

In B. Kleinjohann et al. eds., '*Design and Analysis of Distributed Embedded Systems*'
(Proc. IFIP 17th World Computer Congress, TC10 Stream, Montreal, Can., Aug 2002),
Kluwer, pp.205-215.

Going Beyond Deadline-Driven Low-level Scheduling in Distributed Real-Time Computing Systems

K.H. (Kane) Kim and Juqiang Liu
Department of *Electrical and Computer Engineering*
University of California,
Irvine, CA, 92697, U.S.A
{kane, jqliu} @ ece.uci.edu

Abstract: In real-time computing systems, timing-requirement specifications coming from the application designer are the obvious primary driver for resource allocation. Deadline-driven scheduling of computation-segments has been studied as an advanced mode of scheduling devised to meet the timing requirement specifications. However, it does not reflect additional concerns of the application designer, the damaging impacts of various timing violations on the application. The notion of *risk-incursion function* (RIF) as a framework for specification of such damaging impacts has been established by the first co-author. In this paper, a concrete implementation approach of the RIF-driven resource allocation scheme is discussed first. Then two RIF-based scheduling algorithms are discussed. The results of the experiment conducted to compare the performance of RIF-based scheduling algorithms against that of deadline-driven scheduling algorithms are also provided.

Key words: Real-time, RIF, Risk Incursion Function, RIPF, Risk Incursion Potential Function, Resource Allocation, scheduling, deadline.

1. INTRODUCTION

As the real-time (RT) computing field continues to grow, calls for significant advances in the technology for resource allocation are increasingly heard in industry. Much of the technology practiced in industry for scheduling of RT computations in the past 35 years has been at the level of assigning priorities to processes.

It has been known for long that the deadline-driven scheduling is a step forward from the fixed-priority scheduling in terms of reflecting the application requirements closely [Fin67, Kop97, Liu73, Ser72]. However, deadline-driven scheduling has not been practiced much. Guiding and helping application designers to specify deadlines have been a challenge for a long time.

In recent years, research produced a programming approach such as the *time-triggered message-triggered object* (TMO) programming scheme [Kim97, Kim00] with which programmers can specify start time-windows and completion deadlines of RT computation-segments in convenient manners. The practice of deadline-driven scheduling is bound to increase.

After all, it is natural for RT application programmers to think about start time-windows and completion deadlines rather than priority numbers. The argument that low priorities can be given to unimportant computation-segments is a peripheral argument for using priority-based scheduling approaches. If such treatment occurs in absence of computing resource failures, such computation-segments are trivial ones which could have been omitted to begin with. If computing resource failures occur and the resulting resource shortage dictates sacrificing the abilities to meet all timing requirements of all computation-segments, then less important computation-segments can be sacrificed. If such a decision is represented by assignment of low priorities to the victimized computation-segments, then the priority is really a number representing criticality or importance rather than a timing requirement. Reflecting criticality is an issue orthogonal to that of reflecting timing requirements. Reflecting criticality can be combined with approaches to reflecting timing requirements, including deadline-driven scheduling.

Therefore, reflecting the start time-window specifications and the completion deadline specifications supplied by application designers in low-level (short-term) scheduling is expected to be the direction increasingly adopted in industry during this decade. To go a step further in reflecting application designers' concerns, the scheduler must be designed to reflect not only timing specifications associated with various RT computation-segments but also specifications of damaging impacts of various timing violations to the application. Typically the impacts of violating the completion deadlines of different output actions vary. The violations of some output deadlines may lead to the failure of the entire application, while the application may continue with lowered application service goals when the violations of some other deadlines occur.

We have taken the position that specifying the damaging impacts of various timing violations to the application is at the core of specifying the quality-of-service (QoS) requirements imposed on the application system. The first co-author established the notion of *risk-incursion function* (RIF) as

a framework for specification of such damaging impacts [Kim01]. Basically, the damaging impact potentials incurred by timing violations are called the *risks*. In this paper, we discuss two specific low-level scheduling algorithms devised to reflect RIF specifications. These algorithms function essentially as deadline-driven scheduling algorithms when sufficient computing resources are available for handling the given application. In the situations of computing resource shortage, these algorithms behave toward minimizing the risks to the application. Therefore, these algorithms are advances over deadline-driven scheduling algorithms.

The capabilities of the two algorithms have recently been validated through experiments. The experiment setup and the measurement results are also reported in this paper.

We consider the application software which is constructed in the form of a TMO network. The effectiveness of the TMO programming scheme in easy-to-understand and easy-to-analyze RT distributed programs has been demonstrated in quite a few contexts [Kim97, additional references in <http://dream.eng.uci.edu/tmo/tmo.htm>]. Execution of TMOs can be facilitated by building an execution engine as middleware running on well-established commercial software / hardware platforms. A middleware model named the *TMO support middleware* (TMOSM) architecture has been established and several prototype implementations on different hardware+OS platforms have been developed [Kim99, Kim02].

Three basic practical types of RIFs are then discussed in Section 2 and a design example is also given to illustrate the application of the RIF scheme. Two RIF-based scheduling algorithms are discussed in Section 3 and the results of the experiment conducted to compare the performance of RIF-based scheduling algorithms against that of deadline-driven scheduling algorithms are also provided.

2. THREE BASIC TYPES OF RIFS AND AN EXAMPLE APPLICATION

An RIF is associated with each output function of the system and indicates the amount of risk incurred when the value carried in and the timing of an execution of the output function deviates from the specified range. Using the RIFs as guides, risk incursion potential functions (RIPFs) associated with outputs from various program units, rather than application system outputs going to application environments, are derived, possibly with the aid of tools. Note that assigning an RIPF to an output action is almost the same as assigning an RIPF to the completion event of the computation-segment that leads to the output.

For short-term resource allocation, e.g., allocation of CPU timeslices, method-segments may be treated as basic resource users. However, assignment of effective RIPFs for such fine-grain computation units may often be very difficult and costly. Therefore, an approach considered to be cost-effective in many cases is to associate RIPFs with TMO method completions and each output actions but not with the completions of any smaller computation units.

Here we consider three basic RIF types depicted in Figure 1, which we believe are among the most practical and useful types of RIFs. In the diagrams, the y axis shows the risk incurred, and the x axis shows the output action time.

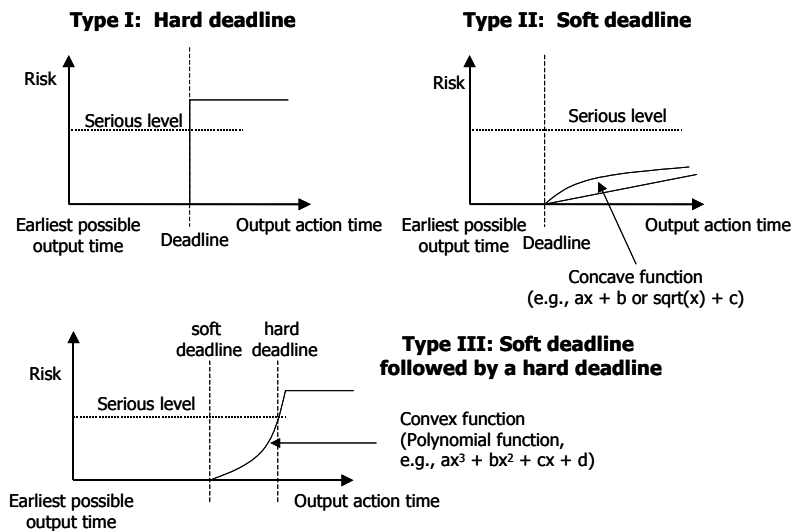


Figure 1. Three basic types of RIF

Type I: For output with a hard deadline. The risk immediately jumps from zero and exceeds the serious level after the deadline is violated.

Type II: For output with a soft deadline. The risk starts increasing from zero in a concave fashion after the deadline is violated. Also, in a certain situation the risk incurred by an inaccurate output may be “erased” after some time interval called the *validity duration* of the risk, e.g., due to a system recovery or the compensating nature of subsequent correctly executed output actions. In this type of RIF, the risk increases so slowly that it may not exceed the serious level within the lifetime of the application or for the validity duration of the risk.

Type III: For output with a soft deadline followed by a hard deadline. The risk starts increasing from zero in a convex fashion after the soft deadline is

violated. Some time after the soft deadline, the risk exceeds the serious level, and that time instant is the hard deadline for this output.

In generating the characteristic descriptions of the resource requests of various computation-segments, i.e., RIFs, by reflecting the QoS requirements, an easily understandable and yet rigorous representation of the system being developed and its application environment is of innumerable value to the system engineers. One such representation approach is the TMO structuring scheme. Here we will briefly present an example of the RIF-based system design procedure in the context of TMO-structured top-down design.

CAMIN (Coordinated Anti-Missile Interceptor Network) is a model of a defense command-and-control network which has been used in the authors' laboratory as an example of an advanced RT distributed computing application. For more detailed description of the functionality of CAMIN, readers are referred to [Kim97]. The application environment in CAMIN is a sky, land, and sea segment (together called *Theater*) – in which moving items are taken seriously. The moving items include at least one valuable target to be defended (that is assumed to be the command ship in the sea here) and flying items, both hostile and non-threatening. The high-level requirements are:

Each flying object should be intercepted if it is considered dangerous;

The valuable target (the command ship) can move around to avoid the dangers posed by flying items.

In the first step, the system engineer describes the application as two TMO's, the Theater TMO and the Alien TMO. Alien has one system output, Alien.SysOut1, which sends missiles and non-threatening flying objects (NTFOs) to Theater. Theater also has one system output going to Alien, Theater.SysOut1, which sends information about the current status of the missiles and commercial airplanes leaving from Theater to the outside. In a sense, Alien determines the workload for Theater.

In the next step, the Theater TMO can be further delineated and decomposed into three TMO's, Army's Command Post TMO (CP), Command Ship TMO (CS), and Theater-Remainder TMO (TR), as shown in Figure 2. The system outputs are:

CP.SysOut1: Send an intercept order to TR.

CP.SysOut2: Send a radar spot-check plan to TR.

CP.SysOut3: Hand-over of data on dangerous items to CS.

CS.SysOut1: Send an intercept order to TR.

CS.SysOut2: Send a radar spot-check plan to TR.

TR.SysOut2: Send radar data (both from a scan and a spot-check) to CP.

TR.SysOut3: Send radar data (both from a scan and a spot-check) to CS.

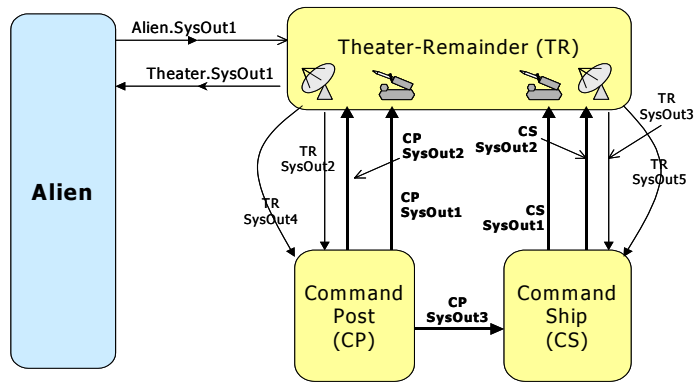


Figure 2. The decomposition of the Theater TMO

Assume that when Alien throws enemy missiles into Theater, it always puts them in the territory covered by the Army's CP and lets them move toward CS. After receiving the radar data from TR, CP must send CP.SysOut2 in time to require more detailed radar data, the spot-check data, to track the enemy missiles. After some tracking, CP must send CP.SysOut1 in time, which is the interception order, because otherwise the interception will be a failure. Since the radar in the command ship (CS TMO) is a short-range radar, CP also needs to send some warning data about dangerous surviving enemy missiles to the CS in advance. Otherwise, CS might not have enough time to intercept the missiles once CP misses them. Therefore, CP uses CP.SysOut3 to send the warnings with historical data on dangerous items to CS. Given these functionalities, a hard deadline may be imposed on CP.SysOut1, and a soft deadline on CP.SysOut2, while CP.SysOut3 may have a soft deadline followed by a hard deadline. Also CP.SysOut1 has higher risk value than CP.SysOut2 and CP.SysOut3 do.

The deadlines for these system outputs are decided based on the consideration of physical constraints. For example, the physical constraints include the frequency and accuracy of the radar data sent to CP, the flying speeds of the interceptors and the missiles, etc.

Suppose the system failure threshold is set to be 1000. Derived from this, the risk value of CP.SysOut1 after its hard deadline is missed is set to 400, the maximum risk value of CP.SysOut2 is set to 50, and the risk value of CP.SysOut3 after its hard deadline is missed is set to 200. These risk incursion values are assigned according to the system engineers' understanding of the relative importance of each system output. Different system engineers might assign slightly different numbers.

After the RIFs are decided, the next step is to further delineate and decompose the CP to multiple TMO's and derive an RIPF for each new TMO. The CP TMO can be decomposed into three TMOs: Radar Data

Queue (RDQ), Flying Object Tracking (FOT), and Interception Plan Data Store (IPDS).

The derivation of RIPFs from an RIF is based on the worst-case execution time analysis and the way the output of each TMO is used by other TMOs or actuators. Determining RIPFs for TMOs is practically to determine RIPFs for methods of TMOs including both method completion actions and specific output actions of each method. TMO consists two types of methods, time-triggered or spontaneous method (SpMs), and service methods (SvMs). Please refer to [Kim97] for the detailed description of TMO structure scheme. After reaching this step, the procedure for deriving RIPFs is completed. The derived RIPF set can be used by resource allocators, such as the processor scheduler, the communication channel scheduler, and the I/O devices scheduler.

3. THE RIPF-DRIVEN SCHEDULERS AND EXPERIMENTAL RESULT

Since the derived RIPF set also incorporates deadline information for each SpM and SvM, the RIPF-driven resource schedulers can schedule various resources at least as efficiently as the deadline-driven resource schedulers do. In this section, we discuss two RIPF-driven algorithms:

Algorithm 1: RIPF-LLF

Step 1: Execute the least-laxity-first (LLF) Algorithm; If a zero total risk arrangement is found, schedule the task set using that arrangement and return; Otherwise, go to the next step.

Step 2: Compare the risk values of the RIPFs when N timeslices (called the vision-window) have elapsed without completing the associated computation unit. Schedule the task with the maximum value. If there are more than one RIPF which have the maximum value, compare the first derivatives of them and schedule the task with the maximum value of the first derivative. If there are more than one RIPF have the maximum value of the first derivative, choose one randomly.

Algorithm 2: RIPF/Laxity

Step 1: Execute the LLF Algorithm; If a zero total risk arrangement is found, schedule the task set using that arrangement and return; Otherwise, go to next step.

Step 2: Calculate the risk value of the RIPFs when N timeslices (called the vision-window) have elapsed, and divide it by its laxity. Schedule the task with the maximum value. If there are more than one RIPF which have the maximum value, compare the first derivatives of them and schedule the task with the maximum value of the first derivative. If there are more than

one RIPF which have the maximum value of the first derivative, choose one randomly.

	Execution Period	Completion Deadline	Average Execution Time	Worst-case Execution Time	Deadline Violation under RIPF scheduler (within 60sec)	Deadline Violation Under EDF scheduler (within 60sec)
Radar->SpM1	100ms	60ms	9.3ms	10ms	141	0
Radar->SpM2	100ms	60ms	5.8ms	6ms	86	4
IPDS->SpM1	200ms	70ms	22.1ms	25ms	0	133
RDQ->SpM1	100ms	40ms	15.0ms	18ms	0	0
FOT->SpM1	500ms	80ms	25.3ms	27ms	5	50

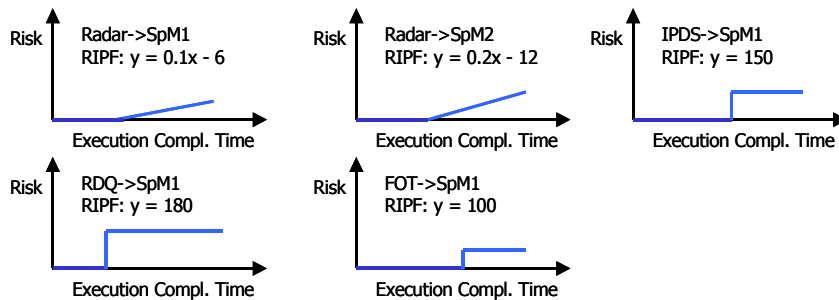


Figure 3. The experimental Data of the RIPF and EDF schedulers

Apparently, the complexities of both Algorithm 1 and Algorithm 2 are $O(n \lg n)$. Running Algorithm 1 should yield fairly good result in most application scenarios, while Algorithm 2 can yield a better result at the cost of slightly longer execution time. The selection of the size of the vision-window may affect the results of these two algorithms also.

Both Algorithm 1 and the EDF algorithm [Liu73] have been implemented in TMOSM, and the example application described in the previous section, CAMIN, has been run under the two scheduling algorithms in order to compare the performance.

Under normal circumstances, all TMOs in CAMIN can complete their tasks within the completion deadlines under both EDF and RIPF schedulers. To simulate the situation where the processor is under heavy demands, the deadlines of SpMs in Radar, FOT, IPDS, and RDQ TMOs were shortened by half. Figure 3 shows the measured data, the average and worst method completion times of SpMs, completion deadlines, and the RIPFs.

Figure 3 shows many deadline violations, and the nature of deadline violations under EDF is quite different from that under RIPF. Possible SpM execution orders under EDF and RIPF schedulers are shown in Figures 4 and 5, respectively.

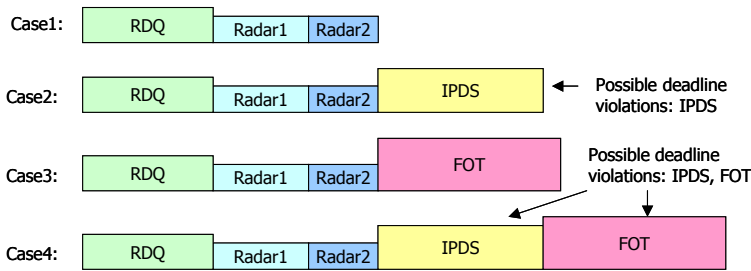


Figure 4. The possible SpM execution orders under the EDF

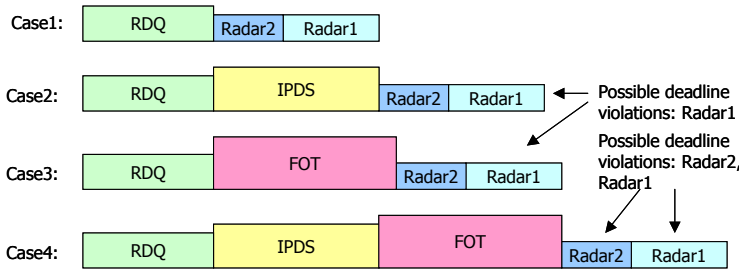


Figure 5. The possible SpM execution orders under the RIPF scheduler

Figure 4 shows that the EDF scheduler dispatches the RDQ SpM first because its deadline, 40ms, is the shortest. Then it dispatches Radar SpM1 (shown as Radar1 in the figure) and Radar SpM2 (shown as Radar2 in the figure), whose deadlines are 60ms each. The invocation intervals of these SpMs are 100ms, while IPDS SpM's invocation interval is 200ms and FOT SpM's invocation interval is 500ms, and thus in Case 1 there are no IPDS and FOT executions which need to be scheduled. When IPDS SpM and FOT SpM appear in Case 2 and Case 3, they are scheduled after the three SpMs mentioned above because their deadlines are 70ms and 80ms, respectively. When both of them appear in Case 4, FOT SpM is scheduled after IPDS SpM. According to the execution times and deadlines listed in Figure 3, there is a fairly good chance that IPDS and FOT miss their deadlines in Case 4.

Similarly, in Figure 5 the RIPF scheduler dispatches RDQ SpM first because the risk value to be incurred when the latter misses the deadline is 180. Then it dispatches IPDS SpM, FOT SpM, Radar SpM1 (shown as Radar1 in the figure), and Radar SpM2 (shown as Radar2 in the figure) according to their RIPFs. Considering the execution times and deadlines listed in Figure 3, there is a fairly good chance that Radar SpM1 and Radar SpM2 miss their deadlines in Case 4.

From these two figures, we can see that under tight resource condition, the EDF scheduler sacrifices the SpMs in IPDS and FOT first since their deadlines are the longest. In contrast, the RIPF scheduler sacrifices the SpMs in Radar first since their risk incursion values are the lowest. Considering the nature of CAMIN, occasional deadline violations of Radar SpMs may not cause serious result, but the deadline violations of FOT, IPDS or RDQ SpMs may cause the failure of missile interception, thus should be avoided if possible. Therefore, the CAMIN application shows better missile interception rates under the RIPF scheduler when the deadlines are set as in Figure 3, i.e., the processor availability is tight.

Our analysis and experiments have thus shown that: 1) If the deadlines of all tasks can be met, the EDF and RIPF schedulers perform equally well; 2) In the case where not all deadlines can be met under EDF, RIPF can do a better job by considering the potential risk values together with the deadline information in the RIPFs, which means less important tasks are sacrificed first.

4. CONCLUSION

Deadline-driven scheduling, although a better alternative to the fix-priority scheduling, does not consider many important QoS requirements from the system designer, such as the impacts of deadline violations. A scheme for specification of not just timing and other QoS requirements associated with system output actions, but also the potential damages that failing to meet the requirements bring in, has been evolving in the authors' laboratory under the name of the risk incursion function (RIF) scheme. Two RIF-driven algorithms were proposed in this paper, and some experimental results on the comparison between RIPF-driven and deadline-driven scheduling algorithms were provided. The research on QoS-driven resource allocation in complex distributed RT systems is still in its early stage. In order to establish it as a widely practicable technology, further research on systematic derivation of RIPFs from the original RIF-based QoS requirements specification is desirable.

Acknowledgements: The research reported here was supported in part by NSF under Grant Numbers 99-75053 (NGS) and 00-86147 (ITR), and in part by US DARPA (NEST) under Contract F33615-01-C-1902 monitored by AFRL. No part of this paper represents the views and opinions of any of the sponsors mentioned above.

REFERENCES

- [Fin67] Mark S. Fineberg and Omri Serlin, "Multiprogramming for hybrid computation", *Proc. AFIPS Conf.*, vol. 31, Anaheim, Nov. 1967, pp. 1-13.
- [Kim97] Kim, K.H., "Object Structures for Real-Time Systems and Simulators", *IEEE Computer*, Vol. 30, No.8, August 1997, pp. 62-70.
- [Kim99] Kim, K.H. Ishida, Masaki, Liu, Juqiang, "An Efficient Middleware Architecture Supporting Time-Triggered Message-Triggered Objects and an NT-based Implementation", *Proc. 2nd IEEE CS Int'l Symp. on Object-Oriented Real-time Distributed Computing (ISORC '99)*, St. Malo, France, May, 1999, pp.54-63.
- [Kim00] Kim, K.H., "APIs for Real-Time Distributed Object Programming", *IEEE Computer*, June 2000, pp.72-80.
- [Kim01] Kim, K.H, Liu, Juqiang, "QoS-driven Resource Management in Real-Time Object Based Distributed Computing Systems", *proc. FTDCS'01 (The 8th IEEE Workshop on Future Trends of Distributed Computing Systems)*, Bologna, Italy, October 2001, pp. 222-230.
- [Kim02] Kim, H.J., Park, S.H., Kim, J.G., and Kim, M.H., "TMO-Linux: A Linux-based Real-time Operating System Supporting Execution of TMOs", to appear in *Proc. ISORC 2002 (5th IEEE CS Int'l Symp. on OO Real-time distributed Computing)*, Washington DC, Apr. 2002.
- [Kop97] Kopetz, H., 'Real-Time Systems: Design Principles for Distributed Embedded Applications', *Kluwer Pub.*, ISBN: 0-7923-9894-7, Boston, 1997.
- [Liu73] C.L. Liu and J.W. Layland. "Scheduling algorithms for multiprogramming in a hard real-time environment", *J. ACM*, 20(1):46-61, Jan. 1973, pp.46-61.
- [Pal00] Pal, P.P., et al, "Using QDL to Specify QoS Aware Distributed (QuO) Application Configuration", *Proc. ISORC 2000 (3rd IEEE Int'l Symp. on Object-Oriented Real-time Distributed Computing)*, March 2000, Newport Beach, CA, pp. 310-319.
- [Ser72] Serlin, O., "Scheduling of time critical process", *proc. AFIPS conf.*, Vol. 40, Atlantic City, NJ, May 1972, pp.925-932.