DYNAMIC RESOURCE ALLOCATION IN DISCRETE EVENT SYSTEMS [1]

Christos G. Panayiotou
Department of Electrical and Computer Engineering
University of Massachusetts, Amherst, MA 01003
panayiot@ecs.umass.edu
and
Christos G. Cassandras
Department of Manufacturing Engineering
Boston University, Boston, MA 02215
cgc@enga.bu.edu

Abstract

In this paper we develop a controller for dynamic resource allocation in Discrete Event Systems (DES) operating in a stochastic environment.. The controller's objective is to allocate a finite number of discrete resources to a set of users so as to achieve optimal system performance. The derived controller uses *concurrent estimation*, a sample path constructability technique for DES, to obtain estimates of the system's performance under a set of hypothetical parameter settings using only information observed from the real system. Subsequently, these estimates are used by an on-line algorithm which reallocates the resources among the various users to achieve our objective. An application to a buffer allocation problem is included along with explicit numerical results illustrating the use of this dynamic allocation scheme.

1 Introduction

Allocation of discrete resources is a problem encountered in many areas. In manufacturing, the Just-In-Time (JIT) approach has introduced the use of the *kanban*, a tag attached to each arriving job in order to maintain a small work-in-process inventory; in this case, a fixed number of *kanban* must be allocated to the various work stations in order to minimize or maximize some performance measure. Other classic examples include the buffer allocation problem in queueing models where a fixed number of buffers must be allocated over a fixed number of servers, and the transmission scheduling problem in radio networks where a fixed number of time slots forming a "frame" must be allocated over several nodes.

[1] This work was supported in part by the National Science Foundation under Grant EEC-9527422, by AFOSR under contract F49620-95-1-0131 and by the Air Force Rome Laboratory under contract F30602-95-C-0242.

In the basic model we will consider, there are K identical resources to be allocated over N user classes so as to optimize some system performance measure (objective function). Let $\mathcal{S}$ be the (discrete) set of feasible resource allocations

$$\mathcal{S} = \{[n_1, \cdots, n_N] : \quad n_j \in \{1, \cdots, K\}\}$$

where n_i is the number of resources allocated to the ith user, and by "feasible" we mean that the allocation may have to be chosen to satisfy some basic requirements, such as 'stability' or 'fairness'. Let $L_i(n_i)$ be the class i cost associated with the number of resources n_i. The *resource allocation* problem we consider is formulated as:

$$\textbf{(RA)} \qquad \min_{s\in\mathcal{S}} \sum_{i=1}^{N} \beta_i L_i(n_i) \quad s.t. \quad \sum_{i=1}^{N} n_i = K$$

where β_i is a weight associated with user class i.

For many systems encountered in practice, closed form expressions for the performance measure $L_i(n_i)$ are difficult to obtain or are simply unavailable. Moreover, systems frequently operate in a stochastic environment (e.g., random demand fluctuations) and therefore, one is forced to resort to estimation techniques (e.g., Monte Carlo simulation) to obtain an estimate of the system's performance $\hat{L}_i(n_i)$ over a specific sample path. In this case, $L_i(n_i)$ in **(RA)** has to be replaced by the expectation of $\hat{L}_i(n_i)$ over all possible sample paths, i.e., $L_i(n_i) = E[\hat{L}_i(n_i)]$. Usually, the actual value of this expectation heavily depends on the distributions that govern the processes which characterize the various events of the system. This implies that the optimal allocation is also dependent on the statistics of the underlying stochastic processes, which, in general, are not stationary. Consequently, the optimal allocation may

change over time. This motivates our objective, i.e., to develop a controller that will react to changes in the underlying processes and will be able to reallocate the resources in a way so that the system performance is improved. In this context, Liberatore et. al. [7] have used Finite Perturbation Analysis (FPA) to extract the performance of a manufacturing system over all feasible buffer allocations using only information observed from the nominal sample path and by employing a simple control scheme that switches to the allocation with the best performance. Our approach is intended to exploit the special structure of certain problems and is based on the *on-line* algorithm proposed by Cassandras and Julka [2]. This is an iterative algorithm driven by "finite differences", and it converges to the optimal allocation for deterministic systems that satisfy some separability and convexity assumptions. At every step of this algorithm the following information is required in order to determine the *next* allocation: *(a)* Performance of the system under the current allocation, and *(b)* Performance of the system under all "neighboring" allocations i.e., allocations that differ by $+1$ and -1 resources in two distinct users. Of course, if a closed form expression for $L_i(n_i)$ is unavailable, one is faced with the problem of obtaining performance estimates for all the necessary allocations.

It is by now well-documented in the literature that the nature of sample paths of DES can be exploited so as to extract a significant amount of information, beyond merely an estimate of the performance measure $J(\theta)$ under some parameter θ such as $dJ/d\theta$ or $\Delta J/\Delta\theta$ (see [1, 6] where several forms of Perturbation Analysis (PA) are described). *Concurrent Estimation* [3] is a technique through which one can construct multiple sample paths of the system under different parameters *using only information available along the given sample path.* Using the information obtained from the nominal sample path we are able to obtain estimates of the performance of the system for all "neighboring" allocations which are then used by the on-line algorithm to reallocate the resources to the various users so as to improve the system's performance.

The main contribution of this paper is in developing a *dynamic* resource allocation scheme for DES and exploring its applicability on a particular application compared to a *static* allocation approach. This scheme combines two basic components: *(a)* A discrete optimization algorithm, and *(b)* An estimation algorithm which will provide all the information required by *(a)*.

2 Concurrent Estimation

In this section, we present an approach based on "concurrent estimation" (see [3]) for solving the *constructability problem (CO)* [5] for general DES. Specifically, arbitrary lifetime distributions are allowed, unlike the *Standard Clock (SC)* approach [8] and the *Augmented System Analysis (ASA)* [5] which are two efficient methods for solving the *(CO)* problem but are limited to models with exponentially distributed event lifetimes. The main idea is to observe the evolution of a sample path of the actual system under the nominal allocation. As the sample path evolves, observed data (e.g., event occurrences and their corresponding occurrence times) are processed to concurrently construct the set of sample paths that would have resulted if the system had operated under a set of different (hypothetical) allocations.

2.1 Sample Path Constructability

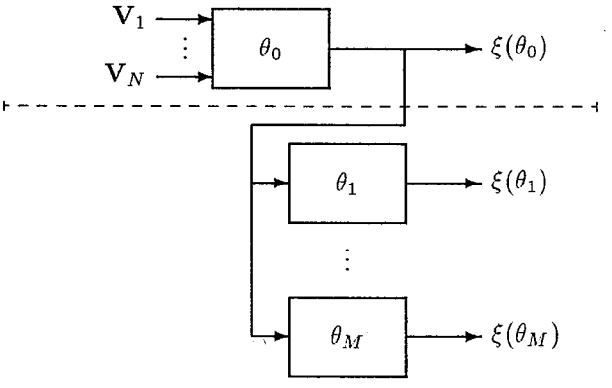

Figure 1: The sample path constructability problem for DES

We consider a DES and adopt the modeling framework of a *stochastic timed state automaton* $(\mathcal{E},\mathcal{X},\Gamma,f,x_0)$ [1]. Here, $\mathcal{E}$ is a countable event set, $\mathcal{X}$ is a countable state space, and $\Gamma(x)$ is a set of feasible (or enabled) events, defined for all $x \in \mathcal{X}$ such that $\Gamma(x) \subseteq \mathcal{E}$. The state transition function $f(x,e)$ is defined for all $x \in \mathcal{X}$, $e \in \Gamma(x)$, and specifies the next state resulting when e occurs at state x. Finally, x_0 is a given initial state. In addition, for simplicity we assume that the DES satisfies the non-interruption condition, i.e., once an event is enabled it cannot be disabled; this is not essential to the derivation of our results however.

Assuming the cardinality of the event set $\mathcal{E}$ is N, the input to the system is a set of event lifetime sequences $\{\mathbf{V}_1,\cdots,\mathbf{V}_N\}$, one for each event, where $\mathbf{V}_i = \{v_i(1), v_i(2),\cdots\}$ is characterized by some arbitrary distribution. For simplicity, we assume that this is an iid sequence, though straightforward extensions are possible. Under some system parameter θ_0 (e.g., an allocation s_0, the output is a sequence $\xi(\theta_0) = \{(e_k,t_k),\ k = 1,2,\cdots\}$ where $e_k \in \mathcal{E}$ is the kth event and t_k is its corresponding occurrence time (see Figure 1). Based on any observed $\xi(\theta_0)$, we can

evaluate $L[\xi(\theta_0)]$, a sample performance metric for the system. For a large family of performance metrics of the form $J(\theta_0) = E[L[\xi(\theta_0)]]$, $L[\xi(\theta_0)]$ is therefore an estimate of $J(\theta_0)$. Defining a set of parameter values of interest $\{\theta_0, \theta_1, \cdots, \theta_M\}$, the sample path constructability problem is stated as:

> *For a DES under θ_0, construct all sample paths $\xi(\theta_1), \cdots, \xi(\theta_M)$ given a realization of lifetime sequences $\mathbf{V}_1, \cdots, \mathbf{V}_N$ and the sample path $\xi(\theta_0)$.*

2.2 Notation and Definitions

First, let $\xi(n, \theta) = \{e_j : j = 1, \cdots, n\}$, with $e_j \in \mathcal{E}$, be the sequence of events that constitute the observed sample path up to n total events. Although $\xi(n, \theta)$ is clearly a function of the parameter θ, we will write $\xi(n)$ to refer to the observed sample path and adopt the notation $\hat{\xi}(k) = \{\hat{e}_j : j = 1, \cdots, k\}$ for any constructed sample path under a different value of the parameter up to k events in that path. It is important to realize that k is actually a function of n, that is $k = g(n)$, since the constructed sample path is coupled with the observed sample path through the observed event lifetimes; however, again for the sake of notational simplicity, we will refrain from continuously indicating this dependence.

Next we define the *score* of an event $i \in \mathcal{E}$ in a sequence $\xi(n)$, denoted by $s_i^n = [\xi(n)]_i$, to be the non-negative integer that counts the number of instances of event i in this sequence. The corresponding score of i in a constructed sample path is denoted by $\hat{s}_i^k = [\hat{\xi}(k)]_i$. In what follows, all quantities with the symbol " $\hat{\cdot}$ " refer to a typical constructed sample path.

Associated with every event type $i \in \mathcal{E}$ in $\xi(n)$ is a sequence of s_i^n event lifetimes

$$\mathbf{V}_i(n) = \{v_i(1), \cdots, v_i(s_i^n)\} \quad \text{for all } i \in \mathcal{E}$$

The corresponding set of sequences in the constructed sample path is:

$$\hat{\mathbf{V}}_i(k) = \{v_i(1), \cdots, v_i(\hat{s}_i^k)\} \quad \text{for all } i \in \mathcal{E}$$

which is a subsequence of $\mathbf{V}_i(n)$ with $k \leq n$. In addition, we define the following sequence of lifetimes:

$$\tilde{\mathbf{V}}_i(n, k) = \{v_i(\hat{s}_i^k + 1), \cdots, v_i(s_i^n)\} \quad \text{for all } i \in \mathcal{E}$$

which consists of all event lifetimes that are in $\mathbf{V}_i(n)$ but not in $\hat{\mathbf{V}}_i(k)$. Associated with any one of these sequences are the following operations. Given some $\mathbf{W}_i = \{w_i(j), \cdots, w_i(r)\}$,

Suffix Addition:

$\mathbf{W}_i + \{w_i(r+1)\} = \{w_i(j), \cdots, w_i(r), w_i(r+1)\}$

Prefix Subtraction:

$\mathbf{W}_i - \{w_i(j)\} = \{w_i(j+1), \cdots, w_i(r)\}$.

Note that the addition and subtraction operations are defined so that a new element is always added as the *last* element (the *suffix* of a sequence), whereas subtraction always removes the *first* element (the *prefix* of the sequence).

Next, define the set

$$A(n, k) = \{i : i \in \mathcal{E}, s_i^n > \hat{s}_i^k\} \tag{1}$$

which consists of all events whose corresponding sequence $\tilde{\mathbf{V}}_i(n, k)$ contains at least one element. Thus, every $i \in A(n, k)$ is an event that has been observed in $\xi(n)$ and has at least one lifetime that has yet to be used in the coupled sample path $\hat{\xi}(k)$. Hence, $A(n, k)$ should be thought of as the set of *available* events to be used in the construction of the coupled path.

Finally, we define the following set, which is crucial in our approach:

$$M(n, k) = \Gamma(\hat{x}_k) - (\Gamma(\hat{x}_{k-1}) - \{\hat{e}_k\}) \tag{2}$$

where, clearly, $M(n, k) \in \mathcal{E}$. Note that $\hat{e}_k$ is the triggering event at the $(k-1)$th state visited in the constructed sample path. Thus, $M(n, k)$ contains all the events that are in the feasible event set $\Gamma(\hat{x}_k)$ but not in $\Gamma(\hat{x}_{k-1})$; in addition, $\hat{e}_k$ also belongs to $M(n, k)$ if it happens that $\hat{e}_k \in \Gamma(\hat{x}_k)$. Intuitively, $M(n, k)$ consists of all *missing* events from the perspective of the constructed sample path when it enters a new state $\hat{x}_k$: those events already in $\Gamma(\hat{x}_{k-1})$ which were not the triggering event remain available to be used in the sample path construction as long as they are still feasible; all other events in the set are "missing" as far as residual lifetime information is concerned.

The concurrent sample path construction process we are interested in consists of two coupled processes, each generated by a timed state automaton. This implies that there are two similar sets of equations that describe the dynamics of each process. In addition, we need a set of equations that captures the coupling between them.

2.3 Timed State Automaton Dynamics

We briefly review here the standard timed state automaton dynamics, also known as a Generalized Semi-Markov Scheme (GSMS) (see [1, 6]). We introduce two additional variables, t_n to be the time when the nth event occurs, and $y_i(n)$, $i \in \Gamma(x_n)$, to be the residual lifetime of event i after the occurrence of the nth event (i.e., it is the time left until event i occurs). On a particular sample path, just after the nth event occurs the following information is known: the state x_n from which we can determine $\Gamma(x_n)$, the time t_n, the residual lifetimes $y_i(n)$ for all $i \in \Gamma(x_n)$, and all event scores s_i^n, $i \in \mathcal{E}$. The following equations describe the dynamics of the timed state automaton.

step 1: Determine the smallest residual lifetime among all feasible events at state x_n, denoted by y_n^*:

$$y_n^* = \min_{i \in \Gamma(x_n)} \{y_i(n)\} \quad (3)$$

step 2: Determine the triggering event:

$$e_{n+1} = arg \min_{i \in \Gamma(x_n)} \{y_i(n)\} \quad (4)$$

step 3: Determine the next state:

$$x_{n+1} = f(x_n, e_{n+1}) \quad (5)$$

step 4: Determine the next event time:

$$t_{n+1} = t_n + y_n^* \quad (6)$$

step 5: Determine the new residual lifetimes for all new feasible events $i \in \Gamma(x_{n+1})$:

$$y_i(n+1) = \begin{cases} y_i(n) - y_n^* & \text{if } i \neq e_{n+1} \text{ and } i \in \Gamma(x_n) \\ v_i(s_i^n + 1) & \text{if } i = e_n \text{ or } i \notin \Gamma(x_n) \end{cases}$$
$$\text{for all } i \in \Gamma(x_{n+1}) \quad (7)$$

step 6: Update the event scores:

$$s_i^{n+1} = \begin{cases} s_i^n + 1 & \text{if } i = e_{n+1} \\ s_i^n & \text{otherwise} \end{cases} \quad (8)$$

Equations (3)-(8) describe the sample path evolution of a timed state automaton. These equations apply to both the observed and the constructed sample paths. Next, we need to specify the mechanism through which these two sample paths are coupled in a way that enables event lifetimes from the observed $\xi(n)$ to be used to construct a sample path $\hat{\xi}(k)$. First, observe that the process described by (3)-(8), applied to $\hat{\xi}(k)$, hinges on the availability of residual lifetimes $\hat{y}_i(k)$ for all $i \in \Gamma(\hat{x}_k)$. Thus, the constructed sample path can only be "active" at state $\hat{x}_k$ if every $i \in \Gamma(\hat{x}_k)$ is such that either $i \in (\Gamma(\hat{x}_{k-1}) - \{\hat{e}_k\})$ (in which case $\hat{y}_i(k)$ is a residual lifetime of an event available from the previous state transition) or $i \in A(n,k)$ (in which case a full lifetime of i is available from the observed sample path). This motivates the following:

Definition 1: A constructed sample path is *active* at state $\hat{x}_k$ after the occurrence of an observed event e_n if, for every $i \in \Gamma(\hat{x}_k)$, $i \in (\Gamma(\hat{x}_{k-1}) - \{\hat{e}_k\}) \cup A(n,k)$.

2.4 Coupling Dynamics

Upon occurrence of the $(n+1)$th observed event, e_{n+1}, the first step is to update the event lifetime sequences $\tilde{\mathbf{V}}_i(n,k)$ as follows:

$$\tilde{\mathbf{V}}_i(n+1,k) = \begin{cases} \tilde{\mathbf{V}}_i(n,k) + v_i(s_i^n + 1) & \text{if } i = e_{n+1} \\ \tilde{\mathbf{V}}_i(n,k) & \text{otherwise} \end{cases} \quad (9)$$

The addition of a new event lifetime implies that the "available event set" $A(n,k)$ defined in (1) may be affected. Therefore, it is updated as follows:

$$A(n+1,k) = A(n,k) \cup \{e_{n+1}\} \quad (10)$$

Finally, note that the "missing event set" $M(n,k)$ defined in (2) remains unaffected by the occurrence of observed events:

$$M(n+1,k) = M(n,k) \quad (11)$$

At this point, we are able to decide whether all lifetime information to proceed with a state transition in the constructed sample path is available or not. In particular, the condition

$$M(n+1,k) \subseteq A(n+1,k) \quad (12)$$

may be used to determine whether the constructed sample path is active at the current state $\hat{x}_k$ (in the sense of Definition 1). The following is a formal statement of this fact.

Lemma 1: A constructed sample path is active at state $\hat{x}_k$ after an observed event e_{n+1} if and only if $M(n+1,k) \subseteq A(n+1,k)$.

Assuming (12) is satisfied, equations (3)-(8) may be used to update the state $\hat{x}_k$ of the constructed sample path. In so doing, lifetimes $v_i(s_i^k + 1)$ for all $i \in M(n+1,k)$ are used from the corresponding sequences $\tilde{\mathbf{V}}_i(n+1,k)$. Thus, upon completion of the six state update steps, all three variables associated with the coupling process, i.e., $\tilde{\mathbf{V}}_i(n,k)$, $A(n,k)$, and $M(n,k)$ need to be updated. In particular,

$$\tilde{\mathbf{V}}_i(n+1,k+1) =$$
$$\begin{cases} \tilde{\mathbf{V}}_i(n+1,k) - v_i(\hat{s}_i^k + 1) & \text{for all } i \in M(n+1,k) \\ \tilde{\mathbf{V}}_i(n+1,k) & \text{otherwise} \end{cases} \quad (13)$$

This operation immediately affects the set $A(n+1,k)$ which is updated as follows:

$$A(n+1,k+1) = A(n+1,k) - \{i : i \in M(n+1,k),\ \hat{s}_i^{k+1} = s_i^{n+1}\} \quad (14)$$

Finally, applying (2) to the new state $\hat{x}_{k+1}$,

$$M(n+1,k+1) = \Gamma(\hat{x}_{k+1}) - (\Gamma(\hat{x}_k) - \{\hat{e}_{k+1}\}) \quad (15)$$

Therefore, we are again in a position to check condition (12) for the new sets $M(n+1,k+1)$ and $A(n+1,k+1)$. If it is satisfied, then we can proceed with one more state update on the constructed sample path; otherwise, we wait for the next event on the observed sample path until (12) is again satisfied. Similar to Lemma 1, we have:

Lemma 2: A constructed sample path is active at state $\hat{x}_{k+1}$ after event $\hat{e}_{k+1}$ if and only if $M(n+1,k+1) \subseteq A(n+1,k+1)$.

The analysis above is summarized below in the form of the following *Time Warping Algorithm (TWA)*.

Time Warping Algorithm *(TWA)*:

1. INITIALIZE

$n := 0$, $k := 0$, $t_n := 0$, $\hat{t}_k := 0$, $x_n := x_0$, $\hat{x}_k = \hat{x}_0$, $y_i(n) = v_i(1)$ for all $i \in \Gamma(x_n)$, $s_i^n = 0, \hat{s}_i^k = 0$ for all $i \in \mathcal{E}$, $M(0,0) := \Gamma(x_0)$, $A(0,0) := \emptyset$

2. WHEN EVENT e_n IS OBSERVED:

2.1 Use (3)-(8) to determine e_{n+1}, x_{n+1}, t_{n+1}, $y_i(n+1)$ for all $i \in \Gamma(x_{n+1})$, s_i^{n+1} for all $i \in \mathcal{E}$.

2.2 Update $\tilde{\mathbf{V}}_i(n+1,k)$ using (9).

2.3 Update $A(n,k)$ using (10)

2.4 Update $M(n,k)$ using (11).

2.5 IF $M(n+1,k) \subseteq A(n+1,k)$ then Goto **3.** ELSE set $n \leftarrow n+1$ and Goto **2.1.**

3. TIME WARPING OPERATION:

3.1 Obtain all missing event lifetimes to resume sample path construction at state $\hat{x}_k$:

$$\hat{y}_i(k) = \begin{cases} v_i(\hat{s}_i^k + 1) & \text{for } i \in M(n+1,k) \\ \hat{y}_i(k-1) & \text{otherwise} \end{cases}$$

3.2 Use (3)-(8) to determine $\hat{e}_{k+1}$, $\hat{x}_{k+1}$, $\hat{t}_{k+1}$, $\hat{y}_i(k+1)$ for all $i \in \Gamma(\hat{x}_{k+1}) \cap (\Gamma(\hat{x}_k) - \{\hat{e}_{k+1}\})$, $\hat{s}_i^{k+1}$ for all $i \in \mathcal{E}$.

3.3 Update $\tilde{\mathbf{V}}_i(n+1,k+1)$ using (13)

3.4 Update $A(n+1,k)$ using (14)

3.5 Update $M(n+1,k)$ using (15)

3.6 IF $M(n+1,k+1) \subseteq A(n+1,k+1)$ then $k \leftarrow k+1$ and Goto **3.1.** ELSE $k \leftarrow k+1$, $n \leftarrow n+1$ and Goto **2.1.**

3 On-Line Resource Allocation Algorithm

In this section we present a modified version of the optimization algorithm *on-line* in [2], adapted to fit our purposes, which we use to adjust the resource allocation vector in order to improve the performance of the system. This algorithm is designed for static deterministic resource allocation problems that satisfy the convexity assumption *A1*.

• *A1*: For all $i = 1, \cdots, N$, $L_i(n_i)$ is such that $\Delta L_i(n_i + 1) > \Delta L_i(n_i)$
where

$$\Delta L_i(n_i) = L_i(n_i) - L_i(n_i - 1), \quad n_i = 1, \cdots, K \quad (16)$$

Actually, in [2] it is proved that for this class of systems the *on-line* algorithm will always converge to the optimal allocation in a finite number of steps.

Often, however, systems operate in stochastic environments. When no closed-form expressions for the performance measure are available, in any *on-line* optimization technique the expected cost is estimated by direct measurements made on the system. In this case we want to emphasize that the performance estimate $\hat{L}_i^\tau(\cdot)$ is a random variable which depends on the length of the observation/simulation interval τ. It turns out that the original *deterministic on-line* algorithm can be modified to work with stationary stochastic systems and it has been shown in [4] that it converges in probability to the optimal allocation as $\tau \to \infty$. In what follows, we wish to consider the case where the underlying distributions are not stationary; for example, tasks are submitted to a computer system at random instants in time and require processing for a random period, but it is expected that the task arrival rate varies with the time of day. Clearly, in the case of such *dynamic* resource allocation problems we cannot allow τ to go to infinity, since waiting for long periods of time before we obtain performance estimates prevents us from reacting to changes in the system and hence making timely resource reallocations. Consequently, it is possible that the system will end up operating with poor allocations for long periods of time.

Following is the *on-line* algorithm in [2], modified to enable the system to be "adaptive" in the sense that it reacts to changes in the underlying event processes.

ALGORITHM: *Dynamic Resource Allocation*

1.0 Initialize: $\mathbf{s}^{(0)} = [n_1^{(0)}, \cdots, n_N^{(0)}]$;
$C^{(0)} = \{1, \cdots, N\}$; $k = 0$.

1.1 Evaluate

$$\hat{\mathbf{D}}^{(k)}(n_1^{(k)}, \cdots, n_N^{(k)}) \equiv [\Delta \hat{L}_1^\tau(n_1^{(k)}), \cdots, \Delta \hat{L}_N^\tau(n_N^{(k)})]$$

2.1 Set $i^* = arg\max_{i \in C^{(k)}}[\hat{\mathbf{D}}^{(k)}(n_1^{(k)}, \cdots, n_N^{(k)})]$

2.2 Set $j^* = arg\min_{i \in C^{(k)}}[\hat{\mathbf{D}}^{(k)}(n_1^{(k)}, \cdots, n_N^{(k)})]$

2.3 Evaluate
$\hat{\mathbf{D}}^{(k)}(n_1^{(k)}, \cdots, n_{i^*}^{(k)} - 1, \cdots, n_{j^*}^{(k)} + 1, \cdots, n_N^{(k)})$

2.4 If $\Delta \hat{L}_{j^*}^\tau(n_{j^*}^{(k)} + 1) < \Delta \hat{L}_{i^*}^\tau(n_{i^*}^{(k)})$ Goto **3.1**
ELSE Goto **3.2**

3.1 Update allocation:
$n_{i^*}^{(k+1)} = n_{i^*}^{(k)} - 1; n_{j^*}^{(k+1)} = n_{j^*}^{(k)} + 1; n_m^{(k+1)} = n_m^{(k)}$ for all $m \in C^{(k)}$ and $m \neq i^*, j^*$;
Set $k \leftarrow k+1$
Reset $C^{(k)} = \{1, \cdots, N\}$, and Goto **2.1**

3.2 Replace $C^{(k)}$ by $C^{(k)} - \{j^*\}$;
IF $|C^{(k)}| = 1$, Reset $C^{(k)} = \{1, \cdots, N\}$, and Goto **2.1** ELSE Goto **2.2**

4 An Application to Buffer Allocation

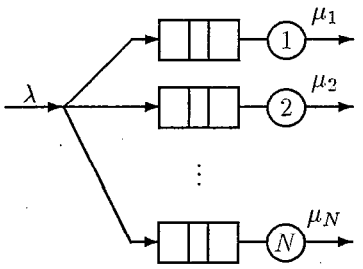

Figure 2: Queueing system with N parallel servers

The *Dynamic Resource Allocation* algorithm together with the *concurrent estimation* scheme, have been applied to the queueing system shown in figure 2 where each server represents a user and each buffer slot represents a resource to be allocated to a user (it is assumed that buffers can be freely reallocated from one server to another). Jobs arrive at the system at a rate λ and are routed to one of the N users with some probability $p_i, i = 1 \cdots N$. Each of the users is servicing jobs at a rate $\mu_i, i = 1, \cdots N$. Jobs that are routed to a user with a full queue are lost and in this case we would like to allocate all K available buffer slots to each one of the users in order to minimize the sum of the probabilities of losing a job from each individual queue due to an overflow.

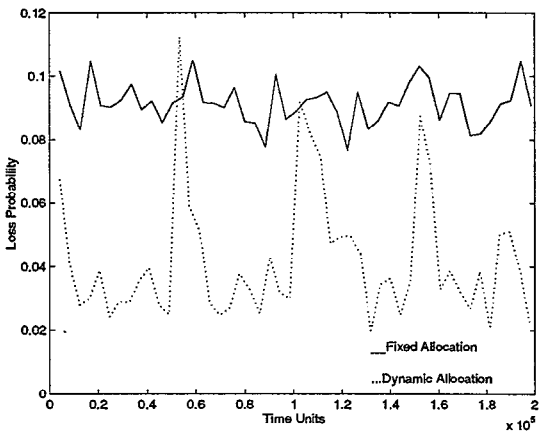

Figure 3: System performance under the *dynamic* allocation scheme vs. a *fixed* resource allocation vector

We consider a system with $N = 4$ and $K = 16$ when the arrival rate $\lambda = 1.3$ and all service rates $\mu_i = 1.0$ for all $i = 1, \cdots, 4$. We also assume that the routing probabilities are not stationary and that they change every 50,000 time units as shown in Table 1. In figure 3 we plot the performance of the *Dynamic Resource Allocation* scheme against the performance of a system with a fixed allocation $\mathbf{s}_f = [4, 4, 4, 4]$. Note that $\mathbf{s}_f$ is the optimal *fixed* allocation over the interval $(0-20,000)$ due to the symmetry of the routing probabilities.

From -	To	distribution
0 -	50,000	$\mathbf{p}_1 = [0.7, 0.1, 0.1, 0.1]$
50,000 -	100,000	$\mathbf{p}_2 = [0.1, 0.7, 0.1, 0.1]$
100,000 -	150,000	$\mathbf{p}_3 = [0.1, 0.1, 0.7, 0.1]$
150,000 -	200,000	$\mathbf{p}_4 = [0.1, 0.1, 0.1, 0.7]$

Table 1: Routing Probabilities

5 Conclusions

In this paper we have developed a control scheme to dynamically reallocate discrete resources in DES in order to improve performance. The scheme consists of two parts: *(a)* Concurrent Estimation, which is used to estimate the performance of the system under a set of different allocations in order to evaluate the finite differences required by the *Dynamic Resource Allocation* algorithm, and *(b)* The *Dynamic Resource Allocation* algorithm, which uses the finite differences to make the reallocation decisions. This control scheme can be applied to DES with separable convex objective functions.

References

[1] Cassandras C.G.,"Discrete Event Systems, Modeling and Performance Analysis", *IRWIN*, (1993).

[2] Cassandras C.G. and Julka V. "Descent Algorithms for Discrete Resource Allocation Problems", *Proc. 33rd IEEE Conf. Decision and Control*, pp. 2639-2644, (1994).

[3] Cassandras C. G. and Panayiotou C. G., "Concurrent Sample Path analysis Of Discrete Event Systems", *Proc. of 35th IEEE Conf. Decision and Control*, pp. 3332-3337, (1996).

[4] Cassandras C. G., Dai L., and Panayiotou C. G., "Stochastic Optimization for Discrete Resource Allocation", subm. to *36th IEEE Conf. Decision and Control*, (1997).

[5] Cassandras C.G. and Strickland S.G., "Observable Augmented Systems for Sensitivity Analysis of Markov and Semi-Markov Processes", *IEEE Transactions on Automatic Control*, Vol. AC-34, pp.1026-1037, (1989).

[6] Ho Y.C. and Cao X., "Perturbation Analysis of Discrete Event Systems", *Kluwer*, Boston, 1991.

[7] Liberatore G., Nicosia S. and Valigi P., "Dynamic Allocation of Buffer Capacity in Discrete Event Systems", subm. to *Journal of Intelligent Automation and Soft Computing*, (1996).

[8] Vakili P., "A Standard Clock Technique for Efficient Simulation", *Operations Research Letters*, Vol.10, pp. 445-452, (1991).