

Adaptive QoS Optimizations with applications to Radar Tracking ^{*}

Sourav Ghosh¹, Jeffery Hansen², and Raguathan (Raj) Rajkumar³ and John Lehoczky⁴

¹ Carnegie Mellon University, Department of Electrical and Computer Engineering
sourav@cs.cmu.edu

² Carnegie Mellon University, Institute for Complex Engineered Systems hansen@cmu.edu

³ Carnegie Mellon University, Department of Electrical and Computer Engineering
raj@ece.cmu.edu

⁴ Carnegie Mellon University, Department of Statistics jpl@stat.cmu.edu

Abstract. In many applications such as sensor networks, mobile ad hoc networking and autonomous systems, the relationship between level of service and resource requirements is not fixed. Environmental factors outside the direct control of the system affect this relationship and may also affect the perceived utility of a given level of service. Radar tracking provides a good example. In radar systems, a fixed amount of radar bandwidth and computing resources must be apportioned among multiple tasks, each of which corresponds to a target. In addition, environmental factors such as noise, heating constraints of the radar and the speed, distance and maneuverability of tracked targets dynamically affect the mapping between the level of service and resource requirements as well as the mapping between the level of service and the user-perceived utility. To be able to handle these tasks, a QoS manager must be adaptive, reacting to dynamic changes in the environment, adjusting the level of service and reallocating resources efficiently. In this paper, we present a dynamic QoS optimization scheme for a radar tracking application based on Q-RAM [1, 2]. Our scheme is able to deal with a large number of operating points in real-time with very acceptable losses in total utility accrued. This result is made possible by an efficient heuristic to compute the concave majorant of a multi-variate function, and an off-line storage-efficient discretization of the static aspects of the problem space. These two contributions will also be useful in many dynamic QoS-driven applications beyond radar tracking.

1 Introduction

Traditional QoS optimization algorithms assume that a collection of tasks compete for resources, with each task receiving some benefit from those resources. The goal is to allocate the resources to tasks in such a way as to optimize the total benefit received by all the tasks. This benefit is often called “utility” [3]. A larger QoS for a task generally

^{*} This work was supported by a DARPA Multidisciplinary University Research Initiative (MURI) program administered by the Office of Naval Research under Grant N00014-01-1-0576 and by DARPA under contract number F33615-00-C-1729.

requires a larger amount of resources and results in larger utility. Furthermore, the QoS-utility curve usually saturates at a high QoS level with further increases in QoS yielding smaller increases in utility [1]. Generally, it is assumed that for a given invocation of a task, the utility received for a particular amount of resources is constant. While this is sufficient for many applications, this may not be the case for other applications.

An example of an application where the mapping between resources and utility may change dynamically is radar tracking. Radar tracking is challenging in that it deals with a large number of dynamic tasks, multiple resources, several practical constraints and real-time operation. Due to the high complexity of high-quality tracking in real-time, radar systems have traditionally used rather static schemes inter-mixed with operator (somewhat error-prone) intuition of which combinations are likely to work. It is important to develop real-time resource allocation algorithms that will lead near-optimal resource allocations over a wide range of conditions.

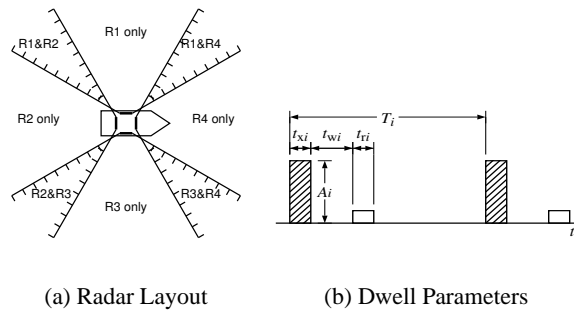


Fig. 1. Radar Model

In this paper, we model a radar system as being composed of radar transceivers, a power source and an array of processors for the signal processing and tracking algorithms. A radar system must also satisfy physical constraints, for example, the dynamics of an object being tracked (such as its speed, acceleration and distance from the radar), and environmental factors such as noise influence the tracking precision. Moreover, the radar system itself imposes certain constraints. For example, not only must the utilization of the resources be below an appropriate utilization bound, other constraints such as the heat dissipation limits of a radar transmitter must also be satisfied. The radar system resource management problems are as follows:

- **Selection of appropriate settings or operating points for tracking tasks:** In this paper, we address this issue. We use the QoS-based Resource Allocation Model (Q-RAM)[1, 4] as the building block of our resource management framework.
- **Ensuring schedulability of the tasks:** We address this in [5].

Many recent studies have focused on phased-array systems, especially radar scheduling. For example, Goddard et al [6] used data flow model for real-time scheduling of radar tracking algorithms. On the other hand, Shih et al[7, 8] addressed the scheduling

issues of radar front-end. In addition, similar to our optimization methodology, they also introduced the idea of “service classes” designed off-line to determine the QoS operating points of tasks.

The rest of the paper is organized as follows. Section 2 presents our model of the radar system. Section 3 presents the QoS optimization schemes in the radar system. In Section 4, we evaluate and compare these schemes. In Section 5, we quantize the problem space and also perform part of the computation off-line. In Section 6, we describe the results from the quantization. Finally in Section 7, we summarize our contributions and provide a brief description of our future work.

2 Our Radar System Model

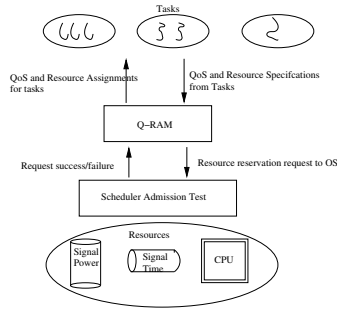


Fig. 2. Q-RAM & Scheduler Admission Control

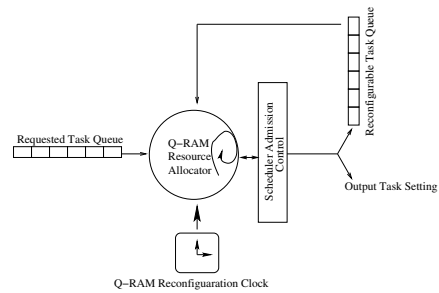


Fig. 3. Dynamic Q-RAM

We assume 4 radar antennas oriented at 90° to each other as shown in Figure 1(a). We also assume that each antenna is capable of tracking targets over a 120° arc. This means that there are regions of the sky that are capable of being tracked by only one radar antenna, as well as regions that can be tracked by two antennas. The antennas are assumed to share a pool of 128 processors (used for tracking and signal processing algorithms) and a common energy source for transmitting and receiving electromagnetic signals.

A single instance of tracking a particular target by sending radar signals and receiving the echo is called a *dwell*, as shown in Figure 1(b). It is characterized in terms of a *transmit power* A_i , a *transmission time* of pulses t_{x_i} ⁵, a *wait time* t_{w_i} and a *receive time* t_{r_i} . Generally, $t_{x_i} = t_{r_i}$, and the wait time is based on the round-trip time of the radar signal (e.g., about 1 *m.s* for a target 100 miles away). Also, while the radar may dissipate some power while receiving, this power is much smaller than the transmit power. For purposes of simplicity, we assume that the receive power is negligible compared to the transmit power. The time between two successive dwells is called the **dwell period** (T_i).

⁵ In practice, the radar transmission time is actually composed of a series of narrow high frequency pulses.

For a particular target, we obtain better tracking information by increasing the transmit time, reducing the dwell period, or increasing the transmission power. The power requirement, in turn, is inversely proportional to the fourth power of distance of the target from the radar as the power of the received signal at the radar is inversely proportional to the fourth power of the distance [9]. Apart from the power output capability of the energy source, there is a limit on the amount of heat dissipation on each radar transmitter.

The tracking information is also dependent on many environmental factors beyond the radar system's control such as the speed, the acceleration, the distance and the type of the target, the presence of noise in the atmosphere and the use of electronic countermeasures by the target. Several tracking algorithms are available to choose from, with each algorithm performing better or worse with respect to computational cycles, noise tolerance, dealing with target maneuverability etc. The main tasks of an antenna are (1) Searching for targets and (2) Tracking the targets. In this paper, we only deal with tracking because search tasks can be considered to be tracking "virtual targets". If real targets are found by a search task, tracking tasks are initiated [5, 6, 7].

In summary, the quality of tracking a particular target depends on environmental factors, target dynamics as well as the usage of resources. In addition, the importance of tracking a particular target depends on the type of the target (e.g., a friend or a foe, a helicopter or a missile), the distance of the target from the radar and the speed of the target towards the radar. We would like higher precision tracking for dangerous targets that are close to the radar, and lower precision but more resource-conserving tracking for less dangerous ones and targets that are far away.

2.1 QoS Model

In this section, we briefly describe our generic QoS model. Let us assume a distributed system with a set of shared resources and a set of n independent tasks τ_1, \dots, τ_n . Each task is assumed to have a set of parameters that can be changed to configure its QoS and resource demands. We will call these *operational dimensions*. Some operational dimensions may also be *QoS dimensions*. A QoS dimension is a single aspect of task quality that is of direct relevance to the user; for example, the frame rate of a video-conference application. Other operational dimensions that may be transparent to the user with respect to quality, but may affect resource demands. For example, the selection of a video coding algorithm, or the path through the network fall into this category. In addition, some other operational dimensions may affect both quality and resource demand, but may not be of direct relevance to the user. An example is the dwell period of a radar tracking task. While a shorter dwell period will increase both tracking quality and resource demand, the dwell period itself is not directly of interest to the user. In such applications, the QoS dimensions may not be directly aligned with the operational dimensions. In fact, the task quality may even depend on factors in the environment in addition to the operational setting of a task. For example, in a radar tracking task the tracking quality can depend not only on the configuration of the task, but on factors such as the distance to the target as well as its speed and type.

Because of the possible disparity between the QoS objectives of the user and the configuration options of the task we separately define an *operational space*, a *quality*

space and an *environment space*. For a task τ_i we define the operational space to be,

$$\Phi_i = \Phi_{i1} \times \cdots \times \Phi_{iN_i^\Phi}, \quad (1)$$

where Φ_{ij} is the j^{th} operational dimension, the *quality space* by

$$Q_i = Q_{i1} \times \cdots \times Q_{iN_i^Q}, \quad (2)$$

where Q_{ij} is the j^{th} QoS dimension, and we define the *environment space* by,

$$E_i = E_{i1} \times \cdots \times E_{iN_i^E} \quad (3)$$

where E_{ij} is the j^{th} environment dimension. We also define a shared *resource space* as:

$$R = R_1 \times \cdots \times R_m, \quad (4)$$

where R_k is the k^{th} shared resource (or resource dimension).

For some types of tasks, the operational dimensions and quality dimensions may be equivalent, and there may be no environment dimensions (e.g., a videoconferencing task), but in general we say that there is a *quality function* $f_i : \Phi_i \times E_i \rightarrow Q_i$ mapping each point in the cross product of the operational space and environment space to a point in the quality space.

Rather than concerning ourselves with the semantics of the values that can be taken on by the operational dimensions, we assume each operational dimension can be characterized by an index value between 1 and ϕ_{ij}^{max} . For operational dimensions that affect application quality (e.g., frame-rate) we assume 1 to be the lowest quality with increasing index values representing increasing quality. For other types of operational dimensions (e.g., the video compression algorithm used) the mapping between the index value and its semantics is arbitrary. We call each possible combination of index values $\{\phi_{i1}, \dots, \phi_{iN_i^\Phi}\}$ a *set-point*. The number of set-points for a task τ_i is $\prod_{j=1}^{N_i^\Phi} \phi_{ij}^{\text{max}}$.

For each task, the user may specify a *utility function* $u_i : Q_i \rightarrow \mathfrak{R}$ mapping each point in the quality space to a real number called the *task utility*. Usually, task utility is defined in terms of the weighted sum of a set of dimension-wise utility functions $u_{ij} : Q_{ij} \rightarrow \mathfrak{R}$. The *system utility* is then simply the sum of the individual task utilities. Note that for a given environmental condition e_i , it is straightforward to map a set-point ϕ_i in the operational space to a utility value using the quality function f and the utility function u . Using this mapping, a resource manager can only concern itself with selecting set-points in the operational space and need not directly consider points in the quality space.

In order for a task to operate at a particular set-point ϕ_i , it generally requires resources. We define a function $g_i : \Phi_i \rightarrow R$ specifying the amount of resources required for a task to operate at each set-point. In a system consisting of multiple resources, a task may need more than one resource for a particular set-point. We refer to this resource requirement as a resource vector, whose each element represents the demand on a particular resource. In this case, we need to estimate the price of a particular resource vector of a set-point. Hence we define a scalar quantity r_i^* we call *compound resource* in terms of a compound resource function $h : R \rightarrow \mathfrak{R}$ mapping a resource vector of

a task to a value representing the overall demand on the system for a particular set of resource requirements. The resources can be of multiple *types* such as processors or radar antenna or network links. In general, if there are l distinct types of resources and m_k denotes the number of resources of type k , a typical compound resource function is given by

$$h(R) = \sqrt{\sum_{k=1}^l \left(\sum_{j=1}^{m_k} (R_{k_j} P_{k_j}) \right)^2}. \quad (5)$$

For brevity, we will also write $h(\phi_i)$ to represent the compound resource required of a task τ_i operating at set-point ϕ_i .

An operational dimension Φ_{i_j} of task τ_i is said to be *monotonic* if for all points in the environment space an increase in the operational index value results in non-decreasing utility and increasing resource requirements across all resource dimensions. All other operational dimensions are said to be *non-monotonic*. Monotonic operational dimensions typically include direct QoS-like dimensions such as video frame-rate, as well as indirect QoS-like dimensions such as the dwell period in radar tracking.

Thus, we have explained how each QoS setting of a task is mapped to a set-point consisting of a resource vector, a compound resource and a corresponding utility value. Next, we describe how we optimize the QoS of each task under the constraints of finite resource capacities.

2.2 QoS Optimization

We can now define the basic problem of QoS-based resource allocation as follows. For each task τ_i in the set τ_1, \dots, τ_n , assign a set-point ϕ_i such that the system utility (sum of the utilities of all tasks) is maximized and no resource utilization exceeds its maximum. Formally, we write this as:

$$\begin{aligned} \text{maximize: } & u(\phi_1, \dots, \phi_n) = \sum_{i=1}^n u_i(\phi_i) \text{ (System Utility)} \\ \text{subject to: } & \forall_{1 \leq k \leq m} \sum_{i=1}^n R_{ik} \leq R_k^{max} \\ & \forall_{1 \leq k \leq m, 1 \leq i \leq n} R_{ik} = g_{ik}(\phi_i) \end{aligned}$$

While finding the optimal resource allocation is NP-hard, the Q-RAM algorithm is able to find a near-optimal solution to this problem in $O(nl \log(nl))$ time [10, 2] where n is number of tasks and l is the maximum number of *input* set-points per task ⁶.

The steps of the basic Q-RAM algorithm [4, 2, 3] can be summarized as follows:

1. For each set-point of every task, compute the *compound resource* and utility values. The compound resource is a measure of the impact on the system of a particular resource requirement vector.
2. Compute the concave majorant (the smallest concave function lying on or above all points) of the set-points in the compound resources versus utility space and eliminate all set-points lying below the concave majorant.

⁶ The above complexity ignores the complexity involved during the task profile generation phase, if any.

3. Each task is initialized to the lowest utility set-point.
4. Choose the task with the highest *marginal utility* between its current point and the next point to the right on its concave majorant and make that the new current point for that task. The marginal utility is defined as the increase in utility divided by the increase in the compound resource.
5. Repeat the previous step until all resources have been allocated.

The concave majorant in the context of Q-RAM is the set of set-points defining a curve under which all of the other set-points lie. An example is shown in Figure 4. Set-points not on the concave majorant need not be considered since they represent operating points for which an increase in utility can be achieved with no increase in resource requirements.

The basic algorithm shown here works best when all operational dimensions are monotonic. Extensions to this algorithm for handling non-monotonic dimensions have also been studied [10]. These heuristics have been proved to be very effective at quickly finding resource allocations that are near-optimal for tasks with a reasonable number of set-points [10].

2.3 Q-RAM and Scheduling

Q-RAM is a resource allocation framework. Since straightforward resource constraints (such as total resource usage of any resource must be less than 100%) are used in Q-RAM, a given resource allocation may result in an unschedulable task set. Therefore, the tasks need to depend on the scheduler admission control to be scheduled at a particular QoS level. If the tasks are not schedulable, Q-RAM needs to reduce the utilization bounds of some of the resources in order to produce a schedulable output. The interaction between Q-RAM and scheduler admission control is demonstrated in Figure 2. This is discussed in detail in [5]

Moreover, Q-RAM optimization needs to be performed at regular intervals (known as the reconfiguration rate) as a background process in a dynamic scenario where the task set is not fixed and tasks are continuously arriving and departing from the system [11]. It accepts all newly arrived tasks, performs optimization along with the existing tasks, and produces the resource allocation settings for all of them as illustrated by Figure 3.

2.4 Radar Resource Model

The radar resources-space consists of the following resource dimensions: radar bandwidth, radar power and computing resources such as CPU and network bandwidth.

Radar Bandwidth Each radar can track only a limited number of targets at a specific time. Assuming the receiving time of a dwell pulse is equal to its transmit time t_{x_i} , since the radar is unused during the waiting period, the utilization of a radar is given by $\sum_{i \in S_j} \frac{2t_{x_i}}{T_i}$ where S_j is the set of tracks being processed by radar j . Clearly, the utilization cannot be more than 100%. If we assume fixed-priority non-preemptive

scheduling, the utilization could be much lower than 100% [12, 13]. Assuming we have a schedulable utilization bound U_b , the utilization constraint can be written as:

$$\sum_{i \in S_j} \frac{2t_{x_i}}{T_i} \leq U_b. \quad (6)$$

Power Resource As mentioned earlier, each radar must also satisfy one or more physical constraints such as limits on heat dissipation. There are two constraints on duty-cycles of radar pulses that control its heat dissipation, **short-term** and **long-term**. The latter is more conservative than the former in terms of heat dissipation from the radar transmitter. In this paper, for simplicity of presentation, we will restrict our consideration to (conservative) long-term duty-cycles only. The short term power constraint is more appropriately considered during the schedulability analysis, which is discussed in [5].

Based on the model presented in the beginning of this section and using Figure 1(b), $t_{x_i}A_i$ is the energy consumed by one dwell ignoring the power during receive time and thus $\sum_i \frac{t_{x_i}A_i}{T_i}$ will be the long-term power consumption for a set of targets. This gives us the power constraint:

$$\sum_{i \in T_j} \frac{t_{x_i}A_i}{T_i} \leq \bar{P}_{\max}, \quad (7)$$

where \bar{P}_{\max} is the maximum continuous power dissipation. We can treat each of these constraints, and any other constraints we might wish to consider, as though they were logical resources. For example, each of the four radars has a utilization resource in which U_b is the maximum amount of that resource available, and $\frac{2t_{x_i}}{T_i}$ is the amount consumed by target i . Likewise, each radar also has a power resource in which \bar{P}_{\max} is the total amount of that resource, and $\frac{t_{x_i}A_i}{T_i}$ is the amount consumed by target i .

Computational Resource In addition to the radar resource, each track requires computing resources to process the radar data, and to predict the next location of the target. The computing resources required depend on the tracking algorithm Π_i used, and the period T_i . We assume that the required CPU is of the form C_{Π_i}/T_i where C_{Π_i} is the coefficient representing the computational cost of algorithm Π_i . If we treat the back-end multiprocessor system as a single resource, then we have the CPU constraint:

$$\sum_i C_{\Pi_i}/T_i \leq C_{\max}, \quad (8)$$

where C_{\max} represents the total processing power of the bank of processors. This abstraction is reasonable as long as the amount of processing required by each of the individual tasks is small compared with the amount available on each of the processors.

Resource Configuration of Radar System Based on the discussion above, we have the following resources in our Radar system: radar bandwidth on 4 radar transmitters, radar heat constraints on 4 radar antennas and 1 global energy source and 1 computational

resource from the ship. Therefore, our resource vector consists of 10 components. The global computational processor allocation has been studied in [10], and we treat it as a single resource.

2.5 Tracking QoS Model

There are two principal QoS dimensions in the quality space of the radar tracking problem:

- *Tracking error*: This is the difference between the actual position and the tracked position of the target. Although one cannot know the true tracking error, many tracking algorithms yield estimates and their precision of a particular tracking result. As mentioned in Section 2.4, this tracking precision is dependent on the availability of the physical resources in addition to the computing resources. A smaller tracking error leads to better tracking precision and hence better quality of tracking. Therefore, we assume that the tracking quality q_{track} is inversely related to the tracking error ϵ .

- *Reliability*: This is the probability that there are no hardware/software failures in a specified time interval. Higher reliability of a task is obtained by the replicated use of resources, such as using two radars to track a single target.

In this paper, we focus only on tracking error, and refer the reader to [10] for a study of how the number of replicas for a particular computation is integrated into Q-RAM.

Next, we list the operational and environmental dimensions of the system.

Operational Dimensions In our tracking model, the operational dimensions are dwell period (T_i), dwell time (t_{x_i}), dwell power (A_i), and choice of the tracking algorithm Π_i .

The above parameters can be controlled by the system designer or the optimizer in order to achieve the desired quality of tracking of a target.

Environmental dimensions The environmental dimensions we consider are the *type* of target ξ_i (e.g., airplane, helicopter, missile etc.), the distance of the target from the radar r_i , the velocity vector of the target \mathbf{v}_i , the acceleration vector of the target \mathbf{a}_i , the active noise or the presence of electro-magnetic interference such as counter-measures n_i , and the angular location of the target in the sky.

Considering all the operational and environmental dimensions, we can write a function,

$$\epsilon_i = E(\underbrace{\xi_i, r_i, \mathbf{v}_i, \mathbf{a}_i, n_i}_{\text{environmental}}, \underbrace{T_i, t_{x_i}, A_i, \Pi_i}_{\text{operational}}), \quad (9)$$

that estimates the tracking error ϵ_i as a function of the position of the target along the environmental and operational dimensions. Higher tracking quality yields higher utility. For the purposes of this paper, we assume that the utility of tracking a target for a certain quality q_{track} is given by the following concave exponential function,

$$U(q_{track}) = w(1 - e^{-\beta q_{track}}), \quad (10)$$

where β is a parameter specific to the ranges of the speeds of three different types of targets (airplane, missile or helicopter). It assumes the utility increases with increase

Table 1. Environmental Dimensions

Parameter	Type	Range	Increments
Distance	All	0.1-10 miles	continuous
Acceleration	All	0.001g-5g	continuous
Noise	All	1-8 levels	discrete
speed	1 (helicopter)	60-160 miles/hr	continuous
	2 (fighter-jet)	160-960 miles/hr	continuous
	3 (missile)	800-3200 miles/hr	continuous
Angle	All	0°-360°	continuous

Table 2. Operational Dimensions

Parameter	Range	Increments
Algorithm	Kalman, Least Squares, $\alpha\beta\gamma$	-
Period	120ms-720 ms	60 ms
Dwell time	0.6ms-30ms	0.6ms
Power	$0.1Kd^4 K - p_m K d^4$	$0.1Kd^4$ ^a

^a K = constant term, d = distance

in tracking precision, which ultimately saturates at a very high precision [3]. The parameter w is a *weight factor* that determines the importance of the target. This is also dependent on the type of target. Moreover, it is also assumed to be proportional to the *speed* and is inversely proportional to the *distance* of the target. The objective of our optimization is to allocate resources to each tracking process such that the total utility is maximized. From our stated assumptions on tracking precision, quality and utility, we obtain the expression of utility as a function of the tracking error,

$$U(\epsilon) = w(1 - e^{-\gamma/\epsilon}), \quad (11)$$

where γ is a function of β and the relation between quality and tracking error. The required values of the operational dimensions in order to obtain a particular value of tracking error from (9) can be translated into the usage of resources.

3 QoS Optimization in Radar

In the previous section, we formulated the radar tracking problem in our Q-RAM-based QoS optimization framework. The assumed ranges of the physical parameters are summarized in Tables 1 and 2. The value of K in Table 2 is assumed to be 1 with miles as a unit of distance, while that of p_m corresponds to the value of utility at $0.95w$ for a particular target, keeping all operational dimensions except power at their best levels.

Our present system assumes a fixed snapshot of the sky at a particular *instant* during which the environmental dimensions of the objects are constant. The resource allocation process needs to be repeated at regular intervals in the dynamically changing environment of the real world. Hence, its efficient execution is of critical importance for the method to be of practical use.

The resource allocation process consists of two main steps:

1. **Task Profile Generation:** Generate the set-points for each target with values assigned to its environmental parameters and picking the ranges of the operational parameters.
2. **Basic Optimization:** Perform Q-RAM optimization as described in Section 2.2.

3.1 Efficiency and Algorithm Complexity

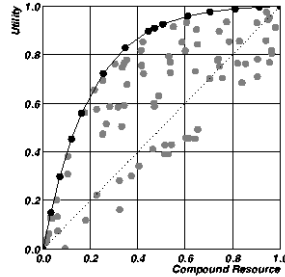


Fig. 4. Slope-Based Traversal of Concave Majorant

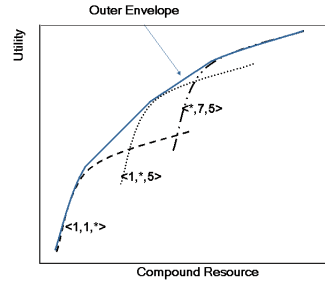


Fig. 5. Incremental Traversal

As mentioned earlier, the computational complexity of the **basic Q-RAM algorithm** is $O(nl(\log(nl)))$, where l is the number of set-points in a task and n is the number of tasks. It has two drawbacks. First, this excludes the task profile generation phase which could be prohibitively expensive in practice for a large l . Second, one of the primary steps of the basic algorithm involves “concave majorant” determination that has the complexity of $O(l \log(l))$ for each task. Consequently, a large l can make this computation very expensive as well. In our radar model, a task’s average number of initial set-points is around 16500. For 1000 targets, we need to generate and input as many as 16.5 million set-points to the optimizer, a very large number given the radar system’s need to make resource allocation decisions in real-time. In addition, in the basic Q-RAM scheme, penalty vectors are computed based on all inputted set-points of all tasks (task profile). The compound-resource function is then applied to all these set-points, and only then is the concave majorant of a task profile computed.

Hence, our first improvement to this basic scheme is to filter set-points during the task profile generation itself so that it relieves the load on the penalty vector computation and concave majorant operation during the subsequent Q-RAM optimization process. Such a scheme is discussed in detail next.

3.2 Concave Majorant at Task Profile Generation

Every tracking task needs three resources namely computation time, power and radar bandwidth. We propose that another level of composite resource cost of each set-point can be defined using the formula: $h^* = \{(r_R/r_R^{\max})^2 + (r_P/r_P^{\max})^2 + (r_C/r_C^{\max})^2\}^{\frac{1}{2}}$

during task profile generation process. The parameters r_R , r_P and r_C are the total amount of radar bandwidth, power and computational resource required by each task for its a given set-point irrespective of the particular radar and CPU to which it could be mapped. r_R^{\max} , r_P^{\max} and r_C^{\max} are the total resource capacities for radar, power and CPU. Unlike the compound resource evaluation during the Q-RAM optimization process, this pre-processed compound resource does not require us to evaluate the penalty vector-based demand of resources which considers all set-points of all the tasks together. Next, we immediately perform the concave majorant operation on these set-points during the task profile generation. We refer to this scheme as *Concave Majorant Only* in subsequent discussions, and it leads to significant load reduction in the concave majorant operation of the optimization step that requires us to evaluate the penalty vector of resources.

3.3 Concave-Majorant Approximation

The complexity of computing the concave majorant can be further improved. We present several pre-processing heuristics that can be performed prior to actually computing the concave majorant. For the sake of illustration, we will first assume that all tasks have only monotonic operational dimensions. Later, we shall consider the case of having some operational dimensions that are non-monotonic.

Slope-based Traversal (ST) Let the *minimum* set-point for a task τ_i for which all operational dimensions are monotonic be defined as $o_i^{\min} = \{1, \dots, 1\}$, and let the *maximum* set-point be defined as $o_i^{\max} = \{v_{i1}^{\max}, \dots, v_{iN^{\phi}}^{\max}\}$. Clearly, all of the set-points in the utility/compound resource space that lie below a “terminating” line from $(u(v_i^{\min}), h(v_i^{\min}))$ to $(u(v_i^{\max}), h(v_i^{\max}))$ as shown in Figure 4 cannot be on the concave majorant. These points can be eliminated immediately without being passed on to the concave majorant step. We call this heuristic “slope-based traversal” (ST). While this heuristic can reduce the time to compute the concave majorant by a constant factor, it must still scan all of the set-points to determine if they are above or below the terminating line.

Fast Set-point Traversals We now consider a set of fast traversal heuristics that do not require computations for all of the set-points. We (temporarily) assume that all operational dimensions are monotonic. A key observation we made is that when the actual concave majorant is generated using all of the set-points for typical tasks, the concave majorant tends to consists of runs of set-points which vary in only *one* dimension at a time with occasional jumps between runs of points. This insight suggests that we can use local search techniques to follow the set-points up the concave majorant. We also know that o_i^{\min} will always be the first point on the concave majorant and o_i^{\max} will always be the last. The different methods presented here differ primarily in the method used to perform the local search. As an example, consider a task with three operational dimensions. If we consider the subset of the set-points $\langle 1, 1, * \rangle$ consisting of all the set-points for which dimensions 1 and 2 have index value 1, these points will tend to

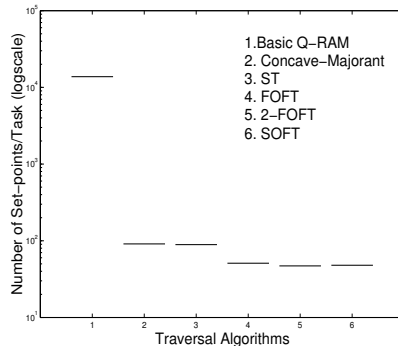


Fig. 6. Average Number of Set-points

form a line as shown in Figure 5. The concave majorant will tend to follow such a line until it switches to some other line, in this case $\langle 1, *, 5 \rangle$ followed by $\langle *, 7, 5 \rangle$.

While the fast traversal heuristics presented in this section are not guaranteed to find the exact concave majorant, in Section 4 we will show that these heuristics produce very good approximations to the concave majorant in our radar QoS optimization and more importantly that the drop in system utility from using the approximations is negligible.

First-Order Fast Traversal (FOFT) In *first-order fast traversal* (FOFT), we keep a current point ϕ_i for each task τ_i which we initialize to ϕ_i^{\min} . We then compute the marginal utility for all the set-points adjacent to ϕ_i . A set-point is adjacent if all of its index values except for one are identical, and the one that differs varies by only one (i.e., they have a Manhattan distance of one). We, in fact, need only consider positive index value changes. We then choose the point that has the highest marginal utility, add it to the concave majorant and make that point the current point. Formally, if ϕ_i is the current point we choose the next current point $\phi_i' = \phi_i + \xi_j$ where j maximizes the marginal utility:

$$\frac{u(\phi_i + \xi_j) - u(\phi_i)}{h(\phi_i + \xi_j) - h(\phi_i)} \quad (12)$$

and where ξ_j is a vector that is zero everywhere except in dimension j where it is equal to 1. We repeat this step until we reach ϕ_i^{\max} . After we have generated this set of points, the resulting curve may not be a concave majorant. Hence, we perform a final concave majorant operation (albeit on a much smaller number of points than before).

The number of set-points generated before the final concave majorant step will be the Manhattan distance between ϕ_i^{\min} and ϕ_i^{\max} which is: $\sum_{j=1}^{N_i^\Phi} (\phi_{ij}^{\max} - \phi_{ij}^{\min})$. Ignoring boundary conditions, at each point we only consider N_i^Φ possible next set-points. This means that when we have d dimensions and k index levels per dimensions then the complexity of this algorithm is $O(kd^2)$.

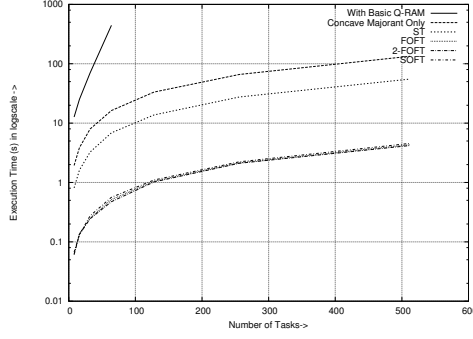


Fig. 7. Q-RAM Execution Time

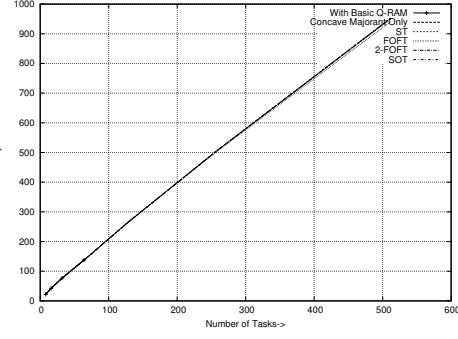


Fig. 8. Q-RAM Utility Variation

Higher-Order Fast Traversal Methods We can generalize the FOFT algorithm to an m -step p -order Fast Traversal algorithm as follows. Just as in the FOFT heuristic, initialize the current point ϕ_i to ϕ_i^{\min} . Then choose the next point $\phi'_i = o_i + z$ where $z \in G_m^p$ such that the marginal utility is maximized and G_m^p is defined as:

$$G_m^1 = \bigcup_{1 \leq j \leq N^O, 1 \leq k \leq m} \{k\xi_j\} \quad (13)$$

$$G_m^p = \{x + y : x \in G_m^1, y \in G_m^{p-1}, x \bullet y \equiv 0\} \cup G_m^{p-1} \quad (14)$$

That is, we look at all the set-points that can be reached from the current set-point by increasing up to p dimensions up to m steps. The FOFT algorithm described above then corresponds to G_1^1 . As with FOFT, we need to perform a final concave majorant operation on the raw points generated by this heuristic. In this paper, in addition to FOFT, we will only consider 2-Step First-Order Fast Traversal (G_2^1) which we will call 2-FOFT, and 1-step, Second-Order Fast Traversal (G_1^2) which we will call SOT.

3.4 Non-Monotonic Dimensions

The fast traversal algorithms described above assume that all of the operational dimensions are monotonic. Unlike monotonic dimensions, non-monotonic operational dimensions generally do not have a structure that can be easily exploited. In our radar problem, tracking period, dwell time and tracking power are the examples of monotonic operational dimensions. However, the choice of tracking algorithm is an example of a non-monotonic operational dimension.

Suppose that some of the operational dimensions are non-monotonic. Then, for every combination of the index values of the non-monotonic dimensions, we simply apply the fast traversal algorithms to the subset that is monotonic. We then form the union of all these results and apply a concave majorant. In the worst case that a task has only non-monotonic dimensions, this simply reduces to a full concave majorant operation. For example, in our radar model, we apply fast-traversal methods for each of the three tracking algorithms separately, and then merge all three results and perform a concave majorant operation.

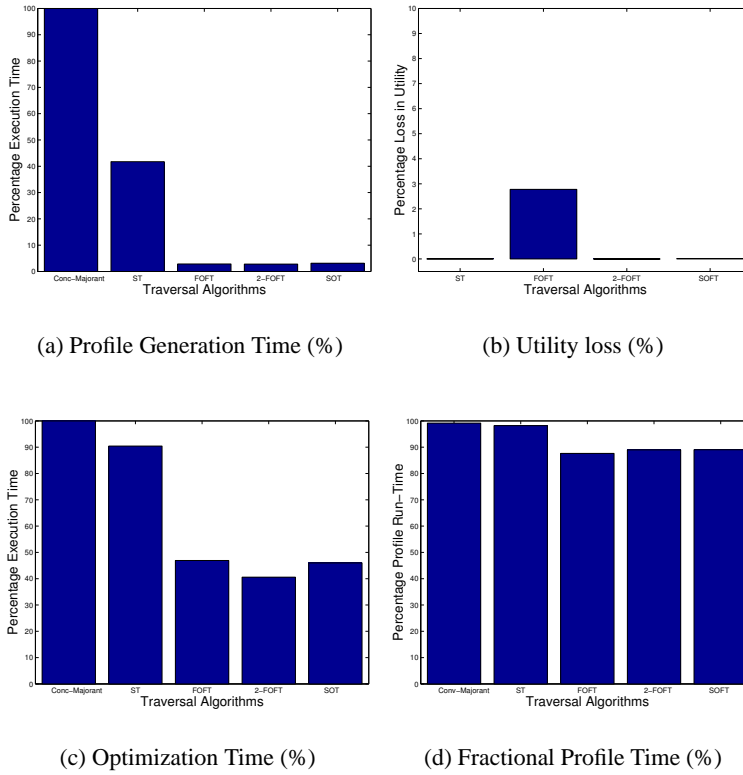


Fig. 9. Comparisons of Traversal Algorithms

4 Experimental Results

Using the settings presented in Tables 1 and 2, we perform the processes of task profile generation and QoS optimization. We vary the number of targets, compute the utility accrued and determine the execution time of the two processes. This is averaged over 50 iterations involving independent sets of targets. The experiments are performed on a 2.0GHz Pentium IV with 256MB of memory.

The number of tasks is varied geometrically between 8 and 512. The number of tasks for basic Q-RAM stopped before 128 when the optimization became intractable. This is due to the fact that approximately 2 million set-points are generated for 128 tasks. Each set-point requires approximately 100 bytes of space. This means that we need more than 200MB just for set-point storage.

The bar-graph in Figure 6 shows the average number of set-points per task after the task profile generation step under different techniques in log-scale. We observe a drop from 13794 to 91 (a drop of 99%) when we apply the concave majorant operation. We

also observe that 2-FOFT reduces the number of points further to 47, which is a 48% drop compared to the full concave majorant scheme.

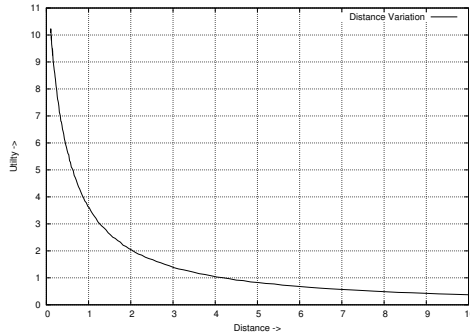


Fig. 10. Utility Variation with Distance

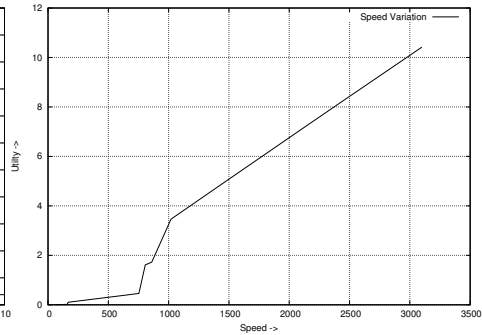


Fig. 11. Utility Variation with Speed

Figure 7 shows the plot of the overall Q-RAM execution time which is the sum of the times for the task profile generation and the subsequent optimization. We notice a huge drop in execution time for the traversal algorithms. For example, for 64 tasks, the run-time drops from 7.4 minutes under basic Q-RAM to 16.28 seconds when we perform the concave majorant. It is further reduced to a minimum of 0.48s under FOFT.

Next, we inspect the quality of the optimization results. Figure 8 shows the variation of utility versus the number of tasks. All the algorithms yield very similar utility values. The worst performer is FOFT, which is smaller by only 1.17% for 512 tasks.

Next, we exhaustively compare these traversal algorithms against the simple Concave Majorant scheme for 1024 tasks. The results shown in Figure 9 are averaged over 100 independent sets of tasks. As shown in Figure 9(b), we obtained the maximum utility loss under FOFT, which is still only 2.7% less than the simple concave majorant scheme. However, all incremental traversal algorithms provide *large* (97%) reductions in the profile generation time as shown in Figure 9(a), as well as *considerable* reductions in the optimization time (close to 50%) as shown in Figure 9(c).

The total run-time of the algorithm is the sum of the profile generation and optimization times. The percentage contribution of the profile generation time in the execution time of the whole process under all traversal techniques is plotted in Figure 9(d). The task profile generation always contributes more than 87% of the overall time which is quite expensive. Unfortunately, even the fastest scheme takes more than 7s for 1024 tasks. Such a large number is unlikely to be acceptable in modern radar systems. We therefore, focus on enhancing the performance of our approach even more.

5 Discrete Profile Generation

We found in the previous section that even efficient profile generation at run-time takes too long. An alternative is to generate the profiles off-line, but the profile space has

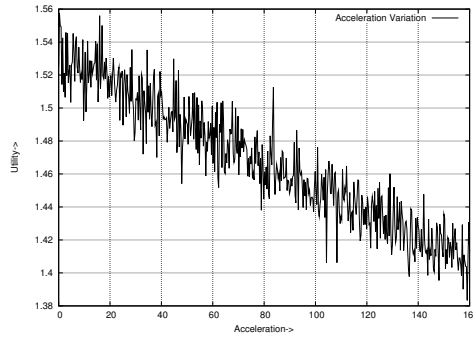


Fig. 12. Utility Variation with acceleration

multiple dimensions with wide ranges. Therefore, off-line computation and storage of profiles requires exponentially large space and becomes unwieldy. The approach we adopt is to quantize each continuous environmental dimension into a collection of discrete regions. Then, we only need to generate a number of discrete task profiles *off-line* for a variety of environmental conditions. At run-time, we only need to map each task into one of the discrete profiles. Any quantization carried out must be such that (a) the storage needs of the discrete profiles are practical, and (b) there is no significant drop in the quality of the tracks (as measured by the total system utility). We study these metrics next. The quantization along any dimension can employ an arithmetic, geometric or other progression. We first perform experiments varying one environmental dimension at a time keeping the others constant and plot the utility values based on our assumed equations. Figures 10, 11 and 12 show the variation of utility under the variations of acceleration, speed and distance respectively.

As observed from the figures, the utility variation can be reasonably approximated by linear regression with respect to speed and acceleration. The plot for speed has approximately *three* linear steps due to the use of *three* different types of targets. This means the arithmetic progression is appropriate for discrete values of speed and acceleration. On the other hand, the utility variation is hyperbolic relative to the distance of the target. Therefore, a geometric progression represents the best scheme for quantizing distance.

For each combination of the quantized environmental dimensions, a profile is generated off-line. Then, during the optimization process, each task is mapped to one of the discrete profiles based on target characteristics. We always round up to conservative estimates on the environmental parameters in order to determine the suitable quantization level for each task. In addition, the weight factor of the profile obtained off-line is adjusted based on the real values of the speed and the distance of the target.

6 Experimentation Results with Discrete Profiles

The aim of these experiments is to determine the loss in the utility relative to the resolutions of the environmental dimensions. Higher resolutions result in utility values closer

to the optimal, but require more storage for off-line profiling. Also, one dimension may be more dominant in influencing the utility value than others, hence we need higher resolution for this dimension. Thus, we have a trade-off between the loss in utility and storage space.

Figure 13 shows the percentage loss in utility from using discrete profiles for different resolutions of acceleration. Each result was taken for 1024 independent tasks over 100 iterations. The resolution of acceleration is varied from 16 to 1024 m/sec² by powers of 2 keeping speed and distance continuous. The plot is a concave curve saturating close to the continuous optimization at higher resolution. Next, we vary the resolution of the distance from 16 to 256 points keeping acceleration and speed continuous. Figure 14 shows the comparative utility variations. The same experiment is repeated by varying the resolution of speed keeping the other two dimensions continuous. Figure 15 shows the result. All the results show concave curves approaching a zero utility loss relative to that obtained under all continuous environmental dimensions. But the variation across the acceleration dimension is much more significant than the other two, and can be up to 25%.

Next, we plot the same results from the above three experiments as the loss in utility at 1024 tasks against the amount of space required for discrete off-line profiling. For each curve, we keep two dimensions continuous (or at their maximum possible resolution) and keep increasing the resolution of one dimension. The amount of storage space required for off-line profiling is proportional to the resolution of a particular dimension. For the speed dimension, it is also proportional to the number of types of targets (3 in our case) since the speed of each type of target is quantized independently. From this, we can obtain the storage requirements and the corresponding utility loss for each setting of the environmental parameters. For example, for a resolution of 16 for distance, the loss is 2.47%, the other factors being continuous. Similarly it is 2.89% for speed at a resolution of 2 and 2.59% for acceleration at a resolution 256. Therefore, for a setting of (16, 2, 256) for distance, speed and acceleration respectively, we incur a utility loss of $1 - \left(\frac{100-2.47}{100}\right)\left(\frac{100-2.89}{100}\right)\left(\frac{100-2.59}{100}\right) = 7.74\%$. If each set-point requires 100 bytes,

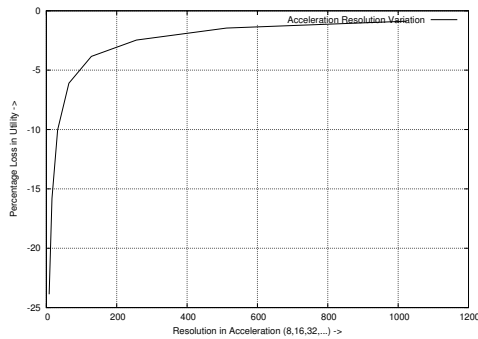


Fig. 13. Utility Loss with Quantized Acceleration

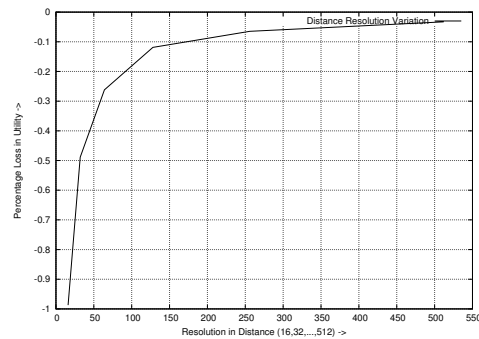


Fig. 14. Utility Loss with Quantized Distance

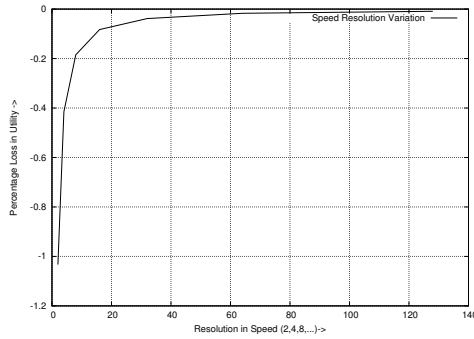


Fig. 15. Utility Loss with Quantized Speed

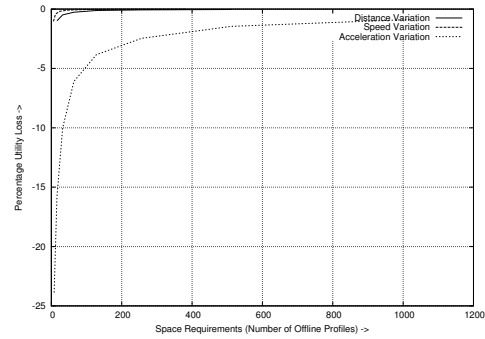


Fig. 16. Utility Loss with Storage Requirements

the total space requirement is $100 \text{ bytes} \times 16 \times (2 \times 3) \times 256 \simeq 2.34 \text{ MB}$, which is certainly very acceptable with today’s memory technology.

As seen in Figure 16, we observe that the *acceleration dimension* is dominant in deciding the quality of optimization, the results from the fact that other two dimensions saturate very quickly. The explanation for the above behavior can be explained from the *weight factor*, which is always computed (see Appendix B) based on the *exact* values of the speed and the distance of the individual targets independent of their quantization. This can be done without incurring additional complexity, and it dramatically minimizes the effect of quantization on speed and distance. With the quantization in place, the on-line profile generation time is reduced to the order of microseconds, and only requires the reading of discrete profiles. Overall, the run-time of the algorithm is mainly contributed by the Q-RAM optimization step. With profile generation now becoming essentially negligible, the total run-time with discrete profiles can be determined from the data presented in Figure 7 and the fraction of time not spent in profile generation presented in Figure 9(d). For example, Figure 7 indicates that with 512 tasks, the total execution time with on-line profile generation is about 5 seconds. Of these 5 seconds, Figure 9(d) indicates that 87% of the time is spent on on-line profile generation. With off-line profile generation taking only microseconds, the optimization time for 512 tasks now takes just 13% of 5 seconds, i.e. 650 ms. Even in the unlikely event that the number of tasks increases to 1024, the total run-time is reduced from 7s for on-line profile generation to only 1s for off-line discrete profile generation with a negligible loss in utility. Hence, utility maximization can be carried out in real-time for 1024 tasks or more⁷.

7 Conclusions and Future Work

We developed a QoS optimization scheme for a model radar system for allocating resources to various tracking tasks. It incorporates the physical and environmental factors that influence the QoS of the tasks. In order to perform QoS optimization dynamically

⁷ Normally, a radar deals with atmost 150 to 200 tracking tasks

in real-time, the profiles of the tasks must be dynamically generated but this leads to unacceptable execution times. We proposed two approaches to solve this problem. First, we showed how only the “relevant” set-points of the tasks are generated using traversal techniques that significantly reduce the complexity of the optimization. A Two-step First-order Fast Traversal scheme (2-FOFT) proves to be the best in reducing computational time significantly with negligible loss in system utility. Next, we showed that the profiles can be generated off-line based on quantization of the environmental dimensions, with very acceptable storage requirements. Only a limited number of profiles need to be generated with quantized values of the environmental dimensions. With such off-line discrete profile generation, the total optimization time takes only 650 *ms* for 512 tasks with minimal loss in accrued utility. Hence, resource allocation with utility maximization can be carried out in real-time.

In this paper, we have dealt with the resource allocation to tracking tasks. Our future work includes the modeling of a large distributed radar system involving multiple ships. Finally, these techniques need to be applied to other dynamic systems such as sensor networks and autonomous systems such as autonomous robotic teams.

References

- [1] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek, “A resource allocation model for qos management,” in *IEEE Real-Time Systems Symposium*, December 1997.
- [2] C. Lee, J. Lehoczky, D. Siewiorek, R. Rajkumar, and J. Hansen, “A scalable solution to the multi-resource qos problem,” in *Proceedings of the IEEE Real-Time Systems Symposium*, December 1999.
- [3] Chen Lee, *On Quality of Service Management*, Ph.D. thesis, Carnegie Mellon University, Aug. 1999.
- [4] C. Lee, J. Lehoczky, R. Rajkumar, and D. Siewiorek, “On quality of service optimization with discrete qos options,” in *Proceedings of the IEEE Real-Time Technology and Applications Symposium*. June 1998, IEEE.
- [5] Sourav Ghosh, Raj Rajkumar, Jeffery Hansen, and John Lehoczky, “Integrated resource management and scheduling with multi-resource constraints,” Tech. Rep. 18-2-04, Institute for Complex Engineering Systems, Carnegie Mellon University, 2004.
- [6] S. Goddard and K. Jeffay, “Analyzing the real-time properties of a dataflow execution paradigm using a synthetic aperture radar application,” in *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium*, June 1997.
- [7] C. Shih, S. Gopalakrishnan, P. Ganti, M. Caccamo, and L. Sha, “Template-based real-time dwell scheduling with energy constraint,” in *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium*, May 2003.
- [8] C. Shih, S. Gopalakrishnan, P. Ganti, M. Caccamo, and L. Sha, “Scheduling real-time dwells using tasks with synthetic periods,” in *Proceedings of the IEEE Real-Time Systems Symposium*, December 2003.
- [9] Michael O. Kolawole, *Radar Systems, Peak Detection and Tracking*, Newnes Press, 2002.
- [10] Sourav Ghosh, Raguathan (Raj) Rajkumar, Jeffery Hansen, and John Lehoczky, “Scalable resource allocation for multi-processor qos optimization,” in *23rd IEEE International Conference on Distributed Computing Systems (ICDCS 2003)*, May 2003.
- [11] Jeffery P. Hansen, John Lehoczky, and Raguathan Rajkumar, “Optimization of quality of service in dynamic systems,” in *Proceedings of the 9th International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS)*, April 2001.

- [12] D. Chen, A.K. Mok, and T.W. Kuo, "Utilization bound revisited," *IEEE Transaction on Computers*, pp. 351–361, 2003.
- [13] J.W. Layland C.L. Liu, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal on ACM*, vol. 2, no. 1, pp. 46–61, 1973.

Table 3. Constants in Error Equation

Name	Type	Values
K_s	Speed constant	1.25×10^{-6}
K_d	Distance constant	2×10^{-6}
K_a	Acceleration constant	Kalman : 5×10^{-4} , Least Squares : 5×10^{-5} , $\alpha\beta\gamma$: 1×10^6
K_n	Noise constant	Kalman : 5×10^{-4} , Least Squares : 5×10^{-5} , $\alpha\beta\gamma$: 1×10^{-6}
K_p	Power constant	7×10^{-2}
K_c	Dwell constant	3×10^{-2}
K_1, K_2, K_3	Weight factor constants	5, 5, 5

A Tracking Error Computation

We model the tracking error as a function of environmental parameters as shown in (9). Next, we make the following assumptions:

- The error increases with the increase in speed (v_i), distance (r_i) and acceleration (a_i).
- Higher signal-to-noise ratio at the receiver reduces error. In addition, the received signal power is directly proportional to the transmitted signal power and is inversely proportional to the 4th power of the distance.
- Longer dwell time (duration of transmission) reduces error.
- The error increases with the increase in dwell period. The error due to acceleration also increases with the increase in dwell period.
- The tracking algorithm like *Kalman* provides best precision under noisy environment. *Least Squares* comes second and $\alpha\beta\gamma$ finishes last. However, the targets with high maneuverability can be best tracked by $\alpha\beta\gamma$ filter, followed by *Least Squares* and *Kalman*.

Based on the above, the assumed expression of error is given in (15). The values of the constants are presented in Table 3.

$$\epsilon_i = T_i \left\{ K_s v_i + K_d r_i + K_a a_i T_i + K_n n_i \left(\frac{1 + K_p r_i^4}{A_i} \right) + \frac{1 + K_c r_i}{t_{x_i}} \right\}; \quad (15)$$

B Weight Factor

The *weight factor* in the expression of utility function in (11) determines the importance of the target. For a speed v_i and a distance r_i , our weight factor is expressed as $w_i = \left(\frac{\xi_i + K_1}{K_2} \right) \left(\frac{v_i}{r_i + K_r} \right)$. The values of K_1, K_2, K_3 are shown in Table 3. ξ_i is an integer corresponding to one of the *three* target types assumed for this model and therefore has the range between [1, 3].