The Journal of Systems and Software 138 (2018) 1–18

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

An input-centric performance model for computational offloading of mobile applications

Adam Rehn*, Jason Holdsworth, John Hamilton, Singwhat Tee

James Cook University, 14-88 McGregor Road Smithfield, Cairns, Queensland, Australia

ARTICLE INFO

Article history:
Received 23 November 2016
Revised 13 June 2017
Accepted 11 December 2017
Available online 14 December 2017

Keywords:
Timing model
Symbolic execution
Computational offloading

ABSTRACT

Computational offloading frameworks are a widely-researched technology for optimising mobile applications through the use of cloud resources. Existing frameworks fail to fully account for the effect of input data characteristics on application behaviour. Comprehensive timing models exist in the literature, but feature information requirements and performance overheads that preclude use on mobile devices. In this paper, we propose a conceptual model for an input-centric view of application performance. Our proposed model simplifies the existing count-and-weights and pipeline timing models to significantly reduce their information and processing requirements, facilitating use on resource-constrained mobile devices. Our proposed model also utilises symbolic execution techniques to account for the effects of application input data characteristics. Validation with both synthetic and real device datasets demonstrates that our model provides an extremely accurate approximation of the count-and-weights model. Results demonstrate the predictive power of our model for linear execution paths with no loops or recursion. Further work with improved symbolic execution techniques may look to expand application of our proposed model to real-world use cases. The proposed input-centric approach provides a promising foundation for incorporating a deeper level of application-specific knowledge into computational offloading framework cost models, with the potential to contribute to higher-quality offloading decisions.

1. Introduction

Mobile devices and cloud computing are technologies whose prominence continues to grow. The confluence of these technologies represents an area of substantial interest. One common way that mobile devices can utilise the power of cloud computing resources is through the use of computational offloading frameworks (Fernando et al., 2013). Offloading frameworks automate the process of leveraging cloud compute resources to perform the processing of application tasks. Offloading is performed adaptively, using information about the application and device state to optimise performance and resource usage (Kumar et al., 2013).

The behaviour of an application can be influenced by its input data in ways that are of interest when performing computational offloading. As an example, consider a function that parses data that has been read from an input file. The function first parses the header fields of the hypothetical data format, which contains a flag specifying if the data is compressed, and another flag specifying if the data is encrypted. The execution paths through the function that include decompression and decryption will be more computationally expensive than the remaining paths that do not. In the case of input data that is not compressed or encrypted, the benefits of offloading the function to a remote server may be outweighed by the costs of transmitting the input and output data over the network. In the case of data that is both compressed and encrypted, offloading the computation may provide a performance benefit that far outweighs the costs of data transmission. The offloading decision can only discern between these two cases if the computational offloading is aware of the influence that the input data's header fields have on the function's behaviour.

Existing computational offloading frameworks rarely account for the full influence of an application's input data on its behaviour. Most existing frameworks focus solely on the size of input data, or track input influence temporally through the use of history-based profiles (Fernando et al., 2013; Kumar et al., 2013; Lewis and Lago, 2015). Very few offloading frameworks directly account for the influence of arbitrary input characteristics (Wang and Li, 2004; Shi et al., 2014; Gao et al., 2014). These characteristics could include the presence or absence of boolean or bitwise flags, ranges that parameter values fall into, or any arbitrary features of the application's input data that are acted upon by the application's code

* Corresponding author.
E-mail addresses: adam.rehn@jcu.edu.au, adam.rehn@my.jcu.edu.au (A. Rehn), jason.holdsworth@jcu.edu.au (J. Holdsworth), john.hamilton@jcu.edu.au (J. Hamilton), singwhat.tee@jcu.edu.au (S. Tee).

https://doi.org/10.1016/j.jss.2017.12.010
0164-1212/

(Gao et al., 2014; Wang and Li, 2004), including abstract characteristics whose relationship to application behaviour may not be immediately obvious. The relationship between arbitrary input characteristics and execution behaviour represents a deep level of application-specific knowledge. The utilisation of this knowledge presents an opportunity for improving the decisions made by offloading frameworks. In particular, predictions of application performance may become far more accurate when produced by a model that takes the nuances of execution behaviour into account (Wilhelm et al., 2008).

As we demonstrate in Section 2.3, existing models of application performance are often designed for offline use, and are poorly suited to the online prediction context of computational offloading. Additionally, many of these models feature information requirements or performance overheads that make their use on resource-constrained mobile devices impractical. To improve the use of application-specific knowledge by computational offloading frameworks, a performance model is needed that takes execution behaviour into account whilst remaining efficient enough for use on mobile devices.

In this paper, we propose a conceptual model for an input-centric view of application performance. The proposed model simplifies the knowledge represented by existing models in order to significantly reduce information requirements and runtime overheads. The model also utilises symbolic execution techniques to take into account the relationship between arbitrary input characteristics and execution behaviour. Although further advances in symbolic execution are necessary before our proposed model can be applied to real-world applications, validation of our model on controlled test cases in Section 5 suggests that it provides a foundation for making efficient and accurate performance predictions at runtime on resource-constrained mobile devices. Once integrated into a computational offloading framework, these performance predictions could then contribute to high-quality offloading decisions.

Our proposed performance model focuses purely on the metric of execution time. Other resource consumption metrics such as energy usage or network transfer costs sit alongside execution time in a computational offloading framework's cost model. Measurement of these complementary metrics, and integration of the generated predictions into a computational offloading framework's decision-making processes, are outside the scope of this paper and are left as future work to be completed when the prerequisite advances in symbolic execution techniques are achieved.

The contributions of this research are as follows:

- We describe the limitations of existing performance prediction models, and identify the characteristics that make a model suitable for adaptation to a mobile device context. Based on these characteristics, we identify which existing models are best suited to such adaptation.
- We present a new conceptual model for predicting application performance in an online prediction context. The proposed model draws from existing performance models and simplifies them, in order to provide a cost-effective approximation of the represented information.
- We propose a language-agnostic and platform-agnostic tooling pipeline that can be used to implement our input-centric performance model for any programming language and any mobile device platform. Using this tooling pipeline, we describe the design and implementation of a prototype of the model that targets the C++ programming language and the Apple iOS mobile platform.
- We validate the accuracy of our model for code with individual execution paths, with respect to the information that it approximates. Validation results demonstrate that the approximation is extremely accurate, both on synthetic datasets and when used with real mobile devices.
- We then validate the predictive power of the model for code with multiple execution paths, running on consumer mobile devices. In addition, we define a heuristic to measure the suitability of a given piece of code for prediction with our model, and examine the influence of the suitability value on the accuracy of the generated predictions.

The rest of this paper is structured as follows. First, we explore the factors that influence application performance, and examine the state of existing performance prediction models. Then, we present our proposed model and describe the method by which it simplifies the information represented by existing models. We then describe the language- and platform-agnostic tooling pipeline, and discuss the implementation of a software prototype using this pipeline. Next, we validate how well our proposed model approximates the existing model that it simplifies for code with individual execution paths, by comparing the results of the two models on both synthetic datasets and benchmark results from real mobile hardware. We describe the experimental methodology used to validate the predictive power of the model for code with multiple execution paths, and present the results of this validation. Finally, we discuss the implications of this research and directions for future work.

2. Background

2.1. Application profiling

Computational offloading frameworks perform two key functions. The first is to manage the migration of an application's execution between the local mobile device and a remote server (Shiraz et al., 2015). The second function is to perform cost-benefit analysis to make optimal offloading decisions that satisfy criteria that have been specified by either the developer or the user (Fernando et al., 2013; Kovachev et al., 2011). This cost-benefit analysis is commonly formulated as an optimisation problem, which aims to satisfy a set of goals within a given set of constraints. In order to predict the cost of performing offloading, frameworks commonly monitor resources on the local device in addition to the behaviour of the application being offloaded. Fig. 1 depicts the common components of the computational offloading cost model.

The component of the offloading cost model that we focus on is application profiling. Application profiling seeks to characterise the performance and resource usage of an application. Profiled characteristics often include execution time, battery usage, and memory footprint (Fernando et al., 2013). Maximising performance and minimising energy consumption are common offloading goals. An application's memory footprint determines the quantity of data that must be transmitted over the network when performing migration between the device and the remote server (Chun et al., 2011). Minimising data transfer is another common goal when making offloading decisions. It is the performance characteristic of application profiling that we focus on in our research.

Application performance is influenced by numerous factors, including characteristics of the application itself and the operating system and hardware that the application is running on (Wang et al., 2002). Characteristics of the application itself can include instruction count, basic blocks and execution paths, input data, and system calls (Reistad and Gifford, 1994; Wilhelm et al., 2008). Factors related to the operating system include context switches and pre-emptive scheduling (De et al., 2007), OS clock ticks and timers, system daemons (Tsafrir et al., 2005), Translation Lookaside Buffer (TLB) mosses, page faults and memory swap-