

On Controllability and Feasibility of Utilization Control in Distributed Real-Time Systems

Xiaorui Wang[†], Yingming Chen[‡], Chenyang Lu[‡], Xenofon Koutsoukos[§]

[†]Department of Electrical and Computer Engineering, University of Tennessee, Knoxville, TN 37996

[‡]Department of Computer Science and Engineering, Washington University in St. Louis, St. Louis, MO 63130

[§]Department of Electrical Engineering and Computer Science, Vanderbilt University, Nashville, TN 37235
 xwang@ece.utk.edu {yingming, lu}@cse.wustl.edu xenofon.koutsoukos@vanderbilt.edu

Abstract—Feedback control techniques have recently been applied to a variety of real-time systems. However, a fundamental issue that was left out is guaranteeing system controllability and the feasibility of applying feedback control to such systems. No control algorithms can effectively control a system which itself is uncontrollable or infeasible. In this paper, we use the multiprocessor utilization control problem as a representative example to study the controllability and feasibility of distributed real-time systems. We prove that controllability and feasibility of a system depend crucially on end-to-end task allocations. We then present algorithms for deploying end-to-end tasks to ensure the system is controllable and utilization control is feasible for the system. Furthermore, we develop runtime algorithms to maintain controllability and feasibility by reallocating tasks dynamically in response to workload variations such as task terminations and migrations caused by processor failures. We implement our algorithms in a robust real-time middleware and report empirical results on an experimental test-bed. We also evaluate the performance of our approach in large systems using numerical experiments. Our results demonstrate that the proposed task allocation algorithms improve the robustness of feedback control in distributed real-time systems.

I. INTRODUCTION

Recent years have seen rapid growth of applying feedback control techniques to real-time computing and communication systems (e.g., [1][2][3][4][5][6]). In contrast to traditional approaches that rely on accurate knowledge about system workload, control-based solutions can provide robust QoS guarantees in *unpredictable* environments by adapting to workload variations based on dynamic feedback. A particularly suitable platform for feedback control is *Distributed Real-time Embedded (DRE)* systems whose workloads are unknown and vary significantly at run-time. For example, task execution times in vision-based feedback control systems depend on the content of live camera images of changing environments [7].

While existing feedback control work on real-time systems has shown promise, several essential issues regarding feedback control have unfortunately received little to no attention. A fundamental problem is guaranteeing system *controllability*. Controllability is an important property of DRE systems. No control algorithm can achieve its control objective if the system itself is uncontrollable. From a system perspective, uncontrollability is commonly caused by the lack of enough actuators in the system to provide complete control for all desired performance metrics. Along with controllability, it is also important to investigate the *feasibility* problem, which is caused by actuation constraints (e.g., rate constraints of periodic tasks in a DRE system). A controllable system may still fail to achieve its desired performance set points when its actuators saturate due to constraints. Therefore, both

controllability and feasibility are important system properties and have to be guaranteed for DRE systems.

In this paper, we use utilization control as an example to study the controllability and feasibility of DRE systems. End-to-end utilization control [8][9] has been demonstrated to be an effective way to guarantee the *end-to-end deadlines* of all periodic tasks in a soft DRE system, despite uncertainties in task execution times and coupling among processors. In end-to-end scheduling [10], the deadline of an end-to-end task is divided into subdeadlines of its subtasks, and so the problem of meeting the end-to-end deadline is transformed to the problem of meeting the subdeadline of each subtask. A well-known approach to meeting all subdeadlines on a processor is guaranteeing that the real CPU utilization of the processor remains below an appropriate *schedulable utilization bound*, under certain scheduling algorithms (*i.e.*, RMS) [10]. In a real DRE system where the invocation rates of some tasks may be adjustable within certain ranges, it is usually preferable to control the utilizations of all processors to stay slightly below their schedulable bounds so that the task rates can become as high as possible without causing any deadline misses. As a result, the value of the system can be maximized [11]. Utilization control can also be deployed in middleware systems to support Quality of Service portability [4], or enhance system survivability by providing overload protection against workload fluctuation [12].

In utilization control, an uncontrollable DRE system is a system for which it is impossible to find a sequence of task rates that take the utilizations of all processors to desired set points specified by the applications. An infeasible system is interpreted as a system which fails to achieve its set points because the invocation rates of its tasks saturate on the boundaries of the allowed rate ranges. As a result of uncontrollability or infeasibility, some processors may become overloaded while some other processors may be poorly utilized at the same time. This kind of workload imbalance and consequent *deadline misses* may cause serious problems in real-time systems. With controllability and feasibility guarantees, we can maximize the system value by running all tasks in highest possible rates without causing deadline misses [11].

In this paper, we show that system controllability and feasibility can be guaranteed by adjusting certain system configurations such as end-to-end task allocation. Specifically, the contributions of this paper are five-fold:

- We formulate the controllability and feasibility problem as an end-to-end task allocation problem.
- We design task allocation algorithms to ensure a system

is controllable and feasible.

- We analyze the impact of workload variations on controllability and feasibility and design efficient online algorithms to dynamically adjust task allocation.
- We integrate our algorithms with a robust real-time middleware to maintain system controllability and feasibility for deployed DRE applications.
- We present both empirical and numerical results to demonstrate the effectiveness of our algorithms.

The rest of this paper is structured as follows. We first review related work in Section II. We then formulate the controllability and feasibility problems in Section III. Section IV analyzes the two problems to provide a theoretical foundation for algorithm design. Section V presents our offline task allocation algorithms. Section VI presents our online allocation adjustment algorithms. Section VII introduces the implementation of the algorithms in a real-time middleware system. Section VIII presents our numerical and empirical results. Finally, Section IX summarizes the paper.

II. RELATED WORK

Control theoretic approaches have been applied to a number of computing systems. A survey of feedback performance control in computing systems is presented in [1]. Many projects that applied control theory to real-time scheduling and utilization control are closely related to this paper. Steere et al. and Goel et al. developed feedback-based schedulers [6][13] that guarantee desired progress rates for real-time applications. Abeni et al. presented control analysis of a reservation-based feedback scheduler [2][14]. Lu et al. developed a middleware service which adopts feedback control scheduling algorithms to control CPU utilization and deadline miss ratio [4]. Feedback control has also been successfully applied to power control [15][16] and digital control applications [3]. For systems requiring discrete control adaptation strategies, hybrid control theory has been adopted to control state transitions among different system configurations [17].

Stankovic et al. [18] and Lin et al. [19] proposed feedback control scheduling algorithms for *distributed* real-time systems with independent tasks. These algorithms do not address the interactions between processors caused by end-to-end tasks, which are commonly available in DRE systems. Diao et al. developed a Multi-Input-Multi-Output (MIMO) control algorithm for load balancing in data servers [20]. However, their algorithm cannot handle actuation constraints which are also common in DRE systems. In contrast, our previous work EUCON [8] and DEUCON [9] are specially designed to handle the constrained MIMO utilization control problem for multiple processors that are coupled due to end-to-end tasks.

Both controllability and feasibility are important system properties wherever MIMO control is necessary. This paper presents the first study on the controllability and feasibility of DRE systems. Recently, Karamanolis et al. raised the problem of designing controllable systems [21]. However, that paper focused only on some practical issues regarding how to get better control performance for Single-Input-Single-Output (SISO) systems. In contrast, our work investigates the fundamental issues defined in control theory such as whether

it is possible to control a DRE system and how to make an uncontrollable system controllable. Feasibility is another important issue. While the feasibility of scheduling tasks [22] has been addressed before in real-time community, in this paper, we focus on the feasibility of controlling DRE systems.

We formulate the controllability and feasibility problem as a task allocation problem in DRE system. Task allocation is a classical problem which has been discussed by many existing projects (e.g. [23][24][25]). The difference between our work and those related projects is that we are trying to guarantee system controllability and feasibility, instead of minimizing communication cost or ensuring load balancing.

III. PROBLEM FORMULATIONS

In this section, we first introduce the system model employed in our work. We then formulate the controllability and feasibility problems.

A. System Model

We adopt an end-to-end task model [10] implemented by many DRE applications. A system is comprised of m periodic tasks $\{T_i | 1 \leq i \leq m\}$ executing on n processors $\{P_i | 1 \leq i \leq n\}$. Task T_i is composed of a chain of sub-tasks $\{T_{ij} | 1 \leq j \leq n_i\}$ located on different processors. The release of subtasks is subject to precedence constraints, i.e., subtask T_{ij} ($1 < j \leq n_i$) cannot be released for execution until its predecessor subtask $T_{i,j-1}$ is completed. If a non-greedy synchronization protocol (e.g., release guard [26]) is used to enforce the precedence constraints, all the subtasks of a task share the same rate as the first subtask. Therefore, the rate of a task (and all its subtasks) can be adjusted by changing the rate of its first subtask.

Our task model has two important properties. First, while each subtask T_{ij} has an *estimated* execution time c_{ij} available at design time, its *actual* execution time may be different from its estimation and vary at run time. Modeling such uncertainty is important to DRE systems operating in unpredictable environments. Second, the rate of a task T_i may be dynamically adjusted within a range $[R_{min,i}, R_{max,i}]$. This assumption is based on the fact that the task rates in many applications (e.g., digital control [27], sensor update, and multimedia [28]) can be dynamically adjusted without causing system failure.

We assume that each task T_i has a *soft* end-to-end deadline related to its period. In an end-to-end scheduling approach [26], the deadline of an end-to-end task is divided into subdeadlines of its subtasks [29][30]. Hence the problem of meeting the deadline can be transformed to the problem of meeting the subdeadline of each subtask. A well known approach for meeting the subdeadlines on a processor is to ensure its utilization remains below its schedulable utilization bound [31][32]. Utilization control is not designed to handle network delays. Network delay may be handled by treating each network link as a processor [26], or by considering the impact of worst-case network delay in subdeadline assignment.

In our previous work [8][9], we have modeled the utilization control problem by establishing difference equations to capture the dynamics of a DRE system with n processors and m end-to-end periodic tasks. The DRE system is described by the following MIMO model:

$$\mathbf{u}(\mathbf{k} + 1) = \mathbf{u}(\mathbf{k}) + \mathbf{G}\mathbf{F}\Delta\mathbf{r}(\mathbf{k}) \quad (1)$$

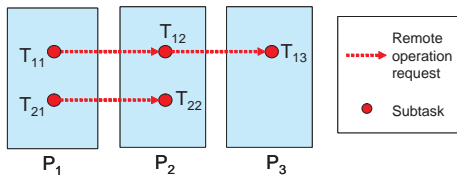


Fig. 1. An example DRE system

The vector $\Delta \mathbf{r}(\mathbf{k})$ represents the changes in task rates. The *subtask allocation matrix*, \mathbf{F} , is an $n \times m$ matrix, where $f_{ij} = c_{jl}$ if a subtask T_{jl} of task T_j is allocated to processor P_i , and $f_{ij} = 0$ if no subtask of task T_j is allocated to processor P_i . \mathbf{F} captures the *coupling* among processors due to end-to-end tasks. $\mathbf{G} = \text{diag}[g_1 \dots g_n]$ where g_i represents the ratio between the change in the actual utilization and its estimation. The exact value of g_i is *unknown* due to the unpredictability in execution times. Note that \mathbf{G} describes the effect of workload uncertainty in a DRE system.

As an example, the DRE system shown in Figure 1 is modeled by (1) with the following parameters:

$$\mathbf{u}(\mathbf{k}) = \begin{bmatrix} u_1(k) \\ u_2(k) \\ u_3(k) \end{bmatrix}, \Delta \mathbf{r}(\mathbf{k}) = \begin{bmatrix} \Delta r_1(k) \\ \Delta r_2(k) \end{bmatrix}$$

$$\mathbf{F} = \begin{bmatrix} c_{11} & c_{21} \\ c_{12} & c_{22} \\ c_{13} & 0 \end{bmatrix}, \mathbf{G} = \begin{bmatrix} g_1 & 0 & 0 \\ 0 & g_2 & 0 \\ 0 & 0 & g_3 \end{bmatrix},$$

B. Controllability Problem

In a MIMO control system, if a sequence of control input variables can be found that takes all control output variables from any initial conditions to any desired final conditions in a finite time interval, the MIMO system is said to be *controllable* [33], otherwise the system is uncontrollable. According to control theory, a MIMO system $\mathbf{x}(\mathbf{k} + 1) = \Phi \mathbf{x}(\mathbf{k}) + \Gamma \mathbf{v}(\mathbf{k})$ with n control outputs $[x_1(k) \dots x_n(k)]$ and m control inputs $[v_1(k) \dots v_m(k)]$ is controllable iff the rank of its *controllability matrix* $\mathbf{C} = [\Gamma \quad \Phi \Gamma \quad \dots \quad \Phi^{n-1} \Gamma]$ is n , the order of the system [33].

Definition A DRE system is *controllable* if there exists a sequence of task rates that take the utilizations of all processors in the system to any desired utilization set points.

In our system model (1), we assume that matrix \mathbf{G} is the identity matrix $\text{diag}[1 \dots 1]$ because system gains are unknown at design time [8]. We will show later that system gains do not affect system controllability. Hence, the controllability matrix of the system model is an $n \times nm$ matrix $\mathbf{C} = [\mathbf{F} \quad \mathbf{F} \quad \dots \quad \mathbf{F}]$. In order to have a controllable DRE system, we have to guarantee the rank of the controllability matrix \mathbf{C} is n , the number of processors in the system.

C. Feasibility Problem

In control theory, the condition of controllability is based on the assumption that there are no actuation constraints (i.e. rate constraints). However, as introduced in our task model, the rate of each task T_i can only be adjusted within a range $[R_{min,i}, R_{max,i}]$, namely $R_{min,i} \leq r_i \leq R_{max,i}$, ($1 \leq i \leq m$). Therefore, a system proved to be controllable may still not be able to achieve the desired utilization set points, as the task rates may saturate.

Definition If a controllable DRE system cannot get to the set points because the rates of one or more of its tasks saturate at the rate boundaries, we say it is *infeasible* to achieve the set points for the system. Otherwise we say utilization control is *feasible* for the system.

In utilization control, although a feasible system is more preferred, it is actually sufficient to just keep the processor utilizations below the desired set points, because overload may cause *deadline misses* and thus is much more undesirable than underutilization in DRE systems. Therefore, in this paper, we focus on *practical feasibility* defined below.

Definition Utilization control is *practically feasible* for a DRE system whose task rate constraints allow the utilizations of all processors to either get to or stay below the desired set points.

An effective solution to the feasibility problem is subtask allocation adjustment. For instance, if a processor in the system remains overloaded because all its subtasks already reach their lower rate boundaries, we may move one subtask away from the processor so that it can have less workload and then recover from overload. While this solution is sufficient for systems where execution times never change, it has to be extended for DRE systems whose execution times may vary unpredictably. In such systems, a previously feasible system may become infeasible at runtime. Continuously monitoring feasibility and migrating subtasks would introduce large runtime overhead. Hence, instead of guaranteeing a system to be feasible for certain execution times, we try to increase the likelihood of the system being feasible even under variations, so that we can reduce the overhead of moving subtasks later at runtime.

We first introduce several definitions.

Definition The *minimum estimated utilization* of processor P_i is defined as the sum of the products of the estimated execution times and the minimum allowed rates of all subtasks on the processor. Specifically, $u_{min,i} = \sum_{T_{jl} \in S_i} c_{jl} R_{min,j}$, where S_i represents the set of subtasks located on processor P_i .

Definition The difference between the set point of processor P_i and its minimum estimated utilization is defined as its *feasibility margin*. Specifically, $\text{margin} = u_{min,i} - B_i$.

When the variations of execution times cause the utilization of P_i to deviate from its set point B_i , a large feasibility margin can give task rates enough space for adaptation so that the utilization can reconverge to the set point. Hence, we want to adjust subtask allocations so that the task rates can stay as far away from their lower boundaries as possible when processors settle at their set points. In other words, our goal is to maximize the feasibility margin for all processors in order to maximize the chance of having a feasible system under variations. If we consider a DRE system infeasible when any processor becomes infeasible, the feasibility problem becomes a problem of maximizing the smallest feasibility margin among all processors in the system. Hence, the feasibility problem can be formulated as finding a subtask allocation to optimize the following objective.

$$\max \left(\min_{1 \leq i \leq n} (|B_i - u_{min,i}|) \right) \quad (2)$$

This optimization problem is subject to two constraints. The first one is a utilization constraint. The minimum estimated

utilization $u_{min,i}$ of each processor P_i is not allowed to be larger than B_i , otherwise the system would be infeasible based on the estimated execution times. The second one is a resource constraint. As a common practical issue in DRE systems, each subtask can only be allocated to a specific set of processors due to resource availability. Note that the set point B_i of each processor P_i is a function of its number of subtasks when the system is scheduled by some algorithms like RMS [32].

IV. THEORETICAL ANALYSIS

In this section, we present the theoretical analysis of the controllability and feasibility problems, which provides a foundation for the design of our task allocation algorithms.

A. Controllability Condition

We first analyze the controllability matrix to see how we can guarantee its rank to be equal to n , the number of processors in the system.

Theorem 4.1: A DRE system is controllable if and only if the rank of its subtask allocation matrix \mathbf{F} is n .

Proof: We prove that the rank of the subtask allocation matrix \mathbf{F} is equal to the rank of the controllability matrix $\mathbf{C} = [\mathbf{F} \ \mathbf{F} \ \dots \ \mathbf{F}]$. We first transform \mathbf{C} to a matrix $\mathbf{C}' = [\mathbf{F} \ 0 \ \dots \ 0]$ by subtracting every column of the first \mathbf{F} from the rest \mathbf{F} 's. Since elementary transformations do not change the rank of a matrix, \mathbf{C} has the same rank as \mathbf{C}' . Clearly, \mathbf{C}' has the same rank as \mathbf{F} . Hence, the system is controllable if and only if the rank of \mathbf{F} is n . ■

Example The DRE system shown in Figure 1 is not controllable because the rank of its subtask allocation matrix \mathbf{F} is 2, while there are 3 processors in the system.

Corollary 4.2: A DRE system with n processors and m end-to-end tasks is *uncontrollable* if $m < n$.

In other words, any DRE system must have more tasks (control inputs) than processors (control outputs) in order to be controllable. Note that $m > n$ is a necessary but not sufficient condition of controllability. When this condition is met, a system is not necessarily controllable. However, as we will show later, we may adjust the subtask allocation matrix of the system to make it controllable. Hence, through task allocation, a system can achieve both feasibility and controllability. Note that when there are not enough tasks (i.e., $m < n$), we can easily use *fewer* processors to run the same DRE applications so that the system becomes controllable and the system value could be maximized [11] with less computing resource.

B. Structural Controllability

As the algorithms we are proposing are used in DRE systems, here we narrow down our attention from *complete controllability* (i.e. controllability defined before) to *structural controllability* [34]. A system is structurally controllable if there exists another system which is structurally equivalent to the system and is completely controllable [34]. Two systems are structurally equivalent if there is a one-to-one correspondence between the locations of the fixed zeros and nonzero items in their controllability matrices [34].

A structurally controllable system may not always be controllable because the elements of two rows/columns of its

TABLE I
IMPACT OF DIFFERENT TYPES OF WORKLOAD VARIATIONS

Variations	Feasibility	Controllability
Task arrival	harmful	harmless
Task termination	harmless	harmful
Processor failure	harmless	conditional harmful
Exec time variation	harmful	harmless

controllability matrix could happen to be proportional, which causes its rank to be smaller than the system order. In our system model, two rows are proportional means that the subtasks on two processors belong to exactly a same set of tasks and the execution times of corresponding subtasks are strictly proportional to each other. Two columns are proportional means that two tasks are deployed on exactly a same set of processors and the execution times of their subtasks on a same processor are strictly proportional to each other. In general, finding proportional rows and columns is computationally expensive [34]. Fortunately, in DRE systems, such cases are very rare due to the high variations in task execution times on modern processors. Therefore, in practice we can easily identify potentially proportional rows and columns in the allocation matrix based on the configuration of DRE applications. Once we identify a set of proportional rows or columns, we combine them as a single row or column for our analysis. As a result, structural controllability ensures controllability in the modified allocation matrix. Hence, our analysis and algorithms only ensure structural controllability. We use controllability and structural controllability interchangeably hereinafter.

C. Impact of Workload Variations

In DRE systems, workload variations often happen and may change subtask allocations which in many ways affect system feasibility or controllability. Hence, it is necessary to investigate their possible impact on system feasibility and controllability. In this paper, we focus on four common types of workload variations: task arrival, task termination, processor failure, and execution time variation. In the following, we analyze the possible impact of each type of variation on controllability as well as on feasibility. If a type of variation does not affect controllability or feasibility, we define it as *harmless* to controllability or feasibility. Otherwise we say it is *harmful*. The categorization of harmless and harmful variations allows us to execute our runtime adjustment algorithms only when harmful variations happen, so we can minimize the runtime overhead.

We first analyze the impact of workload variations on controllability.

Theorem 4.3: Task arrival in a DRE system is harmless to controllability.

Proof: Dynamically adding a task to a DRE system is equivalent to adding a new column to the subtask allocation matrix \mathbf{F} , which does not reduce the rank of \mathbf{F} . ■

Therefore, if the system is controllable, it remains controllable after task arrivals.

Theorem 4.4: Task termination in a DRE system is harmful to controllability.

Proof: Removing a column from the allocation matrix may reduce the rank of the matrix. ■

Theorem 4.5: Processor failure is harmful to controllability if the failed processor has more than $m - n + 2$ subtasks, where m and n are the numbers of tasks and processors, respectively.

Proof: Removing a failed processor from a DRE system leads to removing a row from the allocation matrix \mathbf{F} . Assuming all tasks having subtasks on the failed processor terminate, the failure also results in removing several columns from the allocation matrix. If the rank of matrix \mathbf{F} is originally n , any of its submatrices with size as $n' \times m'$ has the rank as $\min(n', m')$. We assume that after the processor failure, the allocation matrix has its rank as $\min(n - 1, m')$. In order for the matrix to have a rank less than $n - 1$, we need to have $m'' \leq n - 2$. Hence, we need to terminate at least $m - m'' = m - n + 2$ tasks. ■

Execution time variation is harmless to structural controllability because it does not change the structure of the controllability matrix. The impact of different types of workload variations on controllability is summarized in Table I.

We now investigate feasibility by finding which types of variation may reduce the feasibility margin of a system. Clearly, any variations that increase system workload may cause the feasibility margin to decrease. Therefore, execution time variation, task arrival are harmful to feasibility because they may increase the workload of some processors in the system. Task termination reduces the workload of some processors so it is harmless. Processor failure causes task termination so it is also a harmless variation to feasibility. The impact of different types of workload variations on feasibility is also summarized in Table I.

V. OFFLINE TASK ALLOCATION ALGORITHMS

Both the controllability and feasibility problems rely on the development of novel subtask allocation algorithms. In this section, we propose a two-step approach to allocate subtasks in a DRE system. The first algorithm aims to increase the feasibility margin. The second algorithm ensures controllability by adjusting the allocation while minimizing the influence on the feasibility margin.

A. Increasing Feasibility Margin

As suggested by the optimization objective in (2), the feasibility problem is related to both load balancing [25] and variable-size bin packing [10]. It is related to the variable-size bin packing problem because it needs to pack all subtasks to processors and the capacity of a processor shrinks when its number of subtasks increases. It differs from bin packing because the goal here is to balance the workload on each processor, instead of using fewest processors. The problem is closer to the load balancing problem but the difference is that we are trying to maximize the smallest feasibility margin instead of minimizing the highest utilization among all processors. Clearly our problem can be reduced to the load balancing problem which is an NP-hard problem [25]. Here we present a feasibility algorithm which is extended from the existing Max-Min algorithm used for load balancing [25]. The Max-Min algorithm has a good trade-off between solution quality and computation overhead [25].

In our feasibility algorithm, we first sort all subtasks based on their minimum estimated utilization, $u_{min,jl} = c_{ij}R_{min,j}$.

```

(1) Sort all subtasks  $T_{jl}$  based on  $u_{min,jl}$ 
    Enqueue all subtasks in decreasing order
(2) While there is at least one subtask in the queue
    Pop up the first subtask  $T_{jl}$  (which has the largest  $u_{min,jl}$ )
    For each processor  $P_q = cons[T_{jl}, q + +]$ 
        If  $u_{current,q} + u_{min,jl} \leq B_q$ 
             $u_{new,q} = u_{current,q} + u_{min,jl}$ 
            Feasibility margin of  $P_q = B_q - u_{new,q}$ 
        End if
    End for
    Allocate  $T_{jl}$  to proc  $P_i$  with the largest feasibility margin
    If  $T_{jl}$  cannot be allocated to any processor
        Algorithm fails
    End while

```

Fig. 2. Pseudo code of the algorithm to increase feasibility margin

Then we pick the subtask with the currently largest $u_{min,jl}$ and allocate it to the processor that has the largest feasibility margin after this allocation. We continue the process until all the subtasks are allocated. Note that the allocation at each step is subject to both the utilization and resource constraints. The utilization constraint is checked at each step when a subtask is allocated to a processor. If the largest feasibility margin after allocating a subtask to the system becomes negative, the algorithm fails. In that case, more advanced algorithms such as Mixed Integer Programming may be adopted to provide a solution at a cost that could be comparable to the cost of exhaustive search [25].

The detailed algorithm is shown in Figure 2. The resource constraints are represented by an $s \times p$ matrix $cons$, where s is the total number of subtasks in the system and p is the number of processors on which a subtask can execute. Each element $cons[T_{jl}, q]$ is the q^{th} processor that the subtask T_{jl} can be allocated to. We assume all processors are homogeneous here, but the algorithm can be easily extended to systems with heterogeneous processors.

Now we analyze the complexity of this algorithm. The complexity of step 1 is $O(s \log s)$, where s is the total number of subtasks in the system. The complexity of step 2 is sp , where p is the number of processors that a subtask can be allocated to. Hence, the time complexity of the feasibility algorithm is $O(\max(s \log(s), sp))$.

B. Ensuring Controllability

After our feasibility algorithm successfully allocates all subtasks, we check the allocation matrix \mathbf{F} to determine whether the current workload configuration is controllable. If it is, the workload is accepted for deployment on the target DRE system. Otherwise we process the workload with a controllability adjustment algorithm. In the algorithm, for every processor, we search all tasks which have subtasks on the processor to find one task to *dedicate* to the processor. The task is called the *dedicated task* of the processor and its subtasks on the processor are called the *dedicated subtasks*. A task can only be dedicated to one processor. For those processors which fail to find dedicated tasks, we migrate subtasks of some non-dedicated tasks from other processors to them so they can have those tasks dedicated to them.

Theorem 5.1: If every processor in a system has a dedicated task, the system is controllable.

Proof: If every processor has a dedicated task, the allocation matrix can be proved to have full rank (i.e. its rank

equals the order of the system). To prove that, we can move the columns of the matrix so that all tasks can place their dedicated subtasks on the diagonal of the allocation matrix. As described in Section IV-B for structural controllability, there are no two rows or columns that are proportional to each other in the matrix. As a result, a matrix has full rank if there is no zero on its diagonal. Hence, a system is guaranteed to have controllability if every processor has a dedicated task. ■

Note that Theorem 5.1 is both a sufficient and a necessary condition for structural controllability. The rationale behind dedicating tasks to processors can also be explained from a system perspective, each processor can rely on the rate adaption of its dedicated task to achieve its utilization set point, if we assume there is no rate constraints.

Our controllability algorithm first sorts all processors based on their numbers of subtasks. The algorithm dedicates tasks to the processors with fewer subtasks first, because that may reduce the necessity of moving subtasks later. The second step preprocesses the allocation matrix to speed up the later dedicating step. For every processor/task pair in the allocation matrix, we search for a candidate subtask by assuming this processor fails to find its dedicated task and needs a subtask of this task to be moved to the processor. Since subtask migration may affect the feasibility margin of a system, we want to minimize the impact by moving the *best candidate subtask*, which has the smallest minimum estimated utilization and is allowed by the resource constraints to run on the target processor. Hence, for every element (i.e. processor/task pair) in the allocation matrix F , we insert some attributes such as the location of the best candidate. The information will speed up the search process if a processor loses its dedicated task and needs to find a new one at runtime. In the third step, we sort all existing subtasks on each processor based on their minimum estimated utilization. For those previous zero elements (i.e. no subtask exists there), we sort them based on the minimum estimated utilizations of their best candidate subtasks. The reason for sorting them is also to speed up the search process, which is especially important for extending the algorithm to support online task reallocation (as described in Section VI). In the last step, we start the dedicating process. If no task can be dedicated to a processor, we move the best candidate subtask of the first non-dedicated task to the processor. This subtask is guaranteed to have the smallest minimum estimated utilization and so should only cause small impact on the system feasibility margin. The detailed algorithm is shown in Figure 3.

Now we analyze the time complexity of this algorithm. The complexity of the four steps are $O(n \log n)$, $O(sp)$, $O(nm \log m)$ and $O(nm)$, respectively. Hence, the time complexity of the whole controllability algorithm is $O(\max(sp, nm \log m))$.

VI. ONLINE ALLOCATION ADJUSTMENTS

Even though the algorithms presented in the previous section can effectively preprocess workloads before deployment to increase feasibility margin and guarantee controllability, there are two issues we have to address at runtime. First, as the subtask allocation matrix may change at runtime due to

```

(1) Replace all zero elements with maximum integer in matrix  $\mathbf{F}$ 
    Sort all processors in increasing order of number of subtasks

(2) For each subtask  $T_{jl}$  in resource constraints matrix  $cons$ 
    For each of its allowed processor  $P_q$ 
         $\mathbf{F}(q, j) = \min\{u_{min,jl}, \mathbf{F}(q, j)\}$ 
        If  $u_{min,jl} < \mathbf{F}(q, j)$ , best candidate subtask of  $\mathbf{F}(q, j) = T_{jl}$ 
    End for

(3) For each processor in allocation matrix  $\mathbf{F}$ 
    For all existing subtasks
        Sort their subtasks in decreasing order of  $u_{min,jl}$ 
    For all previous zero elements
        Sort their best candidates in increasing order of  $u_{min,jl}$ 
    End for each processor

(4) For each processor  $P_i$  in the allocation matrix  $\mathbf{F}$ 
    For each task  $T_j$  already having subtasks on  $P_i$ 
        In decreasing order of  $T_j$ 
            If  $T_j$  is non-dedicated, dedicate  $T_j$  to  $P_i$ 
    End for
    If all tasks are already dedicated to other processors
        For each previous zero element
            In increasing order of  $u_{min,jl}$ 
                If the task is non-dedicated
                    Move the best candidate subtask to  $P_i$ 
                    Dedicate the task to  $P_i$ 
            End if
        End for
        If cannot find a non-dedicated task, algorithm fails
    End if
    End for each processor

```

Fig. 3. Pseudo code of the algorithm to ensure controllability

workload variations such as task termination, a workload processed with the offline algorithms may become uncontrollable or infeasible. Hence, controllability and feasibility have to be maintained at runtime. Second, as analyzed in the previous section, the controllability and feasibility algorithms introduce some computation overhead. While it is acceptable to run the two algorithms for preprocessing, we need to develop more efficient ones to incrementally adjust workload at runtime.

A. Feasibility Adjustment

According to Table I, two types of variations may reduce the feasibility margin of a system. Among them, execution time variation has been handled by the feasibility margin which is designed to tolerate possible variations to the maximum degree, so that we can avoid the runtime monitoring overhead. To minimize the impact of task arrivals on feasibility and reduce runtime cost at the same time, here we run our feasibility algorithm incrementally only to allocate new tasks for a balance between the two conflicting goals. The algorithm presented in Figure 2 is adopted to sort and allocate only the new arriving tasks. Hence, the computation overhead is now only $O(\max(qn \log(qn), qnp))$, where q is the number of arriving tasks.

B. Controllability Maintenance

According to Table I, there are two situations that may jeopardize the system controllability: task termination and processor failure. The reason that processor failure is harmful is that it may cause one or more tasks to terminate. Hence, we only need to check and maintain controllability when tasks terminate, which can be handled incrementally by the runtime task reallocation algorithm shown in Figure 4. The

```

(1) Remove the terminated task from the allocation matrix
(2) If this task is not dedicated to a processor
    Algorithm successfully ends
(3) Else
    For the processor that the terminated task was dedicated to
    Run step 4 (Fig. 3) to find a dedicated task for the processor
    End if

```

Fig. 4. Pseudo code of the algorithm to maintain controllability online

time complexity of the controllability maintenance algorithm is $O(m)$, where m is the number of tasks in the system.

VII. MIDDLEWARE IMPLEMENTATION

Both the controllability and feasibility algorithms have been integrated in the FC-ORB middleware [12]. FC-ORB implements an end-to-end utilization control algorithm called EUCON [8]. Like any other feedback utilization control algorithm developed for DRE systems, the EUCON algorithm may experience the controllability and feasibility problems and is used as an example platform to demonstrate the effectiveness of our algorithms. The two algorithms are integrated with the FC-ORB controller which is running on a different processor from the controlled system. The middleware architecture of the extended FC-ORB system is shown in Figure 5.

The controllability maintenance algorithm is implemented as a controllability handler. Based on our analysis in the previous section, only task termination affects the controllability of a system. Consequently, the controllability handler is invoked whenever a task terminates at runtime. When that happens, the handler removes the terminated tasks from the control model, and then moves proper subtasks to maintain system controllability. After that the handler re-initializes the controller and resumes the feedback control loop. Similarly, the feasibility adjustment algorithm has been implemented as a feasibility handler to do incremental subtask allocation whenever new tasks are admitted to the system.

To support online task reallocation, we extended FC-ORB to handle subtask migrations demanded by the controller. The migration mechanism works as follows. Each subtask can have a primary instance and a few backup instances on the processors where it has the required resource. In the normal mode, each subtask pushes remote operation requests only to the primary instance of its successor. As a result, the backup instances do not receive any requests and their threads remain idle. After a task migration decision is made by the controller, the predecessor of the migrated subtask switches the connection to the desired backup instance and sends the remote operation requests to it. In the case when the first subtask of a task has to be moved, the controller activates the proper backup instance of the subtask.

VIII. EXPERIMENTS

In this section, we present the results of two sets of experiments. First, we evaluate the offline subtask allocation algorithms using numerical experiments, which allow us to use a large number of randomly generated workloads to stress-test the algorithms in large systems. Second, we present empirical results based on the extended FC-ORB middleware system to demonstrate the effectiveness of the online algorithms.

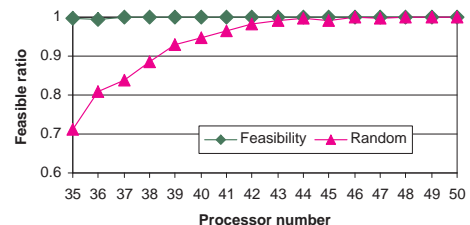


Fig. 6. Feasible ratio under different processor numbers

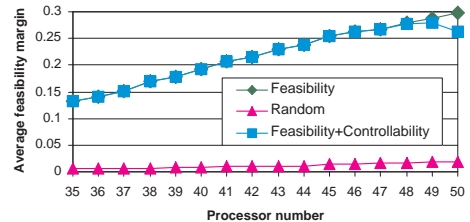


Fig. 7. Feasibility margin under different processor numbers

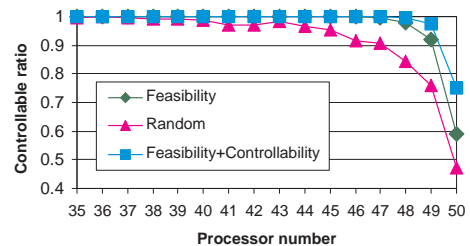


Fig. 8. Controllable ratio under different processor numbers

A. Numerical Results

In all experiments presented in this subsection, the number of tasks has been fixed at 50 (i.e. $m = 50$), while the number of subtasks of each task varies uniformly from 1 to 7. For each task, its lower rate bound, $R_{min,j}$, is randomly generated between 0.01 and 0.1Hz. For each subtask, its minimum estimated utilization varies randomly between 5% and 15% and its execution time is calculated based on its rate and its minimum estimated utilization. Each subtask can only be executed on 5 processors (i.e. $p = 5$), which are randomly chosen from all processors. Because any system with more processors than tasks is uncontrollable, we vary the number of processors (i.e. n) from 35 to 50 to examine the performance of our algorithms when the average number of subtasks on each processor changes. For each value of n , 500 different workload configurations are randomly generated and tested. The schedulable utilization bound of RMS [32], namely $B_i = m_i(2^{1/m_i} - 1)$, is used as the utilization set point of each processor P_i to avoid deadline misses, where m_i is the number of subtasks on this processor.

We compare our algorithms against a baseline algorithm called *Random*. *Random* first ensures there is no *idle* processor by randomly allocating one subtask to each processor, because otherwise the system is clearly uncontrollable. Then the rest subtasks are also randomly allocated to processors under the utilization and resource constraints. If a subtask cannot be allocated to any processor, the algorithm fails.

We first examine the *feasible ratio* (i.e. the fraction of task allocations that are feasible) under both our feasibility algorithm and *Random*. A workload resulted from an allocation is *feasible* if the minimum estimated utilizations of all processors are equal to or lower than their schedulable bounds. Figure 6

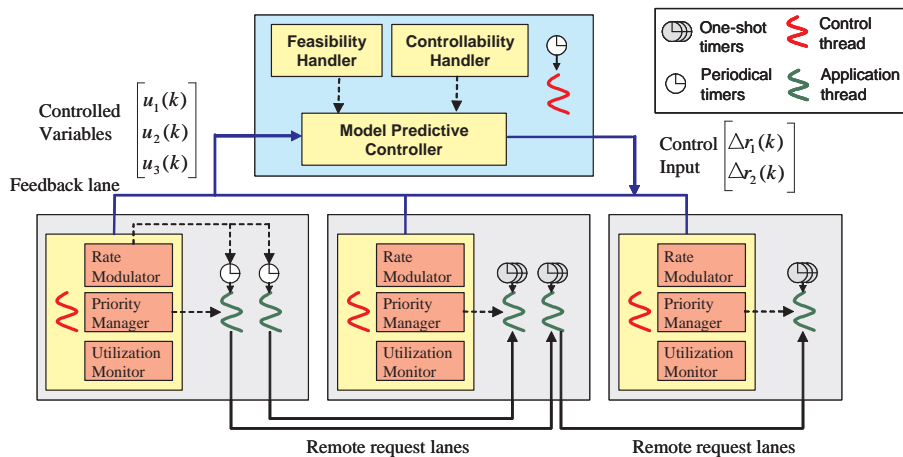


Fig. 5. Middleware architecture of the extended FC-ORB system

shows that the feasibility algorithm achieves higher feasibility ratio than Random when the number of processors is smaller than 44. For example, when processor number is 35, more than 30% of workloads are not feasible under Random, but the ratio is only 1% under the feasibility algorithm. The reason is that when the number of processors is small, each processor has more subtasks, which decreases the probability for Random to find feasible solutions.

As discussed in section III-C, the main goal of our feasibility algorithm is to increase the feasibility margin. Figure 7 plots the average feasibility margin of 250 workloads which are feasible under both the feasibility algorithm and Random. The average feasibility margin under Random is much smaller than that under the feasibility algorithm. That means the feasibility algorithm results in workloads which can tolerate much larger execution time variations. For example, with 48 processors, the workload generated by the feasibility algorithm can remain feasible even when task execution times increase by 28%. When the number of processors increases, the difference becomes larger. That is because when each processor has fewer subtasks, the space for the feasibility algorithm to improve becomes larger.

We then compare the *controllable ratio* (i.e. the fraction of task allocations that are controllable) under Random, the feasibility algorithm and the integrated feasibility and controllability algorithm. Same as before, Random and the feasibility algorithm are applied to all randomly generated workloads without any concern of controllability. In contrast, the integrated algorithm adopts the controllability algorithm introduced in Section V-B to reallocate the subtasks if the workload processed by the feasibility algorithm is diagnosed to be uncontrollable. Figure 8 shows that the controllability algorithm reduces the uncontrollable cases significantly. For example, with 50 processors, the controllable ratio has been increased more than 10%. In addition, the feasibility algorithm can also help improve controllability as its controllable ratio is much higher than Random.

As discussed in Section V, the controllability algorithm will have some impact to the feasibility margin though the algorithm is designed to minimize the impact. Figure 7 shows the impact is only roughly 3%. This result demonstrates that the controllability algorithm can improve system controllability

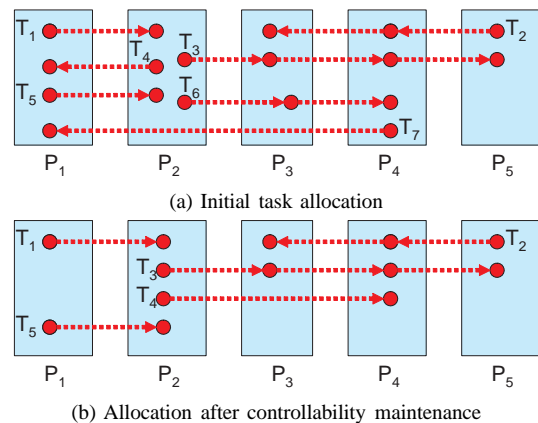


Fig. 9. Workload configuration and variations in controllability experiments significantly only at negligible cost of feasibility margin.

B. Empirical Results

In this subsection, we present the experiments conducted on a real DRE system implemented based on the extended FC-ORB middleware. We first introduce the experimental configurations. We then present the experimental results on controllability and feasibility, respectively by contrasting systems with and without the dynamic algorithms.

1) *Experimental set-up*: We perform our experiments on a testbed of six PCs. All applications and the ORB service run on four Pentium-IV PCs (P_1 to P_4) and one Celeron PC (P_5). P_1 and P_4 are 2.80GHz while P_2 and P_3 are 2.53GHz. P_1 to P_4 all are equipped with 512KB cache and 512MB RAM. P_5 is 1.80GHz and has 128KB cache and 512MB RAM. All application PCs run RedHat Linux 2.4.22. The controller is located on another Pentium-IV 2GHz PC with 512KB cache and 256MB RAM. The controller PC runs Windows XP Professional with MATLAB 6.0. P_1 to P_4 are connected via an internal switch and communicate with P_5 and the controller PC through the departmental 100Mbps LAN.

Our experiments run a medium-sized workload that comprises 7 end-to-end tasks (with a total of 18 subtasks). Figure 9(a) shows how the 7 tasks are distributed on the 5 application processors. The detailed workload parameters are not shown due to space limitations. The subtasks on each processor are scheduled by the RMS algorithm [32]. Each task's end-to-end deadline is $d_i = n_i/r_i(k)$, where n_i is the number of subtasks in task T_i and $r_i(k)$ is the current rate of T_i . Each end-to-end

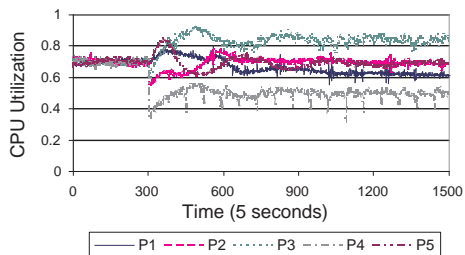


Fig. 10. System becomes uncontrollable after task termination

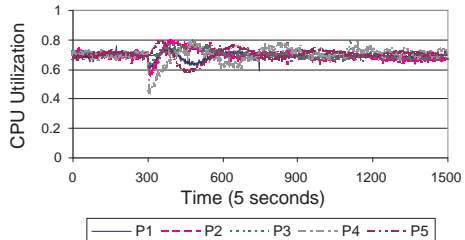


Fig. 11. System becomes controllable after controllability maintenance

deadline is evenly divided into subdeadlines for its subtasks. The resultant subdeadline of each subtask T_{ij} equals its period, $1/r_i(k)$. The utilization set point of every processor is set as 0.7.¹ All (sub)tasks meet their (sub)deadlines if the desired utilization on every processor is enforced. The sampling period of the utilization control service is $T_s = 5$ seconds.

2) *Controllability*: In our first experiment, we run the original FC-ORB with an initial workload shown in Figure 9(a). The rates of all tasks in the workload are selected based on their execution times so that the utilizations of all processors can be initially close to their set points. At time 300×5 seconds, task T_6 and T_7 terminate so the workload becomes uncontrollable. From the experimental results shown in Figure 10, we can see that only the utilizations of processor P_2 and P_5 converge to the desired set points. The utilization of P_1 stays slightly below the set point. P_4 is severely underutilized as its utilization is just 50% while P_3 is overloaded. As processor overload may cause *deadline misses* as shown in our previous work [9], controllability has to be maintained at runtime.

In the second experiment, we run our extended middleware system with the controllability handler activated. All configurations remain the same as in the first experiment. In the controllability analysis, task T_7 is not dedicated to any processor so its termination is ignored. However, task T_6 is dedicated to processor P_4 so we have to migrate a subtask to P_4 after T_6 's termination, because the two existing subtasks on P_4 , T_2 and T_3 are already dedicated to P_3 and P_5 , respectively. As an outcome of the online controllability algorithm, subtask $T_{4,2}$ is migrated from processor P_1 to P_4 (as shown in Figure 9(b)), immediately after the task terminations. From the results shown in Figure 11, we can see that the previously uncontrollable system indeed becomes controllable again. The utilizations of all processors converge to the desired set points. Undesired processor overload and underutilization have been avoided.

3) *Feasibility*: As we analyzed before, controllability maintenance alone is not enough because it may still be infeasible

¹The schedulable utilization bound of RMS [32] may be used as the utilization set point for better utilization.

TABLE II
TASK RATES OF ALL TASKS (S MEANS RATE SATURATED)

	T_1	T_2	T_3	T_4	T_5
Naive	20 (S)	30.6569	5 (S)	29.9830	5.6836
Feasibility	43.1741	20.6556	19.3107	11.6857	5.0194
	T_6	T_7	T_8	T_9	T_{10}
Naive	20 (S)	50.4092	10 (S)	10 (S)	10 (S)
Feasibility	5.0004	50.5294	10.0018	11.1398	10.0008

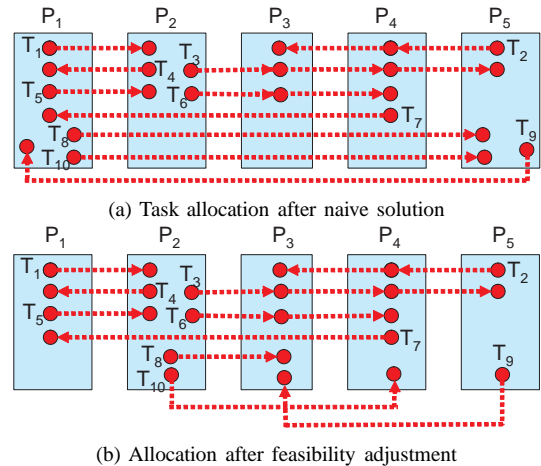


Fig. 12. Workload variations in feasibility experiments

for a controllable system to achieve the desired utilization set points when tasks arrive at runtime. In this set of experiments, we first show that some naive allocations of dynamically arriving tasks make it infeasible for the original FC-ORB to achieve the set points. Same as the previous experiments, the utilizations of all processors in the system initially start from their set points. At time 300×5 seconds, three end-to-end tasks (T_8 , T_9 and T_{10}) are admitted to the system. As an example of possible naive allocations, three subtasks are allocated to P_1 while the other three are allocated to P_5 (as shown in Figure 12(a)). Figure 13 shows that the system becomes infeasible after this allocation. P_1 and P_5 become overloaded while P_2 to P_4 are underutilized. Figure 14 and Table II show the rates of several tasks saturate after the task arrivals. The rates of tasks T_1 , T_3 and T_8 to T_{10} reach their lower boundaries and so cannot be decreased anymore. On the other hand, the rate of task T_6 reaches the upper boundary so cannot be increased any further. As a result of the saturations, no processor can achieve their set points because it is infeasible to do so.

We then run the same experiment on our extended middleware system with the feasibility handler enabled. Whenever there are new tasks admitted to the system, the feasibility handler conducts incremental Max-Min algorithm presented in Section VI to allocate the subtasks. We can see that the new tasks first have a smaller impact on the utilizations of the processors in the system, compared to the naive solution.

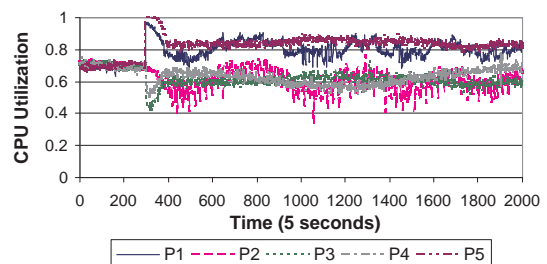


Fig. 13. System becomes infeasible after task arrivals

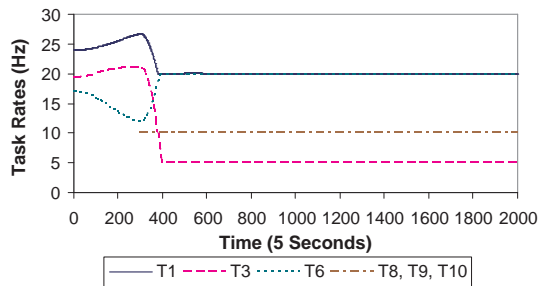


Fig. 14. Task rates saturate at boundaries when system is infeasible

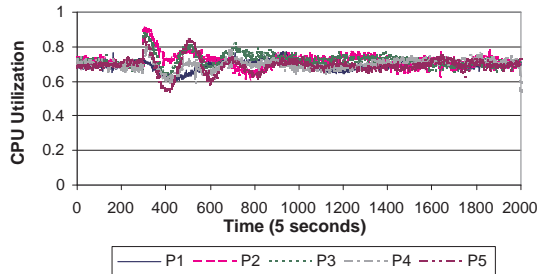


Fig. 15. System remains feasible after feasibility adjustment

That is because the feasibility handler distributes the impact to different processors, as shown in Figure 12(b). As demonstrated by Figure 15, even though the same task rate constrains exist, the system still can achieve the desired utilization set points thanks to the feasibility adjustment. Table II shows that none of the tasks saturate at their rate boundaries. Hence, with feasibility adjustment, it becomes feasible for a previously infeasible system to achieve the desired set points.

IX. CONCLUSION

In this paper, we have shown that both controllability and feasibility are fundamental properties of DRE systems, and so are crucial to the success of feedback control in such systems. Using end-to-end utilization control as an example, we found that uncontrollable or infeasible DRE systems often cause processor overload, deadline misses or undesired low task rates. We then proved that controllability and feasibility depend on end-to-end task allocations. We presented both offline and online task allocation algorithms to ensure system controllability and feasibility both at deployment time and at runtime even when the system is experiencing dynamic workload variations. As a result, a DRE system is guaranteed to meet the end-to-end deadlines of all tasks in the system while being able to run all tasks in the highest possible rates. Furthermore, we integrated our task allocation algorithms in the FC-ORB middleware. The efficacy of our algorithms has been demonstrated through both numerical results and empirical results on a physical test-bed.

REFERENCES

- [1] T. Abdelzaher, J. Stankovic, C. Lu, R. Zhang, and Y. Lu, "Feedback performance control in software services," *IEEE Control Systems*, vol. 23, no. 3, June 2003.
- [2] L. Abeni, L. Palopoli, G. Lipari, and J. Walpole, "Analysis of a reservation-based feedback scheduler," in *IEEE RTSS*, Dec. 2002.
- [3] A. Cervin, J. Eker, B. Bernhardsson, and K.-E. Arzen, "Feedback-feedforward scheduling of control tasks," *Real-Time Systems*, vol. 23, no. 1, pp. 25–53, July 2002.
- [4] C. Lu, X. Wang, and C. Gill, "Feedback control real-time scheduling in ORB middleware," in *IEEE RTAS*, May 2003.
- [5] M. Caccamo, G. Buttazzo, and L. Sha, "Elastic feedback control," in *ECRTS*, Stockholm, Sweden, June 2000.

- [6] D. C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole, "A feedback-driven proportion allocator for real-time scheduling," in *Operating Systems Design and Implementation*, 1999.
- [7] D. Henriksson and T. Olsson, "Maximizing the use of computational resources in multi-camera feedback control," in *IEEE RTAS*, May 2004.
- [8] C. Lu, X. Wang, and X. Koutsoukos, "Feedback utilization control in distributed real-time systems with end-to-end tasks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 16, no. 6, pp. 550–561, June 2005.
- [9] X. Wang, D. Jia, C. Lu, and X. Koutsoukos, "DEUCON: Decentralized end-to-end utilization control for distributed real-time systems," to appear in *IEEE Trans. Parallel Distrib. Syst.*, 2007.
- [10] J. W. S. Liu, *Real-Time Systems*. Prentice Hall, 2000.
- [11] C. Lu, J. A. Stankovic, G. Tao, and S. H. Son, "Feedback control real-time scheduling: Framework, modeling, and algorithms," *Journal of Real-Time Systems*, vol. 23, no. 1/2, pp. 85–126, July 2002.
- [12] X. Wang, C. Lu, and X. Koutsoukos, "Enhancing the robustness of distributed real-time middleware via end-to-end utilization control," in *IEEE RTSS*, 2005.
- [13] A. Goel, J. Walpole, and M. Shor, "Real-rate scheduling," in *RTAS*, 2004.
- [14] L. Abeni, T. Cucinotta, G. Lipari, L. Marzario, and L. Palopoli, "Adaptive reservations in a linux environment," in *RTAS*, May 2004.
- [15] V. Sharma, A. Thomas, T. Abdelzaher, K. Skadron, and Z. Lu, "Power-aware QoS management in web servers," in *IEEE RTSS*, 2003.
- [16] X. Wang, C. Lefurgy, and M. Ware, "Managing Peak System-level Power with Feedback Control," IBM Research, Tech. Rep. RC23835, 2005.
- [17] X. Koutsoukos, R. Tekumalla, B. Natarajan, and C. Lu, "Hybrid supervisory utilization control of real-time systems," in *IEEE RTAS*, 2005.
- [18] J. A. Stankovic, T. He, T. Abdelzaher, M. Marley, G. Tao, S. Son, and C. Lu, "Feedback control scheduling in distributed real-time systems," in *IEEE RTSS*, 2001.
- [19] S. Lin and G. Manimaran, "Double-loop feedback-based scheduling approach for distributed real-time systems," in *HiPC*, 2003.
- [20] Y. Diao, J. L. Hellerstein, A. J. Storm, M. Surendra, S. Lightstone, S. S. Parekh, and C. Garcia-Arellano, "Incorporating cost of control into the design of a load balancing controller," in *RTAS*, 2004.
- [21] C. Karamanolis, M. Karlsson, and X. Zhu, "Designing controllable computer systems," in *USENIX Workshop on Hot Topics in Operating Systems (HotOS)*, Santa Fe, NM, 2005.
- [22] S. Baruah, "Feasibility analysis of preemptive real-time systems upon heterogeneous multiprocessor platforms," *rtss*, vol. 00, pp. 37–46, 2004.
- [23] C.-J. Hou and K. G. Shin, "Allocation of periodic task modules with precedence and deadline constraints in distributed real-time systems," *IEEE Trans. Comput.*, vol. 46, no. 12, pp. 1338–1356, 1997.
- [24] S. Gertphol, Y. Yu, S. B. Gundala, V. K. Prasanna, S. Ali, J.-K. Kim, A. A. Maciejewski, and H. J. Siegel, "A metric and mixed-integer-programming-based approach for resource allocation in dynamic real-time systems," in *IPDPS*, Washington, DC, 2002.
- [25] S. Ali, J.-K. Kim, Y. Yu, S. B. Gundala, S. Gertphol, H. J. Siegel, and A. A. Maciejewski, "Utilization-based techniques for statically mapping heterogeneous applications onto the HiPer-D heterogeneous computing system," *Parallel and Distributed Computing Practices*, 2003.
- [26] J. Sun and J. W.-S. Liu, "Synchronization protocols in distributed real-time systems," in *ICDCS*, 1996.
- [27] D. Seto, J. P. Lehoczky, L. Sha, and K. G. Shin, "On task schedulability in real-time control system," in *RTSS*, Washington, D.C., Dec. 1996.
- [28] S. Brandt, G. Nutt, T. Berk, and J. Mankovich, "A dynamic quality of service middleware agent for mediating application resource usage," in *IEEE RTSS*, Dec. 1998.
- [29] B. Kao and H. Garcia-Molina, "Deadline assignment in a distributed soft real-time system," *IEEE Trans. Parallel Distrib. Syst.*, vol. 8, no. 12, pp. 1268–1274, 1997.
- [30] M. D. Natale and J. Stankovic, "Dynamic end-to-end guarantees in distributed real-time systems," in *IEEE RTSS*, 1994.
- [31] J. P. Lehoczky, "Fixed priority scheduling of periodic task sets with arbitrary deadline," in *IEEE RTSS*, 1990.
- [32] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of ACM*, Vol. 20, No.1, pp. 46–61, Jan. 1973.
- [33] G. F. Franklin, J. D. Powell, and M. Workman, *Digital Control of Dynamic Systems*, 3rd edition. Addison-Wesley, 1997.
- [34] R. W. Shields and J. B. Pearson, "Structural controllability of multiinput linear systems," *IEEE Transactions on Automatic Control*, vol. AC-21, pp. 203–212, 1976.