Technical Report: MS-CIS-05-20
Optimal Control of Software ensuring Safety and Functionality

Arvind Easwaran, Sampath Kannan, Insup Lee
arvinde, kannan, lee@cis.upenn.edu
University of Pennsylvania, Philadelphia

October 11, 2005

Abstract

Existing verification and validation methodologies can detect software violations very effectively but fail to provide any mechanism for correcting faults once they are detected. Detection of faults, their diagnosis and corrective actions are all essential components of any software rectification framework. In this paper, we propose a framework for correction of violations in software systems ensuring that the desired goals of the system are achieved. We describe a stochastic finite state machine used to abstract a software system along with the uncertainty in its operating environment. Safety property violations and satisfaction of functionalities are abstracted using penalties and rewards on the states, respectively. Rectification of software is then formulated as a stochastic optimal control problem over this abstraction. Algorithms polynomial in the size of the abstraction have been developed for solving this optimization problem exactly. The paper also applies the developed framework to a variety of examples from different domains.

1 Introduction

Software systems are designed and implemented to satisfy certain predetermined user requirements. These user requirements identify functionalities desired from the system in addition to safety properties it is expected to possess. Errors in design or implementation often result in the user requirements not being met. Although verification and validation techniques can determine the presence of such errors (especially safety violations), they fail to correct the system in order to prevent their occurrence. Correction techniques must preserve functionality while preventing the occurrence of safety violations. Control theory plays a significant role in control of engineering systems. Analogously, we aim to control a software system minimizing violations of safety properties and simultaneously maximizing the achievable goals in the system.

Formal models help in unambiguously describing software systems and also in arguing formally about properties that the system possesses. Software correction can be facilitated if the system is abstracted as a formal model. Any software system interacts with an unpredictable operating environment. Since the execution of the system depends on the environment, any abstraction of the system must capture this uncertainty. Control actions on

the abstract formal model would result in modifications to the model leading to a correct system. The abstraction must then capture the cost of control that the controller will incur for execution of control actions on the system. Cost of control must depend on the criticality of the part of the system being modified. Controllability of an execution point refers to the ability of the controller to execute control actions at that point. In practice, not all execution points in a software system are controllable and hence abstractions must capture the notion of uncontrollability in the system. Controllers must also preserve functionalities of the system while preventing safety property violations. Abstractions must then identify execution points that either achieve system goals or result in safety property violations.

In this paper, we describe a finite state machine based formal model used to abstract software systems. This model abstracts the uncertainty in the operating environment as well as the cost of control for the system. Rewards associated with satisfaction of goals and penalties for safety violations are used to prevent violations while preserving functionality. Control actions on this model involve blocking a subset of the transitions of the finite state machine. We then formulate the optimal stochastic control problem in terms of the abstracted model and allowed control actions. Stochasticity results in optimization of expected values of control costs, rewards and penalties. Optimality is achieved by minimizing cost of control and penalties and simultaneously maximizing rewards. Minimizing control cost ensures that control is achieved by expending the least effort in terms of modifying the system. Maximizing rewards results in maximizing available functionalities in the system and minimizing penalties results in minimizing the safety violations. Note that if a safety violation is catastrophic, the abstraction can assign infinite penalty to the corresponding system state to ensure that the optimal control policy never allows the system to enter that state. Algorithms with running time polynomial in the size of the abstraction are then described for solving this optimization problem. Related work is discussed towards the end of the paper.

2 Motivation

In this section we describe three problems from different domains which will motivate the need for a general framework for control of software.

2.1 Assuring Quality of Service for Web Services

A web service application consists of a web server providing services to various distributed clients using multiple servers [26]. The distributed clients send service requests to the web server which then executes the requests on its servers. The web service application software basically performs two jobs. It controls admission of all the incoming service requests and schedules the waiting service requests on idle servers. Quality of service (QoS) parameters like maximum latency are associated with incoming service requests. Latency for a service request can be defined as the time between the arrival of the request at the web server and the completion of service of that request by the server. These parameters are previously agreed upon by the client and the server. An architecture for a typical web service application is as shown in Figure 1.

Every service request from a distributed client belongs to the same fixed class or priority. The web server has one accept queue per class. The admission control software monitors the service requests arriving from distributed clients. Based on the control logic of the ad-

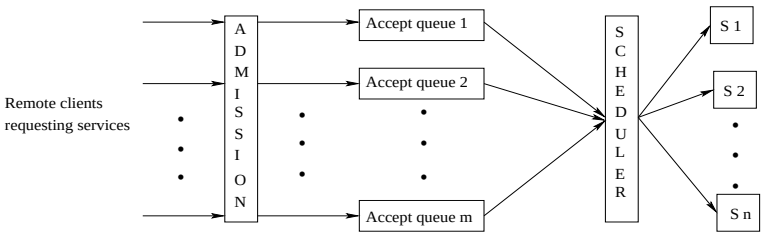


Figure 1: Web Service Software Architecture

mission controller, it will determine whether to accept a particular service request. If the admission controller accepts a service request, then that request is queued into an accept queue determined by the class of the request. The task scheduling software is responsible for scheduling of service requests waiting for service in the accept queues. Whenever any server on the web server becomes idle, the scheduling software will determine the next service request to be processed by the idle server. The scheduler will make use of the current latency (time since arrival) and the priority of waiting requests to determine the next request to be processed. Once allocated to a server, requests will be executed to completion. The goal of the web server software (admission controller and task scheduler) is to ensure that the QoS requirements of all the accepted requests are satisfied (different request classes will in general have different QoS requirements). We will present a more detailed model of this application as an example of the general framework we develop in the next few sections.

2.2 Dynamic Power Management

Application of embedded devices in various domains has resulted in growing importance for reducing power consumption in hardware components. Dynamic power controller is a specialized control software that aims to reduce the power consumption of a hardware component by predicting the arrival of requests that use the component. An architecture for the dynamic power controller along with the system is as shown in Figure 2. The architecture consists of a set of service requesters also known as clients ($SR_1, \cdots, SR_m$) requesting services from the system. The system has a set of request queues of fixed size ($Q_1, \cdots Q_n$), one for each class or priority. We assume that all the requests from a particular client belong to the same class. The service provider (SP) is the hardware component that processes requests waiting for service. The provider can be in one of three states depending on the amount of power it is consuming. It consumes the maximum power when it is in the 'active' state (processing request), the least power when it is in 'sleep' state and moderate power when it is in 'idle' state (waiting for request). Changing the state of SP from 'idle' to 'sleep' or from 'sleep' to 'active' will consume a fixed amount of power and time. The dynamic power manager (DPM) is responsible for controlling the state of the service provider. DPM uses current occupation levels of the queues and prediction of request arrivals to determine the state of the provider. If a new request for a particular class arrives when the queue is full, then that request will be dropped. Further each class has a QoS parameter which determines the maximum allowed latency for that request class. The DPM must not only minimize power consumption but must also reduce request loses and ensure satisfaction of QoS requirements. The DPM determines the state of the provider that results in minimiza-

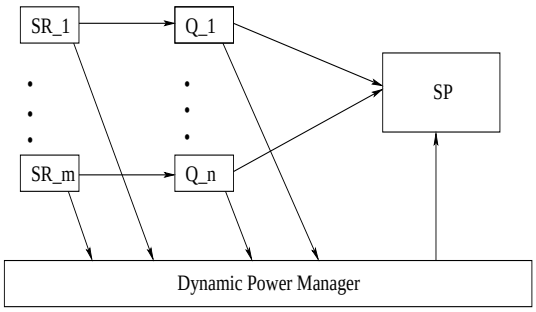


Figure 2: Dynamic Power Controller

tion of power consumption and request loss and simultaneously leads to satisfaction of QoS requirements using its prediction for future request load on the SP.

2.3 Aircraft Routing under Weather Uncertainty

Explosive growth of air travel has resulted in increased delays most of which are weather induced. Traditional routing strategies for aircrafts use the weather predictions to completely avoid bad weather zones in the flight path. Although this strategy results in a safe route, it is extremely conservative in terms of the distance traveled by the aircraft to reach its destination. The weather bureau makes predictions of storms which are revised periodically. These predictions identify storm zones in the airspace along with a probabilistic estimate for the occurrence of storm in that zone between two predictions. Current routing strategies ignore the probabilities completely and avoid all storm zones in their path planning algorithms. They also do not update the planned route periodically in response to new predictions of storms. Nilim et. al. in papers [23, 22] have formulated the aircraft path planning problem under storms as a Markov decision process. The path planner is required to compute a route for the aircraft that minimizes the distance traveled and also ensures that the path is not affected by any storm. Standard optimization algorithms for Markov decision processes like policy iteration using bellman recursion are then employed to determine a route that will minimize the expected distance traveled by the aircraft from a given source to destination. Algorithms for solving optimization problems over Markov decision processes like policy iteration using bellman recursion have poor running times when there are exponentially many actions at each state. This is true for the aircraft path planning problem because two different routes can be optimal under expectation from a given state. We will formulate this path planning problem in our framework which will then result in an efficient algorithm for solving the optimization problem.

3 Finite State Abstraction and Problem Statement

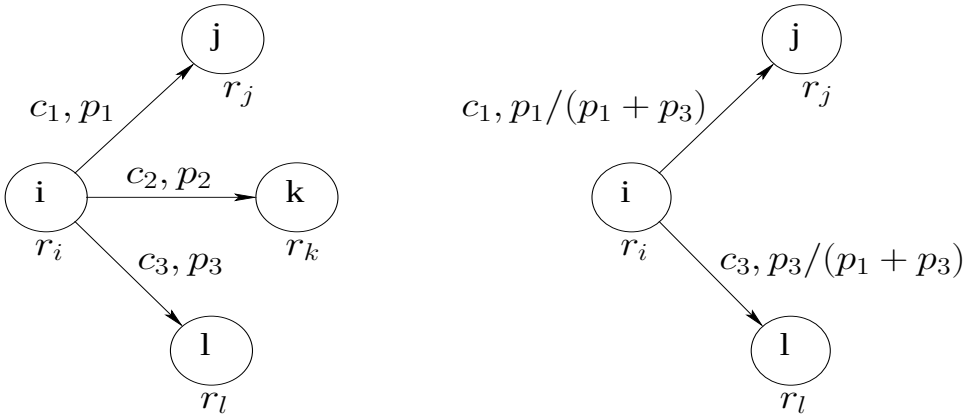
A software system consists of sequential execution of finite number of discrete steps. This system can be represented abstractly by a finite state machine M. To capture the uncertainty in the operating environment, the transitions of M can have associated probabilities. A mapping of states to rewards/penalties helps determine whether they violate safety properties

or satisfy some functionality. Each transition of M represents an action that the software system may execute when the system is at the source state of that transition. In this paper we assume that blocking the execution of these transitions are the only possible control actions. Blocking a transition t in M results in a reduced finite state machine in which transition t is disabled. Control actions blocking transitions are simple to implement and can satisfy the objective of preventing safety violations ensuring maximization of existing functionality. Cost of control can then be associated with transitions which identifies the cost incurred by the system in implementing control actions. To model uncontrollable transitions we let the cost of control for these transitions be infinite.

Definition Formally, a finite state machine abstraction for a software system can be represented by a labeled graph $M = (S, s, T, \Sigma, F)$ where,

- S is the set of states of M and $s \in S$ is the initial state. Assuming there are n states in the model, let the states be numbered 1 thru n with start state $s = 1$ wlog. We assume that all the states $i \in S$ are reachable from the start state.
- T is the set of transitions where $\{i, j\} \in T$ represents a transition from state i to state j.
- Σ is the set of events and $F \subseteq S$ is the set of final states of M. States in F do not have any outgoing transitions.
- Each transition $\{i, j\} \in T$ in the graph has the following parameters:
  - Event $e_{i,j} \in \Sigma$ abstracting a set of instructions executed by the target system.
  - Conditional probability $p_{i,j}$ that the system takes this transition given that the system is in the source state of this transition.
  - Cost of blocking $c_{i,j} \in \mathcal{R}^+$ which represents the cost that the controller will incur if a control action blocks this transition.
- Each state $i \in S$ has a reward/penalty parameter $r_i \in \mathcal{R}$. If state i violates a safety property then r_i is positive. But if the state satisfies some system functionality then r_i is negative.

Let $p_{i,j} = c_{i,j} = 0$ if transition $\{i, j\}$ does not exist in model M. Also let $\forall i \in S \backslash F, \sum_{j=1}^{n} p_{i,j} = 1$. This assumption ensures that when the system reaches a state i in the abstraction, it will take one of the outgoing transitions at that state. A control action on M involves blocking a subset of the transitions. The set of all control actions for M can then be given as $\mathcal{CA}^M = \{A | A \subseteq T\}$ where $A \in \mathcal{CA}^M$ is the set of transitions blocked by control action A. Hence the total number of control actions $|\mathcal{CA}^M|$ for M is equal to $2^{|T|}$ where $|T|$ is the total number of transitions in M. The reduced finite state machine to which control action $A \in \mathcal{CA}^M$ has been applied will be referred to as the controlled system and denoted by M^A. Let $A_i = \{\{i, j\} | (\{i, j\} \in A)\}$ where $A \in \mathcal{CA}^M$ denote the set of outgoing transitions blocked at state i under control action A. Also let $c^A_{i,j}$ be the cost of control for transition $\{i, j\}$ under control action A. $c^A_{i,j} = c_{i,j}$ if $\{i, j\} \in A$ and is 0 otherwise. In a controlled system M^A with non-empty A_i, since some of the outgoing transitions at state i are blocked, conditional probabilities on the remaining unblocked transitions will violate the probability distribution assumption stated earlier. Transition probabilities at state i must then be modified (normalized) to generate a distribution. This situation is shown in Figure 3 which shows a partial



(a) Initial Abstraction

(b) Controlled Abstraction

Figure 3: Dynamic Probabilities for Transitions

finite state machine M consisting of states i, j, k and l. Figure 3(a) shows M prior to application of control action and Figure 3(b) shows the controlled system M^A after a control action $A_i = \{i, k\}$ has been applied. As shown in Figure 3(b), probabilities of the unblocked transitions have to be normalized to maintain the probability distribution property at state i.

Let $p^A_{i,j}$ define the new conditional probability of transition $\{i, j\}$ when control action A is applied to M.

$$p^A_{i,j} = p_{i,j}/(\textstyle\sum_{k:\{i,k\}\notin A} p_{i,k}) \qquad \text{If } (\{i,j\} \notin A)$$
$$p^A_{i,j} = 0 \qquad \text{otherwise}$$

Given a model M we would like to determine a control action A such that the controlled system M^A has the least expected total cost at its start state as compared to any other control action. We will now formulate the objective function for this control problem. Two factors that contribute to the total cost are the control cost for blocking transitions and the net expected penalty and reward in the modified transition system. Blocking a transition may increase the total cost due to a non-negative cost of blocking associated with that transition. If any state in M^A is reachable from the start state, then the reward or penalty at that state will contribute to the total cost as well. If the state is a desired state then it will reduce the total cost and if it is a violating state then it will result in an increase in the total cost. Since the state machine is stochastic, we can calculate expected values for costs incurred at states. We now formally define the objective function for the expected value of the total cost at any state of M for a particular control action A. The optimal stochastic control problem can then be defined as minimizing this objective function at the start state $s = 1$ over all possible control actions $\mathcal{CA}^M$.

Definition The expected cost at a state $i \in S$ under a control action $A \in \mathcal{CA}^M$ for $M = (S, s, T, \Sigma, F)$ is,

$$E_i^A = r_i + \sum_{j=1}^{n} c_{i,j}^A + \sum_{j=1}^{n} (p_{i,j}^A \times E_j^A) \tag{1}$$

At state i, the controller will get a reward or penalty of r_i for reaching that state. It will also incur the blocking cost which is given by $\sum_{j=1}^{n} c_{i,j}^A$ under control action A. Further, the controller will incur an expected cost given by $\sum_{j=1}^{n} (p_{i,j}^A \times E_j^A)$ for all successor states j ($p_{i,j}^A$ is 0 if j is not a successor state of i or if $\{i, j\}$ is blocked under A). E_j^A is the expected total cost at state j under control action A. The optimal stochastic control problem can now be defined as,

Definition Given $M = (S, s, T, \Sigma, F)$ compute,

$$\min_{A \in \mathcal{CA}^M} \{E_1^A\} = \min_{A \in \mathcal{CA}^M} \{r_1 + \sum_{j=1}^{n} c_{1,j}^A + \sum_{j=1}^{n} (p_{1,j}^A \times E_j^A)\} \tag{2}$$

where E_j^A is the optimal expected total cost at state j under control action A.

Since arbitrary transition systems can be decomposed into strongly connected components (SCCs) connected by a directed acyclic graph (DAG), we consider in turn the control of a DAG and the control of SCCs and put these together to achieve control of general systems.

4 Optimal Control of Directed Acyclic Graphs

In this section we will describe a dynamic programming algorithm for solving the optimization problem given by Eq. (2) for the case when M is a directed acyclic graph (DAG). We will also prove for DAGs that the objective function satisfies the optimal substructure property.

4.1 Optimal Substructure Property of the Objective Function

Assume E_1^* is the optimal cost for the start state $s = 1$ and that this optimal cost is incurred when control action A is applied to M i.e, $E_1^A = E_1^*$ and it is given by,

$$E_1^A = r_1 + \sum_{j=1}^{n} c_{1,j}^A + \sum_{j=1}^{n} (p_{1,j}^A \times E_j^A) \tag{3}$$

Let $R_1^A \subseteq S$ denote the set of successor states of state 1 in M^A (states in S with incoming transitions from 1 under control action A) such that $|R_1^A| = k$. Let the expected total cost at each of the states in R_1^A be suboptimal. Consider the k subproblems of determining the optimal expected total costs at the k states in R_1^A where for each state $j \in R_1^A$ optimization is done over the subgraph M_j of M originating from j and only having states reachable from j. Let $\forall j \in R_1^A, E_j^{B_j}$ denote the optimal expected total cost obtained by solving the subproblems where B_j denotes the optimal control action associated with subgraph M_j. Since each M_j is a DAG, the optimal expected total cost at any state of M_j is independent of the transitions and paths that lead to that state. Hence the control actions $\{B_j | j \in R_1^A\}$ do not conflict at any state which occurs in more than one subgraph. In other words, the control actions specified

by the set $\{B_j | j \in R_1^A\}$ at a state l that occurs in more than one subgraph are all equivalent. Since $E_j^{B_j}$ are optimal expected costs for all j in the set R_1^A and E_j^A are sub-optimal, we know that $\forall j \in R_1^A, E_j^{B_j} < E_j^A$. Now, consider the control action $C = A_1 \bigcup_{j \in R_1^A} B_j$. The corresponding expected total cost at state 1 for the entire graph M is given by,

$E_1^C = r_1 + \sum_{j=1}^n c_{1,j}^C + \sum_{j=1}^n (p_{1,j}^C \times E_j^C)$

$= r_1 + \sum_{j=1}^n c_{1,j}^A + \sum_{j=1}^n (p_{1,j}^A \times E_j^{B_j})$ by definition of C and DAG property of M

$< r_1 + \sum_{j=1}^n c_{1,j}^A + \sum_{j=1}^n (p_{1,j}^A \times E_j^A)$ since $\forall j \in R_1^A, E_j^{B_j} < E_j^A$ and $p_{1,j}^A$ is a probability measure

$= E_1^A = E_1^*$ using Eq. (3)

This implies that E_1^* was not optimal which is a contradiction. Hence the objective function given in Eq. (2) satisfies the optimal substructure property and we can use a dynamic programming algorithm to solve the optimization problem.

4.2 Dynamic Programming Algorithm for DAGs

The recursive equation of the objective function in Eq. (2) is given by,

$$E_i^* = r_i + \min_{A_i} \{ \sum_{j=1}^n c_{i,j}^A + \sum_{j=1}^n (p_{i,j}^A \times E_j^*) \} \tag{4}$$

where E_j^* is the optimal cost for state j under some control action A_j and $A = \bigcup_{j=1}^n A_j$.

A naive algorithm will first topologically sort the states using TOPOLOGICAL-SORT algorithm given in [9] in time $O(m+n)$ where m is the number of transitions in M. It will then evaluate the expected costs at states in the order of reverse topological sort using Eq. (4). Since M is a DAG, the optimal costs at all successor states of i would already have been computed before computing the cost at state i if states were considered in reverse topological order. For each state i, the algorithm will evaluate the expected cost for all possible control actions and then pick the control action giving the least expected cost. Since $A_i \subseteq \{\{i,k\} | (\{i,k\} \in T)\}$ for any state i, the time to compute the objective function at that state is $O(2^{od_i})$ where od_i is the out degree of state i. Hence the total running time of the algorithm is $O(m+n) + O(n2^n) = O(n2^n)$. Next we exploit a structural property of M to produce a polynomial time algorithm. We first describe the algorithm and then give a proof of its correctness.

4.2.1 Improved Dynamic Programming Algorithm

The improved algorithm for solving the optimization problem given in Eq. (4) is described in Algorithm 1. This algorithm topologically sorts the state space and evaluates costs at states in reverse order of the sort as in the naive approach. But at each state of M, instead of evaluating the objective function for all possible control actions, it determines optimal control by evaluating the objective function for only a polynomial subset of the control actions. This is possible because of a structural property of M that this algorithm exploits.

Consider some state i of M for which the algorithm is currently determining the optimal control action. Since M is a DAG, the optimal cost at all successor states of i is known. The procedure shown in Algorithm 1 first evaluates the total expected cost at state i assuming all outgoing transitions are unblocked. It then blocks an arbitrary outgoing transition and evaluates the new expected cost. If the new cost is smaller than the current expected cost, then the algorithm permanently blocks this transition. It then repeats this process until blocking transitions leads to no further decrease in the expected total cost at state i. Correctness of this approach is proved in Section 4.2.2 of this paper. The running time of this algorithm at each state i is $O(od_i^2)$ (since in each execution of the loop at least one transition is blocked and each step takes at most $O(od_i)$ time to execute). Hence the total running time over all states is $O(\sum_{i \in S} od_i^2)$ and the total running time of the entire algorithm is $O(m+n) + O(\sum_{i \in S} od_i^2) = O(\sum_{i \in S} od_i^2)$.

Algorithm 1 OptConDAGS

Topologically sort vertex set using TOPOLOGICAL-SORT algorithm
Initialize optimal control action $A = \phi$
//Evaluate costs in reverse topological order
for Each state i in reverse topological order **do**
  Let $E_i^A = r_i + \sum_{j=1}^{n}(p_{i,j}^A \times E_j^*)$ //Assuming all outgoing transitions are unblocked
  //Let Q be a queue containing all outgoing transitions of i in arbitrary order
  while Exists transition in Q whose blocking reduces objective value **do**
    Block transition at the head of Q ($\{i,k\} = pop(Q)$)
    //Compute new objective function value
    $E_i^B = r_i + c_{i,k}^B + \sum_{j=1}^{n}(p_{i,j}^B \times E_j^*)$ where $B = A \cup \{i,k\}$
    //If objective function value reduces, then block $\{i,k\}$
    if $E_i^B < E_i^A$ **then**
      $E_i^A = E_i^B$ and $A = B$
    end if
    //If objective function value does not reduce then push $\{i,k\}$ back into Q
    if $E_i^B \geq E_i^A$ **then**
      $push(\{i,k\})$
    end if
  end while
end for

4.2.2 Proof of Correctness of the Algorithm

Let $S_1 \subset S \setminus \{i\}$ be some arbitrary subset of states and let $S_2 = S \setminus (S_1 \cup \{i\})$. Consider the following equations which denote the expected cost at some state m ($m \in S$) during various stages of the algorithm given in Algorithm 1,

$E_1 = E_m^A = r_m + \sum_{j \in S_2}(p_{m,j}^A \times E_j^A) + \sum_{j \in S_1}(p_{m,j}^A \times E_j^A) + p_{m,i}^A \times E_i^A$ where $A_m = \phi$

$E_2 = E_m^B = r_m + c_{m,i}^B + \sum_{j \in S_2}[(p_{m,j}^A \times E_j^A)/(1 - p_{m,i}^A)] + \sum_{j \in S_1}[(p_{m,j}^A \times E_j^A)/(1 - p_{m,i}^A)]$ where $B = A \cup \{m,i\}$

$E_3 = E_m^C = r_m + \sum_{j \in S_2}[(p_{m,j}^A \times E_j^A)/(1-R)] + M + (p_{m,i}^A \times E_i^A)/(1-R)$ where $R = \sum_{j \in S_1} p_{m,j}^A, C = A \cup \{\{m,j\} | j \in S_1\}$ and $M = \sum_{j \in S_1} c_{m,j}^C$

$E_4 = E_m^D = r_m + \sum_{j \in S_2}[(p_{m,j}^A \times E_j^A)/(1-R-p_{m,i}^A)] + c_{m,i}^D + M$ where $D = C \cup \{m,i\}$

E_1 is the objective function value at the beginning of the algorithm for state m. E_2 is the value when transition $\{m,i\}$ has been blocked. E_3 represents the value of the function when transitions to $j \in S_1$ from m have been blocked but $\{m,i\}$ has not yet been blocked. E_4 represents the value when transitions to $j \in S_1$ from m and transition $\{m,i\}$ have been blocked. We now prove that,

Theorem 4.1 $(E_1 > E_2) \wedge (E_1 > E_3) \Rightarrow (E_3 > E_4)$

Proof Using equations for E_1 and E_2 we get,

$E_1 - E_2 = p_{m,i}^A E_i^A - c_{m,i}^B - \sum_{j \in S_2}[(p_{m,j}^A \times E_j^A)(p_{m,i}^A)/(1-p_{m,i}^A)] - \sum_{j \in S_1}[(p_{m,j}^A \times E_j^A)(p_{m,i}^A)/(1-p_{m,i}^A)] > 0$

Also, from equations for E_1 and E_3 we get,

$E_1 - E_3 = -p_{m,i}^A E_i^A (R/(1-R)) - M - \sum_{j \in S_2}[(p_{m,j}^A \times E_j^A)(R/(1-R)) + \sum_{j \in S_1}(p_{m,j}^A \times E_j^A) > 0$

Substituting this equation in the equation for $E_1 - E_2$ we get,

$p_{m,i}^A E_i^A (1-R-p_{m,i}^A)/[(1-p_{m,i}^A)(1-R)] - c_{m,i}^B - M p_{m,i}^A/(1-p_{m,i}^A) - \sum_{j \in S_2}[(p_{m,j}^A \times E_j^A) p_{m,i}^A/[(1-p_{m,i}^A)(1-R)]] > 0$

$p_{m,i}^A E_i^A (1-R-p_{m,i}^A)/[(1-p_{m,i}^A)(1-R)] - c_{m,i}^B - \sum_{j \in S_2}[(p_{m,j}^A \times E_j^A) p_{m,i}^A/[(1-p_{m,i}^A)(1-R)]] > 0$ since M is positive

$\Rightarrow p_{m,i}^A E_i^A/(1-R) - c_{m,i}^D (1-p_{m,i}^A)/(1-R-p_{m,i}^A) - \sum_{j \in S_2}[(p_{m,j}^A \times E_j^A) p_{m,i}^A/[(1-R)(1-R-p_{m,i}^A)]] > 0$ since $c_{m,i}^B = c_{m,i}^D$

$\Rightarrow p_{m,i}^A E_i^A/(1-R) - c_{m,i}^D - \sum_{j \in S_2}[(p_{m,j}^A \times E_j^A) p_{m,i}^A/[(1-R)(1-R-p_{m,i}^A)]] > 0$ since $R + p_{m,i}^A \geq p_{m,i}^A$

$\Rightarrow E_3 > E_4$ □

This implies that if blocking a transition $\{m,i\}$ reduces the expected cost at state m at some stage in the algorithm, then transition $\{m,i\}$ will be blocked in the optimal control action as well. This proves correctness of the algorithm described in Section 4.2.1.

5 Optimal Control of Strongly Connected Components

In this section we assume wlog that states $\{1, \cdots, n\}$ of a finite state machine M with state space $S = \{1, \cdots, m\}$ where $n \leq m$ form a strongly connected component S_1 (SCC). We also

assume that all control costs, rewards and penalties are finite. Uncontrollable transitions and critical safety violations can be specified by assigning very high but finite costs and penalties to the transitions and states, respectively. We will show that in general, the optimization problem defined in Eq. (2) need not have a solution for strongly connected components. We will then modify the objective function to ensure existence of a unique solution for general SCCs and also describe an efficient algorithm for solving the same.

5.1 Conditions for Existence of Optimal Solution

A strongly connected component is called terminal when all the vertices in the SCC have transitions only to vertices in the same SCC [9]. Any strongly connected component which is not terminal is called a non-terminal SCC. We will show that for a fixed control action, the stochastic control problem has a solution if and only if the SCC is non-terminal. This implies that the optimization problem restricted to S_1 given by Eq. (2) has a solution if and only if S_1 is a non-terminal SCC. We then modify the objective function with discounted rewards and penalties to guarantee existence of a solution for terminal SCCs.

5.1.1 Objective Function Formulation for a Fixed Control Action

From Eq. (2) the objective function restricted to S_1 for a fixed control action A is given as,

$$\begin{aligned} E_1^A &= r_1 + \textstyle\sum_{j=1}^{n}(c_{1,j}^A) + \sum_{j=1}^{n}(p_{1,j}^A \times E_j^A) \\ \text{where } E_j^A &= r_j + \textstyle\sum_{i=1}^{n}(c_{j,i}^A) + \sum_{i=1}^{n}(p_{j,i}^A \times E_i^A) \end{aligned} \tag{5}$$

Let for control action A, $M_j = r_j + \sum_{i=1}^{n}(c_{j,i}^A)$, $p_{j,i}^A = p_{j,i}$ and $E_j^A = E_j$. If S_1 is non-terminal ($n < m$), then we can assume the expected costs for states not in S_1 with transitions from S_1 be known. Further, if a state i in S_1 has a transition to a state k not in S_1, then we assume that the expected cost incurred by state i from state k is absorbed in the constant M_i. Thus the control problem given in Eq. (5) can be represented as a set of n simultaneous equations with n expected cost variables. Let E represent the column vector $(E_1 \cdots E_n)$, M represent the column vector $(M_1 \cdots M_n)$, I denote the $n \times n$ identity matrix and P be the row stochastic matrix,

$$P = \begin{pmatrix} p_{1,1} & p_{1,2} & p_{1,n} \\ p_{2,1} & p_{2,2} & p_{2,n} \\ \cdots\cdots\cdots\cdots\cdots \\ \cdots\cdots\cdots\cdots\cdots \\ p_{n,1} & p_{n,2} & p_{n,n} \end{pmatrix} \tag{6}$$

The set of n linear simultaneous equations with n expected cost variables is then given by,

$$E = M + P \times E \Rightarrow (I - P)E = M \tag{7}$$

Thus a solution to the simultaneous equation set given in Eq. (7) gives the value $E_1 = E_s$ for the objective function given by Eq. (5).

5.1.2 Existence of a Solution for Non-Terminal SCCs

We now show that the matrix $(I - P)$ is non-singular whenever S_1 is a non-terminal SCC. In Eq. (7), P is an irreducible, non-negative and stochastic matrix. From Perron-Frobenius

theorem [24], the largest unique eigenvalue r for P is given by the equation,

$$r = max_i(\sum_{j=1}^{n} p_{i,j}) \quad If\ min_i(\sum_{j=1}^{n} p_{i,j}) = max_i(\sum_{j=1}^{n} p_{i,j})$$
$$min_i(\sum_{j=1}^{n} p_{i,j}) < r < max_i(\sum_{j=1}^{n} p_{i,j}) \qquad Otherwise$$

From the theory of Markov chains [29, 24], we know that for a stochastic, irreducible matrix P, $P^k/s^k \longrightarrow 0$ as $k \longrightarrow \infty$ if and only if $s > r$ where r is the Perron-Frobenius eigenvalue. Putting $s = 1$, we get $P^k \longrightarrow 0$ as $k \longrightarrow \infty$ if and only if $1 > r$ (this is the condition for a transient class). Also, for a real matrix P such that $P^k \longrightarrow 0$ as $k \longrightarrow \infty$, $(I - P)^{-1}$ exists and is given by $(I - P)^{-1} = \sum_{k=0}^{\infty} P^k$. Combining this with the Perron-Frobenius theorem we get that Eq. (7) has an unique solution if and only if $\sum_{j=1}^{n} p_{i,j} \leq 1, \forall 1 \leq i \leq n$ and $\exists i, \sum_{j=1}^{n} p_{i,j} < 1$

This condition implies that Eq. (7) has a unique solution if and only if any one vertex in the SCC has a transition to a state not in the SCC (non-terminal SCC). Hence we have shown that the stochastic optimal control problem can be solved exactly for non-terminal SCCs.

5.1.3 Modified Objective Function for Terminal SCCs

For matrix P in Eq. (7), we have $\sum_{j=1}^{n} p_{i,j} \leq 1, \forall i : 1 \leq i \leq n$ with strict equality for states with transitions only to states in the SCC. If S_1 is terminal, then each row of the matrix $(I - P)$ in Eq. (7) will sum to 0 and hence its determinant will be 0. Hence the stochastic optimal control problem as defined by Eq. (2) need not have a solution for terminal strongly connected components.

In practice, most abstractions for software systems result in a terminal strongly connected component because of the ability of the system to reset itself to the start state from any other state. Hence we will modify the objective function defined in Eq. (2) to ensure existence of a solution for terminal SCCs. We will modify these equations by discounting the contribution of rewards and penalties to the expected cost at a state. Discounted rewards and penalties, derived from the theory of Markov decision processes [6, 27], captures the notion that the contribution of reward or penalty from a state farther in the future must be exponentially smaller than the contribution from a state nearer in the future. Let $0 < \alpha < 1$ be a fixed discount factor for the terminal strongly connected component S_1. The modified objective function with discount factor α is then given by,

$$E_1 = M_1 + \alpha \sum_{j=1}^{n} (p_{1,j} \times E_j) \tag{8}$$

where E_j is defined similarly. In vector notation this equation is given by,

$$E = M + \alpha P \times E \Rightarrow (I - \alpha P)E = M \tag{9}$$

Consider any terminal strongly connected component S_1. Let S_1' be a non-terminal strongly connected component generated from S_1 by adding a new transition from every state in S_1 to a single new dead state. For each old transition in S_1 let its probability of occurrence

in S_1' be multiplied by a factor α. Further, let the conditional probability on every new transition in S_1' be $(1-\alpha)$ and the reward associated with the dead state be 0. The discounted reward control problem given by Eq. (9) for S_1 is then equivalent to the standard control problem given by Eq. (7) for the non-terminal SCC S_1'. Since the matrix αP can be mapped to an equivalent stochastic matrix for a non-terminal SCC as described, from Section 5.1.2 we know that $(I - \alpha P)$ is non-singular and hence the discounted reward control problem given by Eq. (9) has a unique solution for any general SCC.

5.2 Algorithm for Stochastic Optimal Control of SCCs

In Section 5.1, we showed that the stochastic control problem for a fixed control action can be formulated as a set of n linear simultaneous equations with n variables as given by Eq. (9). We also showed that Eq. (9) has a unique solution for any general SCC. A naive algorithm for solving the stochastic optimal control problem solves such simultaneous equations for every possible control action in the abstraction M. Since at each state of M, an exponential number of control actions exist (each transition can be blocked or unblocked), we would be required to solve an exponential set of simultaneous equations. This leads to an algorithm which is exponential in the size of the input abstraction M. In this section, we will formulate this stochastic optimal control problem for SCCs as a linear programming problem (LPP) and give a polynomial time algorithm for solving this LPP.

5.2.1 Stochastic Optimal Control using Ellipsoid Algorithm

The objective function for stochastic optimal control of a finite state machine M is given by the equation,

$$\min_{A \in \mathcal{CA}^M} \{E_1^A\} = \min_{A \in \mathcal{CA}^M} \{r_1 + \sum_{j=1}^{n} (c_{1,j}^A) + \sum_{j=1}^{n} (p_{1,j}^A \times E_j^A)\} \tag{10}$$

where, E_j^A is the expected total cost at state j under control action A. Since the discounted reward control problem for terminal SCCs can be mapped to an equivalent standard control problem for non-terminal SCCs, we only consider the objective function for non-terminal SCCs in this section. We now construct a linear programming problem (LPP) equivalent to the stochastic optimal control problem given by Eq. (10). Let, $E_i, \forall 1 \leq i \leq n$ be the linear program variables where E_i^A are specific values for the variables. Let $M^{A_i} = r_i + \sum_{j=1}^{n} c_{i,j}^A$ and $p_{i,j}^A = p_j^{A_i}$. The equivalent LPP formulation for Eq. (10) is then given by,

Definition Maximize : $E_1 = E_s$

Subject To : $E_i \leq M^{A_i} + \sum_{j=1}^{n} [E_j \times p_j^{A_i}], \forall A_i, \forall 1 \leq i \leq n$

This objective function is certainly convex and hence the ellipsoid algorithm can be used to solve this LPP in time polynomial in the number of LPP variables (polynomial in the size of the abstraction M). Interior point algorithms like Karmarkar's do not perform well in the presence of exponential number of LPP constraints. The ellipsoid algorithm starts by containing the feasible region within a ellipsoid and then generates a sequence of ellipsoids each of successively smaller volume. In each iteration, the algorithm examines the center of the ellipsoid for feasibility. The algorithm then determines a half-space defined by a hyperplane passing through the center. If the current center is not feasible, then the hyperplane can be

obtained by identifying a violating constraint of the LPP using a separation oracle. A separation oracle is a black box, which when given a point in space either identifies a violating constraint in the LPP for that point or declares that the point is feasible. For the ellipsoid algorithm to run in polynomial time, this separation oracle must be able to generate results in polynomial time as well. If the center is feasible, then the algorithm uses the objective function cut to eliminate feasible points with objective value no better than its value at the current center. A new ellipsoid is then generated by finding the minimum volume ellipsoid containing the half-ellipsoid obtained by the intersection of the hyperplane and the current ellipsoid. One can then show that after a polynomial number of iterations, the volume of the ellipsoid is small enough such that the feasible point generated is very close to the optimal value.

In order that the ellipsoid algorithm execute in time polynomial in the number of LPP variables, a polynomial time separation oracle is required. For the LPP given in Definition 5.2.1 a polynomial time separation oracle can be generated using the algorithm described in Section 4.2.1. Given a point in the solution space of the LPP (values for E_i), the oracle needs to either identify a violating constraint if the point is infeasible or declare that the point is feasible. Since the different number of possible control actions for the abstraction M is exponential in the size of the abstraction, the number of LPP constraints are also exponential. Hence a naive algorithm which tests every constraint will take exponential time to determine a violating constraint. Instead, the separation oracle can run the algorithm given in Algorithm 1 at each state of M to identify a violating constraint in polynomial time. For each state i there are exponential number of constraints in the LPP corresponding to the exponential number of control actions at that state. For a given point in the solution space, the algorithm can determine the least value of the expected cost at some state i using the given expected costs at its successor states. This least value of the expected cost at state i corresponds to some control action at state i and hence corresponds to a particular LPP constraint. If this least value determined by the algorithm is smaller than the value of E_i at the current point in the solution space, then the corresponding LPP constraint is a violating constraint. The minimum over all right hand sides of the LPP constraints in Definition 5.2.1 for a state i is equivalent to the objective function as given by Eq. (2) for the algorithm at state i. Thus by running Algorithm 1 at each state of M we can identify a violating constraint for the point if it is infeasible. Since the ellipsoid algorithm executes in time $O(n^6)$ and the separation oracle can run in time $O(\sum_{j=1}^{n} od_j^2)$ for each step, we get a polynomial time algorithm for the optimal stochastic control problem for general SCCs.

5.2.2 Equivalence of Optimization Problems defined by Eq. (10) and Def. 5.2.1

Let the solution to the LPP problem given in Def. 5.2.1 be achieved when control action is $a = \bigcup_{i=1}^{n} a_i$ where a_i is control action at state i. Let the values of LPP variables E_1 thru E_n for this optimal point be E_1^a thru E_n^a respectively. Since E_1^a is an optimal value for the LPP, there are atleast n LPP constraints which are tight for the point $(E_1^a, \cdots, E_n^a)$ such that each E_i appears on the LHS of at least one of them. If there exists some E_k for which no constraint with E_k on the LHS is tight, then E_k can be increased without violating any constraint, thereby leading to an eventual increase in E_1 because the component is strongly connected. In particular, the n constrains corresponding to the control actions $(a_1, \cdots, a_n)$ must be tight. Let these n constraints with LHS E_1 thru E_n be labeled C_1 thru C_n, respectively. Eq.(10) solves sets of simultaneous equations one for each control action and chooses the action

which results in the least value for E_1. The tight satisfaction of constraints $C_1, \cdots, C_n$ also gives a solution to Eq.(10) for control action a.

We now show that any further increase in E_1 will always result in violation of some LPP constraint. Let for some state k the expected total cost $E_k = E_k^a$ be increased by an amount equal to the minimum slackness among all the LPP constraints with E_k on the LHS except constraint C_k. Suppose wlog that $succ_k = \{k+1, \cdots k+j | k+j \leq n\}$ represents the set of successor states of k in M. The LHS of the constraint labeled C_k is now $E_k^{new} = E_k^a + c$ where c is the amount by which E_k was incremented. This implies that constraint C_k is currently being violated. On the RHS of C_k we compute the expected value of total costs at states in $succ_k$. Increase in the RHS value of constraint C_k depends on a weighted average of the increase in the total costs at each state in $succ_k$. To ensure that the constraint is not violated, each of E_{k+1} thru E_{k+j} must increase by an amount equal to c (maximum increase in any cost is c). But this is not possible because the component is a non-terminal strongly connected component. There is at least one path with non-zero probability starting from some state $(k+l) \in succ_k$ that leaves the strongly connected component. Then $E_{k+l}^{new} < E_{k+l}^a + c$ and hence RHS of constraint C_k can never increase by c. Thus constraint C_k will always remain violated and hence the value of E_1 cannot be increased beyond E_1^a. Similarly it can be shown that any value of E_1 which is smaller than E_1^a will not give a solution to Eq.(10) for any control action because there will never be n tight constraints in the LPP. Since no value of E_1 smaller than E_1^a provides a solution for Eq. (10) and no value of E_1 bigger than E_1^a can satisfy the LPP constraints, control action a must be the optimal control action generating the minimum value of expected total cost at state 1. This proves equivalence of the optimization problems defined by Eq. (10) and Def. 5.2.1.

6 Optimal Stochastic Control of Generic Graphs

Section 4 describes a dynamic programming algorithm for the determination of optimal stochastic control for abstractions which are DAGs. Section 5 describes the LPP formulation and the ellipsoid algorithm which can be used to compute the optimal stochastic control for abstractions which are SCCs. In this section we combine these two algorithms to generate an algorithm for determination of optimal stochastic control for any generic abstract finite state machine.

6.1 Algorithm for Generic Finite State Machines

A SCC graph for any finite state machine M is a directed acyclic graph with nodes identifying the strongly connected components of M. Given M with strongly connected components $scc_1, \cdots scc_k$, the SCC graph SCC^M will have k vertices corresponding to the k strongly connected components of M. Further there will be an edge between vertices scc_i and scc_j in SCC^M if and only if there exists an edge between some vertex in scc_i and some vertex in scc_j in M [9]. The algorithm for computing the optimal stochastic control action for M is given in Algorithm 2. This algorithm will first construct the SCC graph SCC^M for M and then topologically sort the vertices of SCC^M. For each vertex scc_i of SCC^M in reverse topological order, if the vertex is a non-trivial SCC (SCC with more than one state in M) then the algorithm would use the LPP formulation and the ellipsoid algorithm described in Section 5. For each possible start state of scc_i (state in M belonging to scc_i and having an in-

coming transition from another SCC of M), the algorithm would run the ellipsoid algorithm and determine the optimal expected cost at that state. The LPP formulation for different start states will differ only in the objective function with the constraints remaining the same. If the vertex scc_i is a trivial SCC then the algorithm would use the algorithm described in Section 4 to compute its optimal cost.

Algorithm 2 OptConGeneric

1: Compute SCC graph SCC^M for M using STRONGLY-CONNECTED-COMPONENTS
2: Topologically sort the vertex set $scc_1, \cdots, scc_k$ of SCC^M using TOPOLOGICAL-SORT //For each vertex of SCC^M compute the expected optimal cost. If vertex is a trivial SCC then use dynamic programming algorithm else use ellipsoid algorithm.
3: **for** Each vertex scc_i in reverse topological order **do**
4: **if** scc_i is a trivial SCC of M representing state j **then**
5: Compute E_j by running the algorithm OptConDAGS at state j
6: Store optimal control action A and optimal cost E_j^A at state j
7: **end if**
8: **if** scc_i is a non-trivial SCC of M **then**
9: Determine states $Start = \{l_1, \cdots l_p\}$ of M in scc_i to which there are incoming transitions in M from other SCCs of M //Identify all possible start states for scc_i //Compute optimal cost for each start state
10: With each state j in $Start$ as start state of scc_i, compute optimal expected cost using ellipsoid algorithm described in Section 5
11: Store optimal control action A and optimal cost E_j^A at state j
12: **end if**
13: **end for**

The SCC graph SCC^M can be constructed in time $O(m+n)$ using the algorithm STRONGLY-CONNECTED-COMPONENTS given in [9] where m is the number of transitions and n is the number of vertices in M. Topological sort of the SCC graph can be done using the algorithm TOPOLOGICAL-SORT given in [9] in time $O(l + k)$ where l is the number of edges and k is the number of vertices in SCC^M ($l \leq m$ and $k \leq n$). For each vertex of SCC^M, if it is a trivial SCC then we run the dynamic programming algorithm once. If it is a non-trivial SCC having j vertices of M, then we run the ellipsoid algorithm at most j times. Each run of the dynamic programming algorithm or the ellipsoid algorithm is polynomial in the size of the abstraction and the ellipsoid algorithm is run at most a polynomial number of times. Hence the total running time of this generic algorithm is also polynomial in the size of the abstraction.

7 Stochastic Optimal Control of Web Servers

In this section, we will abstract the web service application given in Section 2 as a finite state machine. We can then formulate the problem of satisfying QoS requirements by the web server as a stochastic optimal control problem. Solution to this problem will generate an admission controller and a task scheduler for the web server which will guarantee that QoS requirements for all the accepted service requests are satisfied optimally.

7.1 Framework Assumptions

Stochastic optimal control algorithms designed in this paper have running times polynomial in the size of the abstraction. These algorithms would be efficient for a web server framework only if the framework can be abstracted as a finite state machine. In this framework, we assume that the web server has a fixed number of request classes with a finite buffer for the accept queues of each class. Since the queues are finite, the state space of the abstraction for this system will also be finite. Let m be the number of different classes of service requests where a higher class implies higher quality of service. Further, let time be abstracted or discretized as a fixed quantum t and let n be the number of servers on the web server. We assume that the arrival process for each request class is a Poisson arrival process with a fixed arrival rate. We also assume that time abstraction t is small enough such that the probability of more than one arrival for a class within t time units is 0. Let $L_i, \forall 1 \leq i \leq m$ represent the size of the accept queues. For each class i we assume that we are given,

- Poisson arrival rate λ_i for service requests. λ_i is the average number of request arrivals for class i in a time interval of length t. For stability, the request arrival rate for a class must be smaller than the request service rate for that class. Since service time for any request is greater than t (time is discretized in steps of t), λ_i is less than or equal to one.
- Worst case execution time (WCET) for requests. Let $WCET_i$ be the WCET for class i where $WCET_i = Kt$ such that $K \in \mathcal{I}^+$.
- Probability distribution DET_i for execution times of requests. This distribution is defined for an interval of length $WCET_i$ and is discretized with respect to time quantum t.
- Quality of service requirement for the class as a deadline $D_i \geq WCET_i$. D_i is the maximum latency within which any request of class i must be processed by the web server.

7.2 State Space

A state in the abstract state machine consists of,

- State of all the accept queues given by $\forall 1 \leq k \leq m, A_k = \langle AT_k^1, AT_k^2, \cdots, AT_k^{i_k} \rangle$ where $i_k \leq L_k$. $AT_k^j \leq 0$ is the arrival time for the j^{th} request of class k in accept queue A_k. This arrival time is discretized with respect to t and represents the arrival time of the request backward in time i.e., AT_k^j is decremented by t with each progress of time transition. To ensure finiteness of the abstraction, we will store only a finite history of request arrival times. As soon as a particular service request's arrival time exceeds the QoS deadline for that class ($|AT_k^i| > D_k$), AT_k^i will be abstracted to a constant L for all future states indicating that the service request has violated its QoS requirements.
- State of all the servers given as $S = \{\langle AT_k^j, ET_k^j \rangle | (1 \leq k \leq m) \wedge (1 \leq j \leq n)\}$. Further, $|S| \leq n$ where $|S|$ is the cardinality of the set S. AT_k^j is the arrival time in accept queue and $ET_k^j \leq 0$ is the execution time of the request currently executing on server j and belonging to class k. ET_k^j and AT_k^j are decremented by t with each progress of time transition.

- $AR = i$ indicating arrival of a service request of class i in the previous step (any state with incoming transition representing arrival of request). If no request arrived in the previous step, then $AR = \phi$.
- $ER = \langle j_1, \cdots, j_l \rangle$ where $j_l \leq m$ and $j_a < j_b, \forall a < b$ indicating that requests of class j_1 thru j_l have expired in the previous step (any state where the arrival times of requests of classes j_1 thru j_l have just become L thereby indicating violation of QoS requirements). If no request expired in this state, then $ER = \phi$.
- SS indicating whether a start of service transition can occur at this state or not ($SS = true$ indicates that start of service transitions can occur at this state). SS is set to $true$ on completion of service of a request, arrival of a new request in an empty accept queue with idle servers in the system or on progress of time with idle servers in the system.

7.3 Transition Set and State Classification

Different transitions in the abstract state machine include,

- a_k indicating arrival of a new service request for class k. On execution of this transition variable AR gets updated to $AR = k$
- acc indicating acceptance of a new service request into the accept queue. If $AR = k$, then execution of this transition will result in the queue $A_k = \langle AT_k^1, AT_k^2, \cdots, AT_k^{i_k} \rangle$ being updated to $A_k = \langle AT_k^1, AT_k^2, \cdots, AT_k^{i_k+1} \rangle$ with $AT_k^{i_k+1} = 0$ and AR is set to ϕ.
- rej indicating rejection of the recently arrived service request. On execution of this transition, AR gets updated to $AR = \phi$ and the accept queues remain unchanged.
- T indicating progress of time by the fixed quantum t. On execution of this transition, all the arrival times get updated to $AT_k^i = AT_k^i - t$ and all the execution times get updated to $ET_k^i = ET_k^i - t$. For any request at the head of accept queues, if arrival time AT_k^1 satisfies the condition $|AT_k^1| > D_k$, then $AT_k^1 = L$ and $ER = ER \cup k$. SS is set to $true$ if the system has idle servers.
- rem indicating removal of the expired service request at the head of the queue ER from the web server. If $ER = \langle j_k, \cdots, j_l \rangle$, then j_k is removed from queue ER and $AT_{j_k}^1$ is removed from queue A_{j_k}.
- ret indicating that the recently expired request at the head of the queue ER must be retained in the system for delayed servicing. If $ER = \langle j_k, \cdots, j_l \rangle$, then j_k is removed from queue ER.
- c_j indicating completion of service of request executing on server j. Set S gets updated to $S = S \setminus \{\langle AT_k^j, ET_k^j \rangle\}$ and SS is set to $true$.
- s_k^j indicating start of service of request at the head of queue A_k on server j. The server state S gets updated to $S = S \cup \{\langle AT_k^j, ET_k^j \rangle\}$ with $ET_k^j = 0$ and $AT_k^j = AT_k^1$ where AT_k^1 is the arrival time of request at the head of queue A_k. AT_k^1 is removed from queue A_k and if $|S| = n$, then SS is set to $false$.

- na indicating no assignment of service request to any server resulting in variable SS being set to $false$.

Transitions of type T, c_j and a_k with very high control costs are assumed to be uncontrollable. Also, every path in the machine between a s_k^j transition and a c_j transition is assumed to have atleast one occurrence and atmost $WCET_k$ occurrences of transition T. The state space of this abstraction is classified as follows.

- **Environment State:** This is an uncontrollable state with outgoing transitions of type T, c_j and a_k. This state is identified by the boolean expression $(AR = \phi) \wedge (ER = \phi) \wedge (SS = false)$. At this state, either a new service request could arrive based on the poisson arrival process associated with each class or a currently executing request could finish execution based on its execution time distribution or time could progress without any other event occuring.

- **Admission Control State:** This state is identified by a non-empty value for AR. The only outgoing transitions at this state are acc and rej which control the admission of the recently arrived service request. If the accept queue is full, then this control state will only have one outgoing transition labeled rej.

- **Server Assignment State:** This state is represented by the variable SS having value $true$. Outgoing transitions at this state are server assignment transitions (s_k^j) and no assignment transition (na) where j is the lowest numbered idle server at the current state. For each request class k with a non-empty queue, there will be one server assignment transition s_k^j. No assignment transitions will be used by the scheduler to prevent starvation of higher priority requests by lower priority ones.

- **Expired Request State:** This state is represented by a non-empty queue ER. The only outgoing transitions at this state are rem and ret indicating either removal or retainment of the expired request at the head of queue ER.

7.4 Rewards, Control and Stochasticity

Penalties will be associated with states which indicate recent expiry of a service request (AT_k^i set to L in this state). Higher the class of the expired request, higher would be the penalty. Penalties will also be associated with states indicating expiry of requests currently executing on the server. Rewards will be associated with states indicating completion of service prior to the expiry of the request ($(AT_k^j \leq D_k) \wedge (c_j \; has \; just \; occurred)$). Again, higher the request class, higher would be the reward for that state. Appropriate rewards for requests of different classes along with the na transitions will prevent starvation of a higher class request by a lower class request. Start of service (s_k^j), no assignment (na), acceptance or rejection of requests (acc, rej) and removal or retainment of expired requests (rem, ret) are the only controllable transitions in this machine. Control cost incurred by the controller in executing control actions will be 0 for all the transitions.

Consider some control state i (admission control, server assignment or expired request) of this web server abstraction. Since the cost of control of all the outgoing transitions is 0, the optimal expected total cost at this state will be equal to the minimum value of the optimal costs at all the successor states of i. The corresponding control action would then block

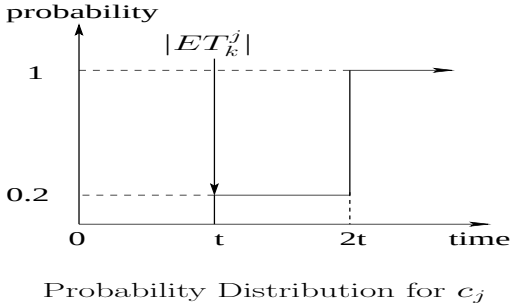


Figure 4: Probability Distribution for Service Time

all the outgoing transitions at state i except the one that leads to the successor state with the least expected cost. If more than one successor state has the same minimum expected cost, then any of them can be chosen as the successor state in the controlled system. Thus the control algorithm at any control state will first identify the successor state with the least expected cost among all the successor states and then will block all but one transition. Since the algorithm is independent of the probability distribution of the outgoing transitions, any probability distribution of these transitions would result in the same optimal control. In particular, assigning equal probabilities to all the outgoing transitions is also a valid assignment.

Control algorithm described in Section 4.2.1 and the LPP formulation given in Section 5.2.1 can be modified for this web server abstraction. Greedy algorithm given in Section 4.2.1 can be simplified to identify the successor state with the least expected cost among all the successor states of any given control state. If there are more than one such successor states, then the algorithm can pick any one of them arbitrarily. It can then generate the control action for this control state by blocking all but one transition. Thus the complexity of the algorithm can be reduced from $O(od_i^2)$ for a control state i to $O(od_i)$. LPP formulation given in Section 5.2.1 can also be simplified by reducing the number of constraints at all the control states. Since optimal control action can result in only a single transition remaining unblocked, we only have a linear number of possible optimal control actions at any state. Hence the number of constraints at any control state in the LPP formulation will also be linear in the out-degree of the state. It is important to note that control algorithms for the web server abstraction are simplified only because transitions do not have any control costs associated with them.

At all the environment states, probability of occurrence of transitions depends on the poisson arrival rates for various classes and also on the distribution of execution times for the currently executing requests. For example let the probability distribution for the execution time of a certain request of class k be as given in Figure 4. Let the current state of the machine be such that the request has already executed for time t ($ET_k^j = -t$) and let there be only one server in the system. We further assume that there are only two classes of requests in the system, one with arrival rate $\lambda_1 = 0.5$ and the other with arrival rate $\lambda_2 = 0.4$. The probabilities on various transitions for this example web server is as shown in Figure 5. Transition a_1 represents the occurrence of an event where a new request of class 1 arrives and no request of class 2 arrives and request on server j does not finish its execution. The probability of occurrence of this event is 0.21 ($(1-0.2) \times (1-(\lambda_2 e^{-\lambda_2})) \times \lambda_1 e^{-\lambda_1}$ normalized with other probabilities). Transitions a_2 and c_j similarly represent arrival of a new request of class 2 and completion of service of request on server j, respectively. Transition T represents

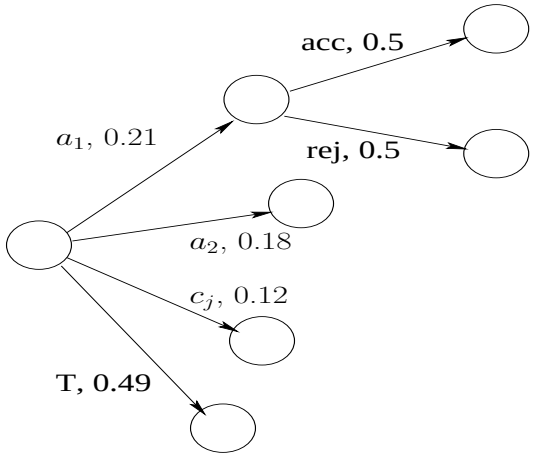


Figure 5: Probability Assignment for Transitions

the event where neither a new request arrives nor the currently executing request finishes its execution. Since any subset of the events a_1, a_2 or c_j can occur, the abstraction must represent all possible interleavings of these events.

8 Dynamic Power Management

We will now abstract the dynamic power controller described in Section 2 in our framework and design an optimal dynamic power manager (DPM).

8.1 Power Controller Abstraction

Let n be the number of request classes in the system where a higher class implies higher quality of service. We assume that the request arrival process of each class i is modeled as a Poisson process with fixed arrival rate λ_i. λ_i is the average number of request arrivals for class i in a time interval of length t (t is the fixed quantum of discretization for time in our abstraction). We further assume that t is small enough for the probability of more than one arrival within the interval to be 0. Each request class i has a worst case execution time ($WCET_i$) for requests. The service time of requests in each class is given by a bounded probability distribution DET_i which is discretized in time quantum t. To guarantee stability, the request arrival rate must be smaller than request service rate and hence we require that $\forall i, \lambda_i \leq 1$. Further each request class has a quality of service parameter $D_i \geq WCET_i$ which gives the maximum latency within which requests in that class must be processed. We assume that pending requests in the queue are scheduled by a fixed priority scheduler that processes a request if and only if there are no pending higher priority requests.

A state in the abstraction is represented by the size of all the request queues and the state of the service provider SP. Size of queue can be further abstracted into either being 'full', 'empty' or 'partial'. Thus any state in the system abstraction has one variable for the state of SP and n variables for the states of the n request queues. It also has other variables like arrival time of requests to keep track of the latency. Loss of request can be determined by a

variable which is set to true only when the queue for a class is full and a new request for that class arrives. Uncontrollable transitions in the abstraction are arrival of a new request for class i (a_i), progress of time by fixed quantum t (T) and completion of service of a request (c). Controllable transitions in the abstraction are changing the state of SP from 'idle' to 'sleep', 'sleep' to 'active' or letting the state remain unchanged. On the execution of a T transition, current time of the system is increased by t. On the occurrence of a_i, the request loss and queue variables for class i are updated appropriately. On the occurrence of a c transition, state of SP is changed to 'idle'. On the occurrence of controllable transitions, the state of SP is changed accordingly. Whenever the state of SP changes from 'idle' to 'sleep' or from 'sleep' to 'active', a T transition will be forced to account for the latency incurred by the system as a result of the state change. Treatment similar to web servers for requests failing to meet QoS requirements will guarantee finiteness of the state space.

The state space of this abstraction can be classified into controllable and uncontrollable states depending on their outgoing transitions. Each state with an incoming c transition indicates completion of service. Depending on the latency incurred by the request a reward or penalty will be associated with such a state. A penalty will be incurred at a state where the request loss variable is true and also at states where the state of SP is 'idle' indicating loss of power. Cost of control will be associated with controllable transitions indicating no change of state (to force a change of state, transitions indicating no change of state have to be disabled). This cost of control must be proportional to the loss of power as a result of the state change. Uncontrollable transitions (arrival, completion and progress of time) will have transition probabilities based on the arrival process and probability distribution for request completion. All the outgoing transitions at a controllable state will have equal probabilities. Executing the optimization algorithm described in this paper will result in simultaneous minimization of power loss, request loss and request latencies.

9 Aircraft Path Planning with Weather Uncertainty

We now describe the aircraft path planning problem formally and abstract it into our framework so that efficient optimization algorithms can be used to compute the optimal route.

9.1 Problem Statement

We consider a two-dimensional airspace split into N square grids for routing of an aircraft. Suppose each square grid i is represented by a two-dimensional point (x_i, y_i). Let (x_1, y_1) be the source point of the aircraft in the airspace and (x_2, y_2) be the destination point. The path planner must generate a route from the source point (x_1, y_1) to the destination point (x_2, y_2). The aircraft is assumed to travel at a constant height with a constant velocity. We also assume that the airspace under consideration is affected by k storms. The weather bureau makes predictions for each of the k storms periodically. We assume time to be abstracted in periods of T where T is the period with which the weather bureau makes predictions. At each prediction stage, the bureau generates a probability distribution $p_m = (p_{m,1}, p_{m,2}, \cdots, p_{m,N})$ for each storm m over the entire abstracted airspace. For each grid i in the airspace, $p_{m,i}$ gives the probability that storm m will be in grid i within time T given the current position of the storm. Given the current position of the storm and assuming that in time T the storm can travel a distance of at most one grid, the number of grids that the storm can reach within an

interval of length T is a very small subset of N. Hence the probability vector p_m will always be sparsely populated with most entries being 0. Given this information, the goal of the path planner is to determine a route for the aircraft from a given source to destination that will minimize the distance traveled by the aircraft. The calculated route must avoid all storms in the airspace and the route of the aircraft must be updated after every new prediction of storms is received i.e., after every T time units.

9.2 Path Planner Abstraction

We now formulate the aircraft path planning problem in our framework. Each state in our abstraction represents the current position of the aircraft in the airspace ((x, y)) along with the positions of each of the k storms. Hence the total state space of the abstraction is $N \times N^k = N^{k+1}$ which is very large. But, since the storm location between successive decision epochs does not change much (it can move to neighboring grids only), the actual state space at each prediction stage is a very small subset of N^{k+1}. Also, in practice the number of storms k will be a small constant and hence the state space will only be polynomially bigger than N. There are two types of transitions in the abstraction.

- Uncontrollable storm transitions which track the movement of storms in space. These transitions are probabilistic (depending on the current weather prediction) and track the movement of storms from one grid to another.
- Controllable aircraft transitions which track the movement of the aircraft from one grid to another neighboring grid. From a given state, all the outgoing aircraft transitions are assumed to have the same probability. The aircraft is equally likely to go to any of the neighboring grids until a particular control action is applied by the path planner.

Outgoing transitions of a particular state in the abstraction are either all storm transitions or all aircraft transitions. States with only storm transitions are called uncontrollable states and those with only aircraft transitions are called controllable states. A high penalty will be associated with states which represent the presence of the aircraft and any one of the k storms in the same grid in airspace. A reward will be associated with states that identify the presence of the aircraft at the destination (x_2, y_2). Storm transitions will have very high control costs associated with them to abstract their uncontrollability. Control costs on controllable transitions (aircraft transitions) will be used to identify transitions that take the aircraft closer to the destination as compared to its current position. Let (x, y) be the current position of the aircraft and $N(x, y)$ denote the set of neighboring grids of grid (x, y). The control cost for a transition that takes the aircraft from its current position (x, y) to a new grid $(l, m) \in N(x, y)$ is given by,

$$c_{l,m} = ||(x, y) - (x_2, y_2)|| - ||(l, m) - (x_2, y_2)|| + |\min_{(i,j)\in N(x,y)}(||(x, y) - (x_2, y_2)|| - ||(i, j) - (x_2, y_2)||)|$$

where $||.||$ represents the euclidian distance between the two grid points and $|.|$ represents the absolute value. This cost function ensures that control costs are all non-negative and proportional to the reduction in the distance of the aircraft from its destination. If the aircraft moves closer to its destination as a result of taking a transition then the control cost for that

transition will be very high thereby discouraging blocking of that transition by control actions. Optimization algorithms given in this paper can now be used to compute an optimal route for the aircraft which minimizes the distance traveled in expectation and simultaneously avoids all the storms.

10 Related Work

A substantial body of work exists in the area of application of linear control theory to software systems. In this approach, a non-linear software system is approximated with a linear model using estimation techniques. Proportional-integral or proportional-integral-derivative controllers are then developed for such linear approximate models. Controllers for controlling the performance of web servers [3, 1, 11, 19, 28], balancing load in networks or distributed systems [35, 8], providing differentiated caching services [21], controlling congestion in networks [5], adaptive web content delivery [2] and dynamically adapting real-time systems to overloads and deadline misses [20] have all been developed. In [30] the authors use a queuing model to first predict the behavior of a network server and then use the prediction for controlling server performance. Any error in prediction is corrected by linear feedback control using a linear approximate model. All these papers develop control techniques to achieve higher level business goals like bounded response time, utilization, service rate etc. A single linear approximate model for software systems in the presence of unpredictable operating environment is difficult to achieve. The controller framework developed in this paper is significantly different from empirical design methodologies employed by these papers. Our abstraction of software as a finite state machine helps to represent the system accurately. Further, the fine-grained state space model allows us to argue about control of safety property violations in addition to achieving the higher level business goals.

In [16] the authors have modeled the network bandwidth allocation problem as a Markov decision process where optimization is in terms of a utility function. The utility function for each user assigns a reward proportional to the amount of bandwidth allocated to the user. In [23, 22] the authors have formulated an aircraft routing problem under storms as a Markov decision process. In [10] the author has described a reachability problem in Markov decision processes where an optimal policy maximizes the probability of reaching a subset of the state space. Standard algorithms for Markov decision processes like policy iteration using bellman recursion are then used to solve these optimization problems. These algorithms have poor worst case running time especially when there are exponentially many different control actions in the model. Abstracting these problems in our framework will result in improved objective functions and will also facilitate use of efficient algorithms described in this paper.

In [34] the authors have modeled an extremely restricted discrete event system using max-plus linear algebra. They have then developed model based prediction and control mechanism where the only control possible is delaying the time instant when the next input is given to the model. Our framework is capable of modeling a more general software system and also provides more generic control actions. Fuzzy control is a non-linear control methodology which uses linguistic rules for controller design. Since computing systems are non-linear, fuzzy control can be applied directly without the need to generate a linear approximate model. [12, 17] have applied fuzzy control to control QoS in web servers and network flow. Fuzzy control is a heuristic control mechanism which lacks the formal analyti-

cal strengths of optimization based techniques that we have developed. Feedback controlled software [13] using pseudo energy functions is a reactive control framework where control actions are executed on detection of deviation of the software from its goal. The framework does not deal with safety violations nor does it consider the stochasticity inherent in software systems.

Dynamic power management (DPM) [25] aims to control the power consumption of various system components based on the history of request arrivals. The paper has developed an objective function that aims to minimize the expected long run average response time of the system under constraints of bounded long run average power consumption and request losses. The DPM problem can be formulated in our framework where proper assignment of rewards and penalties to states can generate an objective function which not only minimizes the response time but also minimizes power consumption and request losses. Hence it would no longer be required to specify arbitrary bounds on power consumption and request losses that the formulation in paper [25] necessitates. Paper [4] describes the application of real-time AI techniques to a flight control system using CIRCA(Cooperative Intelligent Real-Time Control Architecture). The motion of the aircraft along with the unpredictable environment is modeled as a stochastic state machine. The paper then uses stochasticity to eliminate low probability regions in this model from being used for motion planning. CIRCA is then used on the restricted model to generate control actions ensuring prevention of failure and progress towards goal. Use of AI techniques in determining control and exploration of limited state space renders this technique sub-optimal. The paper also assumes that recovery from failure is always possible which is difficult to achieve in software systems.

Recovery oriented computing paradigm [7, 32, 33] aims to develop a software development and verification framework for developing highly available large scale Internet systems. Aspect oriented programming [15, 14] is a software design framework that assists in designing highly modularized software. It complements object oriented methodology and aims to separate concerns or aspects that crosscut multiple classes. Both these frameworks use qualitative techniques to design goal specific software and also involve substantial human understanding. Our framework develops an optimal software design that achieves the desired goals optimally under stochasticity with little human intervention. Edit automata described in [18] is a mechanism for enforcing security policies like access control, availability etc in software. Edit automata does not handle uncontrollable system events and lacks any notion of optimal control under stochasticity. Optimal control of hybrid systems described in [31] models the hybrid system as a Markov chain with continuous dynamics at each state. Control actions in the system modify the continuous dynamics at states driving the system away from bad states. Algorithms given in the paper are heuristic and optimization is done over a finite horizon. We are interested in discrete systems and provide polynomial time exact optimization algorithms for controlling such systems.

11 Conclusion

In this paper we have developed a framework based on stochastic finite state machines for abstracting software systems and their unpredictable operating environment. Safety violations and functionalities in the software are abstracted as penalties and rewards on the states, respectively. We have then formulated the stochastic control problem that aims to minimize the safety property violations and simultaneously maximize the goals achieved by the sys-

tem. Algorithms with running time polynomial in the size of the abstraction have also been developed for solving the optimization problem exactly. Any software system which can be abstracted in the given framework can be optimally controlled using the approach described in this paper.

Improving the running times of the algorithms described in this paper is one direction for future work. Statically analyzing the software system model and computing optimal control would be extremely inefficient if the abstraction is large which is true for many software systems. We are interested in exploring modifications to the framework and algorithms given in this paper for dynamic control of software. At runtime, the dynamic controller can explore a small subset of the system state space and optimally control the explored system.

References

[1] T. Abdelzaher and K. Shin. End-host architecture for qos-adaptive communication. In *In Proceedings of the Fourth IEEE Real-Time Technology and Applications Symposium*. IEEE Computer Society, 1998.

[2] Tarek F. Abdelzaher and Nina Bhatti. Web server qos management by adaptive content delivery. In *In International Workshop on Quality of Service*, 1999.

[3] Tarek F. Abdelzaher and Kang G. Shin. Qos provisioning with qcontracts in web and multimedia servers. In *In Proceedings of the 20th IEEE Real-Time Systems Symposium*. IEEE Computer Society, 1999.

[4] E. M. Atkins, E. H. Durfee, and K. G. Shin. Plan development using local probabilistic models. In *In Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence*, Aug 1996.

[5] Lotfi Benmohamed and Semyon M. Meerkov. Feedback control of congestion in packet switching networks: The case of a single congested node. *IEEE/ACM Transactions on Networks*, 1(6):693–708, 1993.

[6] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control: 2nd Edition*. Athena Scientific, Nashua, NH, 2001.

[7] Goerge Candea, Shinichi Kawamoto, Yuichi Fujiki, Greg Friedman, and Armando Fox. Microreboot - a technique for cheap recovery. In *In Proceedings of the 6th Symposium on Operating System Design and Implementation*, 2004.

[8] Valeria Cardellini, Michele Colajanni, and Philip S. Yu. Request redirection algorithms for distributed web systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(4), 2003.

[9] Thomas Cormen, Charles Leiserson, Ronald Rivest, and Clifford Stein. *Introduction to Algorithms: 2nd Edition*. McGraw-Hill Book Company, New York, NY, 2001.

[10] Beauquier D. Markov decision processes and deterministic buchi automata. *Fundamenta Informaticae*, 49:1–13, 2002.

[11] Y. Diao, Neha Gandhi, Joseph L. Hellerstein, S. Parekh, and D. M. Tilbury. Using mimo feedback control to enforce policies for interrelated metrics with application to the apache web server. In *In Proceedings of the 8th IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2002.

[12] Yixin Diao, Joseph L. Hellerstein, and Sujay Parekh. Optimizing quality of service using fuzzy control. In *In Proceedings of the 13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management*. Springer-Verlag, 2002.

[13] William B. Dunbar, Eric Klavins, and Stephen Waydo. Feedback controlled software systems. In *California Institute of Technology Technical Report: CaltechCDSTR:2003.002*, 2003.

[14] Tzilla Elrad, Mehmet Aksits, Gregor Kiczales, Karl Lieberherr, and Harold Ossher. Discussing aspects of aop. *Communications of the ACM*, 44(10), 2001.

[15] Tzilla Elrad, Robert E. Filman, and Atef Bader. Aspect-oriented programming: Introduction. *Communications of the ACM*, 44(10), 2001.

[16] Suresh Kalyanasundaram, Edwin K. P. Chong, and Ness B. Shroff. Optimal resource allocation in multi-class networks with user-specified utility functions. *Comput. Networks*, 38(5):613–630, 2002.

[17] Srinivasan Keshav. A control-theoretic approach to flow control. In *In Proceedings of the Conference on Communications Architecture & Protocols*. ACM Press, 1991.

[18] Jay Ligatti, Lujo Bauer, and David Walker. Edit automata: Enforcement mechanisms for run-time security policies. *International Journal of Information Security*, 4(1–2):2–16, 2005.

[19] Chenyang Lu, Tarek F. Abdelzaher, John A. Stankovic, and Sang H. Son. Feedback control architecture and design methodology for service delay guarantees in web servers. In *University of Virginia Technical Report: CS-2001-06*, 2001.

[20] Chenyang Lu, John A. Stankovic, Tarek F. Abdelzaher, Gang Tao, Sang H. Son, and Michael Marley. Performance specifications and metrics for adaptive real-time systems. In *In Proceedings of the IEEE Real-Time Systems Symposium*. IEEE, 2000.

[21] Ying Lu, Avneesh Saxena, and Tarek F. Abdelzaher. Differentiated caching services; a control-theoretical approach. In *In Proceedings of the 21st IEEE International Conference on Distributed Computing Systems*. IEEE, 2001.

[22] Arnab Nilim and Laurant El Ghaoui. Robust markov decision processes with uncertain transition matrices. In *UC Berkeley Technical Report: M04/26*, 2004.

[23] Arnab Nilim, Laurant El Ghaoui, Mark Hansen, and Vu Duong. Trajectory-based air traffic management (tb-atm) under weather uncertainty. In *4th USA/EUROPE ATM Research and Development Seminar*, 2001.

[24] J. R. Norris. *Markov Chains*. Cambridge University Press, Cambridge, New York, 1998.

[25] G. A. Paleologo, L. Benini, A. Bogliolo, and G. De Micheli. Policy optimization for dynamic power management. In *In Proceedings of the 35th annual conference on Design automation*. ACM Press, 1998.

[26] P. Pradhan, R. Tewari, S. Sahu, C. Chandra, and P. Shenoy. An observation-based approach towards self-managing web servers. In *10th International Workshop on Quality of Service*, 2002.

[27] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons., New York, 1994.

[28] Anders Robertsson, Bjom Wittenmark, and Maria Kihl. Analysis and design of admission control in web-server systems. In *In Proceedings of the IEEE American Control Conference*. IEEE, 2003.

[29] E. Seneta. *Non-negative Matrices and Markov Chains: 2nd Edition*. Springer Verilag, 1981.

[30] Lui Sha, Xue Liu, Ying Lu, and Tarek Abdelzaher. Queueing model based network server performance control. In *In Proceedings of the 23rd IEEE Real-Time Systems Symposium*. IEEE Computer Society, 2002.

[31] Ling Shi, Alessandro Abate, and Shankar Sastry. Optimal control for a class of stochastic hybrid systems. In *In Proceedings of the 43rd IEEE Conference on Decision and Control*. IEEE, Dec 2004.

[32] Michael M. Swift, Muthukaruppan Annamalai, Brian N. Bershad, and Henry M. Levy. Recovering device drivers. In *In Proceedings of the 6th Symposium on Operating System Design and Implementation*, 2004.

[33] Stanford University and Univ. of California at Berkeley. Recovery oriented computing. In *http://roc.cs.berkeley.edu/*.

[34] T.J.J. van den Boom, B. De Schutter, G. Schullerus, and V. Krebs. Adaptive model predictive control for max-plus-linear discrete event input-output systems. *IEE Proceedings – Control Theory and Applications*, 151(3):339–346, 2004.

[35] L. Zhang, Z. Zhao, Y. Shu, L. Wang, and O. W. W. Yang. Load balancing of multipath source routing in ad hoc networks. In *In Proceedings of IEEE International Conference on Communications*. IEEE, 2002.