ANALYSIS OF OVERRUN STRATEGIES IN PERIODIC CONTROL TASKS

Anton Cervin

Department of Automatic Control, Lund University,
Box 118, SE-221 00 Lund, Sweden
anton@control.lth.se

Abstract: The paper considers period overruns in control tasks with variations in execution time (or response time). A simple model is used, where the input and output operations are assumed to be time-triggered, and the execution-time distribution of the task is assumed to be known. Three overrun strategies, called Abort, Skip, and Queue, are modeled as discrete-time jump linear systems and are analyzed with regard to control performance. The analysis is exemplified on an integrator process. It is argued that the Skip strategy has good performance, is simple to analyze, and is easy to implement in real-time operating systems. Some simple extensions are also considered.

Keywords: computer control, embedded systems, execution times

1. INTRODUCTION

In embedded control systems, the computational resources are limited and must be used as efficiently as possible. In such systems, it may not always be feasible to use high sampling rates and to base the real-time system design on worst-case execution times.

In this paper, we study controllers that experience variations in their execution times from sample to sample. If the control task has not finished its execution by the end of the sampling period, an *overrun* is said to have occurred. The overrun can be viewed as an exception that must be handled by the real-time operating system and by the controller. The variations in execution time may stem from the control application itself, from the real-time system, or from the computing hardware. Examples in the first category include discrete logic and data-dependencies in the control algorithm. In the second category, we find preemption from higher-priority tasks and interrupts [1]. The third category includes hardware effects such as cache misses.

[1] The total execution time, including preemption from higher-priority tasks, is usually referred to as the *response time* of the task.

At the heart of the problem is a trade-off between the sampling period and the probability and consequence of overruns. Using conventional notation from real-time systems, the CPU utilization U of a task is given by

$$U = \frac{C}{T} \tag{1}$$

where C is the worst-case execution time of the task, and T is the task period. It is seen that, for a given value of U, a large enough T must be chosen to accommodate the largest possible execution time. It is well known that an overly long sampling period leads to degraded control performance. Hence, it can be tempting to choose a smaller T than what is dictated by (1). The penalty that must be payed is that of possible execution overruns. If the performance loss due to the overruns is smaller than the performance gain due to the shorter sampling period, then such a design could be considered to be "better" than a classical worst-case design.

The influence of the sampling interval on the control performance is relatively easy to understand and to compute. The consequence of execution overruns is considerably more difficult to predict. The result de-

pends on a large number of factors, including how the I/O is performed, the basic scheduling algorithm used in the operating system, the specific overrun handling method used, the execution-time characteristics of the control task, the controller and plant dynamics, and whether or not the controller can compensate for overruns. Hence, it may not be possible to devise an overrun handling method that is "the best" for all control applications.

To allow for some control analysis, in this paper we will consider a very simple model of a real-time control system. The model is based on the seminal paper on real-time scheduling theory, (Liu and Layland, 1973). The input and output operations of the controller are assumed to be time-triggered and synchronized such that the reading and writing of measurement and control signals occur at the same time. Hence, assuming non-zero computation times, there will be a computational delay of at least one sample in the feedback loop. Furthermore, it is assumed that the execution time of the control algorithm in successive periods is independent and can be described by a stochastic variable with a known distribution. These assumptions will allow us to model the real-time control system as a jump linear system. The performance of the system (as measured by a quadratic cost function) can then be evaluated for different overrun handling strategies.

1.1 Related work

The study of execution overruns is closely related to the analysis of control systems with random delays. Systems with random sampling and random delays are modeled as jump linear systems in (Krasovskii and Lidskii, 1961). Discrete-time jump linear systems are treated in e.g. (Ji *et al.*, 1991). Control systems with random delays and skips are also treated in (Davidson, 1973). Linear-quadratic analysis and control of systems with random network delays are studied in (Nilsson *et al.*, 1998). Jitterbug (Lincoln and Cervin, 2002) is a MATLAB-based toolbox that allows the evaluation of a quadratic cost function for a control system with aperiodic sampling, skips, etc.

Skips and overruns have been studied quite extensively in the real-time literature. Scheduling of systems that allow skips is treated in (Koren and Shasha, 1995) and (Ramanathan, 1997). The latter paper considers scheduling that guarantees that at least k out of n instantiations will execute. A slightly different motivation for skipping samples is presented in (Caccamo and Buttazzo, 1997). Here the main objective is to use the obtained execution time to enhance the responsiveness of aperiodic tasks.

The constant bandwidth server (Abeni and Buttazzo, 1998) is a scheduling mechanisms for soft and/or aperiodic tasks. When a task has an overrun, the deadline is postponed by a period so that the schedulability of other tasks are not jeopardized. A variant of the constant-bandwidth server specifically designed to handle overruns in real-time control systems is presented in (Caccamo *et al.*, 2002). The proposed server, called CBS^{hd}, differs from the original CBS by postponing the deadline only by the amount needed to complete the job. In this way, the task can be scheduled more efficiently and finish earlier.

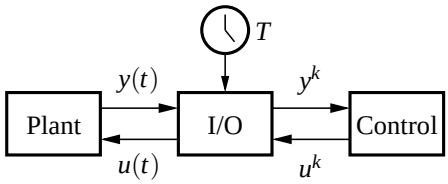

Fig. 1. The system model assumed in this paper. The I/O is time-triggered with the period T.

1.2 Outline

In Section 2, the system model is given, and three different overrun handling strategies are suggested. The strategies are modeled as Markov chains, allowing the closed-loop system to be described as a jump linear system. In Section 3, the analysis is exemplified on an integrator process, and in Section 4, some extensions and relationships to real-time scheduling algorithms are discussed. Finally, Section 5 contains the conclusions.

2. OVERRUN STRATEGIES

2.1 System model and performance evaluation

The real-time control system assumed in this paper is shown in Fig. 1. The plant and the control computer are interfaced by a time-triggered I/O unit that takes measurement samples with from the plant and forwards control signals (using zero-order hold) from the controller, both with the period T.

Each measurement sample generates a job (task instance) in the computer. If the execution time of the job is larger than the sampling period, an overrun occurs and the control signal will not be updated in the next sampling interval.

The plant is described by a linear continuous-time system $P(s)$, and the controller is described by a linear discrete-time system, $C(z)$. The controller is typically designed taking the one-sample delay in the feedback path into account. Both systems are assumed to be disturbed by white noise processes.

The execution time (or response time) of the control algorithm is described by a probability density function $f_c(x)$. An example of an execution-time distribution is shown in Fig. 2. Here, the execution time varies between a common, lower, nominal value c_{nom} and an upper, maximum value c_{max}. This could model a task

which experiences occasional cache misses. In a real system, the execution times from sample to sample are typically not independent. Another modeling problem is that it can be very hard to find the maximum execution time.

Below, we describe three different overrun handling strategies; Abort, Skip, and Queue; and show how they can be modeled using Markov chains. Also, the implementation of the strategies in the real-time system are discussed.

In our models, a Markov chain makes two transitions every period T. Each transition takes $T/2$ seconds, during which the continuous-time dynamics of the plant evolve. When a node is reached, a discrete-time system (the I/O or the controller) associated with the node may be updated. Sampling the plant with the interval $T/2$, the closed-loop system can then be written as a discrete-time jump linear system,

$$x(k+1) = \Phi_n x(k) + \Gamma_n v(k) \qquad (2)$$

where the state vector x collects the plant, controller, and I/O states, v is a discrete-time white noise process, and the transition matrix Φ_n and the input matrix Γ_n depend on the current Markov state n.

Given a Markov chain, it is straightforward to compute the stationary covariance of x by iteration and to evaluate a quadratic cost function for the system. For the calculations in this paper, we have used the Jitterbug toolbox (Lincoln and Cervin, 2002).

2.2 The Abort strategy

From a real-time perspective, a period overrun can be viewed as a missed deadline. If the real-time operating system supports the monitoring of deadlines, an exception can be generated when the overrun occurs, killing the job. This is referred to as the *Abort* strategy. An illustration of the strategy is given in Figure 3. Aborting the current job means that no new control signal will be produced this period, but, at the same time, the next job will have a better chance of finishing before its deadline.

The Abort strategy can be modeled by a simple Markov chain as shown in Fig. 4. The I/O system is updated at the beginning of each period. Then, with a probability of $p = \int_0^T f_c(x)dx$, the job will finish before the period, causing the controller C to be executed. If the controller has executed, the control signal

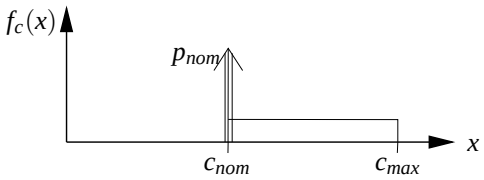

Fig. 2. Example of a probability density function describing the execution-time distribution of a control task.

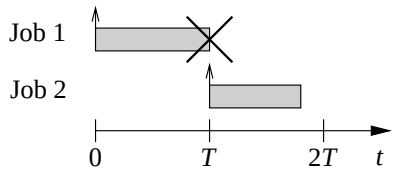

Fig. 3. The Abort strategy.

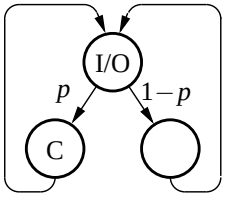

Fig. 4. Markov chain for the Abort strategy.

```
t := StartTime() + T;
loop
  select
    delay until t;
  then abort
    Read_input();
    Compute_control();
    Write_output();
  end
  t := t + T;
  delay until t;
end
```

Listing 1. Pseudo-code for implementation of the Abort strategy.

will be delivered to the plant by the I/O at the beginning of the next period.

Although simple to model, the Abort strategy may be difficult to implement. Some languages, such as Ada (e.g., (Burns and Wellings, 2001)) and Real-Time Java (RTSJ, (Bollella *et al.*, 2000)) have explicit support for timeouts and asynchronous transfer of control (program flow). Listing 1 shows the pseudo-code for the Abort strategy (in an Ada-like language).

Generally, however, operating systems do not support asynchronous transfer. One alternative is to insert extra checkpoints in the code. Here, the clock can be read and it can be determined whether the task is late, causing a branch to the end of the loop. Another possibility is to use a timer and to lower the priority (or equivalent) of the late task, letting it run in the background until it has finished. Meanwhile, a new task, taken from a task pool, is used to execute the next job. Such a scheme introduces additional overhead. Care must also be taken such that the task data is in a consistent state throughout.

2.3 The Skip strategy

In the Skip strategy, subsequent jobs and samples are skipped as long as the current instance has not completed. The strategy is illustrated in Figure 3. Contrasted to the Abort method, the Skip strategy

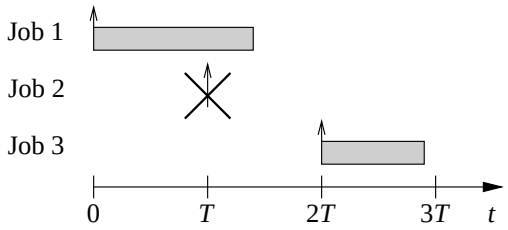

Fig. 5. The Skip strategy. When the first job overruns, the second job is skipped.

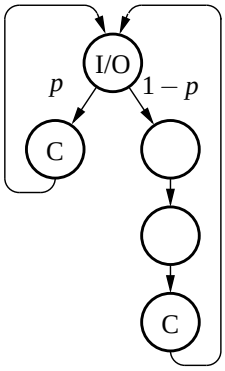

Fig. 6. Markov chain for the Skip strategy, assuming $c_{max} \leq 2T$.

makes certain that new control signals are eventually delivered to the plant. If the overrun covers several periods, many jobs may have to be skipped, however.

The size of the associated Markov chain depends on the ratio of the maximum execution time c_{max} and the period T. The chain for the case $c_{max} \leq 2T$ is shown in Fig. 6. With a probability $p = \int_0^T f_c(x)dx$, the output will be delayed only one period, and in the other case the output will be delayed two periods. For longer maximum execution times, more branches must be added to the chain.

The Skip strategy is very simple to implement in most real-time systems. The pseudo-code is given in Listing 2. The task must keep track of its own time-base, i.e., when it was first released. When a job finishes, it can read the system clock to find out whether it is late or not. It can then set a flag to indicate to the next job whether it should execute the control algorithm.

2.4 The Queue strategy

The Queue strategy can be said to be the default implementation in systems where overruns are not considered. The strategy is illustrated in Figure 7. When an overrun occurs, the following job is queued and can start once the first instance completes.

Allowing the first job to complete, the second job will be delayed, introducing extra input-output latency. Also, since the second job is released late, it is less likely to complete before the third job is released.

```
t := TimeBase() + T;
late := false;
loop
  if not late then
    Read_input();
    Compute_control();
    Write_output();
  end;
  t := t + T;
  if Clock() < t
    delay until t;
    late = false;
  else
    late = true;
  end
end loop
```

Listing 2. Pseudo-code for implementation of the Skip strategy.

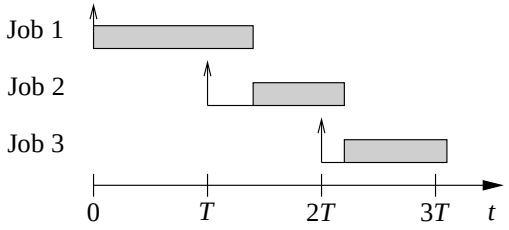

Fig. 7. The Queue strategy. The second job is queued and can start once the first job completes.

Note that, if several long execution times occur in a row, a long queue of jobs may build up in the RTOS. This can lead to very poor control performance. An alternative is to queue only the most recent job. This modified strategy will be referred to as Queue(1).

Modeling the Queue strategy as a Markov chain is more difficult than the other cases. If an arbitrary number of jobs may be queued, the chain will be infinite and must be truncated at some point. Furthermore, the state of the task, i.e., the amount of completed execution time of the job, must be stored between periods. Since this state is continuous in time, it must be discretized.

A Markov chain for the Queue(1) strategy for the case $c_{max} \leq 2T$ is shown in Fig. 8. The left-most node corresponds to the case where there is no queue in the system and the job finishes before the next period. Each branch in the right-hand side corresponds to an overrun of some length $k\delta$, where δ is the chosen discretization interval for the queue length. Since the I/O unit always executes periodically, it may also be necessary to store old inputs (representing an input read by the control task in the previous period). For this purpose, in the model, the I/O delivers two samples; the current and the previous one. The execution of C1 represents the reading of the latest sample, while the execution of C2 represents reading the old sample.

The implementation of the plain Queue strategy is trivial in most systems, since the overrun exceptions are not really handled. The pseudo-code is given in

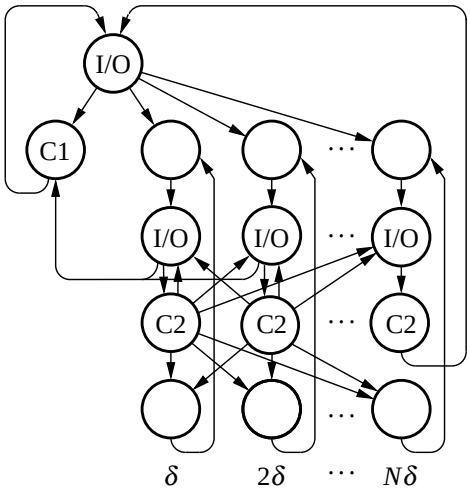

Fig. 8. Markov chain for the Queue(1) strategy, assuming $c_{max} \leq 2T$.

```
t := TimeBase() + T;
loop
  Read_input();
  Compute_control();
  Write_output();
  t := t + T;
  delay until t;
end loop
```

Listing 3. Pseudo-code for implementation of the Queue strategy.

Listing 3. The modified Queue(1) strategy may be implemented using a technique similar to Listing 2.

3. EXAMPLE

The analysis is exemplified on an integrator process,

$$P(s) = \frac{1}{s}$$

which is assumed to be disturbed by a white noise process with unit incremental variance. The controller is designed to minimize the stationary cost function

$$J = \lim_{t\to\infty} \frac{1}{t} \int_0^t y^2(\tau)\, d\tau$$

i.e., the controller is a minimum-variance controller (Åström and Wittenmark, 1997). The sampling interval T and a computational delay of T is assumed in the design.

An execution-time distribution is assumed according to Figure 2, where $p = 0.8$, $c_{nom} = 1$, and $c_{max} = 2$. The performance for each strategy is computed for different sampling intervals between $T = 1$ and $T = 2$. Note that the case $T = 2$ corresponds to a worst-case design, where no overruns are possible. In this case, the cost can be computed to be $\sqrt{3}/3 + 3$. For the Queue strategy, the queue length was discretized with step size of $\delta = T/10$. The results are displayed in Figure 9, where the cost J is plotted as a function of the sampling period T.

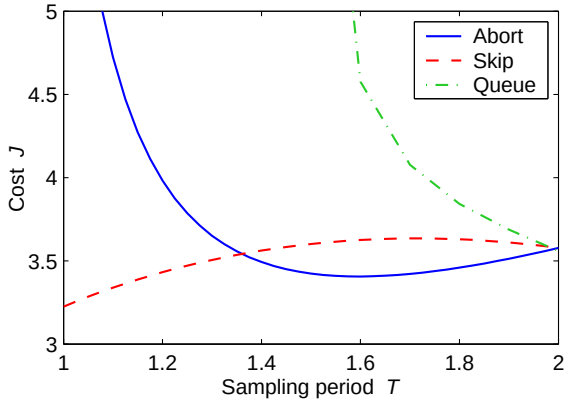

Fig. 9. Comparison of costs in different overrun handling strategies.

The cost of the Abort strategy initially decreases as the period is decreased from c_{max}. This is due to the shorter sampling interval used. As the period becomes shorter, however, the cost increases rapidly due to the larger number of missed outputs. The minimum cost of $J = 3.4$ is obtained for $T = 1.6$.

For the Skip strategy, the results are reversed—the cost initially increases due to occasional skips, but then decreases again as the period becomes shorter. The minimum cost $J = 3.2$ is obtained for the shortest sampling interval, $T = 1$.

For the Queue strategy, the cost increases monotonically with shorter sampling periods. The increase is due to the successive long delays caused by the queuing of jobs. The minimum cost, $J = 3.6$, is obtained for the longest sampling period, $T = 2$.

In this example, the Skip strategy has the best overall performance, assuming that the period can be chosen freely. In some cases, however, the sampling period may be dictated by the application (consider for instance a camera delivering images at a fixed rate), and then the Abort strategy may give better performance. For time-triggered inputs and outputs, the queue strategy does not seem to work very well. This is due to the domino effect that causes repeated missed outputs.

4. EXTENSIONS

The analysis presented in this paper are based on two simplifying assumptions: that the execution times are independent between periods, and that the I/O is time-triggered. Removing either of these assumptions makes the analysis much harder. Having execution-time dependencies between jobs requires the addition of another dimension to the Markov chains. Removing the I/O points, one must consider both sampling jitter and output jitter in the analysis, and these depend on the scheduling policy used, etc.

One case that is easy to model, however, is a model that could be called “Queue–Shift”. The idea is the

```
t := TimeBase() + T;
loop
  Read_input();
  Compute_control();
  Write_output();
  if Clock() < t
    delay until t;
  else
    t := Clock() + T;
  end
end loop
```

Listing 4. Pseudo-code for implementation of the Queue–Shift strategy.

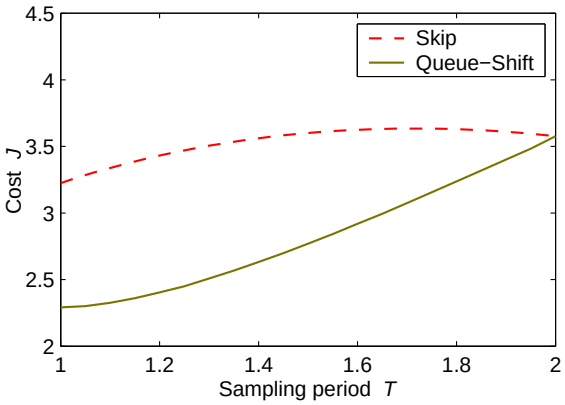

Fig. 10. Comparison of the Skip and Queue–Shift strategies.

following. When a job overruns, the following I/O operations and jobs are shifted by the same amount as the overrun. This of course requires that the I/O unit can be reprogrammed on the fly. The task implementation for this strategy is simple, see Listing. 4.

Due to the minimization of delays in the case of overruns, the Queue–Shift strategy is expected to outperform the other strategies. Fig. 10 compares the cost to the Skip strategy. The Queue–Shift strategy performs better throughout (except, of course, for $T = 2$.)

It is also natural to consider control algorithms that compensate for overruns. For instance, in the case of state feedback from an observer, the Kalman filter can be modified so that it considers the number of samples the previous control signal has been active.

5. CONCLUSION

The problem of overruns due to varying control task execution times has been investigated. Theoretical analysis of simple models has shown that the performance can be improved by choosing shorter sampling periods that what is dictated by worst-case considerations. Of the three basic overrun methods investigated in the example, the Skip strategy seems to be the most robust one. It is also very simple to implement in existing real-time systems.

ACKNOWLEDGMENTS

This work has been sponsored by the EU/IST projects ARTIST and ARTIST2.

REFERENCES

Abeni, Luca and Giorgio Buttazzo (1998). Integrating multimedia applications in hard real-time systems. In: *Proc. 19th IEEE Real-Time Systems Symposium*. Madrid, Spain.

Åström, Karl Johan and Björn Wittenmark (1997). *Computer-Controlled Systems*. Prentice Hall.

Bollella, G., B. Brosgol, P. Dibble, S. Furr, J. Gosling, D. Hardin and M. Turnbull (2000). *The Real-Time Specification for Java*. Addison-Wesley.

Burns, Alan and Andy Wellings (2001). *Real-Time Systems and Programming Languages*. 3rd ed.. Addison-Wesley.

Caccamo, M. and G. Buttazzo (1997). Exploiting skips in periodic tasks for enhancing aperiodic responsiveness. In: *Proc. 18th IEEE Real-Time System Symposium*.

Caccamo, Marco, Giorgio Buttazzo and Lui Sha (2002). Handling execution overruns in hard real-time control systems. *IEEE Transactions on Computers*.

Davidson, Charles (1973). Random sampling and random delays in optimal control systems. PhD thesis. Department of Optimization and Systems Theory, Royal Institute of Technology, Sweden.

Ji, Y., H.J. Chizeck, X. Feng and K.A. Loparo (1991). Stability and control of discrete-time jump linear systems. *Control-Theory and Advanced Applications* **7**(2), 247–270.

Koren, G. and D. Shasha (1995). Skip-over: Algorithms and complexity for overloaded systems that allow skips. In: *Proc. IEEE Real-Time Systems Symposium*.

Krasovskii, N.N. and E.A. Lidskii (1961). Analytic design of controllers in systems with random attributes, I, II, III.. *Automation and Remote Control* **22**(9–11), 1021–1025, 1141–1146, 1289–1294.

Lincoln, Bo and Anton Cervin (2002). Jitterbug: A tool for analysis of real-time control performance. In: *Proceedings of the 41st IEEE Conference on Decision and Control*. Las Vegas, NV.

Liu, C. L. and J. W. Layland (1973). Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM* **20**(1), 40–61.

Nilsson, Johan, Bo Bernhardsson and Björn Wittenmark (1998). Stochastic analysis and control of real-time systems with random time delays. *Automatica* **34**(1), 57–64.

Ramanathan, P. (1997). Graceful degradation in real-time control application using (m,k)-firm guarantee. In: *Proc. 27th Annual International Symposium on Fault-Tolerant Computing*.