STM Concurrency Control for Multicore Embedded Real-Time Software: Time Bounds and Tradeoffs*

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 [14]. 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) [12] 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 [17], and is composable [13]. Multiprocessor TM has been proposed in hardware, called HTM (e.g., [16]), and in software, called STM (e.g., [20]), with the usual tradeoffs: HTM provides strong atomicity [16], 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 [4] and OMLP [6] 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 conclude in Section 8.

*Approved for Public Release. Unlimited Distribution.

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 [7]. In contrast, STM is semantically simpler [14], and is often the only viable lock-free solution for complex data structures (e.g., red/black tree) [10] and nested critical sections [17]. (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 [2, 9, 10, 15, 18, 19].

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

[9] 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.

[18] 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.

[19] 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 [19] 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 [10], 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 [19], [10] 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 [10] are extended in [2], 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 $T_1, T_2, \ldots, T_n$. The k^{th} instance (or job) of a task T_i is denoted T_i^k. Each task T_i is specified by its worst case execution time (WCET) c_i, its minimum period $t(T_i)$ between any two consecutive instances, and its relative deadline $D(T_i)$, where $D(T_i) = t(T_i)$. Job T_i^j is released at time $r(T_i^j)$ and must finish no later than its absolute deadline $d(T_i^j) = r(T_i^j) + D(T_i)$. Under a fixed priority scheduler such as G-RMA, $p(T_i)$ determines T_i's (fixed) priority. Under a dynamic priority scheduler such as G-EDF, a job's priority is determined by its absolute deadline. A task T_j may interfere with task T_i for a number of times during a duration L, and this number is denoted as $G_{ij}(L)$. T_j's workload that interferes with T_i during L is denoted $W_{ij}(L)$.

Shared objects. A task may need to access (i.e., read, write) shared, in-memory objects while it is executing any of its atomic sections, which are synchronized using STM. The set of atomic sections of task T_i is denoted s_i. s_i^k is the k^{th} atomic section of T_i. Each object, θ, can be accessed by multiple tasks. The set of objects accessed by T_i is θ_i. The set of atomic sections used by T_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 T_i that accesses θ. $s_i^k(\theta)$ executes for a duration $len(s_i^k(\theta))$, which is the whole length of the atomic section (and not just the part that accesses θ). Thus, for two objects θ_1 and θ_2 that are accessed within the same atomic section of T_i, $len(s_i^k(\theta 1)) = len(s_i^k(\theta 2))$. If θ is shared by multiple tasks, then $s(\theta)$ is the set of atomic sections of all tasks accessing θ, and the set of tasks sharing θ with T_i is denoted $\gamma(\theta)$. Atomic sections are non-nested.

The maximum-length atomic section in T_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 or equal to that of T_i is $s^i_{max}(\theta)$.

STM retry cost. If two or more atomic sections conflict, the CM will commit one section and abort and retry the others, increasing the time to execute the aborted sections. The increased time that an atomic section $s_i^p(\theta)$ will take to execute due to interference with another section $s_j^k(\theta)$, is denoted $W_i^p(s_j^k(\theta))$.

The total time that a task T_i's atomic sections have to retry is denoted $RC(T_i)$. When this retry cost is calculated over the task period $t(T_i)$ or an interval L, it is denoted, respectively, as $RC(t(T_i))$ and $RC(L(T_i))$.

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, $s_i^k(\theta)$, is aborted and retried by a conflicting atomic section of T_j, $s_j^l(\theta)$, 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 [3] and is shown in Figure 1. Here, the deadline of an instance of T_j coincides with that of T_i, and T_j^1 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:

$$\begin{aligned} W_{ij}^{*}\left(t\left(T_{i}\right)\right) &= \left\lfloor \frac{t\left(T_{i}\right)}{t\left(T_{j}\right)} \right\rfloor .c_{j} + min\left(c_{j}, t\left(T_{i}\right) - \left\lfloor \frac{t\left(T_{i}\right)}{t\left(T_{j}\right)} \right\rfloor .t\left(T_{j}\right)\right) \\ &\leq \left\lceil \frac{t\left(T_{i}\right)}{t\left(T_{j}\right)} \right\rceil .c_{j} \end{aligned} \quad (1)$$

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:

$$\hat{W}_{ij}\left(L\right) = \left(\left\lceil \frac{L - c_{j}}{t\left(T_{j}\right)} \right\rceil + 1\right) .c_{j} \quad (2)$$

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

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

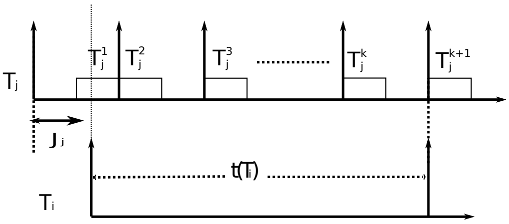


Figure 1: Max interference between two tasks under G-EDF

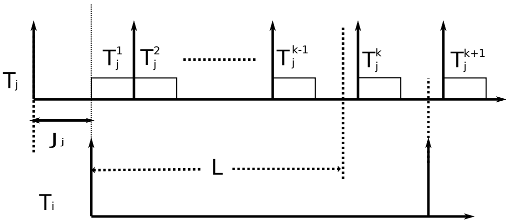


Figure 2: Max interference during part L of $t(T_i)$

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:*

$$RC(T_i) \le \sum_{\theta\in\theta_i}\Big(\Big(\sum_{T_j\in\gamma(\theta)}\Big(\left\lceil\frac{t(T_i)}{t(T_j)}\right\rceil\sum_{\forall s_j^l(\theta)} len\big(s_j^l(\theta) + s_{max}(\theta)\big)\Big)\Big) - s_{max}(\theta) + s_{i_{max}}(\theta)\Big) \quad (4)$$

PROOF. 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, $s_i^k(\theta)$, will experience its maximum delay when it is at its end of the atomic section, and the conflicting atomic section of T_j, $s_j^l(\theta)$, starts. The CM will retry $s_i^k(\theta)$.

Validation (i.e., conflict detection) in STM is usually done in two ways [16]: 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), $s_i^k(\theta)$ will retry for the same time duration, which is $len(s_j^l(\theta) + s_i^k(\theta))$. Then, $s_i^k(\theta)$ can commit successfully unless interferred by another conflicting atomic section, as shown in Figure 3.

In Figure 3(a), $s_j^l(\theta)$ 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 $s_j^l(\theta)$. When T_j finishes its atomic section, T_i executes its atomic section.

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,

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

Thus, the total contribution of all atomic sections of all other tasks that share objects with a task T_i to the retry cost of T_i during T_i's period $t(T_i)$ is:

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

Here, $\left\lceil\frac{t(T_i)}{t(T_j)}\right\rceil\sum_{\forall s_j^l(\theta)} len\left(s_j^l(\theta) + s_{max}(\theta)\right)$ 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:*

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

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

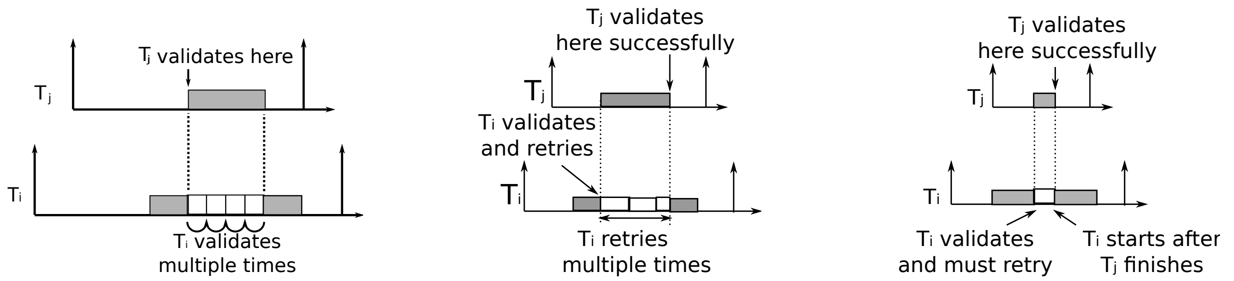
$$\Phi_2 = \Big(\sum_{T_j\in\gamma(\theta)}\Big(\left\lceil\frac{t(T_i)}{t(T_j)}\right\rceil\sum_{\forall s_j^l(\theta)} len\big(s_j^l(\theta) + s^*_{max}(\theta)\big)\Big)\Big) - \bar{s}_{max}(\theta) + s_{i_{max}}(\theta) \quad (8)$$

PROOF. (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:

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

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

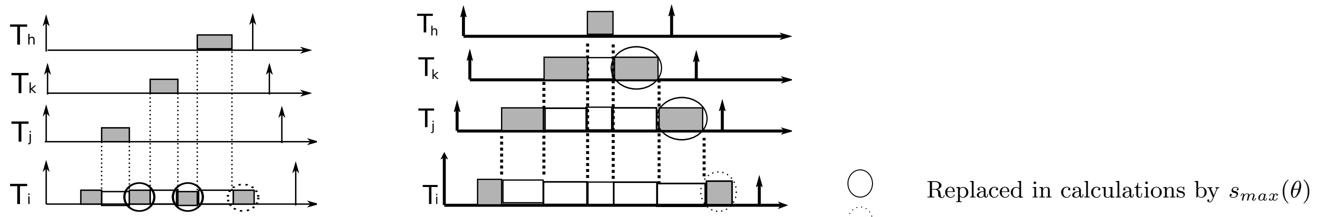


(a) Early validation

(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 $s_j^l(\theta)$



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

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

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

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.

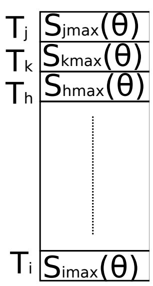


Figure 5: Values associated with $s^*_{max}(\theta)$

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 $Const_j = \left\lceil \frac{t(T_i)}{t(T_j)} \right\rceil \times \sum_{\forall s_j^l(\theta)} len(s_j^l(\theta))$. 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:

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

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 [3], as follows:

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

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:

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

(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 [3]) 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 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.

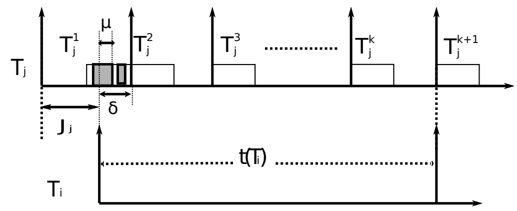


Figure 6: Atomic sections of T_j^1 contributing to $t(T_i)$

For simplicity, we use the following notations:

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

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:

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

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

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

Now, (7) becomes:

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

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

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

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.

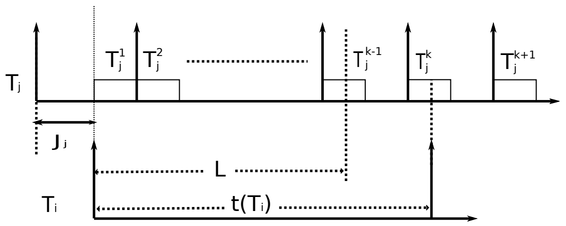


Figure 7: Max interference of T_j to T_i in G-RMA

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:*

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

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. 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 $s_{max}^j(\theta)$, 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^*). $\square$

5.3 Upper Bound on Response Time

The response time upper bound can be computed by Theorem 7 in [3] with some modification to include the effect of retry cost. Thus, this upper bound is given by:

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

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

The FMLP protocol [5] has been shown to be superior to other multiprocessor real-time locking protocols in terms of schedulability, and the global OMLP protocol [6] 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., [11,21]) 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 [6], no upper bounds are presented for these terms, except for $DB(T_i)$, as [6]'s main focus is on FMLP's suspension-based part. Also, the upper bound for $DB(T_i)$ in [6] 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:

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

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:

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

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:

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

6.2 Global OMLP

In [6], global FMLP has a maximum s-oblivious pi-blocking cost of $\Theta(n)$, whereas global OMLP [6], 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):

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

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 [8], 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 [8]'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. Equation (15) can be upper bounded as:

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

where $s_j^l(\theta)$, $s_{max}(\theta)$, $s_{i_{max}}(\theta)$, $s_{max}^*(\theta)$, and $\bar{s}_{max}(\theta)$ 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:

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

The loop retry cost is given by:

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

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 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:

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

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:

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

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 $t(T_j) \geq t(T_i) \therefore \left\lceil \frac{t(T_i)}{t(T_j)} \right\rceil =$

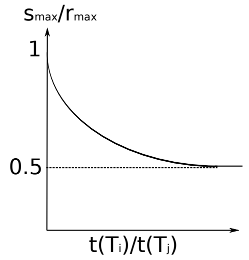


Figure 8: Effect of $\left\lceil \frac{t(T_i)}{t(T_j)} \right\rceil$ on $\frac{s_{max}}{r_{max}}$

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. $\square$

7.2 RCM versus Lock-Free

Claim 5. *For RCM's schedulability to be better or equal to that of [8]'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. Equation (16) is upper bounded by:

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

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:

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

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:

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

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

$$\begin{array}{c|cccc} & j & 1 & 2 & \cdots & n \\ i & & & & & \\ 1 & & 0 & 1 & \cdots & 1 \\ 2 & & 0 & 0 & \ddots & \vdots \\ \vdots & & \vdots & \vdots & \ddots & 1 \\ n & & 0 & 0 & \cdots & 0 \end{array} \qquad \begin{array}{cccccc} & l & 1 & 2 & \cdots & n \\ k & & & & & \\ 1 & & 0 & 0 & \cdots & 0 \\ 2 & & 1 & 0 & & \vdots \\ \vdots & & \vdots & \ddots & \ddots & 0 \\ n & & 1 & \cdots & 1 & 0 \end{array}$$

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,

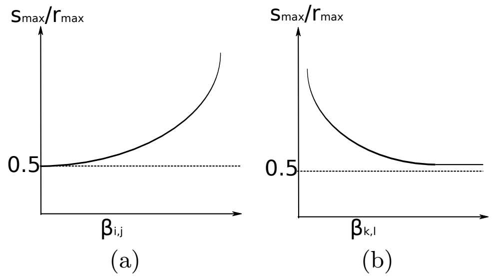


Figure 10: Change of s_{max}/r_{max}: a) $\frac{s_{max}}{r_{max}}$ versus $\beta_{i,j}$ and b) $\frac{s_{max}}{r_{max}}$ versus $\beta_{k,l}$

because lower priority tasks do not affect higher priority ones, so s_{max} is allowed to be much greater than r_{max}.

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. 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:

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

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

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:

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

For RCM, the ratio is:

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

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. CONCLUSIONS

Under both ECM and RCM, a task incurs $2.s_{max}$ retry cost for each of its atomic section due to a conflict with another task's atomic section. Retries under RCM and lock-free are affected by a larger number of conflicting task instances than under ECM. While task retries under ECM and lock-free are affected by all other tasks, retries under RCM are affected only by higher priority tasks.

STM and lock-free have similar parameters that affect their retry costs—i.e., the number of conflicting jobs and how many times they access shared objects with a task T_i, while FMLP and OMLP are affected by the total number of tasks and the number of requests made by T_i. This is because, requests in FMLP and OMLP are arranged in a queue, and the order of the requests in the queue does not change (except for the case of OMLP's priority queue). Thus, STM and lock-free can be compared in terms of parameters affecting their retry costs, while STM and locking protocols can only be compared asymptotically.

The s_{max}/r_{max} ratio determines whether STM is better or as good as lock-free. For ECM, this ratio cannot exceed 1, and it can be 1/2 for higher number of conflicting tasks. For RCM, for the common case, s_{max} must be 1/2 of r_{max}, and in some cases, s_{max} can be larger than r_{max} by many orders of magnitude. For locking protocols, a comparative schedulability of STM to that of FMLP and OMLP depends on the number of tasks and processors.

Thus, no synchronization method fits all applications from a timing standpoint; our results shed light on which method to select under what application conditions. From a programmability standpoint, however, STM is semantically as simple as coarse-grain locks.

Our work has only further scratched the surface of real-time STM. The questions that we ask (see Section 1) are fundamentally analytical in nature, and hence, our results are analytical. However, significant insights can be gained by experimental work on a broad range of embedded software, which is outside our work's scope. For example, what are the typical range of values for the different parameters that affect the retry cost (and hence the response time)? How tight is our retry and response time bounds in practice? Can real-time CMs be designed for other multiprocessor real-time schedulers (e.g., partitioned, semi-partitioned), and those that dynamically improve application timeliness behavior? These are important directions for further work.

9. REFERENCES

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

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

[3] M. Bertogna and M. Cirinei. Response-time analysis for globally scheduled symmetric multiprocessor platforms. In *RTSS*, pages 149–160, 2007.

[4] A. Block, H. Leontyev, B. B. Brandenburg, and J. H. Anderson. A flexible real-time locking protocol for multiprocessors. In *RTCSA*, pages 47–56, 2007.

[5] B. Brandenburg and J. Anderson. A comparison of the M-PCP, D-PCP, and FMLP on LITMUS–RT. In *OPODIS*, pages 105–124, 2008.

[6] B. B. Brandenburg and J. H. Anderson. Optimality results for multiprocessor real-time locking. In *RTSS*, pages 49–60, 2010.

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

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

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

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

[11] J. Goossens, S. Funk, and S. Baruah. Priority-driven scheduling of periodic task systems on multiprocessors. *Real-Time Syst.*, 25:187–205, September 2003.

[12] R. Guerraoui, M. Herlihy, and B. Pochon. Toward a theory of transactional contention managers. In *PODC*, pages 258–264, 2005.

[13] T. Harris, S. Marlow, et al. Composable memory transactions. In *PPoPP*, pages 48–60, 2005.

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

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

[16] A. McDonald. *Architectures for Transactional Memory*. PhD thesis, Stanford University, June 2009.

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

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

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

[20] N. Shavit and D. Touitou. Software transactional memory. In *PODC*, pages 204–213, 1995.

[21] A. Srinivasan and S. Baruah. Deadline-based scheduling of periodic task systems on multiprocessors. *Information Processing Letters*, 84(2):93–98, 2002.