Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelor Thesis Nr. 172

Efficient Solving of Linear Equations on Mobile Devices Utilizing a Cloud-Infrastructure

Steven Großmann

Course of Study:	Softwaretechnik
Examiner:	Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel
Supervisor:	Dipl.-Inf. Christoph Dibak
Commenced:	October 14, 2014
Completed:	April 15, 2015
CR-Classification:	C.1.3, C.2.1, C.2.4

Abstract

As the usage of mobile computer devices like smartphones got popular over the last decade, mobile applications got more important and higher sophisticated. An example of resource demanding applications are numerical simulations. However the resources of mobile devices are not as powerful as on stationary computers due to the mobility aspect. Proceedings like code offloading provide solutions to bypass the limitations of mobile resources. In this bachelor thesis a system was developed to efficiently solve linear equation systems on mobile devices by utilizing a Cloud-Infrastructure. The system focuses on optimizing the resources energy, computation time and result quality according to the network situation. Evaluation tests recorded energy savings of 70% and a computation time reduction of 80% compared to conventional proceedings. Though those values depend on the network situation.

Kurzfassung

In dem vergangenen Jahrzehnt sind mobile Computer Geräte, wie z.B. Smartphones populär geworden, wodurch mobile Anwendungen wichtiger und komplexer wurden. Ein Beispiel für Ressourcen fordernde Anwendungen sind numerische Simulationen. Jedoch sind bei mobilen Geräten, aufgrund des Mobilitätsaspekt, weniger Ressourcen vorhanden als bei stationären Computern. Verfahren wie Code Offloading bieten Lösungen an, um diese Beschränkungen zu umgehen. In dieser Bachelorarbeit wurde ein System entwickelt, das unter Verwendung von Cloud-Infrastruktur ein effizientes Lösungsverfahren bietet, um lineare Gleichungssysteme auf Mobilgeräten zu lösen. Das System fokussiert dabei die Optimierung der Ressourcen Energie, Berechnungszeit und Ergebnisqualität im Bezug auf die Netzwerksituation. Evaluierungstests ergaben Energieeinsparungen von 70% und eine Reduktion der Berechnugszeit von 80%, im Vergleich zu herkömmlichen Verfahren. Jedoch sind diese Werte abhängig von der Netzwerksituation.

Acknowledgements

My special thanks goes to my supervisor Christoph Dibak for his support during research and development for this bachelor thesis.

Further I would like to thank all people of the Institute of Parallel and Distributed Systems who helped providing devices and infrastructure to develope the system and conduct evaluation tests.

Contents

List of Figures

List of Tables

List of Listings

List of Algorithms

1 Preamble

In the past years, a change in the prensence of mobile devices occured. Smartphones and mobile computers became ubiqutious in industry nations. According to the rise of mobile devices, mobile applications became more popular for users of the devices [Gon10]. The applications are using all possible resources of the mobile devices, like movement sensors, cameras, user input and output (IO) devices, etc. . As a consequence thereof, the applications became higher sophisticated and more complex. While the CPUs and memory for mobile devices increased in speed and capacity, they were always behind stationary computers in the context of performance [Sok12]. Another problem that restricts the usage of complex and resource demanding mobile applications, is the battery of the mobile device. As the computational power increased drastically over the years, the capacity of batteries did not improve considerable [Edu10]. That is one reason, why developers and researchers had to reconsider the implementation of mobile applications.

Very popular mobile applications with a high resource demand are e.g., video streaming applications like YouTube. They became popular with the rise of smart phones like the iPhone or Android devices. Those applications can drain a battery of a mobile device within a few hours [Edu10]. Another special scenario for resource demanding and complex applications are simulations. Simulations need a huge amount of input data and according to this, big calculations have to be solved. Many simulations in engineering and natural sciences are numerical simulations. Therefore an important part are equation systems. Such equation systems can grow proportional to the input data of a simulation, like in spline interpolation. The resulting equations contain in most cases sparse and diagonally dominant matrices [Wol94]. As simulation programs are popular on stationary computers since the 1960 [G. 61], they also became popular on mobile devices as they got more processing power [Chr14].

A very important feature of mobile devices, is the possibility of networking. It allows to communicate with other computers of nearly any platform and without restriction of their location. This progress is owed to wireless networks like WiFi or cellular networks like 3G/4G [Edu10]. With those communication techniques, a whole new possibility of usage scenarios appeared. Mobile applications are no longer bound to the hardware they are running on. They can use resources that are connected to the same local area network or the Internet. With this feature, it is possible to move the resource demanding calculations to stronger stationary servers in a cloud infrastructure. This method is called code offloading [Edu10]. With code offloading it is possible to reduce the energy consumption, while increasing the computation speed of applications.

A problem that goes with networking, is for example high latency, low throughput or temporary connection losses. While most big cities in industry nations have a good mobile network connection, smaller cities or rural areas have a lack of stable network connection. Also not all public transportation systems in cities can ensure mobile network connection in tunnels. According to the resulting communication link failures, a protocol has to handle the execution of code in relation to the current network situation.

1.1 Goals

The focus of this thesis is to develop a procedure, to efficiently solve linear equation systems on mobile devices, utilizing a Cloud-Infrastructure. The applied algorithms use an iterative approximation approach and are specialized to solve diagonally dominant matrices. The resources that define the efficiency for this approach are energy, computation time and quality of the solution. The main factor that influences the efficiency is the network situation.

1.2 Structure

This thesis is structured into ten chapters. The first chapter provides a preamble, defines the goals and explains the structure of this thesis. Chapter 2 explains the necessary mathematical preconditions for the algorithms that are discussed in further chapters. The following Chapter 3 describes projects that share related approaches and address equivalent problems to this bachelor thesis. The fourth chapter gives an overview about the results that were found during pre-tests for different algorithms on stationary computers and mobile devices. The system model of the developed approach is explained in Chapter 5. The architecture of the developed approach is described in the chapter System Design. Chapter 7 explains the detailed implementation of the system components of the developed test application and server. The results of the final tests are depicted and evaluated in Chapter 8. Chapter 9 summarizes the findings of this bachelor thesis and gives an outlook about future work. The Appendix is located in the last chapter.

2 Background

This chapter describes the background of efficient proceedings to solve linear equation systems. The Jacobi method serves as representative for efficient iterative algorithms and will be explained in the following section.

2.1 Efficient Solving of Linear Equation Systems

Linear equation systems are widely spread in problems of subjects like engineering or natural sciences. Those equation systems consist of multi-dimensional matrices, meaning several thousand dimensions and more. Since these problems can be solved with electric computers, the dimensions got even bigger [Ach95]. Typical proceedings like the Gaussian elimination are not well applicable for those problems, as they are too complex and slow on big equation systems. A better solution is provided by iterative solving methods. Iterative proceedings are important for solving problems like the method of least squares, spline interpolation or partial differential equations [Ach95]. Most of the matrices that are generated for partial differential equations or spline interpolation lead to sparse matrices, often diagonally dominant [Wol94]. A sparse matrix can be defined as a matrix with very few non zero elements. Proceedings like the Jacobi method take advantage of sparse and diagonally dominant matrices and provide a more simple and faster solving approach than typical proceedings.

2.2 Jacobi Method

The method described by C.G. Jacobi in 1845 calculates a precise approximation result of a linear equation system $A * x = b$ [Ach95]. In each iteration step an approximation of the result is calculated. That means the algorithm converges against the final result in a way, that the approximation of the iteration $i + 1$ has a better precision to the real result than the approximation of the iteration i. Therefore a predefined residual has to determine the termination criteria to stop the algorithm when the desired precision reached. Obviously, the algorithm can speed up if the residual addresses a lower quality.

The matrix of the equation gets decomposed by the Jacobi method into three parts $A = D + E + F$ [Wol94]. Where D is the diagonal matrix, E the strict lower part and F the

strict upper part. For the diagonal elements we assume $\forall d \in D : d \neq 0$. The Jacobi iteration determines the i-th component of the next approximation. ξ_i^k denotes the i-th component of the iterate x_k and β_i the i-th component of the equation result b.

Algorithm 2.1 Jacobi Method Algorithm

```
procedure JACOBI(A,x^0,b)
    length ← length of x^0
    k ← 0
    while termination criteria not reached do
        for i ← 0 to length do
            σ ← 0
            for j ← 0 to length do
                if i ≠ j then
                    σ ← σ + a_ij ξ_j^k
                end if
            end for
            ξ_i^{k+1} = (β_i − σ) / a_ii
        end for
        k ← k + 1
    end while
end procedure
```

That is $x_k = \begin{pmatrix} \xi_0^k \\ \vdots \\ \xi_n^k \end{pmatrix}$ and $b = \begin{pmatrix} \beta_0 \\ \vdots \\ \beta_n \end{pmatrix}$.

This leads to the following equation for every iteration [Wol94]:

$\xi_i^{k+1} = \frac{1}{a_{ii}}(\beta_i - \sum_{j=1, j\neq i}^{n} a_{ij}\xi_j^k)$, $i = 1, ..., n$

And can be rewritten as:

$x_{k+1} = D^{-1}(E + F)x_k + D^{-1}b$

The termination criteria of the Jacobi method is given by the i-th component of the residual vector $(b - A * x_{k+1})_i = 0$. If the result only has to be an approximation, the difference can have a tolerance. The implementations of the test applications in this thesis use $||b - A * x_{k+1}|| < \epsilon$, where ϵ defines the precision of the approximation. Also there is the possibility to terminate at a maximum of Jacobi iterations.

The Jacobi method converges for all equation systems, that have a square matrix and a spectral radius less than 1, i.e. $SpecRad(D^{-1} * (E + F)) < 1$. Out of this, if A is strictly diagonally dominant or an irreducibly diagonally dominant matrix, the presented Jacobi method converges

for any vector x [Wol94]. Algorithm 2.1 shows the code to calculate a linear equation system with the Jacobi method.

3 Related Work

In this chapter projects are presented, that share related approaches utilizing Cloud-Infrastructure to compute high sophisticated and resource demanding problems on mobile devices. Definitions for cloud computing and code offload are also provided in this chapter.

3.1 Cloud Computing and Code Offload

Conventional applications are executed completely on the running device, with local resources. As a contrast to that, there are techniques that enable the execution of applications remotely on another machine [Edu10]. Using the remote technique, the application can profit from the resources of the remote machine, which might be more powerful than the local computer device. The term Cloud Computing thence consists of applications delivered as services over the Internet and hardware and systems software that provide those services in a datacenter [Mic09]. The applications that are delivered as services over the Internet are referred to as Software as a Service (SaaS) [Mic09]. Those services run on datacenter hardware that is provided over a network, meaning the cloud. When using Cloud Computing, an application is executed completely remote. Another attempt is code offloading, meaning that only specific code parts of the applications, mostly resource demanding, are migrated to the cloud [Chr14]. The cloud then cares for the best execution method and returns the results to the mobile device.

The term mobile-cloud-computing can be defined as an extension of Cloud Computing, when at least parts of the foundational hardware consists of mobile devices [Eug09].

3.2 Virtual Cloud Computing Provider for Mobile Devices

While mobile devices have an imersive growth in our society, mobile applications gain more relevance in our everyday life. What goes with this, is that the mobile applications get an increasing grade of complexity [Sok12]. This results in a high consumption of resources of the device and thus additionally in battery energie consumption [Sok12]. As mobile devices have to be portable, they can not be equipped with big and heavy components. Hence mobile devices are not that powerful as their stationary relatives, like desktop computers or servers.

Specific methods had to be developed, to satisfy the demands of complex mobile applications. While mobile devices like laptops and smartphones are able to connect to mobile or local networks, it is possible to distribute or completely move the computation of the mobile device to other devices. Based on this, some attempts use parallelization to distribute the computation of applications on multiple devices [Chr14]. Such a system was developed by Gonzalo et al. [Gon10]. Their motivation was to develop a system that uses the fact of the pervasiveness of mobile devices in our current society. They presented a use case, where a person wants to execute a resource demanding application on a mobile device, that can not compete with those resource needs. Given the pervasive presence of mobile devices, their system distributes the execution on the fly to all mobile devices that are located in the same area or follow the same movement patterns. An obvious problem that goes with that, is that the distribution to mobile devices with the same or less performance seems not to be beneficial because of the communication overhead [Gon10]. Also there is the question for providers, why they should share their resources with others, as it costs them additional energy. The solution for that according to Gonzalo et al., is to find mobile device users that do the same tasks on the current area. An example is an optical character recognition (OCR) translation application that is used in a museum, or persons downloading a P2P file at a university [Gon10]. While there is a place where many mobile device users share the same interests, this would in fact be profitable for all users to share their resources. The drawback of this attempt is the communication overhead, because parallelization requires very frequent state exchanges [Chr14]. While testing the cloud computing framework, Gonzalo et al. recorded that about 44% of the execution time is used by offloading preparation and waiting for results [Gon10]. Also the test results show, that the distributed approach is about 1% slower than a local execution. Though they aimed to improve this in future work.

3.3 Energy Savings with Code Offload

A further technique that utilizes Cloud-Infrastructure is called code offload. Compared to the parallelization attempt, code offload is more efficient in the context of communication overhead. This advantage is used by related work, e.g. to safe energy on mobile devices.

There are many different techniques to safe energy on a mobile device. Examples are optimizations on the IO, turn off the screen when it is not needed or slow down the CPU [Zhi01]. But all of those techniques can influence the user experience or restrict the usage of applications on the device. Also, most of the energy saving attempts considered laptops instead of handhelds [Zhi01]. The better attempt to prevent disadvantages of conventional techniques, is code offload. If a mobile application uses the code offload technique, the offloaded parts will not be executed on the mobile device. As a consequence the energy that would be consumed with that part of code, is saved on the mobile device without restrictions for the user. The only power that is consumed by this, is the power to send the code with its parameters and to receive the results.

3.3.1 MAUI

This advantage is for example used by the MAUI system [Edu10]. MAUI enables a fine-grained energy aware code offload. With MAUI it is easy to adapt existing programs, because of the simple mechanism to add annotations to code that should be offloaded. A special optimization engine makes MAUI to decide at runtime, whether to upload the code or to execute locally. This reduces the effort for the programmer and takes his decision whether to upload the code or not. That is an important advantage, because the code offload is not always better than a local execution. It should be considered that the network connection can vary over time. That means if a method is offloaded once, there may be later a point of time, where it is better to execute the method locally. That is why MAUI has a profiler for energy costs [Edu10]. The profiler measures the devices' energy consumption characteristics at initialization time. Further factors of the profiler are the program characteristics, like running time and resource needs, and the network characteristics. A solver then interprets the statistics of the profiler. Based on this, MAUI decides whether to offload code or not before each method invocation that is annotated with MAUI annotations. While MAUI can be used to offload code, it is not always possible or beneficial to offload specific code. Parts that should not be offloaded, are e.g. user interface code, mobile device IO or code that interacts with external components that would be affected by re-execution [Edu10]. MAUI was tested with three applications: face recognition, an arcade game and a voice-based language translation. The test results show, that MAUI saves energy on applications with intensive calculations. The bests results came out when using WiFi as network connection [Edu10]. WiFi is measured to be three to five times more energy efficient than 3G for using code offload [Edu10].

By contrast, the ThinkAir system of Kosta et al. measured more energy savings when using 3G instead of WiFi [Sok12]. Their explanation is, that when an application uses larger data to transfer, 3G has advantages against WiFi. I.e., WiFi is less energy efficient per bit transmitted than 3G [Sok12].

3.3.2 Dynamic Software Deployment

A similar system to MAUI was developed by Giurgiu et al. [Ioa12]. This system also aims to improve the power consumption and performance on mobile devices by deploying code of mobile applications into cloud infrastructures. However the focus of this system aims the dynamic adaption of code offloading. Familiar to MAUI, it continuously profiles the performance of the application and updates its deployment according to the results. While MAUI uses a static attempt to monitor the device characteristics at initialization, the system of Giurgiu et al. monitors the network and device characteristics, such as CPU load and available storage space, continuously at runtime. That can be important, when the user of the mobile device runs multiple applications at a time. Therefore the computational resources can vary over time. Another point is the user input for an application. A mobile application may not

always be used in the same way, hence the benefit of offloading code can vary, too. With the system of Giurgiu et al., energy savings of about 45% could be reached against traditional approaches [Ioa12].

Both systems, MAUI and the system of Giurgiu et al., had similar findings in monitoring the network status. As there are tools to monitor this status, both systems rejected to use them, because the tools are unnecessary complex. Instead they send ping messages to the servers and wait for the results [Edu10][Ioa12]. This finding explains, that too much monitoring of the system and its environment can reduce the energy savings. Hence it has to be balanced between a high sophisticated profiler and the energy and resources that are consumed by logging the statistics to the profiler.

3.4 Performance Gain with Code Offload

A beneficial part of code offloading is the possibility to move resource demanding code into the cloud. The computers in the cloud, often servers, are in most cases more powerful than the mobile device from which the computation starts. According to this, when using code offload to safe energy, one of the side effects is that the performance increases [Sok12]. This effect was also discovered when using the MAUI system. The application test results show, that MAUI allows to more than double the screen refresh rate of latency sensitive applications [Edu10]. This advantage was also seen in the system of Giurgiu et al., where the system could register 75% of performance gain against traditional approaches without code offload [Ioa12]. That means that the code offload approach enables the development of applications, that need more resources than the mobile device can provide [Edu10].

3.4.1 ThinkAir

Another system called ThinkAir by Kosta et al. [Sok12], aims to improve the execution speed of mobile applications and safe energy at the same time. Their approach not only uses code offload to reach these goals, furthermore they provide on demand resource allocation with virtual machines. That means there is not only one server to compute the offloaded tasks, but any desired amount of machines to execute the code parallelized. The allocation of virtual machines happens dynamically. One machine, the primary default server, is always on. The secondary dynamically scaled power server gets cut in if the primary server detects that the application needs more resources. The obvious drawback of this approach is, that the starting of a virtual machine takes time. But some tasks take much longer time, sometimes hours, so that this time difference gets compensated. This leads to the conclusion, that this approach only makes sense if the application is very complex and resource demanding. These findings are stated by the test results of ThinkAir on different applications like face detection programs, the n-queens problem, a virus scanner and a picture merger [Sok12].

3.4.2 CloneCloud

Familiar to ThinkAir is the CloneCloud [Byu11] system which provides application virtual machines for mobile applications in a cloud. The difference to ThinkAir is, that CloneCloud aims to migrate whole execution threads instead of single functions. Also there is no need to modify the source code of the application with annotations like in MAUI. CloneCloud automatically checks with its static and dynamic analyzing whether to migrate and later re-integrate a thread or not. The test results of CloneCloud show a meaningful improvement of up to 20 times faster application execution against traditional approaches without cloud computing proceedings [Byu11].

3.5 Communication Link Handling

While the previous presented methods address the lack of battery capacity and resources on a mobile device, one important limiting factor is often handled only rudimentary. The limitations of the network can turn out as the bottleneck of the code offload and cloud computing methods. Temporary communication link failures are not an exception, but furthermore the rule [Flo14]. According to Ding et al. [N. 13], over 80% of Android smart phones have only poor signal strength for over 15% of their active usage time [Flo14]. That is due to the frequent movements of smart phone users between areas with different signal strengths [Flo14]. An example would be a person, using the subway or go shopping in different buildings. Always online would be the best condition for code offloading systems. But as described, there are situations where the network is temporary not available or the signal strength is very low, which causes high latency. If those connection failures are not handled in a proper way, the cloud attempt can be slower and use more energy than a local execution. The MAUI system for example, detects failures with simple timeout mechanisms. The reaction to a failure, is to find another MAUI server in the network or to re-execute the code locally [Edu10]. Both reactions are not the best way to handle disconnects, as the attempt to search for an alternative MAUI server in the network costs energy and time. If no server is found, the local re-execution occurs. But this is also not the best solution, because all the progress that is made till the disconnect, is lost.

3.5.1 Preemptable Code Offloading

A better way presents the attempt of Berg et al. [Flo14], which has the goal to minimize energy consumption and increase the application robustness on temporary disconnects. They focus on communication link failures by using preemptable code offloading. That means, their system partitions the application in a way, such that it runs with partial results [Flo14]. They implement this by creating safe points and transmit intermediate results form the server to the mobile device. If a link failure occurs, the mobile device can continue from the state of the

last transmitted safe-point. An important question that the developers had to discuss, is the amount of safe points. Too many safe points can generate too much communication overhead, while too few safe points can cause to much progress loss on a disconnect [Flo14]. Berg et al. implemented the handling of this decision with an offload compiler, that inserts break points before methods that are offload candidates. An offload controller decides whether to offload a method with a break point or not. The safe-point generator is responsible for efficient safe point creation, based on the offloaded code [Flo14].

They evaluated their preemptive code offloading (PreOff) approach against two basic non-preemptable code offloading approaches. The other approaches are BasicOff-ReExec, which means that a local re-execution is initiated as soon as a disconnect occurs, and BasicOff-Wait, which waits for a re-connect to fetch the result of the server. The AllLocal approach, meaning a local execution on the device, is also considered for comparison. The results show, that the efficiency of the different approaches depend on the current communication. In a situation where no link failures occur, both basic non-preemptable approaches are slightly more efficient. But as soon as only one disconnect occurs, the PreOff approach is faster and consumes only 3.5% more energy than the BasicOff approaches [Flo14]. Also the AllLocal approach only consumes less energy than PreOff, when the execution time is very short (a few seconds). Therefore the most efficient approach is PreOff [Flo14].

3.6 Conclusion of the Realted Work

The systems and proceedings presented in this chapter are only an insight into the wide range of available attempts or past projects. All introduced approaches and methods in this chapter conclude, that code offloading proceedings improve an application execution on a mobile device with regards to less energy consumption and faster computation. But there are situations, e.g. on communication link failures or for very short calculations, where a completely local execution is more efficient. Thus the efficiency of a code offload approach depends on the usage scenario of the mobile device user. But with respect to the various situations in which the mobile devices are used, code offload techniques or cloud computing systems are in most cases more efficient than a local execution.

4 Pre-Tests

This chapter describes the pre-tests and presents the according results. It is expected, that solving linear equation systems on stationary computer devices is faster than solving them on mobile devices. It is additionally anticipated, that iterative approximation algorithms can be adapted in various ways to adjust the result quality and execution time. Therefore, two different classifications of computer devices were taken as a base for the measurements. The stationary computer devices like desktop computers or servers, and mobile computer devices like smartphones or laptops. The detailed specifications of the test devices are described in Chapter 10 and the implementation of the tests are described in Chapter 7. The tested algorithms are: Jacobi method, conjugate gradient method, SYMMLQ and decomposition solver.

To evaluate multiple different linear equation systems, random generated equations are provided as input for the algorithms. The algorithm created a random $n * n$ matrix A with property: $\forall a \in A : |a_{i,i}| > \sum |a_{i,j}|$ for $i \neq j \Rightarrow SpecRad(A) < 1$. Further, a random vector x, with length n, was created and a result vector b was computed, with $b = A * x$. The input parameters for the presented algorithms are a matrix A and an equation vector b.

4.1 Linear Equation Solver Measurements

The first measurement shows the speed differences between the solving approaches for different matrix sizes on the same computer device. The exact solver always computes until the exact result is retrieved, whereas the iterative approximation algorithms use termination criteria. The termination can either be caused by a maximum of iterations or by a residual, as it is described in Section 2.2. The test device is a stationary computer (desktop computer) with a four core CPU and a clock frequency of 3 GHz. The random-access memory (RAM) amounts 8 GB. During the test runs, all applications apart of the test program were closed to prevent interruptions or result variances.

4.1.1 Conjugate Gradient and SYMMLQ Scales

The tests in this subsection measure the computation time (y-axis) of the conjugate gradient method and the SYMMLQ solver for equations with a matrix dimension between 100 and 1500

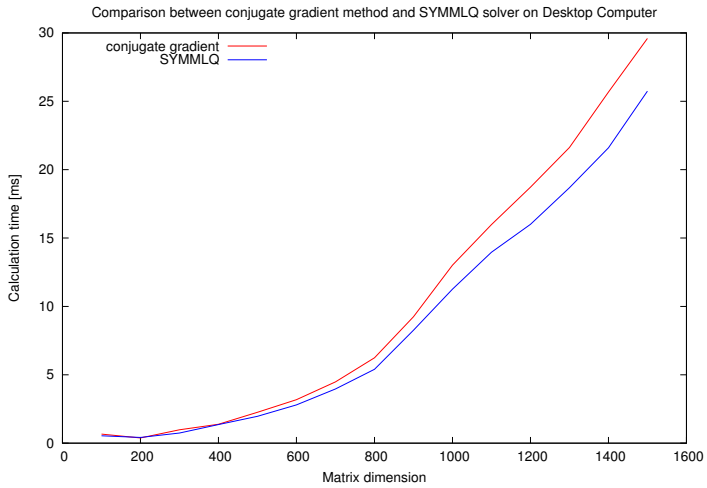

Figure 4.1: Matrix scale for conjugate gradient method and SYMMLQ solver on Desktop Computer

(x-axis). It is expected, that both iterative methods are similar fast and are overall faster than the exact solver. The termination criteria in this test was set to a maximum of 100 iterations and a residual of 10^{-9} for both algorithms.

The test results can be seen in figure Fig. 4.1. Both proceedings, the conjugate gradient method and the SYMMLQ solver compute an equation with a matrix dimension of 1500 in under 30 milliseconds. The speed difference between both methods increases while the matrix dimension scales up. For matrix dimensions less than 800, the SYMMLQ solver is less than one millisecond faster than the conjugate gradient method. At a matrix dimension of 1500, the speed difference amounts four milliseconds.

4.1.2 Jacobi Method and Decomposition Solver

As the self implemented Jacobi method and the decomposition solver are expected to be slower in computation speed compared to the conjugate gradient method and the SYMMLQ solver, they were tested separately. Additionally, it is anticipated that the decomposition solver is the slowest method under the tested algorithms, as it is the only exact and non iterative solver. The test results in figure Fig. 4.2 compare the computation speed in relation to the matrix dimension for the Jacobi method and the decomposition solver. The termination criteria for the Jacobi method are set to a maximum of 100 iterations and a residual of 10^{-9}. The decomposition solver has an exponential growth of computation time while the matrix dimension scales up. Equations with a matrix dimension of 500 are solved in 101.32 milliseconds. For a three times bigger equation with dimension 1500, the computation takes 6290.82 milliseconds. In

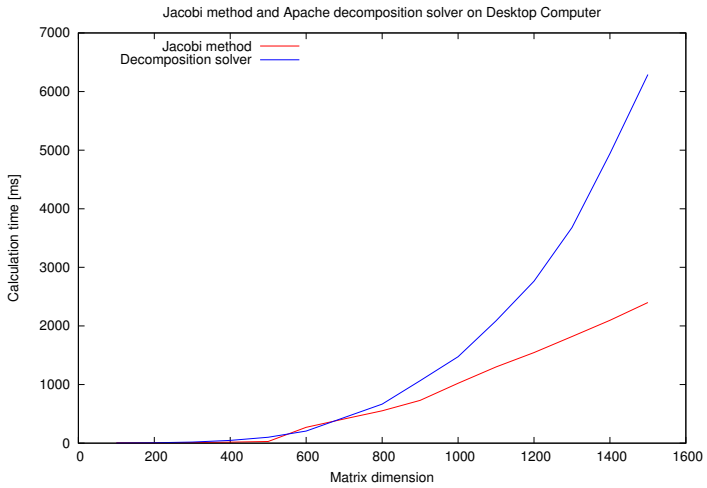

Figure 4.2: Matrix scale for Jacobi method and decomposition solver on Desktop Computer

comparison to the decomposition solver, the Jacobi method has a smaller growth rate. For an equation with dimension 500, the Jacobi method terminates in 29.0 milliseconds, while an equation with dimension 1500 terminates in 2400.1 milliseconds. Both methods, the Jacobi method and the decomposition solver are considerable slower in computation speed than the conjugate gradient method and the SYMMLQ solver. Although as expected, all iterative algorithms are faster than the exact solver.

4.1.3 Jacobi Method Studies

The Jacobi method in figure Fig. 4.2 has an inconsistent growth rate. Though, the spike of computation time between the matrix sizes 500 and 600, is not a single phenomenon. Every test run for the Jacobi method with a termination setup of 100 iterations and a residual of 10^{-9} records the same spike between the dimensions 500 and 600. Hence the next tests consider different termination criteria, as it is expected, that the termination setup influences the spike occurrence. The test runs use four different residuals: 10^{-10}, 10^{-9}, 10^{-8}, 10^{-1}. Figure Fig. 4.3 depicts the results of this test. For an accuracy of 10^{-9} the spike occurs between the matrix dimensions 500 and 600, whereas for an accuracy of 10^{-10} the spike already occurs between the dimensions 100 and 200. The same spike occurs for an accuracy of 10^{-8} at a matrix dimension between 2000 and 2100.

Figure Fig. 4.4 depicts the number of Jacobi iterations for the test runs with different residuals. This provides a more detailed analysis of the test iterations. As can bee seen in the diagram at matrix dimension 500, the number of iterations rises from 13 to 100, for a residual of 10^{-9}. The explanation for this is, when checking the residual termination $||A * x_i - b|| < \epsilon$, the

norm does not change at a given point. This is due to rounding errors, which means for a specific ϵ and a specific matrix dimension, no difference can be calculated between the norms $||A * x_i - b||$ and $||A * x_{i+1} - b||$. Therefore, the reason for the spikes in calculation time is, that the residual does not terminate the algorithm. Instead the second termination criterion occurs, meaning the maximum of 100 Jacobi iterations.

Figure Fig. 4.5 shows the Jacobi iteration growth rate for the residual termination criteria $10^{-8}, 10^{-7}, 10^{-5}, 10^{-1}$. The iteration termination is disabled for this test. With the information of this diagram, two options are available to prevent the spikes. As can been seen in the diagram, the bigger the matrix and the smaller the residual, the more Jacobi iterations are needed to calculate the result. Hence, to prevent the spikes, it should be considered to either reduce the maximum of Jacobi iterations, or to add a third termination criterion. The third termination criterion saves each result of the norm $norm_i = ||A * x_i - b||$ and compares it to the previous norm. If $norm_i = norm_{i-1}$ the algorithm has to terminate, as the result will not get a higher accuracy.

The difference between the iteration and the residual termination can be seen in figure Fig. 4.6. The iteration termination has a maximum of 100 iterations while the residual termination is set to 10^{-9}. This time, for the residual termination the third termination criterion is added to prevent infinite Jacobi iterations. There is a significant difference between the computation times of both methods, as the residual termination has a much slower growth rate.

The findings of the pre-tests state, that iterative algorithms can be adapted in various ways which has different effects. One effect is, that the termination criteria adjust the accuracy of the result. E.g., for a residual of 10^{-9}, the error is about 10^{-13}. The error value means the norm of the difference between the real x vector and the approximated vector x'. The second effect is, that the termination criteria additionally control the execution speed of the algorithms. The less accurate the result has to be, the faster the computation terminates. Additionally, if algorithms like the Jacobi method starts with an estimation of the x vector, the number of iterations can be reduced, as the residual will be reached earlier. If an error value is acceptable for the result, the iterative approximation algorithms are more efficient than exact solving methods.

4.2 Device Comparison

Now that we have analyzed the algorithms on a stationary computer, the next tests consider measurements on mobile devices and comparisons between stationary computers. The algorithms on the mobile devices are expected to terminate slower than on the stationary computers. The mobile device for this tests is a Nexus 5 smartphone with Android version 5.0.1. It has a four core CPU with a clock frequency of 2.26 GHz and 2 GB RAM. A further mobile device is a laptop with a four core 1.6 GHz CPU and 8 GB RAM. The stationary devices are the previously tested desktop computer (4 core CPU with 8 GB RAM), the Curium Server

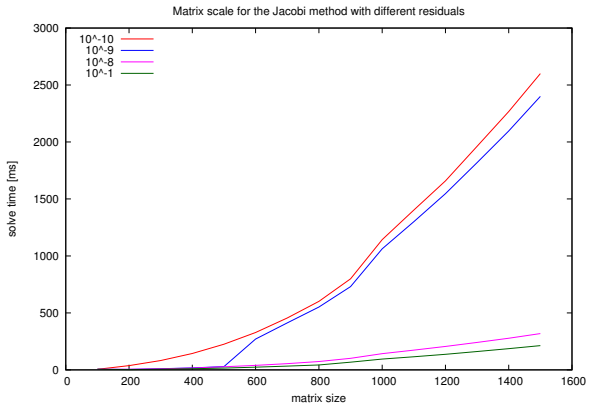

Figure 4.3: Matrix scale for the Jacobi method with different residuals on Desktop Computer

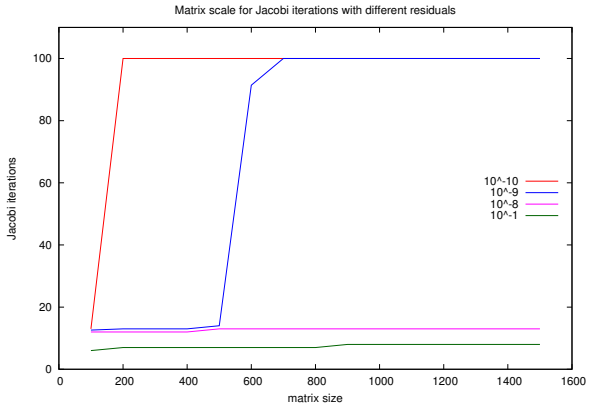

Figure 4.4: Matrix scale for Jacobi iterations with different residuals on Desktop Computer

(8 core CPU with 32 GB RAM) and the Kepler server (32 core CPU with 128 GB RAM). First we test if the different algorithms have equal speed differences on the mobile device and on the Desktop computer. The algorithms are implemented in Java and are therefore equal on every test device. The computer device for the first tests is the Nexus 5 smartphone. All applications except of the test program were closed and the flightmode was turned on during the tests. Figure Fig. 4.7 shows the conjugate gradient method and the SYMMLQ solver, while figure Fig. 4.8 shows the decomposition solver along with the Jacobi method. The results state, that the conjugate gradient method and the SYMMLQ solver are again the fastest solving methods, compared to all tested algorithms. The non convergence of the Jacobi method, e.g. for a matrix size greater than 600 and a residual of 10^{-9}, occurs on the mobile device, too. Hence as assumed, this phenomenon is not device dependent. Thence the Jacobi method is again viewed with either iteration termination, or residual termination along with the third termination

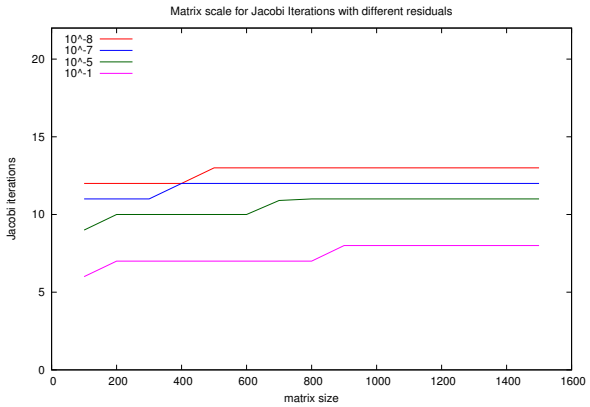

Figure 4.5: Matrix scale for Jacobi (small epsilon scale) iterations with different residuals on Desktop Computer

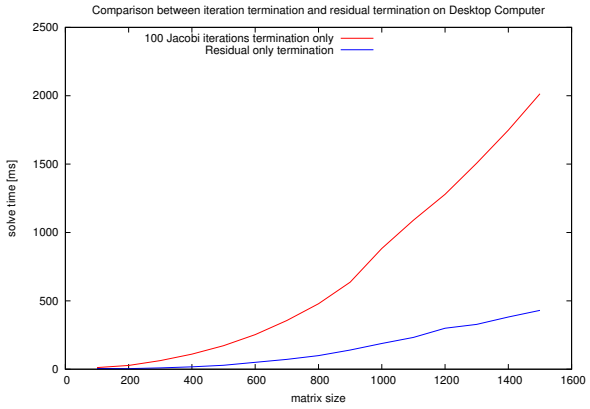

Figure 4.6: Matrix scale for Jacobi method with iteration and residual termination only, on Desktop Computer

criterion as mentioned before. When looking at the Jacobi method, both termination criteria are faster than the exact solver when the matrix dimension scales up. Although for a matrix dimension $x \in [2, 800]$ and using 100 iterations as termination criterion for the Jacobi method, the decomposition solver is faster. The residual termination was set to an accuracy of 10^{-9}, which is always faster than the decomposition solver. This shows that on the mobile device, nearly the same proportions occur between the algorithms.

When comparing the computation times between the stationary desktop computer (Fig. 4.1, Fig. 4.2) and mobile phone Nexus 5(Fig. 4.7, Fig. 4.8) for the test algorithms, it can be seen that the stationary computer is significant faster compared to the mobile device. E.g. the computation time for the Jacobi method with 100 Jacobi iterations amounts about 2.5 seconds

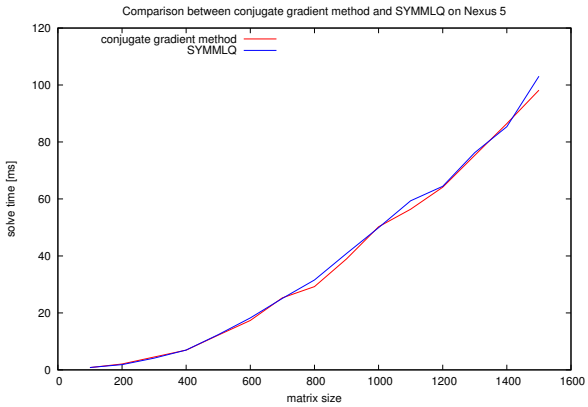

Figure 4.7: Matrix scale for conjugate gradient method and SYMMLQ solver on Nexus 5

on the desktop computer. In contrast, the computation time on the mobile device amounts over 5 seconds. Also the conjugate gradient method on the mobile device is about four times as slow as on the desktop computer. This states the hypothesis, that migrating the computation procedures of solving linear equation systems to stronger computers is profitable in the context of computation time.

Figure Fig. 4.9 shows the difference between the computation times of the conjugate gradient method on the test devices: Nexus 5, Curium, Desktop, Laptop and Kepler. This test scaled the equation dimension from 100 to 4500 with a step size of 100. The diagram shows a drop of calculation time at a matrix size of 3100 on all devices. The reason is, that the conjugate gradient iterations decrease when the matrix dimension increases. At a matrix size of 3100, the algorithm reduced the iterations from seven to six which causes a faster execution. This is due to the update residual implementation of the algorithm and is necessary to prevent rounding errors for the conjugate gradient method [Str02].

Against the hypothesis that executions are always faster on stationary devices compared to mobile devices, the conjugate gradient method executed second fastest when running on the laptop. The speed difference compared to the execution on the Kepler server amounts only three milliseconds. Although the laptop is a mobile device, it has beneficial specifications to solve linear equations. This concludes, that a mobile device has not necessary to be less powerful than a stationary device. Thus, a migration of the equation computation only makes sense, if the mobile device has weaker resources than the stationary device.

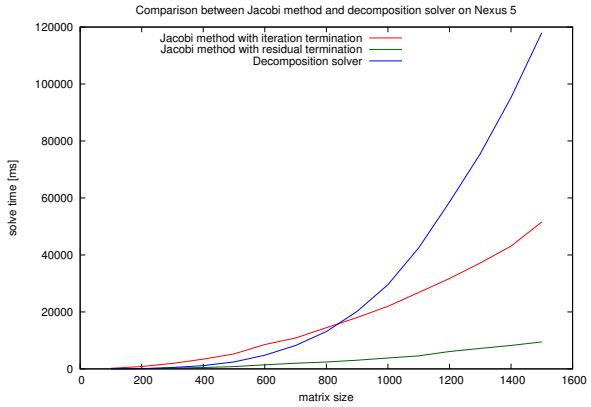

Figure 4.8: Matrix scale for Jacobi method and decomposition solver on Nexus 5

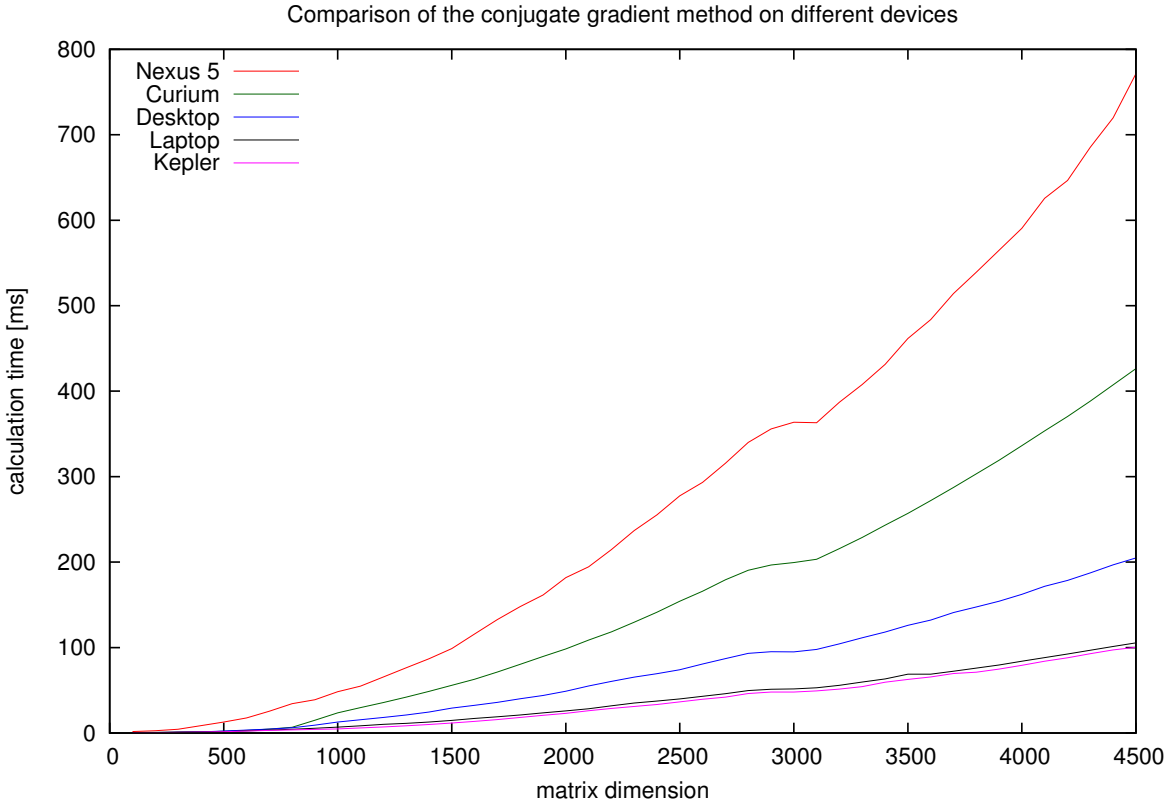

Figure 4.9: Comparison on different computer devices for the conjugate gradient method

5 System Model

This chapter presents the system model to efficiently solve linear equations utilizing a Cloud-Infrastructure. Section 5.1 presents an abstract view about the components of the system. This provides a better understanding about how the components interact with each other and how they work. The requirements for an efficient solving are addressed at the end of this chapter in Section 5.2.

5.1 Architecture and Assumptions

This section describes the system model and according assumptions about it, as it can be seen in figure Fig. 5.1. The two main components are the mobile device and the Cloud-Infrastructure. Those components communicate via wireless network techniques with each other and provide an efficient proceeding to solve linear equations with square and diagonally dominant matrices. There are two execution methods to solve linear equation systems. On the one hand is the local execution, which defines an approach to calculate the equations on the mobile device by using its local resources. On the other hand, the remote approach uses the communication to the Cloud-Infrastructure to distribute the calculation.

5.1.1 Mobile Device

The mobile device in this system is seen as a battery powered device with interfaces to connect to wireless networks. It runs an application that solves linear equation systems. Therefor the mobile device requires a middleware, to initialize the computation of equation systems. Depending on the current situation, the middleware decides whether to solve the equation on the mobile device or to distribute the computation in the cloud. To help making this decision, the middleware has to provide a component that monitors the executions and creates statistics. For local executions, the middleware has to provide algorithms that solve linear equations locally. Note that the decision process does only consider the network situation and not the situation of the mobile device. Utilization of the CPU or varying memory capacity can impact the local execution. For this system model it is assumed, that the CPU utilization and the memory capacity are constant. The resources on the mobile device are the CPU performance, memory capacity and the battery power. It also is assumed, that those resources are not as powerful as the resources on the servers in the Cloud-Infrastructure.

5.1.2 Cloud-Infrastructure

In the cloud, multiple servers can provide their resources to solve resource demanding linear equation systems. A server needs to handle incoming requests and initiate the solving of the received equations. The server also has to provide a backup of every computed result, in case the connection to a client device is lost. The backup is also important if the mobile client wants to pull interim results of running computations. After a specific time range, the results may be deleted to prevent unnecessary memory occupation. The equation solver on the server can be equal to the counterpart on the mobile device. Although it is not necessary to use the same algorithms as on the mobile device, as long as the interface is equal.

On stationary computers, the considered resources are only the CPU performance and the memory capacity. There is no need to consider battery power, as the cloud computers are connected to a power grid. Also the resources should be more powerful as the resources on the mobile devices in this system model.

5.1.3 Connection

The connection of the architecture is based on wireless network techniques. The reason is, that the mobile devices are considered to be moved around between areas with different networks. The connection is initiated and monitored completely by the mobile device. The server just has to be available and handle incoming requests. The network quality situation is defined as good, if the network provides a high throughput and low latency. A bad network quality means, that there is high latency or a low throughput. A network is unstable, if the connection is only temporary available or not available at all, whereas a stable network connection is permanently available. Test messages can be used to monitor the network status. They are defined as messages to test the round trip time (RTT) in the network. Therefor it measures how long it takes for a message to be sent from the mobile device to the server and back again. For connection oriented transmission protocols like the transmission control protocol (TCP), the connection establishment time, like of the three way handshake, can be measured instead of sending a test message. The size of the test messages should only amount a few bytes. Even though bigger messages would provide a more precise network state, small messages do not cause a high communication overhead and save time. This is more important than a precise network state, as the network state can vary quickly.

5.1.4 Linear Equation Systems and Algorithms

The linear equation systems that have to be solved are of the type $A * x = b$. The given values are the diagonally dominant and square $n * n$ matrix A and the equation result vector b with length n. The searched value that has to be computed is the x vector. Therefore the algorithms

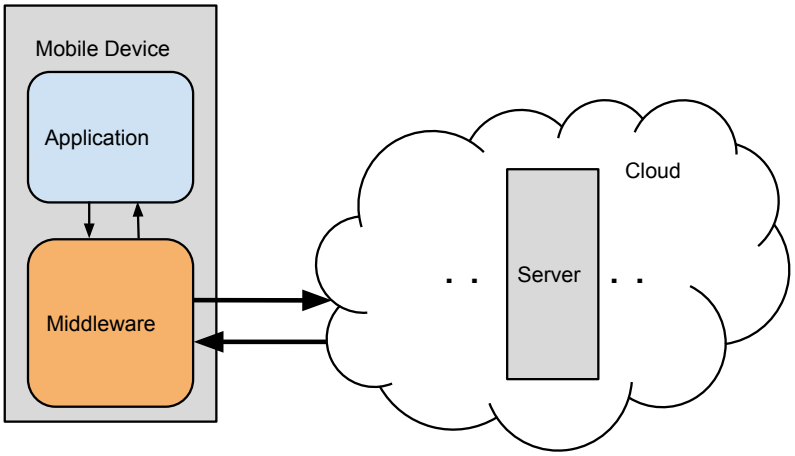

Figure 5.1: System model of the mobile device and the cloud infrastructure

that solve those equations need at least a matrix A and a vector b as input. The output is the x vector with length n. As the algorithms are iterative and do not have to be computed straight, an estimation of the x vector can be provided as input parameter, too.

5.1.5 Energy Model

Executions of tasks on the CPU consume energy. The more sophisticated the task, the more energy is consumed. Hence the calculation of a linear equation system utilizes the CPU capacity highly, which leads to high energy consumption. The network communication between the mobile device and the cloud is also a high utilizing CPU task, but compared to the solving of an equation it lasts shorter. Hence the hypothesis is, that a remote execution is saving energy against a local execution.

5.2 Requirements

The requirement to this system model against a typical local approach is, to optimize the three resources energy, computation speed and result quality. If the network connection has a good quality and is stable, the system model aims to optimize all resources. If the network quality is decreasing or the connection is unstable, the priority switches for the benefit of computation speed. The efficiency of the system model is defined by the balance between those resources according to the current network situation.

Some resources harmonize with each other, while others exclude each other. It depends on the current network situation. If the network situation allows a remote execution, such that it is faster than a conventional local execution, the resource computation speed would be satisfied. Additionally, the energy that would be consumed by a local execution is saved on the mobile device, because the calculation is performed in the cloud. In this situation, the resource energy is optimized, too. Unfortunately not every situation allows the optimization of multiple resources. In specific situations, in context of the network, resources can exclude each other. There are two types of exclusions, the situation dependent and the situation independent. The following argumentations show that it is not possible to optimize all resources equally in every situation, as they can get in conflict with each other.

5.2.1 Situation Dependent Exclusion

The best situation for a fast remote equation calculation with a specific matrix dimension occurs, when the network has a high throughput, low latency and is stable. Let us assume that a remote equation calculation under the best network situation is faster than a local calculation on a mobile device. Also assume that the network connection quality between the mobile device and the cloud infrastructure is varying, which means there is no guarantee that the remote calculation is faster than a local calculation. Therefore the fastest approach to solve the equation system on a mobile device with cloud infrastructure is provided, if the calculation is computed locally while trying to solve it remotely at the same time. The faster execution of both then returns the result and terminates the other attempt. This prevents the remote attempt from delaying the calculation due to high latency in the network or connection losses, because the local computation does not depend on the current network statistics. The drawback of the approach with focus on the fastest calculation is, that it is not energy efficient. Due to the parallel execution of the calculation on both, the mobile device and the cloud infrastructure, no energy is saved on the mobile device compared to a completely local execution. On the contrary, this consumes even more energy than a completely local execution, because the communication overhead between the mobile device and the cloud infrastructure does also consume energy. The most efficient execution of the calculation to safe energy is, when at most everything is calculated in the cloud. This means, that it is not possible to optimize the resources of fast computation speed and energy saving on the mobile device for every situation.

5.2.2 Situation Independent Exclusion

An example for situation independent resource exclusions is, optimizing the resources for result quality and execution speed. The iterations of iterative approaches like the Jacobi method increase if the residual value decreases. I.e., a calculation with a residual of 10^{-9} takes longer than a calculation with a residual of 10^{-1}, as it can be seen in Chapter 4. When trying to reach

a high result quality, a small residual would be chosen. Let us assume high result quality is defined for a residual of 10^{-9}. When trying to reach a fast computation, a residual of less than 10^{-9} is more profitable, because less Jacobi iterations are performed. This leads to a contradiction where the consequence is, that the resources for fast computation and high result quality can not both be optimized as effectively as possible.

6 System Design

This chapter presents the design of the system model in Chapter 5, to efficiently solve linear equation systems utilizing a Cloud-Infrastructure. The design consists of the description of the mobile device components and the cloud server components. The component diagram in figure Fig. 6.1 shows the abstract architecture of the draft. Its components will be described in the following sections.

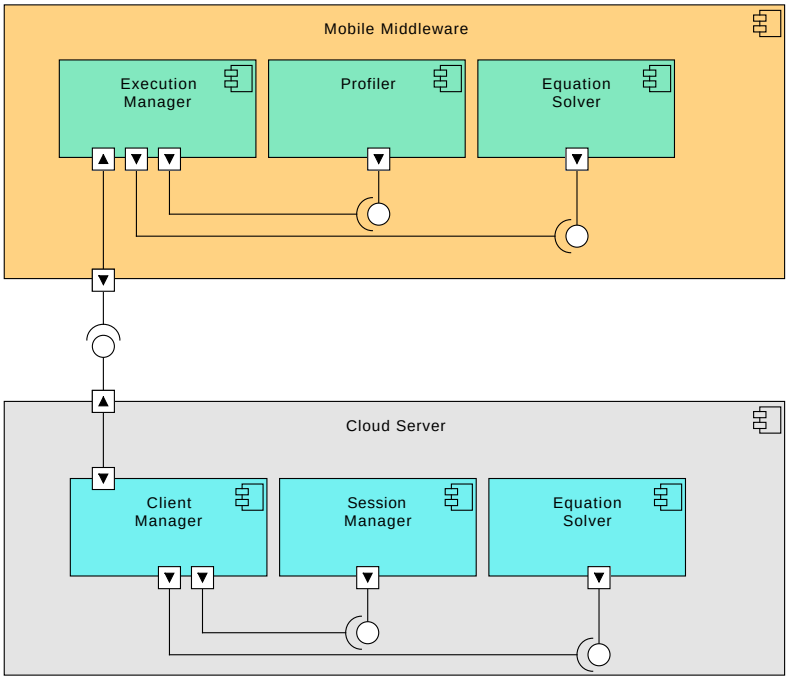

Figure 6.1: Component diagram of the mobile device and the cloud architecture

6.1 Middleware for the Mobile Device

The middleware initializes the solving on the mobile device. It uses a profiler, to decide if a remote or a local computation is more efficient for the current network situation and equation system. Based on the suggestion of the profiler, the execution manager is responsible to initiate either the remote execution or to start a local calculation. As the network situation can vary, the execution manager has to adapt to changes. For local calculations of equation systems, the equation solver provides iterative algorithms to solve the equation system on the mobile device.

6.1.1 Execution Manager

As shown in the requirements analysis, not every resource can be optimized equally in every situation. Although the middleware can distribute the best priorities according to the current situation, which can be defined as situation based resource priority handling. The focus lies on the situation dependent resource exclusions, as the independent have to be defined by the calculation initiator at initialization time.

The best results can be reached, if the network provides a fast and stable connection, as the two resources execution speed and energy saving are both optimized as effectively as possible. Thus, in the beginning the system aims to optimizie all resources, meaning to increase computation speed and safe energy while computing a high result quality. If the network quality is reducing or temporary connection losses occur, the protocol adapts the resource priorities in a way, such that the computation time does not increase considerable. The priority for computation speed is the most important to contain, as the middleware is intended to be used in real time applications on mobile devices. The computation speed can be contained, when reducing the result quality or reducing the attempts to safe energy on the mobile device. The result quality is reduced when the maximum of iterations or the accuracy of the residual is decreased for an algorithm. This can either be done at initialization time or by fetching interim results from running computations. Energy saving attempts can be stopped by concurrently starting a local and a remote execution.

Shared State

If the execution manager has to run a local execution concurrent to a remote execution, it is important that no progress on both methods is lost. That means, if one method has calculated an interim result of the equation, it should be passed to a shared state. In the case that the second execution method starts, it can use the current interim result as an estimation for the final result. This can reduce the computation time of, e.g. the Jacobi method as less Jacobi iterations have to be performed. Although it has to be considered, that the sharing of interim

results can cause regression, as results may be overwritten with less exact results. To prevent this, the current accuracy of the interim result has to be calculated before updating the shared state. It must only overwrite the shared state if $||b - A * x_k|| < ||b - A * x_{k-1}||$, for an equation $A * x = b$.

Triggers

The execution manager has to react to specific triggers. On a trigger activation, the execution manager should change the computation method in a way, such that the result is available as fast as possible for this situation. There are two different trigger types which depend on the network connection: timeouts and connection losses. The timeout trigger is used for test messages that measure the network RTT. If the execution manager records that a test message is not returned within the time the last test message was measured, the network quality decreased. In this case, it has to be considered to start a local execution besides the remote execution or to fetch an interim result.

On connection losses between the mobile device and the server, the execution manager has to start a local execution, as it is not predictable when the network will be available again and no interim results can be fetched. The mobile device can try to re-connect to the server in periodic time intervals. If the connection loss is due to a single network failure, the remote execution can still be faster than a local execution on reconnect. Then also interim results can be retrieved, but only if the equation system was already sent in the first remote execution attempt.

6.1.2 Profiler

As the efficiency for solving linear equation systems depends on the network situation, a component has to be responsible for deciding which execution method is the best at the moment. In cases, where the matrix dimension is too small or the network connection is very slow, it is not efficient to initiate a remote computation. E.g., for small matrix dimensions, the communication between the mobile device and the cloud infrastructure would consume more time and energy than a local execution of the calculation. The same applies if the network connection has a high latency or low throughput. The profiler has the responsibility to filter, in which situation it would be efficient to calculate on a remote machine. Although, the network can vary from time to time, hence it is not possible to provide an exact prediction for the most efficient execution. Therefore the profiler can only suggest estimations. To get better estimations, the profiler should monitor the process of past solving attempts. Meaning, it logs the times of remote and local calculation executions. Together with trivial network RTT tests, the profiler estimates how fast the calculation can be executed either remote or local. This is clearly not the most accurate analyzing method, as more factors, like the current bandwith,

wireless signal strength etc. could be considered, too. But a very accurate network analyzing method costs too much time and consumes additional energy such that the benefit is lost.

Monitoring Executions

The profiler measures the time from the initiation of a local or remote calculation, until the result is returned; if the calculation was successful. Additionally the network is monitored as well, to have an overview about the network situation. However in proceedings of related projects, the authors came to the conclusion, that it is not profitable to use high sophisticated network analyzing tools [Edu10][Ioa12]. Instead they used to send ping messages to measure the network quality. According to those findings, the profiler of the middleware in this system model uses time measurements of test messages. Those time values are connected to absolute remote execution times. This gives a more precise estimation and bounds the remote execution times to the network situation in which this time was achieved.

Adapting Time Measurements

The profiler tries to always adapt to the latest collected statistics. I.e. previous results should be overwritten by later results. Thus, single measured time values can influence the decision process of the profiler. E.g. if a remote execution time value for matrix dimension x is less than the time value for a matrix dimension y with $x > y$, the time value of dimension y is no more appropriate. In this case, both values have to be equalized because a computation with a bigger matrix dimension takes longer than with a smaller dimension. This is stated by the measurements of the pre-tests in Chapter 4. The reason for such time value variances is the varying network situation. I.e. the transmission of the equation can affect inappropriate time values for remote executions.

Decision Handling

Based on the collected statistics, the profiler provides a function to suggest whether to solve an equation remote or local. If a new computation is initiated, the profiler takes the dimension of the equation system and compares the times, that were achieved in past remote and local computations. The attempt that was faster in the past, will probably be faster in the future. But it should also be considered, that the network situation can vary and time results can get better or worse. Hence the profiler should test the current network situation and involve this into the decision process. Let us assume, that for a matrix with dimension $n*n$ the remote computation terminated slower in past executions than a local computation. So for the next calculation with a matrix dimension of $n*n$, the profiler would suggest a local execution. Although, to involve the current network situation, the profiler sends a test message to the server and measures the

round trip time (RTT). If the current RTT is faster than the RTT in the previous attempt, the profiler realizes that the network connection has improved. According to the better network quality, the profiler will suggest a remote execution. This prevents that single bad results in the profiler statistics block future remote executions. In a situation, were the past remote execution was faster than a local execution, the profiler will also suggest a remote execution. This is due to advantages against a local computation attempt. If the network situation is now worse than in the previous execution, the execution manager has to handle the situation change. Note that little variations in the RTTs can occur, therefore an offset should be provided to compensate those variations. It should be considered that the offset prefers a remote execution before a local execution, as the remote attempt can also safe energy against the local attempt. To prevent unnecessary complex constructs, in a situation where a local execution is faster than the RTT of past remote executions, the profiler suggests a local calculation. This only filters out very small equation systems.

6.1.3 Equation Solver

The equation solver provides algorithms to solve linear equation systems with an iterative approach. The advantage of the iterative algorithms is, that the result quality of the equation system can be adapted. The residual termination criteria or the maximum of iterations can be reduced or increased, if the network quality changes.

6.2 Cloud Server

Multiple servers can provider their resources in the cloud. In contrast to the mobile device, the server has just to be available and solve linear equation systems that are sent from mobile devices. The server does not adapt to changes in the network quality nor to disconnects from the clients. If a client loses the connection to the server, the server continues to calculate the equation system and safes the results; as long as the equation system was received successfully. The mobile device then can ask for the results of running or finished equation computations.

6.2.1 Client Manager

The client manager handles incoming connection requests. It has to provide at least two different request managements: a handling for solving a new linear equation system and another for fetching interim results or results of finished computations. If a request for solving a new equation system was received, the server distributes the job to the equation solver. When the result was computed by the solver, the client manager sends a response with the result of the equation system to the mobile device, if still possible. On the other hand, if a request

for an interim result was received, the result has to be retrieved from the session manager, if available, and sent to the mobile device.

6.2.2 Session Manager

The session manager has the responsibility to safe the results from initiated calculations on the server. Every calculation should be bound to a special identifier, so that requests from clients for interim results or finished calculations can properly be handled. Possible identifiers are e.g. mobile device numbers combined with session numbers for equation systems etc. .

6.2.3 Equation Solver

The equation solver on the server can be the same as the solver on the mobile device. However one difference is, that the server has no need to adapt to changes in the network. Hence it always solves until the most accurate result quality is reached, because the mobile device cares for timeouts. After every iteration of an iterative algorithm, the result has to be passed to the session manager, so that the server can respond to interim result requests from the mobile device.

7 Implementation

This chapter discusses the implementation details of the created applications and tests for this bachelor thesis.

7.1 Pre-Test Implementation Details

All measurement tests were implemented in Java. The reason is, that the mobile devices run Android as operating system (OS) and the main language for Android applications is Java. Thence all devices can use the same implementation to have the same conditions. There are no specific libraries for solving linear equation systems in the Java programming language. To solve this problem, the tests use the Apache Commons Mathematics Library [1]. The package org.apache.commons.math3.linear.* provides different solvers for linear equation systems, that were used for the tests. All algorithms of the Apache library do not have special dependencies on different third party libraries. As there are many different algorithms to solve linear equation systems, the tests only use a few algorithms as representative for two main techniques. That is either to solve the equation system exactly or compute iteratively until an approximation of the result is found. For the tests, one exact solver and three iterative approximation algorithms were used. The exact solver is the Apache implementation of a decomposition algorithm. The two other Apache implementations that were used in the tests are the conjugate gradient method and the SYMMLQ solver. A further iterative solver is based on the Jacobi Method and was implemented as described in Chapter 2. The tests use multiple runs for every test iteration. For the matrix scaling tests, every test iteration generated a new equation to prevent single spikes in the computation time.

7.1.1 Apache Commons Math Linear Equations Solver

The three used solvers of the Apache Commons Math library are described in the following. They solve an equation of the form $A * x = b$ for the x vector.

[1] http://commons.apache.org/proper/commons-math/

Decomposition Solver

In the tests the Apache LUDecomposition [2] solver is used to represent an exact linear equation solver. The algorithm uses the LUP-decomposition, which means that the matrix is decomposed into L (lower triangular), U (upper triangular) and P (permutation matrix) that satisfy: $P * A = L * U$. This implementation can only find an exact solution if the matrix is square and there is an exact linear solution of the equation system, i.e. one has $||A * x - b|| = 0$.

Conjugate Gradient Method

The Apache implementation of the conjugate gradient method [3] follows the template of Barrett et al. [R. 94] and provides an approximated solution of the linear equation system, where $r = b - A * x$. The default stopping criterion is $||r|| \leq \delta||b||$, where δ is a user-specified tolerance. The variable r is called the update residual, as it might differ from the true residual due to rounding-off errors. A further termination criterion can occur by reaching the maximum of iterations, which can be seen as an evaluation of the matrix-vector product $A * x$.

Symm LQ Method

A further iterative solver of Apache is the SYMMLQ [4] solver proposed by Paige and Saunders [C. 75]. The difference between the SYMMLQ solver and the conjugate gradient method is, that it does not need a positive definite matrix. Although it does a bit more work if the matrix is positive definite.

7.2 Implementation of the Android Application

To test the efficiency of the developed approach of solving linear equation systems, a sample application was implemented. The application was implemented for the Android platform and is thence written in the programming language Java. The sample application is called Linear Equation Cloud Calculator (LECC) and provides functions to solve equation systems of the type $A * x = b$ for the x vector. Figure Fig. 7.1 depicts the structure for the middleware on the mobile Android device. The main classes in this system are the ConnectionManager, the Profiler and the MatrixExecutionManager. The ConnectionManager provides all functions to handle the data transmission between the mobile device and the cloud server. The Profiler class is the according

[2] http://commons.apache.org/proper/commons-math/apidocs/org/apache/commons/math3/linear/LUDecomposition.html
[3] https://commons.apache.org/proper/commons-math/apidocs/org/apache/commons/math3/linear/ConjugateGradient.html
[4] https://commons.apache.org/proper/commons-math/apidocs/org/apache/commons/math3/linear/SymmLQ.html

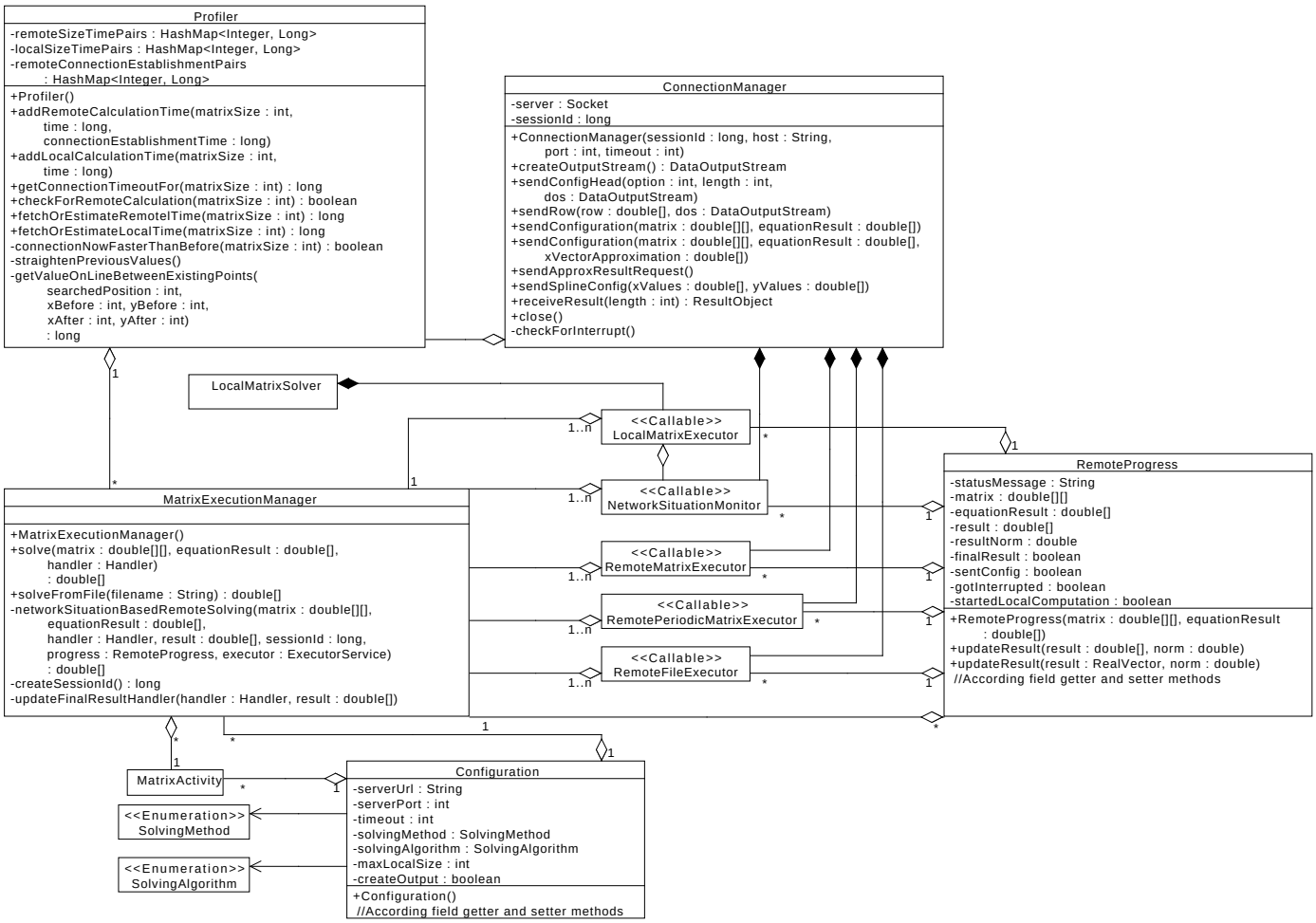

Figure 7.1: Class diagram of the mobile device middleware structure

implementation of the Profiler component and the MatrixExecutionManager the according implementation of the Execution Manager in the system design. A detailed explanation of the classes and the middleware structure is provided in the following subsections.

7.2.1 Middleware Structure

A new equation computation is initiated by the MatrixActivity component. The MatrixExecutionManager then executes the solving process according to the feedback of the Profiler and the current settings in the Configuration. The MatrixExecutionManager uses different

executors to start the calculation of equations. This is necessary due to the resource demanding process to solve an equation system. When computing the equations on the main thread, the application freezes until the result is available. Furthermore network connections have to be realized in background processes in the Android system, otherwise the application crashes. In first tests of the application, the AsyncTask class was used to implement the executors. It is a common way to implement concurrent tasks in the Android system. Also it provides an easy way to handle the communication between background processes and UI threads. The problem that goes along when using AsyncTask classes is, that according to its documentation it should only be used for short processes, at most for a few seconds. E.g. when rotating the screen of the application, the AsyncTask is no more bound to the latest UI. So the task is still computing but no updates on the UI will be received. This can cause memory leaks as unused tasks will not be garbage collected by the runtime, but contain running in the background. In the final implementation of the application, an ExecutorService of the package "java.util.concurrent" was used to implement the processes. To communicate updates to the UI from the background processes, a Handler of the "android.os" package was used. The Handler object can communicate two different messages, either to notify the UI that a new update of the approximated result is available, or that a final result is computed. As the application should process equations with big matrix dimensions like 1000 and more, it has to deal with memory limitations of the Android OS. Every application gets only a specific part of the memory. To bypass this limitation, the application uses the large heap option, which allows to yield the memory of the smartphone.

7.2.2 ConnectionManager

The ConnectionManager class provides the necessary functions to communicate with the server. Two methods are provided to send an equation to the server, meaning the "configuration". One sends the matrix A and the equation result vector b of the equation $A * x = b$, while the other additionally sends an approximation of the x vector. Results of initiated remote executions can be fetched with the "receiveResult" method. This method blocks until the result is received from the server and can only be used if the equation system was sent via the same connection. If the mobile device wants to retrieve interim results, it is not necessary to send the equation system again or to use the same connection. Instead an approximation request can be sent with the according session identification number (sessionId) of the calculation. Further methods provide the sending of separate parts, like the header of a request or single rows of a matrix. On disconnects during any send operation, an IOException is thrown. Those exceptions have to be handled by the calling operation, like e.g. the MatrixExecutionManager. The two methods for sending configurations can be interrupted. This is important if another execution (e.g. a local execution) has already finished, as the send methods block until they completed. If the receive method is interrupted due to a disconnect or malformed responses from the server, it throws an exception, too. In this case, the returning object is null and has therefore be handled by the calling operation.

Listing 7.1 EBNF of request transmission protocol for mobile devices

```
Request = Option, SessionId, [Length, Data];
Option = '0' | '1' | '2'; (* Java int type *)
SessionId = (* Java long type *);
Length = (* Java int type *);
Data = MatrixData | MatrixDataWithEstimation;
Row = (* Java float array *);
MatrixData = Matrix, Row (* equation result vector*);
Matrix = Length * Row;
MatrixDataWithEstimation = Matrix, Row (* equation result *), Row (* estimation vector *);

Options:
0 : new calculation;
1 : new calculation with estimation of x vector;
2 : request existing result;
```

Data Transmission

Listing 7.1 presents the request transmission protocol for the mobile device in Extended Backus-Naur Form (EBNF).

The Socket class of the "java.net" package was used for the network connection, while the sending of the data was implemented with the "java.io.DataStream" classes. To improve the data transmission speed, the DataStream object wrapped a BufferedStream object instead of directly writing to the server output stream. Additionally, the values of the arrays were written into a ByteBuffer object, such that every matrix row is transmitted straight. Although the data transmission speed could further be improved by a data compression. As seen in Section 4.1.3, the error value of the iterative equation solvers amounts about 10^{-13} for a residual of 10^{-9}. All the algorithms use Java double types, which have a size of eight byte. But for an accuracy of 10^{-13} it is enough to use a Java float type, which has a size of only 4 byte. As the smallest positive value of float is $2^{-149} \approx 1.4 * 10^{-45}$, no important decimals are lost if the value is compressed from a double type to a float type. But only as long as all values of the equation are less than the maximum of the greatest float value $(2 - 2^{-23}) * 2^{127}$ and greater than the smallest float value $-(2 - 2^{-23}) * 2^{127}$, which is assumed in this test application. Therefore the ConnectionManager class uses a lossy data compression before sending equations.

7.2.3 Profiler

The profiler is including responsible for monitoring executions of equation calculations. Hence the profiler provides methods to add local and remote execution times. Those times are bound to their matrix dimensions of the equation. Previous results are overwritten to always have the statistics up to date. Although this approach can cause anomalies in the statistics. An anomaly occurs, if e.g. a time value for an equation with dimension x is smaller than the time value for an equation with a dimension less than x. The equalization process uses the following function and sets both compared values to the $newValue$, for which $newValue = rightRemoteValue + \frac{leftRemoteValue - rightRemoteValue}{2}$, with $leftRemoteValue > rightRemoteValue$. Hence, after adding a time value, the "straightenPreviousValues" function is invoked to prevent anomalies in the statistics.

For remote executions it is important to involve the network situation, because the limiting factor of a remote execution depends on the time period to send the equation data to the server. According to that, all remote execution time values are bound to a connection establishment time, meaning the "three way handshake" of TCP. At initialization time, three default time values are added to the profiler to create an initial statistic. These values are based on the execution times that were measured in the pre-tests in Chapter 4.

The "checkForRemoteCalculation" function checks whether a remote execution is beneficial or not. The decision process is explained in figure Fig. 7.2. The profiler is based on previous executions, but it is possible that not every specific equation dimension has already been calculated in the past. Thus the profiler provides the two "fetchOrEstimate" functions. If the profiler gets a request for an estimation of a matrix dimension that is not available, it has to linear interpolate the value between two existing values.

7.2.4 MatrixExecutionManager

The MatrixExecutionManager class provides two different methods, "solve" and "solveFromFile", to initiate the solving of an equation system.

The "solveFromFile" method should be used if the equation dimension of the input file is too big to be processed on the device. Instead of processing the matrices of the equations in two dimensional arrays on the device, this method reads every row of the input file and directly sends it to the server. To process big equations locally on the mobile device, it would be necessary to modify the solving algorithms. This is a sample of how the memory resources on a mobile device can be enhanced by using cloud techniques, without the need to modify existing algorithms.

The "solve" method takes a matrix and an equation vector as input. Additionally a handler has to be passed to communicate updates to the UI. First it is checked, which execution method should be started. This decision is made by checking the settings in the configuration and the

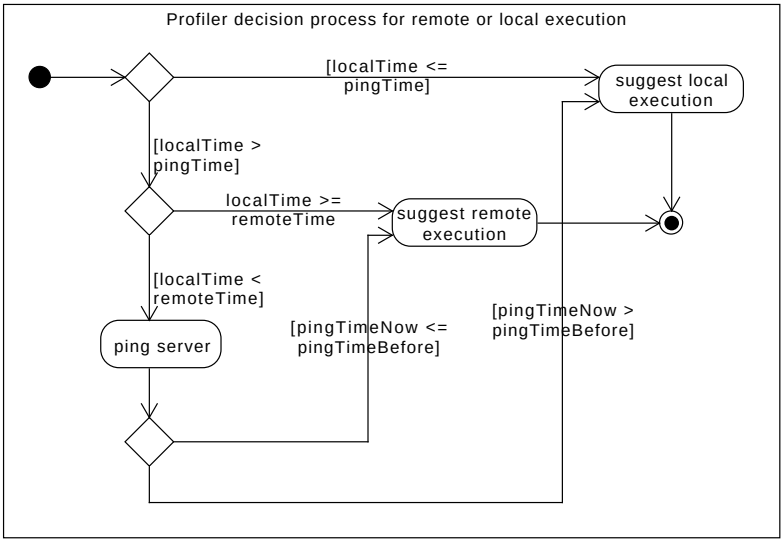

Figure 7.2: Activity diagram of the mobile device profiler for the decision process

profiler. On a local only execution, a LocalMatrixExecutor (LME) object is submitted to the ExecutorService, whereas on a remote only execution a RemotePeriodicMatrixExecutor (RPME) object is submitted. If the profiler of the configuration suggests remote solving, multiple executors will be started concurrently. With the "invokeAny" method of the ExecutorService, a RemoteMatrixExecutor (RME) and a NetworkSituationMonitor (NSM) will be started concurrently. On exceptional interruption of one of those processes, the LME and the RPME are submitted with the "invokeAny" method.

RemoteMatrixExecutor

The RME tries to establish a connection between the mobile device and the cloud server, sends the equation system and waits for the result. No interim results are retrieved and no adaption to changing network situations is made. If any exception is raised during this process, the RME stops. All progress by this executor is saved into a shared state object. The computation time is added to the profiler if the remote execution was successful.

NetworkSituationMonitor

The process of a remote execution is monitored by the NSM. After timeout t, $t = 1000ms$, the NSM checks the current network situation by establishing a connection to the server. If the time to establish the connection takes longer than in the previous attempt, the network

quality decreased. In this case the NSM decides whether to decrease the result quality of the calculation or to stop containing the energy resource. As long as the RME already sent the equation system, the NSM can retreive interim results (reducing priority for result quality). Otherwise a local execution has to be started (reducing priority for energy). If the network quality is still good, the NSM waits for the next timeout to occur and checks the situation again.

RemotePeriodicMatrixExecutor

The RPME has a similar task as the RME. The only difference is, that the RPME does not stop upon exceptions. Instead it tries the same task periodically again, until the final result is returned from the server. If the equation was already sent in earlier connections, the RPME does only make requests for the current result on the server instead of sending the equation again. If the received result is an interim result the RPME will send a new request after one second, otherwise it is a final result and the process can terminate.

LocalMatrixExecutor

The LME starts a new local equation computation. All progress, i.e. after every iteration of an approximation algorithm, is updated in the shared state object. If the residual or the maximum of iterations for the algorithm is reached, the result is returned and the LME process terminates.

7.2.5 RemoteProgress

The middleware uses different executors, which might execute concurrently. Thus to keep all concurrent tasks up to date, it is necessary to provide a shared state object that stores the progress of all running tasks. The matrix, the equation result and the current x vector of the equation can be stored in the RemoteProgress class. Besides that, the norm of the current x vector and a status message can be synchronized. Flag fields can additional be set or checked, to get an overview about the progress for the remote execution. The flag fields, the status message and the current x vector use the Java volatile keyword to be up to date in all threads.

7.2.6 Equation Solver

Local executions can use the LocalMatrixSolver class to solve equation systems. The provided algorithms are the Apache implementations of the conjugate gradient method, the SYMMLQ

Listing 7.2 Application input file for equation systems

```
MatrixDimension
MatrixRow1Column1;...;MatrixRow1ColumnN
.
.
.
MatrixRowNColumn1;...;MatrixRowNColumnN

VectorValue1;...;VectorValueN
```

Listing 7.3 Application output file

```
VectorValue1;...;VectorValueN
```

solver and the decomposition solver. They are provided in the Apache Commons Math library. Additionally the self implemented Jacobi algorithm can be used. When using the Jacobi algorithm, a RemoteProgress object can be passed to the LocalMatrixSolver constructor, which causes the updating of the x vector in the progress object.

7.2.7 Application Input and Output

There are two types of input for the equation solver. The user can either provide a file containing the matrix A and the according vector b, or generate a random equation system as for the pre-tests. The format for an equation input file is presented in listing 7.2.

In the application, the user has to provide the path to the input file, relative to the external storage directory of the Android system. The application can optionally create an output file that contains the result vector of the calculation. The format of the output file is a semicolon separated line with all values of the vector, as presented in listing 7.3. The output file is created in the external storage directory of the Android device.

7.2.8 Configuration

The application provides different solving methods and algorithms. Hence the Configuration class is responible for storing those settings. There are three different solving methods: local, remote, and network situation based (NSB). The local solving method initiates a local execution if a new calculation is started. On the other side, the remote solving method tries to solve the equation in the cloud, without considering the network situation. If no connection can

be established, the remote solving method cannot return a result. Both methods make the execution manager ignore the suggestions of the profiler. As a contrast to that, the NSB solving method makes the profiler to suggest the best solving method for the current situation. If the profiler suggests a remote execution, the network is monitored and the system reacts to changing network conditions. Another setting that is managed by the configuration is the solving algorithm that should be used for the equation calculation. The different algorithms are the self implemented Jacobi method and the Apache implementations of the conjugate gradient method, the SYMMLQ solver, and the decomposition solver. This setting does only affect the local execution and not the execution on the cloud server. Further settings provide connection information like the url to the cloud server, the according port on the server and the default RTT timeout. There is also the option to create an output file with the containing result vector at the end of a successful calculation.

7.3 Implementation of the Cloud Server

Figure Fig. 7.3 shows the structure of the server in a class diagram. The server is implemented in Java. It can manage multiple requests from multiple clients, as every request gets handled in an extra thread. The ClientConnector class listens for new requests from clients and creates a new ClientWorker thread, if a client was accepted. The ClientWorker then filters out the request classification. The option field in the request header defines if the ClientWorker object has to handle a new equation calculation request with or without an estimation of the result. Further options are a result request for running or finished calculations, a spline calculation request or an unknown request. Every request implies that the server sends a response to the client. The EBNF structure in listing 7.4 presents the transmission protocol for a response of the cloud server.

The connection techniques on the cloud server are similar to the techniques on the mobile device. I.e. for the connections, the "java.net.Socket" and for the transmission the "java.io" DataStream classes are used. The compression technique is also the same as on the mobile device, which means that the double values get compressed to float values.

Listing 7.4 EBNF of the server response protocol

```
Response = Type, SessionId, [FinalResult, Data];
Type = '0' | '1'; (* Java int type *)
SessionId = (* Java long type *);
FinalResult = '0' | '1'; (* Java int type *)
Data = (* Java float array *);

Type:
0 : no data in response
1 : result exists; implies that data is sent in response

FinalResult:
0 : result is only an interim result, not finished calculation yet
1 : result is final result
```

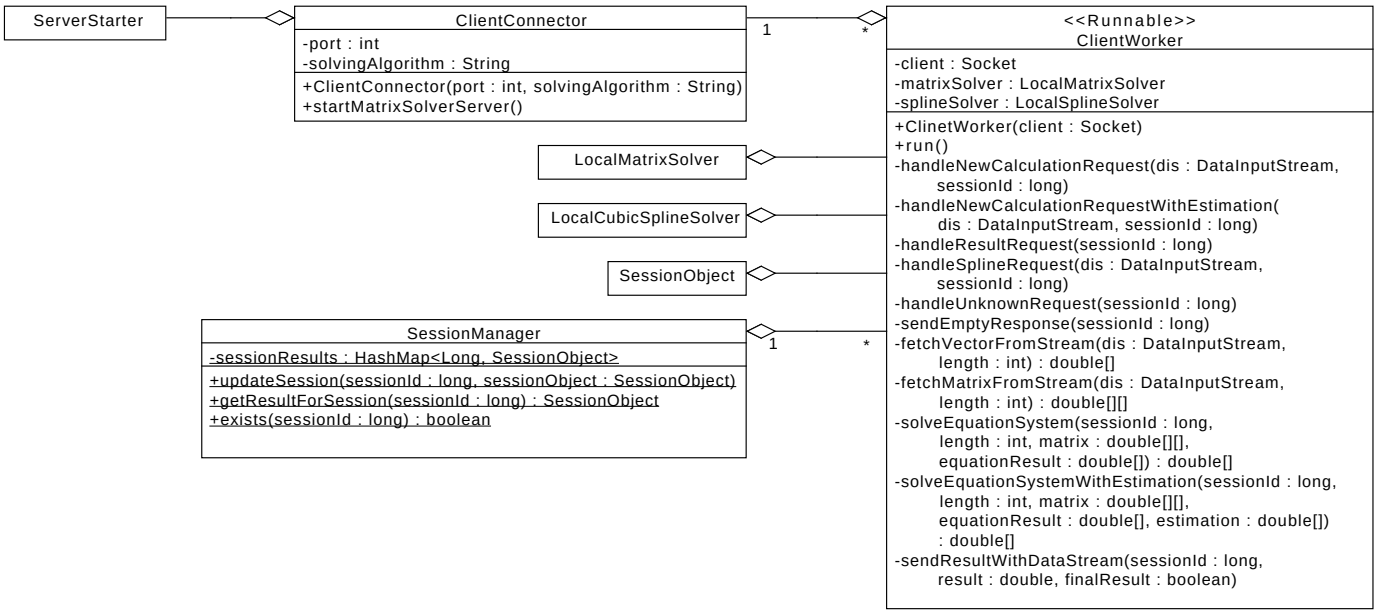

Figure 7.3: Class diagram of the Java cloud server

8 Discussion and Evaluation

In this chapter we evaluate the developed approach to efficiently solve linear equation systems. The presented application in Chapter 7 serves as a sample application. The results of the tests will be presented and discussed.

8.1 Transmission Protocols

The transmission protocol of the developed system is a key component for the efficiency, as the data transfer of the equation system is the only factor that can make a remote execution slower than a local execution. It was first considered to transmit the equations with the User Datagram Protocol (UDP), because it is faster compared to TCP. But although the quality of the result must not be 100% accurate, it is not acceptable to lose whole rows or columns of the equation matrix or vector, as it can influence the calculation considerable. Therefore an approach needed to be developed to control the successful transmission and the correct order of the arrived packages with UDP. Although, the overhead that would be produced by securing the data transmission would compensate the benefits of UDP. When considering this, it is more efficient to use a reliable data transmission protocol instead of UDP.

A further attempt that was tested is the Extensible Markup Language (XML) - Remote Procedure Call (RPC) technique. It is based on the Hypertext Transfer Protocol (HTTP) and provides a reliable data transmission. A data transmission speed comparison between TCP and XMLRPC can be seen in figure Fig. 8.1. This test serves to filter out which transmission protocol is faster. The test sends a matrix with dimension $n * n$ and a vector of length n with TCP, respectively it passes an array with length $n * n$ and n for XMLRPC. The Java float type was used for the values, which has a size of four bytes. Hence the data load for one transmission with matrix dimension $n*n$ is $4*n^2+4*n$ bytes. The wireless network (Home network) for the test is a 2,4 GHz 802.11n network with a data rate of 300 Mbit/s. The real throughput of 11.2 Mbit/s and an average latency of 4.023 ms for 20 packages in 20 seconds was measured before the tests. The test devices are the desktop computer as server (four core 3 GHz CPU, 8GB RAM) and the laptop as client (four core 1.6 GHz CPU, 8 GB RAM). The diagram depicts that XMLRPC is considerable slower for data transmission compared to TCP. For smaller matrix dimensions of 600, the difference between both techniques amounts about 300 ms. This is already a considerable time difference, but when comparing the time difference for a matrix dimension of 1500 and more, it amounts over two seconds. E.g. TCP reaches a transmission time of 7013

ms for a matrix dimension and vector length of 2000, while XMLRPC has a transmission time of 9503 ms. Thus the XMLRPC technique is not an efficient candidate for this system.

Another transmission technique that was considered is the Google Cloud Messaging [1] (GCM) service for Android. It is a service to send and receive messages from an Android device to a cloud server. The server then can distribute the message to other Android devices, too. The message queuing and delivery is completely handled by the GCM service. The system is implemented with three components, the GCM Android application (client App), the GCM Cloud Connection Servers (CCS) and a third party application server (3PS). The CCS are provided by Google and take messages from the 3PS server to send them to the client App and vice versa. The supported connection protocols between the CCS and the 3PS are HTTP and Extensible Messaging and Presence Protocol (XMPP). When using the GCM service, network disconnects must not be handled by the Mobile device middleware, as this is already handled by the GCM protocol. An Android application (on Nexus 5) and an application server (on desktop computer) were implemented, to test the GCM service. The application server uses the HTTP protocol to communicate with the CCS. The application server and the Android application are located in the same wireless network as in the previous comparison test between TCP and XMLRPC. Although the GCM service requires an Internet connection to connect to the CCS. The Internet connection provides 7.9 Mbit/s download data rate and 1 Mbit/s upload data rate. In this test, messages with 22 bytes took more than two seconds to be sent to the application server. This is significant slower than a transmission with TCP, where the transmission of 22 bytes takes one millisecond.

8.2 Computation Time Evaluation

The computation time of the developed remote approach is evaluated against the computation time of a conventional local execution. The remote computation time consists of the data transmission time $t_{transmission}$ and the time to wait for the result response of the server $t_{response}$. The local computation time consists of the time to solve the equation on the mobile device t_{local}. If $t_{transmission} + t_{response} < t_{local} \Rightarrow$ the remote execution terminates faster than the local execution. Both execution methods, meaning local and remote, were tested on the Nexus 5 mobile phone (four core 2.26 GHz CPU, 2 GB RAM). According to the pre-tests, it is expected that the remote execution method is faster than the local execution method. The wireless network environment is the Eduroam network of the University of Stuttgart. The throughput of 80.2 Mbit/s was measured before the test runs, along with the average latency of 1.618 ms for 20 packages within 20 seconds. The test server application is running on the Nimbus server (eigth core 3 GHz CPU, 32 GB RAM) and is permanently available. The wireless network connection was turned on during remote and local executions on the mobile

[1] https://developer.android.com/google/gcm/index.html

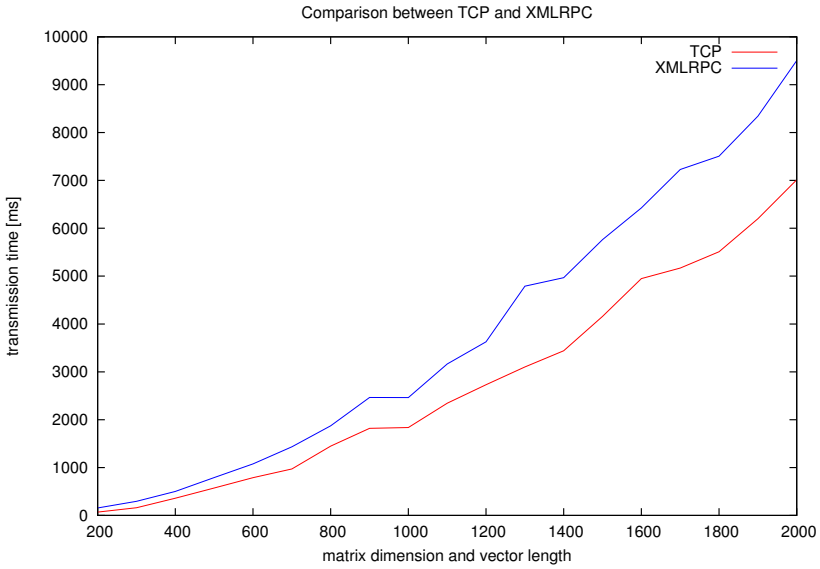

Figure 8.1: Comparison between TCP and XMLRPC. The client device was a laptop while the server was the Desktop computer

phone. The used algorithm is the Jacobi method on the mobile device and on the server. A maximum of 15 iterations and a residual of 10^{-9}, were taken as termination criteria on both devices. The equation dimension was scaled from 100 to 1500. Figure Fig. 8.2 depicts the results of the comparison between the execution methods local and remote in the Eduroam network. The local execution takes 922 ms for an equation with dimension 500, while the remote execution takes 150 ms. For a dimension of 1500, the local execution takes 9048 ms compared to 1894 ms for the remote execution. Although the remote execution time consists of $t_{transmission} + t_{response}$, it is about 80% faster for every equation dimension in this test against the local solving method. This test states the expectation, that the remote solving method is faster than the local method under the given network situation for the Jacobi method.

To state that the network environment influences the efficiency of the remote execution method, a further test was made in another network environment. The test devices in this test are the Nexus 5 mobile phone with the client application and the desktop computer (four core 3 GHz CPU, 8GB RAM) running the server application, which is permanently available. The wireless network (Home network) measured a throughput of 10.4 Mbit/s and an average latency of 4.843 ms for 20 packages in 20 seconds. According to the network quality definition in Section 5.1.3, the Home network has a worser network quality than the Eduroam network. The used solving algorithm on both devices is again the Jacobi method with a residual of 10^{-9} and a maximum of 15 iterations. Figure Fig. 8.3 depicts the results of the comparison between the execution methods local and remote in the Home network. The local execution is faster than the remote execution for equation dimensions 100, 200 and 300. From dimension

400 and greater, the remote execution terminates faster than the local execution. Also, the time difference between both execution methods increases after equation dimension 400. The execution in the Eduroam network is significant faster than the execution in the Home network for all tested equation dimensions. Note that both tests do not only have different network environments, but also different servers. However, for a remote execution without disconnects to the server, the remote execution time $t_{remote} = t_{transmission} + t_{response}$. Therefore, the greater the value of $t_{transmission}$, the greater the value of t_{remote}. Even in the case, that both tests had the same value for $t_{response}$, the remote execution in the Home network would be slower than in the Eduroam network. Hence the efficiency of the remote execution depends on the network quality, meaning the throughput and latency.

Both tests used the Jacobi method as solving algorithm on the server and the mobile phone. In contrast, the conjugate gradient method computes equations faster than the Jacobi method. Thence we analyzed if the remote execution method is still faster than the local execution when the applications uses the conjugate gradient method. The tests in the Eduroam network and in the Home network indicated, that the local computation time of the conjugate gradient method was significant faster compared to the data transmission time in both network environments. So, if the local computation time t_{local} of a task is less than the data transmission time $t_{transmission}$, it is not beneficial to use the remote execution.

8.3 Computation Time Discussion for Instable Connections

Further tests involved disconnects between mobile phone and server. The network situation based (NSB) execution method was used in this test and the profiler suggested a remote execution. This test shows the reaction of the protocol on disconnects. According to the protocol, the mobile application started a local execution as soon as the disconnect occured. The mobile middleware tried to reconnect every second. The equation system could not be sent before the disconnect occured in this test cases. Hence the whole equation system had to be sent again on reconnect. The remote computation time is influenced due to the disconnect and has to be increased by the connection absense time t_a. Additionally it consists of the time difference t_x between the start timepoint of the send operation and the timepoint of the disconnect. On multiple disconnects, the values $t_x + t_a$ have to be summed up. Therefore the remote computation time $t_{remote} = t_{transmission} + t_{response} + \sum_{i=0}^{n} t_{x_i} + t_{a_i}$, having $i = 0, 1, ..., n$.

In test cases where the equation system could be sent before the disconnect occured, the mobile middleware sent an approximation request, with time value t_{approx}, instead of the whole equation system. If the response of the server to the approximation request of the mobile phone does not contain a final result, a new approximation request is sent after one second. This repeats until the server response contains the final result. A further time factor that has to be considered in this case is the time difference t_y between the end of

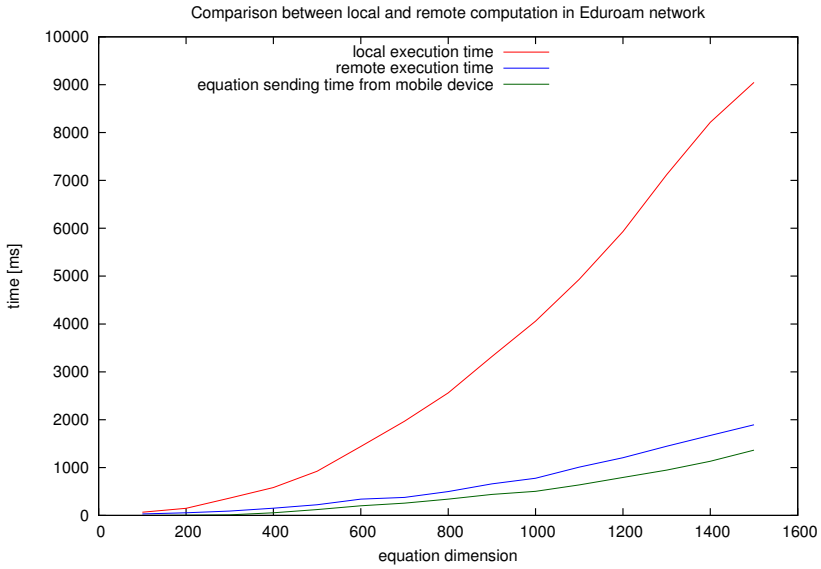

Figure 8.2: Computation speed comparison between local and remote executions, on Nexus 5. The network environment is the Eduroam network and the used solving algorithm is the Jacobi method

the send operation and the disconnect. The formula for t_{remote} in this case is: $t_{remote} = t_{transmission} + t_y + t_a + k * (t_{approx} + t_{response})$, having k as the number of approximation requests until the final result is received as response from the server.

8.4 Energy Consumption Evaluation

The evaluation of the energy consumption requires external components, as the Android system has no exact energy monitor. Also applications like Powertutor [2] are not completely accurate on the available test devices, as this application is developed for older devices. Hence we needed to set up a circuit to measure the energy consumption.

8.4.1 Setup of the Energy Measurements

The circuit diagram is depicted in figure Fig. 8.4 and the resistor values are defined in table 8.1.

[2]http://ziyang.eecs.umich.edu/projects/powertutor/

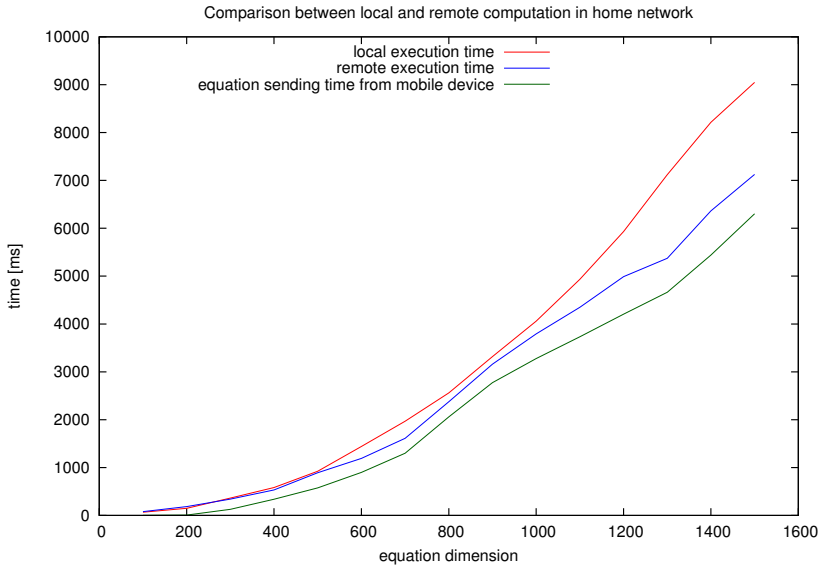

Figure 8.3: Computation speed comparison between local and remote executions, on Nexus 5. The network environment is the Home network and the used solving algorithm is the Jacobi method

Resistor Identifier	Resistor Value
R_M	0.1 Ω
R_1	4.7 Ω
R_2	1.0 $k\Omega$
R_6	330.0 $k\Omega$
R_7	12.0 $k\Omega$

Table 8.1: Resistor value table

As amplifier we used the LM324-N, a low power quad operational amplifier. The test device is the Galaxy Nexus mobile phone (dual core 1.2 GHz CPU, 1 GB RAM). The battery of the mobile phone is detached from the device and inserted separately into the circuit. In place of the battery, an adapter that connects the mobile phone with the circuit is attached (resistor R_H). This guarantees that the mobile phone does not need to be connected to a power source. A voltmeter measures the non inverted amplified voltage U_a, while a computer program logs the values every millisecond. The energy consumption is calculated with $\int_{t_0}^{t_1} u_h(t) * i_h(t)dt$. The network in this tests is the Eduroam wireless network of the University of Stuttgart. The server application is running on the Nimbus server (8 core 3 GHz CPU, 32 GB RAM). An execution start delay was implemented into the android application. This is necessary

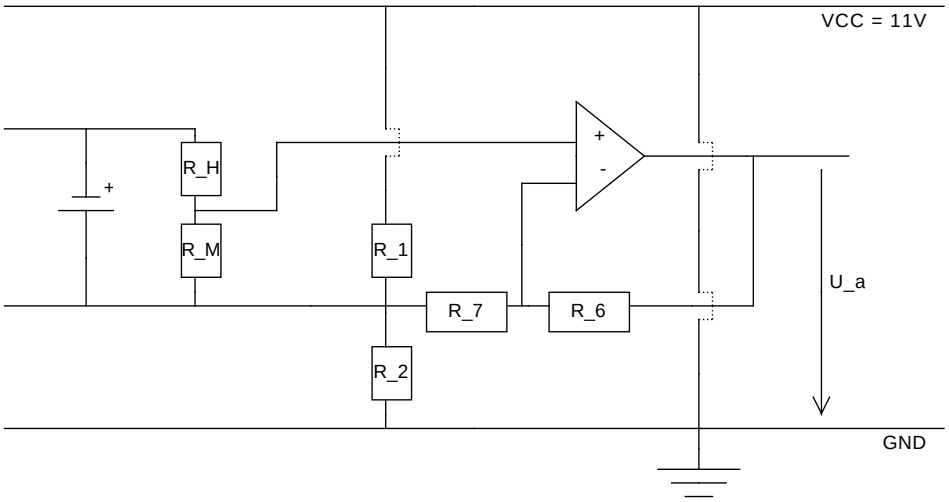

Figure 8.4: Circuit diagram for energy measurements on the Galaxy Nexus

to filter out the effects of touch events on the mobile phone, e.g. when touching the start button for the calculation initiation. The display was turned on during the tests, but with the lowest brightness value. The wireless network connection was turned on during all energy measurements, i.e. during both, remote and local executions. No other applications, except of the test application were running during the tests.

8.4.2 Energy Consumption Comparison between Local and Remote Executions

The test measures the consumed energy when computing an equation with the network situation based (NSB) method. One test has a stable connection to the server, which results in a remote execution, while the other test run does not have a connection to the server, which results in a local execution. The used solving algorithm is the Jacobi method with a termination criteria of 15 iterations. The residual is not considered as termination criteria in this test to have equal algorithm iteration rounds for all tests on the mobile phone and on the server. The equation dimensions for the tests are set to 1200. Ten seconds were taken as delay, before the execution starts. According to the longer execution time of the local execution, the time scale (x-axis) depicts the values between millisecond 10000 and 24000 for the local execution, while the graph of the remote execution has a range from millisecond 10000 to 14000. The power scale ranges from 0 to 8.5 Watt for both execution methods. Figure Fig. 8.5 depicts the energy consumption of the local execution, while figure Fig. 8.6 depicts the energy consumption results of the remote execution. There is a significant difference between both power plots. The local

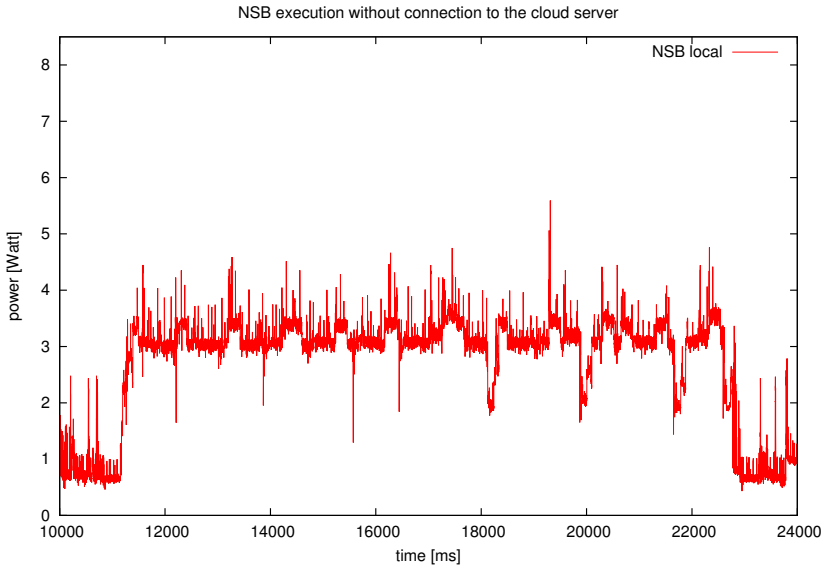

Figure 8.5: Power consumption for local NSB execution without connection to the server and matrix dimension 1200

execution (11699 ms) runs longer than the remote execution (2461 ms), but the highest power values appear in the remote execution. The maximum power value of the remote execution amounts 7.8387 Watt, while the local execution has a maximum power value of 5.5948 Watt. Nevertheless, the local execution consumes overall more energy with a value of 36.3994 Joule compared to the remote execution with 10.3814 Joule. A further test measured the energy for an equation dimension of 1800. In this test case, the local execution consumes 83.6387 Joule over 27368 ms, compared to the remote execution energy consumption of 22.0870 Joule over 4845 ms. Therefore, the saved energy of the remote execution compared to the local execution amounts 26.0180 Joule for a dimension of 1200 and 61.5517 Joule for a dimension of 1800. This means the remote execution saves over 70% of energy against a local computation.

8.5 Resource Priority Discussion

During the tests of the developed approach, different priority distributions were considered. In the system model, the highest priority is set to the execution speed, in case that the network quality decreases. The reason is that the developed approach is intended to be used in real time applications. Thus the next paragraphs argue, why it is not efficient to focus on containing the resource energy or result quality if the network quality decreases.

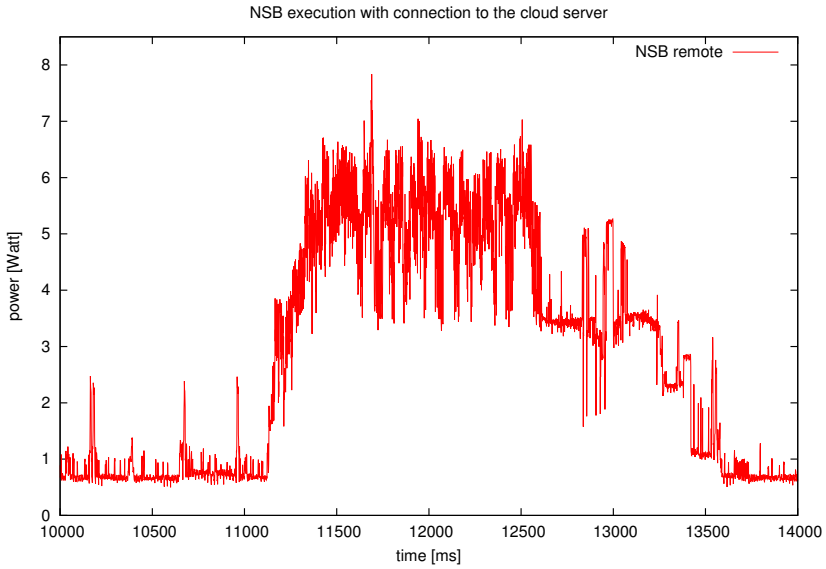

Figure 8.6: Power consumption for NSB execution with connection to the server and matrix dimension 1200

8.5.1 Containing the resource energy

In the case of a decreasing network connection quality, the middleware would realize that it is not energy efficient to send the equation system at this moment, as the sending costs the most energy in this process. Hence the system would wait until the network connection improves. Also to safe energy, no local computation would be initiated concurrently. In a worst case scenario, if the network connection is not improving in quality or not available at all, the system would wait infinite long. Thus it is not profitable to contain the resource energy saving when the network connection quality decreases.

8.5.2 Containing the resource result quality

If the middleware relies on a very accurate result quality, the computation would take considerable time longer. I.e., the highest result quality for solving linear equation systems can be reached when using an exact solver. However, as shown in Chapter 4, exact solvers are considerable slower than iterative solvers. A disadvantage of exact and non-iterative solvers is, that no interim results are available. In the case that no connection to the server is available, a local execution is initiated. If the connection is available at a later point of time, the middleware cannot take the interim result of the local execution and send it to the server along with the equation. So the time period with no network connection is not used efficiently with exact solvers.

9 Conclusion

This bachelor thesis addresses the development of a procedure to efficiently solve linear equation systems on mobile devices. The system therefor utilizes a Cloud-Infrastructure and optimizes the resources energy, computation time and result quality under the given network situation. The solving algorithms are specialized to solve diagonally dominant matrices.

Mobile devices, especially mobile phones, have restricted resource capacities due to its mobility. Related projects present different approaches to solve this problem. The most efficient results are recorded with code offload proceedings. Those proceedings aim to improve the computation time or to safe energy for abstract tasks. By contrast, the developed system in this thesis focuses on linear equation computation tasks and involves the varying network situation. The system takes advantage of diagonally dominant matrices, which occur in numerical simulations. It was assumed, that iterative solvers provide control of computation time and result quality of those equation systems. This assumption is confirmed by conducted pre-tests that analyze different iterative algorithms and non-iterative solvers.

The system model in this thesis consists of two hardware components, a mobile device and a Cloud-Infrastructure. The connection between those components is based on wireless network techniques. The processed input data are diagonally dominant and square equation systems of the type $A * x = b$. The difficulty of this setup is, to deal with variations in the network connection, like disconnects. The system takes advantage of the control of computation time and result quality with iterative algorithms, to adapt to changes in the network connection. If the network provides a good connection quality, the system optimizes the resources computation time, energy and result quality. On the other hand, if the network quality decreases, the system reduces the priority for the resources energy and result quality and focuses on computation time.

The developed system design considers a middleware for the mobile device, that takes care for the execution of equation calculations and adapts to changes in the network connection. The middleware consists of an execution manager, a profiler and an equation solver component. The profiler has the responsibility to monitor executions and provide functions to suggest local or remote executions. The execution manager distributes the equation calculation to the Cloud-Infrastructure or to the local equation solver, based on the network situation. The equation solver component provides iterative algorithms to solve the linear equation systems on the mobile device.

The counterpart of this system, meaning the cloud servers, also consist of three components: a client manager, a session manager and an equation solver. The client manager component handles incoming request of mobile clients and either uses the session manager or the equation solver to handle the request. The session manager maps all equation results to a specific client session id. This is enables the response to interim requests of running calculations. The equation solver on the cloud server has the same job as the counterpart on the mobile device.

An Android application and a server were implemented to evaluate the developed system. The evaluation considered different transmission protocols for the data transmission of the system. The considered protocols were UDP, TCP, XMLRPC and Google Cloud Messaging. Evaluation tests filtered out TCP as the best transmission protocol for the purpose of this system. The computation time evaluations revealed that the developed system can execute calculations 80% faster with a remote execution compared to conventional local executions. Although, the computation time depends on the given network situation. If the network connection has low quality, it is not beneficial to use a remote execution, because the local execution is faster in this case. The same findings are recorded for the energy measurements. With a stable and high quality network connection, energy savings of 70% were measured. A further important finding is that remote executions are only beneficial for long running and resource demanding tasks.

The developed system proved in evaluation tests that it successfully adapts to changes in the network connection and optimizes the resources according to the protocol in the system design.

Future Work

It was not possible to research every aspect in this field of study, due to the restricted scope of this bachelor thesis.

Open points are e.g., how cellular network techniques like 3G/4G influence the energy consumption of remote executions. Related work argued about the efficiency, e.g. the MAUI [Edu10] system measured more energy savings when using WiFi as network connection compared to 3G, whereas the ThinkAir [Sok12] system was more energy efficient with 3G instead of WiFi.

A further point is the use of higher sophisticated monitors or profilers. The system in this bachelor thesis uses a simple monitor for executions and the network connection. This decision is based on findings of the related projects MAUI [Edu10] and the system of Giurgiu et al. [Ioa12]. However, a higher sophisticated monitor could provide detailed predictions about the network situation which could be used to decrease the computation time or energy consumption. A detailed evaluation between a system with simple monitor against a system with a high sophisticated monitor could bring clarity.

The transmission of the equation parameters is the bottleneck of a remote execution. Therefore, if the data to transmit could be reduced, the system would be more efficient. Mathematical problems like partial differential equations do only have few input parameters, but produce huge matrices. This property can be used to provide efficient usage scenarios for solving linear equation systems.

10 Appendix

Measurement Computer Device Specifications

The first device is stationary, non mobile computer device and will be called as "Desktop computer" in the following. It has an AMD Phenom II X4 945 processor (four CPU cores) with a clock frequency of 3 GHz, 8 GB of random-access memory (RAM) and a 500 GB hard disk drive (HDD). The operating system (OS) is a Ubuntu 14.04 LTS Linux distribution. The OS architecture uses 64-bit.

The second device is also stationary and non mobile. It is the server that is provided in the network of the University of Stuttgart, hosted by the Institute of Parallel and Distributed Systems. It has four Intel Xeon CPU X5472 on two sockets (eight CPU cores) with a clock frequency of 3 GHz, 32 GB of RAM and a 270 GB HDD. The OS is GNU/Linux with a 64-bit architecture. In the following it will be called as the "Nimbus" server.

The third device is a mobile device. It is a Nexus 5 smart phone. The processor is a Qualcomm Snapdragon 800 with a clock frequency of 2.26 GHz (four CPU cores). It has 2 GB RAM and a flash memory capacity of 32 GB. The battery of the Nexus 5 has 2.300 mAh. The operating system is a 5.0.1 Android build.

The next mobile device is a Galaxy Nexus smart phone. It has an ARMv7 Processor CPU (dual core) with a clock frequency of 1.2 GHz. The RAM has 1 GB and the flash memory has a capacity of 16 GB. The battery capacity is 1750 mAh. As OS it uses the Cyanogenmod 10.21 + Google Apps (Android 4.3.1) and is unlocked.

A further mobile device is a laptop. It has a Intel i5-4200U processor (quad core CPU) with a clock frequency of 1.6 GHz. The DDR3 RAM amounts 8 GB, while the SSD has 256 GB capacity.

Curium is a stationary server. It has eight cores and 32 GB RAM. The operating system is Linux with a 64-bit architecture.

Kepler is a stationary server. It has 32 cores and 128 GB RAM. The operating system is Linux with a 64-bit architecture.

Bibliography

[Ach95] Achim Basermann. *Iterative Verfahren für dünnbesetzte Matrizen zur Lösung technischer Probleme auf massiv-parallelen Systemen*. 1995. (Cited on page 13)

[Byu11] Byung-Gon Chun, Sunghwam Ihm, Petros Maniatis, Mayur Naik, Ashwin Patti. *CloneCloud: Elastic Execution between Mobile Device and Cloud*. 2011. (Cited on page 21)

[C. 75] C. C. Paige, M. A. Saunders. *Solutions of Sparse Indefinite Systems of Linear Eqautions*. 1975. (Cited on page 44)

[Chr14] Christoph Dibak, Boris Koldehofe. *Towards Quality-aware Simulations on Mobile Devices*. 2014. (Cited on pages 11, 17 and 18)

[Edu10] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, Paramvir Bahl. *MAUI: Making Smartphones Last Longer with Code Offload*. June 18, 2010. (Cited on pages 11, 17, 19, 20, 21, 40 and 66)

[Eug09] Eugene E. Marinelli. *Hyrax: Cloud Computing on Mobile Devices using MapReduce*. 2009. (Cited on page 17)

[Flo14] Florian Berg, Frank Duerr , Kurt Rothermel. *Increasing the Efficiency and Responsiveness of Mobile Applications with Preemptable Code Offloading*. June 2014. (Cited on pages 21 and 22)

[G. 61] G. Gordon. *A General Purpose Systems Simulation Program*. 1961. (Cited on page 11)

[Gon10] Gonzalo Huerta-Canepa, Dongman Lee. *A Virtual Cloud Computing Provider for Mobile Devices*. 2010. (Cited on pages 11 and 18)

[Ioa12] Ioana Giurgiu, Oriana Riva, Gustavo Alonso. *Dynamic Software Deployment from Clouds to Mobile Devices*. 2012. (Cited on pages 19, 20, 40 and 66)

[Mic09] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. *Above the Clouds: A Berkeley View of Cloud Computing*. 2009. (Cited on page 17)

[N. 13] N. Ding, D. Wagner, X. Chen, A. Pathak, Y. C. Hu, A. Rice. *Characterizing and Modeling the Impact of Wireless Signal Strength on Smartphone Battery Drain*. 2013. (Cited on page 21)

[R. 94] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, H. Van der Vorst . *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. 1994. (Cited on page 44)

[Sok12] Sokol Kosta, Andrius Aucinas, Pan Hui, Richard Mortier, Xinwen Zhang. *ThinkAir: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading*. 2012. (Cited on pages 11, 17, 19, 20 and 66)

[Str02] Strakoš Zdenek, Petr Tichý. *On error estimation in the conjugate gradient method and why it works in finite precision computations*. 2002. (Cited on page 29)

[Wol94] Wolfgang Hackbusch. *Iterative solution of large sparse systems of equations*. 1994. (Cited on pages 11, 13, 14 and 15)

[Zhi01] Zhiyuan Li, Cheng Wang, Rong Xu. *Computation Offloading to Save Energy on Handheld Devices: A Partition Scheme*. 2001. (Cited on page 18)

All links were last followed on April 12, 2015.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature