Reducing Delay Jitter of Real-Time Control Tasks through Adaptive Deadline Adjustments

Shengyan Hong and Xiaobo Sharon Hu
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556
{shong3,shu}@nd.edu

M.D. Lemmon
Department of Electrical Engineering
University of Notre Dame
Notre Dame, IN 46556
lemmon@nd.edu

***Abstract*—For many control systems, control performance is strongly dependent on delay variations of the control tasks. Such variations can come from a number of sources including task preemptions, variations in task workloads and perturbations in the physical environment. Existing work has considered improving control task delay variations due to task preemption only. This paper presents a general adaptive framework that incorporates a powerful heuristic aiming to further reduce delay variations. Preliminary results indicate that the heuristic significantly improves existing approaches.**

I. Introduction

For many cyber-physical systems, intelligent coordination between control design and its corresponding computer implementation can lead to improved control performance and/or reduced resource demands [1], [18], [24]. A prime example that benefits from such coordination is regulating delay variations (jitter) in control tasks. For many control systems, control performance strongly depends on delay variations in control tasks. Such variations can come from numerous sources including task preemptions, variations in task workloads and perturbations in the physical environment, and can cause degraded control system performance, such as sluggish response and erroneous behavior. An integrated approach to regulate delay variation has the potential to significantly improve a physical system performance.

There are a number of published papers related to reducing delay variations. A somewhat indirect way of reducing delay variations is to reduce task deadlines, which has been investigated by many researchers, e.g., [3], [4], [10], [11], [19]. A common theme of all these methods is to focus on reducing deadlines of either tasks or subtasks. Because deadlines are only allowed to be reduced, these methods cannot effectively explore the design space where deadlines of certain tasks/subtasks may be increased (within some upper bounds) to reduce the overall delay variations.

Another set of methods are based on a task decomposition based approach where each task is partitioned into three subtasks, i.e., Initial, Mandatory, and Final Subtasks (referred as the IMF model), and the delay variation of the final subtask (corresponding to control update) is minimized. The task-decomposition based methods [2], [5] suffer less, but still obvious performance degradation (compared with direct deadline reduction methods) when deadlines are only allowed to be decreased greedily. The decomposition task model is acceptable for control tasks where only a small amount of data needs to be passed to control update subtasks, otherwise context switching cost could be prohibitive. In addition, these methods require repeated worst-case response time computation under the Earliest Deadline First (EDF) scheduling policy, which can be quite time consuming.

Some recent works have focused on studying the influence of task delay variations on control system performance, and how to reduce such influence by scheduling the tasks intelligently in order to enhance system performance. A Matlab-based toolbox for real-time control performance analysis, which takes the timing effects into account, is presented in [15], [23]. A computational model has been proposed in [14] to provide small jitter and short input-output latencies of control tasks to facilitate co-design of flexible real time control systems. Theory of jitter margin is proposed in [16], and applied in [7], [8], [16] to guarantee the stability and performance of controllers in the target system. Some straightforward jitter control methods to improve control system performance, e.g., task splitting, advancing deadlines and enforcing non-preemption, are evaluated in [11]. [9] proposes a delay-aware period assignment algorithm under fixed priority scheduling to reduce the control performance cost. Some of these works [9], [14], [16] adjust control task periods and change the workload of the control system, which may over or under utilize the control system resources, while some of them [7], [8], [11] are not suitable for on-line use due to the exceedingly long computation time of the schedulability analysis in the algorithms.

When considering the various sources that cause delay variation in a physical system and the significant influence of delay variation on the stability and performance of control systems, it is imperative that a delay variation reduction process be integrated in the control loop so as to regulate delay variation whenever there are strong internal and external perturbations. To accomplish this, we need a delay variation reduction approach that is effective, efficient and adaptive. In this paper, we propose an on-line adaptive approach which directly minimizes delay variations for both decomposable and non-decomposable control tasks simultaneously. The approach leverages the IMF based task model for both types of tasks and formulates the delay variation minimization problem as an optimization problem. An efficient algorithm is designed based on the generalized elastic scheduling heuristic [17]. The efficiency of the algorithm readily supports an adaptive framework which can adjust deadlines of control tasks on-line in response to dynamic changes in workloads.

The rest of the paper is organized as follow. Section II reviews important system model and provides some motivations to our work. Section III presents our heuristic to solve the delay variation reduction problem. Experimental results are presented and discussed in Section IV and the paper concludes with Section V.

II. Preliminaries

In this section, we first introduce necessary notation and scheduling properties and then present some motivation for the problem to be solved.

A. System Model

We consider a computer system which needs to handle a set Γ of N real-time control tasks, $\{\tau_1, \tau_2, \cdots, \tau_N\}$, each with the following attributes: (C_i, D_i, P_i), where C_i is the worst case execution time (WCET) of τ_i, D_i is τ_i's deadline, P_i is its period, and $C_i \le D_i \le P_i$. Without loss of generality, we adopt the IMF task modeling approach introduced in [5]. Specifically, we let τ_i be composed of three subtasks, the initial part τ_{ii} for sampling input data, the mandatory part τ_{im} for executing the control algorithm, and the final part τ_{if} to deliver the control action. Thus, a task set Γ_{IMF} consists of $3N$ subtasks $(\tau_{1i}, \tau_{1m}, \tau_{1f}, ..., \tau_{Ni}, \tau_{Nm}, \tau_{Nf})$, each with the following parameters

$$\tau_{ii} = \{C_{ii}, D_{ii}, P_i, O_{ii}\}$$
$$\tau_{im} = \{C_{im}, D_{im}, P_i, O_{im}\}$$
$$\tau_{if} = \{C_{if}, D_{if}, P_i, O_{if}\}$$

where $O_{i\star}$ is the offset of the corresponding subtask. Note that in order for the IMF model to faithfully represent the original task set, each τ_{ii} must be executed before τ_{im}, which must in turn be executed before τ_{if}. For a non-decomposable task, say τ_i, we simply have $C_{ii} = C_{im} = D_{ii} = D_{im} = 0$, and $C_{if} = C_i$. Some tasks may also be partially decomposable, i.e., we may have non-zero C_{ii} and D_{ii} but $C_{im} = D_{im} = 0$.

To achieve desirable control performance, control actions should be delivered at regular time intervals periodically. However, preemptions, variations in task workloads, and perturbations in the physical environment make each instance of the control actions experience different delays. Similar to [5], we define the delay variation as the difference between the worst and best case response times of the same final subtask relative to its period, i.e.,

$$DV_i = \frac{WCRT_{if} - BCRT_{if}}{P_i}, \tag{1}$$

where $WCRT_{if}$, $BCRT_{if}$ are the worst case response time and best case response time, respectively. The definition of delay variation gives information on the delay variance that a task will suffer in the control action delivery within a period. Our problem then is to minimize the delay variations of all the final subtasks.

We use Earliest Deadline First (EDF) scheduling algorithm. A necessary and sufficient condition for a synchronous task set to be schedulable under EDF is given below.

Theorem 1. *A set of synchronous periodic tasks with relative deadlines less than or equal to periods can be scheduled by EDF if and only if* $\forall L \in K \cdot P_i + D_i \le \min(L_{ip}, H, B_p)$ *the following constraint is satisfied,*

$$L \ge \sum_{i=1}^{N} (\lfloor \frac{L - D_i}{P_i} \rfloor + 1) \cdot C_i \tag{2}$$

where $L_{ip} = \frac{\sum_{i=1}^{N}(P_i - D_i)U_i}{1-U}, U_i = \frac{C_i}{P_i}, U = \sum_{i=1}^{N} \frac{C_i}{P_i}$, $K \in \mathbb{N}$ *(the set of natural numbers including 0),* H *is the hyperperiod, and* B_p *is the busy period [6], [13].*

For an asynchronous task set, the condition in Theorem 1 can be used as a sufficient condition [6].

B. Motivation

We use a simple robotic example, similar to the one in [5], to illustrate the deficiencies of existing approaches for delay variation reduction. The example contains four control tasks, i.e., the speed, strength, position and sense tasks. The tasks and the original delay variations under EDF are shown in columns 1 to 5 of Table I. We consider two representative methods for delay variation reduction, which are described as the followings.

In [2], [5], a task decomposition based method, denoted as TDB, is proposed to reduce delay variations of final subtasks. The algorithm replaces the deadlines of final subtasks by their respective worst case response times in the main loop greedily and efficiently until the algorithm converges. However, the method neglects the subtask dependencies in the IMF task model and tends to generate infeasible solutions in high utilization task sets.

Another greedy algorithm presented in [4] indirectly reduces task delay variations by the deadline scaling based technique, denoted as DSB. The algorithm repeatedly reduces the deadlines of all the tasks by the same scaling factor, until the task set becomes unschedulable. The drawback of the method is that the blind deadline reduction may increase the delay variations, which opposes the goal of the algorithm.

Suppose that decomposing the strength and sense tasks in the robotic example would cause non-negligible context switch overhead and we opt to only partition the speed and position tasks according to the IMF model. Assume that the IMF decomposition is made considering that the initial and final subtasks consume 10% of the execution time of the corresponding control task. By applying the DSB and TDB methods, new delay variation values are obtained and are shown as the first two values in column 6 of Table I. TDB actually fails to find a feasible solution and employs the original deadline assignment, while DSB actually gives worse

TABLE I

A MOTIVATIONAL EXAMPLE CONTAINING FOUR TASKS WITH THE SECOND AND FOURTH TASKS BEING NON-DECOMPOSABLE.

Task Name	Computation Exec. Time	Deadline	Period	Original Delay Variations (%)	New Delay Variations (%) DSB / TDB / DVR	Delay Variations before Reassignment (%) DSB / TDB / DVR	Delay Variations after Reassignment (%) DSB / TDB / DVR
Speed	5000	27000	27000	18.52	44.54 / Fail / 0.44	44.54 / 33.33 / 0.44	41.48 / Fail / 0.44
Strength	8000	30000	320000	1.56	3.36 / Fail / 5.94	33.59 / 28.13 / 59.4	31.25 / Fail / 15.59
Position	10000	45000	50000	32	34.75 / Fail / 1	34.75 / 44 / 1	34 / Fail / 1
Sense	13000	60000	70000	40	32.86 / Fail / 9.27	32.86 / 48.57 / 9.27	32.86 / Fail / 20

delay variations. With the DSB method, three tasks suffer more than 30% delay variation.

Now, assume that at some time interval, the execution rate of the strength task increases by 10 times. If the same deadline assignments are used for the tasks/subtasks, the delay variation of the strength task increases to 33.59% and 28.13% for DSB and TDB, respectively (see the first two values in column 7 of Table I). Since TDB cannot find a feasible solution before the workload change, it still employs the original deadline assignment after the change. Suppose we apply the DSB and TDB methods online in response to the period change, the new delay variation values are shown as the first two values in column 8 of Table I. It turns out that the TDB still fails to find a feasible solution while DSB does not even provide much improvement over the delay variations before the reassignment.

With our proposed approach (DVR), smaller delay variations can be obtained for all the cases considered above. In particular, for each respective scenario, we have applied our approach and the delay variation values are shown as the third number in columns 6-8 of Table I. Though for some tasks, delay variations see a small increase, most of the tasks which suffer from large delay variations due to the other methods are now having much smaller delay variations.

III. OUR APPROACH

A. Problem Formulation

From the previous section, one can see that delay variations could be improved significantly if more appropriate deadline assignments can be identified. In this section, we describe our proposed adaptive delay variation reduction (DVR) approach. DVR is built on three basic elements. First, the general IMF model as given in the previous section is used for both decomposable and non-decomposable tasks. Second, the delay variation reduction problem is formulated as an optimization problem. Third, an efficient heuristic is developed to solve the optimization problem. The heuristic is then incorporated into an adaptive framework.

We adopt the generalized IMF task model described in Section II to represent the task set under consideration. The general IMF model allows both decomposable and non-decomposable tasks to be treated equivalently. Given an IMF task set, there may exist numerous sets of feasible deadlines (D_{ii}, D_{im}, D_{if}) which allow the original task set to be schedulable. However, different sets of deadlines could lead to different delay variations of the original tasks. To find the particular subtask deadline assignment that results in the minimum delay variation, we formulate the deadline selection problem as a constrained optimization problem. Though existing work such as [17], [21] has considered the deadline selection problem as an optimization problem, there are two major differences between our present formulation and theirs. First, our formulation directly minimizes delay variations. Second and more importantly, our formulation leverages special properties of the IMF task model and thus allows much more effective delay variation reduction.

The delay variation minimization problem is to minimize the total delay variation bounds of (1) subject to the schedulability constraints as given in (2) while considering the IMF task model. The decision variables in the problem are subtask deadlines D_{ii}, D_{im} and D_{if} (while L as defined in Theorem 1 is dependent on D_{i*}'s). Specifically, we have

$$\text{min: } \sum_{i=1}^{N} w_i (\frac{D_{if} - C_{if}}{P_i})^2 \tag{3}$$

$$\text{s.t. } \sum_{i=1}^{N} [(\lfloor \frac{L - D_{ii}}{P_i} \rfloor + 1) \cdot C_{ii} + (\lfloor \frac{L - D_{im}}{P_i} \rfloor + 1) \cdot C_{im}$$
$$+(\lfloor \frac{L - D_{if}}{P_i} \rfloor + 1) \cdot C_{if}] \leq L, \forall L \in K \cdot P_i + D_i \leq L_{ip}, \tag{4}$$

$$D_{ii} = D_{im}, \tag{5}$$

$$C_{if} \leq D_{if} \leq \min(D_{im}, D_i - D_{im}), \tag{6}$$

$$C_{im} \leq D_{im} \leq D_i - C_{if}, \tag{7}$$

where L_{ip} and K are defined in Theorem 1. If task τ_i is not decomposable, (6) and (7) are replaced by

$$C_{if} \leq D_{if} \leq D_i, \tag{8}$$

$$D_{im} = 0. \tag{9}$$

To see why the above formulation can lead to valid deadline assignments that minimize delay variations, first note that deadline D_{if} is the upper bound of the WCRT of the final subtask of τ_i, and C_{if} is the lower bound of the BCRT of the final subtask of τ_i. Hence, the objective function in (3) is simply the weighted sum of the squares of worst-case delay variations as defined in (1). The use of w_i allows one to capture the relative importance of control tasks in the objective function. The choice of the objective function is based on two observations. First, the quadratic form effectively reduces the variation of jitter distribution. Second, the formulation leads

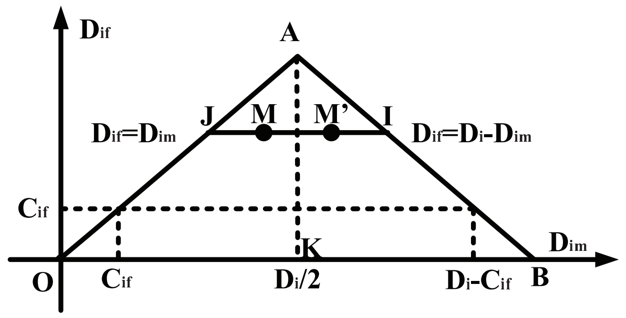

Fig. 1. Feasible deadline region for mandatory/final subtasks.

to an efficient heuristic to solve the quadratic programming problem. We do not directly optimize control performance as such formulation can be quite expensive computationally [15]. (Our experiments will show that the objective function is effective in improving control performance.)

To guarantee schedulability under the IMF model, we have introduced a set of constraints in our formulation. Constraint (4) helps ensure the schedulability of the task set according to Theorem 1. However, this constraint alone is not sufficient since the precedence requirement must be obeyed when executing the initial, mandatory and final subtasks of any decomposable task. Note that ensuring the subtask dependencies during task execution is straightforward. The difficulty lies in capturing this in the schedulability test without either introducing more variables (i.e., O_{ii}, O_{im}, O_{if}) or being overly pessimistic.

To handle the unique challenges due to the IMF task model, we have added several more constraints in addition to (4). To capture the fact that τ_{ii} is always executed before τ_{im}, we can set $D_{ii} \leq D_{im}$ (and hence τ_{ii} has a higher priority than τ_{im} as long as $O_{ii} = O_{im} = 0$). For simplicity, we let $D_{ii} = D_{im}$, assuming that a tiebreak goes to τ_{ii}, which is constraint (5).

Since τ_{if} must start after τ_{im} is completed, we let $O_{if} = D_{im}$. Furthermore, to guarantee that task τ_i finishes by its deadline D_i, we must have $O_{if} + D_{if} \leq D_i$. We thus have $D_{if} \leq D_i - D_{im}$, which leads to one part of (6). The other part of (6), i.e., $D_{if} \leq D_{im}$, reflects the observation that smaller deadlines should be assigned to the final subtask compared to that of the mandatory subtask so as to help reduce delay variation of the final subtask (as this would be the delay variation of interests). (7) constrains the space of D_{im} and is obtained simply by combining $D_{im} \leq D_i - D_{if}$ and $D_{if} \geq C_{if}$. Constraints (8) and (9) replace (6) and (7) for tasks that are not decomposable. Since they are simpler than (6) and (7), our following discussions focus more on constraints (5)-(7).

Based on constraints (5)-(7), Figure 1 depicts the feasible region of (D_{im}, D_{if}), which is bounded by $\triangle ABO$ and corresponds to the search region for the optimal solution to (3)-(7). To make our search more efficient, we would like to reduce the search region as much as possible without sacrificing the optimization solution quality. Theorem 2 provides the basis for reducing the search region.

Theorem 2. *Given a set Γ_{IMF} of N tasks. If the necessary and sufficient condition for schedulability in Theorem 1 is satisfied for a synchronous task set Γ_{IMF} with $(D_{ii} = D_{im}, D_{im}, D_{if})$ for $i = 1, ..., N$, then the same condition is satisfied for a synchronous task set $\Gamma^{'}_{IMF}$ with $(D^{'}_{ii}, D^{'}_{im}, D_{if})$, where $D^{'}_{ii} = D^{'}_{im} \geq D_{im}$ for $i = 1, ..., N$.*

Proof The proof is trivial and omitted. □

Applying Theorem 2 to the search region depicted in Figure 1, one can readily see that point $M^{'}$ on the segment JI leads to a schedulable solution if point M leads to a schedulable solution. Since D_{im} corresponding to $M^{'}$ is larger than that of M, $M^{'}$ is a more desirable solution than M as it leads to a smaller D_{if}. Based on this observation, we can reduce the search region by 1/2 by replacing constraints (6)-(7) in the optimization problem by the following:

$$C_{if} \leq D_{if} \leq D_i - D_{im}, \tag{10}$$

$$\frac{D_i}{2} \leq D_{im} \leq D_i - C_{if}. \tag{11}$$

If task τ_i is not decomposable, constraint (11) is replaced by

$$D_{im} = 0. \tag{12}$$

Hence the jitter minimization can be achieved by solving the optimization problem defined by (3-5), (10-12).

B. DVR Heuristic

Solving the optimization problem specified in (3) together with (4), (5), (10), (11) and (12) is not trivial as it involves dealing with a discontinuous function (the floor function). Heuristic techniques such as the one presented in [17] may be leveraged to solve the problem, but it would take many iterations to reach convergence. In addition, the simplified sufficient condition adopted by [17] either fails to find a solution or finds a very pessimistic solution for task sets with high utilization. We have developed a better heuristic to avoid such problems, which we refer to as DVR.

DVR solves the optimization problem as follows. (The high-level process of DVR is similar to that used in [17], but there are significant differences between DVR and that in [17] in the way that the actual search is conducted.) For an initial solution $(D_{ii} = D_{im}, D_{if})$, the value of L is computed, and an updated set of D'_{if} is obtained by solving the optimization subproblem defined in (3), (4), (5) and (10). The new set of D'_{if} is adjusted to make the solution $(D_{ii} = D_{im}, D'_{if})$ become schedulable, i.e., satisfy the necessary and sufficient condition in Theorem 1; and then D_{im} is updated to D'_{im} by considering not only the constraint on D_{im} but also D_{im}'s effect on future D_{if} selection. These new values, $D'_{ii} = D'_{im}$ and D'_{if}, are then used as the initial solution for the next iteration. This process is repeated in an attempt to find the best set (D_{im}, D_{if}) that minimizes the objective function (3).

Algorithm 1 summarizes the main procedure of DVR. In the algorithm, the current and best solutions found are represented

Algorithm 1 DVR(Γ_{IMF}, Γ, *maxIter*)

```
1: $ePID = Sort\_in\_Task\_Exe\_Time(\Gamma)$
2: $Construct\_Initial\_Solution\_and\_Subproblem(\Gamma_{IMF}, \Gamma, currD)$
3: $L\_busy = Compute\_Busy\_Period(\Gamma)$
4: $state = 0$
5: $duplicate = 0$
6: $BestObjF = +\infty$
7: for $h = 0, h < maxIter, h = h + 1$ do
8:     if $duplicate == \beta$ then
9:         break
10:    end if
11:    $feasibility = Feasibility\_Test(\Gamma_{IMF}, \Gamma, currD, L\_busy, L, time\_demand, h)$ //test whether $currD$ satisfies constraint (4)
12:    if $feasibility == 1$ then //feasible
13:        $ObjF = Obj\_Compute(\Gamma, \Gamma_{IMF}, currD)$
14:        if $|ObjF - BestObjF| < \delta$ then
15:            $duplicate = duplicate + 1$
16:        end if
17:        if $ObjF < BestObjF$ then
18:            $Record\_Current\_Solution(\Gamma, \Gamma_{IMF}, bestD, currD, BestObjF, ObjF)$
19:            $duplicate = 0$
20:        end if
21:        if $state == 1$ then
22:            $Construct\_New\_Subproblem(\Gamma, \Gamma_{IMF}, currD)$ //update $currD_{im} = D_i - currD_{if}$ to formulate a new subproblem
23:        else
24:            $Optimize\_Solution(\Gamma, \Gamma_{IMF}, currD)$ //apply Theorem 3 to compute a new set of $currD_{if}$
25:        end if
26:        $state = !state$
27:    else if $feasibility == 0$ then //not feasible
28:        $overload = time\_demand - L$
29:        if $state == 1$ then
30:            $adjust\_result = Final\_Deadline\_Adjust(\Gamma, \Gamma_{IMF}, currD, overload, L, ePID)$ //adjust $currD_{if}$ to make the found solution schedulable
31:        else
32:            $adjust\_result = Mandatory\_Deadline\_Adjust(\Gamma, \Gamma_{IMF}, currD, overload, L, ePID)$ //adjust $currD_{im}$ to modify the new constructed subproblem
33:        end if
34:        if $adjust\_result == 0$ then //the deadlines cannot be adjusted
35:            break
36:        end if
37:    end if
38: end for
39: return $bestD$
```

by $currD$ and $bestD$, respectively. DVR starts with several straightforward initialization procedures (Line 1-Line 6).

The main loop of DVR spans from Line 7 to Line 38, where DVR searches the best solution $bestD$ to minimize the objective function (3). In each iteration of the main loop, a schedulability check is performed to test whether the current task set Γ_{IMF} satisfies constraint (4) (Line 11). If the current solution satisfies constraint (4), the corresponding objective function value is evaluated by (3) and the new solution is either recorded as the best solution found or discarded (Line 13-Line 20). Furthermore, a set of $currD_{if}$ or $currD_{im}$ will be updated according to the $state$ variable, as shown in Lines 21 to 25. If constraint (4) is not satisfied, subtasks' deadlines are adjusted according to the $state$ variable as shown in Lines 29 to 33 of Algorithm 1. More details about the deadline updates and the use of "$state$" will be given later.

In each iteration of the main loop, DVR updates the final subtasks' deadlines or mandatory subtasks' deadlines according to the $state$ value. DVR updates the final subtasks' deadlines to search an optimal set of D_{if} in the optimization problem described by (3-5) and (10) for a fixed L value, where the most obvious difficulty is how to deal with the discontinuous function (4). Instead of simply adopting the strategy introduced in [17], i.e., removing the floor operator from (4), we propose to use a less pessimistic way to tackle the difficulty. Specifically, we replace constraint (4) by the following two related constraints

$$\sum_{i=1}^{N}[(\frac{L-D_{ii}}{P_i}+1)\cdot C_{ii}+(\frac{L-D_{im}}{P_i}+1)\cdot C_{im} +(\frac{L-D_{if}}{P_i}+1)\cdot C_{if}]=L, L=L_{ip}, \quad (13)$$

$$\sum_{i=1}^{N}[(\lfloor\frac{L-D_{ii}}{P_i}\rfloor+1)\cdot C_{ii}+(\lfloor\frac{L-D_{im}}{P_i}\rfloor+1)\cdot C_{im} +(\lfloor\frac{L-D_{if}}{P_i}\rfloor+1)\cdot C_{if}]\leq L, \forall L\in K\cdot P_i+D_i<L_{ip}, \quad (14)$$

where L_{ip} and K are as defined in Theorem 1, and constraint (13) is guaranteed by L_{ip}'s definition in Theorem 1. It is easy to see that (4) is equivalent to (13) plus (14). We utilize (13) and (14) as follows. Constraint (13) replaces (4) to form a new optimization subproblem defined by (3), (5), (10), (13). Compared with L^* employed in [17], L_{ip} in constraint (13) is simpler to be computed and able to derive less pessimistic solution. Solving the optimization subproblem defined by (3), (5), (10), (13) can be done efficiently by leveraging the Karush-Kuhn-Tucker (KKT) Theorem [26]. Specifically, function $Optimize\_Solution()$ in Line 24 finds the solution according to Theorem 3 for given $L = L_{ip}$ and D_{im} values.

Theorem 3. *Given the constrained optimization problem as specified in (3), (5), (10), (13), for fixed values of $L = L_{ip}$ and mandatory subtask deadlines $D_{im}, \forall i$, let*

$$\tilde{D}=\sum_{i=1}^{N}L\cdot U_i-\sum_{i=1}^{N}D_{im}\cdot(U_{ii}+U_{im})+\sum_{i=1}^{N}C_i-L$$

$$- \sum_{D_{if} \neq D_{ifmax}} D_{ifmin} U_{if} - \sum_{D_{if} = D_{ifmax}} D_{ifmax} U_{if}, \quad (15)$$

$$\tilde{S} = \sum_{D_{if} \neq D_{ifmax}} \frac{U_{if}^2}{w_i} P_i^2. \quad (16)$$

A solution, D_{if}^*, is optimal, if and only if

$$D_{if}^* = \frac{\tilde{D} U_{if}}{\tilde{S} w_i} P_i^2 + D_{ifmin}, \quad (17)$$

where $U_i = \frac{C_i}{P_i}$, $U_{ii} = \frac{C_{ii}}{P_i}$, $U_{im} = \frac{C_{im}}{P_i}$, $U_{if} = \frac{C_{if}}{P_i}$, $D_{ifmin} = C_{if}$ and $D_{ifmax} = D_i - D_{im}$.

Proof Theorem 3 can be proved by applying the KKT conditions. We omit the proof due to the page limit.

Solving this problem for given L_{ip} and D_{im} values results in a set of D_{if} values. However, tasks with this set of deadlines may or may not be schedulable since constraint (13) itself is not equivalent to (4). To check whether the set of deadlines can indeed satisfy the feasibility condition in Theorem 1, we apply (14) to every scheduling point $L < L_{ip}$ to determine if the deadlines found can be satisfied by the tasks. If the solution leads to a schedulable task set, the solution will be used for deriving the deadlines in the next iteration. Otherwise, adjustments to the found deadlines will be made.

In order to make the found solution schedulable, DVR incrementally extends the final subtasks' deadline, which is implemented in function $Final\_Deadline\_Adjust()$ of Algorithm 1 (Line 30). The D_{if}'s value of task τ_{if} is adjusted to a new value such that the activation number of τ_{if}'s jobs within the time interval L_{ip} will be decreased by 1. Such a deadline adjustment is in the decreasing order of the task execution times, and stops as soon as all the job deadlines within L_{ip} can be satisfied by the tasks. If DVR uses any other order of the tasks, e.g., to randomly select a task, to adjust the deadlines, it will take lots of iterations to make the solution schedulable and cause the schedulable solution very pessimistic.

In addition to the final subtasks' deadline update, DVR also updates mandatory subtasks' deadline values by increasing mandatory subtasks' deadlines D_{im} to $D'_{im} = D_i - D_{if}$ in order to construct a new optimization subproblem containing better solutions than the solved subproblems, which is implemented in function $Construct\_New\_Subproblem()$ of Algorithm 1 (Line 22). By setting $D'_{im} = D_i - D_{if}$, constraint (10) is satisfied. More importantly, DVR skips all D''_{im} values that satisfy $D_{im} < D^{''}_{im} < D'_{im}$, because D'_{im} leads to a smaller D_{if} than $D^{''}_{im}$ according to Theorem 2. Additionally, DVR does not increase D_{im} beyond $D'_{im} = D_i - D_{if}$, because it misses the optimal solution contained in the subproblem of $D^{'}_{im}$. In all, the new subproblem formulation, together with the solution initialization described below, makes our search process focus on the subproblems that have a high likelihood of containing the optimal solution. If the constructed new subproblem does not contain any schedulable solution, DVR will incrementally extend the mandatory subtasks' deadlines, which is implemented in $Mandatory\_Deadline\_Adjust()$ and similar to adjusting the final subtasks' deadlines.

To make the best utilization of the new subproblem formulation in the search process, it is important to make the subtasks' deadline initialization corporate well with the update of D_{im}. We let DVR to start the search process from the minimum value of D_{im} and the maximum value of D_{if}, which is implemented in function $Construct\_Initial\_Solution\_and\_Subproblem()$ of Algorithm 1 (Line 2). Specifically, the D_{im} of a decomposable task is set to $\frac{D_i}{2}$, which satisfies constraint (11), while D_{im} of a non-decomposable task is set to 0 according to constraint (12). The initial deadline of the final subtask of any task τ_i is set to $D_i - D_{im}$, according to constraint (10). With the update of D_{im} described above, the upper bound of D_{if} becomes smaller as the number of iterations increases, implying a possible smaller D_{if} by Theorem 3 (and leads to smaller jitter). To accelerate the search process and make DVR more flexible, we also allow user-defined deadlines to be used as an initial solution.

In the main loop, DVR needs stopping criteria to end its search process. We choose variables $BestObjF$ and $duplicate$ to set up the stopping criteria for DVR. $BestObjF$ is the objective function value of the best solution found so far, and $duplicate$ records the number of the found feasible solutions whose objective function values satisfy $|ObjF - BestObjF| < \delta$, where δ is a user-defined parameter. If $|ObjF - BestObjF| < \delta$, $duplicate$ is incremented by 1, as shown in Lines 14 to 16. When the number of "duplicated" solutions is equal to a user-defined parameter β, the program exits from the main loop (Line 9). To handle the case where final subtasks' deadlines do not converge to some fixed values (or when it may take too long for the solution to converge), the algorithm uses another user-defined parameter, $maxIter$, to limit the maximum number of iterations.

The time complexity of DVR is dominated by the busy period computation in Line 3 and the main `for` loop starting at Line 7 of Algorithm 1. The time complexity of the busy period computation algorithm proposed in [31] is $O(\frac{L_{busy}}{C_{min}})$, where L_{busy} is the first busy period and C_{min} is the minimum task computation time. Inside the `for` loop, the most timing consuming operations appear in the $Feasibility\_Test()$ in Line 11. Let N be the number of tasks, $|A|$ be the number of scheduling points inside the first busy period, and $maxIter$ be the maximum iteration count set by the user. The time complexity of the `for` loop is $O(N \cdot |A| \cdot maxIter)$. Thus, the time complexity of DVR is $O(max(\frac{L_{busy}}{C_{min}}, N \cdot |A| \cdot maxIter))$.

We end this section by demonstrating the application of DVR to illustrate the example introduced in Section II-B. Recall that when the strength task increases its execution rate by 10 times, its delay variation also increases by 10 times. We would like to use DVR to find a better set of final subtasks' deadlines. In the initialization phase of DVR, the deadlines are set as $\{currD_{1m}$=26349.9, $currD_{1f}$=640.8; $currD_{2m}$=0, $currD_{2f}$=30000; $currD_{3m}$=43444, $currD_{3f}$=1521.48; $currD_{4m}$=0, $currD_{4f}$=22491$\}$, and $state$ is 0. Since the

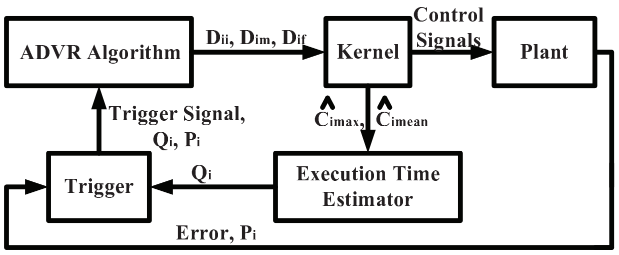

Fig. 2. Adaptive framework for delay variation reduction.

second and fourth tasks are non-decomposable, $currD_{2m}$ and $currD_{4m}$ are 0. In the first iteration of the main loop, the initial task set is tested to be shedulable, and the value of the objective function is 0.49. In addition, $currD$ is recorded in $bestD$. DVR attempts to find a better set of final subtasks' deadlines by applying Theorem 3 in Line 24. The resultant subtasks' deadlines are $\{currD_{1m}$=26349.9, $currD_{1f}$=640.8; $currD_{2m}$=0, $currD_{2f}$=15951.2; $currD_{3m}$=43444, $currD_{3f}$=1521.48; $currD_{4m}$=0, $currD_{4f}$=22491$\}$ and $state$ is changed to 1. In the second and third iteration, $currD$ is checked to be infeasible, and the final subtasks' deadlines are adjusted in function $Final\_Deadline\_Adjust()$. Then in the fourth iteration, the solution $\{currD_{1m}$=26349.9, $currD_{1f}$=640.8; $currD_{2m}$=0, $currD_{2f}$=15951.2; $currD_{3m}$=43444, $currD_{3f}$=1521.48; $currD_{4m}$=0, $currD_{4f}$=27049.9$\}$ obtained in the previous iteration is found to be feasible and the objective function value is 0.10. Then a new subproblem with $\{currD_{1m}$=26359.2; $currD_{2m}$=0; $currD_{3m}$=43478.5; $currD_{4m}$=0$\}$ is constructed in Line 22 and $state$ becomes 0. Such a process is repeated until the algorithm converges.

C. Adaptive Delay Variation Reduction

As we have seen from the motivational example, dynamic workload changes could cause larger delay variations if the original task/subtask deadlines were used. It is desirable to deploy an on-line adaptive framework to adjust task/subtask deadlines when workloads change significantly. The key to such an adaptive framework is an efficient method of solving the optimization problem posed earlier. Based on preliminary results, our heuristic, DVR, seems to satisfy such a requirement. Hence, we propose an adaptive framework built on DVR.

Our proposed framework is similar to the one in [12] and is shown in Figure 2. In this framework, an on-line monitoring mechanism in *Kernel* measures the mean execution time $\hat{c}_i$ and the maximum execution time $\hat{C}_i$, and sends these measured data to *Execution Time Estimator*. *Execution Time Estimator* computes the current execution time estimate Q_i, which is a function of $\hat{c}_i$ and $\hat{C}_i$, and forwards it to *Trigger*. Meanwhile, *Plant* also reports its error, i.e., the difference between the actual and ideal performances of *Plant*, and task period P_i to *Trigger*. When the error and the changes of Q_i and P_i reach some thresholds, e.g., the allowed maximum degradation of the *Plant*'s performance, *Trigger* will signal DVR algorithm to recompute the deadlines and send the results to *Kernel*. With these new results, *Kernel* adjusts *Plant* so as to reduce delay variations. The problem of the threshold formulation is under study using techniques in [22], [27], [32].

IV. Experimental Results

In this section, we first evaluate the performance and efficiency of our heuristic DVR based on randomly generated task sets and compare DVR with the iterative method TDB in [5] and the greedy method DSB in [4]. Then, we illustrate the use of our heuristic in solving real-world problems. Last, we demonstrate the effectiveness of our adaptive framework through the simulation of actual control systems. Our heuristic was implemented in C++, running on a Sun Ultra 20 (x86-64) workstation with Red Hat Enterprise Linux 4.

A. Performance of DVR

We present the following comparisons in this section. First, we demonstrate the performance of DVR by comparing the number of problems it is able to solve with what can be solved by TDB and DSB. Second, to assess the solution quality of DVR, we compare the solutions obtained by DVR with the results by TDB and DSB. Third, to show the efficiency of our heuristic, we compare the DVR's execution time for solving a batch of problems with those of TDB and DSB.

To perform the aforementioned comparisons, similar to [17], 1000 task sets consisting of 5 tasks each were randomly generated for 9 different utilization levels ($U_{level} = 0.1, ..., 0.9$) with a total of 9000 task sets. The utilization level is defined to be $U_{level_i} = \sum_{j=1}^{5} \frac{C_j}{P_j}, i = 0.1, ..., 0.9$. Each task is initially schedulable with (C_j, D_j, P_j), using the necessary and sufficient condition (2) in Theorem 1. In our experiment, we set the maximum hyperperiod, minimum period, and maximum period to $500,000$, $10,000$, and $40,000$, respectively. The precision was specified to be 100, whereas the maximum number of tries was set to $10,000$. The precision denotes the minimum increment in any task period. For example, if the precision is set to 100, a task period could be 5200, but not 5010. For the details of the task set generation, readers can refer to [17].

To allow the decomposable task model adopted by both DVR and TDB to be investigated, we randomly select 3 tasks out of each task set to be decomposable, and assume that the initial and final subtasks consume 10% of the execution time of the corresponding control task. Furthermore, because the constraints of the subtask dependencies in the decomposable task model makes it harder to construct initially feasible task sets, we increase each task's deadline to $\frac{D_j+P_j}{2}$.

In the first experiment, we compare the percentage of solutions found by our heuristic, as opposed to those by TDB and DSB. Figure 3 compares the number of solutions found by DVR, TDB and DSB, respectively. If a task set cannot be solved by a method, its solution is said to be not found by this method. The x-axis shows the different utilization levels, whereas the y-axis shows the percentage of solutions found.

TABLE II
NUMBER OF ITERATIONS FOR SOLVING A TASK SET BY DVR.

Utilization Level	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
Average Number of Iterations	9.43	9.29	9.21	9.33	9.59	9.5	11.43	16.31	25.98
Maximum Number of Iterations	19	19	19	19	19	19	38	58	106

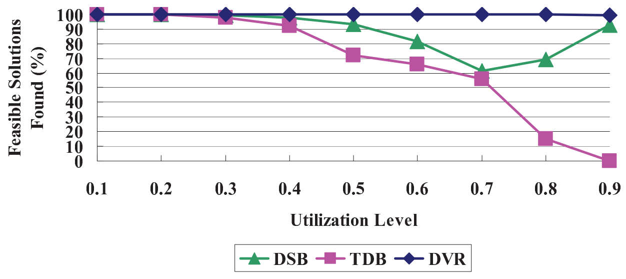

Fig. 3. Comparison of DVR, TDB and DSB in terms of percentage of feasible solutions found.

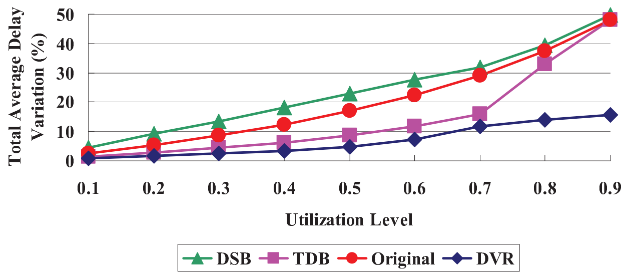

Fig. 4. Comparison of DVR, TDB and DSB in terms of solution quality.

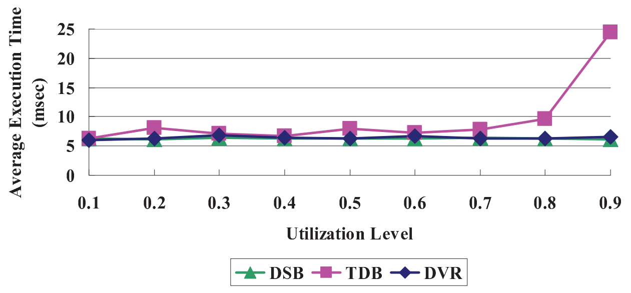

Fig. 5. Comparison of DVR, TDB and DSB in terms of average execution time.

It is clear from the plot that DVR is able to find solutions for all task sets with utilization levels 0.1 to 0.8, while for utilization level 0.9 DVR finds 995 solutions out of 1000 task sets. In constrast, TDB and DSB suffer from various degrees of degradation. With increasing utilization levels, more and more solutions found by TDB cannot satisfy the subtask dependency constraint, i.e., the found solutions are infeasible, because TDB blindly reduces deadlines of final subtasks, neglecting the deadlines of mandatory subtasks. DSB works better than TDB, but it cannot find solutions of many task sets for utilization levels 0.6 to 0.8. Compared with TDB and DSB, DVR performs excellently in obtaining a schedulable solution while guaranteeing subtask dependencies.

The second experiment examines the quality of the solutions found by our heuristic with respect to the original delay variations and that of the solutions found by TDB and DSB. Figure 4 illustrates the solution quality of DVR as well as TDB and DSB. As before, the x-axis shows the different utilization levels. The y-axis shows the average delay variations of the found solutions at each utilization level, i.e., $\frac{\sum_{j=1}^{5}\frac{DV_j}{5}}{1000}$, where DV_j is the delay variation of task τ_j in a task set. To guarantee the fairness of the comparision, the average delay variation of a task set that cannot be solved by a specific method is set to the original average delay variation. The first, and most obvious, observation is that the average delay variations resulted from applying DVR are much smaller than the values by DSB and the original average delay variations. In addition, with increasing utilization levels, such a delay variation difference becomes greater. Actually, DSB gives worse delay variations than the original delay variations at each utilization level, because its blind deadline reduction increases the delay variations. Second, for utilization levels less than or equal to 0.7, TDB performs a little worse than DVR, while for utilization levels greater than 0.7, the performance of TDB degrades drastically. The reason for this is that the numbers of found solutions by TDB for utilization levels greater than 0.7 decrease greatly (see Figure 3), and the average delay variations for these task sets without a solution by TDB are the original delay variations. The results of the first and second experiments show that DVR performs best in applications with various utilizations among the three methods.

To exam whether DVR is suitable for online dynamic deadline adjustments, we study the execution times of DVR and compare them with TDB and DSB in the third experiment. Figure 5 compares the execution times of DVR, TDB and DSB. The x-axis shows the different utilization levels, whereas the y-axis shows the average execution time which it takes for an algorithm to solve the delay variation reduction problem. As shown in Figure 5, TDB spends 24.5 milliseconds on average in searching a solution at utilization level 0.9, and only finds 1 solution out of 1000 task sets finally. DVR and DSB only take 6.58 and 6.21 milliseconds on average to search a solution at utilization level 0.9, respectively, which is almost 4 times faster than TDB. Furthermore, at all utilization levels, TDB always spends longer execution time than DVR and DSB, and DVR has the execution time comparable with that of the greedy method DSB.

To further investigate the convergence characteristics of DVR, the average and maximum numbers of iterations to search for a solution by DVR at various utilization levels are summarized in Table II. As shown in Table II, DVR converges very fast at utilization levels 0.1 to 0.6 with the average number of iterations less than 10 and the maximum number of iterations 19. For higher utilization levels 0.7 to 0.9, the maximum number of iterations increase obviously. However, most of the task sets at high utilization levels can be solved within 50 iterations. For example, 894 task sets at utilization level 0.9 can be solved within 50 iterations, 568 out of which only need 25 iterations or less. The relatively high number of iterations at higher utilization levels is caused by the increased amount of preemption, which requires the adjustment of subtasks' deadlines repeatedly in DVR. However, the execution time of the deadline adjustment is very short, which effectively restrains the average execution time at high utilization levels from increasing drastically.

The experimental results on the execution time and convergence rate of DVR demonstrate that DVR can solve problems with various utilization levels efficiently and have an acceptable convergence rate even for task sets with a high utilization level. The power of DVR lies in its effective problem formulation as well as its efficient search strategy.

B. Experimental data for real-world workloads

Using a large number of randomly generated task sets, we have shown that our heuristic can reduce delay variations greatly in almost all the task sets at various utilization levels. However, it is important to quantify the performance of DVR under real-world workloads. In this section, we compare DVR with TDB and DSB using three real-world applications. The applications we consider are a videophone application, a computerized numerical control (CNC) application, and an avionics application. We will describe the benchmarks one by one, and compare the results obtained by the three methods.

1) *Videophone Application:* A benchmark for a typical videophone application is presented in [30]. The task set is composed of four real-time tasks: video encoding, video decoding, speech encoding, and speech decoding. Delay variations in such an application would degrade the user perceived quality of voice or images. The worst-case execution time and period for each task is given. Task deadlines are randomly generated to be at least 95% of the period by using a uniform distribution. Since the videophone application was proposed in 2001, it is reasonable to reduce each task execution time by 10%. Thus, the utilization of the task set becomes 0.89. Assume that the context switch overhead of partitioning the encoding tasks is non-negligible, so only the video decoding and speech decoding tasks are decomposed based on the IMF model, where the initial and final subtasks consume 10% of the execution time of the corresponding control task.

DVR, TDB and DSB are applied to the application to reduce delay variations. DVR takes 7 iterations to reduce average delay variations from 58.05% to 5.96%, while both TDB and DSB fail to find a feasible solution.

2) *Computerized Numerical Control Application* A benchmark is presented in [20], consisting of eight tasks with measured execution times and derived periods and deadlines of some CNC controller tasks. Consider the situation where the system is overloaded (e.g., the primary computer is down and the backup computer is less powerful). The increased task execution times can cause much higher delay variations and degrade the controller performance significantly. Thus, it is necessary to employ an efficient method to reduce delay variations. To model such a scenario, we multiply each task execution time by a factor of 1.6 and increase the utilization of the task set to 0.78. We assume that all the tasks in the application are decomposed based on the IMF model where the initial and final subtasks consume 10% of the execution time of the corresponding control task.

DVR can reduce the average delay variation from 69.9% to 2.17%, requiring 23 iterations in total, while TDB fails to find a feasible solution and DSB reduces the average delay variation only to 67.8%.

3) *Generic Avionics Application* The authors in [25] present a benchmark containing one aperiodic task and 17 periodic tasks with given period and execution time for each task, for a Generic Avionics Platform (GAP). We randomly assign the task deadlines to be between 50% and 60% of the periods using a uniform distribution. The aperiodic task arrives with a specific deadline and a minimum inter-arrival time. In addition, since the GAP benchmark dates back to 1991, it is reasonable to reduce each task execution time by 20%. The utilization of the task set becomes 0.72. In this application, we still assume that all the tasks are decomposed based on the IMF model where the initial and final subtasks consume 10% of the execution time of the corresponding control task.

DVR reduces the average delay variation from 23.42% to 1.63% within 22 iterations. TDB is unable to find a feasible solution, and DSB actually increases the average delay variation to 30.64% after 324 iterations.

According to the experimental results of the three real-world benchmarks, we show that DVR is capable of reducing delay variations greatly within a limited number of iterations. In contrast, TDB always fails to find a feasible solution for such high utilization level applications, while DSB cannot guarantee to find a satisfactory solution even after quite a few iterations.

C. Performance of Adaptive-DVR

We use a simple system composed of one hard real-time task and three control tasks, similar to the systems employed in [7], [8], [11], to illustrate the efficiencies of DVR for delay variation reduction and control performance improvement. The hard real-time task is a non-control task without jitter and delay requirement except for schedulability, while each control task τ_i actuates a continuous-time system of two stable poles P_i. In order to vary the sensitivity towards delay variations, we employ the method presented in [9] to generate random plants P_i, which are given by

$$P_1(s) = \frac{1000}{(s+0.5)(s-0.2)},$$

$$P_2(s) = \frac{1000}{(s+0.5)(s-0.3)}, \tag{18}$$

TABLE III
A CASE STUDY OF CONTROL TASK DELAY VARIATIONS AND THEIR CONTROL PERFORMANCE BY ADAPTIVELY APPLYING DVR.

Task Name	Computation Exec. Time	Deadline	Period	Delay Variations (%) / Control Performance Original / DSB / TDB / DVR	New Computation Exec. Time	New Deadline	Delay Variations (%) / Control Performance Original / DSB / TDB / DVR
Hard Real-time Task	570	3810	3810	33.7 / 33.7 / Fail / 37.82	1140	2290	7.27 / 23.12 / Fail / 29.52
				NA / NA / NA / NA			NA / NA / NA / NA
Control Task τ_1	1570	10000	10000	18.54 / 18.54 / Fail / 12.84	3140	10000	59.87 / 56.84 / Fail / 8.08
				153.3 / 153.3 / Fail / 112.5			2089.49 / 1246.6 / Fail / 105.2
Control Task τ_2	855	1500	6860	6.25 / 6.25 / Fail / 1.16	1710	5710	45.58 / 48.52 / Fail / 4.17
				82.2 / 82.2 / Fail / 71.9			123.8 / 130.2 / Fail / 74.2
Control Task τ_3	429	1500	8570	9.98 / 9.98 / Fail / 3.83	857	7570	68.14 / 67.94 / Fail / 1.98
				95.3 / 95.3 / Fail / 86.6			240.5 / 240.5 / Fail / 86.6
Average Delay Variation				11.59 / 11.59 / Fail / 5.94			57.86 / 57.77 / Fail / 4.74
Total Performance Cost				330.8 / 330.8 / Fail / 271			2453.79 / 1617.3 / Fail / 266

$$P_3(s) = \frac{1000}{(s+0.2)(s-0.7)}.$$

In [9], [14], the control performance cost J_i of each system is defined to be

$$J_i = \lim_{T \longrightarrow \infty} \int_0^T (y_i(t)^2 + u_i(t)^2)dt, \tag{19}$$

where $y_i(t)$ and $u_i(t)$ are the system output and input controlled by task τ_i, respectively. Thus, the total control performance cost of the continuous-time system is

$$J = \sum_{i=1}^{3} J_i. \tag{20}$$

In addition, the hard real-time task and the three control tasks are generated by the method presented in [9], as shown in columns 1 to 4 of Table III. All the tasks have a transient and location dependent deadlines, similar to the cases presented in [28], [29].

We compare the delay variation values of the control tasks by applying DVR with those obtained by TDB and DSB. Furthermore, given delay variation and sampling period of a control system, we are able to compute control performance cost by using Jitterbug proposed in [15], [23].

The delay variation value and control performance cost of each control task in the original task set and the corresponding values obtained by applying DSB, TDB and DVR are shown in column 5 of Table III. For each task, two rows of data are shown in columns 5 and 8. The top row corresponds to delay variation while the bottom corresponds to control performance. The data clearly show that DVR reduces the average delay variation from 11.59% to 5.94% and improves the total performance cost from 330.8 to 271. However, DSB neither reduces the delay variations nor improves the corresponding total control performance cost at all. TDB fails to find a feasible solution for the task set.

Now, assume that at some time interval, the execution times of all the tasks increase by 2 times, due to the reason that the primary processor is down and the backup processor is much slower. Meanwhile, the state-dependent deadlines of the hard real-time task and the control tasks τ_2 and τ_3 also change to satisfy the user's requirement. The new execution times and deadlines of the tasks are shown in columns 6 and 7 of Table III. If the previous deadline assignments generated by DSB and DVR are reused for the tasks/subtasks, the deadlines of the hard real-time task and the control tasks will be missed. If no delay variation reduction method is applied to the current case, the delay variation and control performance cost of each control task are shown as the first numbers of column 8 of Table III. The average delay variation and total control performance cost increase to 57.86% and 2453.79, respectively, which may not be acceptable to the system.

Suppose we apply the DSB, TDB and DVR methods online in response to the workload and deadline change, the new delay variation values and control performance costs are shown as the second to the fourth numbers in column 8 of Table III. With our proposed approach (DVR), smaller average delay variation 4.74% and total control performance cost 266 can be obtained in the current situation, which overcomes the negative effects on the system caused by the workload and deadline change. However, TDB fails to find a feasible solution while DSB gives average delay variation 57.77% and total control performance cost 1617.3, much worse than the results obtained by DVR.

V. SUMMARY AND FUTURE WORK

We have presented a new approach to reduce delay variations of control tasks. The approach formulates the delay variation reduction problem as an optimization problem that can effectively handles both decomposable and non-decomposable tasks. Based on several key observations, we have devised an efficient heuristic to solve the optimization problem. The efficiency of the heuristic leads to an adaptive framework that can dynamically readjust task/subtask deadlines to keep delay variations small and improve control system performance in the presence of environment perturbations. Experimental results show that our heuristic performs well in delay variation reduction for randomly generated task sets at various utilization levels and three real-world workloads. In addition, we have applied our adaptive framework to a real-time control system and improved its control performance greatly.

As future work, we will further evaluate the proposed approach by implementing it in a real-time operating system and applying it to actual control application. We will explore other kinds of objective functions and constraints that may be more influential on control system performance.

VI. Acknowledgement

This work is supported in part by NSF under grant numbers CNS07-20457 and CNS09-31195.

References

[1] P. Albertos, A. Crespo, I. Ripoll, M. Valles, and P. Balbastre, "Rt control scheduling to reduce control performance degrading," in *ICDC '00*.

[2] P. Balbastre, I. Ripoll, and A. Crespo, "Control tasks delay reduction under static and dynamic scheduling policies," in *RTCSA '00*.

[3] ——, "Optimal deadline assignment for periodic real-time tasks in dynamic priority systems," in *ECRTS '06*.

[4] ——, "Minimum deadline calculation for periodic real-time tasks in dynamic priority systems," *IEEE Trans. Comput.*, vol. 57, no. 1, pp. 96–109, 2008.

[5] P. Balbastre, I. Ripoll, J. Vidal, and A. Crespo, "A task model to reduce control delays," *Real-Time Syst.*, vol. 27, no. 3, 2004.

[6] S. K. Baruah, L. E. Rosier, and R. R. Howell, "Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor," *Real-Time Syst.*, vol. 2, no. 4, 1990.

[7] M. Behnam, "Flexible scheduling for real time control systems based on jitter margin," in *Master Thesis, Malardalen Research and Technology Center, Malardalen University*, 2005.

[8] M. Behnam and D. Isovic, "Real-time control and scheduling co-design for efficient jitter handling," in *Embedded and Real-Time Computing Systems and Applications, 2007. RTCSA 2007. 13th IEEE International Conference on*, Aug. 2007, pp. 516–524.

[9] E. Bini. and A. Cervin, "Delay-aware period assignment in control systems," in *Real-Time Systems Symposium, 2008*, Dec. 3 2008, pp. 291–300.

[10] E. Bini and G. Buttazzo, "The space of edf deadlines: the exact region and a convex approximation," *Real-Time Syst.*, vol. 41, no. 1, pp. 27–51, 2009.

[11] G. Buttazzo and A. Cervin, "Comparative assessment and evaluation of jitter control methods," in *Real-Time and Network Systems, 2007. RTNS 2007.*, Mar. 2007.

[12] G. Buttazzo and L. Abeni, "Adaptive workload management through elastic scheduling," *Real-Time Syst.*, vol. 23, no. 1/2, pp. 7–24, 2002.

[13] G. C. Buttazzo, "Hard real-time computing systems: Predictable scheduling algorithms and applications." Springer, 2005.

[14] A. Cervin, "Integrated control and real-time scheduling," in *PHD Thesis, Department of Automatic Control, Lund Institute of Technology*, April 2003.

[15] A. Cervin and B. Lincoln, "Jitterbug 1.21 reference manual," Feb. 2006.

[16] A. Cervin, B. Lincoln, J. Eker, K.-E. Arzen, and G. Buttazzo, "The jitter margin and its application in the design of real-time control systems," in *Embedded and Real-Time Computing Systems and Applications, 2004. RTCSA 2004. 10th IEEE International Conference on*, Aug. 2004.

[17] T. T. Chantem, X. S. Hu, and M. D. Lemmon, "Generalized elastic scheduling for real-time tasks," *IEEE Trans. Comput.*, vol. 58, no. 4, pp. 480–495, 2009.

[18] A. Crespo, I. Ripoll, and P. Albertos, "Reducing delays in rt control: The control action interval, decision and control," in *IFAC '99*.

[19] H. Hoang, G. Buttazzo, M. Jonsson, and S. Karlsson, "Computing the minimum edf feasible deadline in periodic systems," in *RTCSA '06*.

[20] N. Kim, M. Ryu, S. Hong, M. Saksena, C.-H. Choi, and H. Shin, "Visual assessment of a real-time system design: a case study on a cnc controller," in *RTSS '96: Proceedings of the 17th IEEE Real-Time Systems Symposium*, 1996.

[21] T. Kim, H. Shin, and N. Chang, "Deadline assignment to reduce output jitter of real-time tasks," in *IFAC '00*.

[22] L. Li, M. Lemmon, and X. Wang, "Optimal event-triggered transmission of information in distributed state estimation problems," in *submitted to American Control Conference, 2010.*, 2010.

[23] B. Lincoln and A. Cervin, "Jitterbug: a tool for analysis of real-time control performance," in *Decision and Control, 2002, Proceedings of the 41st IEEE Conference on*, vol. 2, Dec. 2002.

[24] M. Lluesma, A. Cervin, P. Balbastre, I. Ripoll, and A. Crespo, "Jitter evaluation of real-time control systems," in *RTCSA '06*.

[25] C. Locke, D. Vogel, and T. Mesler, "Building a predictable avionics platform in ada: a case study," in *Real-Time Systems Symposium, 1991. Proceedings., Twelfth*, Dec 1991, pp. 181–189.

[26] S. G. Nash and A. Sofer, "Linear and nonlinear programming." McGraw-Hill, 1996.

[27] M. Rabi, G. Moustakides, and J. Baras, "Multiple sampling for estimation on a finite horizon," in *Decision and Control, 2006 45th IEEE Conference on*, Dec. 2006, pp. 1351–1357.

[28] C.-S. Shih and J. W. S. Liu, "State-dependent deadline scheduling," in *RTSS '02: Proceedings of the 23rd IEEE Real-Time Systems Symposium*, 2002, p. 3.

[29] S. Shih and S. Liu, "Acquiring and incorporating state-dependent timing requirements," *Requir. Eng.*, vol. 9, no. 2, 2004.

[30] D. Shin, J. Kim, and S. Lee, "Intra-task voltage scheduling for low-energy, hard real-time applications," *IEEE Des. Test*, vol. 18, no. 2, 2001.

[31] M. Spuri, "Analysis of deadline scheduled real-time systems." INRIA, 1996.

[32] P. Wan and M. Lemmon, "Event-triggered distributed optimization in sensor networks," in *Information Processing in Sensor Networks, 2009. IPSN 2009. International Conference on*, April 2009, pp. 49–60.