Leveraging Structural Information to Ease Approximation Management

Liu Liu
Rutgers University
jack.liuliu@cs.rutgers.edu

Sibren Issacman
Loyola University
isaacman@cs.loyola.edu

Ulrich Kremer
Rutgers Universitay
uli@cs.rutgers.edu

Abstract

Current approximation management approaches establish the correlation between approximate application configurations and overall quality outcomes and resource requirements through off-line training. At runtime, an approximation manger uses the resulting profiles to select an appropriate configuration. The time needed for the training phase of these largely black box approaches can range from hours to even days depending on the size of the configuration search spaces. Training would have to be repeated when porting an application to a new platform, or when executing in unknown environments, making the approach impractical in many cases. Another drawback of the black box approach is the lack of support for application users to express quality preferences for different aspects of an application, which together determine the subjective overall user experience.

This paper introduces RAPID, a framework for writing a class of approximate applications where the application structure is exposed and utilized. Structural information is used (a) by the developer to significantly speed-up the configuration training phase, and (b) by the application user to express relative preferences of different aspects of an application. RAPID formulates configuration selection as an integer programming problem that optimizes quality while respecting resource budgets. Experimental results on six different applications show the benefits of leveraging structural information in approximation management.

1 Introduction

Approximations and redundancies allow mobile and distributed applications to produce answers or outcomes of lesser quality at lower costs. This paper introduces RAPID[1], a new programming framework and methodology for developing approximate applications. Existing strategies use applications mainly as black boxes, then execute and measure applications under different approximation selections[3–5, 7, 9]. The training phase is done off-line, i.e., before application deployment. Configurations are represented by a set of variables whose values can be changed during execution. There are three main concerns about existing approaches.

Scalability: The total number of configurations can grow exponentially in the size of the application as the number of configurable components increases. As a results, the training phase can be extremely expensive since all data points in the configuration space have to be covered (e.g., [8] reports training times of over 64 hours).

Dependencies: Many applications have dependencies between configurable components. Certain combinations of these component values may not be valid either due to data dependencies or for quality concerns. A straightforward alternative to filter out invalid configurations is through profiled tradeoffs, however this requires these undesirable configurations to be included in the training set.

User Involvement: Current work assumes each application comes with a fixed or commonly accepted QoS metric. However, the notion of quality can be highly subjective in practice. Users need a way to express their particular application quality preferences without needing to know complicated details within the "Black-Box".

In this paper, we explore how structural knowledge of the application can mitigate or even eliminate the weaknesses and deficiencies listed above. Structural information reveals correlations between the components (knobs), significantly reducing the feasible configuration space. Training this compressed configuration space can be further optimized by only executing *representative configurations* which allow the reconstruction of the full search space. In addition, the structural composition of applications can be the foundation of a user specified quality metric that expresses preferences among application aspects.

2 The RAPID Framework

RSDG. RAPID uses the **R**edundant **S**ervices **D**ependence **G**raph as its main representation of an application's structure. The RSDG is a directed graph with node and edge weights [6]. Developers describes the structure information of the application by providing the configurable components (nodes) and their dependencies (edges). Weights represent the cost and QoS contribution of a particular node. Cost and initial quality weights are automatically generated through training, QoS weights can be customized later by the user. As a result, RSDG determines whether a configuration is valid and the estimated cost and QoS for that configuration.

[1]For Redundancy, Approximation, Preferences, Implementation, Dependencies; the cornerstones of our approach.

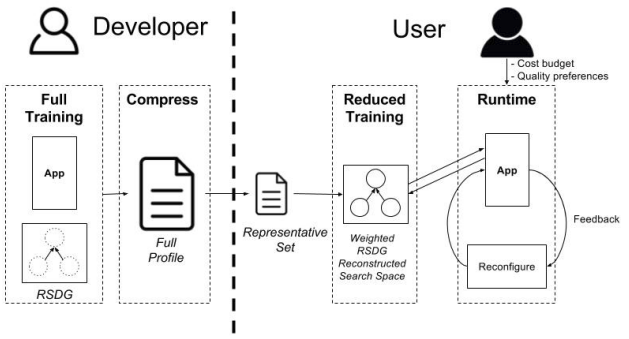
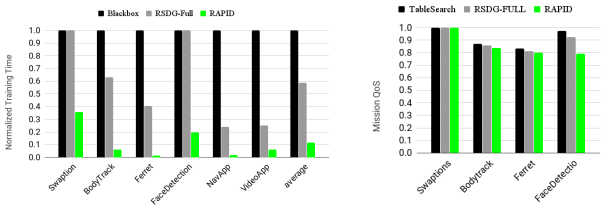


Figure 1. Overview of the RAPID Framework.



(a) Training Time Reduction **(b)** QoS Loss

Figure 2. Evaluation of RAPID

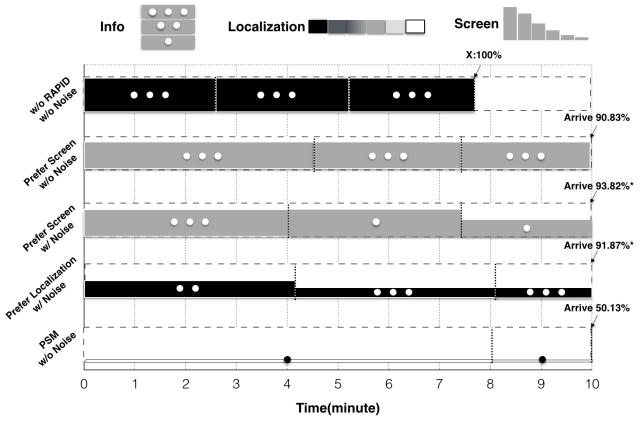


Figure 3. RAPID Runtime and Custom QoS in NavApp. Text above the end of each bar: "X" - failure (exceeds budget). "Arrives(n%)" - completes the mission with n% of budget.

RAPID. Figure 1 shows the basic structure of the RAPID framework. RAPID takes the input program along with the RSDG specified by developers to run a full training on all configurations. It then calculates a small training set, the *Representative Set, RS*, representing the minimum training set when re-training is required. *RS* starts with only the configurations with the highest and lowest cost, and iteratively adds a new configuration until a pre-defined cost model accuracy is reached. The next configuration with the highest potential to improve the accuracy is selected. This set allows a reduced training phase that is able to reconstruct the entire configuration space and its profiles at significantly reduced runtime overheads. Before application execution, users customize QoS by providing relative preferences of application services and their approximations through updating the RSDG quality weights. RAPID optimizes the custom QoS under resource budget. It periodically monitors the runtime behavior and reconfigures if necessary.

3 Experimental Evaluation

We evaluate the benefits of leveraging the structural information in approximation through six applications: three commonly used Parsec Linux benchmarks [1] (Swaptions, Ferret, and BodyTrack), two mobile Android applications developed with RAPID (NavApp: car navigation, VideoApp: video streaming), and one RAPID embedded application (FaceDetection: image processing running on Nvidia TX-1 Board [2]). In Fig 2a, the black bar represents the total required training time using a black-box approach where all combinations of component settings are considered valid. The gray bar shows the time after compressing the search space through dependencies, and the final green bar shows the time for only training the *RS*. On average, training times were reduced by up to 92%. Fig 2b shows the final QoS loss of four benchmarks using the cost model constructed by the three strategies, with 4.9% on average. In summary, RAPID suffers from minor QoS loss compared to other strategies but has significantly lower training overheads.

4 Runtime and Custom QoS

To evaluate the runtime control and the ability of supporting custom QoS, we also performed a series of experiments for each application. Due to space limitations, we only show in Figure 3 experimental results for NavApp, the mobile navigation application. The overhead of RAPID is below 1.5% on all applications in terms of execution time or energy.

5 Conclusion and Discussion

RAPID is the first system that explicitly represents approximations and their dependencies, and optimizes the application behavior in response to user preferences and energy budgets. Having the structure of applications, the required training for constructing the cost model becomes easy and efficient, which enables the application to be quickly profiled when being ported to unknown environments, suffering only a small QoS loss compared to a fully re-trained strategy. RAPID dynamically reconfigure to handle the runtime uncertainty including input dependencies or unexpected runtime behavior. We are currently extending the RSDG representation to support more complex cost models, including highly non-linear models. Furthermore, we are developing tools to ease a developer's effort to write RAPID applications.

References

[1] Christian Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

[2] NVIDIA Corporation. Nvidia jetson tx1 developer kit, 2017.

[3] Anne Farrell and Henry Hoffmann. MEANTIME: Achieving both minimal energy and timeliness with approximate computing. In *2016 USENIX Annual Technical Conference*, Denver, CO, June 2016.

[4] H. Hoffmann. JouleGuard: Energy guarantees for appoximate applications. In *SOSP '15)*, October 2015.

[5] Henry Hoffmann, Stelios Sidiroglou, Michael Carbin, Sasa Misailovic, Anant Agarwal, and Martin Rinard. Dynamic knobs for responsive power-aware computing. In *ASPLOS'11*, March 2011.

[6] U. Kremer and L. Liu. A framework for exploiting redundancies in service-based applications. Technical Report DCS-TR720, Department of Computer Science, Rutgers University, October 2015.

[7] Jongse Park, Xin Zhang, Kangqi Ni, Hadi Esmaeilzadeh, and Mayur Naik. Expax: A framework for automating approximate programming. Technical report, Georgia Institute of Technology, 2014.

[8] Stelios Sidiroglou, Sasa Misailovic, Henry Hoffmann, and Martin Rinard. Managing performance vs. accuracy trade-offs with loop perforation. In *ESEC/FSE'11*, Szeged, Hungary, September 2011.

[9] Xin Sui, Andrew Lenharth, Donald S. Fussell, and Keshav Pingali. Proactive control of approximate programs. In *ASPLOS '16*, 2016.