The Quest for Non-Functional Property Optimisation in Heterogeneous and Fragmented Ecosystems: a Distributed Approach

Mahmoud A. Bokhari
Optimisation and Logistics, School of Computer Science, The University of Adelaide, Australia
Computer Science Department, Taibah University, Kingdom of Saudi Arabia
mahmoud.bokhari@adelaide.edu.au

Markus Wagner
Optimisation and Logistics, School of Computer Science, The University of Adelaide, Australia
markus.wagner@adelaide.edu.au

Brad Alexander
Optimisation and Logistics, School of Computer Science, The University of Adelaide, Australia
bradley.alexander@adelaide.edu.au

ABSTRACT
The optimisation of non-functional properties of software is of growing importance in all scales of modern computing (from embedded systems to data-centres). In mobile computing, smart devices have complex interactions between their hardware and software components. Small changes in the environment can greatly impact the measurements of non-functional properties of software. In-vivo optimisation of applications on a platform can be used to evolve robust new solutions. However, the portability of such solutions’ performance across different platforms is questionable. In this paper we discuss the issue of optimising the non-functional properties of applications running in the Android ecosystem. We also propose a distributed framework that can mitigate such issues.

CCS CONCEPTS
• Hardware → Batteries; • Software and its engineering → Software maintenance tools.

KEYWORDS
Non-functional properties, energy consumption, mobile applications, Android

ACM Reference Format:

1 INTRODUCTION
In-vivo optimisation provides the ground-truth for the behaviour of an application on a given platform, unlike model-based optimisation which only provides what the model was trained on. However, in-vivo optimisation faces several obstacles. The noisy fitness function resulting from sampling non-functional properties, such as speed, memory or energy use, is a key challenge. In general, precise and robust measurement of non-functional software properties is difficult to obtain. Complete isolation of the effects of software optimisation is simply infeasible, due to random and systematic noise from the platform. Furthermore, noise reduction requires re-sampling the new software variants, which makes the fitness function very expensive. The search space is gigantic and can be non-monotonic, requiring a large number of fitness evaluations in order to discover more interesting regions of the solution space. These issues raise the need to incorporate scalable and high performance techniques to help the search process.

2 THE PROBLEM OF HETEROGENEOUS AND FRAGMENTED ECOSYSTEMS
Android ecosystem suffers from fragmentation and it has been a major challenge for software practitioners. It has been shown in the literature this also affects apps quality attributes as Android deployments can be manufacturer-dependent [5]. In addition, operating system updates, by themselves, contribute to about 19% of energy bugs and hogs [6]. In addition to having expensive and noisy evaluation functions, existing non-functional property optimisers might not be able to cope with such environmental issues.

Taking energy minimisation as an example. Despite the increased interest in software energy consumption improvement in the literature, the current body of work does not consider how the improved software variants perform across different platforms. For example the work in Bokhari et al. [1], Bruce et al. [3] used different hardware, but same operating system versions. The former used several Raspberry pi devices running the same Raspbian OS version whereas, the latter used a Nexus 9 device running Android 6 for evolving solutions and cross-validated the Pareto front on Nexus 6 running the same Android version. This would have been ideal assuming the improved solutions’ behaviour (in terms of non-functional properties) is identical across different operating systems or the underlying implementation details of different operating systems are close to identical. Figure 1 shows the power use of the Rebound library against another variant obtained from Bokhari
Both variants were executed alternately on Nexus 6 running Android 6, from fully charged battery to 20%. A Wilcoxon rank sum test results in a \(p\)-value of less than 0.005 indicating a statistically significant difference between the two solutions. In other words, variant A has less energy use compared to the original version.

In our initial experiments, although we used the same settings found in [2] in each experiment (e.g. activating airplane mode and turning the screen off), the overall system’s CPU utilisation on average was 29% (standard deviation \(\sigma = 2\)) on Android 6, while it was 33% (\(\sigma = 8\)) on Android 7. In addition, the execution time of both variants, memory use represented by the amount of heap used, and the number of background processes during running the two variants increased by more than 100% on Android 7. This shows that other non-functional properties are also affected by the changing the flavour of Android.

Our proposed framework is based on a hierarchical (hybrid) model parallelism Gong et al. [4]. It combines the master-slave and coarsen-grain island models to improve effectiveness and scalability. The master node (PC) controls the smart-phones farm, evolves software variants and distributes them on to the farm of heterogeneous smart-phones for energy fitness evaluation. At every 10% of the battery level, the master allows the migration of the individuals between smart-phones. The current best performing solution on Android 6, from fully charged battery to 20% (369 samples). Figure 1: power used in repeated runs on Nexus 6 running Android 6 from fully charged battery to 20% (369 samples).

On the other hand, running the same variants on the Android 7 gives different results. Figure 2 shows the result of running the two solutions in similar settings to the previous trial. Clearly, on Android 7 variant A is not as energy efficient as the original solution. Additionally, the total amount of energy consumed on Android 7 is doubled compared to Android 6.

In our initial experiments, although we used the same settings found in [2] in each experiment (e.g. activating airplane mode and turning the screen off), the overall system’s CPU utilisation on average was 29% (standard deviation \(\sigma = 2\)) on Android 6, while it was 33% (\(\sigma = 8\)) on Android 7. In addition, the execution time of both variants, memory use represented by the amount of heap used, and the number of background processes during running the two variants increased by more than 100% on Android 7. This shows that other non-functional properties are also affected by the changing the flavour of Android.

3 THE FRAMEWORK

Our proposed framework is based on a hierarchical (hybrid) model parallelism Gong et al. [4]. It combines the master-slave and coarse-grain island models to improve effectiveness and scalability. The master node (PC) controls the smart-phones farm, evolves software variants and distributes them on to the farm of heterogeneous smart-phones for energy fitness evaluation. At every 10% of the battery level, the master allows the migration of the individuals between smart-phones. The current best performing solution on Android 6, from fully charged battery to 20% (369 samples). Figure 1: power used in repeated runs on Nexus 6 running Android 6 from fully charged battery to 20% (369 samples).

On the other hand, running the same variants on the Android 7 gives different results. Figure 2 shows the result of running the two solutions in similar settings to the previous trial. Clearly, on Android 7 variant A is not as energy efficient as the original solution. Additionally, the total amount of energy consumed on Android 7 is doubled compared to Android 6.

In our initial experiments, although we used the same settings found in [2] in each experiment (e.g. activating airplane mode and turning the screen off), the overall system’s CPU utilisation on average was 29% (standard deviation \(\sigma = 2\)) on Android 6, while it was 33% (\(\sigma = 8\)) on Android 7. In addition, the execution time of both variants, memory use represented by the amount of heap used, and the number of background processes during running the two variants increased by more than 100% on Android 7. This shows that other non-functional properties are also affected by the changing the flavour of Android.

4 CONCLUSION AND FUTURE WORK

Generally speaking, in-vivo optimisation is influenced by the physical environment in which the evolution occurs. This, to some extent, requires locality of operations used to generate the new genetic materials. “The beautiful flowers in the municipal garden, it is extremely unlikely that they will be fertilised with pollen from a garden on the opposite side of the world”– A.E. Eiben & J.E Smith.

Besides increasing the number of evaluations, the benefit of using this hybrid model is to improve the global search. By varying the evolution settings on every node, more regions of the search space can be explored. In addition, it is possible to have more than one current best individual evolved in each node. Moreover, tournaments across islands can improve solution robustness. This is because the winner of such tournaments has indeed improved energy efficiency across different platforms.