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 (MPC's). In model predictive control, the control signal is obtained by optimization of a cost function in each sample, and the MPC task may experience very large variations in execution time. Unique to this application, the cost function also offers an explicit on-line quality-of-service measure for the task. Based on this insight, a feedback scheduling strategy 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.

1. Introduction

Flexible scheduling of tasks with unknown or varying execution times 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. Design based on worst-case assumptions would in these cases lead to under-utilization of the computing resources. The research has resulted in a number of general approaches such as scheduling of imprecise computations [Liu *et al.*, 1991], statistical scheduling, e.g. [Tia *et al.*, 1995], and value-based scheduling, e.g. [Burns *et al.*, 2000].

In this work, we instead take the application as the starting point. Many controllers, including hybrid controllers and model predictive controllers, exhibit large variations in their execution time, e.g. [Årzén *et al.*, 1999]. In this paper, we will concentrate on scheduling of model predictive controllers (MPC's), which in the terminology of [Liu *et al.*, 1991] can be viewed as "milestone" tasks. In an MPC, the control signal is determined by solving a convex optimization problem in every sample. The result is gradually refined for each iteration in the optimization algorithm, up to a certain bound.

The model predictive control strategy has won widespread industrial use in recent years, e.g. [Richalet, 1993]. Since MPC is based on on-line optimization, it has traditionally been applied to plants with slow dynamics in the process industry. Today, faster computers have allowed MPC to be applied also to plants with faster dynamics. In [Dunbar *et al.*, 2002], for instance, MPC is applied to high-performance flight control experiments. The main advantages of MPC are its ability to handle constraints and its straightforward applicability to large, multi-variable processes.

The main problem with MPC, however, is the long and highly varying execution time of the control algorithm. The execution time in each sample depends on a number of factors: the state of the plant, the current and future reference values, the current disturbances acting on the plant, the number of active constraints on control signals and outputs, etc. 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.

In this paper, we take a first step towards a strategy for dynamic scheduling of model predictive controllers. The main idea is to use feedback from the optimization algorithms to determine (a) when to terminate the optimization and output the control signal, and (b) which of several MPC's that should be scheduled at a given time. One very 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 very tight and natural connection between the control and the scheduling. In the paper, a simulation study is performed, where the new approach is compared to conventional fixed-priority and EDF scheduling. The results show that large improvements in control performance can be achieved by more dynamic scheduling.

1.1 Outline

The rest of this paper is outlined as follows. An introduction to model predictive control is given in Section 2. Section 3 discusses the concept of feedback scheduling, and also contrasts the current approach to some related work. Section 4 contains a case study, where an MPC for a quadruple-tank

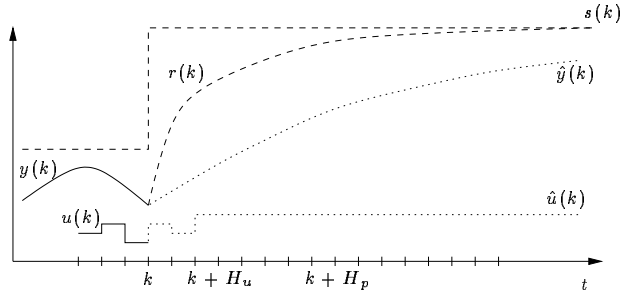

Figure 1 The basic principle of model predictive control.

process is investigated from a real-time perspective. Different scheduling approaches are compared in simulations. Section 5 provides further discussion and possible research directions, while Section 6 gives the conclusions.

2. Model Predictive Control

This section gives a brief overview of model predictive control. In short, MPC is based on three main concepts:

1. Explicit use of a model to predict the output of the controlled process at future discrete time instants, over a *prediction horizon*, H_p.
2. Computation of a sequence of future control actions over a *control horizon*, H_u, by minimizing a given objective function, such that the predicted process output is as close as possible to a desired reference signal.
3. *Receding horizon* strategy, so that only the first control action in the sequence is applied, the horizons are moved towards the future and the optimization is repeated in next sample.

The predicted output values, denoted $\hat{y}(k+i)$ for $i = 1, \ldots, H_p$, depend on the state of the process at the current time k (represented by a collection of past inputs and outputs) and on the future control signals $u(k+i)$ for $i = 0, \ldots, H_u - 1$. If H_u is chosen such that $H_u < H_p$, the control signal is manipulated only within the control horizon and remains constant afterwards, i.e., $u(k+i) = u(k+H_u-1)$ for $i = H_u, \ldots, H_p - 1$, see Figure 1.

The sequence of future control signals $u(k+i)$ for $i = 0, \ldots, H_u - 1$ is computed by optimizing a given cost function. The most often used cost functions are variations of the following quadratic function [Maciejowski, 2002]:

$$V(k) = \sum_{i=1}^{H_p} \alpha_i \left(r(k+i) - \hat{y}(k+i)\right)^2 + \sum_{i=0}^{H_u-1} \beta_i \Delta u(k+i)^2 \quad (1)$$

The first term accounts for minimizing the variance of the process output from the reference, while the second term represents a penalty on the control effort (related for instance to energy). The latter term can also be expressed by using u itself or other filtered forms of u, depending on the problem. The vectors α and β define the weighting of the output error and the control effort with respect to each other and with respect to the prediction step.

A major feature of the MPC algorithm, and the one that has contributed the most to its industrial acceptance, is its ability to handle *constraints*, e.g. saturations of control signals or output signals. These constraints are naturally specified as a part of the optimization problem. At the same time, it is the constraints that make the optimization problem so time-consuming to solve.

The MPC formulation above leads to a quadratic optimization problem with linear inequality constraints. The optimization theory offers strong results for such problems. For example, if a local minimum exists, then it must also be a global minimum. The problem is solved each sample with respect to the control signal increments $\Delta u(k+i)$ for $i = 0, \ldots, H_u - 1$. Only the control signal $u(k) = u(k-1) + \Delta u(k)$ is applied to the process. At the next sampling instant, the process output $y(k+1)$ is available and the optimization and prediction is repeated with the updated values according to the receding horizon principle. The control action $u(k+1)$ computed at time step $k+1$ will be generally different from the one calculated at time step k, since more up-to-date information about the process is available.

3. Feedback Scheduling

Traditional hard real-time scheduling theory assumes that a controller can be described as a periodic task with a fixed period T, a known worst-case computation time C, and a hard deadline D, such that $D = T$. Many controllers however, including hybrid controllers and model predictive controllers, do not fit the traditional task model very well. In particular, these controllers can exhibit very large variations in their execution time. Basing the real-time design on worst-case assumptions can lead to very conservative designs, slow sampling, and, in the end, poor control performance. On the other hand, basing the real-time design on average-case execution times can lead to temporary CPU overloads and starvation of some controllers during runtime. The result can, again, be poor control performance.

One way to handle the large variations in execution time is to introduce feedback in the real-time system. A schematic illustration of a general feedback scheduling system is shown in Figure 2. 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, where the tasks can inform the scheduler that they are about to consume more resources.

In the case of control tasks, there are two main ways

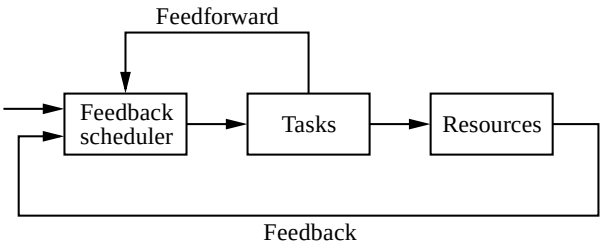

Figure 2 A general feedback scheduling system.

to control the CPU demand. First, one can manipulate the sampling periods of the controllers. This is possible since a controller can typically give satisfactory, although degraded, performance at lower sampling rates than it was designed for. A case study on hybrid controllers where this approach was taken was presented in [Cervin and Eker, 2000]. The feedback in this case consisted of execution-time measurements. The control tasks could also feedforward mode-change information to the scheduler. There was no feedback from the actual control performance.

For model predictive controllers, being milestone tasks, it is more natural to control the CPU demand by manipulating the execution time of the controllers. Also, the standard MPC formulation is based on a sampled-data description with a constant sampling interval, and it would be very difficult (and time-consuming) to allow on-line changes of the sampling period. In the approach suggested in this paper, a scheduling decision is made between each iteration of the optimization algorithm. The feedback information consists of the values of the cost functions of the controllers. No feedforward is needed since the scheduling decisions are so frequent, and since the cost functions automatically include all relevant information, including, for instance, all known future reference changes.

In [Stankovic *et al.*, 1999; Lu *et al.*, 1999], a scheduling algorithm that explicitly uses feedback in combination with EDF scheduling is presented. A PID controller regulates the deadline miss-ratio for a set of soft real-time tasks with varying execution times, by adjusting their requested CPU utilization. It is assumed that tasks can change their CPU consumption by executing different versions of the same algorithm. An admission controller is used to accommodate larger changes in the workload. In [Lu *et al.*, 2000] the same approach is extended. An additional PID controller is added that instead controls the CPU utilization. The two controllers are combined using a min-approach. The resulting hybrid controller scheme, named FC-EDF^2, gives good performance both during steady-state and under transient conditions. Although related to the work presented here, there are important differences. In our approach the tasks that are scheduled are controllers, controlling some physical plants. The performance, or *Quality-of-Control (QoC)* is explicitly used by the feedback scheduler to optimize the overall control performance.

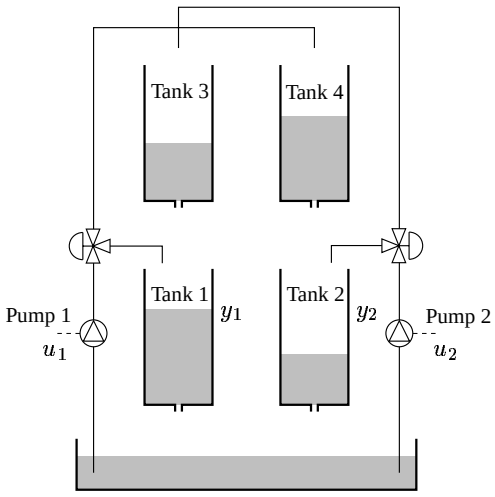

Figure 3 The quadruple-tank laboratory process. From [Johansson, 2000].

4. Case Study

This section contains a case study of a model predictive controller for a quadruple-tank process. 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.

4.1 The Controlled Process

The process used in the case study is the quadruple-tank laboratory process, see Figure 3. This process was first presented in [Johansson, 2000] and is a good example of a multi-variable process. The goal is to control the level of the two lower tanks, 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.

4.2 Real-Time Properties

A 100 seconds simulation scenario, see Figure 4, 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 described above, 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 4. The control perfor-

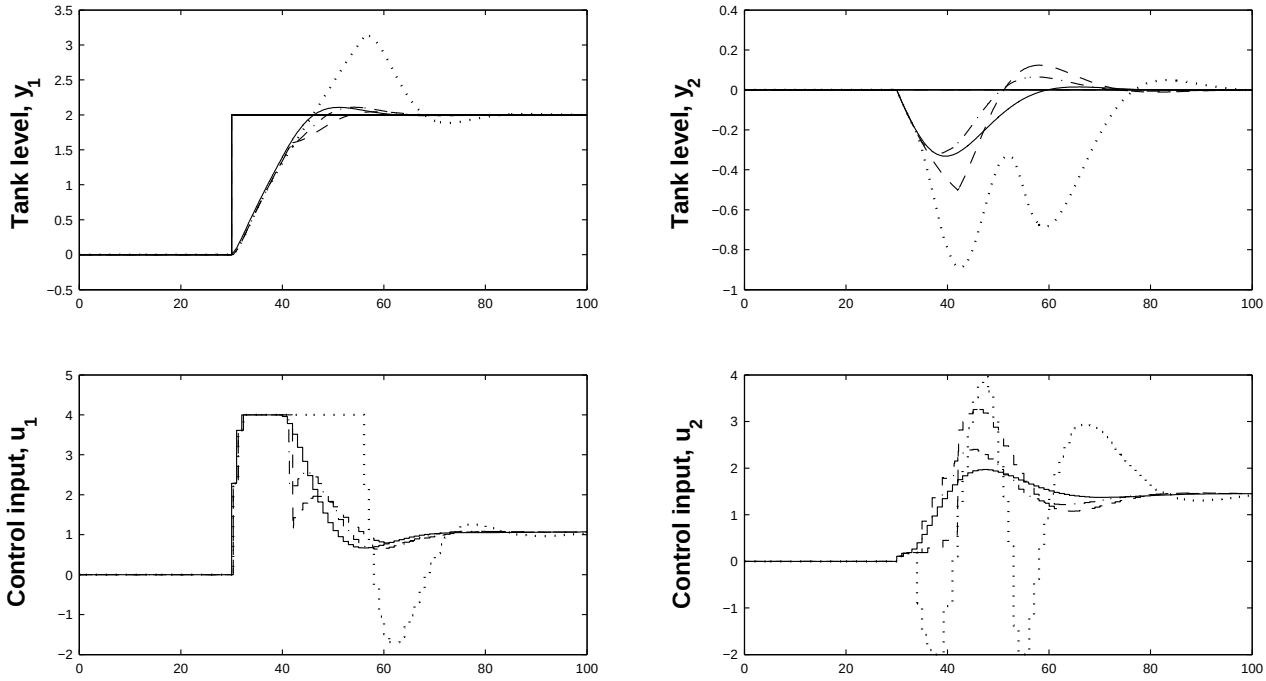

Figure 4 Control performance of a single MPC. The solid curve represents the ideal control performance obtained by not considering the computational delay, and letting the optimization algorithm run to termination in each sample. If the computational time is considered deadlines are missed and the performance degrades (dashed). Enforcing hard deadlines by aborting the optimization at the deadline gives even worse performance (dotted). Best performance is obtained by a dynamic stop criterion (dash-dotted).

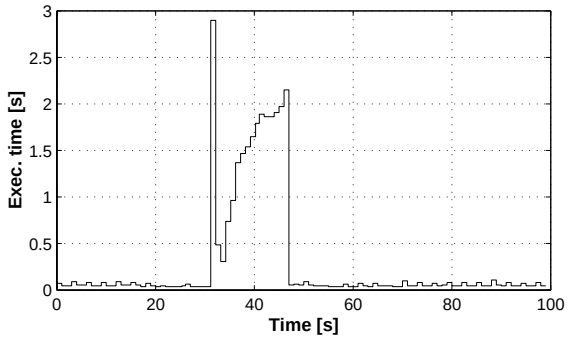

Figure 5 Execution time measurement for the MPC algorithm in the specific simulation scenario given by Figure 4.

mance 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 computation time is negligible, whereas during a step response it may be considerable longer than the period of the control task (one second). It also seen in Figure 4 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.

Figure 6 shows a close-up of how the objective function (1) decreases for each iteration during one of the samples when the execution time is long. It can be seen that the cost function decreases monotonically but not smoothly by each iteration, which is due to the specific method used for the optimization. Different choices of optimization algorithms are discussed further in Section 5.3. While the objective function decreases with each new iteration, it is also expected that the true cost will increase because of the latency. This is illustrated schematically by the dashed curve in Figure 6. This fundamental trade-off between input-output latency and optimization will be studied in the examples below.

4.3 Simulation Environment

The MATLAB/Simulink-based simulator TRUETIME presented in [Henriksson *et al.*, 2002] 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. The real-time simulation environment also allows for system-level communication between the tasks, which will be used to obtain the feedback

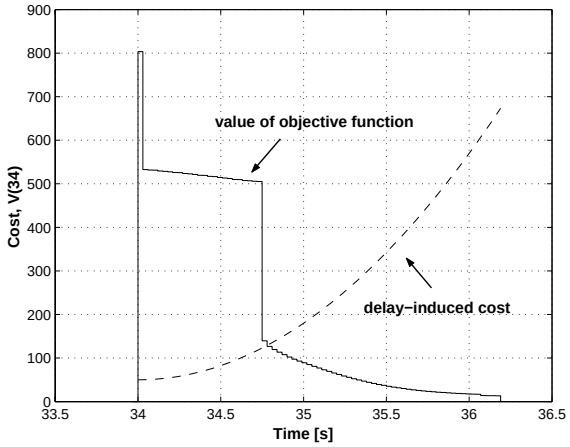

Figure 6 The solid curve shows a close-up of how the objective function (1) decreases for each iteration during a sample when the execution time is long. It is seen that the required execution time for full optimization is well above the period of one second. The dashed curve illustrates the expected loss due to delay.

connection between the controller tasks and the scheduler in the simulation of the feedback scheduling scheme.

4.4 A Single MPC Controller

To get an idea of how the input-output latency affects the control performance, we first investigate the case of a single MPC executing without interference from other tasks. The MPC task is implemented using the standard task model for periodic tasks, i.e., the task is released periodically with the period equal to the sampling interval of the controller, in this case one second. Furthermore the relative deadline of the task is equal to the task period.

In each instance the task will sample the process, perform the optimization, and finally output the computed control signal. The control signal is actuated as soon as the task has completed, i.e., as soon as the optimization has terminated or been aborted. A new instance may not start to execute until the previous instance has completed even if the task executes longer than its period, thus missing its deadline.

If a deadline is missed, which may very well be the case because of the long execution times, the task is immediately released again after completion and the release time is increased by the task period. This will make the task try to ”catch up” to missed deadlines. This situation is studied in a first simulation, where the optimization algorithm is allowed to run to termination in each instance. Since the execution time is sometimes longer than the task period, see Figure 5, there will be considerable sampling jitter and long delays. The result is degraded control performance, as shown by the dashed curves in Figure 4.

An obvious alternative would be to enforce hard deadlines by always terminating the optimization at the end of the period. In this example, however, this turns out to render even worse control performance, which is shown by the dotted curves in Figure 4.

In a third attempt, a dynamic stop criterion is used, where the optimization is terminated after at most two periods, or when the value of the objective function is below a certain threshold. The result is better performance, as shown by the dash-dotted curve in Figure 4. This simulation 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. It is, however, not straightforward to determine when it is best to abort the optimization in relation to the input-output latency. Problems related to this are discussed in Section 5.1.

4.5 Fixed-priority Scheduling

We next consider scheduling of two MPC’s for two identical quadruple-tank processes on the same CPU. The controllers are implemented as synchronous periodic tasks according to the task model described above. 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.

We first consider the situation where the tasks are scheduled in a priority-preemptive real-time kernel using distinct priorities. MPC 2 is given the highest priority, and no termination of the optimization algorithms is performed in this simulation. 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 as shown in Figure 7 (dotted).

4.6 EDF Scheduling

Next, consider scheduling of the two MPC’s using earliest deadline first scheduling. The results are given by the dashed curves in Figure 7, and it is seen that the performance is not considerably improved compared to fixed-priority scheduling. It is a well-known fact that EDF 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. The required execution time decreases just before 60 seconds, and both tasks are then running frequently to try to ”catch up” to their missed deadlines.

4.7 Feedback Scheduling

The previous simulations indicate that fixed-priority scheduling and earliest deadline first scheduling may be

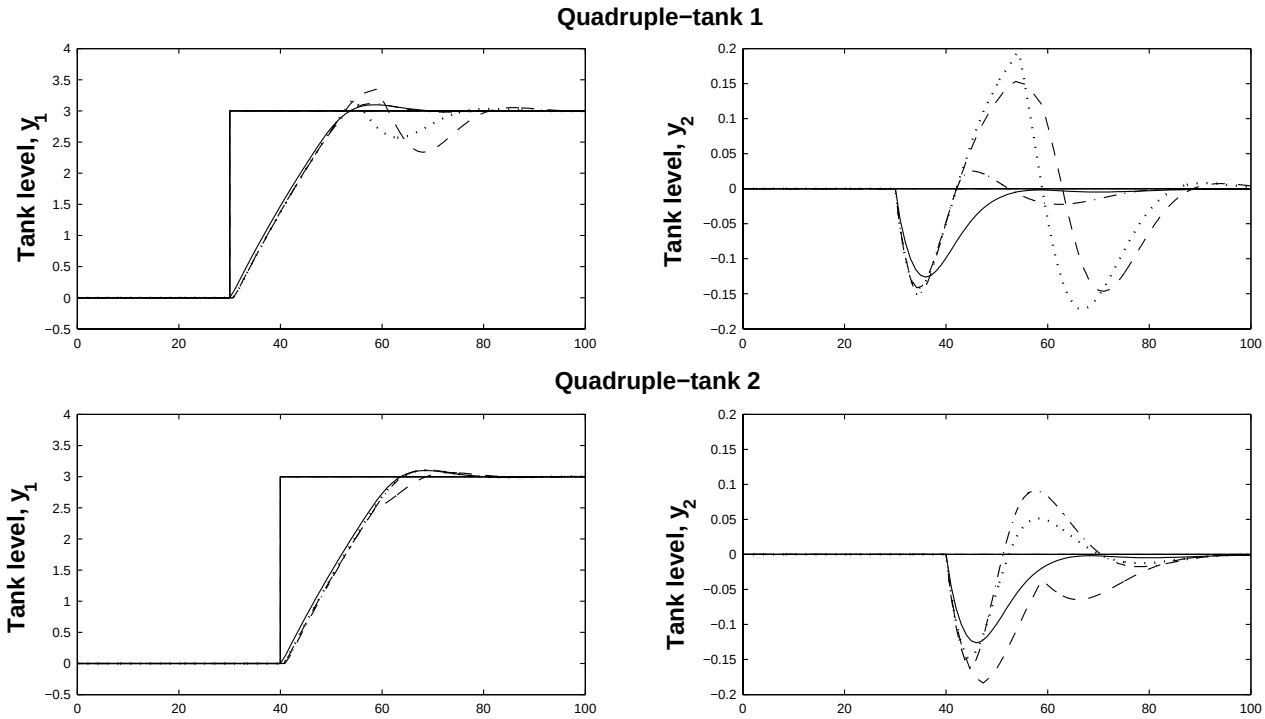

Figure 7 Control performance when scheduling two MPC tasks for two identical quadruple-tank processes. The solid curve shows the optimal performance, where computational delay and interference are neglected. The other curves show a comparison of control performance obtained by fixed-priority scheduling (dotted), EDF scheduling (dashed), and feedback scheduling (dash-dotted). The introduction of the feedback scheduler, using feedback from the cost functions, improves the control performance considerably.

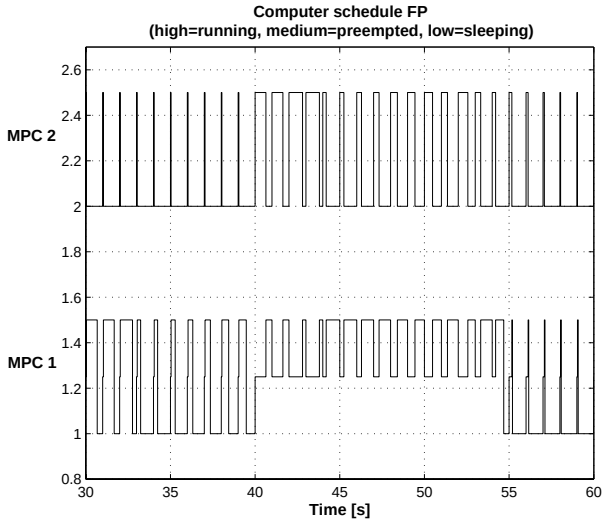

Figure 8 Close-up of the schedule under fixed-priority scheduling. The low-priority controller task (MPC1) is preempted during significant amounts of time with resulting poor control performance.

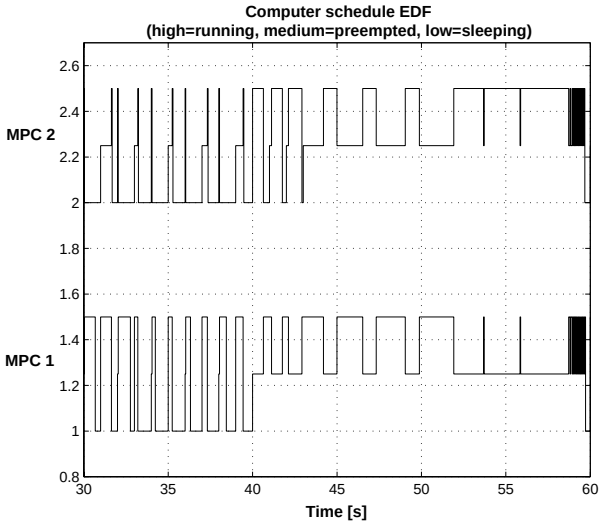

Figure 9 Close-up of the schedule under EDF scheduling. Between 40 and 60 seconds both controllers consume much computing resources and the system becomes heavily overloaded.

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. A better criterion to use in a dynamic scheduling scheme would be the value of the objective functions (1) for the respective controller tasks, which offer a nice on-line quality-of-service measure.

To improve the control performance a feedback scheduler is introduced, which uses feedback from the cost functions to dynamically schedule the two controllers. The feedback scheduler may abort the optimization of the controllers

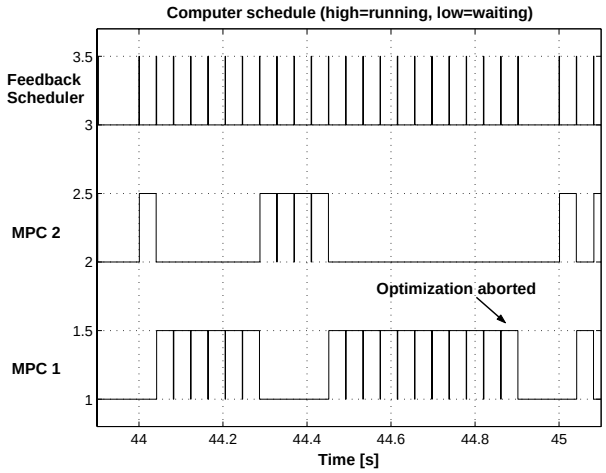

Figure 10 Close-up of the schedule under feedback scheduling. The feedback scheduler (top) distributes the computing on a per-iteration basis based on the values of the cost functions.

to cope with long execution times and to avoid excessive delays. Since both controllers are designed with the same control and prediction horizons, sampling intervals and weighting matrices, it is straightforward to compare the values of the respective cost functions. In other cases, some kind of scaling of the objective functions would have been required. This is discussed in Section 5.2.

The controllers are still implemented as synchronous periodic tasks, but their execution in each instance is governed by the feedback scheduler, which distributes computing resources among the running MPC tasks. The feedback scheduler makes decisions on a per-iteration basis based on the current values of the cost functions of the respective controllers. The MPC with the highest cost will be given the opportunity to perform one iteration. After the iteration the cost function has decreased, and the feedback scheduler will make a new decision as of which MPC to run. For stability reasons the highest priority is to make sure that all MPC's have feasible solutions, i.e., solutions that fulfill the constraints of the optimization problem. Therefore an infeasible solution is associated with an infinite cost.

After each iteration a decision will also be made whether or not to abort the optimization and actuate the plant. This decision depends on the current delay and is made for all active MPC controllers and not only for the one that previously executed. The logic of the feedback scheduler is described in pseudo-code below:

```
LOOP
  for (each active MPC controller) {
    if (delay > limit) {
      abort optimization and actuate plant;
    }
  }
  determine MPC task #i with highest cost;
  let MPC task #i perform one iteration;
END
```

The feedback scheduler and the MPC tasks communicate and synchronize their execution using events. After each invocation of the feedback scheduler it notifies the MPC task that is about to run an iteration. After the iteration the feedback scheduler is notified by another event and repeats the procedure described above.

Simulation results when 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 so for MPC controller 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.

5. Discussion

The simulation results indicate that dynamic scheduling based on feedback from cost functions may be a successful way of dealing with the problem of limited computational time when implementing model predictive controllers. However, a number of non-trivial issues have to be addressed to be able to apply the suggested scheme in a more general setting. Some of these are discussed below.

5.1 Computational Delay

In the MPC formulation used in this paper the computational delay was not accounted for. Standard practice is to include a one-sample delay in the process description and then synchronize the writing of the outputs with the reading of the inputs to enforce this. The computational delay, however, could vary from a very small fraction of the sampling interval up to several sampling intervals. In the dynamic schemes presented in the paper, the control signal was actuated as soon as the optimization terminated, not to induce any unnecessary delay that degrades the performance. Ideally, this should be combined with an adjustment of the prediction matrices in the next sample according to the actual delay.

Another issue is the trade-off between delay and cost during optimization. As time goes, the cost function decreases, but there will also be a penalty due to the input-output delay, see Figure 6. If this penalty could be estimated (as a function computed off-line) it could be possible to terminate the optimization when the cost function plus the penalty starts to increase instead of decrease. Or it could perhaps be possible to include an additional term in the cost function that takes the computational delay into account.

5.2 Comparing 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.

5.3 Different QP-Solvers

There exist two major families of methods for solving quadratic optimization problems with linear inequality constraints, see for example [Maciejowski, 2002]. 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 6, 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, in that the cost typically 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 [Wright, 1997] 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 (this is 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.

6. Conclusions

The paper has discussed feedback scheduling of model predictive controllers, and the potential of the suggested approach has been illustrated by simulations. A case study showed that traditional scheduling approaches, such as fixed-priority scheduling and EDF scheduling, may be inappropriate for scheduling of model predictive controllers. The proposed scheduling approach uses feedback from the cost functions that are used explicitly in the controller algorithms. The feedback scheduler may also abort a task to reduce the input-output latency.

References

Årzén, K.-E., B. Bernhardsson, J. Eker, A. Cervin, K. Nilsson, P. Persson, and L. Sha (1999): "Integrated control and scheduling." Technical Report TFRT-7586. Department of Automatic Control, Lund Institute of Technology, Lund, Sweden.

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

Cervin, A. and J. Eker (2000): "Feedback scheduling of contol tasks." In *Proceedings of the 39th IEEE Conference on Decision and Control*, pp. 4871–4876.

Dunbar, W. B., M. B. William, R. Franz, and R. M. Murray (2002): "Model predictive control of a thrust-vectored flight control experiment." In *Proceedings of the 15th IFAC World Congress on Automatic Control*. Barcelona, Spain.

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

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

Liu, J., K.-J. Lin, W.-K. Shih, A. Yu, J.-Y. Chung, and W. Zhao (1991): "Algorithms for scheduling imprecise computations." *IEEE Transactions on Computers*.

Lu, C., J. Stankovic, T. Abdelzaher, G. Tao, S. Son, and M. Marley (2000): "Performance specifications and metrics for adaptive real-time systems." In *Proceedings of the 21st IEEE Real-Time Systems Symposium*, pp. 13–23.

Lu, C., J. Stankovic, G. Tao, and S. H. Son (1999): "Design and evaluation of a feedback control EDF scheduling algorithm." In *Proceedings of the 20th IEEE Real-Time Systems Symposium*, pp. 56–67.

Maciejowski, J. M. (2002): *Predictive Control with Constraints*. Prentice-Hall.

Richalet, J. (1993): "Industrial application of model based predictive control." *Automatica*, **29**, pp. 1251–1274.

Stankovic, J. A., C. Lu, S. H. Son, and G. Tao (1999): "The case for feedback control real-time scheduling." In *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, pp. 11–20.

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

Wright, S. J. (1997): *Primal-Dual Interior-Point Methods*. SIAM.