Optimal Model-based Policies for Component Migration of Mobile Cloud Services

Rene Gabner*, Hans-Peter Schwefel*†, Karin Anna Hummel‡, Günter Haring‡
*Telecommunication Research Center Vienna, Donau-City-Strasse 1, A-1220 Vienna, {gabner, schwefel}@ftw.at
†Aalborg University, Department of Electronic Systems, Niels Jernes Vej 12, 9220 Aalborg, hps@es.aau.dk
‡University of Vienna, Distributed and Multimedia Systems, Lenaugasse 2/8, A-1080 Wien,
{karin.hummel, guenter.haring}@univie.ac.at

***Abstract*—Two recent trends are major motivators for service component migration: the upcoming use of cloud-based services and the increasing number of mobile users accessing the Internet-based services via wireless networks. While cloud-based services target the vision of Software as a Service, where services are ubiquitously available, mobile use leads to varying connectivity properties. In spite of temporary weak connections and even disconnections, services should remain operational. This paper investigates service component migration between the mobile client and the infrastructure-based cloud as a means to avoid service failures and improve service performance. Hereby, migration decisions are controlled by policies.**

To investigate component migration performance, an analytical Markov model is introduced. The proposed model uses a two-phased approach to compute the probability to finish within a deadline for a given reconfiguration policy. The model itself can be used to determine the optimal policy and to quantify the gain that is obtained via reconfiguration. Numerical results from the analytic model show the benefit of reconfigurations and the impact of different reconfigurations applied to three service types. Immediate reconfigurations are in many cases not optimal, hence a threshold on time before reconfiguration is desirable to control reconfiguration.

I. Introduction

Mobile computing services are becoming increasingly popular. As mobile devices provide only scarce resources and are used by non-experts, installing and operating services on the device has severe drawbacks. As one answer to this challenge, cloud computing offers an infrastructure to implement the vision of *Software as a Service (SaaS)* [1] where software is hosted and maintained at a service cloud in the infrastructure. In addition to taking the burden of installing, updating, and maintaining of software away from the user, the computing, energy, and memory resources of mobile devices are spared.

Mobile access to cloud services on the other hand is challenged by varying wireless link conditions due to environmental effects and by mobility of the devices. As a consequence, degraded link quality and connection losses are likely to happen and may affect service execution times and service availability in SaaS scenarios.

An approach to dynamically migrate service components between the mobile device and the infrastructure-based cloud can alleviate these performance problems. In doing so, the trade-off between increased execution times on client platforms as well as reconfiguration delays required for the component migration and reduced communication needs during execution has to be carefully balanced: while for stable and high-performance client connectivity, service components are best executed in the cloud, in situations of unreliable and weak links, it will be better to execute service components locally on the mobile device.

While the overall approach is plausible and has been presented for static scenarios in previous works of the authors [2], the automated decision process of dynamic service reconfiguration, i.e., dynamic component placement between the cloud and the client, requires advanced means for estimating the effects of component migration prior to taking a decision. The contribution of this paper lies in (i) providing a Markov model for evaluating the performance effects of dynamic reconfigurations in the client/cloud system, and (ii) proposing model-based strategies (policies) to maximize the probability to finish before a given deadline, (iii) and to illlustrate the benefit of the model-based policiiesfor different service component flow examples.

We discuss mobile cloud computing and mobile SaaS approaches in related work (Section II) as well as related methodological approaches in the context of other fields. The novelty of our approach lies in the particular quantitative modeling of a mobile SaaS system by decomposing the service into movable components. The overall scenario containing a mobile client and the cloud, principles of service decomposition and service flow, and migration of components are presented in Section III. Based on this system description, we develop a new analytical model based on Markov chains (Section IV). For reconfiguration of service component placement, a phased approach is used (changing from one configuration to the next at a reconfiguration point in time). The decision for choosing a configuration is modeled by different policies optimizing for maximum probability to finish before a given deadline (Section V). For three different component topologies, we provide analytical results demonstrating and quantifying the achieved benefit by model-based reconfiguration policies (Section VI).

II. Background and Related Work

Mobile computing introduces challenges related to wireless networking, mobile data management, seamless mobile computing, and adaptations due to limited mobile device capabilities. Implications and challenges of mobile computing are described by Imielinski et al. [3] Important aspects are *(i)*

management of location dependent data, *(ii) disconnections*, *(iii) adaptations of distributed algorithms for mobile hosts*, *(iv) broadcasting over a wireless network*, and *(v) energy efficient data access*. When developing distributed computing solutions, thus, these impairments and challenges will cause service failures and significant delays. In our work, we propose utilizing the cloud computing paradigm as well as code migration between the cloud and the mobile device.

A potential disadvantage of running services locally at the mobile device are limited resources in terms of computing power, memory, or battery power. In the past years there was a trend towards offloading services to central cloud environments [4], [5], [6], [7], [8]. The cloud-based SaaS [1] approach is one possibility to avoid development for different mobile target systems. Instead of executing the service locally on the mobile device, it is possible to run it directly on the more powerful and flexible server environment within the cloud network. Totally independent of their operating systems, all clients which are connected to the network server can use the services deployed in the cloud.

To handle access and execution of SaaS services, a special execution environment on the mobile devices is required. Chou and Li [9] present a SOA-based solution for Android phones. Amoretti et al. [10] analyze ways to cope with highly dynamic cloud environments. Possible design approaches for services supporting migratory components in flexible environments are presented in [11].

To access SaaS services running remotely on a server infrastructure, reliable network connectivity and disruption tolerant networks are of particular importance. As this is not always the case in mobile networks, services should be able to adapt to changing environments. Hence, when including mobile devices, virtualization and code mobility can be applied to overcome disconnection periods when connected to the cloud. Hereby, code migration has a long tradition [12], [13], [14]. A model-based performance analysis is proposed for mobile software systems in [15] which allows to compare design alternatives in an early design phase. Our approach is related, but extends these approaches by introducing model-based decision policies for runtime migration based on an analytical model which allows for calculating the expected benefits for reconfiguration (i.e., component movement).

Model-based studies of service adaptations to changing environments are also performed in the context of access network selection: Gronbak et al. [16] develop a Markov model of a file-transfer application where imperfect network state diagnosis is used to take decisions on handover to other available networks. Target is to increase the probability to finish the file transfer within a given deadline. The model results show that under inaccurate diagnosis, policies that fail-over only after a minimum time-threshold is passed, can achieve higher service reliability in certain settings. Analyzing the effectiveness of service reconfiguration utilizing code mobility approaches was the major aim of the work by Grassi et al. [17], [18]. They developed a modeling framework to analyze the performance effectiveness of adaptations in mobile networks. Our approach is clearly related, as one of the aims is to analyze the performance of our solution using an analytical model as first introduced in [2].

III. System Description and Assumptions

The system platform envisioned supports the *Software as a Service* (SaaS) paradigm [1], where, in principle, services are run remotely and accessed from a client device. One major constraint of SaaS is the availability of a stable, *always-on* network connection to the host running the service. Applied in a mobile context, intermittent connectivity caused by disrupted transmissions at the air interface is a major challenge.

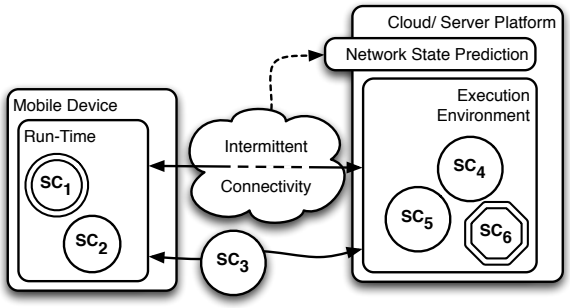

Fig. 1. Overall system architecture.

To overcome this issue we assume that the service consists of several components which can be moved during runtime from the network-side execution platform to the client, e.g., to optimize with respect to a predicted network quality. Figure 1 shows a service example, which consists of six *service components* ($SC_1, SC_2, \cdots, SC_6$). Service components are either executed within a run-time environment on the client or on the cloud, while one service component, SC_3, is currently moved from the cloud infrastructure to the mobile client (i.e., sent as a message over the wireless connection). The module *Network State Prediction* collects and extrapolates information about the network connection between the server and the client which allows for an estimate of network model parameters.

The general system model has been introduced for an investigation of stationary component placements in [2]. Now, we extend the analytical model by adding dynamics via service component migration. In the general case, the service consists of a number of K components. The execution path through the different service components and the execution time for the individual components depend on varying input parameters, and are described by a stochastic process. The service components are assumed to be autonomous in the sense that all needed input data is passed to the component at the component call, so that no interaction with the environment (or with other components) is needed during the component execution period. Hence, network communication is only required when the execution flow is passed to a new component. The service is completed successfully upon the entrance to a specific final component, here, without loss of generality, assumed to be component K.

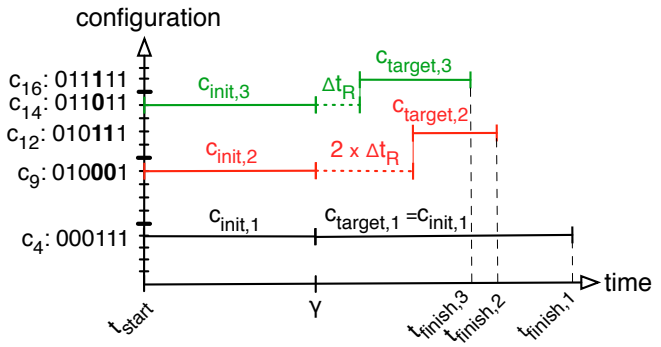

Fig. 2. Illustration of reconfiguration during runtime for three different initial configurations: For $c_{init,1} = c_4$, no reconfiguration is occurring, so the application executes continuously until $T_{finish,1}$. For the other two cases, reconfigurations occur, which leads to a reconfiguration delay (marked dotted) that is in this example proportional to the number of components moved.

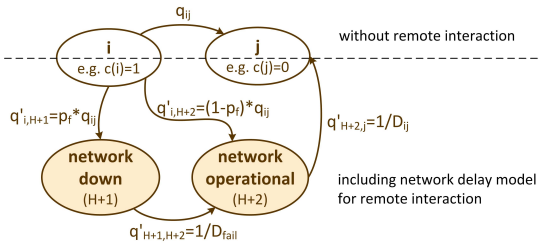

Fig. 3. Extended service component model including network failure and execution time [2].

Individual components can either be placed and executed at the infrastructure-based cloud, or they can be executed on the mobile device. The placement of all K components is called *configuration* and is represented by the binary vector c, where $c_i = 0$ means that component i is placed on the client, while $c_i = 1$ represents the placement on the cloud ($i = 1, \cdots, K$). In case of component execution on the client, the execution time of the component is affected by a factor k_{client}; $k_{client} > 1$ represents the slow-down effect of less available computing resources on the mobile device. When the client-based service component i passes the execution flow to a component j placed on the infrastructure, hence remote, a certain time-period with average duration $D_{rem_{i,j}}$ is required for passing the required input data to the remote component. For simplicity, this delay is assumed independent of the passing direction of the execution flow. Hence, it is identical (and identically distributed), if – vice versa – component i is placed on the cloud side and j is placed on the mobile client. A component call without required remote communication is assumed to be instantaneous.

The network is assumed to show "ON-OFF behavior", i.e., a remote component call is delayed for the remaining OFF period if it is started in OFF state. The semantics of the OFF state include disconnections and low-quality network connections, which both result in longer delays.

Service components can be moved during run-time, but this dynamic reallocation will be at the cost of an additional reconfiguration delay. This reconfiguration delay can depend on the number and type of components moved and furthermore on the network state, see Figure 2 for an illustration. During the time period when a remote component call is progressing, component migration cannot be performed and will be delayed until the remote component call has successfully been finished.

The primary target of the dynamic component placement adaptation is to increase service dependability. We assume here that the service success depends exclusively on the execution time, represented by the random variable T_{finish} with density function $f_T(t)$. For interactive services, the service success could for instance be described by a utility function with service execution time as parameter. For space reasons, we do not consider general utility functions in this paper, but focus on maximizing the probability to finish before deadline, $\Omega := Pr(T_{finish} \leq T_D) = F_T(T_D)$: this corresponds to maximizing a utility function that is a step function, switching from 1 to 0 at deadline T_D.

The main problem addressed in this paper is to derive a mathematical model as basis for the decision whether a reconfiguration is beneficial and what target configuration should be chosen to optimize Ω. The benefits of reconfiguration and the usefulness of the mathematical model are illustrated in selected numerical examples.

IV. System Model

In order to build a quantitative analysis model that allows to investigate different reconfiguration policies, this paper is following a two-step approach: (i) a quantitative model for the static setting is summarized based on previous work for reasons of understandability and (ii) the static setting is extended to a phased model allowing for reconfiguration during runtime. The analytic model can subsequently be used to identify optimal reconfiguration policies.

A. Markov model of the static setting

Assuming that the service is started at time $t = 0$ in the initial configuration c_{init} and kept in this configuration until finalization, the goal is to obtain an analytic model that allows to compute the finishing probability Ω before a certain deadline T_D. In [2], we have introduced a Markovian model for this static scenario, which is summarized in the remainder of this section.

When executed all on the same entity (without remote calls), the service components are represented by K states of a continuous Markov model with a generator matrix $\mathbf{Q}$ (we use the generator form which has negative elements on the main diagonal). A consequence of this Markov assumption is that component execution times are exponentially distributed and that the control flow (the sequence of component calls) is probabilistic, only depending on the current component. These assumptions are however not limiting, since they can be easily removed via the use of phase-type or Matrix exponential distributions and via explicitly including an additional state for control-flow adaptation in the component model.

For a given configuration c of the components on network or client-side, the continuous Markov chain Q is extended as shown in Figure 3 [2]: For every remote component call two additional states are included, one for the network being down (with probability p_f) and one for the operational network. D_{fail} is the delay occurring when the network is down. Note, that this approach avoids a product space representation at the price of the approximation that at the start of every remote component the network is OFF with probability $p_f = \overline{OFF}/(\overline{OFF} + \overline{ON})$. This approximation is exact for long time-periods between remote component calls, while it is conservative for shorter intervals (as the network was up at the last remote transition, and the transient probability of being in state 'ON' approaches its steady-state limit from above). All exponential assumptions can be removed via the use of phase-type distributions.

Let the extended Continuous Time Markov Chain (CTMC) model for component placement vector c be represented by the generator matrix $\tilde{Q}^{(c)}$, which has $K + 2n_r$ states, where $0 \leq n_r \leq (K/2)^2$ is the number of remote component calls. To translate this generator matrix to the Phase-type distribution format of [19], it is necessary to remove the column and row that corresponds to the absorbing final state and to negate the matrix (as the format of [19] uses a representation with positive main diagonal). Via this transformation, the matrix $\tilde{B}^{(c)}$ is obtained together with the appropriately zero-padded entrance vector $\tilde{\pi}_0^{(c)}$, the vector-matrix-pair $< \tilde{\pi}_0^{(c)}, \tilde{B}^{(c)} >$ specifies the Phase-type distribution of the service execution time, and the corresponding metric of interest can then be obtained as

$$\begin{aligned} \Omega &= Pr(\text{finish time under config } c \leq T_D) \\ &= 1 - \tilde{\pi}_0^{(c)} \text{expm}\left(-\tilde{B}^{(c)} T_D\right) \epsilon'. \end{aligned}$$

Here, ϵ' is a column vector of appropriate dimension and expm() is the Matrix exponential function. Note, that these equations allow to calculate the performance metrics in the static setting, given that the service configuration is c for the whole service duration.

B. Reconfiguration policies in dynamic setting

In order to analyze reconfiguration policies that control service component migration during run-time from cloud to the client device and vice-versa, we need to extend the static model from Section IV-A. To obtain an understanding of the behavior, we limit ourselves to the investigation of a single reconfiguration step during the service execution, i.e., at some moment in time $t_{reconfig}$, the configuration is changed from c_{init} to c_{new}, where both are binary vectors of dimension K. We make the following assumptions for this reconfiguration step:

- Reconfiguration can occur during component execution. If $c_{init} \neq c_{new}$, a reconfiguration delay will result, expressed by a random variable with expected value $D_r(c_{init}, c_{new})$. After this delay, the execution of the currently active component will be resumed (in either the same or the new execution environment, depending on the configuration change).
- Although more general distributions can be included, we limit ourselves in the case studies to deterministic reconfiguration delays.
- The reconfiguration delay is considered independent of the network state to facilitate the model description. It is a trivial technicality to remove this.
- If a reconfiguration is triggered at a moment in time when a remote component call is in progress, first this remote component call is finalized and then the reconfiguration procedure with the associated delay is conducted.

In the general case, a policy for the single-reconfiguration case is a function mapping time, service state (component ID), network state, and current configuration to the new configuration:

$$P : \mathbb{R}^+ \, x \, \{1, ..., K\} \, x \, \{ON, OFF\} \, x \, \{0,1\}^K \quad \longrightarrow \quad \{0,1\}^K$$

The inclusion of time is necessary as the optimality criterion is time-based, in particular when using the deadline T_D (see also [16]).

As the intention of the component reconfiguration here is to provide an optimal choice despite fluctuating network state, the policy is actually assumed independent of the current network state, though still optimized for the long-term probabilistic behavior of the network. Hence, we constrain ourselves later to the investigation of policies of the form

$$P' : \mathbb{R}^+ \, x \, \{1, ..., K\} \, x \, \{0,1\}^K \quad \longrightarrow \quad \{0,1\}^K,$$

$$c_{new} = P'(\gamma, a, c_{init}).$$

Hereby, a is the service state at reconfiguration moment $t_{reconfig} \geq \gamma$. The time instant γ is the moment at which reconfiguration is triggered (earliest executed): If at this moment, a component execution is running, $t_{reconfig} = \gamma$, otherwise, the reconfiguration will occur as soon as a remote component call has finalized. Later in Section VI, we will study the impact of this reconfiguration threshold γ on the optimal policies and on the actual performance metrics.

The analytic model therefore needs to be extended to be able to include a single reconfiguration event under policy P'.

C. Phased model for dynamic reconfiguration

For the calculation of the performance metrics in the dynamic setting, we use a phased approach, namely we compute the behavior of the Markov chain corresponding to $\tilde{Q}^{(c_{init})}$ up until reconfiguration trigger at time γ. Then we compute the behavior under $c_{new} = P'(\gamma, a(t_{reconfig}), c_{init})$ for the second phase starting at $t_{reconfig} + D_r$ (where D_r is the reconfiguration delay). Some special consideration has to be given to the cases when a remote component call at time γ is active and hence $t_{reconfig} > \gamma$.

Let π_γ be the probability vector of the CTMC $\tilde{Q}^{(c_{init})}$ at time γ:

$$\pi_\gamma = \tilde{\pi}_0^{(c_{init})} \text{expm}\left(\tilde{Q}^{(c_{init})}\gamma\right).$$

For ease of notation, we introduce the following conditional random variables and (conditional) probabilities:

- R_a is the (residual) time to reach the final state of the service in configuration c_{new} given that the execution is started in service component a. R_a has a Matrix-exponential distribution with representation $< e_a, \tilde{B}^{(c_{new})} >$. e_a is a vector with all components set to zero except component a which set to 1.
- $T_{<\gamma}$ is the time to finish if the service finishes before γ (the configuration is here c_{init}).
- $p_{<\gamma} = \pi_\gamma(\text{absorbing state})$ is the probability that the service finishes until time γ (the configuration is c_{init}).
- $d_\gamma(l,a)$ is the probability that the service execution flow is in a remote call from component l to component a and the network is down; this corresponds to the element in the vector π_γ that represents the corresponding state marked $H+1$ in Figure 3.
- $c_\gamma(l,a)$ is the probability that the service execution flow is in a remote call from component l to component a and the network is up; this corresponds to the element in the vector π_γ that represents the corresponding state marked $H+2$ in Figure 3.

For better readability, the configuration is not stated explicitly in the superscript for these identifiers. Instead, only R_a is used which refers to 'Phase 2' with configuration c_{new}, while all others refer to 'Phase 1' with configuration c_{init}.

The probability Ω that the service finishes before deadline T_D in case of at most one reconfiguration from c_{init} to $P'(\gamma, a, c_{init})$ under reconfiguration delay D_r, can then be computed by (where A is the absorbing state)

$$\begin{aligned}
\Omega \;=\; & p_{<\gamma} + \sum_{a=1,a\neq A}^{K} \Big[\\
& \pi_\gamma(a)\, Pr\left(R_a < T_D - \gamma - D_r(c_{init}, P'(\gamma, a, c_{init}))\right) \\
& + \sum_{l=1,l\neq a}^{K} c_\gamma(l,a)\, Pr\big(R_a + T_c(l,a) < \\
& \quad T_D - \gamma - D_r(c_{init}, P'(\gamma, a, c_{init}))\big) \\
& + \sum_{l=1,l\neq a}^{K} d_\gamma(l,a)\, Pr\big(R_a + T_c(l,a) + T_{down} < \\
& \quad T_D - \gamma - D_r(c_{init}, P'(\gamma, a, c_{init}))\big) \\
& \Big] \qquad (1)
\end{aligned}$$

The three terms contributing to the outer sum for Ω (cases of 'Phase 2') are: (i) in the second line of Eq.(1), cases where reconfigurations occur while component a is executed, (ii) in the third and fourth line, contributions to Ω from cases in which a remote component call from l to component a is being executed while the network is up, and (iii) in the two bottom lines, the cases described originate from cases where a remote component call is executed while the network is down.

Hereby, $T_c(l,a)$ is the random variable describing the time to perform a remote component call from component l to component a given the network is up. T_{down} is the time the network is down. Both are later assumed to be exponentially distributed, but any arbitrary Matrix exponential distribution could be used. The convolution of the random variables is then obtained via concatenating Matrix exponential distributions and the three probabilities result via standard Matrix exponential calculations. If the reconfiguration delay D_r is also Matrix-exponentially distributed, it just needs to be moved to the left-hand side of the equation and also included in the convolution (via concatenation) of Matrix exponentials.

V. Optimal Configuration Selection

The analytic model of the previous section can now be used to determine optimal target configurations and to determine optimal time instances for reconfiguration.

The policies introduced aim at maximizing the probability of a service to finish before the deadline. In addition to the optimization policies, simple policies are introduced for comparison reason.

Model-Based Optimizing Policies: Policies are introduced that determine the optimal target configuration via the use of the analytic model. Optimality here means maximizing the *probability* to finish before the deadline. Two versions are introduced and later numerically compared:

- P'_{state}: This strategy selects the optimal target configuration based on a calculation for the highest probability to finish before the deadline, given the knowledge of the current application state at the reconfiguration time. For each application state, potentially a different target configuration can result.
- P'_{single}: This strategy does not utilize the knowledge of the current application state, hence it only computes a single target configuration that optimizes the probability to finish before the deadline, given that the current service state is determined by the transient state probabilities at the reconfiguration time.

Obviously, the second policy is less informed and hence will perform worse than P'_{state}. Rationale to investigate these two policies is to investigate how much benefit can result from the knowledge of the application state at the price of a higher complexity of policy calculation and policy description. In addition, there may be scenarios in which the current application state is not easily obtainable by the reconfiguration mechanism, so the sub-optimal P'_{single} can only be used.

Algorithmic Realization: Given the application characterization, the initial configuration, the parameters of the network and execution environment, it is of course possible to precompute the above policies and store them for instance in table form. However, in particular when environments parameters are estimated in the operational system, an online computation of the optimized target configuration would be desirable. The computational effort of the target configuration calculation

is therefore interesting, though this paper does not focus on performance optimized calculation procedures.

The optimal target configuration(s) can be simply computed by enumerating all possible target configurations and computing the Ω metric using the phased analytic model introduced in Section IV. For $K-2$ movable application components (as assumed here: the UI component is bound to the client and the service finish state is bound to the server-side) there are $2^{(K-2)}$ target configurations. The computationally most expensive operation in the computation of Ω is the computation of the matrix exponential function. With some optimization, the matrix exponential needs to be computed once for every $\tilde{B}^{(c)}$ matrix. Obviously, the computational cost of the matrix exponential calculation also depends on the size of the $\tilde{B}^{(c)}$ matrices. In the examples later, we use matrices with some tens of states. Calculations for hundreds of states are also rather fast on desktop machines. However, the online computation becomes problematic, if there is a large number of application states, as then the brute-force search among all possible configurations becomes exponentially expensive. Therefore, such online computation with a brute-force search algorithm is only feasible for a low number (several tens) of candidate target configurations. The granularity of the division of the application in migratable components should therefore not be too fine.

Policies for Comparison Purpose: To evaluate the impact of the optimization policies, two simple policies are introduced:

- *No reconfiguration*: This policy keeps the initial configuration (c_{init}) and does not change it. Comparison with this policy allows to investigate whether there is any benefit from dynamic reconfiguration.
- *Client*: This strategy assumes that almost all service components (except the "exit-component") are moved to the client ($c_{client} = (0^*1)$, e.g., considering six components, the configuration is given as: $c_{client} = (000001)$).

VI. Case Study and Results

We illustrate the benefit from the model-based optimized reconfiguration policies and the impact of the reconfiguration time in selected synthetic application examples.

A. Service types

We included three different archetype topologies of service component flows in our investigations as use cases: 'linear' (or sequential) service execution, 'split' service execution, and a star-like service execution. All different service execution topologies have in common, that the first component (here, component 1) is always executed at the client (corresponding to the design of a startup UI) and the last component (here, component 6) has to be executed on the server, e.g., performing saving functions. In previous work [2], we presented the results for static service configurations and the star-like service execution based on an editor service example.

Figure 4a depicts the structure of the 'linear' service execution topology. The state transition rates are all set to 1, thus, the overall mean service execution time is 5 time units.

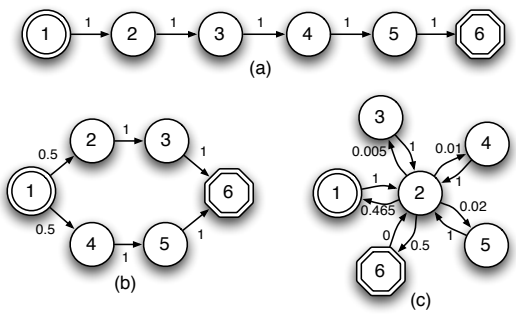

Fig. 4. (a) Linear, (b) split, and (c) star topology. All services start at component 1(at the client) and terminate at component 6 (at the server)

Slow-down factor of execution at the client	$k_{client=10}$
Delay of remote component call [time-units]	$D_{rem} = 1$
Network down time [time-units]	$D_{fail} = 20$
Network failure probability	$p_f = 0.8$

TABLE I
Parameter settings used during the analytical study.

The *'split' service execution topology* is a pattern used by services incorporating choices and following a sequential (linear) pattern in each branch. These choices can result from user selections, e.g., selecting between location based information of nearby sights or a location based reservation service in a tourist service application. Figure 4b visualizes this type of service component topology. The state transition rates are 0.5 from component 1 to component 2 and component 4, otherwise 1. In this service execution topology, the point in time of earliest reconfiguration is expected to have a significant impact on the resulting performance of the service.

The *'star' service execution topology* is a pattern used by services consisting of one major "central" component. As this component is distinguished among all others, it is of high importance where this component is placed in relation to the other components. When modeling a service flow of for instance a text editor, this central component can be a controlling component invoking other functions (such as loading, saving, etc.). Figure 4c visualizes the 'star' topology. Here, the state transition and holding rates model different usage characteristics. The settings used in this study are inspired by the *editor* service example used in [2].

B. Parameter choices

The evaluation is performed for the three different service types (service topologies) introduced in the previous section with the component transition rates given in Figures 4a, 4b, and 4c.

Table I summarizes the default parameters used for the subsequent experimental studies. Note, that k_{client} is a unitless ratio, while D_{rem} and D_{fail} are time-intervals.

For the given default parameters, the expected service finish times for the static setting with two different initial configurations are shown in Table II. The configuration $c = (111111)$ corresponds to all components being executed on a cloud or

	linear	split	star
$E(T_{finish})(c = 111111)$	5	3	1.23
$E(T_{finish})(c = 000111)$	49	38	30.85

TABLE II
T_{finish} FOR SELECTED CONFIGURATIONS FOR ALL THREE COMPONENT TOPOLOGIES.

Relative Deadline	10
Relative Reconfiguration Threshold	5
Reconfiguration delay (per component)	10 time-units

TABLE III
SETTINGS FOR COMPONENT PLACEMENT RECONFIGURATION.

infrastructure-based server, hence no impact of communication and client-side performance behavior occurs. This is the minimum finish time achievable for each service type. Note, however, that this configuration is hypothetical, as all possible configurations assume that the starting component is a UI executed on the client side. Naturally, neither network-related parameters nor the client slow-down factor (as shown in Table I) impact this configuration. In the second configuration selected (i.e., $c = (000111)$), half of the components are on the client and half on the infrastructure-side (cloud) leading to substantially longer mean finish times, which are influenced by the parameters in Table II. As this second configuration represents a balance between network and server-side components with intermediate performance behavior, it will be used as the default initial configuration in the subsequent experiments and analysis.

As we aim for comparing three different service types despite their different execution times we specify parameters of the reconfiguration policies relative to their mean execution times in the base configuration $c = (111111)$ (see Table III for the default value). For instance, the default value of the relative deadline of 10 thereby means an absolute deadline of 50 for the 'linear' model, 30 for the 'split' model, and 12.3 for the 'star' model. Following the requirement that network models should be comparable across service types, reconfiguration delay is not given relative but as absolute number of time-units.

There are several notions for reconfiguration delays like those based on the number of moved components, or depending on their actual size. In this section we consider the notion of a mean delay proportional to the number of components migrated. This resembles the case when component code has to be moved between the cloud and the client device and the size of the code is assumed to be proportional to the number of components.

C. Numerical Results

First, validation of the equations and their implementation via simulation experiments was performed; however these results are not provided here, as they just verify the correctness of the analytic formulas and their implementation.

Figure 5 visualizes a comparison between the different reconfiguration policies as described earlier in Section V and their impact on the probability to finish before deadline, Ω. We thereby vary the (relative) time threshold γ , which is the first possible reconfiguration time. The figure shows that there is an actual benefit from dynamic reconfiguration, if the target configuration is selected carefully as in the model-based approaches. Furthermore, it is possible to increase Ω by postponing the reconfiguration task instead of immediate reconfiguration at the start. For the 'linear' service type, Ω can be improved by approx. 15% for policy P'_{single} for $\gamma \approx 3$ and even 25% for P'_{state} for $\gamma \approx 4$. Whereas the 'split' type allows for less improvement (around 10%) and only with P'_{state}. Smallest benefits are achieved with the 'star' topology. The policy P'_{single} does not show any improvement for the 'split' and 'star' service types compared to no reconfiguration ('Non'), hence, service-state dependent choice of the target configuration is a pre-requisite for improving service performance in these two service examples. When the relative reconfiguration threshold approaches the relative deadline (at value 10), all reconfiguration policies are identical to 'Non', as they have no influence on the transient service performance metric Ω any more. Reconfiguring to a pure client-based configuration decreases Ω for all choices of $\gamma <$ deadline.

Figure 6 presents the optimal component configurations (vectors $c_1 \cdots c_{16}$) for the 'split' topology utilizing the P'_{state} reconfiguration policy. When the service is in States 1, 4, and 5, reconfigurations are actually never performed (for no choice of γ) as the target configuration is always identical to the initial configuration. When the service is in State 2 and relative reconfiguration time is below approximately 3, a target configuration with all components except the UI on the server is chosen. When the current application state is State 3, only this component 3 is migrated to the server – obviously there is no benefit of migrating component 2 any more, as it will not any more be executed in this case. The fact that the target configurations are a discrete set and the optimal target configuration changes at discrete values of the reconfiguration time leads to the two discontinuities in the derivative of the red dashed-dotted curve in Figure 5 at relative reconfiguration time approximately 3 and 7.

VII. SUMMARY AND OUTLOOK

This paper introduced an analytic model for quantitative analysis of component migration during runtime. The proposed model is used to determine optimal reconfiguration policies targeting at maximizing the probability of service completion given a finite deadline. The paper illustrated the usefulness of the model and of the dynamic reconfiguration approaches in three example scenarios of synthetic service types: a 'linear' component model in which service components are executed sequentially, a 'split' component model in which two disjunct branches of sequential component executions are probabilistically occurring, and a 'star' component model in which a central component is repetitively taking over the control flow.

The numerical results show that a careful selection of reconfiguration time and target configuration can substantially

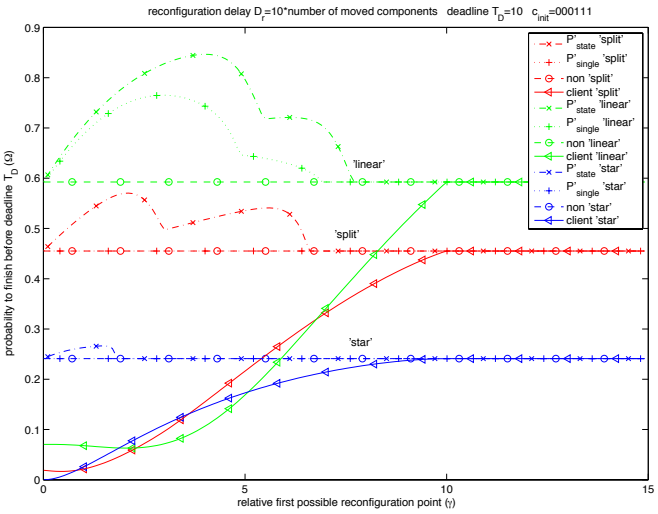

Fig. 5. Dynamic reconfiguration for 'star', 'split', and 'linear' service types - probability to finish before deadline for reconfiguration delays proportional to number of moved components.

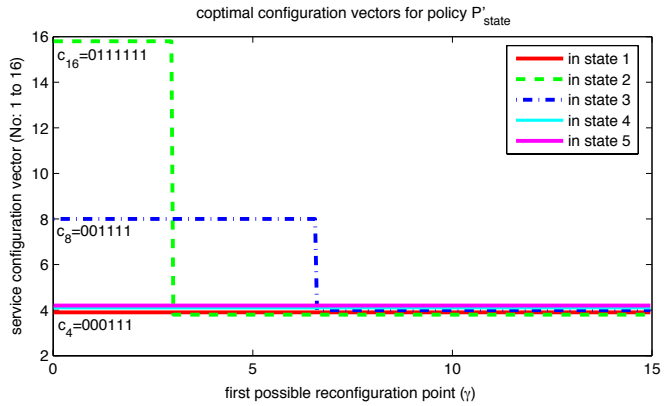

Fig. 6. Optimal configurations for 'split' topology under policy P'_{state}

increase the service performance in the 'linear' and 'split' case, while the benefit is only marginal in the 'star' case.

We focused only on investigating one reconfiguration, however, using the phased approach, the analytic model can be extended to multiple reconfigurations during runtime. Furthermore, the search for optimal reconfiguration policies is currently performed via exhaustive search, enumerating all target configurations and reconfiguration times. Optimized search strategies need to be developed for on-line use of the reconfiguration policy search in settings with large number of service components. In online use of the model-based target configuration calculation, system parameters may be incorrectly estimated; the robustness of the model-based policies to inaccurate input parameters will be investigated as a next step. A systematic case study of realistic real-life service examples using Phase-type approximations of empiric distributions of application component execution times and of network behavior together with a detailed investigation of the computational effort in online deployments will conclude this work in future.

ACKNOWLEDGMENTS

This work has been supported by the Austrian Government and by the City of Vienna within the competence center program COMET.

REFERENCES

[1] K. Bennett, P. Layzell, D. Budgen, P. Brereton, and M. M. L. Macaulay, "Service-based Software: The Future for Flexible Software," in *7th Asia-Pacific Software Engineering Conference*. IEEE Computer Society Press, 2000, pp. 214–221.

[2] R. Gabner, K. A. Hummel, and H.-P. Schwefel, "Modeling movable components for disruption tolerant mobile service execution," *Diaz, M.; Avresky, D.; Bode, A.; Bruno, C.; Dekel, E. (Eds.) CloudComp 2009. LNCS*, vol. 34, no. 1, pp. 231–244, 2010.

[3] T. Imielinski and B. Badrinath, "Mobile Wireless Computing: Challenges in Data Management," *Communications of the ACM*, vol. 37, no. 10, pp. 18–28, 1994.

[4] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, pp. 50–58, April 2010.

[5] B.-G. Chun and P. Maniatis, "Augmented smartphone applications through clone cloud execution," *HotOS'09: Proceedings of the 12th conference on Hot topics in operating systems*, May 2009.

[6] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, ser. MobiSys '10. New York, NY, USA: ACM, 2010, pp. 49–62.

[7] R. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen, and H.-I. Yang, "The case for cyber foraging," in *Proceedings of the 10th workshop on ACM SIGOPS European workshop*, ser. EW 10. New York, NY, USA: ACM, 2002, pp. 87–92.

[8] R. K. Balan, M. Satyanarayanan, S. Y. Park, and T. Okoshi, "Tactics-based remote execution for mobile computing," in *Proceedings of the 1st international conference on Mobile systems, applications and services*, ser. MobiSys '03. New York, NY, USA: ACM, 2003, pp. 273–286.

[9] W. Chou and L. Li, "WIPdroid A Two-way Web Services and Real-time Communication Enabled Mobile Computing Platform for Distributed Services Computing," in *International Conference on Services Computing*. IEEE Computer Society Press, 2008, pp. 205–212.

[10] M. Amoretti, M. C. Laghi, F. Tassoni, and F. Zanichelli, "Service migration within the cloud: Code mobility in sp2a," *High Performance Computing and Simulation (HPCS), 2010 International Conference on*, pp. 196 – 202, 2010.

[11] R. K. Balan, D. Gergle, M. Satyanarayanan, and J. Herbsleb, "Simplifying cyber foraging for mobile devices," in *Proceedings of the 5th international conference on Mobile systems, applications and services*, ser. MobiSys '07. New York, NY, USA: ACM, 2007, pp. 272–285.

[12] C. Mascolo, G. P. Picco, and G.-C. Roman, "Codeweave: Exploring fine-grained mobility of code," *Automated Software Eng.*, vol. 11, pp. 207–243, June 2004.

[13] M. Mamei, F. Zambonelli, and L. Leonardi, "Tuples on the air: A middleware for context-aware computing in dynamic networks," in *Proceedings of the 23rd International Conference on Distributed Computing Systems*, ser. ICDCSW '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 342–. [Online]. Available: http://portal.acm.org/citation.cfm?id=839280.840565

[14] A. Carzaniga, G. P. Picco, and G. Vigna, "Is code still moving around? looking back at a decade of code mobility," in *Companion to the proceedings of the 29th International Conference on Software Engineering*, ser. ICSE COMPANION '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 9–20.

[15] P. Bracchi, B. Cukic, and V. Cortellessa, "Performability modeling of mobile software systems," *Software Reliability Engineering, International Symposium on*, vol. 0, pp. 77–88, 2004.

[16] J. Gronbak, H.-P. Schwefel, and T. Toftegaard, "Model based evaluation of policies for end-node driven fault recovery," in *Design of Reliable Communication Networks, 2009. DRCN 2009. 7th International Workshop on*, October 2009, pp. 367 –374.

[17] V. Grassi and R. Mirandola, "Derivation of markov models for effectiveness analysis of adaptable software architectures for mobile computing," *IEEE Transactions on Mobile Computing*, vol. 2, pp. 114–131, 2003.

[18] V. Grassi, R. Mirandola, and A. Sabetta, "Uml based modeling and performance analysis of mobile systems," in *Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, ser. MSWiM '04. New York, NY, USA: ACM, 2004, pp. 95–104. [Online]. Available: http://doi.acm.org/10.1145/1023663.1023683

[19] L. Lipsky, *QUEUEING THEORY: A Linear Algebraic Approach*, 2nd ed. Springer, 2009.