Reconfiguring Distributed Systems using Markov-Decision Models

Leonard J.N. Franken
KPN Research, P.O. Box 15000
9700 CD Groningen, The Netherlands

Boudewijn R.Haverkort
RWTH Aachen, Distributed Systems Lab.
D-52056 Aachen, Germany

August 5, 1996

Abstract

When reconfiguring distributed systems, either because failures and/or repairs have occurred, or because the workload requirements have changed over time, an alternative configuration has to be selected, thereby taking performability and costs, i.e., quality of service, requirements into consideration. In this paper we will present the use of Markov-decision models to decide upon such reconfigurations. This seems to be a promising approach when the number of possible alternative configurations is limited. In that case, reconfigurations can be prepared well, so that the actual decision process during the real-system operation can be performed pretty fast.

1 Introduction

Quality of service management tries to match the quality of service offered by a system with the user required quality of service. This naturally depends on the *level* of quality of service *guarantee* required by the user. These levels can differ from deterministic, to probabilistic, to, at the lower end, best effort []. The internet service is an example of a best effort service; there is no effective guaranteed quality of service. Deterministic guarantees require that the quality of service level is met at each instance of time, whereas the probabilistic level requires, for example, that 95% of the time the quality of service requirements are met. Because the (total) required quality of service will not remain the same over a period of time, for example, since the number of users might change during the day, it might be efficient and effective to adapt the offered quality of service by a system over the day to the required quality of service. Th! ese adaptions can be realised using a dynamic reconfiguration algorithm; in our case such an algorithm is envisaged as part of the so-called performability manager. Note that system configuration adaptions can be the result of changing user demands but also of changes in the offered quality of service caused by system failures. We focus however on the changing user demands, i.e., changes in the required quality of service.

The measure we use to characterise the quality of service is the performability measure. The relation between performability and quality of service has been expressed earlier by Meyer [9]. In this paper we will use the notion of quality of service and performability to express user requirements. Note that the costs to realise and maintain the required quality of service will be expressed as a separate measure, however, one could argue that costs are also a quality aspect.

This paper is further organized as follows. In Section 2 we summarize our previous work in which the current apporach needs to be embedded. In Section 3 we then give a concise overview of the theory of Markov decision models which we will use in our reconfiguartion process. An application of a Markov decision model in the context of intelligent networking is discussed in Section 4. Finally, we conclude this paper in Section 5.

2 Performability management

We have recently proposed a five-step method for quality of service maintenance by a distributed-system component called the performability manager (see [3]). This management component should become active (and thereby possibly initiate a reconfiguration) once the offered quality of service by the application(s) is not fulfilling the user required QoS, either due to the occurrence failures, or due to changes in workload.

Once triggered, i.e., a decrease in actual performability is perceived (measured) (step 0), the performability manager establishes a reconfiguration in four steps:

1. Creation of alternative configuration(s);
2. Evaluation of the alternative configuration(s);

3. Selection of a configuration;

4. Performing the reconfiguration.

The creation of an alternative configuration encompasses the listing of alternative configurations that might be possible from a pure functional point of view, i.e., can certain process be migrated to other machines? or can certain communications be rerouted? The performability manager can create *one* alternative configuration or a *set* of alternatives, respectively denoted a *proposing* strategy and a *generating* strategy.

With the proposing strategy only one (alternative) configuration is proposed. Proposing algorithms are designed to create one optimal configuration (with respect to some objective function). If the proposed alternative configuration fulfils the requirements, also from a performability point of view, we put it into effect. If not, another alternative configuration needs to be created and evaluated. With the proposing strategy steps 1, 2 and 3 are iteratively dealt with, until an acceptable configuration is found.

Conversely, using the generating strategy a *set* of alternative configurations is generated (step 1) at each trigger. These alternative configurations are evaluated and their performabililty is determined in step 2. After the evaluation we have to select a configuration that fulfils the required quality of service in step 3. For both policies step 4 is the reconfiguration itself.

In this paper, we also discuss the selection of a new configuration from a set of created configurations. Although this does resemble the generating strategy, there are two important differences. With the performability manager, we have assumed, until now, that a trigger for a reconfiguration occurs at random points in time, i.e., we do not have an indication when these triggers will occur. However, if we can predict when certain triggers will occur, we might be able to prepare a reconfiguration strategy, so that the creation of alternative configurations and their evaluation can be done *a priori*. Although this might seem artificial, this situation occurs often in a telecommunication system context where workloads change periodically (with period 1 day), and where planned preventive repair activities alter the system capacity.

We therefore assume that we are able to predict when the triggers occur, and that we are able to create and evaluate the alternative configurations in advance. Then, based on this knowledge, we can prepare a policy according to which the reconfigurations can take place. In order to do so, we employ Markov decision models [1, 14].

3 Markov decision models

In Section 3.1 we give a global overview of Markov decision theory. Then, in Section 3.2 we present an overview of applications of Markov-decision theory in the context of computer and communication system configuration.

3.1 Basic theory

With Markov decision models the decision for a new configuration is based on the current configuration and quality of service , i.e., the usual *Markov property* is valid [13]:

> The future evolution of the system depends only on the current state of the system and not on its (past) history.

Note that in this definition the notion of states is used instead of configurations. The actual decision model is built on these states and the possible transitions between them.

In each state a trigger for a reconfiguration can occur if the achieved and/or required quality of service change(s). In both cases the required quality of service cannot be achieved by the current configuration. Note that if the required quality of service decreases then a reconfiguration can be required in order to reduce the cost for the offered quality of service, i.e., the operational configuration. We call the moments of triggering *decision epochs*. At a decision epoch, an action (in the current state) initiates a transition to a next state. Furthermore, we assume that in between two decision epochs all the quality of service contracts agreed upon are met. At each decision epoch we have to decide which action to perform in order to pass into the next state.

The time between decision epochs, or triggers, can be constant. This results in discrete-time Markov decision models. However, if the times between decision epochs are not constant but random, we can use semi-Markov decision models. Note that under specific conditions we can use the discrete-time model for continues processes. In order to do so, we must be able to discretise the process to formulate the continues problem as a discrete one [18].

Another aspect is the determination of the actions. The required quality of service, for example, can change deterministically or stochastically. A deterministically changing required quality of service implies that at such a change the quality of service for the next stage is required with a probability of 1. So, the action to deal with the changing quality of service can also be selected deterministically. However, if the

quality of service is changing stochastically, then the actions are also stochastically determined.

In combination with the trigger behaviour we can distinguish between these situations using the notion of deterministic and stochastic discrete-time and deterministic and stochastic continuous-time models.

Let us now describe the Markov decision models that are used to determine the reconfiguration policy. For our decision models we assume a finite state space I and a finite set of actions U. If the model is in state i and action r is taken then a state change to a state j takes place with probability $p_{ij}(r_i)$, with $i, j \in I$ and $r_i \in U$. As with Markov reward models, we can assign a reward (or cost) $c_i(r_i)$ to the action r_i taken in state i. For the state transitions a policy or rule R is required that decides what the next state will be using the information of the current state and the required quality of service. More formally, a policy is a set of decision rules describing what action to take in a certain state at a decision epoch.

Markov *decision theory* is concerned with finding an optimal reward from the system. In our case the reward gained is the quality of service offered by the system or costs incurred to realise that quality of service The reward gained from the system can be optimised by creating an optimal reconfiguration policy with respect to the defined rewards. This can be realised by selecting an optimal action with respect to the reward in each state. In doing this we can, for example, in the case of a finite time horizon maximise the total reward and, for example, maximise the expected reward per time unit over a infinite time horizon. The time horizon can be associated with the mission time of a service.

In keeping with the Markovian assumptions, i.e., the decision for the next state only depends on the current state of the system, and if we assume that the planning horizon is infinitely long, then we consider stationary policies. A stationary policy R is a rule that always prescribes a single action r_i whenever the system is found in state i at a decision epoch. More formal, we definefor $n = 0, 1, \cdots$ (see [15]):

$$X_n = \text{system state at the } n\text{-th decision epoch.}$$

Under a given stationary policy R, we have

$$\Pr\{X_{n+1} = j | X_n = i\} = p_{ij}(r_i), \text{with } r_i \in U,$$

regardless of the past history of the system up to time n. Hence, under a given stationary policy R the stochastic process $\{X_n, n \geq 0\}$ is a Markov chain with one-step transition probabilities $p_{ij}(r_i)$. This Markov chain incurs a cost $c_i(r_i)$ each time the system visits its state i. Thus we can invoke results from Markov chain theory to specify the long run average cost per unit time under a given stationary policy.

The case we discuss has a periodicity in the demand, a one day period, i.e., each day the pattern repeats itself for an infinitely long period. For the stationary policy the quality of service requirements must be identical in each period. Hence, all costs associated with the actions and the obtained R remain the same in each period.

We shall now discuss how to obtain the average cost or gain per period. The average gain per period can be derived using the steady state probabilities of the Markov chain. We assume that each Markov chain only contains one ergodic subset of states [15]. Then, we can associate a long term average cost per period by:

$$g(R) = \sum_{i \in I} c_i(r_i)\pi_i(R),$$

where c_i is the one period cost for starting in state i associated with action r_i of strategy R, and π_i is the steady state probability associated with state i of the created Markov chain using policy R. However, we study systems that can be ruled by a number of alternative policies, i.e., different R, each defining its own Markov chain and resulting in a uniquely long-run average cost per period. Out of these alternative policies we want to select an optimal policies. Since the number of policies is M^N, if we have N states and M actions per state, it is (often) not feasible to create all possible policies and select the optimal one. However, an efficient algorithm, i.e., the *policy iteration algorithm*, has been reported by Howard [7] that constructs a sequence of (improved) policies until an average cost optimal policy is found [1, 14, 15].

We now present an intuitive explanation of the steps in the policy iteration algorithm. Step 0 is the determination of an initial rule R; this rule is merely an educated guess for an action $r_i \in U$ at each decision epoch.

In step 1 rule R is used to obtain the long-run average cost function $g(R)$ and the relative cost $v_i(R)$ belonging to each state $i \in I$. This relative value $v_i(R)$ is the increase in cost for the system in state i taking action r_i of rule R instead of starting in state j. Note that a different action in that state would result in different relative cost. $g(R)$ and $v_i(R)$ can be obtained simultaneously by solving the set of linear equations created by rule R and Equation 1 as given below. This set of linear equations has a unique solution if we set the relative cost of one state to zero,

i.e., $v_s = 0$ where s is an arbitrarily chosen state (this is also why we obtain the relative cost for starting in state i instead of j, and not the absolute cost).

Step 2 is the policy improvement step. For each state i in the Markov decision model and every action $r_i \in R$ the relative costs v_i are determined. With the policy improvement step we search for the action $r_j \in U$, but $r_j \notin R$, that results in lower relative costs of starting in state i instead of j. If the relative costs found in this step are less then the costs found with rule R and the action used is not in rule R the rule is updated with the "cheaper" action. With this step a new rule $\widehat{R}$ can be created by selecting these "cheaper" actions. Note that if the newly chosen action yields the same cost, the old action will still be used. In step 3 a comparison is made between the old the and new rule, i.e., between $\widehat{R}$ and R. If they do not differ the iteration is stopped, but if they still differ, another iteration has to be made. In the latter case the algorithm is resumed in step 2 (after having set $R := \widehat{R}$.

In a more compact notation the policy iteration algorithm can be described as follows:

Step 0: initialisation. Choose a stationary rule R.

Step1: value determination step. For the current rule R, compute the unique solution $\{g(R), v_i(R)\}$ to the following system of linear equations (for $i \in I, r_i \in R, v_s = 0$):

$$v_i = c_i(r_i) - g(R) + \sum_{j \in I} p_{i,j}(r_i) v_j \,, \qquad (1)$$

where s is an arbitrarily chosen state.

Step 2: policy-improvement step. For each state $i \in I$, determine an action r_i yielding the minimum in

$$\min_{r_i \in U} \{c_i(r_i) + \sum_{j \in I} p_{i,j}(r_i) v_j\}. \qquad (2)$$

The new stationary rule $\widehat{R}$ is obtained by choosing $\widehat{R} = r_i$ for all $i \in I$ with the convention that $\widehat{R}$ is chosen equal to the old action R when this action yields the minimum in (2)

Step3: convergence test. If the new rule $\widehat{R}$ equals the old rule R, the algorithm is stopped with rule R. Otherwise, go step 1 with R replaced by $\widehat{R}$.

For more background related to the Markov decision models we advise the interested readers the textbooks of Tijms [14, 15] and Daellenbach *et al.* [1] and Ross [11].

3.2 Literature overview

In this section we present a short overview of the literature on the application of Markov decision models in the context of computer-communication systems (another overview is given by White [17]).

Winston describes a maintenance system consisting of a finite set of machines and a single repair facility that may operate at several rates [18]. Machines may fail and are subsequently sent to the repair facility. Discrete-time Markov decision models are used to prove that, under the assumption that costs depend on the repair rate and lost production, the optimal repair rate is a non-increasing function of the number of machines in good condition. A continuous-time maintenance system is considered as a limit of a sequence of discrete-time maintenance systems. For the continuous-time case it is also proven that the optimal repair rate is a non-increasing function of the number of machines in good condition. Note that optimal in both cases is optimal with respect to the cost of the repair rate and lost production, in other words, a minimisation of the repair rate and lost production.

Similar cases as discussed by Winston are the discrete-time Markov maintenance models used to control the queues of repair facilities discussed by Hatoyama [6]. Hatoyama does not discuss the repair rate as Winston, but does study the options for opening and closing repair facilities, when there are machines waiting for repair.

Wartenhorst, in his Ph.D. thesis [16], discusses a similar case to that discussed by Winston. The difference with the work of Winston is that Wartenhorst allows the failure and repair times to be generally distributed, where Winston assumes exponentially distributed failure and repair times. Wartenhorst also allows the repair rate to depend on both the number of failed machines and on the amount of work to be carried out on the unit that is under repair.

Different from the above described work, is the work of Shin *et al.* [12], in which besides repair, dynamic reconfiguration is used for optimal system maintenance or resource control. A resource control decision is needed whenever there is significant change by either component failures or workload changes. The reward (cost) function that is optimised for resource control is based on Meyer's performability measure [9, 10]. The type of models used for optimisation are semi-Markov decision models.

De Meer, in his Ph.D. thesis [8], deals with similar cases as Shin *et al.*. However, De Meer extensively discusses the use of performability measures in relation to Markov decision theory and dynamic reconfiguration.

The work by Shin *et al.* and by De Meer forms the basis of our work. Our aplication are in the area of telecommunication systems. However, we also experiment with available capacity and changing workload. We merely want to point at the applicability of markov decision models in the area of QoS management.

4 Reconfiguring intelligent networks

In this section we describe how to obtain a reconfiguration policy for a so-called number translation service (NTS) in telecommunications systems, using Markov decision models in order to meet a periodically changing required quality of service.

The NTS is an example of a value-added service; it can be provided by telecom-operators using concepts knowns from the area of intelligent networking (see also [5, 4]). An example of the usage of the NTS appears when calling toll-free numbers, or when calling mobile phones. The local telephone switch will recognise the speciality of the number being dialled, and forward it to an evaluation component, typically a database system, where the dialled number is translated to another number that is used internally. While performing this translation, other administrative date might be changed in the database as well, e.g., data related to billing and statistical data. The former also is of importance when employing the service known as 'credit-card calling'.

Since telecommunication networks consist of many switches, telecom-operators normally have some freedom where to locate particular components that together constitute the value-added services. In the above example, it might be such that there is only one database server, or more than one, each working on a particular subset of telephone numbers (distribution), or each working on all telephone numbers (replication). Depending on whether the application requires only read- or also write-queries on the database, one or another solution might be more cost-effective. In this trade-off, also the actual usage pattern, i.e., the workload, of the services plays a role.

Let us now suppose that the overall workload requirements of all NTS end-users is changing deterministically and that the changes have a 24-hour period. In Figure 1 we depict on such a varying workload. On the X-axis 6 stages of 4 hours are distinguished. For each stage the average number of expected calls per second is depicted on the Y-axis. We assume that all users require the same (quaranteed) quality of service in each stage, but we have more or less users and thus calls at each stage. In this study quality of service is expressed as average response time. In order to meet the quality of service requirements, we must install more or less capacity at each stage. In this application, it is assumed that the NTS can operate in one of three configurations [2, 4, Chapter 7], i.e., Ω_1, Ω_2 or Ω_3, each offering a different capacity at different costs. With the first configuration, Ω_1, we use two systems. A system to which the database is allocated and a separate system for the actual NTS application. The second configuration, Ω_2, just uses one system (a centralised solution) which has the database and the NTS application allocated to it. The third configuration, Ω_3, is a combination of Ω_1 and Ω_2, it has one application and uses two (identical) databases. The three configurations differ in availability and capacity of the database. At first sight the third configuration should have the highest capacity, but because of an inefficiency in the application code and the limitations of the software environment in which the application is realised, its capacity is less then that of the second but higher than the fist configuration. the availability only differs with respect to the database, the application is a single point of failure with all three configurations. In this study focus on the capacity issues. Besides installing the required capacity in each stage, we also want to reduce the average cost per unit of time. Note that the installed over-capacity during each stage is lost.

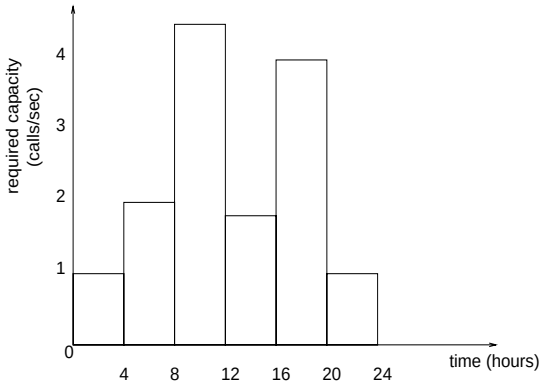

Figure 1: The required load per stage

The process of changing configurations is modelled as a discrete-time Markov decision model as depicted in Figure 2. The states of the decision model are determined by the current stage and the configuration used. The stages are derived from the changing capacity requirements, which, at equidistant points in time, i.e., every 4 hours, result in 6 stages $K = \{1, \cdots, 6\}$. The configurations used are $\Omega = \{\Omega_1, \Omega_2, \Omega_3\}$. This results in the two dimensional discrete state space $I = K \times \Omega$

.

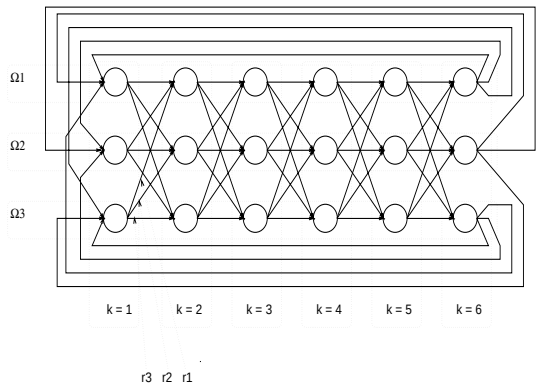

Figure 2: The state space belonging to the NTS

At the end of each stage a reconfiguration can take place, i.e., we have a decision epoch. These are the arrows between the different stages. From the current configuration the system can reconfigure to one of the other two configurations or remain in the current configuration, when going to the next stage. The action for a particular decision epoch depends on the current state and the required quality of service for the next stage. We denote three possible actions by:

$$r = \begin{cases} r_1, & \text{if } \Omega_1 \text{ will be used in the next stage,} \\ r_2, & \text{if } \Omega_2 \text{ will be used in the next stage,} \\ r_3, & \text{if } \Omega_3 \text{ will be used in the next stage.} \end{cases}$$

Note that an action r_j, with $j = 1, \ldots, 3$, is *possible* in the current stage k if the configuration proposed by that action offers the required capacity for the next stage $k \oplus 1$, i.e., if the required capacity for stage $k \oplus 1$ is smaller than or equal to the offered capacity by the state after action r_i. More formally,

$$Q_{\text{req}}^{k \oplus 1} \leq Q_{\text{off}}^{\Omega_x},$$

where $Q_{\text{req}}^{k \oplus 1}$ denotes the *required* capacity in stage $k \oplus 1$ and $Q_{\text{off}}^{\Omega_l}$ denoted the *offered* capacity by configuration Ω_l with $l = 1, \ldots, 3$.

The values of the quality of service parameters of the Markov decision model result from both monitoring results as discussed in [4] and [2, Chapter 7]. The required capacities and cost for each configuration and reconfiguration are educated gueses. The offered capacities are measurement results from real experimental applications. For the NTS we have the following data concerning the required capacities Q_{req}^{k} in each stage:

$$\begin{array}{lll} Q_{\text{req}}^{1} = 1.0, & Q_{\text{req}}^{2} = 2.0, & Q_{\text{req}}^{3} = 4.4, \\ Q_{\text{req}}^{4} = 1.8, & Q_{\text{req}}^{5} = 4.0, & Q_{\text{req}}^{6} = 1.0, \end{array}$$

and the offered capacities by the different configurations $Q_{\text{off}}^{\Omega_l}$:

$$Q_{\text{off}}^{\Omega_1} = 1.8, \quad Q_{\text{off}}^{\Omega_2} = 4.4, \quad Q_{\text{off}}^{\Omega_3} = 3.6.$$

At each decision epoch the immediate costs $c_{(k,\Omega_x)}(r_j)$ are incurred, i.e., the cost associated with the action r_j chosen in state (k, Ω_x). These costs are the sum of the reconfiguration cost, i.e., r_Ω and the operating costs of the configuration used during the next stage, i.e., o_{Ω_x}. The reconfiguration costs are constant, however, these can be chosen to be dependent on the current configuration and the next configuration. Note that if the configuration does not change, no reconfiguration costs are incurred for the action chosen, although, the operation costs remain. Thus, the one step costs incurred in each state at the decision epoch are:

$$\begin{cases} c_{(k,\Omega_1)}(r_1) = o_{\Omega_1}, \\ c_{(k,\Omega_1)}(r_2) = o_{\Omega_2} + r_\Omega, \\ c_{(k,\Omega_1)}(r_3) = o_{\Omega_3} + r_\Omega, \\ c_{(k,\Omega_2)}(r_1) = o_{\Omega_1} + r_\Omega, \\ c_{(k,\Omega_2)}(r_2) = o_{\Omega_2}, \\ c_{(k,\Omega_2)}(r_3) = o_{\Omega_3} + r_\Omega, \\ c_{(k,\Omega_3)}(r_1) = o_{\Omega_1} + r_\Omega, \\ c_{(k,\Omega_3)}(r_2) = o_{\Omega_2} + r_\Omega, \\ c_{(k,\Omega_3)}(r_3) = o_{\Omega_3}. \end{cases}$$

The operating costs and the reconfiguration cost are also educated guesses. The *operating* costs are based on the number of resources and application components in use for the particular configuration:

$$o_{\Omega_1} = 4 \quad o_{\Omega_2} = 5 \quad o_{\Omega_3} = 7.$$

The costs for the *reconfigurations* are assumed to be constant: $r_\Omega = 1$.

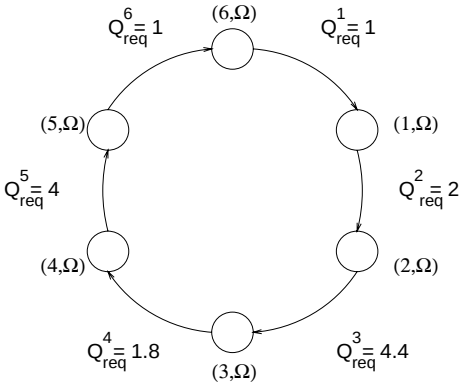

Figure 3: The state transitions of the NTS

The set of possible actions for the NTS, considering that $Q_{\text{req}}^{k\oplus 1} \leq Q_{\text{off}}^{\Omega_x}$, are:

stage	1	2	3	4	5	6
actions	2, 3	2	1, 2, 3	2	1, 2, 3	1, 2, 3

For example, in stage 4 we can only take action 2, which results in configuration Ω_2 for stage 5. Based on these possible actions we have the resulting Markov decision model as depicted in Figure 4, the arcs signifying "illegal" transitions are simply removed. This model is used for the search for the optimal policy.

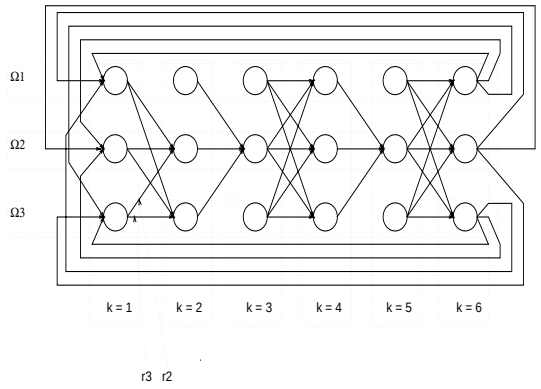

Figure 4: The possible actions taking into account the required capacity

Because of periodic and deterministic demands we will observe that after a number of transient states the model will step through a set of recurrent states. The one step transition probabilities $p_{(k,\Omega_x),j}(r_j)$, with $j \in I$ and $r_j \in U$, for this particular Markov chain are given by (with $k = 1, \cdots, 6$):

$$p_{(k,\Omega_x),j}(r_j) = \begin{cases} 1, & \text{if } j = (k \oplus 1, r_j) \ , \\ 0, & \text{otherwise.} \end{cases} \qquad (3)$$

With the NTS we are dealing with *deterministic state transitions*, i.e., each next state is chosen with a probability of 1. The recurrent states are the states selected by the one-step probabilities, the other states can only occur in an initial situation and are the transient states. We depicted the above in Figure 3.

At this moment we are ready to apply the policy iteration algorithm as described in Section 3. For the NTS we selected the following stationary initial rule $R_{initial}$:

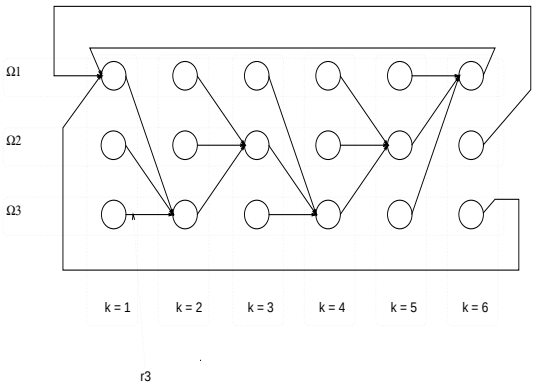

Figure 5: The state transitions belonging to the initial rule

		stage k					
		1	2	3	4	5	6
Ω	1	3	2	3	2	1	1
	2	3	2	3	2	1	1
	3	3	2	3	2	1	1

The state transitions belonging to this rule are depicted in Figure 5. Using the algorithm we created, after three iterations, the following *final* rule R_{final}:

		stage k					
		1	2	3	4	5	6
Ω	1	2	2	1	2	1	1
	2	2	2	2	2	1	1
	3	2	2	1	2	1	1

The average cost per stage for the initial rule is $g(R_{initial}) = 6.167$, then after three iterations the average cost per stage reduced to $g(R_{final}) = 5.0$ for the final rule. The final rule is illustrated in Figure 6. This rule gives the optimal average costs per stage for the NTS, with respect to the long term behaviour, when the workload is changing as described. Note that for this policy the actions in some initial (transient) states are different. These actions can be different because the initial states do not contribute to the long-run average cost per stage [15].

5 Conclusions

In this paper, we have shown the applicability of Markov decision theory for quality of service management in distributed systems. We do realise that we conducted only a simple case study. The described NTS is controlled using a rule that prescribes an action for each state at each decision epoch.

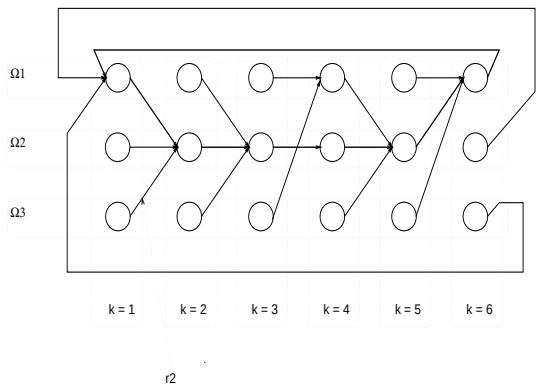

Figure 6: The state transitions belonging to the final reconfiguration rule

The rule derived in this paper is based on the optimisation of an average reward per time unit, i.e., the average reward per time unit of the environment in steady state. The type of reward gives a global optimal gain with respect to costs. In doing so, local non-optimal states can occur. For example, in the 4-th stage of the NTS example we installed more capacity than required, which results in more costs for that one stage. However, the overall costs were minimised because of the avoidance of costly reconfigurations for that particular stage.

When we want to prepare reconfigurations for failure events as well the decision models will get more complex. Failures occur at random points in time and not at deterministic points in time as we could assume for the NTS workload changes. In order to obtain a reconfiguration policy in such cases we need to employ semi-Markov decision models.

Note that if events occur that result in a quality of service change and that have not been accounted for when preparing the rule, one still needs the performability manager (with its five-step approach as indicated in the introduction) to take the appropriate action.

As future work we envisage, among others, the application of Markov-decision theory also to cases where the periods between Quality of service mismatches are not deterministically known *a priori*, so that we can also cope with capacity changes due to randomly occurring failures and repairs in the system.

References

[1] H.G. Daellenbach, J.A. George, and D.C. McNickle. *Introduction to Operations Research Techniques*. Alyn and Bacon, Inc, 1978.

[2] L.J.N. Franken. *Quality of Service Management: a Model-Based Approach*. PhD thesis, University of Twente, the Netherlands, 1996.

[3] L.J.N. Franken and B.R.H.M. Haverkort. The Performability Manager. *IEEE Network*, 8(1):24–32, January 1994.

[4] L.J.N. Franken, R.H. Pijpers, and B.R. Haverkort. Modelling Aspects of Model Based Dynamic QoS Management by the Performability Manager. In G. Haring and G. Kotsis, editors, *Computer Performance Evaluation. Modelling Techniques and Tools. Proceedings of the 7th International Conference, Vienna, Austria*, pages 89–110. Lecture Notes in Computer Science, Volume 794, Springer-Verlag, May 1994.

[5] J.J. Garrahan, P.A. Russo, K. Kitami, and R. Kung. Intelligent Network Overview. *IEEE Communications Magazine: Feature Topic: Toward The Global Intelligent Network*, 31(3):30–38, March 1993.

[6] Y. Hatoyama. Markov Maintenance Models with Control of Queue. *Journal of the Operations Research*, 20(3):164–181, 1977.

[7] R.A. Howard. *Dynamic Programming and Markov Processes*. Wiley, New York, 1960.

[8] H.de Meer. *Transiente Leistingsbewertung und Optimierung rekonfigurierbarer fehlertoleranter Rechensysteme*. PhD thesis, FA University Erlangen-Neurnberg, Erlangen, 1992. *Arbeitsberichte des IMMD*, Vol.25, No. 10.

[9] J.F. Meyer. Performability Evaluation of Telecommunication Networks. In Network Teletraffic Science for Cost-Effective Systems and ITC-12 Services, editors, *M. Bonatti*, pages 1163–1172. IAC, Elsevier Science Publishers B.V. (North Holland), 1989.

[10] J.F. Meyer. Performability: a Retrospective and some Pointers to the Future. *Performance evaluation*, 14(3-4):139–156, February 1992.

[11] J.S. Ross. Average CostSemi-Markov Descision Processes. *Journal of Applied Probability*, pages 649–656, 1970.

[12] K.G. Shin, C.M. Krishna, and Y. Lee. Optimal Dynamic Control of Resources in a Distributed System. *IEEE Transactions on Software Engineering*, 15(10):1188–1197, October 1989.

[13] W.J. Stewart. *Introduction to the Numerical Solution of Markov Chains.* Princeton University Press, 1994.

[14] H.C. Tijms. *Stochastic Modelling and Analysis: a Computational Approach.* John Wiley & Sons, 1986.

[15] H.C. Tijms. *Stochastic Models: an Algorithmic Approach.* John Wiley & Sons, 1994.

[16] P. Wartenhorst. *Performance Analysis of Repairable Systems.* PhD thesis, Katholieke Universiteit Brabant, Tilburg, the Netherlands, 1993.

[17] D.J. White. Real Applications of Markov Decision Processes. *Interfaces*, 15(6):73–83, 1985.

[18] W. Winston. Optimal Control of Discrete and Continuous Time Maintenance Systems with Variable Service Rates. *Operations Research*, 25(2):259–268, 1977.