Run-Time Monitoring of Fixed-Contract Interruptible Algorithms

Martin Adelantado

Onera-Cert
Department of Computer Science
2, Avenue Edouard Belin,
31055 Toulouse Cedex, France
e-mail: Martin.Adelantado@cert.fr
fax: (33) 05-62-25-25-93
http://www.cert.fr/anglais/deri/adele/adele.html

Abstract

Anytime algorithms give intelligent real-time systems the ability to trade deliberation time for quality of results. This capability is essential in domains where computing the optimal result is not computationally feasible or is not economically desirable. Examples of such domains include avionics, air traffic control, process control and mission-critical computations. Run-time monitoring of anytime algorithms defines a framework for reducing the effect of uncertainty on the performance of the system. In this paper, we discuss ongoing work aiming at extending the scope of anytime computing when performance profiles of basic components are not predictable at compile time. More precisely, we describe a two-levels model of run-time monitoring for resource-bounded problem-solving systems: at meta-level, original fixed-contracts are allocated to interruptible tasks according to some expectations on the environment behavior. At resource-level, contracts adjustments are dynamically performed by a scheduler according to the computational resources workload and the quality of the available approximate results, estimated at run-time. This work, supported by the French Ministry of Defense (DRET)[1], has been conducted in collaboration with G. Champigneux and J. Sardou from Dassault-Aviation Company.

Introduction

Anytime computing provides a framework for preventing timing faults and achieving graceful degradation in intelligent real-time systems. The problem in monitoring anytime algorithms is to decide how to allocate processing time among different tasks so as to optimize the total performance of the system. Several monitoring approaches have been proposed. The *compilation* approach (Russel and Zilberstein 1991), (Zilberstein 1996), (Zilberstein 1993), (Grass and Zilberstein 1996) calculates a fixed temporal contract to each component of the system prior to its activation. M. Boddy and Th. Dean (Boddy and Dean 1994) introduced several deliberation scheduling techniques to make time-allocation decisions in time-constrained planning problems. E. J. Horvitz (Horvitz 1990) proposed *flexible inferences* to solve time-critical decision problems, implementing decision-theoretic reasoning about what next action has to be performed. J. Liu et al. (Liu et al. 1991) propose the *imprecise computation* technique to achieve graceful degradation of approximate results. In their *design-to-time* approach to real-time scheduling, A. Garvey et al. (Garvey, Humphrey and Lesser 1993) exploit task interrelationships in situations where multiple methods exist for many tasks that the system needs to solve. A. Mensch and F. Charpillet (Mensch and Charpillet 1996) propose algorithms supporting a joint scheduling approach of predictable tasks and unbounded optional tasks, in real-time knowledge-based systems.

The fixed-contract monitoring strategy is optimal when the domain has predictable utility and the system provides deterministic performance profiles (Zilberstein 1993). Unfortunately, there is in most cases, uncertainty about the rate of improvement of solution quality, or about the future state of the environment. To face these situations, a great deal of research effort has been spent in defining run-time strategies for solving the monitoring problem (Hansen and Zilberstein 1996).

In this paper we discuss a run-time monitoring model based on the following assumptions: First, despite unpredictable changes on the environment behavior, utility of a result can still be predictable. Such assumption allows to calculate an original fixed-contract before activating an anytime process. Second, performance profiles are not

[1] Under contract 9400200051.

predictable. Therefore, expectations on the quality of the future approximate results, are not allowed (myopic approach). Nevertheless, an interruptible task can accurately calculate the quality of the currently available solution. Third, at run-time, computational resources are to be shared between several parallel interruptible algorithms. In the next section we describe the run-time monitoring model we propose through an example taken from actual applications. Then, we examine several monitoring strategies for dynamically adjusting the original fixed-contracts according to the resources workload. Finally, we briefly discuss the significance of this work and current research directions.

Run-Time Monitoring Model and Problem Statement

The run-time monitoring model shown in figure 1, is based on a cooperative scheduling approach between a meta-level scheduler and a resource-level scheduler. Such a model is suitable for designing distributed time-constrained systems in such areas as avionics, air-traffic management or distributed air defense systems.

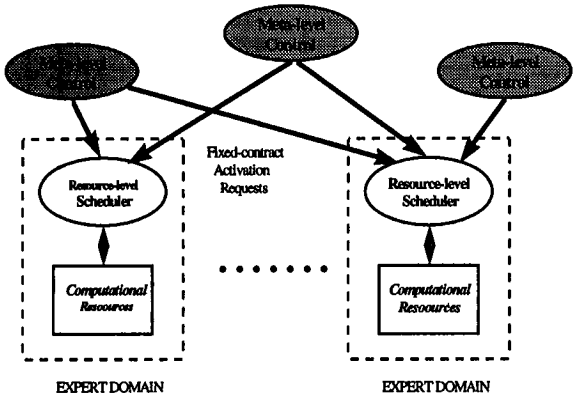

Figure 1: Two-levels run-time monitoring

For example, in an embedded avionics system, the guidance and control function involves *Situation Assessment* to detect abnormal situations (threats, system failures, ...), and *Plan Generation* to create one or several plans to face unexpected problems. In generating plans, a variety of mechanisms and algorithms may be applied. The decision to which plan generation mechanisms to employ is made by a *Coordination* function according to a given strategy including the time allocated to generate a plan and constraints to be employed. For the sake of simplicity, a strategy describes an activation order of the processes.

The Coordination function can be then considered as a meta-level control flow controlling the real-time decision making performed by the Plan Generation function. From the Coordination point of view, Planning consists then of sending tasks activation requests to a set of expert domains performing a plan according to their own operational knowledge. Each request from the meta-level control flow defines a fixed-contract activation of an interruptible process. When the system is designed for simultaneously handling up to N abnormal situations, Planning involves up to N meta-level flows of control, each of them sending asynchronously tasks activation requests to the corresponding expert domains.

Based on such a monitoring scheme, we assume that an interruptible task can easily estimate the quality of the currently available result. Time is then discretized into a finite number of time steps, t_0, t_1, ..., t_n. Similarly, solution quality is discretized into a finite number of levels q_0, q_1, ...,q_n. Whenever an activation request of an interruptible task **T** is sent, the corresponding meta-level control flow **C** decides how much time ΔT should be given to **T** to receive an approximate result. Process **T** returns the approximate result elaborated at the last time step t_i before reaching the deadline, or the correct result if **T** has completed before deadline. Usually, a fixed-contract approach can be used when the complete system is compiled into a contract algorithm. We consider in this paper that an interruptible approach is more suitable since performance profiles are not predictable. Computational resources associated to an expert domain, are controlled by a scheduler which receives activation requests of fixed-contract interruptible tasks from several meta-level control flows. According to the computational resources workload, the resource-level scheduler dynamically adjust the contracts in such a way that the updated contracts are never greater than the original ones.

Resource-Level Monitoring Issues

The framework previously described raises many interesting and challenging questions. In this paper, we focus on the problem of how adjusting the fixed-contracts associated with the tasks activation requests, from the resource-schedulers point of view. In other words, we are not interested in the problem of how time can be reallocated among the remaining computational processes, by the meta-level scheduler. Assume a resource-level scheduler has p, $0<=p<n$, *pending tasks* activation requests $\mathbf{T_1}$, $\mathbf{T_2}$, ...,$\mathbf{T_n}$ with the corresponding fixed-contracts ΔT_1, ΔT_2, ..., ΔT_n. On a single processor system, $\mathbf{T_1}$ is the *running task*, and p=n-1. On a multiprocessor system, k tasks, $\mathbf{T_1}$, $\mathbf{T_2}$, ...,$\mathbf{T_k}$ are running and p=n-k. Based on this formal framework, several *discrete monitoring* policies[2] can be examined. Those policies perform contracts adjustment

[2] From now on, the time spent in run-time monitoring will be neglected.

either upon arrival of a task activation request or whenever a running task has completed.

Basic Contracts Adjustment Algorithm on Single Processor Systems

Upon arrival of the $\mathbf{T}_{n+1}$ task, with a fixed-contract ΔT_{n+1}, a basic contract adjustment algorithm is performed as shown in figure 2, assuming first a single processor system. The goal of this algorithm is to dynamically adjust the pending tasks contracts according to the expected resources overload given in time units. The basic idea consists in equally diminishing the contracts of the running and the pending tasks in such a way that the deadline of $\mathbf{T}_{n+1}$ will be guaranteed. Tasks activation calls are assumed to be serviced in a FIFO order, and running tasks are never preempted. The constant noted **const** ensures that $\mathbf{T}_{n+1}$ will be allocated at least "const" time units.

1. Compute $\Delta = \left(\sum_{i=1}^{n} \Delta T_i\right) - \Delta T_{n+1}$

2. If $\Delta \leq (-\mathbf{const})$ *then* $\Delta T'_{n+1} = (-\Delta)$*; exit*

3. Compute $\Gamma = \dfrac{\Delta + \mathbf{const}}{\mathbf{n}}$

4. For i=1 to n

4.1. Compute $\Delta T'_i = \max(\mathbf{const}, \Delta T_i - \Gamma)$

4.2. Compute $\Delta = \Delta - \Gamma$

4.3. If $\Delta T'_i == \mathbf{const}$ *and* $\mathbf{i! = n}$ *then*

4.4. Compute $\Delta = \Delta + (\mathbf{const} - (\Delta T_i - \Gamma))$

4.5. Compute $\Gamma = \dfrac{\Delta}{\mathbf{n - i}}$

4.6. EndIf

5. EndFor

Figure 2: Adjusting contracts upon arrival of a task activation request

The first step consists of computing the expected timing overload. At the second step, the resources are not overloaded: $\Delta T'_{n+1}$ is accordingly updated, and the algorithm exits. Otherwise, all the contracts associated with the pending tasks have to be adjusted using Γ (step 4). At step 4.1, the adjusted contract is lower bounded by a constant (const>=0). A value of const greater than 0 ensures that all the pending tasks will be allocated at least "const" time units. A value of const equal to 0 allows a task activation request to be ignored when $\Delta T'_i$=0. Steps 4.4 and 4.5 update the timing overload, respectively Γ, whenever a given contract has been "less adjusted" than expected. The worst case occurs if the updated contract $\Delta T'_n$ is equal to "const" when the algorithm ends. In such a case, all the pending tasks contracts have been correctly diminished, except the contract of $\mathbf{T}_n$. Therefore, the deadline of $\mathbf{T}_{n+1}$ is not guaranteed and the algorithm fails. Several solutions are offered to solve this problem. The first one consists in removing the task $\mathbf{T}_{n+1}$. In the second solution, the scheduler selects among $\mathbf{T}_1$ to $\mathbf{T}_{n+1}$, the first task $\mathbf{T}_h$ such as

$$\Delta T'_h - (\mathbf{const} - (\Delta T_n - \Gamma)) > \mathbf{const}$$

and diminishes then the contract of $\mathbf{T}_h$ as follows:

$$\Delta T''_h = \Delta T'_h - (\mathbf{const} - (\Delta T_n - \Gamma))$$

When $\Delta T'_1 < \Delta T_1$, the scheduler updates the abort conditions of the running task $\mathbf{T}_1$ through a process interaction request, in the following way: let t_{start} the starting time of $\mathbf{T}_1$, and t the current time. If $\Delta T'_1$ is greater than t-t_{start}, the adjusted abort condition is indicated to the running task. Otherwise, $\mathbf{T}_1$ is discarded and the last approximate result elaborated so far is returned. The updated contracts computed by this basic algorithm have to be considered as worst-case contracts: first, because the running time already spent by $\mathbf{T}_1$ has been neglected in calculating the expected resource overload, and secondly, because a running task can terminate before its deadline, since the execution time needed to obtain the correct result depends on input data and processing power. In such a case, when the running task $\mathbf{T}_1$ has completed at time t, a time recovering mechanism is performed by the scheduler by adjusting once again the contract of the *ready task* $\mathbf{T}_2$. The contract adjustment equation of the ready task is then the following:

$$\Delta T''_2 = \Delta T'_2 + ((t_{start} + \Delta T'_1) - t)$$

A similar mechanism may be applied by the meta-level scheduler to reallocate the remaining time to the following processes. For example, when the Planning function involves up to N meta-level flows of control, each of them has to send several tasks activation requests to the expert domains. Therefore, each flow of control has to sequentially send R requests to the corresponding resource-level dispatchers. For a given problem, Planning has to be performed in ΔT units of time. Before sending the activation request of the task $\mathbf{T}_i$, the meta-level scheduler estimates the fixed contract ΔT_i according to its knowledge about the future behavior of the Planning process. An extremely simple policy consists of equally allocating time to the remaining processes. Whenever a given ΔT_i has not been wholly used, the meta-level scheduler has the opportunity to give to the following tasks, more time than previously expected.

Several straightforward improvements may be suggested to enhance the basic algorithm behavior. For example, instead of adjusting the contract of the ready task upon completion of the running task, time should be equally allocated among all the pending tasks. Another strategy consists of sorting the pending tasks contracts in decreasing order, and then adjusting only the first contracts until the deadline of T_{n+1} can be guaranteed. The basic algorithm assumes that the worst case budget given in time units by the variable "const", is exactly the same for all tasks. An obvious improvement consists of associating a time budget related to the task's importance.

Basic Contracts Adjustment Algorithm on Multiprocessor Systems

Suppose now a multiprocessor system built around K processors. k tasks $T_1, T_2, ..., T_k$ (k<=K) are running, and there are n-k tasks $T_{k+1}, T_{k+2}, ..., T_n$ pending into a FIFO queue. Upon arrival of the T_{n+1} task, the expected resources overload in time units is computed if and only if k=K. It is given by:

$$\Delta = \min_{i=1}^{k}(\Delta T_i) + \sum_{i=k+1}^{n} \Delta T_i - \Delta T_{n+1}$$

If k<K, a processor is free and T_{n+1} is immediately started. When resources are not overloaded, ΔT_{n+1} is updated as previously described in the single processor algorithm. Otherwise, each pending task contract is expected to be diminished by Γ time units, given by:

$$\Gamma = \frac{\Delta + \mathbf{const}}{\mathbf{n - k + 1}}$$

The loop body at step 4 of the single processor algorithm, is then restricted to the pending tasks and the running task m such as:

$$\Delta T_m = \min_{i=1}^{k}(\Delta T_i)$$

When a task T_c has completed (1<=c<=k) at time t, the contract of the ready task T_{k+1} is updated once again as above described, and then the ready task is activated on the processor c. Allocation of tasks to processors follows then a self-scheduled policy. When a resource-level scheduler controls a multiprocessor, an abort strategy may be suggested: whenever the resources are overloaded, the scheduler selects the running task T_c whose quality Q_c (1<=c<=k) of the last returned result so far, is maximum. T_c is then immediately discarded if Q_c is greater than a given quality threshold.

Priority-Driven Non Preemptive Algorithms

The basic adjustment algorithm we have discussed, assumes that tasks activation calls are serviced in a FIFO order. It is not optimal when the contracts variance is high. A best strategy begins by sorting the pending tasks contracts according to a given criterion. In this section we briefly describe the underlying ideas of the algorithm, when applied on a single processor system.

Assume a list of n+1 pending tasks activation requests $T_1, T_2, ..., T_{k-1}, T_k, T_{k+1}, ..., T_{n+1}$ with the corresponding fixed-contracts $\Delta T_1, \Delta T_2, ..., \Delta t_{n+1}$. This list is obtained by inserting the current task activation request T_k with contract ΔT_k, into the sorted queue in an "earliest deadline first" order. Suppose we have at time t, the following contracts queue

(5, 8, 6, 1, 3)

corresponding to five pending tasks. The expected deadlines at time t, are given by the sorted list

(t+5, t+13, t+19, t+20, t+23)

These deadlines are to be considered as worst-case deadlines since T_1 is already running at time t. Suppose now that the resource scheduler receives at time t an activation request for a task **T** with a contract ΔT, and a deadline $t+\Delta T$. Inserting the task **T** in position k in an "earliest deadline first" order, implies that before insertion, we have the following relation

$$\sum_{i=1}^{k-1} \Delta T_i \le \Delta T < \sum_{i=1}^{k} \Delta T_i$$

For example, if $\Delta T=7$, **T** will be inserted in position 2. This strategy implies that the timing overload from the new request point of view, is given by

$$\Delta = \sum_{i=1}^{k-1} \Delta T_i - \Delta T \le 0$$

- When $\Delta=0$ occurs, the current task **T** has the same deadline as a pending task T_i, and has to be inserted in position k=i+1. For example, if $\Delta T=13$, task **T** has to be inserted in position 3, and the contract queue becomes

(5, 8, 13, 6, 1, 3)

The deadline of **T** cannot then be guaranteed without adjusting the contracts of tasks T_1 and T_2. This is performed according to the basic contracts adjustment algorithm previously discussed, and improved in the following way. In the basic version of the algorithm, an incoming task is discarded whenever $\Delta=0$. An improved mechanism consists in preventing to ignore a pending request whenever is possible. Several solutions are then offered. An obvious

one gives to the incoming task **T** a contract equal to (k-1) and diminishes the contracts of the tasks T_1 to T_{k-1} by one unit of time. Applying this straightforward solution, the contracts queue becomes

(4, 7, 2, 6, 1, 3)

and the deadlines queue is given by

(t+4, t+11, t+13, t+19, t+20, t+23)

Since ΔT_3 has been adjusted such as its deadline t+13 will be guaranteed, the contracts associated with the following tasks T_4 to T_6, have not to be modified.

- When $\Delta<0$, the deadline of **T** can be guaranteed by updating its contract through the equation

$$\Delta T' = (-\Delta)$$

For example, if $\Delta T=15$, task **T** has to be inserted in position 3 and $\Delta T'=2$. The contracts queue becomes then

(5, 8, 2, 6, 1, 3)

Inserting a new contract in position k prevents the deadlines of the tasks T_{k+1} to T_{n+1} to be guaranteed. A straightforward solution consists in updating ΔT_{k+1} in the following way

$$\Delta T'_{k+1} = \Delta T_{k+1} - \Delta T'$$

Remember that inserting the task **T** in position k ensures that

$$\sum_{i=1}^{k-1} \Delta T_i < \Delta T < \sum_{i=1}^{k} \Delta T_i$$

implying that

$$\Delta T - \sum_{i=1}^{k-1} \Delta T_i < \Delta T_k$$

and that

$$\Delta T' = (-\Delta) < \Delta T_k$$

Once the task **T** has been inserted in position k, the previous ΔT_k becomes ΔT_{k+1}, and then

$$\Delta T'_{k+1} = \Delta T_{k+1} - \Delta T' > 0$$

Updating the contract of task T_4 in the previous example, gives the following contracts queue

(5, 8, 2, 4, 1, 3)

and all the contracts will be guaranteed. Consequently, when a task **T** is inserted into the queue in position k, the adjusted contracts of the following tasks are always greater than 0, and then are never discarded. The proposed algorithm is described in the general case, by the figure 3.

1. Find the correct position of the incoming task T such as $\sum_{i=1}^{k-1} \Delta T_i \le \Delta T < \sum_{i=1}^{k} \Delta T_i$

2. Compute $\Delta = \sum_{i=1}^{k-1} \Delta T_i - \Delta T$

3. If $\Delta = 0$ *then*

3.1. Adjust the contracts of task T_1 *to* T_{k-1}

4. Else

4.1. Adjust the contract $\Delta T' = (-\Delta)$

4.2. Adjust the contract $\Delta T'_{k+1} = \Delta T_{k+1} - \Delta T'$

5. EndIf

Figure 3: Adjusting the contracts in an "earliest deadline first" order

When this algorithm is applied, a particular case occurs if the incoming task **T** has to be inserted at time t, in position 1, with a contract ΔT. This case occurs when $\Delta T<\Delta T_1$. Remember that the key point of the scheduling approach is to avoid a running task to be preempted. Consequently, the deadlines queue is only updated upon completion or interruption of the running task T_1. To solve this problem, the scheduler maintains the starting time t_{start} of the running task T_1. The scheduling decision depends then on the value of

$$\Delta = \Delta T_1 - (t - t_{start})$$

- If $\Delta<\Delta T$, the scheduler updates the contract of the task **T** in the following way

$$\Delta T' = \Delta T - \Delta$$

The task **T** is then inserted in position 2 and the contract of task T_3 is updated in order to guarantee the deadlines of the following pending tasks.

- Otherwise, the deadline of **T** will not be guaranteed without stopping the running task. Such a case is the most interesting one, and may be solved through the knowledge of the quality Q of the last approximate result produced so far by T_1. For example, if Q is greater than a given threshold, the scheduler may decide to abort T_1 and to immediately start **T**. Otherwise, the incoming task **T** will be ignored.

Conclusions and Outstanding Issues

Anytime algorithms are being used increasingly for time-critical problem-solving in domains such as planning, database query processing, scheduling and others. This

short paper discusses a cooperative scheduling approach between meta-level flows of control and resource-level schedulers. Several run-time monitoring issues based on contracts adjustment algorithms, have been examined. This framework is suitable when original fixed-contracts may be estimated at run-time and when quality of approximate solutions returned by the anytime components cannot be accurately measured at compile-time. Adjusting fixed-contracts according to run-time workload, facilitates the design of machine-independent problem-solving systems, that can automatically adjust time allocation to achieve graceful performance degradation. The main advantage in monitoring through contracts adjustments lies in the fact that real-time preemptive scheduling is not necessary to meet timing requirements. Therefore, both context switching and protocols to protect the consistency of shared data are avoided. However, how guaranteeing the proper use of non-preemptable shared resources, remains a key problem since interruptible processes are expected to be aborted at any time. Ongoing work addresses this aspect of anytime computing. Further works include investigating extended scheduling protocols when weighted tasks and shared data are required, and developing language supports and run-time system facilities for activating and interacting with fixed-contracts interruptible algorithms.

References

Boddy, M., and Dean, T. 1994
Deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence 67, pp. 245-285, 1994*

Garvey, A.; Humphrey, M.; and Lesser, V. 1993
Task interdependencies in design-to-time real-time scheduling. In Proceedings of the Eleventh National Conference on Artificial Intelligence, 1993

Grass, J,. and Zilberstein, S. 1996
Anytime algorithms development tools. *SIGART Bulletin, Special Issue on Anytime Algorithms and Deliberation Scheduling, Vol. 7, N° 2, 1996*

Hansen, E. A., and Zilberstein, S. 1996
Monitoring the progress of anytime problem solving. In Proceedings of the Thirteen National Conference on Artificial Intelligence. Portland, Oregon, 1996

Horvitz, E. J. 1990
Computation and action under bounded ressources. Ph.D. Dissertation, Department of Computer Science and Medecine, Stanford University, December 1990

Liu, J. W. S.; Lin, K. J.; Shih, W. K.; Yu, A. C.; Chung, J. Y.; and Zhao, W. 1991
Algorithms for scheduling imprecise computations. *IEEE Computer 24(5):58-68, May 1991*

Mensch, A., and Charpillet, F. 1996
Scheduling in the REAKT Kernel: Combining predictable and unbounded computations for maximising solution quality in real-time knowledge-based systems. In Proceedings of Real-Time Systems and Embedded Systems, R.T.S. & E.S.'96. Teknea, January 1996

Russel, S. J, and Zilberstein, S. 1991
Composing real-time systems. In Proceedings of the Twelfth International Joint Conference on Artificial Intelligence. Sydney, Australia, 1991

Zilberstein, S. 1993
Operational rationality through compilation of anytime algorithms. Ph.D. Dissertation, Computer Science Division, University of California at Berkeley, 1993

Zilberstein, S. 1996
Using anytime algorithms in intelligent systems. *AI Magazine, 17(3):73-83, Fall 1996*