On the design of real-time STM contention managers

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

Abstract

We consider software transactional memory (STM) concurrency control in multicore embedded real-time software. We investigate real-time contention managers (CMs) for resolving transactional conflicts, including those based on dynamic and fixed priorities, and establish upper bounds on transactional retries and task response times. We identify the conditions under which STM (with the proposed CMs) is superior to lock-based and lock-free synchronization.

1 Introduction

Embedded systems sense physical processes and control their behavior, typically through feedback loops. Since physical processes are concurrent, computations that control them must also be concurrent, enabling them to process multiple streams of sensor input and control multiple actuators, all concurrently. Often, such computations need to concurrently read/write shared data objects. Typically, they must also process sensor input and react in a timely manner.

The de facto standard for programming concurrency is the threads abstraction, and the de facto synchronization abstraction is locks. Lock-based concurrency control has significant programmability, scalability, and compositionality challenges [24]. Transactional memory (TM) is an alternative synchronization model for shared in-memory data objects that promises to alleviate these difficulties. With TM, programmers write concurrent code using threads, but 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) [22] resolves the conflict by aborting one and allowing the other to proceed to commit, yielding (the illusion of) atomicity. Aborted transactions are re-started, often immediately. In addition to a simple programming model, TM provides performance comparable or superior to highly concurrent fine-grained locking and lock-free approaches [34], and is composable [23]. Multiprocessor TM has been proposed in hardware, called HTM (e.g., [31]), and in software, called STM (e.g., [38]), with the usual tradeoffs: HTM provides strong atomicity [31], has lesser overhead, but needs transactional support in hardware; STM is available on any hardware.

Given STM's programmability, scalability, and compositionality advantages,

we consider it for concurrency control in multicore embedded real-time software. Doing so will require bounding transactional retries, as real-time threads, which subsume transactions, must satisfy time constraints. Retry bounds in STM are dependent on the CM policy at hand (analogous to the way thread response time bounds are scheduler-dependent). Thus, real-time CM is logical.

Designing a real-time CM is straightforward. Transactional contention can be resolved using dynamic or fixed priorities of parent threads, resulting in Earliest-Deadline-First (EDF) CM or Rate Monotonic Assignment (RMA)-based CM, respectively. But what upper bounds exist for transactional retries and thread response times under such CMs and respective multicore real-time schedulers, global EDF (G-EDF) and global RMA (G-RMA)? How does real-time STM compare against locking and lock-free protocols? i.e., are there upper or lower bounds for transaction lengths below or above which is STM superior to locking/lock-free?

We answer these questions. We consider EDF and RMA CMs, and establish their retry and response time upper bounds, and the conditions under which they outperform locking and lock-free protocols. Our work reveals a key result: for most cases, for G-EDF/EDF CM and G-RMA/RMA CM to be better or as good as lock-free, the atomic section length under STM must not exceed half of the lock-free retry loop-length. However, in some cases, for G-EDF/EDF CM, the atomic section length can reach the lock-free retry loop-length, and for G-RMA/RMA CM, it can even be larger than the lock-free retry loop-length. This means that, STM is more advantageous with G-RMA than with G-EDF. These results, among others, for the first time, provide a fundamental understanding of when to use, and not use, STM concurrency control in multicore embedded real-time software, and constitute the paper's contribution.

We overview past and related efforts in Section 2. Section 3 outlines the work's preliminaries. Sections 4 and 5 establish response time bounds under G-EDF/EDF CM and G-RMA/RMA CM, respectively. We consider the FMLP [5] and OMLP [7] protocols as the best locking competitors to STM, given their superiority, and bound their blocking times in Section 6. We compare STM against locking and lock-free approaches in Section 7. We present the idea of FIFO CM and response time of G-EDF/FIFO CM and G-RMA/FIFO CM in Section 8. We conclude in Section 19.

We present a novel CM that can be used with both dynamic and fixed priority (global) multicore real-time schedulers: length-based CM or LCM (Section 9.1). LCM tries to reduce retry cost than ECM and RCM. 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 9.2) and with G-RMA (Section 9.5) schedulers. We identify the conditions under which G-EDF/LCM outperforms ECM (Section 9.3) and lock-free synchronization (Section 9.4). We also identify the conditions under which G-RMA/LCM outperforms RCM (Section 9.6) and lock-free (Section 9.7). We implement the previous synchronization techniques in the Rochester STM framework [30] and conduct experimental studies (Section 18). Our study reveals that G-EDF/LCM and G-RMA/LCM have shorter or comparable retry costs and response times than competitors.

To allow multiple objects per transaction, we design a novel contention manager called PNF (Section 11), which can be used with global EDF (G-EDF) and

global RMA (G-RMA) multicore real-time schedulers. We upper bound transactional retry costs and task response times (Section 12), and formally compare PNF's schedulability with ECM, RCM, LCM, and lock-free synchronization (Section 13). We show that PNF achieves better schedulability than lock-free synchronization for larger atomic section length range than ECM and RCM. Our implementation reveals that PNF yields shorter or comparable retry costs than competitors in case of transitive retry (Section 18).

PNF requires a-priori knowledge of all objects accessed by each transaction. This significantly limits programmability, and is incompatible with dynamic STM implementations [25]. Additionally, PNF is a centralized CM, which increases overheads and retry costs, and has a complex implementation.

We propose the First Bounded, Last Timestamp (or FBLT) contention manager (Section 15). In contrast to PNF, FBLT does not require a-priori knowledge of objects accessed by transactions. Moreover, FBLT allows each transaction to access multiple objects with shorter transitive retry cost than ECM, RCM and LCM. Additionally, FBLT is a decentralized CM and does not use locks in its implementation. Implementation of FBLT is also simpler than PNF.

We establish FBLT's retry and response time upper bounds under G-EDF and G-RMA schedulers (Section 16). We also identify the conditions under which FBLT's schedulability is better than ECM, RCM, G-EDF/LCM, G-RMA/LCM, PNF, and lock-free synchronization (Section 17).

We implement FBLT and competitor CM techniques in the Rochester STM framework [30] and conduct experimental studies (Section 18). Our results reveal that FBLT has shorter retry cost than ECM, RCM, LCM and lock-free. FBLT's retry cost is slighter higher than PNF in case of transitive retry, but it doesn't require a-priori knowledge of objects accessed by transactions, unlike PNF. In case of non-transitive retry, FBLT acheives shorter or comparable retry cost than other synchronization techniques, including PNF.

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 [8]. In contrast, STM is semantically simpler [24], and is often the only viable lock-free solution for complex data structures (e.g., red/black tree) [19] and nested critical sections [34]. (The relationship between lock-free and STM is similar to that between programmer-controlled memory management and garbage collection.)

STM concurrency control for real-time systems has been previously studied in [3, 18, 19, 29, 35, 36].

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

[18] 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 atomic regions with arbitrary duration.

[35] presents real-time scheduling of transactions and serializes transactions based on deadlines. However, the work does not bound retries and response times, nor establishes tradeoffs against locking and lock-free approaches. In contrast, we establish such bounds and tradeoffs.

[36] proposes real-time HTM, unlike real-time STM that we consider. The work does not describe how transactional conflicts are resolved. In contrast, we show how task response times can be met using different conflict resolution policies. Besides, the retry bound developed in [36] 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.

The past work that is closest to ours is [19], which upper bounds retries and response times for EDF CM with G-EDF, and identify the tradeoffs against locking and lock-free protocols. Similar to [36], [19] also assumes that the worst case conflict between atomic sections occurs when the sections are released simultaneously. In addition, we consider RMA CM, besides EDF CM.

The ideas in [19] are extended in [3], which presents three real time CM designs. But no retry bounds nor schedulability analysis techniques are presented for those CMs.

3 Preliminaries

We consider a multiprocessor system with m identical processors and n sporadic tasks $\tau_1, \tau_2, \ldots, \tau_n$. The k^{th} instance (or job) of a task τ_i is denoted τ_i^k. Each task τ_i is specified by its worst case execution time (WCET) c_i, its minimum period T_i between any two consecutive instances, and its relative deadline D_i, where $D_i = T_i$. Job τ_i^j is released at time r_i^j and must finish no later than its absolute deadline $d_i^j = r_i^j + D_i$. Under a fixed priority scheduler such as G-RMA, p_i determines τ_i's (fixed) priority and it is constant for all instances of τ_i. Under a dynamic priority scheduler such as G-EDF, a job τ_i^j's priority, p_i^j, differs from one instance to another. A task τ_j may interfere with task τ_i for a number of times during an interval L, and this number is denoted as $G_{ij}(L)$.

Shared objects. A task may need to read/write shared, in-memory data objects while it is executing any of its atomic sections (transactions), which are synchronized using STM. The set of atomic sections of task τ_i is denoted s_i. s_i^k is the k^{th} atomic section of τ_i. Each object, θ, can be accessed by multiple tasks. The set of distinct objects accessed by τ_i is θ_i without repeating objects. The set of atomic sections used by τ_i to access θ is $s_i(\theta)$, and the sum of the lengths of those atomic sections is $len(s_i(\theta))$. $s_i^k(\theta)$ is the k^{th} atomic section of τ_i that accesses θ. s_i^k can access one or more objects in θ_i. So, s_i^k refers to the transaction itself, regardless of the objects accessed by the transaction. We denote the set of all accessed objects by s_i^k as Θ_i^k. While $s_i^k(\theta)$ implies that s_i^k accesses an object $\theta \in \Theta_i^k$, $s_i^k(\Theta)$ implies that s_i^k accesses a set of objects $\Theta = \{\theta \in \Theta_i^k\}$. $\bar{s_i^k} = \bar{s_i^k}(\Theta)$ refers only once to s_i^k, regardless of the number of objects in Θ. So, $|\bar{s_i^k}(\Theta)|_{\forall\theta\in\Theta} = 1$. $s_i^k(\theta)$ executes for a duration $len(s_i^k(\theta))$. $len(s_i^k) = len(s_i^k(\theta)) = len(s_i^k(\Theta)) = len(s_i^k(\Theta_i^k))$ The set of tasks sharing θ with τ_i is denoted $\gamma_i(\theta)$.

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

while the maximum one among all tasks is $s_{max}(\theta)$, and the maximum one among tasks with priorities lower than that of τ_i is $s^i_{max}(\theta)$.

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 $s^p_i(\theta)$ will take to execute due to a conflict with another section $s^k_j(\theta)$, is denoted $W^p_i(s^k_j(\theta))$. If an atomic section, s^p_i, is already executing, and another atomic section s^k_j tries to access a shared object with s^p_i, then s^k_j is said to "interfere" or "conflict" with s^p_i. The job s^k_j is the "interfering job", and the job s^p_i is the "interfered job".

Due to *transitive retry* (introduced in Section 10), an atomic section $s^k_i(\Theta^k_i)$ may retry due to another atomic section $s^l_j(\Theta^l_j)$, where $\Theta^k_i \cap \Theta^l_j = \emptyset$. θ^*_i denotes the set of objects not accessed directly by atomic sections in τ_i, but can cause transactions in τ_i to retry due to transitive retry. $\theta^{ex}_i (= \theta_i + \theta^*_i)$ is the set of all objects that can cause transactions in τ_i to retry directly or through transitive retry. γ^*_i is the set of tasks that accesses objects in θ^*_i. $\gamma^{ex}_i (= \gamma_i + \gamma^*_i)$ is the set of all tasks that can directly or indirectly (through transitive retry) cause transactions in τ_i to abort and retry.

The total time that a task τ_i's atomic sections have to retry over T_i is denoted $RC(T_i)$. The additional amount of time by which all interfering jobs of τ_j increases the response time of any job of τ_i during L, without considering retries due to atomic sections, is denoted $W_{ij}(L)$. These different preliminaries are shown in Appendix A

4 G-EDF/EDF CM Response Time

Since only one atomic section among many that share the same object can commit at any time under STM, those atomic sections execute in sequential order. A task T_i's atomic sections are interfered by other tasks that share the same objects with T_i. An atomic section of T_i, $s^k_i(\theta)$, is aborted and retried by a conflicting atomic section of T_j, $s^l_j(\theta)$, if $d(T_j) \leq d(T_i)$, by the EDF CM. We will use *ECM* to refer to a multiprocessor system scheduled by G-EDF and resolves STM conflicts using the EDF CM.

The maximum number of times a task T_j interferes with T_i is given in [4] and is shown in Figure 1. Here, the deadline of an instance of T_j coincides with that of T_i, and T^1_j is delayed by its maximum jitter J_j, which causes all or part of T_j's execution to overlap within T_i's period $t(T_i)$.

T_j's maximum workload that interferes with T_i in $t(T_i)$ is:

$$\begin{aligned} W^*_{ij}\left(t\left(T_i\right)\right) &= \left\lfloor \frac{t\left(T_i\right)}{t\left(T_j\right)} \right\rfloor .c_j + min\left(c_j, t\left(T_i\right) - \left\lfloor \frac{t\left(T_i\right)}{t\left(T_j\right)} \right\rfloor .t\left(T_j\right)\right) \\ &\leq \left\lceil \frac{t\left(T_i\right)}{t\left(T_j\right)} \right\rceil .c_j \end{aligned} \tag{1}$$

For an interval $L < t(T_i)$, the worst case pattern of interference is shown in Figure 2, and the workload of T_j is:

$$\hat{W}_{ij}\left(L\right) = \left(\left\lceil \frac{L - c_j}{t\left(T_j\right)} \right\rceil + 1 \right) .c_j \tag{2}$$

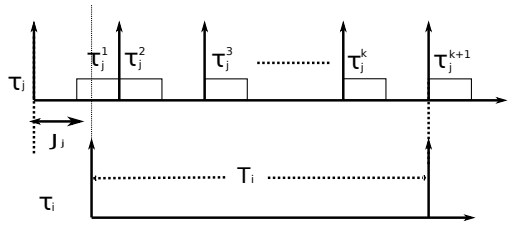


Figure 1: Maximum interference between two tasks under G-EDF

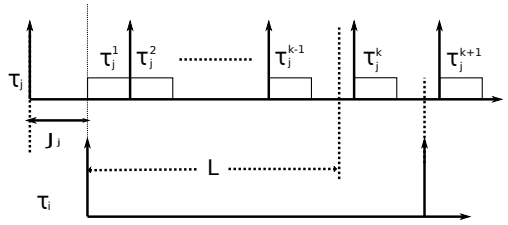


Figure 2: Maximum interference during part L of $t(T_i)$

Thus, the overall workload, over an interval R is:

$$W_{ij}(R) = min\left(\hat{W}_{ij}(R), W^*_{ij}(t(T_i))\right) \tag{3}$$

4.1 Retry Cost of Atomic Sections

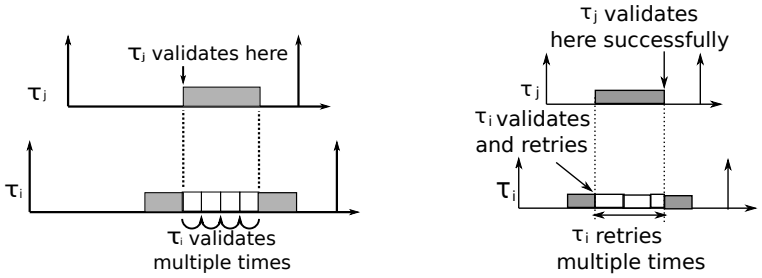
Claim 1 *Under ECM, a task T_i's maximum retry cost during $t(T_i)$ is upper bounded by:*

$$\begin{aligned} RC(T_i) \;\leq\; & \sum_{\theta\in\theta_i}\Big(\Big(\sum_{T_j\in\gamma(\theta)}\Big(\left\lceil\frac{t(T_i)}{t(T_j)}\right\rceil\sum_{\forall s^l_j(\theta)} len\big(s^l_j(\theta) \\ & + \; s_{max}(\theta)\big)\Big)\Big) - s_{max}(\theta) + s_{i_{max}}(\theta)\Big) \end{aligned} \tag{4}$$

Proof 1 Given two tasks T_i and T_j, where T_i has a longer absolute deadline than T_j. When a shared object conflict occurs, the EDF CM will commit T_j and abort and retry T_i. Thus, an atomic section of T_i, $s^k_i(\theta)$, will experience its maximum delay when it is at its end of the atomic section, and the conflicting atomic section of T_j, $s^l_j(\theta)$, starts. The CM will retry $s^k_i(\theta)$.

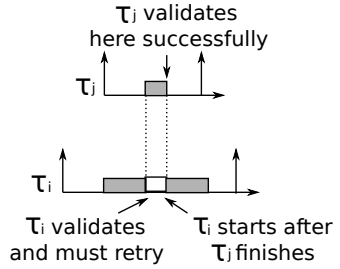
Validation (i.e., conflict detection) in STM is usually done in two ways [31]: a) eager (pessimistic), in which conflicts are detected at access time, b) lazy (optimistic), in which conflicts are detected at commit time. Despite the validation time incurred (either eager or lazy), $s^k_i(\theta)$ will retry for the same time duration, which is $len(s^l_j(\theta) + s^k_i(\theta))$. Then, $s^k_i(\theta)$ can commit successfully unless interferred by another conflicting atomic section, as shown in Figure 3.

In Figure 3(a), $s^l_j(\theta)$ validates at its beginning, due to early validation, and a conflict is detected. So T_i retries multiple times (because at the start of each retry, T_i validates) during the execution of $s^l_j(\theta)$. When T_j finishes its atomic section, T_i executes its atomic section.



(a) Early validation

(b) Lazy validation with $s_i^k(\theta) \leq s_j^l(\theta)$



(c) Lazy validation with $s_i^k(\theta) > s_j^l(\theta)$

Figure 3: Retry of $s_i^k(\theta)$ due to $s_j^l(\theta)$

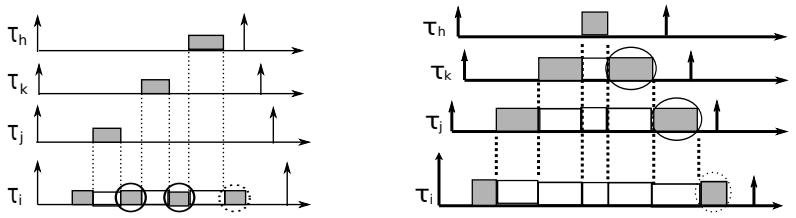
In Figure 3(b), T_i validates at its end (due to lazy validation), and detects a conflict with T_j. Thus, it retries, and because its atomic section length is shorter than that of T_j, it validates again within the execution interval of $s_j^l(\theta)$. However, the EDF CM retries it again. This process continues until T_j finishes its atomic section. If T_i's atomic section length is longer than that of T_j's, T_i would have incurred the same retry time, because T_j will validate when T_i is retrying, and T_i will retry again, as shown in Figure 3(c). Thus, the retry cost of $s_i^k(\theta)$ is $len(s_i^k(\theta) + s_j^l(\theta))$.

If multiple tasks interfere with T_i or interfere with each other and T_i (see the two interference examples in Figure 4), then, in each case, each atomic section of the shorter deadline tasks contributes to the delay of $s_i^p(\theta)$ by its total length, plus a retry to some atomic section in the longer deadline tasks. For example, $s_j^l(\theta)$ contributes by $len(s_j^l(\theta)+s_i^p(\theta))$ in both figures 4(a) and 4(b). In Figure 4(b), $s_k^y(\theta)$ causes a retry to $s_j^l(\theta)$, and $s_h^w(\theta)$ causes a retry to $s_k^y(\theta)$.

Since we do not know in advance which atomic section will be retried due to another, we can safely assume that, each atomic section (that share the same object with T_i) in a shorter deadline task contributes by its total length, in addition to the maximum length between all atomic sections that share the same object, $len(s_{max}(\theta))$. Thus,

$$W_i^p\left(s_j^k(\theta)\right) \leq len\left(s_j^k(\theta) + s_{max}(\theta)\right) \tag{5}$$

Thus, the total contribution of all atomic sections of all other tasks that



(a) Other atomic sections interfere only with $s_i^p(\theta)$

(b) All atomic sections interfere with each other and $s_i^p(\theta)$

Replaced in calculations by $s_{max}(\theta)$

Replaced in calculations by $s_{i_{max}}(\theta)$

Figure 4: Retry of $s_i^p(\theta)$ due to other atomic sections

share objects with a task T_i to the retry cost of T_i during T_i's period $t(T_i)$ is:

$$RC\left(T_i\right) \leq \sum_{\theta \in \theta_i} \sum_{T_j \in \gamma(\theta)} \Big(\left\lceil \frac{t\left(T_i\right)}{t\left(T_j\right)} \right\rceil \sum_{\forall s_j^l(\theta)} len\big(s_j^l(\theta) + s_{max}(\theta)\big)\Big) \quad (6)$$

Here, $\left\lceil \frac{t(T_i)}{t(T_j)} \right\rceil \sum_{\forall s_j^l(\theta)} len\left(s_j^l\left(\theta\right) + s_{max}\left(\theta\right)\right)$ is the contribution of all instances of T_j during $t(T_i)$. This contribution is added to all tasks. The last atomic section to execute is $s_i^p(\theta)$ (T_i's atomic section that was delayed by conflicting atomic sections of other tasks). One of the other atomic sections (e.g., $s_m^n(\theta)$) should have a contribution $len(s_m^n(\theta) + s_{i_{max}}(\theta))$, instead of $len(s_m^n(\theta) + s_{max}(\theta))$. That is why one $s_{max}(\theta)$ should be subtracted, and $s_{i_{max}}(\theta)$ should be added (i.e., $s_{i_{max}}(\theta) - s_{max}(\theta)$). Claim follows.

Claim 2 *Claim 1's retry bound can be minimized as:*

$$RC(T_i) \leq \sum_{\theta \in \theta_i} min(\Phi_1, \Phi_2) \quad (7)$$

where Φ_1 is calculated by (4) for one object θ (not the sum of $\theta \in \theta_i$), and

$$\Phi_2 = \Big(\sum_{T_j \in \gamma(\theta)} \Big(\left\lceil \frac{t\left(T_i\right)}{t\left(T_j\right)} \right\rceil \sum_{\forall s_j^l(\theta)} len\big(s_j^l(\theta) + s^*_{max}(\theta)\big)\Big)\Big) - \bar{s}_{max}(\theta) + s_{i_{max}}(\theta) \quad (8)$$

Proof 2 (4) can be modified by noting that a task T_i's atomic section may conflict with those of other tasks, but not with T_i. This is because, tasks are assumed to arrive sporadically, and each instance finishes before the next begins. Thus, (4) becomes:

$$RC(T_i) \leq \sum_{\forall \theta \in \theta_i} \Big(\Big(\sum_{T_j \in \gamma(\theta)} \Big(\left\lceil \frac{t\left(T_i\right)}{t\left(T_j\right)} \right\rceil \sum_{\forall s_j^l(\theta)} len\big(s_j^l(\theta) + s^*_{max}(\theta)\big)\Big)\Big) - \bar{s}_{max}(\theta) + s_{i_{max}}(\theta)\Big) \quad (9)$$

where, $s^*_{max}(\theta) \in s(\theta)$ and $s^*_{max}(\theta) \notin s_j(\theta)$, because T_j will not cause a retry to one of its instances.

To obtain $\bar{s}_{max}(\theta)$, the maximum-length atomic section of each task that accesses θ is grouped into an array, in non-increasing order of their lengths. $s_{max}(\theta)$ will be the first element of this array, and $\bar{s}_{max}(\theta)$ will be the next element, as illustrated in Figure 5, where the maximum atomic section of each task that accesses θ is associated with its corresponding task. In (9), all tasks but T_j will choose $s_{j_{max}}(\theta)$ as the value of $s^*_{max}(\theta)$, as it is the maximum-length atomic section not associated with the interfering task. But when T_j is the one whose contribution is studied, it will choose $s_{k_{max}}(\theta)$, as it is the maximum one not associated with T_j. This way, it can be seen that the maximum value always lies between the two values $s_{jmax}(\theta)$ and $s_{kmax}(\theta)$. Of course, these two values can be equal, or the maximum value can be associated with T_i itself, and not with any one of the interfering tasks. In the latter case, the chosen value will always be the one associated with T_i, and yet, it will lie between the two largest values.

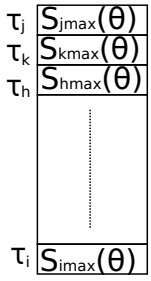


Figure 5: Values associated with $s^*_{max}(\theta)$

This means that the subtracted $s_{max}(\theta)$ in (4) must be replaced with one of these two values ($s_{max}(\theta)$ or $\bar{s}_{max}(\theta)$). However, since we do not know which task will interfere with T_i, the minimum is chosen, as we are determining the worst case retry cost (as this value is going to be subtracted), and this minimum is the second maximum.

Let $p_j = \left\lceil \frac{t(T_i)}{t(T_j)} \right\rceil$, g_j be the number of times T_j accesses θ, and $Const_j = \left\lceil \frac{t(T_i)}{t(T_j)} \right\rceil \times \sum_{\forall s^l_j(\theta)} len(s^l_j(\theta))$. If θ_1's maximum-length atomic section is associated with T_i (i.e., $s_{max}(\theta_1) = s_{i_{max}}(\theta_1)$), all other tasks will choose it, and Φ_1 (the result of (4) for θ_1) will be $\sum_{\forall T_j \in \gamma(\theta_1)} (Const_j + p_j g_j s_{i_{max}}(\theta_1)) - s_{i_{max}}(\theta_1) + s_{i_{max}}(\theta_1)$, whereas Φ_2 (the result of (9) for θ_1) will be $\sum_{\forall T_j \in \gamma(\theta_1)} (Const_j + p_j g_j s_{i_{max}}(\theta_1)) - s_{k_{max}}(\theta_1) + s_{i_{max}}(\theta_1)$. Since $s_{k_{max}}(\theta_1) \leq s_{i_{max}}(\theta_1)$, $\Phi_1 \leq \Phi_2$.

Let the maximum-length atomic section for θ_2 be $s_{d_{max}}(\theta_2)$ ($s_{max}(\theta_2) = s_{d_{max}}(\theta_2)$), and be associated with another task T_d, and not with T_i. Let $s_{k_{max}}(\theta_2) = \bar{s}_{max}(\theta_2)$, which will be the second minimum. Let T_d has g_d atomic sections that share θ_2 with T_i. Then, Φ_1 for θ_2 will result in $\sum_{\forall T_j \in \gamma(\theta_2)} (Const_j + p_j g_j s_{d_{max}}(\theta_2)) - s_{d_{max}}(\theta_2) + s_{i_{max}}(\theta_2)$, and Φ_2 will be $\sum_{\forall T_j \in \gamma(\theta_2) \wedge T_j \neq T_d} (Const_j + p_j g_j s_{d_{max}}(\theta_2)) + Const_d + p_d g_d s_{k_{max}}(\theta_2) - s_{k_{max}}(\theta_2) + s_{i_{max}}(\theta_2)$. So, $\Phi_1 - \Phi_2 = (p_d g_d - 1)(s_{d_{max}}(\theta_2) - s_{k_{max}}(\theta_2))$. Since T_d has at least one job that shares θ_2 with T_i (otherwise, T_d would not be included in $\gamma(\theta_2)$), $p_d g_d - 1 \geq 0$. Since $s_{d_{max}}(\theta_2) \geq s_{k_{max}}(\theta_2)$, $\Phi_1 \geq \Phi_2$.

Thus, given an object θ, Φ_1 may be greater, smaller, or equal to Φ_2. The minimum of Φ_1 and Φ_2 therefore yields the worst-case contribution for θ in $RC(T_i)$. Claim follows.

4.2 Upper Bound on Response Time

To obtain an upper bound on the response time of a task T_i, the term $RC(T_i)$ must be added to the workload of other tasks during the non-atomic execution of T_i. But this requires modification of the WCET of each task as follows. The WCET, c_j, of each interfering task T_j should be inflated to accommodate for the interference of tasks other than T_k, $k \neq j, i$. Meanwhile, atomic regions that access shared objects between T_j and T_i should not be considered in the inflation cost, because they have already been calculated in T_i's retry cost. Thus, T_j's inflated WCET becomes:

$$c_{ji} = c_j - \left(\sum_{\theta \in (\theta_j \wedge \theta_i)} len\left(s_j(\theta)\right) \right) + RC(T_{ji}) \tag{10}$$

where, c_{ji} is the new WCET of T_j relative to T_i; the sum of lengths of all atomic sections in T_j that access object θ is $\sum_{\theta \in (\theta_j \wedge \theta_i)} len(s_j(\theta))$; and $RC(T_{ji})$ is the $RC(T_j)$ without including the shared objects between T_i and T_j. The calculated WCET is relative to task T_i, as it changes from task to task. The upper bound on the response time of T_i, denoted R_i^{up}, can be calculated iteratively, using a modification of Theorem 6 in [4], as follows:

$$R_i^{up} = c_i + RC(T_i) + \left\lfloor \frac{1}{m} \sum_{j \neq i} W_{ij}(R_i^{up}) \right\rfloor \tag{11}$$

where R_i^{up}'s initial value is $c_i + RC(T_i)$.

$W_{ij}(R_i^{up})$ is calculated by (3), and $W_{ij}^*(t(T_i))$ is calculated by (1), with c_j replaced by c_{ji}, and changing $\hat{W}_{ij}(L)$ as:

$$\hat{W}_{ij}(L(T_i)) = max \begin{cases} \left(\left\lceil \frac{L - c_{ji} - \sum_{\theta \in (\theta_j \wedge \theta_i)} len(s_j(\theta))}{t(T_j)} \right\rceil + 1 \right) . c_{ji} \\ \left\lceil \frac{L - c_j}{t(T_j)} \right\rceil . c_{ji} + c_j - \sum_{\theta \in (\theta_j \wedge \theta_i)} len(s_j(\theta)) \end{cases} \tag{12}$$

(12) compares between two terms, as we have two cases:

Case 1. The carried-in job (i.e., a job whose release is before $r(T_i)$ and its deadline is after $r(T_i)$ but before $d(T_i)$, as defined in [4]) of T_j contributes by c_{ji}. Thus, other instances of T_j will begin after this modified WCET, but the sum of the shared objects' atomic section lengths is removed from c_{ji}, causing other instances to start earlier. Thus, the term $\sum_{\theta \in (\theta_i \wedge \theta_j)} len(s_j(\theta))$ is added to c_{ji} to obtain the correct start time.

Case 2. T_j's carried-in job contributes its c_j. Thus, other instances begin after this c_j of the carried-in job (as shown in Figure 2), but the sum of the shared atomic section lengths between T_i and T_j should be subtracted from this carried-in instance, as they are already included in the retry cost.

It should be noted that subtraction of the sum of the shared objects' atomic section lengths is done in the first case to obtain the correct start time of other

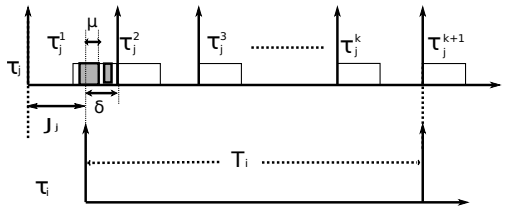


Figure 6: Atomic sections of job T_j^1 contributing to period $t(T_i)$

instances, while in the second case, this is done to get the correct contribution of the carried-in instance. The maximum is chosen from the two terms in (12), because they differ in the contribution of their carried-in jobs, and the number of instances after that.

4.2.1 Tighter Upper Bound

To tighten T_i's response time upper bound, the response time can be calculated recursively over duration R_i^{up}, and not directly over $t(T_i)$, as done in (11). Thus, $RC(T_i)$ will change according to T_i's recursive response time (i.e., R_i^{up}). So, (7) must be changed to include the modified number of interfering instances, in the same way this term is calculated in (3). Also, when calculating this term for the entire $t(T_i)$, a situation like that shown in Figure 6 can happen.

Atomic sections of T_j^1 that are contained in the interval δ are the only ones that can contribute to $RC(T_i)$. Of course, they can be lower, but cannot be greater, because T_j^1 has been delayed by its maximum jitter. Hence, no more atomic sections can interfere during the duration $[d(T_j^1)-\delta, d(T_j^1)]$. Even though only one of T_j^1's atomic sections contributes by length μ to T_i, the effect of this μ will still be the retry of one of the other atomic sections.

For simplicity, we use the following notations:

- $\lambda_1(j,\theta) = \sum_{\forall s_j^l(\theta) \in \left[d\left(T_j^1\right)-\delta, d\left(T_j^1\right)\right]^*} len\left(s_j^{l^*}(\theta) + s_{max}(\theta)\right)$
- $\chi_1(i,j,\theta) = \left\lfloor \frac{t(T_i)}{t(T_j)} \right\rfloor \sum_{\forall s_j^l(\theta)} len\left(s_j^l(\theta) + s_{max}(\theta)\right)$
- $\lambda_2(j,\theta) = \sum_{\forall s_j^l(\theta) \in \left[d\left(T_j^1\right)-\delta, d\left(T_j^1\right)\right]^*} len\left(s_j^{l^*}(\theta) + s_{max}^*(\theta)\right)$
- $\chi_2(i,j,\theta) = \left\lceil \frac{t(T_i)}{t(T_j)} \right\rceil \sum_{\forall s_j^l(\theta)} len\left(s_j^l(\theta) + s_{max}^*(\theta)\right)$

Here, $s_j^{l^*}(\theta)$ is the part of $s_j^l(\theta)$ that is included in interval δ. The term $\left[d\left(T_j^1\right)-\delta, d\left(T_j^1\right)\right]^*$ contains $s_j^l(\theta)$, whether it is partially or totally included in it. If it is partially included, $s_j^l(\theta)$ will contribute by its included length μ.

Now, (7) can be modified as:

$$RC\left(t\left(T_i\right)\right) \leq \sum_{\theta \in \theta_i} min \left\{ \begin{array}{l} \left(\left(\sum_{T_j \in \gamma(\theta)} \lambda_1(j,\theta) + \chi_1(i,j,\theta) \right) \right. \\ \left. -s_{max}(\theta) + s_{i_{max}}(\theta) \right) \\ \left(\left(\sum_{T_j \in \gamma(\theta)} \lambda_2(j,\theta) + \chi_2(i,j,\theta) \right) \right. \\ \left. -\bar{s}_{max}(\theta) + s_{i_{max}}(\theta) \right) \end{array} \right. \tag{13}$$

We can compute $RC\left(T_i\right)$ during a duration of length L, which does not extend to the last instance of T_j. Let:

- $\upsilon\left(L,j\right)=\left\lceil\frac{L-c_j}{t(T_j)}\right\rceil+1$
- $\lambda_3\left(j,\theta\right)=\sum_{\forall s_j^l(\theta)} len\left(s_j^l\left(\theta\right)+s_{max}\left(\theta\right)\right)$
- $\lambda_4\left(j,\theta\right)=\sum_{\forall s_j^l(\theta)} len\left(s_j^l\left(\theta\right)+s^*_{max}\left(\theta\right)\right)$

Now, (7) becomes:

$$RC\left(L\left(T_i\right)\right)\leq\sum_{\theta\in\theta_i} min\begin{cases}\left\{\begin{array}{l}\left(\sum_{T_j\in\gamma(\theta)}\left(\upsilon\left(L,j\right)\lambda_3\left(j,\theta\right)\right)\right)\\-s_{max}\left(\theta\right)+s_{i_{max}}\left(\theta\right)\end{array}\right.\\\left\{\begin{array}{l}\left(\sum_{T_j\in\gamma(\theta)}\left(\upsilon\left(L,j\right)\lambda_4\left(j,\theta\right)\right)\right)\\-\bar{s}_{max}\left(\theta\right)+s_{i_{max}}\left(\theta\right)\end{array}\right.\end{cases}\tag{14}$$

Thus, an upper bound on $RC(T_i)$ is given by:

$$RC(R_i^{up})\leq min\begin{cases}RC(R_i^{up}(T_i))\\RC(t(T_i))\end{cases}\tag{15}$$

The final upper bound on T_i's response time can be calculated as in (11) by replacing $RC(T_i)$ with $RC(R_i^{up})$.

5 G-RMA/RMA CM Response Time

As G-RMA is a fixed priority scheduler, a task T_i will be interfered by those tasks with priorities higher than T_i (i.e., $p(T_j) > p(T_i)$). Upon a conflict, the RMA CM will commit the transaction that belongs to the higher priority task. We will use *RCM* to refer to a multiprocessor system scheduled by G-RMA and resolves STM conflicts by the RMA CM.

5.1 Maximum Task Interference

Figure 7 illustrates the maximum interference caused by a task T_j to a task T_i under G-RMA. As T_j is of higher priority than T_i, T_j^k will interfere with T_i even if it is not totally included in $t(T_i)$. Unlike the G-EDF case shown in Figure 6, where only the δ part of T_j^1 is considered, in G-RMA, T_j^k can contribute by the whole c_j, and all atomic sections contained in T_j^k must be considered. This is because, in G-EDF, the worst-case pattern releases T_i before $d(T_j^1)$ by δ time units, and T_i cannot be interfered before it is released. But in G-RMA, T_i is already released, and can be interfered by the whole T_j^k, even if this makes it infeasible.

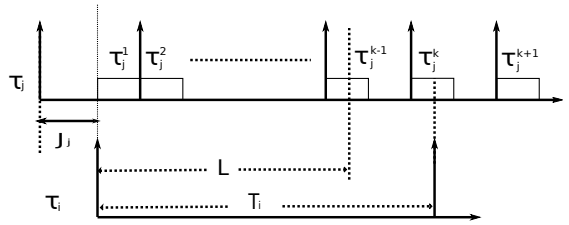


Figure 7: Max interference of T_j to T_i in G-RMA

Thus, the maximum contribution of T_j to T_i for any duration L can be deduced from Figure 7 as $W_{ij}(L(T_i)) = \left(\left\lceil \frac{L-c_j}{t(T_j)} \right\rceil + 1\right).c_j$, where L can extend to $t(T_i)$. In contrast to ECM where L cannot be extended directly to $t(T_i)$, as this will have a different pattern of worst case interference from other tasks.

5.2 Retry Cost of Atomic Sections

Claim 3 *Under RCM, a task T_i's retry cost over duration $L(T_i)$, which can extend to $t(T_i)$, is upper bounded by:*

$$\begin{aligned} RC\left(L\left(T_i\right)\right) \quad \leq \quad & \sum_{\theta \in \theta_i} \left(\left(\sum_{T_j^*} \left(\left(\left\lceil \frac{L - c_j}{t\left(T_j\right)} \right\rceil + 1 \right) \pi\left(j, \theta\right) \right) \right) \right. \\ - \quad & \left. s_{max}^{j}\left(\theta\right) + s_{i_{max}}\left(\theta\right) \right) \end{aligned} \tag{16}$$

where:

- $T_j^* = \{T_j | (T_j \in \gamma(\theta)) \wedge (p(T_j) > p(T_i)\}$
- $\pi(j, \theta) = \sum_{\forall s_j^l(\theta)} len\left(s_j^l\left(\theta\right) + s_{max}^{j}\left(\theta\right)\right)$

Proof 3 Since the worst case interference pattern for RCM is the same as that for ECM for an interval L, except that, in RCM, L can extend to the entire duration of $t(T_i)$, but in ECM, it cannot, as the interference pattern of T_j to T_i changes. So, (14) can be used to calculate T_i's retry cost, with some modifications, as we do not have to obtain the minimum of the two terms in (14), because T_j's atomic sections will abort and retry only atomic sections of tasks with lower priority than T_j. Thus, $s_{max}(\theta)$, $s^*_{max}(\theta)$, and $\bar{s}_{max}(\theta)$ are replaced by $s^j_{max}(\theta)$, which is the maximum-length atomic section of tasks with priority lower than T_j and share object θ with T_i. Besides, as T_i's atomic sections can be aborted only by atomic sections of higher priority tasks, not all $T_j \in \gamma(\theta)$ are considered, but only the subset of tasks in $\gamma(\theta)$ with priority higher than T_i (i.e., T_j^*).

5.3 Upper Bound on Response Time

The response time upper bound can be computed by Theorem 7 in [4] with some modification to include the effect of retry cost. Thus, this upper bound is given by:

$$R_i^{up} = c_i + RC(R_i^{up}) + \left\lfloor \frac{1}{m} \sum_{j \neq i} \hat{W}_{ij}(R_i^{up}) \right\rfloor \tag{17}$$

where $\hat{W}_{ij}(R_i^{up})$ is calculated as in (12), c_{ji} is calculated by (10), and RC is calculated by (16).

6 FMLP and OMLP Blocking Times

A number of protocols have been proposed for synchronization in real-time systems as shown in Table 1. The FMLP protocol [9] has been shown to be

Table 1: Some synchronization protocols

Protocol	Uniprocessor	Multiprocessor					
		Global			Partitioned		
		G-EDF	PFair	Others	P-EDF	Static Priority	Others
PCP [11]	✓						
SRP [2]	✓						
PCP variant [33]						✓	
PCP variant [37]						✓	
MPCP [27, 32]						✓	
DPCP [32]						✓	
[10]					✓		
SRP implementation [28]					✓		
SRP implementation [20]					✓		
[26]			✓				
PPCP [14]				✓			
PIP [14]							
[13]		✓					
FMLP [5, 6]		✓(GSN-EDF)	✓		✓(PSN-EDF)	✓	
OMLP [7]		✓		✓JLSP	✓		✓JLSP

superior to other multiprocessor real-time locking protocols in terms of schedulability, and the global OMLP protocol [7] has been shown to be asymptotically optimal. To formally compare STM against FMLP and global OMLP, we first upper bound their blocking times.

6.1 Global FMLP

FMLP can be used with global and partitioned scheduling. Since we only consider global scheduling, "FMLP" and "global FMLP" mean the same, for the paper's purpose.

FMLP divides shared objects into short resources, s_θ, and long ones, l_θ. Nested resources are grouped together into two groups: $g(s\_\theta)$ that contains only short resources, and $g(l\_\theta)$ that contains only long resources. A request $R_i(g(s\_\theta))$ is made by a task T_i to access one or more resources in $g(s\_\theta)$. This request's length is denoted $|R_i(g(s\_\theta))|$, and the number of times T_i requests short resources is denoted $N_{i,s}$. Similarly, $R_i(g(l\_\theta))$ is T_i's request to a group containing long resources for a duration $|R_i(g(l\_\theta))|$, and $N_{i,l}$ is the number of times T_i requests long resources.

Global FMLP uses a variant of G-EDF (called GSN-EDF which discriminates between linked jobs and scheduled ones) to account for non-preemptive jobs while still using G-EDF for scheduling, Tasks busy-wait on short resources, and suspend on long ones. In both cases, requests for resources are arranged in FIFO queues, and for requesting long resources, the task holding the resource inherits the highest priority of the suspended tasks on that resource. For short resources, there is no need to inherit priorities, as tasks become non-preemptable when acquiring short resources.

Requests for long resources can contain requests for the short resource group,

but the reverse is not true. The protocol allows non-preemptable jobs and bounds the time a job is non-preemptively blocked by a lower priority job as the maximum time a non-preemptive section of the job can be linked to the processor of the higher priority job. This non-preemptive blocking can only happen when the higher priority job is released or resumed.

Three types of blocking can be incurred by any task under global FMLP. These include busy-wait blocking, non-preemptive blocking, and direct blocking. The total blocking time of a job, b_i, is the sum of these three blocking durations. Execution time of each task, e_i, is inflated by this blocking amount ($e_i + b_i$), and is used in any of the G-EDF schedulability tests (e.g., [21,39]) for verifying schedulability.

We upper bound a task's blocking durations due to busy-wait blocking, non-preemptive blocking, and direct blocking, denoted as $BW(T_i)$, $NPB(T_i)$, and $DB(T_i)$, respectively, as follows. (Note that, in [7], no upper bounds are presented for these terms, except for $DB(T_i)$, as [7]'s main focus is on FMLP's suspension-based part. Also, the upper bound for $DB(T_i)$ in [7] does not consider the effect of requesting a short resource within a long one.)

A job T_i^j busy-waits in a FIFO queue when it is scheduled on a processor and it cannot be removed by any other task until its request is satisfied. As busy-waiting tasks are non-preemptable, job T_i^j can be blocked for at most the maximum $m-1$ requests, where each request consists of the sum of the nested requests to some resources in the same group. This process proceeds for each short resource requested by T_i. The busy-wait blocking time, $BW(T_i)$, is therefore:

$$BW(T_i) \leq \sum_{s\_\theta \in \theta_i} \left(max \left[\sum_{k=1, k\neq i}^{min(m,n)-1} |R_k\left(g\left(s\_\theta\right)\right)| \right] \right) \tag{18}$$

A job T_i^j can be non-preemptively blocked, either at its release or when it resumes, by at most the maximum (nested) request to any short resource. The non-preemptive blocking time, $NPB(T_i)$, is therefore:

$$NPB(T_i) = (1 + N_{i,l}).max_{k\neq i} |R_k(g(s\_\theta))| \tag{19}$$

Here, 1 is added to $N_{i,l}$, because T_i can be non-preemptively blocked at its release, in addition to suspension times.

A job T_i^j can be blocked by all other $n-1$ tasks for any long resource. Any of these $n-1$ requests can be a nested request to long resources belonging to the same group. In addition, any of those requests can contain a request to a short resource, and so it can busy-wait on it. Thus, each request in the $n-1$ requests, requiring access to a short resource, can be delayed by at most the maximum $m-1$ requests to the group containing that short resource. The direct blocking time, $DB(T_i)$, is therefore:

$$DB(T_i) \leq \sum_{l\_\theta \in \theta_i} \left[max_{k=1, k\neq i}^{n-1} |R_k\left(g\left(l\_\theta\right)\right)| \right] \tag{20}$$

6.2 Global OMLP

In [7], global FMLP has a maximum s-oblivious pi-blocking cost of $\Theta(n)$, whereas global OMLP [7], which is a suspension-based protocol that supports G-EDF,

as well as any global job-level static priority (JLSP) scheduler, has a $\Theta(m)$ s-oblivious pi-blocking cost, as seen by equation (21):

$$b_i \triangleq \sum_{k=1}^{q} N_{i,k}.2.(m-1).max_{1 \leq i \leq n}\{L_{i,k}\} \tag{21}$$

where $N_{i,k}$ is the maximum number of times T_i requests resource k, and $L_{i,k}$ is the maximum execution time of such a request. $N_{i,k}$ and $L_{i,k}$ are assumed to be constants, so the s-oblivious pi-blocking is $\Theta(m)$, and thus it is optimal.

7 STM versus Locks and Lock-Free

We now would like to understand when STM will be beneficial, and when FMLP/global OMLP and lock-free approaches will be so. We consider the lock-free retry-loop approach for G-EDF in [13], where a retry upper bound is developed by computing the worst-case number of tasks that can execute within one period of the task being considered. This lock-free approach is the most relevant to our work.

7.1 ECM versus Lock-Free

Claim 4 *For ECM's schedulability to be better or equal to that of [13]'s retry-loop lock-free approach, the size of s_{max} must not exceed one half of that of r_{max}; with low number of conflicting tasks, the size of s_{max} can be at most the size of r_{max}.*

Proof 4 Equation (15) can be upper bounded as:

$$RC\left(T_i\right) \leq \sum_{T_j \in \gamma_i} \left(\sum_{\theta \in \theta_i} \left(\left\lceil \frac{t\left(T_i\right)}{t\left(T_j\right)} \right\rceil \sum_{\forall s_j^l(\theta)} \left(2.s_{max}\right) \right) \right) \tag{22}$$

where $s_j^l\left(\theta\right)$, $s_{max}\left(\theta\right)$, $s_{i_{max}}\left(\theta\right)$, $s_{max}^*\left(\theta\right)$, and $\bar{s}_{max}\left(\theta\right)$ are replaced by s_{max}, and the order of the first two summations are reversed by each other, with γ_i being the set of tasks that share objects with task T_i. These changes are done to simplify the comparison.

Let $\sum_{\theta \in \theta_i} \sum_{\forall s_j^l(\theta)} = \beta_{i,j}^*$, and $\alpha_{edf} = \sum_{T_j \in \gamma_i} \left\lceil \frac{t(T_i)}{t(T_j)} \right\rceil .2\beta_{i,j}^*$. Now, (22) can be modified as:

$$RC\left(T_i\right) = \alpha_{edf}.s_{max} \tag{23}$$

The loop retry cost is given by:

$$\begin{aligned} LRC\left(T_i\right) &= \sum_{T_j \in \gamma_i} \left(\left\lceil \frac{t\left(T_i\right)}{t\left(T_j\right)} \right\rceil + 1 \right) .\beta_{i,j}.r_{max} \\ &= \alpha_{free}.r_{max} \end{aligned} \tag{24}$$

where $\beta_{i,j}$ is the number of retry loops of T_j that accesses the same object as that accessed by some retry loop of T_i, $\alpha_{free} = \sum_{T_j \in \gamma_i} \left(\left\lceil \frac{t(T_i)}{t(T_j)} \right\rceil + 1 \right) .\beta_{i,j}$, and r_{max} is the maximum execution cost of a single iteration of any retry loop

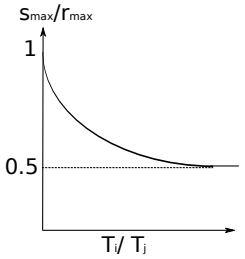


Figure 8: Effect of $\left\lceil \frac{t(T_i)}{t(T_j)} \right\rceil$ on $\frac{s_{max}}{r_{max}}$

of any task. Since the shared objects are the same in both STM and lock free, $\beta_{i,j} = \beta^*_{i,j}$. Thus, STM achieves equal or better schedulability than lock-free if the total utilization of the STM system is less than or equal to the lock-free system:

$$\begin{aligned} \sum_{T_i} \frac{c_i + \alpha_{edf}.s_{max}}{t\,(T_i)} &\leq \sum_{T_i} \frac{c_i + \alpha_{free}.r_{max}}{t\,(T_i)} \\ \therefore \frac{s_{max}}{r_{max}} &\leq \frac{\sum_{T_i} \alpha_{free}/t\,(T_i)}{\sum_{T_i} \alpha_{edf}/t\,(T_i)} \end{aligned} \tag{25}$$

Let $\bar{\alpha}_{free} = \sum_{T_j \in \gamma_i} \left\lceil \frac{t(T_i)}{t(T_j)} \right\rceil .\beta_{i,j}$, $\hat{\alpha}_{free} = \sum_{T_j \in \gamma_i} \beta_{i,j}$, and $\alpha_{free} = \bar{\alpha}_{free} + \hat{\alpha}_{free}$. Therefore:

$$\begin{aligned} \frac{s_{max}}{r_{max}} &\leq \frac{\sum_{T_i} (\bar{\alpha}_{free} + \hat{\alpha}_{free})/t(T_i)}{\sum_{T_i} \alpha_{edf}/t(T_i)} \\ &= \frac{1}{2} + \frac{\sum_{T_i} \hat{\alpha}_{free}/t(T_i)}{\sum_{T_i} \alpha_{edf}/t(T_i)} \end{aligned} \tag{26}$$

Let $\zeta_1 = \sum_{T_i} \hat{\alpha}_{free}/t(T_i)$ and $\zeta_2 = \sum_{T_i} \left(\frac{\alpha_{edf}}{2}\right)/t(T_i)$. Thus, for $\zeta_1 \leq \zeta_2$ depends on $\left\lceil \frac{t(T_i)}{t(T_j)} \right\rceil$. The maximum value of $\frac{\zeta_1}{2.\zeta_2} = \frac{1}{2}$, which can happen if $t(T_j) \geq t(T_i) \;\therefore \left\lceil \frac{t(T_i)}{t(T_j)} \right\rceil = 1$. Then (26) = 1, which is its maximum value. Of course, $t(T_j) \geq t(T_i)$, which means that there is small number of interferences from other tasks to T_i, and thus low number of conflicts. Therefore, s_{max} is allowed to be as large as r_{max}.

The theoretical minimum value for $\frac{\zeta_1}{2.\zeta_2}$ is 0, which can be asymptotically reached if $t(T_j) \ll t(T_i)$, $\therefore \left\lceil \frac{t(T_i)}{t(T_j)} \right\rceil \to \infty$ and $\zeta_2 \to \infty$. Thus, (26) $\to 1/2$.

$\beta_{i,j}$ has little effect on s_{max}/r_{max}, as it is contained in both numerator and denominator. Irrespective of whether $\beta_{i,j}$ is going to reach its maximum or minimum value, both can be considered constants, and thus removed from (26)'s numerator and denominator. However, the number of interferences of other tasks to T_i, $\left\lceil \frac{t(T_i)}{t(T_j)} \right\rceil$, has the main effect on s_{max}/r_{max}, as shown in Figure 8.

7.2 RCM versus Lock-Free

Claim 5 *For RCM's schedulability to be better or equal to that of [13]'s retry-loop lock-free approach, the size of s_{max} must not exceed one half of that of r_{max} for all cases. However, the size of s_{max} can be larger than that of r_{max}, depending on the number of accesses to a task T_i's shared objects from other tasks.*

Proof 5 Equation (16) is upper bounded by:

$$\sum_{(T_j \in \gamma_i) \wedge (p(T_j) > p(T_i))} \left(\left\lceil \frac{t(T_i) - c_j}{t(T_j)} \right\rceil + 1 \right) .2.\beta_{i,j}.s_{max} \tag{27}$$

Consider the same assumptions as in Section 7.1. Let

$$\alpha_{rma} = \sum_{(T_j \in \gamma_i) \wedge (p(T_j) > p(T_i))} \left(\left\lceil \frac{t(T_i) - c_j}{t(T_j)} \right\rceil + 1 \right) .2.\beta_{i,j}$$

Now, the ratio s_{max}/r_{max} is upper bounded by:

$$\frac{s_{max}}{r_{max}} \leq \frac{\sum_{T_i} \alpha_{free}/t(T_i)}{\sum_{T_i} \alpha_{rma}/t(T_i)} \tag{28}$$

The main difference between RCM and lock-free is that RCM is affected only by the higher priority tasks, while lock-free is affected by all tasks (just as in ECM). Besides, the RCM is still affected by $2.\beta_{i,j}$ (just as in ECM). The subtraction of c_j in the numerator in (27) may not have a significant effect on the ratio of (28), as the loop retry cost can also be modified to account for the effect of the first interfering instance of task T_j.

Therefore, $\alpha_{free} = \sum_{T_j \in \gamma_i} \left(\left\lceil \frac{t(T_i)-c_j}{t(T_j)} \right\rceil + 1 \right) \beta_{i,j}$.

Let tasks in the denominator of (28) be given indexes k instead of i, and l instead of j. Let tasks in both the numerator and denominator of (28) be arranged in the non-increasing priority order, so that $i = k$ and $j = l$. Let α_{free}, in (28), be divided into two parts: $\bar{\alpha}_{free}$ that contains only tasks with priority higher than T_i, and $\hat{\alpha}_{free}$ that contains only tasks with priority lower than T_i. Now, (28) becomes:

$$\begin{aligned} \frac{s_{max}}{r_{max}} &\leq \frac{\sum_{T_i} (\bar{\alpha}_{free} + \hat{\alpha}_{free})/t(T_i)}{\sum_{T_k} \alpha_{rma}/t(T_k)} \\ &= \frac{1}{2} + \frac{\sum_{T_i} \hat{\alpha}_{free}/t(T_i)}{\sum_{T_k} \alpha_{rma}/t(T_k)} \end{aligned} \tag{29}$$

For convenience, we introduce the following notations:

$$\begin{aligned}
\zeta_1 &= \sum_{T_i} \frac{\sum_{(T_j \in \gamma_i) \wedge (p(T_j) < p(T_i))} \left(\left\lceil \frac{t(T_i) - c_j}{t(T_j)} \right\rceil + 1 \right) \beta_{i,j}}{t\,(T_i)} \\
&= \sum_{T_i} \hat{\alpha}_{free}/t(T_i) \\
\zeta_2 &= \sum_{T_k} \frac{\sum_{(T_l \in \gamma_k) \wedge (p(T_l) > p(T_k))} \left(\left\lceil \frac{t(T_k) - c_l}{t(T_l)} \right\rceil + 1 \right) \beta_{k,l}}{t\,(T_k)} \\
&= \frac{1}{2} \sum_{T_k} \alpha_{rma}/t(T_k)
\end{aligned}$$

T_j is of lower priority than T_i, which means $D(T_j) > D(T_i)$. Under G-RMA, this means, $t(T_j) > t(T_i)$. Thus, $\left\lceil \frac{t(T_i) - c_j}{t(T_j)} \right\rceil = 1$ for all T_j and

$$\zeta_1 = \sum_{T_i} \Big(\sum_{(T_j \in \gamma_i) \wedge (p(T_j) < p(T_i))} (2.\beta_{i,j}) \Big) / t(T_i)$$

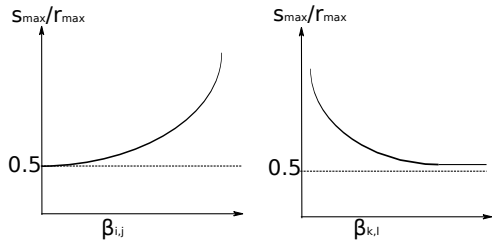
Since ζ_1 contains all T_j of lower priority than T_i and ζ_2 contains all T_l of higher priority than T_k, and tasks are arranged in the non-increasing priority order, then for each $T_{i,j}$, there exists $T_{k,l}$ such that $i = l$ and $j = k$. Figure 9 illustrates this, where 0 means that the pair i, j does not exist in ζ_1, and the pair k, l does not exist in ζ_2' (i.e., there is no task T_l that is going to interfere with T_k in ζ_2), and 1 means the opposite.

i \ j	1	2	$\cdots$	n
1	0	1	$\cdots$	1
2	0	0	$\ddots$	$\vdots$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	1
n	0	0	$\cdots$	0

k \ l	1	2	$\cdots$	n
1	0	0	$\cdots$	0
2	1	0		$\vdots$
$\vdots$	$\vdots$	$\ddots$	$\ddots$	0
n	1	$\cdots$	1	0

Figure 9: Task association for lower priority tasks than T_i and higher priority tasks than T_k

Thus, it can be seen that both the matrices are transposes of each other. Consequently, for each $\beta_{i,j}$, there exists $\beta_{k,l}$ such that $i = l$ and $j = k$. But the number of times T_j accesses a shared object with T_i may not be the same as the number of times T_i accesses that same object. Thus, $\beta_{i,j}$ does not have to be the same as $\beta_{k,l}$, even if i, j and k, l are transposes of each other. Therefore, we can analyze the behavior of s_{max}/r_{max} based on the three parameters $\beta_{i,j}$, $\beta_{k,l}$, and $\left\lceil \frac{t(T_k) - c_l}{t(T_l)} \right\rceil$. If $\beta_{i,j}$ is increased so that $\beta_{i,j} \to \infty$, $\therefore$ (29) $\to \infty$. This is because, $\beta_{i,j}$ represents the number of times a lower priority task T_j accesses shared objects with the higher priority task T_i. While this number has a greater effect in lock-free, it does not have any effect under RCM, because lower priority tasks do not affect higher priority ones, so s_{max} is allowed to be much greater than r_{max}.



(a) (b)

Figure 10: Change of s_{max}/r_{max}: a) $\frac{s_{max}}{r_{max}}$ versus $\beta_{i,j}$ and b) $\frac{s_{max}}{r_{max}}$ versus $\beta_{k,l}$

Although the minimum value for $\beta_{i,j}$ is 1, mathematically, if $\beta_{i,j} \to 0$, then (29) $\to 1/2$. Here, changing $\beta_{i,j}$ does not affect the retry cost of RCM, but it does affect the retry cost of lock-free, because the contention between tasks is reduced. Thus, s_{max} is reduced in this case to a little more than half of r_{max} ("a little more" because the minimum value of $\beta_{i,j}$ is actually 1, not 0).

The change of s_{max}/r_{max} with respect to $\beta_{i,j}$ is shown in Figure 10(a). If $\beta_{k,l} \to \infty$, then (29)$\to 1/2$. This is because, $\beta_{k,l}$ represents the number of times a higher priority task T_l accesses shared objects with a lower priority task T_k. Under RCM, this will increase the retry cost, thus reducing s_{max}/r_{max}. But if $\beta_{k,l} \to 0$, then (29)$\to \infty$. This is due to the lower contention from a higher priority task T_l to a lower priority task T_k, which reduces the retry cost under RCM and allows s_{max} to be very large compared with r_{max}. Of course, the actual minimum value for $\beta_{k,l}$ is 1, and is illustrated in Figure 10(b).

The third parameter that affects s_{max}/r_{max} is $t(T_k)/t(T_l)$. If $t(T_l) \ll t(T_k)$, then $\left\lceil \frac{t(T_k)-c_l}{t(T_l)} \right\rceil \to \infty$, and (29) $\to 1/2$. This is due to a high number of interferences from a higher priority task T_l to a lower priority one T_k, which increases the retry cost under RMA CM, and consequently reduces s_{max}/r_{max}.

If $t(T_l) = t(T_k)$ (which is the maximum value for $t(T_l)$ as $D(T_l) \leq D(T_k)$, because T_l has a higher priority than T_k), then $\left\lceil \frac{t(T_k)-c_l}{t(T_l)} \right\rceil \to 1$ and $\zeta_2 = \sum_{T_k} \frac{\sum_{(T_l \in \gamma_k) \wedge (p(T_l) > p(T_k))} 2\beta_{k,l}}{t(T_k)}$. This means that the system will be controlled by only two parameters, $\beta_{i,j}$, and $\beta_{k,l}$, as in the previous two cases, shown in Figures 10(a) and 10(b). Claim follows.

7.3 FMLP & OMLP versus ECM and RCM

Claim 6 *For ECM's schedulability to be better or equal to that of FMLP or OMLP, $s_{max}/|s\_\theta|_{max}$ (in case of FMLP), where $|s\_\theta|_{max}$ be the maximum short request by any task, and s_{max}/L_{max} (in case of OMLP) must not exceed $O(\frac{m}{n})$, where $L_{max} = max_{\forall i, \forall k} L_{i,k}$. For RCM's schedulability to be better or equal to that of OMLP, s_{max}/L_{max} must not exceed $O(\frac{m}{n})$.*

Proof 6 As FMLP is used with G-EDF (GSN-EDF), we compare only ECM against it. First, we derive upper bounds for the blocking parameters of FMLP. When requests are non-nested, each resource (short or long) will be contained in its own group. Let $N_{i,s}$ be the number of times a task T_i requests a short

resource, $|s\_\theta|_{i,max}$ be the maximum request for a short resource by T_i, $\alpha_{bw} = N_{i,s}.(m-1)$. Now, FMLP's three blocking terms, described in Section 6.1, are upper bounded as follows:

$$\begin{aligned} BW(T_i) &\leq \sum_{s\_\theta \in \theta_i} (m-1).|s\_\theta|_{i,max} \\ &= N_{i,s}.(m-1).|s\_\theta|_{i,max} \\ &\leq N_{i,s}.(m-1).|s\_\theta|_{max} \\ &= \alpha_{bw}.|s\_\theta|_{max} \end{aligned}$$

$$\begin{aligned} NPB\,(T_i) &\leq (1+N_{i,l})\,.max\,(BW_{k\neq i}\,(T_k) + |s\_\theta|_{k,max}) \\ &\leq (1+N_{i,l})\,.max\,(BW_{k\neq i}\,(T_k) + |s\_\theta|_{max}) \\ &= (1+N_{i,l})\,.|s\_\theta|_{max}.max_{k\neq i}\,(N_{k,s}.\,(m-1)+1) \\ &= \alpha_{npb}.|s\_\theta|_{max} \end{aligned}$$

where $\alpha_{npb} = (1+N_{i,l})\,.max_{k\neq i}\,(N_{k,s}.\,(m-1)+1)$.

$$\begin{aligned} DB\,(T_i) &\leq N_{i,l}.\,(n-1)\,.|l\_\theta|_{i,max} \\ &\leq N_{i,l}.\,(n-1)\,.|l\_\theta|_{max} \end{aligned}$$

If $|l\_\theta|_{max} \leq c1.|s\_\theta|_{max}$, where $c1$ is the minimum constant that satisfies this relation, then

$$\begin{aligned} DB\,(T_i) &\leq N_{i,l}.\,(n-1)\,.c1.|s\_\theta|_{max} \\ &= \alpha_{db}.|s\_\theta|_{max} \end{aligned}$$

where $\alpha_{db} = N_{i,l}.\,(n-1)\,.c1$.

The total blocking time of each task is added to the task execution time, and as before, we compare the total utilization of the G-EDF system (with both contention managers) against that under FMLP.

Now, for ECM's schedulability to be better than FMLP,

$$\frac{s_{max}}{|s\_\theta|_{max}} \leq \frac{\sum_{T_i} (\alpha_{bw} + \alpha_{npb} + \alpha_{db}) \Big/ t(T_i)}{\sum_{T_i} \alpha_{edf} \Big/ t(T_i)} \tag{30}$$

From (30), it can be seen that T_i's blocking time, under FMLP, depends on m, n and the number of times it requests resources (in contrast to ECM, under which, T_i's retry cost depends on the number of times a conflicting task T_j requests resources). Thus, if $N_{i,s}$, $N_{i,l}$, and $N_{k,s}$ can all be upper bounded by some constant C_2, which is the maximum number of times any task T_i can request a short or long resource, then the numerator in (30) is $O(n(n+m))$, while the denominator is $O(n^2)$. Therefore:

$$\frac{s_{max}}{|s\_\theta|_{max}} = O\left(\frac{m}{n}\right) \tag{31}$$

This means that, for $n < m$, the contention between tasks under both STM and FMLP is low (even for short resources under FMLP), but FMLP is more

affected by NTB. When $n > m$, contention increases, but FMLP arranges requests in a FIFO queue, so it is less affected than ECM, which suffers from conflicting tasks and instances of each conflicting one. FMLP is not affected by the number of instances of each conflicting task.

Since OMLP's blocking time is bounded by (21)

$$\therefore bi \leq 2.(m-1).L_{max} \sum_{k=1}^{q} N_{i,k}$$

For ECM's schedulability to be better than global OMLP:

$$\frac{s_{max}}{L_{max}} \leq \frac{\sum_{T_i} \left(2.(m-1) \sum_{k=1}^{q} N_{i,k}\right) \Big/ t\left(T_i\right)}{\sum_{T_i} \alpha_{edf} \Big/ t\left(T_i\right)} \tag{32}$$

For RCM, the ratio is:

$$\frac{s_{max}}{L_{max}} \leq \frac{\sum_{T_i} \left(2.(m-1) \sum_{k=1}^{q} N_{i,k}\right) \Big/ t\left(T_i\right)}{\sum_{T_i} \alpha_{rma} / t\left(T_i\right)} \tag{33}$$

If $\sum_{k=1}^{q} N_{i,k}$ is upper bounded by C_3, which is a constant representing the maximum total number of requests for resources by any task T_i, then:

$$\frac{s_{max}}{L_{max}} = O\left(\frac{nm}{n^2}\right) = O\left(\frac{m}{n}\right) \tag{34}$$

for each of (32) and (33). Claim follows.

8 Response time of G-EDF and G-RMA system with FIFO CM

In FIFO CM, transactions are executed in the order they arrive. This way, the effect of an atomic section of T_j to one in T_i is shown in Figure 11(a) for early validation and Figure 11(b) for lazy validation. In both cases, $s_i^l(\theta)$ must retry for at most $len(s_j^p(\theta))$, which is less than that of EDF CM and RMA CM, but a higher priority task can be delayed by a lower one if the latter's atomic section arrives earlier than the former. Besides, if a low priority task, T_i, is executing its atomic section, and a higher priority one, T_j, is released and allocated to the processor of T_i, the order of the atomic section of T_i in the FIFO is delayed after any conflicting atomic section arrives in the preemption interval of T_i, otherwise, tasks other than T_i can be delayed until T_i is allocated a processor again and finishes its atomic section, besides this helps avoiding deadlocks, as T_j itself may need to access the same object accessed by the atomic section of T_i but it cannot commit because every time it tries to do that, CM will enforce it to abort as it arrives later than the atomic section of the preempted task.

Retrial of atomic sections here does not depend on the scheduler, rather it only depends on the arrival pattern, so retrial cost (RC) will be the same for both G-EDF and G-RMA systems, and the maximum pattern of interference will also be the same as shown in Figure 7 (in case of G-EDF, the last instance

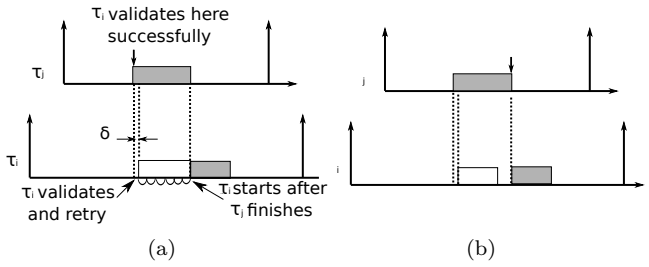


Figure 11: (a)Early validation (b)Lazy validation

of the interfering task with longer absolute deadline than the studied one, can have atomic sections arriving earlier than that of T_i and still conflict with them), but this pattern differs when calculating the contribution of other tasks during the non-atomic part of T_i.

RC can be calculated by (35)

$$\hat{RC}(L(T_i)) = \sum_{\theta \in \theta_i} ((\sum_{T_j \in \gamma(\theta)} ((\lceil \frac{L - c_j}{t(T_j)} \rceil + 1). \sum_{\forall s_j^l(\theta)} len(s_j^l(\theta))))) \quad (35)$$

where the lengths of all atomic sections in all tasks that share object θ with T_i are summed together, and L can extend to cover $t(T_i)$. It should be noted that one instance can have multiple atomic sections that access the same object, then how can an atomic section of T_i be interfered by all of them while it should be interfered by only one of them according to its arrival time, as the others will come after that of T_i? This can happen if T_i was trying to execute its atomic section, then it is replaced by a higher priority job, then T_i gets back to the processor. In the time of preemption, another atomic section of the same conflicting task can come and execute. This situation is shown in Figure 12 where s_i^l retries while s_j^p is executing, then T_k is released, preempting T_i to get its processor and before it finished, s_j^{p+1} executes, so when s_i^l returns after T_k finishes, it will be after s_j^{p+1} and it will have to retry until it finishes. For this, (35) sums the lengths of all atomic sections in each instance that access the same object as the studied task.

The rest of analysis for both schedulers is done as before, only $RC(L(T_i))$ is changed.

In case of G-RMA

For G-RMA, the case shown in Figure 12 can happen only if T_i is the lowest priority among all running tasks, which reduces number of conflicting tasks to only those of higher or equal priority to that of T_i. Meanwhile, if there are m processors, this means that before T_i gets preempted, it will retry for the duration of at most the maximum $m - 1$ atomic sections sharing the same object with T_i whether they belong to higher priority tasks or not.

After T_i gets preempted, no lower priority task can allocate a processor while T_i is waiting, but as T_i can get preempted again and again by higher priority tasks, it can be delayed for each release of a higher priority job, T_i by the

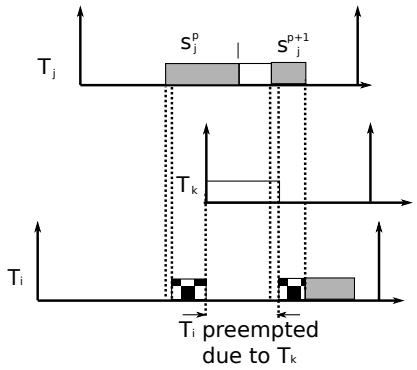


Figure 12: Retrial of s_i^l by multiple atomic sections of s_j^p

maximum $m-1$ higher or equal priority tasks (the first term of the "min" part in (36)), or, it can be delayed by the sum of all atomic sections in higher priority tasks that access the same object as T_i (the second term of the "min" part in (36)).

So, RC in the case of G-RMA can be calculated as in (36)

$$RC^*(L(T_i)) = \sum_{\theta\in\theta_i}(\sum_{k=1}^{min(n-1,m-1)}(\omega(k,\theta)) + (\sum_{T_j^*} min^* \begin{cases} \lceil \frac{L}{t(T_j)} \rceil . \sum_{k=1}^{m-1} \omega_{high}(k,\theta))) \\ (\lceil \frac{L-c_j}{t(T_j)} \rceil + 1) . \sum_{\forall s_j^l(\theta)} len(s_j^l(\theta)))) \end{cases} \quad (36)$$

where:-

- $\omega(k,\theta)$ is the set of $s_{p_{max}}(\theta), p \neq i$ arranged in non-increasing order.
- $\omega_{high}(k,\theta)$ is the set of $s_{p_{max}}(\theta), p \neq i$ arranged in non-increasing order, where $p(T_p) \geq p(T_i)$.
- $min^* = 0$ if $n \leq m$.
- $T_j^* = \{T_j | (T_j \in \gamma(\theta)) \wedge (p(T_j) \geq p(T_i))\}$.

The second term is zero in case number of tasks is less than or equal to number of processors, because T_i will not have to be preempted because of enough number of processors for tasks, and the final value for retrial cost will be the minimum of (35) and (36).

$$RC(L(T_i)) = min(\hat{RC}(L(T_i)), RC^*(L(T_i))) \quad (37)$$

The rest of the analysis for workload of other tasks is the same as before.

In case of G-EDF

G-EDF is similar to G-RMA in that T_i- before it is preempted- it can be delayed by the maximum $min(m-1, n-1)$ tasks, but after it is preempted, no task of longer, or equal, relative deadline than T_i will start before it because it will have a longer absolute deadline. So, RC in the case of G-EDF will be calculated as shown in (38) where L can be extended to $t(T_i)$, and the final RC will be calculated as in (37).

$$RC^*(L(T_i)) = \sum_{\theta\in\theta_i}(\sum_{k=1}^{min(n-1,m-1)}(\omega(k,\theta)) + (\sum_{T_j^*} min^* \begin{cases} \lfloor \frac{L}{t(T_j)} \rfloor . \sum_{k=1}^{m-1} \omega_{high}(k,\theta)) \\ (min \begin{cases} \lceil \frac{L-c_j}{T_j} \rceil + 1 \\ \lfloor \frac{T_i}{T_j} \rfloor \end{cases}) . \sum_{\forall s_j^l(\theta)} len(s_j^l(\theta)))) \end{cases} \quad (38)$$

where:-

- $min^* = 0$ if $n \leq m$.
- $T_j^* = \{T_j | (T_j \in \gamma(\theta)) \wedge (D(T_j) \leq D(T_i)\}$
- The "min" term in "min^*" accommodates for the right number of interference of higher priority tasks during the period L (the top term), or the whole period of T_i (the bottom term) as explained before in the analysis of G-EDF with EDF CM.

Workload of other tasks is the same as before.

9 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 [17], 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.

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

We assume that:

$$c_{ij}^{kl} = len(s_j^l(\theta))/len(s_i^k(\theta)) \quad (39)$$

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.

Algorithm 1: LCM

Data: $s_i^k(\theta) \rightarrow$ interfered atomic section.
$s_j^l(\theta) \rightarrow$ interfering atomic section.
$\psi \rightarrow$ predefined threshold $\in [0, 1]$.
$\delta_i^k(\theta) \rightarrow$ remaining execution length of $s_i^k(\theta)$
Result: which atomic section of $s_i^k(\theta)$ or $s_j^l(\theta)$ aborts

1 **if** $p_i^k > p_j^l$ **then**
2 | $s_j^l(\theta)$ aborts;
3 **else**
4 | $c_{ij}^{kl} = len(s_j^l(\theta))/len(s_i^k(\theta))$;
5 | $\alpha_{ij}^{kl} = ln(\psi)/(ln(\psi) - c_{ij}^{kl})$;
6 | $\alpha = \left(len(s_i^k(\theta)) - \delta_i^k(\theta)\right)/len(s_i^k(\theta))$;
7 | **if** $\alpha \leq \alpha_{ij}^{kl}$ **then**
8 | | $s_i^k(\theta)$ aborts;
9 | **else**
10 | | $s_j^l(\theta)$ aborts;
11 | **end**
12 **end**

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).

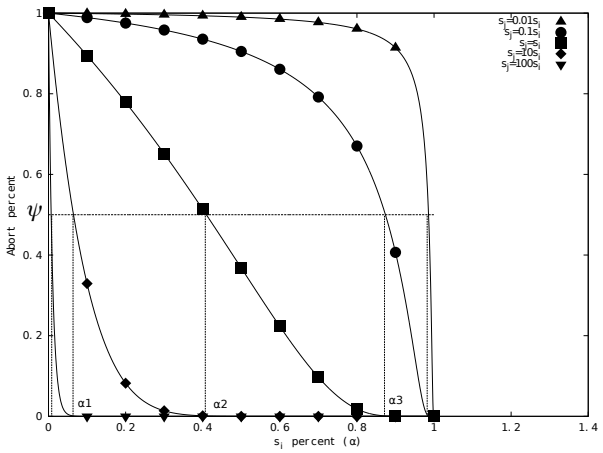


Figure 13: Interference of $s_i^k(\theta)$ by various lengths of $s_j^l(\theta)$

The behavior of LCM is illustrated in Figure 13. 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 (40):

$$f(c_{ij}^{kl}, \alpha) = e^{\frac{-c_{ij}^{kl}\alpha}{1-\alpha}} \tag{40}$$

where c_{ij}^{kl} is calculated by (39).

Figure 13 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 13) of its execution length. As $len(s_j^l(\theta))$ decreases, the corresponding α_{ij}^{kl} increases (as shown in Figure 13, $\alpha 3 > \alpha 2 > \alpha 1$).

Equation (40) achieves the desired requirement that the abort opportunity is reduced as $s_i^k(\theta)$ gets closer to the end of its execution (as $\alpha \to 1$, $f(c_{ij}^{kl}, 1) \to 0$), 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 $s_i^k(\theta)$ 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 centralized CM, which means that, upon a conflict, each transactions has to decide whether it must commit or abort.

Claim 7 *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:*

$$W_i^k(s_j^l(\theta)) \leq \alpha_{ij}^{kl} len\Big(s_i^k(\theta)\Big) + len\Big(s_j^l(\theta)\Big) \tag{41}$$

Proof 7 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 (41). 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.

Claim 8 *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:*

$$W_j^l(s_i^k(\theta)) \leq \Big(1 - \alpha_{ij}^{kl}\Big) len\Big(s_i^k(\theta)\Big) \tag{42}$$

Proof 8 It is derived directly from Claim 7, as $s_j^l(\theta)$ will have to retry for the remaining length of $s_i^k(\theta)$.

Claim 9 *Under LCM, there is no priority inversion.*

Proof 9 Assume 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, 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, $s_a^b(\theta)$ will not abort any one because it is still in its beginning and it is of the lowest priority. Thus, τ_j is not indirectly blocked by τ_a.

- $p_i < p_a < p_j$. Now, even if $s_a^b(\theta)$ interferes with $s_i^k(\theta)$ before α_{ia}^{kb}, and $s_a^b(\theta)$ is allowed to abort $s_i^k(\theta)$, after comparing $s_j^l(\theta)$ and $s_a^b(\theta)$, LCM will select $s_j^l(\theta)$ to commit and abort $s_a^b(\theta)$, because the latter is still at its 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. So, the medium priority instance of τ_a does not increase the delay time of the higher priority instance of τ_j.

- $p_a > p_j > p_i$. Now, if $s_a^b(\theta)$ is chosen to commit, this will not cause a 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 LCM will choose $s_j^l(\theta)$ 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.

Claim 10 *A higher priority job, τ_j^v, can be delayed by lower priority jobs for at most number of atomic sections in τ_j^v.*

Proof 10 By generalizing the cases mentioned in the proof of Claim 9 to any number of conflicting jobs, it can be seen that when an atomic section, $s_j^l(\theta)$, of a higher priority job conflicts with a number of atomic sections belonging to lower priority jobs, $s_j^l(\theta)$ can be delayed only one of them, $s_i^k(\theta)$, for the remaining length of $s_i^k(\theta)$, then $s_j^l(\theta)$ 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.

Claim 11 *The maximum delay suffered by $s_j^l(\theta)$ due to lower priority jobs is caused by the maximum length atomic section accessing object θ, which belongs to a lower priority job than τ_j^b that owns $s_j^l(\theta)$.*

Proof 11 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 (42) 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 (40):

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

$\because ln\psi \leq 0$ and $c_{ij}^{kl}, c_{hj}^{kl} > 0, \therefore$ by substitution from (39)

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

Thus, as the length of the interfered atomic section increases, the delay suffered by the interfering atomic section increases. Claim follows.

9.2 Response Time of G-EDF/LCM

Claim 12 *$RC(T_i)$ for a task τ_i under G-EDF/LCM is upper bounded by:*

$$\begin{aligned} RC(T_i) &= \left(\sum_{\forall \tau_h \in \gamma_i} \sum_{\forall \theta \in \theta_i \wedge \theta_h} \left(\left\lceil \frac{T_i}{T_h} \right\rceil \sum_{\forall s_h^l(\theta)} len\Big(s_h^l(\theta)\Big) \right.\right. \\ &+ \left.\left. \alpha_{max}^{hl} len\Big(s_{max}^h(\theta)\Big) \right) \right) \\ &+ \sum_{\forall s_i^y(\theta)} \Big(1 - \alpha_{max}^{iy}\Big) len\Big(s_{max}^i(\theta)\Big) \end{aligned} \tag{43}$$

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 $s_i^y(\theta)$.

Proof 12 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 14, where one instance of τ_h and τ_h^p coincides with the absolute deadline of τ_i^x.

By using Claims 7, 8, 10, and 11, and Claim 1 in [17] to determine the effect of atomic sections belonging to higher and lower priority instances of interfering tasks to τ_i^x, Claim follows.

Response time of τ_i is calculated by (11) in [17].

9.3 Schedulability of G-EDF/LCM and ECM

We now compare the schedulability of G-EDF/LCM with ECM [17] 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,

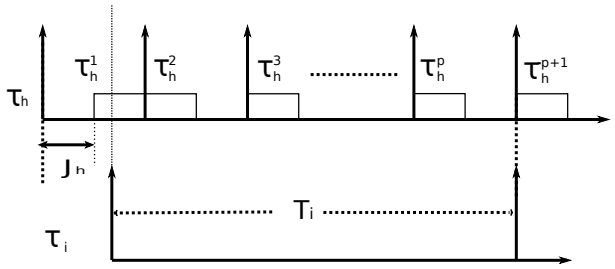


Figure 14: τ_h^p has a higher priority than τ_i^x

then the schedulability of A and B are compared as:

$$\begin{aligned} \sum_{\forall \tau_i} \frac{c_i + RC_A(T_i)}{T_i} &\leq \sum_{\forall \tau_i} \frac{c_i + RC_B(T_i)}{T_i} \\ \sum_{\forall \tau_i} \frac{RC_A(T_i)}{T_i} &\leq \sum_{\forall \tau_i} \frac{RC_B(T_i)}{T_i} \end{aligned} \tag{44}$$

Thus, schedulability is compared by substituting the retry cost added by the synchronization methods in (44).

Claim 13 *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:*

$$\frac{1 - \alpha_{min}}{1 - \alpha_{max}} \leq \sum_{\forall \tau_h \in \gamma_i} \left\lceil \frac{T_i}{T_h} \right\rceil \tag{45}$$

Proof 13 Under ECM, $RC(T_i)$ is upper bounded by:

$$RC(T_i) \leq \sum_{\forall \tau_h \in \gamma_i} \sum_{\forall \theta \in (\theta_i \wedge \theta_h)} \left(\left\lceil \frac{T_i}{T_h} \right\rceil \sum_{\forall s_h^z(\theta)} 2len(s_{max}) \right) \tag{46}$$

with the assumption that all lengths of atomic sections of (4) and (8) in [17] and (43) are replaced by s_{max}. Let α_{max}^{hl} in (43) be replaced with α_{max}, and α_{max}^{iy} in (43) be replaced with α_{min}. As α_{max}, α_{min}, and $len(s_{max})$ are all constants, (43) is upper bounded by:

$$\begin{aligned} RC(T_i) \leq & \left(\sum_{\forall \tau_h \in \gamma_i} \sum_{\forall \theta \in \theta_i \wedge \theta_h} \left(\left\lceil \frac{T_i}{T_h} \right\rceil \sum_{\forall s_h^l(\theta)} (1 + \alpha_{max}) \right. \right. \\ & \left. \left. len\Big(s_{max}\Big) \right) \right) + \sum_{\forall s_i^y(\theta)} \Big(1 - \alpha_{min}\Big) len\Big(s_{max}\Big) \end{aligned} \tag{47}$$

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, (46) becomes:

$$RC(T_i) \leq \sum_{\tau_h \in \gamma_i} 2 \left\lceil \frac{T_i}{T_h} \right\rceil \beta_i len(s_{max}) \tag{48}$$

and (47) becomes:

$$\begin{aligned} RC(T_i) \quad &\leq \quad \beta_i len(s_{max}) \Bigg((1 - \alpha_{min}) \\ &+ \quad \sum_{\forall \tau_h \in \gamma_i} \left\lceil \frac{T_i}{T_h} \right\rceil (1 + \alpha_{max}) \Bigg) \end{aligned} \tag{49}$$

We can now compare the total utilization of G-EDF/LCM with that of ECM by comparing (47) and (49) for all τ_i:

$$\begin{aligned} &\sum_{\forall \tau_i} \frac{(1 - \alpha_{min}) + \sum_{\forall \tau_h \in \gamma_i} \left(\left\lceil \frac{T_i}{T_h} \right\rceil (1 + \alpha_{max}) \right)}{T_i} \\ \leq \quad &\sum_{\forall \tau_i} \frac{\sum_{\forall \tau_h \in \gamma_i} 2 \left\lceil \frac{T_i}{T_h} \right\rceil}{T_i} \end{aligned} \tag{50}$$

(50) 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.

9.4 G-EDF/LCM versus Lock-free

We consider the retry-loop lock-free synchronization for G-EDF given in [13]. This lock-free approach is the most relevant to our work.

Claim 14 *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 [13]. 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).*

Proof 14 From [13], the retry-loop lock-free algorithm is upper bounded by:

$$RL(T_i) = \sum_{\tau_h \in \gamma_i} \left(\left\lceil \frac{T_i}{T_h} \right\rceil + 1 \right) \beta_i r_{max} \tag{51}$$

where β_i is as defined in Claim 13. The retry cost of τ_i in G-EDF/LCM is upper bounded by (49). By comparing G-EDF/LCM's total utilization with that of the retry-loop lock-free algorithm, we get:

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

$$\therefore \frac{s_{max}}{r_{max}} \leq \frac{\sum_{\forall \tau_i} \frac{\sum_{\forall \tau_h \in \gamma_i}\Big(\Big\lceil \frac{T_i}{T_h}\Big\rceil +1\Big)\beta_i}{T_i}}{\sum_{\forall \tau_i} \frac{\Big((1-\alpha_{min})+\sum_{\forall \tau_h \in \gamma_i}\Big(\Big\lceil \frac{T_i}{T_h}\Big\rceil (1+\alpha_{max})\Big)\Big)\beta_i}{T_i}} \tag{52}$$

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:

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

$\therefore$ (52) will be:

$$\frac{s_{max}}{r_{max}} \leq 1 + \frac{\sum_{\forall \tau_i} \frac{\omega-1}{T_i}}{\sum_{\forall \tau_i} \frac{1+\sum_{\forall \tau_h \in \gamma_i}\left\lceil \frac{T_i}{T_h}\right\rceil}{T_i}} \tag{53}$$

By substituting the edge values for ω and $\left\lceil \frac{T_i}{T_h} \right\rceil$ in (53), we derive that the upper bound on s_{max}/r_{max} lies between 1 and 2.

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

(52) becomes

$$\frac{s_{max}}{r_{max}} \leq 0.5 + \frac{\sum_{\forall \tau_i} \frac{\omega-0.5}{T_i}}{\sum_{\forall \tau_i} \frac{1+2\sum_{\forall \tau_h \in \gamma_i}\left\lceil \frac{T_i}{T_h}\right\rceil}{T_i}} \tag{54}$$

By applying the edge values for ω and $\left\lceil \frac{T_i}{T_h} \right\rceil$ in (54), 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 $\alpha_{min} \leq \alpha_{max}$.

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

$\therefore$ (52) becomes:

$$\frac{s_{max}}{r_{max}} \leq 0.5 + \frac{\sum_{\tau_i} \frac{\omega}{T_i}}{2\sum_{\tau_i} \frac{\sum_{\forall \tau_h \in \gamma_i}\left\lceil \frac{T_i}{T_h}\right\rceil}{T_i}} \tag{55}$$

By applying the edge values for ω and $\left\lceil \frac{T_i}{T_h} \right\rceil$ in (55), 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 [17], it lies between 0.5 and 1. Claim follows.

9.5 Response Time of G-RMA/LCM

Claim 15 *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:*

$$\begin{aligned} RC\left(T_i\right) &= \sum_{\forall \tau_j^*} \left(\sum_{\theta \in (\theta_i \wedge \theta_j)} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \lambda_2(j,\theta) \right) \right) \\ &+ \sum_{\forall s_i^y(\theta)} \left(1 - \alpha_{max}^{iy}\right) len\left(s_{max}^i(\theta)\right) \end{aligned} \tag{56}$$

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

Proof 15 Under G-RMA, all instances of a higher priority task, τ_j, can conflict with a lower priority task, τ_i, during T_i. (41) 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 8 and 10 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 [17], the claim follows.

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

9.6 Schedulability of G-RMA/LCM and RCM

Claim 16 *Under the same assumptions of Claims 13 and 15, G-RMA/LCM's schedulability is equal or better than RCM if:*

$$\frac{1 - \alpha_{min}}{1 - \alpha_{max}} \leq \sum_{\forall \tau_j^*} \left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \tag{57}$$

Proof 16 Under the same assumptions as that of Claims 13 and 15, (56) can be upper bounded as:

$$\begin{aligned} RC(T_i) &\leq \sum_{\forall \tau_j^*} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) (1 + \alpha_{max}) len(s_{max}) \beta_i \right) \\ &+ (1 - \alpha_{min}) len(s_{max}) \beta_i \end{aligned} \tag{58}$$

For RCM, (16) in [17] 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:

$$\begin{aligned} & \sum_{\forall \tau_i} \frac{len(s_{max})\beta_i\Big((1-\alpha_{min})+\sum_{\forall \tau_j^*}\Big(\Big\lceil \frac{T_i}{T_j}\Big\rceil+1\Big)(1+\alpha_{max})\Big)\Big)}{T_i} \\ \leq & \sum_{\forall \tau_i} \frac{2len(s_{max})\beta_i \sum_{\forall \tau_j^*}\Big(\Big\lceil \frac{T_i}{T_j}\Big\rceil+1\Big)}{T_i} \end{aligned} \tag{59}$$

(59) is satisfied if $\forall \tau_i$ (57) is satisfied. Claim follows.

9.7 G-RMA/LCM versus Lock-free

Although [13] considers retry-loop lock-free synchronization for G-EDF systems, [13] also applies for G-RMA systems.

Claim 17 *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 [13]. G-RMA/LCM achieves higher schedulability than the retry-loop lock-free approach if the upper bound on s_{max}/r_{max} under G-RMA/LCM is no less than 0.5. Upper bound on s_{max}/r_{max} can extend to large values when α_{min} and α_{max} are very large.*

Proof 17 Retry cost for G-RMA/LCM is upper bounded by (56). Let $\gamma_i = \tau_j^* \cup \bar{\tau_j}$, where τ_j^* is the set of higher priority tasks than τ_i sharing objects with τ_i. $\bar{\tau_j}$ is the set of lower priority tasks than τ_i sharing objects with it. We follow the same definitions of β_i, r_{max} and $RL(T_i)$ given in the proof of Claim (14). Schedulability of G-RMA/LCM equals or exceeds schedulability of retry-loop lock-free if

$$\begin{aligned} \frac{s_{max}}{r_{max}} \leq & \frac{\sum_{\forall \tau_i} \frac{\sum_{\tau_j^*}\Big(\Big\lceil \frac{T_i}{T_j}\Big\rceil+1\Big)}{T_i}}{\sum_{\forall \tau_i} \frac{\Big(1-\alpha_{min}\Big)+\sum_{\tau_j^*}\Big(\Big\lceil \frac{T_i}{T_j}\Big\rceil+1\Big)(1+\alpha_{max})}{T_i}} \\ & + \frac{2\sum_{\forall \tau_i} \frac{\sum_{\forall \bar{\tau_j}}}{T_i}}{\sum_{\forall \tau_i} \frac{\Big(1-\alpha_{min}\Big)+\sum_{\tau_j^*}\Big(\Big\lceil \frac{T_i}{T_j}\Big\rceil+1\Big)(1+\alpha_{max})}{T_i}} \end{aligned} \tag{60}$$

If $p_j < p_i$, $\therefore \left\lceil \frac{T_i}{T_j} \right\rceil = 1$ because the system assumes impilicit deadline tasks and uses G-RMA scheduler. Let ω_1 be size of τ_i^* and ω_2 be size of $\bar{\tau_i}$. $\therefore \omega_1^i \geq 1$ and $\omega_2^i \geq 1$. Otherwise, there is no conflict with τ_i. To find the maximum and minimum upper bounds for s_{max}/r_{max}, the following cases are considered:

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

$$\therefore \frac{s_{max}}{r_{max}} \leq 1 + \frac{\sum_{\forall \tau_i} \frac{2\omega_2^i - 1}{T_i}}{\sum_{\forall \tau_i} \frac{1+\sum_{\tau_j^*}\Big(\Big\lceil \frac{T_i}{T_j}\Big\rceil+1\Big)}{T_i}} \tag{61}$$

As the second term in (61) is alawys positive (because $\omega_2^i \geq 1$), then the minimum upper bound on s_{max}/r_{max} is 1. To get the maximum upper bound on s_{max}/r_{max}, let $\left\lceil \frac{T_i}{T_j} \right\rceil$ approaches its minimum value 1, $\omega_1^i \to 0$ and $\omega_2^i \to n-1$ (the maximum and minimum values for ω_1^i and ω_2^i respectively. n is number of tasks), then

$$\therefore \frac{s_{max}}{r_{max}} \leq (2n-2)$$

Of course, n cannot be lower than 2. Otherwise, there will be no conflicting tasks.

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

$$\frac{s_{max}}{r_{max}} \leq \frac{1}{2} + \frac{\sum_{\forall \tau_i} \frac{4\omega_2^i - 1}{T_i}}{2\sum_{\forall \tau_i} \frac{1+2\sum_{\tau_j^*}\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1\right)}{T_i}} \tag{62}$$

The minimum upper bound for s_{max}/r_{max} is 0.5. This can happen when $T_i \gg T_j$. To get the maximum upper bound on s_{max}/r_{max}, let $\left\lceil \frac{T_i}{T_j} \right\rceil$ approaches its minimum value 1, $\omega_2^i \to n-1$ and $\omega_1^i \to 0$, then

$$\frac{s_{max}}{r_{max}} \leq 2n-2$$

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

This case is rejected because α_{max} must be greater or equal to α_{min}.

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

$$\frac{s_{max}}{r_{max}} \leq \frac{1}{2} + \frac{\sum_{\forall \tau_i} \frac{\omega_2^i}{T_i}}{\sum_{\forall \tau_i} \frac{\sum_{\tau_j^*}\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1\right)}{T_i}} \tag{63}$$

The minimum upper bound for s_{max}/r_{max} is 0.5. This can happen when $T_i \gg T_j$. To get the maximum upper bound on s_{max}/r_{max}, let $\left\lceil \frac{T_i}{T_j} \right\rceil$ approaches its minimum value 1, $\omega_2^i \to n-1$, $\omega_1^i \to 0$, then

$$\frac{s_{max}}{r_{max}} \to \infty$$

From the previous cases, we can derive that upper bound on s_{max}/r_{max} extends from 0.5 to large values. Claim follows.

10 Limitations of ECM, RCM, and LCM

ECM and RCM use dynamic and fixed priorities, respectively, to resolve conflicts. ECM uses G-EDF, and allows the transaction whose job has the earliest absolute deadline to commit first. RCM uses G-RMA, and commits the transaction whose job has the shortest period.

As mentioned before, [16, 17] assumes that each transaction accesses only one object. This assumption simplifies the retry cost (Claims 2 and 3 in [17], and Claims 5, 8 in [16]) and response time analysis (Sections 4 and 5 in [17], and Sections 4.2, 4.5 in [16]). Besides, it enables a one-to-one comparison with lock-free synchronization in [13]. With multiple objects per transaction, ECM, RCM and LCM will face transitive retry, which we illustrate with an example.

Example 1. Consider three atomic sections s_1^x, s_2^y, and s_3^z belonging to jobs τ_1^x,τ_2^y, and τ_3^z, with priorities $p_3^z > p_2^y > p_1^x$, respectively. Assume that s_1^x and s_2^y share objects, s_2^y and s_3^z share objects. s_1^x and s_3^z do not share objects. s_3^z can cause s_2^y to retry, which in turn will cause s_1^x to retry. This means that s_1^x may retry transitively because of s_3^z, which will increase the retry cost of s_1^x.

Assume another atomic section s_4^f is introduced. Priority of s_4^f is higher than priority of s_3^z. s_4^f shares objects only with s_3^z. Thus, s_4^f can make s_3^z to retry, which in turn will make s_2^y to retry, and finally, s_1^x to retry. Thus, transitive retry will move from s_4^f to s_1^x, increasing the retry cost of s_1^x. The situation gets worse as more tasks of higher priorities are added, where each task shares objects with its immediate lower priority task. τ_3^z may have atomic sections that share objects with τ_1^x, but this will not prevent the effect of transitive retry due to s_1^x.

Definition 1 ***Transitive Retry:*** *A transaction s_i^k suffers from transitive retry when it conflicts with a higher priority transaction s_j^l, which in turn conflicts with a higher priority transaction s_z^h, but s_i^k does not conflict with s_z^h. Still, when s_j^l retries due to s_z^h, s_i^k also retries due to s_j^l. Thus, the effect of the higher priority transaction s_z^h is transitively moved to the lower priority transaction s_i^k, even when they do not conflict on common objects.*

Claim 18 *ECM, RCM and LCM suffer from transitive retry for multi-object transactions.*

Proof 18 ECM, RCM and LCM depend on priorities to resolve conflicts between transactions. Thus, lower priority transaction must always be aborted for a conflicting higher priority transaction in ECM and RCM. In LCM, lower priority transactions are conditionally aborted for higher priority ones. Claim follows.

Therefore, the analysis in [17] and [16] must extend the set of objects that can cause an atomic section of a lower priority job to retry. This can be done by initializing the set of conflicting objects, γ_i, to all objects accessed by all transactions of τ_i. We then cycle through all transactions belonging to all other higher priority tasks. Each transaction s_j^l that accesses at least one of the objects in γ_i adds all other objects accessed by s_j^l to γ_i. The loop over all higher priority tasks is repeated, each time with the new γ_i, until there are no more transactions accessing any object in γ_i[1].

In addition to the *transitive retry* problem, retrying higher priority transactions can prevent lower priority tasks from running. This happens when all processors are busy with higher priority jobs. When a transaction retries, the

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

processor time is wasted. Thus, it would be better to give the processor to some other task.

Essentially, what we present is a new contention manager that avoids the effect of transitive retry. We call it, Priority contention manager with Negative values and First access (or PNF). PNF also tries to enhance processor utilization. This is done by allocating processors to jobs with non-retrying transactions. PNF is described in Section 11.

11 The PNF Contention Manager

Algorithm 2 describes PNF. It manages two sets. The first is the m-set, which contains at most m non-conflicting transactions, where m is the number of processors, as there cannot be more than m executing transactions (or generally, m executing jobs) at the same time. When a transaction is entered in the m-set, it executes non-preemptively and no other transaction can abort it. A transaction in the m-set is called an *executing transaction*. This means that, when a transaction is executing before the arrival of higher priority conflicting transactions, then the one that started executing first will be committed (Step 8).

Algorithm 2: PNF Algorithm

Data: *Executing Transaction:* is one that cannot be aborted by any other transaction, nor preempted by a higher priority task;
m-set: m-length set that contains only non-conflicting executing transactions;
n-set: n-length set that contains retrying transactions for n tasks in non-increasing order of priority;
n(z): transaction at index z of the n-set;
s_i^k: a newly released transaction;
s_j^l: one of the executing transactions;
Result: atomic sections that will commit

1 **if** s_i^k *does not conflict with any executing transaction* **then**
2 Assign s_i^k as an executing transaction;
3 Add s_i^k to the m-set;
4 Select s_i^k to commit
5 **else**
6 Add s_i^k to the n-set according to its priority;
7 Assign temporary priority -1 to the job that owns s_i^k ;
8 Select transaction(s) conflicting with s_i^k for commit;
9 **end**
10 **if** s_j^l *commits* **then**
11 **for** *z=1 to size of n-set* **do**
12 **if** *n(z) does not conflict with any executing transaction* **then**
13 **if** *processor available*[2] **then**
14 Restore priority of task owning n(z);
15 Assign n(z) as executing transaction;
16 Add n(z) to m-set and remove it from n-set;
17 Select n(z) for commit;
18 **else**
19 Wait until processor available
20 **end**
21 **end**
22 move to the next n(z);
23 **end**
24 **end**

The second set is the n-set, which holds the transactions that are retrying because of a conflict with one or more of the executing transactions (Step 6), where n stands for the number of tasks in the system. Transactions in the n_set are known as *retrying transaction.*It also holds transactions that cannot currently execute, because processors are busy, either due to processing executing transactions and/or higher priority jobs. Any transaction in the n-set is assigned a temporal priority of -1 (Step 7) (hence the word "Negative" in the algorithm's name). A negative priority is considered smaller than any normal priority, and a transaction continues to hold this negative priority until it is moved to the m-set, where it is restored its normal priority.

A job holding a transaction in the n-set can be preempted by any other job with normal priority, even if that job does not have transactions conflicting with the preempted job. Hence, this set is of length n, as there can be at most n jobs. Transactions in the n-set whose jobs have been preempted are called preempted transactions. The n-set list keeps track of preempted transactions, because as it will be shown, all preempted and non-preempted transactions in the n-set are examined when any of the executing transaction commits. Then, one or more transactions are selected from the n-set to be executing transactions. If a retrying transaction is selected as an executing transaction, the task that owns the retrying transaction regains its priority.

When a new transaction is released, and if it does not conflict with any of the executing transactions (Step 1), then it will allocate a slot in the m-set and becomes an executing transaction. When this transaction is released (i.e., its containing task is already allocated to a processor), it will be able to access a processor immediately. This transaction may have a conflict with any of the transactions in the n-set. However, since transactions in the n-set have priorities of -1, they cannot prevent this new transaction from executing if it does not conflict with any of the executing transactions.

When one of the executing transactions commits (Step 10), it is time to select one of the n-set transactions to commit. The n-set is traversed from the highest priority to the lowest priority (priority here refers to the original priority of the transactions, and not -1) (Step 11). If an examined transaction in the n-set, s_h^b, does not conflict with any executing transaction (Step 12), and there is an available processor for it (Step 13) ("available" means either an idle processor, or one that is executing a job of lower priority than s_h^b), then s_h^b is moved from the n-set to the m-set as an executing transaction and its original priority is restored. If s_h^b is added to the m-set, the new m-set is compared with other transactions in the n-set with lower priority than s_h^b. Hence, if one of the transactions in the n-set, s_d^g, is of lower priority than s_h^b and conflicts with s_h^b, it will remain in the n-set.

The choice of the new transaction from the n-set depends on the original priority of transactions (hence the term "P" in the algorithm name). The algorithm avoids interrupting an already executing transaction to reduce its retry cost. In the meanwhile, it tries to avoid delaying the highest priority transaction in the n-set when it is time to select a new one to commit, even if the highest priority transaction arrives after other lower priority transactions in the n-set.

[2]An idle processor or at least one that runs a non-atomic section task with priority lower than the task holding $n(z)$.

11.1 Illustrative Example

We illustrate PNF with an example. We use the following notions: $s_a^b \in \tau_a^k$ is transaction s_a^b in job τ_a^k. $s_a^b(\theta_1, \theta_2, \theta_3)$ means that s_a^b accesses objects $\theta_1, \theta_2, \theta_3$. $p(s_a^b)$ is the priority of transaction s_a^b. p_i^j is the priority of job τ_i^j. If $s_a^b \in \tau_a^j$, $\therefore p_o(s_a^b) = p_a^j$, where $p_o(s_a^b)$ is the original priority of s_a^b. $p(s_a^b) = -1$, if s_a^b is a retrying transaction; $p(s_a^b) = p_o(s_a^b)$ otherwise. m-set= $\{s_a^b, s_i^k\}$ means that the m-set contains transactions s_a^b and s_i^k regardless of their order. n-set= $\{s_a^b, s_i^k\}$ means that the n-set contains transactions s_a^b and s_i^k in that order, where $p_o(s_a^b) > p_o(s_i^k)$. m-set (n-set) $= \{\phi\}$ means that m-set (n-set) is empty. Assume there are five processors.

1. Initially, m-set= n-set= $\{\phi\}$. $s_a^b(\theta_1, \theta_2) \in \tau_a^b$ is released and checks m-set for conflicting transactions. As m-set is empty, s_a^b finds no conflict and becomes an executing transaction. s_a^b is added to m-set. m-set= $\{s_a^b\}$ and n-set= $\{\phi\}$. s_a^b is executing on processor 1.
2. $s_c^d(\theta_3, \theta_4) \in \tau_c^d$ is released and checks m-set for conflicting transactions. s_c^d does not conflict with s_a^b as they access different objects. s_c^d becomes an executing transaction and is added to m-set. m-set= $\{s_a^b, s_c^d\}$ and n-set= $\{\phi\}$. s_c^d is executing on processor 2.
3. $s_e^f(\theta_1, \theta_5) \in \tau_e^f$ is released and $p_o(s_e^f) < p_o(s_a^b)$. s_e^f conflicts with s_a^b when it checks m-set. s_e^f is added to n-set and becomes a retrying transaction. $p(s_e^f)$ becomes -1. m-set= $\{s_a^b, s_c^d\}$ and n-set= $\{s_e^f\}$. s_e^f is retrying on processor 3.
4. $s_g^h(\theta_1, \theta_6) \in \tau_g^h$ is released and $p_o(s_g^h) > p_o(s_a^b)$. s_g^h conflicts with s_a^b. Though s_g^h is of higher priority than s_a^b, s_a^b is an executing transaction. So s_a^b runs non-preemptively. s_g^h is added to n-set before s_e^f, because $p_o(s_g^h) > p_o(s_e^f)$. $p(s_g^h)$ becomes -1. m-set= $\{s_a^b, s_c^d\}$ and n-set= $\{s_g^h, s_e^f\}$. s_g^h is retrying on processor 4.
5. $s_i^j(\theta_5, \theta_7) \in \tau_i^j$ is released. $p_o(s_i^j) < p_o(s_e^f)$. s_i^j does not conflict with any transaction in m-set. Though s_i^j conflicts with s_e^f and $p_o(s_i^j) < p_o(s_e^f) < p_o(s_g^h)$, s_e^f and s_g^h are retrying transactions. s_i^j becomes an executing transaction and is added to m-set. m-set= $\{s_a^b, s_c^d, s_i^j\}$ and n-set= $\{s_g^h, s_e^f\}$. s_i^j is executing on processor 5.
6. τ_k^l is released. τ_k^l does not access any object. $p_k^l < p_o(s_e^f) < p_o(s_g^h)$, but $p(s_e^f) = p(s_g^h) = -1$. Since there are no more processors, τ_k^l preempts τ_e^f, because the currently assigned priority to $\tau_e^f = p(s_e^f) = -1$ and $p_o(s_g^h) > p_o(s_e^f)$. τ_k^l is running on processor 3. This way, PNF optimizes processor usage. The m-set and n-set are not changed. Although s_e^f is preempted, n-set still records it, as s_e^f might be needed (as will be shown in the following steps).
7. s_i^j commits. s_i^j is removed from m-set. Transactions in n-set are checked from the first (highest p_o) to the last (lowest p_o) for conflicts against any executing transaction. s_g^h is checked first because $p_o(s_g^h) > p_o(s_e^f)$. s_g^h conflicts with s_a^b, so s_g^h cannot be an executing transaction. Now it is time to check s_e^f, even though s_e^f is preempted in step 6. s_e^f also conflicts with s_a^b, so s_e^f cannot be an executing transaction. m-set= $\{s_a^b, s_c^d\}$ and n-set= $\{s_g^h, s_e^f\}$. Now, s_e^f can be retrying on processor 5 if τ_i^j has finished execution. Otherwise, τ_i^j continues running on processor 5 and s_e^f is still

preempted. This is because, $p(s_e^f) = -1$ and $p_i^j > p(s_e^f)$. Let us assume that τ_i^j is still running on processor 5.

8. s_a^b commits. s_a^b is removed from m-set. Transactions in n-set are checked as done in step 7. s_g^h does not conflict with any executing transaction any more. s_g^h becomes an executing transaction. s_g^h is removed from n-set and added to m-set, so m-set= $\{s_c^d, s_g^h\}$. Now, s_e^f is checked against the new m-set. s_e^f conflicts with s_g^h, so s_e^f cannot be an executing transaction. s_e^f can be retrying on processor 1 if τ_a^b has finished execution. Otherwise, s_e^f remains preempted, because $p(s_e^f) = -1$ and $p_a^b > p(s_e^f)$. n-set= $\{s_e^f\}$. Let us assume that τ_a^b is still running on processor 1.
9. s_g^h commits. s_g^h is removed from m-set. τ_g^h continues execution on processor 4. Transactions in n-set are checked again. s_e^f is the only retrying transaction in the n-set, and it does not conflict with any executing transactions. Now, the system has τ_a^b running on processor 1, s_c^d executing on processor 2, τ_k^l running on processor 3, τ_g^h running on processor 4, and τ_i^j running on processor 5. s_e^f can become an executing transaction if it can find a processor. Since p_i^j, $p_k^l < p_o(s_e^f)$, s_e^f can preempt the lowest in priority between τ_i^j and τ_k^l. s_e^f now becomes an executing transaction. s_e^f is removed from the n-set and added to the m-set. So, m-set= $\{s_c^d, s_e^f\}$ and n-set= $\{\phi\}$. If p_i^j, p_k^l were of higher priority than $p_o(s_e^f)$, then s_e^f would have remained in n-set until a processor becomes available.

The example shows that PNF avoids transitive retry. This is illustrated in step 5, where $s_i^j(\theta_5, \theta_7)$ is not affected by the retry of $s_e^f(\theta_1, \theta_5)$. The example also explains how PNF optimizes processor usage. This is illustrated in step 6, where the retrying transaction s_e^f is preempted in favor of τ_k^l.

11.2 Properties

Claim 19 *Transactions scheduled under PNF do not suffer from transitive retry.*

Proof 19 Proof is by contradiction. Assume that a transaction s_i^k is retrying because of a higher priority transaction s_j^l, which in turn is retrying because of another higher priority transaction s_z^h. Assume that s_i^k and s_z^h do not conflict, yet, s_i^k is transitively retrying due to s_z^h. Note that s_z^h and s_j^l cannot exit together in the m-set as they have shared objects. But they both can be in the n-set, as they can conflict with other *executing transactions*. We have three cases:

Case 1: Assume that s_z^h is an executing transaction. This means that s_j^l is in the n-set. When s_i^k arrives, by the definition of PNF, it will be compared with the m-set, which contains s_z^h. Now, it will be found that s_i^k does not conflict with s_z^h. Also, by the definition of PNF, s_i^k is not compared with transactions in the n-set. When it newly arrives, priorities of n-set transactions are lower than any normal priority. Therefore, as s_i^k does not conflict with any other executing transaction, it joins the m-set and becomes an *executing transaction*. This contradicts the assumption that s_i^k is transitively retrying because of s_z^h.

Case 2: Assume that s_z^h is in the n-set, while s_j^l is an executing transaction. When s_i^k arrives, it will conflict with s_j^l and joins the n-set. Now, s_i^k retries due to s_j^l, and not s_z^h. When s_j^l commits, the n-set is traversed from the

highest priority transaction to the lowest one: if s_z^h does not conflict with any other executing transaction and there are available processors, s_z^h becomes an executing transaction. When s_i^k is compared with the m-set, it is found that it does not conflict with s_z^h. Additionally, if it also does not conflict with any other executing transaction and there are available processors, then s_i^k becomes an executing transaction. This means that s_i^k and s_z^h are executing concurrently, which violates the assumption of transitive retry.

Case 3: Assume that s_z^h and s_j^l both exist in the n-set. When s_i^k arrives, it is compared with the m-set. If s_i^k does not conflict with any executing transactions and there are available processors, then s_i^k becomes an executing transaction. Even though s_i^k has common objects with s_j^l, s_i^k is not compared with s_j^l, which is in the n-set. If s_i^k joins the n-set, it is because, it conflicts with one or more executing transactions, not because of s_z^h, which violates the transitive retry assumption. If the three transactions s_i^k, s_j^l and s_z^h exist in the n-set, and s_z^h is chosen as a new executing transaction, then s_j^l remains in the n-set. This leads to Case 1. If s_j^l is chosen, because s_z^h conflicts with another executing transaction and s_j^l does not, then this leads to Case 2.

Claim 20 *The first access property of PNF prevents transitive retry.*

Proof 20 The proof is by contradiction. Assume that the retry cost of transactions in the absence of the first access property is the same as when first access exists. Now, assume that PNF is devoid of the first access property. This means that executing transactions can be aborted.

Assume three transactions s_i^k, s_j^l, and s_z^h, where s_z^h's priority is higher than s_j^l's priority, and s_j^l's priority is higher than s_i^k's priority. Assume that s_j^l conflicts with both s_i^k and s_z^h. s_i^k and s_z^h do not conflict together. If s_i^k arrives while s_z^h is an executing transaction and s_j^l exists in the n-set, then s_i^k becomes an executing transaction itself while s_j^l is retrying. If s_i^k did not commit at least when s_z^h commits, then s_j^l becomes an executing transaction. Due to the lack of the first access property, s_j^l will cause s_i^k to retry. So, the retry cost for s_i^k will be $len(s_z^h + s_j^l)$. This retry cost for s_i^k is the same if it had been transitively retrying because of s_z^h. This contradicts the first assumption. Claim follows.

From Claims 19 and 20, PNF does not increase the retry cost of multi-object transactions. However, this is not the case for ECM and RCM as shown by Claim 18.

Claim 21 *Under PNF, any job τ_i^x is not affected by the retry cost in any other job τ_j^l.*

Proof 21 As explained in Section 2, PNF assigns a temporary priority of -1 to any job that includes a retrying transaction. So, retrying transactions have lower priority than any other normal priority. When τ_i^x is released and τ_j^l has a retrying transaction, τ_i^x will have a higher priority than τ_j^l. Thus, τ_i^x can run on any available processor while τ_j^l is retrying one of its transactions. Claim follows.

12 Retry Cost under PNF

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

Claim 22 *Under PNF, the maximum retry cost suffered by a transaction s_i^k due to a transaction s_j^l is $len(s_j^l)$.*

Proof 22 By PNF's definition, s_i^k cannot have started before s_j^l. Otherwise, s_i^k would have been an executing transaction and s_j^l cannot abort it. So, the earliest release time for s_i^k would have been just after s_j^l starts execution. Then, s_i^k would have to wait until s_j^l commits. Claim follows.

Claim 23 *The retry cost for any job τ_i^x due to conflicts between its transactions and transactions of other jobs under PNF during an interval $L \leq T_i$ is upper bounded by:*

$$RC(L) \leq \sum_{\tau_j \in \gamma_i} \left(\sum_{\theta \in \theta_i} \left(\left(\left\lceil \frac{L}{T_j} \right\rceil + 1 \right) \sum_{\forall \bar{s_j^l}(\theta)} len\left(\bar{s_j^l}(\theta) \right) \right) \right) \tag{64}$$

where $\bar{s_j^l}(\theta)$ is the same as $s_j^l(\theta)$ except for the following difference: if $\bar{s_j^l}$ accesses multiple objects in θ_i, then $\bar{s_j^l}$ is included only once in the last summation (i.e., $\bar{s_j^l}$ is not repeated for each shared object with s_i^k).

Proof 23 Consider a transaction s_i^k belonging to job τ_i^x. Under PNF, higher priority transactions than s_i^k can become executing transaction before s_i^k. A lower priority transaction s_v^f can also become an executing transaction before s_i^k. This happens when s_i^k conflicts with any executing transaction while s_v^f does not. The worst case scenario for s_i^k occurs when s_i^k has to wait in the n-set, while all other conflicting transactions with s_i^k are chosen to be executing transactions. Let $\bar{s_j^l}$ accesses multiple objects in θ_i. If $\bar{s_j^l}$ is an executing transaction, then $\bar{s_j^l}$ will not repeat itself for each object it accesses. Besides, $\bar{s_j^l}$ will finish before s_i^k starts execution. Consequently, $\bar{s_j^l}$ will not conflict with s_i^{k+1}. This means that an executing transaction can force no more than one transaction in a given job to retry. This is why $\bar{s_j^l}$ is included only once in (64) for all shared objects with s_i^k.

The maximum number of jobs of any task τ_j that can interfere with τ_i^x during interval L is $\left\lceil \frac{L}{T_j} \right\rceil + 1$. From the previous observations and Claim 22, Claim follows.

Claim 24 *The blocking time for a job τ_i^x due to lower priority jobs during an interval $L \leq T_i$ is upper bounded by:*

$$D(\tau_i^x) \leq \left\lfloor \frac{1}{m} \sum_{\forall \bar{\tau_j^l}} \left(\left(\left\lceil \frac{L}{T_j} \right\rceil + 1 \right) \sum_{\forall \ddot{s_j^h}} len\left(\ddot{s_j^h} \right) \right) \right\rfloor \tag{65}$$

where $D(\tau_i^x)$ is the blocking time suffered by τ_i^x due to lower priority jobs. $\bar{\tau_j^l} = \{\tau_j^l : p_j^l < p_i^x\}$ and $\ddot{s_j^h} = \{s_j^h : s_j^h$ does not conflict with any $s_i^k\}$. During this blocking time, all processors are unavailable for τ_i^x.

Proof 24 Under PNF, executing transactions are non preemptive. So, a lower priority executing transaction can delay a higher priority job τ_i^x if no other processors are available. Lower priority executing transactions can be conflicting or non-conflicting with any transaction in τ_i^x. They also can exist when τ_i^x is newly released, or after that. So, we have the following cases:

Lower priority conflicting transactions after τ_i^x is released: This case is already covered by the retry cost in (64).

Lower priority conflicting transactions when τ_i^x is newly released: Each lower priority conflicting transaction s_j^h will delay τ_i^x for $len(s_j^h)$. The effect of s_j^h is already covered by (64). Besides, (64) does not divide the retry cost by m as done in (65). Thus, the worst case scenario requires inclusion of s_j^h in (64), and not in (65).

Lower priority non-conflicting transactions when τ_i^x is newly released: τ_i^x is delayed if there are no available processors for it. Otherwise, τ_i^x can run in parallel with these non-conflicting lower priority transactions. Each lower priority non-conflicting transaction $\ddot{s_j^h}$ will delay τ_i^x for $len(\ddot{s_j^h})$.

Lower priority non-conflicting transactions after τ_i^x is released: This situation can happen if τ_i^x is retrying one of its transactions s_i^k. So, τ_i^x is assigned a priority of -1. τ_i^x can be preempted by any other job. When s_i^k is checked again to be an executing transaction, all processors may be busy with lower priority non-conflicting transaction and/or higher priority jobs. Otherwise, τ_i^x can run in parallel with these lower priority non-conflicting transactions.

Each lower priority non-conflicting transaction $\ddot{s_j^h}$ will delay τ_i^x for $len(\ddot{s_j^h})$.

From the previous cases, lower priority non-conflicting transactions act as if they were higher priority jobs interfering with τ_i^x. So, the blocking time can be calculated by the interference workload given by Theorem 7 in [4].

Claim 25 *The response time of a job τ_i^x, during an interval $L \leq T_i$, under PNF/G-EDF is upper bounded by:*

$$R_i^{up} = c_i + RC(L) + D_{edf}(\tau_i^x) + \left\lfloor \frac{1}{m} \sum_{\forall j \neq i} W_{ij}(R_i^{up}) \right\rfloor \tag{66}$$

where $RC(L)$ is calculated by (64). $D_{edf}(\tau_i^x)$ is the same as $D(\tau_i^x)$ defined in (65). However, for G-EDF systems. $D_{edf}(\tau_i^x)$ is calculated as:

$$D_{edf}(\tau_i^x) \leq \left\lfloor \frac{1}{m} \sum_{\forall \bar{\tau_j^l}} \begin{cases} 0 & , L \leq T_i - T_j \\ \sum_{\forall \ddot{s_j^h}} len\left(\ddot{s_j^h}\right) & , L > T_i - T_j \end{cases} \right\rfloor \tag{67}$$

and $W_{ij}(R_i^{up})$ is calculated by (3) in [17].

Proof 25 Response time for τ_i^x is calculated as (3) in [17] with the addition of blocking time defined by Claim 24. G-EDF uses absolute deadlines for scheduling. This defines which jobs of the same task can be of lower priority than τ_i^x,

and which will not. Any instance τ_j^h, released between $r_i^x - T_j$ and $d_i^x - T_j$, will be of higher priority than τ_i^x. Before $r_i^x - T_j$, τ_j^h would have finished before τ_i^x is released. After $d_i^x - T_j$, d_j^h would be greater than d_i^x. Thus, τ_j^h will be of lower priority than τ_i^x. So, during T_i, there can be only one instance τ_j^h of τ_j with lower priority than τ_i^x. τ_j^h is released between $d_i^x - T_j$ and d_i^x. Consequently, during $L < T_i - T_j$, no existing instance of τ_j is of lower priority than τ_i^x. Hence, 0 is used in the first case of (67). But if $L > T_i - T_j$, there can be only one instance τ_j^h of τ_j with lower priority than τ_i^x. Hence, $\left\lceil \frac{L}{T_i} \right\rceil + 1$ in (65) is replaced with 1 in the second case in (67). Claim follows.

Claim 26 *The response time of a job τ_i^x, during an interval $L \leq T_i$, under PNF/G-RMA is upper bounded by:*

$$R_i^{up} = c_i + RC(L) + D(\tau_i^x) + \left\lfloor \frac{1}{m} \sum_{\forall j \neq i, p_j > p_i} W_{ij}(R_i^{up}) \right\rfloor \tag{68}$$

where $RC(L)$ is calculated by (64), $D(\tau_i^x)$ is calculated by (65), and $W_{ij}(R_i^{up})$ is calculated by (2) in [17].

Proof 26 Proof is same as of Claim 25, except that G-RMA assigns fixed priorities. Hence, (65) can be used directly for calculating $D(\tau_i^x)$ without modifications. Claim follows.

13 PNF vs. Competitors

We now (formally) compare the schedulability of G-EDF (G-RMA) with PNF against ECM, RCM, LCM and lock-free synchronization [13, 16, 17]. Such a comparison will reveal when PNF outperforms others. Toward this, we compare the total utilization under G-EDF (G-RMA)/PNF, with that under the other synchronization methods. Inflated execution time of each method, which is the sum of the worst-case execution time of the task and its retry cost, is used in the utilization calculation of each task.

By Claim 24, no processor is available for τ_i^x during the blocking time. As each processor is busy with some other job than τ_i^x, $D(\tau_i^x)$ is not added to the inflated execution time of τ_i^x. Hence, $D(\tau_i^x)$ is not added to the utilization calculation of τ_i^x.

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

$$\begin{aligned} \sum_{\forall \tau_i} \frac{c_i + RC_A(T_i)}{T_i} &\leq \sum_{\forall \tau_i} \frac{c_i + RC_B(T_i)}{T_i} \\ \therefore \sum_{\forall \tau_i} \frac{RC_A(T_i)}{T_i} &\leq \sum_{\forall \tau_i} \frac{RC_B(T_i)}{T_i} \end{aligned} \tag{69}$$

As described in Section 10, the set of common objects needs to be extended under PNF's competitors. Toward this, we introduce a few additional notions.

Let θ_i^{ex} be an extended set of distinct objects that contains all objects in θ_i. Thus, θ_i^{ex} contains all objects accessed by τ_i. θ_i^{ex} can also contain other objects that can cause any transaction in τ_i to retry, as discussed in Section 10. Thus, θ_i^{ex} may contain objects not accessed by τ_i. γ_i^{ex} is an extended set of tasks that access any object in θ_i^{ex}. i.e., γ_i^{ex} contains at least all tasks in γ_i.

There are two sources of retry cost for any τ_i^x under ECM, RCM, LCM and lock-free. First is due to conflict between τ_i^x's transactions and transactions of other jobs. This is denoted as RC. Second is due to the preemption of any transaction in τ_i^x due to the release of a higher priority job τ_j^h. This is denoted as RC_{re}. Retry due to the release of higher priority jobs do not occur under PNF, because executing transactions are non-preemptive. It is up to the implementation of the contention manager to safely avoid RC_{re}. Here, we assume that ECM, RCM and LCM do not avoid RC_{re}. Thus, we introduce RC_{re} for ECM, RCM and LCM first before comparing PNF with other techniques.

Claim 27 *Under ECM and G-EDF/LCM the total retry cost suffered by all transactions in any τ_i^x during an interval $L \leq T_i$ is upper bounded by:*

$$RC_{to}(L) = RC(L) + RC_{re}(L) \tag{70}$$

where $RC(L)$ is the retry cost resulting from conflict between transactions in τ_i^x and transactions of other jobs. $RC(L)$ is calculated by (15) in [17] for ECM and (5) in [16] for G-EDF/LCM. γ_i and θ_i are replaced with γ_i^{ex} and θ_i^{ex}, respectively. $RC_{re}(L)$ is the retry cost resulting from the release of higher priority jobs, which preempt τ_i^x. $RC_{re}(L)$ is:

$$RC_{re}(L) = \sum_{\forall \tau_j \in \zeta_i} \begin{cases} \left\lceil \frac{L}{T_j} \right\rceil s_{i_{max}} & , L \leq T_i - T_j \\ \left\lfloor \frac{T_i}{T_j} \right\rfloor s_{i_{max}} & , L > T_i - T_j \end{cases} \tag{71}$$

where $\zeta_i = \{\tau_j : (\tau_j \neq \tau_i) \wedge (D_j < D_i)\}$.

Proof 27 Two conditions must be satisfied for any τ_j^l to be able to preempt τ_i^x under G-EDF: $r_i^x < r_j^l < d_i^x$, and $d_j^l \leq d_i^x$. Without the first condition, τ_j^l would have been already released before τ_i^x. Thus, τ_j^l will not preempt τ_i^x. Without the second condition, τ_j^l will be of lower priority than τ_i^x and will not preempt it. If $D_j \geq D_i$, then there will be at most one instance τ_j^l with higher priority than τ_i^x. τ_j^l must have been released at most at r_i^x, which violates the first condition. The other instance τ_j^{l+1} would have an absolute deadline greater than d_i^x. This violates the second condition. Hence, only tasks with shorter relative deadline than D_i are considered. These jobs are grouped in ζ_i.

The total number of released instances of τ_j during any interval $L \leq T_i$ is $\left\lceil \frac{L}{T_i} \right\rceil + 1$. The "carried-in" jobs (i.e., each job released before r_i^x and has an absolute deadline before d_i^x [4]) are discarded as they violate the first condition. The "carried-out" jobs (i.e., each job released after r_i^x and has an absolute deadline after d_i^x [4]) are also discarded because they violate the second condition. Thus, the number of considered higher priority instances of τ_j during the interval $L \leq T_i - T_j$ is $\left\lceil \frac{L}{T_j} \right\rceil$. The number of considered higher priority instances of τ_j during interval $L > T_i - T_j$ is $\left\lfloor \frac{T_i}{T_j} \right\rfloor$.

The worst RC_{re} for τ_i^x occurs when τ_i^x is always interfered at the end of execution of its longest atomic section, $s_{i_{max}}$. τ_i^x will have to retry for $len(s_{i_{max}})$. The total retry cost suffered by τ_i^x is the combination of RC and RC_{re}.

Claim 28 *Under RCM and G-RMA/LCM, the total retry cost suffered by all transactions in any τ_i^x during an interval $L \leq T_i$ is upper bounded by:*

$$RC_{to}(L) = RC(L) + RC_{re}(L) \tag{72}$$

where $RC(L)$ and $RC_{re}(L)$ are defined in Claim 27. $RC(L)$ is calculated by (16) in [17] for RCM, and (8) in [16] for G-RMA/LCM. $RC_{re}(L)$ is calculated by:

$$RC_{re}(L) = \sum_{\forall \tau_j \in \zeta_i^*} \left(\left\lceil \frac{L}{T_j} \right\rceil s_{i_{max}} \right) \tag{73}$$

where $\zeta_i^* = \{\tau_j : p_j > p_i\}$.

Proof 28 The proof is the same as that for Claim 27, except that G-RMA uses static priority. Thus, the carried-out jobs will be considered in the interference with τ_i^x. The carried-in jobs are still not considered because they are released before r_i^x. Claim follows.

Claim 29 *Consider lock-free synchronization. Let $r_{i_{max}}$ be the maximum execution cost of a single iteration of any retry loop of τ_i. RC_{re} under G-EDF with lock-free synchronization is calculated by (71), where $s_{i_{max}}$ is replaced by $r_{i_{max}}$. RC_{re} under G-RMA with lock-free synchronization is calculated by (73), where $s_{i_{max}}$ is replaced by $r_{i_{max}}$.*

Proof 29 The interference pattern of higher priority jobs to lower priority jobs is the same in ECM, G-EDF/LCM, and G-EDF with lock-free. The pattern is also the same in RCM, G-RMA/LCM, and G-RMA with lock-free.

13.1 PNF versus ECM

Claim 30 *In the absence of transitive retry, PNF/G-EDF's schedulability is better or equal to ECM's when conflicting atomic sections have equal lengths.*

Proof 30 Substitue $RC_A(T_i)$ and $RC_B(T_i)$ in (69) with (64) and (70), respectively. Let $\theta_i^{ex} = \theta_i + \theta_i^*$, where θ_i^* is the set of objects not accessed directly by τ_i but can cause transactions in τ_i to retry due to transitive retry. Let $\gamma_i^{ex} = \gamma_i + \gamma_i^*$, where γ_i^* is the set of tasks that access objects in θ_i^*. Let:

$$\begin{aligned} g(\tau_i) \quad = \quad & \left(\sum_{\forall \tau_j \in \gamma_i^*} \sum_{\theta \in \theta_i^*} \left(\left\lceil \frac{T_i}{T_j} \right\rceil \sum_{\forall \bar{s_j^k}(\theta)} len\Big(\bar{s_j^k}(\theta) \right. \right. \\ + \quad & \left. \left. s_{max}(\theta)\Big) \right) \right) + RC_{re}(T_i) \end{aligned}$$

where RC_{re} is given by (71). $g(\tau_i)$ includes effect of transitive retry. Let:

$$\eta_1(\tau_i) = \sum_{\forall \tau_j \in \gamma_i} \sum_{\forall \theta \in \theta_i} \left(\sum_{\forall \bar{s_j^k}(\theta)} len\left(\bar{s_j^k}(\theta)\right) \right)$$

$$\eta_2(\tau_i) = \sum_{\forall \tau_j \in \gamma_i} \sum_{\forall \theta \in \theta_i} \left(\left\lceil \frac{T_i}{T_j} \right\rceil \sum_{\forall s_j^{\bar{k}}(\theta)} len\left(s_{max}^j(\theta)\right) \right)$$

$$\eta_3(\tau_i) = \sum_{\forall \tau_j \in \gamma_i} \sum_{\forall \theta \in \theta_i} \left(\left\lceil \frac{T_i}{T_j} \right\rceil \sum_{\forall s_j^{\bar{k}}(\theta)} len\left(s_j^{\bar{k}}(\theta)\right) \right)$$

By substitution of $g(\tau_i)$, $\eta_1(\tau_i)$, and $\eta_2(\tau_i)$, and subtraction of $\sum_{\forall \tau_i} \frac{\eta_3(\tau_i)}{T_i}$ from both sides of (69), we get:

$$\sum_{\forall \tau_i} \frac{\eta_1(\tau_i)}{T_i} \leq \sum_{\forall \tau_i} \frac{\eta_2(\tau_i) + g(\tau_i)}{T_i} \tag{74}$$

Assume that $g(\tau_i)_{\forall \tau_i} \to 0$. From (74), we note that by keeping every $len(s_j^{\bar{k}}(\theta)) \leq len(s_{max}^j(\theta))$ for each τ_i, $\tau_j \in \gamma_i$, and $\theta \in \theta_i$, (74) holds. Due to G-EDF's dynamic priority, $s_{max}^j(\theta)$ can belong to any task other than τ_j. By keeping $len(s_j^{\bar{k}}(\theta)) \leq len(s_{max}^j(\theta))$, then 74 holds. By generalizing this condition to any $s_j^k(\theta)$ and $s_{max}^j(\theta)$, then 74 holds if all atomic sections in all tasks have equal lengths. Claim follows.

13.2 PNF versus RCM

Claim 31 *In the absence of transitive retry, PNF/G-RMA's schedulability is better or equal to RCM's schedulability when a large number of tasks heavily conflict. PNF's schedulability is improved compared with RCM's, when atomic section length increases as priority increases.*

Proof 31 Let $\theta_i^{ex} = \theta_i + \theta_i^*$ and $\gamma_i^{ex} = \gamma_i + \gamma_i^*$, as defined in the proof of Claim 30. Substitute $RC_A(T_i)$ and $RC_B(T_i)$ in (69) with (64) and (72), respectively. Let:

$$g(\tau_i) \quad = RC_{re}(T_i) + \Bigg(\sum_{\forall \tau_j \in (\gamma_i^* \cap \zeta_i^*)} \sum_{\forall \theta \in \theta_i^*} \left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \times \sum_{\forall s_j^{\bar{k}}(\theta)} len\left(s_j^{\bar{k}}(\theta) + s_{max}^j(\theta)\right) \Bigg)$$

where RC_{re} and ζ_i^* are defined by (73). $g(\tau_i)$ includes effect of transitive retry. Let $\gamma_i = \zeta_i^* \cup \bar{\zeta_i}$, where $\bar{\zeta_i} = \{\tau_j : (\tau_j \neq \tau_i) \wedge (p_j < p_i)\}$, thus $\zeta_i^* \cap \bar{\zeta_i} = \phi$.

Let:

$$\eta_1(\tau_i) = \sum_{\forall \tau_j \in (\gamma_i \cap \zeta_i^*)} \sum_{\forall \theta \in \theta_i} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \sum_{\forall s_j^{\bar{k}}(\theta)} len\left(s_j^{\bar{k}}(\theta)\right) \right)$$

$$\eta_2(\tau_i) = \sum_{\forall \tau_j \in (\gamma_i \cap \bar{\zeta_i})} \sum_{\forall \theta \in \theta_i} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \sum_{\forall s_j^{\bar{k}}(\theta)} len\left(s_j^{\bar{k}}(\theta)\right) \right)$$

$$\eta_3(\tau_i) = \sum_{\forall \tau_j \in (\gamma_i \cap \zeta_i^*)} \sum_{\forall \theta \in \theta_i} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \times \sum_{\forall s_j^{k^-}(\theta)} len\left(s_j^{k^-}(\theta) + s_{max}^j(\theta) \right) \right)$$

By substitution of $g(\tau_i)$, $\eta_1(\tau_i)$, $\eta_2(\tau_i)$, and $\eta_3(\tau_i)$ in (69):

$$\sum_{\forall \tau_i} \frac{\eta_1(\tau_i) + \eta_2(\tau_i)}{T_i} \leq \sum_{\forall \tau_i} \frac{\eta_3(\tau_i) + g(\tau_i)}{T_i} \tag{75}$$

When tasks with deadlines equal to periods are scheduled with G-RMA, $T_j > T_i$ if $p_j < p_i$. So, for each $\tau_j \in \bar{\zeta_i}$, $\left\lceil \frac{T_i}{T_j} \right\rceil = 1$. Then:

$$\eta_2(\tau_i) = 2 \sum_{\forall \tau_j \in (\gamma_i \cap \bar{\zeta_i})} \sum_{\forall \theta \in \theta_i} \sum_{\forall s_j^{k^-}(\theta)} len\left(s_j^{k^-}(\theta) \right) \tag{76}$$

Let:

$$\eta_4(\tau_i) = \sum_{\forall \tau_j \in (\gamma_i \cap \zeta_i^*)} \sum_{\forall \theta \in \theta_i} \left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \sum_{\forall s_j^{k^-}(\theta)} len\left(s_{max}^j(\theta) \right)$$

By substitution of (76) and subtraction of $\sum_{\forall \tau_i} \frac{\eta_1(\tau_i)}{T_i}$ from both sides of (75), we get:

$$2 \sum_{\forall \tau_i} \frac{\eta_2(\tau_i)}{T_i} \leq \sum_{\forall \tau_i} \frac{\eta_4(\tau_i) + g(\tau_i)}{T_i} \tag{77}$$

Assume that $g(\tau_i)_{\forall \tau_i} \to 0$. From (77), we note that when higher priority jobs increasingly conflict with lower priority jobs, (77) tends to hold. (77) also tends to hold if $len(s_{max}^{j^-}(\theta))$ in the right hand side of (77) is larger than $len(s_j^{k^-}(\theta))$ in the left hand side of (77), which means atomic section length increases as priority increases. Claim follows.

13.3 PNF versus G-EDF/LCM

Claim 32 *In the absence of transitive retry, PNF/EDF's schedulability is equal or better than G-EDF/LCM's if the conflicting atomic section lengths are approximately equal and all α terms approach 1.*

Proof 32 Assume that $\eta_1(\tau_i)$ and $\eta_3(\tau_i)$ are the same as that defined in the proof of Claim 30. Let:

$$g(\tau_i) = \left(\sum_{\forall \tau_j \in \gamma_i^*} \sum_{\theta \in \theta_i^*} \left(\left\lceil \frac{T_i}{T_j} \right\rceil \sum_{\forall s_j^{k^-}(\theta)} len\Big(s_j^{k^-}(\theta) + \alpha_{max}^{ji} s_{max}(\theta) \Big) \right) \right) + RC_{re}(T_i)$$

$$\eta_2(\tau_i) = \sum_{\forall \tau_j \in \gamma_i} \sum_{\forall \theta \in \theta_i} \left(\left\lceil \frac{T_i}{T_j} \right\rceil \sum_{\forall s_j^{k^-}(\theta)} len\left(\alpha_{max}^{jl} s_{max}^j(\theta) \right) \right)$$

where α^{jl}_{max} is defined in (43). Following the same steps in the proof of Claim 30, we get:

$$\sum_{\forall \tau_i} \frac{\eta_1(\tau_i)}{T_i} \leq \sum_{\forall \tau_i} \frac{\eta_2(\tau_i) + g(\tau_i)}{T_i} \tag{78}$$

Assume that $g(\tau_i)_{\forall \tau_i} \rightarrow 0$. Thus, we ignore the effect of transitive retry and retry cost due to the release of higher priority jobs. Let $len(\bar{s^k_j}(\theta)) = s^j_{max}(\theta) = s$, and $\alpha^{jl}_{max} = \alpha^{iy}_{max} = 1$ in (78). Then, PNF/EDF's schedulability equals LCM/EDF's schedulability if $\left\lceil \frac{T_i}{T_j} \right\rceil = 1, \forall \tau_i, \tau_j$ (which means equal periods for all tasks). If $\left\lceil \frac{T_i}{T_j} \right\rceil > 1, \forall \tau_i, \tau_j$, PNF/EDF's schedulability is better than LCM/EDF's. PNF/EDF's schedulability becomes more better than LCM/EDF's schedulability if $g(\tau_i)$ is not zero. Claim follows.

13.4 PNF versus G-RMA/LCM

Claim 33 *In the absence of transitive retry, PNF's schedulability is equal or better than G-RMA/LCM's if: 1) lower priority tasks suffer increasing number of conflicts from higher priority tasks, 2) the lengths of the atomic sections increase as task priorities increase, and 3) α terms increase.*

Proof 33 Assume that $g(\tau_i)$, $\eta_1(\tau_i)$, and $\eta_2(\tau_i)$ are the same as in the proof of Claim 31. Let:

$$\begin{aligned}
\eta_3(\tau_i) \quad = \quad & \sum_{\forall \tau_j \in (\gamma_i \cap \zeta^*_i)} \sum_{\forall \theta \in \theta_i} \Bigg(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \times \\
& \sum_{\forall \bar{s^k_j}(\theta)} len\left(\bar{s^k_j}(\theta) + \alpha^{jl}_{max} s^j_{max}(\theta) \right) \Bigg)
\end{aligned}$$

$$\begin{aligned}
\eta_4(\tau_i) \quad = \quad & \sum_{\forall \tau_j \in (\gamma_i \cap \zeta^*_i)} \sum_{\forall \theta \in \theta_i} \Bigg(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \\
& \times \quad \sum_{\forall \bar{s^k_j}(\theta)} len\left(\alpha^{jl}_{max} s^j_{max}(\theta) \right) \Bigg)
\end{aligned}$$

Following the steps of Claim 31's proof, $\therefore$(69) becomes:

$$2 \sum_{\forall \tau_i} \frac{\eta_2(\tau_i)}{T_i} \leq \sum_{\forall \tau_i} \frac{\eta_4(\tau_i) + g(\tau_i)}{T_i} \tag{79}$$

Assume that the effect of transitive retry and retry cost due to the release of higher priority jobs is negligible ($g(\tau_i) \rightarrow 0$). (79) holds if: 1) the contention from higher priority jobs to lower priority jobs increases because of the $\left\lceil \frac{T_i}{T_j} \right\rceil + 1$ term in the right hand side of (79); 2) α terms approach 1; and 3) the lengths of the atomic sections increase as priority increases. This makes $len(s^j_{max}(\theta))$ in (79)'s right side to be greater than $len(\bar{s^k_j}(\theta))$ in (79)'s left side. Claim follows.

13.5 PNF versus Lock-free Synchronization

Lock-free synchronization [13, 17] accesses only one object. Thus, the number of accessed objects per transaction in PNF is limited to one. This allows us to compare the schedulability of PNF with the lock-free algorithm.

$RC_B(T_i)$ in (69) is replaced with:

$$\sum_{\forall \tau_j \in \gamma_i} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \beta_{i,j} r_{max} \right) + RC_{re}(T_i) \tag{80}$$

where $\beta_{i,j}$ is the number of retry loops of τ_j that access the same object as accessed by some retry loop of τ_i [13]. r_{max} is the maximum execution cost of a single iteration of any retry loop of any task [13]. $RC_{re}(T_i)$ is defined in Claim 29. Lock-free synchronization does not depend on priorities of tasks. Thus, (80) applies for both G-EDF and G-RMA systems.

Claim 34 *Let r_{max} be the maximum execution cost of a single iteration of any retry loop of any task [13]. Let s_{max} be the maximum transaction length in all tasks. Assume that each transaction under PNF accesses only one object for once. The schedulability of PNF with either G-EDF or G-RMA scheduler is better or equal to the schedulability of lock-free synchronization if $s_{max}/r_{max} \leq 1$.*

Proof 34 The assumption in Claim 34 is made to enable a comparison between PNF and lock-free. Let $RC_A(T_i)$ in (69) be replaced with (64) and $RC_B(T_i)$ be replaced with (80). To simplify comparison, (64) is upper bounded by:

$$RC(T_i) = \sum_{\tau_j \in \gamma_i} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \beta^*_{i,j} s_{max} \right)$$

where $\beta^*_{i,j}$ is the number of times transactions in τ_j accesses shared objects with τ_i. Thus, $\beta^*_{i,j} = \beta_{i,j}$, and (69) will be:

$$\sum_{\forall \tau_i} \frac{\sum_{\tau_j \in \gamma_i} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \beta_{i,j} s_{max} \right)}{T_i} \leq \sum_{\forall \tau_i} \frac{\sum_{\forall \tau_j \in \gamma_i} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \beta_{i,j} r_{max} + RC_{re}(\tau_i) \right)}{T_i} \tag{81}$$

From (81), we note that if $s_{max} \leq r_{max}$, then (81) holds.

14 First Bounded, Last Timestamp CM (FBLT)

With PNF, all objects accessed by each transaction must be known a-priori. Therefore, this is not suitable with dynamic STM implementations [25]. Additionally, PNF is implemented in [15] as a centralized CM that uses locks. This increases overhead.

14.1 Case for FBLT

It is desirable to have a CM with the following goals:

1. reduce the retry cost of each transaction s_i^k due to another transaction s_j^l, just as LCM does compared to ECM and RCM.
2. avoid or bound the effect of transitive retry, similar to PNF, without prior knowledge of accessed objects by each transaction, enabling dynamic STM.
3. decentralized design and avoid the use of locks, thereby reducing overhead.

We propose the *First Bounded, Last Timestamp contention manager* (or (FBLT). FBLT achieves these goals by bounding the number of times each transaction s_i^k is aborted due to other transactions to at most δ_i^k. δ_i^k includes the number of aborts due to direct conflict with other transactions, as well as transitive retry (goal 2). If a transaction s_i^k reaches its δ_i^k, it is added to an $m_$set in FIFO order. In the $m_$set, s_i^k executes non-preemptively. If transactions in the $m_$set conflict together, they use their FIFO order in the $m_$set to resolve the conflict. s_i^k can still abort after it becomes a non-preemptive transaction due to other non-preemptive transactions. The number of aborts for any non-preemptive transaction is bounded by $m - 1$, where m is the number of processors, as will be shown in Section 16.

Thus, the key idea behind FBLT is to use a suitable δ_i^k for each s_i^k before it becomes a non-preemptive transaction. The choice of δ_i^k should make the total retry cost (and thus, the schedulability) of any job τ_i^x under FBLT comparable to the retry cost under ECM, RCM, LCM, and PNF. (In Section 17, we show the suitable δ_i^k for each s_i^k to have equal or better schedulability than other CMs.) Preemptive transactions resolve their conflicts using LCM. Thus, FBLT defaults to LCM if abort bounds have not been violated (goal 1). Each non-preemptive transaction s_i^k uses the time it joined the $m_$set to resolve conflicts with other non-preemptive transactions. Therefore, FBLT does not have to use locks and is decentralized (goal 3).

15 The FBLT Contention Manager

Algorithm 3 illustrates FBLT. Each transaction s_i^k can be aborted during T_i for at most δ_i^k times. η_i^k records the number of times s_i^k has already been aborted up to now. If s_i^k and s_j^l have not joined the $m_$set yet, then they are preemptive transactions. Preemptive transactions resolve conflicts using Algorithm 1 (step 2). Thus, FBLT defaults to LCM when no transaction reaches its δ. If only one of the transactions is in the $m_$set, then the non-preemptive transaction (the one in $m_$set) aborts the other one (steps 15 to 26). η_i^k is incremented each time s_i^k is aborted as long as $\eta_i^k < \delta_i^k$ (steps 5 and 18). Otherwise, s_i^k is added to the $m\_$ set and its priority is increased to m_prio (steps 7 to 9 and 20 to 22). When the priority of s_i^k is increased to m_prio, s_i^k becomes a non-preemptive transaction. Non-preemptive transactions cannot be aborted by other preemptive transactions, nor by any other real-time job. The $m_$set can hold at most m concurrent transactions because there are m processors in the system. $r(s_i^k)$ records the time s_i^k joined the $m_$set (steps 8 and 21). When non-preemptive transactions conflict together (step 27), the transaction with the smaller $r()$ commits first (steps 29 and 31). Thus, non-

Algorithm 3: The FBLT Algorithm

Data: s_i^k: interfered transaction;
s_j^l: interfering transactions;
δ_i^k: the maximum number of times s_i^k can be aborted during T_i;
η_i^k: number of times s_i^k has already been aborted up to now;
m_set: contains at most m non-preemptive transactions. m is number of processors;
m_prio: priority of any transaction in m_set. m_prio is higher than any priority of any real-time task;
$r(s_i^k)$: time point at which s_i^k joined m_set;
Result: atomic sections that will abort

```
1  if s_i^k, s_j^l ∉ m_set then
2  |  Apply Algorithm 1 (default to LCM);
3  |  if s_i^k is aborted then
4  |  |  if η_i^k < δ_i^k then
5  |  |  |  Increment η_i^k by 1;
6  |  |  else
7  |  |  |  Add s_i^k to m_set;
8  |  |  |  Record r(s_i^k);
9  |  |  |  Increase priority of s_i^k to m_prio;
10 |  |  end
11 |  else
12 |  |  Swap s_i^k and s_j^l;
13 |  |  Go to Step 3;
14 |  end
15 else if s_j^l ∈ m_set, s_i^k ∉ m_set then
16 |  Abort s_i^k;
17 |  if η_i^k < δ_i^k then
18 |  |  Increment η_i^k by 1;
19 |  else
20 |  |  Add s_i^k to m_set;
21 |  |  Record r(s_i^k);
22 |  |  Increase priority of s_i^k to m_prio;
23 |  end
24 else if s_i^k ∈ m_set, s_j^l ∉ m_set then
25 |  Swap s_i^k and s_j^l;
26 |  Go to Step 15;
27 else
28 |  if r(s_i^k) < r(s_j^l) then
29 |  |  Abort s_j^l;
30 |  else
31 |  |  Abort s_i^k;
32 |  end
33 end
```

preemptive transactions are executed in FIFO order of the m_set.

15.1 Illustrative Example

We now illustrate FBLT's behavior with the following example:

1. Transaction $s_i^k(\theta_1, \theta_2)$ is released while $m\_set = \emptyset$. $\eta_i^k = 0$ and $\delta_i^k = 3$.
2. Transaction $s_a^b(\theta_2)$ is released while $s_i^k(\theta_1, \theta_2)$ is running. $p_a^b > p_i^k$ and $\eta_i^k < \delta_i^k$. Applying LCM, $s_i^k(\theta_1, \theta_2)$ is aborted in favor of s_a^b and η_i^k is incremented to 1.
3. $s_a^b(\theta_2)$ commits. $s_i^k(\theta_1, \theta_2)$ runs again. Transaction $s_c^d(\theta_2)$ is released while

$s_i^k(\theta_1, \theta_2)$ is running. $p_c^d > p_i^k$. Applying LCM, $s_i^k(\theta_1, \theta_2)$ is aborted again in favor of $s_c^d(\theta_2)$. η_i^k is incremented to 2.

4. $s_c^d(\theta_2)$ commits. $s_e^f(\theta_2, \theta_3)$ is released. $p_e^f > p_i^k$ and $\eta_e^f = 2$. $s_i^k(\theta_1, \theta_2)$ is aborted in favor of $s_e^f(\theta_2, \theta_3)$ and η_i^k is incremented to 3.
5. $s_j^l(\theta_3)$ is released. $p_j^l > p_e^f$. $s_e^f(\theta_2, \theta_3)$ is aborted in favor of $s_j^l(\theta_3)$ and η_e^f is incremented to 1.
6. $s_i^k(\theta_1, \theta_2)$ and $s_e^f(\theta_2, \theta_3)$ are compared again. $\because \eta_i^k = \delta_i^k$, $\therefore s_i^k(\theta_1, \theta_2)$ is added to m_set. $m\_set = \{s_i^k(\theta_1, \theta_2)\}$. $s_i^k(\theta_1, \theta_2)$ becomes a non-preemptive transaction. As $s_e^f(\theta_2, \theta_3)$ is a preemptive transaction, $\therefore s_e^f(\theta_2, \theta_3)$ is aborted in favor of $s_i^k(\theta_1, \theta_2)$, despite p_e^f being greater than the original priority of $s_i^k(\theta_1, \theta_2)$. η_e^f is incremented to 2.
7. $s_j^l(\theta_3)$ commits but $s_g^h(\theta_3)$ is released. $p_g^h > p_e^f$ but $\eta_e^f = \delta_e^f$. So, $s_e^f(\theta_2, \theta_3)$ becomes a non-preemptive transaction. $m\_set = \{s_i^k(\theta_1, \theta_2), s_g^h(\theta_2, \theta_3)\}$.
8. $s_i^k(\theta_1, \theta_2)$ and $s_g^h(\theta_2, \theta_3)$ are now non-preemptive transactions. $s_i^k(\theta_1, \theta_2)$ and $s_g^h(\theta_2, \theta_3)$ still conflict together. So, they are executed according to their addition order to the m_set. So, $s_i^k(\theta_1, \theta_2)$ commits first, followed $s_g^h(\theta_2, \theta_3)$.
9. $s_g^h(\theta_3)$ will continue to abort and retry in favor of $s_e^f(\theta_2, \theta_3)$ until $s_e^f(\theta_2, \theta_3)$ commits or $\eta_g^h = \delta_g^h$. Even if $s_g^h(\theta_3)$ joined the m_set, $s_g^h(\theta_3)$ will still abort and retry in favor of $s_e^f(\theta_2, \theta_3)$, because $s_e^f(\theta_2, \theta_3)$ joined the m_set earlier than $s_g^h(\theta_3)$.

It is seen from steps 2 to 6 that $s_i^k(\theta_1, \theta_2)$ can be aborted due to direct conflict with other transactions, or due to transitive retry. Irrespective of the reason for the conflict, once a transaction has reached its maximum allowed δ, the transaction becomes a non-preemptive one (steps 6 and 7). Non-preemptive transactions have higher priority than other preemptive transactions (steps 6 and 7). Non-preemptive transactions execute in their arrival order to the m_set.

16 Retry Cost and Response Time Bounds

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

Claim 35 *The total retry cost for any job τ_i^x under FBLT due to 1) conflicts between its transactions and transactions of other jobs during an interval $L \leq T_i$ and 2) release of higher priority jobs is upper bounded by:*

$$RC_{to}(L) \leq \sum_{\forall s_i^k \in s_i} \left(\delta_i^k len(s_i^k) + \sum_{\forall s_{iz}^k \in \chi_i^k} len(s_{iz}^k) \right) + RC_{re}(L) \qquad (82)$$

where χ_i^k is the set of at most $m-1$ maximum length transactions conflicting directly or indirectly (through transitive retry) with s_i^k. Each transaction $s_{iz}^k \in \chi_i^k$ belongs to a distinct task τ_j. $RC_{re}(L)$ is the retry cost resulting from the release of higher priority jobs which preempt τ_i^x. $RC_{re}(L)$ is calculated by (6.8) in [15] for G-EDF, and (6.10) in [15] for G-RMA schedulers.

Proof 35 By the definition of FBLT, $s_i^k \in \tau_i^x$ can be aborted a maximum of δ_i^k times before s_i^k joins the m_set. Before joining the m_set, s_i^k can be aborted

due to higher priority transactions, or transactions in the m_set. The original priority of transactions in the m_set can be higher or lower than p_i^x. Thus, the maximum time s_i^k is aborted before joining the m_set occurs if s_i^k is aborted for δ_i^k times.

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

Claim 36 *Under FBLT, the blocking time of a job τ_i^x due to lower priority jobs is upper bounded by:*

$$D(\tau_i^x) = min\left(max_1^m(s_{j_{max},\forall\tau_j^l,\, p_j^l<p_i^x})\right) \tag{83}$$

where $s_{j_{max}}$ is the maximum length transaction in any job τ_j^l with original priority lower than p_i^x. The right hand side of (83) is the minimum of the m maximum transactional lengths in all jobs with lower priority than τ_i^x.

Proof 36 τ_i^x is blocked when it is initially released and all processors are busy with lower priority jobs with non-preemptive transactions. Although τ_i^x can be preempted by higher priority jobs, τ_i^x cannot be blocked after it is released. If τ_i^x is preempted by a higher priority job τ_j^y, then, when τ_j^y finishes execution, the underlying scheduler will not choose a lower priority job than τ_i^x before τ_i^x. So, after τ_i^x is released, there is no chance for any transaction s_u^v belonging to a lower priority job than τ_i^x to run before τ_i^x. Thus, s_u^v cannot join the m_set before τ_i^x finishes. Consequently, the worst case blocking time for τ_i^x occurs when the maximum length m transactions in lower priority jobs than τ_i^x are executing non-preemptively. After the minimum length transaction in the m_set finishes, the underlying scheduler will choose τ_i^x or a higher priority job to run. Claim follows.

Claim 37 *The response time of any job τ_i^x during an interval $L \leq T_i$ under FBLT is upper bounded by:*

$$R_i^{up} = c_i + RC_{to}(L) + D(\tau_i^x) + \left\lfloor \frac{1}{m} \sum_{\forall j \neq i} W_{ij}(R_i^{up}) \right\rfloor \tag{84}$$

where $RC_{to}(L)$ is calculated by (82), $D(\tau_i^x)$ is calculated by (83), and $W_{ij}(R_i^{up})$ is calculated by (11) in [17] for G-EDF, and (17) in [17] for G-RMA schedulers. (11) and (17) in [17] inflates c_j of any job $\tau_j^y \neq \tau_i^x$, $p_j^y > p_i^x$ by the retry cost of transactions in τ_j^y.

Proof 37 The response time of a job is calculated directly from FBLT's behavior. The response time of any job τ_i^x is the sum of its worst case execution time c_i, plus the retry cost of transactions in τ_i^x ($RC_{to}(L)$), plus the blocking time of τ_i^x ($D(\tau_i^x)$), and the workload interference of higher priority jobs. The workload interference of higher priority jobs scheduled by G-EDF is calculated by (11) in [17], and by (17) in [17] for G-RMA. Claim follows.

17 Schedulability Comparison

We now (formally) compare the schedulability of G-EDF (G-RMA) with FBLT against ECM, RCM, LCM, PNF, and lock-free synchronization [13,15–17]. Such a comparison will reveal when FBLT outperforms the others. Toward this, we compare the total utilization under G-EDF (G-RMA)/FBLT with that under the other synchronization methods. In this comparison, we use the inflated execution time of the task, which is the sum of the worst-case execution time of the task and its retry cost, in the utilization calculation of the task.

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

Let $RC_A(T_i)$ and $RC_B(T_i)$ denote the retry cost of a job τ_i^x during T_i using the synchronization method A and synchronization method B, respectively. Now, schedulability of A is comparable to B if:

$$\begin{aligned} \sum_{\forall \tau_i} \frac{c_i + RC_A(T_i)}{T_i} &\leq \sum_{\forall \tau_i} \frac{c_i + RC_B(T_i)}{T_i} \\ \sum_{\forall \tau_i} \frac{RC_A(T_i)}{T_i} &\leq \sum_{\forall \tau_i} \frac{RC_B(T_i)}{T_i} \end{aligned} \tag{85}$$

17.1 FBLT vs. ECM

Claim 38 *The schedulability of FBLT is equal to or better than ECM's when the maximum abort number of any preemptive transaction s_i^k is less than or equal to the number of transactions directly conflicting with s_i^k in all other jobs with higher priority than τ_i's current job.*

Proof 38 By substituting $RC_A(T_i)$ and $RC_B(T_i)$ in (85) with (82) and (6.7) in [15], respectively, we get:

$$\begin{aligned} & \sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k \in s_i} \left(\delta_i^k len(s_i^k) + \sum_{s_{iz}^k \in \chi_i^k} len(s_{iz}^k) \right) + RC_{re}(T_i)}{T_i} \\ \leq & \sum_{\forall \tau_i} \frac{\left(\sum_{\forall \tau_j \in \gamma_i^{ex}} \sum_{\theta \in \theta_i^{ex}} \left(\left\lceil \frac{T_i}{T_j} \right\rceil \sum_{\forall \bar{s_j^h}(\theta)} len\left(\bar{s_j^h}(\theta) + s_{max}^j(\theta) \right) \right) \right) + RC_{re}(T_i)}{T_i} \end{aligned} \tag{86}$$

Let $\theta_i^{ex} = \theta_i + \theta_i^*$, where θ_i^* is the set of objects not accessed directly by τ_i but can cause transactions in τ_i to retry due to transitive retry. Let $\gamma_i^{ex} = \gamma_i + \gamma_i^*$, where γ_i^* is the set of tasks that access objects in θ_i^*. $\bar{s_j^h}(\theta)$ can access multiple objects, so $s_{max}^j(\theta)$ is the maximum length transaction conflicting with $\bar{s_j^h}(\theta)$.

$\bar{s_j^h}(\theta)$ is included only once for all $\theta \in \Theta_j^h$. Each $\theta \in \theta_i^{ex}$ has its own $s_{max}^j(\theta)$. But s_i^h can access multiple objects, denoted as Θ_j^h. So, $s_{max}^j(\theta)$ is replaced by $s_{max}^j(\Theta_j^h)$, where $s_{max}^j(\Theta_j^h) = max\{s_{max}^j(\theta), \forall\theta \in \Theta_j^h\}$. $s_{max}^j(\Theta_j^h)$ is included once for each $\theta \in \theta_i$.

Each job τ_i^x has the same interference pattern from higher priority jobs, τ_j^h, under FBLT and ECM. Hence, $RC_{re}(T_i)$ for τ_i^x is the same under FBLT and ECM. Consequently, (86) becomes:

$$\begin{aligned} & \sum_{\forall\tau_i} \frac{\sum_{\forall s_i^k \in s_i} \left(\delta_i^k len(s_i^k) + \sum_{s_{iz}^k \in \chi_i^k} len(s_{iz}^k) \right)}{T_i} \\ \leq \quad & \sum_{\forall\tau_i} \frac{\left(\sum_{\forall\tau_j \in \gamma_i^{ex}} \sum_{\forall \bar{s_j^h}(\Theta_j^h),\, \Theta_j^h \in \theta_i^{ex}} \left(\left\lceil \frac{T_i}{T_j} \right\rceil len\left(\bar{s_j^h}(\Theta_j^h) + s_{max}^j(\Theta_j^h) \right) \right) \right)}{T_i} \end{aligned} \tag{87}$$

Although different s_i^ks can have common conflicting transactions $\bar{s_j^h}$, no more than one s_i^k can be preceded by the same $\bar{s_j^h}$ in the m_set. This happens because transactions in the m_set are non-preemptive. The original priority of transactions preceding s_i^k in the m_set can be lower or higher than the original priority of s_i^k. Since under G-EDF, τ_j can have at least one job of higher priority than τ_i^x, $\left\lceil \frac{T_i}{T_j} \right\rceil \geq 1$. Thus, each one of the s_{iz}^k term in the left hand side of (87) is included in one of the $\bar{s_j^h}(\theta)$ term in the right hand side of (87). Now, (87) holds if:

$$\begin{aligned} & \sum_{\forall\tau_i} \frac{\sum_{\forall s_i^k \in s_i} \delta_i^k len(s_i^k)}{T_i} \\ \leq \quad & \sum_{\forall\tau_i} \frac{\sum_{\forall\tau_j \in \gamma_i^{ex}} \sum_{\forall \bar{s_j^h}(\Theta_j^h),\, \Theta_j^h \in \theta_i^{ex}} \left(\left\lceil \frac{T_i}{T_j} \right\rceil len\left(s_{max}^j(\Theta_j^h) \right) \right)}{T_i} \end{aligned} \tag{88}$$

Since FBLT is required to bound the effect of transitive retry, only θ_i (not the whole θ_i^{ex}) will be considered in (88). Thus, ECM acts as if there were no transitive retry. Consequently, (88) holds if:

$$\begin{aligned} & \sum_{\forall\tau_i} \frac{\sum_{\forall s_i^k \in s_i} \delta_i^k len(s_i^k)}{T_i} \\ \leq \quad & \sum_{\forall\tau_i} \frac{\sum_{\forall\tau_j \in \gamma_i} \sum_{\forall \bar{s_j^h}(\Theta),\, \Theta \in (\theta_i \cap \Theta_j^h)} \left(\left\lceil \frac{T_i}{T_j} \right\rceil len\left(s_{max}^j(\Theta) \right) \right)}{T_i} \end{aligned} \tag{89}$$

where $s_{max}^j(\Theta) \leq s_{max}^j(\Theta_j^h)$.

For each $s_i^k \in s_i$, there are a set of zero or more $\bar{s_j^h}(\Theta) \in \tau_j$, $\forall\tau_j \neq \tau_i$ that are conflicting with s_i^k. Assuming this set of transactions conflicting with s_i^k is denoted as

$$\nu_i^k = \left\{ \bar{s_j^h}(\Theta) \in \tau_j : \left(\Theta \in (\theta_i \cap \Theta_j^h) \right) \wedge (\forall\tau_j \neq \tau_i) \wedge \left(\bar{s_j^h}(\Theta) \notin \nu_i^l,\, l \neq k \right) \right\}$$

The last condition $\bar{s_j^h}(\Theta) \notin \nu_i^l$, $l \neq k$ in the definition of ν_i^k ensures that common transactions $\bar{s_j^h}$ that can conflict with more than one transaction $s_i^k \in \tau_i$ are split among different ν_i^k, $k = 1, .., |s_i|$. This condition is necessary, because in ECM, no two or more transactions of τ_i^x can be aborted by the same transaction

of τ_j^h, where $p_j^h > p_i^x$. By substitution of ν_i^k in (89), we get:

$$\begin{aligned} & \sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k \in s_i} \delta_i^k len(s_i^k)}{T_i} \\ \leq \quad & \sum_{\forall \tau_i} \frac{\left(\sum_{\forall k=1}^{|s_i|} \sum_{s_j^{\bar{h}}(\Theta) \in \nu_i^k} \left(\left\lceil \frac{T_i}{T_j} \right\rceil len\left(s_{max}^j(\Theta)\right) \right) \right)}{T_i} \end{aligned} \tag{90}$$

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

$$\delta_i^k \leq \frac{\sum_{s_j^{\bar{h}}(\Theta) \in \nu_i^k} \left(\left\lceil \frac{T_i}{T_j} \right\rceil len\left(s_{max}^j(\Theta)\right) \right)}{len(s_i^k)} \tag{91}$$

Since $len\left(s_{max}^j(\Theta)\right) \geq len(s_i^k)$, (91) holds if $\delta_i^k \leq \sum_{s_j^{\bar{h}}(\Theta) \in \nu_i^k} \left\lceil \frac{T_i}{T_j} \right\rceil \cdot \sum_{s_j^{\bar{h}}(\Theta) \in \nu_i^k} \left\lceil \frac{T_i}{T_j} \right\rceil$ is the maximum number of transactions directly conflicting with s_i^k in all jobs with higher priority than p_i^x. Claim follows.

17.2 FBLT vs. RCM

Claim 39 *The schedulability of FBLT is equal to or better than RCM's if*

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

$\sum_{s_j^{\bar{h}}(\Theta) \in \nu_i^{\bar{k}}} \left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right)$ *is number of transactions directly conflicting with* s_i^k *in all jobs with higher priority than* τ_i. $\sum_{u=1,\, s_{u_{max}} \in \epsilon}^{min(n,m)-1} s_{u_{max}}$ *is the sum of the maximum* $m-1$ *transactional lengths in all tasks*

Proof 39 By substituting $RC_A(T_i)$ and $RC_B(T_i)$ in (85) with (82) and (6.9) in [15], respectively, we get:

$$\begin{aligned} & \sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k \in s_i} \left(\delta_i^k len(s_i^k) + \sum_{s_{iz}^k \in \chi_i^k} len(s_{iz}^k) \right) + RC_{re}(T_i)}{T_i} \\ \leq \quad & \sum_{\forall \tau_i} \frac{\left(\sum_{\forall \tau_j^* \in \gamma_i^{ex}} \sum_{\forall \theta \in \theta_i^{ex}} \left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \sum_{\forall s_j^{\bar{h}}(\theta)} len\left(s_j^{\bar{h}}(\theta) + s_{max}^j(\theta)\right) \right) + RC_{re}(T_i)}{T_i} \end{aligned} \tag{92}$$

where $\tau_j^* = \{\tau_j : (\tau_j \neq \tau_i) \wedge (p_j > p_i)\}$.

Let $\theta_i^{ex} = \theta_i + \theta_i^*$, where θ_i^* is the set of objects not directly accessed by any job of τ_i, but can cause transactions in τ_i to retry due to transitive retry. Let $\gamma_i^{ex} = \gamma_i + \gamma_i^*$, where γ_i^* is the set of tasks that access objects in θ_i^*. $s_j^{\bar{h}}(\theta)$ can access multiple objects, so $s_{max}^j(\theta)$ is the maximum length transaction conflicting with $s_j^{\bar{h}}(\theta)$. $s_j^{\bar{h}}(\theta)$ is included only once for all $\theta \in \Theta_j^h$. Each $\theta \in \theta_i^{ex}$ has its own $s_{max}^j(\theta)$. But s_i^h can access multiple objects, denoted as Θ_j^h. So, $s_{max}^j(\theta)$ is replaced by $s_{max}^j(\Theta_j^h)$, where $s_{max}^j(\Theta_j^h) = max\{s_{max}^j(\theta), \forall \theta \in \Theta_j^h\}$. $s_{max}^j(\Theta_j^h)$ is included once for each $\theta \in \theta_i$.

Each τ_i^x has the same interference pattern from higher priority jobs, τ_j^h, under FBLT and RCM. Hence, $RC_{re}(T_i)$ for τ_i^x is the same under FBLT and

RCM. Consequently, (92) becomes:

$$\begin{aligned} & \sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k \in s_i} \left(\delta_i^k len(s_i^k) + \sum_{s_{iz}^k \in \chi_i^k} len(s_{iz}^k) \right)}{T_i} \\ \leq \quad & \sum_{\forall \tau_i} \frac{\sum_{\forall \tau_j^* \in \gamma_i^{ex}} \sum_{\forall \bar{s_j^h}(\Theta_j^h), \Theta_j^h \in \theta_i^{ex}} \left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) len\left(\bar{s_j^h}(\Theta_j^h) + s_{max}^j(\Theta_j^h) \right)}{T_i} \end{aligned} \tag{93}$$

Although different s_i^ks can have common conflicting transactions $\bar{s_j^h}$, no more than one s_i^k can be preceded by the same $\bar{s_j^h}$ in the m_set. This happens because transactions in the m_set are non-preemptive. The original priority of transactions preceding s_i^k in the m_set can be of lower or higher priority than the original priority of s_i^k. Under G-RMA, $p_j > p_i$, which means that $T_j \leq T_i$. Therefore, $\left\lceil \frac{T_i}{T_j} \right\rceil \geq 1$. For each $s_i^k \in s_i$, there are a set of zero or more $\bar{s_j^h}(\Theta_j^h) \in \tau_j^*$ that are conflicting with s_i^k. Assuming this set of transactions conflicting with s_i^k is denoted as $\nu_i^k = \left\{ \bar{s_j^h}(\Theta_j^h) \in \tau_j^* : \left(\Theta_j^h \in \theta_i^{ex}\right) \wedge \left(\bar{s_j^h}(\Theta_j^h) \notin \nu_i^l,\, l \neq k \right) \right\}$.

The last condition $\bar{s_j^h}(\theta) \notin \nu_i^l,\, l \neq k$ in the definition of ν_i^k ensures that common transactions $\bar{s_j^h}$ that can conflict with more than one transaction $s_i^k \in \tau_i$ are split among different ν_i^k, $k = 1, .., |s_i|$. This condition is necessary, because in RCM, no two or more transactions of τ_i^x can be aborted by the same transaction of τ_j^h, where $p_j^h > p_i^x$. By substitution of ν_i^k in (93), we get:

$$\begin{aligned} & \sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k \in s_i} \left(\delta_i^k len(s_i^k) + \sum_{s_{iz}^k \in \chi_i^k} len(s_{iz}^k) \right)}{T_i} \\ \leq \quad & \sum_{\forall \tau_i} \frac{\left(\sum_{\forall k=1}^{|s_i|} \sum_{\bar{s_j^h}(\Theta_j^h) \in \nu_i^k} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) len\left(\bar{s_j^h}(\Theta_j^h) + s_{max}^j(\Theta_j^h) \right) \right) \right)}{T_i} \end{aligned} \tag{94}$$

$\bar{s_j^h}$ belongs to higher priority jobs than τ_i. s_{max}^j belongs to higher priority jobs than τ_i or τ_i itself. s_{max}^j has a lower priority than τ_j. Transactions in the m_set can belong to jobs with original priority higher or lower than τ_i. Thus, (94) holds if for each $s_i^k \in \tau_i$:

$$\delta_i^k len(s_i^k) \leq \left(\sum_{\bar{s_j^h}(\Theta_j^h) \in \nu_i^k} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) len\left(\bar{s_j^h}(\Theta_j^h) + s_{max}^j(\Theta_j^h) \right) \right) \right) - \sum_{s_{iz}^k \in \chi_i^k} len(s_{iz}^k) \tag{95}$$

Then,

$$\delta_i^k \leq \left(\sum_{\bar{s_j^h}(\Theta_j^h) \in \nu_i^k} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) len\left(\frac{\bar{s_j^h}(\Theta_j^h) + s_{max}^j(\Theta_j^h)}{s_i^k} \right) \right) \right) - \sum_{s_{iz}^k \in \chi_i^k} len\left(\frac{s_{iz}^k}{s_i^k} \right) \tag{96}$$

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

transactional lengths of all tasks. $|\chi_i^k| \leq m-1$ and $len(s^j_{max}(\Theta^h_j)) \geq len(s^k_i)$. So, (96) holds if:

$$\delta_i^k \leq \left(\sum_{s^{\bar{h}}_j(\Theta^h_j) \in \nu_i^k} \left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \right) - \sum_{u=1,\, s_{u_{max}} \in \epsilon}^{min(n,m)-1} s_{u_{max}} \tag{97}$$

To bound the effect of transitive retry, only objects that belong to θ_i (not whole θ_i^{ex}) will be considered. Thus, RCM acts as if there were no transitive retry. Thus, ν_i^k is modified to $\bar{\nu_i^k} = \left\{ s^{\bar{h}}_j(\Theta) \in \tau_j^* : \left(\Theta \in \Theta^h_j \cap \theta_i\right) \wedge \left(s^{\bar{h}}_j(\Theta) \notin \nu_i^l,\, l \neq k \right) \right\}$. Since $\bar{\nu_i^k} \subseteq \nu_i^k$, (97) still holds if ν_i^k is replaced with $\bar{\nu_i^k}$. Consequently, (97) holds if:

$$\delta_i^k \leq \left(\sum_{s^{\bar{h}}_j(\Theta) \in \bar{\nu_i^k}} \left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \right) - \sum_{u=1,\, s_{u_{max}} \in \epsilon}^{min(n,m)-1} s_{u_{max}} \tag{98}$$

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

17.3 FBLT vs. G-EDF/LCM

Claim 40 *The schedulability of FBLT is equal to or better than G-EDF/LCM's when*

$$\delta_i^k \leq \left(\sum_{s^{\bar{h}}_j(\Theta) \in \nu_i^k} \left(\left\lceil \frac{T_i}{T_j} \right\rceil \alpha^{j\bar{h}}_{max} \right) \right)$$

α^{jh}_{max} is the maximum α with which s_j^h can conflict with the maximum length transaction sharing objects with s_i^k and s_j^h

Proof 40 By substituting $RC_A(T_i)$ and $RC_B(T_i)$ in (85) with (82) and (6.7) in [15], respectively, we get:

$$\begin{aligned} & \sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k \in s_i} \left(\delta_i^k len(s_i^k) + \sum_{\forall s^k_{iz} \in \chi_i^k} len(s^k_{iz}) \right) + RC_{re}(T_i)}{T_i} \\ \leq & \sum_{\forall \tau_i} \frac{\left(\sum_{\forall \tau_j \in \gamma_i^{ex}} \sum_{\theta \in \theta_i^{ex}} \left(\left\lceil \frac{T_i}{T_j} \right\rceil \sum_{\forall s^{\bar{h}}_j(\theta)} len\left(s^{\bar{h}}_j(\theta) + \alpha^{j\bar{h}}_{max} s^j_{max}(\theta) \right) \right) \right)}{T_i} \\ + & \sum_{\forall \tau_i} \frac{\left(\sum_{\forall s_i^k} \left(1 - \alpha^{ik}_{max}\right) len\left(s^i_{max}\right) \right) + RC_{re}(T_i)}{T_i} \end{aligned} \tag{99}$$

Let $\theta_i^{ex} = \theta_i + \theta_i^*$, where θ_i^* is the set of objects not accessed directly by τ_i, but can cause transactions in τ_i to retry due to transitive retry. Let $\gamma_i^{ex} = \gamma_i + \gamma_i^*$, where γ_i^* is the set of tasks that access objects in θ_i^*. $s^{\bar{h}}_j(\theta)$ can access multiple objects, so $s^j_{max}(\theta)$ is the maximum length transaction conflicting with $s^{\bar{h}}_j(\theta)$. $s^{\bar{h}}_j(\theta)$ is included only once for all $\theta \in \Theta^h_j$. Each $\theta \in \theta_i^{ex}$ has its own $s^j_{max}(\theta)$. But s_i^h can access multiple objects, denoted as Θ^h_j. So, $s^j_{max}(\theta)$ is replaced by

$s^j_{max}(\Theta^h_j)$, where $s^j_{max}(\Theta^h_j) = max\{s^j_{max}(\theta), \forall\theta \in \Theta^h_j\}$. $s^j_{max}(\Theta^h_j)$ is included once for each $\theta \in \theta_i$.

Each τ^x_i has the same interference pattern from higher priority jobs, τ^h_j, under FBLT and G-EDF/LCM. Hence, $RC_{re}(T_i)$ for τ^x_i is the same under FBLT and G-EDF/LCM. Consequently, (99) holds if:

$$\begin{aligned} & \sum_{\forall\tau_i} \frac{\sum_{\forall s^k_i \in s_i}\Big(\delta^k_i len(s^k_i) + \sum_{\forall s^k_{iz} \in \chi^k_i} len(s^k_{iz})\Big)}{T_i} \\ \leq \quad & \sum_{\forall\tau_i} \frac{\Big(\sum_{\forall\tau_j \in \gamma^{ex}_i} \sum_{\forall s^h_i(\Theta^h_j),\, \Theta^h_j \in \theta^{ex}_i} \Big(\Big\lceil \frac{T_i}{T_j} \Big\rceil len\Big(s^{\bar{h}}_j(\Theta^h_j) + \alpha^{j\bar{h}}_{max} s^j_{max}(\Theta^h_j)\Big)\Big)\Big)}{T_i} \\ + \quad & \sum_{\forall\tau_i} \frac{\Big(\sum_{\forall s^k_i} \big(1 - \alpha^{ik}_{max}\big) len\big(s^i_{max}\big)\Big)}{T_i} \end{aligned} \tag{100}$$

Although different s^k_i can have common conflicting transactions $s^{\bar{h}}_j$, no more than one s^k_i can be preceded by the same $s^{\bar{h}}_j$ in the m_set. This happens because transactions in the m_set are non-preemptive. The original priority of transactions preceding s^k_i in the m_set can be of lower or higher priority than the original priority of s^k_i. Under G-EDF/LCM, $\tau_j \neq \tau_i$ can have at least one job of higher priority than the current job of τ_i. Hence, $\left\lceil \frac{T_i}{T_j} \right\rceil \geq 1$. Thus, each one of the s^k_{iz} terms in the left hand side of (100) is included in one of the $s^{\bar{h}}_j(\Theta^h_j)$ terms in the right hand side of (100). Now, (100) holds if:

$$\begin{aligned} & \sum_{\forall\tau_i} \frac{\sum_{\forall s^k_i \in s_i} \delta^k_i len(s^k_i)}{T_i} \\ \leq \quad & \sum_{\forall\tau_i} \frac{\Big(\sum_{\forall\tau_j \in \gamma^{ex}_i} \sum_{\forall s^{\bar{h}}_j(\Theta^h_j), \Theta^h_j \in \theta^{ex}_i} \Big(\Big\lceil \frac{T_i}{T_j} \Big\rceil len\Big(\alpha^{j\bar{h}}_{max} s^j_{max}(\Theta^h_j)\Big)\Big)\Big)}{T_i} \\ + \quad & \sum_{\forall\tau_i} \frac{\sum_{\forall s^k_i} \big(1 - \alpha^{ik}_{max}\big) len\big(s^i_{max}\big)}{T_i} \end{aligned} \tag{101}$$

To bound the effect of transitive retry, only θ_i (not the whole θ^{ex}_i) will be considered in (101). So, G-EDF/LCM acts as if there is no transitive retry. Consequently, (101) holds if:

$$\begin{aligned} & \sum_{\forall\tau_i} \frac{\sum_{\forall s^k_i \in s_i} \delta^k_i len(s^k_i)}{T_i} \\ \leq \quad & \sum_{\forall\tau_i} \frac{\Big(\sum_{\forall\tau_j \in \gamma_i} \sum_{\forall s^{\bar{h}}_j(\Theta), \Theta \in \Theta^h_j \cap \theta_i} \Big(\Big\lceil \frac{T_i}{T_j} \Big\rceil len\Big(\alpha^{j\bar{h}}_{max} s^j_{max}(\Theta)\Big)\Big)\Big)}{T_i} \\ + \quad & \sum_{\forall\tau_i} \frac{\sum_{\forall s^k_i} \big(1 - \alpha^{ik}_{max}\big) len\big(s^i_{max}\big)}{T_i} \end{aligned} \tag{102}$$

where $s^j_{max}(\Theta) \leq s^j_{max}(\Theta^h_j)$. For each $s^k_i \in s_i$, there are a set of zero or more $s^{\bar{h}}_j(\Theta^h_j) \in \tau_j$, $\forall\tau_j \neq \tau_i$ that are conflicting with s^k_i. Assuming this set of transactions conflicting with s^k_i is denoted as $\nu^k_i = \Big\{ s^{\bar{h}}_j(\Theta) \in \tau_j : \big(\Theta \in \theta_i \cap \Theta^h_j\big) \wedge (\forall\tau_j \neq \tau_i) \wedge \Big(s^{\bar{h}}_j(\theta) \notin \nu^l_i,\, l \neq k\Big)\Big\}$.

The last condition $s^{\bar{h}}_j(\theta) \notin \nu^l_i,\, l \neq k$ in the definition of ν^k_i ensures that common transactions $s^{\bar{h}}_j$ that can conflict with more than one transaction $s^k_i \in \tau_i$ are split among different ν^k_i, $k = 1, .., |s_i|$. This condition is necessary, because in G-EDF/LCM, no two or more transactions of τ^x_i can be aborted by the same

transaction of τ_j^h, where $p_j^h > p_i^x$. By substitution of ν_i^k in (101), we get:

$$\begin{aligned} & \sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k \in s_i} \delta_i^k len(s_i^k)}{T_i} \\ \leq & \sum_{\forall \tau_i} \frac{\sum_{k=1}^{|s_i|} \sum_{\forall \bar{s_j^h}(\Theta) \in \nu_i^k} \left(\left\lceil \frac{T_i}{T_j} \right\rceil len\left(\alpha_{max}^{j\bar{h}} s_{max}^j(\Theta) \right) \right)}{T_i} \\ + & \sum_{\forall \tau_i} \frac{\left(\sum_{\forall s_i^k} \left(1 - \alpha_{max}^{ik}\right) len\left(s_{max}^i\right) \right)}{T_i} \end{aligned} \tag{103}$$

$\bar{s_j^h}$ belongs to higher priority jobs than τ_i and s_{max}^j belongs to higher priority jobs than τ_i or τ_i itself. Transactions in the m_- set can belong to jobs with original priority higher or lower than τ_i. Thus, (103) holds if for each $s_i^k \in \tau_i$:

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

This leads to:

$$\delta_i^k \leq \left(\sum_{\forall \bar{s_j^h}(\Theta) \in \nu_i^k} \left(\left\lceil \frac{T_i}{T_j} \right\rceil \right) len\left(\frac{\alpha_{max}^{j\bar{h}} s_{max}^j(\Theta)}{s_i^k} \right) \right) + \left(1 - \alpha_{max}^{ik}\right) len\left(\frac{s_{max}^i}{s_i^k} \right) \tag{104}$$

Since $len\left(\frac{s_{max}^j(\Theta)}{s_i^k} \right) \geq 1$, (104) holds if:

$$\delta_i^k \leq \left(\sum_{\bar{s_j^h}(\Theta) \in \nu_i^k} \left(\left\lceil \frac{T_i}{T_j} \right\rceil \alpha_{max}^{j\bar{h}} \right) \right)$$

. Claim follows.

17.4 FBLT vs. G-RMA/LCM

Claim 41 *The schedulability of FBLT is equal to or better than G-RMA/LCM's when*

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

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

Proof 41 By substituing $RC_A(T_i)$ and $RC_B(T_i)$ in (85) with (82) and (6.9) in [15], respectively, we get:

$$\begin{aligned} & \sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k \in s_i} \left(\delta_i^k len(s_i^k) + \sum_{s_{iz}^k \in \chi_i^k} len(s_{iz}^k) \right) + RC_{re}(T_i)}{T_i} \\ \leq & \sum_{\forall \tau_i} \frac{\sum_{\forall \tau_j^* \in \gamma_i^{ex}} \sum_{\forall \theta \in \theta_i^{ex}} \left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \sum_{\forall s_j^{\bar{h}}(\theta)} len\left(s_j^{\bar{h}}(\theta) + \alpha_{max}^{j\bar{h}} s_{max}^j(\theta) \right)}{T_i} \\ + & \sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k} \left(1 - \alpha_{max}^{ik}\right) len(s_{max}^i) + RC_{re}(T_i)}{T_i} \end{aligned} \tag{105}$$

where $\tau_j^* = \{\tau_j : (\tau_j \neq \tau_i) \wedge (p_j > p_i)\}$. Let $\theta_i^{ex} = \theta_i + \theta_i^*$, where θ_i^* is the set of objects not accessed directly by τ_i but can cause transactions in τ_i to retry due to transitive retry. Let $\gamma_i^{ex} = \gamma_i + \gamma_i^*$, where γ_i^* is the set of tasks that access objects in θ_i^*. $s_j^{\bar{h}}(\theta)$ can access multiple objects, so $s_{max}^j(\theta)$ is the maximum length transaction conflicting with $s_j^{\bar{h}}(\theta)$. $s_j^{\bar{h}}(\theta)$ is included only once for all $\theta \in \Theta_j^h$. Each $\theta \in \theta_i^{ex}$ has its own $s_{max}^j(\theta)$. But s_i^h can access multiple objects, denoted as Θ_j^h. So, $s_{max}^j(\theta)$ is replaced by $s_{max}^j(\Theta_j^h)$, where $s_{max}^j(\Theta_j^h) = max\{s_{max}^j(\theta), \forall \theta \in \Theta_j^h\}$. $s_{max}^j(\Theta_j^h)$ is included once for each $\theta \in \theta_i$.

Each τ_i^x has the same interference pattern from higher priority jobs, τ_j^h, under FBLT and G-RMA/LCM. Hence, $RC_{re}(T_i)$ for τ_i^x is the same under FBLT and G-RMA/LCM. Consequently, (105) becomes:

$$\begin{aligned} & \sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k \in s_i} \left(\delta_i^k len(s_i^k) + \sum_{s_{iz}^k \in \chi_i^k} len(s_{iz}^k) \right)}{T_i} \\ \leq & \sum_{\forall \tau_i} \frac{\sum_{\forall \tau_j^* \in \gamma_i^{ex}} \sum_{\forall s_j^h(\Theta_j^h),\, \Theta_j^h \in \theta_i^{ex}} \left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) len\left(s_j^{\bar{h}}(\Theta_j^h) + \alpha_{max}^{j\bar{h}} s_{max}^j(\Theta_j^h) \right)}{T_i} \\ + & \sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k} \left(1 - \alpha_{max}^{ik}\right) len(s_{max}^i)}{T_i} \end{aligned} \tag{106}$$

Although different s_i^k can have common conflicting transactions $s_j^{\bar{h}}$, no more than one s_i^k can be preceded by the same $s_j^{\bar{h}}$ in the m_set. This happens because transactions in the m_set are non-preemptive. The original priority of transactions preceding s_i^k in the m_set can be of lower or higher priority than the original priority of s_i^k. Under G-RMA, $p_j > p_i$, which means that $T_j \leq T_i$. Hence, $\left\lceil \frac{T_i}{T_j} \right\rceil \geq 1$. For each $s_i^k \in s_i$, there are a set of zero or more $s_j^{\bar{h}}(\Theta_j^h) \in \tau_j^*$ that are conflicting with s_i^k. Assuming this set of transactions conflicting with s_i^k is denoted as $\nu_i^k = \left\{ s_j^{\bar{h}}(\Theta_j^h) \in \tau_j^* : \left(\Theta_j^h \in \theta_i^{ex}\right) \wedge \left(s_j^{\bar{h}}(\Theta_j^h) \notin \nu_i^l,\, l \neq k \right) \right\}$.

The last condition $s_j^{\bar{h}}(\Theta_j^h) \notin \nu_i^l,\, l \neq k$ in the definition of ν_i^k ensures that common transactions $s_j^{\bar{h}}$ that can conflict with more than one transaction $s_i^k \in \tau_i$ are split among different ν_i^k, $k = 1, .., |s_i|$. This condition is necessary, because in G-RMA/LCM, no two or more transactions of τ_i^x can be aborted by the same transaction of τ_j^h, where $p_j^h > p_i^x$. By substitution of ν_i^k in (106), we get:

$$\begin{aligned} & \sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k \in s_i} \left(\delta_i^k len(s_i^k) + \sum_{s_{iz}^k \in \chi_i^k} len(s_{iz}^k) \right)}{T_i} \\ \leq & \sum_{\forall \tau_i} \frac{\left(\sum_{\forall k=1}^{|s_i|} \sum_{s_j^{\bar{h}}(\Theta_j^h) \in \nu_i^k} \left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) len\left(s_j^{\bar{h}}(\Theta_j^h) + \alpha_{max}^{j\bar{h}} s_{max}^j(\Theta_j^h) \right) \right)}{T_i} \\ + & \sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k} \left(1 - \alpha_{max}^{ik}\right) len(s_{max}^i)}{T_i} \end{aligned} \tag{107}$$

$s_j^{\bar{h}}$ belongs to higher priority jobs than τ_i. s_{max}^j belongs to higher priority jobs than τ_i or τ_i itself. s_{max}^j has a lower priority than τ_j. Transactions in the m_set can belong to jobs with original priority higher or lower than τ_i. So, (107) holds if for each $s_i^k \in \tau_i$:

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

This leads to:

$$\begin{aligned}\delta_i^k &\leq \left(\sum_{s_j^{\bar{h}}(\Theta_j^h)\in\nu_i^k}\left(\left(\left\lceil\frac{T_i}{T_j}\right\rceil+1\right)len\left(\frac{s_j^{\bar{h}}(\Theta_j^h)+\alpha_{max}^{j\bar{h}}s_{max}^j(\Theta_j^h)}{s_i^k}\right)\right)\right)\\ &+ \left(1-\alpha_{max}^{ik}\right)len\left(\frac{s_{max}^i}{s_i^k}\right)-\sum_{s_{iz}^k\in\chi_i^k}len\left(\frac{s_{iz}^k}{s_i^k}\right) \qquad (108)\end{aligned}$$

Let $\epsilon = \{s_{u_{max}} : (1 \leq u \leq n) \wedge (s_{u1_{max}} \geq s_{u2_{max}}, u1 < u2)\}$, where n is the number of tasks, and $s_{u_{max}}$ is maximum transactional length in any job of τ_u. Thus, ϵ is the set of maximum transactional lengths of all tasks in non-increasing order. Each $s_{u_{max}} \in \epsilon$ belongs to a distinct task. Thus, $\sum_{s_{iz}^k\in\chi_i^k} len\left(\frac{s_{iz}^k}{s_i^k}\right) \leq \sum_{u=1,\, s_{u_{max}}\in\epsilon}^{min(n,m)-1} s_{u_{max}}$. $\sum_{u=1,\, s_{u_{max}}\in\epsilon}^{min(n,m)-1} s_{u_{max}}$ is the sum of at most maximum $m-1$ transactional lengths of all tasks. $|\chi_i^k| \leq m-1$ and $len(s_{max}^j(\Theta_j^h)) \geq len(s_i^k)$. So, (108) holds if:

$$\delta_i^k \leq \left(\sum_{s_j^{\bar{h}}(\Theta_j^h)\in\nu_i^k}\left(\left\lceil\frac{T_i}{T_j}\right\rceil+1\right)\alpha_{max}^{j\bar{h}}\right)-\sum_{u=1,\, s_{u_{max}}\in\epsilon}^{min(n,m)-1} s_{u_{max}} \qquad (109)$$

To bound the effect of transitive retry, only objects that belong to θ_i (not whole θ_i^{ex}) will be considered. So, G-RMA/LCM acts as if there were no transitive retry. Thus, ν_i^k is modified to $\bar{\nu_i^k} = \left\{s_j^{\bar{h}}(\Theta) \in \tau_j^* : \left(\Theta \in \Theta_j^h \cap \theta_i\right) \wedge \left(s_j^{\bar{h}}(\Theta) \notin \nu_i^l,\, l \neq k\right)\right\}$. Since $\bar{\nu_i^k} \subseteq \nu_i^k$, (109) still holds if ν_i^k is replaced with $\bar{\nu_i^k}$. Consequently, (109) holds if:

$$\delta_i^k \leq \left(\sum_{s_j^{\bar{h}}(\Theta)\in\bar{\nu_i^k}}\left(\left\lceil\frac{T_i}{T_j}\right\rceil+1\right)\alpha_{max}^{j\bar{h}}\right)-\sum_{u=1,\, s_{u_{max}}\in\epsilon}^{min(n,m)-1} s_{u_{max}} \qquad (110)$$

Claim follows.

17.5 FBLT vs. PNF

Claim 42 *Let* $\rho_i^j(k) = \left(\sum_{\forall s_j^{\bar{h}}(\Theta)\in\nu_i^k(j)} len\left(s_j^{\bar{h}}(\Theta)\right)\right) - s_{i_{max}}$, $\tau_j \in \gamma_i^k$. $\rho_i^j(k)$ *is the difference between the sum of transactional lengths of all transactions in* τ_j

conflicting with s_i^k, and the maximum length transaction in τ_i. Let $\sum_{u=1,\, s_{u_{max}} \in \epsilon}^{min(n,m)-1} s_{u_{max}}$ be sum of at most maximum $m-1$ transactional lengths in all tasks. Schedulability of FBLT is better or equal to PNF's when

$$\delta_i^k \leq \left(\sum_{\forall \tau_j \in \gamma_i^k} \left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) len \left(\frac{\left(\sum_{\forall \bar{s_j^h}(\Theta) \in \nu_i^k(j)} len\left(\bar{s_j^h}(\Theta) \right) \right) - s_{i_{max}}}{s_i^k} \right) \right) \quad (111)$$

$$- \sum_{u=1,\, s_{u_{max}} \in \epsilon}^{min(n,m)-1} s_{u_{max}} \quad (112)$$

Proof 42 By substituting $RC_A(T_i)$ and $RC_B(T_i)$ in (85) with (82) and (6.1) in [15], respectively, we get:

$$\sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k \in s_i} \left(\delta_i^k len(s_i^k) + \sum_{s_{iz}^k \in \chi_i^k} len(s_{iz}^k) \right) + RC_{re}(T_i)}{T_i} \quad (113)$$

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

$\bar{s_j^h}(\theta)$ can access multiple objects. $\bar{s_j^h}(\theta)$ is included only once for all objects accessed by it. $RC_{re}(T_i)$ is given by (6.8) in [15] in case of G-EDF, and (6.10) in [15] in case of G-RMA. Substituting $RC_{re}(T_i)$ given by (6.8) and (6.10) in [15] with $RC_{re}(T_i) = \sum_{\forall \tau_j \in \gamma_i} \left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) s_{i_{max}}$, we ensure correctness of (113). If τ_j has no shared objects with τ_i, then the release of any higher priority job τ_j^y will not abort any transaction in any job of τ_i. Thus, (113) holds if:

$$\sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k \in s_i} \left(\delta_i^k len(s_i^k) + \sum_{s_{iz}^k \in \chi_i^k} len(s_{iz}^k) \right)}{T_i} \quad (114)$$

$$\leq \sum_{\forall \tau_i} \frac{\sum_{\forall \tau_j \in \gamma_i} \left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \left(\left(\sum_{\forall \bar{s_j^h}(\Theta),\, \Theta \in \theta_i} len\left(\bar{s_j^h}(\theta) \right) \right) - s_{i_{max}} \right)}{T_i}$$

For each $s_i^k \in s_i$, there are a set of zero or more $\bar{s_j^h}(\Theta) \in \tau_j$, $\forall \tau_j \neq \tau_i$ that are conflicting with s_i^k. Assuming this set of transactions conflicting with s_i^k is denoted as $\nu_i^k(j) = \left\{ \bar{s_j^h}(\Theta) \in \tau_j : (\Theta \in \theta_i) \wedge (\tau_j \neq \tau_i) \wedge \left(\bar{s_j^h}(\Theta) \notin \nu_i^l,\, l \neq k \right) \right\}$. The last condition $\bar{s_j^h}(\Theta) \notin \nu_i^l$, $l \neq k$ in the definition of ν_i^k ensures that common transactions $\bar{s_j^h}$ that can conflict with more than one transaction $s_i^k \in \tau_i$ are split among different $\nu_i^k(j)$, $k = 1, .., |s_i|$. This condition is necessary, because in PNF, no two or more transactions of τ_i^x can be aborted by the same transaction of τ_j^h.

Let γ_i^k be the subset of γ_i that contains tasks with transactions conflicting directly with s_i^k. By substitution of ν_i^k and γ_i in (114) by $\nu_i^k(j)$ and γ_i^k, (114) holds if for each s_i^k:

$$\delta_i^k + \sum_{s_{iz}^k \in \chi_i^k} len(\tfrac{s_{iz}^k}{s_i^k}) \quad (115)$$

$$\leq \sum_{\forall \tau_j \in \gamma_i^k} \left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \left(\left(\sum_{\forall \bar{s_j^h}(\Theta) \in \nu_i^k(j)} len\left(\frac{\bar{s_j^h}(\Theta)}{s_i^k} \right) \right) - len\left(\frac{s_{i_{max}}}{s_i^k} \right) \right)$$

Non-preemptive transactions preceding s_i^k in the m_set can directly or indirectly conflict with s_i^k. Under PNF, transactions can only directly conflict with s_i^k. Thus, s_{iz}^k on the left hand side of (115) is not necessarily included in $\bar{s_j^h}(\Theta)$ on the right hand side of (115). Let $\epsilon = \{s_{u_{max}} : (1 \leq u \leq n) \wedge (s_{u1_{max}} \geq s_{u2_{max}}, u1 < u2)\}$, where n is the number of tasks, and $s_{u_{max}}$ is the maximum transactional length in any job of τ_u. Thus, ϵ is the set of maximum transactional lengths of all tasks in non-increasing order. Each $s_{u_{max}} \in \epsilon$ belongs to a distinct task. Thus, $\sum_{s_{iz}^k \in \chi_i^k} len\left(\frac{s_{iz}^k}{s_i^k}\right) \leq \sum_{u=1,\, s_{u_{max}} \in \epsilon}^{min(n,m)-1} s_{u_{max}}$. $\sum_{u=1,\, s_{u_{max}} \in \epsilon}^{min(n,m)-1} s_{u_{max}}$ is the sum of at most maximum $m-1$ transactional lengths of all tasks. $|\chi_i^k| \leq m-1$. Then (115) holds if:

$$\begin{aligned}\delta_i^k \quad \leq \quad & \left(\sum_{\forall \tau_j \in \gamma_i^k} \left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1\right) len\left(\frac{\left(\sum_{\forall \bar{s_j^h}(\Theta) \in \nu_i^k(j)} len\left(\bar{s_j^h}(\Theta)\right)\right) - s_{i_{max}}}{s_i^k}\right)\right) \\ - \quad & \sum_{u=1,\, s_{u_{max}} \in \epsilon}^{min(n,m)-1} s_{u_{max}}\end{aligned}$$

Since $\rho_i^j(k) = \left(\sum_{\forall \bar{s_j^h}(\Theta) \in \nu_i^k(j)} len\left(\bar{s_j^h}(\Theta)\right)\right) - s_{i_{max}}$, $\tau_j \in \gamma_i^k$, and $\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1\right)$ is the total number of jobs of τ_j interfering with τ_i, claim follows.

17.6 FBLT vs. Lock-free

Claim 43 *Under G-EDF and G-RMA, the schedulability of FBLT is equal or better than that under lock-free synchronization if $s_{max} \leq r_{max}$. If transactions execute in FIFO order (i.e., $\delta_i^k = 0, \forall s_i^k$) and contention is high, s_{max} can be much larger than r_{max}.*

Proof 43 Lock-free synchronization [13, 24] allows only one object to be synchronized at a given time (e.g., a lock-free stack). Thus, for comparing FBLT's schedulability with lock-free, we limit the number of accessed objects per transaction under FBLT to one.

By substituting $RC_A(T_i)$ and $RC_B(T_i)$ in (85) with (82) and (6.17) in [15], respectively, we get:

$$\begin{aligned} & \sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k \in s_i} \left(\delta_i^k len(s_i^k) + \sum_{s_{iz}^k \in \chi_i^k} len(s_{iz}^k)\right) + RC_{re}(T_i)}{T_i} \\ \leq \quad & \sum_{\forall \tau_i} \frac{\left(\sum_{\forall \tau_j \in \gamma_i} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1\right) \beta_{i,j} r_{max}\right)\right) + RC_{re}(T_i)}{T_i} \end{aligned} \tag{116}$$

where $\beta_{i,j}$ is the number of retry loops of τ_j that access the same objects as accessed by any retry loop of τ_i [13] and r_{max} is the maximum execution cost of a single iteration of any retry loop of any task [13].

For G-EDF (G-RMA), any job τ_i^x under FBLT has the same pattern of interference from higher priority jobs as ECM (RCM), respectively. $RC_{re}(T_i)$ for ECM, RCM, and lock-free are given by Claims 25, 26, and 27 in [15], respectively. $\therefore RC_{re}(T_i) = \left\lceil \frac{T_i}{T_j} \right\rceil s_{i_{max}}, \forall \tau_j \neq \tau_i$ for G-EDF/FBLT and G-

RMA/FBLT. $RC_{re}(T_i) = \left\lceil \frac{T_i}{T_j} \right\rceil r_{i_{max}}$, $\forall \tau_j \neq \tau_i$ for G-EDF/lock-free and G-RMA/lock-free. $\therefore$ (122) becomes:

$$\begin{aligned} & \sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k \in s_i} \left(\delta_i^k len(s_i^k) + \sum_{s_{iz}^k \in \chi_i^k} len(s_{iz}^k) \right) + \sum_{\forall \tau_j \neq \tau_i} \left\lceil \frac{T_i}{T_j} \right\rceil s_{i_{max}}}{T_i} \\ \leq & \sum_{\forall \tau_i} \frac{\left(\sum_{\forall \tau_j \in \gamma_i} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \beta_{i,j} r_{max} \right) \right) + \sum_{\forall \tau_j \neq \gamma_i} \left\lceil \frac{T_i}{T_j} \right\rceil r_{i_{max}}}{T_i} \end{aligned} \tag{117}$$

Since $s_{max} \geq s_{i_{max}}$, $len(s_i^k)$, $len(s_{iz}^k)$, $\forall i, z, k$ and $r_{max} \geq r_{i_{max}}$ $\therefore$ (123) holds if:

$$\begin{aligned} & \sum_{\forall \tau_i} \frac{\left(\left(\sum_{\forall s_i^k \in s_i} \left(\delta_i^k + |\chi_i^k| \right) \right) + \sum_{\forall \tau_j \neq \tau_i} \left\lceil \frac{T_i}{T_j} \right\rceil \right) s_{max}}{T_i} \\ \leq & \sum_{\forall \tau_i} \frac{\left(\left(\sum_{\forall \tau_j \in \gamma_i} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \beta_{i,j} \right) \right) + \sum_{\forall \tau_j \neq \tau_i} \left\lceil \frac{T_i}{T_j} \right\rceil \right) r_{max}}{T_i} \end{aligned} \tag{118}$$

$\therefore$ (124) holds if for each τ_i:

$$\begin{aligned} & \left(\left(\sum_{\forall s_i^k \in s_i} \left(\delta_i^k + |\chi_i^k| \right) \right) + \sum_{\forall \tau_j \neq \tau_i} \left\lceil \frac{T_i}{T_j} \right\rceil \right) s_{max} \\ \leq & \left(\left(\sum_{\forall \tau_j \in \gamma_i} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \beta_{i,j} \right) \right) + \sum_{\forall \tau_j \neq \tau_i} \left\lceil \frac{T_i}{T_j} \right\rceil \right) r_{max} \end{aligned} \tag{119}$$

$\therefore$

$$\frac{s_{max}}{r_{max}} \leq \frac{\left(\sum_{\forall \tau_j \in \gamma_i} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \beta_{i,j} \right) \right) + \sum_{\forall \tau_j \neq \tau_i} \left\lceil \frac{T_i}{T_j} \right\rceil}{\left(\sum_{\forall s_i^k \in s_i} \left(\delta_i^k + |\chi_i^k| \right) \right) + \sum_{\forall \tau_j \neq \tau_i} \left\lceil \frac{T_i}{T_j} \right\rceil} \tag{120}$$

It appears from (126) that as δ_i^k and $|\chi_i^k|$ increases, s_{max}/r_{max} decreases. So, to get the lower bound on s_{max}/r_{max}, let $\sum_{\forall s_i^k \in s_i} \left(\delta_i^k + |\chi_i^k| \right)$ reach its maximum value. This maximum value is the total number of interfering transactions belonging to any job τ_j^l, $j \neq i$. The priority of τ_j^l can be higher or lower than the current instance of τ_i. Beyond this maximum value, there will be no more transactions that conflict with s_i^k. Thus, higher values for any δ_i^k beyond the maximum value will be ineffective. $\therefore$ $\sum_{\forall s_i^k \in s_i} \left(\delta_i^k + |\chi_i^k| \right) \leq \sum_{\forall \tau_j \in \gamma_i} \left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right)$. Consequently, (126) will be:

$$\frac{s_{max}}{r_{max}} \leq \frac{\left(\sum_{\forall \tau_j \in \gamma_i} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \beta_{i,j} \right) \right) + \sum_{\forall \tau_j \neq \tau_i} \left\lceil \frac{T_i}{T_j} \right\rceil}{\left(\sum_{\forall \tau_j \in \gamma_i} \left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \right) + \sum_{\forall \tau_j \neq \tau_i} \left\lceil \frac{T_i}{T_j} \right\rceil} \tag{121}$$

Since we are seeking the lower bound on $\frac{s_{max}}{r_{max}}$, let $\beta_{i,j}$ assume its minimum value. Thus, $\beta_{i,j} = 1$. $\therefore$ (127) holds if $\frac{s_{max}}{r_{max}} \leq 1$.

Let $\delta_i^k(T_i) \to 0$ in (126). This means that transactions approximately execute according to their arrival order. Let $\beta_{i,j} \to \infty$, $\left\lceil \frac{T_i}{T_j} \right\rceil \to \infty$ in (126). This implies high contention. Consequently, $\frac{s_{max}}{r_{max}} \to \infty$. Hence, if transactions execute in FIFO order and contention is high, s_{max} can be much larger than r_{max}. Claim follows.

Claim 44 *Under G-EDF and G-RMA, schedulability of FBLT is equal or better than lock-free's if $s_{max} \leq r_{max}$. If transactions execute in FIFO order (i. e., $\delta_i^k = 0, \forall s_i^k$) and contention is high, s_{max} can be much larger than r_{max}.*

Proof 44 Lock-free synchronization [13, 17] accesses only one object. Thus, the number of accessed objects per transaction in FBLT is limited to one. This allows us to compare the schedulability of FBLT with the lock-free algorithm.

By substituting $RC_A(T_i)$ and $RC_B(T_i)$ in (85) with (82) and (6.17) in [15] respectively.

$$\begin{aligned} & \sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k \in s_i} \left(\delta_i^k len(s_i^k) + \sum_{s_{iz}^k \in \chi_i^k} len(s_{iz}^k) \right) + RC_{re}(T_i)}{T_i} \\ \leq & \sum_{\forall \tau_i} \frac{\left(\sum_{\forall \tau_j \in \gamma_i} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \beta_{i,j} r_{max} \right) \right) + RC_{re}(T_i)}{T_i} \end{aligned} \tag{122}$$

where $\beta_{i,j}$ is the number of retry loops of τ_j that access the same objects as accessed by any retry loop of τ_i [13]. r_{max} is the maximum execution cost of a single iteration of any retry loop of any task [13]. For G-EDF(G-RMA), any job τ_i^x under FBLT has the same pattern of interference from higher priority jobs as ECM(RCM) respectively. $RC_{re}(T_i)$ for ECM, RCM and lock-free are given by Claims 25, 26 and 27 in [15] respectively. $\therefore$ $RC_{re}(T_i) = \left\lceil \frac{T_i}{T_j} \right\rceil s_{i_{max}}, \forall \tau_j \neq \tau_i$ covers $RC_{re}(T_i)$ for G-EDF/FBLT and G-RMA/FBLT. $RC_{re}(T_i) = \left\lceil \frac{T_i}{T_j} \right\rceil r_{i_{max}}, \forall \tau_j \neq \tau_i$ covers retry cost for G-EDF/lock-free and G-RMA/lock-free. $\therefore$ (122) becomes

$$\begin{aligned} & \sum_{\forall \tau_i} \frac{\sum_{\forall s_i^k \in s_i} \left(\delta_i^k len(s_i^k) + \sum_{s_{iz}^k \in \chi_i^k} len(s_{iz}^k) \right) + \sum_{\forall \tau_j \neq \tau_i} \left\lceil \frac{T_i}{T_j} \right\rceil s_{i_{max}}}{T_i} \\ \leq & \sum_{\forall \tau_i} \frac{\left(\sum_{\forall \tau_j \in \gamma_i} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \beta_{i,j} r_{max} \right) \right) + \sum_{\forall \tau_j \neq \gamma_i} \left\lceil \frac{T_i}{T_j} \right\rceil r_{i_{max}}}{T_i} \end{aligned} \tag{123}$$

Since $s_{max} \geq s_{i_{max}}$, $len(s_i^k)$, $len(s_{iz}^k)$, $\forall i, z, k$ and $r_{max} \geq r_{i_{max}}$ $\therefore$ (123) holds if

$$\begin{aligned} & \sum_{\forall \tau_i} \frac{\left(\left(\sum_{\forall s_i^k \in s_i} \left(\delta_i^k + |\chi_i^k| \right) \right) + \sum_{\forall \tau_j \neq \tau_i} \left\lceil \frac{T_i}{T_j} \right\rceil \right) s_{max}}{T_i} \\ \leq & \sum_{\forall \tau_i} \frac{\left(\left(\sum_{\forall \tau_j \in \gamma_i} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \beta_{i,j} \right) \right) + \sum_{\forall \tau_j \neq \tau_i} \left\lceil \frac{T_i}{T_j} \right\rceil \right) r_{max}}{T_i} \end{aligned} \tag{124}$$

$\therefore$ (124) holds if for each τ_i

$$\begin{aligned} & \left(\left(\sum_{\forall s_i^k \in s_i} \left(\delta_i^k + |\chi_i^k| \right) \right) + \sum_{\forall \tau_j \neq \tau_i} \left\lceil \frac{T_i}{T_j} \right\rceil \right) s_{max} \\ \leq & \left(\left(\sum_{\forall \tau_j \in \gamma_i} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \beta_{i,j} \right) \right) + \sum_{\forall \tau_j \neq \tau_i} \left\lceil \frac{T_i}{T_j} \right\rceil \right) r_{max} \end{aligned} \tag{125}$$

$\therefore$

$$\frac{s_{max}}{r_{max}} \leq \frac{\left(\sum_{\forall \tau_j \in \gamma_i} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \beta_{i,j} \right) \right) + \sum_{\forall \tau_j \neq \tau_i} \left\lceil \frac{T_i}{T_j} \right\rceil}{\left(\sum_{\forall s_i^k \in s_i} \left(\delta_i^k + |\chi_i^k| \right) \right) + \sum_{\forall \tau_j \neq \tau_i} \left\lceil \frac{T_i}{T_j} \right\rceil} \tag{126}$$

It appears from (126) that as δ_i^k, as well as $|\chi_i^k|$, increases, then s_{max}/r_{max} decreases. So, to get the lower bound on s_{max}/r_{max}, let $\sum_{\forall s_i^k \in s_i} \left(\delta_i^k + |\chi_i^k| \right)$

reaches its maximum value. This maximum value is the total number of interfering transactions belonging to any job τ_j^l, $j \neq i$. Priority of τ_j^l can be higher or lower than current instance of τ_i. Beyond this maximum value, there will be no more transactions to conflict with s_i^k. So, higher values for any δ_i^k beyond maximum value will be ineffective. $\therefore \sum_{\forall s_i^k \in s_i} \left(\delta_i^k + |\chi_i^k|\right) \leq \sum_{\forall \tau_j \in \gamma_i} \left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right)$. Consequently, (126) will be

$$\frac{s_{max}}{r_{max}} \leq \frac{\left(\sum_{\forall \tau_j \in \gamma_i} \left(\left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right) \beta_{i,j} \right)\right) + \sum_{\forall \tau_j \neq \tau_i} \left\lceil \frac{T_i}{T_j} \right\rceil}{\left(\sum_{\forall \tau_j \in \gamma_i} \left(\left\lceil \frac{T_i}{T_j} \right\rceil + 1 \right)\right) + \sum_{\forall \tau_j \neq \tau_i} \left\lceil \frac{T_i}{T_j} \right\rceil} \quad (127)$$

As we look for the lower bound on $\frac{s_{max}}{r_{max}}$, let $\beta_{i,j}$ assumes its minimum value. So, $\beta_{i,j} = 1$. $\therefore$ (127) holds if $\frac{s_{max}}{r_{max}} \leq 1$.

Let $\delta_i^k(T_i) \to 0$ in (126). This means transactions approximately execute in their arrival order. Let $\beta_{i,j} \to \infty$, $\left\lceil \frac{T_i}{T_j} \right\rceil \to \infty$ in (126) . This means contention is high. Consequently, $\frac{s_{max}}{r_{max}} \to \infty$. So, if transactions execute in FIFO order and contention is high, s_{max} can be much larger than r_{max}. Claim follows.

18 Experimental Evaluation

Having established different CMs' retry and response time upper bounds, and the conditions under which each CM is prefered, we now would like to understand how different CMs' retry and response times in practice (i.e., on average). Since this can only be understood experimentally, we implement all synchronization techniques and conduct experimental studies.

18.1 Experimental Setup

We used the ChronOS real-time Linux kernel [12] and the RSTM library [30]. We modified RSTM to include implementations of ECM, RCM, LCM, PNF and FBLT 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 2. Each task runs in its own thread and has a set of atomic sections. 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.

18.2 LCM Results

Figure 15 shows the retry cost (RC) for each task in the some task sets in Table 2, where each task has a single atomic section of length equal to 0.2 of the task

Table 2: Task sets a) 4 tasks. b) 5 tasks. c) 8 tasks. d) 10 tasks. e) 12 tasks. c) 20 tasks.

(a)

$P_i(\mu s)$	$c_i(\mu s)$
1000000	227000
1500000	410000
3000000	299000
5000000	500000

(b)

$P_i(\mu s)$	$c_i(\mu s)$
500000	150000
1000000	227000
1500000	410000
3000000	299000
5000000	500000

(c)

$P_i(\mu s)$	$c_i(\mu s)$
1500000	961000
1875000	175000
2500000	205000
3000000	129000
3750000	117000
5000000	269000
7500000	118000
15000000	609000

(d)

$P_i(\mu s)$	$c_i(\mu s)$
400000	75241
750000	69762
1200000	267122
1500000	69863
2400000	152014
4000000	286301
7500000	493150
10000000	794520
15000000	1212328
20000000	1775342

(e)

$P_i(\mu s)$	$c_i(\mu s)$
400000	58195
750000	53963
1000000	206330
1200000	53968
1500000	117449
2400000	221143
3000000	290428
4000000	83420
7500000	380917
10000000	613700
15000000	936422
20000000	1371302

(f)

$P_i(\mu s)$	$c_i(\mu s)$
375000	9000
400000	8000
500000	8000
600000	14000
625000	375000
750000	19000
1000000	26000
1200000	17000
1250000	21000
1500000	33000
1875000	39000
2000000	43000
2500000	18000
3000000	90000
3750000	28000
5000000	126000
7500000	231000
10000000	407000
15000000	261000
30000000	369000
375000	8000
30000000	407000

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 can be delayed 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 16 shows the response time of each task of the task sets in Table 2 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.

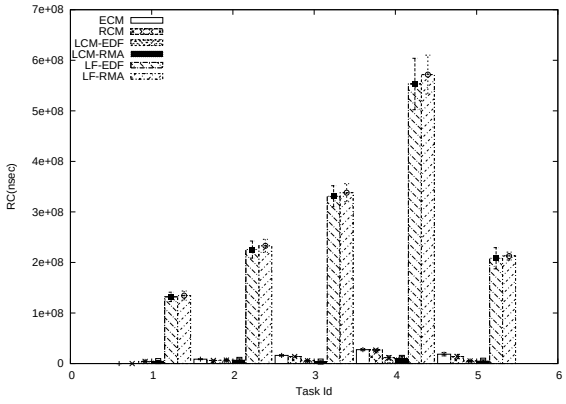
We repeated the experiments by varying the number and length of atomic sections. These results are shown in Appendix B. Each figure's label has three parameters x, y, z. x specifies the relative total length of all atomic sections to the length of the task. y specifies the maximum relative length of any atomic section to the length of the task. z specifies the minimum relative length of any atomic section to the length of the task. These figures show a consistent trend with previous results.

18.3 PNF Results

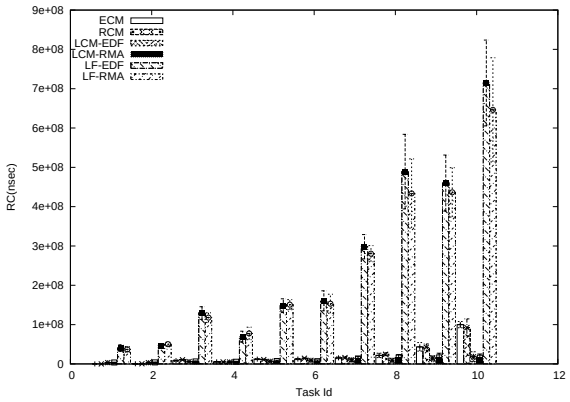
The difficulty in testing with PNF is to incur transitive retry cases. Tasks are arranged in non-decreasing order of periods, and each task shares objects only with the previous and next tasks. Each task begins with an atomic section. Thus, increasing the opportunity of transitive retry.

Figure 17 shows average retry cost under ECM, RCM, LCM, PNF and lock-free for each task set. Figure 18 shows average retry cost for only contention managers for each task set. The x-axis has three parameters a, b, c. a specifies the relative total length of all atomic sections to the length of the task. b specifies the maximum relative length of any atomic section to the length of the task. c specifies the minimum relative length of any atomic section to the length of the task. Each data point in the figure has a confidence level of 0.95. Only one object per transaction is shared in Figures 17 and 18.

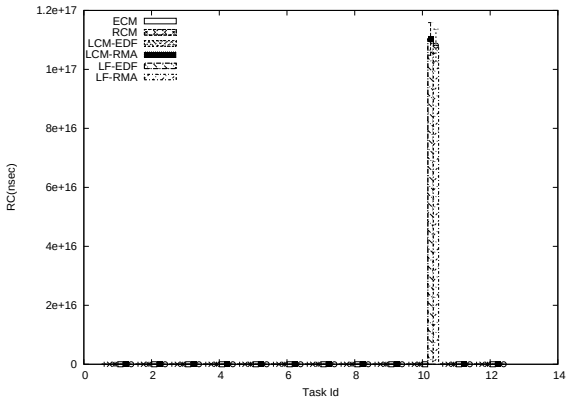
Lock-free is the longest technique as it provides no conflict resolution. PNF better or comparable retry cost than ECM, RCM and LCM. As we move from 4 to 8 to 20 task set, retry costs of different contention managers get closer to each other. This is explained by noting that each task set in Table 2 is organized in non-decreasing order of periods, and c_i/T_i for almost each τ_i is low. Besides,



(a) Task set 1

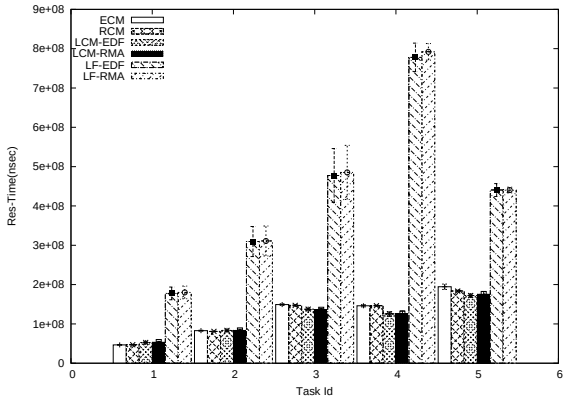


(b) Task set 2

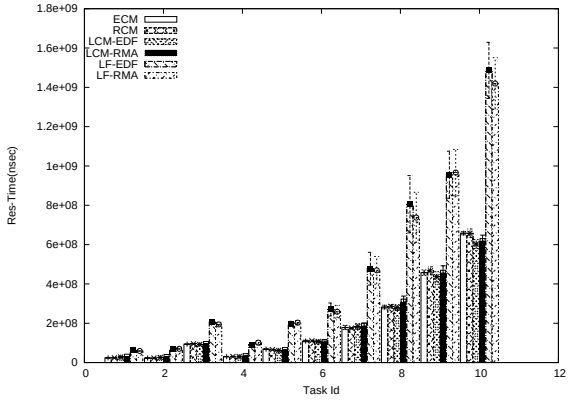


(c) Task set 3

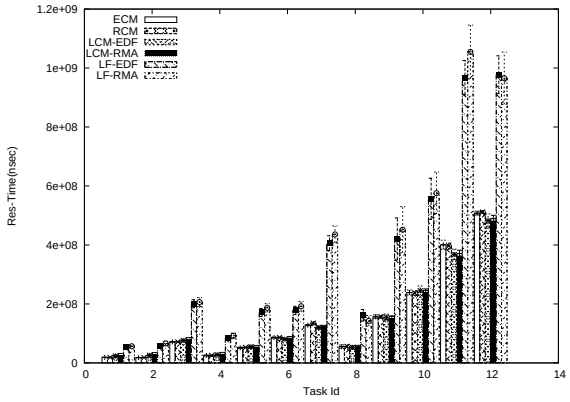
Figure 15: Task retry costs under LCM and competitor synchronization methods



(a) Task set 1



(b) Task set 2



(c) Task set 3

Figure 16: Task response times under LCM and competitor synchronization methods

each task shares objects only with the previous and next tasks, and tasks are released at the same time to enforce transitive retry. While the first instances of all tasks have a high potential of conflict, the contention level decreases with time for higher number of tasks. Thus, for the 20 task set, contention level is the lowest. Hence, retry costs of all contention managers get closer as number of tasks increases.

We compared retry cost for different contention managers with multiple objects per transaction and different levels of read/writer operations. Figure 19 shows retry cost of the three task sets sharing 20 objects per transaction, with 40% write operations and 60% read operations. The same experiment is repeated in Figure 20 with 80% write operations, and 20% read operations. Figure 21 repeats the same experiment with 100% write operations. The same previous three experiments were repeated in Figures 22, 23 and 24 with 40 objects per transaction. Figures 19 to 24 show consistent trends with Figure 18 except that retry cost of PNF is shorter than the others even with increasing number of tasks. For the 20 task set, PNF retry cost is a little shorter than LCM, but much better than ECM and RCM. This happens because of sharing multiple objects per transaction. Thus, contention level is increased than in sharing 1 object per transaction. Besides, transitive retry exists which makes PNF better than the others.

18.4 FBLT Results

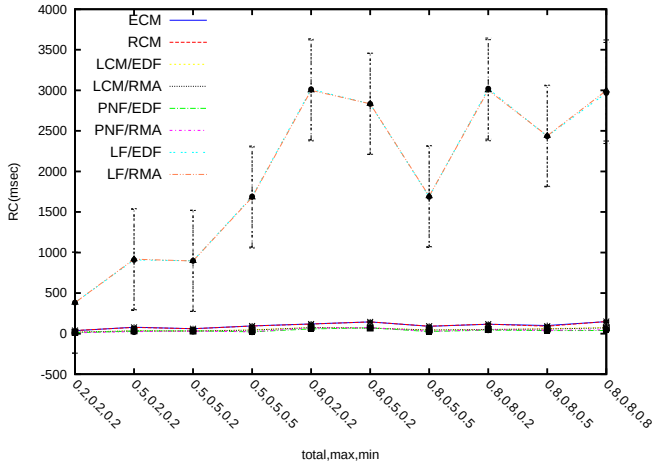
We repeated the experiments in Section 18.3 including FBLT. While Figure 26(a) includes all synchronization methods, Figure 26(b) excludes lock-free. From these figures, we observe that lock-free has the largest retry cost, as it provides no conflict resolution. FBLT has the largest retry cost among CMs, because transactions share only one object in this case. For multiple objects per transaction, FBLT provides equal or shorter retry cost than LCM, as shown in Figures 27 and 28. PNF has an advantage over FBLT. However, PNF requires a-priori knowledge of all objects accessed by each transaction, whereas FBLT does not. Consequently, retry cost under PNF is shorter than that under FBLT. For 8 and 20 task sets, FBLT's retry cost is comparable to PNF's as shown in Figures 34 to 32. So, experiments show that FBLT's retry cost can be shorter than that under ECM, RCM, and LCM, and can be comparable to that of PNF's.

PNF was designed to avoid transitive retry. Previous experiments compares retry cost of different CMs in case of transitive retry. Figures 27 to 35 compare retry costs of different CMs in case of non-transitive retry. FBLT achieves shorter or comparable retry cost to other CMs including PNF.

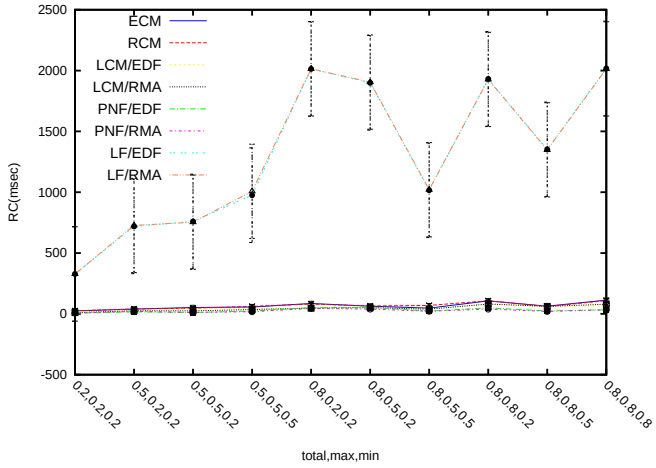
19 Conclusions

We designed, analyzed, and experimentally evaluated four real-time CMs. Designing real-time CMs is straightforward. The simplest logic is to use the same rationale as that of the underlying real-time scheduler. This was shown in the design of ECM and RCM. ECM allows the transaction with the earliest absolute deadline (i.e., dynamic priority) to commit first. RCM allows the transaction with the smallest period (i.e., fixed priority) to commit first. We established upper bounds for retry costs and response times under ECM and RCM, and identified the conditions under which they have better schedulability than lock-free synchronization.

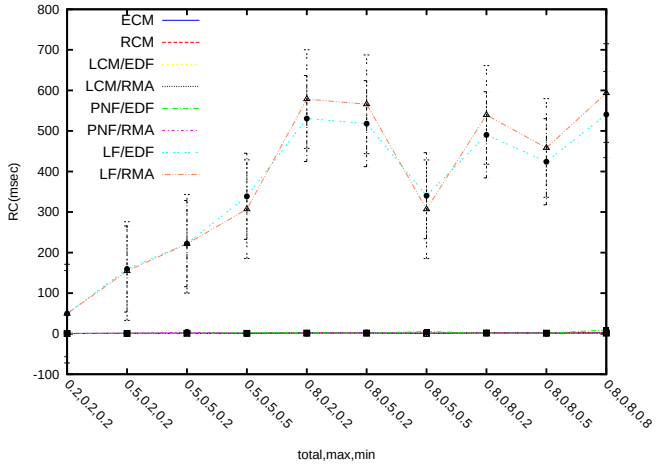
Under both ECM and RCM, a task incurs $2.s_{max}$ retry cost for each of its atomic sections due to a conflict with another task's atomic section. Retries under RCM and lock-free synchronization are affected by a larger number of



(a) 4 tasks

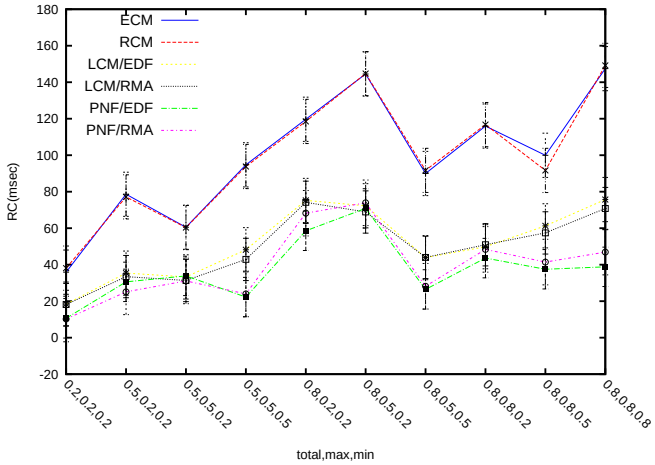


(b) 8 tasks

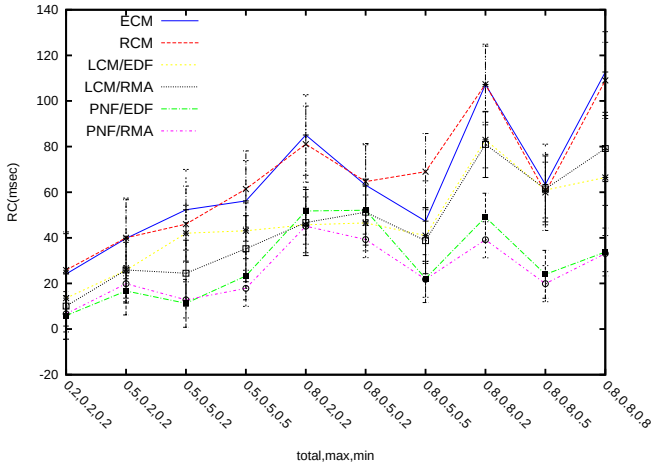


(c) 20 tasks

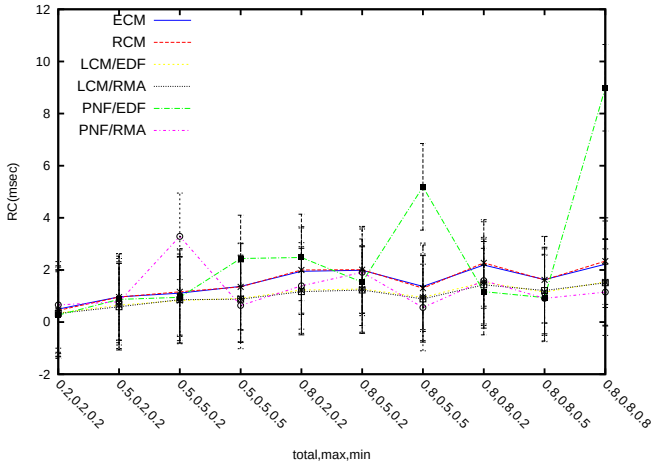
Figure 17: Average retry cost for 1 object per transaction for different values of total, maximum and minimum atomic section length under all synchronization techniques



(a) 4 tasks

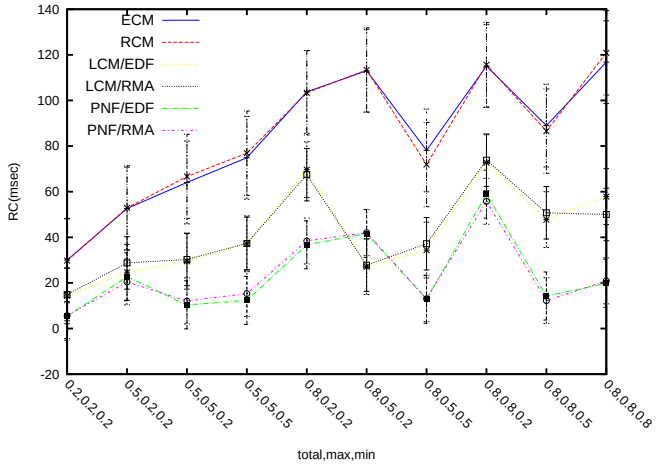


(b) 8 tasks

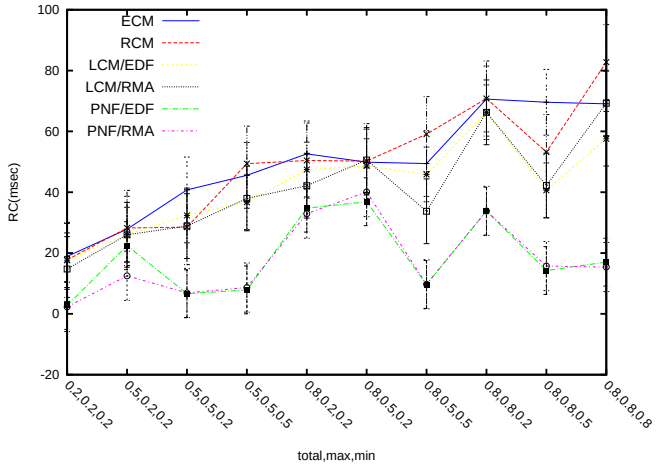


(c) 20 tasks

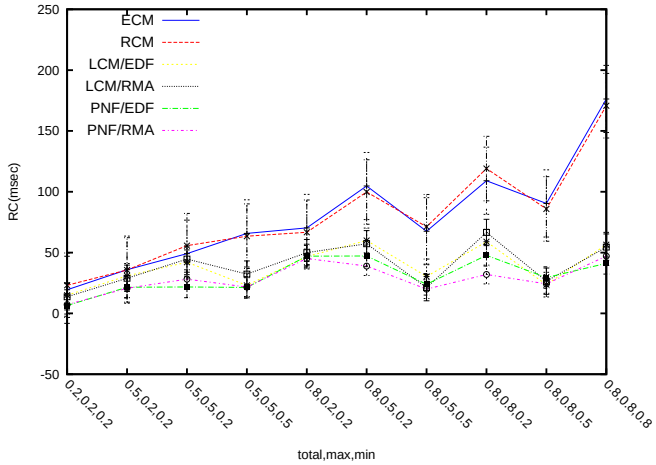
Figure 18: Average retry cost for 1 object per transaction for different values of total, maximum and minimum atomic section length under contention managers only



(a) 4 tasks

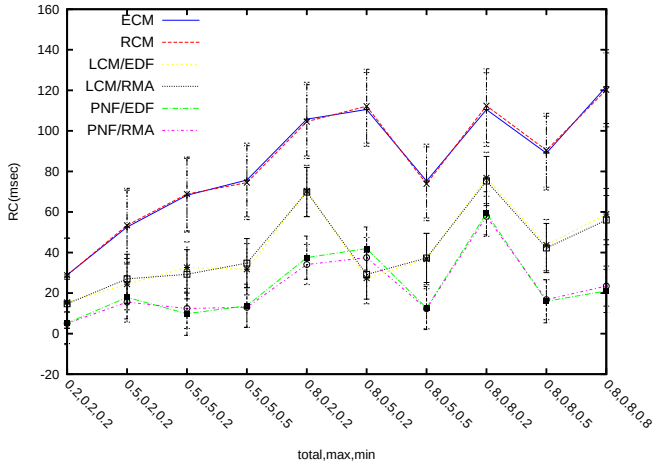


(b) 8 tasks

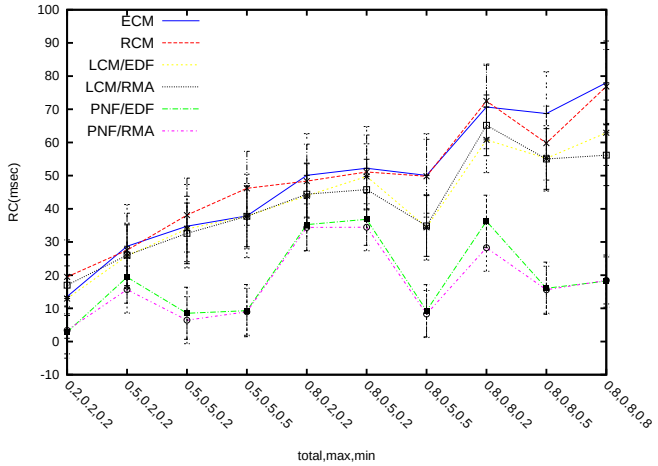


(c) 20 tasks

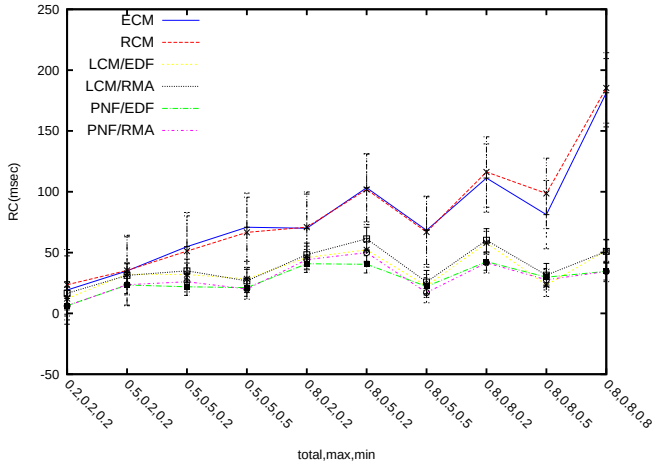
Figure 19: Average retry cost for 20 objects per transaction, 40% write operations for different values of total, maximum and minimum atomic section length under different CMs



(a) 4 tasks

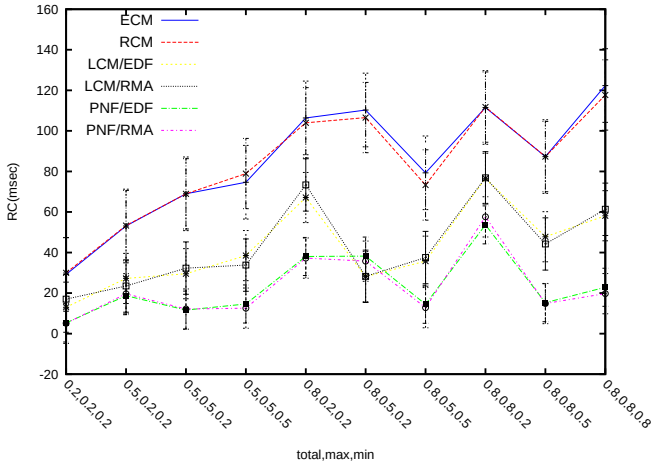


(b) 8 tasks

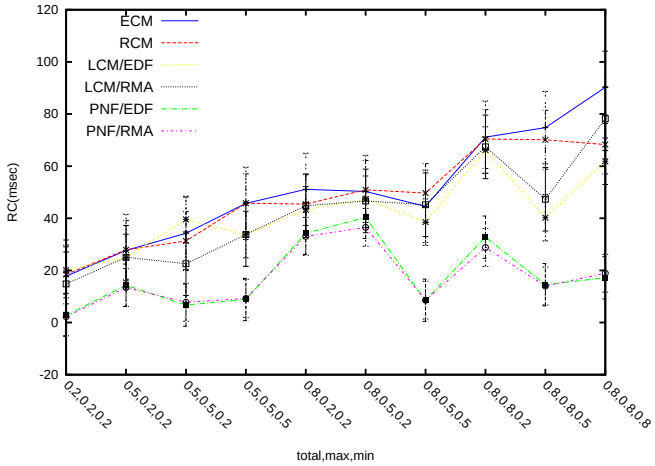


(c) 20 tasks

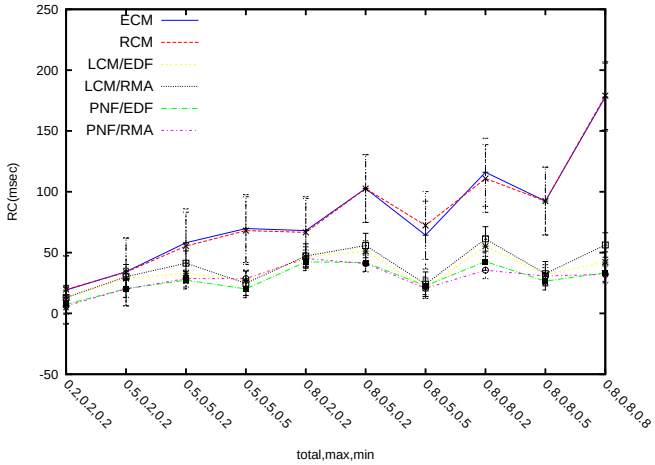
Figure 20: Average retry cost for 20 objects per transaction, 80% write operations for different values of total, maximum and minimum atomic section length under different CMs



(a) 4 tasks

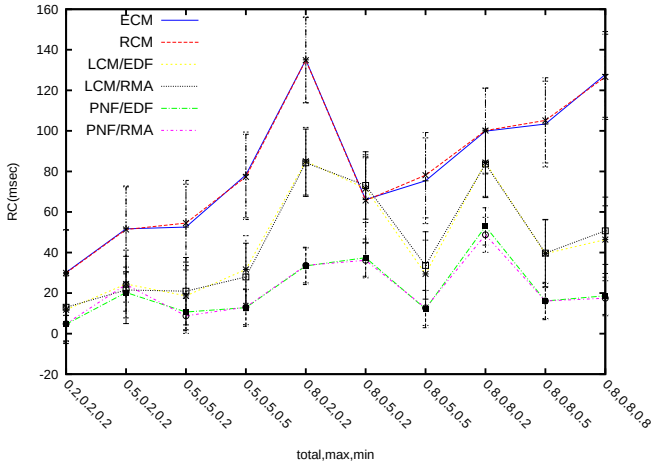


(b) 8 tasks

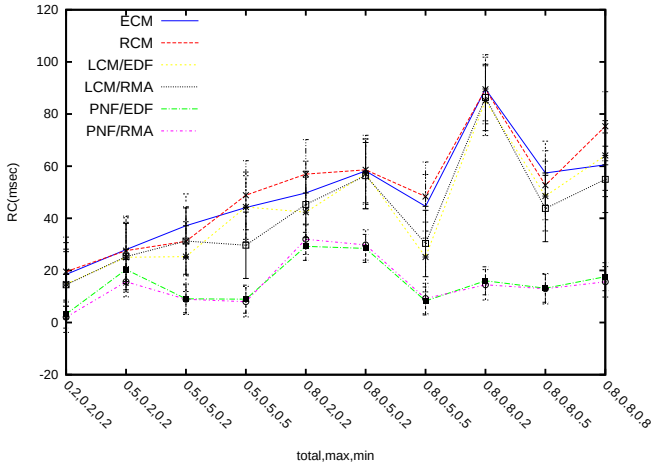


(c) 20 tasks

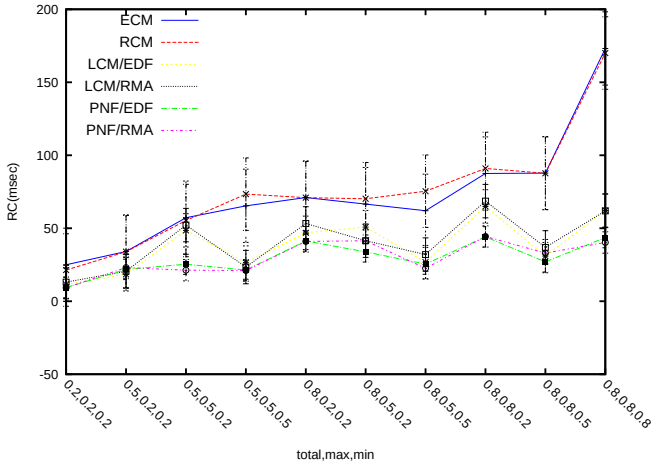
Figure 21: Average retry cost for 20 objects per transaction, 100% write operations for different values of total, maximum and minimum atomic section length under different CMs



(a) 4 tasks

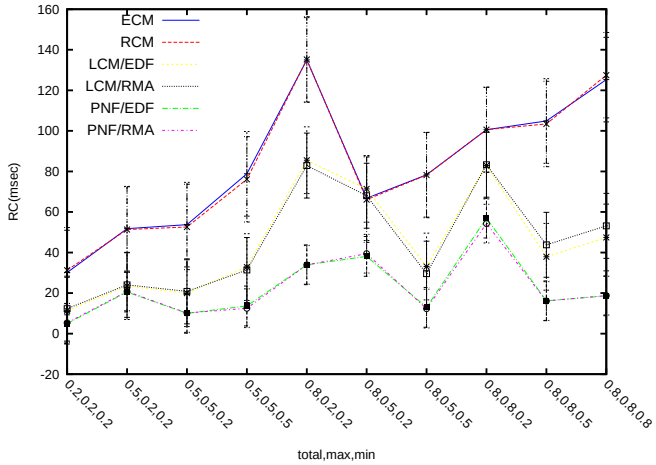


(b) 8 tasks

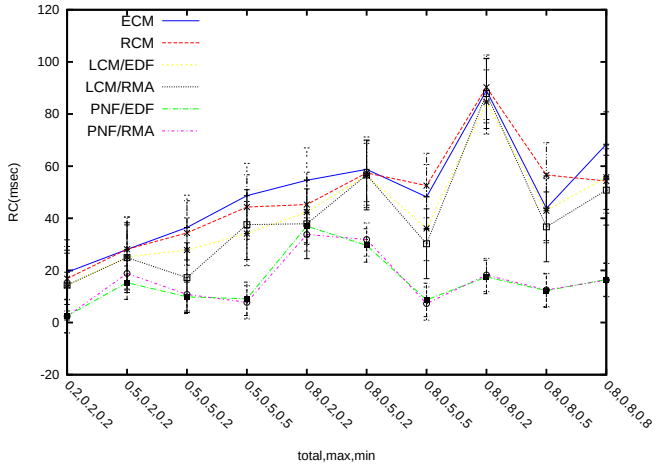


(c) 20 tasks

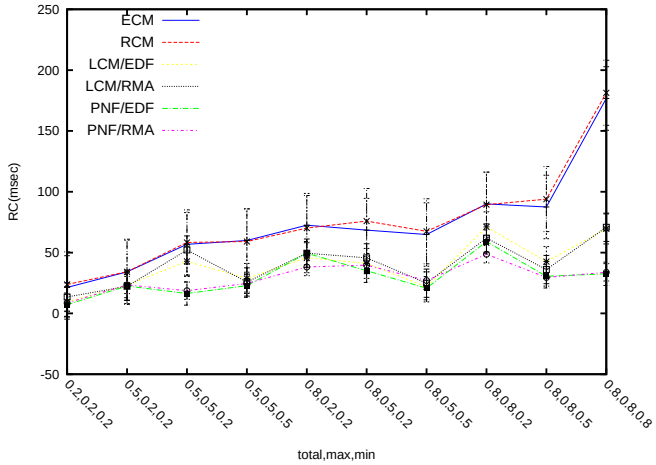
Figure 22: Average retry cost for 40 objects per transaction, 40% write operations for different values of total, maximum and minimum atomic section length under different CMs



(a) 4 tasks

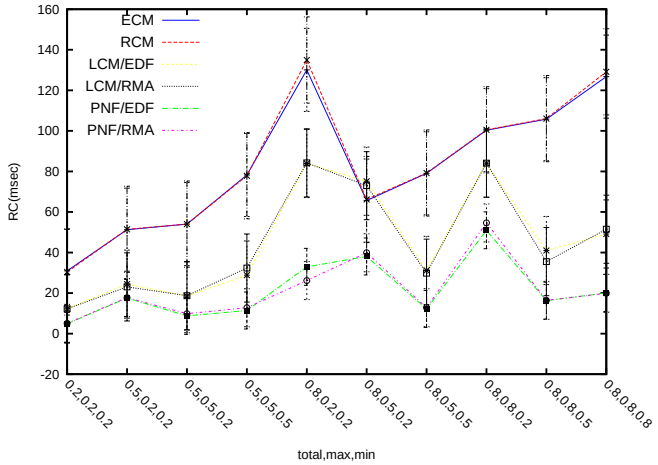


(b) 8 tasks

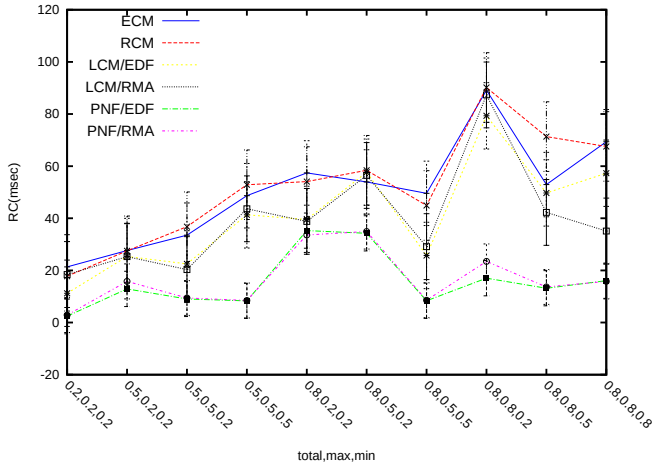


(c) 20 tasks

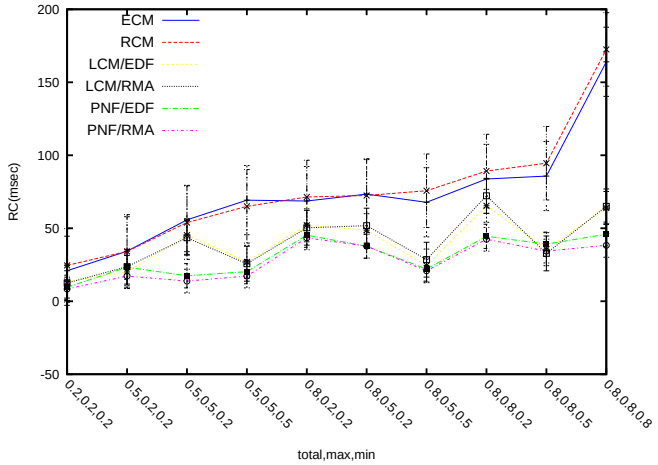
Figure 23: Average retry cost for 40 objects per transaction, 80% write operations for different values of total, maximum and minimum atomic section length under different CMs



(a) 4 tasks

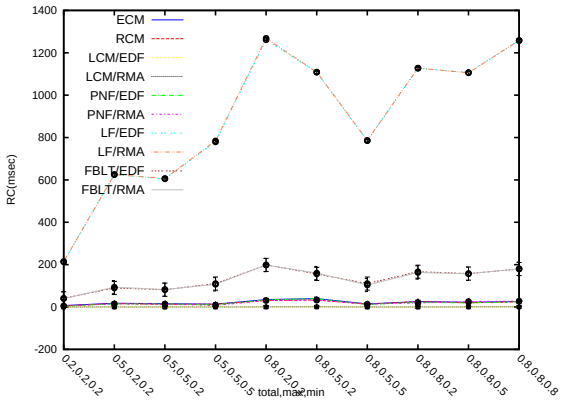


(b) 8 tasks

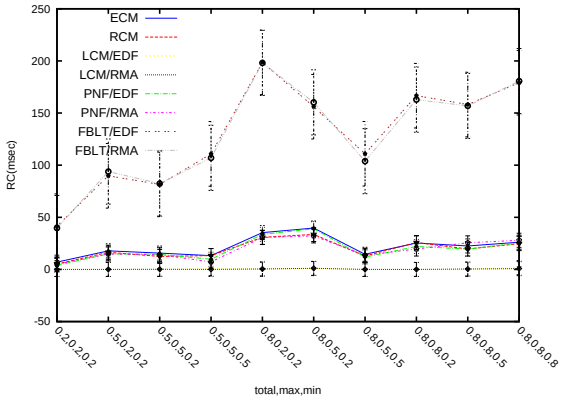


(c) 20 tasks

Figure 24: Average retry cost for 40 objects per transaction, 100% write operations for different values of total, maximum and minimum atomic section length under different CMs

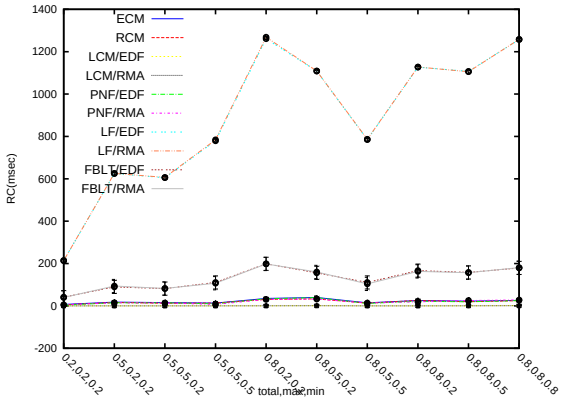


(a) ECM, RCM, LCM, PNF, FBLT, Lock-Free

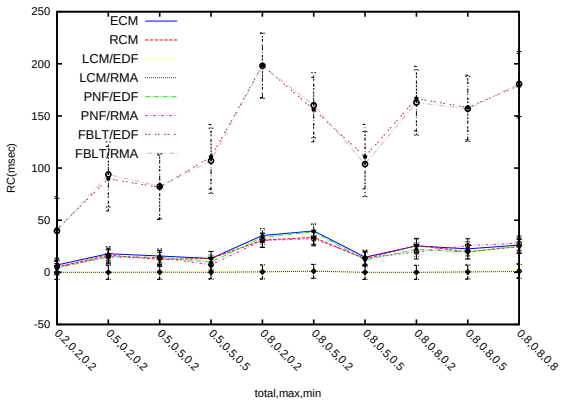


(b) ECM, RCM, LCM, PNF, FBLT

Figure 25: Avgerage retry cost (one object/transaction).



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



(b) ECM, RCM, LCM, PNF, FBLT

Figure 26: Avgerage retry cost (one object/transaction).

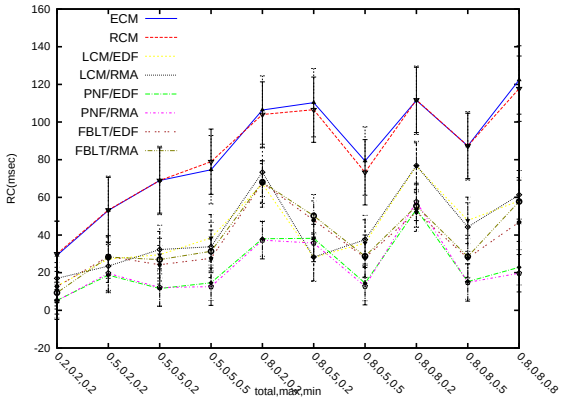


Figure 27: Avgerage retry cost (20 shared objects, 4 tasks).

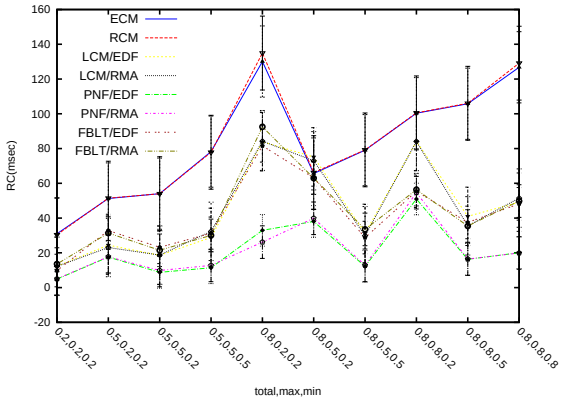


Figure 28: Avgerage retry cost (40 shared objects, 4 tasks).

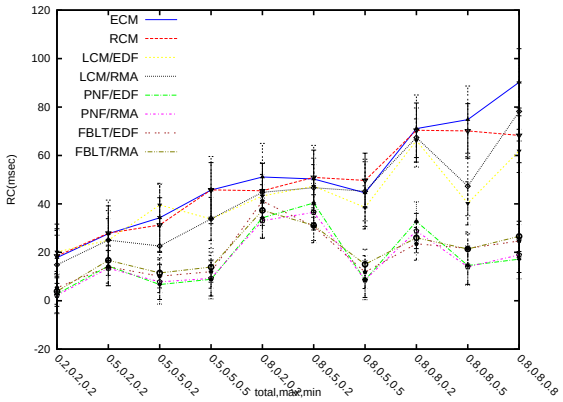


Figure 29: Avgerage retry cost (20 shared objects, 8 tasks).

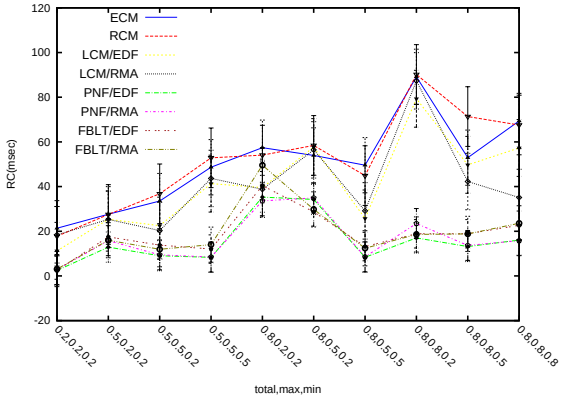


Figure 30: Avgerage retry cost (40 shared objects, 8 tasks).

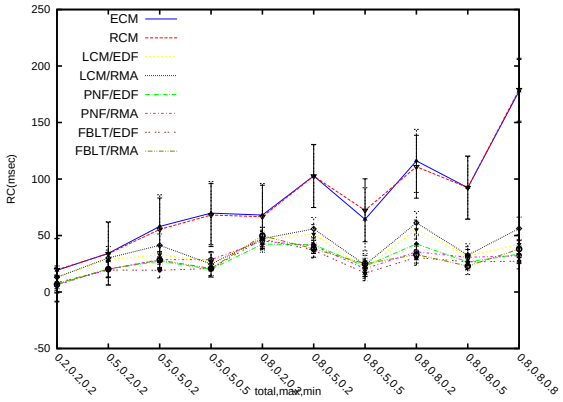


Figure 31: Avgerage retry cost (20 shared objects, 20 tasks).

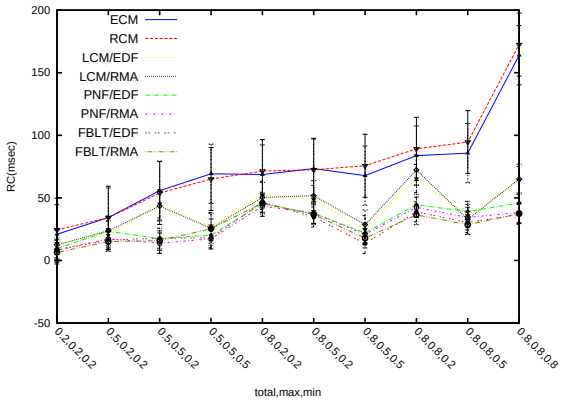


Figure 32: Avgerage retry cost (40 shared objects, 20 tasks).

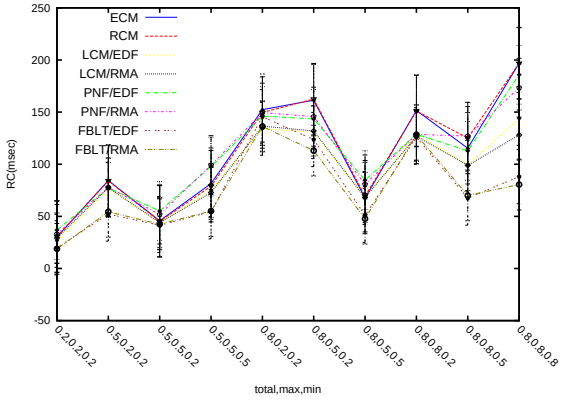


Figure 33: Avgerage retry cost (20 shared objects, 4 tasks).

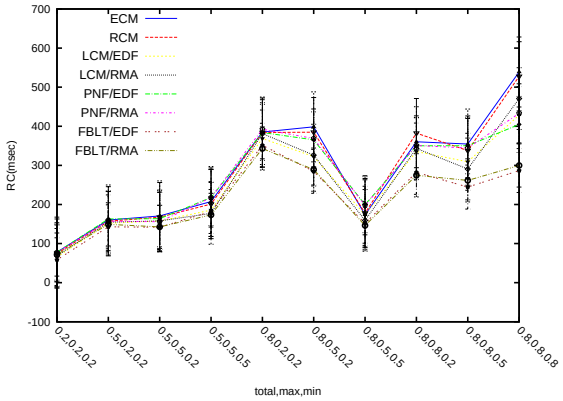


Figure 34: Avgerage retry cost (20 shared objects, 8 tasks).

A Notations

T_i	Task i.
T_i^j	j^{th} Instance (job) of task i.
$t(T_i)$	Minumum period of T_i.
$D(T_i)$	Relative deadline of any instance of T_i
$r(T_i^j)$	Release time of job T_i^j.
$d(T_i^j)$	Absolute deadline of T_i^j.
c_j	WCET of any instance of T_j
$p(T_i)$	Priority of T_i.
c_{ji}	The new WCET of T_j relative to the studied task T_i. This length changes according to the studied task.
θ	An object that can be accessed by any task.
θ_i	Set of objects accessed by T_i without repeatition.
$\gamma(\theta)$	Set of tasks that share object θ with T_i.
$s_i(\theta)$	Set of atomic sections of T_i that access object θ.
$s(\theta)$	Set of atomic sections in all tasks that share object θ.
$s_i^k(\theta)$	The k^{th} atomic section of T_i that accesses object θ.
s_i^k	The k^{th} atomic section of T_i regrdless of which object it accesses.
s_i	Set of atomic sections in T_i.
$len(s_i^k(\theta))$	Length of $s_i^k(\theta)$.
$len(s_i(\theta))$	Sum of all atomic sections in T_i that access object θ.
$s_{max}(\theta)$	The maximum atomic section in all tasks that share object θ.
$s_{i_{max}}(\theta)$	The maximum atomic section in T_i that accesses object θ.
$s_{max}^i(\theta)$	The maximum atomic section in all tasks with priority lower than or equal to that of T_i that share object θ.
$W_i^p(s_j^k(\theta))$	Contribution or workload by $s_j^k(\theta)$ in the retrial cost of $s_i^p(\theta)$.
$RC(T_i)$	Maximum transactional retrial cost of atomic sections in T_i due to conflict between atomic sections.
$RC(L(T_i))$	The same as $RC(T_i)$ but calculated only in a period of length L.
$RC(t(T_i))$	The same as $RC(T_i)$ over the whole $t(T_i)$.
$G_{ij}(L)$	Number of interferences made by T_j to T_i during period of length L.
$W_{ij}(L)$	Workload contributed by T_j to T_i during period of length L.
s_θ	A short resource.
l_θ	A long resource.
$g(s\_\theta)$	A group containing only short resource.
$g(l\_\theta)$	A group containing only long resources.
$R_k(g(s\_\theta))$	Request made by T_k to the $g(s\_\theta)$.
$R_k(g(l\_\theta))$	Request made by T_k to the $g(l\_\theta)$.
$\lvert R_k(g(s\_\theta))\rvert$	The size of the request and it equals the sum of all nested accesses to resources in $g(s\_\theta)$ made by T_k.
$\lvert R_k(g(l\_\theta))\rvert$	The sum of nested requests by T_k to the group containing l_θ, plus $max_{s\_\theta \in \theta_k}[\sum_{h=1,h\neq k}^{min(m,n)-1} \lvert R_h(g(s\_\theta))\rvert]$, if s_θ can be called inside l_θ.
$N_{i,l}$	The number of times T_i^j requests long resources.
$N_{i,s}$	The number of times T_i^j requests short resources.

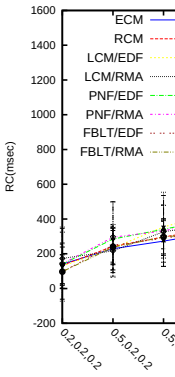
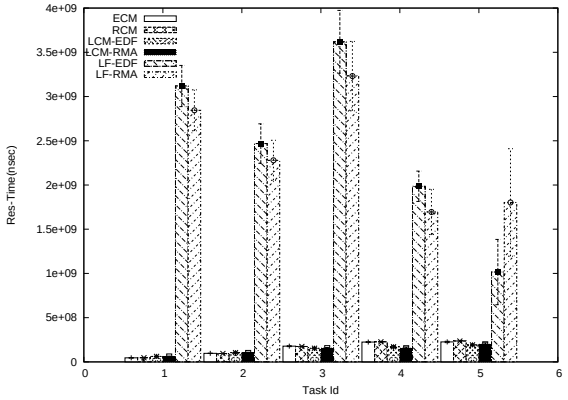


Figure 35: Avgerage

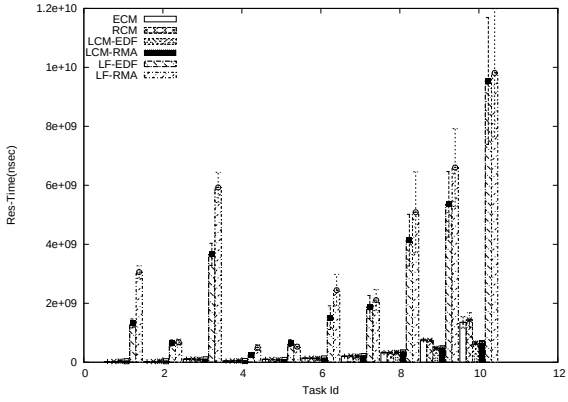
B Extended Results

-

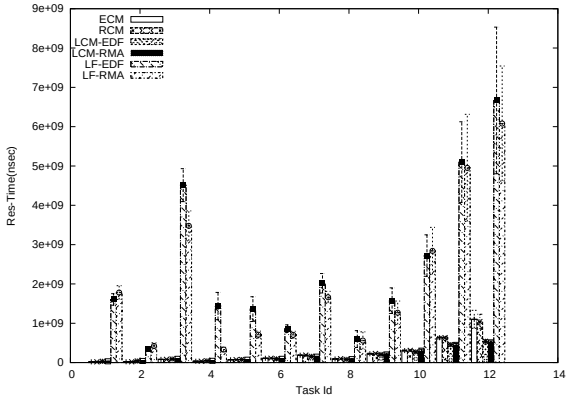
–



(a) Task set 1

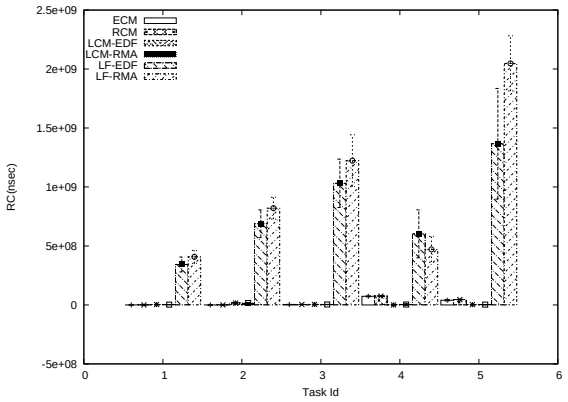


(b) Task set 2

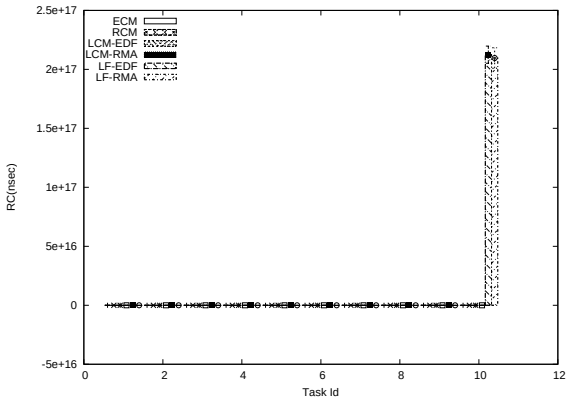


(c) Task set 3

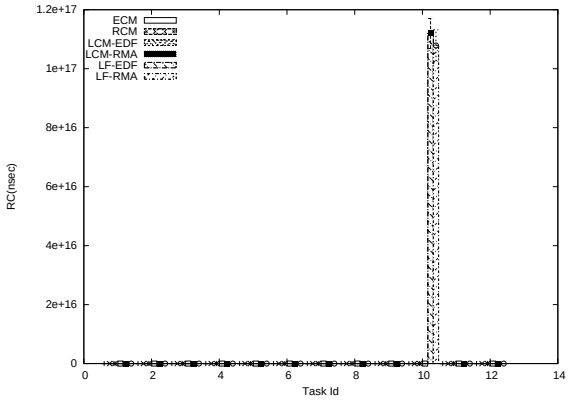
Figure 37: Task response times under LCM and competitor synchronization methods (0.5,0.2,0.2)



(a) Task set 1

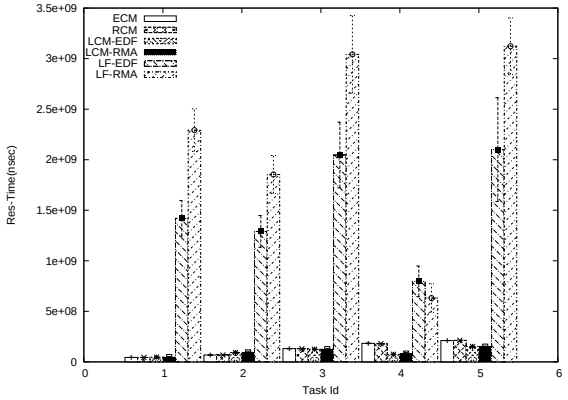


(b) Task set 2

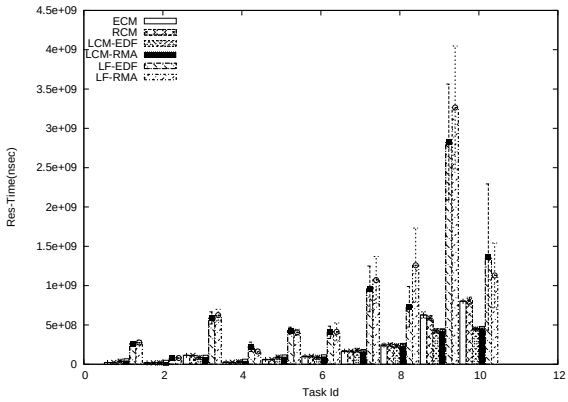


(c) Task set 3

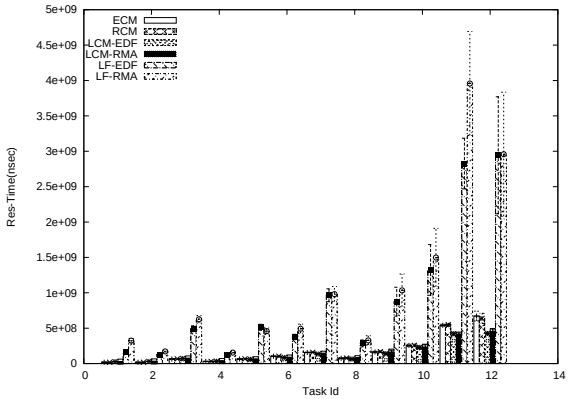
Figure 38: Task retry costs under LCM and competitor synchronization methods (0.5,0.5,0.2)



(a) Task set 1

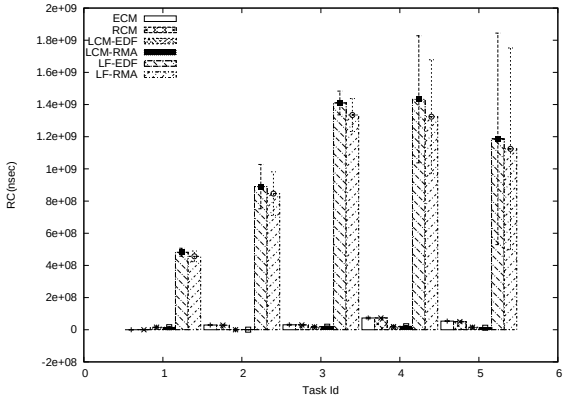


(b) Task set 2

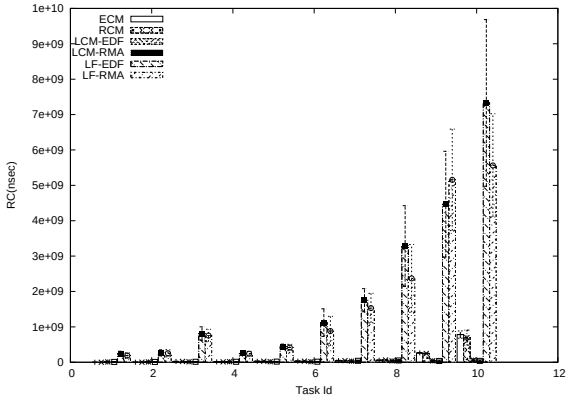


(c) Task set 3

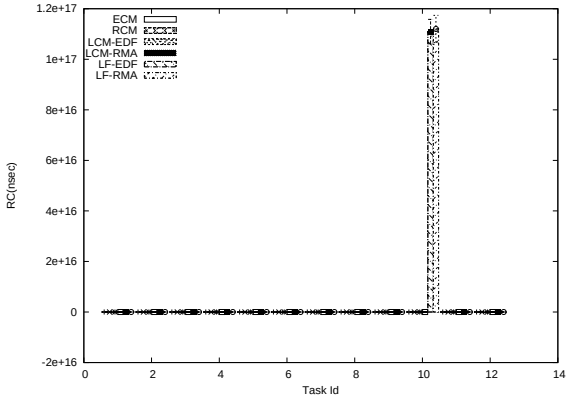
Figure 39: Task response times under LCM and competitor synchronization methods (0.5,0.5,0.2)



(a) Task set 1

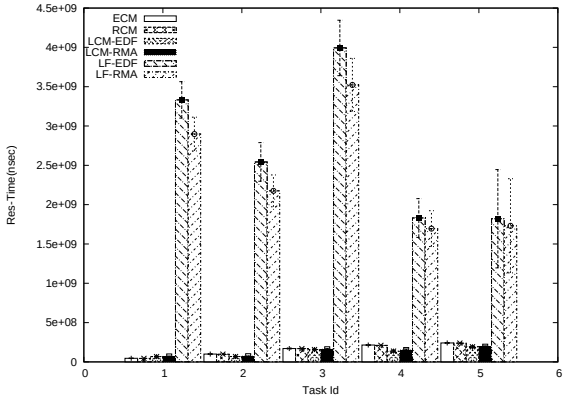


(b) Task set 2

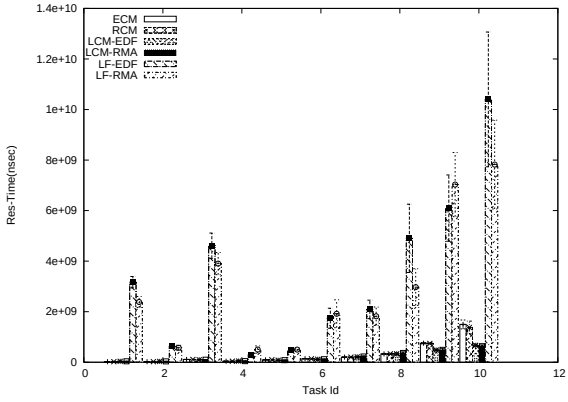


(c) Task set 3

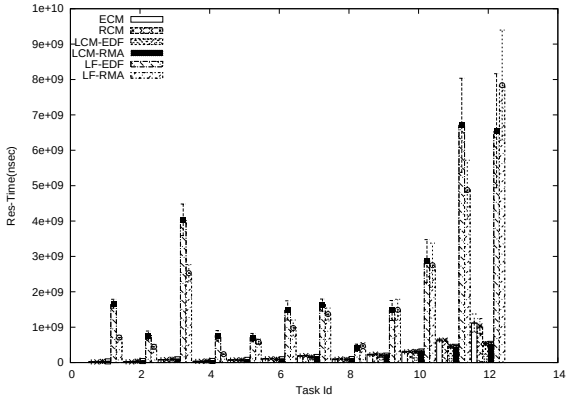
Figure 40: Task retry costs under LCM and competitor synchronization methods (0.5,0.5,0.5)



(a) Task set 1

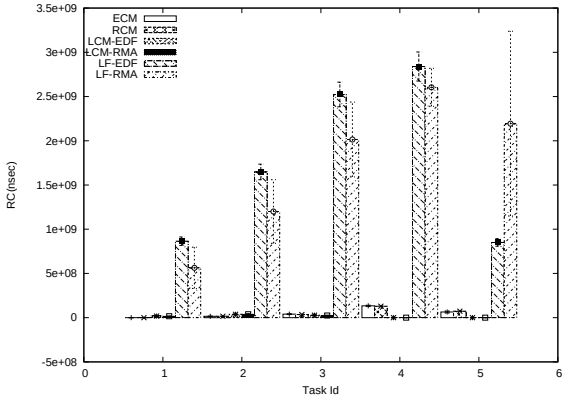


(b) Task set 2

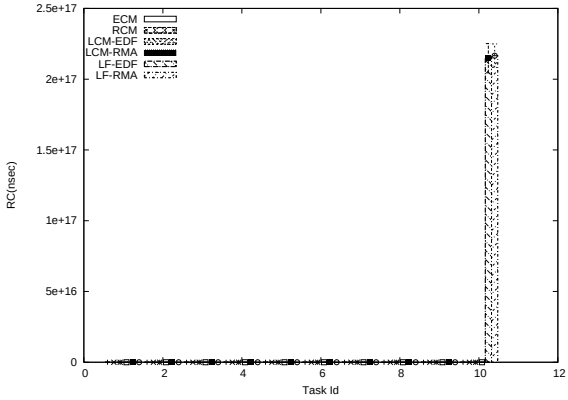


(c) Task set 3

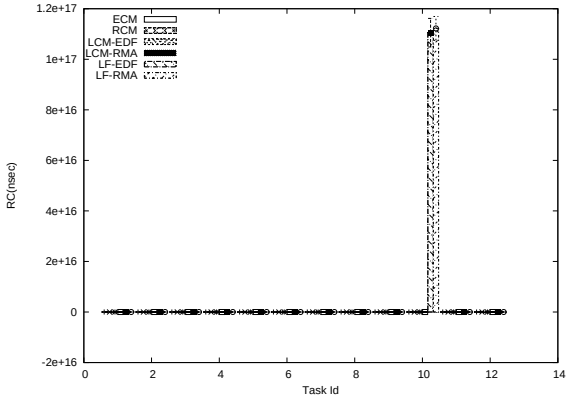
Figure 41: Task response times under LCM and competitor synchronization methods (0.5,0.5,0.5)



(a) Task set 1

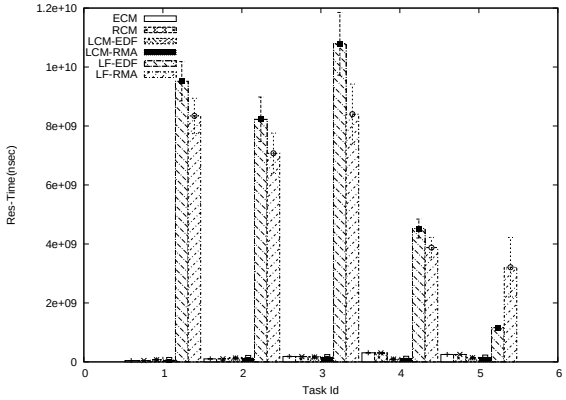


(b) Task set 2

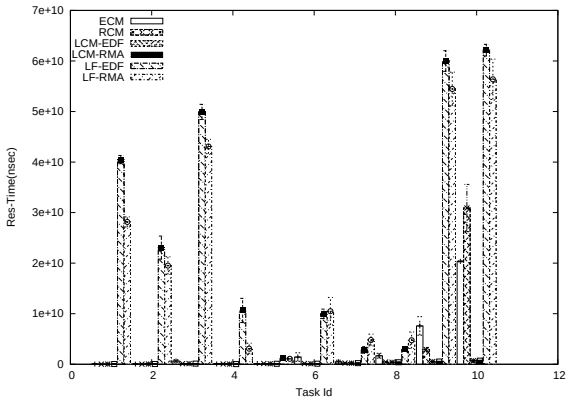


(c) Task set 3

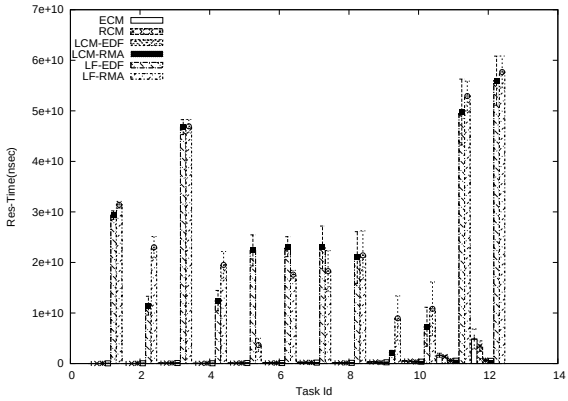
Figure 42: Task retry costs under LCM and competitor synchronization methods (0.8,0.2,0.2)



(a) Task set 1

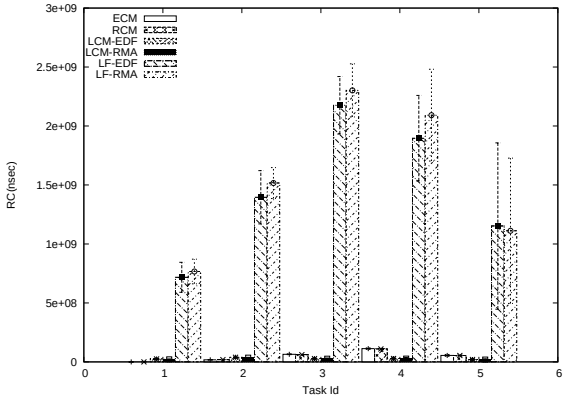


(b) Task set 2

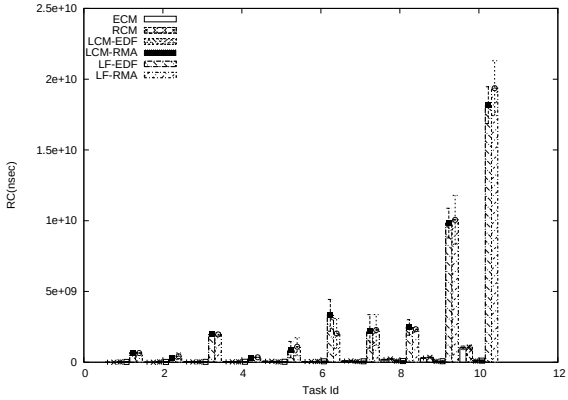


(c) Task set 3

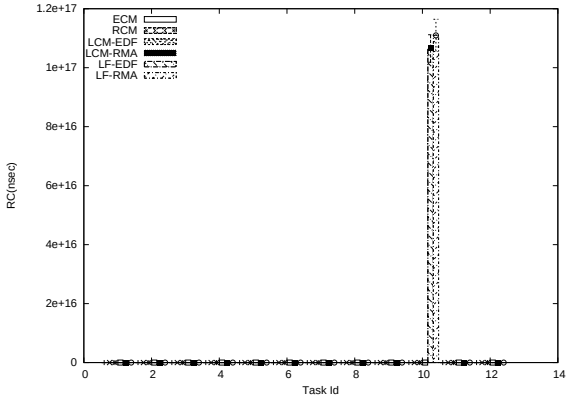
Figure 43: Task response times under LCM and competitor synchronization methods (0.8,0.2,0.2)



(a) Task set 1

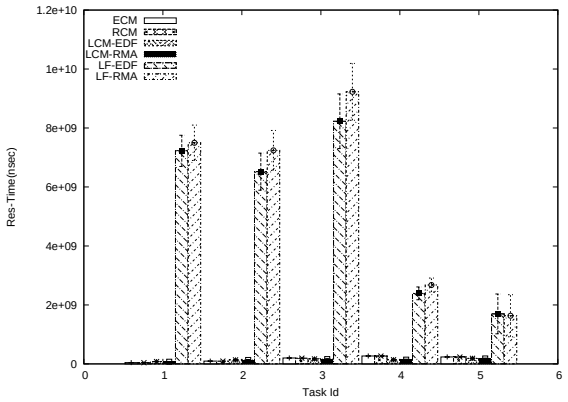


(b) Task set 2

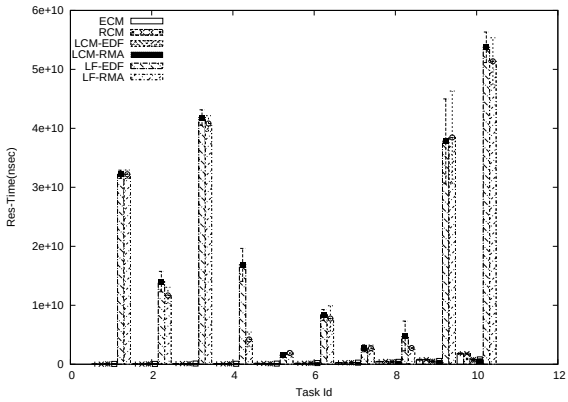


(c) Task set 3

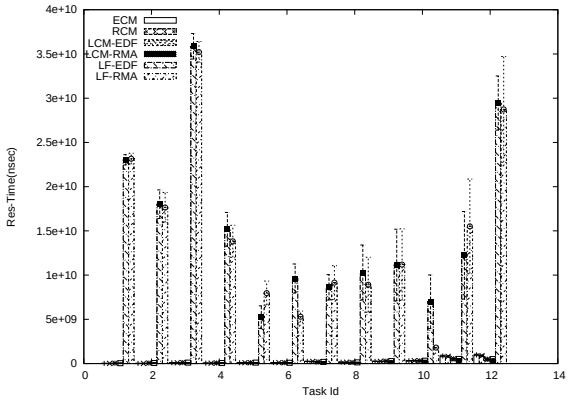
Figure 44: Task retry costs under LCM and competitor synchronization methods (0.8,0.5,0.2)



(a) Task set 1

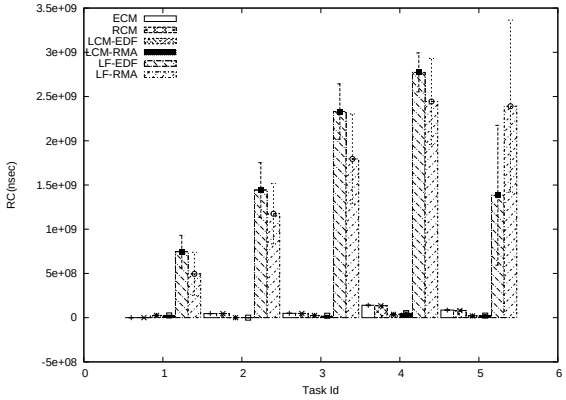


(b) Task set 2

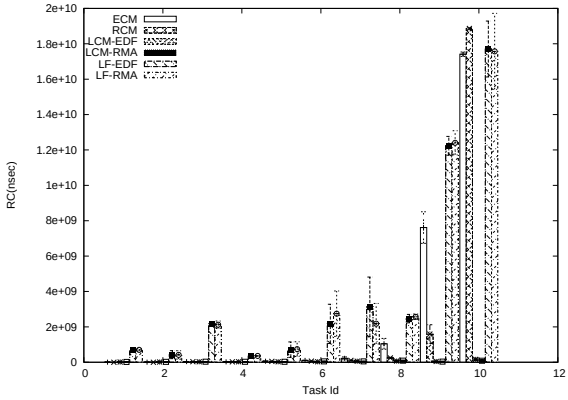


(c) Task set 3

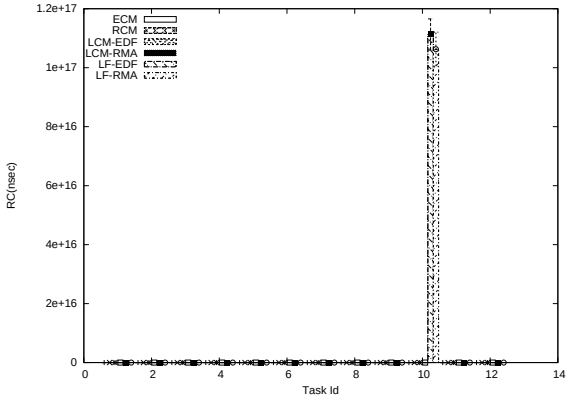
Figure 45: Task response times under LCM and competitor synchronization methods (0.8,0.5,0.2)



(a) Task set 1

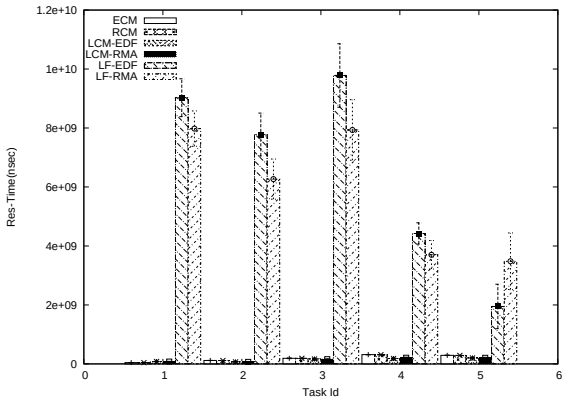


(b) Task set 2

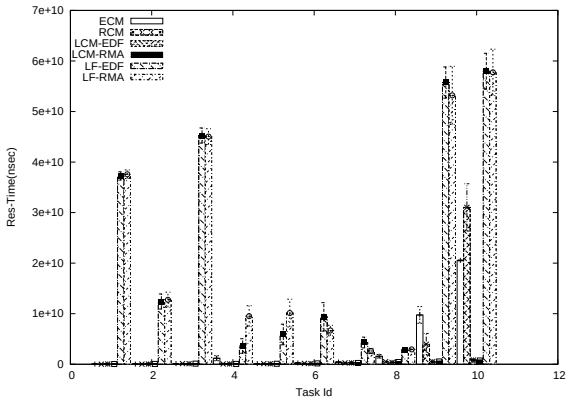


(c) Task set 3

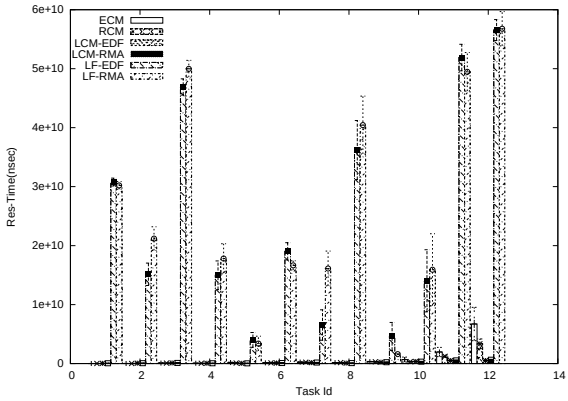
Figure 46: Task retry costs under LCM and competitor synchronization methods (0.8,0.5,0.5)



(a) Task set 1

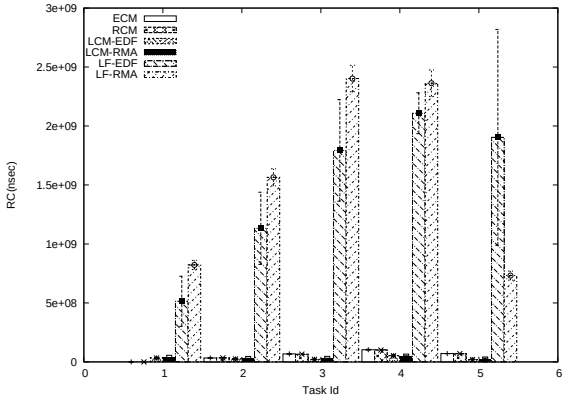


(b) Task set 2

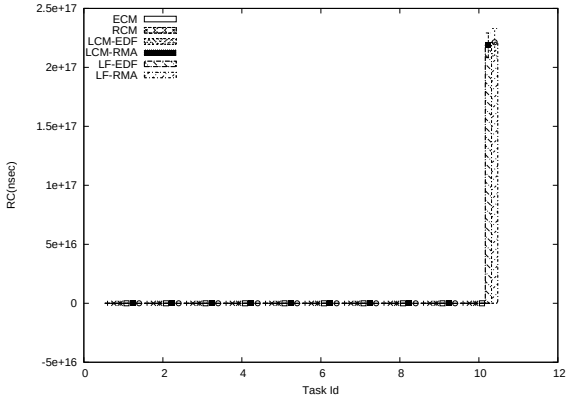


(c) Task set 3

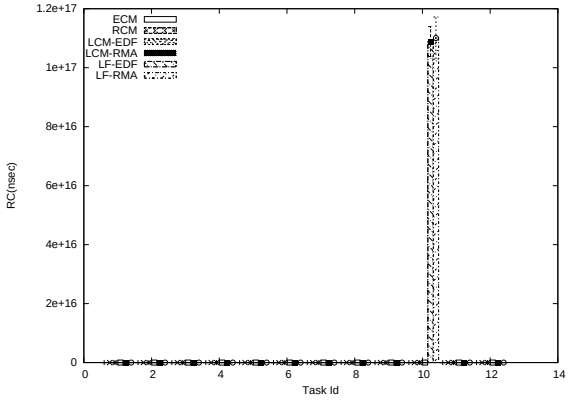
Figure 47: Task response times under LCM and competitor synchronization methods (0.8,0.5,0.5)



(a) Task set 1

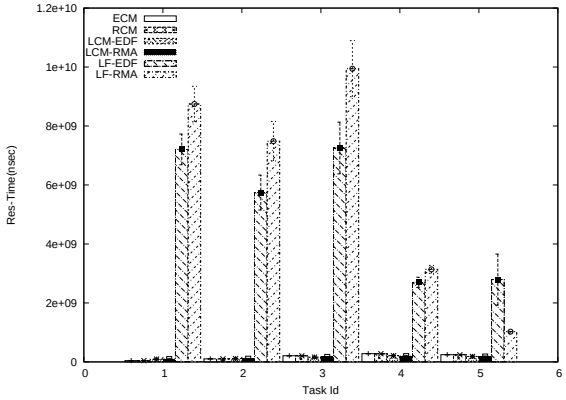


(b) Task set 2

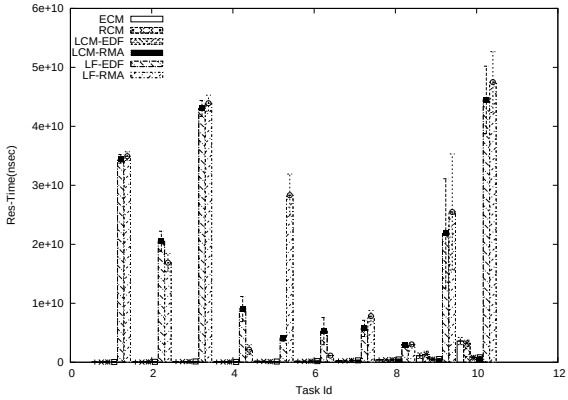


(c) Task set 3

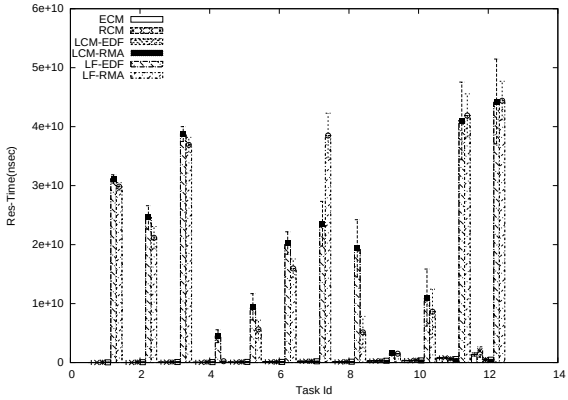
Figure 48: Task retry costs under LCM and competitor synchronization methods (0.8,0.8,0.2)



(a) Task set 1

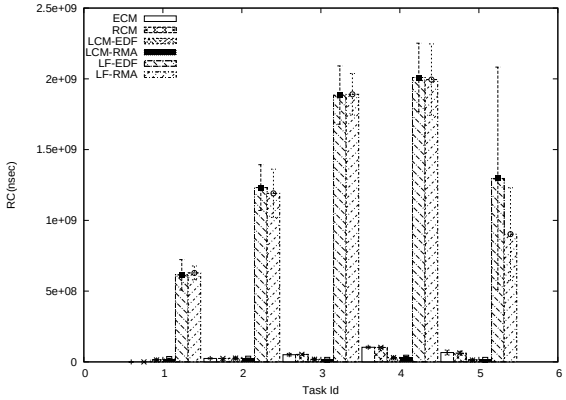


(b) Task set 2

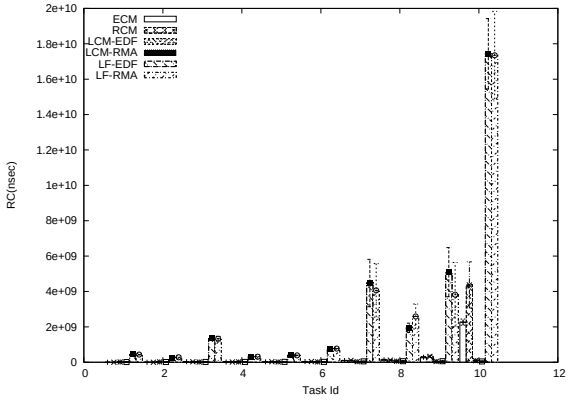


(c) Task set 3

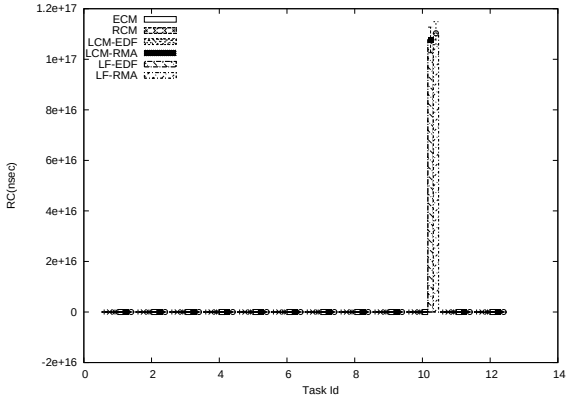
Figure 49: Task response times under LCM and competitor synchronization methods (0.8,0.8,0.2)



(a) Task set 1

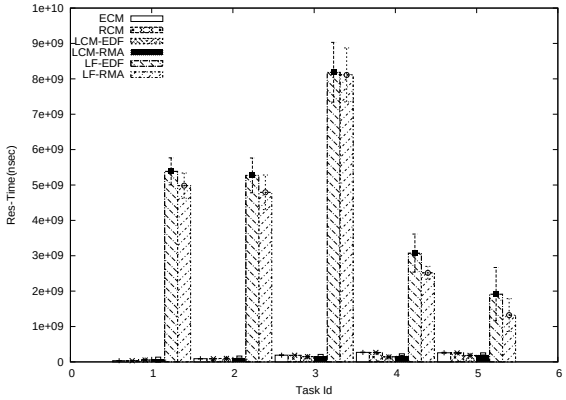


(b) Task set 2

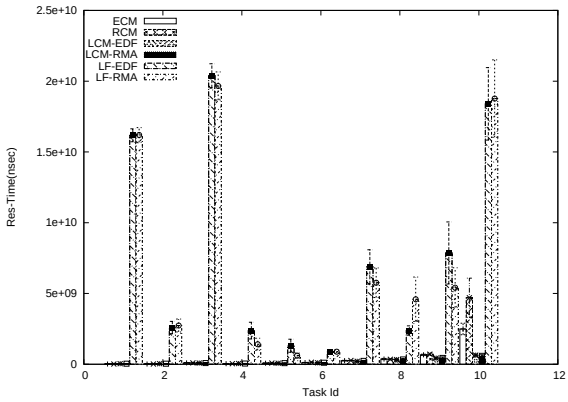


(c) Task set 3

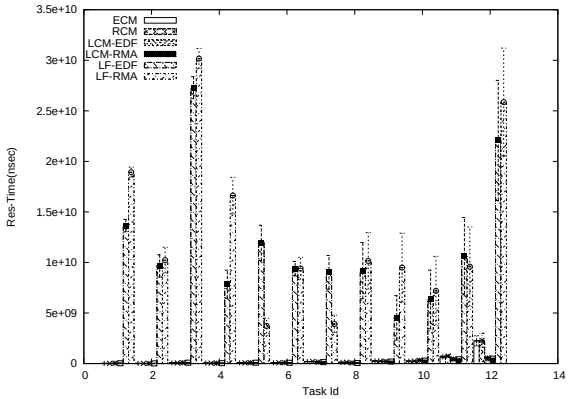
Figure 50: Task retry costs under LCM and competitor synchronization methods (0.8,0.8,0.5)



(a) Task set 1

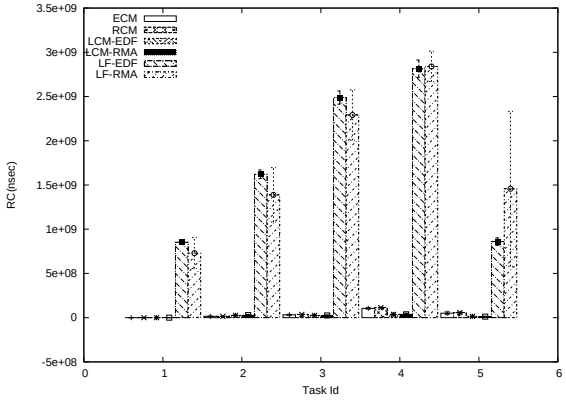


(b) Task set 2

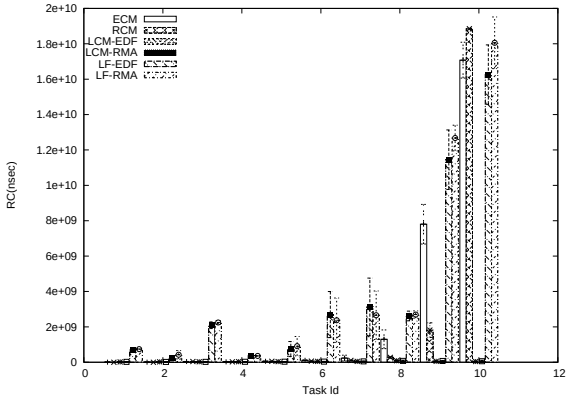


(c) Task set 3

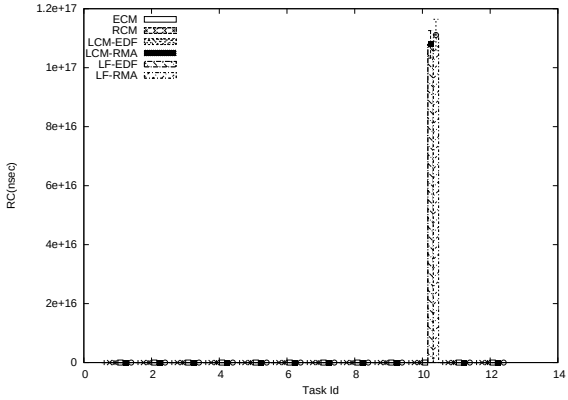
Figure 51: Task response times under LCM and competitor synchronization methods (0.8,0.8,0.5)



(a) Task set 1



(b) Task set 2



(c) Task set 3

Figure 52: Task retry costs under LCM and competitor synchronization methods (0.8,0.8,0.8)