Real-Time Systems, 28, 5–37, 2004

Online QoS Optimization Using Service Classes in Surveillance Radar Systems

CHANG-GUN LEE cglee@ece.osu.edu
Department of Electrical and Computer Engineering, Ohio State University, Columbus, OH 43210, USA

CHI-SHENG SHIH cshih@csie.ntu.edu.tw
Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan

LUI SHA lrs@uiuc.edu
Department of Computer Science, University of Illinois, Urbana, IL 61801, USA

Abstract. Many application level qualities are functions of available computation resources. Recent studies have handled the computation resource allocation problem to maximize the overall application quality. However, such QoS problems are fundamentally multi-dimensional optimization problems that require extensive computation. Therefore, online usage of optimization procedures may significantly reduce the computation resource available for applications. This raises the question of how to best use the optimization procedures for dynamic real-time task sets. In dynamic real-time systems, it is important to improve the performance by re-allocating the resources adapting to dynamic situations. However, the overhead of changing task parameters (i.e., algorithms and frequencies) for resource re-allocation is non-negligible in many applications. Thus, too frequent change of resource allocation may not be desirable. This paper proposes a method called service classes configuration to address the QoS problem with dynamic arrival and departure of tasks. The method avoids online usage of optimization procedures by offline designing templates (called service classes) of resource allocation, which will be adaptively used depending on online situations. The service classes are designed by best trading-off the accuracy of dynamic adaptation against the overhead of resource re-allocation. A simplified radar application is used as an illustrative example.

Keywords: real-time system, optimal resource allocation, quality of service, online QoS management, service class, surveillance radar system

1. Introduction

QoS research recognizes that application level qualities are functions of available computation resources. For example, feedback control systems can provide better control performance with higher rates of sampling and control actuation. Recent studies (Seto et al., 1996, 1998) allocate CPU resources in a way that maximizes the control performance. QoS-based resource allocation model (Q-RAM) is a generalized approach that handles several quality attributes and several different forms of utility functions (Rajkumar et al., 1997, 1998; Lee et al., 1998). The QoS problem is fundamentally a multi-dimensional optimization problem that requires extensive computation.

For a system with a static task set, we can apply the optimization procedure offline to find the optimal resource sharing and it can be constantly used online. However, more challenging QoS applications have a dynamically changing task set. For example, a surveillance radar system must simultaneously track multiple targets. Tracking quality of each target depends on the amount of computing resource allocated to its track task. A higher sampling rate and more sophisticated algorithms, which require more computing resource, will produce better tracking quality. However, the largest possible amount of resource for each task is bounded due to finite computing resource shared among multiple track tasks. Thus, it is important to allocate computing resources in such a way that the weighted sum of tracking qualities is maximized. The challenge is that targets come and go. Different numbers of targets and target types create different QoS optimization problem online. And the QoS optimization is a computationally expensive problem that may compete with tracking tasks for limited computing resources if it is applied online.

As a result, existing real-time scheduling methods typically allocate resources based on a worst case scenario. Consider an example system with the requirement that it must simultaneously track five air planes and seven missiles in the worst case. However, the actual workload state of the system will change under the worst case as targets dynamically arrive and depart. The set of all possible workload states in this simplified example can be depicted as in Figure 1 where each cell represents a workload state and its row and column numbers represent air plane and missile counts at the state. As the workload state dynamically changes, the optimal resource sharing for the best tracking quality also changes. Thus, the resource allocation should be adaptive to online workload changes. The resource allocation based on the worst case scenario is inefficient since it cannot adapt to workload changes. On the other hand, recomputing the QoS problem online has too high an overhead.

This paper investigates how to best use the optimization procedures for dynamic real-time applications. We tackle this problem by a method called service classes configuration. A service class is a pre-calculated resource allocation. During the design phase, we design a set of service classes (CL_1, CL_2, and CL_3 in Figure 1) that are optimized for different workload states (shaded states in Figure 1). A light workload service class (e.g., CL_1) can provide a large amount of resource to each task, resulting in high quality of each. On the other hand, a heavy workload service class (e.g., CL_3) can

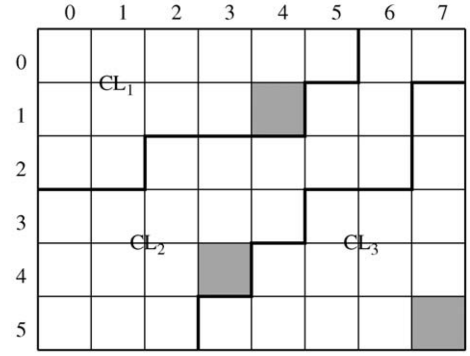


Figure 1. An example set of service classes.

only assign a significant amount of resource to certain important tasks while others must have less resource in order to handle more tasks. Each service class covers a certain region of workload states. A light workload service class can be used only when the system workload is low. A heavy workload service class can be used when workload is either high or low. However, it is preferable to use the light workload class for low workload. The set of workload states is partitioned into a number of regions each of which is covered by its corresponding service class as shown in Figure 1.

The more service classes we have, the closer the online workload state can be to the state for which the covering service class is optimized. For example, if we can have the same number of service classes as the total number of workload states, each service class can be optimized for each workload state. However, many service classes require frequent service class switching as the online workload changes. Switching service classes causes resource re-allocation to each task that requires change of its sampling frequency and/or algorithms. For a task to change its sampling frequency, it needs extra time to reconfigure several objects related to the sampling frequency. Changing algorithm also requires reconfiguration time such as cache replacement with the new algorithm codes and memory allocation used by the new algorithm. Thus, the set of service classes should be designed considering the trade-off between capability of online adaptation and reconfiguration overhead by service class switching.

Another important issue in the design of service classes is the scalability to the explosion of the workload state space since the number of workload states explodes when there are many different target types. Therefore, designing a set of service classes has the following challenges:

- How can we select service classes and how many service classes are needed?
- How can we best define the region of workload states covered by each service class?
- How can we minimize reconfiguration overhead's impact to schedulability during service class switching?
- How can we make the service class design method scalable to the explosion in number of workload states?

The proposed service classes configuration effectively handles these inter-dependent issues. In this paper, we propose reconfiguration scenarios and sufficient conditions for switching service classes guaranteeing all deadlines. Based on these, the service classes configuration method finds a maximal set of service classes without explicitly reserving resource for reconfigurations. Using the service classes, we can effectively adapt to online workload to achieve a high degree of QoS while always guaranteeing deadlines.

The rest of this paper is organized as follows: Section 2 uses a surveillance radar system to illustrate the service class problem. Section 3 explains the proposed approach to service classes configuration. Section 4 presents experimental results. Section 5 summarizes the related work. Finally, Section 6 presents our conclusions and future work.

2. Problem Description, Terminologies, and Assumptions

The goal of this work is to develop a method that maximizes the application level QoS using online switching of offline computed service classes. To introduce the application context, we assume a phased array radar system (Kuo et al., 2000) with an array of antennae that can steer the radar beam electronically.

In such systems, radar beam steering commands are issued by three kinds of tasks with different missions: search, confirmation and track tasks. The search task periodically scans the entire surveillance space to detect the appearance of new targets. Once a new target is detected, a confirmation task is created to identify the type of target. When it is identified, a track task is created and starts tracking the target until it leaves or is destroyed.

These search, confirmation and track tasks are scheduled on the processor resource issuing radar control commands. Their radar control commands are scheduled on the antenna resource. Figure 2 (Baugh, 1973) depicts the two resources. The scheduling of radar control commands on the antenna resource is non-preemptive in nature. On the other hand, the task scheduling on the processor can be implemented either in a preemptive way or in a non-preemptive way. Modern radar systems use a general purpose processor (Billetter, 1989, 1987) and thus we assume such processor scheduling tasks in a preemptive way. Specifically, we assume the preemptive earliest deadline first (EDF) (Liu and Layland, 1973) scheduling of tasks on the processor resource. Also, to focus on the service class abstraction rather than the details of radar operations, we assume that certain amounts of resource budgets are reserved for search tasks by using periodic polling servers and for confirmation tasks by using aperiodic servers (Strosnider et al., 1995; Ghazalie and Baker, 1995; Deng et al., 1997; Spuri and Buttazo, 1996). Therefore, in the rest of this paper, we will focus on the optimal usage of the remaining resource budget by track tasks to maximize the overall tracking quality.

Once a track task is created to track a target object, it periodically samples the target location and estimates the next location with a particular sampling frequency as shown in Figure 3. Table 1 (Kuo et al., 2002) lists the nominal period value ranges depending on the required precision level and target speed and distance. As long as the target speed and distance are in a certain range, the track task can be modeled as a periodic task with a constant sampling frequency. In Section 3.4, we will explain how to handle the sampling frequency change caused by a significant change of speed and distance.

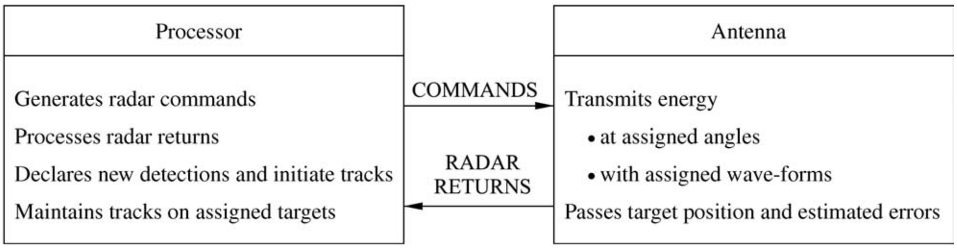

Processor
Generates radar commands
Processes radar returns
Declares new detections and initiate tracks
Maintains tracks on assigned targets
COMMANDS
RADAR RETURNS
Antenna
Transmits energy
• at assigned angles
• with assigned wave-forms
Passes target position and estimated errors

Figure 2. Two resources in a radar system.

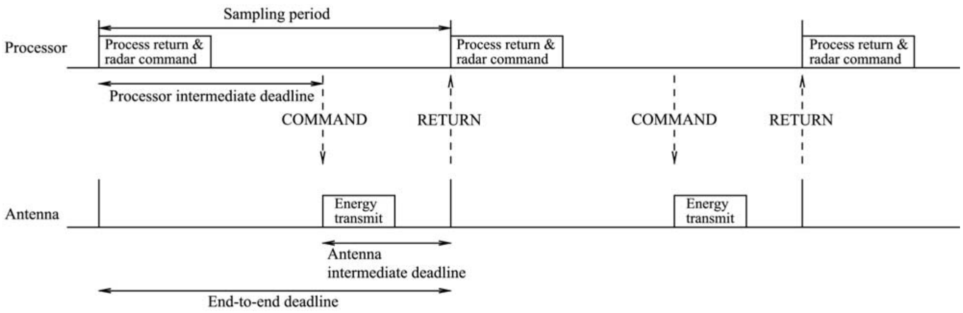

Sampling period
Processor
Process return & radar command
Process return & radar command
Process return & radar command
Processor intermediate deadline
COMMAND
RETURN
COMMAND
RETURN
Antenna
Energy transmit
Energy transmit
Antenna intermediate deadline
End-to-end deadline

Figure 3. Execution of a track task.

As shown in Figure 3, a track task can be modeled as an end-to-end task (Bettati, 1994; Sun, 1997) whose first subtask is executed on the processor and the second one is executed on the antenna. Each subtask should meet its intermediate deadline to meet the end-to-end deadline. To make the explanation simple, we will momentarily ignore the effect of the antenna scheduling considering only the processor resource. In Section 3.5, we will explain how our service class approach can be extended to the dual resource problem with processor and antenna resources.

Track tasks are differently classified by target types they are tracking. For example, in the surveillance radar application, targets can be classified into different types such as missile, hostile air plane, unknown, etc. The number of different target types is denoted by K. Due to the resource limitation, for each target type $k \in \{1, \ldots, K\}$, we assume that the maximum number of targets that the system must simultaneously track, denoted by N_k, is given as a system specification. Thus, the worst case workload is denoted by a K-tuple $(N_1, N_2, \ldots, N_K)$. Each possible system workload state or just state s is denoted by $(n_1^s, n_2^s, \ldots, n_K^s)$, $0 \leq n_1^s \leq N_1$, $0 \leq n_2^s \leq N_2, \ldots, 0 \leq n_K^s \leq N_K$. We call the set of all possible system states workload space denoted by WS. Figure 4 shows the workload space for a system where the number of target types, that is, K, is two.

A track task for each target type $k \in \{1, \ldots, K\}$ produces different tracking quality depending on the amount of allocated processor resource. For example, tracking a missile with higher sampling frequency and/or more sophisticated algorithms will require more processor resource but produce better tracking quality. Thus, the tracking quality of each target type $k \in \{1, \ldots, K\}$ is modeled as a function of its sampling frequency f_k and computation time C_k, that is, $V_k(f_k, C_k)$. For now, we assume that each track task for a type $k \in \{1, \ldots, K\}$ uses a fixed algorithm and thus requires a constant computation time

Table 1. Nominal period values of track tasks.

Tracking precision level	Period
High-precision track	(100 ms, 250 ms)
Precision track	(100 ms, 250 ms)
Normal track	(250 ms, 2000 ms)

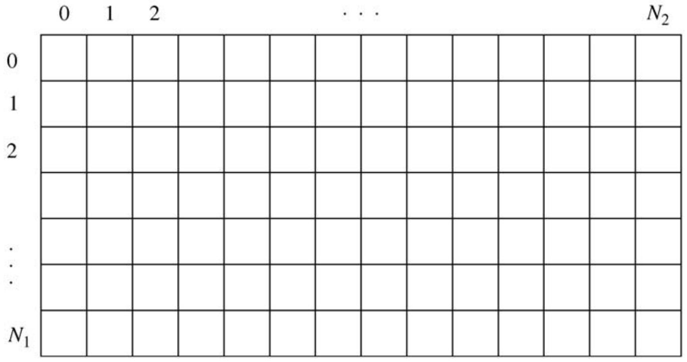


Figure 4. Workload space $\mathrm{WS} = \{s = (n_1^s, n_2^s) \mid 0 \le n_1^s \le N_1, 0 \le n_2^s \le N_2\}$.

C_k at each sampling period. Only its sampling frequency f_k is assumed to be adjustable. Later, we will show how our approach can be generalized to handle the situation where both C_k and f_k are adjustable by using the Q-RAM approach (Rajkumar et al., 1997, 1998; Lee et al., 1998). Additional quality attributes can be handled in a similar way.

Generally, the control performance depending on the sampling frequency is modeled as an exponential function (Seto et al., 1996; Caccamo et al., 2000). In a radar system, the system keeps a record (track) of each target and remembers its current position, heading, speed, etc. as the target moves. The records are updated periodically at sufficiently high frequencies so as to maintain a specified level of confidence in their accuracy. The higher is the update frequency to keep track of it, the more accurate the record is. The accuracy improvement by increasing the update frequency is significant at the beginning but becomes only marginal once the accuracy is saturated with a high enough frequency. Such property allows us to apply the exponential performance model proposed by Seto et al. (1996). We assume that the tracking performance, also called tracking quality, for target type k can be defined with the following form of exponential function:[1]

$$V_k(f_k) = w_k\left(1 - \mathrm{e}^{-\alpha_k f_k + \beta_k}\right) \tag{1}$$

where the sampling frequency f_k is a controllable variable that should be larger than or equal to $f_{k,\min}$ to avoid loss of tracking. The parameters w_k, α_k, and β_k specify the weight (importance of the target type), sensitivity to sampling frequency, and control value of minimum achievable performance, respectively. These parameters for each target type can be given by nonlinear regression techniques (Bates and Watts, 1988) after thoroughly investigating target characteristics such as speed, maneuverability, and lethality. Figure 5 shows examples of tracking quality functions for two different target types: one is for a target type more critical and more sensitive to the sampling frequency and the other less critical and less sensitive to the sampling frequency. Note that the minimal sampling frequencies for these two target types are not the same. The minimal sampling frequency of one target type is determined by the physical property of the target such as the maximal speed and maneuvering capability. Figure 5 shows that by properly choosing

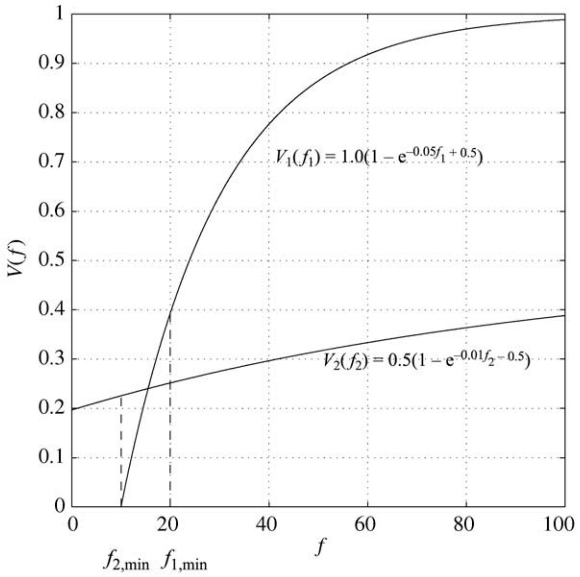


Figure 5. Examples of tracking quality function.

$f_{k,\min}$, w_k, α_k, and β_k, we can draw many different shapes of tracking quality functions that can properly model physical properties of different target types.

With these settings, the processor resource allocation problem can be formally described as follows:

Problem Description Which sampling frequency f_k^s should be assigned to each target type $k \in \{1, \ldots, K\}$ at each system state $s \in \mathrm{WS}$, to maximize the total sum of expected tracking quality $(\sum_{s \in \mathrm{WS}} p(s) \sum_{k \in \{1,\ldots,K\}} n_k^s V_k(f_k^s)$ where $p(s)$ is the probability that the system is at state $s)$ while guaranteeing all deadlines and minimal requirements $f_{k,\min}$?

We handle this problem with a set of service classes each of which is defined as a pre-calculated sampling frequency allocation. The set of service classes is denoted by $\{\mathrm{CL}_1, \mathrm{CL}_2, \ldots, \mathrm{CL}_M\}$ where $\mathrm{CL}_i (1 \leq i \leq M)$ is each service class in the set. A service class CL_i is optimized for a different workload state defined as a base state. The base state of CL_i is denoted by $\mathscr{S}^{\mathrm{CL}_i} = (n_1^{\mathrm{CL}_i}, \ldots, n_K^{\mathrm{CL}_i})$ where $n_k^{\mathrm{CL}_i} (1 \leq k \leq K)$ is the number of target instances of type k. The CL_i's frequency allocation optimized for $\mathscr{S}^{\mathrm{CL}_i}$ is denoted by $\mathscr{F}^{\mathrm{CL}_i} = (f_1^{\mathrm{CL}_i}, \ldots, f_K^{\mathrm{CL}_i})$ where $f_k^{\mathrm{CL}_i} (1 \leq k \leq K)$ is the sampling frequency for target type k. The region of the workload space that is covered by a service class CL_i is called CL_i's covering region.

As mentioned before, changing service classes, that is, changing sampling frequencies and/or algorithms is not free. For changing them, a task needs extra time for reconfiguration such as re-initialization of objects, cache replacement, and memory allocations. We call this extra time reconfiguration cost. For simplicity, the reconfiguration cost for each type of track task is assumed same and denoted by C_r. It can be easily relaxed by using different C_r values for different types.

3. Design of Service Classes

3.1. *Reconfiguration for Service Class Switching*

In this section, we derive sufficient conditions for service class switching without explicitly reserving resource for the reconfiguration cost of switching. This section assumes that service classes and their covering regions were already computed and stored in a table. Thus, whenever the workload state changes, by looking at the table, we can determine the current covering service class (denoted by GLOBAL_CLASS) and its associated frequency allocation. Our method to find a set of service classes and covering regions will be explained in Section 3.2.

Let us consider an example in Figure 6 where the arrival of a new target triggers service class switching from CL_1 to CL_2. In the example, the system is originally in a state with two targets that is served by CL_1. The track tasks of the two targets τ_1 and τ_2 have computation times $C_1 = 3$ and $C_2 = 4$. CL_1 has pre-calculated frequencies $f_1 = 1/6$ and $f_2 = 1/8$ for the two track tasks. Thus, the two track tasks are using all processor resource $(3/6 + 4/8 = 1)$.

When a new target is detected at time 11, the scheduler changes the system state and notices that the new state is covered by the new service class CL_2. The track task for the new target τ_3 has computation time $C_3 = 1$ and the new service class CL_2 has pre-calculated frequencies $f_1 = 1/9$, $f_2 = 1/10$, and $f_3 = 1/4$. Once it is noticed, the scheduler changes the global variable GLOBAL_CLASS from CL_1 to CL_2. We assume that the time for changing GLOBAL_CLASS is negligible. Whenever the scheduler is invoked at release time of a job, it uses GLOBAL_CLASS to assign the job's deadline and local variable LOCAL_CLASS. Therefore, jobs released after time 11 will have deadlines determined by periods of CL_2 and local variables LOCAL_CLASS = CL_2.

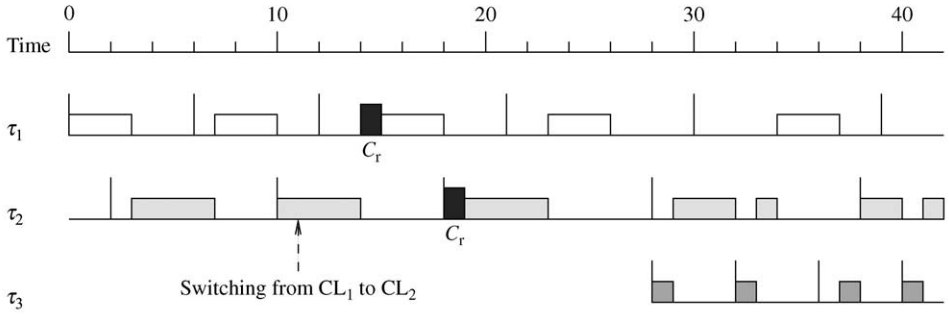


Figure 6. Example of service class switching by an arrival.

Every track task checks its LOCAL_CLASS at the beginning of execution at each sampling period. Thus, the first instance of each track task released after time 11 can notice that its LOCAL_CLASS $= \mathrm{CL}_2$ is different to the previous one, that is, CL_1. Therefore, they consume $C_\mathrm{r} = 1$ (marked as dark shaded tall box in the figure) to reconfigure objects related to the sampling frequency such that successful tracking with the new sampling frequency of CL_2 can be performed. After all existing track tasks are reconfigured, the new track task is started at time 28.[2]

To avoid potential loss of target tracks, we should guarantee deadlines all through the time line. Before the service class switching time at time 11 in Figure 6, we can guarantee all deadlines if we make the total utilization in CL_1 $(3/6 + 4/8)$ less than or equal to 1. Also, after the completion time of all reconfigurations at time 28 in Figure 6, all deadlines are guaranteed if we make the total utilization in CL_2 $(3/9 + 4/10 + 1/4)$ less than or equal to 1. However, during the time interval of reconfiguration from time 11 to time 28 in Figure 6, three types of jobs are mixed: jobs using sampling periods of CL_1, jobs with reconfiguration time C_r using sampling periods of CL_2, and jobs using sampling periods of CL_2. Therefore, to guarantee all deadlines in the time interval, a certain relation between CL_1 and CL_2 should be satisfied. We derive such a relation in Theorems 1 and 2. In the following, a job J_i with release time r_i and deadline d_i is said to be active at time t if $r_i < t \leq d_i$.

THEOREM 1 *A system of preemptable jobs is schedulable according to the EDF algorithm if the sum of densities $C_i/(d_i - r_i)$ of all active jobs J_i is no greater than 1 at all times (Liu, 2000).*

THEOREM 2 *Consider a system state change from s_1 to s_2 by arrivals of new targets that triggers a service class switching from CL_1 to CL_2. Let $U^{\mathrm{CL}_1}(s_1)$ and $U^{\mathrm{CL}_2}(s_2)$ be the total utilization factors at states s_1 and s_2, that is, $\sum_{\tau_i \text{ at } s_1} C_i f_i^{\mathrm{CL}_1}$ and $\sum_{\tau_i \text{ at } s_2} C_i f_i^{\mathrm{CL}_2}$, respectively. If the following conditions hold, all deadlines are guaranteed.*

$$U^{\mathrm{CL}_1}(s_1) \leq 1,$$
$$U^{\mathrm{CL}_2}(s_2) \leq 1,$$
$$C_i f_i^{\mathrm{CL}_1} \geq (C_i + C_\mathrm{r}) f_i^{\mathrm{CL}_2}, \textit{ for all tasks } \tau_i \textit{ at } s_1.$$

Proof: Suppose that the service class switching occurs at time t_s and all reconfigurations complete at t_f. Before t_s, the sum of densities of all active jobs is no greater than 1 since $U^{\mathrm{CL}_1}(s_1) \leq 1$. During the interval $[t_s, t_f)$, the density of some task τ_i at s_1 becomes $(C_i + C_\mathrm{r}) f_i^{\mathrm{CL}_2}$ and then $C_i f_i^{\mathrm{CL}_2}$. Since $C_i f_i^{\mathrm{CL}_1} \geq (C_i + C_\mathrm{r}) f_i^{\mathrm{CL}_2} \geq C_i f_i^{\mathrm{CL}_2}$ and only tasks at s_1 are scheduled before t_f, the sum of densities of all active jobs at all times in $[t_s, t_f)$ is no greater than 1. From the time t_f, the sum of densities of all active jobs is no greater than 1 since $U^{\mathrm{CL}_2}(s_2) \leq 1$. By Theorem 1, all deadlines are guaranteed. ■

If the conditions in Theorem 2 hold between two service classes CL_1 and CL_2, the reconfiguration in the reverse direction, that is, from CL_2 to CL_1, by departures of existing targets can be performed as in Figure 7 guaranteeing all deadlines. In Figure 7,

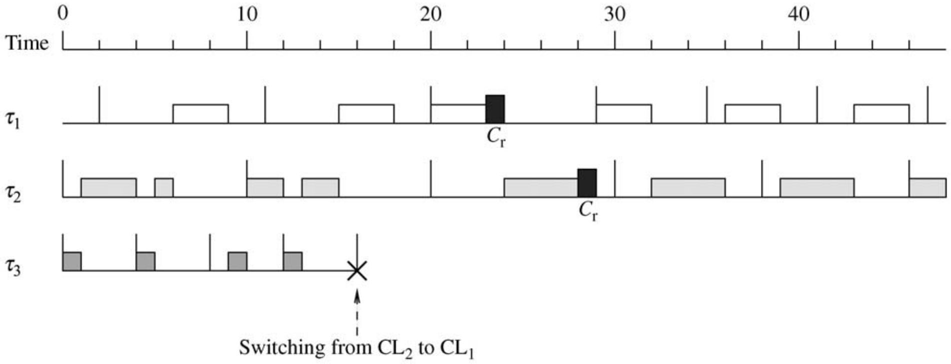


Figure 7. Example of service class switching by a departure.

the system is originally at a state with three track tasks ($\tau_1 = (C_1 = 3, f_1^{\mathrm{CL}_2} = 1/9)$, $\tau_2 = (C_2 = 4, f_2^{\mathrm{CL}_2} = 1/10)$, $\tau_3 = (C_3 = 1, f_3^{\mathrm{CL}_2} = 1/4))$ covered by CL$_2$. At time 16, a target τ_3 departs. Thus, the scheduler changes the system state and notices that the new state is covered by the new service class CL$_1$, whose predefined frequency allocation is $(f_1^{\mathrm{CL}_1} = 1/6,\ f_2^{\mathrm{CL}_1} = 1/8)$. Then, the scheduler changes the global variable GLOBAL_CLASS from CL$_2$ to CL$_1$. After time 16, the scheduler will use GLOBAL_CLASS = CL$_1$ to determine the local variables LOCAL_CLASS of all released jobs. The first instance of each task after time 16 notices the service class switching from CL$_2$ to CL$_1$ at the beginning of the execution by checking its LOCAL_CLASS. Unlike the previous type of switching caused by arrivals of new targets in Figure 6, for this type of switching, the new period of CL$_1$ will be applied from the second instance after the service class switching. That is, in Figure 7, the periods of the first instances after time 16 are still those of CL$_2$ and the new periods of CL$_1$ will be applied from the next instance. Thus, the first instance executes with the original period and then reconfigures objects at the end of execution such that the new period can be used from the next instance. Theorem 3 has the formal proof.

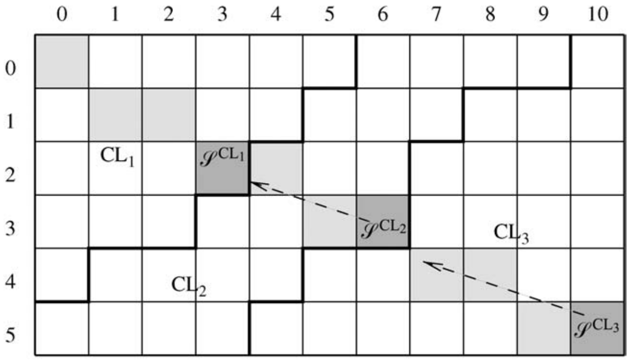
THEOREM 3 *Consider a system state change from s_2 to s_1 by departures of existing targets that triggers a service class switching from* CL$_2$ *to* CL$_1$. *If the conditions in Theorem 2 hold, all deadlines are guaranteed.*

Proof: Suppose that the service class switching occurs at time t_s. Let the release time of the first job after t_s be t_b. Before t_b, the sum of densities of all active jobs is no greater than 1 since $U^{\mathrm{CL}_2}(s_2) \leq 1$. After t_b, the density of some task τ_i at s_1 becomes $(C_i + C_\mathrm{r})f_i^{\mathrm{CL}_2}$ and then $C_i f_i^{\mathrm{CL}_1}$. It is clear that jobs of departed tasks are not active after t_b. That is, only jobs of tasks in s_1 can be active after t_b. Since $(C_i + C_\mathrm{r})f_i^{\mathrm{CL}_2} \leq C_i f_i^{\mathrm{CL}_1}$ and $U^{\mathrm{CL}_1}(s_1) \leq 1$, the sum of densities of all active jobs at all times after t_b is no greater than 1. By Theorem 1, all deadlines are guaranteed. ■

3.2. Heuristic Approach to Designing Service Classes

In this section, we give a heuristic design method to find a maximal set of service classes without explicit reservation for reconfigurations by using the conditions explained in Section 3.1. The overall idea of our approach is as follows. We position base states of service classes on the diagonal of the workload space as shown in Figure 8 where the diagonal states are lightly shaded and base states are darkly shaded. A state on the diagonal is represented by $(\lceil N_1(i/N)\rceil, \ldots, \lceil N_K(i/N)\rceil)$ where $i = 0, \ldots, N$ and $N = \max_{k\in\{1,\ldots,K\}} N_k$. When $i = N$, it represents the worst case workload state $(N_1, \ldots, N_K)$. When $i = 0$, it represents the zero workload state $(0, \ldots, 0)$. Among all diagonal states, we choose a subset of them as a set of base states[3] using a method explained later. For the chosen base states, we optimize the sampling frequency allocation using a non-linear optimization technique, which will also be explained later. The sampling frequency allocation optimized for each base state forms a service class. The covering region of each service class is determined by checking the schedulability at each state using the frequency allocation of the service class. In this way, we can partition the workload space into a number of covering regions that are covered by corresponding service classes as shown in Figure 8.

To guarantee all deadlines even if service class switching happens, we need to make the conditions in Theorem 2 hold for any pair of two service classes. We also need to guarantee the minimum sampling frequency of each type of target even at the worst case workload state. To do so, we start with the base state on the worst case workload state $(N_1, \ldots, N_K)$. The service class based on the worst case workload is denoted as CL_M.


0
1
2
3
4
5
6
7
8
9
10
0
1
2
3
4
5
CL1
$\mathscr{S}^{\mathrm{CL}_1}$
$\mathscr{S}^{\mathrm{CL}_2}$
CL3
CL2
$\mathscr{S}^{\mathrm{CL}_3}$

$\mathscr{F}^{\mathrm{CL}_1} = (f_1^{\mathrm{CL}_1} = 100\text{ Hz},\ f_2^{\mathrm{CL}_1} = 80\text{ Hz})$

$\mathscr{F}^{\mathrm{CL}_2} = (f_1^{\mathrm{CL}_2} = 70\text{ Hz},\ f_2^{\mathrm{CL}_2} = 60\text{ Hz})$

$\mathscr{F}^{\mathrm{CL}_3} = (f_1^{\mathrm{CL}_3} = 50\text{ Hz},\ f_2^{\mathrm{CL}_3} = 40\text{ Hz})$

Figure 8. Service class design on the diagonal.

When the workload state is $(n_1, \ldots, n_K)$, the optimal frequency allocation problem is formulated as follows:

$$\begin{aligned}
&\text{maximize}\\
&\qquad \sum_{k=1}^{K} n_k V_k(f_k) = \sum_{k=1}^{K} n_k w_k\left(1 - \mathrm{e}^{-\alpha_k f_k + \beta_k}\right)\\
&\text{subject to} \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (2)\\
&\qquad \sum_{k=1}^{K} n_k C_k f_k \le 1,\\
&\qquad f_k \ge f_{k,\min}, \qquad \text{for all } k = 1, \ldots, K.
\end{aligned}$$

In the formulation, the objective function $\sum_{k=1}^{K} n_k V_k(f_k)$ represents the total tracking quality we can achieve at state $(n_1, \ldots, n_K)$ if we use sampling frequencies $(f_1, \ldots, f_K)$. The first constraint says that the total utilization should be no greater than 1 for the schedulability. The second constraint forces each sampling frequency f_k to be larger than or equal to the minimum sampling frequency $f_{k,\min}$ of the corresponding target type k.

Since both the objective function and the constraint functions are convex, we can solve the problem using the nonlinear optimization technique based on Kuhn–Tucker conditions (Peressini et al., 1980) as in (Seto et al., 1996). We present the solution in the following. The proof is similar to that in (Seto et al., 1996). Interested readers can be referred to Seto et al. (1996).

$$f_k = \begin{cases} f_{k,\min}, & k = 1, 2, \ldots, p \\ \frac{1}{\alpha_k}(\beta_k + \ln \Gamma_k - Q), & k = p+1, \ldots, K \end{cases}$$

where

$$\Gamma_k = \frac{w_k \alpha_k}{C_k},$$

$$Q = \frac{1}{\sum_{k=p+1}^{K} (n_k C_k / \alpha_k)} \left(\sum_{k=1}^{p} n_k C_k f_{k,\min} + \sum_{k=p+1}^{K} \frac{n_k C_k}{\alpha_k} (\beta_k + \ln \Gamma_k) - 1 \right)$$

$f_1, \ldots, f_K$ are ordered according to $f_{1,\min}, \ldots, f_{K,\min}$ which are arranged as

$$\Gamma_1 \mathrm{e}^{-\alpha_1 f_{1,\min} + \beta_1} \le \Gamma_2 \mathrm{e}^{-\alpha_2 f_{2,\min} + \beta_2} \le \cdots \le \Gamma_K \mathrm{e}^{-\alpha_K f_{K,\min} + \beta_K}$$

and $p \in \{1, \ldots, K\}$ is the largest integer such that

$$\sum_{k=1}^{p} n_k C_k f_{k,\min} + \sum_{k=p+1}^{K} \frac{n_k C_k}{\alpha_k} \left(\alpha_p f_{p,\min} + \ln \frac{\Gamma_k}{\Gamma_p} + \beta_k - \beta_p \right) \ge 1$$

The above solution when $(n_1, \ldots, n_K) = (N_1, \ldots, N_K)$ is the frequency allocation for CL_M. To find the base state of the next service class CL_{M-1}, we scan the diagonal states $(\lceil N_1(i/N) \rceil, \ldots, \lceil N_K(i/N) \rceil)$ (changing i from $N-1$ to 0) to check if the following conditions can hold. These conditions are representations of the first and third conditions in Theorem 2 (the second condition was already satisfied when designing CL_M).

$$\sum_{k \in \{1,\ldots,K\}} \left\lceil N_k \frac{i}{N} \right\rceil C_k f_k \leq 1 \tag{3}$$

$$f_k \geq \frac{(C_k + C_\mathrm{r})}{C_k} f_k^{\mathrm{CL}_M} \quad \text{for all } k \in \{1,\ldots,K\} \tag{4}$$

At state $(\lceil N_1(i/N)\rceil, \ldots, \lceil N_K(i/N)\rceil)$, if the minimum required utilization U_min to satisfy the condition (4) (such U_min is simply given by $\sum_{k\in\{1,\ldots,K\}} \lceil N_k(i/N)\rceil C_k((C_k + C_\mathrm{r})/C_k)f_k^{\mathrm{CL}_M})$ is less than or equal to 1, the above conditions can be met. Otherwise, there cannot be a solution that satisfies the condition (3). When workload is heavy (i.e., when i is large), U_min is likely to be larger than 1. As scanning the diagonal states from $i = N - 1$ to $i = 0$, we can find the first state where U_min becomes less than or equal to 1. At the diagonal state, we make CL_{M-1}. When finding the optimal frequency allocation for CL_{M-1}, we use the formulation in (2) by replacing $f_{k,\mathrm{min}}$ with $((C_k + C_\mathrm{r})/C_k)f_k^{\mathrm{CL}_M}$.

By repeating this process until we reach the diagonal state $(0,\ldots,0)$ as shown in Figure 8, we can find $\mathrm{CL}_{M-2}, \mathrm{CL}_{M-3}, \ldots, \mathrm{CL}_1$. This is the maximal addition of service classes on the diagonal states satisfying conditions (3) and (4).

This process can be intuitively understood as follows. Satisfying the conditions (3) and (4) when adding service classes makes enough distances between two neighboring base states. The utilization gap due to this distance can handle the reconfiguration costs during service class switching.[4] In this way, we can design each service class without explicitly reserving a budget for the reconfigurations.

Once we find the frequency allocations for $\mathrm{CL}_1, \mathrm{CL}_2, \ldots, \mathrm{CL}_M$, the next step is to determine their covering regions. The service class that covers a system state $(n_1, \ldots, n_K)$ is determined by checking the following feasibility condition:

Covering feasibility condition A state $(n_1, \ldots, n_K)$ can be covered by a service class CL_i if

$$\sum_{k=1}^{K} n_k C_k f_k^{\mathrm{CL}_i} \leq 1 \tag{5}$$

The covering service class of a state $(n_1, \ldots, n_K)$ is defined as the first service class (from CL_1 to CL_M) that satisfies the above feasibility condition. Using this method, we can determine the covering regions of the service classes partitioning the workload space. The final shape of the partitions by the covering regions looks like Figure 8.

THEOREM 4 *For any system state change that triggers a switching between any pair of service classes, all deadlines are guaranteed.*

Proof: For any pair of service classes CL_i and CL_j $(i < j)$, $f_k^{\mathrm{CL}_i} \geq ((C_i + C_\mathrm{r})/C_i)f_k^{\mathrm{CL}_j}$ for all $k \in \{1, \ldots, K\}$. For any two system states s_i and s_j, respectively covered by CL_i and CL_j, let $U^{\mathrm{CL}_i}(s_i)$ and $U^{\mathrm{CL}_j}(s_j)$ be the utilization factors at s_i and s_j. By the covering feasibility condition (5), $U^{\mathrm{CL}_i}(s_i) \leq 1$ and $U^{\mathrm{CL}_j}(s_j) \leq 1$. Therefore, all conditions in Theorems 2 hold. This follows the theorem. ■

3.3. Generalization of the Proposed Approach

Until now, we assumed that the computation time C_k is fixed and only the sampling frequency f_k is adjustable. Also, we assumed that the tracking quality function $V_k(f_k)$ is defined as a continuous convex function. This section shows how to relax those assumptions using the Q-RAM approach (Rajkumar et al., 1997, 1998; Lee et al., 1998).

Consider a situation where a track task for a target type $k \in \{1, \ldots, K\}$ has two algorithm options (Algorithms 1 and 2) and 5 sampling frequency options (10, 20, 30, 40, and 50 Hz). Using Algorithms 1 and 2 at each sampling period takes 2 and 3 ms, respectively. With these options, we can draw two tracking quality functions $V_k^{\mathrm{Algo1}}(f_k)$ and $V_k^{\mathrm{Algo2}}(f_k)$ as shown in Figure 9(a). These functions can be simply translated into functions of assigned resource R_k since $C_k f_k = R_k$. Figure 9(b) shows the translated functions $V_k^{\mathrm{Algo1}}(R_k)$ and $V_k^{\mathrm{Algo2}}(R_k)$. By taking the maximum of the two functions, we can obtain the final tracking quality function $V_k(R_k)$ (the thick line in Figure 9(b)):

$$V_k(R_k) = \max(V_k^{\mathrm{Algo1}}(R_k), V_k^{\mathrm{Algo2}}(R_k)) \tag{6}$$

$V_k(R_k)$ is represented by a list of discrete R–V (resource–value) pairs in increasing R-order as follows:

$$V_k(R_k) = \left\langle \begin{pmatrix} V_{k1} \\ R_{k1} \end{pmatrix}, \ldots, \begin{pmatrix} V_{kq_k} \\ R_{kq_k} \end{pmatrix} \right\rangle$$

Associated with each R_{kj} $(1 \leq j \leq q_k)$, its computation time and frequency option selection $Q_{kj} = (C_{kj}, f_{kj})$ is determined from Equation (6) by noting the option that gave the maximum.

If such $V_k(R_k)$ is given for each target type $k \in \{1, \ldots, K\}$, the optimal resource allocation problem at state $(n_1, \ldots, n_K)$ is formulated as follows:

$$\begin{aligned} &\text{maximize} \\ &\qquad \sum_{k=1}^{K} n_k V_k(R_k) \\ &\text{subject to} \\ &\qquad \sum_{k=1}^{K} n_k R_k \leq 1, \\ &\qquad R_k \geq R_{k,\min}, \quad \text{for all } k = 1, \ldots, K \end{aligned} \tag{7}$$

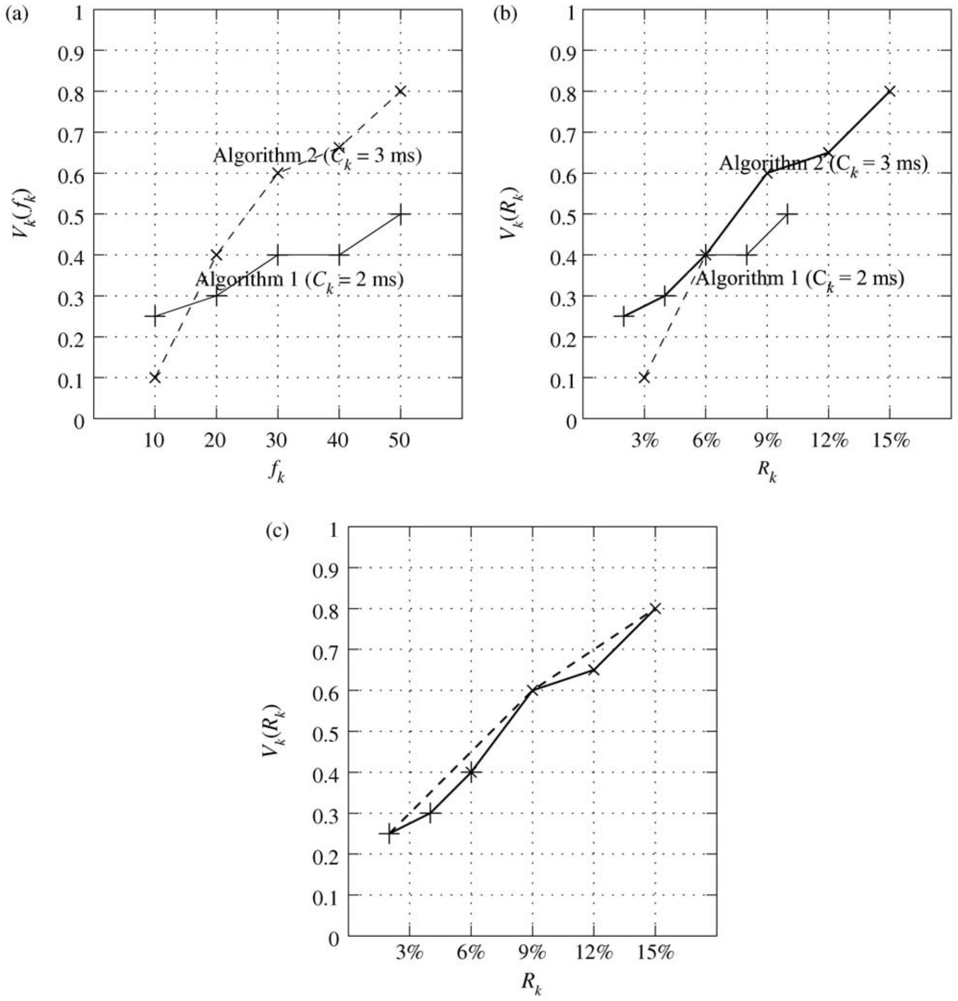


Figure 9. Tracking quality function with multiple dimension discrete options.

This problem can be solved by optimal or near-optimal algorithms in Rajkumar et al. (1998) and Lee et al. (1998). We present a near-optimal algorithm that solves the above problem by constructing the convex hull for each $V_k(R_k)$ as shown in Figure 9(c), which is a variation of the one in Lee et al. (1998). In the following algorithm, R^c denotes the residual resource capacity; $s\_list[j].tid$, $s\_list[j].R$, and $s\_list[j].V$ denote the target type id, the associated R and V values of the corresponding R–V pair; R_k denotes the resource allocated for target type k.

Near-Optimal Algorithm Find resource allocation R_k (and associated $Q_k = (C_k, f_k)$) for each target type k, $1 \leq k \leq K$ when the system state is $(n_1, \ldots, n_K)$

```
Input: $(n_1, \ldots, n_K)$, $(V_1(R_1), \ldots, V_K(R_K))$, $(R_{1,\min}, \ldots, R_{K,\min})$
Output: Option selection of each type $(Q_1, \ldots, Q_K)$ and
    total value $V$
begin procedure
 1. for $k = 1$ to $K$ do
 2.     in $V_k(R_k) = \left\langle \begin{pmatrix} V_{k1} \\ R_{k1} \end{pmatrix}, \ldots, \begin{pmatrix} V_{kq_k} \\ R_{kq_k} \end{pmatrix} \right\rangle$, find the first $m$ such that $R_{km} \geq R_{k,\min}$
 3.     $R_k := R_{km}$
 4.     $V'_k(R_k) :=$ convex_hull_frontier $(V_k(R_k))$ with the start point of $\begin{pmatrix} V_{km} \\ R_{km} \end{pmatrix}$
 5. end for
 6. $s\_list :=$ merge_and_sort$(V'_1(R_1), \ldots, V'_K(R_K))$
 7. $R^c := 1 - \sum_{k=1}^{K} n_k R_k$
 8. if $R^c < 0$ return (``insufficient resource'')
 9. for $j = 1$ to $|s\_list|$ do
10.     $k := s\_list[j].tid$
11.     $\beta := (s\_list[j].R - R_k) \times n_k$
12.     if $\beta \leq R^c$ then $R^c := R^c - \beta$, $R_k := s\_list[j].R$
13.     else continue
14. end for
15. determine $(Q_1, \ldots, Q_k)$ from $(R_1, \ldots, R_k)$
16. $V = \sum_{k=1}^{K} n_k V_k(R_k)$
17. return$(Q_1, \ldots, Q_k, V)$
end procedure
```

The loop from line 1 to 5 assigns the minimum resource allocation to each target type. Also, it finds the convex_hull_frontier of each $V_k(R_k)$ with the starting point of the R–V pair that first satisfies the minimum resource requirement. Line 6 merges all convex_hull_frontiers into a single list *s_list* in decreasing R–V-gradient-order. In line 7, the residual resource capacity R^c is set to the the amount of resource left by the minimal resource allocation. If R^c is negative, we cannot satisfy even the minimum resource requirements and hence immediately return. From line 9 to 14, we further assign resources starting from the type k with the largest R–V-gradient until the residual capacity R^c is exhausted. After finding the resource allocation for each type $(R_1, \ldots, R_K)$, line 15 determines its corresponding option selection of each type $(Q_1, \ldots, Q_k)$. The total value V we can achieve by using the resource allocation $(R_1, \ldots, R_K)$ at state $(n_1, \ldots, n_K)$ is given in line 16.

One thing we need to take care is to satisfy the conditions in Theorem 2 between two neighboring service classes CL_i and CL_{i+1} to guarantee all deadlines. The first and second conditions are automatically satisfied by the formulation in (7). The third condition is represented as follows in this context:

$$C_k^{\mathrm{CL}_i} f_k^{\mathrm{CL}_i} \geq (C_k^{\mathrm{CL}_{i+1}} + C_{\mathrm{r}}) f_k^{\mathrm{CL}_{i+1}}, \quad \text{for all } k = 1, \ldots, K$$

To satisfy this condition when designing CL_i,

$$R_{k,\min} = (C_k^{\mathrm{CL}_{i+1}} + C_\mathrm{r})f_k^{\mathrm{CL}_{i+1}}$$

Remember that $C_k^{\mathrm{CL}_{i+1}}$ and $f_k^{\mathrm{CL}_{i+1}}$ are known values when designing CL_i since we design service classes from CL_M to CL_1.

3.4. Sub-typing of a Physical Target Type Considering Target States

The tracking value of a target type k that can be achieved by using resource R_k in fact depends on not only the target type (e.g., missile, hostile airplane, airplane, etc.) but also target states such as distance and speed. For example, if a target is far away, we can achieve an appropriate tracking value by using a low sampling frequency. On the other hand, for tracking a close target, we must use a high sampling frequency to achieve a desirable tracking value. Also, tracking a fast moving target requires a higher sampling frequency than tracking a slow moving target.

If we specify the tracking quality function independently to target states, we should assume the worst case target state, that is, the closest distance and the maximum speed. Such tracking quality specification may result in an ineffective resource allocation that unnecessarily gives much resource to a target in a non-critical state.

Our service class design approach can handle this problem by allowing finer-grained classification of target types. In other words, a single (physical) target type can be further classified into a number of sub-types (logical target types) depending on the target state. For example, we can divide the surveillance space into a finite number of regions (e.g., critical region, tracking region, and non-critical region)[5] as shown in Figure 10. A target of (physical) type k (e.g., missile) in different regions can be regarded as different (logical) types (e.g., missile in critical region, missile in tracking region, and missile in non-critical region). If the target crosses the region boundary, we can handle this as if a departure of old (logical) type and an arrival of new (logical) type happen.

With this finer-grained type classification considering target states, all possible (logical) target types are represented as follows:

$$\left(\underbrace{\mathrm{Type}_{11},\ldots,\ \mathrm{Type}_{1L_1}}_{\mathrm{Type}_1},\ldots,\ \underbrace{\mathrm{Type}_{K1},\ldots,\ \mathrm{Type}_{KL_k}}_{\mathrm{Type}_K}\right)$$

where L_k is the number of sub-types of (physical) type $k \in \{1,\ldots,K\}$.

The maximum number of target instances is also specified for each sub-type as follows:

$$\left(\underbrace{N_{11},\ldots,N_{1L_1}}_{N_1},\ldots,\underbrace{N_{K1},\ldots,N_{KL_k}}_{N_K}\right)$$

For example, if the original specification on the maximum number of target instances of type k was $N_k = 10$ (i.e., 10 missiles), the fine-grained specification can be given as

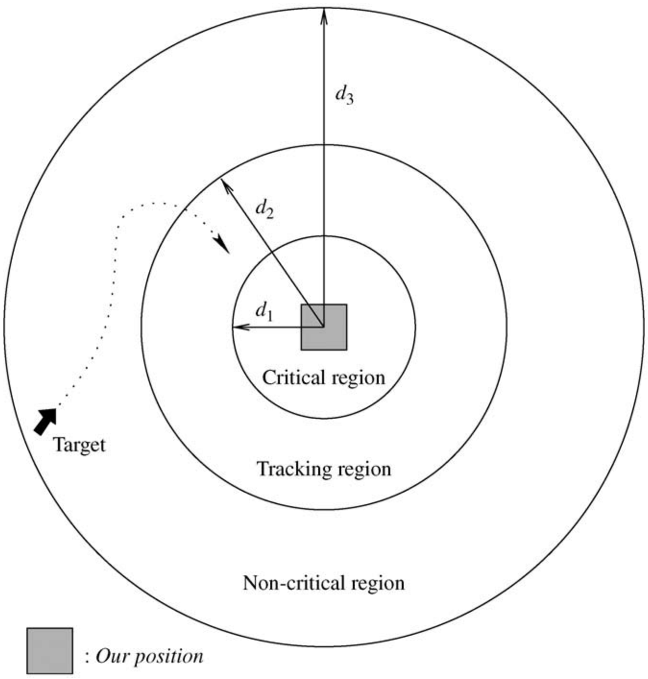


Figure 10. Division of surveillance space into discrete regions.

$N_{k1} = 3$, $N_{k2} = 5$, and $N_{k3} = 2$ (i.e., 3 critical region missiles, 5 tracking region missiles, and 2 non-critical region missiles).

The tracking quality function associated to fine-grained target types Type_{kl} can now be specified as

$$V_{kl}(f_{kl}, C_{kl})$$

The remaining process of service class design is the same as explained in Sections 3.2 and 3.3.

One serious problem of fine-grained type classification is explosion of workload space. Assuming $\Sigma_{l=1}^{L_k} N_{kl} = N_k$ for all $k \in \{1, \ldots, K\}$, the number of workload states for the original type classification is $N_1 \times \cdots \times N_K = \sum_{l=1}^{L_1} N_{1l} \times \cdots \times \sum_{l=1}^{L_K} N_{Kl}$ but that for the fine-grained classification is $\prod_{l=1}^{L_1} N_{1l} \times \cdots \times \prod_{l=1}^{L_K} N_{Kl}$. This raises the scalability issue of online QoS management.

Our service class based QoS management is scalable to the explosion of workload space because the number of service classes used online is bounded by $\max_{\text{all } k,l} N_{kl}$ and time consuming optimizations are performed offline. However, the approach that uses optimal resource allocations for all different states become more and more impractical as workload space explodes because the number of resource allocations managed online also explodes.

3.5. Dual Resource Problem: Processor and Antenna

Until now, we explained our service class approach focusing on the processor resource. This subsection explain how our approach can be extended to handle the situation of two different types of resources: processor and antenna.

As explained in Figure 3, a track task can be modeled as an end-to-end tasks with two subtasks: processor subtask and antenna subtask. If we use a nongreedy synchronization protocol (Bettati, 1994; Sun, 1997) between the processor and antenna, each subtask can be modeled as an independent periodic task on each resource. For assigning the intermediate deadline for each subtask to meet the end-to-end deadline, we can use one of deadline-assignment algorithms (Kao and Garcia-Molina, 1993, 1994; Sun, 1997) reported in the literature. Here, we use the proportional deadline algorithm (Kao and Garcia-Molina, 1993; Sun, 1997) that assigns a fraction of the end-to-end deadline to each subtask in proportion to its computation time. Thus, the intermediate deadlines of the processor and antenna subtasks of a track task of type k are given as follows:

$$\text{intermediate deadline of processor subtask} = \frac{1}{f_k}\left(\frac{C_k^{\text{pro}}}{C_k^{\text{pro}} + C_k^{\text{ant}}}\right)$$

$$\text{intermediate deadline of antenna subtask} = \frac{1}{f_k}\left(\frac{C_k^{\text{ant}}}{C_k^{\text{pro}} + C_k^{\text{ant}}}\right)$$

where f_k is the sampling frequency and C_k^{pro} and C_k^{ant} are computation times on the processor and antenna, respectively.

With these settings of intermediate deadlines, we can perform the schedulability analysis on each resource independently when designing the service classes. Considering the intermediate deadline, the processor schedulability condition (the first and second conditions in Theorem 2) should be changed as follows:

$$\sum_{k=1}^{K} n_k C_k^{\text{pro}} f_k \left(\frac{C_k^{\text{pro}} + C_k^{\text{ant}}}{C_k^{\text{pro}}}\right) \leq 1$$

The utilization gap constraint (the third condition in Theorem 2) for the reconfiguration cost on the processor is now represented as follows:

$$C_k^{\text{proc}} f_k^{\text{CL}_i} \left(\frac{C_k^{\text{pro}} + C_k^{\text{ant}}}{C_k^{\text{pro}}}\right) \geq (C_k^{\text{proc}} + C_{\text{r}}) f_k^{\text{CL}_{i+1}} \left(\frac{C_k^{\text{pro}} + C_k^{\text{ant}}}{C_k^{\text{pro}}}\right)$$

for all $k = 1, \ldots, K$.

In addition, we need to check the schedulability of the antenna resource. When doing this, we should note that the antenna is non-preemptive. So, the worst case non-preemptive portion $b = \max_{k \in \{1,\ldots,K\}} C_k^{\text{ant}}$ should be included as the blocking term B to the schedulability condition. Assuming the non-preemptive EDF scheduling, the schedulability condition considering the blocking effect is as follows:

$$\sum_{k=1}^{K} \left(n_k C_k^{\text{ant}} f_k \left(\frac{C_k^{\text{pro}} + C_k^{\text{ant}}}{C_k^{\text{ant}}}\right)\right) + B \leq 1$$

where $B = \max_{k \in \{1,\dots,K\}} bf_k((C_k^{\text{pro}} + C_k^{\text{ant}})/C_k^{\text{ant}})$.

With these new constraints, we can apply the same approach to design a set of service classes considering both processor and antenna resource.

4. Experimental Results

In this section, we compare the expected tracking quality of our approach with those of two alternative approaches: (1) worst case based approach and (2) all states based approach. The worst case based approach constantly uses the resource allocation optimized for the worst case workload. Thus, it does not require any online overhead for reconfiguration. On the other hand, the all states based approach uses a different resource allocation for each different workload state. Each resource allocation is optimized for the corresponding state. However, to guarantee deadlines while switching resource allocations, a budget for the reconfiguration cost C_{r} is explicitly reserved by adding C_{r} to the execution time C_k of each task when calculating the resource allocation. As an upper bound of the expected performance, we also present the ideal performance that is achieved by using the optimal resource allocation for each state ideally assuming *zero* reconfiguration cost.

Before explaining our experiments, we first discuss on the actual impact of the reconfiguration cost. For this, we implemented a Kalman filter algorithm for target tracking, and measured the computation time on a 500 MHz AMD K6-II processor. The Kalman filter consists of three code segments: (1) sampling frequency independent initialization part, (2) sampling frequency dependent initialization part, and (3) track estimation from return signal part. The first part is executed only once. The second part is executed once whenever the sampling frequency changes. The third part reflects the normal execution at each sampling period. We used the cycle counting ability of the AMD K6 processor to measure cycles taken by each part. The results are shown in Table 2.

We can regard the cycles taken by the sampling frequency dependent initialization part, that is, 4428 cycles, as the reconfiguration cost C_{r} we have to pay when changing the sampling frequency. The cycles taken by the first sampling period, that is, 8068, cycles can be regarded as the normal computation time C_k in the absence of cache. In this specific experiment, the reconfiguration cost C_{r} takes more than 50% of the normal computation time C_k. Note that the cycles of the first and second sampling periods have a big difference. From the second sampling period, the cycles are similar. This is because the first sampling period is executed when the code does not reside in the cache while the others are executed with the cached code. From this, we can estimate the reconfiguration overhead caused by algorithm change, which may result in cache replacement. As long as we use the same algorithm code, the code would be in the cache, so the normal computation time would be about 3500 cycles. If we change the algorithm, the first sampling period takes longer, 8068 cycles. Thus, the extra cost C_{r} is about 4500, which is more than 120% of the normal computation time C_k. From these measurements, we can notice that the reconfiguration cost is non-negligible.

Table 2. Kalman filter cycle measurements.

Code parts	Measured cycles
Sampling frequency independent initialization	19,643
Sampling frequency dependent initialization	4428
Track estimation from return signal	
1st sampling period	8068
2nd sampling period	3636
3rd sampling period	3522
4th sampling period	3491
⋮	⋮

Section 4.1 presents the experimental results where tracking quality functions are defined as continuous convex functions of sampling frequency. In Section 4.2, we show the results for discrete tracking quality functions of multiple QoS attributes, that is, sampling frequency and computation time.

4.1. Experiments with Continuous Convex Quality Functions of Single QoS Attribute

To keep the exposition simple, we use only two different target types, that is $K = 2$. We also fix the maximum number of target instances of each type as $N_1 = 15$ and $N_2 = 31$. The computation time C_k and the minimum sampling frequency $f_{k,\min}$ for each type $k \in \{1, 2\}$ are also fixed. These fixed parameters are summarized in Table 3.

To see the effect of the shapes of tracking quality functions, we performed the experiment for two sets of tracking quality functions: One set (Set 1) is highly sensitive to sampling frequency and the other (Set 2) is less sensitive. The parameters for the two sets are presented in Figure 11. In both Sets 1 and 2, the weights w_k and the minimal sampling frequencies $f_{k,\min}$ in Equation 1 are the same. However, smaller values of sensitivity α_k are used in Set 2 to see the effect of the sensitivity. The control values β_k in Set 2 are chosen such that the resulting minimum achievable performances are similar to those of Set 1 to make the comparison simple.

In the experiment, we assume poisson arrival of each type $k \in \{1, 2\}$ with arrival rate λ_k. Also, the time for which a target instance of each type k stays in the system is assumed to follow the exponential distribution with the parameter μ_k. These parameters are tightly related to the expected tracking performance since they determine the probability that the system is at each workload state. Therefore, we will investigate the effect of these

Table 3. Experimental parameters (fixed).

Parameters	Values
No. of different target types (K)	$K = 2$
Max no. of target instances of each type (N_k)	$N_1 = 15, N_2 = 31$
Computation time of each type (C_k)	$C_1 = 2\,\text{ms}, C_2 = 1.5\,\text{ms}$
Min sampling frequency of each type $(f_{k,\min})$	$f_{1,\min} = 10\,\text{Hz}, f_{2,\min} = 10\,\text{Hz}$

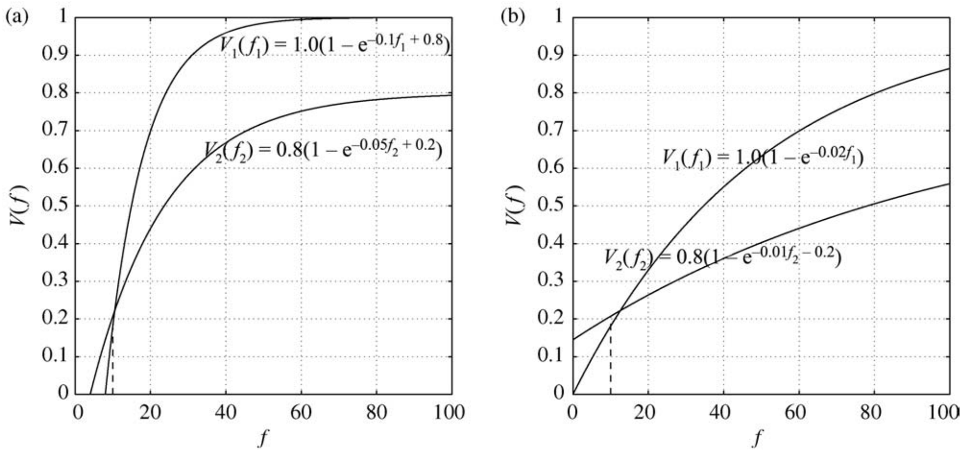


Figure 11. Tracking quality functions for Sets 1 and 2.

parameters as changing their values. We change those values such that the resulting average workload ρ ranges from 10 to 98% of the worst case workload. The time to perform a reconfiguration also severely affects the performance. Thus, we will also perform the experiment as changing the reconfiguration time. For illustrative purposes, we will use the normalized reconfiguration time C_r, which is the percentage of reconfiguration time relative to the average of computation times, that is, $(C_1 + C_2)/2$. Table 4 summarizes these changing parameters. The number in parentheses represents the default value used when we are changing other parameters.

Figure 12(a) and (b) compare the results as increasing the normalized reconfiguration time C_r from 0 to 100% for the two sets of tracking quality functions. Since the worst case based approach does not require any reconfiguration, its performance is independent of C_r. When C_r is very small, the overhead for reconfiguration is negligible. Thus, the performance of all states based approach is almost same as the ideal. The performance of our approach is also very close to the ideal but slightly below. The reason that our performance is slightly worse than that of all states based approach when C_r is very small can be explained as follows; Our approach uses resource allocation optimized only for a subset of diagonal states even for other states. On the other hand, the all states based approach can use the nearly ideal resource allocation for all individual states if the reconfiguration time is negligible.

As C_r increases, the performances of both of our and all states based approaches decrease. However, our approach decreases more slowly than the all states based

Table 4. Experimental parameters (Changing).

Parameters	Values
Average workload (ρ)	$\rho = 10\% \sim (50\%) \sim 98\%$
Normalized reconfiguration time (C_r)	$C_r = 0\% \sim (20\%) \sim 100\%$

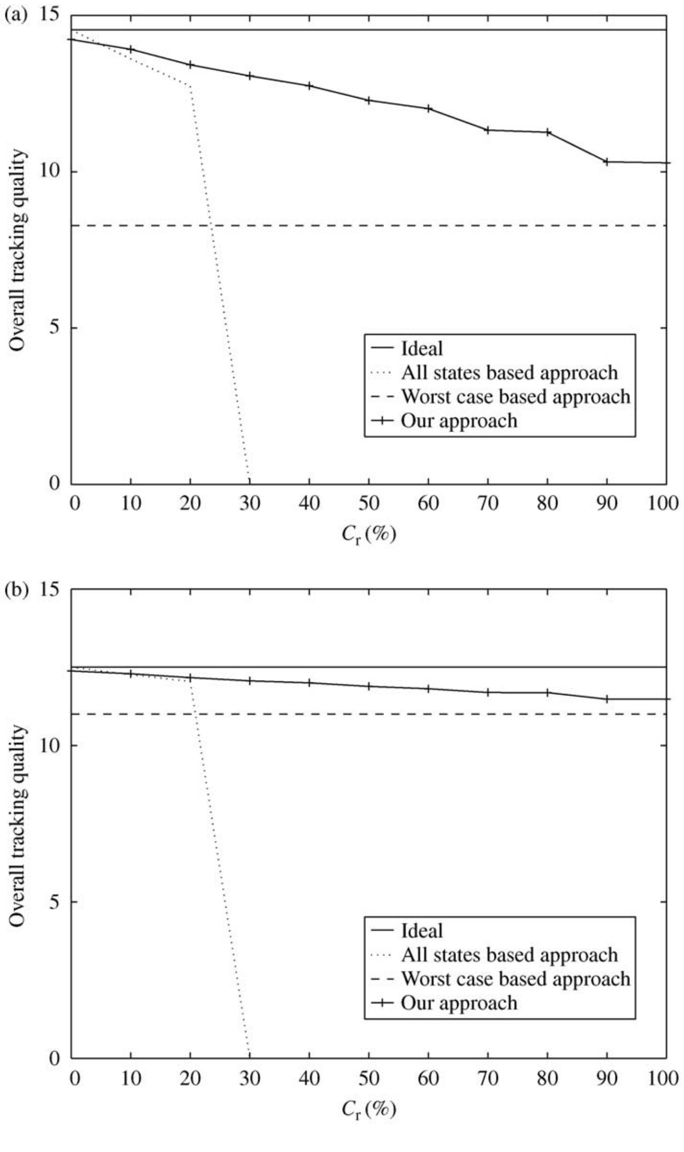


Figure 12. Overall tracking quality varying reconfiguration time C_r (with continuous quality functions).

approach. This is because the former does not explicitly reserve reconfiguration budget by adding only a proper number of service classes while the latter does to handle reconfigurations that happen whenever the state changes. Note that when C_r becomes 30%, the performance of all states based approach suddenly drops to zero since it cannot guarantee even the minimum requirements if we explicitly reserve the resource for 30% or more reconfiguration time. When C_r becomes very large, our approach automatically degenerates to the worst case based approach and thus its performance eventually merges to that of the worst case based approach. These trends are observed in both of Figure 12(a) and (b). Only difference is that the performance gap is clearer when tracking quality functions are sensitive to sampling frequencies (Figure 12(a) Set 1) than when they are less sensitive (Figure 12(b) Set 2).

Figure 13 shows the sum of tracking qualities of targets at each workload state when $C_r = 20\%$. In Figure 3, x, y, and z represent the number of target instances of Type 1, the number of target instances of Type 2, and the sum of tracking qualities, respectively. In both of Figure 13(a) and (b), the top-left, top-right, bottom-left, and bottom-right curves correspond to ideal, all states based approach, worst case based approach, and our approach, respectively. Since the worst case based approach uses a fixed frequency allocation regardless of workload state, the quality of each individual target is same at each workload state. Thus, the sum of tracking qualities at each state linearly increases as the number of targets increases. On the other hand, all other three can assign higher sampling frequencies for individual targets when workload is not heavy. Thus, the performances more quickly increase than the worst case approach at the beginning. As the workload becomes heavy, the sampling frequencies that can be assigned to individual targets become smaller and approach to those of the worst case based approach. The ideal performance eventually converges to that of the worst case based approach since zero reconfiguration cost is assumed. Our approach also converges to the worst case based approach when the workload is heavy, since no resource is need to be reserved for reconfigurations.[6] However, when the workload is heavy, all states based approach is worse than the worst case based approach since a large amount of resource is reserved for reconfigurations and only the remaining amount of resource is used for track tasks.

One interesting observation regarding the performance of the all states based approach in Figure 13(a) (top-right) is that there is a performance drop in heavy workload. This comes from the quality function shapes of Set 1 in Figure 11(a). For those functions, even a small increase of sampling frequency over the minimum can provide a significant quality improvement. When the workload is not so heavy, the all states based approach can allow a reasonable increase of sampling frequency over the minimum even after the reservation for reconfigurations. Thus, each individual track task can provide a high quality at such workload. However, when the workload is very heavy, only the minimum sampling frequency (or very close to the minimum, at most) can be allowed for each individual track task due to a large amount of reservation for reconfigurations. Thus, each track task can only provide the minimum quality. Therefore, it happens that the sum of the tracking qualities at heavy workload is smaller than that of lighter workload although the former state has a larger number of targets. Such performance drop in heavy workload is not observed in Figure 13(b), since its corresponding quality functions (Figure 11(b)) gradually increase.

Figure 14(a) presents the overall tracking qualities obtained by changing the average workload ρ from 10 to 98% for the tracking quality functions of Set 1. When ρ is very low, the system stays at low workload states at most of time. Thus, the performance at low workload states is the dominant factor of the overall performance. At low workload states, it is possible to allocate enough frequencies to tasks even though a certain amount

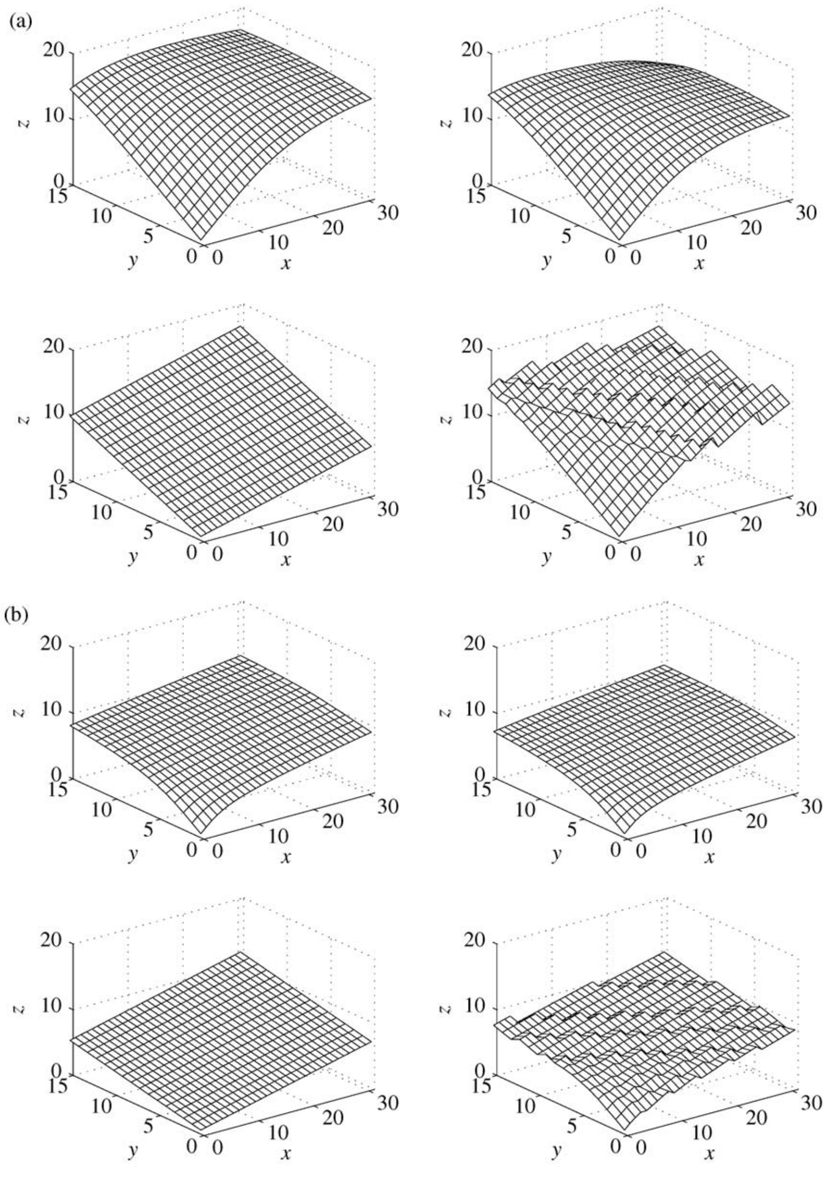


Figure 13. Sum of tracking qualities at each workload state.

of budget is reserved for reconfigurations. Therefore, the all states based approach that can adapt to the workload state shows the performance close to the ideal performance. Our approach also can adapt to the workload state with prepared service classes and thus shows almost ideal performance. However, since the worst case based approach cannot adapt to light workload states, its performance is much worse than the others.

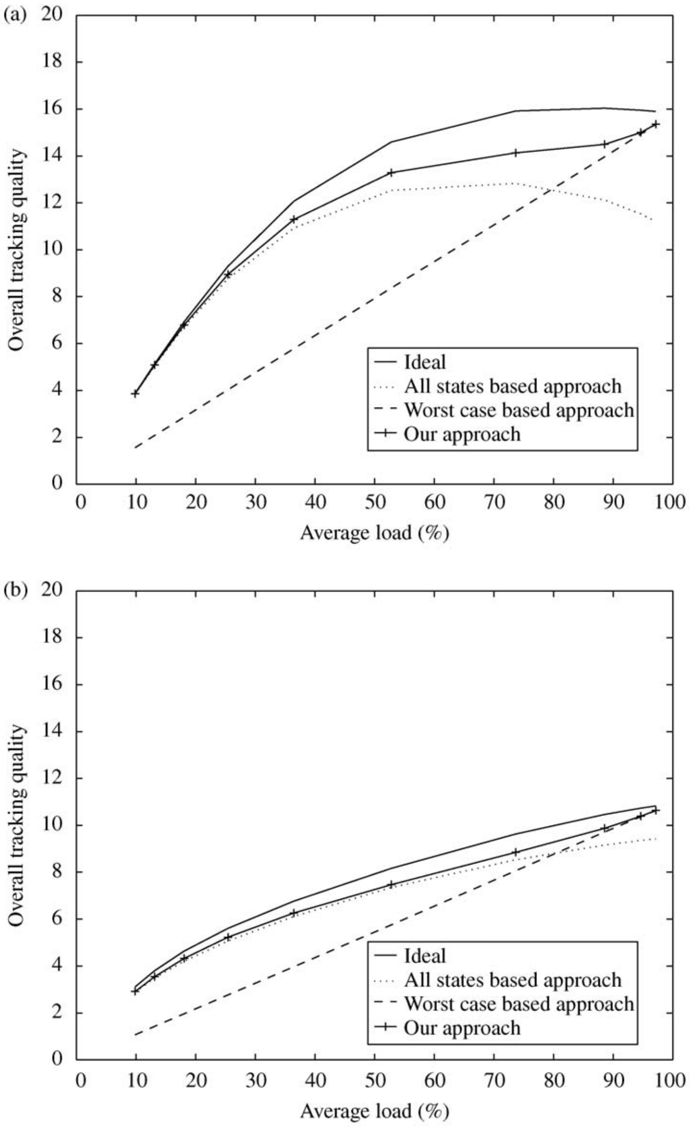


Figure 14. Overall tracking quality varying average workload ρ (with continuous quality functions).

As ρ increases, the system stays at medium and heavy workload states for a larger portion of time. Thus, providing good performance at those workload states becomes important. However, all states based approach has a small amount of available resource due to the explicitly reserved budget for reconfigurations. This small amount of resource quickly becomes insufficient for providing good quality as the system becomes heavy loaded. Therefore, the performance of all states based approach only slowly increases as

ρ increases and eventually starts to decrease when the heavy workload states become dominant. The performance decrease is due to the performance drop in heavy workload as explained in Figure 13(a). In contrast, the performance of our approach keeps increasing with the increase of ρ since our approach does not explicitly reserve resource for reconfigurations. When ρ is very high, our approach eventually merges to the worst case based approach that is optimized only for the worst case. Although less clear, similar trends are still observed in Figure 14(b) that shows the results for the set of less sensitive tracking quality functions.

4.2. Experiments with Discrete Quality Functions of Multiple QoS Attributes

As in Section 4.1, we use two different target types, that is, $K = 2$ and the maximum number of target instances of each type of $N_1 = 15$ and $N_2 = 31$. We also perform the experiment changing the average workload ρ and the normalized reconfiguration time C_r. For these changing parameters, we use the same values in Table 4.

However, tracking quality functions are defined differently. We assume that each track task have two algorithm options (Algo1 and Algo2) and nine sampling frequency options (10, 15, 20, 25, 30, 35, 40, 45, and 50 Hz). This means that there are 18 (2 × 9) option points among which each task can choose one. Tracking a target of type 1 with Algo1 and Algo2, respectively, takes computation times $C_1^{\text{Algo1}} = 2\,\text{ms}$ and $C_1^{\text{Algo2}} = 3\,\text{ms}$. On the other hand, tracking a target of type 2 with Algo1 and Algo2 takes $C_2^{\text{Algo1}} = 1.5\,\text{ms}$ and $C_2^{\text{Algo2}} = 2\,\text{ms}$. The experimental tracking value achieved by each option point is presented in Table 5.

Figure 15 shows experimental results obtained as changing C_r from 0 to 100%. As in Figure 12 for continuous quality functions, our approach outperforms the worst case based and all states based approaches under most of C_r values. Only when C_r is very small, the performance of our approach is slightly below that of all states based approach. All other trends are similar to Figure 12.

Figure 16 compares the results as increasing ρ from 10 to 98%. Also in this case, all trends observed in Figure 14 for continuous quality functions are retained. That is, our approach steadily outperforms other two approaches in the entire range of ρ whereas other two defeats each other depending on the ρ value.

Table 5. Tracking values at discrete quality points.

		Tracking value								
Type	C (ms)f\(Hz)	10	15	20	25	30	35	40	45	50
Type 1	$C_1^{\text{Algo1}} = 2\,\text{ms}$	0.10	0.30	0.50	0.60	0.65	0.70	0.74	0.77	0.80
	$C_1^{\text{Algo2}} = 3\,\text{ms}$	0.05	0.10	0.30	0.50	0.70	0.85	0.93	0.95	1.00
Type 2	$C_2^{\text{Algo1}} = 1.5\,\text{ms}$	0.10	0.30	0.40	0.50	0.55	0.60	0.62	0.67	0.70
	$C_2^{\text{Algo2}} = 2\,\text{ms}$	0.15	0.20	0.30	0.60	0.65	0.70	0.72	0.79	0.80

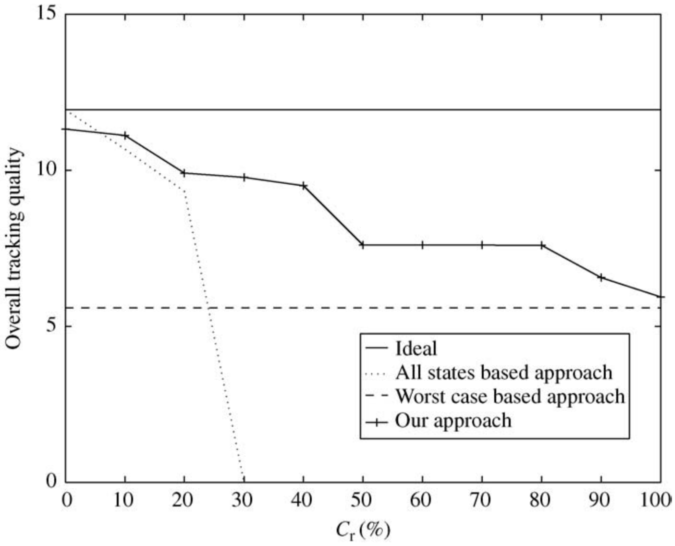


Figure 15. Overall tracking quality varying reconfiguration time C_r (with discrete quality functions).

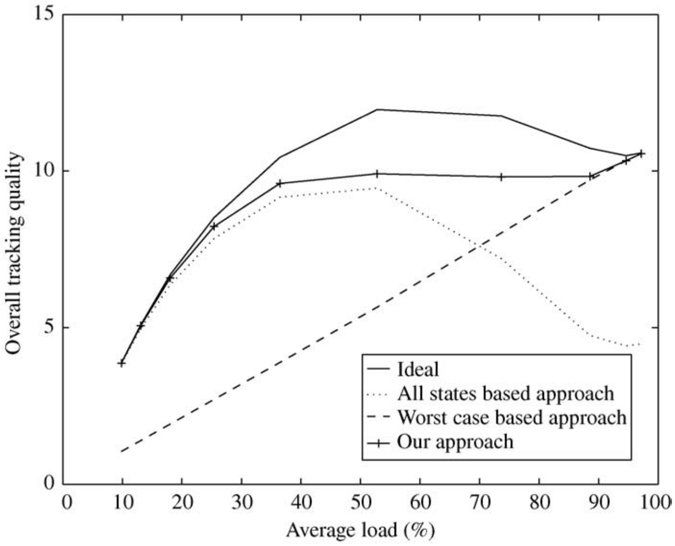


Figure 16. Overall tracking quality varying average workload ρ (with discrete quality functions).

5. Related Work

Many recent studies (Huizing and Bloemen, 1996, Chang et al., 1997; Kuo et al., 2000) have dealt with radar system design problems using real-time technology. For example, Kuo et al. (2000) formalize the workload of typical radar systems as periodic and aperiodic real-time tasks and propose effective task allocation and scheduling algorithms to process the tasks in a real-time fashion. They also provide a way to estimate the system capacity, which is essential to determine the needed amount of resource to meet the system specification in terms of workload. Chang et al. (1997) model each task in a radar system as a sequence of jobs with precedence relations and contending hardware resources such as digital signal processors, communication channels, and memory. To schedule these jobs within deadlines using as less resources as possible, they propose a scheduling algorithm that effectively resolves resource contention.

This paper also deals with task scheduling in radar systems in a real-time fashion. However, unlike the previous approaches, we employ the fact that tracking quality depends on the amount of assigned processing resource. Based on this, we try to maximize the tracking performance with the given amount of resource by effectively changing resource allocation depending on the workload.

The fact that the performance of a control task depends on the amount of assigned resource has been pointed out in many recent studies (Seto et al., 1996, 1998; Chandra and Sha, 1999; Caccamo et al., 2000). Seto et al. (1996, 1998) model the performance of each control task as a function of its sampling frequency. For a given set of control tasks with such performance functions, they propose a method that optimizes frequencies such that the weighted sum of performance of the control tasks is maximized. Chandra and Sha (1999) propose an integrated approach that can handle both reliability and schedulability of control tasks when finding their optimal sampling frequencies. Caccamo et al. (2000) handle the optimal frequency allocation problem when the normal and worst case computation times have a large variation.

Rajkumar et al. (1997, 1998) and Lee et al. (1998) extend the optimal resource allocation problem to a wider range of applications with requirements of multiple QoS dimensions (discrete and/or continuous) such as timeliness, security, and data quality. They present an analytic model called Q-RAM for QoS management in such applications. In the model, the utility gained by improvement along each QoS dimension with more resource is represented as a utility function. The resource allocation algorithms in Q-RAM can find the optimal or sub-optimal solutions that maximize the total system utility. A recent study in Lee et al. (1999) handles a more complex problem of allocating multiple resources (like CPU and network bandwidth) to satisfy the QoS needs of multiple tasks with multiple QoS dimensions.

Our approach is based on these optimization techniques. However, we deal with resource allocation for dynamic hard real-time systems. To avoid the online overhead for optimization, we use the optimization techniques only at the design phase to calculate the set of service classes. Also, we modify the optimization problem to handle multiple instances of the same type and to satisfy the conditions between service classes required for guaranteeing deadlines.

6. Conclusions

In this paper, we propose an approach to online QoS management based on service classes. This approach is illustrated in a simplified radar system applications. In surveillance radar systems, many different targets arrive and depart dynamically. This creates many difficulties in the online resource management problem for maximizing the tracking quality.

Our approach uses pre-computed templates of resource allocations called service classes as the basis for online QoS optimization. Experimental results show that this approach outperforms two traditional approaches in a wide range of system parameters.

However, there are open problems. First, the service class design assumes only the EDF scheduling, in which the optimization problem is much easier. Extension of the proposed approach to rate monotonic scheduling (Liu et al., 1973; Sha et al., 1994) is an open problem. Second, the conditions used in this paper for guaranteeing deadlines while switching service classes are only sufficient and not necessary. Thus, the maximum number of service classes we can add is conservatively limited. Finally, several interdependent problems for the optimal service class design are handled by heuristics. The optimal design of service classes is still an open problem. These limitations will be handled in future research since it is an ongoing study.

In spite of these limitations, we believe that the design approach based on the service class concept is an important step for online QoS management of many dynamic real-time systems. For example, in a multimedia service system through a shared bandwidth, the QoS of a media stream can be modeled as a function of its bandwidth share. If a large bandwidth is available, a high QoS can be delivered by transmitting all enhancement layers of a scalable encoded stream (e.g., MPEG2 (II, 1992) and fine-granular scalable video coding (Thomas)). Otherwise, we can deliver a reasonably degraded QoS by transmitting only a subset of layers. In this environment, the optimal resource sharing to maximize the total QoS of all active streams is important. With the dynamically changing active streams, it is impractical to apply an optimization procedure online due to its complexity. Thus, the pre-optimized resource sharing (i.e., the service class) for a specific workload region will help the optimal usage of limited resource without online optimization overheads. We are currently studying on such extension of our service class approach.

Acknowledgments

This work was sponsored in part by the MURI program N00014-01-0576, by the ONR N0004-02-0102, and by a grant from Lockheed Martin 1-5-36137.

Notes

1. Other forms of functions might better model the tracking quality. However, finding such functions is beyond the scope of this paper.
2. The confirmation task keeps tracking the new target until its track task is started. However, the time is short because the new track task can be started within 2 × max(existing tasks' periods) even in the worst case. The initial configuration of the new track task is performed by the confirmation task.

3. We consider diagonal states as candidates of base states due to the following reasons. First, it is hard to predict the ratio among target counts of different types and the ratio dynamically changes. The optimization based on diagonal states (at the centers of the ratio range) can provide a stable performance that is less sensitive to the change of the ratio. Second, diagonal states forms a totally-ordered set in terms of workload. As we will see later, this allows switching between any pair of service classes if the switching between neighboring ones is possible.
4. One way to reduce the necessary utilization gap is to use two different frequencies—one only for service class transition and the other for steady state in a single service class. However, the application of this idea is not trivial since using transition period will require one more period change resulting in another reconfiguration. We will investigate this in further study.
5. Dividing a continuous state space into a finite number of discrete states creates a large number of target subtypes but allows fine-grained tracking quality specification. Thus, finding an effective division of continuous state space is a challenging issue but is beyond the scope of this paper.
6. Some bumps in our approach is due to the boundaries of service classes.

References

Bates, D. M., and Watts, D. G. 1988. *Nonlinear Regression Analysis and Its Applications.* Wiley.

Baugh, R. A. 1973. *Computer Control of Modern Radars.* RCA M&SR-Moorestown Library.

Bettati, R. 1994. End-to-end scheduling to meet deadlines in distributed systems. Ph.D. thesis, Department of Computer Science, University of Illinois at Urbana-Champaign.

Billetter, D. R. 1987. Computers and data processing in radar. In M. J. M. Scanlan (ed.), *Modern Radar Techniques.* Collins Technical Books, Chapter 3.

Billetter, D. R. 1989. *Multifunction Array Radar*, Chapter 7. Artech House.

Caccamo, M., Buttazzo, G., and Sha, L. 2000. Elastic feedback control. In *Proceedings of the 12th Euromicro Conference on Real-Time Systems.*

Chandra, R., and Sha, L. 1999. On scheduling tasks in reliable real-time control systems. In *Proceedings of the 20th Real-Time Systems Symposium.*

Chang, C., Chen, C.-C., Chen, Y.-L., and Huang, F.-S. 1997. Real-time scheduling in a programmable radar signal processor. In *Proceedings of the 4th International Workshop on Real-Time Computing Systems and Applications (RTCSA'97).*

Deng, Z., Liu, J. W. S., and Sun, J. 1997. A scheme for scheduling hard real-time applications in open system environment. In *Proceedings of 9th Euromicro Workshop on Real-time Systems*, pp. 191–199.

Ghazalie, T. M., and Baker, T. P. 1995. Aperiodic servers in deadline scheduling environment. *Real-Time Systems Journal* 9(1): 31–68.

Huizing, A. G., and Bloemen, A. A. F. 1996. An efficient scheduling algorithm for a multifunction radar. In *IEEE International Radar Conference*, pp. 359–364.

II, I. 1992. ISO CD11172-2: Coding of moving pictures and associated audio.

Kao, B., and Garcia-Molina, H. 1993. Deadline assignment in distributed soft real-time systems. In *Proceedings of 13th IEEE International Conference on Distributed Computing Systems*, pp. 428–437.

Kao, B., and Garcia-Molina, H. 1994. Subtask deadline assignment for complex distributed soft real-time tasks. In *Proceedings of 14th IEEE International Conference on Distributed Computing Systems*, pp. 172–181.

Kuo, T.-W., Chao, Y.-S., Kuo, C.-F., and Chang, C. 2002. Real-time dwell scheduling of component-oriented phased array radars. In *IEEE 2002 Radar Conference.*

Kuo, T.-W., Kuo, C.-F., and Chang, C. 2000. Real-time digital signal processing of component-oriented phased array radars. In *Proceedings of the 21st Real-Time Systems Symposium.*

Lee, C., Lehoczky, J., Rajkumar, R., and Siewiorek, D. 1998. On quality of service optimization with discrete QoS options. In *Proceedings of the IEEE Real-time Technology and Applications Symposium.*

Lee, C., Lehoczky, J., Siewiorek, D., Rajkumar, R., and Hansen, J. 1999. A scalable solution to the multi-resource QoS problem. In *Proceedings of the 20th Real-Time Systems Symposium.*

Liu, C. L., and Layland, J. W. 1973. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM* 20(1): 46–61.

Liu, J. W. S. 2000. *Real-Time Systems*. Prentice Hall, p. 219.

Peressini, A. L., Sullivan, R. E., and Uhl, J. J. J. 1980. *Convex Programming and the Karish-Kuhn-Tucker Conditions*. Springer-Verlag.

Rajkumar, R., Lee, C., Lehoczky, J., and Siewiorek, D. 1997. A resource allocation model for QoS management. In *Proceedings of the 18th Real-Time Systems Symposium.*

Rajkumar, R., Lee, C., Lehoczky, J., and Siewiorek, D. 1998. Practical solutions for QoS-based resource allocation problems. In *Proceedings of the 19th Real-Time Systems Symposium.*

Seto, D., Lehoczky, J. P., and Sha, L. 1998. Task period selection and schedulability in real-time systems. In *Proceedings of the 19th Real-Time Systems Symposium.*

Seto, D., Lehoczky, J. P., Sha, L., and Shin, K. G. 1996. On task schedulability in real-time control systems. In *Proceedings of the 17th Real-Time Systems Symposium.*

Sha, L., Rajkumar, R., and Sathaye, S. S. 1994. Generalized rate-monotonic scheduling theory: a framework for developing real-time systems. *Proceedings of the IEEE* 82(1), 68–82.

Spuri, M., and Buttazo, G. 1996. Scheduling aperiodic tasks in dynamic priority systems. *Real-Time Systems Journal* 10(2): 179–210.

Strosnider, J. K., Lehoczky, J. P., and Sha, L. 1995. The deferrable server algorithm for enhanced aperiodic responsiveness in hard real-time environments. *IEEE Transactions on Computers* 44(1): 73–91.

Sun, J. 1997. Fixed priority scheduling of end-to-end periodic tasks. Ph.D. thesis, Department of Computer Science, University of Illinois at Urbana-Champaign.

Thomas, C. B. Efficient Fine Granular Scalable Video Coding. URL:citeseer.nj.nec.com/442763.html.

Chang Gun Lee received the B.S., M.S., and Ph.D. degrees in computer engineering from Seoul National University, Korea, in 1991, 1993, and 1998, respectively. He is currently an assistant professor in the Department of Electrical and Computer Engineering, Ohio State University, Columbus. Previously, he was a research scientist in the Department of Computer Science, University of Illinois at Urbana-Champaign from March 2000 to July 2002 and a research engineer in the Advanced Telecomm. Research Lab., LG Information and Communications, Ltd. from March 1998 to Feburary 2000. His current research interests include real-time systems, complex embedded systems, QoS management, and mobile communications. He is a member of the IEEE Computer Society.

Chi-Sheng Shih has been an assistant professor at the Department of Computer Science and Information Engineering at National Taiwan University since February 2004. He received the B.S. in Engineering Science and M.S. in Computer Science from National Cheng Kung University in 1993 and 1995, respectively. After his two-years military service, he continued his study at the Department of Computer Science at University of Illinois at Urbana-

Champaign in 1998. In 2003, he received his Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign. His main research interests are real-time systems and database systems. Specifically, his main research interests focus on object-relational database query optimization, real-time operating systems, real-time scheduling theory, embedded software, and software/hardware co-design for system-on-chips.

Lui Sha obtained his Ph.D. from Carnegie Mellon University in 1985. Currently, he is a professor of Computer Science in the University of Illinois at Urbana-Champaign. He was elected to be an IEEE Fellow in 1998 and awarded by IEEE Technical Committee on Real-Time Systems for ‘‘Outstanding technical contributions and leadership’’ in 2001.