PRACTICAL ANALYSIS OF CLOUD OFFLOADING STRATEGIES FOR HEAVY AI PROCESSING MOBILE GAMES

Adriana DRĂGHICI[1], Andrei VOINESCU[2], Nicolae ȚĂPUȘ[3]

Modern mobile devices, such as smartphones and tablets, are offering a wide range of entertainment applications that require an increasing amount of computational and storage resources. To address such needs, developers transfer the load from the mobiles to remote resources, usually in a Cloud infrastructure. The focus of our work is to find policies for balancing the load between devices and remote resources using offloading techniques. We apply these techniques on a popular artificial intelligence intensive mobile game, OpenTTD and seek to reduce the server-side load by leveraging the devices computing capabilities.

Keywords: mobile offloading, provisioning, Android, OpenTTD

1. Introduction

Mobile devices, such as smartphones and tablets in particular have become ubiquitous and are transforming our daily activities and even our education process due to their vast application domain. Most applications rely on connectivity and remote storage, and those from the gaming category make a lot of use also of the computational power of mobile devices or remote servers. While the number of users increases, the games must maintain the quality of experience and provision more remote resources. Completely moving the load to a remote platform leads to greater costs, so we investigate a way of running parts of the application on devices while preserving the user experience.

Our contributions include several offloading policies and an evaluation testbed we designed and implemented for conducting game profiling and offloading scenarios. We have chosen to apply and test our offloading research on OpenTTD's mobile version because it is an open-source real-time strategy (RTS) game which follows a clear game loop with several stages and loosely coupled components which may be executed remotely. The benefits we look for in our

[1] Faculty of Automatic Control and Computer Science, University POLITEHNICA of Bucharest, Romania, e-mail: adriana.draghici@cs.pub.ro
[2] Faculty of Automatic Control and Computer Science, University POLITEHNICA of Bucharest, Romania, e-mail: andrei.voinescu@cs.pub.ro
[3] Faculty of Automatic Control and Computer Science, University POLITEHNICA of Bucharest, Romania, e-mail: nicolae.tapus@cs.pub.ro

policies and mechanisms fall into two criteria: improvement of in-game experience and reduced resource consumption which can lead to lower energy costs.

We address the issue of computational mobile offloading [1] [2], a technique used for sending heavy computations to servers and then waiting for their results and applying them in the applications. Researchers have transferred computations at different levels, from methods [3] and chuncks of code to threads, processes [4] and virtual containers [5] [6] [7]. Our technique proposes the offloading of certain game components, in particular OpenTTD's artificial intelligence components.

OpenTTD [8] is an open source simulation game based on Transport Tycoon Deluxe, in which players develop their own transport company. The goal of the game is to build a successful company, and in order to do that the players need to use modes of transportation for goods and passengers in an efficient way, which generates high revenue. The players can choose between several types of transport and a lot of possible scenarios, each with a specific map size, landscape, industries and settlements. The game supports several operating systems, including Android. Users can compete with each other or against artificial intelligence (AI) components, in multiplayer sessions supporting up to 255 players. In multiplayer mode, there are a maximum of 15 companies, so the players can cooperate in the development of a company and compete against the companies controlled by other players (including AI players). There is also an interest from the community in how the AIs compete with each other, and several AI tournaments have been organized in the last few years.

1.1. OpenTTD Offloading

OpenTTD offers a dynamic game environment, in which the objects on the map continuously evolve - towns grow, vehicles age and may need repair or replacement etc. Moreover, when several companies share the same map, the environment becomes a competitive one, so the planning must be made considering the other companies' actions (a rival company may finish a road that we intended to build etc). This means that AIs must take planning decisions fast in order to remain competitive. While the users play, the game has built-in mechanisms that control the evolution of the environment that consists in towns, industries and vehicles, but they are not very complex and some work poorly without more guidance from the user, the vehicle movement for instance. The more interesting part of this game are the artificial intelligence agents that can compete with the real players, or work along them.

These agents, especially the competitive ones, must employ strategies and algorithms for efficiently building their company and obtaining more income than

the other players, which make them a more appropriate choice for computation offloading. Moreover, in multiplayer mode, players compete on the same map, so they influence one another (one player ends up connecting a specific industry faster than the others who wanted to do the same), which also means that the planning must be done fast. This planning speed requirement is another reason in favor of offloading, permitting the AI to run its algorithms faster. Moreover, profiling has shown [9] that their route planning algorithms need significant amounts of processing power, and coupled with the fact that route decisions must be taken as fast as possible, it seems that they would benefit from offloading.

After identifying our target elements, we need to establish the mechanism(s) we employ for offloading them. Research on mobile offloading mostly focuses on sending application threads to servers in the cloud. There are also fine- grained methods [10] which decide through static and dynamic profiling the methods that will be run remotely, on cloned devices. In the next section we will describe our current method, which is based on OpenTTD's features.

1.2. OpenTTD's AIs

The purpose of artificial intelligence (AI) components in OpenTTD, but also in other games, is to provide a challenge for users in single-player mode. The players compete with one or several AIs on the same map in order to have the most profitable company. It is a way for the players to improve their game strategies, but also to learn from the AIs approaches. Therefore these components are an important part of OpenTTD and their underlying algorithms also make them an appropriate target for offloading.

There are currently approximately 40 AIs, most of them being competitive. The non-competitive ones practically build or control elements on the player's map, such as WmDOT, which just builds a highway network. While some AIs are complex and control many features in an efficient way, some are specialized, being created for a specific purpose, such as just controlling trains, or building complex networks of roads and efficiently using road vehicles. The challenge for a competitive AI is to use intelligent strategies and algorithms for creating and controlling vehicles in such way that it generates a lot of income and obtains greater profit than the competitors.

In our AI analysis we focused on what vehicles it can control, what it can transport, what strategies it uses in order to make profit, how it adapts to the dynamic environment and what path algorithms it uses. We also considered what the users seem to want from an AI (based on the discussions on the official forums), which are vehicle diversity, how stable it is and its competitive results.

A game feature that we must consider in our policies and experiments is the map we use and its characteristics. The community offers many predefined

scenarios, most of them based on real world geographical regions, varying from continents and countries to small regions and islands. We are interested in the size of these scenarios, in the number and density of resources and towns (sparse/dense) and in the terrain type (very flat, flat, hilly, mountainous). The game performance and also the AIs' performance are closely related to the map's characteristics. Some AIs can't work well on large maps, nor when a map is dense, it has a lot of possible resources that the game must render and control and the AI must consider in their planning algorithms. The terrain type influences the costs of roads, leveling areas being expensive, some structures such as airports must be built on flat ground, so the AI must include all these factors in its decision mechanism in order to remain profitable.

2. Offloading Mechanism

Olteanu A. [9] [11] proposes four types of offloading mechanisms for loop-based applications such as OpenTTD: offloading various components, intermittent offloading, offloading with partial data and parallel offloading. Our implementation currently supports serial and parallel offloading of components at a coarse-grained level (we do not offload methods and threads, but entire game elements). For OpenTTD we have identified the AI agents as candidates for offloading. AIs are computationally intensive elements, and require more complex algorithms than the usual game mechanics.

An AI's structure follows a game loop in which it plans the actions to perform on existing vehicles, what new vehicles to buy, and what roads to build in order to connect industries and towns and generate a greater income. Usually, for each of these tasks we have separate management components (different "Manager classes"). For example, AdmiralAI, in each game-loop iteration, calls the managers for aircrafts, trucks, trains and buses, which perform the actual computations and decisions of which route to build, which vehicles to buy etc. In such cases we could consider offloading the 'manager' classes functions, but this is an approach that depends on the structure of the AI's code, and it requires static analysis and observation of each AI's design.

We currently have a simple approach for our offloading mechanisms, which takes advantage of the OpenTTD's multiplayer capability. Therefore, instead of having specially designed containers that run chunks of code remotely, we could consider running the entire AI on non-mobile OpenTTD instances on cloudlet machines or in the cloud. Since in the multiplayer mode, the transport companies share the same map, having an AI that runs on a cloudlet and connects to the same server as the device client, will look the same for users as an AI running locally. What will differ will be the in-game performance, the

device's resource consumption and the requirement to have an active Internet connection.
Our aim is to make the offloading process transparent to the user, such that when a command to start an AI is given, the application connects to a re- mote resource which will start an OpenTTD instance for the given AI. On the cloudlet/cloud side we need to start an OpenTTD server and several clients, one for each AI we want to offload. This strategy is not very cost effective and does not scale well.
Why partial AI offloading? Previous research [9] has focused only on total offloading of AI components, but empirical analysis of AIs performance have shown that some AIs are not computationally intensive, and there is no visible in-game time advantage between their execution remotely and that on the de- vice. Some of these AIs don't support many features or use simpler algorithms. In these cases, there is no need to increase the load on the remote resources by offloading these AIs.

3. Offloading Policies

We intend to evaluate how offloading affects OpenTTD's performance and resource consumption on Android. We have first identified which game com- ponents would bring the most benefits by not being executed on the device, in this case the AI agents, and then we chose a method for offloading them.
In previous work [9] the policy is to offload all AIs. This process requires many server-side resources, since for each AI we should start a separate OpenTTD instance. In order to optimize both our device-side application performance and resource consumption, but also the number of remote resources we use, we intend to partially offload the AIs.
We propose the following offloading policies, classified based on an in-depth analysis of all available AIs both from a technical point of view but also from a popularity and usage standpoint.
To proceed in designing AI-based partial offloading strategies, we performed an in-depth analysis of all available OpenTTD's AIs[4], considering not only their technical aspects but also their usage and popularity. Based on this evaluation, we propose the following offloading policies: *User-centric offloading*, *Feature-centric offloading*, *Random offloading* and *No Offloading*.
For User-Centric offloading the system chooses the AIs that are the most important to the user. In Feature-centric offloading the system chooses the AIs based on the features they support and we have implemented two policies, one

[4] the available AIs in 2013-2014

based on the types of features and one on the number of features. For Random offloading we choose randomly one to keep on the device and offload the rest. In the No offloading policy we do not offload the AI if it is the only one started on the device.

3.1. User-centric offloading

This policy is focused on increasing the user experience, and works under the assumption that the most popular and interesting AIs should have priority for offloading than the simpler, less popular ones.
What does interesting mean? Currently it is difficult to estimate which AIs are the most popular or the most appreciated by the community. We have chosen a few metrics and applied them on 37 AIs in order to obtain a list of 10 main ones. We consider that this policy would benefit if we had an in-game system to receive feedback from users by rating a few game aspects.
The metrics we considered are: downloads ranking, the number of features, the results during AI competitions, the community interest and their implementation complexity.

The list of AIs and their current number of downloads is available on OpenTTD's BaNaNaS[5] content service. We have observed that this ranking is not very relevant for their popularity, because 5 AIs that were old, abandoned and inactive were in top 15. Moreover, the AIs with the most downloads are also the oldest around. There are a few good AIs, such as FastPTPAI, DictatorAI, appre- ciated by the community, which are quite new (less than 3 years), but their number of downloads is, of course, small in comparison with the more than 3 years old AIs. To make this a more relevant popularity criteria it would have been useful to have annual or monthly download rankings.

Users are also interested in the diversity offered by an AI, referring to the dif- ferent vehicles they can control. There are a few competitive AIs (AdmiralAI, AIAI, Dictator AI, NoCAB, SimpleAI, TerronAI, TransAI) that can control most or all types of vehicles available, but usually the AIs are specialized in using a certain type of transport and offer support for another 1-2. The AIs which are highly competitive, usually qualify or win the finals of AI tournaments, and employ more complex strategies and algorithms.
The AIs we have chosen for now are the following: AdmiralAI, DictatorAI, NoCAB, PathZilla, Roadrunner, Rondje, SimpleAI, TerronAI and trAIns.
The policy we propose is the following: for each AI, start it remotely if it is in the list of user-centric AIs. If the number of remaining AIs is still higher than

[5] https://bananas.openttd.org/en/ai

the maximum number of AIs permitted on device, choose randomly the ones to offload.

3.2. Feature-centric offloading

The main purpose of OpenTTD's AIs is to permit the users to challenge themselves by competing against them in single-player mode. Therefore, the community has an active interest in implementing competitive AIs, some specialized in a certain type of vehicles, some in fast route planning, some in maintaining large fleets of various vehicles and so on.

When performing partial AI offloading, which means that we run remotely just some of the AIs the user plays against, we could also consider to prioritize those with more costly features. Our analysis of the 37 AIs identifies several features that we consider important for an AI. By feature we mainly refer to the type of vehicles it can control. There are advantages and disadvantages in favor of each type of vehicle, which play an important role in the complexity of an AI. The downside of our approach in classifying the AIs based on their features is that we don't consider the exact behavior of an AI. As an example, OtviAI is the "perfect copy-cat" of other AIs behavior, so, its performance depends on the other AIs, not on its own strategies.

We consider two directions for offloading AIs based on their characteristics: use the AI's main feature as offloading criteria or make offloading decisions based on the number of features it supports.

The feature-type policy is the following: when given a set of AIs, we offload them considering the priority of each type of feature. First we offload the multi-feature AIs, then the train AIs, road AIs, air, ships and non-competitive AIs. This policy has the disadvantage that some AIs can be specialized in one vehicle type, but also support other types, and are more complex than AIs specialized in that particular type, but not supporting other vehicles. Therefore, we can employ a policy based on the AI's number of features to distinguish between AIs that fall in the same category, or use it stand-alone, instead of other AI offloading policies.

The policy based on the number of features an AI supports, is the following: when given a set of AIs, we offload them considering the number of features they offer, from the higher to the lower number. If two or more AIs have the same number and we need to choose just some of them to offload, we choose them randomly.

4. Evaluation

4.1. Experimental Setup

In sections 2 and 3 we describe our policies and mechanisms for offloading components of OpenTTD. We are currently offloading just the artificial intelligence agents, which require more computation than the game's environment control. In order to perform a wide range of experiments and to test all the AIs, we have developed an automated evaluation testbed. This system allows us to easily run experiments, measure and plot the resources consumption. It is capable of starting instances of OpenTTD on various platforms (on Android devices or on desktops and servers), of controlling the offloading mechanism and of monitoring the running games and their hosts. The researcher just describes the scenario in terms of which AI to load, the map, which devices to use and the running time and the system takes this file, starts and controls the experiment and outputs its performance measurements and plots.

In our experiments we have used a slightly modified OpenTTD's Android port by Pelya[6], which has more than 1 million installs and tens of thousands of active users. On the cloudlet side, we used OpenTTD's 1.3.3 version with some changes that allow us to offload the AIs.

We conducted our tests on an Asus TF101 2-core@1GHz running Android 4.0.3, for the device clients and on a 2-core@2.4Ghz desktop running Linux Mint 15, for the server and the offloaded instances.

4.2. Scenarios and Results

In our OpenTTD AIs analysis we have looked into the user-centric and feature- centric characteristics of 37 AIs. For each of the policies we proposed, we created a list of AIs that have priority for offloading. We have tested several combinations of AIs and applied our policies on them. We defined experiments consisting of complex AIs, specialized AIs and simpler AIs, we varied their number, their combinations and their maps. The duration for each of our tests varies between 7 and 15 minutes, depending on the test scenario.

We have observed that our policies are highly dependent on the subset of AIs we choose to play against, so, after this empirical analysis, we implemented a mechanism that chooses the most suitable offloading policy for the AIs the user chooses to start.

[6] https://play.google.com/store/apps/details?id=org.openttd.sdl

As metrics, we used both device-related ones such as the CPU load and the memory consumption but also a game-specific metric, the in-game time. The in-game time metric is relevant because its evolution depends on how fast the AI performs its operations. This is the type of game where the player builds and observes the evolution of its economy, and the bigger the in-game time the better. For example, between a slow device with no offloading and a device where AIs run remotely the in-game time can differ by days and even weeks in an interval of a few real time minutes, and the users are interested in not wasting time in seeing their companies evolve.

Before starting the policies evaluations, we used our testbed to benchmark each AI in a single-AI on device scenario, without offloading. We have first conducted these tests in order to observe each AI's performance and used these results as a metric in our AI dependent policies. Unfortunately, the outcome was less helpful than expected, most of the AIs having similar resource consumption and identical in-game time evolution, as shown in Fig. 1. We have also identified an interesting aspect, that without an AI the game uses significantly more memory than when having at least one AI (we have run the experiment several times to assure that those values for the No-AI test are not outliers). So far, we have no official explanation for this behavior from the game contributors we contacted.

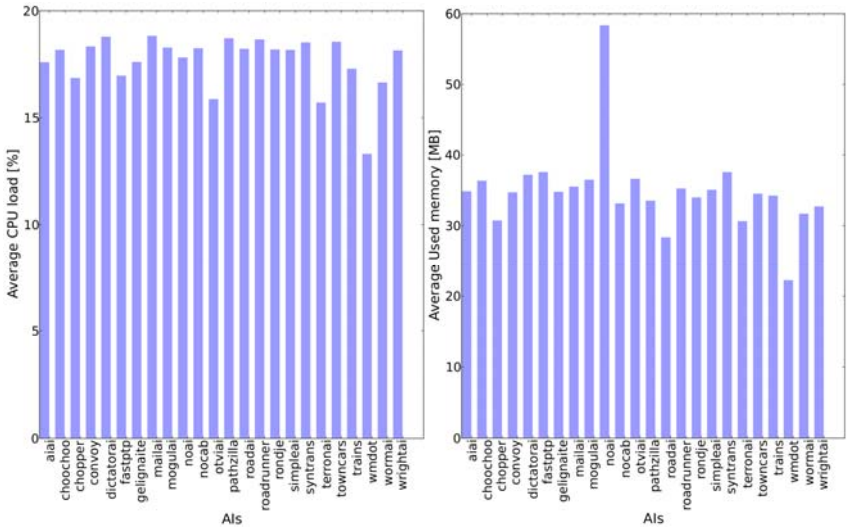

Fig. 1. Comparison of average CPU load and used memory for 24 AIs for a period of 15 minutes.

We approached the evaluation by comparing the partial offloading policies between them and also against a baseline, when all AIs run on device. Due to OpenTTD's restrictions we cannot currently keep more than one AI on the

device in multi-player mode. This affects us not only in our policy enforcement, but also in our performance comparison, since the baseline's AIs don't run in multi-player mode.

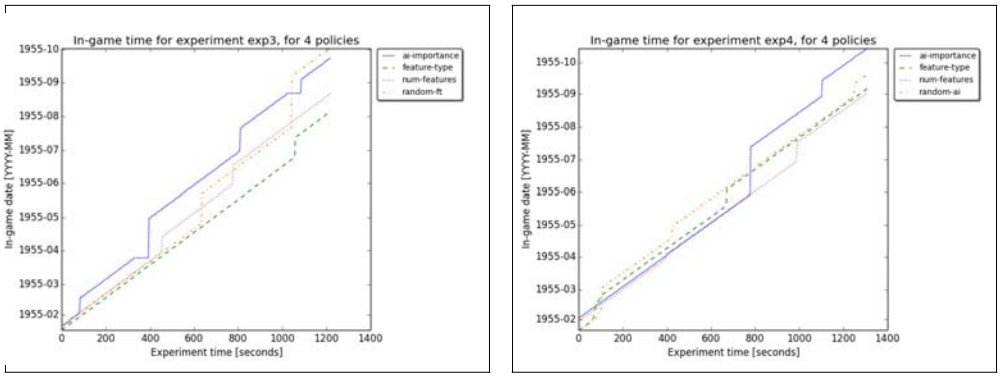

Fig. 2. Comparison of in-game time for two experiments.
"exp3": NoCAB, Terron, trAIns, Convoy, Chopper, WrightAI, MogulAI . "exp4": AIAI, SimpleAI, trAIns, Convoy, PathZilla, WormAI

The tests we conducted for partial offloading varied the number of AIs from 1 to 15 running simultaneously for a period of up to 15 minutes. We have also monitored the AIs behavior during the experiments, they all started correctly and started to grow their companies (building vehicles, stations, roads etc). From the metrics we used, the more relevant for our comparison was the in- game time. Even for tests with fewer AIs (three, four or five) it showed clear and consistent differences between our policies. For example, in Fig. 2 we show two experiments involving seven and six AIs, which were repeated for each of our policies, depicted in the plot as: "random-ai", where we choose a randomly which and how many AIs to offload, "feature-type", where we choose based on their features, "feature-num", based on the number of features supported by each AI and "ai-importance", the user-centric policy. The in- game time for these experiments varied by 2 days, the best policy in this case being the user-centric one. On the other hand, for the CPU load and memory consumption metrics, the performance differences were not so clear, as evidenced in Figs. 3 and 4.

Seeing how little the CPU and memory load differ during the experiments in which we applied our policies on various combinations of AIs, made us look a bit further into the AI-on device benchmarking, and we performed several tests, ranging from one to seven AIs running simultaneously on the device, for various combination of AIs (both simple and complex ones).

We observed no resource performance penalty imposed by increasing their number or varying the type of AI, as shown in an example in Figs. 5 and 6. We believe that this behavior occurs because of the multi-player restrictions imposed by the game's mobile version.

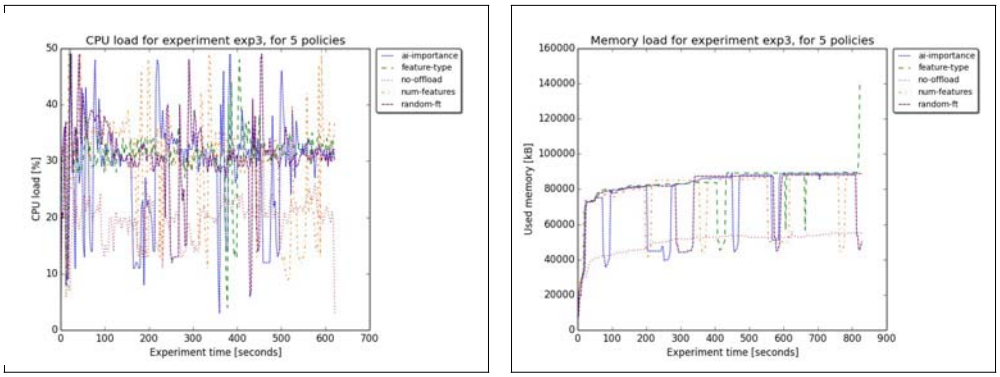

Fig. 3. Comparison of the average CPU load and memory consumption for an offloading scenarios of the following AIs: NoCAB, Terron, trAIns, Convoy, Chopper, WrightAI, MogulAI

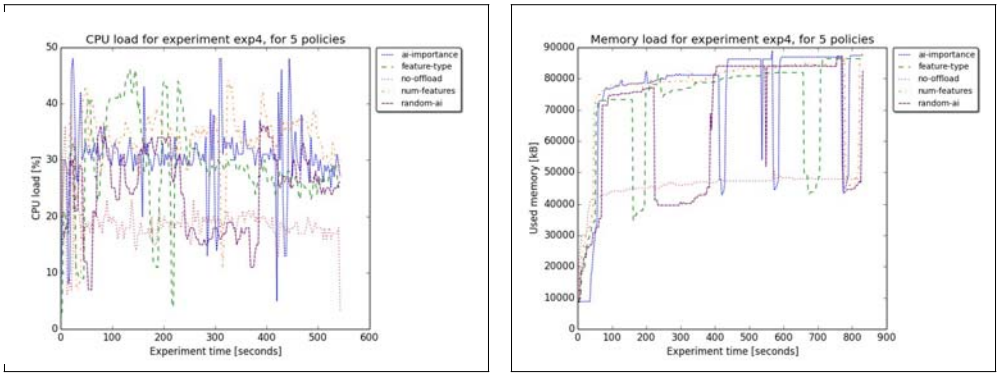

Fig. 4. Comparison of the average CPU load and memory consumption for an offloading scenarios of the following AIs: AIAI, SimpleAI, trAIns, Convoy, PathZilla, WormAI

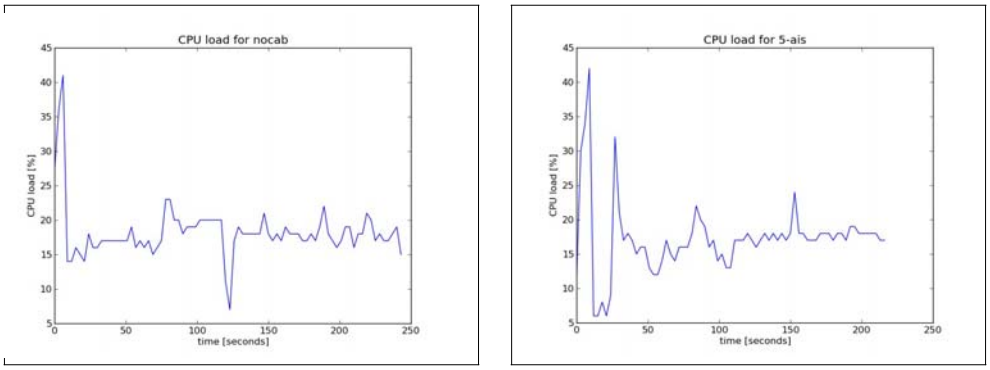

Fig. 5. CPU load for an experiment with one AI running locally and for an experiment with five AIs running locally

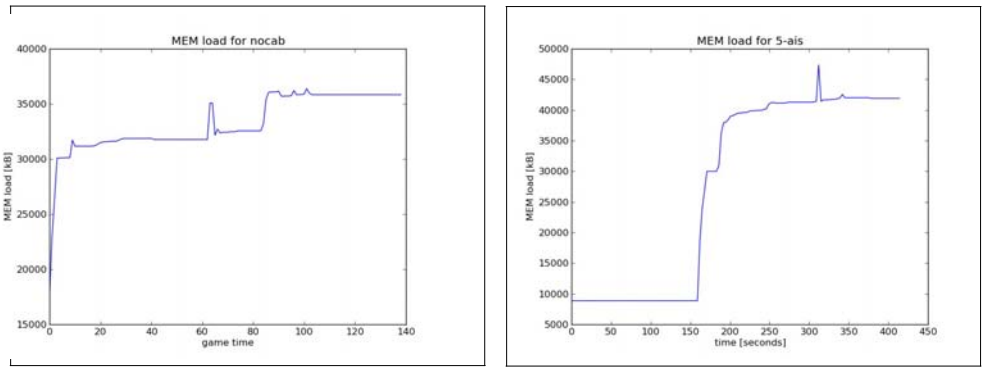

Fig. 6. Memory load for an experiment with one AI running locally and for an experiment with five AIs running locally

5. Conclusions

Computational offloading research seeks to improve both the user experience and infrastructure costs for mobile devices applications. A category of applications that may benefit from such techniques are the real-time strategy games, which have an important user base and market share [12] and make intensive use of computational resources. Its loosely coupled artificial intelligence components and clear game loop make OpenTTD a suitable candidate for our offloading policies and mechanisms.

We have classified and evaluated offloading policies of AI components and proposed policies for balancing the load between the device and the servers while not altering the user's experience. To evaluate these policies and the offloading mechanisms we designed and implemented an evaluation testbed capable of starting and controlling game instances on mobile devices and on

servers, monitoring both device and in-game metrics and providing specific plots. We have identified some improvements, such as in-game time evolution, but we could not empirically identify any significant advantage in terms of resource consumption on the devices. We consider this fact to be caused by the baseline of our experiments: due to current Android OpenTTD implementation restrictions, we could run any number of AIs in parallel on a single device, but in single player, not multiplayer mode.

The current work focuses on AI offloading and all the policies and mechanisms we propose can be applied only to it. A finer-grained direction for our research would be to offload parts of the application's computations, including parts of an AI code. We have identified a loop model for the application execution and for the AIs, in how they update their game components, how they plan and apply strategies. This would lead us to implement the method distribution offloading mechanism, in which computational intensive methods are executed by a different machine and their results and application state is sent over the network. Another improvement of the current research would be to perform the evaluations of AI policies and mechanism on machines in the cloud and study the impact of partial and total AI offloading on their load.

REFERENCES

[1] *K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava*, "A survey of computation offloading for mobile systems," Mobile Networks and Applications, vol. 18, no. 1, pp. 129–140, 2013.

[2] *A.-C. Olteanu and N. Tapus*, "Offloading for mobile devices: A survey," UPB Scientific Bulletin, Series C, vol. 76, Iss. 1, pg. 3-16, 2014.

[3] *R. Kemp, N. Palmer, T. Kielmann, and H. Bal*, "Cuckoo: a computation offloading framework for smartphones," in Mobile Computing, Applications, and Services, pp. 59–79, Springer, 2012.

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

[5] *M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies*, "The case for vm-based cloudlets in mobile computing," Pervasive Computing, IEEE, vol. 8, no. 4, pp. 14–23, 2009.

[6] *M. Satyanarayanan*, "Cloudlets: At the leading edge of cloud-mobile convergence," in Proceedings of the 9th international ACM Sigsoft conference on Quality of software architectures, pp. 1–2, ACM, 2013.

[7] *S. Simanta, K. Ha, G. Lewis, E. Morris, and M. Satyanarayanan*, "A reference architecture for mobile code offload in hostile environments," in Mobile Computing, Appli- cations, and Services, pp. 274–293, Springer, 2013.

[8] "OpenTTD game." https://www.openttd.org/en/. last accessed 17-Aug-2015.

[9] *A.-C. Olteanu*, Extending the Capabilities of Mobile Devices through Cloud Offloading. PhD Thesis, University Politehnica of Bucharest, 2013.

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

[11] *A.-C. Olteanu, N. Tapus, and A. Iosup*, "Extending the capabilities of mobile devices for online social applications through cloud offloading," in Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on, pp. 160–163, IEEE, 2013

[12] "Mobile gaming market share": http://venturebeat.com/2014/04/25/ apple-vs-google-a-world-view-on-the-mobile-gaming-war/view-all/. last accessed on 17- Aug-2015.