On Data-driven Network Performance Modeling for Mobile Cloud Computing

Karin Anna Hummel
Johannes Kepler University Linz
Email: karin_anna.hummel@jku.at

Rene Gabner
Johannes Kepler University Linz
Email: k0849674@students.jku.at

Hans-Peter Schwefel
Aalborg University
Email: hps@es.aau.dk

***Abstract*—Computationally intensive mobile apps may be migrated to a cloud infrastructure for faster remote execution. Decreased execution time and lower energy consumption at the mobile device are the expected benefits when offloading the application to the cloud. The migration decision can be taken based on a continuous-time Markov model that considers network quality, cloud and mobile device capabilities, as well as migration costs, as we have shown in previous work. One of the influencing dynamic characteristics is the network performance. In this work, we focus on characterizing network performance under node mobility in terms of throughput and latency. Our final goal is to derive a mobile performance model that goes beyond an on-off network model. The analysis is based on performance measurements taken on a train while commuting. By clustering the measurement data, we derive a realistic network model.**

I. Introduction

The growing ubiquity of mobile devices such as smartphones, watches, and glasses, gives rise to a variety of networked and cloud-supported applications in the domains of gaming, daily-activity monitoring, and digital image processing, to name a few. The paradigm corresponding to these systems is *mobile cloud computing*, which extends the capabilities of the mobile device by offloading computing tasks and data from the mobile device into a more powerful infrastructure – the cloud [1], [4]. Resulting savings in computation and energy consumption at the mobile devices are desirable features provided by mobile cloud computing.

Yet, access to the application during temporary weak or intermittent connectivity is also desired. The concept of dynamic *code and data migration* allows to place code and data optimally, either on the mobile device or in the cloud. This way, connectivity problems as well as slow computation can be alleviated. Code migration is a well-known concept and has a long tradition in the field of networked systems [3], [8].

Modern code and data migration is supported by runtime migration frameworks that strive for various optimization goals, such as minimization of execution time, reduction of energy consumption at the mobile device, reduction of network communication and delays, and combinations of these goals. An exhaustive overview of migration frameworks can be found in [7]. One example framework is MAUI [4].

To decide whether and when to migrate, a Markov model can suggest optimal solutions during runtime, as we have shown in previous work [5]. Related to our approach, Markov models are also employed to describe file transfer under imperfect network state diagnosis [6] and to improve reliability by investigating whether certain time constraints are met [2].

Among others, communication is a key influencing and challenging factor for migration. Thus, we aim at realistic modeling of network performance under node mobility. We show that a model derived from real data can be integrated with a Markov model of a mobile cloud application. We extend our previous work [5] and the state of the art, firstly, by presenting an analysis of real performance traces collected during train rides. Secondly, we provide a method based on clustering to derive network performance states from the real data collected on the move and, thirdly, we discuss how the model can be employed.

II. Mobile Cloud Computing System

A mobile cloud computing system consists of mobile devices, the cloud infrastructure, and network links between the mobile devices and the cloud. Applications may be executed in a runtime environment on the mobile device or in the cloud – or may be migrated between the client and the cloud.

A. Infrastructure

Figure 1 visualizes the main components of a mobile cloud computing system. Mobile devices host apps that are primarily executed locally. These devices are characterized by CPU power, memory, and battery capacity – all limited resources. The network typically consists of a wireless access network, such as Wi-Fi or 4G, and the Internet backbone provided by network operators. In the majority of configurations, the access network can be considered as the bottleneck (wireless medium, shared access, data rate restrictions based on contracts). Finally, the cloud is characterized predominantly by its capacity, processing power, and memory. For example, the Amazon cluster provides overall computing power in the range of 100s of teraflops (10^{12} floating-point operations per second). Mobile cloud computing allows to benefit from this high-performance computing power in a cloud infrastructure by offloading computation.

While the benefits of mobile cloud computing are evident, the situation changes when network connectivity worsens. Then, it may be beneficial to run an application locally and move it back to the cloud later, requiring mechanisms to migrate code and data during runtime.

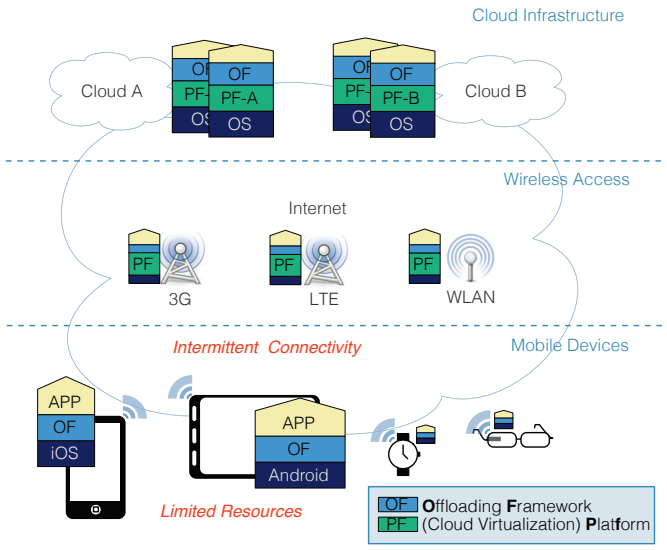

Fig. 1. Mobile cloud computing infrastructure: cloud resources are accessed by mobile users through a variety of mobile devices supported by Internet and wireless access technologies.

B. Migration of Code and Data

Mobile cloud applications can be either migrated as a whole to the virtualized cloud or split into components, which can be moved separately together with their data. Decomposing applications is more flexible, yet, more complex and usually needs support of the developer to define which parts of the application can be offloaded.

As faster movement of smaller components is beneficial in intermittent environments, we consider partitioned applications and movement of components. The placement of components is described as the *configuration* of the application. We assume an appropriate framework that supports the practical aspects of migration and runtime support on the mobile device as well as on the cloud and focus on the migration decision in the following. The migration decision aims at optimizing a specific characteristic, such as the energy consumption. We will now present how an analytical model may be leveraged to take migration decisions.

III. System Model

The system model comprises an application model and a network model. The application model is based on the components of the application and their placement as described by a *configuration vector* v, which is a bit-vector: value 0 is used for placement at the mobile client, value 1 is used for placement in the cloud. The dependencies among components are represented by the component flow sequence. One of the simplest flow sequences is a linear flow from the first to the last component of the application, where every component is visited only once, here termed *line topology* (line, star, and tree/branch topologies are described in [5]).

To find the optimal configuration vector v, the trade-off between the benefit of a configuration and possible migration costs has to be evaluated. The benefits of migration can be characterized by the savings in execution time, or energy consumption at the device provided by a possible new configuration. Costs occur in terms of additional time (also, network load and processing load). With our model, it is possible to formulate optimization goals reflecting the quality of service requirements of the application in terms of time constraints (minimizing execution time and maximizing the probability to finishing before a deadline) and energy constraints.

Yet, the migration decision depends on multiple factors. In particular, user behavior that determines the application flow and the wireless network are non-deterministic. Thus, modeling the problem in a stochastic manner is a viable approach.

A. Application Model

We represent the execution of k application components by the k states of a continuous-time Markov model with a generator matrix Q. The outgoing transition rates of each state are determined by the average duration of the execution of that component and by the probability of the subsequent component; execution times are thus considered to be exponentially distributed.[1] Final components of the application are absorbing states in the Markov model. From such a representation of the application, the computation of moments and the probability density function of the application execution time can be achieved via standard phase-type computations. A detailed description of the model can be found in [5].

To model the limited computational power of a mobile device, the Markov model is extended by a slow-down factor F when executed on the mobile device. A communication delay D for remote calls is added to the execution time and the reconfiguration delay is expressed by a random variable with an expected value of R.

B. Network Model

All delays occurring for remote calls are determined by the size of the transmitted data and the network performance. The basic network model is a simple on-off model with a failure probability p which is described in Section IV-A and extended in Section IV-D.

C. Migration Decision

To decide about which configuration to chose, we evaluate the performance of the application in two phases:

- In **phase one**, we compute the performance of the Markov model corresponding to the current configuration $c_{current}$ up until reconfiguration trigger at time T, where T is varied to find the optimal reconfiguration point.
- In **phase two**, we compute the behavior of the Markov model under the new configuration c_{new} starting at $T+R$, i.e., after reconfiguration.

Among all possible new configurations and $c_{current}$, the optimal configuration is searched for with respect to the

[1]Note that the exponential assumptions are not limiting, since they can be relaxed by the use of phase-type or matrix exponential distributions.

minimum execution time or the highest probability to reach a deadline, the minimum energy consumption, or the highest probability to finish within a given energy budget.

In detail, the algorithm performs a search over a range of reconfiguration points T. For each T, the algorithm computes the performance of the application under the current configuration $c_{current}$ and all possible new configurations, including costs for reconfiguration. By comparison, the best reconfiguration point T is selected and the best configuration, c_{new}, is picked.

The computational effort of this exhaustive search grows with the number of potential reconfiguration points and the number of components. Small applications with about ten components and 20-100 reconfiguration points can be computed in a few seconds on a common laptop. Yet, there is a need to limit the search effort for large-scale applications.

IV. On Realistic Network Characterization

The network behavior is integrated in the continuous-time Markov model by adding network-related states. We start with describing the integration of a simple on-off network model. Then, we describe how multiple network quality levels can be derived from real data and further integrated with the application model.

A. On-Off Network Model

The network behavior can be expressed by a model that mimics on-off behavior. At instances of network communication, which occur exclusively at remote component calls in the described model, the network is down with a probability of p and an exponentially distributed additional delay D is added to the overall delay. In case the network is up (with probability $1-p$), the delay for the remote call is proportional to the size of the data that are transferred. Such behavior is included in the Markov model by adding two extra states for each remote component transition as visualized in Figure 2.

This network model is a coarse-grained abstraction of the real network behavior. Though it is useful for analytical investigations, a more realistic network model with different levels of network quality is desired and will now be derived from experimental data.

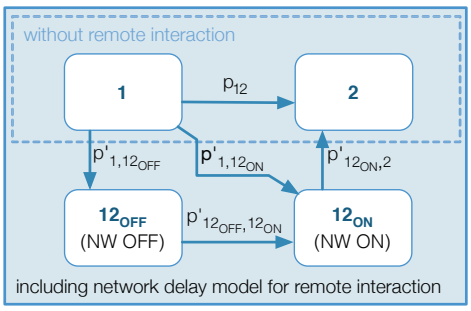

Fig. 2. Extension of the continuous-time Markov model by an on-off network behavior: The transition from state 1 to state 2 may be either direct (local execution) or through states 12_{ON} and 12_{OFF} (remote execution), modeling the case that a connection is available or not; the transition probabilities depend on the failure probability p and the corresponding off-time.

Fig. 3. Mobility trace of measurement campaign: 153 km train-ride in Austria during one hour (March 1, 2018).

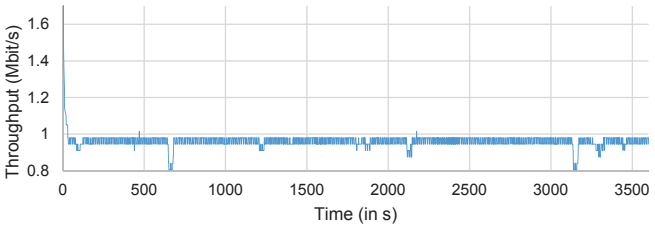

Fig. 4. Throughput sample time series of one hour (March 1, 2018); values are smoothed with a moving average window of 20 s.

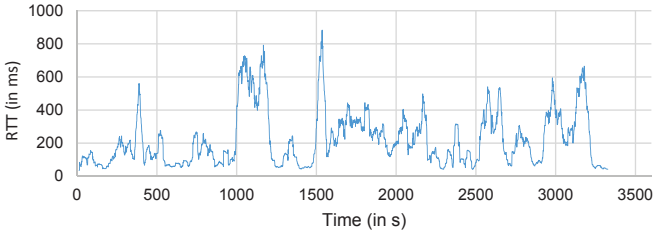

Fig. 5. Round trip time (RTT) sample series about one hour (March 1, 2018); values are smoothed with a moving average window of 20 s.

B. Experiment Setup

Measurements are taken on ten one-hour trips by train in Austria. The mobile measurement device is connected either directly through 3G/LTE or onboard Wi-Fi. The iperf tool[2] is used to measure standard TCP throughput to a server hosted by Johannes Kepler University of Linz, Austria. The command line tool ping measures the latency in terms of round trip time (RTT) to this same server. Network measurements are taken once per second. Overall, 32081 measurements are collected.

Figures 3,4, and 5 show the mobility and performance traces of one experiment. One can observe that the round trip time varies more than the throughput that is dependent on the limitations given in the access network. Nevertheless both measures show decreased performance several times during the trip, corresponding to tunnels and weak connectivity regions during the ride. (Note that we did not aim at measuring maximum throughput possible, but we were interested in continuous probing while moving.)

[2]https://iperf.fr/

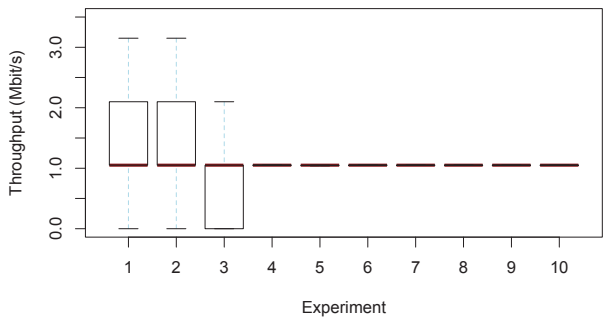

Fig. 6. Throughput measurements depicted in boxplots of each of the ten experiments (about 3000 observations per experiment): median (red horizontal line), quartiles (box horizontal boundaries), and outliers (whisker lines extending the boxes vertically).

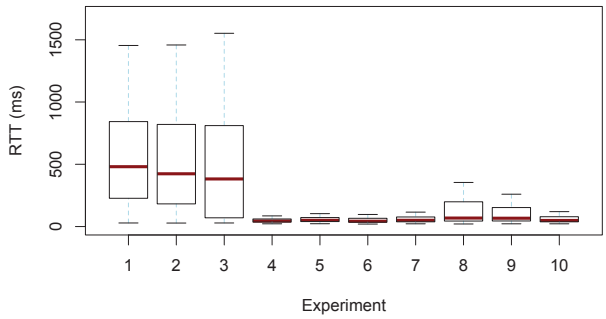

Fig. 7. Round trip time measurements depicted in boxplots of each of the ten experiments (about 3000 observations per experiment): median (red horizontal line), quartiles (box horizontal boundaries), and outliers (whisker lines extending the boxes vertically).

C. Measured Network Performance

A summary of the ten experiments is shown in Figures 6 and 7. Only small variations in throughput can be detected. Round trip time varies more and is expected to be the primary indicator of network quality degradation. The first three experiments were performed by connecting directly to 3G/LTE while the others were conducted using the on-board train Wi-Fi. It can be observed that the direct 3G/LTE measurements show higher variability and generally weaker results.

To derive and configure the network states, we employ moving averages and clustering. As network performance is not influenced by a single peak performance degradation, the raw values are transformed by a moving average algorithm with a window size of 20 s (trials with varying window sizes of 10 s, 20 s, and 30 s show similar results). K-means clustering with two to 20 clusters is performed and evaluated by analyzing the decrease of the sum of squares within the clusters. The typical flattening of this curve indicates to select a cluster size of three, where the ratio of the sum of squares in a cluster and the overall sum of squares yields 0.89. We find that the round trip time (latency) is the predominant factor, whereas the throughput varies heavily in each cluster. Table I summarizes the clustering results.

State (cluster)	TP (Mbit/s)	RTT (ms)	Fraction
$S_1(C_1)$	1.03	104.86	0.72
$S_2(C_2)$	1.32	553.82	0.15
$S_3(C_3)$	1.46	1192.87	0.07
S_4	OFF	OFF	0.06

TABLE I
STATES DERIVED FROM CLUSTERS, CORRESPONDING THROUGHPUT (TP) AND ROUND TRIP TIME (RTT) CLUSTER CENTERS, AND FRACTION OF MEASUREMENTS PER CLUSTER.

D. Multiple Network States

Based on the clustering result, the on-off model depicted in Figure 2 is extended. The network on-state 12_{ON} is replaced by the three states S_1, S_2, and S_3, corresponding to the derived clusters summarized in Table I. The off-state is $S_4 = 12_{\text{OFF}}$.

The transition probabilities to change in one of the on-states are $p_1 = 0.72$, $p_2 = 0.15$, and $p_3 = 0.07$, respectively, where

$$\sum p_i = p'_{1,12_{\text{ON}}} = 1 - p.$$

The transition probability to change into the off-state is $p'_{1,12_{\text{OFF}}} = p = 0.06$. The remote component call delay D can be calculated based on the datasize to be transmitted, the round trip time (RTT) and the throughput. As the round trip time is the predominant factor, we add it to the transmission delay of the remote call data weighted by a factor f_w:

$$D = f_w \times \text{RTT} + \frac{\text{Datasize}}{\text{Throughput}}. \quad (1)$$

It has to be noted that the number of clusters as well as the measured round trip time and throughput are specific for the train ride use case. However, applying clustering and using the information from the clustering results to build a multi-state network model is of general relevance.

V. NUMERICAL RESULTS

To demonstrate the potential of the mobile cloud computing model, we describe its behavior for a simple synthetic application consisting of six components $C_1, ...C_6$, where the component flow is sequential (line topology). C_1 refers to the user interface fixed to the mobile device, C_6 is the final state storing the results and is bound to the infrastructure. We assume that the first three components are initially placed on the mobile device and the last three in the cloud, so that $v = (000111)$. We allow one reconfiguration and investigate the best point in time to reconfigure among 20 reconfiguration points. The slow down factor for computation on the mobile device is $F = 0.1$ and the delay added when in off-state is 5 s. Among possible optimization goals, we select the probability to finish within a given energy budget.

We present numerical results achieved with the on-off network model, and the impact of the more advanced four-state network model.

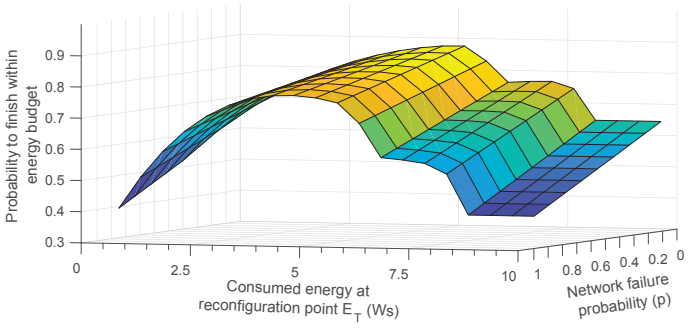

Fig. 8. On-off Network Model: Probability to finish the application within the given energy budget of 10 Ws, with varying reconfiguration point defined by E_T, the energy consumed until a point in time T, and varying network failure probability p.

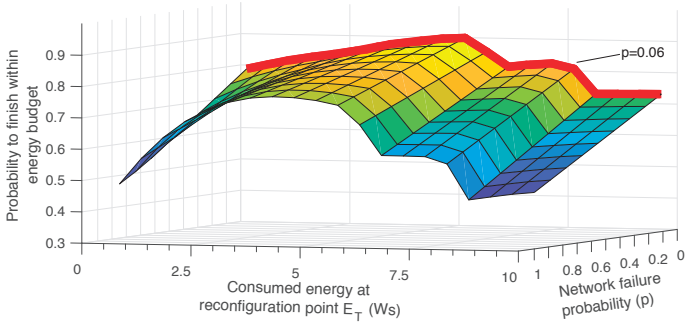

Fig. 9. Four-state Network Model: Probability to finish the application within the given energy budget of 10 Ws, with varying reconfiguration point defined by E_T, the energy consumed until a point in time T, and varying network failure probability p; red line depicts the curve at $p = 0.06$ (train ride dataset).

A. On-Off Network Model

The on-off model is configured by the remote execution delay $D = 4.03$ s, which is calculated by applying Equation 1 to averages of the RTT and throughput of the on-states (Table I), where Datasize $= 150$ kByte and RTT factor $f_w = 5$.

Figure 8 shows the influence of the network failure probability p and the reconfiguration point on the probability to finish within the energy budget. The maximum finishing probability calculated by the model is 0.88 at $E_T = 5.5$ Ws, network failure probability p lowest. The minimum finishing probability is 0.41. A decrease of p generally increases the finalization probability (this is expected). In case of high failure probability p, reconfiguration at the right instance of time has the highest impact on the finalization probability because there are only a few optimal reconfiguration points, all around $E_T = 5$ Ws. Late reconfiguration is not beneficial.

B. Four-state Network Model

To show the potential of the four-state network model, we vary the failure probability p again from 0 to 1 but keep the relation between the three on-states as derived from the number of cluster members, resulting in: $p_1 = (1-p) \times 0.77, p_2 = (1-p) \times 0.16$, and $p_3 = (1-p) \times 0.07$. The delay in state S_i is named D_i and calculated as stated in Equation 1, where Datasize $= 150$ kByte and RTT factor $f_w = 5$. This results in $D_1 = 1.67$ s, $D_2 = 3.68$ s, and $D_3 = 6.79$ s.

Figure 9 shows the probabilities to finish within the energy budget using the four-state network model. The finishing probabilities calculated by the model range from 0.45 to 0.91 with a peek at $E_T = 5.5$ Ws, low network failure probability. Compared to the on-off model (Figure 8), the finishing probability is generally higher. This is due to the dominant state S_1 (highest entering probability) with lowest round trip time, which correctly results in higher finishing probabilities.

The figure also shows the resulting probability to finish with the exact state configuration derived from the train ride dataset (red line in the figure, $p = 0.06$). On the train, the networking conditions allow to finish within our selected energy budget with probabilities ranging from 0.74 to 0.91.

VI. Conclusion

We showed that a continuous-time Markov model can be used to describe the application flow with application components either placed on a mobile device or in the cloud. The migration decision is based on searching for the best placement of application components in terms of probability to finish within a specific energy budget. As the network performance is a key characteristic in the model, we analyzed real performance measurements taken on train rides and clustered them along throughput and latency (round trip time). In our experiment setting, three network on-states are derived from the results of clustering which allows a more fine-grained modeling of the network states than with a binary on-off network model with impact on the overall model outcome.

References

[1] 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. *Comm. of the ACM*, 53:50–58, 2010.

[2] D.R. Avresky, S. J. Geoghegan, and Y. Varoglu. Evaluation of Software-Implemented Fault-Tolerance (SIFT) Approach in Gracefully Degradable Multi-Computer Systems. *IEEE Trans. on Reliability*, 55(3):451–457, 2006.

[3] A. Carzaniga, G.P. Picco, and G. Vigna. Is Code Still Moving Around? Looking Back at a Decade of Code Mobility. In *Comp. Proc. of 29th Int. Conf. on Software Engineering*, ICSE COMPANION '07, pages 9–20. IEEE, 2007.

[4] E. Cuervo, A. Balasubramanian, D. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl. MAUI: Making Smartphones Last Longer with Code Offload. In *Proc. of 8th Int. Conf. on Mobile Systems, Applications, and Services*, MobiSys'10, pages 49–62. ACM, 2010.

[5] R. Gabner, H.-P. Schwefel, K.A Hummel, and G. Haring. Optimal Model-Based Policies for Component Migration of Mobile Cloud Services. In *Proc. of 10th IEEE Int. Symp. on Network Computing and Applications*, NCA '11, pages 195–202. IEEE, 2011.

[6] K. Hjgaard-Hansen, T. K. Madsen, and H.-P. Schwefel. Addressing the Influence of Hidden State on Wireless Network Optimizations using Performance Maps. In *Proc. of 15th Int. Conf. on Next Generation Wired/Wireless Networking*, NEW2AN '15. Springer, 2015.

[7] A.R. Khan, M. Othman, S.A. Madani, and S.U. Khan. A Survey of Mobile Cloud Computing Application Models. *IEEE Communications Surveys Tutorials*, 16(1):393–413, 2014.

[8] C. Mascolo, G.P. Picco, and G.-C. Roman. CODEWEAVE: Exploring Fine-Grained Mobility of Code. *Automated Software Eng.*, 11:207–243, 2004.