POSTER: Exploiting Approximations for Energy/Quality Tradeoffs in Service-Based Applications

Liu Liu[†], Sibren Isaacman[*], Abhishek Bhattacharjee[†], Ulrich Kremer[†]
[†] Rutgers University, NJ [*] Loyola University Maryland, MD

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 service-based applications with approximations and redundancies. While traditional redundancy optimizations select among service alternatives without changing user observable application outcomes, approximations change application semantics since they produce outcomes of reduced quality. As a result, users have to be involved in the service selection process. Previous work [1, 3, 5, 7] addresses this "quality specification" challenge by requiring the user or application programmer to provide quality functions for different aspects of the input/output behavior of an application. The quality reduction of an application is defined as the sum over all aspects' weighted differences of the ideal output (no approximations) quality and the approximate output quality. Though RAPID uses a similar approach to quality specification, it focuses on service qualities and their interactions. This approach provides a more intuitive way for the user to assess the benefits of approximations relative to resource or cost savings.

To support service-based quality specification and search space explorations, RAPID introduces a graph representation that encodes an application's services, their approximation levels, and dependencies between them. Finding the best service configuration under a given resource budget becomes a constrained, dual-weight graph optimization problem. Existing strategies use applications mainly as black boxes, and execute and measure applications under different approximation selections[1, 2, 3, 5, 7]. This training phase is done off-line, i.e., before application deployment. Without knowledge of the application structure and its dependence patterns, search spaces are potentially large. As a result, off-line training will be costly (e.g., in [6], the training time over 64 hours). RAPID represents applications through a weighted graph and derives a set of representative configurations that satisfies the mean or max error of an energy cost model. When porting application to another platform, only a short on-line training phase is needed which runs a computed set of representative configurations to determine node and edge cost (e.g.: energy) weights on the new target platform. The explicit representation also enables easy exploration of an application's service selection space under different resource constraints and user priorities.

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

2. THE RAPID FRAMEWORK

Observation 1: Optimization spaces are large. The best possible application behavior may depend on input characteristics, available redundancy and approximation levels, user preferences (see below), energy and resource budgets, failure scenarios, and adverse environmental conditions. An exhaustive off-line training period could take weeks for certain applications. However, RAPID uses an explicit graph representation allowing programmers to express dependencies without much effort. Experiments have shown that pruning can reduce the search space size by up to 89%.

Observation 2: Quality is subjective. Quality is a concept that is often difficult to formalize, and does not generalize across different value/outcome domains. RAPID represents quality indirectly through relative preferences among "user critical" services, rather than through their actual quality metrics, if such formal metrics exist at all. Relative preferences also encode individual user expectations, which may vary significantly.

Framework Overview. Figure 1 shows the basic structure of the RAPID framework. The application designer, or *system expert*, determines the services used by the application and the dependencies between them. The resulting system graph is executed by an automatic tool to determine the energy weights for all services and the energy needed for possible service interactions. The *user* expresses relative priorities of all the services that she considers important.

The application can be run within RAPID's dynamic runtime environment or can be evaluated using static analysis tools.

RAPID's key feature is its ability to balance and adjust the quality and selection of approximate and redundant services across the entire service web. Service reconfiguration is needed as a response to changing quality requirements, service availabilities (failures), or energy budgets.

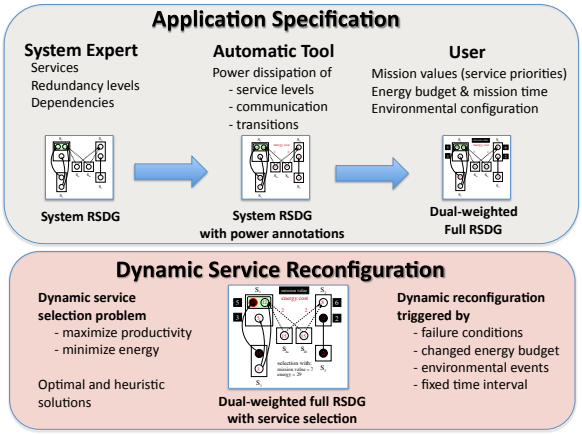

Figure 1: Key Features of the RAPID Framework.

Redundancy Graph. The **R**edundant **S**ervices **D**ependence **G**raph (RSDG) is RAPID's main representation for service-based applications. The RSDG is a directed graph with node and edge weights [4]. Nodes represent implementations of a service or groups of service implementations. Edges represent data or resource dependencies between service implementations. Weights represent the energy cost and mission value of a particular node or edge. RAPID determines energy weights by performing an automatic value propagation strategy during an initial off-line training period followed by online training on a small set of representative configurations.

Experimental Evaluation. To assess the practical, end-to-end effectiveness of our energy-aware approximation management strategy, a prototype system was implemented and evaluated. The evaluation uses three very different sample applications: a mobile street navigation system (NavApp), an autonomous service robot (Robot), and YouTubeApp, the popular video player. None of the applications have a numerical overall quality metric. Instead, each aspect relevant to the user's experience has an intuitive quality notion a user can reason about. This includes the screen brightness, information level, and desired localization precision in NavApp, the appreciation for a particular brand of soda in Robot, or the screen brightness and image quality in YouTubeApp. To demonstrate the expressiveness, correctness, and overhead of RAPID, we performed five experiments for each application: Exp1) Normal execution without RAPID. Exp2) Execution with RAPID. Exp3) RAPID execution with noise. Exp4) RAPID execution with noise and different user preference. Exp5) Execution under "Low-Power-Mode", where each service is set to the lowest level. The experimental results for NavApp are shown in Figure 2.

3. SUMMARY OF CONTRIBUTIONS

Experimental results using our three practical sample applications (NavApp, Robot, and YouTubeApp) show that the

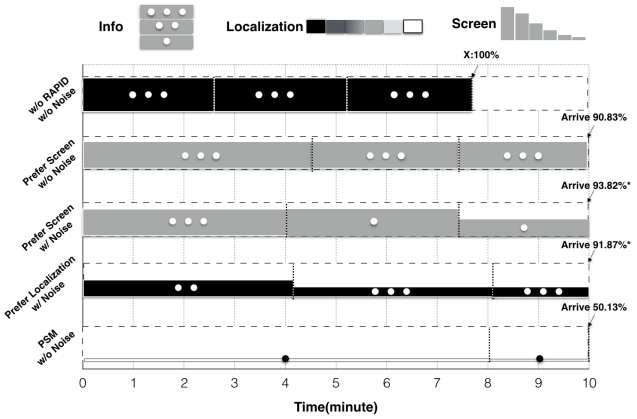

Figure 2: NavApp configurations over time for different priorities. Bar Height: Screen brightness level. Bar Color: Frequency/Accuracy of location data pulling. Dot number: Map type displayed on screen. Text above the end of each bar: "X" - fails the mission (exceeds energy budget). "Arrives(x%)" - completes the mission with x% of energy budget.

RAPID framework is effective and efficient. Optimal service approximation levels can be computed quickly with little runtime overhead (<1%), enabling fast reconfigurations. RAPID is the first system that explicitly represents approximations, redundancies, and their dependencies in a unified framework, and optimizes the application behavior in response to user preferences and energy budgets. Physical metrics of interest in each application (e.g., screen brightness) improve by >24% while using RAPID, compared to intelligent heuristics. Finally, without RAPID, execution for our use cases terminates after 70% of the total mission and underutilizes the budget by 56% under a straightforward energy saving mode. In contrast, RAPID always completes the mission and maximizes utility of the budget while never violating the budget constraints even in a noisy environment.

4. REFERENCES

[1] 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.
[2] H. Hoffmann. JouleGuard: Energy guarantees for appoximate applications. In *SOSP '15)*, October 2015.
[3] 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.
[4] 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.
[5] 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.
[6] 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.
[7] Xin Sui, Andrew Lenharth, Donald S. Fussell, and Keshav Pingali. Proactive control of approximate programs. In *ASPLOS '16*, 2016.