ClockWork: a Real-Time Feasibility Analysis Tool

Pierre G. Jansen, Ferdy Hanssen, Sape Mullender
Dept. of Computer Science — University of Twente
E-mail: {jansen,hanssen}@cs.utwente.nl

May 21, 2002

Abstract

ClockWork shows that we can improve the *flexibility* and *efficiency* of real-time kernels. We do this by proposing methods for scheduling based on so-called *Real-Time Transactions*. *ClockWork* uses Real-Time Transactions which allow scheduling decisions to be taken by the system. A programmer does not need to be aware of synchronisation due to the sharing of resources and may have the illusion of a *run-to-completion* semantics even under preemptive scheduling protocols. The *ClockWork* tool presented here analyses the schedulability of a set of RT Transactions for a variety of protocols and visualises the result in a graphical form[1].

Keywords real-time, transactions, schedulability, inheritance, run-to-completion semantics, feasibility analysis

Real-Time Transactions

Real-time (RT) systems have become an important part of all the ubiquitous computers found in our every day environment. These systems span a wide variety of different tasks, from simple to very complex, all requiring some form of timely behaviour. This wide availability of RT systems is recognised at the University of Twente as well as the need for flexible and precise protocols to guarantee the schedulability of a set of RT tasks.

We propose to model these tasks as a set of Real-Time Transactions (RTTs). RTTs are normal periodic tasks. RTT τ_i is defined by its deadline D_i, period T_i, run-cost C_i, set of resources used ρ_i, and inherited deadline Δ_i, inspired by priority inheritance [11]. The determination of the inherited deadline Δ depends on the resource usage and can be determined offline. Once all inherited deadlines of all tasks in the task set are computed, a task can behave as a RTT. The scheduler does not need to have further knowledge of resources for scheduling and resource synchronisation, even if preemptive scheduling is used. The system can do the scheduling according to very simple scheduling rules based on Earliest Deadline First (EDF), Rate Monotonic (RM) or Deadline Monotonic (DM).

RTTs give the guarantee that shared resources are used under mutual exclusion, also when the scheduling protocols are preemptive. This is an important advantage because the application programmer does not have to be confronted with synchronisation problems: no *wait* and *sync* or p and v are needed. This facilitates the correctness of the programs considerably; the application programmer may have the illusion of *run-to-completion* semantics.

A RT task should complete before its deadline. A feasibility analysis serves to verify if this is the case for all tasks. Such an analysis can be simple; e.g. a task set of n tasks, scheduling with preemptive EDF, with $D_i = T_i$ for all tasks is feasible if

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \qquad \text{// See [7]}$$

However, in case resources are used or $D_i < T_i$, more complex methods, originated by [1] and refined by [4], are needed to verify the feasibility. *ClockWork* can do this analysis as is shown in the following section.

ClockWork

We will introduce our RT feasibility analysis tool by means of a typical example. For theoretical foundations we refer to [5]. Consider task set Γ_1, as defined in table 1. In order to take into account the feasibility we need to compute the *Processor Demand* function $H(t)$ [4] as well as the *Work Load* function $W(t)$ [4], defined as:

$$H(t) = \sum_{i=1}^{n} \left\lfloor \frac{t - D_i + T_i}{T_i} \right\rfloor C_i$$

[1] The research is executed at the University of Twente in the real-time "Tukker" project.

	D_i	T_i	C_i	Δ_i	ρ_i
τ_1	3	4	1	3	{a}
τ_2	4	6	1	3	{A B}
τ_3	5	7	1	5	{c}
τ_4	6	9	2	4	{b}

Table 1: Specification of Γ_1

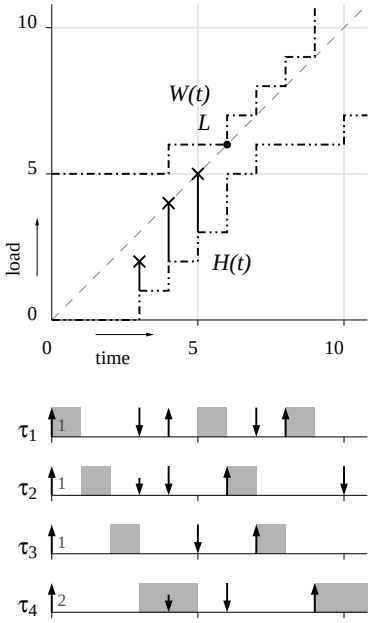

Figure 1: Analysis of Γ_1.

$$W(t) = \sum_{i=1}^{n} \left\lceil \frac{t}{T_i} \right\rceil C_i$$

In figure 1 the 45 degree line is the time-line and reflects the time elapsed since the zero point. $W(t)$ is the accumulative load that is inserted in the system and may intersect the time-line at the idle point L; a point at which all work load must have been resolved, presuming that the processor is working if there is any load. $H(t)$ is the accumulative load that needs to be resolved before the respective deadlines. Without giving the details of the correctness we can state that a task set is schedulable in the idle interval $[0, L]$ if $H(t) \leq t$. If $H(t) > t$ at some point in $[0, L]$ we can find a trace of transactions such that a deadline is exceeded and consequently the set is not schedulable (that is not feasible).

If shared resources are used or if the task set is not preemptive then the problem of *blocking* may disturb the feasibility analysis. Blocking occurs if a transaction with a *short* deadline has to wait for the preemption of a transaction with a *long* deadline beyond the deadline of the blocked task. This can happen due to non-preemption, either in a non-preemptive system or in a preemptive system that uses shared resources. In the case of shared resources, the system inhibits preemption if such a shared resource is already in use. Also preemption is inhibited if the number of running blockers could be greater than one for any other transaction. This property limits the possible blocking to a maximum of one blocker. This reduces the complexity of the schedulability analysis considerably.

In task set Γ_1 presented there are three different types of resources: a, b and c. Small letters indicate read resources while capital letters indicate write resources. A read resource can be blocked by any write resource of the same sort but not by another read resource. Write resources of the same sort may block each other. For instance τ_1 can be blocked by τ_2 over A and τ_2 can be a blocker over B.

Scheduling of transactions is executed under EDF rules with one additional restriction: a released transaction τ_i with absolute deadline d_i and relative deadline D_i may preempt a running transaction τ_r with absolute deadline d_r and inherited deadline Δ_i iff

$$(d_i < d_r) \wedge (D_i < \Delta_r)$$

Such scheduling is called EDFI, where the I stands for *inheritance*.

Blocking can effectively be computed by substituting inherited deadlines for the resources. In table 1 τ_2 inherits the deadline of τ_1 over A, τ_3 inherits its own deadline and τ_4 inherits the deadline of τ_2. The inherited deadlines are represented in the Δ-column and are indicated as Δ_i.

Possible blocking of the transactions can easily be computed by the following blocker relation: τ_i is blocked by τ_j if

$$\Delta_j \leq D_i < D_j$$

And the maximum blocking of τ_i is

$$\max\{C_j \mid \Delta_j \leq D_i < D_j\}$$

We can account for the maximum blocking by correction of the function $H(t)$ with the vertical blocking lines at the respective deadlines of the tasks. For instance at $t = 3$ the maximum blocker is C_2 with a maximum blocking of 1, at $t = 4$ this is $C_4 = 2$ and at $t = 5$ this is $C_4 = 2$ (indirect blocking to prevent multiple blocking).

We state that if $H(t)$, with the blocking correction, exceeds the time-line at the given deadlines then we can construct a possible deadline excess and the set cannot be schedulable anymore. The proof is given in [5]. If $H(t)$ including blocking corrections stays below the time-line in the interval $[0, L]$, then the set is feasible.

The example shown works for transactions under EDF and the principle of what *ClockWork* can do is shown. However, *ClockWork* can do much more: it can do feasibility analyses for the following protocols and its associated attributes in any combination:

- EDF, RM or DM
- preemption or not
- nested critical sections
- use of multiple unit resources
- slack time reservation

The complexity of the algorithm and the blocking computation is polynomial. Except when multiple unit resources are considered, then the complexity of the blocking computation becomes exponential.

The *ClockWork* tool is written in Perl and PostScript. The current version comprises ca. 1800 lines of Perl code and ca. 1200 lines of PostScript code. It is a command-line tool, which reads a textual input file and produces PostScript output. A web interface (written using ca. 350 lines of PHP code), is also available at `http://wwwes.cs.utwente.nl/feas/` and produces PostScript and PDF output.

Current work

The RTT protocol is currently working under RT-Plan 9 [8], RT-Linux [10], and Linux-RTAI [9]. Because the implementation of the protocol itself as well as the feasibility analysis is straight-forward, we also use the protocol for the RT communication in the network of the At Home Anywhere project. RTT is also a good candidate for our video server Clockwise [2]. It is furthermore used as a theory of reference in the periodic jukebox server [3, 6].

Conclusion

ClockWork is a graphical tool and demonstrator that can do the schedulability analysis for a wide variation of real-time scheduling protocols based on RT transactions with deadlines such as RM, DM and EDF. Variations such as preemption or no preemption, nested critical sections, read/write resources and multiple unit resources can be specified. It shows that the scheduling protocols give the user a run-to-completion semantics, even under preemption. *ClockWork* also shows that in most cases the feasibility analysis can be done in polynomial time and that it can be used on-line for admission control in real-time operating systems. The methods of *ClockWork* are currently used in several RT-Linux clones and in Plan 9.

References

[1] N.C. Audsley, A. Burns, M.F. Richardson, A.J. Wellings, Hard Real-Time Scheduling: The Deadline Monotonic Approach, *Internal report*, Dept. of Comp. Science, Univ. of York, 1991

[2] P. Bosch, S.J. Mullender, P.G. Jansen, Clockwise: A Mixed-Media File System, *Conference proceedings of the IEEE ICMCS'99*, Firenze, Italy, pp. 277–281, June 1999

[3] F. Hanssen, P.G. Jansen, M. Lijding, Using an early quantum server to build a periodic jukebox server, *in draft*

[4] P.G. Jansen, R. Laan, The stack resource protocol based on real-time transactions, *IEE Proceedings Software*, **146(2)**:112–119, 1999

[5] P.G. Jansen, A Generalised Scheduling Theory based on Real-Time Transactions, *in draft*

[6] M. Lijding, S. Mullender, P.G. Jansen, Scheduling in Hierarchical Multimedia Archives, *Submitted for publication*, 2002

[7] C.L. Liu and J.W. Layland, Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment, *Journal of the ACM*, **20(1)**:40–61, 1973

[8] R. Pike, D. Presotto, S. Dorward, B. Flandrena, K. Thompson, H. Trickey, P. Winterbottom, Plan 9 from Bell Labs, *Computing Systems*, **8(3)**:221–254, 1995, `http://www.cs.bell-labs.com/sys/doc/`

[9] Real-Time Application Interface for Linux web site, `http://www.aero.polimi.it/~rtai/`

[10] Real-Time Linux web site, `http://www.fsmlabs.com/`

[11] L. Sha, R. Rajkumar, J.P. Lehoczky, Priority Inheritance Protocols: An Approach to Real-Time Synchronization, *IEEE Transactions on Computers*, **39(9)**:1175–1185, 1990