Utilizing Networked Mobile Devices for Scientific Simulations

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von

Christoph Benjamin Dibak

aus Schorndorf

Hauptberichter:	Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel
Mitberichter:	Prof. Dr.-Ing. Wolfgang Nowak
	Prof. Dr. Christian Becker

Tag der mündlichen Prüfung: 10.01.2020

Institut für Parallele und Verteilte Systeme (IPVS)
der Universität Stuttgart
2020

Acknowledgements

This thesis would not have been possible without the support by so many people. First of all, I would like to thank my doctoral advisor and mentor Prof. Kurt Rothermel, for his guidance, valuable feedback, helpful support, and the opportunity to work on this interesting topic at the IPVS. Furthermore, I would like to thank my co-supervisor Prof. Wolfgang Nowak for his helpful comments and support, especially during the initial part of my studies and for the collaboration that lead to the work on surrogate models and data assimilation. I would also like to thank Prof. Christian Becker for kindly acting as a referee for this work.

The scientific collaboration leading to the publications on the distributed reduced basis method would not have been possible without Prof. Bernard Haasdonk and Dr. Andreas Schmidt. Thank you very much for you detailed explanations and helpful conversations.

I would like to thank Prof. Nigel Davies and Mateusz Mikusz for providing the opportunity for the three-month visit to Lancaster University, which has been an inspiring, but also enjoyable experience.

Research requires frequent interaction with senior researchers. Special thanks to Dr. Frank Dürr for many interesting and long discussions and for valuable feedback. I would also like to thank Dr. Boris Koldeholfe for supporting me during the initial year of my PhD studies.

The workplace is not the same without so many supportive and like-minded colleagues. I would like to thank Thomas Kohler for sharing the office and Eva Strähle for all the support in administrative matters. Thanks to all of the VS team for sharing coffee and thrilling kicker experiences. I would also like to thank my colleagues from the SimTech Graduate School for lots of interesting conversations during our Stammtisch and PhD-Student Weekends.

Science is not possible without financial support. I would like to thank the DFG for funding within the Cluster of Excellence in Simulation Technology (EXC 310/2). Clearly, this thesis would not have been possible without the establishment of this interdisciplinary research cluster.

Finally, I would like to thank my family and friends for their encouragement, support, and for providing the necessary distraction to re-focus on scientific matters. Last but certainly not least, I would like to thank Andrea for her love and support on good and bad days, and for always finding ways to cheer me up when I was struggling with my work.

Abstract

Numerical simulations on mobile devices create new applications supporting engineers and scientists in the field. Boosted by novel augmented reality devices, in-field analysis of complex systems allow engineers to make better decisions and predict the behavior of such systems by assuming different parameters before making risky and costly decisions.

Mobile simulations are challenging as battery-powered mobile devices are only equipped with slow processors and are limited in energy resources. At the same time, mobile devices are only connected via wireless communication subjected to environmental conditions that might cause slow bandwidths or even disconnections to remote computing resources. Nevertheless, concepts presented in this thesis assume a distributed computation between mobile device and a powerful remote server.

This thesis covers three major areas of the research field of mobile simulations. First, it provides concepts for distributed execution between server and mobile device in case of frequent disconnections. Second, it provides concepts using computationally less complex surrogate models for faster computation on the mobile device while still utilizing remote resources. Third, it provides concepts utilizing model order reduction for fast execution on mobile devices by pre-computing and adaptation of reduced models on a connected server.

Evaluations show that concepts presented in this thesis significantly increase the performance of mobile simulations. In the case of disconnections, the number of deadline misses is reduced by 61 % while reducing the energy consumption by more than 74 % compared to a simplified approach. Concepts utilizing surrogate models speed-up the computation of the simulation by a factor of 6.5. Lastly, concepts utilizing model order reduction reduce the time for the computation of simulation results by a factor of 131 while using 73 times less energy for the specific test application.

Zusammenfassung

Die Ausführung von numerischen Simulationen auf Mobilgeräten ermöglicht neue Anwendungen um Ingenieure und Wissenschaftler vor Ort zu unterstützen. Solche mobile Simulationen ermöglichen es die Auswirkungen von Änderungen an komplexen Systemen vor Ort zu untersuchen und dadurch Risiken zu senken bevor teure oder zeitaufwändige Änderungen vorgenommen werden. In Kombination mit neuesten Augmented-Reality Geräten können Ingenieure und Wissenschaftler so mehr Erkenntnisse über ein komplex System sammeln und somit bessere Entscheidungen treffen.

Mobile Simulationen zu realisieren ist mit vielen Herausforderungen verbunden. Batteriegetriebene Mobilgeräte sind mit langsamen Prozessoren ausgestattet und verfügen nur über stark limitierte Energieresourcen. Gleichzeitig sind solche Geräte nur über drahtlose Kommunikation verbunden, welche externen Einflüssen ausgesetzt ist. Dadurch kann drahtlose Kommunikation keine festen Bandbreiten garantieren und es kann zu zeitweisen Verbinungsabbrüchen kommen. Trotzdem schließen Konzepte, die in dieser Arbeit vorgestellt werden, externe Rechenressourcen mit ein, da diese unerlässlich für eine Ausführung von komplexen Simulationen sind.

Diese Arbeit deckt drei Gebiete des Forschungsgebiets über mobile Simulationen ab. Zuerst werden Konzepte für die verteilte Ausführung zwischen Server und Mobilgerät im Falle von zeitweisen Verbinungsabbrüchen vorgestellt. Zweitens werden Konzepte vorgestellt, welche weniger komplexe Stellvertretermodelle zu komplexen Simulationsmodellen ausnutzen um eine schnellere Berechnung auf dem Mobilgerät zu ermöglichen. Drittens stellt diese Arbeit Konzepte vor um Ergebnisse aus dem Forschungsgebiet der Modellreduktion auf mobile Simulationen zu übertragen, um durch Vorberechnung und verteilte Adaption der reduzierten Modelle eine schnellere Ausführung ermöglichen.

Untersuchungen haben gezeigt, dass die Konzepte dieser Arbeit die Leistungsfähigkeit des verteilten Gesamtsystems bei der Ausführung von mobilen Simulationen deutlich verbessern. Im Fall von zeitweisen Verbinungsabbrüchen sind

die hier vorgestellten Verfahren in der Lage gegenüber vereinfachten Verfahren, vorbestimmte Zeitschranken in weiteren 61 % der Fälle einzuhalten und mehr als 74 % an Energie einzusparen. Konzepte, welche vereinfachte Simulationsmodelle ausnutzen beschleunigen die Simulation um einen Faktor von 6.5. Zuletzt konnten wir zeigen, dass Konzepte, welche Modellreduktionsmethoden verwenden die Berechnung um einen Faktor von 131 beschleunigen und 63 fach weniger Energie benötigen.

Contents

1

Introduction

1.1 Motivation

With the 2007-introduced iPhone by Apple, ubiquitous and mobile computing devices revolutionized the interaction with software and enabled the development of new disruptive services such as Uber, Foodora, or Spotify. Those services require either the customer or part of the service to be mobile and therefore not only changed our private lives, but also the workplace for many people. With emerging augmented reality devices even more disruptive application are to be expected in our private and working lives. Augmented reality enables virtual objects to be overlayed over the real world and allow not only ubiquitous access to software but a novel form of interaction that constantly adapts to the world around us.

Another, more long-term, disruptive technology from computer science are numerical simulations. As of today, simulations are heavily used for engineering and scientific applications, especially for applications where human lives or financial interests are at high risk. Therefore, in engineering, today's rapid development of new products would not be possible without numerical simulations and virtual prototyping [RWMS18]. In science, simulations are established as the third form for gaining scientific knowledge besides experiments and theory. They are required to validate theories and provide estimates, especially in situations where experiments are not possible as the scenario is unobservable. For instance, when the scenario is unlikely or unique (e.g., the big bang), too long (e.g., climate change, atomic waste disposal), too short (e.g., combustion processes in engines),

too big (e.g., weather forecasts, the expanse of the universe), or too small (e.g., chemical reactions, cancer cells).

The goal of this thesis is to combine ubiquitous computing and numerical simulations to *mobile simulations* enabling novel applications to assist engineers, scientists, and decision-makers in the field. Such mobile simulations enable better decision making, e.g., when an engineer requires to decide on changing some parameters, and even better interaction, e.g., when an architect discusses different options how to react to unexpected incidents with her client directly on site using augmented reality devices.

As a scenario for mobile simulations, we consider an engineer in a machine hall during the assembly of new machinery. While most steps will be long planned before the assembly process, unexpected incidents requiring on-site decisions are to be expected. In this scenario, the engineer has to place a hot tube in accordance with the heat resistance of surrounding materials. Using mobile simulations on her augmented reality glasses, the engineer can see how heat would spread inside the tube and propagate to surrounding materials as if the machine would be operational. In addition, she can change parameters reflecting different conditions on the operation of the machine and surroundings (e.g., air flow, temperature, and pressure). Some parameters can be taken in accordance to sensor readings available to the mobile device, or collected in the cloud, e.g., from similar conditions at other locations. Using mobile simulations, the engineer can choose a placement for the tube that optimally fits and complies with its surroundings.

Providing results of complex numerical simulations to mobile devices is challenging as such devices are limited in energy and computing resources. Mobile devices have to be small to be carried by users and they require mobile energy resources in form of a battery. This requires processors to be energy efficient and results in significantly lower performance compared to server or desktop processors. Such processors alone cannot compute complex numerical simulations. We, therefore, have to take connected computing resources into account, i.e., a remote server. However, mobile devices are connected via wireless com-

munication, which is subjected to environmental conditions leading to dynamic throughput, dynamic latency, and disconnections of the communication link, requiring even more challenges to be solved.

While past decades have seen significant research on numerical applications and ubiquitous computing as separate fields, only a few researchers combined both fields of computer science. Numerical applications were among the first applications on electronic computers and have been addressed by early computer scientists such as Turing [Tur48]. There have always been highly specialized computing facilities, i.e., supercomputers, to solve numerical problems and the field of high-performance computing is centered around such massive, scalable, but immobile computing facilities. While more computing resources were mostly invested in higher quality simulation results, recent research in the field of model order reduction also addressed applications requiring fast results and even mobile applications. However, this research is missing the utilization of powerful server resources, as well as optimizations tailored to modern mobile devices for fewer energy consumption and latency.

On the mobile computing side, Mark Weiser coined the term ubiquitous computing in the early 1990's with his work describing what we now know as tablets and smartphones [Wei91, WB96]. His idea was to make computing invisible and happen in the background supporting humans in their daily lives. While this research led to concepts for distributing complex applications in the infrastructure consisting of mobile devices and background servers, there has been only a few concepts for quality-aware applications on mobile devices, such as numerical simulations.

The goal of this thesis is therefore to combine both research fields. To this end, we will first formulate research questions in the next section, before describing the detailed contribution of this thesis, the project background, and the structure of this thesis.

1.2 Research Focus and Goals

This thesis focuses on four research areas, the analysis of simple approaches, robust distributed execution, efficient quality-aware execution, and how methods from model order reduction can be utilized. These research areas are explained in the following along with the major research questions covered in this thesis.

1.2.1 Analysis of Approaches for Mobile Simulations

Complex numerical applications require high computational resources and efficient execution methods. While mobile devices became ever more powerful providing multiple GHz of processing power and multiple processor cores, they still have to be energy efficient and therefore clearly lack behind stationary processors. However, mobile devices are equipped with fast IEEE 802.11 WiFi and LTE which can be used for offloading complex computations to remote servers. The first objective is therefore to analyze the limitations of modern mobile devices for scientific simulations that are either completely offloaded or executed on the mobile device itself. The main research questions are:

1. How efficient is the execution solely on the mobile device?
2. What components need to be identified in a distributed architecture to include server resources in the simulation?
3. How efficient is streaming of results from nearby servers to the device?

1.2.2 Robust Distributed Execution for Soft Real-Time Simulations

In order to provide timely results for the user, we need to define the mobile simulations as soft real-time applications. Soft real-time applications should provide the result before a given deadline. However, in contrast to hard real-time applications, deadline misses are undesirable but not forbidden, e.g., in soft real-time applications, the user still has some value in receiving the solution after the

deadline, wherein hard real-time applications receiving the solution after the deadline has no value at all.

Distributed processing between the server and mobile device faces the problem of frequent disconnections of the wireless communication channel. We assume that wireless communication channels are only unavailable for a limited time and are eventually recovered. In this setup, we consider the following questions:

4. How often is the mobile device disconnected from the server? We consider the mobile device to be disconnected if multiple packets sent from the server never arrive.
5. How can we timely detect disconnections?
6. Assuming a soft real-time mobile simulation, how can we reduce the number of deadline misses in case of frequent disconnections, i.e., how to react on detected disconnections?

1.2.3 Efficient Quality-aware Execution

Quality plays an essential role in simulations. Intuitively, the higher the quality, the higher the complexity of the computation. Depending on the application, too low-quality results are useless, while too high-quality results waste resources. Additionally, in mobile environments, resources are dynamically available. For instance, the data rate in LTE depends on the current location, usage of other subscribers in the same cell, and physical properties such as humidity [LLM+09]. To provide the best experience, i.e., fast simulation results, to the user, the following research questions need to be answered:

7. How to provide a method for users to define quality constraints for mobile simulations?
8. How to utilize available resources for fast distributed computation of simulation results for the mobile user?
9. Given a low data rate wireless communication link, how can we provide fast simulation results to the user that still fulfill quality constraints?

1.2.4 Utilizing Model Order Reduction

Model order reduction (MOR) concepts are already established in the field of numerical simulations for building reduced problems providing an approximate solution to the original problem [Haa16]. The general idea is to train a reduced model in a pre-computation phase to provide desired quality constraints during runtime. To this end, training data needs to be available before the generation of the reduced model. This training data consist of the parameter range of the simulation that will be used during the execution. While MOR has been intensively studied in high-performance computing (HPC), only a few researchers applied such concepts in mobile computing settings. For our ubiquitous computing scenario, the following open research questions remained:

10. How to efficiently distribute the computation between mobile device and server? How to assess the communication overhead depending on the size of the reduced model?

11. Often, the parameter range for simulations is not known before the execution. How to enable quality constraints for dynamic query ranges that are not available at the pre-computation phase?

12. How can energy consumption and latency be reduced during runtime? Can the distribution be modified to provide results with lower energy consumption or lower computational complexity?

13. Can we generate application dependent reduced models to provide results with fewer resource consumption, i.e., can we modify the pre-computation phase to improve the execution during runtime on the mobile devices?

As we have now defined the scope and research questions of this thesis, the next section will provide details about the contributions of methods.

1.3 Contributions

This thesis combines and extends previously published results presented in [DK14, DDR15, DSD+17, DDR17, DHS+18, DNDR19]. The major contributions are sorted into concepts for robust execution, concepts utilizing surrogate models, and concepts utilizing model order reduction (MOR).

Concepts for Robust Execution

This thesis provides concepts for the distributed execution of time-based numerical applications between mobile device and server in case of disconnections. Providing a soft real-time deadline, presented methods are able to reduce the deadline misses by using prediction of the availability of the network. Another method additionally provides lower energy consumption in such scenarios. Evaluations of approaches were based on real-world data taken from cellular networks in the Stuttgart area and show that approaches are able to keep deadline constraints in more than $61\,\%$ while saving up to $74\,\%$ of energy compared to a simplified approach.

Concepts for robust execution have been published in [DDR15]. The author of this thesis contributed around 80 % of the scientific content.

Concepts Utilizing Surrogate Models and Data Assimilation

Second, this thesis provides novel concepts for the distributed execution of quality-aware numerical applications using low-quality surrogate models and data assimilation techniques. To this end, this thesis provides an intuitive definition of quality based on reference configuration and surrogate configuration required on the mobile device. Additionally, this thesis provides methods to guarantee quality constraints on the mobile device with drastically reduced requirements for the wireless communication channel. Evaluations using a test bed with emulated wireless communication channel and system-on-chip (SoC) devices show that our approaches are able to speed up the distributed computation by a factor of 6.5 while reducing required data rates of the wireless

communication channel.

Concepts for utilizing surrogate models for mobile simulations are currently in submission [DNDR19]. The author of this thesis contributed 80 % of the scientific content.

Concepts Utilizing Model Order Reduction

Third, this thesis provides methods utilizing methods from model order reduction (MOR) for parameterized stationary simulation problems. It provides several methods for different situations, e.g., an adaptive method to provide guaranteed quality for untrained parameter ranges, methods to reduce energy consumption and runtime in the online phase, and a method to provide better reduced models for faster and energy efficient computation during runtime. Methods can be combined and our evaluations show that we are able to speed up the computation by over 131 times while using 73 times less energy for the specific test application.

Concepts utilizing model order reduction have been published in [DSD+17], [DDR17], and [DHS+18]. The author of this thesis contributed around 70 %, 90 %, and 70 % respectively to the scientific contents of these publications.

In addition to these three major contributions, this thesis provides architectures for middleware implementations for all three scenarios and compares results with simple approaches to either compute everything on the mobile device or to stream simulation results from a remote server. These architectures were published in [DK14], where the author of this thesis contributed around 60 % of the scientific content.

1.4 Project Background: SimTech

The research project for mobile simulations has been funded by the German Research Foundation (DFG) at the Cluster of Excellence in Simulation Technology (SimTech) at the University of Stuttgart. SimTech was established in 2007 as an effort to combine and bring together many institutes of the University of Stuttgart to support interdisciplinary research on and with simulations. SimTech's goal is to

promote simulations to an integrative systems science, instead of having isolated numerical approaches, which still is the way many engineers and scientists use simulations as of today.

To support the goal of providing an integrative system science, SimTech identified five visions: (V1) Computational Material Design, including simulation-based optimization for highly sophisticated new materials; (V2) Integrative Virtual Prototyping, supporting design and development for modern production; (V3) Interactive Environmental Engineering, combining simulations, data assimilation, optimization, and risk assessment for applications like carbon dioxide (CO_2) injection into deep geologic formations; (V4) Simulation Cyber Infrastructures for using modern cloud and mobile infrastructures to support engineers and scientists using simulation technologies; and (V5) Overall Human Model, to understand and influence biological processes inside humans to heal diseases.

Towards these visions and goals, research is structured into 7 project networks (PNs), combining nearly 70 individual projects. The PNs are (PN 1) Material Design: Multi-scale and Multi-field Simulations of Materials; (PN 2) High-Performance Simulations Across Computer Architectures; (PN 3) Dynamical Systems: Reduction Optimization and Control; (PN 4) Coupled Problems in Biomechanics and Systems Biology; (PN 5) Multi-phase and Multi-physics Modeling; (PN 6) Cyber Infrastructures and Beyond; and (PN 7) Reflexion and Contextualisation.

Each PN consists of multiple projects. For instance, PN 6, Cyber Infrastructures and Beyond, where the mobile simulation project was incorporated, consisted of 7 Projects: (1) Modeling of Multi-Scale and Multi-Physics Simulations; (2) Execution of Multi-Scale and Multi-Physics Simulations; (3) Bootware: Efficient Execution of Uncertain Computations; (4) Data Provisioning for Data-Driven and Ubiquitous Simulations; (5) Utilizing Networked Mobile Devices for Scientific Simulations; (6) Interactive Visual Analysis of Big Simulation Data; and (7) Natural Interaction with Ubiquitous Simulation Systems. Projects (1) to (3) focused on providing scientific workflows for modeling of experiments and execution in the cloud. Project (4) focused on how simulation data can be stored in databases and how

scientists can model data. Project (6) and (7) focused on visual user interaction with simulations and big data.

To transfer concepts researched in the cluster of excellence, the cluster not only publishes scientific papers on conferences and journals, but also exchanges results, problems, and ideas with industry. To this end, the cluster established an industrial consortium and invites speakers from industry into the SimTech Colloquium. Additionally, SimTech raises social awareness about simulation technology in the general public, e.g., with the exhibition 'In the Digital Lab', which took place in 2017 in the Carl-Zeiss-Planetarium Stuttgart.

1.5 Structure

The rest of this thesis is structured as follows: Chapter 2 provides the required background for the rest of the thesis, including model order reduction and mobile computing. Chapter 3 introduces an overview of the system, including the system model. Chapters 4 to 6 introduce concepts for mobile simulations. First, concepts for robust execution of time-dependent simulations in harsh environments with frequent disconnections are discussed in Chapter 4. Second, concepts for efficient calculation of simulations on mobile device and server for various scenarios, including low data rate scenarios are discussed in Chapter 5. Third, concepts utilizing model order reduction methods are discussed and how they can be improved for mobile simulations are discussed in Chapter 6. After introducing the concepts, the thesis is concluded with the conclusion and outlook in Chapter 7.

2

Background

This chapter presents background for numerical simulations on networked mobile devices. It discusses numerical simulations, model order reduction, data assimilation techniques, such as the ensemble Kalman filter, and background from mobile computing.

2.1 Numerical Simulations

This section provides background on numerical simulations used throughout this thesis. We assume that the simulated system is described by means of partial differential equations (PDE). To provide a better overview of the process of designing, modelling, and computing of simulations, the simulation pipeline is introduced. After the simulation pipeline, background on discretization, numerical solvers, and quality in simulations are discussed.

2.1.1 Simulation Pipeline

The simulation pipeline describes the steps for building and using simulations for engineering and scientific purposes. Different definitions of the simulation pipeline exist. The simulation pipeline described in this section is a slightly

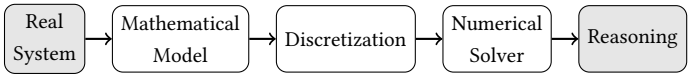

Figure 2.1: The simulation pipeline.

modified version taken from [BZBP13].

Figure 2.1 depicts a schematic representation of the major steps for mobile simulations: the real system, the mathematical model, the discretization, the solver, and the reasoning. The steps are described in more detail in the following.

Real System The real system is the system to be simulated. The real system might be observed by sensors, providing sensor data to be used in the simulation process and for verification of the simulation. As a running example throughout this section, we consider a hot metal plate containing temperature sensors.

Mathematical Model Depending on the aspects of the simulation, the mathematical model contains different aspects of the real system. For instance, if we are interested in traffic simulations of cars, we might model the behavior of individual cars on a map. However, if we are interested in the heat distribution on a hot metal plate, we are interested on a continuous phenomena. From physics, we know formulas for the heat equation. Such equations contain parameters and can be applied to different geometries. We assume parameters and geometries of the problem to be known.

Discretization The mathematical model captures the behavior of the system over continuous space at infinitely many space points. However, computers are only able to handle a limited number of space points. Therefore, the mathematical model needs to be discretized. Typical methods for discretization are finite differences or finite elements. In this thesis, only finite differences are used.

Numerical Solver After the discretization, an algebraic problem has to be solved to provide an approximate solution. Depending on the problem and its mathematical properties, a specific numerical solver has to be used. For instance, if we define the problem with the hot metal plate on sparse matrices, we require a sparse matrix solver for the specific data structure. Depending on the mathematical properties of the matrix, a specific sparse matrix solver can be chosen to guarantee fast convergence.

Reasoning In this thesis, reasoning is everything beyond the calculation of the solution of the simulation problem. For instance, reasoning can be performed by human users, requiring visualization of the solution, e.g., on an augmented reality device. Another form of reasoning can be in the context of a cyber physical system, where the solution is used to control certain aspects of the real world. In that case, reasoning is performed by the controller.

The focus of this thesis is on parts of the mathematical model, the discretization of the model, and parts of the numerical solver. The other parts, mainly reasoning and description of the system, are outside the scope of this thesis and have to be implemented by experts in the specific areas. In the following, we will shortly discuss further background on the mathematical model, the discretization, and the numerical solver.

2.1.2 Mathematical Model

This section provides a short overview on the mathematical model. Further information can be found in [BZBP13].

Partial Differential Equations

Partial differential equations (PDE) describe the behavior of the system. The system state is an unknown function u that describes the current situation of the system. The function domain of u is the temporal and spatial domain of the simulation. For simplicity, the temporal domain will be an interval $t \in [0, t_{end}]$ and the spatial domain will be the two-dimensional unit square with $x \in [0, 1]^2$.

Partial differential equations contain multiple derivatives of space and time. Intuitively, they describe how changes in time or surrounding space affect the system. For instance, the heat equation in two dimensions is given as

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}, \tag{2.1}$$

where u is the unknown function, t is time, and x and y are the axes of the two dimensional space.

There are two fundamentally different kinds of simulation models, stationary and time-dependent. Stationary simulation models do not include varying time and describe one extreme condition for certain parameters. In contrast, time-dependent simulation models describe how the system evolves over time.

Boundary Conditions

The boundary of the spatial and temporal simulation domain needs to be treated differently. Partial differential equations do not contain any information on how the boundary should be treated. The two most common boundary conditions are the Neumann conditions and the Dirichlet conditions. Dirichlet conditions set u on the boundary to a fixed value while Neumann conditions fix the first derivatives of the function in direction to the boundary. Formally, if discrete points on the boundary are given in set B and $b \in B$, Dirichlet conditions claim $u(b) = c$ for constant c. Analogously, Neumann conditions claim $u'(b) = c$, where u' is the derivative of u in direction of the boundary. There are many other boundary conditions describing different physical phenomena. However, in this thesis, we will only use Dirichlet and Neumann conditions.

Full Specification of the Simulation Problem

Using the PDE and boundary conditions, we can now formulate a full specification of the simulation problem, which in this thesis is mostly an *initial value problem*. In an initial value problem, the initial conditions of the system at time $t = 0$ are provided. Initial conditions can be provided by external measurements, e.g., sensor data, or are provided by the user. The full specification of an initial value problem can be described as

1. the PDE describing the "internal" behavior of the system;
2. the space and time domain of the simulation;

3. boundary conditions describing the behavior on the spatial boundary; and
4. initial conditions describing the conditions of the system at time $t = 0$.

Having described how the mathematical description of the simulation problem is provided, we discuss how this description can be transformed into an algebraic problem in the following section.

2.1.3 Discretization

Solutions of simulation problems described by means of partial differential equations are functions over continuous spaces. Such functions cannot be represented in digital computers. Therefore, simulation problems have to be discretized and solved on a finite number of points in space and time.

There are multiple methods for the discretization of differential equations. The most common discretization methods are finite differences, finite elements, and finite volumes. Notice that the discretization of time-dependent problems might use another discretization technique for time than for space discretization, hence there could also be a mix of methods for solving one simulation problem. All discretization techniques transform differential equations to algebraic equations, e.g., a set of linear equations. These equations can then be solved in the next step using problem specific numerical solvers (see Figure 2.1).

In this thesis, only finite differences are used. However, concepts described in this thesis can also be applied for other discretization methods. Finite differences can be directly derived from the difference quotient $(f(x+h) - f(x))/h$, which, as limit $h \to 0$, is used as the definition of the derivative of the function f at position x. The general idea of finite differences is to fix h and describe the function only at discrete positions $x_i = x_0 + i \cdot h$. Finite differences are therefore usually implemented on an equidistant regular grid spanning time and space. In one space dimension, this equidistant grid has mesh width h. The finer the mesh width, the bigger will be the algebraic equation that has to be solved in the next step. For one dimension, the values of the solution can be stored in a vector, where the i-th entry of the vector represents the solution at position $i \cdot h$. For

two or more dimensions, the solution can still be stored in a vector, but needs to be encoded in a special form, e.g., row-major or column-major.

Name	Diff. Eq.	Formula
Forward Difference	$\partial f/\partial x$	$(f(x_{i+1}) - f(x_i))/h$
Central Difference	$\partial f/\partial x$	$(f(x_{i+1}) - f(x_{i-1}))/2h$
2nd Order Central	$\partial^2 f/\partial x^2$	$(f(x_{i+1}) - 2f(x_i) + f(x_{i-1}))/h^2$

Table 2.1: Common finite differences for first and second-order derivatives.

Table 2.1 lists common finite differences used for the discretization of terms of 1st order derivatives and 2nd-order derivatives. Notice that the continuous parameter x is used in the differential equations, whereas discrete parameters x_i are used in the differential equations as fixed points in the discretization grid.

Using finite differences, algebraic equations with unknowns u_i can be formulated. For time-dependent simulations, there are two different classes of methods for discretization, namely explicit methods and implicit methods. Explicit methods consider the future state as evolution of only the current state. This allows to define a straight forward problem resulting in a matrix A that has to be multiplied with the current state u_i in order to calculate the next state u_{i+1}. Implicit methods on the other hand consider both, future state and current state as input and calculate the next state as result of linear equations. Linear equations are formulated as matrix problem with given matrix A, given right-hand side f and unknown next state u_{i+1}. For the next state, the equation $Au_{i+1} = f$ has to hold. This is implemented in a numerical solver.

There exist numerous special methods for discretization that are used for certain simulation problems. For instance, for diffusive phenomena, the forward time central differences space (FTCS) scheme is commonly used. As the name suggests, the FTCS scheme uses forward differences for the time discretization and central differences for the space discretization. Additionally, for multiple dimensions, there are also alternative direction implicit (ADI) methods that use one-dimensional implicit space discretization for alternating directions. The goal of such methods is to reduce the complexity of the underlying algebraic problem

such that it can be solved quickly by still providing good approximations for the true solution of the differential equation.

While explicit methods can calculate the solution directly using the matrix product of the last state, implicit methods require the solution of an algebraic equation. How such equations can be solved is part of the next Section.

2.1.4 Numerical Solvers

The last section described how simulation problems are discretized and transformed into algebraic problems. This section briefly describes how the algebraic problem can be solved using numerical solvers (see Figure 2.1). Usually, the algebraic problem is given as set of linear equations, i.e., a matrix A and a right-hand-side f. The solution of the problem is an unknown vector u fulfilling $A \cdot u = f$. For simplicity, we assume that the algebraic problem is mathematically well-defined, i.e., there exists exactly one solution. Solving linear equation problems has a long history dating back to 2000 BC, and involved famous scientists such as Isaac Newton, Carl Friedrich Gauss, John von Neumann, and Alan Turing [Grc11, Tur48]. Hence, a variety of methods exist for solving linear equations for different properties of the matrix A. This section therefore only provides the basic concepts of exact and iterative solvers before briefly discussing the impact of numerical libraries.

Exact Solvers

The basic Gaussian Elimination technique taught in school is used for solving linear equations by subtracting and multiplying equations until the equations are transformed to a triangular form, i.e., the matrix A has only non-zero entries below or at the diagonal, as can be seen in the following equation, where $*$

represents any entry.

$$A = \begin{pmatrix} * & 0 & 0 \\ * & * & 0 \\ * & * & * \end{pmatrix} \tag{2.2}$$

Solving $A \cdot u = f$ for such a triangular matrix A and unknown u is straight forward and only has linear complexity. The algorithm starts at the first entry, where it assigns $u[0] \leftarrow f[0]/A[0,0]$. Having calculated all previous entries of u, the next entry i is calculated as

$$u[i] \leftarrow \frac{f[i] - A[i,0] \cdot u[0] - \cdots - A[i,i-1] \cdot u[i-1]}{A[i,i]}. \tag{2.3}$$

The linear complexity of solving triangular systems motivates one common decomposition: the LU-Factorization. The LU-Factorization decomposes the matrix A into an upper (U) and lower (L) triangular matrix with $A = L \cdot U$. Intuitively, having such a factorization allows for fast solution of $A \cdot u = f$ in a two-step algorithm, where first the triangular system $L \cdot y = f$ and then $U \cdot u = y$ are solved to obtain y and u. Solving these two systems only requires linear complexity with slight variations of the above algorithm for triangular systems. LU-Factorization is especially useful when multiple sets of linear equations need to be solved with the same matrix A but different vectors f, as the factorization only involves A but not f.

Iterative Solvers

In simulations, we are always concerned about the quality-to-complexity trade-off. Solving the exact solution might result in long calculations that might also be subjected to numerical instabilities. Another concept than solving the equation exactly is therefore to construct a sequence of approximate solutions $\{u_i\}$ converging to the real solution u. Calculating a new element in this sequence is one iteration. Iterative solvers support multiple stopping criteria, e.g., after a

fixed number of iterations or after the difference of the previous approximate solution is lower than a defined threshold. Depending on the application, a suitable stopping criteria can be chosen.

One famous iterative solver is provided by the Jacobi Method. For solving $A \cdot u = f$, the Jacobi Method splits the matrix A into two parts D and R, where D simply includes all entries on the diagonal of matrix A and R the remainder. To this end, R has only 0 entries on the diagonal and $A = D + R$. The next solution is then calculated as $u_{i+1} = D^{-1}(f - R \cdot u_i)$. This is motivated by the equation $D \cdot u = f - R \cdot u$ and as D can be easily inverted.

Many different iterative solvers exist for different properties of the matrix A. The properties depend on the conditions required to guarantee convergence of the constructed sequence. In the example of the Jacobi Method, the matrix A has to be diagonally dominant, i.e., the diagonal value is greater than the sum of absolute values of other entries in the same row.

One important property of iterative solvers is that they can use the concept of linear operators, i.e., they do not require construction of the full matrix but only to implement the matrix-vector product with the problem matrix A. This allows to save memory by not storing the problem matrix and reduces memory access. For instance, the Jacobi Method only requires multiplication with the inverse diagonal D^{-1} and multiplication with the remainder R, which can both be implemented directly without constricting the matrix A. Similarly, the often applied conjugate gradient solver only requires the multiplication of vectors with the problem matrix A [S+94].

Numerical Libraries

Having described different methods for solving discrete simulation problems, we shortly want to discuss performance of such algorithms and why numerical libraries are so important.

Performance of numerical algorithms heavily depends on the utilization of processor features. For instance, modern single instruction multiple data (SIMD) instructions process 8 double precision floating point numbers in one instruc-

tion [Coo18]. Using such features significantly speeds up the calculation, e.g., for solving triangular matrix equations. However, using specific processor features is complex and might require to use assembly instructions or compiler-specific tricks, making the code less portable and harder to write and maintain.

Basic Linear Algebra Subsystem (BLAS) and Linear Algebra PACKage (LAPACK) libraries provide generic standardized routines that can be used for efficient platform-independent implementations of numerical algorithms. There are different BLAS and LAPACK libraries available, e.g., the open source Automatically Tuned Linear Algebra Software (ATLAS) library or Intel Math Kernel Library (MKL). All those libraries provide the same routines and are optimized for specific processor architectures utilizing specific SIMD instructions and other optimizations [RJ15]. The same technique is also used when implementing numerical software for supercomputers, where such libraries are provided by the manufacturer.

2.1.5 Quality

This section briefly discusses the definition of quality, which is important for all steps in the simulation pipeline (cf. Figure 2.1).

Quality Degradation

A perfect quality simulation can never exist, since many small effects contribute to the overall behavior of the real-world system. To this end, all parts of the real system with only little impact to quality might be omitted to speed-up the computation of the result. Such decisions on the quality-to-complexity trade-off have to be made during all steps in the simulation pipeline:

- During the modelling process, the model will only capture a small number of effects and behavior of the real system. Depending on the granularity of the model, different effects can be included. However, with more details in the model, higher complexity for the computation is required.

- During discretization, depending on the mesh-width, not all effects of the mathematical model can be observed. For instance, in a flow simulation, turbulent flows require a very fine-grained discretization grid compared to laminar flows [BZBP13]. Knowing which effects are to be expected during runtime may be therefore essential to choose a suitable discretization.

- For the numerical solver, iterative solvers can reduce the quality of the solution by reducing the number of iterations and provide significantly faster results.

Since every step in the simulation pipeline affects all subsequent steps, the impact of quality decisions on the overall quality has to be evaluated carefully. Additionally, applications using numerical simulations have to define their own quality requirements. To define parts of such requirements, the mathematical concepts of error and residual can be used. These concepts are introduced in the following section.

Error and Residual

For definition of quality, two mathematical quantities are important: error and residual. Consider an algebraic equation $A \cdot u = f$, where A is a given matrix, f is the given right-hand side and u is the unknown solution vector. Given an approximate solution $\tilde{u}$ to this equation, the error e and the residual r are defined as follows:

$$e = \tilde{u} - u \tag{2.4}$$

$$r = A \cdot \tilde{u} - f = A \cdot e \tag{2.5}$$

Intuitively, the residual measures the difference by relaxation of f, while the error measures the difference compared to the true solution.

2.2 Model Order Reduction

This section describes how Model Order Reduction (MOR) can be used for fast computation of approximate results of parameterized simulation problems. Parameterized simulation problems have many applications from multi-query optimization problems to low-latency computation when some parameters can be provided only during runtime.

The concept of Model Order Reduction (MOR) reduces the computational complexity of numerical simulations by generating a computationally less complex model providing fast approximate results for different parameters. There are different methods for MOR, e.g., Proper Orthogonal Decomposition [KGVB05], Krylov Methods, or the Reduced Basis Method (RBM) [Haa16]. We will focus on the reduced basis method.

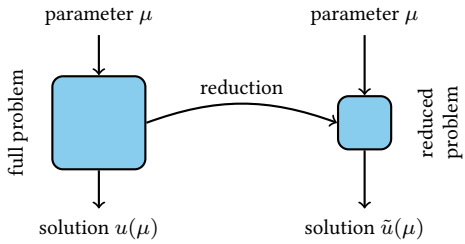

Figure 2.2: Model order reduction translates full simulation problems to reduced simulation problems, requiring significantly fewer computing resources.

The idea of RBM is to generate a low-dimensional search space for approximate solutions of the original simulation problem (cf. Figure 2.2). This is implemented using solutions of the full problem. To this end, the full problem has to be solved multiple times with different parameters during a pre-computation phase. Using these solutions, RBM constructs the reduced problem, which can then be used to provide fast approximate solutions.

Before we provide more details about RBM, the next section will first introduce the parameterized full problem and the reduced problem.

2.2.1 The Parameterized Full Numerical Problem

As described above in Section 2.1.3, the simulation problem can be translated in a set of linear equations represented as matrix equation of the form $A \cdot u = f$, where A is a given matrix, f is a given right-hand side and u is the solution.

Simulation models contain parameters describing different properties of the system that can be changed. Such parameters can be used to interact with the system, e.g., to insert sensor readings or user input. To express the dependency on parameters, we formulate the algebraic problem as

$$\mathbf{A}(\mu) \cdot \mathbf{u}(\mu) = \mathbf{f}(\mu) \tag{2.6}$$

where μ is a vector including all parameters of the simulation. In the RBM context, this problem is called the *full problem*.

2.2.2 Parameter Separable Matrices

The essential part of RBM is the parameter separability of the matrix $A(\mu)$ and the right-hand side $f(\mu)$. Parameter separability of a vector or matrix M is given if we know scalar functions θ_i that map from the parameter space to real numbers and matrices M_i that have the same shape as M such that

$$\mathbf{M}(\mu) = \sum_i \theta_i(\mu)\mathbf{M}_i. \tag{2.7}$$

Notice that we assume that the sum in this equation is finite. Such a representation can be derived from the model equation, e.g., after the discretization using finite differences, or using Empirical Interpolation Methods [BMNP04].

2.2.3 The Reduced Problem

The RBM represents approximate solutions of the full numerical problem as linear combination of *snapshots*. Snapshots are pre-computed and linearly independent solutions for typical parameters. The snapshots form the snapshot matrix V.

Therefore, the approximation to the real solution $u(\mu)$ is $Vu_V(\mu) \approx u(\mu)$, where vector $u_V(\mu)$ is called the reduced solution. The size of the reduced solution is the number of snapshots.

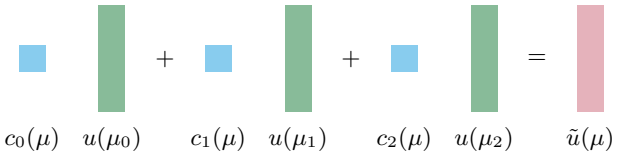

Figure 2.3: The number of unknowns (blue) is much smaller than for the full problem.

Figure 2.3 visualizes the multiplication of coefficients $c_i(\mu)$ of a reduced solution $u_V(\mu) = (c_0(\mu), c_1(\mu), \cdots)^T$ with snapshot matrix $V = (u(\mu_0); u(\mu_1); \cdots)$ and demonstrates why the reduced problem is much faster to solve. While snapshots and the approximate solution are vectors, coefficients $c_i(\mu)$ are scalars. Typically, the number of coefficients is around 10, while one snapshot can have up to one million entries. In this example, the reduced problem has 3 unknown coefficients, while the full simulation result would have up to one million unknowns, making the full problem much harder to solve.

For finding coefficients in $u_V(\mu)$, we will now formulate the reduced problem. To this end, we use $Vu_V(\mu) \approx u(\mu)$ and rewrite Equation 2.6 as $A(\mu)Vu_V(\mu) \approx f(\mu)$. This is an overdetermined system. Therefore, we multiply the full problem from left with V^T, which yields our reduced problem $V^TA(\mu)Vu_V(\mu) = V^Tf(\mu)$. We call $A_V(\mu) := V^TA(\mu)V$ the reduced matrix with snapshot matrix V. Notice that $A_V(\mu)$ is again a separable matrix. The matrices (V^TA_iV) in this separation can be pre-computed, and the matrix $A_V(\mu)$ can be rapidly assembled:

$$\begin{aligned}\mathbf{V}^T\mathbf{A}(\mu)\mathbf{V} &= \mathbf{V}^T\left(\theta_1\left(\mu\right)\mathbf{A}_1 + \cdots + \theta_m(\mu)\mathbf{A}_m\right)\mathbf{V} \\ &= \theta_1(\mu)\mathbf{V}^T\mathbf{A}_1\mathbf{V} + \cdots + \theta_m(\mu)\mathbf{V}^T\mathbf{A}_m\mathbf{V}\end{aligned}$$

Similarly, $f_V(\mu) := V^Tf(\mu)$ can be partially pre-computed and rapidly assembled.

The process of computing a reduced solution is now to solve

$$A_V(\mu) \cdot u_V(\mu) = f_V(\mu)$$

and then to reconstruct $u_V(\mu)$ to the full problem space by multiplication with V. Solving the low-dimensional problem is much faster than solving the full problem, as the low-dimensional problem has only the size of the number of snapshots, which is typically much smaller than the full problem size.

2.2.4 Error Estimation

As described in the previous section, in numerical simulations, the trade-off between accuracy and computational effort is made on many levels. One key property for serious simulation applications is to estimate or indicate the error. We briefly discuss how a fast error indicator can be implemented for reduced models.

Using a reduced model instead of solving the full problem typically degrades the quality of the solution. To express the quality of the solution, we use the residual norm of the approximation as error indicator (see Section 2.1.5). The residual r is the difference of $A(\mu)Vu_V(\mu)$ and the right-hand side $f(\mu)$. If the residual r is 0, the approximation $Vu_V(\mu)$ is the exact solution $u(\mu)$ of the algebraic problem $A(\mu)u(\mu) = f(\mu)$. To this end, the residual is a very common and well-known value for the accuracy of the simulation model.

For RBM, the residual of the approximate solution from the reduced problem can be computed very efficiently by using pre-computed parameter separable matrices. Starting with the definition of the residual r as $r = A(\mu)Vu_V(\mu) -$

$f(\mu)$, the norm of the residual $\|r\|^2 = r^T r$ can be computed as follows:

$$\|\mathbf{r}\|^2 = \mathbf{u}_V(\mu)^T \underbrace{\mathbf{V}^T\mathbf{A}(\mu)^T\mathbf{A}(\mu)\mathbf{V}}_{\mathbf{r}_1} \mathbf{u}_V(\mu) \tag{2.8a}$$

$$- \mathbf{u}_V(\mu)^T \underbrace{\mathbf{V}^T\mathbf{A}(\mu)^T\mathbf{f}(\mu)}_{\mathbf{r}_2} \tag{2.8b}$$

$$- \underbrace{\mathbf{f}(\mu)^T\mathbf{A}(\mu)\mathbf{V}}_{\mathbf{r}_3} \mathbf{u}_V(\mu) \tag{2.8c}$$

$$+ \underbrace{\mathbf{f}(\mu)^T\mathbf{f}(\mu)}_{r_4}. \tag{2.8d}$$

Notice that r_1 to r_4 can be expressed as separable matrices. These matrices can be pre-computed after the basis construction. To this end, the size of the matrices to be computed only depends on the number of snapshots, which is much lower than the size of the full problem matrix $A(\mu)$. Also notice that we do not need to calculate the full solution for this error indicator and require therefore much less overhead compared to calculating the full solution. The error indicator is therefore suitable to measure quality during the basis generation process, as it will be described in the following section.

2.2.5 The Basis Generation Process

Using the residual as error indicator, the snapshots can be computed from a training set of parameters using a *greedy* approach [VPRP03].

Figure 2.4 depicts the pseudo code of the greedy basis generation method. The user provides a set of training parameters ($train\_set$) and a maximum threshold for the residual (max_res). The initial basis can be either an existing basis, or the reduced basis based on the evaluation of a random training parameter. The algorithm will terminate when the generated basis provides a residual norm lower than the provided threshold for all parameters in the training set. In every iteration of the loop, one solution of the full problem is computed and added to the basis. For this computation, the parameter that yields the maximum residual

```
1:  function GreedyBasisGeneration(train_set, max_res)
2:      basis ← initial basis
3:      residuals[μ] ← basis.residual(μ)   ∀μ ∈ train_set
4:      while max(residuals) ≥ max_res do
5:          μ* ← max(residuals).key
6:          solution ← solution of the full problem for μ*
7:          AddSnapshot(basis, solution)                ▷ See Figure 2.5
8:          residuals[μ] ← basis.residual(μ)   ∀μ ∈ train_set
9:      end while
10:     return basis
11: end function
```

Figure 2.4: Greedy approach for generation of a reduced basis, where $train\_set$ is a set of parameters and max_res is the maximum residual threshold.

norm using the current basis is chosen.

Normalization and Orthogonalization

Using the snapshots directly as solutions of the numerical simulation might lead to poor numerical stability. For instance, consider two snapshots s_1 and s_2 as results of the full simulation problem with just one different entry in both large vectors. This difference might, even for humans, be hard to see and also might lead to numerical instabilities of the solution. To provide better numerical stability, only the most relevant information of the second vector should be added to the basis. Adding only the most relevant part of snapshots can be implemented using orthonormal bases, i.e., bases that are both, normal and orthogonal.

Orthonormal bases can be implemented using the iterative Gram-Schmidt procedure when adding a new snapshot to the basis [Fis08]. This procedure calculates what part of the snapshot can be already expressed using the current basis and what information is new. It then only adds this new information to the basis. The same concept is repeated for every new snapshot added to the basis.

Figure 2.5 depicts pseudocode of the Gram-Schmidt procedure. Adding one additional snapshot requires to calculate the product with all orthonormalized vectors in the basis in line 5. Therefore, adding one snapshot is in $\mathcal{O}(|B|)$, where

1: **function** AddSnapshotOrthonormal(basis B, snapshot s)
2: **if** B is $\emptyset$ **then**
3: **return** $\left\{ \frac{s}{\|s\|} \right\}$
4: **end if**
5: $\tilde{s} \leftarrow \sum_{b \in B} \langle s, b \rangle b$
6: $b \leftarrow s - \tilde{s}$
7: **return** $B \cup \left\{ \frac{b}{\|b\|} \right\}$
8: **end function**

Figure 2.5: Function for adding a snapshot s as solution to the simulation problem to an already orthonormal basis B.

$|B|$ is the number of snapshots in the basis.

2.2.6 Limitations of the Reduced Basis Method

The RBM converts the full-dimensional problem into a lower-dimensional problem by computation on snapshots. The dimension of the reduced problem depends on the number of snapshots. The number of snapshots depends on the characteristics of the problem and the quality required by the user. The here introduced method works only for stationary problems without any changes to the geometry. However, in recent literature, RBM approaches for time-dependent problems or problems with changing geometry have already been proposed [Haa16, RHP08]. Nonetheless, advanced RBM approaches are beyond the scope of this thesis.

2.3 Data Assimilation

Data assimilation is the process of combining uncertain sensor data with uncertain simulation models and uncertain initial state [LSZ15b]. Uncertainties are typically expressed in form of covariances describing noise in the respective quantities. While there are different methods for data assimilation available, we focus on the Kalman filter (KF) and the ensemble Kalman filter (EnKF). KF and EnKF gained lots of attention since they have been used for navigation of

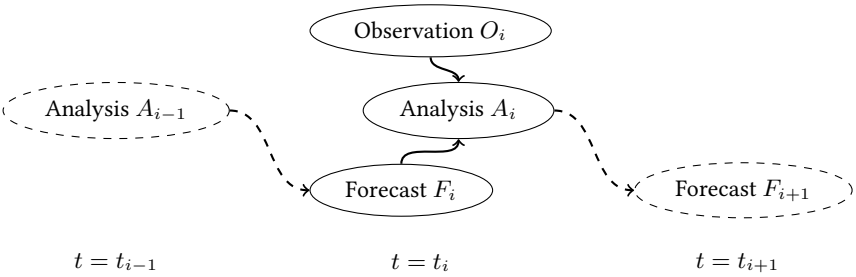

Figure 2.6: Formal model for data assimilation.

spacecrafts in the Apollo program in the 1960s [GA10]. Since then, it has been used in many applications, e.g., for attitude determination [HMS03]. This section will first describe the KF, by introducing forecast, analysis, observations, analysis step, Kalman gain, forecast covariance, and analysis covariance. After a brief summary of the concepts for the KF, the EnKF will be explained.

2.3.1 Formal Model of Forecast, Analysis, and Observations

Formally, data assimilation is an iterative process combining information on forecast state and sensor observations to a common state, referred as analysis state (cf. Figure 2.6). Forecasts F_i are the result of the simulation model on the previous analysis A_{i-1}. Observations O_i contain new information about the state of the real system, e.g., using sensors. Goal of data assimilation is to combine observations O_i and forecast state F_i to a common analysis state A_i. Initially, the initial state of the simulation is used as forecast state F_0.

All quantities F_i, O_i, and A_i contain uncertainties that can be expressed by covariance matrices. There are three quantities that contribute to uncertainties: Initial uncertainty, process uncertainty of the simulation model, and measurement uncertainty of observations. Initial uncertainty is the uncertainty associated to the initial state F_0. This uncertainty might traverse all future analysis and forecast states during the execution unless observations are included. Process uncertainty is the uncertainty added by the simulation model. Even if the previous analysis

A_{i-1} is perfectly accurate, the simulation model will introduce some error during computation of the forecast state F_i. Measurement uncertainty are assigned to observations O_i. This way, measurement errors and noise of respective sensors can be included into the model.

The forecast F_i can be calculated from the previous analysis state A_{i-1} by applying the simulation model on A_{i-1}. For the explanation of the Kalman filter, we assume that the simulation model is described in a matrix S_i such that $F_i = S_i \cdot A_{i-1}$. The following section explains the calculation of the next analysis state A_i using the introduced quantities.

2.3.2 Analysis Step and Kalman Gain

The previous section introduced the forecast F_i and the observations O_i. In this section, we explain how forecast and observations are combined to form the combined analysis state A_i. To this end, we will require the Kalman gain K_i, and covariance matrices expressing the errors in observation, forecast, and analysis.

To calculate analysis A_i, the current forecast F_i and observation O_i are required. The KF and EnKF combine these quantities using a matrix expressing weights of how much the observation and how much the forecast can be trusted [LSZ15a]. This matrix is called the Kalman gain K_i. Using the Kalman gain, the general formula for the analysis is

$$A_i = F_i + K_i(O_i - H_i F_i) \tag{2.9}$$

where H_i is a selection matrix mapping only values from the forecast that relate to positions of the sensor observations in O_i. Intuitively, H_i maps from the state space of the forecast model to the observation space of sensors.

The Kalman gain rates the importance of observations over the forecast. If the Kalman gain is high, the observations have more weight, as the measurements have been identified to be accurate and the forecast model provided only unstable results. If the Kalman gain is low, the forecast model has more weight, as the measurements have been identified to be inaccurate and estimates have only a

small error. To this end, the formula in one dimension of the Kalman gain K is given as follows:

$$K = \frac{\text{forecast error}}{\text{forecast error} + \text{observation error}}. \tag{2.10}$$

In multiple dimensions, the Kalman gain is calculated from the covariance matrices for the forecast C_i^F and for the observations C^O. The general formula can be directly derived from Eq. 2.10:

$$K_i = C_i^F H_i (H_i C_i^F H_i^T + C^O)^{-1}. \tag{2.11}$$

In contrast to Eq. 2.10, the forecast covariance is transformed to the smaller observation space using H_i. This is required to combine the covariance matrices of the forecast C_i^F and the observation C^O. Additionally, it reduces the complexity for the matrix inversion, which in the 1d case is a simple division (cf. Eq. 2.10).

2.3.3 Observation Error, Forecast Error, and Analysis Error

For the calculation of the Kalman gain K_i, observation covariance C^O, the forecast covariance C_i^F, and the analysis covariance C_i^A are required [LSZ15a]. The observation covariance C^O is assumed to be known from the technical properties of the sensor providing the observations. Such information are typically provided by the manufacturer. The forecast covariance C_i^F can be calculated directly from the covariance of the previous analysis state C_{i-1}^A and the error of the simulation model C^S. It is calculated as follows:

$$C_i^F = S \cdot C_{i-1}^A S^T + C^S. \tag{2.12}$$

The only missing matrix is now the covariance of the analysis steps A_i. This matrix can be calculated using the current forecast covariance C_i^F, the selection matrix H_i and the Kalman gain K_i. It is calculated as follows:

$$C_i^A = (I - K_i H_i) C_i^F, \tag{2.13}$$

where I denotes the one matrix in the required dimension. Intuitively, this formula uses the values where observations exist and reduce the forecast covariance depending on the Kalman gain K_i. As mentioned before, entries in the Kalman gain are between 0 and 1 and reflect the weight of how observations should be taken into account. If the observations are accurate, the entry in K_i will be 1. In this case, this entry after the analysis has to be perfectly accurate and will be 0. If the observation is useless and the entry in K_i is 0, the entry from the covariance of the forecast step C_i^F will be taken, as the error could not be reduced in the analysis.

2.3.4 Summary of the Kalman Filter

Having introduced all required concepts for the Kalman filter (KF), we now provide a short summary of the required calculation for one step of the Kalman filter.

1: **function** KalmanStep(A_{i-1}, C^A_{i-1}, O_i, H_i)
2: $F_i \leftarrow S \cdot A_{i-1}$ ▷ Forecast
3: $C_i^F \leftarrow S \cdot C^A_{i-1} S^T + C^S$ ▷ Covariance of Forecast
4: $K_i \leftarrow C_i^F H_i (H_i C_i^F H_i^T + C^O)^{-1}$ ▷ Kalman Gain
5: $A_i \leftarrow F_i + K_i(O_i - H_i F_i)$ ▷ Analysis
6: $C_i^A \leftarrow (I - K_i H_i) C_i^F$ ▷ Covariance of Analysis
7: **return** (A_i, C_i^A)
8: **end function**

Figure 2.7: One step of the Kalman filter. The input is the last analysis A_{i-1}, the covariance of the last analysis C^A_{i-1}, and the current observations O_i and position of the observations decoded in matrix H_i. The output is a tuple of the current analysis state A_i and the covariance of this analysis state C_i^A, that can then be used in the next step as parameters.

Figure 2.7 depicts the algorithm for the Kalman filter. The filter is an iterative process where the output consisting of the analysis state A_i and analysis covariance C_i^A will be used as input for the next state (cf. Figure 2.6). The initial state A_0 has to be provided by the application with an initial covariance C_0^A reflecting

the uncertainty of A_0.

Vector O_i represents sensor readings at one time-step and matrix H_i decodes the mapping from state space to observation space, i.e., it maps corresponding values in A_i (and F_i) to values in O_i. When there are no new observations for one time-step, the Kalman gain has to be zero and the analysis is directly taken from the forecast. In this case, A_i will be set to F_i and C_i^A to C_i^F.

The space complexity of the Kalman filter depends on the number of discretization points of the simulation. This number is also the size of vectors A_i and F_i. With linearly growing number of discretization points of the simulation, vectors A_i and F_i will also grow linearly. However, space complexity of C_i^A will grow quadratically, which raises problems on machines with only few memory, like mobile devices. For such devices, we therefore found another method for data assimilation better suited: the ensemble Kalman filter. This data assimilation method will be discussed in the following section.

2.3.5 The Ensemble Kalman Filter

The Kalman filter (KF) as described in the previous section has two problems. First, it has to store and track the full covariance matrix C_i^A, which requires quadratic memory compared to one forecast simulation state. Especially for complex simulation models with hundreds or thousands of unknowns, this is the major bottleneck. Second, it is only applicable for forecast models that are can be expressed in matrix multiplication, i.e., only explicit methods and not implicit methods (cf. Section 2.1.3).

This section introduces the ensemble Kalman filter (EnKF), which has a lower memory overhead and can be used for explicit and implicit methods [HM98, Eve03]. The biggest difference to the KF is that the EnKF replaces analysis covariance matrices C_i^A with sample covariances. These sample covariances are generated from a set of randomized replicas of the original state, the so called ensemble members (cf. Figure 2.8). For every time-step, all ensemble members follow the behavior of the simulation model. Intuitively, the distribution of ensemble members in one step represents the uncertainty of the current state.

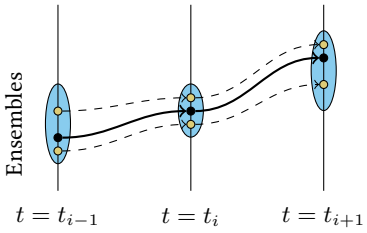

Figure 2.8: Ensemble members (yellow) tracking the real state (black) of the system over time.

It has been shown that even complex applications only require few ensemble members [HM98]. Therefore, processing and storing ensemble members is much faster than using the full covariance matrix as in the KF.

In the following, we briefly describe how ensemble members are generated and how the Kalman gain can be calculated from the sample covariance.

Generation of Ensemble Members

One of the challenging parts for the EnKF is the generation of ensemble members [Eve04]. The idea of the EnKF is that the mean of ensemble members should track the real state of the system (cf. Figure 2.8). Therefore, simply adding white noise to the initial state does not provide desired results, since this would only track the real system state at the current time-step and not necessarily the next time-step. For instance, consider a heat simulation with positive values and zero on the boundary. If we would simply add white noise to an initial state, bigger positive values would "cool down" faster than negative values "heat up" from surroundings. Thus, the mean value of ensembles would be negative.

Generation of ensemble members is problem dependent, as it can be seen in the previous example. In this example, we could only use uniform distributed positive values for perturbation. Following the system behavior, positive changes of the temperature will be cooled down in the next step and therefore allow the mean value of all temperatures to track the real system state. For other applica-

tions, generation of ensemble members has to be implemented with meaningful perturbation that can be provided by an expert in the corresponding field.

Calculation of the Sample Covariance

The sample covariance is calculated directly from the ensemble members using the mean value and the ensemble perturbation matrix [BJvLE98, Eve03]. For the formal description, we first need to introduce notation and definitions of ensemble members, the ensemble matrix, the ensemble perturbation matrix and the mean matrix.

The ensemble matrix E_i is the matrix that holds all m ensemble members of time-step i. Formally, the m ensemble members are denoted as $e_i^{(j)}$ and the ensemble matrix is defined as $E_i = (e_i^{(1)}; \ldots; e_i^{(m)})$.

For the calculation of the sample covariance, we require two other matrices, the ensemble perturbation matrix E_i' and matrix $\overline{E_i}$ which stores the mean of the ensembles in each column. The latter matrix is defined as $\overline{E_i} = E_i 1_m$, where the matrix 1_m is a $m \times m$ matrix with all values set to $1/m$. Using this matrix, the ensembles perturbation matrix can be defined as $E_i' = E_i - \overline{E_i}$.

Having defined these matrices, we can now define the sample covariance matrix as

$$C_i^E = \frac{1}{m-1} E_i' (E_i')^T. \tag{2.14}$$

Where the scalar division by $m - 1$ is used as Besel's correction to provide unbiased results for different ensemble sizes. Once the sample covariance C_i^E has been calculated, the Kalman gain is calculated analogously to Eq. 2.11 as

$$K_i = C_i^e H_i^T (H_i C_i^e H_i^T + C_i^O)^{-1}. \tag{2.15}$$

This Kalman gain can then be used to update the current state as described in Eq. 2.9.

To summarize this section, we explained two popular data assimilation tech-

niques, namely, the Kalman filter (KF) and the ensemble Kalman filter (EnKF). While the KF uses full covariance matrices, the EnKF uses only a sample covariance, which is only an estimate of the full covariance matrix. However, by using only the sample covariance, the EnKF has a much lower complexity making it the preferred data assimilation technique for mobile devices.

2.4 Mobile Computing

In the following, background information on wireless communication, network availability models, and throughput measurement for wireless communication links is provided.

2.4.1 Wireless Communication Technologies

Mobile devices do not have any wired connections and therefore need to use wireless communication. There are different standards for wireless communication depending on the user case. For cellular networks providing world-wide ubiquitous access, Long Term Evolution (LTE) is the major standard. For wireless local area networks (WLAN), the IEEE 802.11 family is the dominant standard, with IEEE 802.11 ac being the most recent standard for WLAN. In the following, few details about the physical layer is provided, before more details are given on cellular networks and WLAN.

The Wireless Physical Layer

The wireless physical layer consists of radio waves that travel through space. Physically, waves can be described by frequency, amplitude, and phase. Depending on the frequency, radio waves propagate differently. As rule of thumb, waves with shorter frequencies have higher range and travel better through solid materials like walls [Wal02]. However, radio waves can be reflected by obstacles leading to multiple paths of the same wave. Such effects can lead to fading of the signal at the receiver.

To be used as digital medium, analog radio waves need to be translated into bits. There are multiple methods for such modulation techniques [Wal02]. One often used method is quadrature amplitude modulation (QAM), where frequency and phase of the radio wave is translated into (multiple) bits.

To support multiple channels, signals need to be multiplexed. One of the mostly used multiplexing technique is orthogonal frequency-division multiplexing (OFDM), where orthogonal frequencies are constructed such that, for multiple frequencies at a specific time, the receiver sees only the signal from the sender [Wal02]. Protocols such as LTE or IEEE 802.11 ac use OFDM and QAM.

Cellular Networks and Long Term Evolution

Cellular networks are wide area networks for ubiquitous access everywhere at any time. The network is built using fixed-location transceivers forming cell areas. Transceivers are connected via a high-speed backend network to serve mobile devices in their cells.

Long Term Evolution (LTE) is today's mostly used standard for cellular networks by the 3rd Generation Partnership Project (3GPP). It promises high data rates of more than 300 Mbit/s for the downlink in perfect conditions and a target latency of less than 5 ms between mobile device and base station [LLM+09, PYLFT16].

In the following, some fundamentals are provided about cellular networks and LTE. Further details can be found in [Wal02, PYLFT16].

Location of the Mobile Device When a packet has to be sent to a mobile device, the approximate location of the device has to be known to forward the packet to the responsible transceiver [Wal02, p. 229]. If the location was not known, the network would need to search for the responsible transceiver, which requires traffic in all cells nation-wide. Nation-wide traffic has to be avoided, as it does not scale with the number of mobile devices. Therefore, the provider stores the approximate location as set of transceivers where the mobile device is located. The responsible cell for the mobile device can then be found using

paging only in cells of these transceivers.

Scheduling of Downlink Packets to Mobile Devices Scheduling of downlink packets in one cell is a trade-off between spectral efficiency and fairness [CPG+13]. The shared downlink channel is divided into resource blocks (RB), spanning time duration for one frequency domain. Coding in one RB depends on the Signal to Inference plus Noise Ratio (SINR) of the mobile device. For instance, if the reception of the device is good, i.e., it has high SINR, a higher bitrate will be assigned. This results in better overall spectral efficiency, if downlink packets are only sent to devices with high SINR. However, if spectral efficiency would be optimized, devices on the edge or in other conditions leading to low SINR will not receive any packet and thus would be treated unfair. The trade-off how to schedule packets depending on the bitrate and spectral efficiency is a parameter set by the provider of the cellular network.

Retransmissions Being the basic protocol for most applications on today's Internet, the transmission control protocol (TCP) is known to perform poorly in wireless networks due to packet loss in the physical layer [Ela02]. To provide better performance for TCP applications, LTE therefore implements re-transmissions on the media access control (MAC) layer using a so called hybrid automatic repeat request (HARQ) mechanism [LLM+09]. As TCP also implements retransmissions, there are two layers implementing retransmissions when using TCP over LTE.

Wireless Local Area Networks

Wireless local area networks (WLAN) are used for wireless access to the local network of home networks or companies. WLAN is used synonymously with the IEEE 802.11 protocol family. Over recent years, many IEEE standards have been implemented, leading to faster communication or extended range [Soc16].

As of today, most deployed WLANs use IEEE 802.11n or the succeeding IEEE 802.11ac. Both are based on OFDM, and multiple antennas (MIMO) to provide high data rates of up to 600 Mbit/s for n or even up to 3.4 Gbit/s for ac. However,

MIMO is mostly used to provide streams for multiple mobile devices and not for connecting two stations.

2.4.2 Energy Consumption

Energy consumption on mobile devices is critical as it affects battery runtime and therefore how long the users can interact with the device. While scientists and engineers are working on higher energy density batteries, battery power will always be limited and cannot keep up with energy demands of modern processors [AT08]. Therefore, mobile devices require energy-efficient processing and communication technologies as well as understanding of the implications of algorithms on power consumption [DLMT16]. To this end, power consumption has been observed and modelled on different levels of the mobile device, e.g., on hardware component level [BBV09, CH10], on system call level of the operating system [PHZ+11], on application code level [FS99, HLHG13], and even on browser level [TAN+12]. In the following, we will go into details about the power consumption of components on a mobile device.

Power Saving for Processors

Modern processors implement power saving techniques in form of power states [Coo18]. There are two different power states provided by the Advanced Configuration and Power Interface (ACPI), namely P-States and C-States. While P-States define the power state during active execution, C-States define the power state when the processor is halted. These power states save power by lowering frequency, lowering voltage, or switching off sub-systems in the processor.

Controlled by the operating system, the decision to go into higher power states is a trade-off between performance and energy. For instance, transition between the highest C-State (C4) to the active state (C0) takes two orders of magnitude longer than the transition from C1 to C0 [Coo18]. Notice that, during this state transition, power is consumed by the processor. Toggling between power states too frequently therefore may waste energy. To this end, the operating system

requires heuristics on the future workload to decide on power states.

As the application programmer might know best whether high workload is to be expected, the operating system provides mechanisms to change power management decisions to userspace. For instance, Android provides wake locks that restrict the system to go into higher power states in order to operate at maximum performance. This enables applications to provide fast results for heavy computations and even may result in lower power consumption [VJLA12, LXCT16].

Power Saving for Wireless Communication

In contrast to processors, power saving for wireless communication needs to be coordinated between multiple stations. For instance, IEEE 802.11 implements a power saving poll mode, where the mobile device instructs the access point to send periodic frames informing the mobile device of incoming traffic. The mobile device can then send the communication module into sleep mode and only wake it up when a new frame from the access point is scheduled.

Deciding on the frequency how often the communication module wakes up is a trade-off between latency and energy consumption. The longer the communication module is at sleep, the more energy is saved. However, if the communication module sleeps longer, the mobile device will not be informed that new packets are waiting. Therefore, different frequencies will be chosen depending on the expected load on the wireless link.

Sending data from the mobile device can also be scheduled to save power. For instance, in cellular networks, energy can be reduced by sending data in bulk [BDR13, BBV09]. This way, the communication module can switch to lower sleep modes when no data has to be sent.

3

System Overview

This chapter provides an overview of the system for the execution of numerical simulations on networked mobile devices. First, the system components will be introduced, followed by a generic system model. This generic system model will be refined for the respective concepts in the following chapters.

3.1 System Components

This section introduces the components of the system that is used by concepts and methods in the following. While all approaches share the same hardware model, the model of the numerical simulation is different for different concepts and methods.

3.1.1 Distributed Mobile Environment

The distributed mobile environment consists of computing resources, i.e., the mobile device and the server, and the wireless communication link connecting computing resources (cf. Figure 3.1).

Computing Resources

The system consists of two heterogeneous computing resources of two classes. The first class represents mobile devices, while the second class represents stationary servers.

Mobile devices are very resource constraint. To be mobile, they have to rely on mobile energy resources, i.e., they are battery-powered. To serve user requests

Figure 3.1: Mobile device and server connected via wireless communication.

as long as possible, processors on the mobile device have to be energy efficient, and are therefore significantly slower than stationary computing resources.

Servers are stationary computing resources. As servers do not have to rely on battery power, they have strong processors at much higher clock rates and are equipped with special vector processing units that can use the same instruction on multiple data. Both enable the server to provide results for computational resources much faster than devices of the mobile class.

Wireless Communication

Mobile device and server communicate via wireless communication. Wireless communication is subjected to dynamic data rates, latency, and might face even disconnections. In general, we assume a 3/4G cellular network or IEEE 802.11 WiFi with data rates between a few kbit/s to multiple Mbit/s.

3.1.2 Numerical Simulation

The numerical simulation is the application that is executed on top of the distributed mobile environment. Throughout this thesis, two models of the numerical simulation are considered: parameterized stationary simulations and time-dependent simulations. As time-dependent simulations can be seen as a sequence of stationary problems, the latter will be explained first.

Parameterized Stationary Simulations

Parameterized stationary simulations can be transferred to the solution of the algebraic problem $A(\mu) \cdot x(\mu) = f(\mu)$, where μ is a vector containing multiple

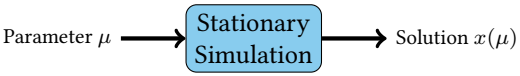

Figure 3.2: Model of parameterized stationary simulations.

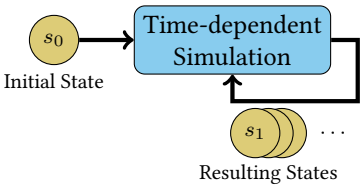

Figure 3.3: Model of time-dependent simulation providing simulation results s_i at discrete time t_i with initial state s_0.

parameters, $A \in \mathbb{R}^{n \times n}$ is a given parameter dependent matrix, $f \in \mathbb{R}^n$ is a given vector, and $x(\mu) \in \mathbb{R}^n$ is the parameter dependent unknown solution (cf. Figure. 3.2). The size n of matrices and vectors is called the problem size and can be chosen by the user.

All parameters μ_i of the stationary problem can be expressed as one *parameter vector* $\mu = (\mu_1, \mu_2, \dots)$. Parameters are called *bounded*, if the user provides μ_i^{min} and μ_i^{max} prior to runtime such that $\mu_i^{min} \leq \mu_i \leq \mu_i^{max}$ for all parameters μ_i during runtime. Notice that this might not be the case for all simulation parameters.

The user accepts some degradation of quality to speed up the computation, as long as a threshold for quality can be defined. This way, the user can decide for herself, if she wants to have a fast low quality solution or a slightly slower high quality solution. To provide a well-known interface to the user, quality of the simulation needs to be defined in form of residual or absolute error of the approximate solution to the problem (cf. Section 2.1.5).

Time-Dependent Simulation

In contrast to stationary simulations, time-dependent simulation require the solution of a sequence of algebraic problems as result of the time-discretization

process. Time-discretization divides the continuous time into a finite number of time states. Each state represents the system state at a fixed time and is required to compute the next time state (cf. Figure 3.3).

As for the stationary case, time-dependent simulations might also require external parameters that can only be assigned shortly before or directly at runtime, i.e., input from external sensors. Therefore, the solution can not just be cached but needs to be calculated every time the result of the simulation is requested.

Quality-degradations will be accepted by the user as long as the error can be quantified in a meaningful way. As the simulation requires multi-dimensional discretization, assessment of quality is more complex as for the stationary case and requires different concepts.

3.2 Generic System Model

After having introduced the components of the system, we provide a generic system model. As concepts presented in this thesis have a different focus, i.e., provide robustness against disconnections, provide fast execution using surrogate models, and provide fast execution using model order reduction techniques, the generic system model will be refined in the respective chapters.

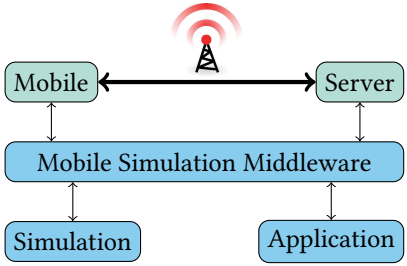

Figure 3.4: Generic system model

In our generic system model, we assume a distributed mobile simulation middleware (cf. Figure 3.4). This middleware runs on top of both compute nodes, the mobile device and the server. Furthermore, we assume that the simulation

can be executed on both nodes, which might require specialized implementations to utilize specific hardware features on the compute nodes. Additionally, the application requiring simulation results and specifying requests to the simulation, is usually only implemented on the mobile device to support the user with the specific task.

Concepts and methods consider different kinds of simulation problems, i.e., either stationary simulation problems or time-dependent simulation problems. While stationary simulation problems only provide one solution, time-dependent simulation problems provide a number of solutions. While the computation of the result can be executed on the server or the mobile device, the result has to be made available to the user on the mobile device. Additionally, methods supporting degradation of quality to speedup the computation have to include a quality constraint that is defined based on a well-known and intuitive quality metric.

The wireless link connecting mobile device and server has properties such as latency, bandwidth, and availability. Depending on the concept, these properties are either measurable or can be detected by the mobile simulation middleware. The middleware can also request profiling of the simulation on specific compute nodes to detect bottlenecks of the computation.

Further details on the system model and requirements can be found in the respective chapters introducing the concepts.

4

Increasing Robustness Against Disconnections

Wireless communication faces frequent disconnections. Disconnections are caused by multiple effects. One effect is that the mobile device got out of range of any nearby wireless access point. Other effects are inferences in the wireless channel, environmental obstacles, too fast mobility for handover between access points, problems in synchronization of too many wireless subscribers, or issues in the fixed back-end network of the provider.

As introduced before, we assume two compute nodes, where the next step of a time-dependent simulation can be executed, the mobile device or the server (see Section 3.2). Computation on the server requires to stream the result to the mobile device, while computation on the mobile device does not require a wireless link to the server. Therefore, if the network is disconnected, a mobile simulation middleware could decide to continue the computation on the mobile device. However, this might result in higher latency for the result or in increased energy consumption, as computation on the mobile device is much slower and requires mobile energy resources.

Figure 4.1 depicts a scenario for the execution on two compute nodes in case of disconnections. For the computation of state S_3, the middleware has two options. Either, it will provide the time step from the server to the mobile device, where it has to wait until the device is reconnected, or it will compute the result of time step S_3 on the mobile device with potential high energy cost and higher latency cost.

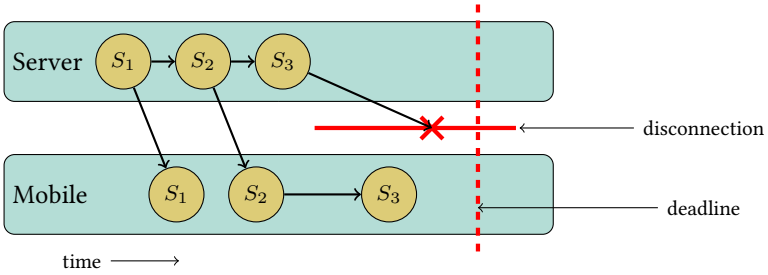

Figure 4.1: Robustness against disconnections is required to provide timely results.

In this chapter, we are considering time-based simulations with a soft deadline until all results for multiple time steps are available on the mobile device. Quality of experience depends on the soft deadline and can be set by the application programmer. For computation of simulation results, both, mobile device and server, are considered. Additionally, we consider disconnections of the wireless communication link between mobile device and server.

We assume that computation of the complex numerical simulation on the mobile device costs more energy than communication of results. If computation would cost less energy, we would simply execute the simulation on the mobile device and would not need to take the network or disconnections into account. Goal is to minimize the energy consumption in case of disconnections with constraints on latency until all results are available on the mobile device.

Parts of this chapter have been published in [DDR15].

4.1 System Model

For robust execution, we consider the system model of the distributed mobile environment with a time-dependent simulation as application (see Section 3.2).

The solver for the time-dependent simulation problem is implemented for two compute nodes, server and mobile device. These two nodes are heterogeneous.

On the one hand, the server is located inside a data center. It is well connected to other computing resources nearby and therefore can scale to current workload on demand. On the other hand, the mobile device runs on battery and is therefore constrained in energy. It has to rely on energy efficient processors being much slower than server processors. Therefore, we assume the server processor to be much faster than the mobile processor.

Mobile device and server are connected over a wireless link provided via cellular networks (3/4G). Such a link experiences communication errors on the wireless link, which will result in packet loss. There are two kinds of communication errors in wireless networks, single packet errors and burst packet errors [GF04]. While the former affects single packets only, the latter affects multiple packets over a period. Single packet errors are easily avoided by introducing retransmission on the link layer as it is implemented in LTE [LLM+09]. Therefore, the majority of errors are burst errors (cf. Section 4.8). Notice that loosing all packets over a period causes temporary disconnection.

To support different quality-of-service requirements, the user provides a soft deadline for the computation. The soft deadline is specified as time $t_{\max}$, where the computation should be finished. However, results of the simulation are required even after the deadline. Therefore, providing the solution before $t_{\max}$ is desired but not a hard requirement.

Computation on the mobile device and communication between mobile device and server effects energy consumption and therefore the limited energy stored in the battery of the mobile device. In order to give the user best experience, energy consumption of the application should be minimized.

Every simulation has a different energy profile for computing the result on the mobile device and for communicating results from a remote server to the mobile device. In this chapter, we assume that the simulation costs much more energy to compute on the mobile device than to communicate the results from a remote server. Notice that when this assumption would be wrong, we could simply execute all simulation steps on the mobile device and would not need to take unreliable wireless communication into account. However, preliminary

tests showed that numerical computations take very long on today's mobile devices while new communication techniques such as LTE promise very high data rates with low energy consumption [CPG+13]. Additionally, especially when high-quality results of the simulation are required, the computational overhead grows faster than the communication overhead, e.g., consider turbulent flows that require computation of multiple intermediate steps that are not required for reasoning or visualization of the results on the mobile device. Therefore, we assume that communicating results from the server is the most efficient solution as long as the mobile device is connected.

4.2 Problem Statement

This section introduces a detailed problem statement for minimizing energy consumption for iterative computation of the states in $S = \{s_1, \ldots, s_n\}$ to be finished within time period e with timespan $t_{\max}$. Each state $s_i \in S$ can be computed on the mobile device, the server, or both. In case of server computation, the result has to be transferred over the wireless link to the mobile device. The decision, whether a state should be computed on the mobile device or the state should be transferred over the network is given by a schedule (M, T), consisting of two sets: M represents computation on the mobile device, and T states to be transferred over the network.

Computation on the mobile device is represented in the set M. Every element $(s_i, t_i) \in M$ represents the computation of state s_i on the mobile device starting at time t_i. For this computation, the state s_{i-1} has to be available on the mobile device at time t_i. We assume all computations on the mobile device to take equal time t_M until the computation is finished and the result is available. The set $\hat{M}(t)$ represents the results of computations on the mobile device at time t and is defined as $\hat{M}(t) = \{(s_i, t_i) \in M : t_i + t_M \leq t\}$. Notice, that we consider sequential execution only. Therefore, no two states can be computed in parallel.

For computation on the server, we have to decide on the time when the result should be sent to the mobile device. This decision is represented in T within

the schedule (M, T). Every element $(s_i, t_i) \in T$ represents the transfer of state s_i from the server to the mobile device, starting at time t_i. The time when the message will be received on the mobile device can be either finite or, in cases of disconnections, infinite. The unknown function $d : \mathbb{R}^+ \to \mathbb{R}^+ \cup \{\infty\}, t \mapsto d(t)$ represents the time of delivery of a message sent at time t. At any point in time t, at most one message can be sent. If a message is sent at time t and $d(t) < \infty$, the message is delivered on the mobile device at time $d(t)$. If $d(t)$ is infinite, the message is lost. We define $\hat{T}(t) = \{(s_i, t_i) \in T : d(t_i) \leq t\}$ as the set of all messages successfully delivered on the mobile device at time t_i.

In the case of disconnections, we want to reduce the energy overhead on the mobile device. As stated in the system model, we assume that remote execution of the simulation is most efficient in terms of energy and latency for the mobile device. However, in the case of disconnections, local computation on the mobile device might be required to meet the deadline for the simulation result. In this case, we want to minimize the additional energy consumption caused by disconnections on the mobile device, i.e., the number of simulation steps that need to be computed on the mobile device.

Using the introduced notation, we can formulate our optimization problem as

$$\begin{aligned} \min \quad & |M| \\ s.t. \quad & \hat{M}(t_{\max}) \cup \hat{T}(t_{\max}) = S \end{aligned}$$

That is, we want to minimize the number of additional computations on the mobile device, under the constraint that all states should be available on the mobile device at time $t_{\max}$.

4.3 Architecture

In this section we present our architecture to solve PDEs on the two computation nodes, the mobile device and the server. This architecture is partly based on [DK14].

Our system consists of four components (1) a *scheduler* to distribute the computation among the computation nodes, (2) a *statistics component* to collect data about the availability of the wireless link on the mobile device, (3) a *disconnection detector*, and (4) a *predictor* to predict the length of a temporary disconnection based on the statistics (cf. Figure 4.2) All components run on the mobile device.

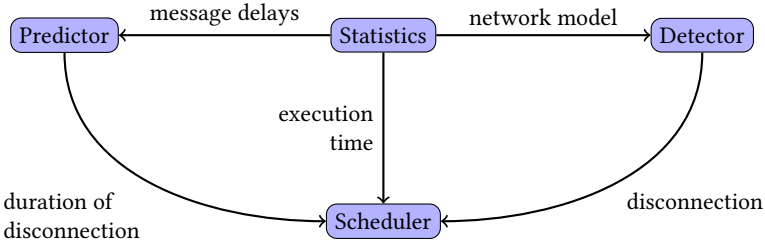

Figure 4.2: Overview of the architecture

The scheduler decides on computation and communication on and between the compute nodes. The goal of the scheduler is to minimize energy consumption on the mobile device, while fulfilling real-time constraints. As we assume computations to be very complex and much more energy intensive than communication of results, the scheduler computes all states on the server and sends them to the mobile device as long as the wireless link is available. However, when the mobile device is temporarily disconnected, the scheduler has two options (1) waiting for the link to become available or (2) starting the computation on the mobile device. The scheduler will use the predicted duration of disconnection, provided by the predictor and information about the execution time on the mobile device provided by the statistics component to make this decision.

The statistics component collects information about the execution time for the computation of states on the mobile device, information about the transfer time of packets from the server to the mobile device, and information about the availability of the wireless link. Information about execution time will be needed by the scheduler. This information will be collected every time the mobile device computes a state. The information about packet transfer time and availability of

the link will be collected using probe messages periodically sent from the server to the mobile device. If such a message is received, the statistics component informs the scheduler about the availability of the link.

The disconnection detector monitors the state of the wireless link. It uses the same probe messages sent periodically by the server as the statistics component. If no such message is received on the mobile device for a longer period of time, the detector will inform the scheduler about the mobile device being disconnected from the server. To detect disconnections, the detector uses a timeout mechanism. To choose this timeout, the detector needs further informations about the link characteristics from the statistics component.

The predictor gives a prediction about the duration of temporary disconnections of the mobile device. To this end, it uses recent data about the link availability provided by the statistics component to learn a Markov Chain. Using this Markov Chain, the expected duration of a temporary disconnection can be computed. The predictor is invoked by the scheduler once disconnection is detected.

The following sections will provide detailed descriptions about each component of the architecture.

4.4 Scheduling Computation Steps

The scheduler decides on when and where to execute compute steps and if results of computations should be sent from the server to the mobile device, i.e., construct the sets M and T for a schedule (M, T).

Notice, it is never beneficial to send states from the mobile device to the server, as (1) we assume server computation to be much faster, (2) computation on the server to not induce any cost, as it does not cost energy on the mobile device and (3) it would increase the energy consumption for communication on the mobile device. Sending a state from the mobile device to the server would therefore only cost time. However, we want to optimize energy consumption of the mobile device, which is to minimize the number of additional computations on the mobile

device. Therefore, the scheduler tries to compute as many steps as possible on the server and will start computation on the mobile device only if it might be absolutely necessary, e.g., if the risk of missing the deadline is high.

The scheduler operates in three different modes, “connected”, “disconnected”, and “recovery”. On the start of the computation, the scheduler operates in “connected” mode. The “disconnected” mode is triggered by the disconnection detector. The statistics component triggers the “recovery”, once the scheduler was in “disconnected” state and the link became available. The state transition from “recovery” to “connected” is handled by the scheduler itself (cf. Figure 4.3).

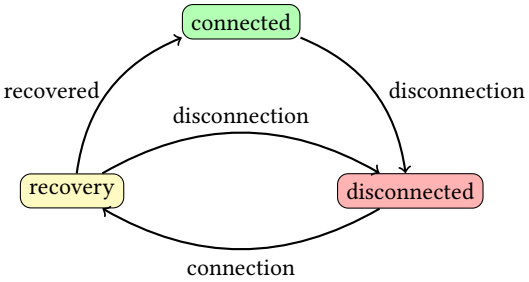

Figure 4.3: Modes and mode transition of the Scheduler

In “connected” mode, all computation is executed on the server and results are sent to the mobile device. For garbage collection on the server, the mobile device sends cumulative acknowledgments to the server. If the server receives such a message for state s_i, it will forget about all prior states s_j with $j < i$. Preliminary tests, which will be presented in the next section, show that this mode is the most efficient in terms of computations on the mobile device, since server computation is very fast compared to computation on the mobile device.

If a disconnection between mobile device and server is detected, the disconnection detector triggers the “disconnected” mode of the scheduler. The scheduler has two options during disconnections (1) to wait for the link to become available again or (2) to compute on the mobile device. The two alternatives have different effect on the objective function. Waiting does not introduce any additional cost,

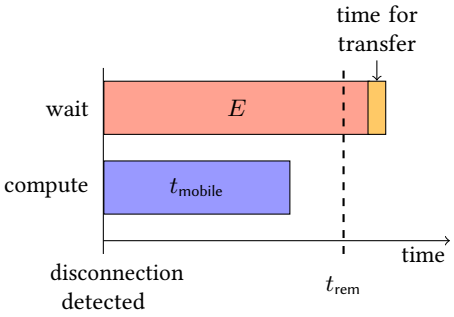

Figure 4.4: Scheduling decision based on the expected time of the disconnection.

while computation increases the additional energy consumption, and, therefore, the objective function of the optimization problem. However, if the scheduler waits too long, the deadline for the computation cannot be kept and the constraint is violated.

To decide between the two alternatives, the scheduler uses two values: an estimate on the duration of the disconnection E and an estimate on the time to finish the full computation on the mobile device t_M (cf. Figure 4.4). These values are provided by the predictor and the statistics component. The scheduler evaluates two predicates. The first predicate evaluates to true if, according to the predicted disconnection duration E, the deadline cannot be kept. This is denoted as $t + E > t_{\max}$, where t is the current time and $t_{\max}$ is the deadline. The second predicate evaluates to true if computation on the mobile device is faster than waiting and can be denoted as $t_M < E$. Only if both predicates are true, the scheduler decides to start computation on the mobile device. While the scheduler is in "disconnected" mode and decides to wait, it will periodically verify its decision based on new predictions of the components. Notice, that uncertainty bounds for the prediction can be introduced to adapt the accuracy of the predictor and the respective cellular network situation, for example the mobility scenario.

When the scheduler is in "disconnected" mode and the statistics component

receives messages from the server, it triggers "recovery" mode. In this mode, the scheduler pauses all computation on the mobile device. It sends a recovery request to the server, containing the numbers of missing states. The server answers by sending previously computed states to the mobile device. Once the mobile device received all requested states computed on the server during disconnection, the scheduler returns to "connected" mode. Notice that the detector component can also trigger "disconnected" mode when in "recovery" mode.

4.5 Statistics Component

The statistics component collects data about three aspects: (1) timing information for solving states on the mobile device, (2) packet transfer time to the mobile device, and (3) the availability of the wireless link. All data is stored directly on the mobile device.

As mobile devices have different processors and execution time varies from device to device, timing information about the computation is needed for prediction of execution time. Execution time depends on the method used for solving the PDEs. Typically, such methods reduce the problem to algebraic equations, like in the heat equation example given in Section 2.1. Solving these equations is the hardest part of solving PDEs. The statistics component collects information about how long it took the specific device to solve the algebraic equations depending on the problem size. For example, for a given matrix size, it provides the mean time for computation as well as an upper and lower bound on the computation time. This information is used by the scheduler to estimate computation time on the mobile device.

In preliminary tests, we evaluated the time to solve matrix equations on different device classes. We choose the Conjugate Gradient method [S+94] which is often used when applying the finite elements method (FEM). For portability reasons, we choose the Java CG-Solver implemented in the Apache Commons Math library. Figure 4.5 depicts the mean time for solving linear equations on different device classes. We chose four different device classes: (1) classical sta-

tionary desktop PCs, (2) mobile laptops, (3) Smartphones like the LG Nexus 5, and (4) small and cheap system on a chip (SoC) like the Raspberry Pi 2. The error bars represent the maximum and minimum time for solving the algebraic equation. All three values, mean time, minimum time, and maximum time are provided by the statistics component. Maximum and minimum execution time are very close. Therefore the mean execution time describes the actual execution time of any execution very accurately. Also notice the difference in execution time on mobile devices, like Smartphones, and servers. This strongly supports our assumption of the server being much faster than the mobile device.

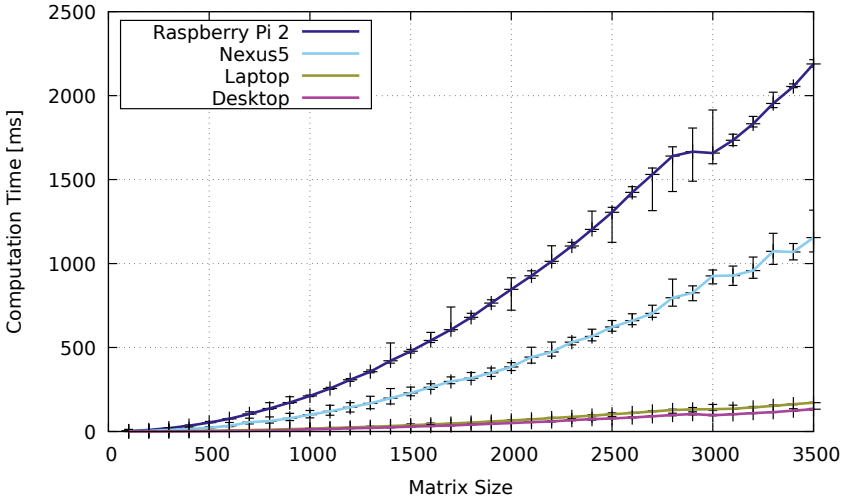

Figure 4.5: Mean time to solve linear equations on different device classes. Error bars represent maximum and minimum time.

In order to make the decision when to consider the mobile device to be disconnected, the detector needs detailed timing information about the transfer time of packets sent from the server to the mobile device. The server periodically sends probe messages to the mobile device. These messages contain timing information. Probe messages are sent via a connectionless protocol, like UDP, and are not retransmitted by the transport layer. The statistics component receives

probe messages and computes the relative delay of the packet. Notice that this relative delay is only used for comparison between delays of different packets. For this task, the delay does not have to be exact and clocks do not have to be synchronized.

For predicting the duration of disconnections, the statistics component collects information about lost packages for the predictor. Therefore, probe messages, periodically sent by the server, contain a sequence number. Using this sequence number, the statistics component is able to detect lost packages. This information is used by the disconnection duration predictor to predict the duration of disconnection once a disconnection is detected.

4.6 Detecting Disconnections

This section describes the detector component for detecting disconnections. Detecting disconnections as early as possible is an important task, as the system can react faster to the new situation.

There are two basic approaches for detecting disconnections: (1) detecting disconnections by lower layer information, e.g., signal-to-noise ratio (SNR), or (2) detecting disconnections using a timeout mechanism for reception of periodically sent probe messages. The former method requires additional link layer information like SNR, which might not be available on the application layer and which highly depends on the link layer protocol. We therefore focus on the timeout mechanism. The timeout mechanism introduces additional messages and needs a carefully chosen timeout value. However, it respects the layer model and does not need any additional information from lower layer protocols.

For the timeout method, the server periodically sends packets to the mobile device using a connectionless protocol, such as UDP. The mobile device registers reception of the packages. If no package arrives after the timeout t_{timeout}, the mobile device considers itself to be disconnected from the server. The challenge is to choose the timeout value t_{timeout}. Choosing t_{timeout} is a trade-off between detection time and rate of false positive disconnection events. If t_{timeout} is too

large, the detection time is increased, which might lead to suboptimal execution, e.g., the scheduler might start mobile computation too late and the deadline for the computation is missed. If t_{timeout} is too small, the detector might signal disconnection shortly before the missing probe packet arrives. This false detection might lead to wrong execution strategies and unnecessary computations on the mobile device, leading to increased energy consumption.

In order to deal with this trade-off, we choose t_{timeout} dynamically, depending on the actual situation of the execution. We use a lower timeout when the risk for missing the deadline is high. To this end, we consider the time when the computation could be finished by just using the mobile processor t_M. We subtract this value from the deadline, so that the values get smaller when we reach the critical point to finish the computation on the mobile device. Too small timeout values do not make sense, therefore we set the timeout to be at least $t_{\min}$.

To determine good values for $t_{\min}$, we did preliminary tests. We set up a server and a laptop. The server was well connected to the campus network. The laptop was located inside a train and connected to the server over 3/4G cellular network. The server sent messages to the laptop every 50 ms using UDP. The messages contained a 2 byte sequence number only. Server and laptop recorded timing information when messages were sent and received. Using this timing information we were able to compute the variance of message delay.

Figure 4.6 depicts the empirical distribution function of received messages over time. The time is calibrated to minimum transfer time of any packet. Figure 4.6 show that the empirical distribution function has a long tail. The longest message delay of any received message was 6227 ms. The number of messages never received on the mobile device was 16.42 %.

As a result of this tests, we found the 90 % quantile as a good trade-off between time to detection and number of false detections. The 90 % quantile guarantees to detect 90 % of received packets correct, while detecting 10 % of later received packages as disconnections. The 90 % quantile of the preliminary tests was 103 ms. Notice that any quantile can be easily identified based on historical data stored by the statistics component or requested by a central database for the

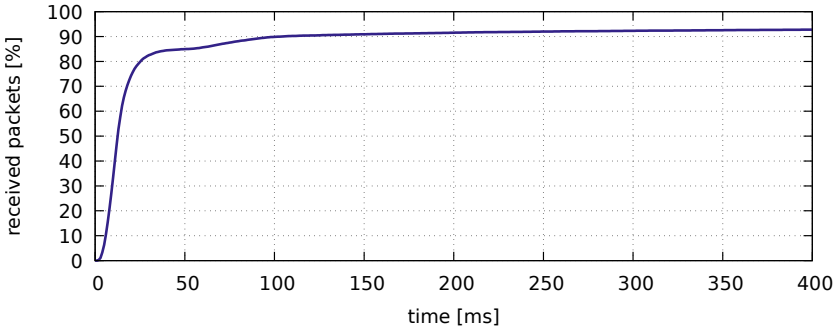

Figure 4.6: Empirical distribution of received packets over time.

given provider and location.

4.7 Predicting the Duration of Disconnections

In the previous section, we showed how to detect disconnections. This section covers the prediction of the duration of a disconnection based on collected data. For this prediction, we learn the characteristics of the network using recent data in a Markov Chain and use the expected error duration of this Markov Chain as prediction.

Higher-order Markov Chains are a common method to model burst errors in wireless networks [GF04]. We will use second-order Markov Chains to predict the link state, which might be either available (A) or disconnected (D). Thus, states of the Markov Chain are boolean. Second-order Markov Chains give the probability of the next state based on the current and the previous state. If a denotes the previous state and b denotes the current state of the link, $P((a, b) \rightarrow c)$ is the probability of c being the next state. However, Markov Chains are memoryless and do not depend on the longer history of previous link states.

For learning the Markov Chain, we use recent data about the link availability. Learning can be implemented by counting the number of three consecutive link states (cf. Figure 4.7). The probability $p = P((x, y) \rightarrow A)$, can be derived

by counting the combinations $a = \#\{(x, y, A)\}$ and $d = \#\{(x, y, D)\}$ for all combination of three consecutive link states in recent data starting with x followed by y. The probability p can then be set to $p = a/(a + b)$. In the special case of $a+b = 0$, the value should be set to $1/2$ to keep principles of probabilities.

For predicting the duration of a disconnection, we use the expected value of disconnections. If the device gets disconnected, the state of the Markov Chain will be shortly (A, D) and then remain (D, D). The probability p of the wireless link to become available again is in every step $p = P((D, D) \to A)$. The expected duration of a disconnection E can therefore be derived as

$$E = 1/p$$

We use this value to predict the duration of the disconnection. Notice, if the Markov Chain is not based on enough data, p might not represent the actual properties of the network. In this case, the predictor returns a very low estimate to collect more data. Markov Chains can also be shared among users of the same cellular network provider in the same region.

Figure. 4.7 shows how learning and prediction can be implemented. Notice, all functions are implemented in $\mathcal{O}(1)$. Therefore, this approach for predicting link availability can be implemented very efficiently and can be applied in an online fashion.

4.8 Evaluation

In the previous sections, we explained our method for solving PDEs on a mobile distributed infrastructure. In this section we will evaluate our method. First we will explain our evaluation setup, including the application, devices, and our setup for measuring real-world data. Second, we will use the collected data to evaluate the disconnection detector. Last, we evaluate our approach with our real-world collected data on the availability of the wireless link.

```
1:  backlog ← new queue()
2:  counter ← [0, ..., 0]
3:  procedure LearnLinkState(link state a)
4:      backlog.append(a)
5:      (a, b, c) ← first three link states in backlog
6:      counter[(c, b, a)] ← counter[(c, b, a)] + 1
7:      if |backlog| ≥ max_backlog then
8:          (x, y, z) ← last three link states in backlog
9:          counter[(x, y, z)] ← counter[(x, y, z)] − 1
10:         backlog.pop()
11:     end if
12: end procedure
13: function GetProb((a, b) → c)                    ▷ returns P((a, b) → c)
14:     sum ← counter[(a, b, c)] + counter[(a, b, ¬c)]
15:     if sum = 0 then return 1 / 2
16:     end if
17:     return counter[(a, b, c)] / sum
18: end function
19: function ExpectedDurationDisconnection
20:     prob ← GetProb((D, D) → A)
21:     if prob = 0 then return ∞
22:     end if
23:     return 1 / prob
24: end function
```

Figure 4.7: Learning and prediction using Markov Chains for the link state can be realized efficiently. Link states are represented as booleans, denoted as either available (A) or disconnected (D).

4.8.1 Evaluation Setup

Our evaluation setup consist of three parts, the application, the compute nodes and the wireless network.

As application, we assume a well-known textbook example, namely, the heat equation. We already introduced the heat equation as an example in Section 2.1.

It is given as

$$\frac{\partial u}{\partial t} - \nabla^2 u = 0.$$

and describes the evolution of the temperature u at any position x in an object over time t. We choose 1,400 equidistant steps for discretization of the positions x at any fixed time t. During the evaluations, the time discretization was chosen randomly, while the deadline was fixed. We ensured that the computation can be finished on the server. We assume to use an implicit scheme for the discretization. This method needs to solve a system of $1,400$ linear equations in every time step.

For the computation nodes, we assume the mobile device to be equally equipped as an LG Nexus 5 Smartphone and the server to have four cores at 2.6 GHz. For computation of one time step, the devices have to solve a system of $1,400$ linear equations. According to our preliminary tests in Section 4.5 (cf. Figure 4.5), solving such a system on these nodes would cost roughly 200 ms on the mobile device and 20 ms on the server. We neglect any additional time for other computations.

For the network, we use different assumptions for link availability, throughput and latency. For the availability of the link, we collected real-world data on the train and per-pedes as already described in Section 4.6. We sent packets every 50 ms from the server to the mobile device and recorded detailed timing information on the devices. For the final evaluation, we used a different cellular provider and different routes than in the preliminary tests. For throughput and latency we used the specification of recently deployed LTE. LTE promises to provide throughput of up to 100 Mbit/s [ZM07]. However, technology deployed today mostly provides only up to 50 Mbit/s. One state plus metadata fits in 12 UDP packets, where each packet has 512 byte payload. Using a 50 Mbit/s link, we are able to send over 950 states per second. We therefore simply assume unlimited bandwidth.

Figure 4.8 depicts the relative arrival times of packets sent by the server and the state of the link in one of our collected real-world datasets. The state is assumed

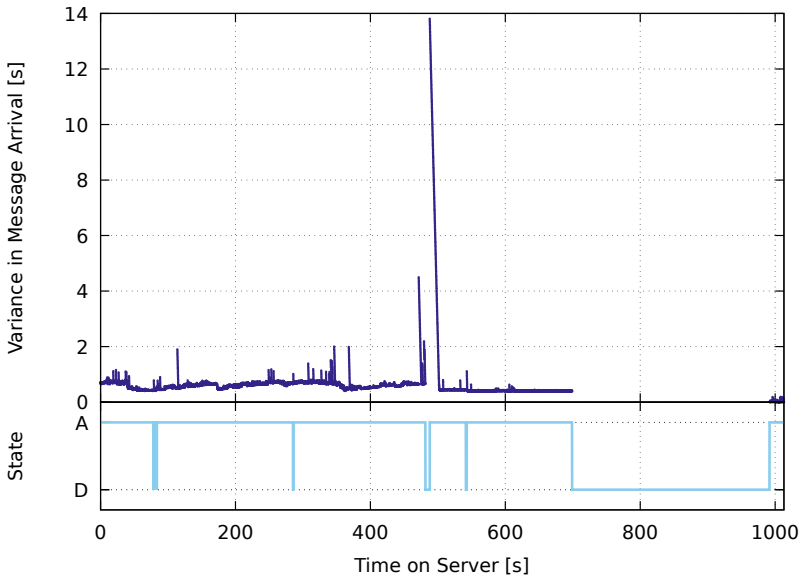

Figure 4.8: One of the sample collected per-pedes and on the train. On top, the variance in message arrival is depicted. On the bottom, the link state ('A': available, 'D': disconnected) is depicted.

to be connected if packets sent by the server eventually arrived on the mobile device. The graph shows two important pieces of information. First, latency can be up to nearly 14 s. Second, high latency of particular packets does not imply disconnection. For instance at 471 s, the mobile device was not disconnected, but packages had a very high latency of 4.5 s compared to other messages. One possible explanation for this high latency are retransmissions of the cellular link layer in areas with bad reception. Especially at underground stations on the train, but also under small obstacles such as bridges, we observed an increased message delay.

4.8.2 Evaluation of the Disconnection Detector

For the evaluation of the disconnection detector, we use the real-world data of cellular networks. In Section 4.6, we already described how we did preliminary tests and found a 90 % quantile for the minimum timeout for detection of disconnections a good choice. However, in the data collected for the evaluation, the 90 % quantile is 301 ms, which is much higher as in the preliminary tests, where it was 103 ms. The median message transfer time of eventually received messages was 54 ms, whereas the maximum was 7153 ms.

Figure 4.9 depicts the latency of messages, the disconnections and the results of the detector of a sample trace. The deadline was at the end of the plot. We used received messages to simulate progress of the computation. Our dynamic timeout mechanism only detects messages at the end of the computation, where detection is critical in order to meet the deadline. It assumes to have detected six disconnections. However, five are false positives, all with message latency over 464 ms, which are received eventually. One of the false positives is received even after 3.9 s, whereas the only real disconnection correctly detected by the detector, has a latency of 417 ms. In other words, the probe message with the smallest delay was a disconnection, while the other messages were eventually received. Any timeout mechanism detecting the real disconnection has to detect the others.

4.8.3 Evaluation in an LTE Scenario

In the remainder of this section, we will evaluate our method to solve PDEs on mobile devices against two other approaches, namely a pure offloading approach and our approach without the disconnection predictor.

The pure offloading approach simply computes all states on the server and sends the results to the mobile device. While the link is disconnected, there is no progress on the mobile device. However, the mobile device sends a retransmission request to the server, once the link is recovered from disconnection. If the server receives a retransmission request, it answers with data of states available on the

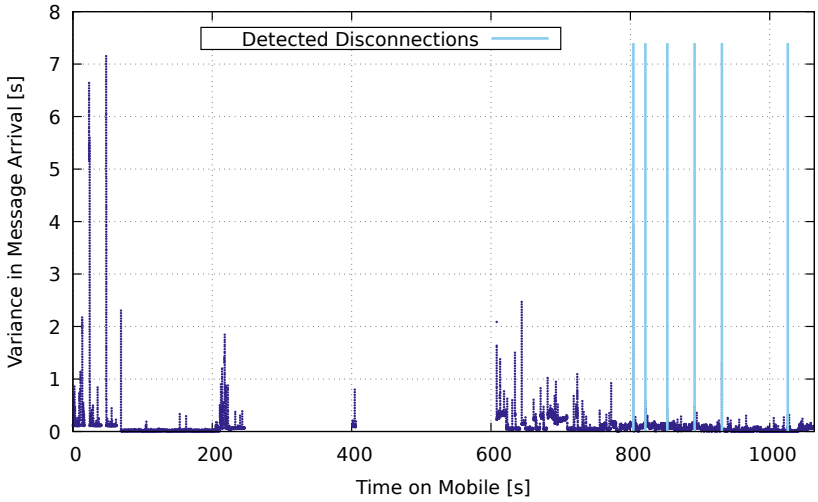

Figure 4.9: Sample data and decision of the disconnection detector. The detector returns 5 disconnections, depicted as light blue lines. Only the third disconnection is an actual disconnection, the others are false-positives reported by the detector.

server but not yet available on the mobile. The pure offloading approach does not use the mobile processor. Therefore, it provides the optimal solution, if the constraint on the deadline can be fulfilled.

Our approach without the disconnection predictor does not have an estimate on the length of disconnections. It therefore starts computation of any missing states on the mobile device, once the device is disconnected. If the connection is recovered, the mobile device sends a retransmit message to the server and stops the mobile computation. The server will then answer with all its available states, requested by the mobile device.

To evaluate the performance of the three methods, we used different deadlines and tested each of the methods with random time discretization. We were interested in two aspects: (1) the fraction of simulation runs, where the deadline could be kept and (2) the additional energy consumption, expressed as the time

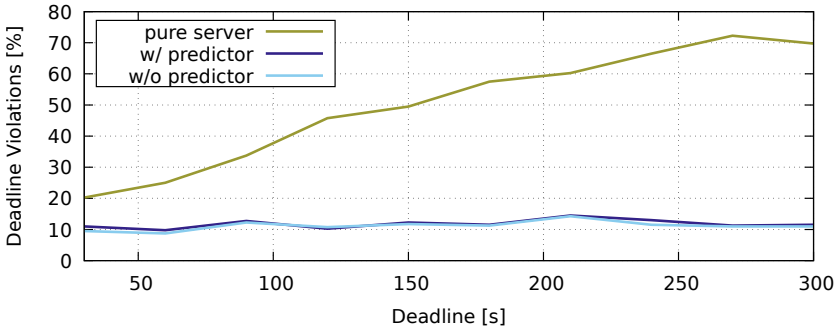

Figure 4.10: Fraction of deadline misses over variable deadline of the three approaches.

of the mobile processor usage.

Figure 4.10 depicts the fraction of deadline misses over the deadline. As the deadline is later, the pure offloading approach misses more and more deadlines. However, our approaches with and without the predictor have a very constant rate of deadline misses, independent of the actual deadline. Compared to the pure offloading approach, our approach without the predictor is able to increase the fraction of kept deadlines by 61.25 % and our approach with the predictor by 61 %. If the application requesting the results is able to handle 10 % to 15 % deadline misses, our approaches can be used, while the pure offloading approach does not provide sufficient results.

Figure 4.11 depicts the performance in terms of the optimization goal. Whereas the pure offloading approach yields optimal solutions with no mobile processor use, it mostly does not fulfill the constraint on the deadline of the computation. Our two methods use the mobile processor and are able to keep the deadline more often. However, they also use the mobile processor and therefore do not yield energy-optimal solutions. Comparing our approaches with and without the predictor, the approach with the predictor is able to reduce energy consumption by up to 74 %. Therefore, the predictor is an essential component to provide better results in terms of energy consumption on the mobile device.

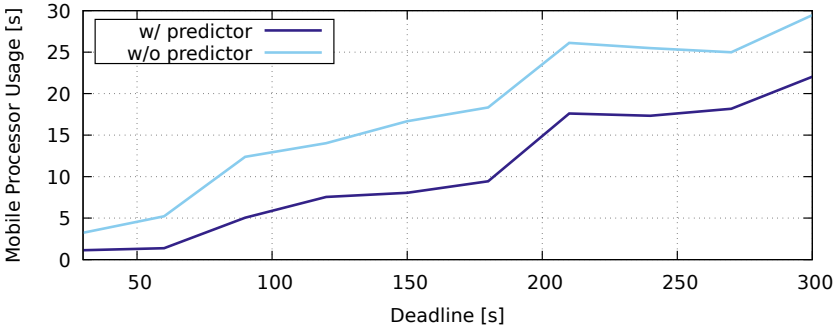

Figure 4.11: Energy consumption on the mobile device of our methods with and without the predictor.

Overall, our approaches with and without the predictor are able to fulfill the constraint on the deadline much better than the pure offloading approach. However, using the predictor, we are able to save a constant time of mobile processor utilization. Especially in a scenario such as AR or MCPS, where the solver has to provide results continuously, this constant time adds up to a linear improvement in energy consumption on the mobile device.

4.9 Related Work

This section discusses related work for providing robust execution of simulations in distributed environments. Related work is structured in classical methods from high-performance computing and code offloading methods for execution in distributed mobile environments.

4.9.1 High-Performance Computing Methods

Classically, numerical simulations are computed on supercomputers using high-performance computing (HPC) methods [RD15]. HPC assumes a set of distributed but closely located nodes connected via a wired dedicated low-latency and high data rate network. Additionally, nodes are considered to be homogeneous and

have the same hardware.

Usually, communication between nodes is implemented using the message passing interface (MPI) standard [RJ15]. Starting with standardization in 1993 [MPI93], MPI provides methods to distribute data and tasks across computing nodes for parallel computation. This makes it easier for the application programmer of numerical simulations to implement scalable, massively parallel applications.

The MPI standard assumes reliable communication provided by the underlying communication subsystem [MPI15]. Therefore, in case of disconnections, the underlying subsystem has to handle recovery. However, TCP for instance does not implement recovery after disconnections. Therefore, other mechanisms need to be implemented by the user [GL04, HSML07].

Limitations of High-Performance Computing Methods

While the MPI standard can be used to implement mobile simulations, they require handling of disconnections by the user. Therefore, disconnection detection and recovery needs to be implemented separately.

In general, HPC methods cannot be used for mobile applications, as they assume reliable communication and homogeneous architectures. In contrast, mobile devices and servers are heterogeneous and usually have different processor architectures. Additionally, the cost of communication is much higher for wireless communication technology than for closely distributed computation nodes in supercomputers. To this end, HPC applications are also considered to be relatively static compared to mobile applications that need to dynamically react to different situations.

In contrast, the approach presented in this chapter considers heterogeneous computing nodes and presents methods for deciding on local and remote computation.

4.9.2 Code Offloading

In code offloading, mobile applications are split into modules, where the execution of modules is distributed between mobile device and connected server. To this end, approaches reduce the latency [RSM+11, GRA12, YKC+13] or the energy consumption on the mobile device [CBC+10, CIM+11] for the execution.

The main problem is to decide how to distribute modules of the application. One solution for this problem is reduced to a graph partitioning problem, where nodes represent different modules and edges represent dependencies between modules, e.g., one module calls a function of another module. Nodes and edges are adjunct with cost for computation and communication on different nodes. To find the best partitioning of modules, the graph has to be partitioned into two distinct sets with respect to energy, time, throughput, latency, volume, and space constraints.

Safe-Pointing Recent publications in code offloading also deal with the problem of disconnections of the wireless network [BDR14, ZNW15, Ber18]. Berg et al. proposed to use safe-points of the application that will be sent to the mobile device from time to time. When a disconnection is discovered, the last used safe-point can be used on the mobile device to eventually start the computation on the mobile device and therefore save energy and time for reconstruction of the same state as on the server.

Limitations of Code Offloading

Code offloading is application agnostic, which may lead to non-optimal distribution in case of specific applications. Time-dependent numerical simulations only support one execution thread, where the result of the previous state is required for calculation of the current state. Code offloading therefore can only split the application into a row of modules where it has to execute one after another.

While solving one step of the numerical simulation is more performant when it is calculated on parallel processors, the state of the simulation is much smaller

after each time step is calculated. Therefore, if safe-pointing is applied during parallel executions, it may result in non-optimal communication strategies at a much higher cost for the calculation during disconnections. Additionally, safe-pointing is implemented on a virtual machine layer, which results in higher overhead than application-specific serialization.

4.10 Summary

In this chapter, we described how to solve simulation problems using mobile devices and servers connected over wireless communication links. Wireless links such as implemented via 3/4G cellular networks are subject to burst errors, where multiple successive packets are not delivered and the link is unavailable for a longer period, i.e., the device is temporarily disconnected. By using prediction on the availability of the wireless link, we showed that we can improve the energy consumption on the mobile device during disconnections while fulfilling constraints on the deadline of the computation.

While frequent disconnections are one of the major problems in mobile computing, mobile simulations also have to provide timely results in case of low data rates. The following chapter therefore considers to use surrogate models to find a trade-off between computation resources on the mobile device and current communication resources of the wireless network.

5

Using Surrogate Models for Efficient Solution of Time-Dependent Problems

In the previous chapter, we provided concepts to execute the simulation either on the mobile device or on the server. In evaluations, we have seen that execution on the server and streaming results to the mobile device was much faster than execution on the mobile device in cases of no disconnections. In this chapter, we will explore a different form of distribution of the execution between mobile device and server that is very specific to simulations and quality-dependent algorithms. In particular, we will utilize the concept of surrogate models, i.e., low-quality and low-complexity simulation models. As surrogate models have much lower complexity than the original full simulation model, they are suitable for execution on the mobile device. Intuitively, the server can run the same surrogate model in addition to the full simulation model and provide updates depending on the quality requirements of the user. In this way, the execution requires little more overhead on the server side, but much less overhead on the network and the mobile processor.

This chapter is structured as follows. First, more details on the system model are provided, followed by the problem statement. The following three sections provide three methods to solve the problem. After the methods have been introduced, they are evaluated in a synthetic testbed based on real-world data. The last two sections in this chapter discuss related work and provide a summary.

Parts of this chapter have been published in [DNDR19].

5.1 System Model

This section introduces our system model for offloading of time-dependent numerical simulations. We first describe our model for the time-dependent simulation and then provide our model of the mobile environment, consisting of mobile device, remote server, and wireless communication network.

5.1.1 Time-Dependent Simulations

Time-dependent numerical simulations are based on differential equations describing the behavior of the system w.r.t. continuous time and space. Such equations need to be discretized in order to be solved. Time-discretization divides the continuous time into $n_t + 1$ time steps. Each step represents the system state S_i at fixed time t_i. For simplicity, we assume that the time of the first step is $t_0 = 0$ and the time for the last step is $t_{n_t} = 1$. Then, the time resolution is $\Delta t = 1/n_t$ (cf. Figure 5.1).

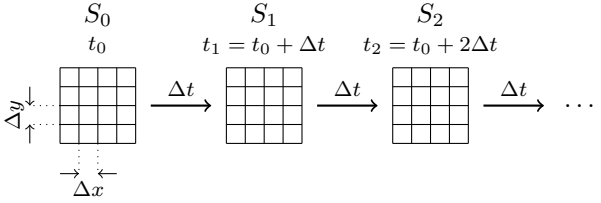

Figure 5.1: Time discretization and space discretization of numerical simulations.

Next, we first describe how the simulated system at each time step is discretized in space, how the transition between time steps is implemented, and how the computation can be optimised to provide computationally cheaper approximations of the simulation problem.

Representation of Time-States and Transition Between States

Time-states S_i represent the state of the simulated system at discrete points in time. While space is defined continuously in the differential equation, it also needs to be discretized. To this end, the system is only observed at fixed points in space, e.g., at points forming a grid with mesh width Δx (cf. Figure 5.1). Values of the simulation at these points form a vector. The size of the vector depends on the spatial discretization. If finer discretization is required, the size of the vector is increased. The size of the vector later also depends on the complexity of the computation.

Transition between time states is implemented by solving an algebraic problem in a numerical solver. The output of the solver is the state vector of the next time state S_{i+1}. Input into the solver is the old state vector S_i and problem-specific information, e.g., a problem matrix and a vector forming the algebraic problem. Typically, there is a choice between multiple classes of algebraic problems for the same differential equation, leading to different trade-offs between quality and complexity of the computation. For instance, simulating heat propagation using the heat equation yields various discretization methods that can be generalized into two classes: implicit methods and explicit methods. While implicit methods are computationally more expensive, they provide better quality than explicit methods. Such decisions on the trade-off between quality and complexity motivate the use of surrogate models.

Reference Model and Surrogate Model

We assume two different implementations of the model, a reference model and a surrogate model. The reference model defines the "ground truth" of the simulation. It is defined over fine-grained discretization grids, enabling accurate predictions of future system states. While it provides accurate results, it is expensive to compute. On the other hand, the surrogate model is computationally less expensive while providing only a lower quality than the reference model. Surrogate models can be obtained by using an explicit method rather than an

implicit method or by changing Δx of the discretization grid to have a lower space resolution.

We will later compare the reference model and the surrogate model at the same time step. To compare results of both models, the vector of the reference model has to be mapped to the same dimensionality as the vector of the surrogate model. We assume that this mapping is provided by a transformation matrix $T_{R \to S}$. Additionally, to simplify the notation for comparison between models, we assume that time-discretization of the reference model and the surrogate model is the same. However, the reference model could also be configured to compute multiple, say n_{ref} steps for one surrogate step. This way, the reference model will have a time discretization with $\Delta t_{ref} = \Delta t / n_{ref}$ and we are able to compare results of the reference model and the surrogate model every n_{ref} time steps.

Mixing Simulation Models for Approximate Solutions

The solutions of one simulation model form a chain of time steps (cf. Figure 5.1). To provide better quality, the chains of the reference model can be used to update the surrogate model chain. Each of these updates forms a new chain of approximate solutions. For instance, the surrogate model is updated at time step, say 5, to set its state to the state of the reference model. The resulting chain of simulation results may be significantly different compared to the original surrogate simulation chain without the update.

5.1.2 System Components

To compute results of the numerical simulation, the system consists of two compute nodes, namely the mobile device and the server. The server is located in a central location in the network and receives data from sensors (cf. Figure 5.2).

The mobile device is carried by the user. The user directly interacts with the mobile device and requests simulation results. The mobile device has an energy-efficient but slow processor. In contrast to the server, it is very resource-limited

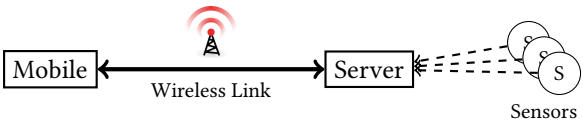

Figure 5.2: The two compute nodes, server and mobile device, and connected sensors.

and depends on batteries providing limited energy.

The server receives data from cloud-connected sensors and collects and stores relevant data to form the initial state for the simulation. Therefore, the initial state for the simulation is only available on the server and needs to be communicated to the mobile device before any simulation model can be executed.

Server and mobile device are connected via wireless communication, e.g., 3/4G cellular networks or IEEE 802.11 WiFi. Wireless communication is subjected to dynamic latency and throughput, which might be very low in some cases.

5.2 Problem Statement

After describing the system model, this section describes the problem statement. We first define quality for approximate solutions and then define the optimization goal of the system.

Quality of approximate solutions is defined by comparing approximate solutions to the reference model using a user-defined norm $\|\cdot\|_U$. Let S_i^A denote the approximate solution and S_i^R denote the reference solution for time step $i = 0, \dots n_t$. The quality of time step i is then defined as $q_i^A = \|S_i^A - T_{R\to S} S_i^R\|_U$, where $T_{R\to S}$ is the transformation matrix between reference model results and surrogate model results (cf. Section 5.1.1). The overall quality of the approximate

solution for all time steps is then defined as

$$Q_A = \max_{i=0,\dots,n_t} q_i \tag{5.1}$$

$$= \max_{i=0,\dots,n_t} \left\| S_i^A - T_{R\to S} S_i^R \right\|_U . \tag{5.2}$$

One example for the user-defined metric $\|\cdot\|_U$ is to compare approximate solution and reference model solution by the maximum difference at any point, e.g., the maximum temperature difference of a heat simulation.

Having defined quality, we can now define the overall goal of the system. The goal of the system is to minimize the latency until approximate solutions are available on the mobile device. Approximate solutions have to fulfill quality constraints given by the user, i.e., the user provides $Q_{\max}$ and the solution has to fulfill $Q_A \leq Q_{\max}$. This way, the user can define the maximum difference of the approximate solution to the reference simulation and the system provides an approximate solution as fast as possible.

5.3 Stream Approach

We briefly describe the straight forward streaming approaches. These approaches only serve as baseline and for comparison to our approaches that will be presented in the next sections. Streaming approaches compute all steps of the simulation on the server and communicate results to the mobile device. We introduce two approaches, the *simple stream approach* and the slightly more sophisticated *advanced stream approach*.

The simple stream approach computes the reference simulation on the server and communicates all steps to the mobile device. Therefore, all results on the mobile device have the best possible quality and $Q_A = 0$.

The advanced stream approach also computes the reference simulation on the server. However, it will reduce the quality of the simulation states before they are sent to the mobile device. In particular, it will reduce the resolution of the simulation to the surrogate model discretization. This way, the quality

is still $Q_A = 0$, while the volume of data communicated over the wireless communication link is significantly reduced.

While the simple stream approach represents the result of an unbalanced partitioning, which could be the result of code offloading for the simulation problem, the advanced stream approach is able to reduce the communication overhead at no quality loss. However, the advanced stream approach still has to communicate all simulation steps over the network. The following sections will introduce our approaches, which require much lower communication overhead over the stream approaches.

5.4 Full Update Approach

While the previously introduced stream approaches are able to meet any quality requirements, they do not consider the mobile device for computing simulation results. Therefore, this section introduces the full update approach, which reduces the requirements on the wireless link as it uses computation on the mobile device.

The general idea of this approach is to execute the same simulation on the surrogate model simultaneously on the mobile device and the server. Thus, the server knows which (approximate) results have been calculated by the mobile device using the surrogate model. By comparing to the exact solution of the reference model, the server can send updates to the mobile device at selected states, whenever the surrogate model yields results of insufficient quality.

Figure 5.3 depicts the components for the full update approach. Reference model and surrogate model are models of the simulation that implement time transition of the states. The update decision component will decide whether to send an update to the mobile device. The mobile state tracker holds the last known state of the simulation on the mobile device for the previous state. On the mobile device, the update integration component combines possible outputs of the surrogate computation.

While the reference model and the surrogate model have already been introduced in the previous sections, we will explain the remaining components in the

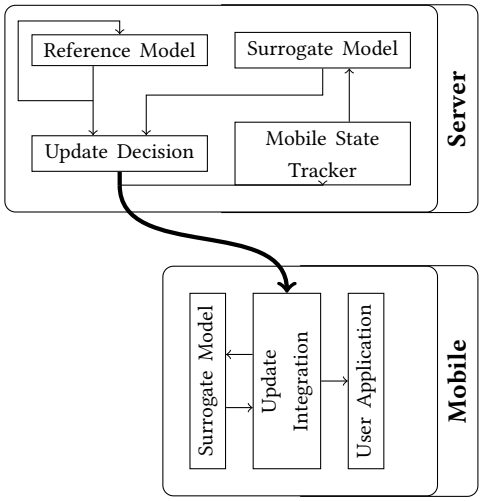

Figure 5.3: Overview of the full update approach

following subsections.

5.4.1 Mobile State Tracker

The mobile state tracker provides the previous state of the mobile device on the server. As the surrogate model is deterministic and will return the exact same result as on the mobile device, the server can use the result even before it has been computed on the mobile device.

For the initialization, the initial state needs to be communicated to the mobile device in reduced-model resolution. The mobile state tracker will then be initialized with the same initial state.

5.4.2 Update Decision

The update decision is based on the requirements of the user. To this end, the update decision component receives the current state of the reference model and the surrogate model. It computes the difference of the states after transforming

the reference state to the same spatial discretization grid as the surrogate state. Afterwards, it will check whether the quality of the result of the surrogate model is sufficient. If it is sufficient, it will send a quality certification message to the mobile device. If it is not sufficient, it will send an update of the vector representing the current reference state to the mobile device.

Notice that, before sending, the update is transformed to the spatial discretization level of the surrogate model since this provides the quality such that the mobile device can continue calculating future states from the updated model.

The update decision component will also update the tracked state on the server. If a certification message was sent, it will use the result from the surrogate model. If an update was sent, it will update the mobile state with the result from the reference model.

5.4.3 Update Integration

The update integration component is executed on the mobile device and receives messages from the server. It may invoke the surrogate model and provides the result as current simulation state to the user application.

If the update integration component receives a certification message, it will use the result from the surrogate model and provide the result to the user application. If it receives an update message, it parses the state and provides it directly to the user application since the result then is directly derived from the reference model. To this end, the update integration module always has to wait for the next message from the server. If no certification message is available, it cannot provide any result to the user application since the result of the surrogate model might not provide sufficient results.

There are optimistic and pessimistic implementations of the full update approach. The optimistic implementation assumes that no update has to be made and the computation using the surrogate model is sufficient. The pessimistic implementation assumes that the surrogate model will not provide sufficient quality. To this end, the optimistic approach will always start the computation of the next state in the background, whereas the pessimistic approach only computes

on request by the server.

Compared to the stream approach, the full update approach will reduce the traffic of the wireless communication link, as certification messages will be much smaller than streaming results. The traffic of the network now depends on the accuracy of the surrogate model. However, this approach adds slightly more overhead on the server side, which now has to compute the surrogate model in parallel to the reference model. In the next section, we will see how we can further reduce the traffic of the network by just sending partial updates.

5.5 Partial Update Approach

The previously introduced full update approach always has to communicate full state updates for surrogate states violating quality requirements. This results in a huge communication overhead even in cases where only a small part of one surrogate state violates quality constraints. In this section, we therefore introduce our partial update approach, which is based on the full update approach, but will only update parts of the vector representing the approximate solution from the surrogate model (cf. Figure 5.4).

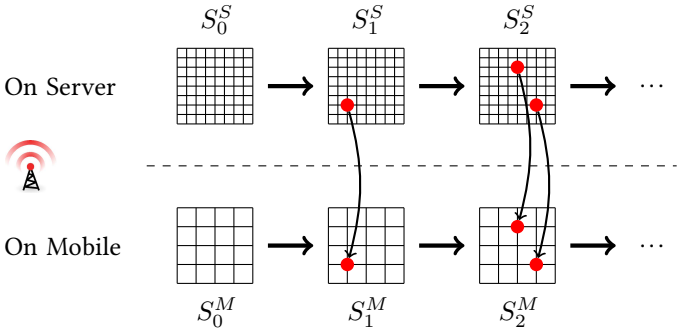

Figure 5.4: Partial point updates.

Updating single values of the simulation is not straight forward since the simulation model might be sensitive to external changes of the simulation state. For

instance, if we consider a heat simulation, the simulation model and numerical calculation assumes the solution to be continuous, while when randomly updating values, the solution becomes discontinuous, i.e., updated values add sharp edges to the simulation state. Such discontinuities lead to numerical instabilities which would never occur in normal calculations for the simulation and which the model might not be able to recover. Therefore, more sophisticated approaches are required.

To reduce the discontinuity when updating single values, we use data assimilation techniques. Data assimilation emerged from weather simulations, where sensor data updates the simulation state, which leads to similar problems [LSZ15b]. To prevent such problems, data assimilation techniques identifies the correlation between parts of the simulation state. When updating one value, data assimilation uses this correlation and updates all correlated values respectively. Therefore, the number of discontinuities is highly reduced and the simulation model quickly recovers from single point updates.

The idea of the partial update approach is to apply data assimilation and treat single point updates as sensor observations. In this case, sensor observations are perfect, since they are taken from the reference simulation, which is our ground truth. This simplifies the calculation of data assimilation methods, since they normally assume inaccurate observations.

In the following, we first briefly describe our data assimilation of choice, the ensemble Kalman filter, before we discuss how the partial update approach changes the update decision and update integration from the full update approach.

5.5.1 The Ensemble Kalman Filter

The ensemble Kalman filter (EnKF) provides a solid and frequently applied framework for data assimilation [Eve03]. The general idea of the EnKF is to use multiple states to track uncertainties (cf. Figure 5.5). These states are called ensemble members. Initially, ensemble members are generated using random perturbation of the initial state. For every simulation state, the next state of the ensemble members is computed using the surrogate model. The number of ensemble mem-

bers n_e can be small. It has been shown that even some complex applications do not require more than 50 ensemble members [HM98, Kep00].

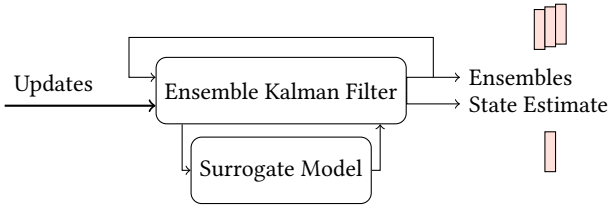

Figure 5.5: Simplified operating principle of the ensemble Kalman filter.

Generation of Ensemble Members

We generate ensemble members by perturbation of a reference state S_i. To this end, we add a random vector to the initial state to form an ensemble member $e_i^{(j)} = S_i + r^{(j)}$. The random vector $r^{(j)}$ is sampled such that the mean of ensemble members track the state of the reference simulation, e.g., using the standard error between reference and surrogate model.

In order to have the same result available on the mobile device as on the server, the computation has to be deterministic. To provide random numbers to the EnKF, we therefore use a deterministic random number generator with well-defined seed for the random vector. As the seed should change for every state, the server chooses a basic seed during initialization. We then use a deterministic function of the basic seed and the state number to calculate the seed for state perturbation of the current state.

Combining Simulation Model and Observations

Combining the state of the surrogate model and partial updates consists of two steps, namely the forecast step and the analysis step. In the forecast step, the surrogate model is applied for all current ensemble members $E_i = (e_i^{(1)}; \ldots; e_i^{(n_e)})$. This generates the forecast ensembles for the next ensemble members $F_{i+1} = (f_{i+1}^{(1)}; \ldots; f_{i+1}^{(n_e)})$.

Partial updates are communicated as set of pairs (*position*, *value*) where, for every updated value, the position of this value in the surrogate state vector is given. This representation is translated into an update vector u_{i+1} containing just the values and a measurement operator H_{i+1} mapping respective entries of the surrogate state vector to the update vector.

For the analysis step, the next state has to be combined with partial updates u_{i+1} by using the so called Kalman gain K_{i+1}. The Kalman gain defines the sensitivity of the difference of partial update u_{i+1} and forecast state F_{i+1}.

The analyzed ensemble members are then calculated as

$$e_{i+1}^{(j)} = f_{i+1}^{(j)} + K_{i+1}(f_{i+1}^{(j)} - H_{i+1}u_{i+1}).$$

The analyzed simulation state as output for the user is the ensemble mean of all analyzed ensemble members. Further details about the EnKF and the computation of the Kalman gain K_{i+1} can be found in Section 2.3.

5.5.2 Update Decision

For identification of parts that need updating, we introduce the concept of violation points. Violation points are points in the result of the surrogate model that violate the quality constraint. How these points can be calculated depends on the norm used for specifying the quality. In general, we distinguish between maximum norm and any other norm.

If the maximum norm is used for the quality constraint, every point has a maximum distance to its corresponding point in the reference state. Violation points are therefore all points that differ too much from the current reference state.

For other norms, e.g., the Euclidean norm, the computation of violation points is slightly more complex and requires an iterative process. To this end, we build the set of points that require updating. Initially, this set is empty. If quality requirements using the current updates cannot be met, the point with the maximum error to the reference model is included in the updates. This is repeated until the

quality requirements are met.

Once the decision on the set of points to update is made, the update is sent over the network. Additionally, on the server side, the update is applied by the mobile state tracker in order to derive the same state as on the mobile device. In contrast to the full update approach, the mobile state tracker not only keeps track of the current simulation state on the mobile device, but also of ensemble members expressing the uncertainty of the current state.

5.5.3 Update Integration

The update integration component on the mobile device receives the update from the server. It holds the current ensemble members of the surrogate model. To this end, it will calculate the prediction model and prepares all steps in the calculation of the EnKF to provide the next state of the simulation for the user application.

Notice that before using the EnKF, we used the Kalman filter as data assimilation technique. The Kalman filter tracks uncertainty of the states using a covariance matrix. This matrix is quadratic to the problem size and therefore much more computationally expensive.

5.6 Evaluation

The previous sections introduced the full update approach and the partial update approach. This section evaluates both approaches against the two streaming approaches described in Section 5.3, which are the state-of-the art for providing simulation results to mobile devices. In this evaluation, we consider different mobile network setups and different assumptions on the accuracy of the surrogate model. As benchmark simulation problem, we are using a 2d heat simulation based on the well-known heat equation. Before describing details of evaluation results, we first introduce the evaluation setup.

5.6.1 Setup

We evaluated our approaches on a distributed test bed consisting of a Raspberry Pi 3 as mobile device and a powerful server. The Raspberry Pi 3 uses a system-on-chip (SoC) hardware similar to the SoCs used by mobile devices. It features a quad-core Broadcom ARM CPU at 1.2 GHz and 1 GB RAM. The server is a commodity off-the-shelf server featuring a quad-core Intel Xeon E3 CPU at 3.4 GHz and 16 GB RAM.

We emulated the cellular network connecting mobile device and server using the Linux Kernel Packet Scheduler on both nodes. To this end, we added queueing disciplines that restrict the data rate using a token bucket filter (TBF) and delaying packets using the netem module. To set parameters of the TBF and for the delay, we measured the performance of real cellular networks using HSDPA and LTE. We found that, in extreme conditions, data rates can be as low as 50 kbit/s with around 1 second latency over longer periods. However, as we assume data rates to increase in the future and as our approaches are much better for lower data rates, we assume a data rate of 1 Mbit/s.

Our approaches and the simulation are implemented in Python (version 2.7.13) and NumPy (1.14.3). To accelerate the computation, NumPy was linked with OpenBLAS (0.2.19), which is available for the server and mobile architecture. Serialization is implemented using Protobuf (3.5.2), and data was communicated using TCP as transport protocol. We used background threads and queues in order to send data parallel to processing. As deterministic random number generator, we use the Mersenne twister sequence [MN98] as implemented in Python.

As simulation problem, we choose the popular and well understood 2d heat equation with Dirichlet boundary conditions. We implemented two numerical solvers, one using explicit Forward-Time Central-Space (FTCS) discretization and the other using the Alternating Direction Implicit method (ADI) with the Crank-Nicolson method for 1d discretization. Throughout the evaluation, we used the explicit FTCS implementation as surrogate model and the implicit ADI implementation as reference model. For the initial state, we choose random values in the interval $[0, 1]$ and set the boundary to 0.

The execution of the simulation depends on many parameters for discretization and accuracy of the numerical model. We ran our evaluations with different parameters and received results similar to the results reported in this section. For the final evaluations, we used the following default parameters. We assume the temporal discretization as $\Delta t = 0.0001$ with 100 states. The maximum error allowed between reference and surrogate model was 2^{-7}, i.e., we allowed less than $1\,\%$ of error between reference model and surrogate model.

To compare different surrogate models, we defined quality levels as uniform grids for space discretization. This way, we can use different discretization grids and define surrogate models on each of the levels. To this end, our uniform grid implementation consists of different levels, where each higher level includes points of all lower levels plus points in between of all existing points. The number of points on this levels quickly grows, e.g., the later often referred level 5 contains 1089 points, while level 6 contains 4225 points.

In the following, we will first evaluate the impact of quality onto the number of full state updates and sizes of partial state updates, since the number of updates required might depend on the required quality. We will then evaluate the latency for varying data rate of the network, full update probability, sizes of partial updates, and surrogate problem size.

5.6.2 Accuracy of the Surrogate Model

The accuracy of the surrogate model impacts two quantities: (1) the number of points requiring partial updates and (2) the distribution of the number of points in states violating quality constraints. To this end, we introduced the terms *violation states* and *violation points*. Violation states are states of the simulation that do not fulfill quality requirements and therefore need updating in the full update approach. Similarly, violation points are points in one simulation state that are required in partial updates. Intuitively, violation states and violation points depend on the maximum error that is allowed for the application. To provide an overview of the distribution of violation states and violation points, we recorded them for different error bounds (maximum error) in the 2d heat

equation with random initial state.

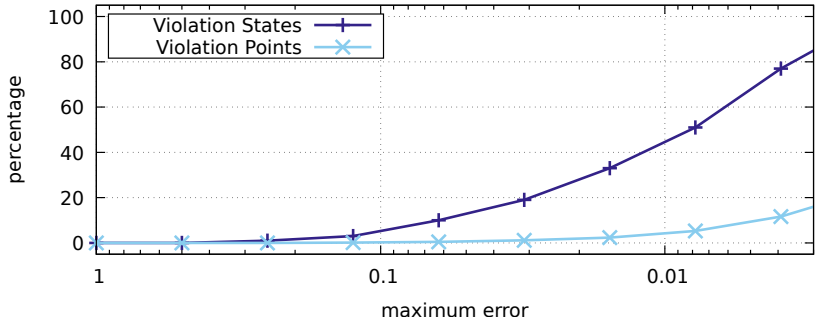

Figure 5.6: Ratio of violation states and violation points.

Figure 5.6 depicts the percentage of states that are required as updates, and percentage of points per state requiring updating for partial updates. While some partial updates require many point updates, the majority of partial updates only require few points. For maximum error 2^{-7}, around $50\,\%$ of states require updating, while only $5.2\,\%$ of points need to be updated. We therefore assume a state update probability of 0.5 in the following, if not stated otherwise. Notice that this reduced the volume of data to be communicated from server to mobile device by $50\,\%$.

5.6.3 Impact of Channel Data Rate

Mobile devices face varying data rates of the wireless communication channel. Especially in areas with bad signal strength, e.g., indoors in basements, data rates drop to low rates down to 50 kbit/s. This is inline with the Shannon-Harley Theorem which would require higher bandwidth in cases of higher signal-to-noise ratio to keep constant data rates.

We measured the overall latency of the approaches for different data rates. Latency is taken from the time the program is started on the mobile device until all results are available in sufficient quality on the mobile device. For the

streaming approach, this includes sending the initial request to the server and then receiving all states of the simulation. For the full update approach, this includes sending the initial request and then computing the surrogate model on the mobile device, whereby the server either acknowledges the surrogate model quality or sends an update to correct quality of the surrogate model. For the partial update approach, this includes sending the initial request, receiving the initial state and then executing the ensemble Kalman filter alongside the surrogate model while potentially receiving point updates for each state from the server.

After setting different data rates, we ran our approaches multiple times and recorded the median latency. All other parameters are set to the default configuration introduced in Section 5.6.1.

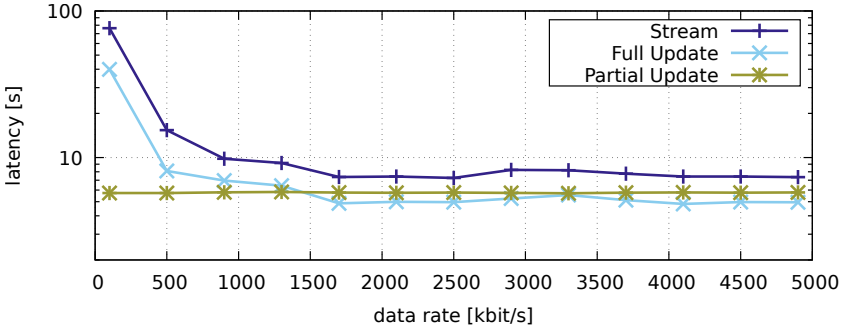

Figure 5.7: Latency of the four approaches over data rate.

Figure 5.7 depicts results for the three approaches over the data rate of the wireless channel. For low data rates, the partial update approach provide results up to 13.3 times faster than streaming, while the full update approach is only 9.4 times faster. However, for high data rates, the full update approach is marginally faster than the partial update approach. Both have a speedup of 50 % compared to streaming. In general, data rate has only very little impact on the partial update approach, while it effects stream and full update approach. For varying data rates, the combined approach is therefore the best choice, while the full

update approach might be considered for higher data rates.

We found that most of the overhead of streaming is caused by the deserialization on the mobile device. However, our implementation is based on Protocol Buffers, which is considered to be one of the fastest serialization formats [SM12]. Partial update approach and full update approach have to serialize and deserialize much fewer data compared to streaming as they do not need to send all states over the network. The biggest bottleneck for the partial update approach remains the execution of the ensemble Kalman filter.

5.6.4 Impact of Update Probability

To evaluate the impact of the accuracy of the surrogate model on latency of the approaches, we previously evaluated our approaches with a synthetic probability of updates (cf. Section 5.6.2). To this end, we recorded the median latency for the different approaches for multiple simulation runs with different update probability. We assumed that updates are uniformly distributed. All other parameters are as our default configuration introduced in Section 5.6.1.

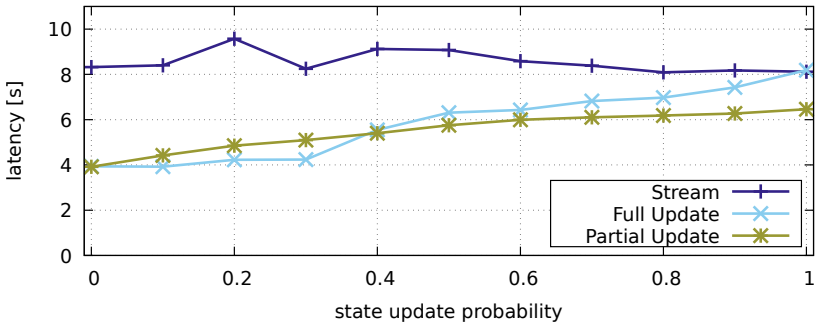

Figure 5.8: Latency over update probability.

Figure 5.8 depicts latency over update probability of simulation states. The latency of the stream approach is practically constant, since it sends all states of the simulation over the network. The full update approach is up to 2.1 times

faster than the stream approach for no updates but converges to the latency of the stream approach when all states require updating. Performance of the partial update approach only changes gradually, since more updates only marginally change the communication overhead. For less than $40\,\%$ of state update probability, the full update approach has a better performance than the partial update approach. However, if all states require updating, the partial update approach has a speedup of $26\,\%$ compared to streaming and full update approach.

5.6.5 Impact of the Size of Partial Updates

In addition to varying update probability, we also considered different sizes of partial updates. To this end, we run the approaches and introduced fake updates. Each state had a probability of 0.5 to require an update. Each update had a fixed size of violation points that require updating by partial updates. All other parameters are as our default configuration in Section 5.6.1.

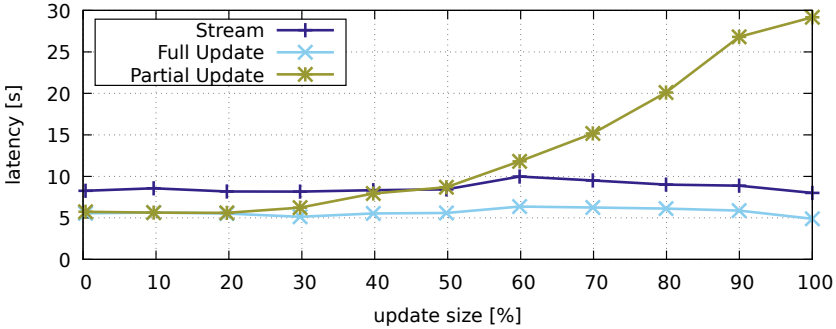

Figure 5.9: Latency over update size.

Figure 5.9 depicts latency over update size. As the size of the partial update does not effect the stream approach and full update approach, only the latency of the partial update approach gradually increases. Latency is even higher than for the stream approach, since sending partial updates requires decoding of the position of the updated points. This makes a partial update with all points

updated bigger than a full update. Additionally, the overhead for calculation of the ensemble Kalman filter is increased for more updates. For more than 50 % updates, streaming is more efficient than the partial update approach. However, for up to 20 %, the partial update approach provides results in the same time as the full update approach. As shown in Section 5.6.2, the percentage of violation points is typically below 20 %.

5.6.6 Impact of the Surrogate Problem Size

Lastly, we want to measure the impact of different surrogate problem sizes. If the surrogate problem grows, the stream approach has to communicate more data. However, for the full update approach and the partial update approach, also the computational overhead is increased. We want to measure the impact for different space discretization of the surrogate model. All parameters are taken as described in Section 5.6.1.

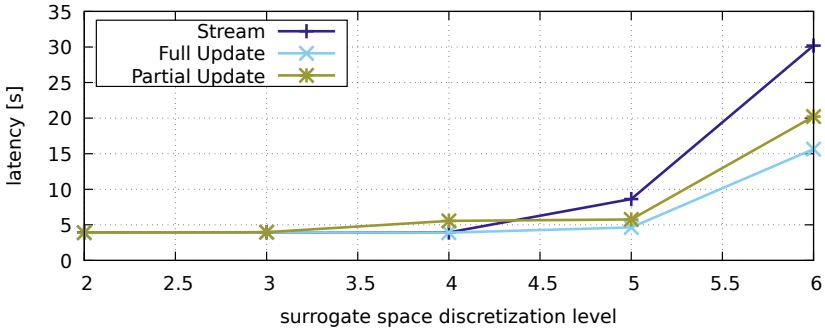

Figure 5.10: Latency over surrogate model space discretization level.

Figure 5.10 shows the latency of the approaches over different discretization level. Notice that for level 6, the reference model has the same space discretization as the surrogate model. However, the surrogate model is implemented as implicit method, so the two models provide different results. As for increased discretization level the number of unknowns grow exponentially, latency of the

approaches grow linearly with increased number of unknowns. However, the full update approach and partial update approach provide much better results for high discretization levels. In particular, the full update approach provides a speedup of up to 1.9, while the partial update approach is only 33 % faster than the streaming approach. Notice that the mobile device is limited to at most discretization level 6 due to memory limitations.

Concluding the evaluations, full update approach and partial update approach are significantly better than streaming. The full update approach is best for high data rates, high update size, low update probability, and high surrogate problem size. The partial update approach is best in cases of low data rates, low update size, and high update probability. Approaches can be combined by simply using the partial update approach for updates lower than 20 % and otherwise send full updates to benefit from both update types.

5.7 Related Work

This section discusses related work for reducing the latency of mobile simulations. While high performance computing (HPC) and code offloading were already discussed in context of robust execution in Section 4.9, this section discusses methods for increasing the performance for these concepts. Additionally, anytime computing and recent work in quality-aware execution for mobile applications are discussed.

5.7.1 High Performance Computing

Traditionally, numerical simulations are executed as high performance computing (HPC) applications on supercomputers [RD15]. Supercomputers consist of many closely distributed nodes connected via low latency and high data rate networks. This enables to run parallel programs on the same data using protocols such as, e.g., the message passing interface (MPI).

Surrogate modelling is not new and is actively researched and applied, especially for optimization, prototyping, or sensitivity analysis. For instance, Gorissen

et al. describe the Matlab SUrrogate MOdelling (SUMO) toolbox to generate surrogate models based on data and applications [GCD+10]. The toolbox uses different models, e.g, artificial neural networks, splines, or support vector machines, to generate problem-specific surrogate models.

In the simulation community, approaches exist that use profiling in order to make better decisions on the trade-off between accuracy and latency of the computation [LDBNR13]. However, these approaches do not provide a distribution between nodes.

Limitations of High Performance Computing

As discussed in Section 4.9, traditional methods from HPC are not suitable as they assume low delay and high data rates between computation nodes. In contrast, wireless communication has high delay and low data rates.

While our approaches could use any surrogate model, providing quality bounds for surrogate models is very problem dependent. In contrast, our approaches compares the result of the surrogate model with a reference model and can guarantee maximum errors between reference and surrogate model.

Making good trade-offs between accuracy and latency of the calculation is important for the surrogate model. Therefore, approaches using profiling to provide good surrogate models are orthogonal to our approach, while they do not provide any mechanism for distribution of the computation.

5.7.2 Code Offloading

Besides aspects discussed in Section 4.9, some code offloading methods were developed to increase the performance of mobile applications [RSM+11, CIM+11, GJM+12]. The general idea is to partition the mobile application and assign weights for offloading to partitions.

Limitations of Code Offloading

Code offloading is application agnostic and does not consider quality. Therefore, code offloading can only offload either an implementation of the surrogate model or the reference model. However, offloading the surrogate model does not provide quality constraints for the user and offloading the reference model results in higher overhead on communication and computation on the mobile device.

5.7.3 Anytime Computing

Another concept that can be used for quality-aware execution on mobile devices is anytime computing [Zil96,SN01]. In anytime computing, algorithms are required to provide an approximate result when interrupted at any time. The idea is to use an iterative algorithm converging to the correct solution. If the algorithm is stopped before calculation of the correct solution, the current value is provided as approximation.

Limitations of Anytime Computing

While there are anytime algorithms for numerical simulations, e.g., multi-grid methods or iterative solvers of algebraic equations, calculation of the reference model would be required on the mobile device. Therefore, using anytime algorithms would have a much higher computational overhead on the mobile device compared to our distributed approach.

5.7.4 Quality-Aware Execution of Mobile Applications

Recently, Pandey et al. proposed a framework for approximate computing on mobile devices and connected servers [PP17]. Their framework uses a workflow-based representation of computation states that yield approximate results. During runtime, their framework selects the quality in real-time and decides on offloading.

Limitations of Existing Quality-Aware Approaches

In contrast to the approaches presented in this chapter, their framework does not utilize state based computation of simulations or parallel processing of a mobile surrogate model on the server. To this end, in contrast to our approaches, all data has to be communicated, or our approaches using the surrogate model would need to be manually modelled as workflow.

5.8 Summary

In this chapter, we presented methods utilizing surrogate models to increase the performance of mobile simulations. The goal was to provide fast results with guaranteed quality of the simulation result. To this end, our approaches compute the simulation in a user-defined reference quality on the server. On the mobile device, a surrogate model providing lower quality at much fewer computation time will be executed.

We presented three approaches. The first approach was to simply stream results to the mobile device in surrogate quality. The second approach was to compute the surrogate model on the mobile device and the server. The server detects whether quality constraints are not fulfilled and then sends a full state update as correction to the mobile device. In the third approach, we considered partial updates to reduce communication overhead. To combine the surrogate state with partial updates, we use tools from data assimilation, namely the ensemble Kalman filter. This is required to maintain mathematical properties of the simulation model.

Evaluations on our test bed based on a Raspberry Pi and a connected server showed that the approaches are able to provide fast simulation results, even in cases of low data rates. Compared to our streaming approach, our approaches increase the performance of the system by up to over 13 times. The performance depends on the actual data rate and the size and frequency of required updates.

Methods described in this chapter can be generalized to different simulations and numerical applications, since the ensemble Kalman filter provides a very

generic framework (see Section 2.3.5). Requirements for the choice of the surrogate model and the reference model for our approaches are (1) existence of a mapping from the reference model state to the surrogate model state, and (2) the implementation of the surrogate model has to be deterministic. The surrogate model can be very simple but fast to compute and might not respect all physical properties of the real world. However, the simpler the surrogate model, the more communication overhead is to be expected during execution.

As we have seen, surrogate models are a very powerful technique to enable simulations on resource constraint mobile devices. While we mainly used surrogate models that change the discretization of the numerical simulation, active research in mathematics provides an orthogonal approach to generate reduced models by using methods from model order reduction. One such methods is the reduced basis method, which we will use in the next chapter for even more efficient calculation on the mobile device.

6

Using Model Order Reduction for Efficient Solution of Stationary Problems

The previous two chapters focused on concepts for time-dependent simulations, whereas this chapter will introduce concepts for stationary simulation problems. While time-dependent problems require to compute a chain of time states, stationary problems require only one solution. Therefore, previously introduced concepts might not provide efficient computation in the mobile environment. In this chapter, we provide concepts using model order reduction techniques for stationary simulation problems. In particular, we will use the reduced basis method (RBM) for distributing the computation between server and mobile device.

As introduced in Section 2.2, RBM uses the original full simulation model to generate a reduced simulation model (cf. Figure 6.1). The full simulation model provides parameter-dependent snapshots that will be used in the reduced simulation model to provide fast approximation for a specific parameter. This way, approximate solutions for parameters only known at runtime can be provided with very low latency compared to the full simulation model.

The general idea of concepts developed in this chapter is to compute the full simulation model only on the server. The mobile device does all necessary steps to provide the parameter-dependent approximate solution from the reduced model. It will constantly check quality constraints defined by the user and might require refined reduced bases from the server. In particular, we provide methods

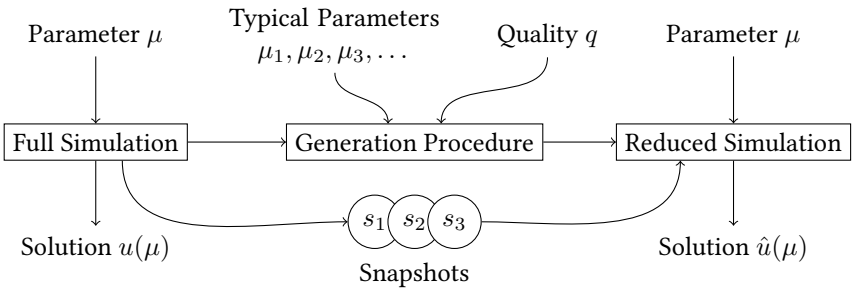

Figure 6.1: Overview over the reduced basis method (RBM) using solutions of the full simulation problem as snapshots in the reduced simulation problem.

for the following problems of mobile simulations: (1) We allow the user to define quality constraints based on training data and maximum error, (2) adapt the reduced model when quality constraints can not be met, (3) save energy for the execution of the reduced model by reducing the number of data required for the computation, and (4) provide methods for alternative basis generation algorithms to provide better performance on the mobile device during runtime.

Parts of this chapter have been published in [DSD+17, DDR17, DHS+18].

6.1 System Model

Before introducing the problem statement and concepts for utilizing RBM, this section describes our assumptions on hardware and software components, as well as the interfaces between the components in our mobile simulation middleware. Figure 6.2 depicts an overview of the system components and interfaces.

6.1.1 System Components

The system consists of two compute nodes, namely the mobile device and the server. Both nodes are connected via a wireless communication channel. Furthermore, the system consists of two software components provided by the

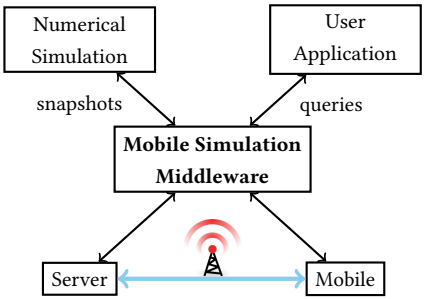

Figure 6.2: System Model

application programmers, the numerical simulation and the user application, and the middleware, which defines the distribution of computations.

The mobile device is the augmented reality headset carried by the user. Energy consumption on the mobile device is critical as it is battery-powered. There are two distinct energy consumers on the mobile device, the processor and the communication module.

In contrast to mobile devices, the server provides fast execution. It can be scaled-up by using specialized hardware, such as GPUs for efficient computation of numerical codes, or scaled-out by adding more servers in a cloud infrastructure.

For the wireless communication channel between mobile device and server, we assume data rates of multiple Mbit/s, as provided by state of the art wireless communication technologies like IEEE 802.11 (WiFi) or 4G cellular networks.

The numerical simulation is implemented by the simulation expert. The simulation problem is implemented as a separable matrix $A(\mu)$ and a separable vector $f(\mu)$ representing the simulation problem $A(\mu)u(\mu) = f(\mu)$ as described in Section 2.2. Parameters of the simulation model are represented by a vector μ. Additionally, the simulation expert has to define the quality requirements of the application. The quality has to be specified by two parameters. The first parameter, say $\mathcal{D}$, is the discretization of the full problem. The second, say r_{max}, is the maximum residual value, which is an indicator for the error introduced by the RBM.

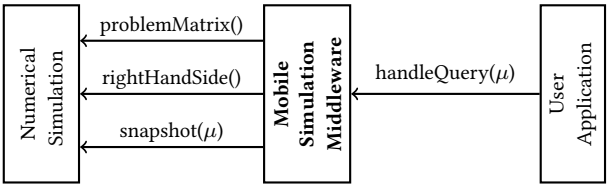

Figure 6.3: Interfaces for the mobile simulation middleware.

The user application is implemented by the application programmer. It sends queries to the middleware. Queries contain a parameter vector μ, which encodes sensor data or user input. When the query is answered, the user application visualizes the simulation results on the augmented reality headset.

The mobile simulation middleware connects the components. It executes code on the server and on the mobile device. Intuitively, the reduced basis method will be used to answer queries with low latency on the mobile device, and the compute-intensive pre-computation of the reduced basis will be performed on the server.

6.1.2 Interfaces

The numerical simulation and the user application provide interfaces for the mobile simulation middleware. Figure 6.3 shows an overview of all interfaces. There are three methods of the numerical simulation to be called by the middleware and one method called by the user application.

The numerical simulation has to implement three interfaces providing the problem matrix, the right-hand side, and solutions to the simulation problem. The problem matrix and the right-hand side has to be provided in parameter separable form. This call is only depending on the quality parameter $\mathcal{D}$. The interface to provide solutions of the simulation problem, called *snapshot*, provides $u(\mu)$ as the solution of the problem $A(\mu)u(\mu) = f(\mu)$ depending on the parameter. Notice that the implementation of the interface to provide snapshots is optional. The mobile simulation middleware could also use a generic algorithm to solve this

problem. However, the simulation expert usually knows which solver should be used to solve the simulation problem efficiently.

The user application sends queries to the mobile simulation middleware. Queries contain the parameter μ. The middleware will return an approximate solution which fulfills the quality requirements given by the simulation expert.

6.2 Problem Statement

Before introducing approaches and concepts, we first provide the problem statement for the execution of mobile simulations using the RBM. The major objective of the system is to reduce two quantities: energy consumption and execution time. At the same time, the system has to ensure to provide user-given quality constraints on the approximate solution that is returned to the user application.

We assume that the user provides a maximum residual $r_{\max}$ as quality constraint to the approximate solution provided to the user application. The optimization goal is then to reduce latency of the computation T and energy consumption on the mobile device E. Mathematically, the goal is to optimize the system for the following optimization problem:

$$\min \quad T + E \tag{6.1}$$

$$\text{s.t.} \quad \|A(\mu)\hat{u}(\mu) - A(\mu)u(\mu)\| < r_{\max}. \tag{6.2}$$

Notice that measuring T and E heavily depends on the method for the distribution between mobile device and server. Reducing latency of the computation usually also reduces the energy consumption of the approach as the device requires energy even while idle. Also notice that in this equation, the parameter μ is unknown before runtime and the solution $u(\mu)$ might never be computed, but the RBM provides a framework to check the quality constraints. Furthermore, the residual depends on the actual problem matrix. To this end, the residual provided has to be chosen carefully by the simulation expert knowing the constraints and requirements of the simulation result for the user applications.

To provide a low latency and low energy solution, the following sections will provide different methods with different distributions for solving the simulation task using the reduced basis method. We will start with the basic approach that requires a-priori knowledge of the parameters to provide the quality constraint of the formal problem.

6.3 Basic Approach

In the following, we present our approaches for the efficient execution of mobile simulations using the Reduced Basis Method. We first present a basic approach in this section, which is then further extended to improve adaptability and energy efficiency in the following sections.

The basic approach for processing queries with different parameters on the mobile device consists of four steps: (1) generation of the reduced basis on the server; (2) communication of the reduced basis from the server to the mobile device where the reduced basis is stored on the internal storage; (3) loading the reduced basis from the internal storage of the mobile device; (4) processing queries on the mobile device using the reduced basis.

The generation of the basis is executed on the server. To this end, the mobile device sends a request to the server which contains all information needed for the basis generation process. This includes the training set and the minimum quality as maximum residual threshold, which depends on the application. The training set can be given by the domain expert or, in applications where sensor values are read, the mobile device can first collect some sensor data, statistically obtain the distribution of the parameter μ, and then use this distribution to create the training set for the reduced basis.

Once the basis has been generated on the server, it is sent to the mobile device. The mobile device stores the basis on internal storage. Notice that the pre-computation of the reduced basis can take multiple minutes, depending on the numerical simulation code, the training set, and the number of snapshots needed to achieve the quality as specified. However, this step is only needed once

for initialization and should not be performed when latency-sensitive queries need to be processed.

Data	Size
Snapshots	$n \cdot d$
Reduced Problem Matrix	$S_A \cdot n^2$
Reduced Right Hand Side	$S_f \cdot n$
Residual Computation Matrices	$S_A^2 n^2 + 2 S_A S_f n + S_f^2$

Figure 6.4: Size of the reduced basis in floating point numbers.

Figure 6.4 lists the size of the data communicated and stored on the mobile device. The size of the data depends on the number of snapshots n, the number of discretization points of the full problem d, and the number of summands in the separation of the problem matrix S_A and the right-hand side S_f. The number of discretization points d, which depends on $\mathcal{D}$, is by far the largest part, typically multiple thousand floating point numbers (in our evaluation up to 65536 with $\mathcal{D} = 256$). The number of snapshots depends on the residual and is typically below 30 in our experiments. Numbers S_A and S_f are constant for a given problem. In our evaluation these values were 4 and 1.

After the basis is stored in a file on the mobile device, this file needs to be read by the middleware on the mobile device. As the file size for the reduced basis can grow rapidly, reading the data from the file can take up to seconds. However, this step is needed only once and can be performed when the user starts the user application, long before the first query will be received by the middleware. The basis can then be stored in memory for processing of multiple queries.

Processing a query is then straightforward as described in Section 2.2.3. First, we need to assemble the reduced system and then compute the solution of the reduced problem. After that, we need to multiply the solution with the snapshots to get an approximate solution of the full problem. In addition to the approximate solution, we also calculate the residual of this approximation and provide this information to the application. Notice that fulfilling the quality constraints for queries with parameters outside the range of the training set cannot be guaran-

teed using this approach. However, it is known that the quality of the result does depend on the region of the parameters rather than the density or specific choice of parameters in the training set [Haa16]. Therefore, for queries with parameters inside the range of the training set, the resulting approximation should have high quality. Furthermore, for many practical problems, the parameter region is known a priori by physical constraints. For example, if one parameter is the heat conductivity of some material, the application can request the reduced basis in the range of all materials to be used for the specific purpose, e.g., all exhaust tubes ever used by the company.

The basic approach has several drawbacks. First, the parameter range needs to be known before the basis generation process. If the parameter range changes, e.g., because the range of sensor values changes, the approach has to start from scratch. We therefore present an adaptive approach in the next section. Second, another problem is the latency and energy overhead introduced by reading the reduced file from internal storage of the mobile device. This is significantly improved using the subspace approach, which will be presented in Section 6.5.

6.3.1 Analysis of Communication Overhead

Using the Reduced Basis Method as described in the previous sub-section reduces the computational overhead on the mobile device significantly. However, it introduces communication overhead to send the basis from the server to the mobile device. Note that this overhead is less critical since the basis is stored on internal storage and can be re-used many times so the communication cost w.r.t. energy, amortizes over time. Moreover, the basis can be transferred while the mobile device is connected to a fixed power source while re-charging its battery. In this case, transferring the basis does not induce any energy overhead.

Still, we are interested in the size of the basis that needs to be transferred and stored on the mobile device. The data structure for a reduced basis contains(1) the snapshots stored in the matrix V; (2) the reduced problem matrix $A_V(\mu)$; (3) the reduced right hand side $f(\mu)$; (4) matrices for computation of the error, in particular the residual.

The snapshots are stored in matrix V. Each column of this matrix is a snapshot, which can be represented as D floating point numbers. Therefore, the size of the snapshots matrix is $n \cdot D$, where n is the number of snapshots in the reduced basis and D is the discretization of the full problem.

The reduced problem matrix $A_V(\mu)$ is a separable matrix. The separable matrix is represented as $A_V(\mu) = \theta_1 A_1 + \theta_2 A_2 + \cdots + \theta_{S_A} A_{S_A}$. The number of summands S_A depends on the actual problem. Each matrix A_i in the separation has $n \times n$ entries. The size for storing the separable matrix is therefore $S_A \cdot n^2$. Additionally, the functions θ_i need to be stored, which introduces a constant size linear in S_A.

The reduced right hand side $f(\mu)$ is also separable, but instead of a matrix, $f(\mu)$ is a vector. Analogously to $A_V(\mu)$, if $f(\mu)$ has S_f summands, it can be stored in $S_f \cdot n$ floating point numbers. In addition, the functions θ_i need to be stored, which introduces a small overhead. The theta functions can be stored as a string, as the only operation for the theta function is multiplication with other theta functions, which can then be realized as string concatenation.

The residual matrices are the result of the multiplication of separable matrices and are therefore again separable matrices (cf. Section 2.2.4). When multiplying two separable matrices, the number of summands of the result is the multiplication of the number of summands in the original matrices. Therefore, the size of Equation 2.8a in Section 2.2.4 is $S_a^2 n^2$. The size of the rest of the matrices can be derived analogously. The full residual matrices can be stored in $S_A^2 n^2 + 2 S_A S_f n + S_f^2$ floating point numbers.

Therefore, we can store the data for the reduced basis in

$$n^2(S_A^2 + S_A) + n(N + S_f + 2 S_a S_f) + S_f^2$$

floating point numbers. Thus, the size of the data will grow quadratically with the number of used snapshots. However, in general, S_A is very small in comparison to D. For example, for the diffusion-advection equation introduced in Section 2.2.4, S_A is 4, while D can be in ranges up to $1024 \times 1024 \approx 10^6$.

```
1:  function onQueryReceived(q)
2:      μ ← parameter of request q
3:      basis ← basis available on mobile
4:      if basis.residual(μ) ≤ max_res then
5:          return approximate solution using basis
6:      end if
7:      send μ to server; receive basis update
8:      apply basis update to basis
9:      return approximate solution using basis
10: end function
```

Figure 6.5: Pseudocode for the adaptive approach.

6.4 Adaptive Approach

If the parameter range and distribution are not known a priori, the basic approach might not be able to fulfill the constraint on quality for all queries. We therefore introduce an adaptive approach next that refines the basis during runtime. This approach is more flexible and also suitable for harder simulation problems, i.e., problems that need more snapshots to fulfill the user requirements.

6.4.1 Overview

The adaptive approach builds upon the basic approach. Similar to the basic approach, some initial reduced basis is made available on the mobile device as described in the previous section. However, in contrast to the basic approach, when a new query q arrives, the adaptive approach first computes the residual of the approximate solution provided by the RBM. If the residual fulfills the quality requirements of the application, the query will be answered with the approximate solution. If the residual does not fulfill the requirements of the application, the mobile device will request an update of the reduced basis from the server. Once the mobile device receives the update, it can again compute the approximate solution, which will—as a property of the RBM—return the exact solution of the full problem. Figure 6.5 depicts the pseudocode of the adaptive approach.

In the following, we will describe the parts of the approach, including the computation of the error indicator and content of the server request, and the processing of the update on the server.

6.4.2 Error Indicator and Server Requests

In addition to the basic approach, for handling query q, the mobile device has to compute error indicators for the approximate solution provided by the RBM. This error indicator represents the quality of the approximate solution. One very generic error indicator is the residual. The computation of the residual can be implemented very efficiently by exploiting the parameter separability (cf. Section 2.2.4).

Once the mobile device has computed the error indicator, it can check whether the quality bounds of the user can be met. If the result is insufficient, the mobile device will request a basis update from the server. This basis update contains the parameter μ of the query and the identifier of the reduced basis which is currently used. As an identifier, the parameters of the snapshots and the discretization of the underlying numerical simulation can be used.

6.4.3 Computation on the Server

When an update request with parameter μ and an identifier of the reduced basis is received by the server, the server first loads the properties of the reduced basis. It then computes a solution of the full problem with parameter μ and the discretization settings of the reduced basis. After computation of the full solution, this solution is orthogonalized to other basis vectors and is normalized to obtain more robust numerical systems. The server then computes the updated separable problem matrix and the separable right-hand side (cf. Section 6.3). Last, the updated residual matrices are computed. All of these operations require high-dimensional and costly operations. However, the most time consuming operation is the computation of the full solution on the server. Therefore, there is only little overhead compared to a pure offloading approach, where only the

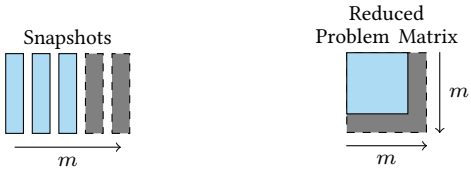

Figure 6.6: Subspace approach chooses m of n available snapshots in the order given during the basis generation approach and changes the reduced problem matrix, the reduced right-hand side, and the residual matrices (last two not depicted).

full problem solution is computed on the server.

6.4.4 Basis Updates

Once the server has computed the update of the reduced basis available on the mobile device, it sends the update back. The update includes a snapshot and updates for the separable matrices. Most entries of the matrices can be re-used, and the update does only contain one column and one row vector of the matrices. Nevertheless, the size of the update grows linearly with the number of snapshots included in the reduced basis. However, for a small number of snapshots, the dominant part is still the snapshot of the full problem. Therefore, the overhead to only communicating the full problem result is very small (for instance only $1.13\,\%$ for a 2D problem with 256^2 points, $S_A = 4, S_f = 1$, and 20 snapshots).

6.5 Subspace Approach

In our analysis of the basic approach, we found that reading the snapshots from internal storage is the major energy-consuming part. We therefore present in this and the following section approaches for reducing the number of snapshots needed for query processing on the mobile device. In this section, we present the *subspace approach*, which limits the computation of the problem to a subspace of the vector space spanned by all snapshots.

The reduced basis is generated such that it fulfills quality requirements for all parameters in the training set. However, for one specific parameter μ, it might be sufficient to compute on fewer snapshots. In the subspace approach, we therefore limit the computation to the first m snapshots in the order given by the reduced basis. Therefore, if n snapshots $s_1, \ldots, s_n$ are given, we want to find $m \leq n$ such that the quality constraint is still fulfilled and compute an approximation only on $s_1, \ldots, s_m$ (cf. Figure 6.6). This saves us from reading $n - m$ snapshots while still fulfilling the quality requirements of the user.

The subspace approach is divided into two problems. First, we explain how we can reuse the data structure of the matrix for computation on a subspace. Second, we explain how we can find the snapshot given the quality constraint by the user. Last, we shortly discuss how this approach can be combined with the adaptive approach.

6.5.1 Computation on Subspaces

For computing a solution on the reduced basis spanned only by the snapshots $s_1, \ldots s_m$, we can reuse the existing data structure. We can compute on sub-matrices which are created when trimming rows from the right and columns from the bottom.

For the reduced problem matrix, we just need the first m rows and the first m columns. Similarly, we only need the first m entries of the reduced right-hand side. The residual matrices can be trimmed similarly. Notice that the right-hand side and the reduced problem matrix are separable matrices. Trimming the separable form of the matrices therefore includes trimming multiple matrices.

The reuse of the data structure is essential at this point. Re-computation of the reduced problem matrix would otherwise involve the high dimensional problem matrix $A(\mu)$ and the snapshots. Using the sub-matrices, neither the problem matrix, nor the snapshots are needed for computation of the residual for the subspaces.

6.5.2 Subspace Selection

Now that we know how to compute a solution of the reduced problem by computation on the sub-matrices, we want to find m, such that computing on snapshots $s_1, \ldots, s_m$ gives us a solution that fulfills the quality requirements of the user. We call the subspace spanned by the first m snapshots $S(m)$.

In order to find m for $S(m)$, we use a linear search. When a query arrives with parameter μ, we first load the reduced problem matrix and the residual matrices into memory. We then loop, starting with $m = n$, compute the subspace $S(m)$, and compute the residual for parameter μ on $S(m)$, until we find the lowest m such that $S(m)$ fulfills the quality requirements. Once this m is known, we load the m snapshots from the file into memory and reconstruct the reduced solution in the full problem space.

The linear search could also be bottom-up starting with one snapshot or could be replaced by a bisection approach. However, this would result in longer search time when the number of snapshots needed is high.

The subspace approach can also be used with the adaptive approach. If the quality check for $m = n$ fails, the mobile device can request a basis update from the server. We then have a three-level storage model, where snapshots are either stored in-memory, on internal storage, or on the server.

6.6 Reorder Approach

For the subspace approach, the order of the snapshots is fixed. This might lead to suboptimal solutions, e.g., when the query has the same parameter as the last snapshot. In this example, the subspace approach needs to choose all snapshots. If we would reorder the snapshots according to the importance of the snapshots, then the snapshot with the same parameter would be the first and the subspace with only the first snapshot would already be sufficient. This motivates our reorder approach, which we introduce in this section as a preceding step to the subspace approach. The reorder approach operates on pre-computed data in order to allow the subspace approach to reduce the number of snapshots needed

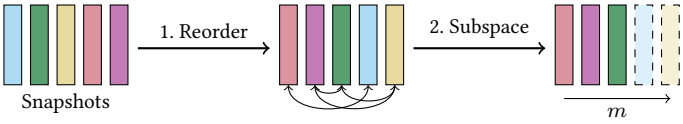

Figure 6.7: The reorder approach permutes the snapshots before choosing subspace with m snapshots depending on parameter μ.

for the computation.

Figure 6.7 depicts the idea. First snapshots are reordered. Then a subspace is chosen using the previously introduced subspace approach. The reordering depends on the query parameter μ. As we will show, finding a reordering can be implemented on the pre-computed data such that it can be executed efficiently and fast on the mobile device.

There are two problems for reordering snapshots: (1) find a suitable reordering for parameter μ, and (2) perform the reordering by re-using pre-computed data. To improve latency and energy cost, we need to find a good order of the $n!$ possible orders in the first step and solve the second problem by only using pre-computed data and not require any high-dimensional operations of the numerical simulation.

6.6.1 Finding an Order

Goal for reordering the snapshots is to allow the subspace approach in a subsequent step to reduce the number of required snapshots. Therefore, the best order of snapshots would be in decreasing order of their importance, i.e., the next snapshot provides the minimum error among all snapshots that are remaining. This way, the subspace approach with m snapshots will have the minimum computable error bounds for any m snapshots. Notice that the reordering is executed during runtime and depends on parameter μ.

Our reordering mechanism is based on the following formal observation that holds for normalized snapshots. If u is the reduced solution for the subspace

with m snapshots and $\tilde{u}$ is the reduced solution for the subspace with $m-1$ snapshots, the difference in the approximate solutions after the reconstruction is

$$\left\| \sum_{i=1}^{m} \mathbf{u}^{(i)} \mathbf{s}_i - \sum_{i=1}^{m-1} \tilde{\mathbf{u}}^{(i)} \mathbf{s}_i \right\| \leq \left| u^{(m)} \right| + \left\| \sum_{i=1}^{m-1} \mathbf{s}_i \left(\mathbf{u}^{(i)} - \tilde{\mathbf{u}}^{(i)} \right) \right\|, \tag{6.3}$$

where $u^{(i)}$ is the i-th entry in vector u and we assume that snapshots are normalized. This motivates to move the snapshots with lowest absolute coefficient to the end. We therefore order the snapshots according to the absolute value of their reduced solution in descending order. Notice that this step does not need the reconstruction of the approximation and therefore no snapshots need to be loaded into memory. Additionally, normalizing the snapshots increases numerical stability [Haa16].

1: **function** FINDREORDER(μ)
2: $\left(u_V^{(i)}(\mu)\right)_{i=1}^{n} \leftarrow$ coefficients of the reduced solution for parameter μ
3: $t \leftarrow \left\{ \left(|u_V^{(i)}(\mu)|, i \right) \right\}_{i=1}^{n}$
4: sort t using the first element of the tuples
5: **return** order of snapshots as second elements in t
6: **end function**

Figure 6.8: Finding reordering for normal bases

Figure 6.8 depicts the pseudo code for finding the reordering. We need the precomputed reduced problem in memory, which consists of the reduced problem matrix A_V and the reduced right-hand side f_V. These are separable matrices which do depend on the snapshot matrix V (cf. Section 2.2.3). We first compute the coefficients as solution $u_V(\mu)$ of the reduced problem $A_V(\mu)u_V(\mu) = f_V(\mu)$ and sort them according to their absolute value. The reordering is represented as a list, where the j-th position has value i when the j-th snapshot should be moved to position i.

6.6.2 Reordering Precomputed Data

The reordering can be represented as a permutation matrix P. Reordering can then be executed by multiplying the existing snapshot matrix V with the permutation matrix P. Using this approach, we can reuse the pre-computed data, such as the separable reduced problem matrix, the separable right-hand side.

The reordering can be applied to the separable reduced problem matrix by permuting rows and columns. We use the pre-computed data for the snapshot matrix V and show that we can reorder this data in order to obtain the pre-computed data for matrix $V' = VP$, where P is the permutation matrix of our reordering. The available pre-computed data is $A_V(\mu) := V^T A(\mu) V$. We can reuse this data for the reordered snapshots V', since $A_{V'}(\mu) = (VP)^T A(\mu)(VP) = P^T V^T A(\mu) VP = P^T A_V(\mu) P$. To obtain the reduced problem matrix, we only need to multiply with permutation matrix P^T from left, which is a permutation of the columns, and permutation matrix P from right, which is a permutation of the rows.

The separable right-hand side and the residual matrices can be reordered by ordering of the entries in the vectors analogously to the reduced problem matrix.

Notice, that the permutation of the matrix can be implemented much faster than multiplication with the permutation matrix by reordering rows and columns in the underlying data structure of matrices and vectors. However, we used the notation for multiplication to show the correctness of the approach.

To improve numerical stability, snapshots can be orthonormalized for the previous approaches. However, in this approach, it is beneficial to only normalize the snapshots. Orthogonalization, e.g. by using Gram-Schmidt, reinforces the order and reduces the flexibility of the reorder approach.

6.7 Reorder Basis Generation

In order to optimize the number of snapshots to be used in the reorder approach, we next present a new approach for basis generation. This approach takes into account the online computation using the reorder approach and is therefore

```
1: function ReorderResidual(basis, l, μ)
2:     r ← FindReorder(basis, μ)
3:     b ← ApplyReorder(basis, r)
4:     s ← CutBasis(b, l)                          ▷ As in subspace approach
5:     return s.residual(μ)
6: end function
7: function ReorderBasisGeneration(T, a, max_res)
8:     s ← solution of full problem for random μ ∈ T
9:     snapshots ← ∅
10:    basis ← reduced basis from snapshots
11:    while ∃μ ∈ T : ReorderResidual(basis, |snapshots| − a, μ) >
   max_res do
12:        μ* ← max(residuals).key
13:        s ← solution of the full problem for μ*
14:        snapshots ← snapshots ∪ {s/‖s‖}
15:        compute new basis from snapshots
16:    end while
17:    return basis
18: end function
```

Figure 6.9: Pseudocode for reorder basis generation for training set T, number of additional snapshots a, and maximum residual max_res.

able to generate better reduced bases on the server. To this end, we modify the previously introduced greedy basis generation (cf. Section 2.2) to allow for better performance for the reorder approach.

In contrast to the greedy basis generation approach, we define the m-residual as the residual after the execution of the reorder approach and choosing only m snapshots (see Figure 6.9). This way, we use the results as provided by our approach already during the construction of the reduced basis.

However, this approach for optimizing for a subset of the full solution space introduces some numerical problems as snapshots are not strictly linearly independent. We therefore use the Moore-Penrose pseudoinverse to compute the solution. The pseudoinverse provides a least square solution, which can even be used when snapshots are linearly dependent.

Figure 6.9 depicts the basis generation procedure for the reorder approach. In contrast to greedy basis generation, we consider the residual after the reorder computation. We omit a number of snapshots for the decision which parameter to use for the next basis refinement. The snapshots will be first sorted and then cut to simulate the reorder approach. However, the number of snapshots to be omitted should not be too large to avoid overfitting. Parameter a, which defines how many snapshots should be omitted, is problem dependent. In preliminary tests, we found good solutions with $a = 3$.

Our original approach was to cut the basis to a constant number of snapshots. However, this approach leads to overfitting to the training set and therefore to very large reduced bases that only provides good results for the training set but not for test queries. Having a constant offset to the greedy basis approach avoids overfitting and produces only slightly bigger bases which enables the reorder approach to have more choices when deciding on a reordering.

6.8 Evaluation

In this section, we present the evaluation of our five approaches using the Reduced Basis Method (RBM) with respect to energy efficiency and execution time. For comparison, we also implemented two simple solutions without the RBM, the *server-only* and the *mobile-only* approach. The server-only approach sends the combination of parameters to the server. The solution of the full simulation problem is computed on the server and sent back to the mobile. The mobile-only approach computes the full simulation problem on the mobile device.

In our evaluation, we consider different performance metrics. First, we evaluate the quality of reduced bases with different sizes and compare different basis generation methods. Then, we evaluate the runtime and energy consumption for two workloads, single queries and multiple queries.

6.8.1 Evaluation Setup

Before we present the evaluation results, we introduce our evaluation setup. The setup consists of two different mobile devices, a mobile network, and the setup for measuring the energy consumption. We also provide details about the used libraries in the implementation and the simulation problem used for the evaluation.

Two different devices were used for the runtime evaluation, a Samsung Note 4 (SM-N910F) and a Samsung Galaxy S7 (SM-G930F). Both devices use the Android platform (version 6.0) based on the Linux kernel (version 3.10). The results for both devices were very similar for most evaluations. Therefore, if not stated otherwise, results of the approaches presented in this section are taken using the Note 4.

For wireless communication, we used IEEE 802.11 (WiFi). Using `ping`, the measured latency between mobile device and server was between 1.4 ms and 6.7 ms with an average of 3.9 ms. `iPerf` measured bandwidths between mobile and server between 55 MBits/s and 74 MBits/s.

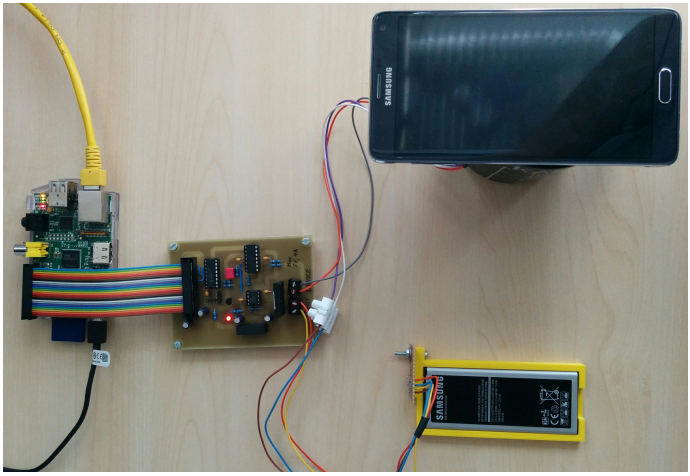

Figure 6.10: Equipment for measuring energy consumption.

For the energy measurements, we used a custom measurement board with analog-to-digital converter connected to a Raspberry Pi[1]. We designed a battery holder and battery replacement in order to perform energy measurements in situ. Figure 6.10 shows our energy evaluation setup. All energy measurement values in this section are absolute values taken from the Note 4 with fixed screen brightness. The power consumption of the device in idle mode was 0.4 W.

As simulation problem for the evaluation, we used the stationary diffusion-advection equation. This equation can be used to simulate the heat in an object as for the application for placing a hot tube as mentioned in the introduction. The equation has three parameters, one for the diffusion (μ_{diff}) and two for advection (μ_{advx} and μ_{advy}). For the implementation, we discretized the equation using finite differences. As numerics library, we used the Apache Commons Maths library (version 3.6), which is the most popular Java numerics library, and NumPy (1.13.0) and SciPy (0.19.1), which provide hardware optimization on the server. Additionally to the pure Java implementation, we implemented the mobile-only approach natively using the Android Native Development Kit (NDK) and the Eigen C++ library in version 3.2.8.

6.8.2 Basis Generation Methods

Next, we evaluate the performance of the basis generation methods and how many snapshots are needed during query processing on the mobile device.

In order to quantify the number of snapshots needed to reach a certain quality, we created three different training sets $A, B,$ and C. The training sets were chosen such that $A \subset B \subset C$, where parameters μ_{advx} and μ_{advy} spanned different parts of the parameter space expressing different behavior of the model ($[0, 40]^2$ for A, $[-40, 40] \times [0, 40]$ for B, $[-40, 40]^2$ for C). Parameter μ_{diff} was for all three training sets in $[10, 20]$. All intervals were discretized with step width 1.0.

To quantify the relation between the size of the reduced basis and quality, we used the three training sets, executed the greedy basis generation algorithm (cf. Section 2.2), and recorded the maximum residual of test sets. The test sets

[1] available at `https://github.com/duerrfk/rpi-powermeter`

A_t, B_t, C_t consist of 1000 random points in the range of A, B, C. The discretization of the full problem was 256×256, i.e. 256 points in x and y direction.

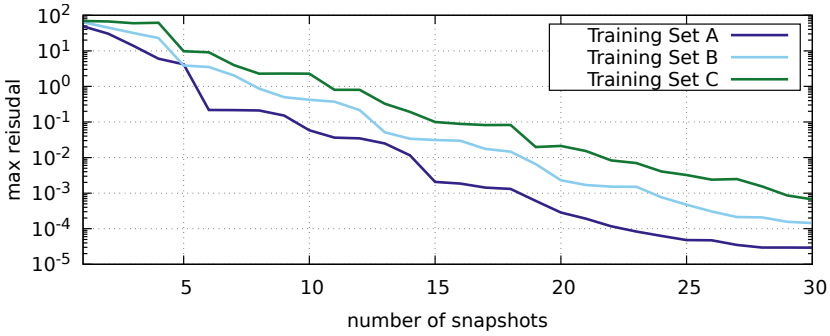

Figure 6.11: Quality of the RBM with different number of snapshots.

In Figure 6.11 depicts the relation between number of snapshots and quality for each training set. Training set A, which has smallest variation in the parameters, has the lowest maximum residual for a fixed number of snapshots. Notice that the residuum is measured in the Euclidean norm. To get a better impression, this norm is always bigger than the maximum absolute difference of any two points in the resulting vector. Therefore, a residual of, say 0.1, means that the result multiplied by the problem matrix results in a vector which differs in all 256^2 entries at most 0.1 from the right-hand side. Therefore, for many practical applications, a basis with 10, 15, or 20 snapshots would provide sufficient quality for this problem.

The number of snapshots required in the subspace approach to provide fixed quality constraints depends on the actual parameter. To evaluate the impact on the parameter we recorded how many snapshots are required for different parameters.

Figure 6.12 depicts the number of snapshots required for different parameter combinations for two parameters of a reduced basis with training set B and 24 snapshots. The total number of 24 snapshots is required only for a small corridor range between -10 and 10 for one of the parameters. Big parts of

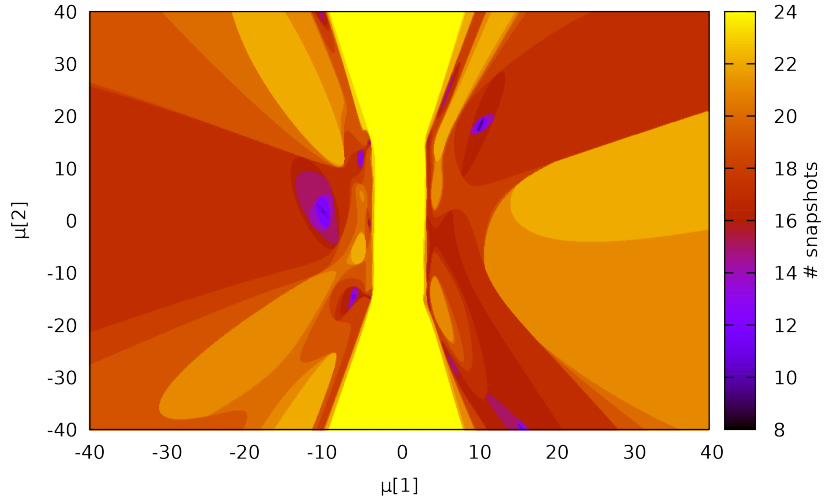

Figure 6.12: Number of snapshots needed in the subspace approach for different parameters.

the parameter space require only 80 % or less of the total number of snapshots. Notice that parameter areas that require only very little snapshots are near the snapshots chosen by the greedy basis generation algorithm. This way, knowing the distribution of parameters in the queries can be used for generating better reduced bases in these parameter regions.

After evaluating the number of snapshots required for the subspace approach for different parameters, we now quantify the number of snapshots needed during online computation for different basis generation methods. The number of snapshots needed for the subspace approach and the reorder approach is dynamically depending on parameter μ. We also want to quantify the fraction of snapshots typically needed for these approaches compared to the basic approach, which always uses all snapshots available in the reduced basis.

Figure 6.13 depicts the number of snapshots needed for the reorder and the subspace approach in multiple basis generation runs. It also compares the greedy basis generation approach to our reorder basis generation approach. Using the

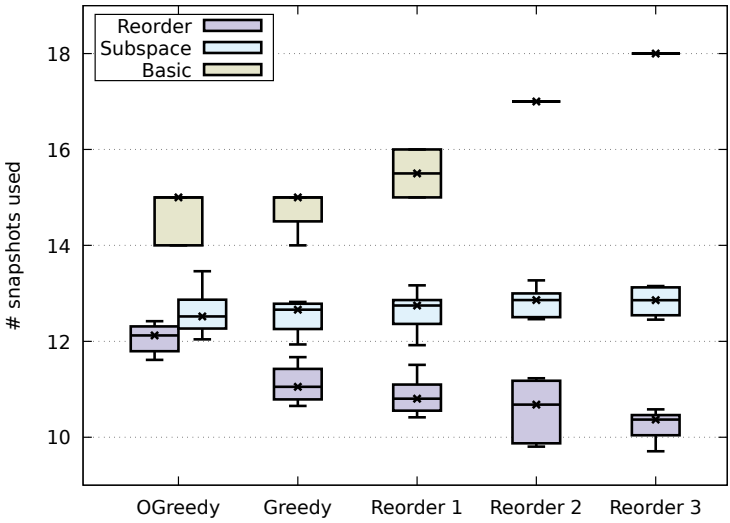

Figure 6.13: Mean number of snapshots needed in different approaches for different reduced basis generation methods. OGreedy is the orthogonal greedy basis. Reorder k is the reorder basis generation method with k additional snapshots.

reorder basis generation approach, the reorder approach performs better than using the greedy bases. In particular, for the reorder basis with 3 additional snapshots the median mean number of snapshots for the reorder approach is 69.1 % lower than for the basic approach using an orthonormal greedy basis. On an orthonormal basis, the median mean number of snapshots of the subspace approach is only 83.5 % compared to the basic approach. Notice that for the reorder basis generation, the bigger k is in the additional number of snapshots, the harder it is to generate the reduced basis because of numerical instabilities.

As these results suggest, we assume in the following that the subspace approach will only need 83.5 % of the snapshots and the reorder approach only 69.1% of the snapshots for the diffusion advection equation. Whenever we refer to the reorder approach, we assume that we use a reorder basis with 3 additional snapshots.

6.8.3 Runtime

We compare the runtime of simulation runs for the different approaches on different mobile devices for both, single queries and multiple queries. Runtime of the adaptive approach can be split into runtime for the local case, when the available reduced basis provides sufficient quality, and runtime for the remote case, when the reduced basis needs an update from the server. The subspace approach was evaluated using $83.5\,\%$ of the snapshots. For each skipped snapshot, it had to compute the residual and the subspace. For the reorder approach, we assume that a reorder basis was generated and that the approach only needs 69.1 % of the snapshots. We included all computations for the overhead. For the mobile-approach, we used two implementations, one using pure Java and one using the Android Native Development Kit (NDK) in C++. We repeated the measurements for different discretizations of the underlying full simulation problem and for different numbers of snapshots.

Figure 6.14 depicts the average runtime for the processing of single queries for different sizes of the full problem. The full problem discretization is equidistant on both axes of the 2D domain. Therefore, for instance, for discretization $\mathcal{D} = 32$, a matrix equation with a $32^2 \times 32^2$ matrix has to be solved. The used reduced bases had 15 snapshots. Most results from the Galaxy S7 were the same as for the Note 4. Only the performance of the mobile-only approach using the NDK was significantly better on the Galaxy S7. For simplicity, only the better results from the S7 are depicted. All other results depicted in Figure 6.14 are from the Note 4. Results from the local case of the adaptive approach are very similar to results of the basic approach. Therefore, only the remote case of the adaptive approach is depicted. The server-only approach is over 280 times faster than the mobile-only approach in pure Java. The basic approach is again over 5 times faster than the server-only approach. The subspace approach is over $51\,\%$ faster than the basic approach. The reorder approach is only 3 % faster than the subspace approach. This is partly caused by the random access on the data file, which can be read as one big bulk operation in the subspace approach.

As the improvement of the reorder approach until dimension $\mathcal{D} = 256$ is

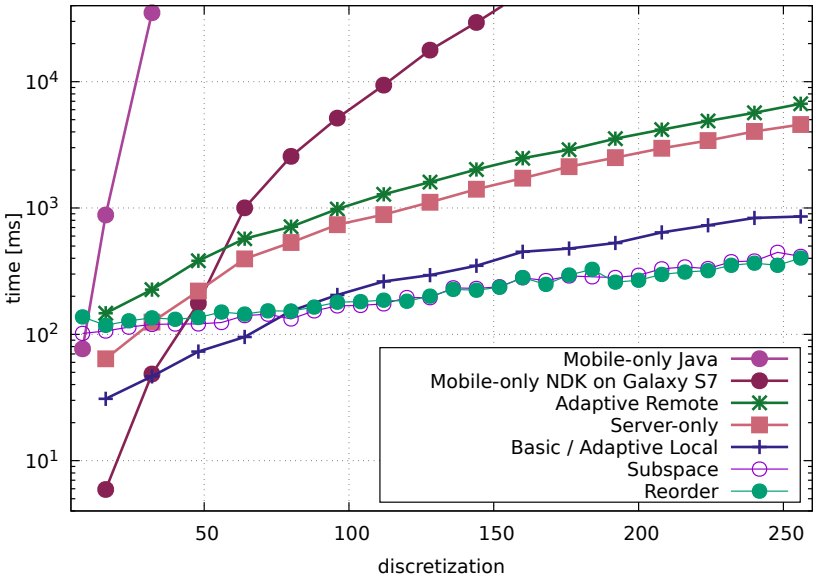

Figure 6.14: Runtime for varying problem discretizations.

not significant, we evaluated the subspace and the reorder approach for bigger reduced problems of dimension $\mathcal{D} = 512$ and $\mathcal{D} = 1024$. While, for the previous evaluation, snapshots had sizes of up to 500 KB, for dimension $\mathcal{D} = 1024$, one snapshot has 8 MB. Figure 6.15 depicts results of this evaluations, where the reorder approach is 10 % faster for dimension $\mathcal{D} = 512$ and 16.5 % faster for dimension 1024 comparing the median execution times.

As both implementations of the mobile-only approach perform very poorly, we compare our approaches in the following only with the server-only approach.

Next, we compare the runtime for processing single queries with varying number of snapshots in the reduced basis. Figure 6.16 depicts the results with full problem dimension $\mathcal{D} = 256$. As the server-only approach computes the full problem, it does not depend on the snapshot size. With growing number of snapshots, our approaches need more time. The speedup of the basic approach

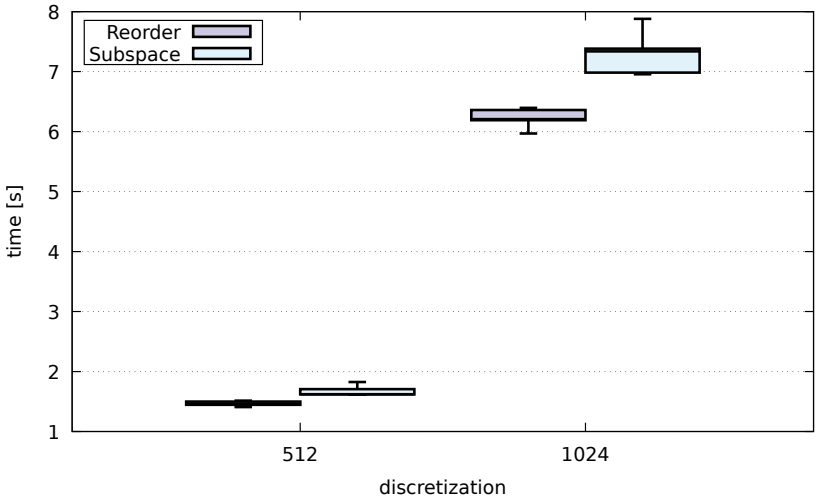

Figure 6.15: Runtime for very high dimensions of 512 and 1024 with 15 snapshots.

against the server-only approach is over 13.2 for 4 snapshots and decreases to 2.5 for 32 snapshots. However, with 64 snapshots, the speedup of our approaches against the server-only approach is still above 1.3 for the basic approach. The subspace approach is 45 % faster than the basic approach. As the reorder approach is only 2.8 % faster than the subspace approach, we see that the number of snapshots does not have too much impact on the performance of the reorder approach when snapshots are small.

Runtime for Multi-Query Applications

Many applications require multiple queries, e.g., to continuously visualize simulation results for augmented reality. For such applications, our basic and adaptive approach can be split into two parts: a setup and a query part. During setup part, snapshots are loaded from internal storage, whereas during the query part solution for a specific parameter is computed. Notice that in this case, the runtime of the query part is much more important, as it will be executed many times

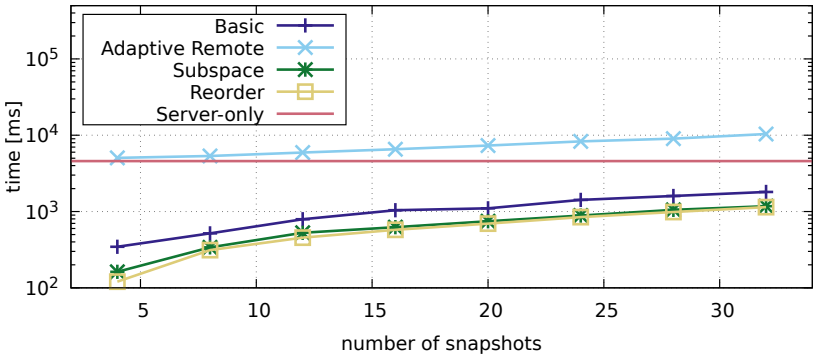

Figure 6.16: Runtime for single queries with varying snapshot number.

compared to the setup part, which will only be executed once.

Figure 6.17 depicts the runtime for setup phase and processing of queries for different sizes of the full problem with 15 snapshots. It shows that the runtime for one query is in general one order of magnitude higher than for the setup phase.

Figure 6.18 depicts the runtime for setup phase and processing of queries for different number of snapshots with discretization $\mathcal{D} = 256$. It also shows one order of magnitude lower runtime for answering queries than for the initial setup phase. Processing queries even with 32 snapshots and $\mathcal{D} = 256$ only takes 63 ms. Query processing using the basic approach on the mobile device is over 131 times faster than the server-only approach for $\mathcal{D} = 256$ and 15 snapshots.

6.8.4 Energy Consumption

Energy is a very important resource for battery-powered mobile devices. Therefore, we evaluated the energy consumption of all four approaches with varying full-problem discretization, as well as for varying number of snapshots.

Figure 6.19 depicts the energy consumption for varying discretizations of the full problem. The initial bases had 15 snapshots. Updates in the adaptive approach consume more energy than the server-only approach. The energy consumption of

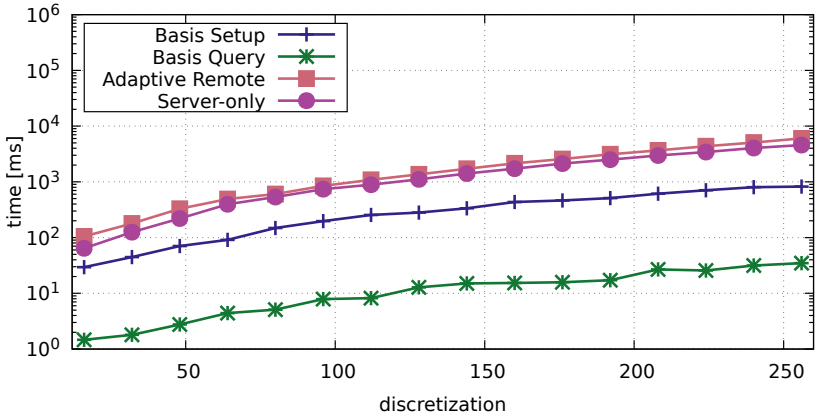

Figure 6.17: Runtime for multiple queries with varying full problem dimension.

the local case of the adaptive approach is very similar to the basic approach. Both only need 68 % of energy compared to the server-only approach. The subspace approach needs 34 % of the energy of the basic approach, while the reorder approach saves another 18 % of energy compared to the subspace approach. The mobile-only approaches, which are not depicted, consume significantly more energy. The NDK version consumed 3.9 J for discretization $\mathcal{D} = 32$ and over 84 J for $\mathcal{D} = 64$. The pure Java implementation consumes more than 80 J for $\mathcal{D} = 32$.

In addition to varying the discretization of the underlying full problem, we also evaluated the impact of the basis size on the energy consumption of our approaches. Figure 6.20 depicts the energy consumption for different numbers of snapshots with full problem size $D = 256$. As already seen for the runtime, also the energy consumption increases for higher number of snapshots. The server-only and the mobile-only approaches are not affected by the number of snapshots.

Our approaches can reduce the energy consumption for single queries significantly, especially when the discretization of the full problem is high and the number of snapshots needed is low. The adaptive approach consumes less energy than the server-only approach, if more than $8\,\%$ of the queries can be answered

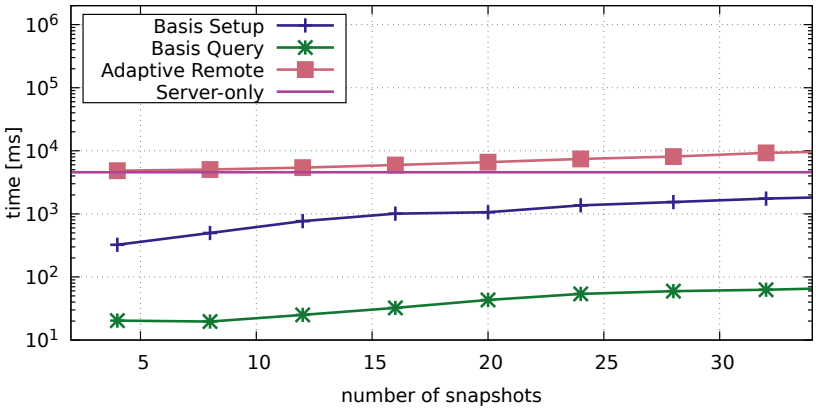

Figure 6.18: Runtime for multiple queries with varying snapshots number.

locally for $\mathcal{D} = 256$ and 5 snapshots. If the snapshot size is increased to 15, the adaptive approach is still beneficial when more than 58 % of the requests can be answered locally. The subspace approach saves over 32 % of energy compared to the basic approach with 20 snapshots. In addition, the reorder approach saves another 13 % compared to the subspace approach.

We also considered the energy consumption for multiple queries for the basic and the adaptive approach. Figure 6.21 depicts the power during the setup phase and the queries for a basis with 64 snapshots. Between the operations, the device was idle. Most energy is needed for reading the reduced basis from internal storage. After the basis is available in memory, processing one query only takes less than 0.17 J. For a basis with 15 snapshots and $\mathcal{D} = 256$, the median energy consumption for one query was 0.04 J. This is 73 times less energy as for the server-only approach.

Overall, the evaluation showed that our approaches significantly improve latency and energy consumption especially for processing queries after the reduced basis is available in memory. In this case, the basic approach achieves a speedup of over 131 compared to the server-only approach. At the same time, it reduces energy consumption by a factor of 73. For single queries, the subspace

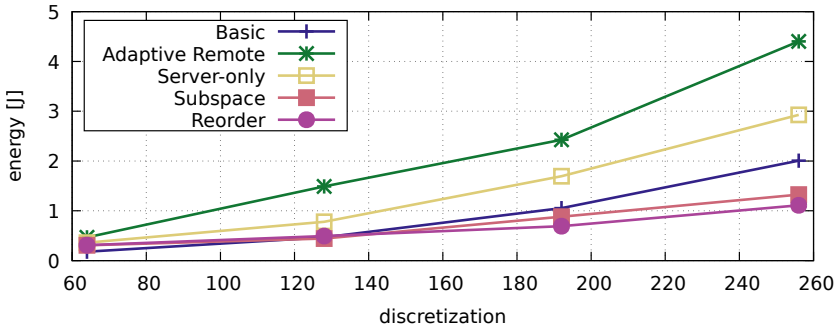

Figure 6.19: Energy consumption for different discretizations of the full problem.

approach reduces the energy consumption in the setup phase and therefore needs 34 % less energy than the basic approach. Additionally, our reorder approach is able to save again 18% of energy compared to the subspace approach and over 62 % of energy compared to the server-only approach.

6.9 Related Work

This section discusses related work for providing results of stationary problems on mobile devices. As seen in Section 4.9 and Section 5.7, concepts in high performance computing and code offloading can be used. In contrast to previously discussed related work (cf. Section 4.9), we will now focus on both, runtime reduction and energy efficiency. Additionally, previously discussed time-dependent problems required the solution of many algebraic equations, whereas stationary problems only require the solution of one algebraic problem.

6.9.1 High Performance Computing

As explained in Section 5.7, high performance computing (HPC) solutions cannot scope with our approaches to provide fast results on heterogeneous and wirelessly connected mobile devices. However, HPC concepts can be used for the server-side computation for generation of the reduced basis. Therefore, they might not

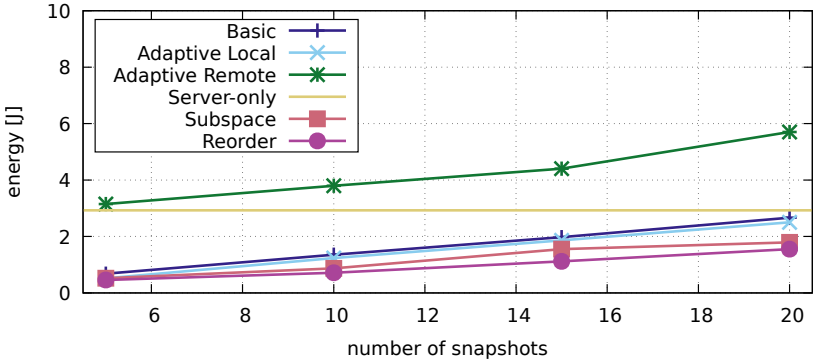

Figure 6.20: Energy consumption for different number of snapshots.

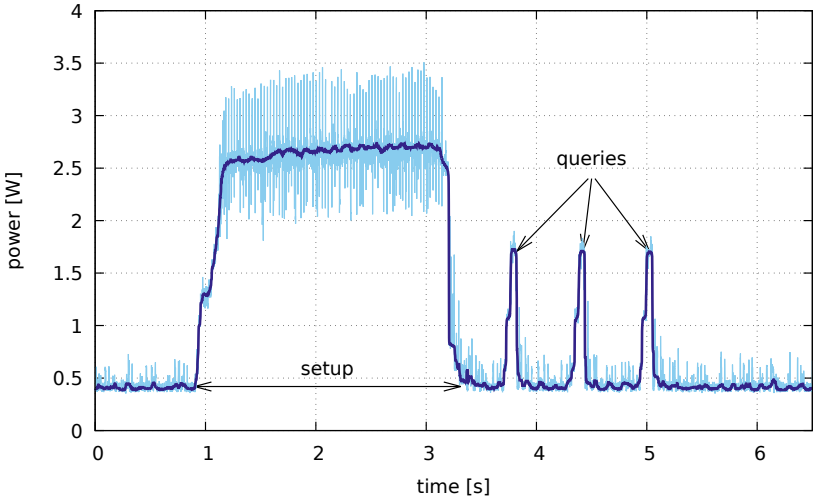

Figure 6.21: Energy consumption of the basic approach processing three queries after initial setup.

provide a distribution between mobile device and server, but can be used to scale the server to be used with the approaches presented in this thesis. However, this would have no impact on the energy resources required on the mobile devices.

6.9.2 Code Offloading

Code offloading has been already discussed in Section 4.9.2 and Section 5.7.2, where the main focus was on reducing the latency for the solution of time-dependent simulation problems. We shortly discuss limitations of code offloading for stationary problems and not only latency, but also energy consumption on the mobile device.

Some code-offloading works focus on reducing the energy consumption on the mobile device [CBC+10, CIM+11, KAH+12]. As for latency-optimized code-offloading, energy-optimized code-offloading partitions the application into two parts, where one part is executed on the mobile device and the other part is executed on the server. The partition is created by minimizing the energy consumption of the different parts of the application that will be executed on the mobile device. Additionally, the energy consumption for the communication between components that will run on the server and that will run on the mobile device has to be taken into account. To create the partition, Cuervo et al. use integer linear programming (ILP) and a profile of the different parts of the application.

Limitations of Code Offloading

As pointed out in Section 5.7.2, code offloading concepts are application agnostic. For stationary problems, this might lead only to the two solutions of either computing everything on the server and then communicating the solution the mobile device, or to compute everything on the mobile device itself. The reason for this is that all possible non-extreme partitions would result in more overhead as methods for solving algebraic equations require all parts to solve $A \cdot x = f$. Additionally, code offloading does not consider quality of the solution and

therefore does not reduce the original full simulation problem as we do by using the reduced basis method.

6.9.3 Model Order Reduction

We are not the first to execute the RBM on constrained computing devices. Huynh et al. already proposed to use the RBM for deployment of thin computing platforms [HKPP11]. In this approach, the precomputation of the reduced basis is executed prior to deployment, and the approximation using the reduced basis is executed after deployment.

Limitations of Existing MOR Approaches

In contrast to our approaches, Huynh et al. do not consider the networking capabilities of the devices. Therefore, their approach is restricted to one single reduced basis that cannot be changed after the deployment. Especially when the parameter region of queries changes over time, their approach is not able to provide approximate solution with quality constraints. Our approach, in contrast, is able to adapt to other parameter regions or quality constraints. Additionally, they did not optimize their approach for energy consumption or latency and do not provide advanced approaches such as the subspace approach or the reorder approach.

6.9.4 Generic Quality-Aware Approaches

Recently, Pandey et al. proposed a mobile distributed framework for quality-aware applications [PP16, PP17]. Their approach adapts the quality of computations in order to achieve better resource efficiency of pervasive mobile applications. To this end, they construct workflows that reduce the quality of the application and meet the requirements of the user.

Limitations of Generic Quality-Aware Approaches

The approaches by Pandey et al. do not provide well-defined quality of the result. To be useful, simulations require guarantees on the quality, e.g., by providing a maximum residual value. Additionally, as for code offloading, there are no real distributed concepts that can be used over a wireless link to solve the algebraic problems. Therefore, we argue that numerical simulations need a more specific method such as the reduced basis method together with specific quality metrics that are well-understood and can be defined by simulation experts.

6.9.5 Approximate Computing

The basic idea of approximate computing is to reduce the accuracy of calculations in favor of reduced energy consumption, runtime, less powerful hardware, etc. For instance, Xu et al. presented an approach in [XMK16] that reduces the refresh rate of memory to save energy, which at the same time increases the probability of bit errors. A second example is the IMPACT system by Gupta et al. [GMP+11] that implements an imprecise adder for low-power approximate computing.

Approximate computing has already been applied for scientific simulations [SBKW16, SBW17]. Such hardware-centric solutions are complementary to our approach and can be used to solve the reduced problem more energy efficient.

6.10 Summary

In this chapter, we presented a middleware for enabling complex numerical simulations on resource-constrained mobile devices by distributing the simulation between mobile device and a server infrastructure. Such a middleware is needed for interactive simulations in the field, e.g., an engineer using a head-mounted augmented reality device who wants to simulate the heat in an object to adjust its placement according to the surrounding materials. We presented four approaches for solving this problem using the Reduced Basis Method (RBM), which pre-computes a reduced representation of the simulation to reduce the evaluation time. The first approach was to pre-compute a reduced basis on the server and

send this basis to the mobile device. In order to calculate an approximation of the simulation, no further communication is necessary. The second approach was more interactive and utilized the fast error indicator of the RBM. Using this indicator, the mobile device can efficiently check whether the quality demands of the application are fulfilled. If the quality is not sufficient, the mobile device requests a basis update from the server. After such an update, the mobile device is able to answer queries with similar parameters completely autonomous without communication with the server. Goal of the third and fourth approach is to reduce the number of data to be read from internal storage, which we identified as major energy consumer. In addition, we also presented a novel approach for the pre-computation, which further reduces the data needed from internal storage during runtime.

We evaluated our approaches on real mobile devices in a real wireless network. We showed that our approach has lower energy consumption and is multiple times faster compared to two simple approaches. In particular, we showed that our approach, once it has performed a setup phase, is over 131 times faster and consumes 73 times less energy compared to offloading everything to a connected server. Still, our approach keeps quality requirements as requested and reports an error indicator to the user.

7

Conclusions and Outlook

7.1 Conclusions

Numerical simulations on mobile devices will enable new classes of applications for supporting engineers and scientists in the field. Especially in combination with augmented reality and sensor input, such mobile simulations will support a new form of pervasive applications for interaction, interpretation, and solution of complex physical problems. However, to realize complex numerical simulations on mobile devices, limitations of the devices have to be taken into account and require new concepts for a distributed execution between connected servers and the device itself.

This thesis introduced three major concepts for enabling numerical simulations on mobile devices. Depending on the actual application and the mobile environment, different concepts were discussed to provide (1) a robust execution in cases of frequent disconnections of the wireless link between server and mobile device, (2) a low-latency and faster distributed execution using data assimilation techniques and low-quality surrogate models of the simulation, and (3) concepts for utilizing model order reduction techniques to reduce the complexity of the simulation by pre-computation of reduced simulation models for parameter dependent stationary simulation problems.

In cases of frequent disconnections, we provided methods for deciding on the placement of the computation of future simulation states of a time-dependent simulation. Our methods predict the length of the disconnection, using historic data and Markov Chains, compare the options, and then decide on the placement.

In evaluations, we showed that our methods are able to decrease soft-deadline misses by more than 61 % compared to pure offloading. At the same time our methods save up to 74 % of energy compared to a simplified method.

The execution of the simulation can be changed in quality to provide significantly faster results with lower quality. This is used by concepts utilizing surrogate models for fast execution. The basic idea is to execute surrogate models on the mobile device and execute both, the full model and the surrogate model, on the server. This way, only updates to the surrogate model require communication to the mobile device, and thus save bandwidth and latency for communication of simulation results. Evaluations showed that our methods are up to 6.5 times faster than pure offloading while still meeting required quality constraints.

For parameterized stationary simulations, we utilized model order reduction techniques to generate reduced models that can be executed directly on the mobile device. We focused on the reduced basis method (RBM), that uses snapshots of the original simulation model in a pre-computation step to provide the reduced model. To this end, we provided methods to adapt the reduced model to provide simulation results with guaranteed quality constraints. Additionally, we provide methods to reduce the energy consumption and latency for the computation on the mobile device by reducing the required snapshots depending on the parameter of the execution and by modifying the original basis generation method. Our methods are able to speedup the computation by over 131 times while using 73 times less energy.

All of these concepts can be combined to enable a full mobile simulation middleware that provides fast execution even in the case of disconnections. As there exist RBM methods for time-dependent simulation models, reduced models can be used as surrogate models and refined where required using data assimilation techniques and server resources. Model order reduction is still an active field of research and more types of simulations will be supported and it is to be expected that new concepts will improve execution and generation of the reduced model for specific problems. However, there are still many other concepts that can be used to improve the distributed execution between mobile

devices and servers even further.

7.2 Outlook

This thesis opens a broad area of future work. In particular, there are hardware improvements, trends in local computing resources, and recent achievements in the model order reduction community that can be considered.

7.2.1 Hardware Improvements

There are two hardware trends requiring new concepts for the distributed execution of mobile simulations: 5G and approximate computing. While both are not yet available, they will require new concepts for the execution of complex applications, such as numerical simulations.

The next generation of cellular networks, 5G, will provide significantly different characteristics, e.g., co-existence of different wireless communication methods [AEK+16]. This might require new concepts for execution in harsh environments and for facing disconnections, especially as mobile devices might be connected by multiple different communication technologies at the same time. Concepts could utilize a low bandwidth communication link, e.g., to acknowledge results of a surrogate model when results are within quality constraints of the user.

Additionally, specialized approximate computing hardware might be introduced on mobile devices to save energy resources. As discussed in Section 6.9.5, approximate computing approaches reduce the voltage of chips degrading the quality of the result [XMK16]. While some works already discussed the execution of simulations on approximate computing hardware [SBKW16, SBW17], they did not focus on mobile devices and it remains unclear how such specialized hardware will be controllable by the application.

7.2.2 Utilizing Local Computing Resources

Another topic for future work is the utilization of local computing resources in fog computing environments [BMZA12], cloudlets [SBCD09, SSX+15], and multi-tier offloading infrastructures [BDR16]. Closer resources provide low-latency results and might provide better scalability of the overall system. Especially when fast simulation results are required, e.g., live-integration of sensor data, nearby resources can help to reduce the overall latency as the data has not to be communicated to the central cloud. As the network between fog node and cloud is wired, more traditional approaches could be used [FPS06]. However, also completely new concepts could be realized for such systems. For instance, the cloud provides an intermediate reduced model of the full simulation model and the mobile device will only use significantly less snapshots of this intermediate model with significantly lower quality at much lower complexity.

7.2.3 Recent Achievements in Model Order Reduction

Recent trends in the model order reduction community combine machine learning and the reduced basis method to provide solutions even for nonlinear problems [HU18]. As machine learning is very popular on mobile devices, frameworks such as Tensor Flow support mobile hardware architectures and might provide fast inference of already trained models [ABC+16]. However, concepts using machine learning require to train the model and might require more general concepts for enabling nonlinear models on mobile devices.

Publications

[DK14] Christoph Dibak and Boris Koldehofe. Towards Quality-aware Simulations on Mobile Devices. 44. Jahrestagung der Gesellschaft für Informatik e.V. (GI) Workshop, 2014.

[DDR15] Christoph Dibak, Frank Dürr, and Kurt Rothermel. Numerical Analysis of Complex Physical Systems on Networked Mobile Devices. International Conference on Mobile Ad-hoc and Sensor Systems (MASS), IEEE, 2015.

[DSD+17] Christoph Dibak, Andreas Schmidt, Frank Dürr, Bernard Haasdonk, and Kurt Rothermel. Server-Assisted Interactive Mobile Simulations for Pervasive Applications. International Conference on Pervasive Computing and Communications (PerCom), IEEE, 2017.

[DDR17] Christoph Dibak, Frank Dürr, and Kurt Rothermel. Demo: Server-Assisted Interactive Mobile Simulations for Pervasive Applications. International Conference on Pervasive Computing and Communications (PerCom) Workshops, IEEE, 2017.

[DHS+18] Christoph Dibak, Bernard Haasdonk, Andreas Schmidt, Frank Dürr, and Kurt Rothermel. Enabling Interactive Mobile Simulations Through Distributed Reduced Models. Pervasive and Mobile Computing, Elsevier, 2018.

[DNDR19] Christoph Dibak, Wolfgang Nowak, Frank Dürr, and Kurt Rothermel. Using Surrogate Models and Data Assimilation for Efficient Mobile Simulations, 2019. Preprint online: arxiv.org/abs/1911.10344.

Bibliography

[ABC+16] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[AEK+16] M. Ayyash, H. Elgala, A. Khreishah, V. Jungnickel, T. Little, S. Shao, M. Rahaim, D. Schulz, J. Hilt, and R. Freund. Coexistence of wifi and lifi toward 5g: concepts, opportunities, and challenges. *IEEE Communications Magazine*, 54(2):64–71, February 2016.

[AT08] Michel Armand and J-M Tarascon. Building better batteries. *Nature*, 451(7179):652, 2008.

[BBV09] Niranjan Balasubramanian, Aruna Balasubramanian, and Arun Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement*, pages 280–293. ACM, 2009.

[BDR13] Patrick Baier, Frank Dürr, and Kurt Rothermel. Opportunistic position update protocols for mobile devices. In *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing*, pages 787–796. ACM, 2013.

[BDR14] Florian Berg, Frank Dürr, and Kurt Rothermel. Optimal predictive code offloading. In *Proceedings of the 11th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, pages 1–10. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2014.

[BDR16] Florian Berg, Frank Dürr, and Kurt Rothermel. Increasing the Efficiency of Code Offloading in n-tier Environments with Code Bubbling. In *Proceedings of the 13th Annual International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*, November 2016.

[Ber18] Florian Berg. *Efficient Code Offloading Techniques for Mobile Applications*. Dissertation, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, March 2018.

[BJvLE98] Gerrit Burgers, Peter Jan van Leeuwen, and Geir Evensen. Analysis scheme in the ensemble kalman filter. *Monthly Weather Review*, 126(6):1719–1724, 1998.

[BMNP04] Maxime Barrault, Yvon Maday, Ngoc Cuong Nguyen, and Anthony T Patera. An 'empirical interpolation' method: application to efficient reduced-basis discretization of partial differential equations. *Comptes Rendus Mathematique*, 339(9):667–672, 2004.

[BMZA12] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16, New York, NY, USA, 2012. ACM.

[BZBP13] Hans-Joachim Bungartz, Stefan Zimmer, Martin Buchholz, and Dirk Pflüger. *Modellbildung und Simulation: eine anwendungsorientierte Einführung*. Springer-Verlag, 2013.

[CBC+10] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 49–62. ACM, 2010.

[CH10] Aaron Carroll and Gernot Heiser. An analysis of power consumption in a smartphone. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'10, pages 21–21, Berkeley, CA, USA, 2010. USENIX Association.

[CIM+11] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. Clonecloud: Elastic execution between mobile device and cloud. In *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11, pages 301–314, New York, NY, USA, 2011. ACM.

[Coo18] Intel Coorporation. Intel 64 and ia-32 architectures optimization reference manual, 2018.

[CPG+13] F. Capozzi, G. Piro, L. A. Grieco, G. Boggia, and P. Camarda. Downlink packet scheduling in lte cellular networks: Key design issues and a survey. *IEEE Communications Surveys Tutorials*, 15(2):678–700, Second 2013.

[DDR15] Christoph Dibak, Frank Dürr, and Kurt Rothermel. Numerical Analysis of Complex Physical Systems on Networked Mobile Devices. In *Proceedings of the 12th IEEE International Conference on Mobile Ad hoc and Sensor Systems (MASS 2015)*, Oct 2015.

[DDR17] C. Dibak, F. Dürr, and K. Rothermel. Demo: Server-assisted interactive mobile simulations for pervasive applications. In *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 68–70, March 2017.

[DHS+18] Christoph Dibak, Bernard Haasdonk, Andreas Schmidt, Frank Dürr, and Kurt Rothermel. Enabling interactive mobile simulations through distributed reduced models. *Pervasive and Mobile Computing*, 45:19 – 34, 2018.

[DK14] Christoph Dibak and Boris Koldehofe. Towards Quality-aware Simulations on Mobile Devices. In *Proceedings of the 44. Jahrestagung der Gesellschaft für Informatik e.V. (GI) (Informatik 2014)*, Lecture Notes in Informatics (LNI). Gesellschaft für Informatik (GI), Sep 2014.

[DLMT16] Erik D Demaine, Jayson Lynch, Geronimo J Mirano, and Nirvan Tyagi. Energy-efficient algorithms. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, pages 321–332. ACM, 2016.

[DNDR19] Christoph Dibak, Wolfgang Nowak, Frank Dürr, and Kurt Rothermel. Using surrogate models and data assimilation for efficient mobile simulations, 2019. Preprint online: arxiv.org/abs/1911.10344.

[DSD+17] Christoph Dibak, Andreas Schmidt, Frank Dürr, Bernard Haasdonk, and Kurt Rothermel. Server-assisted interactive mobile simulations for pervasive applications. In *Proceedings of the 15th IEEE International Conference on Pervasive Computing and Communications (PerCom 2017)*, pages 111–120. IEEE, Mar 2017.

[Ela02] Hala Elaarag. Improving tcp performance over mobile networks. *ACM Computing Surveys (CSUR)*, 34(3):357–374, 2002.

[Eve03] Geir Evensen. The ensemble kalman filter: Theoretical formulation and practical implementation. *Ocean dynamics*, 53(4):343–367, 2003.

[Eve04] Geir Evensen. Sampling strategies and square root analysis schemes for the enkf. *Ocean dynamics*, 54(6):539–560, 2004.

[Fis08] Gerd Fischer. *Lineare Algebra*. Vieweg + Teubner, 2008.

[FPS06] Rohit Fernandes, Keshav Pingali, and Paul Stodghill. Mobile mpi programs in computational grids. In *Proceedings of the Eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel*

Programming, PPoPP '06, pages 22–31, New York, NY, USA, 2006. ACM.

[FS99] Jason Flinn and Mahadev Satyanarayanan. Powerscope: A tool for profiling the energy usage of mobile applications. In *Second IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*, pages 2–10. IEEE, 1999.

[GA10] M. S. Grewal and A. P. Andrews. Applications of kalman filtering in aerospace 1960 to the present [historical perspectives]. *IEEE Control Systems*, 30(3):69–78, June 2010.

[GCD+10] Dirk Gorissen, Ivo Couckuyt, Piet Demeester, Tom Dhaene, and Karel Crombecq. A surrogate modeling and adaptive sampling toolbox for computer based design. *Journal of Machine Learning Research*, 11(Jul):2051–2055, 2010.

[GF04] Andrei Gurtov and Sally Floyd. Modeling wireless links for transport protocols. *ACM SIGC*, 34(2):85–96, 2004.

[GJM+12] Mark S Gordon, D Anoushe Jamshidi, Scott Mahlke, Z Morley Mao, and Xu Chen. Comet: code offload by migrating execution transparently. In *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, pages 93–106, 2012.

[GL04] William Gropp and Ewing Lusk. Fault tolerance in message passing interface programs. *The International Journal of High Performance Computing Applications*, 18(3):363–372, 2004.

[GMP+11] Vaibhav Gupta, Debabrata Mohapatra, Sang Phill Park, Anand Raghunathan, and Kaushik Roy. Impact: Imprecise adders for low-power approximate computing. In *Proceedings of the 17th IEEE/ACM International Symposium on Low-power Electronics and*

Design, ISLPED '11, pages 409–414, Piscataway, NJ, USA, 2011. IEEE Press.

[GRA12] Ioana Giurgiu, Oriana Riva, and Gustavo Alonso. Dynamic software deployment from clouds to mobile devices. In *Middleware 2012*, pages 394–414. Springer, 2012.

[Grc11] Joseph F Grcar. Mathematicians of gaussian elimination. *Notices of the AMS*, 58(6):782–792, 2011.

[Haa16] Bernard Haasdonk. Reduced basis methods for parametrized PDEs – a tutorial introduction for stationary and instationary problems, 2016. Chapter in P. Benner, A. Cohen, M. Ohlberger and K. Willcox: "Model Reduction and Approximation for Complex Systems", SIAM, Philadelphia.

[HKPP11] D.B.P. Huynh, D.J. Knezevic, J.W. Peterson, and A.T. Patera. High-fidelity real-time simulation on deployed platforms. *Computers & Fluids*, 43(1):74 – 81, 2011. Symposium on High Accuracy Flow Simulations. Special Issue Dedicated to Prof. Michel Deville Symposium on High Accuracy Flow Simulations.

[HLHG13] Shuai Hao, Ding Li, William GJ Halfond, and Ramesh Govindan. Estimating mobile application energy consumption using program analysis. In *Software Engineering (ICSE), 2013 35th International Conference*, pages 92–101. IEEE, 2013.

[HM98] P. L. Houtekamer and Herschel L. Mitchell. Data assimilation using an ensemble kalman filter technique. *Monthly Weather Review*, 126(3):796–811, 1998.

[HMS03] Christopher Hide, Terry Moore, and Martin Smith. Adaptive kalman filtering for low-cost ins/gps. *Journal of Navigation*, 56(1):143–152, 2003.

[HSML07] J. Hursey, J. M. Squyres, T. I. Mattox, and A. Lumsdaine. The design and implementation of checkpoint/restart process fault tolerance for open mpi. In *2007 IEEE International Parallel and Distributed Processing Symposium*, pages 1–8, March 2007.

[HU18] Jan S. Hesthaven and Stefano Ubbiali. Non-intrusive reduced order modeling of nonlinear problems using neural networks. *Journal of Computational Physics*, 363:55–78, 2018.

[KAH+12] Sokol Kosta, Andrius Aucinas, Pan Hui, Richard Mortier, and Xinwen Zhang. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *INFOCOM, 2012 Proceedings IEEE*, pages 945–953. IEEE, 2012.

[Kep00] Christian L. Keppenne. Data assimilation into a primitive-equation model with a parallel ensemble kalman filter. *Monthly Weather Review*, 128(6):1971–1981, 2000.

[KGVB05] Gaetan Kerschen, Jean-claude Golinval, ALEXANDER F. VAKAKIS, and LAWRENCE A. BERGMAN. The method of proper orthogonal decomposition for dynamical characterization and order reduction of mechanical systems: An overview. *Nonlinear Dynamics*, 41(1):147–169, Aug 2005.

[LDBNR13] Philipp C Leube, Felipe PJ De Barros, Wolfgang Nowak, and Ram Rajagopal. Towards optimal allocation of computer resources: Trade-offs between uncertainty quantification, discretization and model reduction. *Environmental Modelling & Software*, 50:97–107, 2013.

[LLM+09] Anna Larmo, Magnus Lindstrom, Michael Meyer, Ghyslain Pelletier, Johan Torsner, and Henning Wiemann. The lte link-layer design. *Communications Magazine, IEEE*, 47(4):52–59, 2009.

[LSZ15a] Kody Law, Andrew Stuart, and Konstantinos Zygalakis. *Discrete Time: Filtering Algorithms*, pages 79–114. Springer, 2015.

[LSZ15b] Kody Law, Andrew Stuart, and Kostas Zygalakis. *Data assimilation.* Springer, 2015.

[LXCT16] Yepang Liu, Chang Xu, Shing-Chi Cheung, and Valerio Terragni. Understanding and detecting wake lock misuses for android applications. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 396–409. ACM, 2016.

[MN98] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998.

[MPI93] The MPI Forum. Mpi: A message passing interface. In *Proceedings of the 1993 ACM/IEEE Conference on Supercomputing*, Supercomputing '93, pages 878–883, New York, NY, USA, 1993. ACM.

[MPI15] The MPI Forum. Mpi: A message-passing interface standard – version 3.1, 2015.

[PHZ+11] Abhinav Pathak, Y Charlie Hu, Ming Zhang, Paramvir Bahl, and Yi-Min Wang. Fine-grained power modeling for smartphones using system call tracing. In *Proceedings of the sixth conference on Computer systems*, pages 153–168. ACM, 2011.

[PP16] P. Pandey and D. Pompili. Mobidic: Exploiting the untapped potential of mobile distributed computing via approximation. In *2016 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 1–9, March 2016.

[PP17] Parul Pandey and Dario Pompili. Exploiting the untapped potential of mobile distributed computing via approximation. *Pervasive and Mobile Computing*, 2017.

[PYLFT16] Alberto Paradisi, Michel Daoud Yacoub, Fabrício Lira Figueiredo, and Tania Regina Tronco, editors. *Long Term Evolution: 4G and Beyond*. Telecommunications and Information Technology. Springer, Cham, 1st ed. 2016 edition, 2016.

[RD15] Daniel A. Reed and Jack Dongarra. Exascale computing and big data. *Commun. ACM*, 58(7):56–68, June 2015.

[RHP08] G. Rozza, D. B. P. Huynh, and A. T. Patera. Reduced basis approximation and a posteriori error estimation for affinely parametrized elliptic coercive partial differential equations. *Archives of Computational Methods in Engineering*, 15(3):229, May 2008.

[RJ15] James Reinders and Jim Jeffers. *High Performance Parallelism Pearls Volume Two: Multicore and Many-core Programming Approaches*. Morgan Kaufmann, 2015.

[RSM+11] Moo-Ryong Ra, Anmol Sheth, Lily Mummert, Padmanabhan Pillai, David Wetherall, and Ramesh Govindan. Odessa: enabling interactive perception applications on mobile devices. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 43–56. ACM, 2011.

[RWMS18] U. Rüde, K. Willcox, L. McInnes, and H. Sterck. Research and education in computational science and engineering. *SIAM Review*, 60(3):707–754, 2018.

[S+94] Jonathan Richard Shewchuk et al. An introduction to the conjugate gradient method without the agonizing pain, 1994.

[SBCD09] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *IEEE Pervasive Computing*, 8(4):14–23, Oct 2009.

[SBKW16] A. Schöll, C. Braun, M. A. Kochte, and H. Wunderlich. Efficient algorithm-based fault tolerance for sparse matrix operations. In

2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pages 251–262, June 2016.

[SBW17] A. Schöll, C. Braun, and H. Wunderlich. Energy-efficient and error-resilient iterative solvers for approximate computing. In *2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 237–239, July 2017.

[SM12] Audie Sumaray and S. Kami Makki. A comparison of data serialization formats for optimal efficiency on a mobile platform. In *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication*, ICUIMC '12, pages 48:1–48:6, New York, NY, USA, 2012. ACM.

[SN01] Mahadev Satyanarayanan and Dushyanth Narayanan. Multi-fidelity algorithms for interactive mobile applications. *Wireless Networks*, 7(6):601–607, 2001.

[Soc16] IEEE Computer Society. Ieee standard for information technology – telecommunications and information exchange between systems local and metropolitan area networks – specific requirements – part 11: Wireless lan mediaum access control (mac) and phyiscal layer (phy) specifications, 2016.

[SSX+15] Mahadev Satyanarayanan, Pieter Simoens, Yu Xiao, Padmanabhan Pillai, Zhuo Chen, Kiryong Ha, Wenlu Hu, and Brandon Amos. Edge analytics in the internet of things. *IEEE Pervasive Computing*, 14(2):24–31, 2015.

[TAN+12] Narendran Thiagarajan, Gaurav Aggarwal, Angela Nicoara, Dan Boneh, and Jatinder Pal Singh. Who killed my battery?: analyzing mobile browser energy consumption. In *Proceedings of the 21st international conference on World Wide Web*, pages 41–50. ACM, 2012.

[Tur48] Alan M Turing. Rounding-off errors in matrix processes. *The Quarterly Journal of Mechanics and Applied Mathematics*, 1(1):287–308, 1948.

[VJLA12] Panagiotis Vekris, Ranjit Jhala, Sorin Lerner, and Yuvraj Agarwal. Towards verifying android apps for the absence of no-sleep energy bugs. In *HotPower*, 2012.

[VPRP03] Karen Veroy, Christophe Prud'Homme, Dimitrios V Rovas, and Anthony T Patera. A posteriori error bounds for reduced-basis approximation of parametrized noncoercive and nonlinear elliptic partial differential equations. In *Proceedings of the 16th AIAA computational fluid dynamics conference*, volume 3847, pages 23–26. Orlando, Florida, 2003.

[Wal02] Bernhard H. Walke. *Mobile radio networks*, volume 2. John Wiley & Sons, 2002.

[WB96] Mark Weiser and John Seely Brown. Designing calm technology. *PowerGrid Journal*, 1(1):75–85, 1996.

[Wei91] Mark Weiser. The computer for the 21st century. *Scientific american*, 265(3):94–104, 1991.

[XMK16] Q. Xu, T. Mytkowicz, and N.S. Kim. Approximate computing: A survey. *Design & Test, IEEE*, 33(1):8–22, Feb 2016.

[YKC+13] S. Yang, Y. Kwon, Y. Cho, H. Yi, D. Kwon, J. Youn, and Y. Paek. Fast dynamic execution offloading for efficient mobile cloud computing. In *International Conference on Pervasive Computing and Communications (PerCom)*, pages 20–28, March 2013.

[Zil96] Shlomo Zilberstein. Using anytime algorithms in intelligent systems. *AI magazine*, 17(3):73, 1996.

[ZM07] Jim Zyren and Wes McCoy. Overview of the 3gpp long term evolution physical layer. *Freescale Semiconductor, Inc., white paper*, 22, 2007.

[ZNW15] Y. Zhang, D. Niyato, and P. Wang. Offloading in mobile cloudlet systems with intermittent connectivity. *IEEE Transactions on Mobile Computing*, 14(12):2516–2529, Dec 2015.