Feedback Scheduling of Model Predictive Controllers

Henriksson, Dan; Cervin, Anton; Åkesson, Johan; Årzén, Karl-Erik

Published in:
Proceedings of the 8th IEEE Real-Time and Embedded Technology and Applications Symposium

DOI:
10.1109/RTTAS.2002.1137395

2002

Feedback Scheduling of Model Predictive Controllers

Dan Henriksson, Anton Cervin, Johan Åkesson, Karl-Erik Årzén
Department of Automatic Control
Lund Institute of Technology
Box 118, SE-221 00 Lund, Sweden
{dan,anton,jakesson,karlerik}@control.lth.se

Abstract

The paper presents some preliminary results on dynamic scheduling of model predictive controllers (MPCs). In an MPC, the control signal is obtained by on-line optimization of a cost function, and the MPC task may experience very large variations in execution time from sample to sample. Unique to this application, the cost function offers an explicit, on-line quality-of-service measure for the task. Based on this insight, a feedback scheduling strategy for multiple MPCs is proposed, where the scheduler allocates CPU time to the tasks according to the current values of the cost functions. Since the MPC algorithm is iterative, the feedback scheduler may also abort a task prematurely to avoid excessive input-output latency. A case study is presented, where the new approach is compared to conventional fixed-priority and earliest-deadline-first scheduling. General problems related to the real-time implementation of MPCs are also discussed.

1. Introduction

Flexible scheduling of tasks with unknown or varying execution time has attracted considerable attention in the real-time research community during the last decade. The main motivation has been to reduce some of the pessimism that is inevitable when applying traditional real-time scheduling theory to such tasks. The research has resulted in a number of general approaches, such as scheduling of imprecise computations [10], statistical scheduling, e.g. [18], and value-based scheduling, e.g. [4].

In this work, we instead take the application as the starting point, focusing on scheduling of model predictive controllers (MPCs). In an MPC, the control signal is determined by on-line optimization of a cost function in every sample. The optimization problem is solved iteratively, with highly varying execution time depending on a number of factors: the state of the plant, the current and future reference values, the disturbances acting on the plant, the number of active constraints on control signals and outputs, etc.

The MPC strategy, see e.g. [6, 13], has won widespread industrial use in recent years, the main advantages being its ability to handle constraints and its straightforward applicability to large, multi-variable processes. However, because of the computational demands of the control algorithm, MPC has traditionally only been applied to plants in the process industry, with slow dynamics and low requirements on fast sampling.

The industrial practice has been to run the MPC algorithm on a dedicated computer, and to decrease the complexity of the problem so that overruns are avoided. The problem of scheduling the MPC as a real-time task has not been adequately studied, and scheduling of several MPC tasks on the same computer has never been considered. To this end, we here take a first step towards a strategy for dynamic scheduling of MPCs.

One way to handle large variations in execution time dynamically is to introduce feedback in the real-time system. A schematic illustration of a general feedback scheduling system is shown in Figure 1. The idea is to feed back the actual use of critical resources (in our case, CPU time) to the scheduler and to continuously adjust the tasks' demand of resources according to the current situation. In some cases it is also possible to use feedforward, i.e., the tasks can inform the scheduler that they are about to consume more resources.

For control tasks, there are two main ways to control the CPU demand: by manipulating the task periods, or by manipulating the execution times. The first approach has been explored in several papers. In [3], modulation of sampling rates in robot control systems is considered. [14] studies reallocation of control tasks and on-line adjustment of sampling rates in a multi-processor system. In [5], a case study with hybrid controllers is presented, where the sampling rates are adjusted to avoid CPU overloads. [1] considers quality-of-service negotiation in flight control systems, and treats task periods and deadlines as negotiable parameters.

Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02)
0-7695-1739-0/02 $17.00 © 2002 IEEE

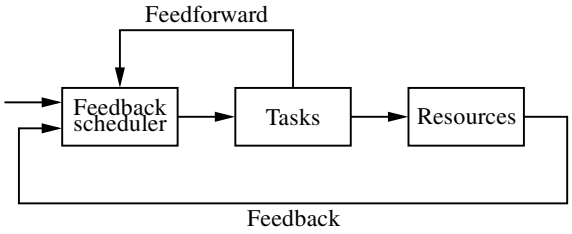

Figure 1. A general feedback scheduling system. The scheduler adjusts the tasks' demands based on feedback from the current use of critical resources. The tasks may also inform the scheduler that they are about to consume more resources (feedforward).

For MPCs, it is more natural to control the CPU demand by manipulating the execution times of the controllers. In the terminology of [10] MPCs can be viewed as "milestone" tasks. In each sample, the quality of the control signal is gradually refined for each iteration in the optimization algorithm, up to a certain bound. This makes it possible to abort the optimization before it has reached the optimum, and still obtain an acceptable control signal. Another nice feature of MPC is that it is not only *possible* to extract a real-world quality-of-service measure from the controller, but the control algorithm is indeed *based* on the same measure. This enables a tight and natural connection between the control and the scheduling.

The main idea proposed in this paper is to use feedback information from the optimization algorithm to determine (a) when to terminate the optimization and output the control signal, and (b) which of several MPCs that should be scheduled at a given time. No papers that the authors are aware of consider adjustment of the execution times of control tasks. Also, none of the papers mentioned above include feedback from the actual performance of the feedback control loops. In the present paper, a simulation study is performed, where the new feedback scheduling approach is compared to conventional fixed-priority and earliest-deadline-first scheduling. The results show that improvements in control performance can be achieved by more dynamic scheduling.

The rest of this paper is outlined as follows. A mathematical description of MPC is given in Section 2, followed by a discussion of issues related to the real-time implementation of MPCs in Section 3. The main ideas for feedback scheduling of MPCs are presented in Section 4. Section 5 contains a case study of an MPC for a quadruple-tank process, illustrating the ideas from Section 4. Section 6 provides some further discussion and Section 7 gives the conclusions.

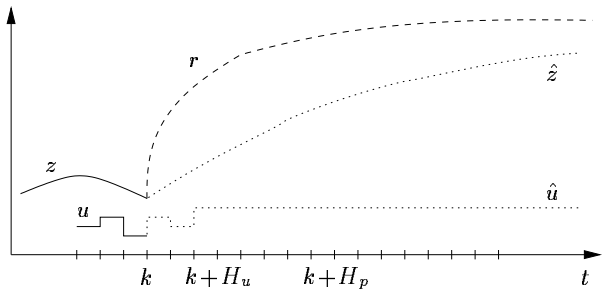

Figure 2. The basic principle of MPC.

2. Model predictive control

This section presents a basic formulation of the model predictive control problem, taken from [11]. The formulation assumes a linearized, discrete-time model of the controlled plant on the form

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= C_y x(k) \\ z(k) &= C_z x(k) \end{aligned} \tag{1}$$

where x is the state vector, u is the vector of control signals, y is the vector of measured outputs, z is the vector of controlled outputs, and A, B, C_y and C_z are constant matrices. Linear constraints may be specified on the outputs as well as on the control signals. The constraints model physical limitations in the plant, such as actuator limitations, rate limitations, level constraints, etc.

The basic principle of MPC is illustrated in Figure 2. The goal is to make the controlled outputs z follow the reference trajectory r as close as possible. In each sample, the predicted outputs $\hat{z}(k+1) \ldots \hat{z}(k+H_p)$ are computed as a function of the current state $x(k)$, the current measurements $y(k)$, and the predicted control signals $\hat{u}(k) \ldots \hat{u}(k+H_u-1)$. H_p is the *prediction horizon*, and H_u ($H_u \leq H_p$) is the *control horizon*. The horizons are chosen in relation to the time constant of the plant under control.

The vector of predicted control signals, $\mathcal{U}^T(k) = \left[\hat{u}(k) \ \ldots \ \hat{u}(k+H_u-1)\right]^T$, constitute the decision variable in an optimization problem. When the optimization problem has been solved, only the first control signal is applied to the process, i.e., $u(k) = \hat{u}(k)$. In the next sampling period, new measurements are taken and the whole procedure is repeated again. Note that the control action $u(k+1)$ computed at time step $k+1$ will generally be different from the predicted control action $\hat{u}(k+1)$ computed at time step k, since more up-to-date information about the process will be available. The predicted control signals from the previous sample can, however, be used as a good initial guess in the current sample.

Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02)
0-7695-1739-0/02 $17.00 © 2002 **IEEE**

The optimization problem to be solved in each sample consists of minimizing a quadratic cost function on the form

$$V(k) = \sum_{i=1}^{H_p} \|\hat{z}(k+i) - r(k+i)\|_Q^2 + \sum_{i=0}^{H_u-1} \|\hat{u}(k+i)\|_R^2 \quad (2)$$

subject to constraints on the control signals and the output signals. The first term in the cost function penalizes deviations from the reference trajectory, while the second term penalizes the control effort. The weighting matrices Q and R constitute tuning parameters which must be adjusted to give satisfactory control performance.

The MPC formulation above leads to a quadratic optimization problem with linear inequality constraints. After some manipulation, the problem can be stated as

$$\begin{aligned} &\text{minimize} \quad \mathcal{U}^T(k)\mathcal{H}\,\mathcal{U}(k) - \mathcal{U}^T(k)\mathcal{G}(k) \\ &\text{subject to} \quad \Omega\,\mathcal{U}(k) \leq \omega(k) \end{aligned} \quad (3)$$

where $\mathcal{H}$ and Ω are constant matrices, and $\mathcal{G}(k)$ and $\omega(k)$ are matrices which depend on the current state. The problem is convex, which means that the optimization algorithm is guaranteed to converge to the optimum and terminate within a finite number of steps.

The number of iterations required for the optimization will be different from sample to sample. Major factors influencing the optimization time are the number of active constraints on control signals and outputs, reference changes, and unmodeled disturbances acting on the plant. The quality of $\mathcal{U}(k-1)$ (the vector of predicted control signals computed in the previous sample) also has a large impact on the optimization time, since it is used as a starting point for the optimization in the current sample.

3. Real-time implementation of MPCs

The mathematical description of MPC above says little about the real-time implementation, i.e., how the controller should be implemented as a periodic task, how sampling and actuation should be performed, what should be done in the event of a task overrun, etc. This is of course a general problem for all control algorithms. The problems do get more pronounced with MPC, however, because of the long and highly varying execution times.

3.1. Periodic sampling

The discrete-time process description (1) assumes that the measurement signals y are sampled with a constant sampling interval. This may not hold if the sampling operation is performed within the task body—a higher-priority task or a long-running previous task instance may delay the start of the task and hence also the sampling operation. Sampling jitter will degrade the control performance and may in extreme cases even lead to instability, see e.g. [19].

Three main approaches are possible. The first approach is to accept the sampling jitter and view it as an additional disturbance that will (hopefully) be regulated by the control system. The second approach is to take the sampling jitter into account in the process description and the control algorithm and try to actively compensate for it. This would lead to a time-varying process and controller structure. The approach would require huge amounts of computations in each sample to set up to optimization problem (3), since all the involved matrices would be time-varying. The third approach is to achieve periodic sampling by performing the sampling operation within a timer interrupt handler or by using a dedicated, high-priority sampling task. The drawback with this approach is the increase in implementation complexity and, in the case of a dedicated sampling task, an increased number of context switches.

3.2. Computational delay and actuation

Computational delay (i.e., the delay from sampling to actuation) is a major problem in MPC. The process description (1) allows for a *constant* computational delay to be specified. This implies that the actuation should be performed at a fixed offset from the sampling in each period. Again, this may not hold, depending on the implementation. Since computational delay degrades the performance of a control loop, a common choice is to output the control signal as soon as possible, i.e., immediately after the computation has finished, see e.g. [2]. Due to preemption from higher-priority tasks and the highly varying execution time of the optimization algorithm itself, this can lead to significant output jitter.

Similar to the case of sampling jitter above, three main approaches are possible. The first approach is to accept the output jitter. This has the advantages of a simple implementation and shorter delays on average. The second approach is to account for the varying delay in the MPC formulation. Again, this makes for a very difficult and time-consuming optimization problem. The third approach is to eliminate the jitter by performing the actuation operation in an interrupt handler or in a dedicated, high-priority actuation task. The actuation operation may in fact be performed together with the sampling operation at the beginning of the next period, thereby enforcing a one-sample delay in the controller. The drawback with this approach is that unnecessarily long delays are introduced in the control loop.

3.3. Overruns

The execution time of the MPC algorithm is bounded, since the quadratic programming problem is convex. Hence, in theory, it would be possible to design the real-time system such that all deadlines are met (under the common assumption deadline = period). Basing the design on the worst-case execution time may not be feasible, however. The choice of sampling period is dictated by the plant dynamics and the desired closed-loop performance, see e.g. [2], and too slow sampling leads to degraded control performance. Also, since the average execution time of the optimization algorithm is significantly shorter than the worst-case execution time, it may simply be a waste of computing resources to require all deadlines to be met.

Knowing that task overruns are possible, proper care must be taken in the implementation. One way to deal with overruns is to abort the optimization at the deadline and output the (non-optimal) control signal that has been computed so far. This approach has the advantage of not delaying the next task instance (and, depending on the implementation of the sampling, also the next sampling instance). Another interesting possibility is to let the optimization continue for yet some time in the next sampling interval, hence stealing time from the next task instance. The intuition behind this approach is that it may pay off in the future to do a really good job now. Spending extra time on a difficult optimization problem in the current sample will mean that good starting values for the optimization will be available in the next couple of samples. It is important to keep the overrun bounded, however, not to endanger the stability of the control system. See e.g. [16], where the effects of a "one-shot" (isolated) overrun are analyzed. More issues related to stability are discussed below.

3.4. Stability concerns

MPC was used in industry several years before formal stability results were available [6]. Stability was not a critical issue, since MPC was typically used for high-level control of plants with slow and stable dynamics. In recent years, several ways to guarantee stability for linear MPC have been proposed, see e.g. [12]. For example, one can include a terminal constraint in the optimization problem, i.e. to require that

$$\hat{z}(k + H_p) = r(k + H_p) \tag{4}$$

Most stability results assume that the optimization is allowed to complete in each sample. This can be problematic in a real-time implementation where the computing resources are scarce. An interesting result regarding suboptimal MPC is given in [17], where it is shown that feasibility of the solution rather than optimality is a sufficient condition for stability, provided that a suitable stability constraint (such as (4) above) is added in the MPC formulation. This means that it is permissible to abort the optimization as long as a feasible solution has been found.

None of the existing stability results for MPC handles a time-varying computational delay, however. Stability results for ordinary computer-controlled systems with random delays, see e.g. [9, 15], are unfortunately not directly applicable to MPC (where the feedback is implicit in the receding-horizon strategy). This is an area where further research is needed.

4. Feedback scheduling of MPCs

In this section, we consider feedback scheduling of first one and then several MPCs. The feedback information used for scheduling decisions (c.f. Figure 1) consists of the values of the cost functions (2) for the different controllers. This information is used to determine

1. for each controller, when to terminate the optimization and output the control signal, and
2. which of several ready controllers that should be scheduled for execution.

The CPU demand is adjusted by manipulation of the execution time of the controllers. This is the natural approach for MPCs, being "milestone" tasks. On-line changes of the sampling intervals are unrealistic, due to the associated computational overhead. The standard MPC formulation in Section 2 assumes constant sampling intervals, and on-line changes of the sampling periods would require, among other things, time-consuming re-computation of the matrices $\mathcal{H}$ and Ω in the optimization problem (3) in each sample.

Throughout, we will assume that the sampling operations of each controller are performed in a timer interrupt handler or in a dedicated, high-priority task. This way, there will not be any sampling jitter, even in the case of task overruns. Furthermore, we will assume that the control signals are actuated as soon as the task has finished (or has been aborted).

4.1. Scheduling of a single MPC

Even scheduling of a single MPC task is nontrivial. The fundamental question is, when does it pay off to optimize further? Figure 3 shows how the cost function (2) decreases for each iteration during a typical optimization run. While the objective function decreases with each new iteration, it is also expected that the true cost will eventually start to increase because of the latency. This is illustrated schematically by the dashed curve.

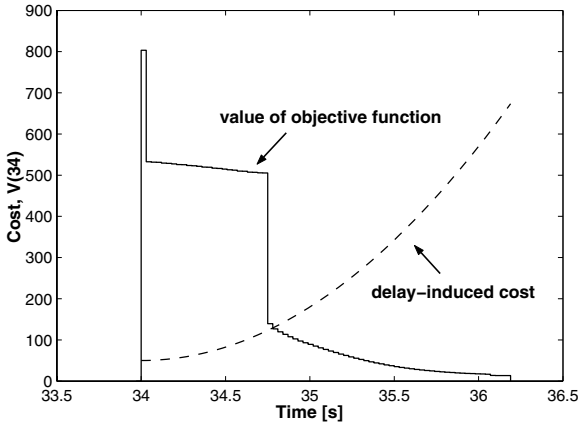

Figure 3. The solid curve shows how the objective function (2) decreases for each iteration during a typical optimization run. The dashed curve illustrates the expected loss due to delay.

Ideally, the increase in cost due to latency for a given MPC task should be computed analytically. A natural stop criterion for the optimization would then be when the total cost starts to increase instead of decrease. However, calculating the delay-induced cost turns out to be a very difficult problem, because of the complex and non-linear nature of the MPC. Until further theoretical results are obtained in this area, we suggest a simpler stop criterion based on the specification of a maximum allowed delay for the controller. The pseudo-code for the controller would then look like this:

```
t = currentTime();
do forever {
   read sample;
   do {
      perform one optimization iteration;
      delay = currentTime() - t;
   } while (!optimum && delay < MAXDELAY);
   actuate plant;
   t = t + T;
   waitUntil(t);
}
```

Note that the code above allows for occasional overruns. If the maximum delay is larger than the period, the next task instance may be delayed until the current instance has finished. This can actually be beneficial from a control performance point of view, which will be shown in the case study in Section 5.

4.2. Scheduling of several MPCs

The naive approach when scheduling several MPC tasks would be to implement each task according to the pseudo-code above, and then to schedule them as ordinary periodic tasks using fixed-priority or earliest-deadline-first scheduling. The main problem with this approach however, is that the (possibly dynamic) priorities of the tasks do not reflect their computational demand nor their relative importance at each point in time.

A better approach is therefore to let the current values of the MPC cost functions act as dynamic task priorities. A scaling of the cost functions may be necessary to achieve a fair comparison. The pseudo-code of the feedback scheduler looks like this:

```
do forever {
   for (each active MPC task i) {
      if (delay_i > MAXDELAY_i) {
         abort optimization;
         actuate plant;
      }
   }
   determine MPC task i with highest cost;
   let task i perform one iteration;
   if (optimum_i) {
      actuate plant;
   }
}
```

To guarantee at least nominal stability of the controllers (ignoring the latencies, see Section 3.4), a controller which does not have a feasible solution yet is associated with an infinite cost (this is typically achieved within the first iteration). When an MPC task has just received new samples, it is also associated with an infinite cost. This ensures that the cost function will become properly updated as soon as possible to reflect current situation in the controlled process.

5. Case study

In this section, a case study of an MPC for a quadruple-tank process will be presented to illustrate the main ideas from Section 4. The real-time properties of the controller are examined, and scheduling of first one and then two controllers under different strategies is investigated by simulation.

5.1. The controlled process

The process used in the case study is the quadruple-tank laboratory process, see Figure 4. This process was first presented in [8] and is a good example of a multi-variable process. The goal is to control the level of the two lower tanks,

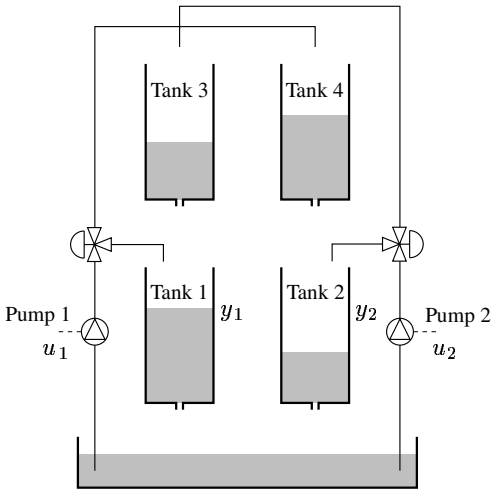

Figure 4. The quadruple-tank laboratory process. From [8].

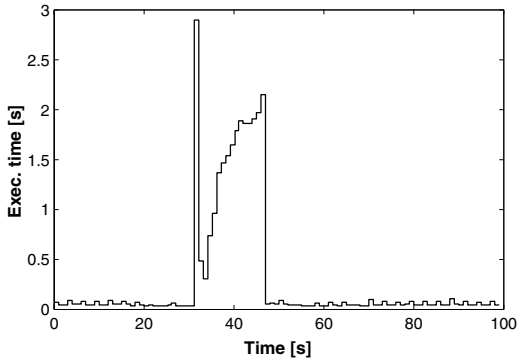

Figure 5. Execution time measurement for the MPC algorithm in the specific simulation scenario given by Figure 6.

y_1 and y_2, using the two pumps, u_1 and u_2. The flow from pump 1 is divided such that a fraction γ_1 enters tank 1 and $1-\gamma_1$ enters tank 4. Likewise, the flow from pump 2 is divided such that a fraction γ_2 enters tank 2 and $1-\gamma_2$ enters tank 3. The cross-coupling in the process makes it hard to achieve good performance using standard PID loops.

The process is linearized around a stationary point and is sampled with an interval of one second. This gives a standard discrete-time state-space model of the process to be used as the internal model by the MPC. In the following, y_1, y_2, u_1, and u_2 will denote deviations from the stationary point.

5.2. Real-time properties

A 100 seconds simulation scenario, see Figure 6, is studied, where at time t = 30 s, a step reference change is commanded in y_1. Also, a step load disturbance enters in y_2 at the same time. The disturbance is not modeled by the internal model, and will therefore make the optimization problem harder. The reference value for y_2 is zero throughout the simulation.

The result of an ideal simulation of the simulation scenario is given by the solid curves in Figure 6. The control performance is optimal and is obtained by letting the optimization algorithm run to termination in each sample and setting the computational delay to zero in the simulation.

A Java implementation of the controller was used to measure the execution time in the described scenario. The result is shown in Figure 5. A considerable difference in execution time can be noticed between different operating conditions. During steady-state operation, the required computational time is negligible, whereas during a step response it may be considerable longer than the period of the control task (one second). It is also seen in Figure 6 that the input signal u_1 operates at its constraint (4 V). The active constraint makes the optimization problem harder and contributes to the increase in execution time.

The varying execution times is due to the large difference in the number of iterations required by the optimization algorithm. In the following simulations it is assumed that the execution time only depends on the number of iterations, and the execution time for each iteration is set to 40 ms.

5.3. Simulation environment

The MATLAB/Simulink-based simulator TRUETIME [7] is used to simulate controller task execution in a real-time kernel in parallel with the continuous-time dynamics of the quadruple-tank process. The detailed co-simulation makes it possible to study the effect of different scheduling policies and execution scenarios on the control performance.

5.4. A single quadruple-tank controller

To get an idea of how the input-output latency affects the control performance, we first study the case of a single MPC executing without interference from other tasks. In all implementations considered below, the sampling is performed by a separate high-priority task.

In a first simulation, the optimization algorithm is allowed to run to termination in each instance. The resulting control performance is shown by the dashed curves in Figure 6, and as a consequence of the delays caused by the varying execution times the performance degrades.

Next, the implementation from Section 4.1 is used, with the delay limit set to two periods. The resulting control per-

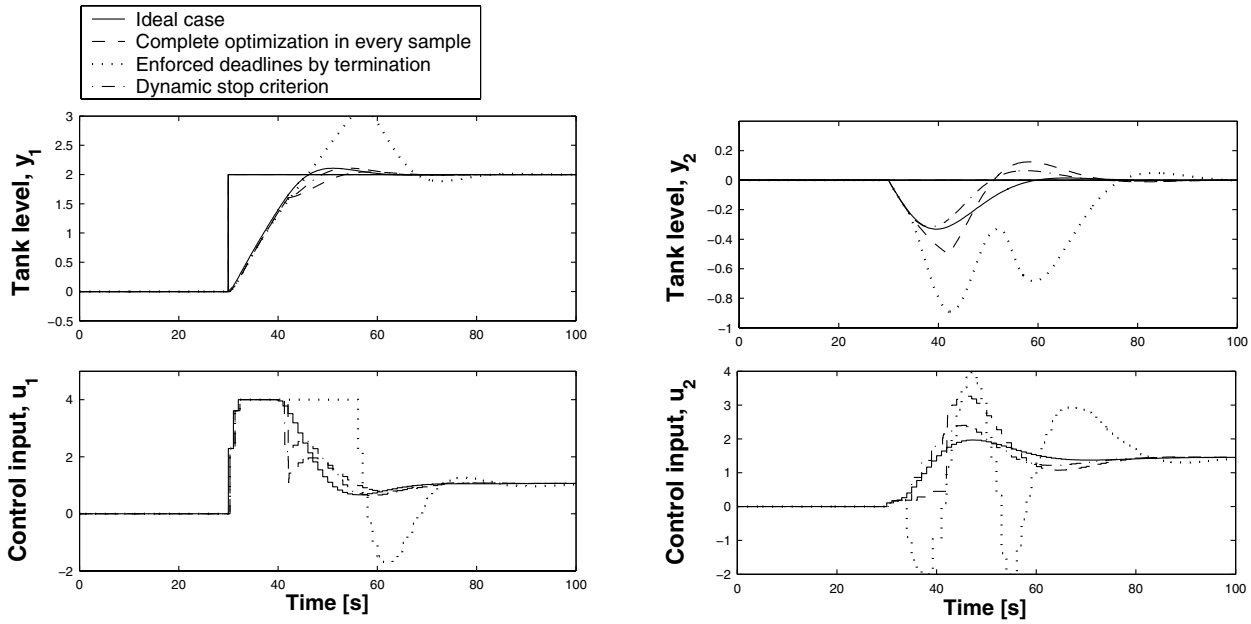

Figure 6. Comparison of control performance in the single MPC case for different implementations. Best performance (dash-dotted) is obtained by exploiting the dynamic trade-off between optimization and delay. Enforcing hard deadlines by aborting prolonged computations at the deadline gives bad performance (dotted). The dashed curves show a standard implementation with no abortion.

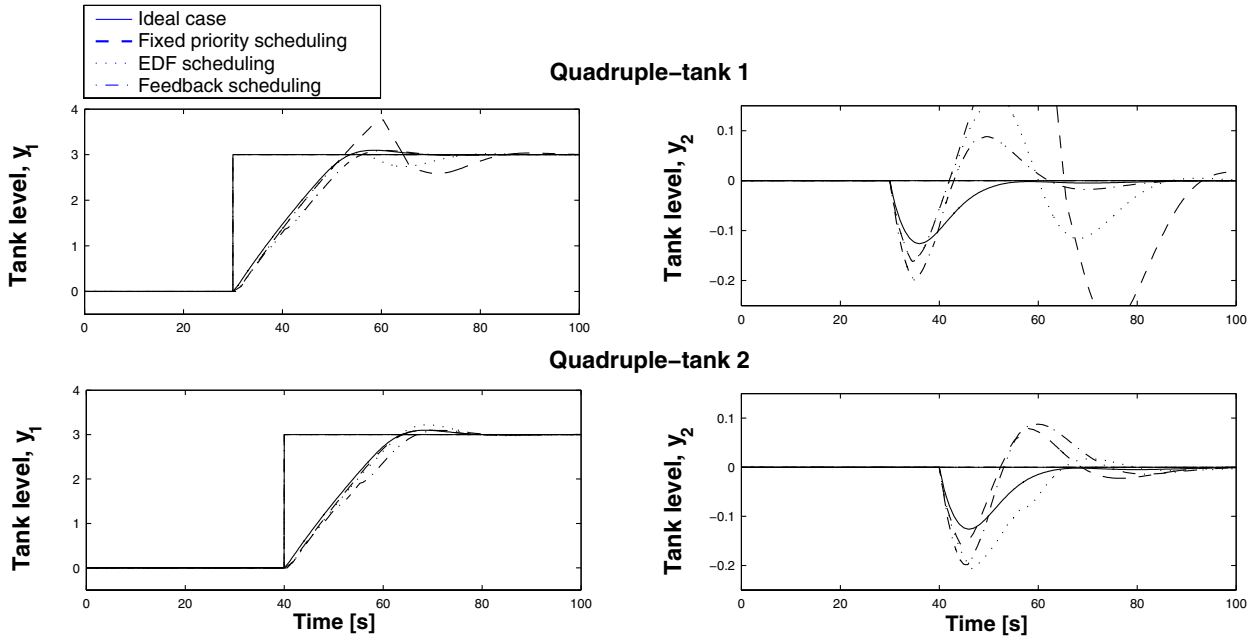

Figure 7. Scheduling two MPC tasks for two identical quadruple-tank processes. The curves show a comparison of control performance obtained by fixed-priority scheduling (dashed), earliest-deadline-first scheduling (dotted), and feedback scheduling (dash-dotted). The introduction of the feedback scheduler, using feedback from the cost functions, improves the control performance considerably.

Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02)
0-7695-1739-0/02 $17.00 © 2002 **IEEE**

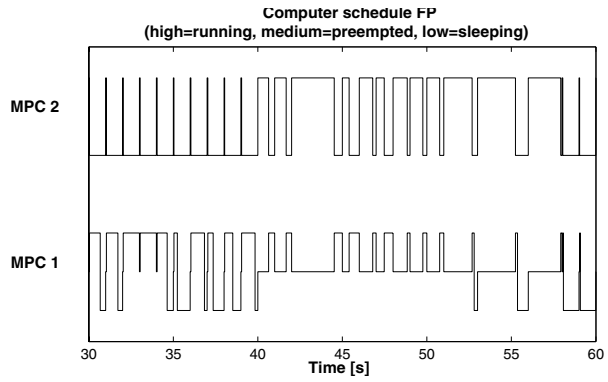

Figure 8. Close-up of the schedule under fixed-priority scheduling. The low-priority controller task (MPC 1) is preempted during significant amounts of time with resulting poor control performance.

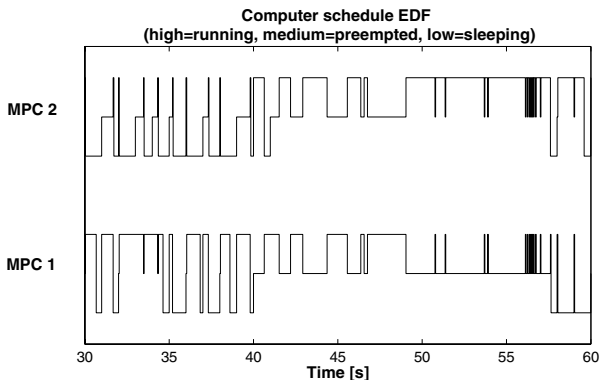

Figure 9. Close-up of the schedule under earliest-deadline-first scheduling. Between 40 and 55 seconds both controllers consume much computing resources and the system becomes heavily overloaded. Just before 60 seconds both tasks are running frequently to "catch up" to previously missed deadlines.

formance is quite close to the optimal, as shown by the dash-dotted curves in Figure 6. The value for the delay limit was found by extensive simulations.

An obvious alternative would have been to enforce hard deadlines by always terminating the optimization at the end of the period. In this example, this turns out to render really bad control performance, which is shown by the dotted curves in Figure 6. Obviously, the optimization is sometimes terminated too early in this case. Optimizing further may not only improve the quality in this sample, but also in subsequent samples (because the last solution is used as initial guess for the optimization in next sample). Hence, always aborting at the deadline may mean that we get a long sequence of poor control signals.

The simulations indicate that dynamic scheduling, where the MPC task is sometimes allowed to miss a deadline, may give better results than terminating the optimization at the deadline or allowing the optimization to run to completion.

5.5. Two quadruple-tank controllers

We next turn to scheduling of two MPCs for two identical quadruple-tank processes on the same CPU. The simulation scenario for the first controller is the same as before, whereas the reference change and step load disturbance occur ten seconds later for the second controller. The solid curves in Figure 7 show the result of an ideal simulation, with full optimization, no interference and no delay.

Fixed-priority scheduling

We first consider the situation where the tasks are scheduled in a priority-preemptive real-time kernel using distinct priorities, with MPC 2 given the highest priority. In addition to the delays caused by the long execution times, MPC 1 will now also experience delay due to preemption from the high-priority controller task. In the time interval 40-55 s, when both processes are in their transient phases, MPC 1 is preempted during significant amounts of time as seen in the computer schedule in Figure 8. The result is poor control performance, especially the response in y_2 for the first tank. This is shown by the dashed curves in Figure 7.

Earliest-deadline-first scheduling

Next, consider scheduling of the two MPCs using earliest-deadline-first scheduling. The results are given by the dotted curves in Figure 7, and it is seen that the performance is somewhat improved compared to fixed-priority scheduling. However, the response in y_2 for the first tank could still be better. It is a well-known fact that earliest-deadline-first performs bad during overload, where the scheduled system often will experience a *domino effect* in missed deadlines. The degraded performance will in turn influence the execution times, since the actual process output will deviate from the predicted output. This increase in execution time will further worsen the situation. Figure 9 shows a close-up of the computer schedule in the interesting time interval. Just before 60 seconds the execution times decrease significantly. Both tasks are then running frequently to try to "catch up" to their missed deadlines.

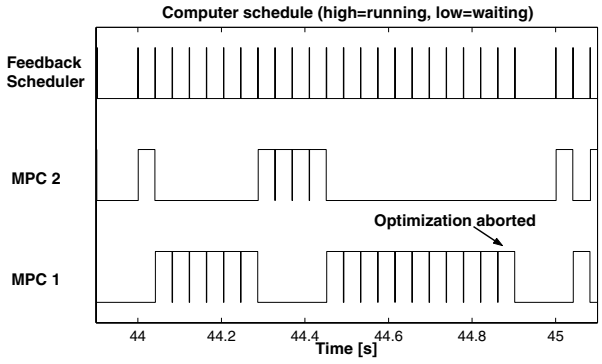

Figure 10. Close-up of the schedule under feedback scheduling. The feedback scheduler (top) distributes the computing on a per-iteration basis based on the values of the cost functions.

Feedback scheduling

The previous simulations indicate that fixed-priority scheduling and earliest-deadline-first scheduling may be inappropriate for scheduling of MPC tasks. The main reason for this is that the relative importance of each task is dynamic and depends on reference changes, disturbances, etc. It is therefore impossible to do relevant off-line priority assignment and dynamic scheduling based on deadlines alone may not be enough.

Instead the feedback scheduler from Section 4.2 is introduced. Both controllers are designed with the same control and prediction horizons, sampling intervals and weighting matrices, making it straightforward to compare the values of the respective cost functions. The feedback scheduler and the MPC tasks communicate and synchronize their execution using events.

Simulation results using the proposed scheme are given by the dash-dotted curves in Figure 7. It can be seen that the control performance has improved, especially for MPC 1. A close-up of the distribution of computing resources in the sample between 44 and 45 seconds is shown in Figure 10. Here it is seen how the execution is divided between the MPC tasks on a per-iteration basis. In the simulation, one scheduling decision is assumed to take 1 ms.

6. Discussion

The simulation results indicate that dynamic scheduling based on feedback from cost functions may be a successful way to deal with limited computational time in the implementation of MPCs. However, to be able to apply the suggested scheme in a more general setting several issues have to be addressed.

One issue, commented on in Section 4.2, is how to make fair comparisons of different cost functions. When scheduling several MPC tasks, the strategy suggested in this paper was to give priority to the controller with the highest current value of its cost function. However, comparing cost functions directly may not be appropriate when the controllers have different sampling intervals, prediction horizons, magnitude of disturbances, etc. In those cases, it would be necessary to scale the cost functions to obtain a fair comparison. The scheduling could also use feedback from the derivatives of the cost functions, as well as the relative deadlines of the different controllers.

Another important issue is the choice of algorithm for the solution of the convex optimization problem. There exist two major families of methods for solving quadratic optimization problems with linear inequality constraints, see e.g. [11]. The traditionally most used is the *Active Set* method, which was used in the examples in this paper. In this algorithm an active set, the set of active inequality constraints, is introduced. As the algorithm proceeds, constraints are added and removed from the active set until the optimal solution is found. A drawback with this method, as seen in Figure 3, is that the cost function may decrease very irregularly as the number of active constraints changes. This makes it difficult to know how close the optimum is and whether it will pay off to optimize further.

In recent years *Interior Point* methods have won widespread use as an alternative to active set methods. Interior point methods may be more suitable in a dynamic setting, since the cost decreases more smoothly by each iteration. It is then easier to estimate how much it will pay off to optimize further—the scheduler could look at the time derivatives of the cost functions to decide which MPC that should run.

For most optimization problems it is possible to formulate another, often simpler problem, called the *dual problem*. One property relating the original problem and its dual, is that their respective objective functions obtain the same value at the optimum. A particularly attractive feature of certain interior point methods [20] is that the algorithm offers an estimation of the difference between the values of the respective objective functions at each iteration. This is a useful feature, since it gives an indication of how close to the optimal point the solution at hand is, and may be used to decide whether to terminate the algorithm or not. This could be a better indication to the scheduler of what MPC task that needs attention than just looking at the current cost.

As described above, premature termination of an optimization run in one MPC may be justified in order to improve the overall control performance. Given that the algorithm at hand has found a feasible solution (considered as a requirement), any of the algorithms may be terminated before the optimum is found. The quality of the solution is

then determined by how close to the optimum the solution is. Potentially, this means that an interior-point method is preferable, since it offers an estimation of how far off from the optimal value a solution at a given iteration is.

7. Conclusions

The paper has discussed feedback scheduling of model predictive controllers (MPCs). The MPC strategy is based on on-line optimization of a cost function, and exhibit large variations in execution time from sample to sample. Since the optimization algorithm is iterative it may be aborted in advance, still producing an acceptable control signal. This is used to trade-off prolonged computations versus input-output latency. The proposed scheduler also uses feedback from the cost functions that are used explicitly in the controller algorithms to dynamically schedule a set of MPC tasks. A case study is presented, where the suggested scheme is shown to perform better than conventional fixed-priority and earliest-deadline-first scheduling.

Acknowledgments

The work has been supported by LUCAS – the VINNOVA-funded Center for Applied Software Research, and by ARTES – the Swedish Network for Real-time Research and Education.

References

[1] T. F. Abdelzaher, E. M. Atkins, and K. Shin. QoS negotiation in real-time systems and its application to automated flight control. *IEEE Transactions on Computers*, 49(11), 2000.

[2] K. J. Åström and B. Wittenmark. *Computer-Controlled Systems*. Prentice Hall, third edition, 1997.

[3] G. Beccari, S. Caselli, M. Reggiani, and F. Zanichelli. Rate modulation of soft real-time tasks in autonomous robot control systems. In *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, pages 21–28, York, England, 1999.

[4] A. Burns, D. Prasad, A. Bondavalli, F. D. Giandomenico, K. Ramamritham, J. Stankovic, and L. Stringini. The meaning and role of value in scheduling flexible real-time systems. *Journal of Systems Architecture*, 46:305–325, 2000.

[5] A. Cervin and J. Eker. Feedback scheduling of control tasks. In *Proceedings of the 39th IEEE Conference on Decision and Control*, pages 4871–4876, Dec. 2000.

[6] C. E. Garcia, D. M. Prett, and M. Morari. Model predictive control: Theory and practice – a survey. *Automatica*, 25(3):335–348, 1989.

[7] D. Henriksson, A. Cervin, and K.-E. Årzén. Truetime: Simulation of control loops under shared computer resources. In *Proceedings of the 15th IFAC World Congress on Automatic Control*, Barcelona, Spain, July 2002.

[8] K. H. Johansson. The quadruple-tank process: A multivariable process with an adjustable zero. *IEEE Transactions on Control Systems Technology*, 8(3), May 2000.

[9] H. Kim and K. Shin. On the maximum feedback delay in a linear/nonlinear control system with input disturbances caused by controller-computer failures. *IEEE Transactions on Control Systems Technology*, 2(2):110–122, June 1994.

[10] J. Liu, K.-J. Lin, W.-K. Shih, A. Yu, J.-Y. Chung, and W. Zhao. Algorithms for scheduling imprecise computations. *IEEE Transactions on Computers*, 1991.

[11] J. M. Maciejowski. *Predictive Control with Constraints*. Prentice-Hall, 2002.

[12] M. Morari and J. H. Lee. Model predictive control: past, present and future. *Computers and Chemical Engineering*, 23:667–682, 1999.

[13] J. Richalet. Industrial application of model based predictive control. *Automatica*, 29:1251–1274, 1993.

[14] K. Shin and C. Meissner. Adaptation of control system performance by task reallocation and period modification. In *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, pages 29–36, 1999.

[15] K. G. Shin and X. Cui. Computing time delay and its effects on real-time control systems. *IEEE Transactions on Control Systems Technology*, 3(2):218–224, 1995.

[16] K. G. Shin and H. Kim. Derivation and application of hard deadlines for real-time control systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(6):1403–1413, November/December 1992.

[17] P. O. M. Stokaert, D. Q. Mayne, and J. B. Rawlings. Suboptimal model predictive control (feasibility implies stability). *IEEE Transactions on Automatic Control*, 44(3):648–654, March 1999.

[18] T.-S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.-C. Wu, and J. W.-S. Liu. Probabilistic performance guarantee for real-time tasks with varying computation times. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium*, 1995.

[19] M. Törngren. Fundamentals of implementing real-time control applications in distributed computer systems. *Real-time systems*, 14(3), 1998.

[20] S. J. Wright. *Primal-Dual Interior-Point Methods*. SIAM, 1997.