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

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

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

ABSTRACT

We consider software transactional memory (STM) for 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-free synchronization.

Categories and Subject Descriptors

C.3 [**Special-purpose and application-based systems**]: Real-time and embedded systems

Keywords

Software transactional memory, response time, real-time

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 concurrent programming is the threads abstraction, and the de facto synchronization abstraction is locks. Lock-based concurrency control has significant programmability, scalability, and compositionality challenges [10]. 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) [8] 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 to highly concurrent fine-grained locking and lock-free approaches [13], and is composable [9]. Multiprocessor TM has been proposed in hardware, called HTM (e.g., [12]), and in software, called STM (e.g., [16]), with the usual tradeoffs: HTM provides strong atomicity [12], 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, such as global EDF (G-EDF) and global RMA (G-RMA)? Since lock-free synchronization overcomes some of the performance challenges (but not scalability and compositionality challenges) of locks, how does it compare with STM? i.e., are there upper or lower bounds for transaction lengths below or above which STM is superior to 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 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 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 compare STM with lock-free approaches in Section 6. We conclude in Section 7.

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 [4]. In contrast, STM is semantically simpler [10], and is often the only viable lock-free solution for complex data structures (e.g., red/black tree) [7] and nested critical sections [13].

STM concurrency control for real-time systems has been previously studied in [2, 6, 7, 11, 14, 15].

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

[6] 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.

[14] 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 lock-free synchronization. In contrast, we establish such bounds and tradeoffs.

[15] proposes real-time HTM. In contrast, we propose real-time STM, and therefore do not need transactional support in hardware. In addition, the retry bound developed in [15] 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 [7], which upper bounds retries and response times for EDF CM with G-EDF, and identifies the tradeoffs against locking and lock-free protocols. Similar to [15], [7] 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 [7] are extended in [2], which presents three real-time CM designs. However, no retry bounds or schedulability analysis are presented for those CMs in [2].

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, τ_i^j's priority, p_i^j, is determined by its absolute deadline. A task τ_j may interfere with task τ_i for a number of times during a duration L, and this number is denoted as $G_{ij}(L)$. τ_j's workload that interferes with τ_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 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. The set of atomic sections used by τ_i to access θ is $s_i(\theta)$, and the sum of the lengths of those atomic sections is $len(s_i(\theta))$. $s_i^k(\theta)$ is the k^{th} atomic section of τ_i that accesses θ. $s_i^k(\theta)$ executes for a duration $len(s_i^k(\theta))$. 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 τ_i is denoted $\gamma_i(\theta)$. Atomic sections are non-nested. Each atomic section is assumed to access only one object; this allows a head-to-head comparison with lock-free synchronization [5]. (Allowing multiple object access per transaction 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 $s_{max}^i(\theta)$.

STM retry cost. If two or more atomic sections conflict, the CM will commit one section and abort and retry the others, increasing the time to execute the aborted sections. The increased time that an atomic section $s_i^p(\theta)$ will take to execute due to interference with another section $s_j^k(\theta)$, is denoted $W_i^p(s_j^k(\theta))$. The total time that a task τ_i's atomic sections have to retry is denoted $RC(\tau_i)$. When this retry cost is calculated over the task period T_i or an interval L, it is denoted, respectively, as $RC(T_i)$ and $RC(L)$.

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 τ_i's atomic sections are interfered by other tasks that share the same objects with τ_i. The EDF CM will abort and retry an atomic section of τ_i, $s_i^k(\theta)$ due to a conflicting atomic section of τ_j, $s_j^l(\theta)$, if the absolute deadline of τ_j is less than or equal to the absolute deadline of τ_i. Hereafter, 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 τ_j interferes with τ_i is given in [3] and is illustrated in Figure 1. Here, the deadline of an instance of τ_j coincides with that of τ_i, and τ_j^1 is delayed by its maximum jitter J_j, which causes all or part of τ_j^1's execution to overlap within T_i. From Figure 1, it is seen that τ_j's maximum workload that interferes with τ_i (when there are no atomic sections) in T_i is:

$$\begin{aligned} W_{ij}(T_i) &\leq \left\lfloor \frac{T_i}{T_j} \right\rfloor c_j + min\left(c_j, T_i - \left\lfloor \frac{T_i}{T_j} \right\rfloor T_j\right) \\ &\leq \left\lceil \frac{T_i}{T_j} \right\rceil c_j \end{aligned} \quad (1)$$

For an interval $L < T_i$, the worst case pattern of interference is shown in Figure 2. Here, τ_j^1 contributes by all its c_j, and d_j^{k-1} does not have to coincide with L, as τ_j^{k-1} has a

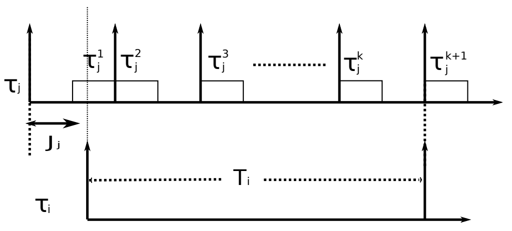


Figure 1: Maximum interference between two tasks, running on different processors, under G-EDF

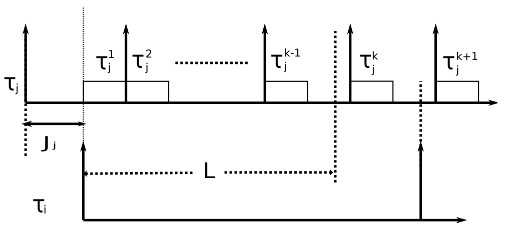


Figure 2: Maximum interference during an interval L of T_i

higher priority than that of τ_i. The workload of τ_j is:

$$W_{ij}\left(L\right) \leq \left(\left\lceil \frac{L - c_j}{T_j} \right\rceil + 1\right) c_j \tag{2}$$

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

$$W_{ij}\left(R\right) = min\left(W_{ij}\left(R\right), W_{ij}\left(T_i\right)\right) \tag{3}$$

where $W_{ij}(R)$ is calculated by (2) if $R < T_i$, otherwise, it is calculated by (1).

4.1 Retry Cost of Atomic Sections

CLAIM 1. *Under ECM, a task T_i's maximum retry cost during T_i is upper bounded by:*

$$RC\left(T_i\right) \leq \sum_{\theta\in\theta_i}\left(\left(\sum_{\tau_j\in\gamma_i(\theta)}\left(\left\lceil\frac{T_i}{T_j}\right\rceil\sum_{\forall s_j^l(\theta)} len\big(s_j^l(\theta) + s_{max}(\theta)\big)\right)\right) - s_{max}(\theta) + s_{i_{max}}(\theta)\right) \tag{4}$$

PROOF. Consider two instances τ_i^a and τ_j^b, where $d_j^b \leq d_i^a$. When a shared object conflict occurs, the EDF CM will commit the atomic section of τ_j^b while aborting and retrying that of τ_i^a. Thus, an atomic section of τ_i^a, $s_i^k(\theta)$, will experience its maximum delay when it is at the end of its atomic section, and the conflicting atomic section of τ_j^b, $s_j^l(\theta)$, starts, because the whole $s_i^k(\theta)$ will be repeated after $s_j^l(\theta)$.

Validation (i.e., conflict detection) in STM is usually done in two ways [12]: a) eager (pessimistic), in which conflicts are detected at access time, and 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 it is 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 τ_i^a retries multiple times (because at the start of each retry, τ_i^a validates) during the execution of $s_j^l(\theta)$. When τ_j^b finishes its atomic section, τ_i^a executes its atomic section.

In Figure 3(b), τ_i^a validates at its end (due to lazy validation), and detects a conflict with τ_j^b. Thus, it retries, and because its atomic section length is shorter than that of τ_j^b, it validates again within the execution interval of $s_j^l(\theta)$. However, the EDF CM retries it again. This process continues until τ_j^b finishes its atomic section. If τ_i^a's atomic section length is longer than that of τ_j^b, τ_i^a would have incurred the same retry time, because τ_j^b will validate when τ_i^a is retrying, and τ_i^a 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 τ_i^a or interfere with each other and τ_i^a (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 of 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 shares the same object with τ_i^a) 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,

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

Thus, the total contribution of all atomic sections of all other tasks that share objects with a task τ_i to the retry cost of τ_i during T_i is:

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

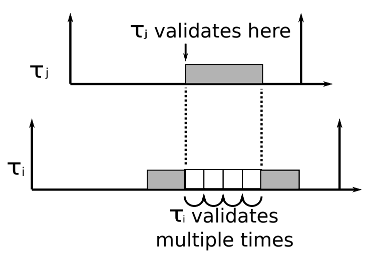
Here, $\left\lceil\frac{T_i}{T_j}\right\rceil\sum_{\forall s_j^l(\theta)} len\left(s_j^l\left(\theta\right) + s_{max}\left(\theta\right)\right)$ is the contribution of all instances of τ_j during T_i. This contribution is added to all tasks. The last atomic section to execute is $s_i^p(\theta)$ (τ_i's 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:*

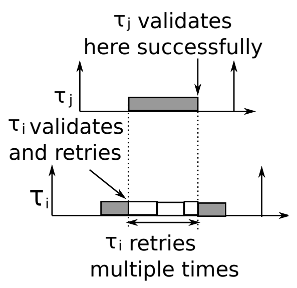
$$RC(T_i) \leq \sum_{\theta\in\theta_i} min(\Phi_1, \Phi_2) \tag{7}$$

where Φ_1 is calculated by (4) for one object θ (not the sum of objects in θ_i), and

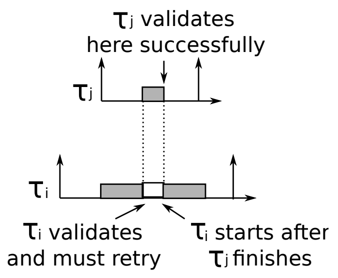
$$\Phi_2 = \left(\sum_{\tau_j\in\gamma_i(\theta)}\left(\left\lceil\frac{T_i}{T_j}\right\rceil\sum_{\forall s_j^l(\theta)} len\big(s_j^l(\theta) + s_{max}^*(\theta)\big)\right)\right) - \bar{s}_{max}(\theta) + s_{i_{max}}(\theta) \tag{8}$$



(a) Early validation

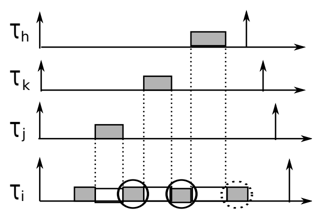


(b) Lazy validation with $len(s_i^k(\theta)) \leq len(s_j^l(\theta))$

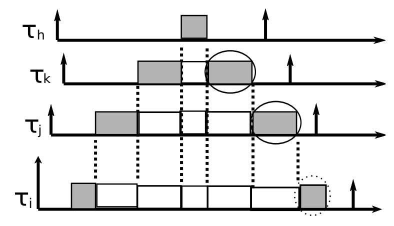


(c) Lazy validation with $len(s_i^k(\theta)) > len(s_j^l(\theta))$

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 $s_i^p(\theta)$

○ 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

where s^*_{max} is the maximum atomic section between all tasks, except τ_j, accessing θ. $\bar{s}_{max}(\theta)$ is the second maximum atomic section between all tasks accessing θ.

Proof. (4) can be modified by noting that a task τ_j's atomic section may conflict with those of other tasks, but not with τ_j. This is because, tasks are assumed to arrive sporadically, and each instance finishes before the next begins. Thus, (5) becomes:

$$W_i^p\left(s_j^k(\theta)\right) \leq len\left(s_j^k(\theta) + s^*_{max}(\theta)\right) \tag{9}$$

To see why $\bar{s}_{max}(\theta)$ is used instead of $s_{max}(\theta)$, 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. According to (9), all tasks but τ_j will choose $s_{j_{max}}(\theta)$ as the value of $s^*_{max}(\theta)$. But when τ_j is the one whose contribution is studied, it will choose $s_{k_{max}}(\theta)$, as it is the maximum one not associated with τ_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 τ_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 τ_i, which still lies between the two largest values.

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 τ_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.

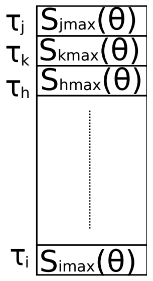


Figure 5: Values associated with $s^*_{max}(\theta)$

Since it is not known a-priori whether Φ_1 will be smaller than Φ_2 for a specific θ, the minimum of Φ_1 and Φ_2 is taken as 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 τ_i, the term $RC(T_i)$ must be added to the workload of other tasks during the non-atomic execution of τ_i. But this requires modification of the WCET of each task as follows. c_j of each interfering task τ_j should be inflated to accommodate the interference of each task τ_k, $k \neq j, i$. Meanwhile, atomic regions that access shared objects between τ_j and τ_i should not be considered in the inflation cost, because they have already been calculated in τ_i's retry cost. Thus, τ_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 τ_j relative to τ_i; the sum

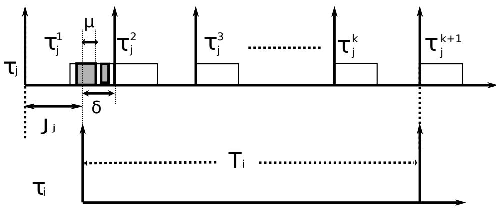


Figure 6: Atomic sections of job τ_j^1 contributing to period T_i

of lengths of all atomic sections in τ_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 τ_i and τ_j. The calculated WCET is relative to task τ_i, as it changes from task to task. The upper bound on the response time of τ_i, denoted R_i^{up}, can be calculated iteratively, using a modification of Theorem 6 in [3], as follows:

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

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

$W_{ij}(R_i^{up})$ is calculated by (3), and $W_{ij}(T_i)$ is calculated by (1), with c_j replaced by c_{ji}, and changing (2) as:

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

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

Case 1. τ_j^1 (shown in Figure 2) contributes by c_{ji}. Thus, other instances of τ_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. τ_j^1 contributes by its c_j, but the sum of the shared atomic section lengths between τ_i and τ_j should be subtracted from the contribution of τ_j^1, 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 τ_j^1. The maximum is chosen from the two terms in (12), because they differ in the contribution of their τ_j^1s, and the number of instances after that.

4.2.1 Tighter Upper Bound

To tighten τ_i's response time upper bound, $RC(\tau_i)$ needs to be calculated recursively over duration R_i^{up}, and not directly over T_i, as done in (11). So, (7) must be changed to include the modified number of interfering instances. And if R_i^{up} still extends to T_i, a situation like that shown in Figure 6 can happen.

To counter the situation in Figure 6, atomic sections of τ_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 τ_j^1 has been delayed by its maximum jitter. Hence, no more atomic sections can interfere during the duration $[d_j^1-\delta, d_j^1]$.

For simplicity, we use the following notations:

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

Here, $s_j^{l^*}(\theta)$ is the part of $s_j^l(\theta)$ that is included in the interval δ. Thus, if $s_j^l(\theta)$ is partially included in δ, it contributes by its included length μ.

Now, (7) can be modified as:

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

Now, we compute $RC(L)$, where L does not extend to the last instance of τ_j. Let:

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

Now, (7) becomes:

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

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

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

where $RC(R_i^{up})$ is calculated by (14) if R_i^{up} does not extend to the last interfering instance of τ_j; otherwise, it is calculated by (13). The final upper bound on τ_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 τ_i will be interfered by those tasks with priorities higher than τ_i (i.e., $p_j > p_i$). Upon a conflict, the RMA CM will commit the transaction that belongs to the higher priority task. Hereafter, 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 τ_j to a task τ_i under G-RMA. As τ_j is of higher priority than τ_i, τ_j^k will interfere with τ_i even if it is not totally included in T_i. Unlike the G-EDF case shown in Figure 6, where only the δ part of τ_j^1 is considered, in G-RMA, τ_j^k can contribute by the whole c_j, and all atomic sections contained in τ_j^k must be considered. This is because, in G-EDF, the worst-case pattern releases τ_i^a before d_j^1 by δ time units, and τ_i^a cannot be interfered before it is released. But in G-RMA, τ_i^a is already released, and can be interfered by the whole τ_j^k, even if this makes it infeasible.

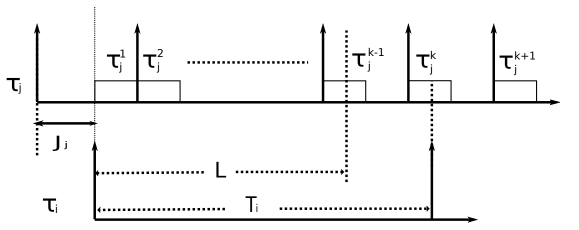


Figure 7: Max interference of τ_j to τ_i in G-RMA

Thus, the maximum contribution of τ_j^b to τ_i^a for any duration L can be deduced from Figure 7 as $W_{ij}(L) = \left(\left\lceil \frac{L-c_j}{T_j} \right\rceil + 1\right) c_j$, where L can extend to T_i. Note the contrast with ECM, where L cannot be extended directly to 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 τ_i's retry cost over duration L, which can extend to T_i, is upper bounded by:*

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

where:

- $\tau_j^* = \{\tau_j | (\tau_j \in \gamma_i(\theta)) \wedge (p_j > p_i)\}$
- $\pi(j,\theta) = \sum_{\forall s_j^l(\theta)} len\left(s_j^l(\theta) + s_{max}^j(\theta)\right)$
- $s_{max}^{min}(\theta) = min_{\forall \tau_j^*}\{s_{max}^j(\theta)\}$

Proof. 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 T_i, but in ECM, it cannot, as the interference pattern of τ_j to τ_i changes. Thus, (14) can be used to calculate τ_i's retry cost, with some modifications, as we do not have to obtain the minimum of the two terms in (14), because τ_j's atomic sections will abort and retry only atomic sections of tasks with lower priority than τ_j. Thus, $s_{max}(\theta)$, $s_{max}^*(\theta)$, and $\bar{s}_{max}(\theta)$ are replaced by $s_{max}^{min}(\theta)$, which is the minimum of the set of maximum-length atomic sections of tasks with priority lower than τ_j and share object θ with τ_i. This is because, the maximum length atomic section of tasks other than τ_j differs according to j. Besides, as τ_i's atomic sections can be aborted only by atomic sections of higher priority tasks, not all $\tau_j \in \gamma(\theta)$ are considered, but only the subset of tasks in $\gamma(\theta)$ with priority higher than τ_i (i.e., τ_j^*). Claim follows. $\square$

5.3 Upper Bound on Response Time

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

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

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

6. STM VERSUS LOCK-FREE

We now would like to understand when STM will be beneficial compared to lock-free synchronization. The retry-loop lock-free approach in [5] is the most relevant to our work.

6.1 ECM versus Lock-Free

Claim 4. *For ECM's schedulability to be better or equal to that of [5]'s retry-loop lock-free approach, the size of s_{max} must not exceed one half of that of r_{max}, where r_{max} is the maximum execution cost of a single iteration of any lock-free retry loop of any task. 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:

$$RC(T_i) \leq \sum_{\tau_j\in\gamma_i}\left(\sum_{\theta\in\theta_i}\left(\left\lceil\frac{T_i}{T_j}\right\rceil\sum_{\forall s_j^l(\theta)}(2.s_{max})\right)\right) \quad (18)$$

where $s_j^l(\theta)$, $s_{i_{max}}(\theta)$, $s_{max}^*(\theta)$, 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 τ_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 $\alpha_{edf} = \sum_{\tau_j\in\gamma_i}\left\lceil\frac{T_i}{T_j}\right\rceil.2\beta_{i,j}^*$. Now, (18) can be modified as:

$$RC(T_i) = \alpha_{edf}.s_{max} \quad (19)$$

The loop retry cost is given by:

$$\begin{aligned} LRC(T_i) &= \sum_{\tau_j\in\gamma_i}\left(\left\lceil\frac{T_i}{T_j}\right\rceil+1\right).\beta_{i,j}.r_{max} \\ &= \alpha_{free}.r_{max} \end{aligned} \quad (20)$$

where $\beta_{i,j}$ is the number of retry loops of τ_j that accesses the same object as that accessed by some retry loop of τ_i, and $\alpha_{free} = \sum_{\tau_j\in\gamma_i}\left(\left\lceil\frac{T_i}{T_j}\right\rceil+1\right).\beta_{i,j}$. 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 that of the lock-free system:

$$\begin{aligned} \sum_{\tau_i}\frac{c_i+\alpha_{edf}.s_{max}}{T_i} &\leq \sum_{\tau_i}\frac{c_i+\alpha_{free}.r_{max}}{T_i} \\ \therefore \frac{s_{max}}{r_{max}} &\leq \frac{\sum_{\tau_i}\alpha_{free}/T_i}{\sum_{\tau_i}\alpha_{edf}/T_i} \end{aligned} \quad (21)$$

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

$$\begin{aligned} \frac{s_{max}}{r_{max}} &\leq \frac{\sum_{\tau_i}(\bar{\alpha}_{free}+\hat{\alpha}_{free})/T_i}{\sum_{\tau_i}\alpha_{edf}/T_i} \\ &= \frac{1}{2} + \frac{\sum_{\tau_i}\hat{\alpha}_{free}/T_i}{\sum_{\tau_i}\alpha_{edf}/T_i} \end{aligned} \quad (22)$$

Let $\zeta_1 = \sum_{\tau_i}\hat{\alpha}_{free}/T_i$ and $\zeta_2 = \sum_{\tau_i}\left(\frac{\alpha_{edf}}{2}\right)/T_i$. The maximum value of $\frac{\zeta_1}{2.\zeta_2} = \frac{1}{2}$, which can happen if $T_j \geq T_i$ $\therefore$

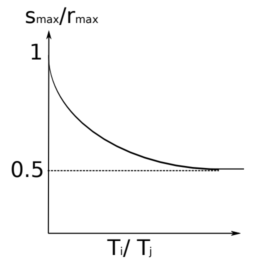


Figure 8: Effect of $\left\lceil \frac{T_i}{T_j} \right\rceil$ on $\frac{s_{max}}{r_{max}}$

$\left\lceil \frac{T_i}{T_j} \right\rceil = 1$. Then (22) = 1, which is its maximum value. $T_j \geq T_i$ means that there is a small number of interferences from other tasks to τ_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 $T_j \ll T_i$, $\therefore$ $\left\lceil \frac{T_i}{T_j} \right\rceil \to \infty$ and $\zeta_2 \to \infty$. Thus, (22) $\to 1/2$.

$\beta_{i,j}$ has little effect on s_{max}/r_{max}, as it is contained in both the 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 (22)'s numerator and denominator. However, the number of interferences of other tasks to τ_i, $\left\lceil \frac{T_i}{T_j} \right\rceil$, has the main effect on s_{max}/r_{max}. This is illustrated in Figure 8. Claim follows. $\square$

6.2 RCM versus Lock-Free

Claim 5. *For RCM's schedulability to be better or equal to that of [5]'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:

$$\sum_{(\tau_j \in \gamma_i) \wedge (p_j > p_i)} \left(\left\lceil \frac{T_i - c_j}{T_j} \right\rceil + 1 \right) .2.\beta_{i,j}.s_{max} \tag{23}$$

Consider the same assumptions as in Section 6.1. Let $\alpha_{rma} = \sum_{(\tau_j \in \gamma_i) \wedge (p_j > p_i)} \left(\left\lceil \frac{T_i - c_j}{T_j} \right\rceil + 1 \right) .2.\beta_{i,j}$. Now, the ratio s_{max}/r_{max} is upper bounded by:

$$\frac{s_{max}}{r_{max}} \leq \frac{\sum_{T_i} \alpha_{free}/t\left(T_i\right)}{\sum_{T_i} \alpha_{rma}/t\left(T_i\right)} \tag{24}$$

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, RCM is still affected by $2.\beta_{i,j}$ (just as in ECM). The subtraction of c_j in the numerator of (23) may not have a significant effect on the ratio of (24), as the loop retry cost can also be modified to account for the effect of the first interfering instance of task T_j. Therefore, $\alpha_{free} = \sum_{\tau_j \in \gamma_i} \left(\left\lceil \frac{T_i - c_j}{T_j} \right\rceil + 1 \right) \beta_{i,j}$.

Let tasks in the denominator of (24) be given indexes k instead of i, and l instead of j. Let tasks in both the numerator and denominator of (24) be arranged in the non-increasing priority order, so that $i = k$ and $j = l$. Let α_{free} in (24) be divided into two parts: $\bar{\alpha}_{free}$ that contains only tasks with priority higher than τ_i, and $\hat{\alpha}_{free}$ that contains only tasks with priority lower than τ_i. Now, (24) becomes:

$$\begin{aligned} \frac{s_{max}}{r_{max}} &\leq \frac{\sum_{\tau_i} (\bar{\alpha}_{free} + \hat{\alpha}_{free})/T_i}{\sum_{\tau_k} \alpha_{rma}/T_k} \\ &= \frac{1}{2} + \frac{\sum_{\tau_i} \hat{\alpha}_{free}/T_i}{\sum_{\tau_k} \alpha_{rma}/T_k} \end{aligned} \tag{25}$$

For convenience, we introduce the following notations:

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

τ_j is of lower priority than τ_i, which means $D_j > D_i$. Under G-RMA, this means, $T_j > T_i$. Thus, $\left\lceil \frac{T_i - c_j}{T_j} \right\rceil = 1$ for all τ_j and $\zeta_1 = \sum_{\tau_i} (\sum_{(\tau_j \in \gamma_i) \wedge (p_j < p_i)} (2.\beta_{i,j}))/T_i$. Since ζ_1 contains all τ_j of lower priority than τ_i and ζ_2 contains all τ_l of higher priority than τ_k, and tasks are arranged in the non-increasing priority order, then for each $\tau_{i,j}$, there exists $\tau_{k,l}$ 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 τ_l that will interfere with τ_k in ζ_2), and 1 means the opposite.

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

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 τ_j accesses a shared object with τ_i may not be the same as the number of times τ_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 $\left\lceil \frac{T_k - c_l}{T_l} \right\rceil$. If $\beta_{i,j}$ is increased so that $\beta_{i,j} \to \infty$, $\therefore$ (25) $\to \infty$. This is because, $\beta_{i,j}$ represents the number of times a lower priority task τ_j accesses shared objects with a higher priority task τ_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. Hence, s_{max} is allowed to be much greater than r_{max}.

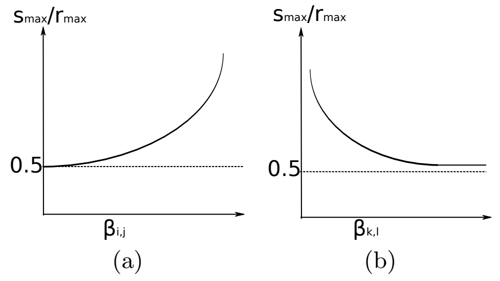


Figure 10: Change of s_{max}/r_{max}: a) $\frac{s_{max}}{r_{max}}$ versus $\beta_{i,j}$ and b) $\frac{s_{max}}{r_{max}}$ versus $\beta_{k,l}$

Although the minimum value for $\beta_{i,j}$ is 1, mathematically, if $\beta_{i,j} \to 0$, then (25) $\to 1/2$. 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 illustrated in Figure 10(a). If $\beta_{k,l} \to \infty$, then (25) $\to 1/2$. This is because, $\beta_{k,l}$ represents the number of times a higher priority task τ_l accesses shared objects with a lower priority task τ_k. Under RCM, this will increase the retry cost, thus reducing s_{max}/r_{max}. But if $\beta_{k,l} \to 0$, then (25)$\to \infty$. This is due to the lower contention from a higher priority task τ_l to a lower priority task τ_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_k/T_l. If $T_l \ll T_k$, then $\left\lceil \frac{T_k - c_l}{T_l} \right\rceil \to \infty$, and (25) $\to 1/2$. This is due to a high number of interferences from a higher priority task τ_l to a lower priority task τ_k, which increases the retry cost under RCM, and consequently reduces s_{max}/r_{max}.

If $T_l = T_k$ (which is the maximum value for T_l as $D_l \le D_k$, because τ_l has a higher priority than τ_k), then $\left\lceil \frac{T_k - c_l}{T_l} \right\rceil \to 1$ and $\zeta_2 = \sum_{\tau_k} \frac{\sum_{(\tau_l \in \gamma_k) \wedge (p_l > p_k)} 2\beta_{k,l}}{t_k}$. 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, illustrated in Figures 10(a) and 10(b). Claim follows. □

7. CONCLUSIONS

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 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. 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.

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.

Acknowledgments

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

8. 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] 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.

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

[6] 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.

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

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

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

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

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

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

[13] 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.

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

[15] M. Schoeberl, F. Brandner, and J. Vitek. RTTM: Real-time transactional memory. In *ACM SAC*, pages 326–333, 2010.
[16] N. Shavit and D. Touitou. Software transactional memory. In *PODC*, pages 204–213, 1995.