Model-based decision framework for autonomous application migration

Nickelsen, Anders; Olsen, Rasmus Løvenstein; Schwefel, Hans-Peter

Published in:
Lecture Notes in Computer Science

DOI (link to publication from Publisher):
10.1007/978-3-642-21713-5_5

Publication date:
2011

Document Version
Early version, also known as pre-print

Model-based decision framework for autonomous application migration

Anders Nickelsen[1], Rasmus L. Olsen[1], and Hans-Peter Schwefel[12]

[1] Dept. of Electronic Systems, Aalborg University, Denmark,
{an|rlo|hps}@es.aau.dk
[2] FTW, Vienna, Austria, schwefel@ftw.at

Abstract. The state of the run-time environment impacts the performance of networked applications and furthermore influences the quality of the user experience of the applications. By *migrating* the application between the user's devices that give different experiences, a good overall user experience can be maintained, without loosing the application session. It is non-trivial for a system to automatically choose the best device for migration. The choice must maximize the experience quality and take into account that a migration delays the user's work-flow and even may fail. Moreover, the environment state is not directly observable, and needs to be estimated which leads to inaccuracy. We model the automatic migration trigger as a stochastic optimization problem and we propose to use a hidden Markov model combined with a Markov Decision Process (MDP) to solve the problem. The solution generates policies to choose target device for migration that gives the optimal user experience when enforced in simulated scenarios. We derive conclusions on which scenarios the model-based approach performs better than a greedy approach, also when considering inaccurate state estimation.

Keywords-migratory applications; configuration choice; Markov decision processes; quality-of-experience

1 Introduction

The state of the run-time environment impacts the performance of networked applications and furthermore influences the quality of the user experience of the applications. For example, if excessive packet loss on a wireless connection or in a network router reduces the end-to-end throughput, a video streaming application may experience glitches or outage periods. In addition, users have access to many different types of devices with network access that can run the same applications. Therefore, we investigate methods to move active application instances between devices to maintain a good user experience in scenarios with dynamic networks. The process of moving the application is called *migration* [1]. We consider migratory applications that can be moved between the user's devices. The combination of an application running on a device and the settings of the device (CPU, network interface, display resolution, etc.) is called a *configuration*. A

configuration gives the user a certain experience quality, influenced by the state of the environment. A migratory application has several available configurations and the objective of the migration is to change the active configuration. Migration is performed without restart of the application session, allowing the user to continue the work-flow after migration.

Migratory applications run on top of a migration middleware (such as described in [2]). The purpose of the middleware is to trigger migration automatically and orchestrate the migration procedure. The orchestration procedure takes time which interrupts the user's work-flow and degrades the experience quality. The duration of the migration procedure depends on the migration strength. In a strong migration both application code and state is transferred, where in a weak migration only parts of the application stack are transferred [1]. We consider the migration procedure as atomic and our models are independent of migration strength. Different strengths only produce different model parameters values in terms of migration delay distribution and failure probability. The automatic migration trigger must balance the trade-off between configurations with high experience quality and the reduced experience quality caused by the orchestration procedure. The triggers are based on the environment state. However, the true state is not always observable and must be estimated from observable parameters. If state estimation is inaccurate, wrong migrations are triggered, which interrupt the user and may end up in worse configurations.

To balance the trade-off, we treat the automatic trigger as a stochastic optimization problem. We model the environment behavior, the state estimation, the application experience behavior and the migration procedure as stochastic processes. The stochastic models are used to generate policies that specify in which state to trigger migration and which configuration to migrate to. We evaluate two different policy generation approaches: 1) a greedy approach that optimizes the user experience in the next evaluation step and 2) a Markov Decision Process (MDP) approach which can account for the effect of future decisions and optimizes for the long-term user experience. We design an enforcement framework around the policies with functions to observe the environment, estimate the environment state and trigger and perform migration. Based on a simulated migration scenario, we evaluate the effect on the experience quality of: 1) the estimation approach, 2) the policy generation approach and 3) the environment behavior. Through result analysis we illustrate the usefulness of including future decisions and the impact of inaccurate environment state estimation.

The concept of migration upon environment state changes is increasingly important. With the rise of *dynamic adaptive systems* (component-based systems which are able to change their composition, [3,4,5,6,7]), applications may be deployed in environments for which they were not originally designed. The application domain can be expanded if they have underlying support to choose the optimal configuration for such unknown environments. The optimal configuration is also relevant in dynamic applications that interact with users [8] since the success of these applications depends on the user's experience. Previous research has shown that model-based policy generation approaches allow for

robust decisions; video stream application stream decoding [9], placement policy of processing tasks in virtual environments [10] and QoS management policies [11]. The quality of the user's experience is modeled by reward functions that map physical properties into experience quality. Methods to derive reward functions are covered in detail in existing literature. For instance mapping network properties into quality of multi-media applications can be studied in [12,13,14] and is not treated further here.

The rest of the paper is structured as follows. In section 2, we present background details of the migration process. In section 3, we describe a model of the migratory system to base the decision framework design on, which is presented in section 4. Then, in section 5, we describe the analysis of the dependencies between the framework parameters and the performance of the policy generation approaches. The paper is concluded in section 6.

2 Background on the migration process

The events of a migration process between two devices are illustrated in Figure 1. The migration framework is realized as a middleware that contains a decision framework that monitors the state of the environment and automatically determines the best configuration to use (see [2] for more details). If the chosen configuration is not equal to the one currently applied, a migration is triggered. The application instance on the source device is paused and a state object, with sufficient information to resume the application elsewhere, is extracted and downloaded onto a central migration server. As the download may be performed over any available network connection, the performance of the network connection affects the duration of the download, and hence, the duration of the migration. Also, any faults that can occur on the network connection may cause the download or migration signaling to fail. After the state is downloaded, a new application instance is initialized on the target device. The state object is then uploaded to the target device and inserted into the new application instance. To complete migration, the new instance is resumed in the original state.

3 Model of the migratory framework

A model of the migratory framework is illustrated in Figure 2. We specify the environment as a set of N states, $\mathcal{S} = \{s_1, ..., s_N\}$. The states are abstractions of the environment parameters, which are considered relevant if they impact the performance of the application or the migration process. We model the behavior of the environment as a Markov model. $P(s_i, s_j)$ denotes the probability of a transition from state s_i at time t to state s_j at time $t + 1$. The next configuration of the application is decided periodically by the migration middleware. A set of M configurations, $\mathcal{C} = \{c_1, ..., c_M\}$, constitutes the set of possible next configurations. The middleware has to decide which configuration c' to

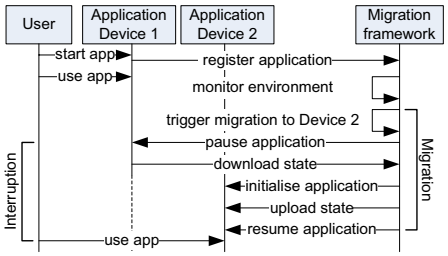


Fig. 1. Migration process events.

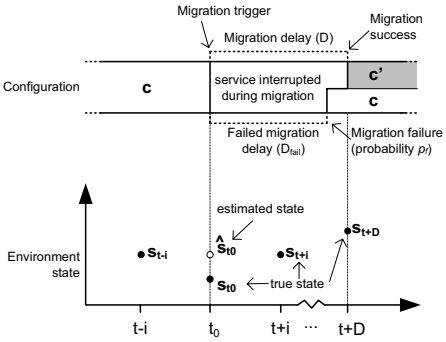


Fig. 2. Time-line migration process.

change to from the current configuration c. If $c \neq c'$, a migration $c \rightarrow c'$, is attempted. The migration duration in time-units, D, is random and characterized by a distribution $f_{D,c\rightarrow c'}(\tau)$ that depends on the size of the state object that needs to be transferred and the environment state. If the migration is successful, the application is in configuration c' after time D. If the migration fails, with probability p_f, the duration until the system returns to configuration c may be characterized by a different distribution $f_{fail,c\rightarrow c'}(\tau)$. For simplicity, we assume $f_{D,c\rightarrow c'}(\tau) = f_{fail,c\rightarrow c'}(\tau)$ in this paper. Furthermore, the delay may be dependent on the environment state, however we consider them independent in this work.

4 Design of the decision framework

The decision framework consists of several components required to generate and enforce decisions. Figure 3 shows the components of the framework. The components are active at different times in the migration process. Decision policies are generated offline in the *policy generation* component. A decision policy, $\pi(c, s)$, specifies which configuration to choose when in configuration c and in state s. Two approaches for policy generation are evaluated in this work. The first approach is *greedy* and generates policies that optimize for the best immediate experience. We call this approach *instantaneous*. The second approach is based on a *Markov Decision process* which generated policies that optimize for the best long-term average experience. Both approaches assume that the state of the environment is known when the decision is made. However, this is not always possible since states that are not directly observable may affect application performance. In such cases the true environment state is hidden to the enforcement component, and in order to enforce decisions from a policy, the true state must be estimated based on parameters that can be observed. When the system is online, the environment state, $\hat{s}(t)$, is estimated in the *state estimation* component based on observations made by the *observation* component. The es-

timated state is used to enforce decisions from policies in the *policy enforcement* component.

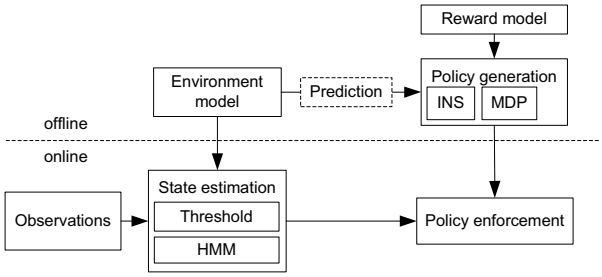


Fig. 3. Overview of the decision framework in the migration middleware.

4.1 State estimation and prediction

To enforce a policy online, the current application configuration c and the estimated environment state $\hat{s}(t)$ must be known. The observable parameters are sampled at periodic intervals during run-time and the most likely environment state is estimated. We investigate how two different estimation methods perform in the decision framework, namely a threshold approach (TH) and a hidden Markov Model (HMM) approach [15]. The two methods estimate the most likely environment state $\hat{s}$ based on observable parameters. The TH method assumes a direct mapping between observations and environment state. The HMM method models the environment Markov model and assumes a stochastic relation between the true state and the observed parameters. During run-time, the Forward-Backward algorithm is used to calculate the probability distribution over the set of states given the sequence of previous and current observations. By using a Marginal Posterior Mode (MPM) estimate [16] (without look-ahead window), the most likely state of the environment can be calculated. Transition probabilities of the HMM are the same as the environment model P. Observation probability distributions, $P(obs|\mathcal{S})$, depend on how the environment affects the observable parameters. Concrete values for both matrices are specified later in the evaluation examples.

To predict the most likely estimated state at time $t + D$ (after migration), the behavior of the Markov model is used, such that the predicted state $\hat{s}(t+D)$ is set equal to the most likely state D steps in the future. This is calculated from $\text{argmax}\, e_i \cdot P^D$, where e_i is an 1xN vector, where element i is equal to one if s_i is the most likely system state and the other elements equal zero.

4.2 Reward model

Reward is defined as a combination of the satisfaction of experiencing a certain application quality and the dissatisfaction of the interruption from migration. Reward of an application configuration in a certain system state is characterized by $R(c, s)$. The dissatisfaction is characterized by two parameters of the migration process; the probability of migration failure, p_f and the migration delay, D. These two parameters influence the user experience negatively and can therefore be used to characterize how performing a migration should be penalized. We represent the penalty of waiting during migration, η, as a function of the waiting time $\eta(D)$. The penalty function can have different shapes (linear, logarithmic, quadratic, exponential) representing different user dissatisfaction patterns. In our work, we define the function linearly as $\eta(D) = -\alpha \cdot D$, where α is a constant and represents the reduction in reward per time-step experienced by the user while waiting on the migration to finish. The linear function is chosen for simplicity, and can easily be changed to other shapes.

4.3 Instantaneous policy generation

The instantaneous (INS) approach chooses the configuration c' that generates the highest expected reward for the predicted state estimate after the migration $c \rightarrow c'$, such that

$$\pi_{INS}(c, \hat{s}(t+D)) = \underset{c' \in C}{\operatorname{argmax}} E[r(c, c', \hat{s}(t+D))]. \tag{1}$$

The reward expression $r(\cdot)$ represents the reward of migrating from c to c' when the predicted state after migration is $\hat{s}(t+D)$. The expression includes the failure probability and random migration duration as follows.

$$\begin{aligned} &E\left[r(c, c', \hat{s}(t+D))\right] = \\ &(1-p_f) \sum_{\tau=0}^{\infty} \left[R(c', \hat{s}(t+\tau)) - \eta(\tau)\right] f_{D, c \rightarrow c'}(\tau) + \\ &p_f \sum_{\tau=0}^{\infty} \left[R(c, \hat{s}(t+\tau)) - \eta(\tau)\right] f_{fail, c \rightarrow c'}(\tau) \end{aligned}$$

The INS approach is *greedy*. It is also conservative, in the sense that it maximizes the reward for the immediate state after migration (positive), but accounts for all migration penalty during migration (negative).

4.4 Markov Decision Process policy generation

The Markov Decision Process (MDP) approach can consider the effect of future decisions on the long-term average reward and generate a policy with the decisions that optimize that. An optimal policy, $\pi_{MDP}(c, s)$, can be calculated offline using the MDP before running the application. We use value iteration

with a finite window of 1000 future decisions to find the optimal policy by use of the Bellman equation for dynamic programming. The values of the window size was found through experiments and is large enough for the policies to converge, meaning that a larger window will only affect computation time, but not change the policy. See [17] for more details on the MDP framework, dynamic programming and value iteration.

The general form of the Markov model used in our decision framework is depicted in Figure 4.a. For each available configuration we assign a reward function $R(c, s)$ that maps a reward value to each environment state for a given configuration. There exist different reward functions for the different configurations. In Figure 4.b the scenario of migrating between configurations $c \rightarrow c'$ is depicted. The actions of the MDP map directly to the available configurations, which can be chosen at each time-step. To model the migration process, an intermediate delay state is introduced between the configurations. When the migration is triggered, the system enters the delay state with probability 1. The reward model of the delay state is defined according to $\eta(D)$, which means that in the linear case of this framework, the reward of the delay state is simply α. In the delay state the migration may be further delayed with probability p_d, fail back to c with probability $p_f \cdot (1 - p_d)$ or succeed to c' with probability $(1 - p_f) \cdot (1 - p_d)$. The definition of p_d is based on the migration delay D. When D is geometrically distributed then $p_d = 1 - \frac{1}{\bar{D}+1}$, where $\bar{D}$ is the mean migration delay.

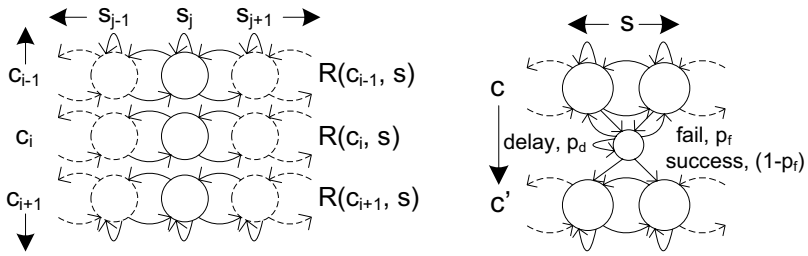


Fig. 4. Model of the migratory system. (a) is the general evolution where the reward (R) only depends on the environment if the configuration is fixed. (b) shows how a migration $c \rightarrow c'$ may change the active configuration and thereby the reward function if successful.

5 Case study and numerical results

We evaluate and compare the performance of the different estimation methods and generation approaches by simulating the migration framework. First, we systematically analyse the differences between two policy generation methods to learn where they are different, and how the difference impacts the application

quality. Next, we evaluate the accuracy of the two state estimations methods, including the ideal estimator as reference. Finally, we analyse two example scenarios of the entire framework to learn how the different parameters of the full framework impact the application quality. We evaluate three state estimation methods combined with the two policy generation methods in two different scenarios, which reflect two different network conditions.

5.1 Simulation model

The environment is modeled as a network with two states, representing a normal and a congested network, $\mathcal{S} = \{N, C\}$. P, the transition behavior within $\mathcal{S}$, is specified by p, the probability of a transition from N to N, and q, the probability of the transition from C to C, as specified in Table 1. The application is a video streaming application with two available configurations on two difference devices; D1) high definition on large display device, and D2) standard definition on mobile device with smaller screen), $\mathcal{C} = \{D1, D2\}$. The reward values of the two configurations, R, are listed in Table 1. These values express that the user gets a higher perceived quality with increasing resolution on the large display device (i.e. R=4 for (c,s)=(1,1) compared to R=3 for (c,s)=(2,1)). Moreover, the impact of packet loss is larger when $c = D1$ than $c = D2$ because the higher resolution requires more available bandwidth on the large display. As migration penalty values, we use $p_f = 0.1$ and use a geometrically distributed delay D with parameter $p_d = 0.95$. This characterizes a mean migration delay of 2 seconds, when time-unit per time-step is 100ms (equal to $\bar{D} = 20$). The failure probability and delay mean are sampled from experimental work on migration prototypes, described in detail in [18]. For the dissatisfaction function $\eta(D) = -\alpha \cdot D$, we use $\alpha = 0.01$, which has been determined a reasonable value through experiments, as it depends on a combination of the scale of the rewards and the mean delay. The parameter values are summarized in Table 1.

For network observations we use packet loss ratio in both scenarios with two possible states (low, high), representing measurable packet loss in a congestion-prone network, $obs = \{L, H\}$. The probability distributions of the observations in each network states, b, are listed in Table 1. The distributions reflect the situation where high packet loss is primarily observed when the network is congested. This would be the case when buffers in a bottleneck router overload and packets are dropped consequently, but the network otherwise is stable. We simulated 30 runs of the network model in 10000 steps and obtained sequences of true network states $s(t)$, observations from the threshold estimator and estimated network states $\hat{s}(t)$ from the HMM estimator.

5.2 Generation approach impact on generated policies

To understand the differences between the INS and the MDP generation approaches, we analyse which policies they generate under equal conditions. We use the full parameter spectrum of the environment states, defined by p and q in P (cf. Table 1), since it represents the scope of environment parameters of

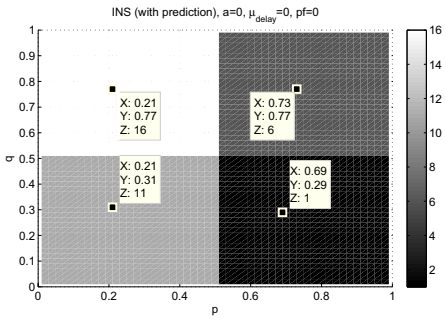
block	*description*	*name*	*value*
environment	transition probabilities	p,q	{0.01, ..., 0.99}
estimation	observation matrix	b	$\begin{bmatrix} 0.9 & 0.1 \\ 0.3 & 0.7 \end{bmatrix}$
migration model	reward	R(c,s)	$\begin{bmatrix} 4 & 1 \\ 3 & 2 \end{bmatrix}$
	waiting penalty	α	0.01
	mean delay	$\bar{D}$	20
	failure probability	p_f	0.1
MDP	window size	w	1000

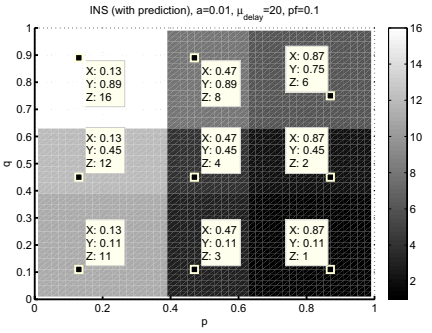
Table 1. Model parameters of the different components used in the evaluation.

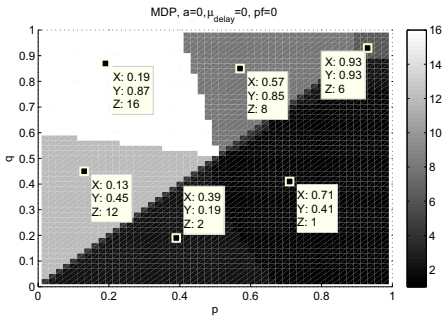
the example that characterize external, uncontrollable network state behavior. These properties are relevant to investigate as they cannot be tuned for better performance. We use a forward prediction step in the INS approach for the comparison, to be able to use p and q in both approaches. Figure 5 shows the generated policies (z-values) over the full spectrum of p and q for both approaches in the case of ideal migration ($\bar{D} = 0, p_f = 0$). There exist many regions in the parameter space where the policies are not equal.
For the INS (Figure 5, top), the regions are divided into four quadrants at $p = 0.5$ and $q = 0.5$. At the upper left (z=16) and lower right (z=1) quadrants, the policies specify to go and stay in D2 or D1 configurations, respectively, indifferent of the system state. To the lower left (z=11), the policy specifies to choose D1 configuration when in congested network state and D2 when in normal network state. The upper right region (z=6) chooses opposite to the lower left.
For the MDP (Figure 5, lower), the regions are separated by the line $p = q$. Below this line ($p > q$), the policy specifies to either to change to D1 always (z=1) or in some cases (z=2) to stay in D2 when in congested network state. Above the line ($p < q$), several different policies are used that will eventually all change to D2 always (z=8,12,16). In the small region ($p > 0.9$ and $q > 0.9$) the MDP behaves similar to the INS by choosing D1 in normal network state and D2 in congested network state (z=6).

The differences in the overall policy distributions are due to the way the two policy generation methods optimize their policies. Even though they both optimize for utility, the INS method only regards 1 future decision, whereas the MDP regards 1000. The different distributions show that it has an impact on the policy whether the future decisions are considered.

Figure 6 shows the policies over the full spectrum of p and q for both approaches in the case of non-ideal migration ($\bar{D} = 20, p_f = 0.1$). The INS policy distribution clearly changes compared to ideal migration. Instead of 4 regions, 9 regions exist, with the original 4 regions still existing though smaller. In a square-formed region in the center, a new policy is seen (z=4), which is the policy that chooses to stay in the current/initial configuration at all times. The additional 4 new policies between each pair of former quadrants are combinations of the original policy pairs. For example, this means that the policy in the middle low-







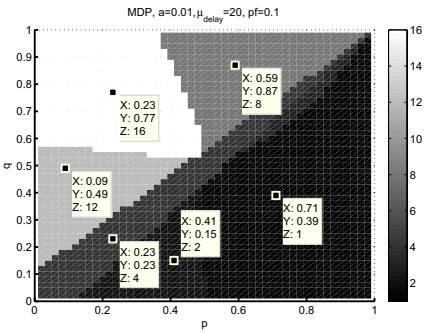


Fig. 5. Policies generated by INS and MDP approaches for different parameter settings, ideal migration.

Fig. 6. Policies generated by INS and MDP approaches for different parameter settings,, non-ideal migration.

est square (z=3) combines choosing D1 always (the policy to the right, z=1) and choosing D2 in normal network state (the policy to the left, z=11) in the way that it chooses D1 always except if in D2 and normal network state. The MDP policy distribution is very similar to the non-ideal case, except that the small region upper right is gone and there is a new region around $p = q$, that contains the same policy as the center square (z=4) in the INS figure, namely not to change configuration when network state changes. The no-change policy (z=6) is found in both INS and MDP distributions when the migration is non-ideal which can be explained by the added probability of failure and long migration delay, which decreases the expected average utility that both methods optimize for. In the border-line regions of both approaches, a higher penalty means a higher probability of decreased utility when migrating, and as a consequence, migration is not chosen in the specific cases.

5.3 Generation approach impact on application quality

To understand the impact on the quality of the different policies, we simulated runs of the decision framework using all policies from both approaches in the

entire parameter space, for ideal and non-ideal migration respectively. For each approach we calculated the average utility in each simulation with a specific policy, and calculated means over the repetitions. We repeated simulations to obtain significant differences within 95% confidence intervals. In these simulations we used an ideal state estimator. The differences in mean average quality between the approaches at each point (p, q) are shown in Figure 7 for ideal migration and in Figure 8 for non-ideal migration. Black means no difference (because INS and MDP policies are the same) and the brighter the tone, the larger the difference. The range of average utility varies between 1.6 and 4 for the different points (p, q), so a difference of 3 is relatively large.

In the ideal migration case of Figure 7, we see that in two regions ($p < 0.5$ and $q < 0.5$) and ($p > 0.5$ and $q > 0.5$) the policies of the MDP approach generate higher average utility values than the policies of the INS approach. In the non-ideal migration case, as shown in Figure 8, differences are detected in the lower left and upper right regions. The corner regions are similar to those in ideal migration, but smaller in area and with larger average utility differences. The evaluation results from the example show that in some cases the use of the MDP gives an advantage. The advantage depends on the penalty of the migration, which is a key property of the MDP approach.

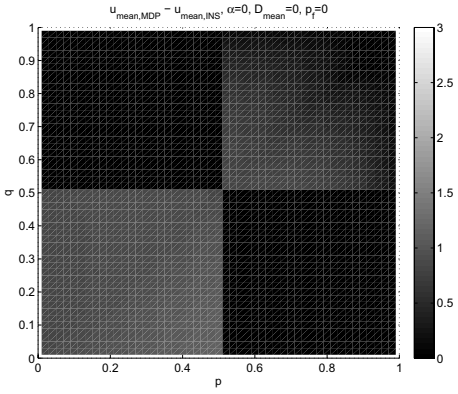


Fig. 7. Average utility difference between INS and MDP approaches for different settings of p and q, with ideal migration.

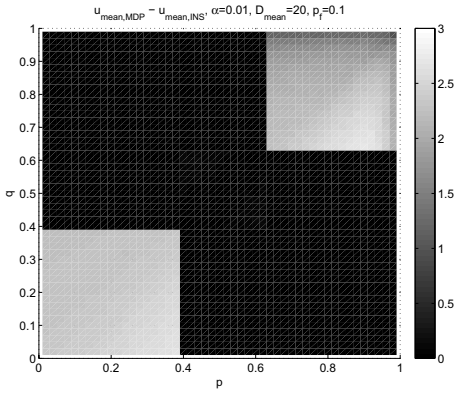


Fig. 8. Average utility difference between INS and MDP approaches for different settings of p and q, with non-ideal migration.

5.4 State estimation accuracy

The above performance evaluation is performed with ideal state estimation. In the following we describe properties of a non-ideal state estimator and the impact on the performance of the generation approach. Instead of using the entire (p,q)-spectrum, we limit the analysis to two scenarios. In scenario 1, we simulate

a network that has a low rate of state changes. In scenario 2, we simulate a fast-changing network with a high rate of changes. The differences between the two networks are expressed in the state transition probabilities. In scenario 1, the probability of changing state is very low (0.1-0.5 changes per second on average), which is seen in Table 2. In scenario 2, it is opposite, such that the probability of changing state is very high (9.5-10 changes per second on average), as seen in Table 3. Since the instantaneous approach assumes a slowly changing

$s_i \rightarrow s_j$	N	C
N	p=0.99	0.01
C	0.05	q=0.95

Table 2. Transition probabilities $P_1(s_i, s_j)$) (scenario 1)

$s_i \rightarrow s_j$	N	C
N	p=0.01	0.99
C	0.95	q=0.05

Table 3. Transition probabilities $P_2(s_i, s_j)$ (scenario 2)

environment, it is expected to perform well with in the first scenario and poorly in the second. As the MDP generates the optimal policies, it should perform best in both scenarios.

We measured the estimator accuracy as the ratio between the number of correctly estimated states and the number of true states. The results are shown in Figure 9. From the figure we see that overall the HMM has a better accuracy, and that the increased state change rate in scenario 2 impacts the estimation accuracy of both approaches. The ideal state estimator is shown for reference.

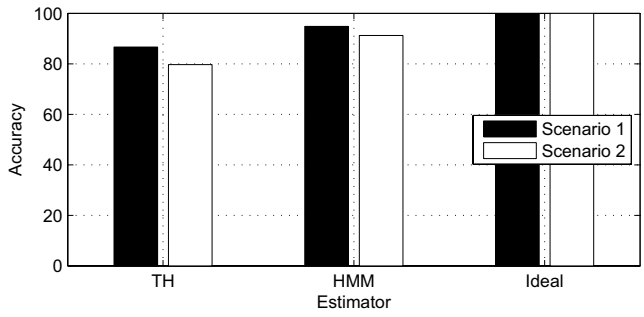


Fig. 9. Accuracy of the estimators, including the theoretical ideal estimator.

5.5 Inaccurate state estimation impact on performance

We evaluate the performance of the combined state estimation methods and generation approaches by calculating the average quality achieved during a simulation run.We use both scenarios for evaluation to illustrate the robustness of

the MDP approach. To get a clear understanding of the impact of state estimation inaccuracy, we used the INS approach without state prediction, INS', in this evaluation. The policy of the INS' approach is the same in both scenarios as it only depends on the reward distribution and not the transition probabilities. The policy is defined as

$$\pi_{INS'}(c,s) = \begin{bmatrix} 1 & 2 \\ 1 & 2 \end{bmatrix}.$$

The interpretation of the policy is that if normal network state is observed (column 1) then $c' = D1$ is decided. If high packet loss is observed (column 2) then $c' = D2$ is decided. The optimal policy generated by the MDP in scenario 1 is equal to the INS'-policy. For scenario 2, the MDP generates a policy opposite to the previously used

$$\pi_{MDP,\mathrm{s2}}(c,s) = \begin{bmatrix} 2 & 1 \\ 2 & 1 \end{bmatrix}.$$

Based on the sequences of chosen configurations and true states, we calculated the average utility achieved during the simulation runs. The mean average utilities in both scenarios are shown in Figure 10, where also results from using the theoretical ideal estimator are shown. The differences are significant in all cases within a 95% confidence interval. In scenario 1, only the state estimation method makes a difference since the policies are equal. In scenario 2, the policies are opposite. Since the MDP policy is optimal, the average quality increases with increasing estimation accuracy. However, the instantaneous policy is opposite the optimal and always makes the worst decision. This explains the decrease in average quality with the increasing estimator accuracy. The fact, that the INS' approach produces the lowest average utility with the ideal estimator in scenario 2 is due to the behavior of the INS policy. Since they are opposite of the optimal policy, INS' performance benefits from any inaccuracy in the state estimation.

6 Conclusion and future work

We have proposed a decision framework for service migration which uses a model-based policy generation approach based on a Markov Decision Process (MDP). The MDP approach generates an optimal decision policy and considers the penalty of performing the migration (failure probability and delay) when implemented in the migratory system. One key property of the MDP approach is the ability to consider the effect of future decisions into the current choice. With the MDP approach, policies are enforced based on the state of the environment. As the state is not always observable, we introduce a hidden Markov model (HMM) for estimating the environment state based on observable parameters. An example system was simulated to evaluate the MDP approach by comparing to a simpler instantaneous approach. The average user experience quality of the generation approach was compared to a simple instantaneous approach that does not consider future decisions. The comparison used a model of a slowly changing

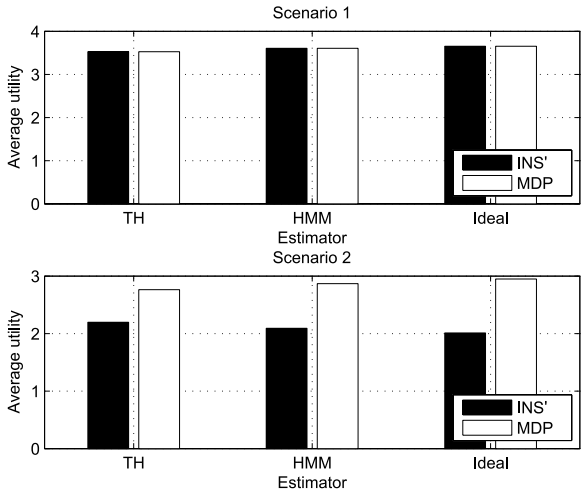


Fig. 10. Average utility comparison between the three estimation methods and the two policy generation methods in both network scenarios.

network and showed that the introduction of the HMM alone gives benefits, as the achieved average quality was slightly higher for the MDP approach than the instantaneous approach. When the network model was changed to include more rapid changes, the MDP approach produced the highest quality and followed the dynamics of the environment more precisely. With our results we are able to quantify the gain in performance of considering future decisions.
Future work should extend the capabilities of the framework to enable online generation of policies to support changing environment models, and ultimately learn environment parameters online to change policies during run-time.

Acknowledgments

This work was partially supported by the EU ICT FP7 project 'Open Pervasive Environments for iNteractive migratory services – OPEN', see www.ict-open.eu. The Telecommunications Research Center Vienna (FTW) is supported by the Austrian Government and by the City of Vienna within the competence center program COMET.

References

1. T. Illmann, F. Kargl, M. Weber, and T. Kruger, "Migration of mobile agents in java: Problems, classification and solutions," in *Proc. of the Int. ICSC Symposium on Multi-Agents and Mobile Agents in Virtual Organizations and E-Commerce (MAMA00)*, Citeseer, 2000.

2. A. Nickelsen, F. Paternò, A. Grasselli, K.-U. Schmidt, M. Martin, B. Schindler, and F. Mureddu, "OPEN: Open pervasive environments for migratory interactive services," in *12th international conference on Information Integration and Web-based Applications and Systems (iiWAS)*, ACM, 2010.
3. H. Klus, D. Niebuhr, and A. Rausch, "A component model for dynamic adaptive systems," in *International workshop on Engineering of software services for pervasive environments*, p. 28, ACM, 2007.
4. J. Fox and S. Clarke, "Exploring approaches to dynamic adaptation," in *Proceedings of the 3rd International DiscCoTec Workshop on Middleware-Application Interaction*, pp. 19–24, ACM, 2009.
5. C. Hofmeister and J. Purtilo, "Dynamic reconfiguration in distributed systems: Adapting software modules for replacement," in *Distributed Computing Systems, 1993., Proceedings the 13th International Conference on*, pp. 101–110, IEEE, 2002.
6. J. Hillman and I. Warren, "An open framework for dynamic reconfiguration," in *Proceedings of the 26th International Conference on Software Engineering*, pp. 594–603, IEEE Computer Society, 2004.
7. S. Kalasapur, M. Kumar, and B. Shirazi, "Dynamic service composition in pervasive computing," *IEEE Transactions on Parallel and Distributed Systems*, pp. 907–918, 2007.
8. F. Paternò, C. Santoro, and A. Scorcia, "User interface migration between mobile devices and digital tv," in *HCSE'08*, p. 292, Springer-Verlag, 2008.
9. C. Wüst and W. Verhaegh, "Quality control for scalable media processing applications," *Journal of Scheduling*, vol. 7, no. 2, pp. 105–117, 2004.
10. D. Vengerov, "Dynamic adaptation of user migration policies in distributed virtual environments," 2009.
11. R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli, "Dynamic qos management and optimisation in service-based systems," *Software Engineering, IEEE Transactions on*, 2010.
12. H. Schwefel, S. Praestholm, and S. Andersen, "Packet Voice Rate Adaptation Through Perceptual Frame Discarding," in *IEEE Global Telecommunications Conference, 2007. GLOBECOM'07*, pp. 2497–2502, 2007.
13. C. Luna, L. Kondi, and A. Katsaggelos, "Maximizing user utility in video streaming applications," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 2, pp. 141–148, 2003.
14. S. Shenker, "Fundamental design issues for the future Internet," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 7, pp. 1176–1188, 1995.
15. L. Rabiner and B. Juang, "Introduction to hidden Markov models.," *IEEE ASSP MAG.*, vol. 3, no. 1, pp. 4–16, 1986.
16. K. Salamatian and S. Vaton, "Hidden markov modeling for network communication channels," in *Proceedings of the 2001 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, p. 101, ACM, 2001.
17. M. Puterman, "Markov decision processes: Discrete stochastic dynamic programming," *IMA Journal of Management Mathematics*, 1994.
18. K. Højgaard-Hansen, H. C. Ngyuen, and H.-P. Schwefel, "Session mobility solution for client-based application migration scenarios," in *Wireless On-demand Network Systems and Services (WONS)*, 2011.