Numerical Analysis of Complex Physical Systems on Networked Mobile Devices

Christoph Dibak, Frank Dürr, Kurt Rothermel
Institute of Parallel and Distributed Systems
University of Stuttgart, Stuttgart, Germany
Email: first.last@ipvs.uni-stuttgart.de

***Abstract*—Recently, a new class of mobile applications has appeared that takes into account the behavior of physical phenomenon. Prominent examples of such applications include augmented reality applications visualizing physical processes on a mobile device or mobile cyber-physical systems like autonomous vehicles or robots. Typically, these applications need to solve partial differential equations (PDE) to simulate the behavior of a physical system. There are two basic strategies to numerically solve these PDEs: (1) offload all computations to a remote server; (2) solve the PDE on the resource-constrained mobile device. However, both strategies have severe drawbacks. Offloading will fail if the mobile device is disconnected, and resource constraints require to reduce the quality of the solution.**

Therefore, we propose a new approach for mobile simulations using a hybrid strategy that is robust to communication failures and can still benefit from powerful server resources. The basic idea of this approach is to dynamically decide on the placement of the PDE solver based on a prediction of the wireless link availability using Markov Chains. Our tests based on measurement in real cellular networks and real mobile devices show that this approach is able to keep deadline constraints in more than 61 % of the cases compared to a pure offloading approach, while saving up to 74 % of energy compared to a simplified approach.

Keywords*—*mobile cloud computing; numerical applications; mobile cyber-physical systems; augmented reality;

I. Introduction

Recent years have seen two new kinds of mobile applications: mobile cyber-physical systems (MCPS) and augmented reality (AR). MCPS are used to control the behavior of robots [1], self-driving cars [2], or even blood values in human bodies [3] where sensor data is used to control aspects of the physical world. AR displays information about objects into the field of vision of the user [4]. For instance, the information displayed could be the result of a physical simulation interesting for an engineer, e.g., showing the flow of air in a room, the distribution of heat in an objected based on sensor readings, or even the combination of airflow and heat in a multi-physics simulation.

Many such physical phenomenons controlled by MCPS or visualized by AR applications are described by means of partial differential equations (PDE) [3]. PDEs describe how a function representing the system has to behave. Solutions to a PDE are therefore functions. For applications it is sufficient to approximate the solution of a PDE numerically. There are a number of techniques to approximate the solution of a PDE, e.g., the finite elements method (FEM), the finite differences method (FDM), or the finite volumes method (FVM), among others [5].

Traditionally, PDEs have been solved on huge compute clusters in high performance computing. However, for MCPS and AR applications, the solution needs to be available on mobile devices. Mobile devices are very restricted in computing power. Therefore, current MCPS and AR solve PDEs either on a connected server or compute results in low-quality on the mobile device itself. While the latter may give insufficient results, the former is subject to disconnections of the wireless communication link.

In order to avoid both problems, we present a new method for solving PDEs on a distributed infrastructure consisting of a mobile device and a server. Our method uses both nodes for computation to benefit from fast compute resources while connected and still be tolerant to disconnections. In order to adapt to the dynamic quality of the wireless network, we present a method of how to detect disconnections and how to predict the duration of disconnections statistically. Using this information, our method is able to decide on whether to wait for the link to become available or to start computation on the mobile device, once disconnection is detected.

In detail, our contributions in this paper are (1) the identification of a new problem, solving PDEs on mobile devices using a distributed infrastructure; (2) a method of how to provide energy efficient computation on the mobile device while fulfilling time constraints, suitable for mobile real-time applications; and (3) an evaluation of our method based on real-world data taken from cellular networks in the region of Stuttgart.

Our evaluations show that using our method we are able to keep the deadline in more than 61 % of the cases compared to a pure remote computation in a real-world setting including disconnections. At the same time, we still can reduce the additional computation overhead on the mobile device to a constant, compared to a simplified approach. Computation overhead on the device also represents additional energy consumption. Therefore, our method carefully meets decisions on the trade-off between energy and execution time, as needed for real-time applications, with only small energy cost for the mobile processor.

The reminder of this paper is structured as follows: Section II presents the system model followed by the problem statement for solving PDEs on mobile device in Section III. Section IV presents an architecture and overview of our method. Sections V–VIII present our method to solve PDEs in

Published in Proceedings of the 12th IEEE International Conference on Mobile Ad hoc and Sensor Systems (MASS 2015), Dallas, USA, October 2015

a mobile and distributed environment. In Section IX we evaluate our methods against a simple approach, before presenting related work in Section X and concluding the paper with an outlook on future work.

II. System Model

This section introduces our system model consisting of the numerical solver as the application, the mobile and fixed compute nodes, the wireless link connecting the nodes, and an energy model for the mobile device.

Many processes in science and engineering are described by means of partial differential equations (PDEs). PDEs define how a function, describing properties of a physical system, has to behave. For example, function u depends on time t and space x and describes the transfer of heat in an object, which is known to fulfill the PDE

$$\frac{\partial u}{\partial t} - \nabla^2 u = 0. \tag{1}$$

For applications, it is sufficient, and in many cases the only practical way, to approximate a solution, i.e., a function fulfilling the properties of the PDE. Finding an approximate solution is called *solving* the equation and is implemented in a *solver*. There are a number of methods available of how PDEs can be solved, e.g., implicit or explicit methods for finite differences, for the heat transfer example. For such methods, the solution is not approximated continuously, but at discrete points in time and space. For example, for the heat transfer, the initial values for all discrete points in physical space at time t_0 might be given at discrete points and we want to compute the evolution of the system for time $t > t_0$ at those points. This is also called an *initial value problem* and is common in many engineering applications. For finding an approximate solution for this problem, we can split time into equidistant time slots, $t_i = t_0 + i \cdot \Delta t$, and compute the approximation at each discrete point in space for each t_i. We call the approximation at one t_i a *state* and denote it as s_i. For computation of state s_i, the state s_{i-1} at time t_{i-1} is required. For explicit methods, this computation is simply one equation for each point in time. However, for implicit methods, which have better numerical stability, the computation of a state requires the solution of algebraic equations [5] (see Fig. 1). We assume the application needs n states. The set of all states to be computed by the solver is denoted as $S = \{s_1, \ldots, s_n\}$.

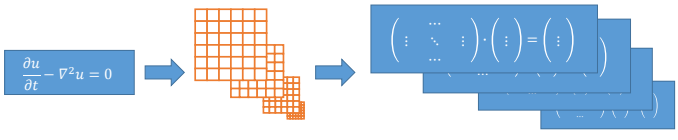


Fig. 1. Implicit discretization method transforms PDEs to algebraic equations

The solver, as described above, is implemented for two compute nodes, server and mobile device. These two nodes are very 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 layer which will result in packet loss on higher layers. There are two kinds of communication errors in wireless networks, single packet errors and burst packet errors [6]. While the former effects single packets only, the latter effects all packets over a period. Single packet errors are easily avoided by introducing retransmission on the link layer as it is implemented in LTE [7]. Therefore, the majority of errors are burst errors (cf. Section IX). Notice that loosing all packets over a period causes temporary disconnection.

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. We consider computations on the mobile device as the dominant factor for energy consumption and neglect the energy consumption for communication. Especially when computation of one step for the application might involve multiple intermediate steps, not being necessary for visualization or reasoning but for improved quality, e.g., for finer coupling between different solvers for multi-physics simulations, computation becomes the predominant factor of energy consumption. Therefore, we will minimize the number of steps computed on the mobile device.

III. 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 at wall clock time $t_{\max}$. 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 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, 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 send to the mobile device. This decision is part of a schedule (M, T) and represented in 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 received on the mobile device at time t_i.

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 computations on the mobile device, under the constraint that all states should be available on the mobile device at time $t_{\max}$.

IV. 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 [8].

Our system consists of four components (see Fig. 2), (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. All components run on the mobile device.

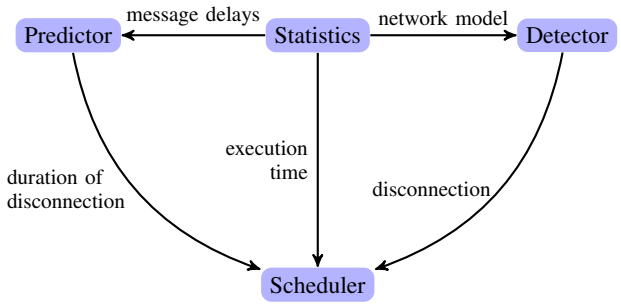


Fig. 2. Overview of the architecture

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 temporary disconnected, the scheduler has two options (1) waiting for the link to become available (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 packet 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.

V. 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 and (2) computation on the server to not induce any cost, as it does not cost energy 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, which is to minimize the number of 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 (see Fig. 3).

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

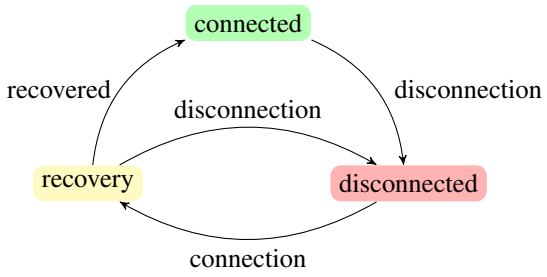


Fig. 3. Modes and mode transition of the Scheduler

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 disconnection (1) wait for the link to become available again (2) compute on the mobile device. The two alternatives have different effect on the objective function. Waiting does not introduce any additional cost, while computation increases 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 on 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. These values are provided by the *predictor* and the *statistics component*. The scheduler evaluates two predicates. The first predicate evaluates 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 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 from the "recovery" mode.

VI. 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 II. 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 [9] 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 depicts the mean time for solving linear equations on different device classes. We chose four different device classes: (1) classical stationary desktop PCs ; (2) mobile laptops ; (3) Smartphones like the LG Nexus 5 ; (4) small and cheap 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.

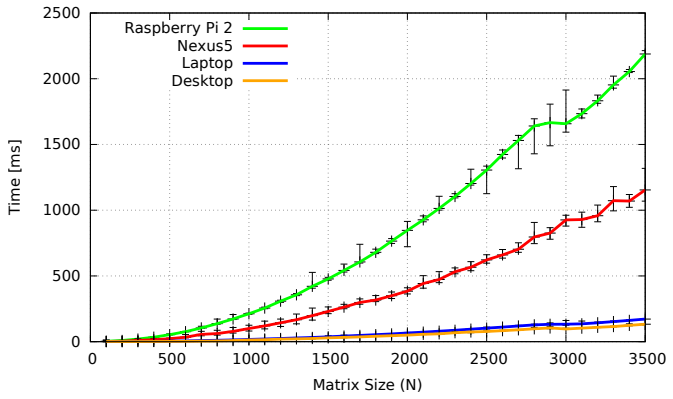


Fig. 4. Mean time to solve linear equations on different device classes. Error bars represent maximum and minimum time.

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.

VII. 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 depend 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 3G 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 5 depicts the empirical distribution function of received messages over time. The time is calibrated to minimum transfer time of any packet. Figure 5 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 %.

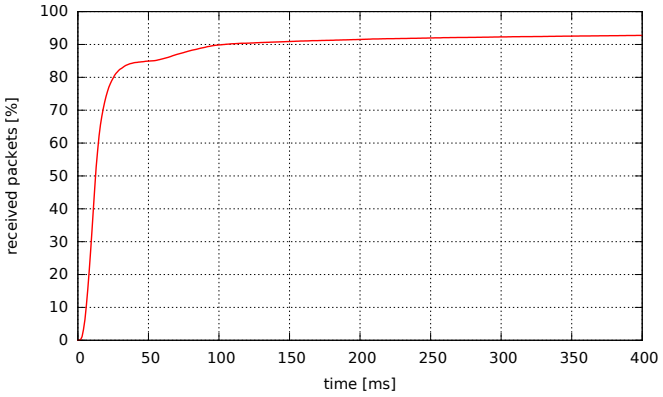


Fig. 5. Empirical distribution of received packets over time.

As a result of this tests, we found the 0.9-quantile as a good trade-off between time to detection and number of false detections. The 0.9-quantile guarantees to detect 90 % of received packets correct, while detecting 10 % of later received packages as disconnections. The 0.9-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 given provider and location.

VIII. 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 [6]. 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, $P((a,b) \to c)$ 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 (see Fig. 6). The probability $p = P((x,y) \to A)$, 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. 6 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.

```
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: function GetProb((a, b) → c)      ▷ returns P((a, b) → c)
12:     sum ← counter[(a, b, c)] + counter[(a, b, ¬c)]
13:     if sum = 0 then return 1 / 2
14:     return counter[(a, b, c)] / sum
15: function ExpectedDurationDisconnection
16:     prob ← GetProb((D, D) → A)
17:     if prob = 0 then return ∞
18:     return 1 / prob
```

Fig. 6. 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).

IX. 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.

A. 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 II. It is given as

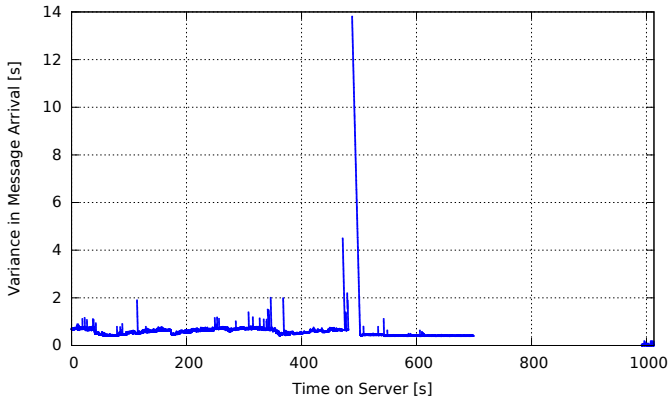
$$\frac{\partial u}{\partial t} - \nabla^2 u = 0.$$

and describes the evolution of heat 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.

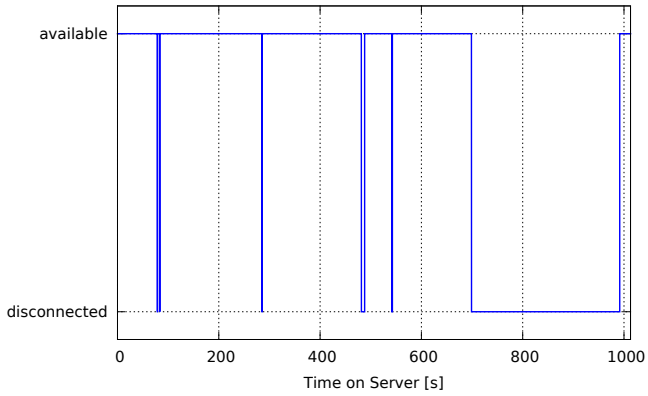
For the computation nodes, we assume the mobile device to be equally equipped as an LG Nexus 5 Smartphone and the server to have four cores at 2.6 GHz. For computation of one time step, the devices have to solve a system of $1,400$ linear equations. According to our preliminary tests in Section VI (see Fig. 4), 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 VII. 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 different cellular provider and 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 [10]. 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 7a depicts the relative arrival times of packets sent by the server over time in one of our collected real-world datasets. Notice the very high latency of nearly 14 s. Figure 7b depicts the link state of the mobile device, which is either available or disconnected, of the same dataset. Notice that 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



(a) Variance in message arrival over time



(b) State of the cellular network over time

Fig. 7. One of the samples collected per-pedes and on the train. Both figures are based on the same data.

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.

B. Evaluation of the Disconnection Detector

For the evaluation of the *disconnection detector* we use the real-world data of cellular networks. In Section VII, we already described how we did preliminary tests and found a 0.9-quantile for the minimum timeout for detection of disconnections a good choice. However, in the data collected for the evaluation, the 0.9-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 8 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, the *detector* has detected correctly, 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.

C. Evaluation in an LTE Scenario

In the reminder 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

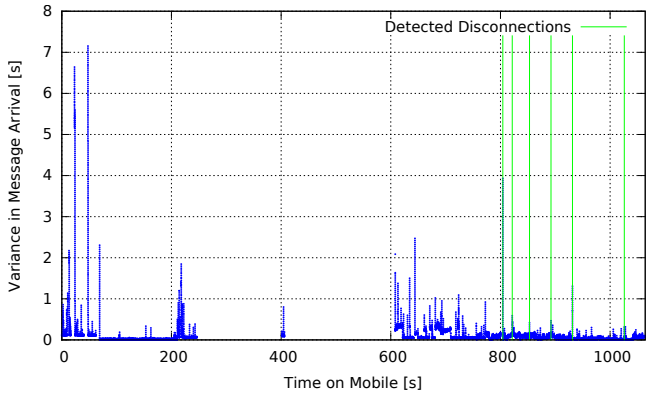


Fig. 8. Sample data and decision of the disconnection detector. The detector returns 5 disconnections, depicted as green lines. Only the third disconnection is an actual disconnection, the others are false-positives reported by the detector.

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 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 energy consumption, expressed as the time of the mobile processor usage.

Figure 9 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 on 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.

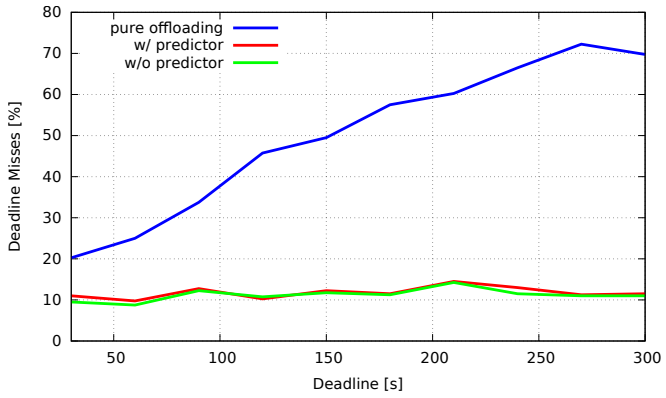


Fig. 9. Fraction of deadline misses over variable deadline of the three approaches.

Figure 10 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 optimal solutions. Comparing our approaches with and without the predictor, the approach with the predictor is able to reduce energy consumption by more than 74 %. The difference in energy consumption with and without the predictor can be described as a constant. Therefore, the predictor is an essential component to provide better results in terms of energy consumption on the mobile device.

Overall, our approaches with and without the predictor are able to fulfill the constraint on the deadline much better than

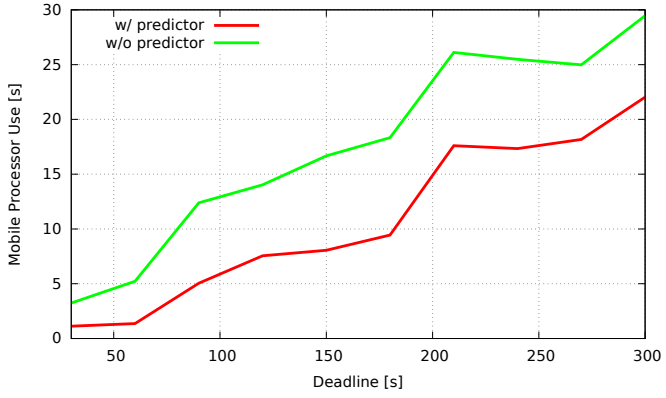


Fig. 10. Energy consumption on the mobile device of our methods with and without the predictor.

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.

X. Related Work

This section discusses related work of how PDEs can be solved for mobile devices.

Classically, PDEs are computed on clusters in high performance computing (HPC). HPC assumes a set of homogeneous and well connected compute nodes. They often have the exactly same processor model and are fast and reliable connected via a wired network. Usually, the computation task is distributed using MPI [11]. This protocol distributes the task to a number of processors. To provide the result to a mobile device, either the result is computed on the cluster and is transferred to the mobile device, or the mobile device is included into the computation using MPI.

Other related research fields are mobile code-offloading approaches and cyber foraging. These approaches utilize the mobile processor and additional remote resources. Code-offloading uses resources provided by a single server located in a data center. Cyber foraging utilizes unused computing resources nearby [12], [13]. Both approaches either try to speed up the application [14], [15] or to save energy [16], [17]. The main problem is the decision of how to distribute different parts, called modules, of the application to optimize for energy or execution time. 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.

Both classical and mobile approaches provide insufficient results for our problem. The classical HPC approach does not provide any solution in the case of disconnections. Many mobile approaches use remote execution to reduce the energy consumption or execution time, when the link and resources are available and do not provide any solution in the case of disconnections. Recent code-offloading approaches are able to cover disconnections [18], [19]. However, those approaches rely on the transfer of one single application state, which makes merging of remotely executed and locally executed states impossible and therefore significantly increases the volume and latency of data to be transferred over the network.

XI. Conclusion & Future Work

We described a new problem for solving PDEs on a distributed infrastructure, which will be needed for future applications in augmented reality and Mobile Cyber-Physical Systems. We also described how we can solve this problem using a mobile device and a server connected over a wireless communication link. Wireless links such as over 3/4G cellular network are subject to burst errors, where multiple successive packets are

not delivered and the link is not available for a longer period, i.e., the device is temporary 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.

In the future we also want to focus on available bandwidth and the ability of numerical applications to change quality.

ACKNOWLEDGMENT

The authors would like to thank Steven Großmann for his help in the evaluation. Furthermore, the authors would like to thank the German Research Foundation (DFG) for financial support of the project within the Cluster of Excellence in Simulation Technology (EXC 310/2) at the University of Stuttgart.

REFERENCES

[1] R. R. Rajkumar, I. Lee, L. Sha, and J. Stankovic, "Cyber-physical systems: the next computing revolution," in *Proceedings of the 47th Design Automation Conference*. ACM, 2010, pp. 731–736.

[2] J. Kim, H. Kim, K. Lakshmanan, and R. R. Rajkumar, "Parallel scheduling for cyber-physical systems: Analysis and case study on a self-driving car," in *Proceedings of the ACM/IEEE 4th International Conference on Cyber-Physical Systems*. ACM, 2013, pp. 31–40.

[3] A. Banerjee and S. Gupta, "Analysis of smart mobile applications for healthcare under dynamic context changes," *Mobile Computing, IEEE Transactions on*, vol. 14, no. 5, pp. 904–919, May 2015.

[4] J. Carmigniani, B. Furht, M. Anisetti, P. Ceravolo, E. Damiani, and M. Ivkovic, "Augmented reality technologies, systems and applications," *Multimedia Tools and Applications*, vol. 51, no. 1, pp. 341–377, 2011.

[5] K. W. Morton and D. F. Mayers, *Numerical solution of partial differential equations: an introduction*. Cambridge university press, 2005.

[6] A. Gurtov and S. Floyd, "Modeling wireless links for transport protocols," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp. 85–96, 2004.

[7] A. Larmo, M. Lindstrom, M. Meyer, G. Pelletier, J. Torsner, and H. Wiemann, "The lte link-layer design," *Communications Magazine, IEEE*, vol. 47, no. 4, pp. 52–59, 2009.

[8] C. Dibak and B. Koldehofe, "Towards quality-aware simulations on mobile devices," in *Proceedings of the 44. Jahrestagung der Gesellschaft für Informatik e.V. (GI) (Informatik 2014)*. Gesellschaft für Informatik (GI), September 2014, pp. 89–100.

[9] J. R. Shewchuk, "An introduction to the conjugate gradient method without the agonizing pain," 1994.

[10] J. Zyren and W. McCoy, "Overview of the 3gpp long term evolution physical layer," *Freescale Semiconductor, Inc., white paper*, 2007.

[11] J. Gracia, C. Niethammer, M. Hasert, S. Brinkmann, R. Keller, and C. W. Glass, "Hybrid mpi/starss - a case study," *CoRR*, vol. abs/1204.4086, 2012.

[12] M. Sharifi, S. Kafaie, and O. Kashefi, "A survey and taxonomy of cyber foraging of mobile devices," *Communications Surveys Tutorials, IEEE*, vol. 14, no. 4, pp. 1232–1243, Fourth 2012.

[13] D. Lima, H. Miranda, and F. Taïani, "Towards a new model for cyber foraging," in *The 13th Workshop on Adaptive and Reflective Middleware (ARM 2014), in conjunction with ACM/IFIP/USENIX ACM International Middleware Conference 2014, Bordeaux, France*, 2014.

[14] M.-R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: enabling interactive perception applications on mobile devices," in *Proceedings of the 9th international conference on Mobile systems, applications, and services*. ACM, 2011, pp. 43–56.

[15] I. Giurgiu, O. Riva, and G. Alonso, "Dynamic software deployment from clouds to mobile devices," in *Middleware 2012*. Springer, 2012, pp. 394–414.

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

[17] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *Proceedings of the sixth conference on Computer systems*. ACM, 2011, pp. 301–314.

[18] F. Berg, F. Dürr, and K. Rothermel, "Increasing the efficiency and responsiveness of mobile applications with preemptable code offloading," in *Mobile Services (MS), 2014 IEEE International Conference on*. IEEE, 2014, pp. 76–83.

[19] F. Berg, F. Durr, and K. Rothermel, "Optimal Predictive Code Offloading," in *Proceedings of the 11th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*. IEEE Computer Society, 2014, pp. 1–10.