CANDIS: Heterogenous Mobile Cloud Framework and Energy Cost-Aware Scheduling

Sebastian Schildt, Felix Büsching, Enrico Jörns, Lars Wolf
Technische Universität Braunschweig
Institute of Operating Systems and Computer Networks (IBR)
Mühlenpfordtstraße 23, 38106 Braunschweig, Germany
Email: {schildt | buesching | joerns | wolf}@ibr.cs.tu-bs.de

***Abstract*—We present CANDIS, a framework that can distribute computing tasks to a computing cloud consisting of mobile devices running Android as well as computers. We argue how integrating mobile devices into an IT infrastructure makes ecologic sense while saving costs at the same time. We show how fluctuating energy spot market prices due to demand and varying supply could be fed into a CANDIS scheduler to further reduce costs of computation by shifting it to times of cheap energy prices. Thus, the presented approach can increase a business' sustainability, save money and – if applied on a larger scale – help to stabilize electricity grids when industrialized countries continue their move to renewables.**

***Index Terms*—Cluster Computing; Smartphone; Mobile Cluster**

I. Introduction

The computational power available to mobile devices such as smartphones or tablets is increasing rapidly. Currently this sector of IT industry is growing exceptionally fast, leading to more and more mobile devices in operation. Due to this growth and innovation speed, the mobile sector outpaces the development of classical personal computing hardware, and the gap in terms of performance is narrowing [1]. This has lead to some ideas to leverage the computing power of those devices, especially when they are idle[1][2][3][4].

With mobile devices offering "usable" amounts of computing power it makes economic and ecologic sense to leverage as much computing power from these devices as possible during their limited lifetime. Even if specialized high performance server hardware might provide better performance per watt ratios, a classical server will also consume much more power when it is idle. It will also need more supporting infrastructure such as space and air conditioning, and thus imposes a much higher TCO compared to mobile devices. Additionally, mobile devices are already existing, so if by using spare computing cycles they can displace the cost, energy and material consumption from the production and operation of a high performance server, the net gain of such an approach increases. It is estimated that only around 25% of the total energy consumption of a mobile device can be attributed to usage, while 75% of a mobile device's energy budget is used during production [5]. Therefore, from a sustainability point of view, the usage of already existing mobile devices should be increased.

In [1] we tried to get a grip on the raw computing power available to mobile devices by borrowing the proven LINPACK benchmark from the high-performance computing (HPC) community. However, in the short term there might be more fruitful applications than HPC for mobile devices. In this paper we focus on the following scenario from [1]: A business decides to offload some of its computation to mobile devices. During working hours while employees sit in their offices and charge their mobile devices a backend infrastructure offloads tasks to the employee's phones.

In this paper we present CANDIS, a software framework that has been designed for these kinds of scenarios. While it can be argued whether the saying "Java is the new COBOL", is entirely true, the majority of enterprise applications today are written in Java. This is a good fit for the Android ecosystem, which also uses Java as its primary language. CANDIS is therefore based on Java and can distribute work to different mobile devices and gather the results. The advantages of this approach are twofold: Existing business logic written in Java can be reused and does not need to be recoded. At the same time, this allows CANDIS to not only use mobile devices as computation devices, but also normal desktop or server hardware. This allows for an approach where as much work as possible is done on mobile devices, while the remaining work before a deadline can be seamlessly shifted to a high performance cloud.

We will also present a simulation showing that CANDIS has the potential to reduce the energy costs by using a price aware scheduler. To achieve this we assume a billing model where the consumer can follow spot market electricity price variations. With the appropriate business models in place, this will also provide incentives which would help to "smoothen" electricity usage, adapting it to the current network situation, which becomes more important with an increasing amount of renewable energies on the grid.

II. CANDIS Architecture

Figure 1 shows the general setup of the system. The system consists of n heterogeneous clients which are wirelessly connected to a controlling server. While CANDIS focuses on mobile devices but it is not limited to Android tablets or smartphones. Normal desktop or server computers can be integrated as well. This basic setup is very scalable. Scalability is only limited by the performance of the controlling server and the capacity of the network links. Large systems can easily be partitioned by employing more than one control server, and the network side can be scaled by using multiple WiFi APs.

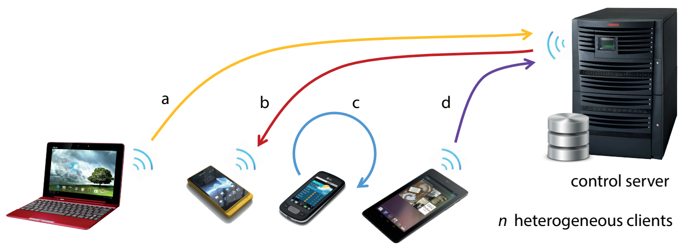


Figure 1. CANDIS system overview

A. System Overview

Every time a new device is going to be part of a mobile cloud, it first has to register at the control server ((a) in Figure 1). The server keeps track of all currently connected nodes and manages the distribution of computation tasks. Pending tasks are distributed to registered and connected nodes (b). The nodes start the computations (c) and finally send the results back to the controlling server (d) who might aggregate the results.

In this concept of distributed computing there are three different parties involved, as can be seen in Figure 2. First of all the developer, who implements the problem to be solved. Second, the control server, which manages the clients and distribution of tasks. And, last but not least, several clients which perform the actual computational tasks.

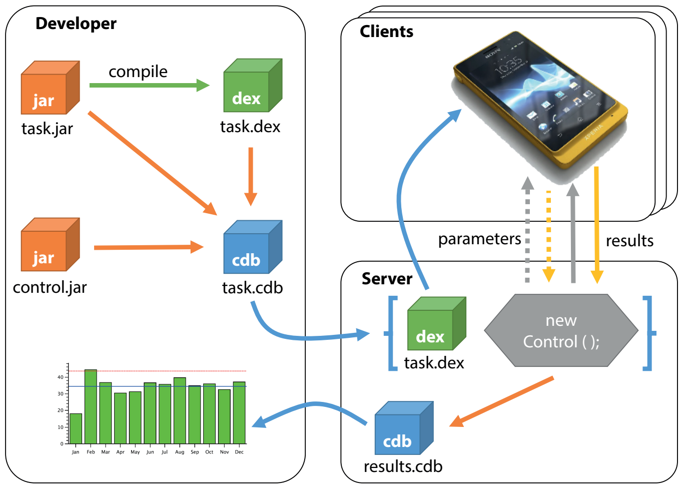


Figure 2. CANDIS project lifecycle

1) Developer: The developer implements one or more tasks which shall be computed by the distributed system. These tasks are written in Java and are packaged within the *task.jar* file – including a description of the parameters and the format of the results. By using standard Java, legacy code can be easily reused and integrated when developing a CANDIS application. The *task.jar* file is compiled to a Dalvik Executable *task.dex* file – a compiled Android application code file, which can be executed on Android tablets or smartphones. The *control.jar* file defines the behavior of the server. All three files are stored in the *task.cdb* container. This container is then transferred to the server. If there are multiple different tasks to compute, different containers can be loaded into the server.

2) Server: At the server the *task.cdb* file is opened and – depending on the information in the containing *control.jar* – a new controlling instance is started, which allows managing the clients according to the task. This includes the initialization of a scheduler for the allocation and distribution of the tasks. The Android executable *task.dex* is extracted and can be transferred to registered devices. Because the *task.jar* contains equivalent code, the server can also distribute tasks to clients using a normal JVM such as standard servers. The results of the distributed calculations are collected and aggregated by the server.

3) Clients: Every time a client is registered and chosen by the server, the *task.dex* can be transferred to the particular device. After that, the parameters for the computations can be sent by the server, thus once a client is seeded with the appropriate code, the overhead for submitting new tasks is reduced, as only the input data (parameters) need to be transferred. After any successful computation, a client will send the results to the server.

B. Scheduling Strategies

The scheduling – performed by the server according to the strategies chosen by the developer – can be adapted for several use cases. Its main purpose is to divide the parameters for the tasks into reasonable slices and send these slices to the clients. When n clients are registered at the server, the most straight forward way to implement the scheduling is to just divide the problem into n equal slices and wait for the results to come by. This is surely not the best strategy, as – in a heterogeneous system – there may be huge differences in computational power, memory usage, and power consumption between the individual devices. In the worst case, a powerful client is able to compute its task in very short time, while the server needs to wait for the results from a under-performing client for a long time and most clients in the system are idling.

Dividing the set of parameters into $\gg n$ slices for n nodes enables the server to send smaller slices to the clients and to send new slices every time a result from the particular client arrives. This leads to a better utilization of the framework, as highly efficient nodes will get to calculate more slices and underachieving clients will compute fewer. However, with more and smaller slices, also the communication overhead raises. Just like in Generalized Processor Sharing (GPS), infinitesimal small slices would lead to an optimal overall utilization, but also to an infinitive communication overhead. The tradeoff between the size of the slices and the communication overhead is dependent on the specific task and the overall number and capacity of nodes; thus, there is no simple and static setup for quantity and size of the slices.

1) Self-Assessment of the Clients: A simple, practical method to optimize task distribution and slicing is the self-assessment of the clients. Every time a client registers at the server, it appends its computational capability and resources to the registration message. Based on this data, the server can determine the size of the slices he sends to each client. This is very efficient as no additional communication is needed for this strategy. Nevertheless, due to the possible different requirements of diverse tasks, the self assessment can only give an estimation of the actual capability for a specific task.

2) Task-related Benchmark: The problem with self-assessment is, that performance for a given task is influenced by much more factors than simple metrics such as core count or available memory. How fast is the storage system? To which performance metrics is a given task especially sensitive? Therefore the dotted lines in Figure 2 denote an optional benchmark phase for each client, which is directly tailored to a specific task. According to the time needed to finish the benchmark, the server can estimate the nodes capability for the current task. It can divide the remaining parameters according to the estimated individual performance of every single node. This will maximize utilization while minimizing communication overhead.

III. Evaluation

The Candis framework has been implemented for Android operating system (OS Version $\geq$ 2.3). We evaluated Candis on a wide range of devices. For the evaluation we used several quad-core Asus Nexus 7 tablets. The Sony LT26i (Xperia S) and the Samsung GT-I8160 (Ace 2) using dualcore CPUs represent current smartphones. To assess the performance that can be expected from currently widespread low-midrange Android handsets the single-core the Huawei U8860 (Honor) and HTC Nexus One are included.

To get a rough idea about the performance of those devices we implemented two Candis tasks. The *hash* task is a simple distributed brute force hash-cracking application. For demonstration purposes, this task computes hashes consisting of five characters each. The first two characters are predetermined by the server and the second three characters will have to be alternated by the clients. We setup the task in such a way, that a solution will never be found, as we did not want to confuse the results by surprisingly finding a match. The second test is the *XSLT* task, which transforms a large XML document into JSON format using XSLT. This task is more representative of real workloads.

The performance of the individual test devices in those tasks is depicted in Figure 3, which is scaled to 100% equaling the slowest device for each task. It can be seen that the performance varies widely between devices and that the more modern multicore devices can profit from tasks that can exploit multiple cores.

A. Scalability

In [1] we have already shown, that despite the less then perfect communication infrastructure and other challenges a mobile cluster can scale using a common benchmark. Figure 4 shows the results for the distributed *XSLT* and *hash* tasks running on 1 to 5 Nexus 7 tablets. This experiment shows, that the framework scales well. While the synthetic *hash* task scales best, the *XSLT* task, which should be more in line with the workload to be expected for real Candis applications, also scales reasonably well.

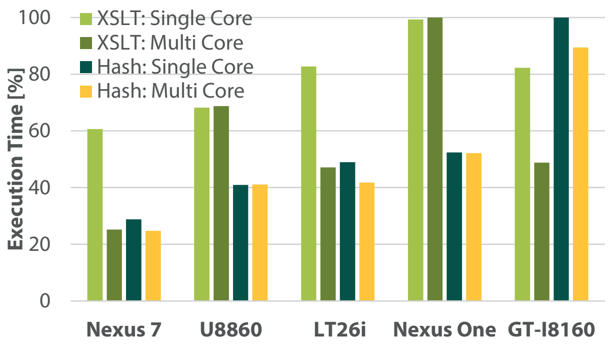


Figure 3. Performance comparison using the *hash* and *XSLT* tasks

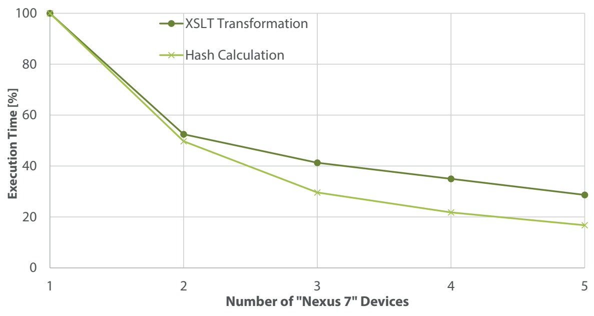


Figure 4. Scalability of the test tasks

B. Scheduling Heterogeneous Clients

With a heterogeneous set of devices, the effect of scheduling becomes significant. A given task will obviously run faster on more powerful devices with more cores (see Figure 3). A simple *slicing scheduler* (see Section II-B) would not take this into account. Therefore a *benchmarking scheduler* based on the simple principle “from each according to his ability, to each according to his need”[6] has been used, to distribute the workload in a fairer way. The scheduler tries to assess the performance of devices before distributing tasks (see Section II-B2).

In Figure 5 we scaled the weakest device (GT-I8160) to 100 % and compared the simple scheduler with the *benchmarking scheduler*. Using the *slicing scheduler*, the most powerful device would only run for $\approx$ 36 % of the time and idle the remaining time, waiting for the slower devices to finish. The device utilization of all devices can be increased by using the more intelligent *benchmarking scheduler*, because devices get individually sized chunks of work, according to their computation capability The total computation time is also lowered as the server does not have to wait for the weaker devices to finish their tasks; thus, the overall efficiency of the framework is significantly increased.

C. Smartphone energy usage

The most energy intensive components in a smartphone are the cellular modem, the display and the CPU[7]. As mobile devices can put currently unused components into a low power state, only the amount of energy used by the CPU is the relevant factor when evaluating the impact of computation

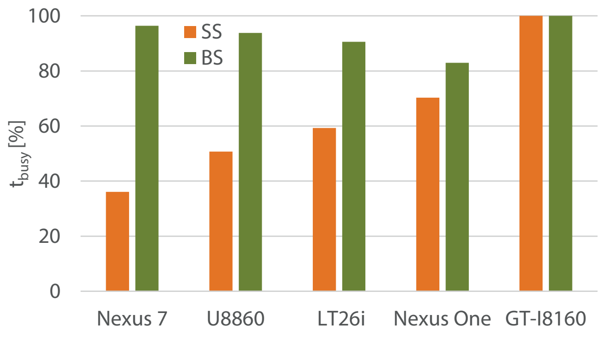


Figure 5. Utilization time using the simple and benchmarking scheduler

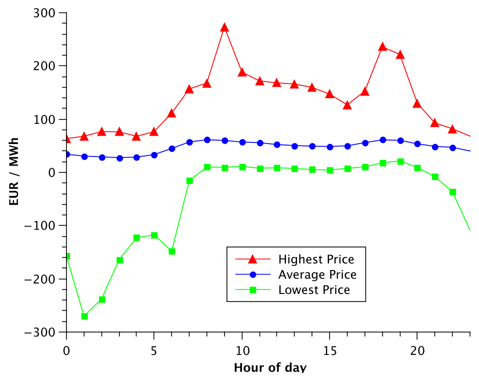


Figure 6. Average EPEX spot prices during weekdays in 2012

with mobile devices. Detailed measurements of the energy consumed by different components in tablets during different usage scenarios has been performed by the Anandtech website using equipment provided by Intel[8][9]. Measurements have been done with platforms using the Nvidia Tegra3 (Coretex A9), Qualcomm APQ8060A (Krait) and Intel Atom Z2760 SoCs.

The Tegra3 is a widespread Quad Cortex-A9 architecture that is also used in the Nexus 7 tablets used for the evaluation, while the Qualcomm developed Krait microarchitecture marks the current high end segment in mobile computation. For the Tegra3 Anandtech reports an average 70.2 mW idle power consumption. For the purpose of looking at the feasibility of computing on mobile devices it can be said that all architectures use significantly below 100 mW when idle. The CPU intensive Sunspider Javascript benchmark is used in [9] to stress the SoC. Power consumption for all architectures goes up into the 1 W area.

IV. Energy spot market scheduling

A. *Electricity Prices*

While most consumers have electricity contracts that let them pay a fixed price per kWh, the prices on the market between electricity producers and brokers vary much more. Normally, a market should find a balance between producers and consumers, between supply and demand. For example usually during the night hours much less electricity is used, because businesses are closed and people sleep. It is common practice that electricity providers sell this off-peak power to consumers for a cheaper price. The reason is, that base load power stations such as big coal and nuclear power plants can not ramp their electricity production up or down very fast. Therefore, they also generate significant amounts of electricity in low demand situations. In recent years, the mismatches between supply and demand have gotten more severe, especially in Germany, due to the growth of renewable energies combined with aging grids already operating at their maximum capacity: There might be lots of wind during the night. Because base load power stations still produce energy there is an excessive supply of electricity which drives prices down. In order to match supply and demand, market mechanisms have been established. The currently largest energy exchange in central Europe is the European Energy Exchange (EEX)[1] located in Germany. In 2011 1,075 TWh have been traded through the EEX. A subsidy of the EEX, EPEXSpot runs a spot market for electricity. On EPEX contracts can be made up to 45 minutes before the start of an hour. 314 TWh of electricity have been traded through the EPEX spot market in 2011. EPEX is the market closest to the generation of electricity. If the supply or demand side changes, this is immediately reflected in the prices.

To get an idea about price fluctuations, in Figure 6 we plot the average prices for electricity during the hours of a working day in 2012. It can be seen, that average prices tend to be lower during the late night and early morning hours. However, the variances between specific days can be huge as can be seen by the minimal and maximal average prices for each hour. For example the lowest peak at -270.11 per MWh EUR for 01:00 clock results from Tuesday, 25th of December when EPEX prices fell to an all-time low in 2012. The lowest observed price in that hour was 473 EUR/MWh. A total of 1,774.0 MWh have been traded in the hours between 01:00 and 02:00 clock yielding an average price of -270.11 EUR/MWh. That is right: At the right time you can actually receive money for being able to consume electricity.

Figure 7 takes a more detailed look at the spot market prices for each day in 2012. There are two observations that can be made: First, there is no recognizable pattern, and secondly there is enough dynamic in the market that promises an economic benefit if you are flexible enough to shift energy usage to times of low prices. The *average* realized prices achieved each day vary between EUR -46,50 and EUR 139,63 per MWh.

B. *Shifting loads*

Shifting loads in order to save money and helping to stabilize the grid has already been extensively researched for home appliances[10][11] or industrial applications[12]. Compared to these load shifting scenarios, technically, for IT systems it is much easier to shift loads: Most equipment can go from low to high power states in a matter of milliseconds. Even cold reserves that need to be booted up, can be ready within

[1]http://www.eex.com/en/

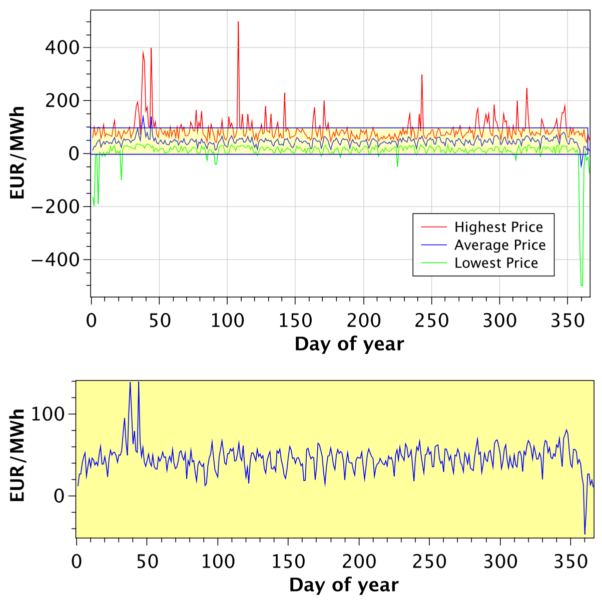


Figure 7. EPEX spot prices in 2012

minutes. This is much faster than most industrial processes and there is less inertia to be considered than for example load shifting a fridge. For many IT processes, the time or computation power needed is known beforehand, and scheduling mechanisms to meet deadlines is a very well researched topic. Most technologies to profit from highly volatile energy prices are already in place. Being able to use surplus energy at low prices, makes the efficiency of the employed hardware less important. Even if a mobile phone or vintage hardware can not match a new server MFlop/J wise it makes sense to use them at times of low energy prices, making it economic to use hardware for a longer time. Ecologically this helps in setting off the environmental cost induced during the production of a device.

C. *Getting access to spot market prices*

While the detailed construction mechanisms of energy markets are out of scope for this paper, and out of the expertise of the authors, we want to sketch a basic business model allowing businesses to profit from the energy spot market without being a participant themselves. Basically an electricity provider with an EPEX license is needed. Instead of offering standard static pricing plans, this provider will add some service charge on top of market prices and offer hourly changing rates to the customer. Based on the price given for the next hour the user will decide how much energy he wants to use in the coming hour. Hybrid models can be conceived where a consumer will still order some electricity for its expected baseload using longer-term contracts and use the spot market pricing model only for consumption above a predetermined threshold. Alternatively, a customer might opt to install two separate individually metered circuits. In the following subsection we show how this model can be combined with a very simple scheduling mechanism to cost-optimize long running computations.

D. *Cost-optimizing energy scheduling*

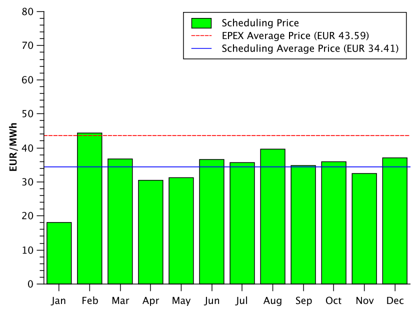
In this simulation we take a look at the savings that can be achieved by shifting the usage of energy to cheaper times. For this experiment we assume that a company has a monthly task (for example a billing process) that needs to be finished at the 15th of each given month. To simplify analysis, we will not look into the amount of computation needed for the calculation nor the exact devices. Instead, we assume a fixed amount of time needed for the calculation given the average amount suitable of devices online and available at any given time. The employed scheduling is simple: If prices fall below threshold t, electricity will be used for that hour. To prevent deadline misses, the prices will be ignored if the deadline would be missed otherwise.

In the first simulation we assume the necessary computation devices to be available 24/7 and assume 72 hours of raw computation time. The goal is to beat the average german spot market energy price (43.59 EUR/MWh in 2012). The results can be seen in Figure 8(a) and 8(b) for $t = 50$ and $t = 30$ respectively. It can be seen, that for some months the average price spent will be above t. This happens, when the scheduler did not find enough suitable timeslots before the deadline approaches and demands to pay any price. For both thresholds it can be seen that the achieved average price even exceeds the yearly EPEX average in February. This is due to unusually high prices in February that can also be seen in figure 7. In general the higher t yields a slightly higher average price, but also smoothens the price variances.

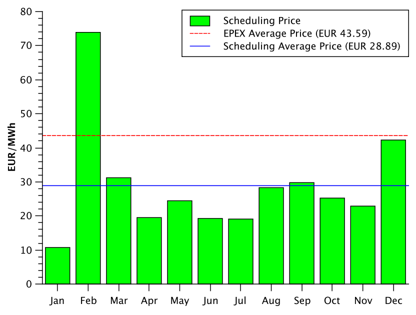
Figure 8(c) shows the situation when only considering working times, which are relevant when using mobile devices that belong to employees and are taken home each day. For this we assume devices to be present between 08:00 and 18:00 on weekdays. During the working hours the EPEX spot price in 2012 was EUR 54.10. We use a threshold of $t = 50$. The scheduling achieves an average price of EUR 47,35 over the year. Probably more advanced scheduling mechanisms would be able to achieve consistently lower prices without the spikes during more expensive months. However, even the simple scheduling presented here can achieve significant savings provided an electricity provider with a suitable billing model steps up, linking consumers more closely to volatile energy markets.

V. Counterpoint: Does it all make sense?

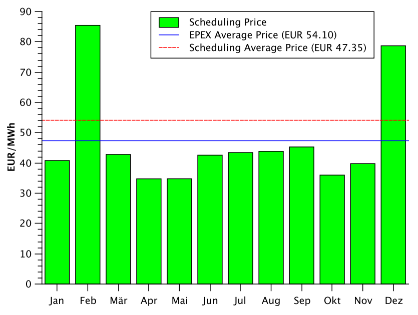
The results in the last section demonstrated the feasibility of load-shifting based on energy prices. However, since some high-end servers might be more energy efficient than mobile devices and are available 24/7, it might be argued that using mobile devices does not make sense in light of this. This however defies the most important point regarding mobile devices: *They are there*. Whether they are used or not, people will still bring them and probably plug them in at work. While the extra energy spent for calculation might be more efficiently turned into computation using specialized servers, it is important to note that the total cost of owning and operating a server far exceeds the cost of energy used during computation: The server needs to be payed for, it takes up valuable space and possibly accounts for some extra costs



(a) 24/7 operation, price threshold t=EUR 50



(b) 24/7 operation price threshold t=EUR 30



(c) Working hours operation, price threshold t=EUR 50

Figure 8. Cost optimizing energy scheduling

due to air conditioning. On the other hand, mobile devices impose no investment costs, because either employees bought them themselves in an BYOD scheme (a recent study predicts, that by 2017 50% of all employees will bring their own devices to work[13]), or the costs are already justified and accounted for due to their primary use: Communication needs of a business' employees. Therefore, being able to offset a fraction of classical IT infrastructure by mobile devices makes economic sense. In the few cases where the deadlines are at risk of being missed, because the available computing power provided by mobile devices is not predictable exactly, an architecture like CANDIS can seamlessly switch computation to standard hardware. If the usecase is carefully planned, these situations should not occur too frequently, so that in the few cases where some extra computation power is needed, the work might be shifted to a pay-per-use cloud provider.

VI. Conclusion

We presented the CANDIS framework for distributing computing tasks to a cloud of mobile devices. CANDIS is available as open source[2]. It can partition tasks based on the computation power of the participating clients. Its architecture allows it to be used on mobile devices using the Android operating system as well as on common server hardware. Using mobile devices makes economic and ecologic sense: As they are already available only the additional energy used for computation needs to be considered as cost factor, while being able to replace a server with mobile devices results into real cost savings for a business. Ecologically, this improves the ecologic balance of mobile devices by offsetting the energy and materials expended at production with more intensive usage while completely eliminating the ecologic burden of compute servers that are replaced by mobile devices. We have shown that with cost-aware scheduling CANDIS can profit from varying electricity prices, minimizing the costs for the operation of mobile clouds.

In some ways, mobile device clouds are to computation what renewables are to today's electricity generation: They can not supply the whole demand yet, but the resource is mostly available with a small amount of uncertainty, continuously renewing itself and just waits to be harvested. The potential will increase, and there will be more and more usecases were you really can have your cake and eat it too!

[2]https://github.com/ejoerns/candis

References

[1] F. Busching, S. Schildt, and L. Wolf, "DroidCluster: Towards Smartphone Cluster Computing – The Streets are Paved with Potential Computer Clusters," in *2012 32nd International Conference on Distributed Computing Systems Workshops*. IEEE, Jun. 2012, pp. 114–117. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6258144

[2] E. E. Marinelli, "Hyrax: Cloud Computing on Mobile Devices using MapReduce," Master's thesis, Carnegie Mellon University, Sep. 2009. [Online]. Available: http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA512601

[3] NativeBOINC - Native port of the BOINC client to Android devices. [Online]. Available: http://nativeboinc.org

[4] J. R. Eastlack, "Extending volunteer computing to mobile devices," Master's thesis, New Mexico State University, 2011. [Online]. Available: http://onlinelibrary.wiley.com/doi/10.1002/cbdv.200490137/abstracthttp://boinc.berkeley.edu/Thesis_Eastlack_Nov09.pdf

[5] J. Yu, E. Williams, and M. Ju, "Analysis of material and energy consumption of mobile phones in China," *Energy Policy*, vol. 38, no. 8, pp. 4135–4141, Aug. 2010. [Online]. Available: http://dx.doi.org/10.1016/j.enpol.2010.03.041

[6] K. Marx, "Critique of the gotha programme," 1875.

[7] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone," *Proceedings of the 2010 USENIX conference on ...*, 2010. [Online]. Available: http://www.usenix.org/event/usenix10/tech/full_papers/Carroll.pdf

[8] A. L. Shimpi, "The x86 Power Myth Busted: In-Depth Clover Trail Power Analysis," December 2012. [Online]. Available: http://www.anandtech.com/show/6529/busting-the-x86-power-myth-indepth-clover-trail-power-analysis

[9] ——, "The ARM vs x86 Wars Have Begun: In-Depth Power Analysis of Atom, Krait & Cortex A15," January 2013. [Online]. Available: http://anandtech.com/show/6536/arm-vs-x86-the-real-showdown

[10] K. Wacks, "Utility load management using home automation," *IEEE Transactions on Consumer Electronics*, vol. 37, no. 2, pp. 168–174, May 1991. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=79325

[11] L. Ha, S. Ploix, E. Zamai, and M. Jacomino, "Tabu search for the optimization of household energy consumption," in *2006 IEEE International Conference on Information Reuse & Integration*. IEEE, Sep. 2006, pp. 86–92. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4018470

[12] S. Ashok and R. Banerjee, "Load-management applications for the industrial sector," *Applied Energy*, vol. 66, no. 2, pp. 105–111, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0306261999001257

[13] D. A. Willis. (2013) Bring Your Own Device: The Facts and the Future. Gartner Inc.