Hybrid Supervisory Utilization Control of Real-Time Systems

X. Koutsoukos R. Tekumalla B. Natarajan
Institute for Software Integrated Systems
Department of Electrical Engineering and Computer Science
Vanderbilt University
Nashville, TN 37235
{Xenofon.Koutsoukos,Radhika.r.tekumalla}@vanderbilt.edu,
bala@isis.vanderbilt.edu

C. Lu
Department of Computer Science
and Engineering
Washington University in St. Louis
St. Louis, MO 63130
lu@cse.wustl.edu

Abstract

Feedback control real-time scheduling (FCS) aims at satisfying performance specifications of real-time systems based on adaptive resource management. Existing FCS algorithms often rely on the existence of continuous control variables in real-time systems. A number of real-time systems, however, support only a finite set of discrete configurations that limit the adaptation mechanisms. This paper presents Hybrid Supervisory Utilization CONtrol (HySUCON) for scheduling such real-time systems. HySUCON enforces processor utilization bounds by managing the switchings between the discrete configurations. Our approach is based on a best-first-search algorithm that is invoked only if reconfiguration is necessary. Theoretical analysis and simulations demonstrate that the approach leads to robust utilization bounds for varying execution times. Experimental results demonstrate the algorithm performance for a representative application scenario.

1. Introduction

Traditional real-time scheduling theories [7][16] depend on accurate *a priori* knowledge of the system workload to provide real-time performance guarantees. Despite their success, they cannot easily be applied to many real-world systems since precise *a priori* knowledge of application behavior is not often available. For example, in large scale distributed real-time and embedded (DRE) systems, dynamic operating conditions due to partial failures and changes in configurations due to mode transitions can cause unexpected and acute changes to resource usage patterns. In many cases, if variations are undetected and unmanaged they can induce drastic changes in system performance [12].

Adaptive solutions to accommodate unexpected changes in the operating environment are usually based on hand-tuned and heuristic strategies which could induce instability. Such solutions tend to be stove-piped, requiring rework when ported to a new operating condition or platform. Further, sequences of such environmental changes over time can lead to chronic loss of confidence in system reliability and utility. Addressing these concerns requires new adaptive techniques to be developed on solid theoretical foundations that can provide *analytic* guarantees on system performance.

To overcome limitations of traditional scheduling theories and heuristic-based adaptive solutions, a number of *feedback control real-time scheduling (FCS)* algorithms (*e.g.*, [1][4][5][12][17]) have been developed recently. While traditional approaches usually adopt open-loop scheduling schemes, FCS algorithms employ software feedback control loops that dynamically adjust resource allocation in response to workload changes. Furthermore, FCS algorithms are modeled and designed using rigorous control-theoretic methodologies. As a result, these algorithms can provide robust and analytical performance guarantees despite uncertainties.

Although existing FCS algorithms have shown promise, several important practical issues have not been addressed. In particular, existing algorithms often assume that the system has *continuous* control variable(s) that can continuously be adjusted. While this assumption holds for certain classes of systems, there are many classes of real-time systems, such as avionics and total-ship computing environments that only support a *finite a priori* set of discrete configurations. The control variables in such systems are therefore intrinsically *discrete*. For instance, rate adaptation is a commonly used mechanism for controlling the CPU utilization and deadline miss ratio of a system [5][13][17]. While existing algorithms generally assume task rates can be set to any value within a range, the possible task rates in some applications may be *discrete* due to practical constraints. For instance in a sensor-to-weapon shooter system, changing the task rates imply changing the rate at which sensed imagery data are published. Setting the task rates to any value within a range is not practical since the hardware that senses the data may not have a high-resolution timer needed to precisely program the tasks. Some multimedia applications (e.g., Multi Bit Rate video) also only support a few predefined rates. Several other adaptation strategies may also provide only discrete control variables. Examples include task admission control, switching communication protocols, algorithms, or different levels of replication.

Traditional control theories such as linear control cannot effectively handle discrete control variables, especially when the number of possible values is small. To support adaptation strategies with discrete variables, hybrid (continuous/discrete) control algorithms must be used. This paper presents Hybrid Supervisory Utilization CONtrol (*HySUCON*) for enforcing utilization bounds in real-time systems by adaptively selecting the task rates from a finite discrete set. Although rate adaptation is used as a concrete example, our methodology is generally applicable to systems operating in a finite set of real-time and performance configurations.

The primary contributions of the paper are:

1. Design of a HySUCON, a hybrid supervisory control scheduling algorithm based on utilization control.
2. Theoretical analysis of HySUCON for providing theoretical performance guarantees.
3. Simulation results that demonstrate robust utilization performance for varying execution times.
4. Evaluation of the approach using experimental results for a representative application scenario.

The rest of the paper is organized as follows. Section 2 reviews related work. The problem is formulated in Section 3. Section 4 presents the development and analysis of HySUCON. Section 5 presents the simulations results and Section 6 the experimental results. Conclusions are discussed in Section 7.

2. Related Work

A survey of feedback performance control in computing systems is presented in [3]. Recent research on applications of control theory to real-time scheduling and utilization control is directly related to this paper. Steere et al., developed a feedback scheduler [17] that coordinated the share of CPU cycles allocated to threads. Abeni et al., presented control analysis of a reservation-based feedback scheduler [4]. Cervin et al. presented a feedback scheduler that adapts task rates for digital control systems [5]. The FCS/nORB system [13] provides a middleware service that dynamically controls CPU utilization by adjusting the invocation rates of remote operation invocations on a real-time Object Request Broker. All the above solutions continuously adjust control variables (*e.g.*, CPU shares or task rates).

Abdelwahed et al. introduced a general hybrid control approach for managing computing systems based on a finite set of control inputs [1]. This approach employs an exhaustive search algorithm to evaluate a performance measure for all possible operating states during a prediction horizon in order to select the best control input. While the framework can be applied for rate adaptation, the exhaustive search introduces significant overhead and is not suitable for real-time systems.

Several other FCS algorithms [2][12] used admission control and service level adjustment as adaptation mechanism for controlling the CPU utilization. Although both mechanisms are discrete in nature, the control algorithms were designed based on approximate fluid models with continuous variables. While such approximations may be adequate for systems with a large number of tasks, they can become inadequate for systems with a small number of tasks and service levels.

3. Problem Formulation

This paper considers a real-time system comprised of tasks $\{T_i \mid 1 \le i \le n\}$ executing on a single processor P. Each task T_i is invoked periodically at a rate $R_i(k)$ at sampling instant k. The rate $R_i(k)$ is assumed to take values in a finite set of discrete rates, $R_i(k) \in \{R_i^{(1)}, R_i^{(2)}, ..., R_i^{(m_i)}\}$ where $R_i^{(j)}$ is the j^{th} possible rate of task T_i. The sampling period of the controller, denoted by T_s, is selected to be larger than the maximum task period.

Each task T_i is characterized by the following attributes:

- C_i : Estimated execution time of T_i
- C_{BC_i} : Best case execution time of task T_i
- C_{WC_i} : Worst case execution time of task T_i
- $R_i(k)$: Current rate of task T_i
- $AE_i(k)$: Actual execution time of task T_i at k

The deadline for each task is assumed to be equal to its period. The actual execution times $AE_i(k)$ are bounded but not known a priori. It is assumed that $C_{BC_i} \le AE_i(k) \le C_{wc_i}$.

The processor P is characterized by the following attributes:

- $B(k) = \sum_i C_i R_i(k)$: Estimated CPU utilization at k
- $U(k)$: CPU utilization during sampling period k
- U_s : Desired utilization set-point
- G_a : Ratio of actual change in utilization to estimated change in utilization.

This paper presents a supervisory control algorithm that dynamically assigns the rates $R_i(k), i = 1,\dots n$ in order to ensure that the processor utilization $U(k)$ stays in a predetermined region $|U(k) - U_s| \le \varepsilon$ since for varying execution times and discrete task rates it is not be feasible to drive the utilization exactly to the set-point.

The algorithm employs a feedback control loop (see Section VI for more details) that dynamically adjusts task rates to enforce the control objective. The controller is located on a separate processor or shares a processor with some applications and must be scheduled as the highest-priority task in order to effectively control utilization under overload conditions. The processor has a *utilization monitor* and a *rate modulator*. A separate TCP connection connects the controller with the pair of utilization monitor and rate modulator on the processor. The user inputs to the controller include the utilization set-point U_s, the parameter ε, and the finite sets of rates for each task. The *control variable* is the processor utilization $U(k)$. The *control inputs* from the controller are the changes to task rates $\Delta\boldsymbol{R}(k) = [\Delta R_1(k),..., \Delta R_n(k)]^T$, where $\Delta R_i(k) = R_i(k) - R_i(k\text{-}1)$, $1 \le i \le n$. The supervisory controller employs the following mechanisms:

1. The utilization monitor on the processor that sends the utilization $U(k)$ in the last sampling period to the controller.
2. The controller computes the new task rates $\boldsymbol{R}(k)$ and sends them to the rate modulator on the processor P.
3. The rate modulator changes the task rates according to $\boldsymbol{R}(k)$.

The system is controlled by a hybrid supervisory controller which receives the utilization $U(k)$ that takes values in the interval [0,1] (continuous set) and sends the rates $\boldsymbol{R}(k)$ that take value in a discrete finite set. The controller changes the task rates only if needed in order to keep the utilization in the region $|U(k) - U_s| \le \varepsilon$.

Next, we establish a dynamic model that characterizes the relationship between the control input $\Delta\boldsymbol{R}(k)$ and the controlled variable $U(k)$. Let $\Delta R_i(k)$ denote the change to task rate, $\Delta R_i(k) = R_i(k) - R_i(k\text{-}1)$. The *estimated change to utilization*, $D_b(k)$, is given by

$$D_b(k) = \sum_{1 \le i \le n} C_i \Delta R_i(k) .$$

Note that $D_b(k)$ is based on the estimated execution time. Since the actual execution times may be different from their estimation, the utilization $U(k)$ can be written as

$$U(k) = U(k\text{-}1) + G_a D_b(k) \qquad (1)$$

where the *utilization gain* G_a represents the ratio between the change to the *actual* utilization and its estimation $D_b(k\text{-}1)$. For example, $G_a = 2$ means that the actual change to utilization is twice of the estimated change. The exact value of G_a is *unknown* due to the unpredictability of subtasks' execution times.

A task running at a higher rate contributes a higher value to the application at the cost of higher CPU utilization. However, large CPU utilization may cause tasks to miss their deadlines. To ensure that the tasks meet their deadlines, the utilization must be kept below its schedulable utilization bound [10]. This paper is concerned with the problem of ensuring that the processor utilization does not exceed the desired utilization set point. The control problem is to design a feedback control algorithm that selects the task rates $R_i(k)$ based on the utilization $U(k)$ so that the distance between the utilization and the set-point does not exceed the desired bound, i.e. $|U(k) - U_s| \le \varepsilon$.

4. Hybrid Supervisory Control

This section presents the design and analysis of the supervisory controller. Our approach is based on supervisory control of hybrid systems [8][9].

4.1. Control Formulation

Given the model of the real-time system, a supervisory controller can be designed to control the processor utilization. The continuous control approaches presented in [12][14] are not suitable because they cannot deal with the discrete nature of the task rates. To overcome this problem, we adopt the hybrid control approach from [9]. The approach is an extension of supervisory control of hybrid systems [8] for piecewise linear hybrid dynamical systems with disturbances and can be applied to the hybrid model of the real-time system that includes unknown execution times. The control specifications are formally described using finite state machines and include safety (e.g. the utilization should remain in region *S*) and eventual execution of actions (e.g. if the utilization is in a region *A*, it must be driven to *S* in a specified time interval).

The control approach is based on a piecewise linear partition of the state space that is used to formulate the specifications. For a single processor real-time system, we can partition the state space in three regions as shown in Figure 1. The partition can be described by a mapping (similar to an A/D converter) $\pi : [0,1] \rightarrow \{B, S, A\}$ that takes as argument the continuous utilization and returns the corresponding region.

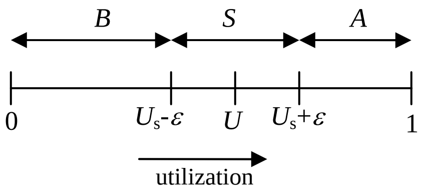

Figure 1. Partition of the state space

For all tasks to be schedulable, the controller should keep the utilization out of region *A*. This can be achieved by selecting the task rates for the worst-case execution times leading to a conservative design that will underutilize the system.

In this paper we take a different approach. We allow the utilization to enter region *A*, but upon detecting such an event the controller is invoked to drive the utilization back to *S* in a fixed time interval. The advantages of the approach are: (i) it increases the average utilization contributing a higher value to the application, (ii) the controller is invoked only when the utilization is out of the schedulable region reducing the control overhead, and (iii) it provides guarantees for driving the utilization to the schedulable region. Of course, when $U(k) \in A$ schedulability cannot be guaranteed which means that for a small fixed time interval some tasks may miss their deadlines. Hence HySUCON is more suitable for soft real-time systems that may tolerate a small percentage of transient deadline misses.

To ensure that the system is not underutilized, the controller is also invoked when the utilization drops in region *B*. In this case, the controller will increase the task rates to drive the utilization back to *S* in a fixed time interval.

To guarantee that the utilization will reach the desired region *S*, the controller must select a suitable set of task rates. This is accomplished by solving a discrete optimization problem online based on utilization feedback.

4.2. Discrete Optimization

If the processor utilization enters region *A* or *B*, then the controller is invoked in order to change the rates and drive the utilization back to the region *S*. The new rates are computed by solving a discrete optimization problem. We are using a Best First Search (BFS) algorithm. BFS is a greedy algorithm which performs a depth first search based on a heuristic. The heuristic is defined as the estimated distance to the goal, where the goal is defined as the desired change in utilization $\Delta U = |U(k) - U_s|$. The controller will compute the new task rates to minimize the heuristic. To reduce the control overhead, the search stops after finding a feasible solution that will drive any state from *A* to *S* that is not necessarily optimal.

First, we consider the case when $U(k) > U_s + \varepsilon$. Figure 2 illustrates the discrete optimization algorithm. The heuristic is initialized as $h(i, j) = \Delta U$. Suppose that at time *k*–1, the rate of the task T_i is $R_i(k-1)$ and h^* is the optimal value of the heuristic. If at time *k* we select $R_i^{(j)}$ for T_i, a lower bound for the estimated change in utilization will be $\alpha_{ij} = (R_i^{(j)} - R_i(k-1))C_{BC_i}$ where C_{BC_i} is the best case execution time of T_i. The heuristic is then updated as $h(i, j) = h^* + \alpha_{ij}$, $\alpha_{ij} < 0$. The algorithm initially computes the heuristic for each task rate $R_i^{(j)}$. The task rate with the lowest evaluation is selected since the heuristic measures the estimated distance to the goal. The algorithm terminates when $h^* \leq \varepsilon$ which means that the total expected change in utilization is close to the goal.

Similarly, if $U(k) < U_s + \varepsilon$, the controller will increase the utilization by increasing the rates of selective tasks. An upper bound of the estimated change in the utilization is defined as $\beta_{ij} = (R_i^{(j)} - R_i(k-1))C_{WC_i}$ where C_{WC_i} is the worst case execution time of T_i and the heuristic is then computed by $h(i,j) = h^* + \beta_{ij}$.

```
BFSα()
$\Delta U := |U(k) - U|; h^* := \Delta U; I := \varnothing;$
while $h^* > \varepsilon$ and $I \neq \{1,...n\}$
    for all $i \notin I$
        for $j = 1,\dots,m_i$
            $\alpha_{i,j} = (R_i^{(j)} - R_i(k-1))C_{BC_i}$
            if $\alpha_{ij} < 0$
                $h(i,j) := h^* + (R_i^{(j)} - R_i(k-1))C_{BC_i}$
            end if
        end for
    end for
    $h^* = \min_{i,j}\{|h(i,j)|\};$
    $(i^*, j^*) = \arg\min_{i,j}\{|h(i,j)|\};$
    $R_{i^*}(k) := R_{i^*}^{j^*}; I = I \cup \{i^*\};$
end while
```

Figure 2. The discrete optimization algorithm

The behavior of the controller is illustrated by the finite state machine of Figure 3. When $U(k)$ exits the region S to A, an event generated by the system (plant event σ_{SA}) triggers the controller that selects the new rates by solving the BFS algorithm based on the lower bounds for the estimated utilization α_{ij}. The new rates will drive the utilization back to S. Similarly, when the utilization falls to B, the controller uses the algorithm based on the lower bounds β_{ij}.

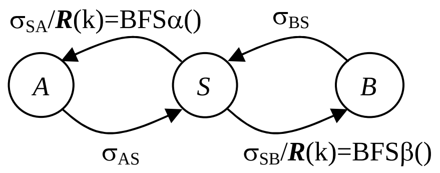

Figure 3. Finite state machine of the controller

4.3. Analysis

Feedback algorithms for adaptive utilization control are designed so that the closed loop system is stable to guarantee that the utilization will converge to the desired set-point independent of the unknown task execution times. Previous work [12][14] is based on the assumption that the set-point is an equilibrium of the system. When only a finite set of rates are available for each task, the single equilibrium assumption does not hold since for different rates the system will have different or no equilibria. However, it is still possible to design controllers for switching between the available rates so that the system stays close to the set-point. Such behaviors are similar to those of conventional stable systems close to equilibrium points and can been described using the notion of *safety* [9].

Definition 1 Given the system (1), the region $|U(k) - U_s| \leq \varepsilon$ is said to be *safe* if for every $U(0)=U_0$, there exists $K=K(U_0)\geq 0$ such that $|U(k) - U_s| \leq \varepsilon$ for any $k \geq K$.

This definition implies that the utilization trajectory will always remain in an ε-region of the set-point. The main challenge for real-time systems in using the notion of safety for control design is that the utilization gain G_a is not known. For such uncertain systems, one could design the controller for the worst case execution times. Such a design would result in underutilization of the processor to allow for a large safety margin.

To overcome these difficulties, our approach employs a notion of *practical stability*, which can be viewed as a relaxed definition of safety.

Definition 2 Let $|U(k) - U_s| > \varepsilon$ at time step k for a fixed parameter ε. If there exists finite $l \geq 0$ such that $|U(k+l) - U_s| \leq \varepsilon$ the system (1) is said to be *practically stable.*

The parameter l corresponds to the "rising-time" and as in conventional control design it is desirable to minimize l while keeping a small overshoot.

We define the lower utilization and upper utilization levels that are guaranteed to be achieved independent of the actual execution times as

$$\underline{U} = \min_{R_i, 1 \leq i \leq n}\left(\sum_{i=1}^{n} C_{WC_i} R_i\right) \text{ and } \overline{U} = \max_{R_i, 1 \leq i \leq n}\left(\sum_{i=1}^{n} C_{BC_i} R_i\right).$$

Theorem 1 There exist control policy for selecting the task rates for the system (1) so that the closed loop system is practically stable if $\underline{U} < U_s + \varepsilon$ and $\overline{U} > U_s - \varepsilon$.

Proof The condition $\underline{U} < U_s + \varepsilon$ $(\overline{U} > U_s - \varepsilon)$ guarantees that for every utilization $U(k_0) \in A$ (or B) there exist task rates R_i and integer k, $0 < k - k_0 \leq l$ so that $U(k) < U_s + \varepsilon$ $(U(k) > U_s - \varepsilon)$. Therefore, for any value $U(k)$ there always exists a combination of rates $\boldsymbol{R}(k) = [R_1(k),...,R_n(k)]^T$ that will drive the utilization towards in the ε-region of the set-point U_s .

Practically, the parameter ε must be selected large enough to avoid oscillations of the utilization between the regions *A* and *B*. The oscillatory behavior is also related to the parameter *l* that specifies how fast the utilization should return the region of the set-point. The presented BFS algorithm will achieve that in one sampling period (i.e. $l = 1$). As in conventional control this could lead to overshoot and oscillatory behavior. An alternative approach could be to compute the change in utilization based on an estimated value of the utilization gain G_a instead the best case and worst case execution time. This will decrease the oscillation but will increase the time *l* the utilization will reach the schedulable region. Due to length limitation, this paper focuses only in the case when $l = 1$.

Although, the safety conditions described above guarantee the existence of an appropriate control policy, we have to analyze the proposed algorithm and develop guidelines for the selection of the parameters. In the following, we assume that $U(k) > U_s + \varepsilon$ and the control task is to reduce the utilization. The case $U(k) < U_s - \varepsilon$ is symmetrical and is omitted.

First, we analyze the BFS algorithm vs. an exhaustive search (ES) algorithm that at every sampling period computes the expected utilization for all possible combinations of task rates and selects the one with the closer distance to the set-point. The conditions of Theorem 1 imply that the ES algorithm can always find an optimal combination of task rates. Let $|\Delta U^*_{ES}|$ denote the optimal estimated change in utilization derived using the ES algorithm, then we have $U(k) - |\Delta U^*_{ES}| < U_s + \varepsilon$ (and further $|U(k) - |\Delta U^*_{ES}| - U_s|$ is minimized).

Theorem 2 Assume that Theorem 1 holds and let $|\Delta U^*_{BFS}|$ denote the expected change in utilization derived using the BFS algorithm. The BFS algorithm will also reduce the utilization so that $U(k) - |\Delta U^*_{BFS}| < U_s + \varepsilon$.

Proof Consider the following two possible cases: (i) ES decreases the rates for all tasks, i.e. $\Delta R_i < 0, i = 1, \ldots, n$. In this case, because of its greediness property, BFS can select the new rates so that $|\Delta U^*_{BFS}| \geq |\Delta U^*_{ES}|$. In practice, our algorithm will stop the search if $U(k) - |\Delta U_{BFS}| < U_s + \varepsilon$ to reduce the overhead. (ii) ES decreases the rates of some tasks while increasers some others. In this case, since BFS can only decrease the rates, we have also $|\Delta U^*_{BFS}| \geq |\Delta U^*_{ES}|$. Therefore, in both cases, $U(k) - |\Delta U^*_{BFS}| < U_s + \varepsilon$.

The case $U(k) - |\Delta U^*_{BFS}| > U_s - \varepsilon$ can be proved in a similar manner.

The potential drawback of the BFS vs. the ES algorithm is related to the oscillation of the utilization. The ES and BFS algorithms differ on the minimum change in utilization they can achieve. Specifically, we have $\min_{R_i,\, 1 \leq i \leq n} |\Delta U^*_{BFS}| \geq \min_{R_i,\, 1 \leq i \leq n} |\Delta U^*_{ES}|$ since the BFS algorithm will either only increase or only decrease the rates. Hence, the BFS algorithm may overshoot the set-point especially since the change in utilization is based on the best case execution times.

An exhaustive search algorithm is not suitable for real-time systems since it is exponential, i.e. $O(m^n)$ where *n* is the number of tasks and *m* is the maximum number of task rates for a task. Our control algorithm is based on a best-first search using a heuristic. From Figure 2, the while loop is executed at most *n* times. Since each time we have to sort the values of the heuristic, the total time complexity is $O(n^3m^2)$ (depending on the sorting algorithm). Practically, the algorithm is much faster since it usually stops before searching all tasks and only negative (positive) values of the expected change $\alpha_{ij}\,(\beta_{ij})$ in utilization are used. The controller overhead is also evaluated using the simulation and experimental results.

5. Simulation Results

5.1. Simulator Environment

We have designed an event-driven simulator using Model Integrated Computing and the Generic Modeling Environment (GME) tool [6]. The simulator implements the real-time system controlled by HySUCON, the utilization monitor, and the rate modulator. The tasks are scheduled using the Rate Monotonic Scheduling (RMS) algorithm [11]. The environment provides a visual notation for specifying the tasks. Simulation code is generated automatically from the task specification. The advantage of the simulation framework is the automatic code generation of the controller. This is accomplished by an interpreter engine that prior to simulation, generates the controller code.

5.2. Workloads

We use two different workload/system configurations in our experiments. The workload parameters are characterized by experimentation and are used by the event-driven simulations in order to compare the performance with the experimental results. SIMPLE is a workload consisting of three tasks as shown in Table 1. All three tasks are floating point matrix multiplications with each task operating the same function but on matrices of different dimensions. Task A performs matrix multiplication on two matrices of dimensions 100x200 and 200x100, task B on matrices of dimensions 100x300 and 300x100, and task C on matrices of dimensions 100x400 and 400x100. The tasks were executed on a 2.8 GHz, Intel Pentium IV processor running Red Hat Linux. Execution time statistics were obtained using a high resolution timer and are shown in Table 1. Each task executed separately and the data was collected over 5000 samples. The wide range observed in the execution times can be attributed to the cache misses and pipelined architectures of modern day processors.

In the simulation environment, tasks are defined using the parameters in Table 1. The execution time for each task at every sampling period is drawn from a uniform distribution between its best- and worst-case execution times in our profiling experiments. The sampling period was 600 *ms* in both the simulation and the experimental setup.

The second configuration, MEDIUM that was used only for the experimental results, simulates a more complex workload consisting of 10 tasks. The tasks are also matrix multiplications of matrices different dimensions. Their parameters were obtained as in SIMPLE.

Table 1. Task parameters in SIMPLE

	Task A	Task B	Task C
C_{BC_i} (ms)	21	31	43
C_{WC_i} (ms)	68	86	81
R_i^j (KHz)	1/75 1/100 1/200 1/300 1/400 1/500	1/75 1/100 1/200 1/300 1/400 1/500	1/75 1/100 1/200 1/300 1/400 1/500
$R_i(0)$ (KHz)	200	200	200

5.3. Baselines

We compare HySUCON against two baseline algorithms, OPEN and FC-U. OPEN is an open-loop algorithm that uses fixed task rates. It assigns task rates a priori based on estimated execution times so that the processor utilization is near the set-point. As it can been seen from the task parameters in SIMPLE, it is not possible to establish tight bounds on the task execution times. OPEN will cause underutilization when execution times are overestimated and over-utilization when they are underestimated. As a baseline OPEN allows us to evaluate the benefits of a supervisory controller.

FC-U uses a single input single output PI controller presented in [12]. FC-U also computes the changes to task rates based on measured utilization but assumes that the task rates take values on a continuous set (i.e. $R_{i.\min} \le R_i \le R_{i.\max}$). FC-U can be used in our case by quantizing the continuous task rates computed by the controller to the closest available value. As a baseline FCS allows to compare the supervisory controller with simple linear control.

5.4. Real-Time Simulation for SIMPLE

This section provides the simulation results for the SIMPLE task scenario obtained using the event-driven real time simulator and compares HySUCON's performance with OPEN and FC-U controller. CPU utilization set point U_s was set to 0.69 and ε to 0.10 Simulations run for 60 *sec*, with sampling being performed at every 600 *ms* (T_s).

Figure 4 shows the CPU utilization $U(k)$. $U(k)$ changes as the execution time of the periodic tasks varies randomly. For example, at time $18T_s$, the processor is underutilized at 0.57. HySUCON responds to the deviation from the utilization set point by increasing task A's rate from 1/400 to 1/300. As a result the utilization increases from 0.57 to 0.74 in the next time step.

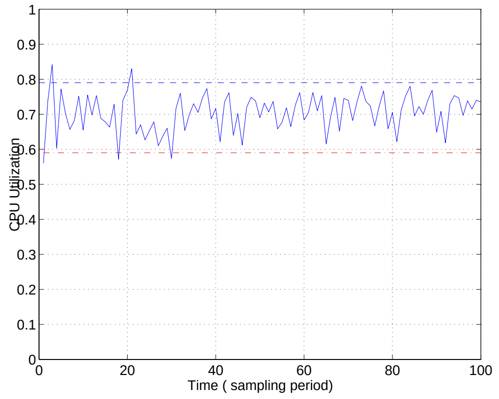

Figure 4. CPU Utilization for SIMPLE

Similarly, in the case of over-utilization the controller will decrease some of the rates. For example, at $21T_s$, where the utilization is 0.83, HySUCON reduces rate of task B from 1/200 to 1/300, which results in a subsequent reduction in utilization to 0.64 in the next time step. The average utilization 0.704 is with standard deviation 0.054. Figure 5 shows the task rates selected by the controller.

In contrast, Figure 6 shows an overloaded processor under OPEN unable to adapt to the workload conditions. As a result, CPU utilization levels at 100%.

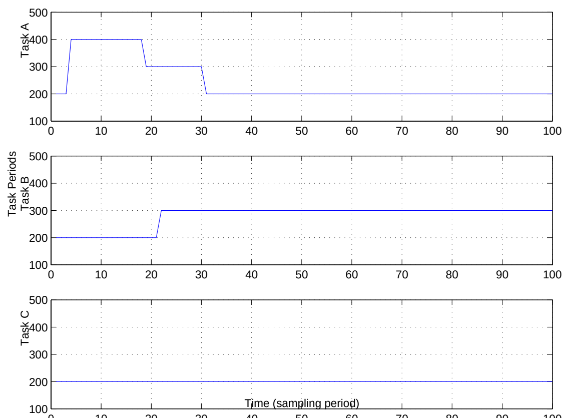

Figure 5. Task Periods for SIMPLE

Figure 7 and Figure 8 illustrate FC-U's performance for the SIMPLE workload. While the FC-U controller responds to both over and under utilization by selecting different rates, its performance is limited by the discrete rate set provided by the user. In the case of discrete rates, the utilization is not guaranteed to converge to the set point and FC-U performs much worse than HySUCON. A larger and finer rate set would reduce the quantization error that occurs when FC-U quantizes its chosen rate to the user defined rate and a lower quantization error would lead to better performance.

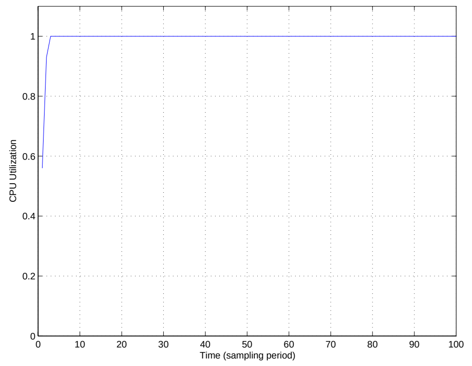

Figure 6. OPEN CPU utilization

6. Experimentation

6.1. Experimental Setup

In our experimental setup, we evaluated a real-time application whose task rates can be changed discretely using callbacks from a controller. The distributed system setup consists of 2 Intel Pentium IV, 2.8 GHz processors running Red Hat Linux. The client processor performs CPU intensive computations and sends utilization updates every sampling period to the remote server. HySUCON residing on the server receives these updates from the client and computes the new task rates. It communicates the new rates to the rate modulator, which subsequently changes the rates.

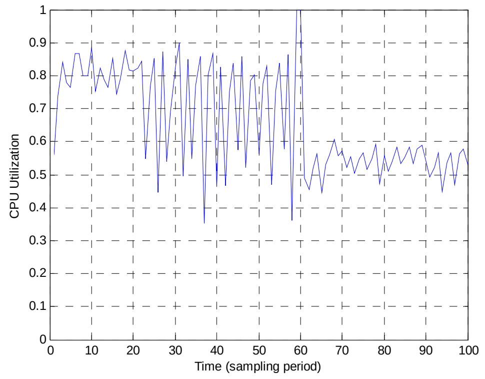

Figure 7. FC-U CPU Utilization

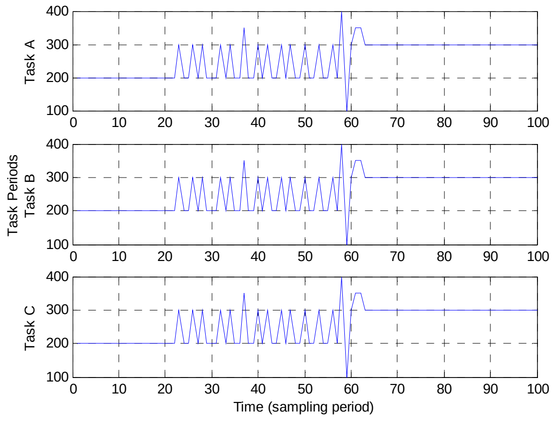

Figure 8. FC-U Task Periods

We assumed that the application is implemented as a Half Sync/Half Async [18] with a queue receiving events at different task rates. The setup has the following components: (i) A Half Sync/Half Async setup as a real-time application, (ii) a component for sensing the resource utilization and sending it to a controller, and (a) a component for accessing new task rates from the controller.

Real-time application: Our real-time application is deadline driven by messages and the processing of each message has a deadline. The messages are inserted into a queue at periodic intervals of time dictated by the periods. The messages are then processed by any thread (every thread has a equally likely chance of processing the message) waiting on the queue and performs the intended job (matrix multiplication in our setup).

Sensing resource-utilization: After every message is processed, the thread which processes a request senses the CPU utilization on the machine and sends to a controller which runs on a different machine. We used a distributed middleware (ACE/TAO) to implement this, since it provides all the required mechanisms.

Accessing New Task Rates: Our experimental setup also has a provision to contact the controller to receive new task rates at periodic time intervals. The new task rates are computed by the controller based on the latest utilizations received from the application by the controller.

6.2. Experimental Results for SIMPLE

This section discusses the experimental setup results obtained from HySUCON for the SIMPLE workload. CPU utilization set point U_s was set to 0.69 and ε to 0.10.

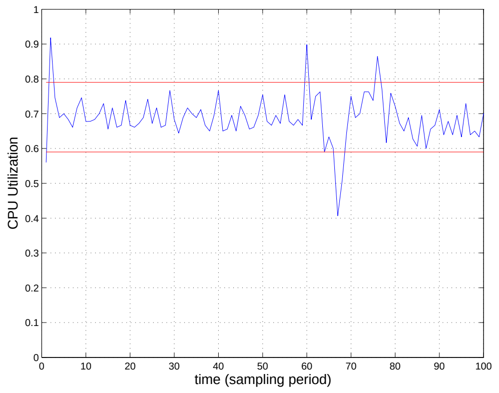

Figure 9. CPU Utilization for SIMPLE

Task execution times vary dynamically at run time under the SIMPLE configuration. As a result, the client CPU utilization varies dynamically, as shown in Figure 9. At time $60T_s$, the processor faces over-utilization at 0.90, as a result of which, the controller decreases the rate of task A from 1/200 to 1/75. This, results in a decrease in the utilization to 0.68 in the next time instant. Similarly the controller reacts to under-utilization by increasing the rate of A from 1/75 to 1/200 at $68T_s$, where the CPU Utilization is 0.9. The average utilization is 0.673 and the standard deviation is 0.053. The task rates are given in Figure 10.

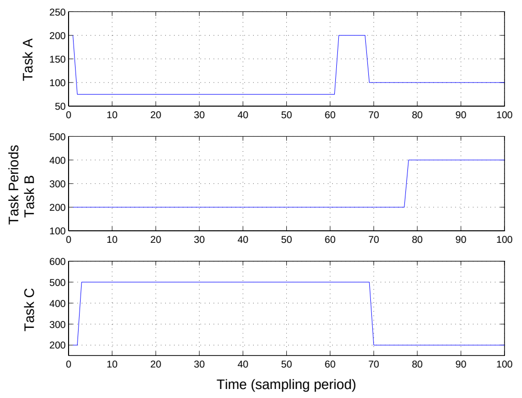

Figure 10. Task Periods for SIMPLE

6.3. Experimental Results for MEDIUM

The MEDIUM task scenario consisting of 10 tasks demonstrate the performance of HySUCON for larger workloads. The same client-server setup was used as described in the previous section. Input parameters for the controller algorithm, such as best-case and worst-case execution times for the 10 tasks were obtained from experiments conducted by running the same 10 tasks one at a time. As shown in Figure 11, the CPU utilization varies due to the dynamic task execution times. HySUCON controls the utilization in a region of the set-point. For instance, at time $47T_s$, the client reports under-utilization at 0.8277 and the controller responds by decreasing the task rates which results in an decrease in utilization to 0.76. Similarly the controller reacts to over-utilization, as seen at time $160T_s$ where the utilization drops to 0.58 by increasing the rates and hence the utilization to 0.65. The average utilization is 0.997 and the standard deviation is 0.054.

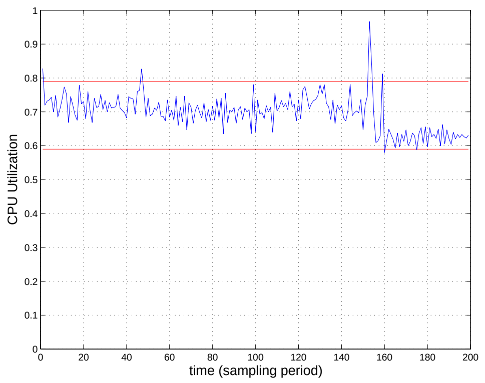

Figure 11. HySUCON CPU Utilization

6.4. Control Overhead

To estimate the run-time overhead of the controller, we measured the execution time of the controller component running on the server. In the simulations with the SIMPLE workload on a 2.8 GHz, Intel Pentium IV processor, the average controller overhead was 0.033 *msec.* In the HySUCON experimental setup, the controller overhead was calculated as the interval from the time the server receives the utilization to the time the controller compute the new task rates. For the SIMPLE workload, the average overhead was 0.069 *ms* and for the MEDIUM scenario the average overhead was 0.155 *ms*.

7. Conclusions

We have presented a hybrid supervisory control algorithm for feedback control real-time scheduling of single-processor real-time systems. Our theoretical analysis and are experimental results demonstrate the algorithm provides robust utilization bounds in the presence of varying execution times. Future work includes adapting the method for task admission control by including rates with zero values as well as the extension of the approach to multi-processor systems with end-to-end tasks.

Acknowledgements
This work is supported in part by DARPA under the Adaptive and Reflective Middleware Systems (ARMS) program (grant NBCHC030140).

8. References

[1] S. Abdelwahed, N. Kandasamy, and S. Neema, "Online Control for Self-Management in Computing Systems," IEEE Real-Time and Embedded Technology and Applications Symposium (*RTAS'04*), May 2004.

[2] T.F. Abdelzaher, K.G. Shin, N. Bhatti, "Performance Guarantees for Web Server End-Systems: A Control-Theoretical Approach," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 13, No. 1, Jan 2002.

[3] T.F. Abdelzaher, J.A. Stankovic, C. Lu, R. Zhang, and Y. Lu, "Feedback Performance Control in Software Services," *IEEE Control Systems*, 23(3): 74-90, June 2003.

[4] L. Abeni, L. Palopoli, G. Lipari, and J. Walpole, "Analysis of a Reservation-based Feedback Scheduler," IEEE Real-Time Systems Symposium (*RTSS 2002*), Dec 2002.

[5] A. Cervin, J. Eker, B. Bernhardsson, and K.-E. Årzén, "Feedback-Feedforward Scheduling of LQG-Control Tasks," *Real-time Systems Journal*, 23, 2002.

[6] G. Karsai, J. Sztipanovits, A. Ledeczi, T. Bapty. Model-Integrated Development of Embedded Software. *Proceedings of the IEEE*, 91(1):145-164, 2003.

[7] M.H. Klein, T. Ralya, B. Pollak, and R. Obenza, *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems,* Kluwer Academic Publishers, 1993.

[8] X.D. Koutsoukos, P.J. Antsaklis, J.A. Stiver and M.D. Lemmon, "Supervisory Control of Hybrid Systems," *Proceedings of IEEE*, 88(7), 1026-1049, 2000.

[9] X. Koutsoukos and P. Antsaklis, "Safety and Reachability of Piecewise Linear Hybrid Dynamical Systems Based on Discrete Abstractions", *Journal of Discrete Event Dynamic Systems: Theory and Applications.* 13(3), 203-243, 2003.

[10] C. Liu and J.Layland, "Scheduling Algorithms for Multi-programming in a Hard-Real-Time Environment," *Journal of the ACM*, 20(1), 46-61, 1973.

[11] J.W.S. Liu, *Real-Time Systems*. Prentice Hall, 2000.

[12] C. Lu, J.A. Stankovic, G. Tao, and S.H. Son, "Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms," *Real-Time Systems Journal*, Special Issue on Control-theoretical Approaches to Real-Time Computing, 23(1/2): 85-126, July/September 2002.

[13] C. Lu, X. Wang, and C.D. Gill, "Feedback Control Real-Time Scheduling in ORB Middleware," IEEE Real-Time and Embedded Technology and Applications Symposium (*RTAS 2003*), Washington DC, May 2003.

[14] C. Lu, X. Wang, and X. Koutsoukos, "End-to-end Utilization Control in Distributed Real-Time Systems," International Conference on Distributed Computing Systems, *ICDCS 2004*, Tokyo, Japan, Mar. 2004.

[15] D. Seto, J.P. Lehoczky, L. Sha, and K.G. Shin, "On Task Schedulability in Real-Time Control Systems", IEEE Real-Time Systems Symposium, December 1996.

[16] J.A. Stankovic, M. Spuri, K. Ramamritham, and G.C. Buttazzo, *Deadline Scheduling for Real-Time Systems – EDF and Related Algorithms*, Kluwer Academic Publishers, 1998.

[17] D.C. Steere, A. Goel, J. Gruenberg, D. McNamee, C. Pu, and J. Walpole, "A Feedback-driven Proportion Allocator for Real-Rate Scheduling," *OSDI*, Feb 1999.

[18] D.C. Schmidt and C. D. Cranor. "Half-Sync/Half-Async Pattern for Efficient and Well-structured Concurrent I/O", Pattern Languages of Program Design, Addison-Wisley, 1996