On Bounding Response Times under Software Transactional Memory in Distributed Multiprocessor Real-Time Systems

Sherif F. Fahmy
ECE Dept., Virginia Tech
Blacksburg, VA 24061, USA
fahmy@vt.edu

Binoy Ravindran
ECE Dept., Virginia Tech
Blacksburg, VA 24061, USA
binoy@vt.edu

E.D. Jensen
The MITRE Corporation
Bedford, MA 01730, USA
jensen@mitre.org

Abstract

We consider multiprocessor distributed real-time systems where concurrency control is managed using software transactional memory (or STM). For such a system, we propose an algorithm to compute an upper bound on the response time.The proposed algorithm can be used to study the behavior of systems where node crash failures are possible. We compare the result of the proposed algorithm to a simulation of the system being studied in order to determine its efficacy. The results of our study indicate that it is possible to provide timeliness guarantees for multiprocessor distributed systems programmed using STM.

1 Introduction

Recently, there has been a shift in the computer industry from increasing clock rates to designing multicore and hyperthreading architectures in the quest to produce faster computers [16]. This paradigm shift means that programmers can no longer depend primarily on faster processors to increase the speed of their programs. Improving software performance now depends on writing correct, concurrent, code. Unfortunately, the traditional method of using locks and condition variables places a heavy burden on programmers – one that, it is believed, they are not well suited for [11].

The difficulty of reasoning about lock-based concurrency control can cause a large number of subtle program errors. Among the more common errors encountered are deadlocks, livelocks, lock convoying, and, in systems where priority is important, priority inversion.

Thus, alternatives to lock-based concurrency control have become the subject of intense study in the literature. A particularly promising alternative to lock-based concurrency control (for non-I/O code) is transactional memory. There have been significant recent efforts to apply the concepts of transactions to shared memory. Such an attempt originated as a purely hardware solution and was later extended to deal with systems where transactional support was migrated from hardware to software, see [5].

While STM has many promising features, it is not a silver bullet. Some problems associated with STM include handling irrevocable instructions such as I/O, the weak atomic semantics of some of the current implementations [7] and the overhead of retries. Despite this, its semantic simplicity makes it a very promising alternative to lock-based concurrency control.

In this paper, we present an algorithm for computing a worst-case bound on the response time of tasks in a real-time distributed multiprocessor system (we define a distributed multiprocessor system as a distributed system where each node is a multiprocessor), where where failures may occur and concurrency control is managed using STM.

2 Previous work

Since the seminal papers about hardware and software transactional memory were published, renewed interest in the field has resulted in a large body of literature on the topic (e.g, see [5]). This body of work encompasses both purely software transactional memory systems, and hybrid systems where software and hardware support for transactional memory are used together to improve performance. Despite this large body of work, to the best of our knowledge, very few papers investigate STM for distributed systems [6, 9, 12].

There has also been a dearth of work on real-time STM systems. Notable work on transactional memory and lock-free data structures in real-time systems include [1, 2, 10, 13]. However, most of these works only consider uni-processor systems (with [10] being a no-

978-3-9810801-5-5/DATE09 © 2009 EDAA

table exception). In [13], the authors consider a single processor system with priority scheduling. In [1,2], the schedulability analysis is utilization based and only considers a single processor. In [10], a multiprocessor machine is considered but the scheduling is restricted to a single node which is a multiprocessor.

In this paper, we consider using STM as the concurrency control mechanism for (non-I/O code in) distributed real-time systems. Toward this, we propose a method for computing an upper bound on the worst-case response time of periodic tasks, programmed using STM, running in a distributed real-time system that employs PFair [3] scheduling.

3 Roadmap

In this paper, we provide timeliness assurances for multiprocessor distributed systems programmed using STM. Pfair scheduling is an optimal scheduling algorithm for multiprocessor real-time systems during underload conditions [3]. Therefore, we propose an algorithm for computing an upper bound on the worst-case response times of tasks running on distributed multiprocessor systems scheduled using the Pfair discipline. Towards that end, we first show how Pfair scheduling on a single processor can be represented as EDF scheduling of the transactional model proposed in [15]. We then show how an upper bound can be placed on Pfair scheduled systems on a multiprocessor and extend this result using holistic analysis to deal with distributed systems. We then show how to incorporate the retry overhead of STM into the analysis. Finally, we show how our analysis can be extended to deal with crash failures.

4 Pfair scheduling on a single processor

We consider periodic tasks scheduled using the Pfair discipline on a single processor. Each task, τ_i, is characterized by its period, T_i, and its execution time C_i. We assume that the deadline of each task is equal to its period. The idea of Pfair scheduling [3] is to divide the processor time among the tasks in proportion to their rates (defined as $wt(\tau_i) = C_i/T_i$). To do this, a task, τ_i, is subdivided into several quanta sized subtasks, τ_{ij}, pseudo-release times, $r(\tau_{ij})$, and pseudo-deadlines, $d(\tau_{ij})$, are derived for these subtasks and then they are scheduled using the EDF discipline (ties are broken using tie breaking rules [3]).

In the rest of the paper, we refer to pseudo-deadlines and pseudo-release times as simply deadlines and release times for simplicity. For synchronous tasks, the deadlines and release times of subtasks can be derived using the following equations:

$$r(\tau_{ij}) = \left\lfloor \frac{j-1}{wt(\tau_i)} \right\rfloor \quad (1)\ , \quad d(\tau_{ij}) = \left\lceil \frac{j}{wt(\tau_i)} \right\rceil \quad (2)$$

For asynchronous tasks, assume that task τ_i releases its first subtask at time r and let τ_{ij} ($j \geq 1$) be this task. The release time and deadline of each subtask τ_{ik} ($k \geq j$) can be obtained by computing the term $\Delta(\tau_i) = r - \lfloor (j-1)/wt(\tau_i) \rfloor$ and adding it to Equations (1) and (2).

In [15], the authors show how it is possible to perform schedulability analysis for EDF systems programmed using "transactions". A transaction, as used in [15], is a sequence of tasks that belong to a single programming context. A sort of precedence constraint is placed on the relative execution times of these tasks by using offsets and jitters. It can be easily shown that this transactional model can be used to represent tasks scheduled using the Pfair discipline.

Specifically, since each task in Pfair scheduling is subdivided into subtasks and these subtasks belong to the same execution context, we can represent each task as a transaction. It now becomes necessary to obtain values for the jitter and offsets to specify the precedence constraints of the subtasks (i.e, τ_{ij} can only start executing after τ_{ij-1} has completed executing). In Section 4.1 we show how this is performed.

4.1 Application to Pfair scheduling

In this Section, we show how the analysis of [15] can be applied to Pfair scheduled systems. As mentioned before, in Pfair scheduling, each task, τ_i, is subdivided into several quantum length subtasks, τ_{ij}. Therefore, we first need to determine the scheduling parameters of these subtasks. The execution time of each subtask is one quantum (i.e., $C_{ij} = Q$, where Q is the duration of a scheduling quantum). To keep the analysis simple, we shall assume $Q = 1$, it is trivial to extend this analysis for $Q \geq 1$. Each subtask retains the period of its parent task, T_i, and d_{ij} is set to the value of Equation (2). We now turn our attention to the offset, ϕ_{ij}, and jitter, J_{ij} of our subtasks.

The initial values of the offsets, ϕ_{ij} are set as follows:

$$\phi_{ij} = \begin{cases} \sum_{\forall k<j} C_{ik} & \text{if } j > 1 \\ 0 & \text{if } j = 1 \end{cases} \quad (3)$$

and all jitters are set to the pseudo-release times of each subtask as computed in Equation (1), i.e., $J_{ij} = r(\tau_{ij})$. We then perform the analysis in [15] and update jitters, after the analysis, as follows:

$$J_{ij} = \begin{cases} R_{ij-1} - \phi_{ij} & \text{if } j > 1 \\ 0 & \text{if } j = 1 \end{cases} \quad (4)$$

where R_{ij} is the response time of τ_{ij}. This process is repeated until two successive iterations produce the same

response time, at which point we have the worst-case response time of the tasks in the system.

5 Multi-processors

Now that we have established that Pfair scheduling can be represented as EDF scheduling using a transactional model, we turn our attention to providing an upper bound on the worst-case response time of Pfair scheduled multi-processor systems. The following theorem shows how this can be done

Theorem 1 (Theorem 6 in [4]) *An upper bound on the response time of a task τ_k in an EDF-scheduled multiprocessor system can be derived by the fixed point iteration on the value R_k^{ub} of the following expression, starting with $R_k^{ub} = C_k$:*

$$R_k^{ub} \leftarrow C_k + \left\lfloor \frac{1}{m} \sum_{i \neq k} I_k^i(R_k^{ub}) \right\rfloor \quad (5)$$

with $I_k^i(R_k^{ub}) = \min(W_i(R_k^{ub}), J_k^i(D_k), R_k^{ub} - C_k + 1)$

The proof of Theorem 1 can be found in [4]. The term $W_i(R_k^{ub})$ is the maximum workload offered by τ_i during a period of duration R_k^{ub}, $J_k^i(D_k)$ is the maximum number of interferences by τ_i that can occur before the deadline of τ_k, and $R_k^{ub} - C_k + 1$ is a natural upper bound on the interference of any task on τ_k (because the response time, R_k^{ub}, is naturally composed of a period of time during which τ_k executes, C_k, and some interferences $R_k^{ub} - C_k + 1$). In order to take advantage of Equation (5), we need to derive expressions for these terms using our transactional model.

First, let us consider the term $W_i(R_k^{up})$. Before presenting the analysis we change the notation to $W_i(R_{ab}^{up})$, since we will be analyzing τ_{ab}. The maximum workload offered by τ_i during a period of length R_{ab}^{up} is equal to the maximum number of jobs of τ_i that can execute during that period . Remember, however, that, in Pfair scheduling, each task, τ_i, is divided into several subtasks to create a transaction Γ_i. So, in essence, when computing the term $W_i(R_{ab}^{up})$, we are computing the worst-case contribution of transaction Γ_i. From the analysis in [15], we know that the worst-case contribution of Γ_i to the response time of a task being analyzed during a period of length R_{ab}^{up} occurs when one of its tasks τ_{ik} coincides with the start of the period. We also know that when τ_{ik} coincides with the beginning of the period, the worst-case contribution of a task τ_{ij} during a period of length R_{ab}^{up} is:

$$W_{ijk}(R_{ab}^{up}) = \left(\left\lfloor \frac{J_{ij} + \varphi_{ijk}}{T_i} \right\rfloor + \left\lceil \frac{R_{ab}^{up} - \varphi_{ijk}}{T_i} \right\rceil \right)_0 C_{ij} \quad (6)$$

where $\varphi_{ijk} = T_i - (\phi_{ik} + J_{ik} - \phi_{ij}) \mod T_i$. Thus, the worst-case contribution, workload, of a transaction Γ_i to the response time of a task τ_{ab} when τ_{ik} coincides with the beginning of the period is:

$$W_{ik}(R_{ab}^{up}) = \sum W_{ijk}(R_{ab}^{up}), \quad \forall j \in \Gamma_i \quad (7)$$

and the upper bound on the contribution of Γ_i is:

$$W_i^*(R_{ab}^{up}) = \max(W_{ik}(R_{ab}^{up})), \quad \forall k \in \Gamma_i \quad (8)$$

Likewise, we can determine a value for $J_k^i(D_k)$ from the analysis in [15]. Specifically:

$$J_{ijk}(D_{ab}) = \left(\left\lfloor \frac{J_{ij} + \varphi_{ijk}}{T_i} \right\rfloor + \left\lfloor \frac{D_{ab} - \varphi_{ijk} - d_{ij}}{T_i} \right\rfloor + 1 \right)_0 C_{ij} \quad (9)$$

$$J_{ik}(D_{ab}) = \sum J_{ijk}(D_{ab}), \quad \forall j \in \Gamma_i \quad (10)$$

$$J_k^i(D_k) = J_{ab}^i(D_{ab}) = J_i^*(D_{ab}) = \max(J_{ik}(D_{ab})), \quad \forall k \in \Gamma_i \quad (11)$$

Unlike in [15], we do not have to take into account the interference of tasks belonging to the transaction being analyzed for two reasons. First, by definition of Pfair scheduling, our subtasks have precedence constraints. Thus, when analyzing a subtask we are sure that all previous subtasks have finished executing and all subsequent subtasks are yet to start. Second, since we assume that deadlines are equal to periods for transactions (see Section 4), we are sure that while analyzing a particular transaction its predecessor's deadline has already passed (and so it is no longer in the system) and its successor is yet to start (otherwise the period would have been over and the current transaction's deadline would have already passed).

Finally, we need to set the value of the offset and jitter for the subtasks in our analysis. We take a pessimistic approach and set the jitter of each subtask to one quanta after the deadline of its preceding subtask i.e.,

$$J_{ij} = \begin{cases} d_{ij-1} + 1 & \text{if } j > 1 \\ 0 & \text{if } j = 1 \end{cases} \quad (12)$$

and the offset of each subtask is set to the best case completion time of its predecessor as in Equation (3).

6 Distributed Multiprocessor Systems

Now that we have shown, in Section 5, how to obtain an upper bound on the response time of tasks scheduled using the Pfair discipline on a multiprocessor, we can extend our analysis to a distributed system. Specifically, we can use the variant of holistic analysis developed in [15], i.e., task offsets are initially set to the best-case completion time of their predecessor:

$$\phi_{ij} = \sum_{i \leq k \leq j} (\delta_{ik} + C_{ij}) + \delta_{ij} \quad \forall 1 \leq j \leq N_i \quad (13)$$

where δ_{ij} is the communication delay between nodes i and j. The jitters are all set to zero and the response time, R_{ij}, is computed using the analysis in Section 5. Then, jitters are modified as follows:

$$J_{i1} = 0 \tag{14}$$

$$J_{ij} = R_{ij-1} + \delta_{ij} - \phi_{ij} \quad \forall 1 < j \leq N_i \tag{15}$$

Offsets remain unchanged. Essentially, this means that the jitters are modified so that each task, τ_{ij}, is released at most δ_{ij}, the communication delay, time units after the completion of its predecessor τ_{ij-1}. We then compute the response times again using the analysis in Section 5. This process is repeated until the result of two successive iterations are the same, at which point we have obtained the response time for each task. If the response times do not diverge, the process above is guaranteed to converge to the solution since the process is monotonic in its parameters. Naturally, during the computation only the contribution of the tasks running on the same processor is taken into account when computing the response times.

7 Considering STM

For the sake of this analysis, we consider atomic regions programmed using STM (e.g, [8]). We assume that each atomic region is kept short and that all atomic regions have computation cost at most s. We now show how the retry overhead of these atomic regions can be incorporated into our analysis. In [10], Anderson *et. al.* show how the overhead of lock-free code can be incorporated into Pfair scheduled systems.

Here, we show how this analysis can be applied to atomic regions programmed using STM. We assume that any two atomic regions that execute concurrently can interfere with each other. We further assume that a retry can only occur at the completion of an atomic region (i.e., validation of the transaction is performed before it commits). The second assumption implies that the number of retries of an atomic region is at most the number of concurrent accesses to atomic regions by other tasks. Finally, we assume that each atomic region is small and spans, at most, two quanta. The last assumption makes sense since atomic regions are usually designed to be small in order to reduce the likelihood of interferences and hence retries.

The idea behind the analysis is to compute the worst-case overhead introduced by the retry behavior of atomic regions. This overhead is then added to the execution time of each task to compute its worst-case demand on the processor. Using these new execution times, the analysis in Sections 5 and 6 can be used to compute the response time on a distributed system.

We assume that each task, τ_i, has N_i atomic regions and accesses atomic regions at most AA_i times during each quantum. Thus, the maximum number of interferences that can occur to τ_a in a single quantum is:

$$I_a = \mathbf{maxsum}_{M-1}\{AA_i | i \neq a\} \tag{16}$$

where M is the number of processors. Also, since we assume that an atomic region can span at most two quanta, and hence can only be preempted once, the overhead introduced by a single access to an atomic region in τ_a can be computed as in Equation (17).

$$O_a^{one} = s + (2I_a + 1)s \quad (17)\ , \quad O_a = O_a^{one} \times N_a \quad (18)$$

The term $(2I_a+1)s$ in Equation (17), represents the overhead of retries. The $2I_a$ represents the maximum number of retries that may occur in the two quanta that the operation spans, and the 1 added to $2I_a$ represents the retry that may occur due to interference that occurred while the task was preempted in the middle of its atomic region (note that by our assumption this can only occur once). Now that we have computed the overhead of one atomic region, we can compute the overhead of the N_a atomic regions using Equation (18).

Thus, we can set the new execution time of task τ_a to $C_a = O_a + C_a$ and then perform the analysis of Sections 5 and 6 using this new value.

8 Handling Failures

In this section we show how the analysis in Section 5 can be extended to take failures into account. We assume that each quantum-sized subtask in the system has an associated exception handler that can be used to restore the system to a safe state in case of failure, and that this exception handler has execution time C_{ji}^h and relative deadline d_{ij}^h. The absolute deadline of the handler is relative to the time failure occurs, t_f, thus the absolute deadline of the handler is $D_{ij}^h = t_f + d_{ij}^h$. Since we cannot determine t_f a priori, we assume a worst-case scenario where each job executes to completion, using up as much processor time as possible, and then an error occurs that triggers its exception handler. It is this worst-case scenario from which we derive the deadline of the exception handlers.

8.1 Analysis

In order to incorporate exception handlers in the analysis, it is necessary to extend Equations (6) and (9) to take their overhead into account.

We assume that a critical instant occurs at time t_B (note that for this analysis we do not need to know the value of t_B, we just assume that it exists) and compute the worst-case contribution of the exception handlers

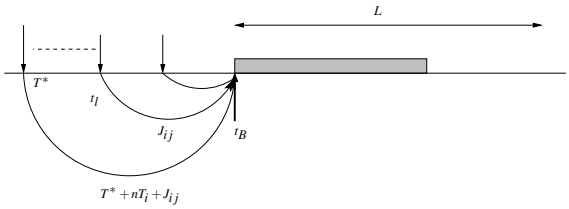

Figure 1: Scenario for calculating worst-case contribution

during a period of length L starting at t_B. From [15], we know that there are

$$n_1 = \left\lfloor \frac{J_{ij} + \varphi_{ijk}}{T_i} \right\rfloor \tag{19}$$

jobs of τ_{ij} that can start at the beginning of the period being studied after suffering some jitter. Each of these activations has an associated exception handler that can start, at most, at time $t_B + d_{ij}$. Thus, when computing the maximum number of interferences that can occur during a period of length L, we only consider these exception handlers if $d_{ij} < L$. There are

$$n_2 = \left\lceil \frac{L - \varphi_{ijk}}{T_i} \right\rceil \tag{20}$$

activations that will occur within a period of duration L. The latest start time of the exception handlers of these activations are

$$S = \varphi_{ijk} + (p-1)T_i + d_{ij} \quad \forall p = 1 \cdots n_2 \tag{21}$$

We only consider exception handlers for which $S < L$. We also need to compute the overhead of exception handlers whose activations do not contribute to the overhead. This may occur when an activation finishes before the critical instant t_B, but its exception handlers execute after t_B. In Figure 1, the first activation that can be delayed to start at the beginning of the period being studied is depicted as starting at t_l.

However, if failures are not considered, jobs that start before t_l, such as the one depicted as starting at time T^* in Figure 1, do not contribute to the analysis because even if these jobs were delayed J_{ij}, their activation time would fall before t_B. It is, now, possible for these tasks to contribute to the worst-case response time if their exception handlers start after t_B. The latest start time of a handler whose job arrives at T^* is:

$$S = T^* + d_{ij} \tag{22}$$

If S is greater than or equal to t_B then the handler will contribute to the busy period. In other words:

$$T^* + d_{ij} \geq T^* + nT_i + J_{ij} \tag{23}$$

which gives us:

$$n < \frac{d_{ij} - J_{ij}}{T_i} \tag{24}$$

Since n is an integer, Equation (24) resolves to:

$$n = \left(\left\lceil \frac{d_{ij} - J_{ij}}{T_i} \right\rceil - 1 \right)_0 \tag{25}$$

The zero subscript indicates that negative values are considered zero (in this case, such an event indicates that none of the jobs starting before t_l can contribute to the busy period). Thus we can compute the contribution of the exception handlers of τ_{ij} to the response time of τ_{ab} in a period of duration L, when the task whose activation time coincides with t_B is τ_{ik} using Algorithm 1. Thus, Equation (6) becomes:

$$W_{ijk}(R_{ab}^{up}) = \left(\left\lfloor \frac{J_{ij} + \varphi_{ijk}}{T_i} \right\rfloor + \left\lceil \frac{R_{ab}^{up} - \varphi_{ijk}}{T_i} \right\rceil \right)_0 C_{ij} + W_{ijk}^h(R_{ab}^{up}) \tag{26}$$

and Equation (9) becomes:

$$J_{ijk}(D_{ab}) = \left(\left\lfloor \frac{J_{ij} + \varphi_{ijk}}{T_i} \right\rfloor + \left\lfloor \frac{D_{ab} - \varphi_{ijk} - d_{ij}}{T_i} \right\rfloor + 1 \right)_0 C_{ij} + W_{ijk}^h(D_{ab} - d_{ij}^h) \tag{27}$$

Algorithm 1: $W_{ijk}^h(L)$

```
sum = (⌈(d_ij − J_ij)/T_i⌉ − 1)_0 C_ij^h;
if d_ij < L then
    sum ← sum + ⌊(J_ij + φ_ijk)/T_i⌋ C_ij^h;
n_2 ← ⌈(L − φ_ijk)/T_i⌉;
for 1 ≤ p ≤ n_2 do
    S ← φ_ijk + (p − 1)T_i + d_ij;
    if S < L then
        sum ← sum + C_ij^h;
return sum;
```

The rest of the analysis remains unchanged.

9 Experiments

In this section, we perform a number of experiments to verify the validity of the analysis presented. In our first set of experiments, we determine whether or not the analysis presented in Section 5 can be used to derive suitable upper bounds for the response times of Pfair scheduled tasks. Toward that goal, we conducted a number of experiments to determine the Deadline Satisfaction Ratio (or DSR) of the analysis in Section 5.

We perform the analysis for ten tasks on multiprocessors that contain 4, 6 and 8 processors. For each of these systems we fix the execution time of the tasks and vary the periods to obtain utilizations between 0 and m, where m is the number of processors. We ran our experiment 700 times and recorded the average DSR for each utilization. Figure 2 depicts the result of the experiments. The utilization on the x-axis is normalized with respect to the number of processors used.

The results indicate that the analysis is tighter for a smaller number of processors, but that it provides a good bound for response time in most cases (for example, the average DSR in our experiments does not drop below 0.8 until close to the 0.8, normalized, system utilization point). In the next set of experiments, we compute the ratio of the response time obtained using the proposed analysis to a system simulated using [14].

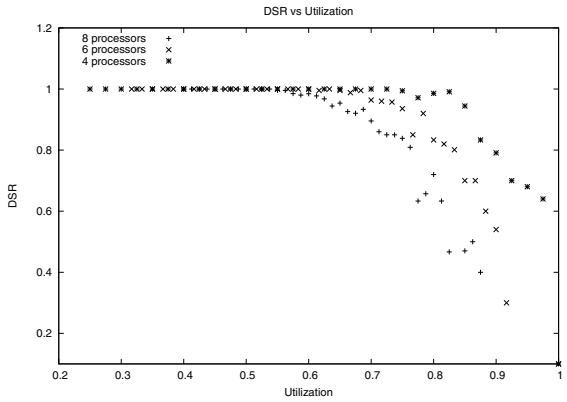

Figure 2: DSR vs. Utilization

For this experiment, we fixed the number of processors and nodes, fixed the execution times and varied the periods to obtain utilizations between 0 and m. Figure 3 depicts the result of our experiments. As can be seen, the response time analysis becomes more pessimistic as the value of s increases due to the dependence of the analysis on Equation (18). Other sources of pessimisim include Equations (5) and (12).

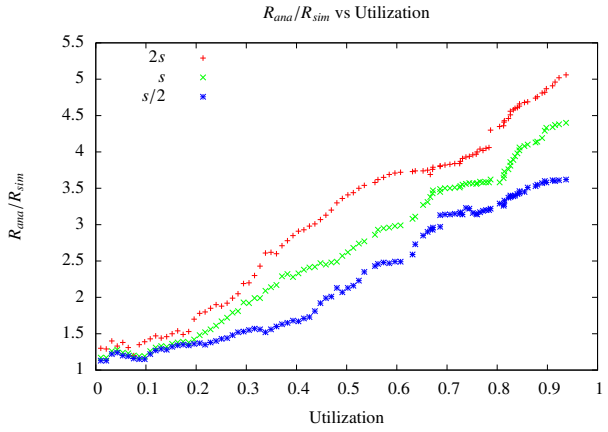

Figure 3: R_{ana}/R_{sim} vs. Utilization

10 Conclusion

We presented an algorithm for computing an upper bound on the worst-case response time for tasks on a multiprocessor distributed real-time system where concurrency control is programmed using STM.

With this result, it is now possible to include STM in the repertoire of real-time programming tools on such architectures. Future work includes tightening the analysis by considering slack (as in [4]), and considering non-periodic tasks and overload scheduling. Other directions include investigating other methods [7] for incorporating STM in distributed real-time systems.

References

[1] J. Anderson and S. Ramamurthy. A framework for implementing objects and scheduling tasks in lock-free real-time systems. In *IEEE RTSS*, pages 92–105, 1996.

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

[3] J. H. Anderson and A. Srinivasan. Mixed pfair/erfair scheduling of asynchronous periodic tasks. *J. Comput. Syst. Sci.*, 68(1):157–204, 2004.

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

[5] J. Bobba, R. Rajwar, and M. Hill. Transactional memory bibliography. `http://www.cs.wisc.edu/trans-memory/biblio/swtm.html`.

[6] R. L. Bocchino, V. S. Adve, and B. L. Chamberlain. Software transactional memory for large scale clusters. In *PPoPP '08*, pages 247–258, 2008.

[7] S. F. Fahmy, B. Ravindran, and E. D. Jensen. On scalable synchronization for distributed embedded real-time systems. In *SEUS*, 2008. `http://www.real-time.ece.vt.edu/seus08.pdf`.

[8] T. Harris and K. Fraser. Language support for lightweight transactions. In *OOPSLA*, pages 388–402. 2003.

[9] M. Herlihy and Y. Sun. Distributed transactional memory for metric-space networks. *Distributed Computing*, 20(3):195–208, 2007.

[10] P. Holman and J. H. Anderson. Supporting lock-free synchronization in pfair-scheduled real-time systems. *J. Parallel Distrib. Comput.*, 66(1):47–67, 2006.

[11] E. A. Lee. The problem with threads. *Computer*, 39(5):33–42, 2006.

[12] K. Manassiev, M. Mihailescu, and C. Amza. Exploiting distributed version concurrency in a transactional memory cluster. In *PPoPP '06*, pages 198–208. 2006.

[13] J. Manson, J. Baker, A. Cunei, S. Jagannathan, M. Prochazka, B. Xin, and J. Vitek. Preemptible atomic regions for real-time java. *RTSS*, 0:62–71, 2005.

[14] P. Pagano, P. Batra, and G. Lipari. A framework for modeling operating system mechanisms in the simulation of network protocols for real-time distributed systems. *IPDPS*, 0:160, 2007.

[15] J. Palencia and M. G. Harbour. Offset-based response time analysis of distributed systems scheduled under edf. *ECRTS*, 00:3, 2003.

[16] H. Sutter. The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobb's Journal*, 30(3), 2005.