} \to 0, f(0, \alpha) \to 1 \)).

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

**Claim 1.** Let \( s_j^k(\theta) \) interfere once with \( s_j^k(\theta) \) at \( \alpha_{ij}^{kl} \). Then, the maximum contribution of \( s_j^k(\theta) \) to \( s_j^k(\theta) \)’s retry cost is:

\[
W_i^k(s_j^k(\theta)) \leq \alpha_{ij}^{kl} \text{len}(s_j^k(\theta)) + \text{len}(s_j^k(\theta)) \tag{3}
\]

(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, \( \tau^b_j \), may have to abort and retry due to a lower priority job, \( \tau^a_i \), if \( s_j^k(\theta) \) interferes with \( s_j^k(\theta) \) after the \( \alpha_{ij}^{kl} \) percentage. \( \tau^b_j \)’s retry time, due to \( s_j^k(\theta) \) and \( s_j^k(\theta) \), is upper bounded by:

\[
W_i^b(s_j^k(\theta)) \leq \left( 1 - \alpha_{ij}^{kl} \right) \text{len}(s_j^k(\theta)) \tag{4}
\]

**Claim 3.** Under LCM, there is no priority inversion.

**Claim 4.** A higher priority job, \( \tau^b_j \), can be delayed by lower priority jobs for at most number of atomic sections in \( \tau^a_i \).

**Claim 5.** The maximum delay suffered by \( s_j^k(\theta) \) due to priority inversion is caused by the maximum length atomic section accessing object \( \theta \), which belongs to a lower priority job than \( s_j^k(\theta) \).

4.2 Response Time of G-EDF/LCM

**Claim 6.** \( RC(T_i) \) for a task \( \tau_i \) under G-EDF/LCM is upper bounded by:

\[
RC(T_i) = \left( \sum_{\forall \tau_k \in C_i, \forall \theta \in \theta_i, \forall \text{h}} \left[ \frac{T_i}{T_h} \sum_{\forall \text{h}(\theta)} \text{len}(s_h(\theta)) + \alpha_{\text{max}}^{kl} \text{len}(s_{\text{max}}(\theta)) \right] \right)
+ \sum_{\forall \text{h}(\theta)} \left( 1 - \alpha_{\text{max}}^{\psi} \right) \text{len}(s_{\text{max}}(\theta)) \tag{5}
\]

where \( \alpha_{\text{max}}^{kl} \) is the \( \alpha \) value that corresponds to \( \psi \) due to the interference of \( s_{\text{max}}(\theta) \) by \( s_j^k(\theta) \), \( \alpha_{\text{max}}^{\psi} \) is the \( \alpha \) value that corresponds to \( \psi \) due to the interference of \( s_{\text{max}}(\theta) \) by \( s_j^k(\theta) \).

Response time of \( \tau_i \) 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 $\tau_i$ by adding the retry cost suffered by $\tau_i$. 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:

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

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

CLAIM 7. Let $s_{max}$ be the maximum length atomic section accessing any object $\theta$. Let $\alpha_{max}$ and $\alpha_{min}$ be the maximum and minimum values of $\alpha$ for any two atomic sections $s^i_1(\theta)$ and $s^i_2(\theta)$. Given a threshold $\psi$, schedulability of G-EDF/LCM is equal or better than ECM if for any task $\tau_i$:

$$1 - \alpha_{min} \over 1 - \alpha_{max} \leq \sum_{\forall \tau_i} \left[ T_i \over T_h \right]$$

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 8. Let $s_{max}$ denote $len(s_{max})$ and $\tau_{max}$ 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 $s_{max}/\tau_{max}$ 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 9. Let

$$\lambda_j(\theta) = \sum_{\forall s^i_j(\theta)} len(s^i_j(\theta)) + \alpha_{max}^i len(s_{max}(\theta))$$

where $\alpha_{max}^i$ is the $\alpha$ value corresponding to $\psi$ due to the interference of $s_{max}(\theta)$ by $s^i_j(\theta)$. The retry cost of any task $\tau_j$ under G-RMA/LCM during $T_i$ is given by:

$$RC(\tau_j) = \sum_{\forall \tau_j(\theta)} \left( \sum_{\forall s^i_j(\theta)} \left( \left[ T_i \over T_j \right] + 1 \right) \lambda_j(\theta) \right)$$

$$+ \sum_{\forall s^i_j(\theta)} \left( 1 - \alpha_{max}^i \right) len(s^i_j(\theta))$$

where $\tau_j = \{ \tau_j(\theta) \in \tau_i \land (p_j > p_i) \}$.

The response time is calculated by (17) in [7] with replacing $RC(RSTM)$ with $RC(T_i)$.

4.6 Schedulability of G-RMA/LCM and RCM

CLAIM 10. Under the same assumptions of Claims 7 and 9, G-RMA/LCM’s schedulability is equal or better than RCM if:

$$1 - \alpha_{min} \over 1 - \alpha_{max} \leq \sum_{\forall \tau_j} \left( \left[ T_i \over T_j \right] + 1 \right)$$

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 0.0129653375μs, and the average time taken by one CAS-loop operation on any core is 0.0292546250 μs.

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 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
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

<table>
<thead>
<tr>
<th>Task set 1</th>
<th>Task set 2</th>
<th>Task set 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>T _i (μs)</td>
<td>c _i (μs)</td>
<td>T _i (μs)</td>
</tr>
<tr>
<td>500000</td>
<td>150000</td>
<td>400000</td>
</tr>
<tr>
<td>1000000</td>
<td>227000</td>
<td>750000</td>
</tr>
<tr>
<td>1500000</td>
<td>410000</td>
<td>1000000</td>
</tr>
<tr>
<td>3000000</td>
<td>290428</td>
<td>1500000</td>
</tr>
<tr>
<td>5000000</td>
<td>290428</td>
<td>20000000</td>
</tr>
</tbody>
</table>

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 suffer 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.

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].

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 $\tau_j$’s last job instance during $\tau_j$’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 $\alpha_{\min}$ and $\alpha_{\max}$. 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.

7. REFERENCES


Supplementary Material

This section includes proofs of all Claims.

S.1 Proof of Claim 1
S.2 Proof of Claim 2

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

S.3 Proof of Claim 3

Proof. Assume three atomic sections, $s_h^k(\theta)$, $s_i^k(\theta)$, and $s_j^k(\theta)$, where $p_j > p_i$ and $s_j^k(\theta)$ interferes with $s_i^k(\theta)$ after $\alpha_{ij}^k$, and $s_j^k(\theta)$ will have to abort. At this time, if $s_i^k(\theta)$ interferes with the other two atomic sections, then LCM decides which transaction to commit by comparing the two transactions. So, we have the following cases:

- $p_a < p_i < p_j$. Now, $s_i^k(\theta)$ will not abort any one because it is still in its beginning and it is of lower priority. Thus, $\tau_i$ is not indirectly blocked by $\tau_a$.
- $p_i < p_a < p_j$. Now, even if $s_i^k(\theta)$ interferes with $s_i^k(\theta)$ before $\alpha_{ia}$, and $s_h^k(\theta)$ is allowed to abort $s_i^k(\theta)$, after comparing $s_i^k(\theta)$ and $s_h^k(\theta)$, LCM will select $s_i^k(\theta)$ to commit and abort $s_h^k(\theta)$, because the latter is still at its beginning, and $\tau_i$ is of higher priority. If $s_h^k(\theta)$ is not allowed to abort $s_i^k(\theta)$, the situation is the same, because $s_i^k(\theta)$ was already retrying until $s_i^k(\theta)$ finishes. So, the medium priority instance of $\tau_a$ does not increase the delay time of the higher priority instance of $\tau_i$.
- $p_a > p_j > p_i$. Now, if $s_i^k(\theta)$ is chosen to commit, this will not cause a priority inversion for $\tau_i$ because $\tau_a$ is of higher priority.
- If $\tau_i$ preempts $\tau_i$, then LCM will compare only between $s_j^k(\theta)$ and $s_h^k(\theta)$. If $p_a < p_j$, then $s_j^k(\theta)$ will commit because of its task’s higher priority and $s_h^k(\theta)$ is still at its beginning. Otherwise, $s_j^k(\theta)$ will retry, but this will not be priority inversion, because $\tau_a$ is already of higher priority than $\tau_i$. If $\tau_a$ does not access any object but it preempts $\tau_i$, then LCM will choose $s_j^k(\theta)$ to commit, since only already running transactions are competing together.

From the previous cases, it appears that priority inversion can never happen under LCM. Claim follows. \qed

S.4 Proof of Claim 4

Proof. By generalizing the cases mentioned in the proof of Claim 3 to any number of conflicting jobs, it can be seen that when an atomic section, $s_j^k(\theta)$, of a higher priority job conflicts with a number of atomic sections belonging to lower priority jobs, $s_i^k(\theta)$ can be delayed only one of them, $s_j^k(\theta)$, for the remaining length of $s_i^k(\theta)$, then $s_j^k(\theta)$ can run. Thus, the worst case scenario happens when each atomic section in the higher priority job is delayed by one of the atomic sections belonging to lower priority jobs. Claim follows. \qed

S.5 Proof of Claim 5

Proof. Assume three atomic sections, $s_h^k(\theta)$, $s_i^k(\theta)$, and $s_j^k(\theta)$, where $p_j > p_i$, $p_j > p_i$, and $\text{len}(s_i^k(\theta)) > \text{len}(s_h^k(\theta))$.

Now, $\alpha_{ij}^k > \alpha_{ij}^k$ and $\alpha_{ij}^k < c_{ij}^k$. By applying (4) to obtain the contribution of $s_h^k(\theta)$ and $s_i^k(\theta)$ to the priority inversion of $s_j^k(\theta)$ and dividing them, we get:

$$\frac{W_1(s_h^k(\theta))}{W_2(s_i^k(\theta))} = \frac{(1 - \alpha_{ij}^k) \text{len}(s_i^k(\theta))}{(1 - \alpha_{ij}^k) \text{len}(s_h^k(\theta))}$$

By substitution for $\alpha$ from (2):

$$\frac{W_1(s_h^k(\theta))}{W_2(s_i^k(\theta))} = \frac{(1 - \frac{\text{len}(s_i^k(\theta))}{\text{len}(s_h^k(\theta))}) \text{len}(s_i^k(\theta))}{(1 - \frac{\text{len}(s_i^k(\theta))}{\text{len}(s_h^k(\theta))}) \text{len}(s_h^k(\theta))}$$

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

S.6 Proof of Claim 6

Proof. The maximum number of higher priority instances of $\tau_h$ that can interfere with $\tau_i^k$ is $\frac{T_i}{T_h}$, as shown in Figure 2, where one instance of $\tau_h$ and $\tau_i^k$ coincides with the absolute deadline of $\tau_i^k$.

By using Claims 1, 2, 4, and 5, and Claim 1 in [7] to determine the effect of atomic sections belonging to higher and lower priority instances of interfering tasks to $\tau_i^k$, claim follows. \qed

S.7 Proof of Claim 7

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

$$RC(T_i) \leq \sum_{\forall \tau_h \in \tau_i} \sum_{\forall \theta \in \theta_h} \left[ \frac{T_i}{T_h} \sum_{\forall s_i^k(\theta)} 2\text{len}(s_{\max}) \right]$$

with the assumption that all lengths of atomic sections of (4) and (8) in [7] and (5) are replaced by $s_{\max}$. Let $\alpha_{\max}$ be replaced with $\alpha_{\max}$, and $\alpha_{\max}$ in (5) be replaced with $\alpha_{\min}$. As $\alpha_{\max}$, $\alpha_{\min}$, and $\text{len}(s_{\max})$ are all constants, (5) is upper bounded by:

$$RC(T_i) \leq \left( \sum_{\forall \tau_h \in \tau_i} \sum_{\forall \theta \in \theta_h} \left( \frac{T_i}{T_h} \sum_{\forall s_i^k(\theta)} (1 + \alpha_{\max}) \text{len}(s_{\max}) \right) + \sum_{\forall s_i^k(\theta)} (1 - \alpha_{\min}) \text{len}(s_{\max}) \right)$$

If $\beta_1^k$ is the total number of times any instance of $\tau_h$ accesses shared objects with $\tau_i$, and $\beta_2^k = \sum_{\forall \theta \in \theta_h} (1 - \alpha_{\min}) \sum_{\forall s_i^k(\theta)}$. Furthermore, if $\beta_2^k$ is the total number of times any instance of $\tau_i$ accesses shared objects with any other instance, then $\beta_2^k = \sum_{\forall \theta \in \theta_i} (1 - \alpha_{\min}) \sum_{\forall s_i^k(\theta)}$. Then, $\beta = \max\{\beta_1^k, \beta_2^k\}$ is the maximum number of accesses to all shared objects by any instance of $\tau_i$ or $\tau_h$. \qed
Thus, (10) becomes:

\[
RC(T_i) \leq \sum_{\forall \tau_i \in \gamma_i} \left( \frac{T_i}{T_h} \right) \beta_i \text{len}(s_{max})
\]

(12)

and (11) becomes:

\[
RC(T_i) \leq \beta_i \text{len}(s_{max}) \left( 1 - \alpha_{\min} \right) + \sum_{\forall \tau_h \in \gamma_i} \left( \frac{T_i}{T_h} \right) \left( 1 + \alpha_{\max} \right)
\]

(13)

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

\[
\sum_{\forall \tau_i} \left( 1 - \alpha_{\min} \right) + \sum_{\forall \tau_h \in \gamma_i} \left( \frac{T_i}{T_h} \right) \left( 1 + \alpha_{\max} \right) \leq \sum_{\forall \tau_h \in \gamma_i} 2 \left( \frac{T_i}{T_h} \right)
\]

(14)

is satisfied if for each \( \tau_i \), the following condition is satisfied:

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

\[
1 - \alpha_{\min} \leq \sum_{\forall \tau_h \in \gamma_i} \left( \frac{T_i}{T_h} \right)
\]

Claim follows. \( \Box \)

### S.8 Proof of Claim 8

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

\[
RL(T_i) = \sum_{\forall \tau_i \in \gamma_i} \left( \frac{T_i}{T_h} \right) \left( 1 + \alpha_{\min} \right) \beta_i \text{len}(T_i) \]

(15)

where \( \beta_i \) is as defined in Claim 7. The retry cost of \( \tau_i \) 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:

\[
\sum_{\forall \tau_i} \left( 1 - \alpha_{\min} \right) + \sum_{\forall \tau_h \in \gamma_i} \left( \frac{T_i}{T_h} \right) \left( 1 + \alpha_{\max} \right) \beta_i \text{len}(s_{max}) \\
\leq \sum_{\forall \tau_i} \sum_{\forall \tau_h \in \gamma_i} \left( \frac{T_i}{T_h} \right) \left( 1 + \alpha_{\max} \right) \beta_i \text{len}(s_{max})
\]

\[
\therefore \frac{s_{max}}{r_{max}} \leq \sum_{\forall \tau_i} \frac{1}{\sum_{\forall \tau_h \in \gamma_i} \left( \frac{T_i}{T_h} \right) \left( 1 + \alpha_{\max} \right) \beta_i}
\]

(16)

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

- \( \alpha_{\min} \rightarrow 0, \alpha_{\max} \rightarrow 0 \)

\[
\therefore (16) \text{ will be:}
\]

\[
\frac{s_{max}}{r_{max}} \leq 1 + \frac{\sum_{\forall \tau_i} \omega \frac{1}{\omega_i}}{1 + \sum_{\forall \tau_h \in \gamma_i} \left( \frac{T_i}{T_h} \right) \left( 1 + \alpha_{\max} \right) \beta_i}
\]

(17)

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

- \( \alpha_{\min} \rightarrow 0, \alpha_{\max} \rightarrow 1 \)

\[
(16) \text{ becomes}
\]

\[
\frac{s_{max}}{r_{max}} \leq 0.5 + \frac{\sum_{\forall \tau_i} \omega \frac{0.5}{\omega_i}}{2 + \sum_{\forall \tau_h \in \gamma_i} \left( \frac{T_i}{T_h} \right) \left( 1 + \alpha_{\max} \right) \beta_i}
\]

(18)

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

- \( \alpha_{\min} \rightarrow 1, \alpha_{\max} \rightarrow 0 \)

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

- \( \alpha_{\min} \rightarrow 1, \alpha_{\max} \rightarrow 1 \)

\[
(16) \text{ becomes:}
\]

\[
\frac{s_{max}}{r_{max}} \leq 0.5 + \frac{\sum_{\forall \tau_i} \frac{\omega}{\omega_i}}{2 + \sum_{\forall \tau_h \in \gamma_i} \left( \frac{T_i}{T_h} \right) \left( 1 + \alpha_{\max} \right) \beta_i}
\]

(19)

By applying the edge values for \( \omega \) and \( \left( \frac{T_i}{T_h} \right) \) 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. \( \Box \)

### S.9 Proof of Claim 9

**Proof.** Under G-RMA, all instances of a higher priority task, \( \tau_j \), can conflict with a lower priority task, \( \tau_i \), during \( T_i \). (3) can be used to determine the contribution of each conflicting atomic section in \( \tau_j \) to \( \tau_i \). Meanwhile, all instances of any task with lower priority than \( \tau_i \) can conflict with \( \tau_i \) during \( T_i \). Claims 2 and 4 can be used to determine the contribution of conflicting atomic sections in lower priority tasks to \( \tau_i \). Using the previous notations and Claim 3 in [7], the claim follows. \( \Box \)

### S.10 Proof of Claim 10

**Proof.** Under the same assumptions as that of Claims 7 and 9, (8) can be upper bounded as:

\[
RC(T_i) \leq \sum_{\forall \tau_j} \left( \left( \frac{T_i}{T_j} \right) + 1 \right) (1 + \alpha_{\max}) \text{len}(s_{max}) \beta_i
\]

\[
+ (1 - \alpha_{\min}) \text{len}(s_{max}) \beta_i
\]

(20)

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

\[
RC(T_i) \leq \sum_{\forall \tau_j} \left( \left( \frac{T_i}{T_j} \right) + 1 \right) 2 \beta_i \text{len}(s_{max})
\]
By comparing the total utilization of G-RMA/LCM with that of RCM, we get:

$$
\sum_{\forall \tau_i} \frac{1}{\beta_i} \left( \left( \frac{T_i}{T} \right) + 1 \right) \left( 1 + \alpha_{max} \right)
\leq \sum_{\forall \tau_i} \frac{2\left( \sum_{\forall \tau_i} \left( \frac{T_i}{T} \right) + 1 \right)}{T_i}
$$

(21)

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

S.11 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.

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)

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