Tools for Empirical and Operational Analysis of Mobile Offloading in Loop-Based Applications

Alexandru-Corneliu OLTEANU, Nicolae ŢĂPUŞ
University "Politehnica" of Bucharest, Romania
alexandru.olteanu@cs.pub.ro, nicolae.tapus@cs.pub.ro

Offloading for mobile devices is an increasingly popular research topic, matching the popularity mobile devices have in the general population. Studying mobile offloading is challenging because of device and application heterogeneity. However, we believe that focusing on a specific type of application can bring advances in offloading for mobile devices, while still keeping a wide range of applicability. In this paper we focus on loop-based applications, in which most of the functionality is given by iterating an execution loop. We model the main loop of the application with a graph that consists of a cycle and propose an operational analysis to study offloading on this model. We also propose a testbed based on a real-world application to empirically evaluate offloading. We conduct performance evaluation using both tools and compare the analytical and empirical results.
Keywords: *Mobile Computing, Offloading, Cloud Computing, Performance Evaluation*

1 Introduction

Modern handheld devices, such as smartphones and tablets, offer portability, increased computational power, and communication capabilities. Thus, they are becoming an attractive option for users to interact with each other and with their environment.

The convergence of mobile and distributed computing has been studied for a number of years, with results in system design [1] [2], job scheduling [3] [4], resource discovery [5] [6] and so on. Mobile integration with various other types of computing takes many forms, such as mobile cloud computing, offloading, delegation, cyber foraging, and data staging. Offloading is a form of transferring tasks of various granularities to remote resources. Offloading and delegation are very similar approaches to use remote resources and sometimes they are considered to be complementary, as in the work published by Flores [7]. Cyber Foraging is an opportunistic approach of using remote resources from mobiles. Satyanarayanan [8] introduced this concept in 2001 as a pervasive computing technique, and work within the same research team [9] led to a scripting language for cyber foraging. Verbelen et al. [2] introduced AIOLOS, a middleware to improve mobile application performance through cyber foraging and Kristensen [10] introduced scheduling concepts in the topic.

Offloading for mobile devices is an increasingly popular research topic, matching the popularity mobile devices have in the general population. With the first research efforts targeted specifically on mobile devices dating back in the 1990's, in the past couple of years a vast material on offloading for mobile devices has been published.

Several offloading systems have been proposed, as middleware, frameworks, or services. However, we find that few solutions reach the point to have a big impact on live systems and applications. *Studying mobile offloading is challenging because of device and application heterogeneity.* However, we believe that focusing on a specific type of application can bring advances in offloading for mobile devices, while still keeping a wide range of applicability. Thus, our approach is to conduct an application domain exploration and select a family of applications on which to conduct analysis and evaluation of various offloading mechanisms.

In this paper, we conduct experimental and analytical evaluation for mobile offloading, as a step towards an integrated offloading system [11]. We also compare the two sets of results to study the trade-offs of the two methods.

DOI: 10.12948/issn14531305/17.4.2013.01

2 Related Work

Analyzing power consumption is an interesting research topic, as shown by various projects that can be found in the literature. Caroll and Heiser [12] design multiple micro-benchmarks to associate the power costs to modules of a mobile system. They try to determine the power consumption of different parts like CPU, GSM and WiFi. Zhang et al. [13] describe a power module construction technique that they use to characterize 3G, GSM, WiFi, CPU and screen, and to introduce PowerTutor, an Android application that can use these models for power consumption estimation on any device. We use such tools to estimate the power consumption of different components, but we found little solutions on estimating power consumption from the Bluetooth radio.

Balasubramanian et al. [14] show that 3G, GSM and WiFi incur a high tail energy overhead. Pering et al. [15] describe methods to reduce the power consumption by switching between Bluetooth and WiFi. The Bluetooth module has power consumption as much as 10 times lower than WiFi. WiFi is intended for high-bandwidth and 100 meters coverage while Bluetooth is designed for low-bandwidth and a coverage of 10 meters. The authors also describe that power consumption in idle mode compared to active mode is 4 times smaller for WiFi and 6 to 10 times smaller for Bluetooth. We find this significant as it justifies the need of advanced algorithms that power down these interfaces when communication is not necessary.

Various other papers try to determine algorithms and technologies for reducing energy consumption, by modeling optimum power as a Nash equilibrium [16], by employing back-off methods for synchronization [17], or introducing an active-sleep duty cycle [18]. We propose an adaptive algorithm for reducing the polling rate in location-aware applications, leveraging the fact that new queries are not necessary if the location has not changed significantly.

In our work we apply techniques of Computer Systems Performance Analysis that are well established in the field, providing very useful tools in the form of analytical modeling. Kleinrock [19] provides a very thorough introduction in the modeling of stochastic systems of flow using queuing theory. By understanding the stochastic processes that describe the arriving stream and the structure and discipline of the service facility, one can mathematically obtain measures of performance and effectiveness. Menasce [20] also uses queuing networks to obtain descriptive models for various types of systems, which serve as basis for quantifying performance models. Jain [21] offers a thorough reference to fundamental and practical engineering principles of computer systems evaluation and King [22] extends the principles on communication topics.

3 Tools for Operational and Empirical Analysis of Loop-Based Applications

In our study, we focus on loop-based applications. A loop-based application is one in which most of the functionality is given by iterating an execution loop. All the Online Social Applications, for which we built a Workload Model in [23], have such a loop, as it can be seen in the State Transition Diagram, the component that models the states in which a mobile online social application can be. Moreover, for Online Social Applications, independent of their technology: Tightly Coupled Simulations, Web 2.0 or Streaming, some amount of functionality already occurs remotely, this being one of the criteria for which we chose them to conduct our studies.

3.1 Formalism for Operational Analysis of Mobile Offloading

We model the main loop of the application with a graph that consists of a cycle, with stages numbered from 1 to n, as depicted in Figure 1. An example of such a loop is also given in Figure 1, showing the main processing loop of OpenTTD that consists of stages such as input capture, synchronization on the server, simulation, and rendering.

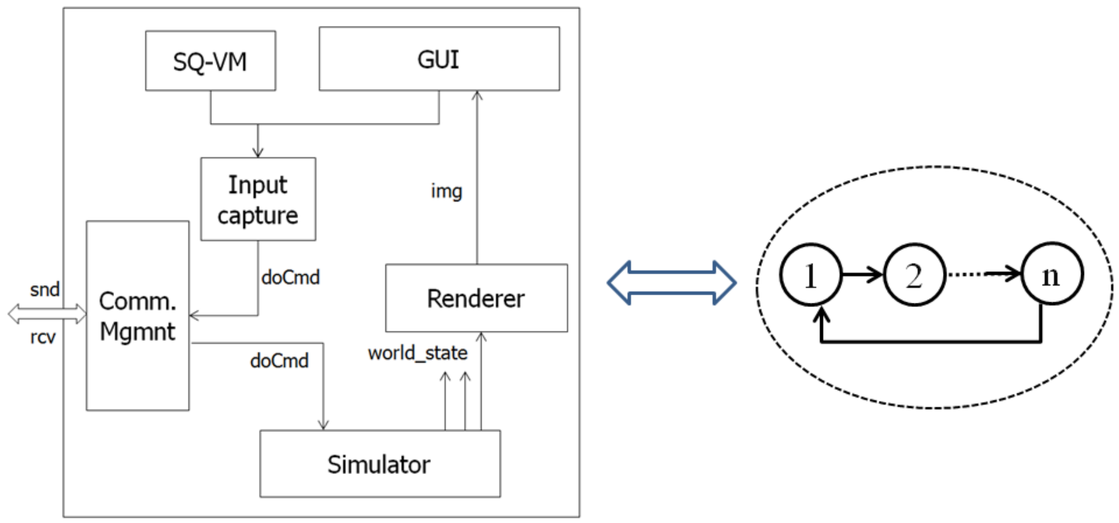

Fig. 1. Graph model of OpenTTD (left) and loop-based applications in general (right)

Based on this loop, and derived from the metrics and the benefit assessment models described in our taxonomy [26] we assess the benefit of offloading in terms of time needed to iterate the processing loop as:

$$T = \sum_{i=1}^{n} T^{i}_{p|r} + \sum_{i=1}^{n} \frac{Q^{i}}{B(s,d)}$$

where:

- T - time needed to perform computation or a data transfer
- p/r - local/remote processing
- Q - quantity of data to be transferred
- B(s,d) - bandwidth when transferring data from source *s* to destination *d*

In the remainder of this section we make some simplifications, to keep the formulas concise, at the expense of some precision. First, we consider the download speed and the upload speed roughly the same size:

$$B(local, remote) = B(remote, local) = B$$

We also consider that passing data among local stages, as well as passing data among remote stages, is much quicker than passing data from local to remote and vice versa, and, therefore, we consider it as infinite:

$$B(local, local) \to \infty, B(remote, remote) \to \infty$$

In this analysis, we address loop-based applications and we model their processing with a cyclic graph. We are also focused on applications that already have some form of communication. We depict this by having a stage being processed in the environment (see Figure 2a), as, for example, OpenTTD has the synchronization stage being processed on the server.

By offloading components we refer to moving the processing of additional stages in the environment. For example, in OpenTTD, besides doing the synchronization, we might remote the execution of the simulation as well. We generally note with *k* the stage where local processing turns into remote processing and with *l* the stage where processing returns on the device. Offloading is thus represented by moving indexes *k* and *l* within the loop, as described by Figure 2b. We use this formalism to express the offloading decision logic, for the three main benefit assessment models: performance, energy and cost.

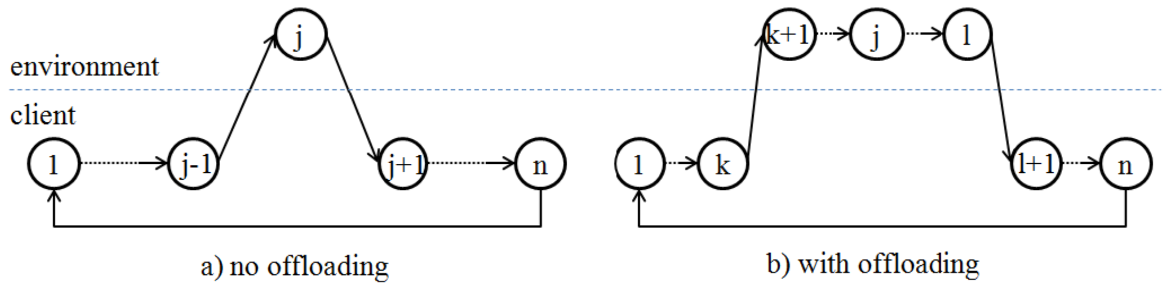

Fig. 2. Graph model of loop-based applications when offloading

Inspired by [24] [25], we express performance as the time it takes to perform the loop:

$$T(k,l) = \sum_{i=1}^{k} T_p^i + T_t^k + \sum_{i=k+1}^{l} T_r^i + T_t^l + \sum_{i=l+1}^{n} T_p^i$$

where T_p^i is the local processing time of stage *i*, T_r^i is the remote processing time of stage *i*, and T_t^i is the transferring time of output data from stage *i*. Transmission time can be expressed as the quantity of data transmitted divided by the transmission speed. At this stage we make the simplification that the sending and receiving speeds are similar. Local and remote processing times can be measured directly, and can also be expressed as the quantity of code to be executed divided by the CPU speed. The equation becomes:

$$T(k,l) = \sum_{i=1}^{k} T_p^i + \sum_{i=k+1}^{l} T_r^i + \sum_{i=l+1}^{n} T_p^i + \frac{Q^k + Q^l}{B}$$

In general, offloading is beneficial if remote processing has a better performance than local processing, or, equivalently, if the penalty for transmitting the data to the remote resource is less than gain in time obtained from using a remote resource more capable than the local one, expressed by the inequality:

$$\frac{Q^k + Q^l}{B} < \sum_{i=k+1}^{l} (T_p^i - T_r^i)$$

Thus, the offload decision becomes the optimization problem expressed as:

$$T_{min} = \min_{k,l} \left\{ \sum_{i=1}^{k} T_p^i + \sum_{i=k+1}^{l} T_r^i + \sum_{i=l+1}^{n} T_p^i + \frac{Q^k + Q^l}{B} \right\}, 1 < k < j < l < n$$

This operational analysis can be extended for the offloading mechanisms defined in our Exploratory Space, such as partial offloading and parallel offloading [11].

3.2 Design and Implementation of a Testbed for Mobile Offloading

To conduct empirical evaluation of various offloading mechanisms, we design and implement a testbed based on OpenTTD, a popular open-source game, the open-source version of Transport Tycoon Deluxe, a business simulation game developed by Chris Sawyer in 1994.

As an open-source application, OpenTTD has a community of developers that continuously update the game and publish all sources in a repository. Moreover, other developers port the game to other platforms. For example, the Android port by Pelya has hundreds of thousands of active users.

We select OpenTTD because it is a real popular application, which falls in the category of online social applications, implemented as a tightly coupled simulation, for which we have a workload model, as proposed in [23]. Tightly Coupled Simulations are, among Online Social Applications, the type of applications that provide opportunity for the broadest range of experiments, because the largest amount of processing takes place on the mobile device. We conduct static and dynamic analysis on OpenTTD. We use *Vprof* to profile OpenTTD on a Linux system and *Vtune* on an Android system.

The basic functionality of the game consists of iterating a loop, which means we can apply our operational analysis described in Section 3.1.

In multiplayer mode, OpenTTD follows a client-server architecture that currently supports up to 255 simultaneous users on the same map, one of which must host the server. There are some notable efforts to expand that number of users with a Massively Multiplayer Online version of OpenTTD, named At Large. In our testbed, we augment the client-server architecture, by tapping into the normal data flow of the system and adding an observer that also has the ability to control the experiments (see Figure 3). The testbed enables experiments for conducting offloading from various clients to either a cloudlet device, in the same LAN, or with powerful cloud infrastructure, over the Internet.

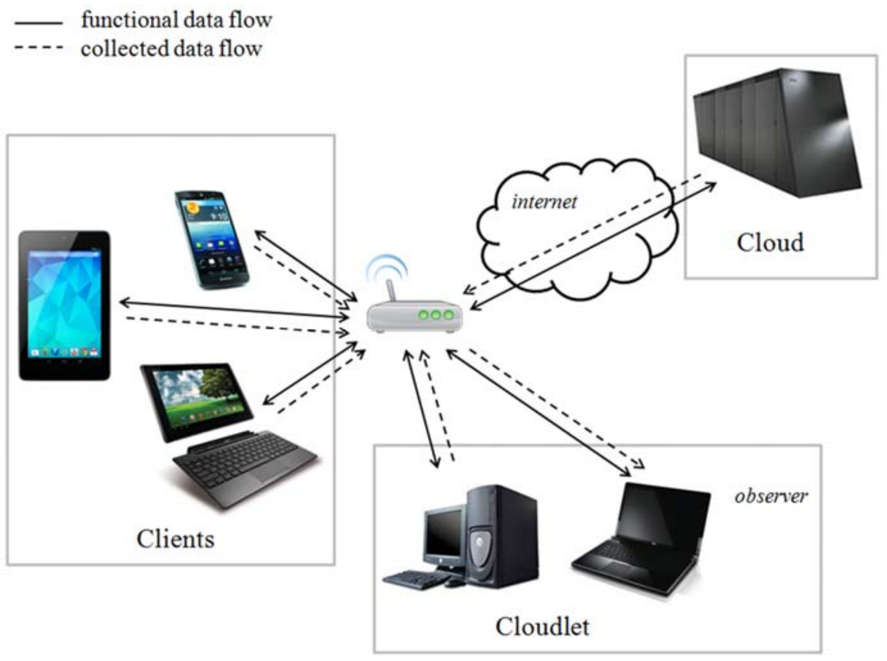

Fig. 3. General Architecture of the Testbed

We implement the testbed by making several changes to the community version. To conduct experiments, we need to repeat them for a number of times and make sure they all behave the same way. Thus we replace the human player with an AI running on the device and our implementation follows its operations on the screen. So, starting a game with the same AI on the same scenario will recreate in each run the same usage patterns and the same operations. To enable running AI players on the server, we adapt the code, starting with the modifications implemented by Otto Visser, from Delft University of Technology. In addition, we also adapt the starting procedure so that we can easily start the game on the mobile device and on the server through scripts. When starting Android applications, it is not possible to give the native code command line arguments, so it was necessary to implement some additional configuration settings in the original configuration file.

We also implement instrumentation in all the components of the architecture, to collect various metrics, as depicted in Figure 4:

- on the device, within OpenTTD: we measure game specific metrics, like frames per second and in-game time, as well as component statistics, in terms of processing time and quantity of data;
- on the device, at application level: we run several profilers, such as *vprof* and *VTune*, and tools, such as *top* and *iftop* for Android, to collect metrics such as CPU load, memory load, and network load;
- on the device, in the kernel: hooks and system calls can be placed for forwarding to higher levels essential information captured from hardware counters on the physical device, like the C-states; we have investigated C-states for a better understanding on CPU loads [27], but we do not detail them in this testbed;
- at the network level, software known as package sniffers, such as Fiddler and Wireshark, capture packets and help with statistics, such as sent and received packets, sent and received bytes, session length, inter-arrival times, and the size of the input and output data;
- on the server side: we measure the number of clients and several hardware-related metrics, such as CPU load, memory load, and network load.

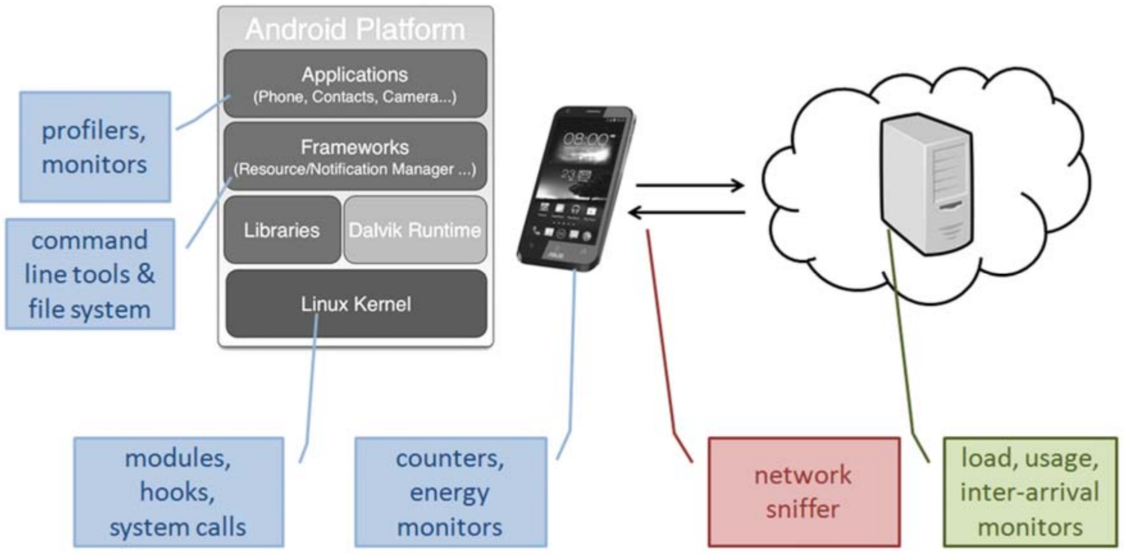
Android Platform
Applications
(Phone, Contacts, Camera...)
Frameworks
(Resource/Notification Manager ...)
Libraries
Dalvik Runtime
Linux Kernel
profilers, monitors
command line tools & file system
modules, hooks, system calls
counters, energy monitors
network sniffer
load, usage, inter-arrival monitors

Fig. 4. Instrumentation in our Offloading System

We use *top*, *iftop* and the *sysfs* to monitor internal resources on the Android device. Network load is monitored using *iftop* and *tcpdump*. We use the tools AWS provide for the cloud servers and simple Linux tools for our cloudlet machines.

The OpenTTD implementation already handles issues such as lagging clients and out of order commands. Each client holds a tick-

based internal counter that serves as a reference for the client in executing commands. A comprehensive error handling system can press the client to speed up on executing older commands, and can even go to the point of kicking out a client that cannot keep the pace. We have created a suite of *bash* scripts that, when run from the observer, trigger the game on the clients and the server, start the measuring tools, wait for the time specified for the experiment and cleanly close all the programs that they started. All communication with the Android clients is done through *adb*, or Android Debug Bridge, a tool offered by Google, which enables file transfers and remote connections to the Android devices. Communication with the other devices in the cloudlet and the cloud is done through *SSH* and *SCP*, two popular communication protocols for remote connections and file transfers, with many clients on both Windows and Linux. At the end of the experiment, all the results are centralized on the observer, where other scripts automatically compute statistics and plot charts using *gnuplot*.
The system does not take a dynamic decision, but is instructed by the observer which tests to run, because we are interested in comparing various mechanisms. We implement several variants of OpenTTD, to offload different amounts of components. We also allow file configuration for varying parameters, such as graphics and user behavior.

4 Performance Evaluation Results

For the empirical evaluation, we use the testbed described in Section 3.2 and for the analytical evaluation we use the operational analysis described in Section 3.1, as well as the workload traces presented in [23].

4.1 Analytical Evaluation Results

We compare offloading the AI input collection stage with no offloading, using the operational analysis proposed in Section 3.1. We also summarize in Table 1 the values we use, based on the values from workload modelling (see [23]).

Table 1. Data used in analytical evaluation

Stage name and number	Local Processing	Remote Processing	Quantity of Data
	Tp [ms]	*Tr* [ms]	*Q* [byte]
Human Input Collection (1)	1.2	0.7	40
AI Input Collection (2)	48.2	4.3	40
Command Synchronisation (3)	188.5	101.9	120
Simulation (4)	9.3	2.6	1 068 576
Rendering (5)	5.7	13.7	4 096 000

Bandwidth
B [Mbps]
8

In the base case, without offloading, *k=j=l=3*, indicating the remote operation of command synchronization, the only stage that needs to be performed remotely:

$$T(3,3) = T_p^1 + T_p^2 + T_r^3 + T_p^4 + T_p^5 + \frac{Q_2 + Q_3}{B} = 166.45ms$$

In a similar way, when offloading Stage 2, the AI input collection, *k=2, j=l=3*, and the formula becomes:

$$T(2,3) = T_p^1 + T_r^2 + T_r^3 + T_p^4 + T_p^5 + \frac{Q_1 + Q_3}{B} = 122.55ms$$

The smaller time for remote processing Stage 2 compared with local processing, as well as the comparable output data size of Stage 1 and Stage 2, make offloading Stage 2 a better option than no offloading.

In comparison, trying to offload Stage 4, the simulation would lead to:

$$T(3,4) = T_p^1 + T_p^2 + T_r^3 + T_r^4 + T_p^5 + \frac{Q_2 + Q_4}{B} = 1178.71ms$$

This would be an example when offloading is not beneficial. Due to the large size of the output data, one iteration of the loop would take more than a second, which means that the client device will be removed from the multiplayer game because it is too slow.

We now consider Stage 2' to describe running 4 AI players instead of 1. Having multiple AI players take a decision during an iteration is an embarrassingly parallel task, thus running *N* AI players on a serial processor would take *N* times the time. This, for *N=4* players, the base case can be represented as:

$$T(3,3) = T_p^1 + T_p^{2'} + T_r^3 + T_p^4 + T_p^5 + \frac{Q_2 + Q_3}{B} = 311.05ms$$

and offloading Stage 2' becomes:

$$T(2',3) = T_p^1 + T_r^{2'} + T_r^3 + T_p^4 + T_p^5 + \frac{Q_1 + Q_3}{B} = 135.45ms$$

Thus, when running 4 AI players locally, the 200 ms loop period is exceeded, and the client will be removed from the multiplayer game. In this case, offloading is necessary, as it brings the iteration time below the 200 ms threshold.

4.2 Empirical Evaluation Results

For this experiment, we explore the benefits of offloading the AI Input Collection component of the game. We represent the AI Input Collection as a stage in the processing loop of OpenTTD (see Section 3.1). Profiling shows that the AI players can consume significant amounts of processing power when computing routes from one city or resource to another, which they do by using algorithms such as A*, especially when the tree representation of the world is large.

Message

Network-Game connection lost

Fig. 5. OpenTTD running on a tablet (left) has to keep up with the rest of the clients, otherwise it will be removed from the game by the server (right)

However, the AI players have a bursty behavior, and compute such routes only when they acknowledge they have enough in-game money. It is therefore interesting to see what impact has offloading the Squirrel Virtual Machine, which runs the AI scripts and offers an AI input.

For this experiment, we use the testbed (see Section 3.2) we propose based on our *openttd-repeatable* implementation, which allows us to repeat the same scenario multiple times. We run the client on an Asus TF101 2-core @1GHz running Android 4.0.3 and the server on a Sony Vaio 4-core @2.3GHz running Ubuntu 10.04. There is one AI player running on the client to simulate a real player.

Using this experiment, we explore one of the four main offloading mechanisms identified in our Exploratory Space [11]. We compare not offloading anything, running AIs on the client device, with offloading the AI component, which is running the AIs on the server. We use our testbed and, as offloading target, the cloudlet represented by a Sony Vaio laptop.

The scenario we investigate is that a human player has a match against a number of AIs. We have started with 16 AIs, but running the game locally with that many AIs proves to be unplayable on our device. So, to collect data for a control test run, we repeatedly decreased the number of AIs until the game became usable on our device. We have settled to 4 AIs, that we picked from the most popular in the community, namely *OtviAI*, *AIAI*, *ChooChoo* and *Chopper*.

Figure 6 reports two of the metrics we collect, showing the first 10 minutes of the experiment. The iteration time is a performance metric, and shows how long an iteration of the data takes to be processed through the whole loop. The CPU Load is also a performance metric, which shows how much of the CPU is used in the system.

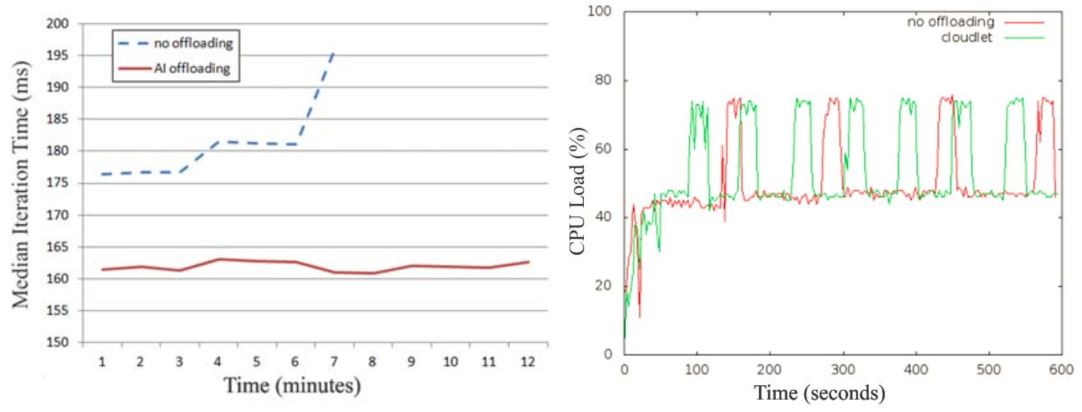

Fig. 6. Experimental results showing iteration time (left) and CPU load (right)

We log the iteration time in our modified version of OpenTTD every time an iteration completes. With roughly ten iterations per second, during a 10 minute window we would have roughly 6000 readings. To present our findings in a friendly way, we aggregate the readings on a per-minute basis, and we only report the median value in the left-hand chart in Figure 6. It can be seen how in the no-offloading version, the iteration time increases in the first 5-7 minutes of the game and gets above the threshold in the 8th minute, when the client is removed from the multiplayer. On the other hand, the median of the offloaded version is not affected much and stays all the time in the range of 160-165ms.

The left-hand chart in Figure 6 shows a CPU load of 40-50% most of the time, which corresponds to a high usage of one of the two cores. The spikes are triggered by auto-saves, which, in our setting, take place once per in-game month. The auto-saves are performed on a separate thread, which explains why the CPU load exceeds the 50% threshold.

Both charts indicate that, without offloading, the game slows down to compensate for the lack of processing power of the client device.

4.3 Comparison of Analytical and Empirical Evaluation Results

In this section we compare the results we obtained during our experiments with the ones obtained by applying the operational analysis, to validate our understanding on the two offloading mechanism. In our operational analysis, we refer to a couple of performance metrics: *total iteration time* and *quantity of data transmitted over the network.*

Table 2 summarizes the significant figures in our comparison. We consider three cases:

- *Base Case 1: No network* - we run OpenTTD in single-player mode on the mobile client; no communication is performed by OpenTTD, but it still has to run 4 AI players;
- *Base Case 2: No offloading* - we run OpenTTD in multiplayer mode, on the mobile client and the laptop; now the client needs to send commands to the server for synchronization, but nothing else is offloaded;
- *Experiment: AI offloading* - we run OpenTTD in multiplayer mode, on the mobile client, and on the laptop several clients run an AI player each, but they are all forced to work on a single core, so they operate in serial.

In the base cases, the game stops in the middle of the experiment, being kicked out by the server as being too slow, so we are not able to compute an average consistent with the method from the other experiments. However, since the client was removed from the game automatically, it must have exceeded the threshold of 200ms repeatedly.

For quantity of data, we sum up the data transmitted over the network, both sent and received, over a period of 8 minutes that represents the maximum period for which we have data in all 4 cases. In this sum, we exclude the initialization phase, which does not belong to the processing loop. To estimate the analytical result, we assume an average of 10 iterations per second, and thus a total of 4800 of round-trips, which include sending to server data of 40 bytes each and receiving 120 bytes.

Table 2. Comparison between empirical and analytical results

Scenario		Average iteration time			Total data in 8 min	
		T [ms]			Q [bytes]	
		empirical	analytical		empirical	analytical
No network		>200	265.64		64 211	0
No offloading		>200	311.05		1 307 300	768 000
AI offloading		155.7	135.45		1 463 240	768 000

It can be noted that our observation, that running 4 AI players is prohibitive for clients that do not offload, is consistent for both methods. However, this is also the cause why we were not able to assess empirically the exact performance for the base cases.

In terms of data transfers, the analytical results do not match the empirical results, probably because in reality there is more information passed between client and server than the commands. In the no network case, we can see an overhead of 64kB of data that seems to be caused by other applications, since OpenTTD has no network enabled. We consider this background noise, having two orders of magnitude less than the real messages, and relatively constant, as it is probably caused by services that synchronize in the background. Even when subtracting this value from the empirical values, they still remain significantly higher than the ones from the analytical evaluation. Therefore, it seems that OpenTTD also transmits other data and this data is larger when offloading AIs. However, data for serial and parallel AI offloading does not differ much, which is consistent with the analytical results.

We find that the operational analysis is a promising tool. Although the actual values are significantly different than the empirical ones, many of the ideas behind the offloading mechanisms are supported by both sets of results.

5 Conclusions

The recent popularity of smart mobile devices is motivating many manufacturers to produce devices for many types of consumers, thus leading to orders of magnitude in heterogeneity. The processing unit may vary from single-core CPU, to quad-core CPU, and even to hybrid architectures that include a multi-core CPU and a GPU. The battery lifetime may vary from tens to hundreds of hours in standby, and is greatly influenced by user behavior, as intense device usage can reduce its battery life to barely a few hours. The *main challenge* we are facing is to assess the benefits of offloading under such heterogeneity.

We believe that focusing on a specific type of application can bring advances in offloading for mobile devices, while still keeping a wide range of applicability. The idea of applying this research specifically to loop-based applications is *new*. From the point of view of the technology used to implement the applications, our approach covers both web-based applications, applications that send messages with the user actions to be performed on the server, and tightly coupled simulations, which do some heavy processing on the device and usually communicate only control messages. Through our workload model (see [23]) and our operational analysis (Section 3.1) we generalize both types of technologies and refer to any kind of loop-based application. We conduct an innovative design space exploration, based on the exploratory space we propose in [26], through empirical and analytical evaluation.

In this paper, we propose a formalism that can be used to conduct operational analysis of the offloading mechanisms in the Exploratory Space (in Section 3.1), applied to any application that functions by iterating over a processing loop. We also propose a testbed for empirical evaluation (in Section 3.2). In Section 4.1 and Section 4.2 we present the results of analytical and empirical evaluation, respectively. Finally, in Section 4.3 we compare analytical evaluation results with empirical evaluation results.

We find that the operational analysis is a promising tool. Although the actual values are significantly different than the empirical ones, many of the ideas behind the offloading mechanisms are supported by both sets of results.

Acknowledgements

The authors would like to thank Alexandru Iosup and Otto Visser from Delft University of Technology for their invaluable support in designing the offloading system and for their effort to implement a massive multiplayer online version of OpenTTD, which served as basis for the first steps in the implementation of the testbed.

This work is funded by the Sectoral Operational Programme Human Resources Development 2007-2013 of the Romanian Ministry of Labour, Family, and Social Protection through the Financial Agreement POSDRU/107/1.5/S/76909.

References

[1] B.G. Chun et al. "Clonecloud: elastic execution between mobile device and cloud." *Proceedings of the sixth conference on Computer systems*. ACM, 2011.

[2] T. Verbelen, et al. "AIOLOS: Middleware for improving mobile application performance through cyber foraging." *Journal of Systems and Software* 85.11 (2012): 2629-2639.

[3] J. Ghosh, et al. "On profiling mobility and predicting locations of wireless users." *Proceedings of the 2nd international workshop on Multi-hop ad hoc networks: from theory to reality*. ACM, 2006.

[4] K.A. Hummel, and J. Gerda, "A robust decentralized job scheduling approach for mobile peers in ad-hoc grids." *Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE International Symposium on. IEEE*, 2007.

[5] D. Bruneo, et al. "Communication paradigms for mobile grid users." *Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on. IEEE*, 2003.

[6] A.T. Gomes, et al. "DICHOTOMY: A resource discovery and scheduling protocol for multihop ad hoc mobile grids." *Cluster Computing and the Grid, 2007. CCGRID 2007. Seventh IEEE Interna-*

tional Symposium on. IEEE, 2007.

[7] H. Flores, and N.S. Satish, "Adaptive Code Offloading and Resource-intensive Task Delegation for Mobile Cloud Applications."

[8] M. Satyanarayanan et al. "The case for vm-based cloudlets in mobile computing." *Pervasive Computing*, IEEE 8.4 (2009): 14-23.

[9] R.K. Balan, et al. "Simplifying cyber foraging for mobile devices." *Proceedings of the 5th international conference on Mobile systems, applications and services*. ACM, 2007.

[10] M. D. Kristensen and O. B. Niels, "Scheduling and development support in the scavenger cyber foraging system." *Pervasive and Mobile Computing* 6.6 (2010): 677-692.

[11] A.C. Olteanu, A. Iosup and N. Ţăpuş, Extending the capabilities of mobile devices for online social applications through cloud offloading. In *The 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, pg. 160–163

[12] A. Carroll and H. Gernot, "An analysis of power consumption in a smartphone." *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*. 2010.

[13] L. Zhang et al. "Accurate online power estimation and automatic battery behavior based power model generation for smartphones." *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. ACM, 2010.

[14] N. Balasubramanian, A. Balasubramanian and A. Venkataramani, "Energy consumption in mobile phones: a measurement study and implications for network applications." *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*. ACM, 2009.

[15] T. Pering et al. "Coolspots: reducing the power consumption of wireless mobile devices with multiple radio interfaces." *Proceedings of the 4th international conference on Mobile systems, applications and services*. ACM, 2006.

[16] W. Yu, G. Ginis and J.M. Cioffi, "An adaptive multiuser power control algorithm for VDSL." *Global Telecommunications Conference, 2001. GLOBECOM'01. IEEE*. Vol. 1. IEEE, 2001.

[17] A. Agarwal and M. Cherian, Adaptive backoff synchronization techniques. Vol. 17. No. 3. ACM, 1989.

[18] T. Van Dam, and K. Langendoen, "An adaptive energy-efficient MAC protocol for wireless sensor networks." *Proceedings of the 1st international conference on Embedded networked sensor systems*. ACM, 2003.

[19] L. Kleinrock, *Theory, volume 1, Queueing systems*. Wiley-interscience, 1975.

[20] D. A. Menascâe et al. *Performance by design: computer capacity planning by example*. Prentice Hall Professional, 2004.

[21] R. Jain, *The art of computer systems performance analysis*. Vol. 182. Chichester: John Wiley & Sons, 1991.

[22] P. J. B. King, *Computer and communication systems performance modelling*. Prentice Hall International (UK) Ltd., 1990.

[23] A.C. Olteanu, A. Iosup and N. Ţăpuş. "Towards a workload model for online social applications: ICPE 2013 work-in-progress paper." *Proceedings of the ACM/SPEC international conference on International conference on performance engineering*. ACM, 2013.

[24] M. Ferber et al. "Resource allocation for cloud-assisted mobile applications." *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*. IEEE, 2012.

[25] I. Ivan and C. Ciurea. "Using very large volume data sets for collaborative systems study." *Informatica Economica* Journal 13.1 (2009).

[26] A.C. Olteanu and N. Ţăpuş, “Offloading for mobile devices: A survey.” *UPB Scientific Bulletin*

[27] A. Gherghina, A.C. Olteanu, and N. Tapus. "Measuring performance and energy

consumption when offloading from mobile devices." *Systems and Computer Science (ICSCS), 2013 2nd International Conference on*. IEEE, 2013.

Alexandru-Corneliu OLTEANU has graduated University Politehnica of Bucharest, the Computer Science and Engineering Department in 2009, with a graduating project at the National University of Singapore on GPU computing. He is currently a Teaching Assistant and PhD student in the same department within University Politehnica of Bucharest. As a Teaching Assistant, he specializes on distributed systems and mobile computing. His PhD studies focus on cloud offloading for mobile devices, with internships at Delft University of Technology, the Netherlands, and University of Applied Sciences in Dresden, Germany. Alexandru led, since 2007, the organization team for the ACM ICPC South-Eastern Europe Programming Contest, which gathers more than 150 participants from 8 countries from all over South-Eastern. He also participated as researcher in several national and international programs from 2010 to present.

Nicolae ŢĂPUŞ has graduated the Politehnic Institute of Bucharest in 1972 specializing in computers. He finished his PhD in Computer Science in 1982. Professor Ţăpuş is currently a professor and doctoral supervisor at the University Politehnica of Bucharest, the Head of the Computer Science and Engineering Department and Vice-President of the university's Senate. He is the author of numerous studies and also the project director of numerous research projects in the fields of Computer Architecture, Computer Networks, Wireless Sensor Networks, Personal Computers, Simulation Languages, publishing more than 116 papers in the country and abroad, including 9 books and 12 textbooks. He got the Traian Vuia Award of the Romanian Academy (1977) and Scientific Creativity Award of the Ministry of Education (1984). Professor Ţăpuş is a Senior Member of IEEE and Chairman of the Romanian Chapter, as well as CATC Coordinator at the Romanian CISCO. He led and participated in numerous national and international programs as Project Director from 1994 to present.

DOI: 10.12948/issn14531305/17.4.2013.01