© 2020

Liu Liu

ALL RIGHTS RESERVED.

TRADING QUALITY FOR RESOURCE CONSUMPTION THROUGH APPROXIMATION MANAGEMENT

by

LIU LIU

A dissertation submitted to the

School of Graduate Studies

Rutgers, The State University of New Jersey

In partial fulfillment of the requirements

For the degree of

Doctor of Philosopy

Graduate Program in Computer Science

Written under the direction of

Ulrich Kremer

And approved by

_____

_____

_____

_____

_____

New Brunswick, New Jersey

JANUARY 2020

ABSTRACT OF THE DISSERTATION

Trading Quality for Resource Consumption through Approximation Management

by Liu Liu

Dissertation Director:

Prof.Ulrich Kremer

The goal of traditional optimizations is to map applications onto limited machine resources such that application performance is maximized while application semantics (program correctness), is preserved. Semantics is thought of as a unique mapping from inputs to outcomes. Relaxing application semantics through approximations has the potential of orders of magnitude performance improvements by trading off outcome quality for resource usage. Here, an execution outcome is not only based on its inputs but also resource availability and user quality expectations.

Emerging approximation techniques provides various ways to trade-off output quality for lower resource consumption. However, as a developer, the guidance and support on how to utilize the power of approximation in everyday applications are limited and rarely discussed in recent works. The offline training overhead to support approximation is usually huge, but often be treated as ”free.” Besides, it is surprising that end-users involvement is always overlooked when determining the quality notion, which should be highly subjective. Finally, supporting approximation in a multi-programming environment is crucial to let approximation be widely accepted as a general technique.

In this dissertation, I introduce Rapids, **R**econfiguration, **A**pproximation, **P**references,

Implementation, **D**ependencies, and **S**tructure, a framework for developing and executing applications suitable for dynamic configuration management for approximate computing. The main contribution of Rapids is its design to address the above concerns through exploiting the different expertise/strengths of the three actors (developers, users, applications) involved. I conduct comprehensive experiments and show that Rapids is adaptive and extendable by providing customizable configuration spaces for developers and the support for customizable quality for end-users. It has low overheads and small cross-platform porting costs. I also introduce an extension of Rapids, Rapids-M (**Rapids** for **M**ulti-programming), which is the first system that discusses cross-application approximation management. The target is to understand and overcome the challenges in approximation management fundamentally, then let both developers and end-users benefit from approximation with little extra efforts so that a wider audience can accept the technique.

ACKNOWLEDGEMENTS

Like every new student, the first thing I was told was that pursuing a Ph.D. would not be easy. The 6-year adventure of my Ph.D. life was arduous, unforgettable, but still happy. Thanks to all the people besides me who make this journey an enjoyable experience of my life.

I'm eternally grateful to my advisor, Prof.Ulrich Kremer (Uli), for his constant support and guidance. Uli's talent in programming languages provides invaluable insights into the research topic, which is relatively new. Most new ideas and major signs of progress along the road were made in front of the whiteboard in his office. Besides the academic support, I was also encouraged by Uli when facing difficulties, for example, when the conference deadline was 3 AM in the night, or when our submission got rejected, or I was desperately struggled with a hard problem for a long time and about to give up. Uli was also supportive and generous when it comes to my plan for the future. With all these supports, my Ph.D. life has been more organized and more productive than it supposed to be.

I would like to sincerely thank Prof.Sibren Isaacman, who gave numerous practical suggestions and pinpointed multiple detailed flaws in both the design and implementation of my work. Besides, the help of formalizing and polishing the language for all submissions was huge. His ideas and patience in academic writing made an enormous contribution to our submissions.

Also, I'd like to thank all the members of my committee, for bringing up potential working directions and useful comments on both research and presentation skills. Thank you, Prof.Abhishek Bhattacharjee, Prof.Eric Allender, Prof.Richard Martin, Prof.Desheng Zhang, and Linbin Yu. I much appreciate Desheng for bringing up fresh and different ideas and angles that bring up the possibility of applying approximation on diverse fields of work, like urban computing.

Then, I want to thank my internship managers during two of my summers, Nick Ko-

rostelev from Google Infrastructure, and Linbin Yu from Facebook Adaptive Apps. I enjoyed both summers and gained invaluable industry experiences, including the coding style of a higher standard and a better idea of system design. These cannot be achieved if without Nick and Linbin's help and guidance.

Besides, I want to express my gratitude to all the people standing beside me for their support. Eric Cong, my first friend at Rutgers, supported me whenever my life gets bumpy. Yixin Gu, my oldest friend, starting from middle school, always provided useful advice surviving all the dark moments of my life. Thanks to Zi Yan from RUArch Lab, Yan Zhu, Lin Zhong from CBIM, Fang Wang, Junjie Ouyang, and Zuohui Fu for making my life much more joyful. Also, I would like to thank Yi Dong for staying by my side and taking care of me during this most critical period in my life.

Lastly, I want to thank my parents, Xueli Liu and Yang Liu, for supporting me with unconditionally loving and care. I could never "survive" this without your help.

My research presented in this dissertation is supported by the grant CSR-1617551 from National Science Foundation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

From small digital devices such as mobile phones, tablets, or ARM [1] boards to personal computers and large databases, the trend for higher computation resource demand is inevitable, especially after the recent breakthroughs in artificial intelligence. Figure 1.1 shows a brief history of the iPhone's battery capacity and main memory. Compared to its first generation in 2007, the latest iPhone (11ProMax) released in 2019 is equipped with around 31× more memory and 3× more battery capacity. Admittedly, increasing the computation power or battery capacities does play an essential role in improving the application performance or extending the life length. However, such approach is quite expensive in terms of both monetary cost and resource consumption. The retail price for an iPhone-1 in 2007 was $499, and the most recent iPhone-11ProMax costs $1099 - around $855 in 2007 dollars considering the inflation.

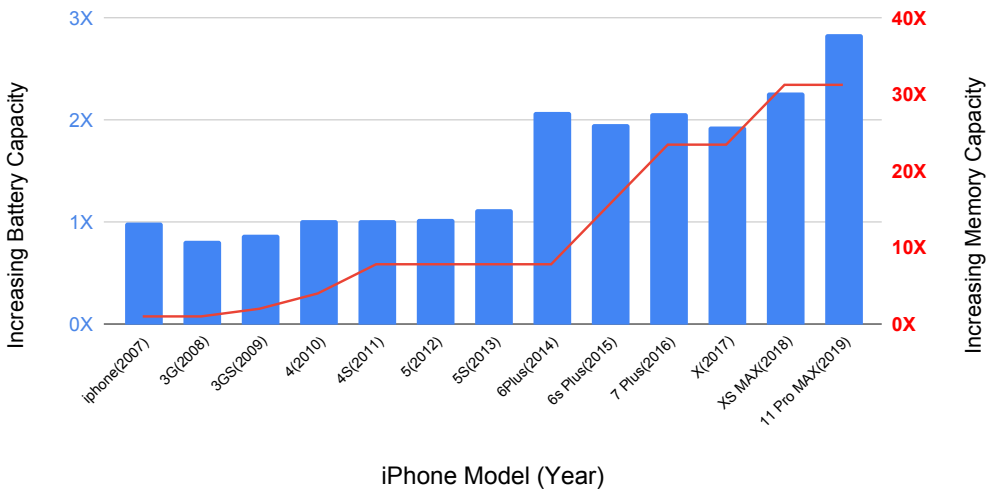


Figure 1.1: iPhone: a Typical Example of Supporting Higher Demands through Increasing Resource Availability

An alternative to increasing resource availability at a high cost is to reduce the resource

demand. For example, “Dark Theme” [2] have been introduced in the most recent development guideline for Android development. The most important purpose, listed as the first benefit on the guideline, is to reduce the power usage. IOS also introduced a similar approach “Dark Mode” [3] in 2019. More aggressively, both Android and IOS systems have “Power Saving Mode” which conserves the battery power through limiting CPU performance, reducing screen brightness, turning off the touch key light, vibration feedback, etc. All these approaches reduce the resource demands through providing users with an “approximate” version of the original service with a reasonably lower quality of service that the users can tolerate.

The idea of “Approximation” is quite fundamental in nature as a principle to survive under limited resources, e.g., animals lowering body functions during winters when food can be hardly found. In the world of computer science, viewing a program as a representation of a mathematical function that maps the input to output values has been a fundamental notion in specifying program semantics and proving program properties. Further, implementations of mathematical functions have always involved sacrificing precision/accuracy for an effective and efficient computable solution. For example, computer architectures support fixed and floating-point operations of specific bitwidth (e.g., single and double precision), coherency models provide a particular view of a shared memory address space (e.g., strong coherency vs. “relaxed” weak coherency [4]), and many applications are based on models that only approximate an actual physical process based on the knowledge of the limited memory and computing resources of the target architecture or computing substrate (e.g., the spatial and temporal resolution in grid-based fluid dynamics [5] codes). In other words, computer architects, OS designers, and application developers have been no strangers to the idea of approximation and its trade-offs.

More recently, the idea of using “Approximation” to optimize program behaviors for resource constraints has gained prominence as a solution for the dilemma of the high demand and the low resource availability. These resource constraints exist on multiple plat-

forms to mobile phones [6, 7], underwater vehicles [8, 9], or large-scale data centers [10]. Such constraints may also exist in multiple forms including execution time, latency, peak power, energy consumption, etc. The potential of approximation creates a new dimension of application performance to explore and inspires multiple works that focus on adapting application behavior to resource constraints. In the world of approximation, users are willing to tolerate an application outcome of lower quality that respects a given resource constraint, rather than having the application fail "halfway through" or produce no outcome at all [11, 7]. In this dissertation, my work focuses on execution time as the main (user level) resources that is managed, with a few examples of energy consumption. However, the techniques introduced in this work can be extended to other resources as well assuming appropriate measurement infrastructures.

One important question, also a common misunderstanding, is whether "Approximation" is just a technique to run applications more efficiently. The answer to this common question is "Yes, and No". First of all, "Approximation" does not affect any application behavior when the resources are not limited. Secondly, the efficiency improvement is indeed the most apparent change one notices after an application is approximated, but it always comes with a price. As an approach intended to reduce the application resource consumption when resources are limited, the main difference between approximation and other techniques that just aim to "speed-up" the execution can be realized through a comparison: To complete a single-threaded application with an execution time requirement, the "Parallelization" technique can potentially shorten the execution time by leveraging the multi-core system after transforming the application to the multi-thread version. Such improvement can be achieved without sacrificing any output quality. However, if the shortened execution time still violates the constraint, the "Approximation" will be the technique that can further extend the speed-up, but with a price. It may run the application under different settings, like skipping iterations, or even dropping code blocks. Such strategies could lower the output quality while cutting down the consumption, as long as such quality

degradation is acceptable.

1.1 Notions and Definitions in Approximation

Approximation is suitable for applications in which end-users are willing to tolerate a reasonable amount of quality loss for better performance under constrained resource availability. Such applications include video/audio players [12, 13], face detection, object recognition, or machine learning. For these applications, there are several necessary conditions for approximate applications that I target in this dissertation:

- There is a correlation between resource consumption and semantically sound application outcomes, i.e., applications have to be tunable, with a set of configurable components and their corresponding settings defining the configuration space of the application;
- Application outcomes under a feasible configuration can be compared and ranked according to a quality metric;
- There are well-defined points during application execution that allow safe and efficient reconfiguration of the application, i.e., transition from the current to the target configuration.

Figure 1.2 shows a piece of code from a benchmark application, Ferret [14]. Ferret is a image similarity query application that accepts image queries and returns the top-K images in a database ranked by content-similarity for each query. The application has two phases. It first locates the top-$2K$ images using a Multi-Probe LSH [15] algorithm. Then, it calls a routine that computes the Earth Mover's Distance (EMD) [16] to rank the images and return the top-K. This example is a typical use-case of applying approximation:

- Different values assigned to the three variables on *Line-2/3/4* can affect the program behavior in terms of the execution time and output results.

```
1   ...
2   int hash = 8; // [2, 8]
3   int probe = 20; // [2,20]
4   int itr = 500; // [2, 25]
5   ...
6   while (ent = readdir(pd)) { // pd points to a dir containing 3500 images
7         if (do_query(ent, hash, probe, itr) != 0) // query for top−K similar images
8            return −1;
9         reconfigure(&hash, &probe, &itr); // re−assign values to knobs if necessary
10     }
11  ...
```

Figure 1.2: Example Approximate Application

- The results (top-K similar images) can be further evaluated through a quality metric, G-score [17].

- The application reads and processes query images sequentially which allows to re-assign values to the three variables (*Line-9*).

I use the example application to explain some of the basic terms used throughout the dissertation:

1.1.1 Basic Terms

Mission and Work Unit: In this dissertation, I call a full execution of an applications a mission. A mission can be partitioned into a collection of work units. For example, if the mission is to process a video, then processing each single frame can be considered a "work unit". Such design gives the runtime control an opportunity to examine the work progress and monitor the resource consumption of the work units that have been accomplished.

The concept similar to work unit was first introduced by Hoffman and etc [18], called "heartbeat". It is defined as the point of completion of each iteration in the outer-most loop. In other words, target applications for approximation need to have the pattern of executing multiple iterations of a main control loop. At the moment of completing each iteration, the application reads the next unit of input, processes this unit, produces the corresponding

output, then executes the next iteration of the loop. Note that the "input unit" here only represents the data preparation for the next iteration, and does not necessarily have to be an I/O operation. As the application processes each input unit, it also reads the configuration provided by the control system and determines the algorithm to be used. A "work unit" in this work is not limited to a "heartbeat". It can be in other forms as well, for example, "one second of execution".

For Ferret*: As shown in Figure 1.2, the* ***Mission*** *is to "query 3500 images in the directory" (completing all iterations of the loop on Line-6), and each single image is a* ***Work Unit*** *(do_query() on Line-7).*

Knobs and Configurations: In approximate applications, the configurable components (program variables, or certain execution paths) can be viewed as tunable "knobs", i.e., entities that, when changed, alter the program execution behavior thus impacting the quality and cost of the application's outcome. A configuration defines an execution plan in which each knob gets assigned a particular value. Different configurations can trigger different execution logics as shown on *Line-7* where these values are passed into *do_query()* as parameters. Each knob in an application may come with a value range, which defines the possible values the knob can have. The purpose of having such constraints could be for security, functionality, or other purposes.

For Ferret*, there are three knobs, namely "hash", "probe", and "itr", specified on Line-2 / 3 / 4. Each of these knob can have a value within a specific range. The first one, "hash", sets the number of hash buckets per table within the range of 2∼8. "probe" sets the number of probing buckets in the multi-probe phase from 2∼20. "itr" sets the maximum number of iterations when computing the EMD (Earth Mover's Distance) within 2∼25.*

Configuration Space and Feasible Configurations: The Cartesian product of the knobs with their value ranges define the full configuration space. If no other constraints exist, a configuration is considered valid (feasible) if all knobs are assigned values that fall within

Table 1.1: Example Profile for Ferret. The row with configuration marked bold has lower cost than the row above it, but produces higher quality

Configurations			Cost	Quality
hash	itr	probe	(second)	(score)
2	2	2	52.38	55.95
...				
2	14	10	55.02	57.40
4	25	12	61.56	80.58
6	2	2	60.53	90.08
...				
8	25	20	78.38	100

their ranges. The universe of feasible configurations is the configuration space. The size of the configuration space is determined by the number of knobs, and the number of values per knob. It is also the size of the optimization problem (searching for optimal configuration).

For Ferret*: The three knobs, each with integer values, form up the configurations space with the size of* $7 \times 19 \times 24 = 3192$.

Budget and Cost: The budget is a user-defined constraint that defines the maximum resource consumption (execution time / energy) she/he is willing to spend on the mission. Different configurations may have different levels of resource demands when processing each work unit. The overall cost of an application is defined as the resource consumption to finish the application, which can be estimated by multiplying the cost-per-unit by the number of work units.

For Ferret*, the column "Cost" in Table 1.1 shows the different cost-per-unit in terms of execution time when running under different configurations (knobs being assigned different values) as shown in the first three columns. The first and last row show the cheapest and the most expensive configuration, respectively.*

Default Configuration: Each application should come with a default configuration where each knob is assigned a default value from the developer. An application running under the default configuration is the non-approximate version of the execution. The result from the

execution under the default configuration has the highest quality, which is also called the "reference result" in Capri [19]. In most cases, the default configuration is also the most expensive configuration.

Quality: The quality metric is a method to map the execution result to a numerical value. In approximation, the output quality can be evaluated by comparing the results from executions under approximate configurations against the result under the default configuration (non-approximate execution).

For Ferret*, the output of a query image is a sorted list containing the ID of the top-K images. I use a common ranking function "G-score" [17] that computes the score of the result (top-k similar images) by comparing the result with the "reference result". The last column in Table 1.1 reports the quality for each configuration in* Ferret*. More details about the metric can be found in Chapter 3. Note that in Table 1.1, the execution of* $\{$ *hash=2, probe=14, itr=10*$\}$ *runs faster than* $\{$ *hash=4, probe=25, itr=12*$\}$ *(lower cost), however produces a lower quality, 57.4, than the slower one, 80.58. Configurations can be ranked by the quality that allows the control system to determine if one configuration is "better" than another.*

Optimal configuration: The optimal configuration under a budget is the configuration producing the highest quality with a cost less than the budget. If the maximum and the minimum cost of all configurations are C_{max} and C_{min}, then the default configuration is usually the optimal configuration when budget$\geq C_{max}$, unless there exist other cheaper configurations that produce the same quality (highest quality). The cheapest configuration is the optimal configuration if the budget $= C_{min}$ or is too low to afford all other configurations. The problem of searching for such optimal configurations under different budgets is not trivial, because 1) the configuration space is huge, and 2) more expensive configurations do not always have higher quality. There are two main types of approaches for this problem, namely model-based and control-based approaches. Both approaches first perform a

profiling phase to construct the cost and quality models. The model-based approaches, like Capri [12], or this work, construct the cost and quality models such that they can be used to predict the estimated cost and quality for different configurations. Then the system can locate the optimal configuration by prediction and searching algorithms. Control-based approaches like PowerDial [20], or Jouleguard [6], leverages control theory methods that use closed mathematical formulations such as differential equations to gradually converge to a configuration which satisfies the budget constraints. Both types of approaches have their own benefits and drawbacks. As pointed out in Caloree [21], Model-based approaches can accurately model the cost and quality of complex, interacting knobs, but does not address system dynamics; Control-based approaches [20, 6] adjust to dynamic changes, but struggle with complex knob interactions.

For Ferret*, as described above, configurations with lower cost do not always produce lower quality.*

Reconfiguration: The optimal configuration under a budget can be found through different approaches as described above. However, the application may encounter fluctuating performance during runtime compared to the measured cost during training. Upon the completion of each work unit, the online controlling system has the chance to re-evaluate the optimality of the current configuration and decide whether to switch to a new configuration. In general, the reconfiguration can be defined as a periodic check that first monitor the resource usage, then take actions if the optimal configuration changes. During each reconfiguration, the system first checks the current resource usage and the work progress (Monitor). The budget per work unit for the remainder of the execution will be re-evaluated. If the current configuration is no longer optimal under the updated budget, a new configuration might be selected (Reconfigure). This feedback loop of monitor/reconfigure aims to make sure the total cost of the execution does not violate the budget constraints in dynamic environments. The over/under estimation of cost could be due to multiple reasons with some most important ones listed below:

- Input Dependencies: The cost to process different inputs can be significantly different. For example, in a video encoding application, processing a frame can be much faster if the current frame is identical or similar to its previous frames.

- System Noise: Since the target application will not be the only application running in the system, computation resources will be shared across all active applications.

- Model Accuracy: For applications with knobs with continuous settings, it is infeasible to train all configurations and observe the costs. Instead, a model can be constructed by observations from a sub-sampled configurations set. The inherit error in the model could lead to the under/over estimation in the runtime. Determining the sampling strategy for better model accuracy is one of the main focuses of this work.

For Ferret*, the reconfiguration can happen on (Line-9 in Figure 1.2) after every single iteration (work unit). Alternatively, it can be performed after the completion of every* n *iterations by the developer.*

1.1.2 More Complications to be addressed in Approximation

Continuous Knobs: As previously discussed, the size of the configuration space is the Cartesian Product of all knob values. However, this only applies to the case where all knobs have a finite number of values (discrete). For many applications, the value of a knob could be continuous (e.g., floating point value) or semi-continuous (integer values within a large range). In this case, the size of the configuration space may be considered infinite.

For Ferret*, the value for knob "itr" ranges from 2 to 25. In some related works including MEANTIME [22] or JoulGuard [6], the knob was treated as a knob with discrete values sampled from the range. In my work, the knob can be treated as a continuous knob. The distinction between discrete and continuous leads to different models I use, which will be discussed in detail in Chapter 4.*

Inter-Knob Dependencies: For many applications, in addition to the per-knob constraints, there can be inter-knob constraints (dependencies), e.g., $if\ a \leq 10\ then\ b \geq 20$. Such constraints are quite common in real world applications for multiple reasons, including security constraints, low quality configuration filtering, etc. The size of the configuration space could potentially be reduced by pruning the undesirable or invalid configurations. However, it increases the complexity of solving the optimization problem since additional constraints are placed on the configuration space.

For Ferret*, Equation 1.1 shows an example of the inter-knob constraints. The constraint indicates that if there are only a few iterations for EMD calculation (*$2 \leq itr \leq 5$*) in the multi-probe stage, then more hash buckets (*$hash = 8$*) should be generated in the first stage to guarantee that the candidates are already accurate enough. Overall, the size of the configuration space gets pruned to 39.4% by having 8 of such constraints. More details will be discussed in Chapter 3.*

$$(2 \leq itr \leq 5) \Rightarrow (hash = 8) \tag{1.1}$$

Custom Quality: For many applications, the output quality is provided as a hard metric, e.g, PSNR [23] in video encoding/decoding. However, there exists a large collection of applications with highly subjective, soft quality notions, which requires involvement from end-users. As different users may have different quality preferences, it is hard to effectively adjust the quality model to encode users' preferences once being constructed. A larger audience can benefit from approximation if such quality customization can be supported.

For Ferret*,* $Coverage$ *(how many real top-K results are included in the execution result) and* $Ranking$ *(how accurate is the top-K images being ranked) are two different sub-metrics to users.* $Coverage$ *is calculated by the first addend in Equation 3.3. Given a fixed* k*, higher* z *results in higher* $Coverage$*, i.e., more "correct" results being returned.* $Ranking$ *is the rest of the equation. Results with lower ranks have less effect on the score*

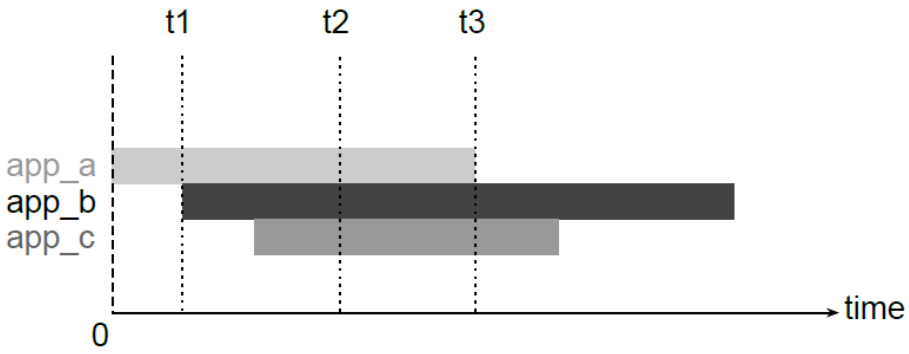


Figure 1.3: Example of cross-application approximation management where executions of three active applications overlapping with each other

of Ranking than results with high ranks (top results).

Cross-Application Approximation: When multiple approximate applications are running together, the performance impact can be more severe than minor environmental noise. In one of my experiments, an application may suffer from up to 15× slowdown in a multi-programming environment. Solely relying on reconfiguration to tackle this issue will not be sufficient since the new selection still uses an incorrect model created when no other application was executing. Also, applications may start and exit in random order as shown in Figure 1.3, which may lead to more frequent but unnecessary reconfigurations. For example, when App-C is about to start, the newly introduced workload from B will create a change in system resource consumption, and then potential performance influence to all active applications. The triggered reconfiguration in each application may further change the resource consumption. Predicting the slowdown from cross-application interference is crucial to make approximation available in the multi-programming environment.

The goal of approximation is to configure the application dynamically with the optimal configuration, such that the resource consumption does not violate the budget, but still produces the highest possible quality. Recent research on approximation have made rapid progress on the three main aspects as follows:

- Semantics: How to express configuration space and quality notions.

- Model: How to construct the cost and quality models.
- Runtime: When and how to perform runtime configuration control to adapt to uncertainties.

The remainder of the introduction will cover these aspects in detail. I will also discuss the limitations of the existing approaches and introduce my proposed solutions.

1.2 Three Main Problems of Approximation

1.2.1 Configuration Space Specification

Designing a proper tool that is expressive and flexible enough for configuration space specification but introduces minimal efforts to the developers is crucial. The size of the configuration space for an application with even a few knobs could still be huge. Once the configuration space is defined, all configurations in the space are assumed to be "approved" by the developers (feasible). Defining the configuration space is crucial, as the important stage for developers to carefully specify all possible constraints, if any, in order to protect the application from failure or unexpected behavior. However, the options provided for developers to define the configuration space and fully utilize the power of approximation are quite limited.

Developers need proper tools to specify the knobs that can be approximated and their attributes. In most cases, a knob is in the form of a variable in the program, e.g, "hash", "itr", "probe" in the previous Ferret example. To this end, language / runtime library based approaches have received increased attention where approximation is an explicit part of a program's semantics. Here I list two representative types of approaches:

- Auto-detect: Some researchers proposed to automatically extract the knobs through either static or dynamic analysis on the program itself, e.g., a LLVM [24] pass embedded to pin-point the variables which can be approximated.

- Manual-label: Another type of research provides language extensions for developers to specify knobs manually, e.g., a notation before a variable to let the system know that the variable can be approximated.

Both approaches aim to help developers to set up the configuration space as the boundary when searching for optimal configurations, or the training set for model construction. However, both approaches may have inherent drawback, i.e., Auto-detect may limit the developers' control over knob attributes specification and Manual-label may put too much burden and responsibility on the developers.

1.2.2 Cost and Quality Model Construction

Designing an efficient training process for the cost and quality model construction is crucial. Once the configuration space is defined, the system needs to know the cost and quality for each configuration in the space which is called the $profile$ of an application. This $profile$ will be used to determine if a configuration is valid when a budget is provided, or if a configuration is better than another when they deliver different levels of quality. An example of such a $profile$ is shown in Table 1.1 where each row shows the expected cost (per work unit) and quality for a specific configuration.

Current approaches [12, 20, 25, 6, 26], including this work, construct such $profile$ through a training process where all or sub-sampled configurations are executed on the target machine. For each trained configuration, the cost and the quality are observed and recorded. For Cost models, the measuring strategy is quite straight-forward if the measurement tools are available. For example, a timer would be enough to measure execution time, and a power gauge (e.g., Trepn [27] on Android) in the system can report the energy/power consumption. For Quality models, the measurement involves a specific evaluation process to examine the execution result and report a numerical value as the quality metric.

The purpose of constructing the cost model $f_c()$ and quality model $f_q()$ is to define the optimization problem for configuration (c_i) selection when a budget is provided. Equa-

tion 1.2 shows the general optimization problem. The $consumption$ represents the resource used by successfully executed work units which could by measured during runtime. $\widetilde{consumption}$ is the estimated consumption calculated by summing up all the cost of the configuration used for processing each executed unit. The difference between the measured and the estimated consumption is considered environmental uncertainty.

$$\begin{aligned} & obj : maximize(f_q(c_i)) \\ & s.t.\ f_c(c_i) * remaining\_units \leq (budget - consumption) \\ & \widetilde{consumption} = \sum_i f_C(c_i), i = 1, 2, ...last\_finished\_unit \\ & consumption - \widetilde{consumption} = uncertainty \end{aligned} \tag{1.2}$$

Constructing the profile can be quite time-consuming. For an application with n knobs, each of which having m settings on average, the size of the configuration space is $O(n^m)$. Training all configurations in the space is usually not possible due to the size of the configuration space, even though some proposed approaches do perform such time-consuming process. In MEANTIME [22], one of the applications takes over weeks to be trained.

1.2.3 Runtime Reconfiguration

Runtime reconfiguration should be enhanced to deal with multiple types of dynamic disturbance. For each instance of a program execution, the application user provides a runtime budget which is assumed to be fixed for the particular execution instance. Ideally, the "consumption" in Equation 1.2 should be close to "$\tilde{consumption}$". However, this may not be the case due to multiple reasons: input dependencies, environmental noises, unpredictable behaviors, cross-application interference, etc. Once the difference between the expected and real consumption ("uncertainty") gets large enough such that the optimal configuration changes, a reconfiguration is required.

The goal of reconfiguration is to ensure that the cost of the application completion is

eventually below the budget. The reconfiguration is performed between work units, and that it can often be implemented by just setting knob values [20] or calling different versions of functions [28]. There are two main types of approaches:

- Consumption Monitor: Most proposed approaches including my work follow the reconfiguration process as described in the previous section that reconfigures based on the feedback of work progress and consumption feedback. It provides strong support for most dynamic environment noises, and limited support for input dependencies by treating it as a type of noise.

- Input Inspection: Some other researchers, including Chameleon[29], proposed approaches to reconfigure before processing each work unit, e.g., select different configurations based on features of input data. It can deal with input dependencies but not environmental noises.

No matter which type of approach is used, it relies on instant online measurement (work progress, consumption, input data feature) as the guidance to update the problem shown in Equation1.2. The most important limitation of "Input Inspection" is that it introduces heavy runtime overhead for inspecting input data, only to deal with input dependencies. That is also why this work follows the most common approach, "Consumption Monitor". However, there is another important source of disturbance in approximation management, the cross-application interference, which cannot be properly handled by all existing approaches. In my experiments, the performance degradation (disturbance) caused by the interference can be up to 15×, which is much more severe than all other types of disturbance sources.

1.3 Limitations of Current Approaches

Though, approximation seems to be an ideal solution for running applications under constrained resources, however, it has not yet seen widespread use in industry. Based on the

real experience of developing approximate applications, I observed multiple roadblocks.

1.3.1 Lack of Expressive Development Model

- Knob Specification

A generic and expressive enough development framework is needed to properly support the developers to integrate approximation strategies into existing applications without introducing too much effort. In Powerdial [20], developers can only specify command line arguments rather than actual knobs. In FlexJava [30], developers can specify the minimal quality requirements. However, it is still left to the compiler [24] to which extent the specified approximation is exploited. In EnerJ [31], the developers are responsible for carefully annotating each variable and determining whether the variable can be stored or operated on using approximate hardware. All these approaches either put too much responsibilities on the developers, or are not expressive enough to let developers participate in defining the approximation strategy.

- Constraints Specification

Existing approaches only focus on the specification on a per-knob level. Developers are unable to encode inter-knob relations (e,g. $if\ k_A <= 10\ then\ k_B >= 5$), which further limit their expressiveness. Such correlations between knobs and their desired values can be treated as a developer's insight about the application.

Though, a training phase with proper quality filters can identify those "bad" configurations that result in "bad" results or even lead to failure. This only applies to the naive case where 1) all configurations are trained, and 2) all knobs are discrete. If any of these two conditions is not satisfied, there will be configurations that fail to be filtered. Additionally, the developers' insights can be extremely valuable, not only to fine-tune the configuration space, but also to encode runtime restrictions. For example, although a configuration may produce an acceptable outcome, a developer may decide to eliminate the configuration for

safety reason or application feature related purpose. Currently, the constraint would only be handled by some form of conditional statements in the source code. Such configuration management is therefore cumbersome and potentially error prone.

1.3.2 Naive Cost / Quality Model Construction

- Cost Model Training Time

The training time for application could be extremely long, however it is often considered to be free in existing works. It is not considered part of the overhead of a particular approach. One possible reason of why this overhead is ignored is that those works do not consider the distinction between the developers and the users. In other words, the model is trained and will be executed on the same type of machine. This prevents the approximation technique from being adopted in areas where the model is highly device-dependent, e.g., mobile applications. The cost model has to be re-constructed through training each machine users may have. However, it would be impractical for the developers to train the application for all possible devices before shipping the application to users. An effective retraining process on the user's machine would be ideal. It should be performed when downloading/installing an application onto the machine. Note that in this work, I target applications whose output quality will not change when being executed on different machines. In other words, making performance or other hardware-related metrics (cache miss, memory footprint) the quality is out of the scope of my work.

- Quality Notion Definition

Another assumption made in existing approaches is that there exists a pre-defined quality metric for all applications. Assuming "quality" as a hard metric (a metric with a fixed and objective evaluation method), like the Cost (either time or energy), makes the quality model construction easier. However, such one-size-fits-all approach forces the users to accept a quality notion that may not make sense to him/her. For several applications, multiple

metrics can be used for evaluating quality [32]. For example, both clustering accuracy and mean centroid distance can be used as metrics for k-means clustering [33]. In addition, no distinction is made between the application developer and application user, limiting user-level customized quality and higher-level reasoning.

1.3.3 Insufficient Support for Multi-Programming

All existing approaches are developed based on the assumption that the target application is the only running application on the platform, where the performance disturbances are treated as noise. In a multi-programming scenario, n applications are executed (active) at the same time on a target platform. A user assigns an unique resource budget to each application. Multiple concurrent executions of the same application are possible, each with their individual execution time budgets. The multi-application configuration selection problem determines a configuration for each active application such that (1) an application can finish within the user specified time budget, if such configurations exist, and (2) the overall quality across all active applications is as high as possible. Since all applications execute on the same hardware platform, they interfere with each other through resource sharing (e.g., memory hierarchy, CPUs / cores, buses / communication networks). Different configurations may have different resource footprints and different quality outcomes, making an optimal or close to optimal selection of configurations across all applications a significant challenge.

Control theoretical approaches aim to describe the system using closed mathematical formulations such as differential equations (e.g., [20]) to adjust runtime behaviors through reconfiguration based on the observed system state, which seems to be a candidate solution in this case. However, as pointed out by Caloree [21], these approaches could easily fall into local optimal solution compared to learning based approaches due to the inherit nature of control theory. Based on my observations, there are also several drawbacks of directly extending single-application control strategies to multi-programming environments,

including frequent reconfiguration, high failure rate, low output quality, etc. Unfortunately, none of the existing approaches ever considers this special and common case in real world environments.

1.3.4 Summary of Challenges

To summarize, I list the main challenges for approximation:

- How to expressively define and construct the configuration space. As a developer, hardcoding inter-knob constraints (dependencies) in source code is error-prone and it is not easy to directly encode such dependencies into a machine learning model, especially when the constraints are rather complex as shown in Equation 1.3.

$$(2 \leq itr \leq 5) \Rightarrow (hash = 8) \wedge (4 \leq probe \leq 6) \tag{1.3}$$

- How to effectively assess the cost and quality of application outcomes using different configurations. There are hundreds or thousands of configurations even for a relatively simple example application with only three knobs. Training the entire configuration space is usually time-consuming and infeasible if the training has to be done on the users' end, e.g., after shipping the application to a user's device whose cost were unknown. There should be an effective way to intelligently sample the configuration space and construct the cost and quality models on a much smaller training set.

- How to support the subjective application quality metric from different users. The overall quality metric will change accordingly when the user tunes the preferences on different sub-metrics. There should be a smarter way to quickly reconstruct the quality model instead of performing a new round of training and re-evaluating the results.

- How to extend the one-app control strategy to the multi-programming environment. Treating a set of n applications as a single, meta application would allow these strategies to be applied to the multi-programming case. However, the resulting size of the combined configuration space is exponential in n, making this approach infeasible in practice. Moreover, the performance model is constructed based on the observations obtained from running the application under a stable environment. However, in multi-programming environment, the more dynamic and unpredictable behaviors could pose major performance impact to each active application. The globally optimal configuration selection requires the system to understand the mutual impact of "local" configuration selections on the "global" runtime environment..

1.4 Thesis

1.4.1 Thesis Statement

In this dissertation, I introduce the scenario where there is the need to sacrifice quality for a reduced resource consumption using the technique of approximation. I also point out that even the state-of-the-art approaches have overlooked and failed to solve some of the key issues which are major roadblocks of using approximation to benefit developers and users. My work can be summarized as:

"Ensure mission completeness with resource constraints in uncertain environments by utilizing the resource wisely and effectively."

1.4.2 Contribution

Approximation can be highly application-specific. Multiple research groups have proposed a variety of specific approximation techniques, including Loop Perforation [26], Precision Scaling [34], Instruction Memoization [35], Task Dropping [10], Data Sampling [36], etc. Rather than proposing new approximation techniques, this dissertation focuses on designing a generic development framework and a runtime execution model for developing and

Table 1.2: Roles and Challenges of the different Stakeholders

	Expertise	**Challenge**
Developer	Implementation	Configuration Space
User	Behavior Expectation	Custom Quality
Framework	Model construction Runtime Control	Porting/Re-training Models Environmental Disturbance

executing approximate applications with those techniques mentioned above. I designed and implemented Rapids, a framework that allows developers to expressively integrate approximation strategies when designing and developing applications, enables an efficient application training process, and enables users to fine-tune the application by providing customize quality preferences. Beyond that, this dissertation also makes the first step into cross-application approximation management. I developed Rapids-M, the first system to extend approximation management to a multi-programming environment that can handle the much more dynamic environments.

Approximation management can be done effectively and efficiently in both single and multi-application environments addressing the shortcomings listed above. My thesis addresses the shortcomings as follows:

1.4.3 Rapids: A Framework for Single Application Approximation Management

- Separation of roles:

One key design feature of Rapids is that it makes an explicit distinction between the application developer, the application user, and the framework itself. Each of these three components has a particular role to play as summarized in Table 1.2. The idea is to partition the whole problem to different roles according to their expertise. The developers have key insights about the application implementation, but might not be aware of the expectation from different individual users. Different users may need to fine-tune the application to their own needs, but lack the knowledge of details on the implementation and models. The framework should be focusing on the model construction and online configuration. It

should be flexible enough to deal with a variety of models and applications, rather than specifically tailored for particular approximation strategies.

- KDG*, a data structure as application representation:*

I designed KDG (**K**nob **D**ependency **G**raph), a compact DAG (Directed Acyclic Graph) based representation as the central data structure that glues all three roles of approximation together. The graph encodes all knobs with their types, ranges, and inter-knob constraints (dependencies). The KDG is a compact and complete representation of the configuration space. Developers can express insight into applications, including each knob's type, setting range, and inter-knob dependencies. The KDG allows developers to customize the configuration space based on size, safety requirements, or the desirability for the target user community of the application. It also serves as the foundation of the Cost/Quality model construction during the training. During runtime, the KDG provides an effective mechanism for users to easily fine-tune the application behavior without knowing the underlying implementation details.

- *Virtual Knobs, knobs for users:*

In the KDG, I introduce another new concept, Virtual Knobs, as a way to fill in the semantic gap between the subjective quality and specific low-level knob settings, allowing users to easily tune the application behavior at a level of abstraction that makes sense to them, without understanding the implementation details. These high-level virtual knobs can be designed and implemented by the developers using the framework. The resulting configuration space with its default quality model can thus be customized by the user to fit his / her particular needs or expectations. Customized quality models are dynamically created just-in-time in response to users' virtual knobs preferences expressed before application execution.

- *Representative Set, enables fast retraining:*

To construct the performance model, the expensive and exhaustive training will only be performed once. During the training, Rapids calculates a smaller training set, called the *Representative Set* (RS), through a smart sampling strategy alongside constructing the model. RS enables fast model reconstruction when applications are ported to other platforms.

- Effective Problem Formulation:

Rapids can compile the KDG along with the budget provided by the user to a Mixed Integer Quadratic Programming Problem (MIQCP) [37]. The compilation also supports conceptually continuous knobs, thereby avoiding the precision loss due to sampling strategies needed by previous approaches. The generated problem can be solved efficiently during runtime with low overheads to determine the optimal configuration.

- Flexible Execution Model:

The Rapids runtime system handles the minor performance disturbance due to environmental noises through constant monitoring and periodic reconfiguration. This feedback-loop can be fine tuned by developers, including the monitoring frequency, reconfigure threshold, cost violation tolerance, etc.

1.4.4 Rapids-M: The first System for Cross-Application Approximation Management

- Partitioning the problem:

The novel design feature of Rapids-M is to handle the explosion of the configuration space by partitioning the problem into two sub-problems. The first problem provides an estimated global view of the environment and generates a coarse-grained answer for each application. Then, each application uses the global result to search for optimal configurations locally. Such a global-local approach reduces the complexity of the problem and outperforms other approaches by having lower reconfiguration frequency, lower failure rate, and higher overall output quality.

- Configuration Buckets, reduces configuration space:

I introduce a new concept called $bucket$ that clusters the large number of application configurations into a few buckets based on their resource demand similarities. Such clustering strategies reduce the configuration search space size when n applications are running together from $O((m^k)^n)$ to $O(b^n)$ where m and k are the number of knobs and settings per knob for an application respectively, and b is the number of configuration buckets per application. For all of our applications, b is usually several magnitudes smaller than m^k.

- p-model and m-model, efficient slowdown prediction:

I also introduce two separate models that enable the per-configuration slowdown prediction. In my experiments, I observed that different configurations in an application may suffer from different levels of performance degradation given the same workload in the system. This makes the problem hard to solve, since the same application with different configurations are to be treated as totally different applications when predicting the slowdown. The m-model and p-model, along with the use of buckets, are the basis to define a heuristic that gives accurate enough slowdown prediction with a low overhead.

- Global Runtime Manager:

Rapids-M includes a runtime manager that enables the isolation between applications when searching for optimal configurations, i.e., no inter-application communication is required. This centralized manager monitors the progress of each application, and re-evaluates the optimal configuration for each application whenever the environment changes, for example, when an application terminates, a new applications join, or other environmental changes.

1.5 Evaluation Summary

The evaluation of Rapids is based on a benchmark suite of eight applications running on Linux and Android systems. Four of these applications have customizable quality notions.

I evaluate the system with respect to the three main contributions: 1) Developers: Configuration space reduction from developer encoded insights, 2) Framework: Training time reduction and the model accuracy, 3) Users: Improvement of user-preferred sub-metrics relative to default metrics. Finally, I evaluate the runtime performance by measuring the overhead and the overall output quality. Benchmark applications show an average configuration space reduction of 68.7%. Two RS strategies further reduce the configuration space and result in a training time reduction of 87.2% or 92.4% compared to state-of-the-art approaches while maintaining cost prediction errors of less than 2.5% across all applications. Instead of using a control-theoretical approach, configuration management is formulated as a mixed integer quadratic constrained optimization problem which is solved directly. The overhead of dynamic reconfiguration for execution time or energy consumption remained below 3.2%, with 1.84% on average. Customized quality results in significantly improved user-preferred quality outcomes, with 1.76× improvement on average across a range of user supplied resource budgets, and over 3× improvement in some cases. In general, by a little extra efforts from developers, the training overhead can be greatly reduced by filtering out invalid configurations. Comparing to enforce constraints in source code, using KDG is a more convenient approach, not only because a graph representation is more intuitive, but also for avoiding the studious and error prone hard-coding. Enabling custom quality is a novel idea that takes users' opinion into consideration when deciding the quality metric.

In multi-programming environments, Rapids-M achieves 3.4% higher success rate when the target quad-core system is not busy (up to 4 active apps), and 22.75% higher when busy compared to existing approaches in which each application adapts itself individually. This translates to 2.6% (not busy) and 52.99% (busy) higher overall output quality. Furthermore, Rapids-M achieves such improvement with an average of 40% fewer performed reconfigurations.

I evaluated the approach on both the framework design and the application performance to motivate the future study in approximation. A new designing philosophy for approximate

applications and techniques to support approximation in multi-programming environments makes a valuable contribution to a wider acceptance of approximation.

1.6 Organization

The remainder of the dissertation is organized as follows:

- Chapter-2 summarizes the state-of-the-art approaches on multiple aspects of approximation and point out the respective limitations. These limitations are also parts of the motivation of this thesis.

- Chapter-3 describes eight benchmark applications for evaluation, 6 of which are LINUX applications adopted from open-source benchmark projects, and the other 2 are designed under Rapids from scratch. These applications are selected to cover multiple real world use cases where users can benefit from approximation.

- Chapter-4 introduces Rapids, a framework for application-level configuration and quality management. At the end of the chapter, I report some of the key evaluation results on Rapids (e.g., Execution quality improvement, custom quality support).

- Chapter-5 introduces Rapids-M, an extension of Rapids that targets the approximation management in a multi-programming environment. At the end of the chapter, I report some of the key evaluation results in execution quality improvement.

- Chapter-6 describes the detailed implementation of both Rapids and Rapids-M, including the framework infrastructure overview, code architecture, API's for hooking up applications and fine-tuning the framework, training process, required effort from developers, and user involvement. Both Rapids and Rapids-M have been made publicly available on Github.

- In Chapter-7, I report the evaluation of Rapids and Rapids-M on the training time reduction, the model accuracy, the runtime output QoS, and the support for custom

QoS.

- Chapter-7 concludes and discusses future research directions.

CHAPTER 2
STATE OF THE ART

Multiple groups of researchers have made tremendous progress on different aspects in the field of approximation. In this chapter, I list some representative works that motivated, inspired, or are highly related to my work. These works are categorized based on their different focus areas.

- **Development Support for Approximation**: On one hand, developing approximate applications requires interactions with the application programmer/developer since configuration management is a semantic issue. On the other hand, the burden on the developer needs to be reduced through new abstractions and automatic techniques. Most existing approaches for adaptive configuration management target applications that have been written without approximation in mind. Compiler-based automatic techniques [20, 38, 39, 40, 41, 42] identify program variables or functions as "knobs". This is similar in spirit to the "dusty deck" approach to automatic parallelization and vectorization which has been only partially successful [43, 44, 45]. Approaches like [46, 47, 20, 21] use automated configuration identification to minimize application developers' effort. However, a "dusty deck" approach does not allow the program developer to express insights into relevant structures and behaviors of an application, e.g., relative constraints between pairs of knob values. Alternatively, language-based approaches provide language extensions [48, 49, 50, 51] or runtime libraries for developers, making approximation part of the semantics of the program. The application developer is responsible for writing code to implement configuration or quality management. For example, in EnerJ [31], developers may annotate variables as "approximate" or "precise" so that the execution can be deployed on different hardware to save energy. In Petabricks [28], developers can provide alternative, approximate im-

plementations of functions that can be selected at runtime. FlexJava [30] is in the same spirit as EnerJ but using fewer annotations and allowing some level of quality management. There are many more language / library based systems with different abstractions to manage configurations and quality. However, these languages have only limited, if any, support for reconfiguration. Most importantly, the language semantics approach solely relies on the developer to hardcode the configuration management strategy, a challenging and potentially error prone task.

- **Model Construction**: Current approaches establish a correlation between configurations and an application's resource cost (performance or energy) and quality outcome through a profiling (training) phase [20, 12, 25, 6, 26], where all or randomly sub-sampled [21] configurations are executed on the target machine, and their costs and quality are measured and recorded. The cost of training is proportional to the size of the configuration space which may lead to significant training times in practice. For example, [22] reports profiling times of more than two weeks for an evaluated application. This is typically not acceptable if the application is to be deployed by users on new machines which would require a full retraining. Therefore, enabling a fast offline profiling is crucial in many cases when the application needs to be executed on different target machines.

Moving from a discrete to continuous configuration space also requires a model that can capture the non-linear behavior of the system. [52] and [53] propose to control a non-linear system using piece-wise linear approximation that translates the problem into a mixed integer programming problem (MIP)[54]. Further, the ability to port the configuration space management system to new hardware/software platforms is important but new hardware/-software configurations may not be known a priori. The result may be significant porting costs because inevitably the training process has to be redone due to the change on resource availability (different memory/cache size, number of cores, etc). A random sampling strategy (e.g., Caloree[21] takes 20 random samples) could help speed up the process. But in order to work well, it needs some insurance of a reasonable sample coverage to determine

configuration costs and qualities. For example, 2 samples may be enough to accurately construct the model for some applications, while some other applications may require way more than 20 samples. In my work, Rapids carefully selects the samples based on their importance to constructing an accurate model, and the number of samples is determined by the application itself and the model error tolerance.

- **Hardware Architectural Support**: Hardware support for probabilistic and approximate computing has also received significant attention in recent years allowing the use of faulty hardware and operations [55, 56, 57, 58, 59, 60, 61, 62] or operations [63, 64]. However, they require the application developers to carefully identify the configurable components, but do not allow developers to express more complex correlations between components and their settings. Also, different approximation technique requires different hardware designs, e.g., approximate storage, approximate instruction execution, etc. Each of these approaches is not general enough to encode other techniques than what it was designed for. A practical developing framework[65, 66, 67] for configurable applications with end-to-end support[68] for developers has remained an unsolved challenge.

- **Resource Consumption Prediction**: Exploiting application performance degradation has been explored by several groups. Significant research focuses on constructing the cost model. Learning based models [12, 69, 70, 13, 71, 19, 72] predict performance through either input and/or execution features, whose accuracy is bound by the richness of the data set. Additionally, examining each input introduce significant runtime overhead (e.g., the rendering logic for a web-page can only be determined after extracting and evaluating the features of the page in Chamelon [29]). Though Rapids-M uses a similar learning-based approach to predict the slowdown for each configuration by examining the features of active applications, such examination only happens once when the global environment changes rather than for each input data (work unit). Control theoretical approaches [73, 6, 20, 74, 22] aim to deal with runtime disturbance. However, to directly extend these approaches

to a multi-programming environment, the model has to be built on the entire search space which is infeasible due to the search space size. Even getting the profile for a single large application may take weeks ([22]).

- **Quality Prediction**: Probabilistic and approximate programming uses probability variables and their distributions to predict the output quality[75, 76, 77, 78, 79, 80, 81]. This type of research focuses on representations of the distributions and operations induced by operations on their associated probabilistic approximate variables. In the database community, approximation has been used to minimize the query overhead given a predicted statistical error bounds of query results [82, 83], and more recently in the context of Map-Reduce [84] applications [85, 10]. Proving and/or verifying approximation error bounds has also been the topic of ongoing research [86, 87]. However these approaches perform repeated training if quality metrics changes, thus cannot effectively enable users to express different quality preferences.

- **Custom Quality**: Existing approaches further assume some pre-defined quality notion where each application "comes with" a quality function for its output, e.g., PSNR for video processing [13]. Akturk et al. [88] categorizes the quality metric used in common approximation applications including some of my benchmark applications (Bodytrack, Swaptions). However, many applications have subjective quality notions, so the application user needs to be involved in defining the quality function. For example, a face-detection application may use F-score [89] as the quality metric, which is defined as the harmonic mean of the recognition precision and recall. However, the precision can be more important than recall when used in target recognition. On the other hand, higher recall can help the application performance when used in crowd counting. Allowing users to express such preference requires the system to be flexible enough that the quality model can be quickly updated when such preferences change. It is too time consuming to repeat the training process to reflect any changes to the quality notion.

- **Dynamic Reconfiguration**: The ability to dynamically reconfigure is crucial in approximate computation to adapt to the inherent error in constructed models or unpredictable disturbance during runtime. The former issue could be a result of input dependencies or normal runtime noise. Approaches like [12, 29] fine-tune the cost model by evaluating each input. PowerDial [20], Jouleguard [6] and other control-thereotical approaches [90, 91, 92] continuously monitor the resource usage, and reconfigure when considered beneficial. However, none of these approaches can be directly extended to multi-programming environment.

- **Cross-application Interference Prediction**: Optimizing the behaviors of groups of applications in a multi-programming environment has been the goal of different research efforts [93, 94, 71]. Models for predicting application interference have been investigate due to the non-linear impact of resource sharing on individually observed application slowdowns. D-Factor [95] explores the inter-application performance degradation through computing the slow-down factor measured by the degradation when running with computation or memory-intensive stressers. However, D-Factor requires the measurement/observation of the current system footprint for the prediction. The ESP system [71] is similar to our approach since it measures specific system footprints for different applications. However, since approximation is not considered, each application has only a single footprint, resulting in a very small set of samples over which to train their model. Also, ESP is limited since it requires the training process to be performed for k applications running simultaneously to produce a model that is able to predict the slow-down among a subset of those k applications. In contrast, our approach uses comprehensive configuration spaces for each application. Also, each application is trained individually, making our approach more flexible and scalable since groups of applications do not have to be known and to be trained as a group in advance.

- **Specific Approximation Techniques**: There exists a large family of approximation tech-

niques, and these techniques can be highly application-specific. This thesis aims to develop a system to seamlessly exploit approximation techniques developed by others, instead of introducing new techniques. Each of the following techniques can potentially be encoded in Rapids as a knob. For applications with iterative computations, Loop Perforation [26, 96] gives approximate answers by skipping certain iterations. Compute and memory-intensive applications with precision tolerance can benefit from precision scaling [34, 97, 98]. Memoization is also a technique used to speed up floating point calculation by reusing results of similar computation instructions [35, 99]. Dropping inputs through sampling is commonly used as the approximation for applications with large set of inputs [50, 10, 100]. Similarly, dropping tasks or jobs is common for applications under large multi-task frameworks, e.g., on GPU [36] or Map-Reduce frameworks[85, 10, 101].

CHAPTER 3
SAMPLE APPLICATIONS

To assess the practical, end-to-end effectiveness of Rapids, I implemented and evaluated a prototype system. Besides the Ferret application introduced in Chapter 1, the evaluation uses eight different sample applications/workloads in total, six of which are Linux applications and the other two are on Android platforms. Five of the Linux applications are from widely used benchmarks with known quality metrics, namely Swaptions[14], Bodytrack[14], Ferret[14], SVM[102], and NeuralNet (NN)[102]. The other one, FaceDetection uses the OpenCV standard library[103], and the two Android applications, NavApp and VideoApp, are freshly developed using the Rapids framework. The eight workloads have been chosen to illustrate the features and benefits of the Rapids and Rapids-M frameworks while allowing others to assess the effectiveness through applications used by related work.

- Swaptions is a financial analysis application to calculate the pricing of a portfolio of swaptions. For each swaption, it performs a Monte-Carlo [104] simulation and reports the result pricing. The computation is based on an iterative simulation algorithm. The execution time and accuracy increases as the number of simulations increases.

Knob: (1). It controls the number of iterations to simulate, within [100000, 1000000]. Larger number of iterations leads to higher overhead but can better ensure the convergence of the calculation.

Dependencies: I do not encode dependencies in Swaptions because it is a single-knob application.

QoS Metric: The QoS loss for each swaption can be computed using the vector distortion [105] described in Equation 3.1. n and w_i are the total number of elements in the

vector and their weights, and in the case, $n = w_i = 1$. $\hat{y}$ is the computed price from the execution and y is the price when executing with highest setting. The total QoS loss is the average loss across all swaptions.

$$\sum_{i=1}^{n} w_i * abs((\hat{y}_i - y_i)/y_i) \tag{3.1}$$

- Bodytrack is a computer vision application that tracks a set of human body components from a video frame by frame. Bodytrack employs an annealed particle filter to track the pose using edges and the foreground silhouette as image features, based on a 10 segment 3D kinematic tree body model. These two image features were chosen because they exhibit a high degree of invariance under a wide range of conditions and because they are easy to extract. An annealed particle filter was employed in order to be able to search high dimensional configuration spaces without having to rely on any assumptions of the tracked body such as the existence of markers or constrained movements.

Knob: (2). One sets the number of annealing layers with 5 settings from {1,2,3,4,5}. The other sets the number of particles to track within [100, 4000]. Intuitively, tracking more particles in each annealing layer results in more accurate estimation on body components for having more estimations. More layers gives more opportunities to refine the estimation. However, tuning up any of the two knobs can increase the execution time for performing more calculation.

Dependencies: Dependencies exist between the lower annealing layers and higher particle number layers. The intuition is that if a frame is processed with fewer layers, more particles should be considered for a meaningful tracking result.

QoS Metric: Each output line is a vector which represents the position of different body components. I use Equation 3.1 for QoS with y once again representing the highest settings and w_i set proportional to the size of the body component being tracked.

- Ferret is a ranking application that accepts image queries and returns the top-K images in a database ranked by content-similarity. The application has two phases. It first locates the top-$2K$ images using a Multi-Probe LSH [15] algorithm. Then, it calls a routine that computes the Earth Mover's Distance (EMD) to rank the images and return the top-K.

Knob: (3) The first sets the number of hash buckets per table within [2, 8]. The second sets the number of probing buckets in the multi-probe phase from [2, 20]. Similarly to Ferret, having more hash buckets or probing more tables could result in higher overhead but can cover more candidates to select the better top-$2K$ results. The third sets the maximum number of iterations when computing the EMD within [20, 500]. Setting a lower value on this knob could cause earlier termination of the EMD calculation before the convergence.

Dependencies: Dependencies exist between the first and the second knob by having edges among the lower number of hash buckets and higher number of probing buckets. A table with a small number of buckets has to probe more buckets to find more accurate results. There are also edges between lower numbers of buckets and higher numbers of iterations. This gives more accurately ranked top-K results given relative poor top-2K candidates.

QoS Metric: The output of a query image is a sorted list containing the ID of the top-K images. I use a common ranking function that compares two lists ($list_1, list_2$) of results for a query:

$$\sum_{i \in Z} |rank_1(i) - rank_2(i)| - \sum_{i \in S} rank_1(i) - \sum_{i \in T} rank_2(i) \tag{3.2}$$

Here, Z is the set of result images appearing in both $list_1$ and $list_2$. S and T are the sets of images appearing exclusively in $list_1$ and $list_2$. $rank_1$ is the rank of an image in $list_1$, and $rank_2$ the rank in $list_2$. Given N results for each query, the results for the equation range from 0 to $-(N+1) * N$.

$$\begin{aligned} err =& 2 * (k - z)(k + 1) + \\ & \sum_{i \in Z} |r_1(i) - r_2(i)| - \sum_{i \in S} r_1(i) - \sum_{i \in T} r_2(i) \end{aligned} \tag{3.3}$$

with $Q = 1 - err/k(k+1)$

Here, Z is the set of images appearing in both $list_1$ and $list_2$ of size z. S and T are the sets exclusively in $list_1$ and $list_2$ of size k. r_1 and r_2 are the ranks of an image in $list_1$ and $list_2$.

Customization: $Coverage$ and $Ranking$ are exposed as virtual knobs to users. $Coverage$ is calculated by the first addend in Equation 3.3. Given a fixed k, higher z results in higher $Coverage$, i.e., more "correct" results being returned. $Ranking$ is the rest of the equation. Ranking "wrong" results lower yields better $Ranking$.

- FaceDetection is a vision application that detects human faces from a series of input images. First a multi-level image pyramid [106] scans each level looking for a face of a fixed size. If a face is found, it filters out false positives by examining nearby pixels. Optionally, it performs a round of filtering by checking the presence of eyes (one or two) in each detected area.

Knob: (3) The first determines the number of pyramid levels, ranging from $\{5,10,15,20\}$. The next determines the number of neighbors to examine in the filtering phase $\{0,4,8\}$. Higher value on the first two knobs gives a more accurate recognition result for having a larger window for examination for each recognized face but require more computation. The last controls the threshold of the minimum number of eyes detected $\{0,1,2\}$. Larger settings on this knob enforces a stricter rule to determine whether a face is truly a face. It may lead to lower recall, but will definitely help the precision. Turning this knob to 0 will save a significant amount of computation time because this step will be skipped in the program. All three knobs are discrete only in FaceDetection.

Dependencies: I do not encode dependencies in FaceDetection because 1) all configurations produce a reasonable results, and 2) each of the two sub-metrics benefits from an opposite knob settings from the other, therefore there does exist obvious dependencies.

QoS Metric: I adopt the standard measurement of recognition performance, the F-measure [89]

using Equation 4.7. By default, $k1 = k2 = 1$ (F1-score). *Customization:*$Precision$ and $Recall$. When $k1 > k2$, precision weighs more than recall, and vice-versa.

- SVM and NN are two supervised learning applications that classify input images. They run 1000 iterations on a set of labeled training data and construct a SVM (Support Vector Machine) and a NN (Neural Network) model for classification.

Knob: (3) There are three configurable knobs. The first determines the learning rate, within [1e-7,1e-5]. The next knob is Discrete only and it determines the batch size {64, 128, 256, 512, 1024}. The last one determines the regularization rate within [5000, 25000]. Intuitively, a larger batch size would provide a more accurate gradient direction with a higher cost. For learning rate, a small learning rate would converge slower after each iteration, however a more aggressive learning rate may miss the optimal point and result in bad quality.

Dependencies: I intentionally do not encode dependencies in these apps to show that Rapids can be treated as a parameter tuning strategy in machine learning, i.e., locating the optimal parameters for highest accuracy through quality model.

QoS Metric: Prediction accuracy as shown in Equation 3.4.

$$accuracy = \frac{correct\_prediction}{total\_input} \tag{3.4}$$

- NavApp is a navigation application to guide a user from location A to B. It uses the Google-Maps API to compute estimated travel time and constantly shows the user's current location. Runtime uncertainty comes from the user behavior, real-time traffic and the Google Map API's prediction error.

Knob: (4) One knob controls the screen brightness from [1%, 100%]. The second is Discrete only and controls the map layout {basic, satellite, and hybrid}. The third sets the polling frequency of location reading {5s, 8s, 10s}. The forth controls the GPS {on, off}.

Dependencies: I inserted edges between the information and the screen brightness from the basic consideration of human ability to interpret simpler information on darker screens. It also have dependencies from localization frequency to map display.

QoS Metric: NavApp is a real world application that composes different real service components. There is not a concrete well-defined QoS metric for such an application. I define the QoS metric to be a weighted sum over all individual sub-metrics.

Customization: Priorities can be given to $brightness$, $localization$, and $information$ as shown in Equation 3.5.

$$\begin{aligned} Q &= w_b * brightness + w_l * localization + w_i * information \\ brightness &= Screen/100.0 \\ localization &= \frac{PollingFreq/3.0 + GPS/2.0}{2} \\ information &= Map/3.0 \end{aligned} \tag{3.5}$$

- VideoApp allows the user to watch a high-resolution video locally or stream a lower-quality video from a remote server.

Knob: (4) One controls the screen brightness as in NavApp. The second controls the video frame-rate from {15fps, 30fps, 45fps, 60fps}. The third controls the video resolution from {144P, 240P, 480P, 720P}. The last controls the network from {On, Off}

Dependencies: I used edges from lower frame-rate to lower resolutions to avoid unreasonable situations, e.g. displaying a 720P/15fps or a 240P/60FPS video. Also, I have dependencies among network and resolutions to either deliver high quality video from the server or a low quality video locally

QoS Metric: As in NavApp, VideoApp does not come with a well-defined overall quality metric. I again adopted the weighted sum as in NavApp to calculate the overall performance as shown in Equation 3.6.

Customization: Priorities can be given to $brightness$, $smoothness$, and $resolution$.

$$\begin{aligned} Q &= w_b * brightness + w_s * smoothness + w_r * resolution \\ brightness &= Screen/100.0 \\ smoothness &= FPS/60.0 \\ resolution &= Resolution/720.0 \end{aligned} \tag{3.6}$$

Table 3.1 shows the significant opportunities for approximation in all the sample applications. $Min\ Cost$ reports the lowest cost relative to the default setting in execution time or power consumption. $Min\ Quality$ reports the quality degradation under the default quality metric. The last two columns summarize the opportunity for both developers and users to participate in Rapids.

Table 3.1: Approximation Opportunity

	Min Cost	Min Quality	Dependency	Custom Quality
Swaptions	10.08%	57.78%	-	-
SVM	57.98%	49.96%	-	-
NN	76.55%	42.8%	-	-
Bodytrack	7.51%	40.84%	yes	-
FaceDetection	27.43%	41.23%	-	yes
Ferret	37.26%	54.91%	yes	yes
NavApp	56.12%	22.2%	yes	yes
VideoApp	67.62%	27.7%	yes	yes

- Environments for Experimental Evaluation:

For the sample applications, developers performed the off-line training phase on a Linux machine or an Android phone. Application users ran the applications on an embedded Linux board or a separate phone. Cost is execution time for Swaptions, Bodytrack, Ferret, and FaceDetection; cost is energy consumption for NavApp and VideoApp. Key specifications are:

LINUX machines $Developer$: 6-Cores at 3.7GHz, —- 16GB RAM at 2666MHz; $Target$: Nvidia TX1 [107], 1.9 GHz 64-bit 4-core —- 2 MB L2 cache. **Android machines:** $Developer$: Nexus-5: 2.26GHz 4-core processor — 2GB RAM. 4.95-inch screen, 1080x1920 pixels; $Target$: Nexus-6: 2.26GHz 4-core processor — 3GB RAM. 5.96-inch screen, 1440x2560 pixels.

CHAPTER 4

RAPIDS

In this chapter, I introduce Rapids, a system to support approximation management in the single-application scenario. The main contributions and theoretical foundation are described in the first three sections. At the end, I report some of the key results comparing to other works. A more detailed discussion of the implementation and a thorough experimental evaluation can be found in Section 6.1 and Section 7.1, respectively.

4.1 Introduction

The objective of application-level configuration management is to trade application output quality for resource consumption under user-supplied budget constraints. However, not all applications can be reconfigured. There are several conditions for approximate applications that this work targets.

(A) There is a useful correlation between resource usage and application outcomes. Applications have to be tunable, with a set of *knobs* and their corresponding settings, i.e., there has to be a feasible *configuration space*.

(B) There are well-defined points during application execution that allow safe and efficient reconfiguration of the application, i.e., transition from the current to a new target configuration.

(C) The application outcomes under different configurations can be compared and ranked according to a quality metric. Optionally, the application quality can be customizable in a way that makes sense to application users.

I will use NavApp as an illustrating example in this chapter instead of Ferret because 1) the example of NavApp shows a different type of use case in contrast to Ferret where energy is used as the cost metric, and 2) the KDG for NavApp includes OR edges which

were not presented in Ferret.

NavApp - an illustrating example: NavApp provides driving directions to the desired destination and runs on mobile platforms. In NavApp, the configurable components include the brightness of the screen, the way the route is communicated and displayed (map, satellite image, text-only), and the spatio-temporal precision of the actual reported location. Each such component (knob) introduces corresponding energy costs. For example, a brighter screen consumes more energy than a dimmer screen, rendering a route on a satellite image for display is more energy expensive than displaying the direction in a simple text box, or polling the GPS more frequently to get more precise localization needs more power. However, there typically are dependencies among these knobs. For example, displaying a high-resolution satellite image on a dimmed screen may not be feasible due to low readability.

The user has to provide an energy budget (e.g., 5% of currently remaining battery energy) and relative preference of the different components (e.g., route display quality is more critical than localization precision). The latter defines the desired user-customized quality experience. Based on the provided destination and overall energy budget, NavApp determines the predicted travel time, thereby determining the power constraint, i.e., energy per second in milliwatts, (each second of execution is considered a work unit) for the entire application execution. The goal is to give an as-even-as-possible quality experience, choosing a configuration that maximizes the user-specified knob priorities while respecting the overall energy budget. Reconfiguration would be necessary if the power constraint (budget per work unit) changes, for instance, due to a longer or shorter predicted mission time. For example, if the driver takes a wrong turn, the remaining energy budget may not sustain the current service quality level, and therefore, a lower quality configuration would be chosen for the remainder of the trip.

Any system that supports application-level approximations has to define the configuration space, allow the specification of a custom quality notion, model the cost and quality of

each configuration, and provide an efficient algorithm to perform reconfigurations. Existing approaches [25, 20, 6, 12, 26] treat applications primarily as "black boxes" with some program variables exposed as "knobs" that may be manipulated in order to influence the cost and quality during the execution. A training phase for all or randomly sub-sampled [21] configurations is used to establish a correlation between knob settings and the observed program quality and resource usage. Once training is complete, a selected configuration can be modified to adapt to real-time constraints using control-theoretical strategies. With the explosion of the configuration space, the training cost can be huge (up to weeks [22]). Such training overhead is unacceptable, especially when the model has to be re-built acceptably quickly when porting the application to another device, or if updating the current cost or quality models becomes necessary [69]. I explore how structural knowledge about the application and their knobs and value ranges can be leveraged to significantly reduce the configuration space that needs to be trained to build accurate cost and quality models for the entire feasible configuration space. The configuration space reduction leads to a reduced training time, in some cases by orders of magnitude. In order to help application users get involved in the quality determination (if needed), I propose a quality model that is flexible enough for users to fine-tune the application behavior without understanding the underlying implementation details.

I introduce Rapids (**R**econfiguration, **A**pproximation, **P**references, **I**mplementation, **D**ependencies, and **S**tructure; some key aspects of my approach.), a new programming and runtime framework that considers the structure of applications for customized configuration management. For each application and target system, application developers first define the application structure with its knobs and the dependencies between them. Rapids performs an initial training to construct the cost and quality models, and encode them along with structural information in the KDG (**K**nob **D**ependence **G**raph). Rapids also calculates a small set of representative configurations to be used as a training set to significantly reduce the cost for reconstruct the cost model. In Rapids, the application developer can

optionally implement "virtual knobs" that allow users to express their quality expectations at a level of abstraction that makes sense to them. These high-level virtual knobs close the semantic gap between subjective quality and specific, low-level knob settings. The resulting configuration space with its default, developer-provided quality model can thus be customized by the user to fit his / her particular needs or expectations. During runtime, application users specify the relative importance of each knob for his/her overall, customized quality experience without detailed knowledge of the application structure. Rapids then uses its structure-aware quality model, the customized knob priorities, and the provided budget to adapt application behaviors with the goal of maximizing overall quality while respecting the provided budget. Rapids automatically reconfigures if the observed resource consumption does not match the predicted consumption. The contributions of this chapter are:

1. KDG: A new structural approach to specifying approximate applications with knob dependencies. In KDG, I introduce multiple new concepts:

    - Dependencies: A new way to encode inter-knob constraints in optimization problems that significantly reduces the feasible configuration space by leveraging the developers' insights.
    - Representative Set: A much smaller subset of configurations that can be used to effectively rebuild the cost model when porting applications to unknown platforms.
    - Virtual Knobs: A bridge closing the gap between the detailed underlying implementation and the high level quality notions that allows users to customize application behavior without much effort.

2. A new strategy to formalize the optimization problem that determines the optimal configuration under a budget based on KDG.

3. The implementation of the Rapids programming and runtime system, and its evaluation on a range of important tunable applications. Rapids and its application benchmark suite will be made publicly available, allowing my results to be reproduced and extended.

The evaluation of Rapids is based on a benchmark of eight applications running on Linux and Android systems. Four of these applications have customizable quality notions. For the applications with customizable quality, the default quality model only achieves as little as 34.8% of the quality, compared to the customized case, with 81.9% on average across the three applications. Rapids allows users to tune the developer-provided default quality model, significantly improving an application's quality outcome with 1.76× improvement on average across a range of user-supplied resource budgets, and over 3× improvement in some cases. Benchmark applications show an average configuration space reduction of 68.7%. The two RS (Representative Set) strategies further reduce the configuration space and result in a training time reduction of 87.2% or 92.4% compared to state-of-the-art approaches, while maintaining cost prediction errors of less than 2.5%. Instead of using a control-theoretical approach, configuration management is formulated as a Mixed Integer Quadratic Constrained Programming (MIQCP) problem that can be solved directly. The overhead of dynamic reconfiguration for execution time or energy consumption remained below 3.2%, with 1.84% on average.

4.2 Application Representation: KDG

To understand Rapids, I must first introduce the KDG, a compact DAG based data structure that encodes a developer's insights about the structure of the application through the graph's nodes and edges. It is a representation of the configuration space of an application and is used for offline cost / quality model construction and online optimization. It provides the basis for formulating configuration selection as a constrained optimization problem.

4.2.1 Developers' Insight as Structure

The KDG allows developers to encode information including knob type, per-knob value range and inter-knob dependency through the graph structure.

Nodes represent knob settings. The KDG supports two types of nodes: Discrete and Continuous. Each knob consists of a collection of Discrete nodes or a single Continuous node. Each Discrete node within a knob is associated with a specific setting. A Continuous node represents a possible value range of a setting.

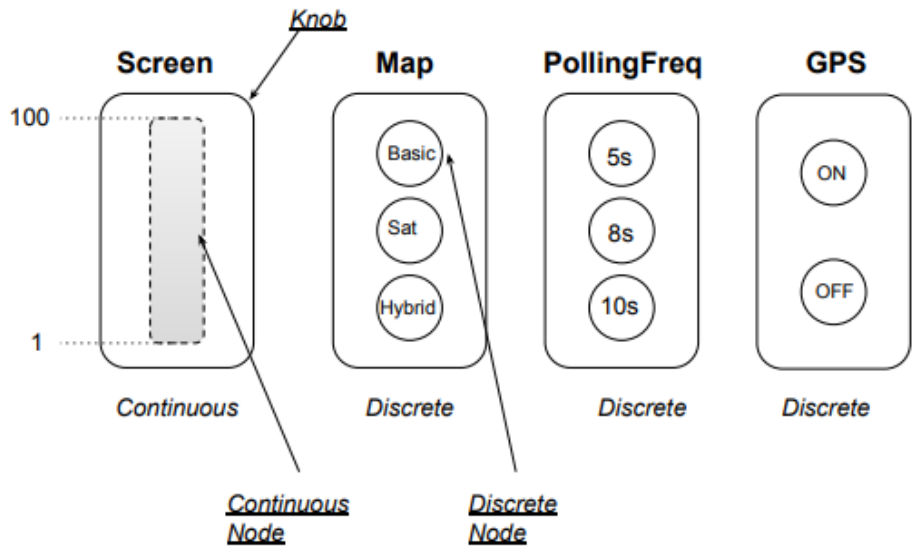


Figure 4.1: KDG nodes in NavApp

Figure 4.1 shows the knobs in NavApp. There are 4 knobs, 3 of which have discrete nodes: "Map", "GPS", and "PollingFreq". The last knob "Screen" has a continuous node.

Edges represent dependencies between knob settings. Edges are directed with the sink depending on the source. Edges thus encode developers' insight into inter-component dependencies. They are not required for system function, but allow the developer to help guide the system and user experience. For discrete nodes, dependencies are on the entire node. In a continuous node, a dependency may be on a range of possible values, i.e., on a *segment*.

To make KDG flexible, the KDG supports two types of dependencies, *AND* and *OR*. *AND* dependencies allow a node to be dependent on a set of different nodes which are *all* needed to satisfy the dependence. In contrast, *OR* dependencies allow a node to require *at least one* node in a knob or set of knobs. That is, *OR* dependencies are grouped: at least one source node has to be selected from each *OR* group. A node *AND*-depends on all independent *OR* groups.

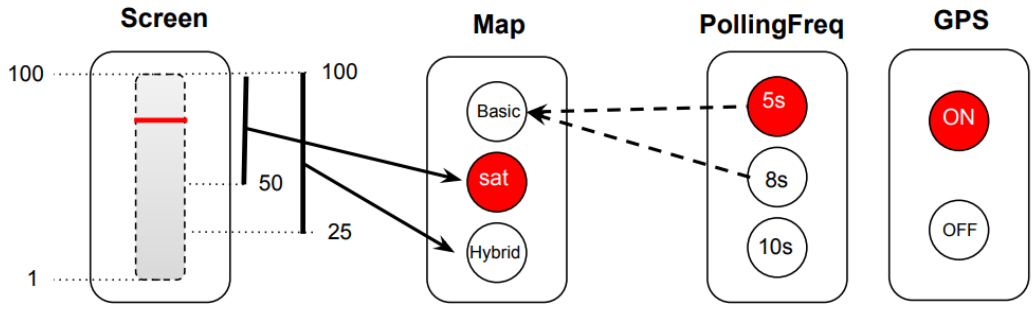


Figure 4.2: KDG nodes with edges in NavApp

In Figure 4.2, node "Sat"has an *AND* dependency on segment [50,100] in **Screen**. If a satellite image is rendered as the map layout, the screen brightness must be at least 50% because the background on satellite images are darker and is hard to interpret on a dark screen. Similarly, "Hybrid" requires at least 25% brightness for having additional highlighted road name and information overlay on the satellite image. "Basic" has *OR* dependencies on 5s, 8s in **PollingFreq** since without satellite image guidance, a basic layout requires more accurate localization for users to interpret the current location. These constraints reflect the developer's design and assessment of a desirable configuration space. Rapids supports a high level specification of the KDG as shown in Figure 6.2. I believe that this representation is much easier to understand and maintain than alternative implementations, for example, through conditional statements embedded and distributed across the source code.

Configurations are represented in the KDG as a selection of nodes or values. For knobs with Discrete nodes, a single node will be selected. For knobs with a Continuous node, a specific value will be selected. Figure 4.3 demonstrates a particular configuration. The

KDG is a compact representation of the entire configuration space where each configuration has to satisfy the edge dependencies.

In all previous work, a particular configuration in an application can be represented as a variable vector. This can also be represented by KDG with only one type of node (discrete or continuous), and no dependencies.

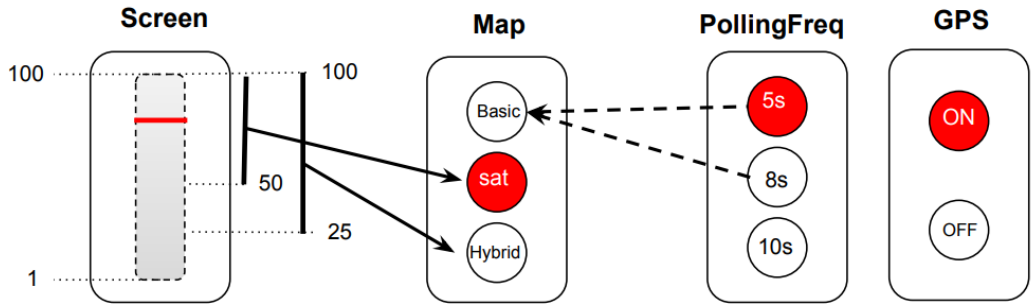


Figure 4.3: KDG sample configuration: Screen=75 (value), Map="sat", PollingFreq="5s", GPS="ON".

4.2.2 Cost Model using KDG

The KDG provided by the developer contains only structural information. A full KDG includes the structure, weights (both for cost and quality models), and user preference (as weight augmentation). A training phase is performed to collect a set of data points that map configurations to corresponding costs which are used to build a regression model that in turn provides weights to the KDG. The overall cost C of a specific configuration is calculated as the sum of (1) the contribution by each individual discrete C_{dis} and continuous C_{cont} knob, and (2) the contribution of the combined effects of each pair of knobs C_{corr}, as shown in Equation 4.1. Based on observation, a second-order linear regression model that captures the first and second order parameter values (weights) for each node, and the pair-wise knob coefficients is accurate enough for the weight prediction on both discrete and continuous nodes.

$$C = C_{dis} + C_{cont} + C_{corr} \tag{4.1}$$

Node Weights in the KDG represent the contribution of a particular node to the overall cost. For a *discrete* node, the weights for each node j in a knob i is represented by single value c_i^j. For a *continuous* node, the weight for a knob i with value range $[min_i, max_i]$ is represented by a function $F_c^i()$ that maps a value v_i within the range to its contribution. Equations 4.2 and 4.3 show the cost contribution of the discrete and continuous nodes, respectively. Only the selected discrete nodes ($v_i^j = 1$) will contribute to the overall cost.

$$C_{dis} = \sum_i \sum_j c_i^j \times v_i^j,\ v_i^j \in 0, 1 \tag{4.2}$$

$$C_{cont} = \sum_i F_c^i(v_i),\ min_i <= v_i <= max_i \tag{4.3}$$

Correlated Weights model the combined contribution of pairs of knobs. This design captures knob correlations that are more complex than simple addition. For example, in a nested loop with the two knobs representing the loop bounds, the total cost is proportional to $\#outer\_iteration * \#inner\_iteration$. Experimentally, I found that modeling quadratic relations between knobs was sufficient to capture program behaviors. Note that a numerical value $\bar{v}$ is required for discrete nodes with categorical values.

$$C_{corr} = \sum_m \sum_n corr_m^n \times (\bar{v}_m \times \bar{v}_n) \tag{4.4}$$

4.2.3 Custom Quality Metrics/Models and Virtual Knobs

Quality and cost metrics rank outcomes of application executions under different configurations. Quality metric Q *measures* an aspect of observed application outcomes. A model $\tilde{Q}$ for a metric Q *predicts* the expected measurements for a configuration without applying the metric to the observed outcome. In other words, a model $\tilde{Q}$ approximates the measured metric Q for a configuration. Even though there are input dependencies in many applications, it is assumed that configurations giving better outcomes during the training process

are also very likely to produce higher quality in real execution. The better the model, the lower the expected metric prediction errors. The produced models are used by Rapids to select the highest quality configuration under a user-provided resource budget.

Customized Quality Sub-Metrics: In general, quality is a subjective metric and therefore needs application users' involvement. If an application comes with a pre-fixed quality metric that cannot be customized, for example PNSR [13] or SSIM [108] for video playback, the construction process will be identical to the cost model except that the observation is no longer the cost but the measured quality metric value. For those applications with customizable quality, it requires at least two distinct quality sub-metrics, i.e., two distinct ways to rank configuration quality. These sub-metrics should be easily reasoned about by the application users and may have different preferences among different target users. In the presence of customizable quality, two distinct quality preferences may result in different configuration selections under the same budget, each maximizing the distinct subjective quality.

Definition: Custom Quality using Sub-Metrics — An application has custom quality if the quality Q can be represented by a function F over several weighted quality metrics $(q_1, \ldots q_n)$, $n \geq 2$. The quality metrics q_1 to q_n are referred to as sub-metrics. Users can provide relative preferences on each sub-metric through their weights.

$$Q = F([(w_1, q_1), \ldots (w_n, q_n)]) \tag{4.5}$$

Example: I will use another application, FaceDetection, as an example to illustrate the custom quality because the quality metric in FaceDetection is one typical use case where custom quality is important to different users. In FaceDetection and other similar classification problems, *F-Score* is widely used as the overall quality metric. A larger family of customized metrics can be expressed based on the precision p and recall r sub-

metrics, as defined in Equation 4.6.

$$p = \frac{TP}{TP + FP}, \qquad r = \frac{TP}{TP + FN} \tag{4.6}$$

where TP denotes True Positive, FP False Positive, and FN False Negative. Users may express different importance of sub-metrics p or r through providing different weights w_p for precision and w_r for recall in Equation 4.7. For example, the *F1-score* can be specified by choosing $w_p = w_r = 1$.

$$\begin{aligned} Q_{face} = F_{face}([w_p, p], [w_r, r]) = (1 + \beta^2) \cdot \frac{p \cdot r}{(\beta^2 \cdot p) + r} \\ , with\ \beta = \frac{w_r}{w_p} \end{aligned} \tag{4.7}$$

The two high-level sub-metrics, "Precision" and "Recall", can be easily reasoned about by application users. Given the same output result, these two metrics will have the same values since they have a fixed evaluation strategy f_p, f_r. However, the overall quality might have different values if users provide different weights to these two metrics.

Custom Quality Models: For single quality metrics Q, the quality model $\tilde{Q}$ can be constructed offline similar to the cost model as discussed in Section 4.2.2. Each configuration measurement collected from training now includes the measured quality metric in addition to the measured cost metric. The resulting quality model has coefficients for single knobs and pairs of knobs.

Rapids allows customizable quality metrics through developer-defined custom quality metrics with default weights at application development time, and user-specified sub-metric weights at application execution time. For each employed quality sub-metric q_i, Rapids builds a model $\tilde{q_i}$ through training in its "offline" phase. Table 4.1 shows the set of configurable knobs (left column) used to implement models for the recall and precision sub-metrics (right column) for the example FaceDetection application.

Table 4.1: Knobs and Sub-Metrics in FaceDetection

Configurable Knobs	Sub-Metrics
Neighbour_Pixel, Decomposition_Level, Eye_Detection_Enabled	Precision, Recall

Just before application execution, a user may customize his / her quality expectation metric Q by providing weights w_i for each sub-metric q_i as shown in Equation 4.5. Thus, Rapids must compute the quality model $\tilde{Q}$ "online" based on the known sub-metric models $\tilde{q_i}$ and the user supplied weights w_i. Unfortunately, there is often no obvious way to use the set of knobs and pairs of knobs coefficients of the individual sub-metric models $\tilde{q_i}$ to effectively compute $\tilde{Q}$.One approach is to express $\tilde{Q}$ as a mapping over weighted individual knob coefficients as illustrated in Equation 4.8.

$$\tilde{Q}([k_1, \ldots k_n]) => \tilde{Q}([w_1 * k_1, \ldots w_n * k_n]) \tag{4.8}$$

However, this straight-forward extension requires the user to know how to tune the weights on the detailed knobs, e.g., #_Neighbour_Pixel, #_Decomposition_Level, and Eye_Detection in FaceDetection, which is non-intuitive and complex because how these low-level implementation related knobs affect the overall quality is unclear. Another approach would be to re-evaluate the execution output with the new Q and reconstruct $\tilde{Q}$. However, this requires the retaining of outputs/results obtained during training and will result in significant space and time overheads. Finally, a full retraining would also be possible, but I consider this approach to be prohibitively expensive in terms of execution time, rendering it infeasible.

Instead, Rapids predicts all sub-metrics for each trained configuration in the training phase and calculates the overall quality using the developer provided function (F in Equation 4.5) with the quality models $\tilde{q_i}$ instead of the quality metrics q_i. Also, providing preferences on sub-metrics are much more straightforward. This is more efficient since q_i requires actual application execution results while $\tilde{q_i}$ only needs the configurations.

The set of calculated quality values yields the overall quality model $\tilde{Q}$ by solving the

regression problem as discussed above, and described in more detail in Section 4.3.1. This approach eliminates the overhead of backing-up execution results and the re-evaluation process. Experiments show that the overhead of dynamically constructing quality models is negligible, less than half a second for all four of the applications with customizable quality. This overhead occurs once just before application execution.

Virtual knobs: As discussed above, nodes in the KDG are associated with application-level objects (e.g. program variables) and their possible value settings, which together define the configuration space. Both cost and quality metrics are defined over this "concrete" configuration space. However, in order to allow application users to customize their quality experience, the knobs of a concrete configuration may be too low level to allow users to make an informed choice. Therefore, Rapids introduces a set of higher-level, "virtual" knobs for the sole purpose of allowing users to reason and manage their quality expectations. The key here is that now users can fine-tune the quality on the level of sub-metrics instead of concrete knob settings and their low-level quality notions.

To support user preferences among sub-metrics, Rapids developers can selectively expose these metrics as virtual knobs to users. Each virtual knob corresponds to a specific sub-metric. The idea of virtual knobs is similar to an interface between the users and the KDG. Unlike the configurable knobs in KDG, virtual knobs do not have quality/cost weights. Users tune these knobs to express relative preferences among sub-metrics. In the FaceDetection example, users can tune the two virtual knobs, "precision" and "recall" and the system can automatically update the quality model accordingly.

4.3 Problem Specification

As described in the previous section, the KDG structure is used to determine whether a certain configuration can be chosen. Weights form the basis of the cost/quality models. Virtual knobs provide a user-level interface to customize the quality model. There are four key problems:

1. KDG Weights Derivation: How to derive the weights that define the models.
2. Effective Training: How to make model construction efficient.
3. Finalizing KDG: How virtual knobs influence the optimal quality configuration.
4. Runtime Optimization: How to calculate the optimal configuration to maximize quality given a weighted KDG and a cost budget.

The solution to the third problem is presented in Section 4.2.3. The solutions to the other problems are discussed below.

4.3.1 KDG Weight Derivation

Table 4.2: Symbols Used in Problem Definition

Symbol	Description
K_i	Knob i
N_i^j	Node j in K_i (Discrete)
S_i^j	Segment j in K_i (Continuous)
v_i	Knob i 's value
$v(N_i^j)$	Discrete Node j 's numerical value
min_i, max_i	Boundary of K_i (Continuous)
min_i^j, max_i^j	Boundary of S_i^j
n	Size of training set
$\hat{y}_k, y_k$	Prediction and real measurement for Configuration k
C_i^j	Contribution (weight) of discrete N_i^j
a_i^j, c_i^j	Weight function params for S_i^j
$\alpha_{[m][n]}$, $\beta_{[m][n]}$, $\gamma_{[m][n]}$	Correlation function params between K_m, K_n
$CC_{x(i)}$	Contribution of K_i, x = {Dis, Cont}
$CC_{corr(m,n)}$	Correlated contribution of K_m and K_n
p_i	User provided priority for K_i
ϵ	error threshold

Rapids assumes that the application is performing a long running task which can be partitioned into multiple work units. KDG models the expected cost for finishing a single work unit. At any given moment during runtime, the cost model will be used as a hard

metric to predict the cost consumption required to finish all the remaining work units. The quality model is used to rank all configurations.

The weights are extracted through a value propagation strategy from training. Suppose the application is trained through a number of configurations each with the same training input. Rapids records the observed cost or quality y for each configuration k. The weight value or function parameters of the nodes can be calculated by solving the problem on Line 4.9. The objective is to minimize the error between the predicted value and the observed value. The prediction is composed of three components, (a) the contribution of discrete nodes, (b) continuous nodes, and (c) the correlation between different knobs. I use the same model for both quality and cost. To simplify the discussion, I will only present the process of deriving the cost weights.

$$Minimize : \sum_{k=1}^{n} (y_k - \hat{y_k})^2 \tag{4.9}$$

$$\forall k : \hat{y}_k = \sum_i (CC_{dis(i)} + CC_{cont(i)}) + \sum_{i,j} CC_{corr(i,j)} \tag{4.10}$$

Discrete Nodes: In each configuration, only those nodes being "selected" will contribute to the overall cost. The total contribution of discrete nodes can be formulated in a specific configuration as shown in Line 4.11, where N_i^j = 1 if the node is selected in the configuration and 0 otherwise.

$$CC_{dis(i)} = \sum N_i^j * C_i^j \tag{4.11}$$

Continuous Nodes: Unlike Discrete nodes, the cost contribution of a Continuous node is represented by a function. I use a piece-wise-linear approach to represent these functions. To demonstrate this method, consider a set of configurations with m values (excluding the upper and lower bound) in N_i. The range of K_i can be partitioned to $m + 1$ segments as

$S_{i,1}, S_{i,2}, ...S_{i,m+1}$. A linear function is associated with each segment. Given a configuration with the knob value, I first determine the segment that this value falls into. Only one segment will be selected, i.e., only one of the S_i^j can be set to 1 in any K_i. The contribution for continuous nodes is formulated as shown in Line 4.12, where v_i is the value of node each configuration:

$$CC_{cont(i)} = \sum_j S_i^j * (a_i^j * v_i + c_i^j) \tag{4.12}$$

Correlation: I use a quadratic term to represent the contribution of each correlated pair of knobs (m,n). The value v for discrete nodes shall be provided from the developer in case each node in a knob is not numerical. Equation 4.13 shows the formulation of the correlated contribution.

$$CC_{corr(m,n)} = \alpha_{[m][n]} * v_m^2 + \beta_{[m][n]} * v_m * v_n + \gamma_{[m][n]} * v_n^2 \tag{4.13}$$

4.3.2 Effective Training

The size of the configuration space is exponential in the number of nodes for discrete knobs and number of segments for continuous knobs. In Section 4.3.1 above, I describe how to derive a discretized, exhaustive configuration space together with accurate cost and quality models represented as weights of the KDG. When porting an application to a new target hardware/software architecture, the model construction process discussed in Section 4.3.1 needs to be performed again for the new platform. This can be rather expensive. One reason this cost was ignored in all previous works is that those works do not consider the distinction between the developers and the users. In other words, the model is trained and will be executed on the same machine. To some extent, this prevents approximation techniques from being adopted in areas where the cost model is highly device-dependent, e.g., mobile applications. However, it is not practical to train models for all possible devices.

Therefore, an effective retraining process that can be executed on the users' end would be ideal.

The selection of the training set impacts the training cost (time, energy) to construct the model. Intuitively, a larger training set yields better a model at a greater cost. To significantly reduce the cost of retraining the models, Rapids introduces the notion of a *Representative Set* (RS), a subset of configurations that is sufficient to accurately reconstruct the entire cost/quality model at the potential cost of a slight accuracy loss.

Definition: RS with error threshold 'ϵ': A subset of all configurations that allows the construction of a cost model with an average prediction error $\leq \epsilon$.

During the construction of the full models, the computation of the representative set is also performed offline on the development platform. The RS computation is based on the configuration space of the model construction and its measured observations for each configuration. This is considered the ground truth. Rapids implements two different RS construction methods. Experiments show that these two strategies are both effective in significantly reducing the training set sizes for the benchmark applications. Selection-based RS takes longer to construct but usually has smaller size. Developers can choose which one to use according to the application and the accuracy of both strategies.

Partition-based RS: For each knob, Rapids first considers only its highest and lowest settings. RS is initialized with these configurations. Rapids evaluates the prediction accuracy of the model constructed from these configurations against the ground truth by solving the problem described in Equation 4.9. If the developer defined error bound ϵ is not satisfied, Rapids partitions each knob by a factor of 2, i.e. adding one more setting in the middle of two selected settings per knob. This process iterates until the error bound ϵ is satisfied or a pre-defined maximum partition level is reached.

Selection-based RS: Rapids initializes RS with only two configurations, the most and least expensive configuration. Subsequently, Rapids iterates through all unselected

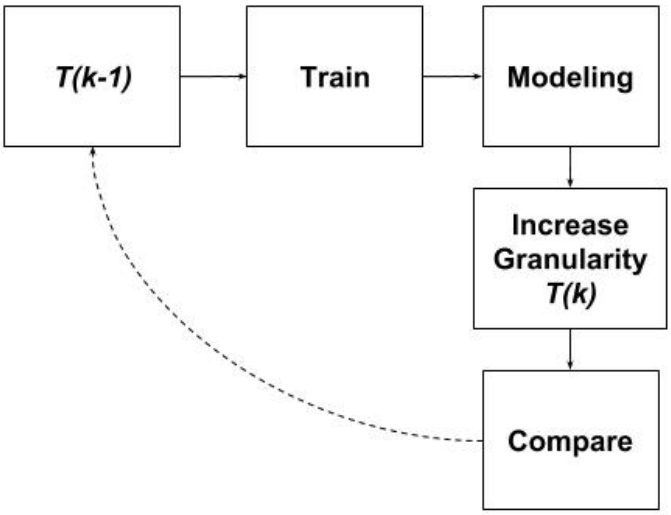


Figure 4.4: Partition-based RS Construction

configurations and constructs the model using that configuration and the current RS. The one configuration that yields the highest prediction accuracy will be added to the RS. The termination condition is again falling within error threshold.

4.3.3 Runtime Optimization Problem Formulation

Given a fully weighted KDG and a user-provided budget at application execution time, Rapids computes the optimal solution by solving a MIQCP [37] problem as specified by Equation 4.14. The problem formulation consists of an objective function and several classes of constraints to enforce the validity of the computed solution. The goal is to find the configuration that delivers the highest quality application outcome within the cost budget while satisfying all knob attributes and edge dependencies. For example, any solution has to respect knob dependencies, can only select a single setting per knob, and has to respect the user provided resource budget. The last line shows the optional constraints where developers can enforce specific knob(s) to be selected or not selected in the solution.

$$
\begin{aligned}
Maximize: & \sum_i (CC^q_{dis(i)} + CC^q_{cont(i)}) + \sum_{m,n} CC^q_{corr(m,n)} \\
s.t. & \sum_i (CC^c_{dis(i)} + CC^c_{cont(i)}) + \sum_{m,n} CC^c_{corr(m,n)} \leq B \\
& \forall AND(s_o -> s_i),\ s_i - s_o \leq 0\ //edges \\
& \forall OR(so_1, so_2, ...so_n -> si),\ si - \sum_i so_i \leq 0 \\
& \forall N_i^j, N_i^j = 1 \rightarrow v_i = v(N_i^j)\ //node \\
& \forall S_i^j, S_i^j = 1 \rightarrow min_{S_i^j} \leq v_i \leq max_{S_i^j}\ //segment \\
& \forall i, j, \sum_j S_i^j \leq 1\ //single\ node\ per\ knob \\
& \forall i, j, \sum_j N_i^j \leq 1\ //single\ seg\ per\ knob \\
& * \forall i^*, N_{i^*} = 1|0\ //optional\ PRESET\ knob
\end{aligned}
\tag{4.14}
$$

where all CC^q's are calculated with the same approach as cost but with measured overall quality with user-provided priorities. The second line ensures that the overall cost does not exceed the provided budget B. The third and forth lines show constraints for AND and OR edges. The fifth line ensures that knob values are within their segment ranges. The next constraint requires that only one segment or node can be chosen within each knob. Lastly, developers can optionally provide extra constraints to force particular nodes to be selected, or have specific values. This is useful especially when developers need to turn on or turn off some features in the application (e.g., turn off GPS) without changing the model. Rapids solves the optimization problem using the off-the-shelf solver *gurobi* [109].

Extra constraint in Weight Derivation: The correlated contribution derived from the model construction may make the problem shown in Equation 4.14, which makes it impossible to be solved efficiently. I tackle this problem by adding additional constraints 4.15 that make the matrix-Q [109] PSD (Positive Semi Definite) facing the problem to be con-

Table 4.3: Search Space Pruning and Specification Effort, RS Calculated with Error Bound ϵ = 5%

Application	Knob(#)	Conf	Discrete(#)	Spec (#loc)	KDG(#)	RS_P(#)	RS_S(#)
Swaptions	1	10	(10^{+})	2	-	2 (20%)	2 (20%)
Bodytrack	2	50	$(5\times10^{+})$	5	28 (56%)	15 (30.0%)	12 (24.0%)
Ferret	3	700	$(7\times10^{+}\times10^{+})$	12	276 (39.4%)	14 (2.0%)	4 (0.6%)
FaceDetection	3	90	$(3\times3\times10^{+})$	4	-	54 (30%)	18 (20%)
SVM	3	250	$(5\times5\times10^{+})$	4	-	8 (3.2%)	4 (1.6%)
NN	3	250	$(5\times5\times10^{+})$	4	-	8 (3.2%)	4 (1.6%)
NavApp	4	180	$(10^{+}\times3\times3\times2)$	7	40 (22.2%)	12 (6.7%)	6 (3.8%)
VideoApp	4	320	$(10^{+}\times4\times4\times2)$	11	24 (7.5%)	12 (3.75%)	4 (1.25%)

vex. Doing so will have certain impact on the accuracy of the model, however the problem cannot be solved by the tool I use if the constraint is not satisfied. If Gurobi [109] or other tools can support solving non-convex models in the future, this constraint can be removed.

$$\beta^{2}_{[m][n]} <= 4\alpha_{[m][n]} * \gamma_{[m][n]} \tag{4.15}$$

4.4 Key Results

I present some of the key evaluation results for Rapids in terms of space reduction, and the output quality. Detailed evaluation on other aspects, e.g., model accuracy, overhead, control sensitivity, etc, can be found on Section 7.1.

4.4.1 Space Reduction

One main benefit of allowing developers to express inter-knob dependencies through edges in KDG is to tailor the configuration space by filtering out invalid configurations. The reduction in the size of the space could both lead to a reduced the training and online optimization time. To support such feature in Rapids, I minimize the extra effort required from the developers by automating most of the training process and providing useful API's for developers to integrate their applications into the system.

Pruned Search Space: Table 4.3 shows the space pruning and cost prediction error rate for all the applications. $Conf$ reports the number of all possible configurations where every

combination of knob setting is considered "valid." KDG reports the number of "valid" configurations after developers encode dependencies using #loc lines of specification code to customize the configuration space in the KDG using the number of lines for specification in column $Spec$. Developers should have a good understanding of their applications and can therefore provide meaningful dependencies among knobs.

RS_P and RS_S report the size of RS with an error threshold ϵ of 5%. To give a meaningful comparison for knobs with continuous settings, the settings of each continuous node are discretized by uniform sampling where the number of sampled data points is shown with a plus mark in the third column labelled $Conf\ Discrete$.

4.4.2 Output Quality

It is impractical to compute the optimal quality configuration for each application under different budgets due to the enormous size of the configuration space. However, the relative quality of the selected configurations can be assessed through comparison of different sampling strategies. The strategies used for comparison purpose are described below:

1. **FULL**: All configurations, no developer encoded dependencies (treated as oracle).
2. **KDG**: Configurations with dependencies.
3. **Rand-20**: A heuristic used in CALOREE [21] that constructs the model with 20 random configurations.
4. **RS_P(RAPIDS)**: Partition-based RS.
5. **RS_S(RAPIDS)**: Selection-based RS.

Here, for coverage purpose, I run each application with 10 different time/energy budgets. For each run i, $budget = min + i * (max - min)/10$, where i =1 to 10 and min/max are the minimum (cheapest setting) / maximum (default setting) time requirement. The results are shown in Figure 4.5. The "error bars" report the range of qualities under the

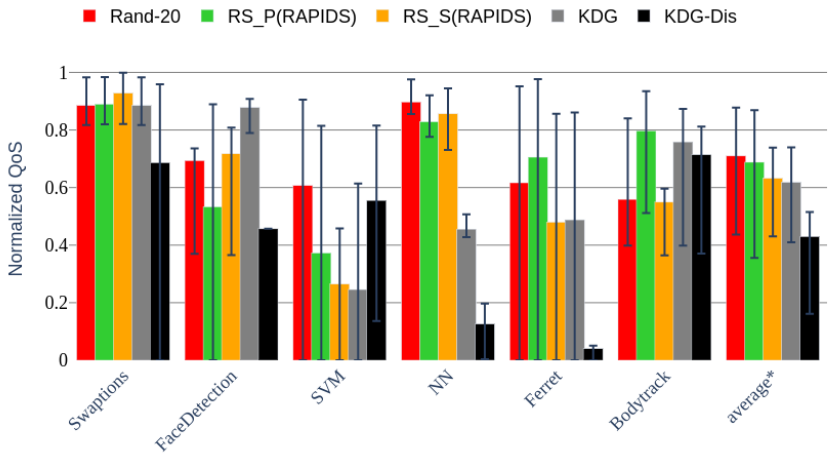


Figure 4.5: Normalized Quality over Different Budgets for Linux Applications. Bar height: Mean Quality; Error Line: Quality Range

different budgets, and the height of the solid bars show the average quality under all budgets. The qualities are normalized relative to the highest quality configuration under an unlimited budget. On average, **RS_P** and **RS_S** achieve 9.5% and 9.6% higher quality than **Rand-20**, respectively. The size of RS is smaller than 20 in all applications (fewer observations), and Rapids stops to take more observations if the cost model accuracy is achieved. Therefore, **Rand-20** achieves higher quality in some applications for having a better model constructed from more observations. Having way more observations is also not a good approach for over-fitting, e.g., for NN, **KDG** performs worse than both **Rand-20** or Representative Set.

Overall, developers can express insights on the application structure through KDG with a few lines of code, which results in a pruned search space over multiple magnitudes smaller. After performing the exhaustive training only once, Rapids computes an extremely compact training set, *Representative Set*, yielding fast re-training times and resulting in only a small QoS loss (on average <= 7.5%) compared to the oracle.

Table 4.4: User-Preferred Sub-Metrics Value Improvement

	Preferred Sub-Metrics and Improvement		
Ferret	coverage 3.2x	ranking 3.59x	
FaceDetection	precision 1.13x	recall 1x	
NavApp	brightness 1.45x	localization 1.5x	information 1.5x
VideoApp	brightness 1.29x	smoothness 1.26x	resolution 1.62x

4.4.3 User Preferred Sub-Metric Comparison

User provided preferences change application behavior when selecting the optimal solution for a budget. It is expected to see improvement on the user preferred sub-metrics when preference is provided. For the four applications with customizable quality, Table 4.4 reports the average improvement in different sub-metrics under the same budget when users tune preferences. The recall in FaceDetection does not change when preference increases because the optimal selection with the default quality metric produces the highest recall. Overall, improvements can be up to 3.2×, with 1.76× on average.

CHAPTER 5
RAPIDS IN MULTI-PROGRAMMING ENVIRONMENT

In this chapter, I introduce Rapids-M, an extension to Rapids that handles cross-application approximation management in a multi-programming environment.

5.1 Introduction

In a multi-programming scenario, n applications are executed (active) at the same time on a target platform. A user assigns a unique time budget to each application. Multiple concurrent executions of the same application are possible, each with their individual execution time budgets. The multi-application configuration selection problem determines a configuration for each active application such that (1) an application can finish within the user specified time budget, if such configurations exist, and (2) the overall quality across all active applications is as high as possible.

Direct Extension towards Multi-Programming Environment: The ability of reconfiguration makes Rapids capable of adapting the application to dynamic environments. In all of my experiments, Rapids manages to finish the mission even with limited resource availability. However, as described in Chapter-1, directly extending existing approaches, including Rapids, to multi-programming environments can suffer from multiple problems which do not exist in single-app scenario. Figure 5.1 shows the execution trace of running multiple applications together, each given enough budget to successfully finish the mission with default setting and managed by an individual Rapids controller. The dashed line with a cross at the end indicates that the execution fails during the middle of the application. That is, Rapids cannot find a feasible configuration to complete the mission. The failure is due to the lack of a global view of running applications, and the naive adaption method

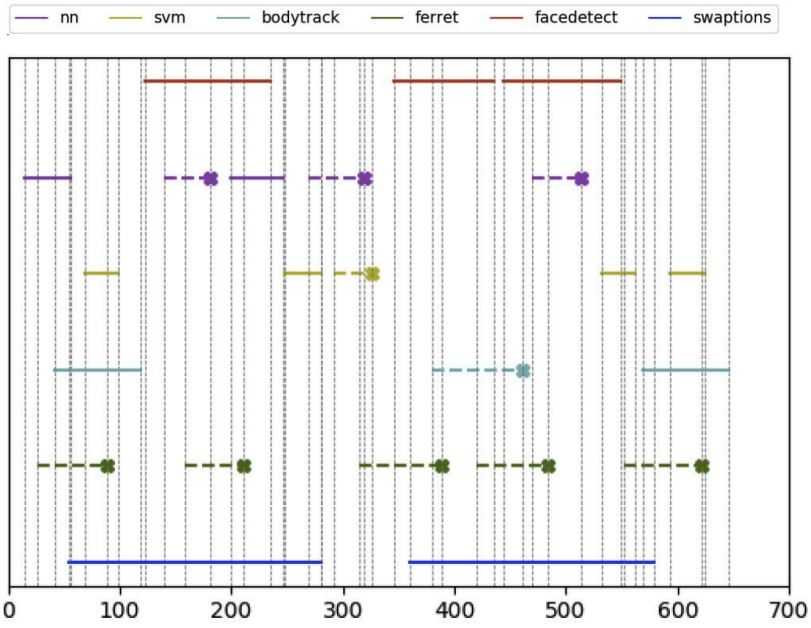


Figure 5.1: Execution Trace of Applying Rapids on Multi-Programming Environment

that simply treats other applications as environmental noise.

Aside from execution failure, there are more problems of directly extending any existing work to multi-programming environment including frequent reconfigurations, and low quality output. The next section will describe Rapids-M, designed to handle this situation.

Global Configuration Problem: At system-defined points in time, t_x, determine a global configuration vector [c_1, c_2, ... c_n] with one entry for each active application $1 \leq i \leq n$, where an entry is either a valid configuration c_i or $terminate$, such that
(1) $\sum_i^n Qual\_Metric_i(c_i)$ is maximized ($terminate$ entries are ignored), and
(2) for each active application i, its remaining work units at time t_x can be successfully executed within its remaining execution time budget under c_i, or the application is terminated.

Multi-programming environments are inherently unpredictable since applications may start at any point in time with different execution time budgets, thereby changing the available resources for each of the currently active applications, currently active application.

The focus of this work is to 1) model and quantify the interference of different active applications and their configurations, and 2) choose high quality configurations for each application so as to maximize overall global quality. If the multi-programming environment changes due to initiation or termination of applications, this approach will recompute the overall global configuration using the interference and prediction models.

Since all applications execute on the same hardware platform, they interfere with each other through resource sharing (e.g., memory hierarchy, CPUs / cores, buses / communication networks). Different configurations may have different resource footprints and different quality outcomes, making an optimal or close to optimal selection of configurations across all applications a significant challenge. In previous work, single application performance models are constructed by applying machine learning strategies on all or a subset of the application's configuration space. Treating a set of n applications as a single, meta application would allow these strategies to be applied to the multi-programming case. However, the resulting size of the combined configuration space is exponential in n, making this approach infeasible in practice. Moreover, the performance model is constructed based on the observations obtained from running the application under a stable environment. Online adaption is only designed to deal with input dependencies or runtime noise, but not the interference from other applications.

This chapter discusses the design and evaluation of a new local-global-local approach that allows a systematic exploration of the combined configuration search spaces of all active applications. It first reduces the space of the problem by clustering configurations (locally) in single applications into equivalence groups, called *buckets*. Buckets combine configurations according to their similar resource demands and performance slowdown characteristics – the two dimensions of the summary strategy for reducing exploration space sizes. Since the performance degradation of an application is due to resource availability on the machine, each bucket also comes with 1) a performance model that predicts the application slowdown given the system environment, and 2) the resource demand by configurations in

the bucket. Across applications (globally), a machine model is constructed to predict the overall system workload for any bucket combinations from active applications, including the option of not selecting a bucket for an application. These combinations are exhaustively evaluated, resulting in the optimal bucket combination with the highest overall global quality. This bucket combination has to be *feasible*, i.e., each bucket in the combination has to contain at least one configuration that satisfies the execution time constraint (budget) of the associated application as provided by the user. Finally, a (local) selection within each bucket will be performed to allow individual applications to react to minor platform uncertainties and input dependencies that can be handled without a global reconfiguration, thereby avoiding reconfiguration overhead.

Rapids-M (**RAPIDS** for **M**ultiprogramming) is a prototype implementation of the new local-global-local approach to configuration management across multiple applications. Experimental results show that application configurations can be partitioned into a small number of buckets allowing the system to produce global configurations of high quality that are able to successfully execute applications under user provided execution time budgets, when possible. Machine learning models are used to model the platform-specific configuration interactions on the level of target system footprints (m-model), and to determine application specific models for configuration slow-downs in response to varying overall system loads (p-model). Both, m-model and p-models allow the prediction of system and configuration behaviors, i.e., assessing the mutual interference and benefits of configuration selections.

Experimental results on six applications and different execution traces show the effectiveness of Rapids-M and its implementation. Runtime overhead is defined as the additional execution time needed to solve the global selection problem, the local problem, and any resulting dynamic reconfigurations. Training times for the m-model and p-models are also reported, capturing Rapids-M's offline overhead. Compared to existing approaches in which each application adapts itself individually, on a 4-core machine, Rapids-M achieves

3.4% higher success rate when the system is not busy ($\leq$4 active apps), and 22.75% higher when busy. This translates to 2.6% (not busy) and 52.99% (busy) higher overall output quality. Furthermore, Rapids-M achieves such improvement with an average of 40% fewer performed reconfigurations.

5.2 Rapids-M Framework Overview

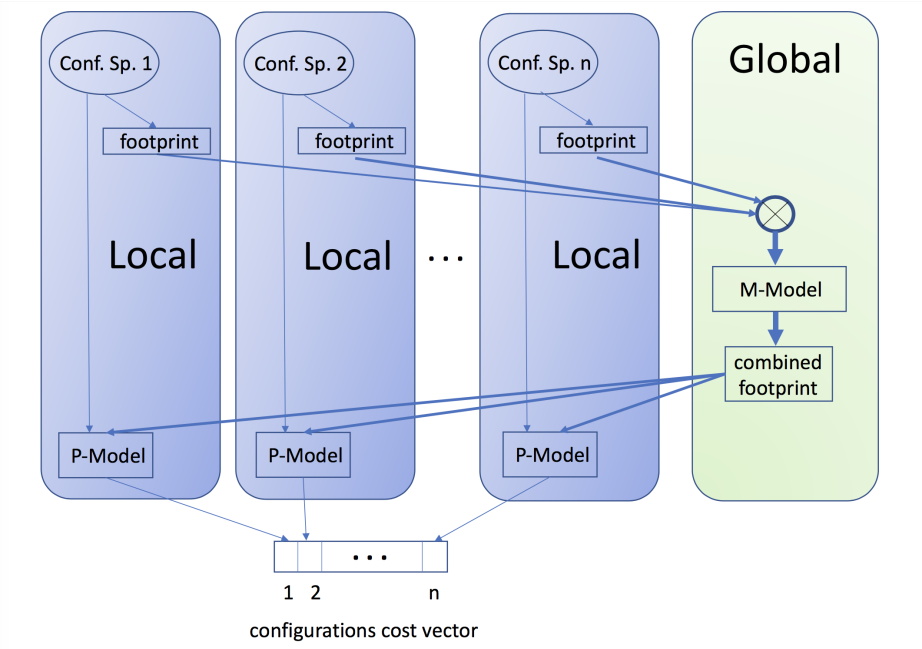


Figure 5.2: Approach Overview

Rapids-M uses the "standard" notion of a configuration as defined in Rapids, and also by most existing adaptive configuration management approaches for single applications [46, 20, 21]. Rapids-M is a framework that manages the configurations of concurrently active applications with the goal of choosing individual application configurations such that all individual resource constraints are satisfied while maximizing the overall, combined quality of all active applications.

A single application's configuration space is the Cartesian product of all knob value ranges, where each value range is discrete, i.e., each knob has finitely many value settings. Assuming that a single application has k knobs with discrete ranges of m values, the re-

sulting configuration space is $O(m^k)$. Existing approaches constructs performance models to determine the best configuration for a given optimization objective at runtime. Using the straight-forward approach of treating all active applications as a single, meta application, the size of the resulting configuration space is $O((m^k)^n)$ for n applications which can be multiple orders of magnitude bigger and therefore infeasible to explore. However, configuration space exploration is necessary because of the mutual interdependence of the individual application configurations due to target system resource.To the best of my knowledge, the Rapids-M framework is the first to address configuration management of multiple active applications.

One main design feature of Rapids-M is to model cross-application configuration interference not at the configuration space level, but on the system footprints associated with each configuration. Figure 5.2 shows the general idea of such abstraction. A *system footprint* is a vector of hardware counters that characterize the use of different system resources by an executing application. Footprints can be used to represent a configuration's resource demands, and also resource demands of groups of active applications. Since resource sharing is based on target machine resource contention, system footprints are the right abstraction to represent the impact of such sharing. This strategy has two main advantages: (1) many configurations of an application may have the same system footprint, and (2) the impact of other applications and their configurations on the performance of a given application's configuration can be modeled based on the combined system footprint of these other application configurations. In other words, for the assessment of a configuration's performance modeled as an expected slow-down, only the combined system footprint/-workload of other applications is relevant, and not their particular configuration selections. The resulting summary information is the key to allowing effective configuration space exploration management across multiple applications. This summary information is computed and exploited with a local offline model training phase, followed by an online global configuration and online local configuration selection phase.

In the offline phase, single application configurations are clustered into groups with similar system footprints and similar slowdown behavior in response to overall system workloads. Such groups of configurations are referred to as *buckets*. The individual application configuration spaces are exhaustively explored and each configuration's system footprint is recorded together with its cost and quality under different system workloads. This data is used to train the p-model that captures the slowdown for each configuration in response to different system workloads. The interaction of different workloads and configuration footprints is captured by the m-model which is trained on data obtained by measuring runtime properties of configuration system footprints executing with randomly generated, "stresser" workloads.

At runtime, the global optimization manager uses the constructed p-models and m-model to assess the impact of global events (e.g., start/exit of applications) on resource availability, and to select the set of local configurations that maximize the overall quality under the changed resource availability. A globally optimal bucket along with the predicted slowdown for all configurations in this bucket is assigned to each application.

The local controller relies on optimization strategies used in single-application scenario, e.g., Rapids, for configuration selection with the predicted slowdown. However, the local configuration managers can only select configurations within the bucket assigned by the global manager. Therefore, I will concentrate the discussion on Rapids-M's offline component and the online global configuration manager.

5.3 Rapids-M Offline Phase

The three key models and abstractions that are generated in Rapids-M's offline phase are the target system's m-model, p-models and their associated buckets.

5.3.1 Resource Usage Prediction: M

The performance of an application can significantly degrade when the overall system resource utilization is high. For example, Bodytrack suffers from as high as 15X slow-down in terms of execution time. Predicting the overall environment is crucial before estimating the performance degradation of an application under multi-programming environment. More specifically, how would the system workload change when a new application starts if there are already other active apps running?

Rapids-M trains the m-model with a Linux "stress" tool [110] that introduces arbitrary workloads to the system, including I/O, CPU utilization, harddisk access, and etc. The system footprint is measured by Intel's performance counter monitor (PCM) [111]. Vector V represents the system footprint where each entry corresponds to a particular system metric considered by Rapids-M, for example, [*FREQ, Mem-READ, Mem-WRITE, IPC, L2HIT, L3HIT, L2MISS, L3MISS*]. For each training data point, Rapids-M collects two running instances, which could be an instance of "stress" tool or an application, each with different workloads. It records the system footprint when each instance executes in isolation (V_1, V_2) and together ($V_{1,2}$). Rapids-M constructs a separate regression model for each feature in the vector based on the data collected. Suppose m features are collected in each vector. Equation 5.1 shows the model construction for the k-th feature in $V_{1,2}$. The first m columns of X are list of V_1s and the last k columns are list of V_2s, i.e., each row of input is $[V_1, V_2]$ and its corresponding output is $V_{1,2}$. The goal is to locate β that minimizes the error ϵ.

$$V_{1,2}[k] = X\beta + \epsilon \tag{5.1}$$

The m-model is a collection of such models each predicts a particular feature in $V_{1,2}$. When running n applications together, the overall system footprint is estimated by applying M iteratively:

$$V = M \otimes ...(M \otimes (M \otimes (V_1, V_2), V_3)...V_n) \tag{5.2}$$

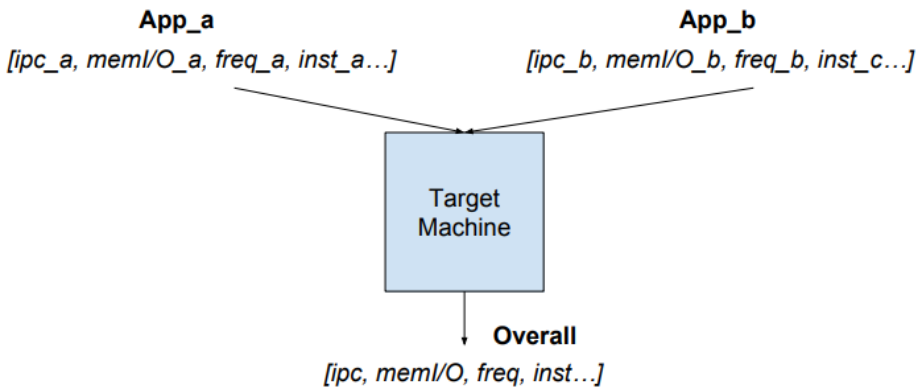


Figure 5.3: m-model Construction

5.3.2 Performance Prediction: P

Rapids-M predicts the performance degradation for each application under different environment by a performance regression model p-model. p-model is trained by collecting the application slow-down under different environments. During the training, Rapids-M first records the execution time for an application under configuration c when running alone. It then runs the configuration under a number of different environments created by running another "stresser". This stresser could be a another application or an instance of "stress" tool. Under each environment, Rapids-M records the overall system footprint V and the execution slow-down (α). The regression model is to minimize the error between prediction $\bar{\alpha}$ and observed slow-down α. Note that a unique p-model is constructed for each bucket. During construction, configurations that are not part of the bucket are not considered. Figure 5.4 shows the construction of p-model.

Different models may be better fit for a particular footprint feature in m-model prediction or the slowdown in p-model, and they may not be relevant to all features in the vector. The solution to the latter is discussed in ESP [71] by first filtering out insignificant features, then training a higher-order model with the remaining features. However, this approach has the drawbacks that is uses a single, linear model approach for all applications, and does not distinguish between different configurations. Also, to support the slowdown prediction for

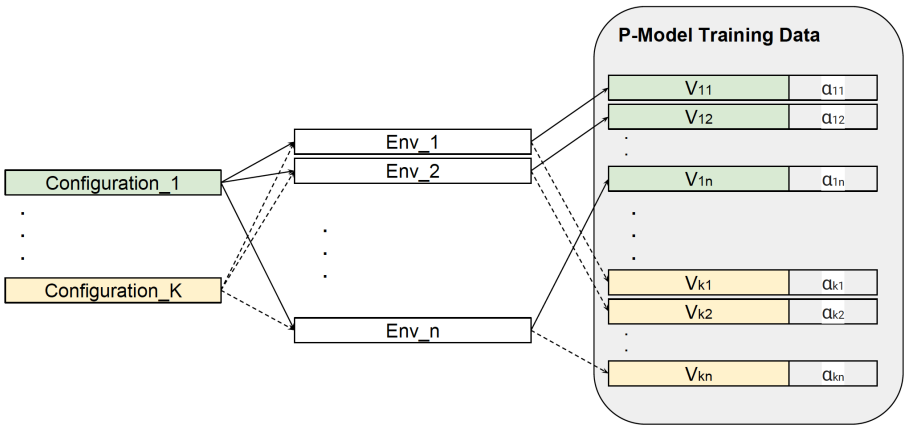


Figure 5.4: p-model Construction

up to k applications, ESP needs to collect the training data by actually running k applications together. In contrast, the p-models and m-model are trained individually. Rapids-M also maintains a model-pool with multiple available models, including Regular Linear-Regression (LR) regression [102], Elastic-Net (EN) regression with cross-validation [102], Lasso (LS) regression with cross-validation [102], Bayesian-Ridge (BR) regression [102], and a fully connected Two-Layer-Neural-Network with 50 neurons and 'relu' activation function (NN) [112]. When constructing the models, Rapids-M trains all models in the pool and picks the one with highest accuracy. For each of these candidate models except NN, Rapids-M first iteratively selects the top-K important features in the footprint. Then, it decides whether to use a higher order regression by comparing the models from linear and higher order features.

5.3.3 Bucket Determination

Observation shows that configurations in an application can behave differently in both introducing workload and performance degradation under a particular environment. However, these configurations can still be clustered into a limited number of groups. Configurations within each group share similar behavior: introduce similar workload to the system and suffer from similar slow-down given a particular environment.

Figure 5.5 shows the dendrogram of hierarchically clustered configurations in one of

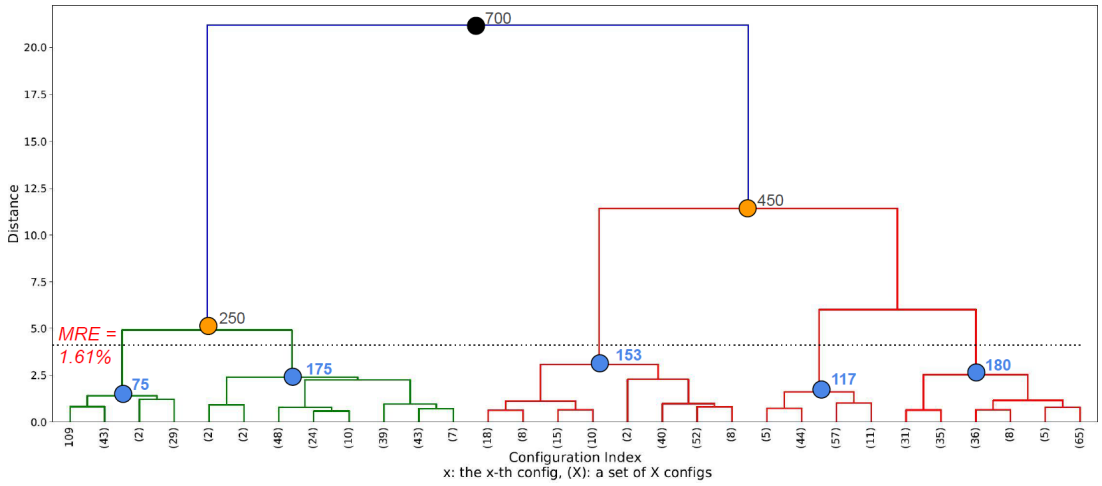


Figure 5.5: Application Configuration Affects Environment
Vertical Bar Shorter $\Longrightarrow$ Higher Inner-Cluster Similarity; Black Bold Number: Number of configs in a cluster; Red Percentage Number: MRE of performance model prediction

the benchmark applications (Ferret) by system footprints. The X-axis shows the index of all 700 configurations. For simplicity, I truncate the indexes of N configurations on the X-axis and represent them by (N). Branches in the graph show the result of clustering. For example, all 700 configurations are clustered together at the "root" of the tree (Black Dot). When moving downward, two "sub-tree"s (clusters) are formed with size 250 and 450 (Yellow Dots). The Y-axis shows the average euclidean distance within a group. The height of each "sub-tree" reveals the closeness of all configurations in the cluster. Configurations are clustered into buckets. Configurations within each bucket have higher similarity in terms of system footprints when moving down the dendrogram.

The number of buckets can be determined by the distance threshold, i.e., the maximum euclidean distance within a cluster. In Figure 5.5, if the threshold is 4, all configurations can be clustered into 5 bucket (dashed line). For each bucket, a performance model is constructed to capture the relationship between slow-down and the execution environment. In Figure 5.5, the red number on the left represents the average prediction error (Mean Relative Error) for all 5 buckets. Using buckets reduces the configuration search space size from $O((m^k)^n)$ to $O(b^n)$ where b is the number of configuration buckets per application.

The bucket design has to satisfy two aspects of similarity: 1) switching between config-

urations belonging to the same bucket will not introduce significant impact to other applications, i.e., lower in the dendrogram, and 2) all configurations within a bucket suffer from the same performance degradation under a particular environment, i.e., lower MRE. The number of buckets could range from 1 (all configuration have a similar system footprint), to N (all configurations have a unique footprint). Having more buckets result in higher similarity for included configurations, while increasing the problem size. On the other hand, fewer buckets could hurt the accuracy of m-model and the p-model. I implement a variant of Hierarchical Clustering[113] in Rapids-M.

Input: $all\_configs$, T_{dis}, T_{acc}
Result: buckets
buckets = [all_configs];
// initial configuration partition
buckets = h_cluster(buckets, criterion='dis', T_{dis})
err,worst_id = evaluate(buckets);
while *err* $\geq T_{acc}$ **do**
    // refine clustering of the worst bucket
    tmp_buckets = h_cluster(buckets[worst_id], criterion='number',2);
    buckets[worst_id] = tmp_buckets;
    err, worst_id = validate(buckets)
end
return buckets;

Algorithm 1: Bucket Determination

Algorithm 1 describes the approach. I first run a standard Hierarchical Clustering procedure to generate the initial buckets, satisfying the distance threshold T_{dis}. Then, I evaluate each bucket by training the p-model with 70% of its observations, then validating with the remaining 30%. If the p-model accuracy T_{acc} theshold is not satisfied, I iteratively refine the bucket that have the worst p-model accuracy until the threshold is satisfied. In Rapids-M, I use T_{dis}=4 and T_{acc}=6%. The choice of these numbers was based on the experiences with the sample applications.

5.4 Rapids-M Online Configuration Manager

The goal of Rapids-M is to find a global optimal configuration for all active applications. By grouping configurations into buckets, the size of the search space is reduced from all combinations of application configurations to all combinations of application buckets. This strategy can reduce the required search space by multiple orders of magnitude. The global optimization problem is solved in two steps: 1) finding the globally optimal bucket for each application, and 2) finding the optimal configuration within each bucket. The global manager provides each local manager with a particular globally optimal configuration, together with all feasible configurations in the bucket to which the optimal configuration belongs. The latter information allows the local manager to change configurations if needed without impacting configuration choices in other applications. Algorithm 2 describes the runtime algorithm to compute the optimal global configuration.

Invariant of Algorithm 2: If the predicted slow-downs (p-models) and the predicted configuration interactions (m-model) are correct (accurate within an error threshold), the selected local configurations are globally optimal under the user defined time constraints and priority weights assuming all active applications can successfully finish the remainder of their execution. The global manager is invoked each time a new application becomes active, an active application terminates, or an active application requires a new bucket assignment. Global reconfiguration may also be triggered every fixed time interval, or may be requested on demand, i.e., a local controller reports that no feasible configuration in the assigned bucket $fCgs$ set meets the application's runtime constraint due to system uncertainties.

Input: Set of n applications with user specified execution time constraints T_i, for $1 \leq i \leq n$. For each application, set of buckets with associated p-models, bucket footprints, and cost / quality models. A target machine m-model.

Result: Termination or Bucket selection for each application i, b_i.

foreach *bucket combinations [*b_1, $b_2 \ldots b_n$*]* **do**

determine global footprint (gfp) using m-model:

gfp = $\bigotimes_M$ (fp(b_1), fp(b_2), $\ldots$ fp(b_n))

determine vector of slow-down factors (sdf) for each bucket using the bucket-specific p-models p_b

sdf = [$p_{b_1}(gfp)$, $p_{b_2}(gfp)$, $\ldots$ $p_{b_n}(gfp)$]

foreach *bucket* b_i **do**

determine set of feasible configurations (fCgs) that satisfy the execution time constraint T_i^{rem} for the remainder of the execution:

$fCgs(b_i)$ =

$\{c \in b_i \mid sdf[i] * cost(c) * \#remaining\_workunits(c) \leq T_i^{rem} \}$

if $fCgs == \emptyset$ **then**

reject bucket combination and break

end

compute the maximal quality configuration $mQCg(b_i)$ of all feasible configurations of b_i:

$mQCg(b_i) = c$ with $qual(c) \geq qual(c_x)$ for all $c_x \in fCgs(b_i))$

end

compute the global quality GQ([b_1, $b_2 \ldots b_n$]) as a

$$\sum_i^n qual(mQCg(b_i))$$

end

Select valid (non rejected) combination of buckets with maximal GQ: [b_1^{maxQ}, $b_2^{maxQ} \ldots b_n^{maxQ}$]

Return to each application i its bucket b_i^{maxQ} and feasible configurations $fCgs(b_i^{maxQ0})$. If no bucket assignment for an application, select "terminate".

Algorithm 2: Global Configuration Manager

5.5 Key Results

In this section, I report some of the key evaluation results for Rapids-M in terms of reducing reconfiguration frequency, improving output quality, success rate, and lowering the rejection rate. Detailed evaluation on other aspects, e.g., model accuracy, overhead, selection optimality, etc, can be found in Section 7.1.

5.5.1 Strategies Used for Comparison

Since to the best of my knowledge Rapids-M is the first system that performs configuration management across sets of applications, the alternative strategies are constructed as extensions to existing single application approaches or multi-application strategies for other problems (e.g., ESP for scheduling). Determining the optimal solution requires exhaustive physical measurements which is not possible due to the size of the configuration space. I define the following alternative configuration selection strategies. The strategies differ in what information they use to determine each applications configuration.

ContextOblivious (CO): Applications ignore the fact that they share resources with others applications. Configuration decisions are made based on the initial assigned budget and the difference between the assumed and actual resource availability is treated as "environmental noise". At times of reconfiguration, the available remaining budget is adjusted by this observed noise.

AwareShare (AS): This strategy is partially inspired by ESP[114]. It first measures the overall environment of different combinations of benchmark applications under their default configurations (ignoring buckets). Then the environment is used together with Rapids-M's p-models to select optimal configurations for each application. This strategy is used only as an oracle in static evaluation where the combination of running applications is known before execution and the environment can be measured.

Rapids-M **(RM):** This strategy utilizes the full power of Rapids-M: configurations are

clustered into buckets, the m-model predicts the system environment, and p-models predict slowdown.

Rapids-M with Rush-To-End (RM-Rush): This strategy extends Rapids-M with a rush-to-end feature that artificially increases the predicted slowdown up to 1.5X when 60% of the work units are completed and the remaining budget is no more than 10% of predicted cost. This feature is designed as an insurance policy that tries to avoid failing an execution after most of the work has been done.

EqualShare (ES): Each application divides its assigned resource budget (execution time) by the number of concurrently active applications. This reduced budget is used to determine the application's configuration.

Always Low(LOW): This strategy always picks the lowest setting for all applications.

Table 5.1: Strategies Used for Comparison

	Utilizes m-model	Slowdown for N apps	Available in Evaluation
CO	-	1.0	Static / Dynamic
AS	-	P(measured)	Static
RM	✓	P(M())	Static /Dynamic
RM-Rush	✓	rush(P(M()))	Dynamic
ES	-	N	Static / Dynamic
LOW	-	-	Dynamic

Table 5.1 summarizes the differences between the strategies. The slowdown prediction aggressiveness increases going down the table. For each possible group of two to six sample applications and three different budget constraint levels (high, medium, low), I measured the overall combined quality of the applications.

5.5.2 Evaluation Metrics

I evaluate the performance of Rapids-M by dispatching different applications starting at random times with a given budget. Each application gets to reconfigure during the execu-

tion. I first generate a series of execution traces showing when to start which application by giving each application infinite budget such that all applications finish successfully with the highest setting producing the highest possible output quality. For each generated trace, I repeat the trace with shrinking budgets to force reconfiguration using different strategies. Different strategies are evaluated on four aspects:

- Rejection Rate: Failing to find a feasible configuration at the application start time. This could happen when the strategy over-predicts the slowdown.
- Success Rate: Finishing the execution within budget.
- Reconfiguration: Number of reconfiguration due to performance changing. More frequent reconfiguration is usually caused by the mis-match between the real execution time and the predicted cost. Since the same cost model is shared across all strategies, the performance difference translates to the accuracy of the slowdown prediction.
- Output Quality: Normalized overall output quality from 0 to 1 for applications that successfully finish. The quality achieved by the lowest setting is 0. A failed execution is penalized by a negative quality of -0.5. An application has to terminate successfully with a valid configuration in order to be considered in the overall quality. In other words, their is no partial quality notion if an application "dies".

It is important to note that even "bad" strategies for the global configuration case can do rather well for an entire execution since the prediction error can be somewhat compensated for at each reconfiguration point. However, the number of local reconfigurations are expected to significantly increase when using "bad" models.

5.5.3 Improvement on Overall Output Quality

To demonstrate the performance in different scenarios, I run the experiment with a threshold N such that the simulator will stop dispatching new applications when there are N

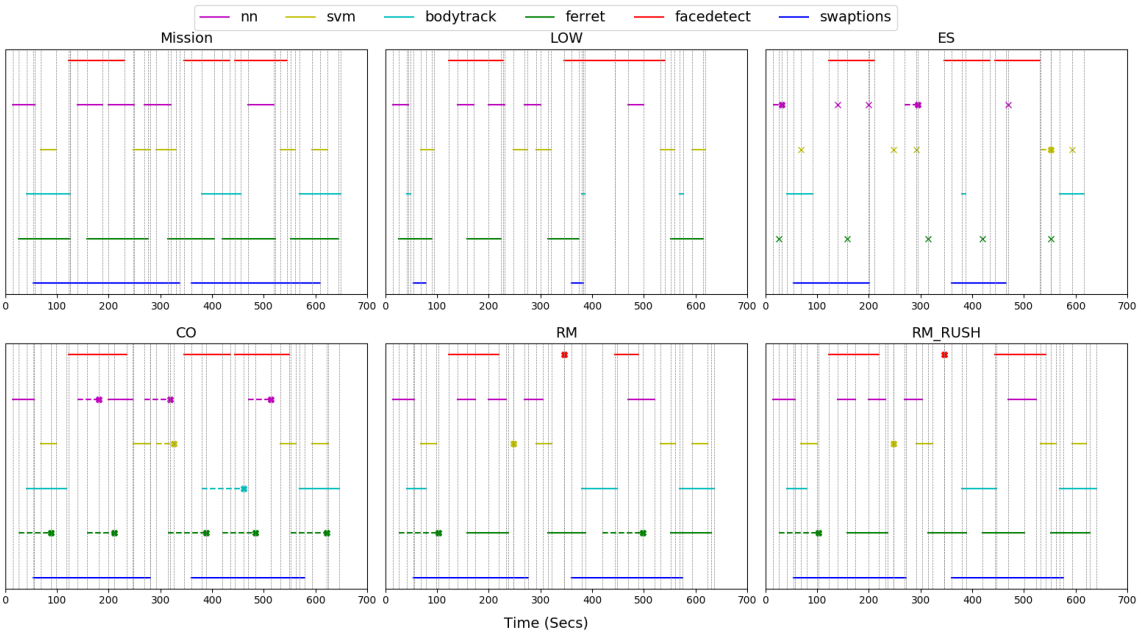


Figure 5.6: Sample 10-minute execution trace with up to 4 active applications using different strategies, budget scale=1.0

active applications. Figure 5.6 shows an example of a 10-minute-execution trace where N=4. As shown in the graph, most executions got rejected by ES because of the over estimation on slowdown. On the other hand, for CO, executions are more likely to fail during the middle of execution for under estimating the slowdown. Only 2 runs failed under Rapids-M, and these runs are "rescued" by adding the 'rush_to_end' strategy.

Figure 5.7 shows the results of 18 traces, each repeated with multiple budget settings using different strategies. To summarize, ES predicts the slowdown so aggressively that it rejects most executions and has 54.4% of Rapids-M's success rate. On a 4-core machine, Rapids-M achieves a 3.4% higher success rate than existing approaches (modeled by CO) when the system is not busy ($\leq$4 active apps), and 22.75% higher when busy. This translates to 2.6% and 52.99% higher output quality. Furthermore, Rapids-M achieves its improvement while reducing the number of reconfigurations to 40% of CO. RM_RUSH further improves the quality by an average of 1.6% higher by enabling more applications to finish.

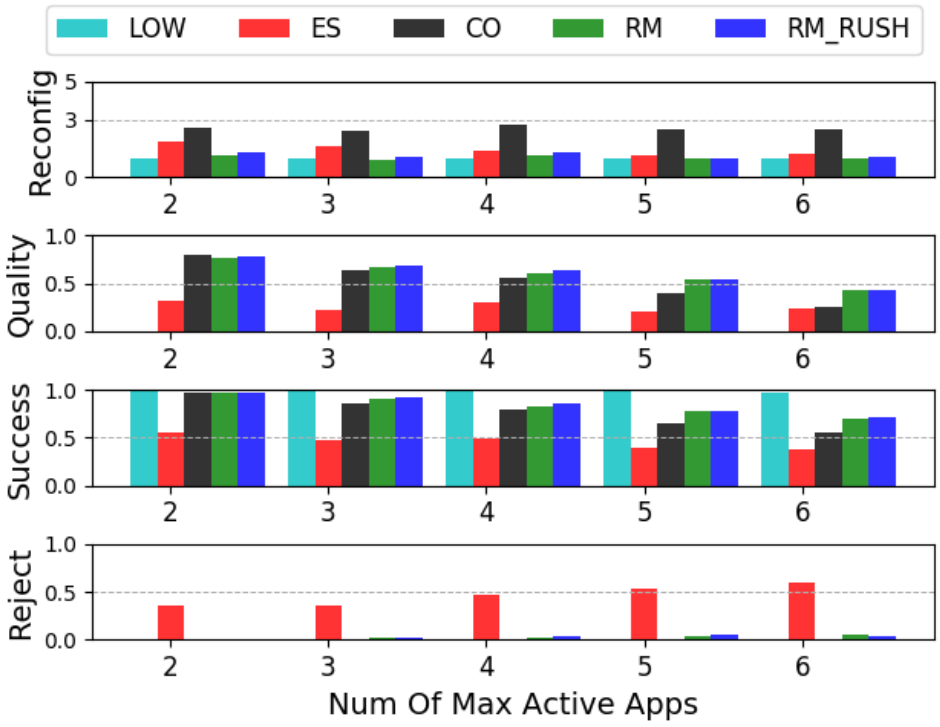


Figure 5.7: Dynamic Configuration Selection Comparison

CHAPTER 6

IMPLEMENTATION

In this chapter, I first introduce the main design feature of Rapids, including the development workflow and how application developers and application users can interact with the system. Rapids-M shares most of the infrastructure with Rapids, but with an extra layer of control logic plus a global server implementation.

6.1 The Rapids Framework

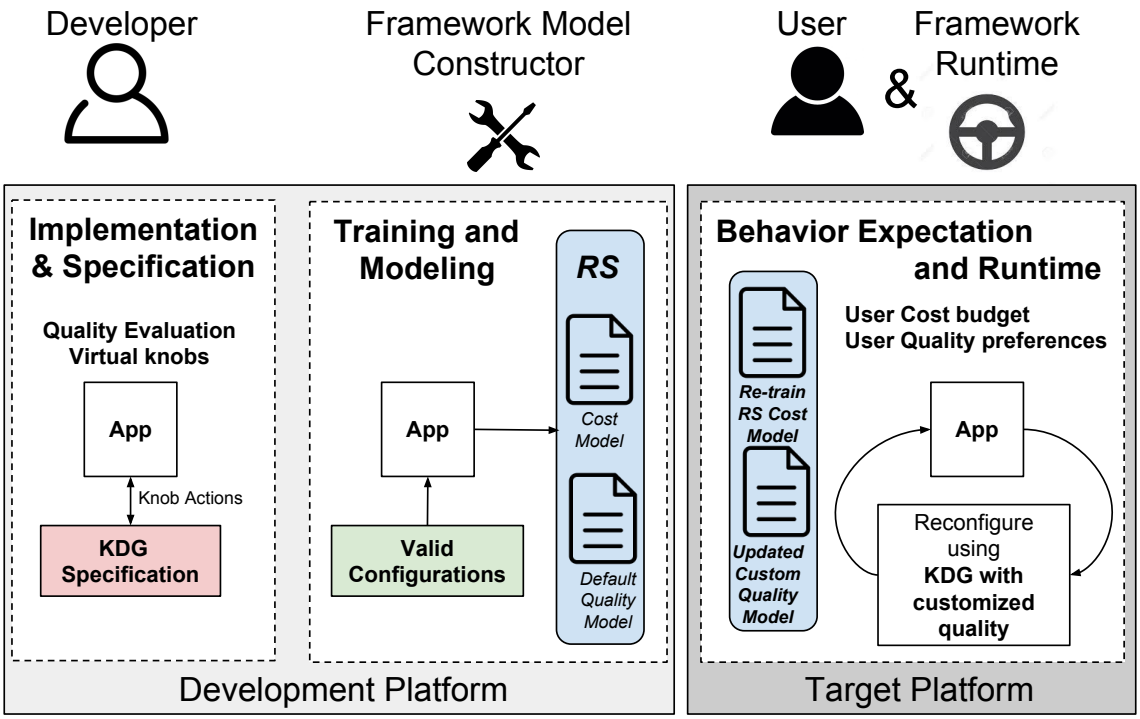


Figure 6.1: Rapids Overview

Rapids provides an end-to-end framework to write and execute reconfigurable applications. Rapids's work flow consists of three main phases: (1) application specification and implementation (done by the *Developer*), (2) automatic training and modeling (done by

$Knobs$:
$continuous\_knob\_name\ C\ [v_{min}, v_{max}]$;
...
$discrete\_knob\_name\ D\ \{v_1, v_2, ...v_n\}$;
...
$Dependencies$:
$sink\_knob\_name\{v\} < -source\_knob\_name\{v\}\ AND|OR\ source\_knob\_name[v_{min}, v_{max}]$;
...
$SubMetrics$:
$sub\_metric\_name_1, sub\_metric\_name_2, ...$

Figure 6.2: Developer: Structural KDG for NavApp

the *Model Constructor*), and (3) custom quality model specification and construction, budget specification, runtime monitoring, and reconfiguration (done by the *User & Runtime Framework*).

Developers' Effort: Rapids provides a profiling platform and a runtime library that allows application developers to communicate important application properties to the framework. To do the profiling, developers need to prepare:

- *KDG specification*: A file to specify insights including knob type, value range, and dependencies. Rapids uses the file to generate the KDG structure to represent the configuration space used to train cost and quality models. An example specification for the NavApp application is shown in Figure 6.2.

- *Evaluation Module*: A Python module that includes a) command and command-line arguments for application execution, b) the optional sub-metric (q_i in Eq 4.5) evaluation strategy, and c) an overall QoS function (F in Eq. 4.5). The execution framework is provided by extending a class from a base class $AppMethods$, included in the framework, and implementing at least the three functions shown in Figure 6.3.

- *Source code modifications*: In the application code, the developer inserts library calls as shown in Figure 6.4 to a) bind particular actions (e.g. change a variable's value) to knobs

```
1  from rapidlib_linux import AppMethods
2  Class MyAppMet(AppMethods):
3      def get_command(self, configs=None):
4          # return the execution cmd
5      def get_submetrics(self):
6          # return all submetric values
7      def get_final_qos(self, weights, submetrics):
8          # return overall quality
```

Figure 6.3: Developer: Evaluation Module Implementation

```
1  #include "RSDGMission.h"
2  ...
3  mission = RSDGMission(DESCRIPTION_FILE);
4  //mission->promtCustomQuality();
5  mission->regKnob(knobNames, &paras);
6  mission->setUnit(TOTAL_WORK_UNIT, UNIT_PER_CHECK);
7  mission->setBudget(BUDGET);
8  for (int i=0; i<TOTAL_WORK_UNIT; i++)
9  {
10     work(paras); // actual work
11     mission->finishOneUnit();
12 }...
```

Figure 6.4: Developer: Application Source Code Modification to Involve Rapids

and their settings (Line 5), b) inform Rapids about the execution progress (Line 12), (e.g., how much remaining work) c) expose virtual knobs corresponded to sub-metrics to users if the developer chooses to support custom quality (Line 4), and d) accept user defined budgets (Line 7). More detailed API's to fine-tune the mission are also available. I believe that the additional demand on the developers is reasonable and well within their expertise.

Training and Model Construction: On the development platform, Rapids generates an exhaustive training set over all knob settings, eliminating all invalid configurations according to the developer's *KDG specification*. The developer provided *Evaluation Module* is used to calculate costs and sub-metrics (or default metrics), which in turn build the models and RS as discussed in Section 4.3.2.

Users' Effort and Runtime Control: Before execution of the main routine, users can optionally express their quality preferences through a developer provided interface. Rapids will then finalize the quality model. If no customized quality is specified, the developer defined default model is used. Users then provide a resource budget (time or energy). During runtime, Rapids tracks the remaining resource budget relative to the remaining work. The runtime system continuously monitors the application and performs automatic re-configuration if needed to maintain the maximum possible quality while respecting the provided budget even under changing conditions due to system uncertainties.

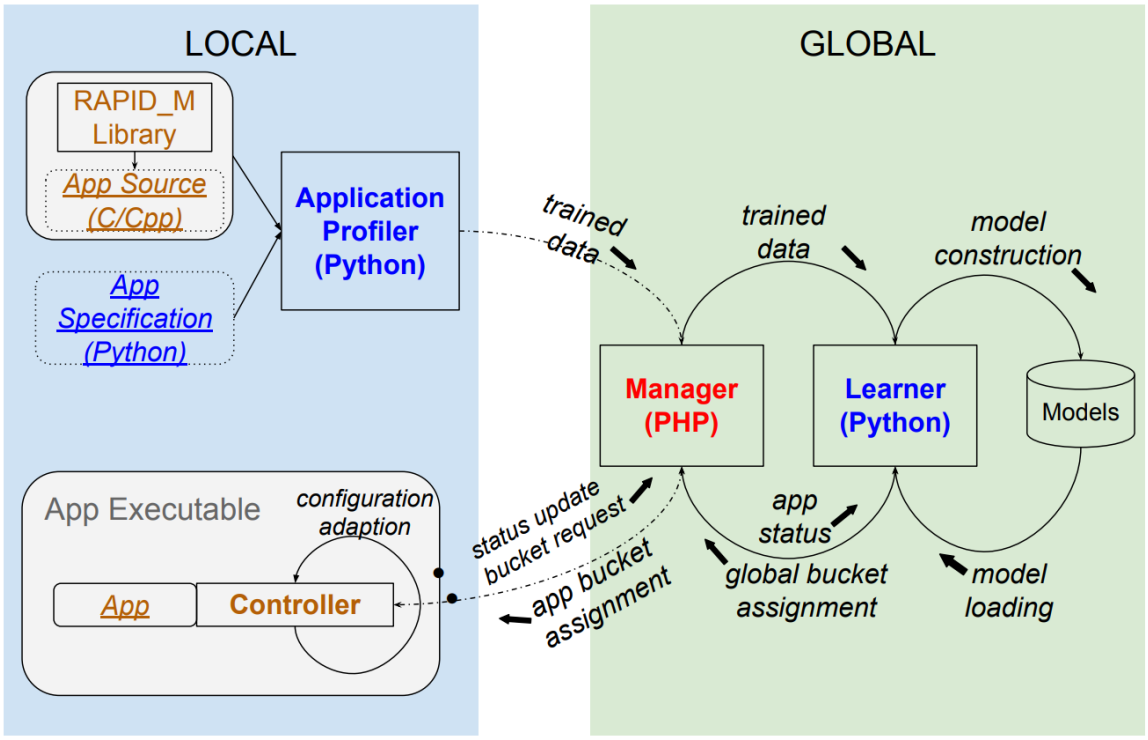


Figure 6.5: Rapids-M Framework Implementation Overview:
Boxes with solid border: provided by Rapids-M; Boxes with dashed border: required from developers

6.2 Rapids-M Implementation

The Rapids-M framework is implemented as a set of offline and online modules. During new application development or adapting an existing application for execution within Rapids-M, the application developer has to provide information to Rapids-M's offline local module via provided, simple APIs as the same in Rapids. This information enables the offline application profiler to automatically collect profiling data for an application, which is needed for offline construction of p/m-models and bucket partitioning, and online configuration management to track application progress through monitoring completion of work units. The offline model training is performed on the target platform to produce the system footprints and stresser workloads. The artificial stresser workloads are generated by LINUX's "stress" tool [110].

In contrast to other approaches (e.g., [114]), Rapids-M collects each application's pro-

file data, and constructs the models independently of other applications, making Rapids-M easily scalable. The offline generated information is represented in an application profile and stored on the Rapids-M server, which also hosts the online global configuration manager. At runtime, i.e., just before application execution, the application user specifies an execution time budget (cost budget). During application executions, the global configuration manager keeps track of all active applications' progress and their remaining budget. It then determines bucket assignments for each application using Algorithm 2. Application start and termination events trigger the reevaluation of the local bucket assignments, in addition to explicit requests from local controllers. Each active application has its own local controller, which communicates with the global configuration manager to receive bucket assignments or to request global bucket reassignments. The local controller is responsible for selecting the optimal configuration within its assigned bucket. Configurations in the same bucket shares similar footprints, thereby local reconfiguration can be performed safely without impacting the overall system footprint visible by other applications, i.e., performance impact on others is small.

Figure 6.5 shows the overview of Rapids-M framework. I implement and evaluate the system on a single-socket machine with 4 Cores at 3.7GHz, and 16GB RAM at 2666MHz. The Rapids-M profiler and learner (model construction and prediction) is implemented in Python and has ˜12.2K lines of code. Rapids-M development framework is implemented in Python for developers to hook up the applications for training. The local runtime controller is implemented as a C++ library (˜6.5K lines of code) with simple API's to be integrated into source code by developers. During runtime, it communicates with the online global configuration manager (in PHP). The manager communicates with the learner via sockets.

Rapids-M constructs the m-model for the target machine on the server using the locally created stresser profiles. The construction took 173 seconds to complete without feature selection.

Local Application Profiler: The profiler is designed as a data collector for the developers

Table 6.1: Data Collected by Rapids-M Profiler

	Data	Description	Usage
base-run	c	cost when alone	calculate slow-down
	v	footprint when alone	m-model construction
	q	output quality	online selection
stress-run	$[c_{as}]$	cost when app+stressers	calculate slow-down
	$[v_s]$	footprint of stressers	m-model Construction
	$[v_{as}]$	overall footprint of app+stressers	m-model Construction p-model Construction

to train the application for Rapids-M required models. The profiler requires the application specification from developers, i.e., individual knob settings, output quality evaluation, and execution instructions.

Table 6.1 shows the profiling strategy of Rapids-M. For each configuration C_i, Rapids-M profiler runs $1 + K$ tests. The first run is a *base-run* that the application runs alone on the target system using C_i and measure the Base Cost: c, Output Quality: q, and System Footprint: v. Then it runs the application K times each with a different "stresser". After each run, Rapids-M records the System Footprint of the stresser when running alone: v_s, the Overall System Footprint: v_{as}, and the execution time of the application c_{as} executing with the stresser.

Table 6.2 summarizes the offline overhead of development in Rapids-M. The second column reports the number of lines of code changes required from the developer to integrate Rapids-M into the origin source code. The column "# configs" reports the total number of configurations in the application. Note that in Rapids-M, all knobs have discrete settings. The column "data profiling" reports the time required to collect the data from each application. The last column reports the time to construct the buckets and their p-models. As shown in the table, the effort required from the developer is minimal, as compared to the relatively large configuration space. The data collecting phase can be time-consuming and the overhead is proportional to the complexity of the application.

Global Learner: After the profiler(s) collects the training data, the data are sent over

Table 6.2: Rapids-M Implementation with Offline Overheads

	# Lines of Source Code Changes	Offline Training Overhead		
		# configs	data profiling	bucket/p-model constr
Swaptions	14	10	44.4 mins	22.72 secs
Bodytrack	17	50	72.5 mins	35.69 secs
Ferret	20	700	8.7 hours	189.79 secs
Facedetect	20	90	28.2 hours	374.34 secs
SVM	20	250	12.4 hours	21.32 secs
NN	20	250	27.6 hours	15.95 secs

to a global server for model construction. During the training, the learner performs the following tasks:

- Construct / Update the m-model for the machine: using all the observed system footprint from all applications, $X = [v_s, v_a]$ $Y = [v_{as}]$
- Compute the bucket for the application: using the observed system footprint of all configurations: $[v]$
- Construct the p-model for the bucket: using the overall system footprint and the slowdown for the application: $X = [v_{as}]$, $Y = [c_s/c]$
- Update the bucket based on the prediction accuracy

Global Manager: During the initialization phase, after the learner finishes the bucketization / p-model construction for the application and updates the m-model, the models will be stored on the server. Summary information will be kept as *app_profile* for runtime control.

During runtime, the manager keeps track of the state of all applications on the machine. There are three cases when running applications need to contact the server:

- Before execution: The application notifies the server that it is about to run so that the server can predict the slow-down for all the currently active applications.

- Re-configuration: The application actively requests a new bucket assignment when no configuration in the current bucket can satisfy the budget constraint.
- Routine Check: The application periodically checks with the server for updated bucket assignment. This is needed since the assignment can be changed when new applications start on the same machine. By default, Rapids-M performs a routine check after each 10% of the total work units.
- After execution: The application reports the termination of the execution to let the server re-evaluate the slow-down for the remaining applications.

Each application on the machine is in a state of either ACTIVE or IDLE. All ACTIVE applications will also be associated with a *budget*. The manager updates the global bucket selection whenever a new request comes in, except "Routine Check."

Local Controller: The global manager returns the optimal bucket assignment for an application along with an optimal configuration within the bucket based on the remaining budget and execution progress reported by the application. The local controller deploys the configuration. However, unexpected disturbances like input dependencies may affect the real execution time for the application. The local controller adapts the application behavior by re-selecting the optimal configuration within the assigned bucket.

CHAPTER 7
EVALUATION

In this chapter, I report the detailed evaluation results of both Rapids and Rapids-M.

7.1 Rapids: Single-App Scenario Evaluation

I evaluate the system with respect to the work's three main contributions: 1) *Developers:* Configuration space reduction from developer encoded insights, 2) *Machine:* Training time reduction and the model accuracy, 3) *Users:* Improvement of user-preferred sub-metrics relative to default metrics. Finally, I evaluate the runtime performance by measuring the overhead and the overall output quality.

7.1.1 Problem Size Reduction from Developer's Insight

In Section 4.4, I report the configuration space reduction by leveraging structural insights from the developers. The pruned search space results in reduced training time but at the same time may introduce some errors in the prediction.

Reduced Training Time: Filtering out "invalid" configurations through the KDG or calculating RS both prune the configuration space and thus reduce the training time. The total training time may not linear in the number of trained configurations due to the execution time difference for each configuration. I report the total training time for constructing the cost model through different approaches; i) **FULL**: All configurations, no developer encoded dependencies ii) **KDG**: Configurations with dependencies, iii) **Rand-20**: A heuristic used in CALOREE [21] that constructs the model with 20 random configurations, iv) **RS_P(RAPIDS)**: Partition-based RS, and v) **RS_S(RAPIDS)**: Selection-based RS.

As shown in Figure 7.1, allowing developers to specify dependencies among knobs

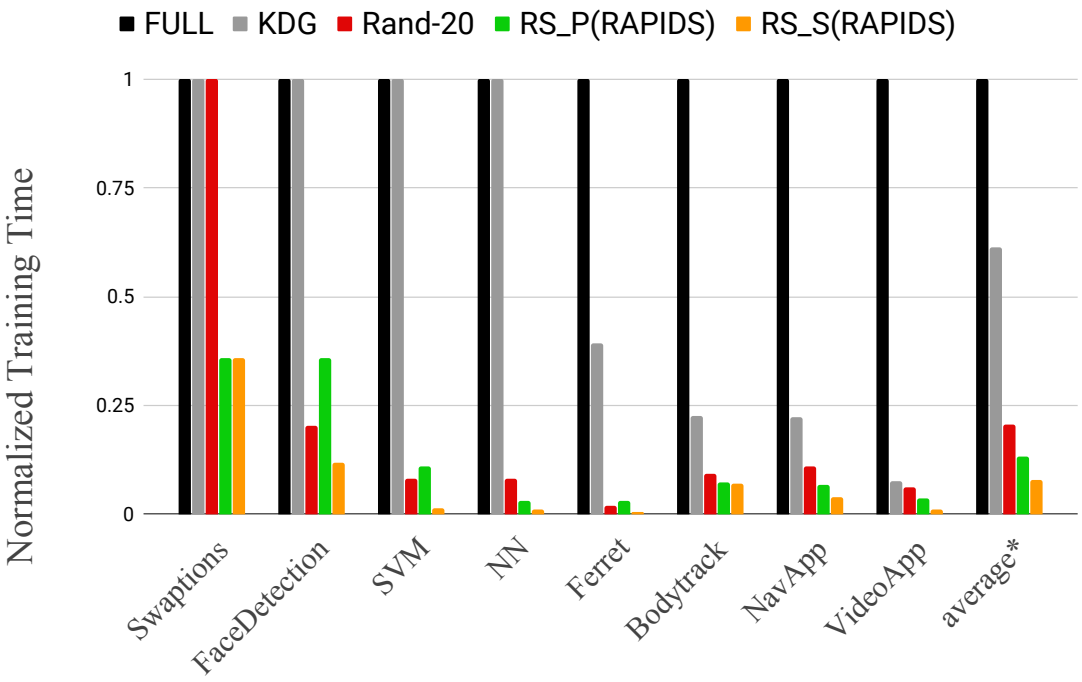


Figure 7.1: Normalized Required Training Time, higher the slower

reduces the training time by an average of 38.55%. The RS calculated by Rapids further reduces the training time to 12.76% (7.5% on average), which translates to up to 13× faster retraining when porting to other devices. When using training for a pre-defined number of samples (e.g, 20), the training time is reduced to an average of 21.24%. The significant reduction in training cost of the RS approach enables Rapids to quickly retrain itself when applications are ported to unknown target devices.

7.1.2 Model Validation

Rapids constructs the cost model quickly based on a pruned space (KDG) and/or RS. However, a reduced training set may lead to a higher prediction error. To this end, I evaluate the accuracy of such reconstructed models on the target machines. For fairness, **FULL** is not considered because all configurations excluded in the **KDG** are "invalid". **KDG** serves as the oracle with $accuracy = 1.0$ because the model is built with all observations. Fig 7.2 shows the corresponding prediction errors across all valid configurations. On average,

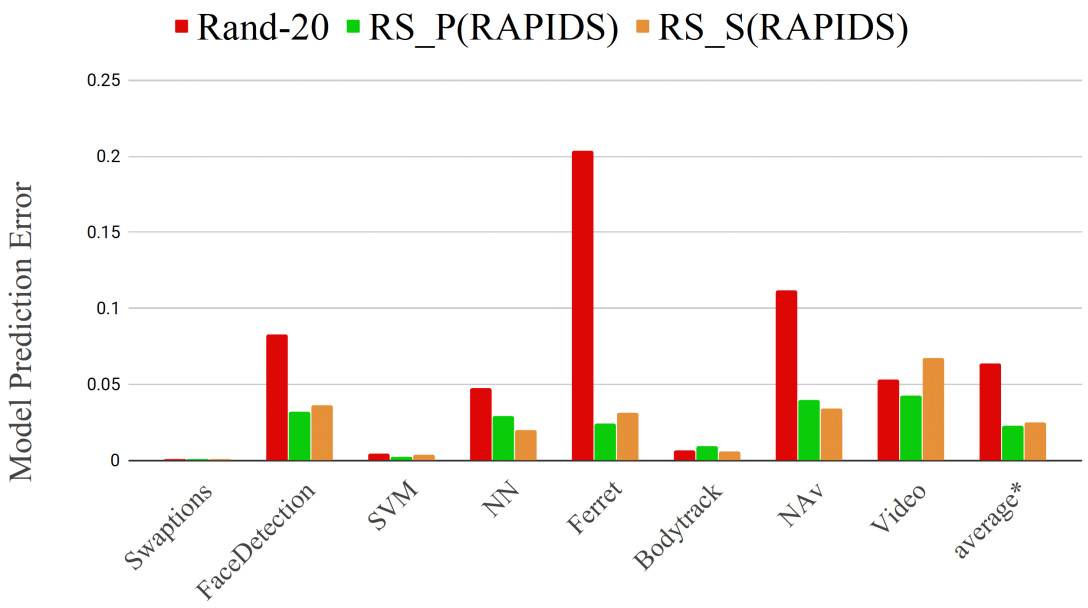


Figure 7.2: Model Prediction Error on Target Machine, lower the better

the model constructed from **Rand-20** results in a 6.39% prediction error (averaged in 10 runs). **Rand-20** performs well in simple applications like Swaptions, but has high error for complex applications. In Ferret, the model has an average error of 20.4% with a maximum of 65.1%. The two RS strategies have a much smaller average error: 2.4% and 3.1%.

7.1.3 Application Output Quality

Clearly, model accuracy can impact the configuration selection, i.e., the achieved overall configuration quality. The result was shown in Section 4.4.2 and the overall quality of the execution using both RS strategies outperform $Rand - 20$.

7.1.4 Custom Quality

Application developers can provide a customizable, higher level quality notion through virtual knobs where users can express their preferences for a particular quality outcome at a level of abstraction that makes sense to them. In turn, Rapids automatically builds the required quality models to support configuration selection and reconfiguration. I evaluate this benefit on the four applications with custom quality metrics.

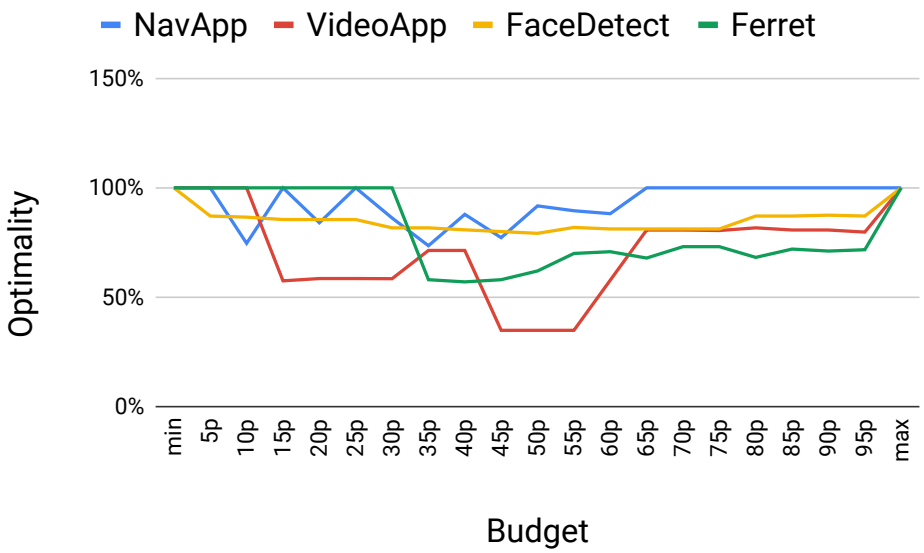


Figure 7.3: Optimality of Default Solution under Custom Quality across Different Budgets. X-axis: Budget Percentage, 50p:50%.

- *Optimality Evaluation:* Under a given budget, the optimal configuration may vary between the default (Q) and the custom (Q_c) quality metric. The default optimal configuration (c) under Q may provide significantly lower quality ($Q_c(c)$) than the custom optimal configuration (c_c) under Q_c. I show this optimality reduction under different execution time budgets from min to max time requirements in 5% increments in Figure 7.3 min and max are the times required for the highest and lowest quality configurations. By enumerating different possible user priorities from 1.0 to 2.0 in increments of 0.1 on one sub-metric, I report the average relative quality value $Q_c(c)/Q_c(c_c)$. The result show that there is substantial quality improvement due to customization across all applications, in particular under intermediate budgets where tradeoff decisions are most crucial.

- *User Preferred Sub-Metric Comparison:* The improvement on the preferred sub-metric was shown in Section 4.4.3. Here I report a detailed result for Ferret to illustrate how different user-provided preferences affect the sub-metric. As shown in Fig 7.4, I evaluate the *Coverage* and *Ranking* sub-metrics by running the application with budget of *1/2 * (*max *- *min*)*. I adjust the preference of one of the metrics from *1.0* to *2.0* (shown in solid lines)

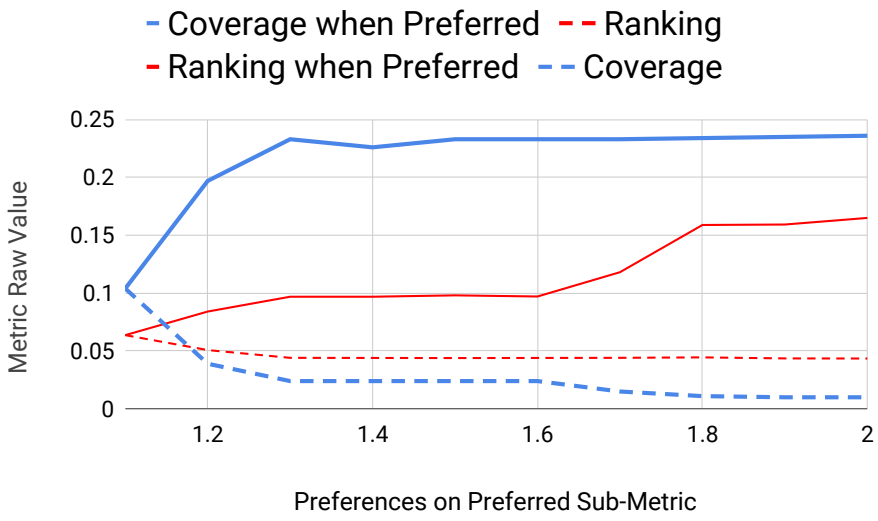


Figure 7.4: Change of Sub-Metric in Ferret. *Solid*: preferred; *Dashed*: not preferred

and keep the other as *1.0* (shown in dashed lines). For example, solid-blue and dashed-red lines represent the change of sub-metric values when "Coverage" has an increasing preference. Figure 7.4 shows an improvement on preferred sub-metrics as preference increases. The metric that is not preferred suffers. This is expected since knob settings that benefit one sub-metric may hurt the other.

7.1.5 Reconfiguration and Overhead

Rapids constructs the performance model based on a training set of input from each application. During runtime, the predicted and the real performance can be different for three main reasons: 1) Embedded prediction error in the model, 2) Application input dependencies, and 3) Dynamic runtime environment. Table 7.1 shows the input dependency of different applications. Bodytrack, FaceDetection, and Ferret have the most significant deviation of cost per each work unit.

Rapids overcomes these issues by constantly monitoring the resource usage and performing reconfiguration if necessary during runtime. The full reconfiguration procedure in Rapids has three steps:

Table 7.1: Application with Input Dependency

	Training_Input	Test_Input	Mean	Std
Swaptions	5	50	2191.22	25.72
SVM	2	10	2658.3	11.98
NN	2	10	3165.2	57.26
Bodytrack	30	261	235.61	16.16
FaceDetection	50	861	393.27	65.13
Ferret	20	3500	87.85	80.19

1. Monitor ($\sim$1ms): Calculate the new budget per work unit by checking the remaining budget and execution progress

2. Problem Solving ($\sim$17ms): Generate and solve the new optimization problem and retrieves the new configuration.

3. Result Deployment ($\sim$1ms): Apply the new configuration to application.

If no solver is available on the target device (e.g., Gurobi [109] cannot be deployed on ARM), Rapids contacts a remote server for results. The overhead of each remote reconfiguration average 191ms/133ms (ARM/Android). In NavApp and VideoApp, I report the overhead as additional energy consumption.

The *Monitor* (step-1) frequency is 10% of the total work units by default and can be tuned by the developers. Figure 7.5 shows the different runtime behavior of the applications when the monitor frequency changes. In this experiment, I run all applications under three different budget (0.2, 0.5, 0.8 $\times$ ($budget\ range$)) multiple times with different monitoring granularity. The experiment aims to investigate deeper into the effect of different monitor frequencies so that the developers can also follow this work to determine what frequency to use during their development processes.

- *Budget Utilization and Output Quality:* The figure on the left of Figure 7.5 reports the change of the budget utilization when the frequency increases. In general, having a higher monitor frequency could potentially increase the budget utilization for having a more chances to update the configuration according to the current consumption. Lower monitor-

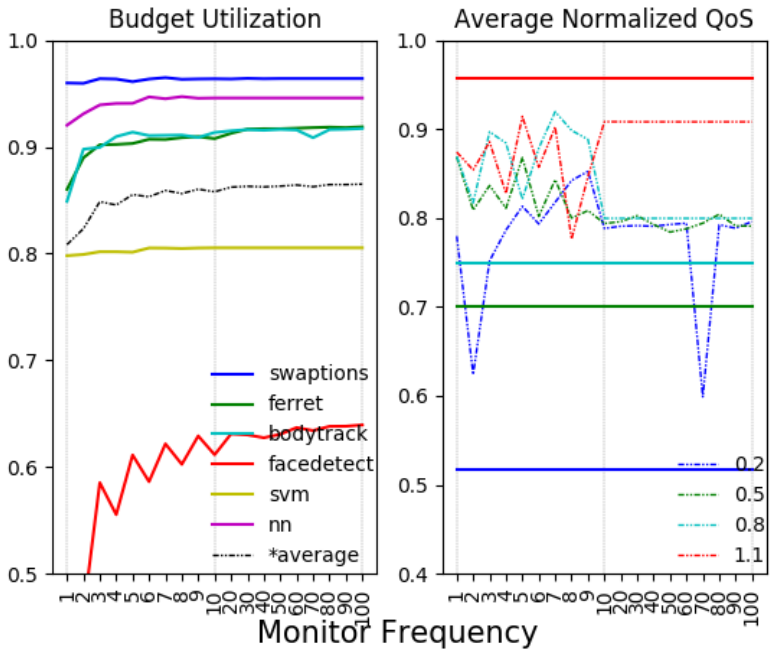


Figure 7.5: Monitor Frequency and Budget Utilization. X-axis: number of budget examination performed, x=1: monitor only once, x=100: monitor 100 times (or the finest granularity supported by application);

ing frequencies can lead to missing of potential reconfigure opportunities. As shown in the graph, the utilization rate grows rapidly from $frequency = 2to10$. Then, it keeps relatively stable after the frequency increases even higher. For applications with higher input dependency, a more frequent reconfiguration rate could better utilize the user-provided budget.

The figure on the right of Figure 7.5 reports the change on normalized QoS. Due to the different output quality reaction to different provided budget, rather than reporting individual output quality, I only report the average normalized quality for all applications. There are four pairs of lines in the graph, where the dashed line in each pair shows the normalized output quality under different monitoring frequencies. The solid line shows the corresponding "pseudo optimal" static configuration by offline search. Each pair of lines show the result for a particular budget setting. The output quality starts low at the beginning for all budget settings(when only reconfigure once), and remains at the same level after

x=10. Interestingly, the output quality by reconfiguration is always higher than pseudo optimal when X=1 under budget=0.2, 0.5, and 0.8. This is because that the "pseudo optimal" configuration is found within the search space formed up by discretized continuous knobs. The real optimal configuration can be missing in such discrete space but can be found by having a model constructed for continuous knobs. When budget=1.1 and X=1, the output quality is lower than pseudo optimal because of the error in the quality model. However, the quality increases as the monitor frequency increases by benefiting from reconfiguration.

7.1.6 Overhead Optimization

Ideally, systems like Rapids can monitor the budget usage and working progress after every work unit to avoid wasting budget or violating budget constraints. However, excessively frequent monitoring and reconfiguration can lead to higher overhead without noticeable quality gain. To minimize the overhead, I embedded two optional optimization strategies in Rapids:

1. *Budget Optimization*: Terminating the reconfiguration procedure after "Monitor" (Step-1 as described in section 7.1.5) if the new budget per work unit is within 5% range of the previous value.

2. *Configuration Optimization*: Terminating the reconfiguration procedure after "Problem Solving" (Step-2) if all the knob values in the new configuration are within 1% of their previous values.

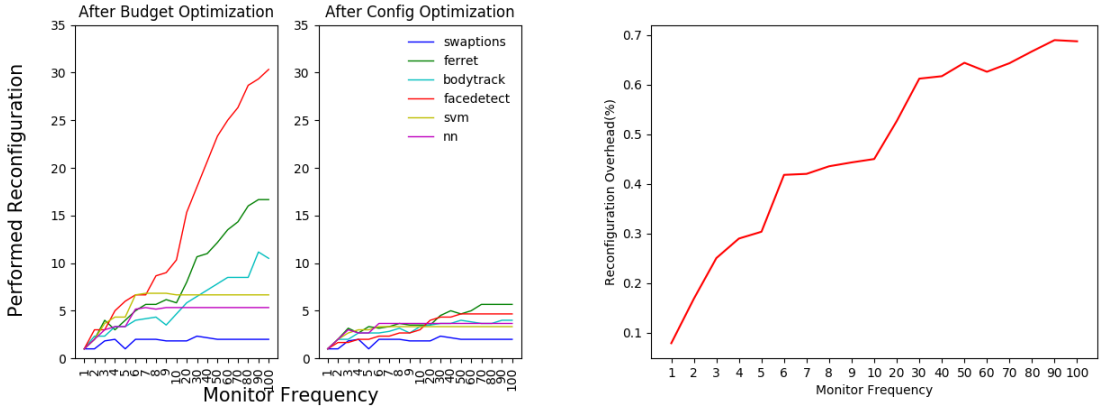
The intuition behind the two methods is to avoid less meaningful reconfigurations if the resulting new configuration will be identical or similar to the current configuration. I conduct two series of experiments to evaluate such optimizations.

- *Actual Performed Reconfiguration:* Figure 7.6a shows the number of reconfigurations that are actually performed by Rapids given different monitoring frequencies. For each point on each line in the left figure, the difference between the X value (monitor frequency) and

the Y value (actual performed reconfiguration) is the number of "Solving+Deployment" (step-2/3) being skipped by the budget optimization. The difference between the left and right figure is the number of "Deployment" (step-3) being skipped by the configuration optimization.

As shown in the graph, with under a granularity of 1% (100 on X-axis), all the applications reconfigure less than 10 times of "Deployment" (step-3) after optimized. Finally in Figure 7.6b, I report the portion of total execution time across all applications under different monitoring frequencies. The total overhead time has a similar trend as the left graph in Figure 7.6a because of the overhead being dominated by the problem solving (step-2).

As discussed, more frequent reconfigure frequency does not yield noticeable quality gain. Even though the total overhead is still below 1% when X=100, SYSTEM is set to have a default monitor interval of 10% of the entire work units, i.e. reconfigure 10 times.



(a) Performed Reconfiguration under different Monitor Frequency

(b) Reconfiguration Overhead to total execution time

Figure 7.6: Reconfiguration Overhead

Table 7.2 reports the overall Rapids overhead for all the applications. As shown in the table, the overhead is usually below 3% comparing to the entire execution, which is considered negligible.

Table 7.2: Overhead in All Applications

	w/ Rapids	w/o Rapids	Overhead
Swaptions	461	458	1.09%
Bodytrack	264	260	1.44%
Ferret	551	544	1.22%
FaceDetection	92	89	3.2%
NN	23.1	22.5	2.67%
SVM	35.7	34.6	3.17%
NavApp*	-	-	<0.05%
VideoApp*	-	-	<0.05%

7.1.7 Summary of Key Results

Specifying knob dependencies using Rapids requires minimal developer effort with around 20 lines on average for the benchmarks, which makes Rapids an expressive, flexible, and easy-to-use. Such dependencies can help Rapids filter out invalid configurations, therefore shorten the training time by an average of 38.55%. This empowers the developers with more control over defining the application structure.

Based on the KDG, two Representative Set (RS) strategies enables training time reduction up to 2 orders of magnitude. The resulting model from the reconstruction using RS has an average error around 3% and outperform the $Rand - 20$ approach which causes significant errors for 3 of our applications (Ferret, NavApp, and FaceDetection). This implies that choosing the training set carefully is crucial and have to be application specific.

The *virtual knobs* enable users to customize the quality outcomes at a level of abstraction that makes sense to them. For our benchmark applications, Rapids manages to improve the preferred metric to up to 3.2× and 1.76× on average. This result justifies the necessity and the feasibility of supporting custom quality.

The runtime overhead of Rapids is small for all of our benchmark applications in terms of execution time or energy consumption. Deeper investigation into the monitor frequency show that the budget utilization can be improved by performing more frequent reconfigurations. However, such improvement is not found in the quality, as opposed to what I expect. This could be due to multiple reasons, including how quality can be input

dependent, or how the input are ordered in the sequence, etc. The lesson learned from the investigation in overhead is that having a moderate amount of reconfiguration should be enough for most applications. That is also why I pick "10" as the default monitor frequency in all of my other experiments. However, overhead can also be determined by the length of the mission. I measured that the average raw cost (time) of performing a full reconfiguration is around 20ms. This makes Rapids not suitable for applications/missions whose total execution time is on the magnitude of milliseconds.

7.2 Rapids-M: Cross-Application Evaluation

For Rapids-M, I first evaluate the model accuracy of both p-model and m-model. The error in the performance model can affect the performance of configuration selection. However, if online reconfiguration is allowed, even a bad model could also achieve an acceptable result due to the online adaption, as shown in Section 5.5. Therefore, another set of experiments is performed to evaluate the performance of Rapids-M compared to multiple existing approaches in a static mode with no reconfiguration. In particular, I report the stability of different strategies by comparing the number of performed reconfigurations. Then I compare the failure rate and output quality to show how Rapids-M outperforms existing strategies.

7.2.1 Model Validation

Rapids-M predicts the performance degradation (slowdown) of an application by first predicting the overall system footprint using the m-model and then the per-application slowdown with the corresponding p-model. The prediction accuracy relies on the quality of both m-model and p-models.

Bucketing and p-models: To validate both the *necessity* and the *benefits* of bucketing, I perform a series of tests to show that the prediction accuracy of per-configuration slowdown is high enough with only a few buckets.

Table 7.3: Selected Model for all Benchmarks.

	# configs & # buckets	Model	Poly	MRE
Swaptions	10 & 1	BR	T	2%
Bodytrack	50 & 2	BR / BR	T / T	1%/2%
Ferret	700 & 5	EN / BR / BR / BR / EN	T / T / T / T / F	2%/1% / 2% /1%/2%
FaceDetect	90 & 3	EN / BR / BR	T / T / T	1% /1% /1%
SVM	250 & 4	LS / LR / BR / BR	T / T / F / T	1%/2% /1%/2%
NN	250 & 5	BR / BR / BR / LS / BR	T / T / T / F / T	2%/1% /1% /3% /1%

m-model: I then evaluate the accuracy of using m-model to predict the overall system footprint.

p+m Model: Finally, I evaluate the entire strategy by predicting the performance degradation under a controlled environment with the artificially introduced workload.

Rapids-M constructs the p / m models from a set of publicly available models as discussed in Section 5.3: 1) Neural Net **(NN)**[112] with 1 hidden layer and 50 neurons, using 'relu' as the activation function; 2) Linear regression **(LR)**[102]; 3) Lasso **(LS)**[102] regression with cross-validation; 4) Elastic net **(EN)**[102] regression with cross-validation; and 5) BayesianRidge **(BR)**[102] regression. The Rapids-M framework uses a modular design and allows developers to add more models to the model pool.

Bucketing with p-model*:* The purpose of bucketing configurations is to reduce the search space size from the number of all combination of configurations to the number of combinations of buckets. However, this approach is constrained by requiring a good prediction of both environment and slowdown. Configurations are clustered based on generated system workloads using a hierarchical clustering strategy. The number of buckets is determined by the accuracy of the per-bucket slowdown model (p-model).

In Table 7.3, the column named "# configs & # buckets" shows the number of buckets constructed from the total number of configurations. Column "Model" and "Poly" report

Table 7.4: Selected Features For All Benchmarks.

Application	Selected Feature											
	EXEC	FREQ	INST	INSTnom%	IPC	L2MISS	L2MPI	L3HIT	L3MPI	L3MISS	PhysIPC%	MEM
Swaptions	*	*	*	*	*	*	*	*	*	*	*	*
Ferret	*	*	*	*	*	*	*	*	*	*	*	*
Bodytrack	*				*		*				*	*
SVM	*	*	*	*	*	*	*	*	*	*	*	*
NN	*	*				*		*				*
FaceDetect	*	*	*			*	*	*	*	*	*	*

the type of model and whether the model use polynomial features or not. Column "MRE" reports the per-bucket mean relative error of slow-down prediction. This different models and their settings being selected reveal that there isn't a unique model which fits for all applications. Comparing to ESP [71] where Elastic Net is used across multiple applications, Rapids-M locates the best model on an application bucket level.

Table 7.4 reports the selected features for each application using the criterion of having a R2 score higher than 0.92. Different selections on features reveal that not all features are important to specific applications. For example, Bodytrack is less sensitive to most of the cache-related features (L2MISS/HIT, L3MISS/HIT). By not considering those not-so-important features, the complexity of the model can be reduced.

Rapids-M reduces the overall problem size by clustering large configuration spaces into a few buckets. The approach reduces the size of the following optimization problem while still providing an a slow-down prediction with an error below 3%, and 1.5% on average, which justifies the feasibility of such approach.

System Profile Prediction with m-model*:* I then validate the system profile (resource consumption) prediction of the m-model. On average, Rapids-M provides a R2 score of 0.93 for all features. Different models yield different prediction accuracy and require different training times. Rapids-M first locates the best model that yields the highest R2 score. Then it selects the least number of input features for the model to achieve an accuracy threshold. In the experiment, I use R2 $\geq$ 0.92 as the criterion. Figure 7.7 shows the R2 score for all features using different models. For all regression methods except Neural Net (**NN**), I report the R2 score for both linear and polynomial input features. For each feature, the last

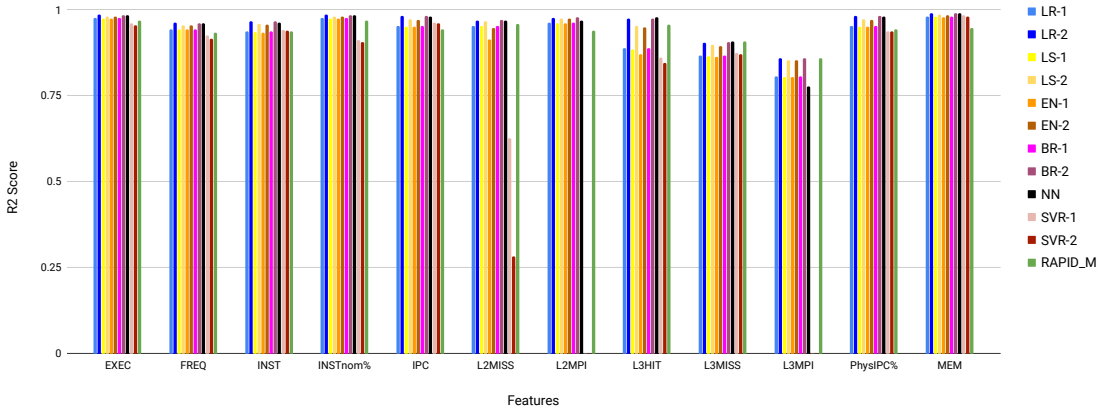


Figure 7.7: Prediction R2 score on All Features using Different Models; RAPID_M: best model with feature selection

bar shows the Rapids-M selected model with highest R2 and selected feature.

Table 7.5 reports the training overhead. The column "MAX" reports the longest times for all models in seconds. Column "Best" lists the time for constructing the best model using all features with highest score. Column "Rapids-M" reports the time required by Rapids-M to construct the best model with the selected features. Column "Ratio" compares the time for Rapids-M and "Best". It shows that Rapids-M reduces the m-model training time to less than 5% for most features. For for some features (e.g., "L3MISS"), the selected model can take longer to converge when only using selected features. The last two columns in Table 7.5 report the selected model type and the model accuracy in terms of relative error.

To summarize, more expensive models do not always yield more accurate predictions. A subset of features may be good enough for a given model. Different system features may be best fit into different models and may not require all available features. Rapids-M selects the best model for each individual feature and filters out less relevant features. This selection phase reduces the training time by 42% on average (in many cases, more than 99%) while maintaining the R2 score at 0.93 comparing to the best model at 0.96.

Table 7.5: Best Fit Model For All System Features. **MAX** is longest training time (secs); **Best** is training time (secs) to produce best model using all features; Rapids-M is time (secs) to construct it model using selected features; **Ratio** is Rapids-M / Best.

	MAX	Best	RAPID_M	Ratio	Model	MRE
EXEC	56.31	0.24	0.01	<=0.01	LR	12.9%
FREQ	48.83	0.24	0.01	<=0.01	NN	7.4%
INST	49.50	0.23	0.01	0.03	BR	4.7%
INSTnom%	51.25	0.23	0.01	<=0.01	LR	12.9%
IPC	42.91	0.43	0.01	<=0.01	BR	5.3%
L2MISS	88.44	39.8	39.96	1.00	BR	29.8%
L2MPI	137.95	0.45	0.01	<=0.01	BR	38.2%
L3HIT	360.29	40.61	45.98	1.13	NN	3.1%
L3MISS	81.24	46.48	86.23	1.86	NN	19.4%
L3MPI	88.20	0.41	0.39	0.96	BR	27.2%
PhysIPC%	87.54	0.48	0.01	<=0.01	BR	5.3%
MEM	88.51	0.48	0.01	<=0.01	BR	10.0%

7.2.2 Optimality

Table 7.5 reports the accuracy of the m-model. Its error will be propagated to the predicted slow-down error (see Figure 7.3), potentially negatively impacting the bucket and configuration selection. I conduct another simulation to investigate how such error propagation affects the optimality of the selection.

The experiment first runs an application with a stresser and records (1) the footprint of the application, fp_a, (2) the footprint of the stresser fp_s, (3) the overall environment env, and (4) the application slowdown sd. The stresser could be another application or an "stress" instance. With the performance profile of the application, the optimal configuration c_{opt} can be computed using sd. I then compute the selected, "optimal" configuration c_p by using the slowdown predicted by applying the p-model to env. Finally, I predict the environment env_m by applying the m-model to [fp_a, fp_s], and predict the slowdown using the p-model on env_m, then make the selection c_{m+p}. The difference between c_{opt} and c_{m+p} or c_p is caused by the error in the p-model with or without the m-model. I evaluate such difference in terms of (1) hit rate: whether the selection is identical, and (2) quality loss:

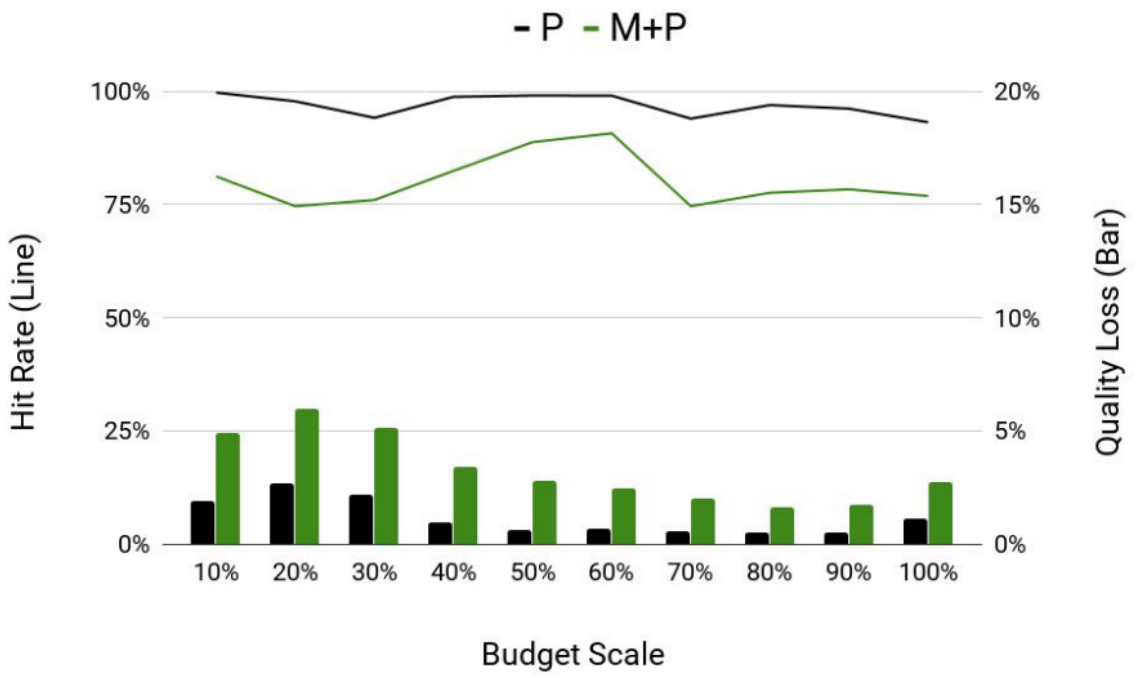


Figure 7.8: Impact of Error Propagation on Optimality under Different Budget Scale; P: Applying p-model to the measured system environment; P+M: Rapids-M approach that applies the p-model to the output of the m-model

relative loss of quality caused by the error, calculated by $|(\tilde{q} - q)|/q$ where $\tilde{q}$ and q denote the quality of the configuration picked by the strategy and the optimal configuration. Since c_{opt} is the optimal configuration found offline, any selection that yields a higher quality than c_{opt} will also be considered a loss of quality for under estimation of the cost.

I run the simulation covering the whole range of possible budgets from 10% to 100% of $MAX - MIN$, where MAX and MIN represent the highest and lowest cost of all configurations when running alone.

Figure 7.8 shows the optimality evaluation result of the simulation. The X-axis represents the available budgets percentage. The lines following the left axis report the average correctness of the configuration selection across all applications, e.g, 100% means that the optimal configuration from Rapids-M gives the exact same configuration as the oracle. The bars following the right Y-axis shows the relative loss of quality. Note that the "P_ONLY" is not used in the real deployment of Rapids-M but is only intended to show the impact of

prediction error caused by m-model. This is because the bucket selection is determined by invoking m-model, however it is assumed to be pre-defined in this simulation. As shown in the graph, if the environment is known, the error in p-model contributes to $\leq$10% of selection difference as the optimal. The errors of combining m and p-model (Rapids-M) introduces 19.9% of different selection on average. However, the wrongly selected configuration is close enough to the optimal configuration that only contributes to less than 6% of output quality loss with an average of 3.3%. This indicates that the embedded errors in M or P-Model only affect the output quality to a limited extent.

7.2.3 Global Reconfigurations

I conduct several experiments to show the effectiveness of key components of our approach. These experiments concentrate on showing the performance of Rapids-M in producing configuration selections of high quality. The evaluation methodology compares possible configuration selection strategies with Rapids-M on different sets of concurrently active applications.

- Review of Rapids-M*'s Features:*

Reconfiguration is a central feature of Rapids-M. There are two main reasons that make reconfiguration in Rapids-M more complex compared to the single application scenario. The first (global) is the main focus of Rapids-M, namely the changes to the execution environment and therefore available resources due to applications entering and exiting their executions. The second (local) reason is related to individual applications reconfiguring to compensate for small disturbances during runtime but at the same time avoiding affecting other applications. A special feature of the Rapids-M framework is the fact that global reconfigurations are handled through bucket selection, while local reconfiguration is handled through reconfiguration within the bucket, thereby not impacting other application's configurations. Local reconfigurations are handled by individual applications through Rapids or other existing techniques without contacting the Rapids-M server, making them much

more efficient.

7.2.4 Static Evaluation

In Section 5.5, I report the improvement of output quality in Rapids-M compared to other approaches when running multiple applications, each with a specific budget. The result were collected under the condition that each application can re-configure. In this section, I introduce another set of experiments that assess the effectiveness of the different strategies under a controlled (static) environment where the number of executed applications is fixed. An optimal strategy would select single configurations for each application that together maximize the global quality. Each application starts at the same time and reconfiguration is disabled. The result will indicate how well the strategies work in modeling the interactions across active applications. If an application finishes before other applications, it will restart with the same configuration until all applications finish at least once thereby maintaining the overall system environment.

Strategies are evaluated on three aspects:

- Violation: Execution time $\geq$ provided budget.
- Misprediction: No feasible configuration can be found by the strategy though there exists at least one. (Usually caused by over-prediction on slow-down.)
- Exceeding Rate: The rate reports by what fraction the execution time exceeds the budget. E.g, rate=1.0 indicates the overall runtime is twice the budget.

Figure 7.9 shows the performance of the computed global configurations by the different selection strategies when the budget is high (each application gets a budget with scale = 150%). The scale is defined the same as in Figure 7.8 that $budget = MIN + x\% * (MAX - MIN)$ when $scale = x\%$, where MAX and MIN represent the highest and lowest cost of all configurations when running alone. Figures 7.10 and 7.11 show the result when the budget is squeezed to a lower value, 100% and 80%. The results show that:

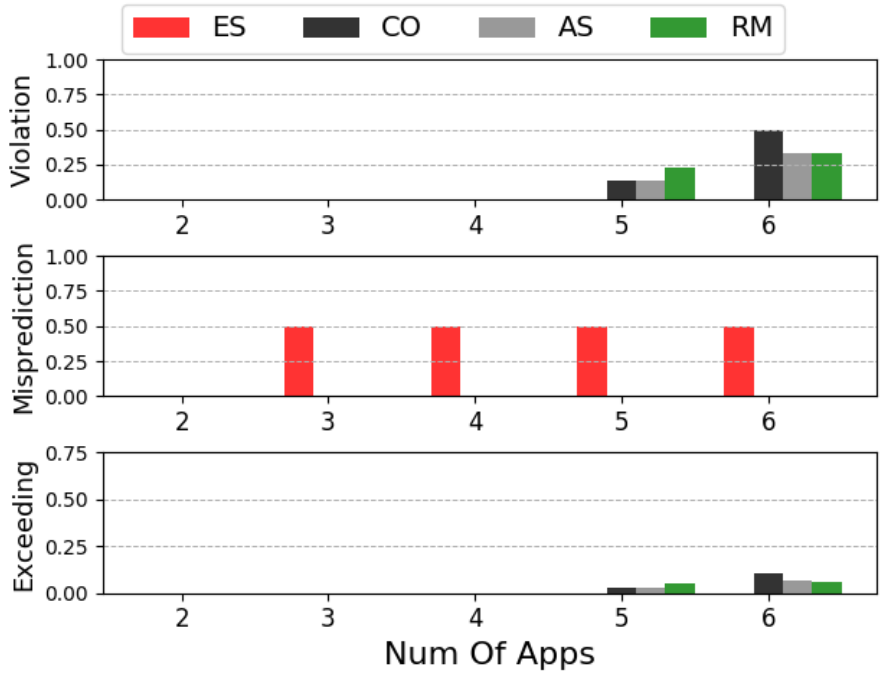


Figure 7.9: Static Configuration Selection Comparison with Enough Budget (scale=150%). ES: Equal Share, CO: Context Oblivious, AS: Aware Share, RM: Rapids-M, details was introduced in Section 5.5.

- For budget violation, 12.7% of executions using CO violate the budget when the system is not busy ($\leq$4 active apps), and 56.1% when busy. Rapids-M has a violation rate of 6.11% and 40.5% respectively. As a comparison, the designed oracle AS violates 3.5% and 26.6%. The violation by the oracle may be caused by either errors in the p-model, input dependencies, or minor system effects since no reconfiguration is performed.

- For those executions that violates the budget, Rapids-M exceeds the budget by an average of 6.6%. CO exceeds the budget by 11.45% on average and up to 50%.

- ES predicts the slowdown too aggressively and 40.74% of executions die because no feasible configuration can be provided resulting in misprediction.

Generally, budget violation could be "rescued" by reconfiguration during runtime. However, more frequent violations along with higher exceeding rate puts more pressure on the

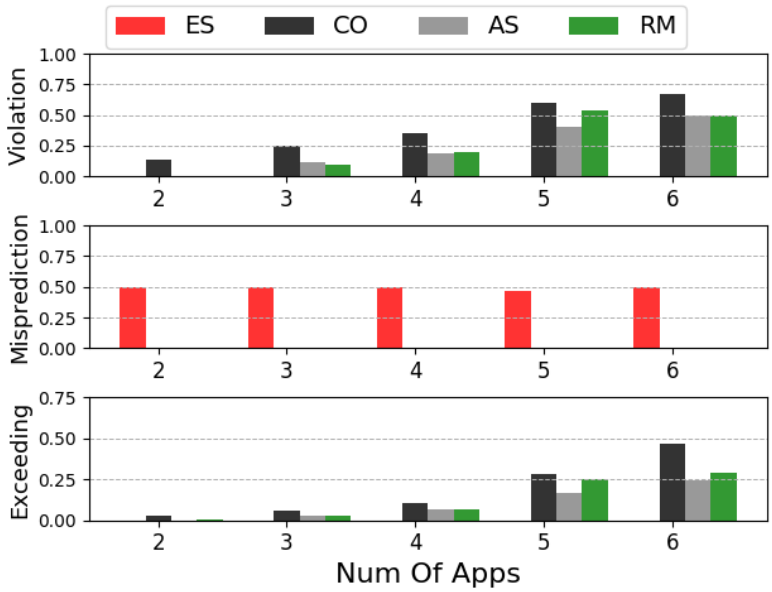


Figure 7.10: Static Selection Comparison with Moderate Budget (scale=100%)

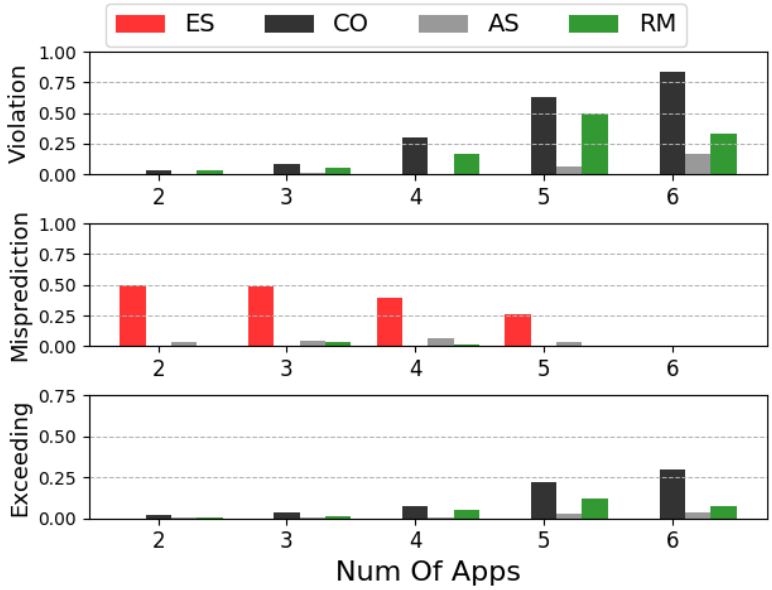


Figure 7.11: Static Selection Comparison with Limited Enough Budget (scale=80%)

dynamic reconfiguration. This experiment shows that Rapids-M lowers the violation rate by 33.9% compared to CO and has an average exceeding rate of 6.6%.

7.2.5 Reconfiguration Overheads

Rapids-M and existing approaches for single application configuration selection incur

overheads due to training of different models. These off-line training times can be substantial, sometimes in the order of days. The overhead is determined by the number of configurations and the length of per-configuration training for collecting meaningful results. For the six benchmark applications, Rapids-M took in average 10 hours to collect data for the p-models, ranging from $\leq$ 1 hour up to 28 hours for complicated applications. The model construction time with the data took 109 seconds on average, up to 6 minutes.

The evaluation methodology for the dynamic runtime overhead is based on a test environment that initiates different benchmark applications in different order with arbitrary delays. An overall resource budget is selected randomly within the low to high budget range of the applications. All quality metrics are normalized and weighted equally. Applications contact the server at the beginning and end of their execution, and at pre-set intervals to check whether a new bucket has been assigned by the server. The Rapids-M's server calls are synchronous, i.e., the application has to wait until the server returns an answer. If a new bucket is selected, the application has to perform a reconfiguration within the assigned bucket. The dynamic overhead is measured as the average time an application has to wait for an answer from the server and the local reconfiguration time if the bucket assignment changed. The average execution time overhead experienced by each application was less than 5% of their execution time. On average, every fifth call to the Rapids-M server resulted in a reconfiguration, with each reconfiguration performed in less than 0.2% of the overall execution time.

Overall, the average turn around time of requesting a bucket assignment is $\sim$ 201ms. The major part of the overhead comes from the online computation of the forward pass through M and P-Model. Therefore, I believe the overhead can be further reduced by selecting a more powerful server and/or a network with a lower latency. The overhead for local reconfigurations is small as in Rapids.

7.2.6 Summary of Key Results

In Rapids-M, configurations are grouped based on similarity of footprint and the slowdown behavior. Such bucketization strategy enables the reduction of search space to orders of magnitudes. In general, the number of buckets is defined by the nature of the application, the machine it runs on, and the threshold used in Algorithm 1. A stricter accuracy or distance threshold may improve the accuracy of the models, but may also introduce more buckets which contradicts the idea of space reduction. In my experiments, I set the threshold to be 6% and 4.0 as the accuracy and distance threshold. For applications, I discretized all continuous knobs with 10 settings. In this setting, the ratio of the number of buckets and configurations per application is on average 3.6%. By having 6 applications, the reduction on the combined search space is over 9 orders of magnitude. Such reduction enables to examine the globally optimal bucket combination of all six active applications even when they are all active.

For each of such bucket, Rapids-M constructs a specialized performance model (p-model) with average prediction error of $\leq$3%. Combined with the m-model, which has a less accurate prediction around 14% per feature, Rapids-M introduces $\leq$5% of quality loss compared to the oracle across different budget settings. This reveals that my approach is still feasible even the m-model is not perfect.

In the dynamic experiments, I created 15 different execution traces covering different number of maximum active applications. These traces are repeated using different budget settings. The result shows that over-estimating (ES) or under-estimating (CO) the slowdown can both cause severe impacts to the execution in terms of success rate and reject rate. These failed execution can hurt the overall quality. Compared with these approaches, Rapids-M makes configuration selections that lower the average budget violation rate by 33.9% with an average exceeding rate of 6.6%. At runtime, Rapids-M successfully finishes 22.75% more executions which results in a 1.52$\times$ improvement of output quality under high system loads.

At the early stage of developing Rapids-M, I found that most of the overhead are spent on the loading the models into memory. I optimized the implementation by making the learner a in-memory service. This optimization greatly reduces the overhead of Rapids-M. For my benchmark applications, the overhead of Rapids-M is within $\leq 1\%$ of the application's execution time.

Also, as a prototype system, the manager/learner are running on the same machine as the applications. The interference from the infrastructure of Rapids-M was not modeled in the P or m-model. I believe the overall performance of Rapids-M can be further improved, both in terms of overhead and application quality, after the system is deployed on a dedicated separate server.

CHAPTER 8
CONCLUSION AND FUTURE WORK

8.1 Conclusion

In this dissertation, I explored multiple aspects of approximation, including a novel development framework, offline training, user involvement, and online control. I pinpointed the limitations of existing approaches on each of these aspects and proposed Rapids and Rapids-M. I then conducted multiple experiments with various benchmarks, including applications ported from existing benchmark suites and newly built from scratch.

In Chapter-4, I showed that by exploring the application structural information, both developers and end-users could benefit from Rapids. First, I showed that Rapids allows developers to easily express insights like inter-knob constraints through the newly designed graph-based representation, KDG, with minimal extra efforts. Then I explained how the configuration space could be tailored by KDG through specifying dependencies. Such dependencies can help Rapids filter out invalid configurations, therefore shorten the training time by an average of 38.55%. This empowers the developers with more control over defining the application structure. Beyond that, I also introduced the *Representative Set*, which allowed a significantly faster retraining process (reduction up to 2 orders of magnitude) when applications got ported to unknown platforms. The *virtual knobs* enable users to customize the quality outcomes at a level of abstraction that makes sense to them. For our benchmark applications, Rapids manages to improve the preferred metric to up to 3.2× and 1.76× on average.

In Chapter-5, I introduced Rapids-M. To the best of my knowledge, Rapids-M is the first framework that enables effective and efficient approximation / configuration management across approximate applications that execute concurrently on the same target system.

Reconfigurations are initiated when applications start or finish their executions, or when system divergence is detected since such events typically result in changes in overall system loads. The global impact of a local configuration selection is predicted through local configuration system footprints, a global model that combines different footprints, and a performance model that considers the overall system environment. This local/global strategy allows the specification of configurations that can be considered equivalent, thereby Rapids-M significantly reduces the configuration search space of each application by up to two orders of magnitude through clustering configurations with similar behaviors into buckets, thereby allowing the exploration of the entire overall search space. For each of such bucket, Rapids-M constructs a specialized performance model with average prediction error of $\leq$3%, and a machine model for predicting overall system environments. Applications can be trained independently. Such a design along with the reduced search space makes Rapids-M scalable. Experimental results using six applications and different concurrent traces of application start and exit events show that Rapids-M is able to select globally optimal configurations. Compared to other possible approaches, Rapids-M makes configuration selections that lower the average budget violation rate by 33.9% with an average exceeding rate of 6.6%. At runtime, Rapids-M successfully finishes 22.75% more executions which results in a 1.52$\times$ improvement of output quality under high system loads. For our benchmark applications, the overhead of Rapids-M is within $\leq$1% of the application's execution time.

8.2 Future Work

All the discussed work targeted to make a more extensive audience accepts the approximation. However, there are still several potential improvements and directions worth to be explored.

In Rapids,

1. User Study: Currently, I myself act as the "developer", the "user", and also the de-

veloper of Rapids. This could lead to a subjective or biased evaluation of Rapids in terms of the easiness, expressiveness of the system. A user-study could be conducted on groups of developers to enrich the application collection and validate the usability of Rapids. Similarly, a user-study on end-users can also help to justify the necessity of supporting custom quality. I encourage continued efforts in these directions, to make approximation a more practical technique.

2. Advanced Model Construction: I use the *Representative Set* to enable a faster re-training process by constructing the cost model only based on observations of a few selected configurations. However, the model recovered from limited observations usually suffers from a slightly lower accuracy. It suggests that there is an opportunity to refine the model based on future observations, for example through *Reinforced Learning*. Or even more aggressively, gradually construct the cost model on new machines directly from the scratch using reinforced learning. I believe such incorporation of modern machine learning approach could make Rapids even more powerful in the case of distributing applications to heterogeneous platforms.

3. Larger Deployment: KDG is designed as a compact representation of cooperating knobs. These knobs are typically variables or code-paths in an application. However, knobs can encode more complex services. For example, a particular node in IOT [115], or an edge device in edge computing clouds [116]. For these larger use-cases, KDG can be used to deal with problems where the constrained resources are bandwidth, latency, computing power peak, etc.

In Rapids-M,

1. Domain Specific Models: The system footprint prediction (m-model) has a not-so-great prediction accuracy due to the high variance in the low level hardware metrics. However, these metrics can highly affect the application performance. In the current Rapids-M, m-model was constructed by choosing the best model from a pool of

off-the-shelf models. A more dedicated model focusing on the specific problem of Rapids-M is facing could be a more impactful direction towards making Rapids-M much more reliable.

2. High Intensity Evaluation: Currently, Rapids-M was tested on a load that has a maximum of 6 active applications. A more intense evaluation can be performed with 20+ active applications. Increasing the intensity might expose new and more interesting problems and open up new areas that worth investigating.

Lastly, my current results were collected through experiments using a limited number of applications deployed on a few platforms. A broader deployment on various machines ranging from more constrained platforms (e.g., wearable devices) to much more powerful platforms (e.g., data centers) with different applications could be a huge next step to bring Rapids, Rapids-M, and the concept of approximation to a wider audience.

BIBLIOGRAPHY

[1] D. Seal, *ARM Architecture Reference Manual*, 2nd. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2000, ISBN: 0201737191.

[2] G. Inc. (2019). Dark Theme Development Guides. https://developer.android.com/guide/topics/ui/look-and-feel/darktheme.

[3] A. Inc. (2019). Dark Mode Development Guides. https://developer.apple.com/design/human-interface-guidelines/ios/visual-design/dark-mode/.

[4] D. J. Sorin, M. D. Hill, and D. A. Wood, "A primer on memory consistency and cache coherence," *Synthesis Lectures on Computer Architecture*, vol. 6, no. 3, pp. 1–212, 2011.

[5] R. Bridson, *Fluid simulation for computer graphics*. AK Peters/CRC Press, 2015.

[6] H. Hoffmann, "JouleGuard: Energy guarantees for appoximate applications," in *Symposium on Operating Systems Principles (SOSP)*, Monterey, CA, 2015.

[7] I. Constandache, S. Gaonkar, M. Sayler, R. R. Choudhury, and L. Cox, "Enloc: Energy-efficient localization for mobile phones," in *INFOCOM 2009, IEEE*, IEEE, 2009, pp. 2716–2720.

[8] B. Chen and D. Pompili, "Uncertainty-aware localization solution for under-ice autonomous underwater vehicles," in *Proceedings of the 9th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, Seoul, Korea, 2012, pp. 308–316.

[9] A. Alvarez, R. Stoner, and A. Maguer, "Performance of pumped and un-pumped CTDs in an underwater glider," in *OCEANS 2013 IEEE - San Diego*, 2013, pp. 1–5.

[10] I. Gori, R. Bianchini, S. Nagarakatte, and T. Nguyen, "ApproxHadoop: Bringing approximations to MapReduce frameworks," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Istanbul, Turkey, 2015, pp. 383–397.

[11] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, "MCDNN: An approximation-based execution framework for deep stream processing under resource constraints," in *Proceedings of the 14th Annual*

International Conference on Mobile Systems, Applications, and Services, ser. MobiSys '16, Singapore, Singapore: ACM, 2016, pp. 123–136.

[12] X. Sui, A. Lenharth, D. S. Fussell, and K. Pingali, "Proactive control of approximate programs," in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '16, Atlanta, Georgia, USA: ACM, 2016, pp. 607–621, ISBN: 978-1-4503-4091-5.

[13] R. Xu, J. Koo, R. Kumar, P. Bai, S. Mitra, S. Misailovic, and S. Bagchi, "Videochef: Efficient approximation for streaming video processing pipelines," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, 2018, pp. 43–56.

[14] C. Bienia, "Benchmarking modern multiprocessors," PhD thesis, Princeton University, 2011.

[15] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe lsh: Efficient indexing for high-dimensional similarity search," in *Proceedings of the 33rd international conference on Very large data bases*, VLDB Endowment, 2007, pp. 950–961.

[16] Y. Rubner, C. Tomasi, and L. J. Guibas, "The earth mover's distance as a metric for image retrieval," *International journal of computer vision*, vol. 40, no. 2, pp. 99–121, 2000.

[17] J. Bar-Ilan, M. Mat-Hassan, and M. Levene, "Methods for comparing rankings of search engine results," *Computer networks*, vol. 50, no. 10, pp. 1448–1463, 2006.

[18] H. Hoffmann, J. Eastep, M. D. Santambrogio, J. E. Miller, and A. Agarwal, "Application heartbeats: A generic interface for specifying program performance and goals in autonomous computing environments," in *Proceedings of the 7th international conference on Autonomic computing*, ACM, 2010, pp. 79–88.

[19] X. Sui, A. Lenharth, D. S. Fussell, and K. Pingali, "Proactive control of approximate programs," *ACM SIGOPS Operating Systems Review*, vol. 50, no. 2, pp. 607–621, 2016.

[20] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard, "Dynamic knobs for responsive power-aware computing," in *ASPLOS '11*, Newport Beach, California, USA, 2011.

[21] N. Mishra, J. D. Laferty, and H. Hofmann, "Caloree: Learning control for predictable latency and low energy," in *ASPLOS '18*, Williamsburg, Virginia, USA, 2018.

[22] A. Farrell and H. Hoffmann, "MEANTIME: Achieving both minimal energy and timeliness with approximate computing," in *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, Denver, CO, Jun. 2016, pp. 421–435.

[23] Q. Huynh-Thu and M. Ghanbari, "The accuracy of PSNR in predicting video quality for different video scenes and frame rates," *Telecommunication Systems*, vol. 49, no. 1, pp. 35–48, 2012.

[24] C. Lattner and V. Adve, "Llvm: A compilation framework for lifelong program analysis & transformation," in *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization*, ser. CGO '04, Palo Alto, California: IEEE Computer Society, 2004, pp. 75–, ISBN: 0-7695-2102-9. [Online]. Available: `http://dl.acm.org/citation.cfm?id=977395.977673`.

[25] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. Rinard, "Dynamic knobs for responsive power-aware computing," in *Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems*, ser. ASPLOS, Newport Beach, California, USA: ACM, 2011, pp. 199–212, ISBN: 978-1-4503-0266-1.

[26] S. Sidiroglou, S. Misailovic, H. Hoffmann, and M. Rinard, "Managing performance vs. accuracy trade-offs with loop perforation," in *ESEC/FSE'11*, Szeged, Hungary, 2011.

[27] Q. Inc. (2015). Tredn Power Profiler. https://developer.qualcomm.com/software/trepn-power-profiler.

[28] J. Ansel, C. Chan, Y. L. Wong, M. Olszewski, Q. Zhao, A. Edelman, and S. Amarasinghe, "Petabricks: A language and compiler for algorithmic choice," in *Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '09, Dublin, Ireland: ACM, 2009, pp. 38–49.

[29] M. Dong and L. Zhong, "Chameleon: A color-adaptive web browser for mobile OLED displays," in *Proceedings of the 9th International Conference on Mobile System, Applications, and Services*, ser. MobiSys '11, Washington D.C., USA, 2011, pp. 85–98.

[30] J. Park, H. Esmaeilzadeh, X. Zhang, M. Naik, and W. Harris, "Flexjava: Language support for safe and modular approximate programming," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ACM, 2015, pp. 745–757.

[31] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "EnerJ: Approximate data types for safe and general low-power computation," in *ACM Conference on Programming Language Design and Implementation (PLDI)*, San Jose, California, USA, 2011.

[32] S. Mittal, "A survey of techniques for approximate computing," *ACM Computing Surveys (CSUR)*, vol. 48, no. 4, p. 62, 2016.

[33] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," in *Proceedings of the 50th Annual Design Automation Conference*, ACM, 2013, p. 113.

[34] M. A. Anam, P. N. Whatmough, and Y. Andreopoulos, "Precision-energy-throughput scaling of generic matrix multiplication and discrete convolution kernels via linear projections," in *The 11th IEEE Symposium on Embedded Systems for Real-time Multimedia*, IEEE, 2013, pp. 21–30.

[35] C. Alvarez, J. Corbal, and M. Valero, "Fuzzy memoization for floating-point multimedia applications," *IEEE Transactions on Computers*, vol. 54, no. 7, pp. 922–927, 2005.

[36] S. Byna, J. Meng, A. Raghunathan, S. Chakradhar, and S. Cadambi, "Best-effort semantic document search on gpus," in *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, ACM, 2010, pp. 86–93.

[37] I. K. Center. (). Miqcp: Mixed integer programs with quadratic terms in the constraints, [Online]. Available: `https://www.ibm.com/support/knowledgecenter/SSSA5P_12.7.1/ilog.odms.cplex.help/CPLEX/UsrMan/topics/discr_optim/mip_quadratic/03_introMIQCP.html`.

[38] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Architecture support for disciplined approximate programming," in *ACM SIGPLAN Notices*, ACM, vol. 47, 2012, pp. 301–312.

[39] D. Mahajan, A. Yazdanbakhsh, J. Park, B. Thwaites, and H. Esmaeilzadeh, "Prediction-based quality control for approximate accelerators," in *Second Workshop on Approximate Computing Across the System Stack, WACAS*, 2015.

[40] L. McAfee and K. Olukotun, "Emeuro: A framework for generating multi-purpose accelerators via deep learning," in *2015 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, IEEE, 2015, pp. 125–135.

[41] A. K. Mishra, R. Barik, and S. Paul, "Iact: A software-hardware framework for understanding the scope of approximate computing," in *Workshop on Approximate Computing Across the System Stack (WACAS)*, 2014, p. 52.

[42] V. Vassiliadis, K. Parasyris, C. Chalios, C. D. Antonopoulos, S. Lalis, N. Bellas, H. Vandierendonck, and D. S. Nikolopoulos, "A programming model and runtime system for significance-aware energy-efficient computing," in *ACM SIGPLAN Notices*, ACM, vol. 50, 2015, pp. 275–276.

[43] M. J. Wolfe, *Optimizing Supercompilers for Supercomputers*. Cambridge, MA, USA: MIT Press, 1990, ISBN: 0262730820.

[44] H. Zima and B. Chapman, *Supercompilers for Parallel and Vector Computers*. New York, NY, USA: ACM, 1991, ISBN: 0-201-17560-6.

[45] K. Kennedy and J. R. Allen, *Optimizing Compilers for Modern Architectures: A Dependence-based Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002, ISBN: 1-55860-286-0.

[46] S. Wang, C. Li, H. Hoffmann, S. Lu, W. Sentosa, and A. I. Kistijantoro, "Understanding and auto-adjusting performance-sensitive configurations," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ACM, 2018, pp. 154–168.

[47] J. Park, X. Zhang, K. Ni, H. Esmaeilzadeh, and M. Naik, "Expax: A framework for automating approximate programming," Georgia Institute of Technology, Tech. Rep., 2014.

[48] R. St Amant, A. Yazdanbakhsh, J. Park, B. Thwaites, H. Esmaeilzadeh, A. Hassibi, L. Ceze, and D. Burger, "General-purpose code acceleration with limited-precision analog computation," *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3, pp. 505–516, 2014.

[49] J. Bornholt, T. Mytkowicz, and K. McKinley, "Uncertain<t>: A first-order type for uncertain data," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Salt Lake City, UT, 2014, pp. 51–66.

[50] J. Ansel, Y. L. Wong, C. Chan, M. Olszewski, A. Edelman, and S. Amarasinghe, "Language and compiler support for auto-tuning variable-accuracy algorithms," in *International Symposium on Code Generation and Optimization (CGO 2011)*, IEEE, 2011, pp. 85–96.

[51] A. Yazdanbakhsh, D. Mahajan, B. Thwaites, J. Park, A. Nagendrakumar, S. Sethuraman, K. Ramkrishnan, N. Ravindran, R. Jariwala, A. Rahimi, *et al.*, "Axilog: Language support for approximate hardware design," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, EDA Consortium, 2015, pp. 812–817.

[52] E. Incerto, M. Tribastone, and C. Trubiani, "Software performance self-adaptation through efficient model predictive control," in *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, IEEE Press, 2017, pp. 485–496.

[53] A. L. Cervantes, O. E. Agamennoni, and J. L. Figueroa, "A nonlinear model predictive control system based on wiener piecewise linear models," *Journal of Process Control*, vol. 13, no. 7, pp. 655–666, 2003.

[54] S. Burer and A. N. Letchford, "Non-convex mixed-integer nonlinear programming: A survey," *Surveys in Operations Research and Management Science*, vol. 17, no. 2, pp. 97–106, 2012.

[55] R. Hegde and N. R. Shanbhag, "Energy-efficient signal processing via algorithmic noise-tolerance," in *Proceedings. 1999 International Symposium on Low Power Electronics and Design (Cat. No. 99TH8477)*, IEEE, 1999, pp. 30–35.

[56] A. Sampson, J. Nelson, K. Strauss, and L. Ceze, "Approximate storage in solid-state memories," *ACM Transactions on Computer Systems (TOCS)*, vol. 32, no. 3, p. 9, 2014.

[57] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Architectural support for disciplined approximate programming," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, London, UK, 2012.

[58] J. Miguel, M. Badr, and N. Jerger, "Load value approximation," in *International Symposium on Microarchitectures*, Cambridge, UK, 2014, pp. 127–139.

[59] D. Khudia, B. Zamirai, M. Samadi, and S. Mahlke, "Rumba: An online quality management system for approximate computing," in *International Symposium on Computer Architecture (ISCA)*, Portland, OR, 2015, pp. 554–566.

[60] S. Misailovic, M. Carbin, S Achour, Z. Qi, and M. Rinard, "Chisel: Reliability- and accuracy-aware optimizations of approximate computational kernels," in *International Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA)*, Portland, OR, 2014, pp. 309–328.

[61] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in *2011 24th Internatioal Conference on VLSI Design*, IEEE, 2011, pp. 346–351.

[62] A. Rahimi, A. Ghofrani, K.-T. Cheng, L. Benini, and R. K. Gupta, "Approximate associative memristive memory for energy-efficient gpus," in *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, EDA Consortium, 2015, pp. 1497–1502.

[63] Y. Fang, H. Li, and X. Li, "Softpcm: Enhancing energy efficiency and lifetime of phase change memory in video applications via approximate write," in *2012 IEEE 21st Asian Test Symposium*, IEEE, 2012, pp. 131–136.

[64] A. Ranjan, S. Venkataramani, X. Fong, K. Roy, and A. Raghunathan, "Approximate storage for energy efficient spintronic memories," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, IEEE, 2015, pp. 1–6.

[65] J. Kramer and J. Magee, "Self-managed systems: An architectural challenge," in *2007 Future of Software Engineering*, IEEE Computer Society, 2007, pp. 259–268.

[66] R. De Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel, *et al.*, "Software engineering for self-adaptive systems: A second research roadmap," in *Software Engineering for Self-Adaptive Systems II*, Springer, 2013, pp. 1–32.

[67] J. Andersson, L. Baresi, N. Bencomo, R. de Lemos, A. Gorla, P. Inverardi, and T. Vogel, "Software engineering processes for self-adaptive systems," in *Software Engineering for Self-Adaptive Systems II*, Springer, 2013, pp. 51–75.

[68] R. Tawhid and D. Petriu, "Integrating performance analysis in the model driven development of software product lines," in *International Conference on Model Driven Engineering Languages and Systems*, Springer, 2008, pp. 490–504.

[69] P. Jamshidi, M. Velez, C. Kästner, N. Siegmund, and P. Kawthekar, "Transfer learning for improving model predictions in highly configurable software," in *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2017 IEEE/ACM 12th International Symposium on*, IEEE, 2017, pp. 31–41.

[70] V. Dalibard, M. Schaarschmidt, and E. Yoneki, "Boat: Building auto-tuners with structured bayesian optimization," in *Proceedings of the 26th International Conference on World Wide Web*, International World Wide Web Conferences Steering Committee, 2017, pp. 479–488.

[71] N. Mishra, J. D. Lafferty, and H. Hoffmann, "Esp: A machine learning approach to predicting application interference," in *Proceedings of the International Conference on Autonomic Computing*, ser. ICAC, Columbus, Ohio, USA, 2017.

[72] J. Knight and N. Leveson, "An experimental evaluation of the assumption of independence in multiversion programming," *IEEE Transactions on Software Enginnering*, vol. 12, no. 1, pp. 96–109, 1986.

[73] W. Baek and T. M. Chilimbi, "Green: A framework for supporting energy-conscious programming using controlled approximation," in *Proceedings of the 2010 ACM SIGPLAN conference on Programming language design and implementation*, ser. PLDI '10, Toronto, Ontario, Canada: ACM, 2010, pp. 198–209, ISBN: 978-1-4503-0019-3. DOI: 10.1145/1806596.1806620. [Online]. Available: http://doi.acm.org/10.1145/1806596.1806620.

[74] H. Zhang and H. Hoffmann, "Maximizing performance under a power cap: A comparison of hardware, software, and hybrid techniques," in *ASPLOS'16*, Atlanta, Georgia, USA, 2016.

[75] A. Gordon, T. Henzinger, A. Nori, and S. Rajamani, "Probabilistic programming," in *International Conference on Software Engineering (ICSE)*, Hyderabad, India, 2014.

[76] N. Ramsey and A. Pfeffer, "Stochastic lambda calculus and monads of probability distributions," in *ACM Symposium on Principles of Programming Languages (POPL)*, Portland, OR, 2002, pp. 154–165.

[77] A. Pfeffer, "IBAL: A probabilistic rational programming language," in *International Joint Conference on Artificial Intelligence (IJCAI)*, Seattle, WA, 2001, pp. 733–740.

[78] J. Borgstrom, A. Gordon, M. Greenberg, J. Margetson, and J. V. Gael, "Meaure transformer semantics for bayesian machine learning," in *European Symposium on Programming (ESOP)*, Saarbrucken, Germany, 2011.

[79] W. Gilks, A. Thomas, and D. Spiegelhalter, "A language and program for complex bayesian modelling," *Journal of the Royal Statistical Society, Series D (The Statistician)*, vol. 43, no. 1, pp. 169–177, 1994.

[80] N. Goodman, V. Mansinghka, D. Roy, K. Bonawitz, and J. Tenenbaum, "Church: A language for generative models," in *Conference in Uncertainty in Artificial Intelligence (UAI)*, Helsinki, Finland, 2008, pp. 220–229.

[81] S. Park, F. Pfenning, and S. Thrun, "A probabilistic language based on sampling functions," in *ACM Symposium on Principles of Programming Languages (POPL)*, Long Beach, CA, 2005, pp. 171–182.

[82] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy, "The aqua approximate query answering system," in *ACM Sigmod Record*, ACM, vol. 28, 1999, pp. 574–576.

[83] K. Zeng, S. Gao, B. Mozafari, and C. Zaniolo, "The analytical bootstrap: A new method for fast error estimation in approximate query processing," in *ACM SIGMOD'14*, Snowbird, UT, 2014, pp. 277–288.

[84] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *Symposium on Operating System Design (OSDI)*, San Francisco, CA, 2004.

[85] N. Laptev, K. Zeng, and C. Zaniolo, "Early accurate results for advanced analytics on mapreduce," *Proceedings of the VLDB Endowment*, vol. 5, no. 10, pp. 1028–1039, 2012.

[86] M. Carbin, D. Kim, S. Misailovic, and M. C. Rinard, "Proving acceptability properties of relaxed nondeterministic approximate programs," in *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '12, Beijing, China: ACM, 2012, pp. 169–180, ISBN: 978-1-4503-1205-9. DOI: 10.1145/2254064.2254086. [Online]. Available: http://doi.acm.org/10.1145/2254064.2254086.

[87] S. Chaudhuri, S. Gulwani, R. Lublinerman, and S. Navidpour, "Proving programs robust," in *ESEC-FSE'11*, 2011.

[88] I. Akturk, K. Khatamifard, and U. R. Karpuzcu, "On quantification of accuracy loss in approximate computing," in *Workshop on Duplicating, Deconstructing and Debunking (WDDD)*, vol. 15, 2015.

[89] N. Chinchor, "Muc-4 evaluation metrics," in *Proceedings of the 4th conference on Message understanding*, Association for Computational Linguistics, 1992, pp. 22–29.

[90] M. H. Santriaji and H. Hoffmann, "Grape: Minimizing energy for gpu applications with performance requirements," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, IEEE, 2016, pp. 1–13.

[91] M. Maggio, A. V. Papadopoulos, A. Filieri, and H. Hoffmann, "Self-adaptive video encoder: Comparison of multiple adaptation strategies made simple," in

2017 IEEE/ACM 12th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), IEEE, 2017, pp. 123–128.

[92] ——, "Automated control of multiple software goals using multiple actuators," in *Proceedings of the 2017 11th joint meeting on foundations of software engineering*, ACM, 2017, pp. 373–384.

[93] M. M. Alves and L. M. d. A. Drummond, "A quantitative model for predicting cross-application interference in virtual environments," *arXiv preprint arXiv:1610.04309*, 2016.

[94] R. Nathuji, A. Kansal, and A. Ghaffarkhah, "Q-clouds: Managing performance interference effects for qos-aware clouds," in *Proceedings of the 5th European conference on Computer systems*, ACM, 2010, pp. 237–250.

[95] S.-H. Lim, J.-S. Huh, Y. Kim, G. M. Shipman, and C. R. Das, "D-factor: A quantitative model of application slow-down in multi-resource shared systems," *ACM SIGMETRICS Performance Evaluation Review*, vol. 40, no. 1, pp. 271–282, 2012.

[96] M. Samadi and S. Mahlke, "Cpu-gpu collaboration for output quality monitoring," in *1st Workshop on Approximate Computing Across the System Stack*, 2014, pp. 1–3.

[97] P. Düben, S. Yenugula, J. Augustine, K Palem, J. Schlachter, C. Enz, T. Palmer, *et al.*, "Opportunities for energy efficient computing: A study of inexact general purpose processors for high-performance and big-data applications," in *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2015, pp. 764–769.

[98] B. Shim, S. R. Sridhara, and N. R. Shanbhag, "Reliable low-power digital signal processing via reduced precision redundancy," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 5, pp. 497–510, 2004.

[99] A. Rahimi, L. Benini, and R. K. Gupta, "Spatial memoization: Concurrent instruction reuse to correct timing errors in simd architectures," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 60, no. 12, pp. 847–851, 2013.

[100] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn, "Flikker: Saving dram refresh-power through critical data partitioning," *ACM SIGPLAN Notices*, vol. 47, no. 4, pp. 213–224, 2012.

[101] G. Hu, S. Rigo, D. Zhang, and T. Nguyen, "Approximation with error bounds in spark," in *2019 IEEE 27th International Symposium on Modeling, Analysis, and*

Simulation of Computer and Telecommunication Systems (MASCOTS), IEEE, 2019, pp. 61–73.

[102] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[103] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[104] C. Z. Mooney, *Monte carlo simulation*. Sage Publications, 1997, vol. 116.

[105] M. Rinard, "Probabilistic accuracy bounds for fault-tolerant computations that discard tasks," in *Proceedings of the 20th annual international conference on Supercomputing*, ACM, 2006, pp. 324–334.

[106] E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt, and J. M. Ogden, "Pyramid methods in image processing," *RCA engineer*, vol. 29, no. 6, pp. 33–41, 1984.

[107] N. Corporation, *Nvidia jetson tx1 developer kit*, 2017. [Online]. Available: `http://www.nvidia.com/object/embedded-systems-dev-kits-modules.html`.

[108] Z. Wang, L. Lu, and A. C. Bovik, "Video quality assessment based on structural distortion measurement," *Signal processing: Image communication*, vol. 19, no. 2, pp. 121–132, 2004.

[109] M. I. P. Tool. (2017). Gurobi optimizer 7.25, [Online]. Available: `http://www.gurobi.com/products/gurobi-optimizer`.

[110] A. Waterland. (2002). The stress workload generator, [Online]. Available: `https://people.seas.harvard.edu/~apw/stress/`.

[111] Intel. (2012). Intel performance counter monitor - a better way to measure cpu utilization, [Online]. Available: `www.intel.com/software/pcm`.

[112] F. Chollet, *Keras*, `https://github.com/fchollet/keras`, 2015.

[113] R. C. Dubes and A. K. Jain, *Algorithms for clustering data*, 1988.

[114] N. Mishra, J. D. Lafferty, and H. Hoffmann, "Esp: A machine learning approach to predicting application interference," in *2017 IEEE International Conference on Autonomic Computing (ICAC)*, IEEE, 2017, pp. 125–134.

[115] K. Ashton *et al.*, "That 'internet of things' thing," *RFID journal*, vol. 22, no. 7, pp. 97–114, 2009.

[116] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, 2016.

[117] L. Liu. (2019). RAPIDS Online Portal, [Online]. Available: https://niuye8911.github.io/rapidlib-linux.

[118] ——, (2019). RAPIDS Repository on Github, [Online]. Available: https://github.com/niuye8911/rapidlib-linux.

[119] M. I. P. Tool. (1999). Lp_solve 5.5, [Online]. Available: http://lpsolve.sourceforge.net/5.5.

APPENDIX A
TRAINING TUNING

The local infrastructure of Rapids and Rapids-M are shared in multiple aspects, including 1) KDG specification, 2) code instrumentation, and 3) training framework. A detailed step-by-step guide of developing an approximate application can be found online through the web portal [117] of Rapids. The source code of Rapids trainer can be downloaded on RAPIDS-Repo [118]. In this chapter, I introduce the training procedure and some key parameters to tune the training process. Then, I will describe how developers can tune the training on a per-application level.

A.1 4-Stage Training Process Tuning

The entry point of the training procedure is the script named "rapid.py". The training process for Rapids or Rapids-M can be partitioned into 4 parts:

1. KDG Structure Generation: generates the skeleton of KDG structure.
2. Training: profiles the application execution time and quality.
3. KDG Weight Population: derives the weights in KDG from the data collected in training.
4. RS calculation: Calculate the *Representative Set* from the KDG.

All these four stages have tunable parameters to fine tune the procedure to developers' need. These parameters are listed as constants at the beginning of the script.

- KDG **Structure Generation:** In this step, the script takes the KDG specification file and generates the structure of KDG in the XML format with all knobs attributes encoded. If

there exists any continuous knob in the KDG, Rapids will discretize the application to 10 discrete settings. Developers can change the granularity by giving different values to the constant $GRANULARITY$.

- Training: In this step, the script will train the application using the valid configurations determined by the KDG.

For Rapids, each configuration will be trained once and collect the observation of its cost and quality. The result for cost observations will be stored in `RAPIDS_ROOT\modelConstr\Rapids\APP_NAME\APP_NAME-cost.fact`. The file for quality will be under the same directory named `APP_NAME-quality.fact`.

For Rapids-M, more data are required for M and p-model construction. Therefore, the system footprint for each configuration will also be collected during the training. Besides, each configuration will also be trained against a "stresser" to observe the overall footprint and the corresponding slowdown. As a result, the slowdown information will be stored in three separate files:

- `APP_NAME-sys.csv`: the footprint for each configuration when running alone (for bucket construction)
- `APP_NAME-perf.csv`: the overall footprint and the slowdown when running with stresser (for p-model training)
- `APP_NAME-mperf.csv`: the footprint of the stresser and the application when running alone, and the overall footprint. (for m-model construction)

For each configuration, the number of "stresser"s to train each configuration with can be changed by updating the constant $NUM\_OF\_FIXED\_ENV$ (default = 30).

- KDG **Weight Population:** In this step, Rapids will use the collected observations to derive the weights for each node or the functions representing the continuous knobs. If the application contains continuous knobs, the KDG will represent the knob as a series

of segments. Developers can change the criterion on two aspects when determining the segmentation granularity (partition depth):

- MAX_PARTITION (default=5): the max depth of partitioning the knob value range. The number of the discretized point in the range is 2^n when n is the depth.
- PARTITION_ERR_THRESHOLD (default = 0.05) : the target error threshold to determine whether the partition depth is good enough.

The result of this step is a fully populated cost KDG for a quality KDG for the default overall quality and all the sub-metrics. The result KDG will be named as `APP_NAME-cost.rsdg` under the same directory as other outputs. For quality KDGs, they will be named `APP_NAME-mv[0/1/2...n].rsdg` , where 0 stands for the default quality and [1∼n] represent the sub-metrics.

- *RS* Calculation: In this step, Rapids will construct the Selection-based Representative Set. Developers can change the constant $RS\_THRESHOLD$ (default = 0.05) to define the error threshold for RS construction.

A.2 Per-Application Training Tuning

In this section, I describe how developers can customize the training process for each individual application. For each application, the developer can provide the training script with a configuration file containing the application-level configuration. Figure A.1 shows the template configuration file and Table A.1 explains the fields developers can configure.

```
{
  "appName": "APP_NAME",
  "appPath": "PATH/TO/YOUR/EXECUTABLE",
  "appMet": "PATH/TO/THE/PYTHON/APP_METHODS.py",
  "appDep": "PATH/TO/THE/RSDG/SPECIFICATION_FILE
    ",
  "withQoS": 1,
  "RS": 0,
  "qosRun": 0,
  "overheadRun": 0,
  "withSys": 0,
  "withPerf": 0,
  "withMModel": 0
}
```

Figure A.1: Per-Application Training Configuration

Table A.1: Per-Application Training Configuration Fields Explanation

parameter	Description
withQos	collect quality value
RS	construct KDG
withSys	collect footprint
withPerf	collect slowdown
withMModel	collect “Stesser” footprint
qosRun	perform quality evaluation as in Figure 4.5
overheadRun	perform overhead evaluation as in Figure 7.5 and Figure 7.6a

APPENDIX B

RUNTIME TUNING

After training the application, a run_config file will be generated for each application containing some key information about the application to be used in the runtime. Figure B.1 shows the skeleton of the configuration file. Developers can use the file to configure the execution of the application, and are highly encouraged to design their own GUI to expose certain fields in the file to the user.

Table B.1 explains the key fields in the file. The first column explains the designed purpose of particular fields. For example, "budget" and "preferences" (current implementations for the "Virtual Knobs") are supposed to be tuned by users. Developers are highly encouraged to implement their own GUI to accept users input for these fields and update in the file accordingly for Rapids or Rapids-M runtime to finalize the quality model.

Table B.1: Runtime Control Configuration Field Explanation

Field Name		Description
Expose to user	budget	budget in seconds
	preferences	preferences to sub-metrics
Used by runtime	rapidScript	Finalize the quality KDG according to "preferences"
	appMet	
For developers	UNIT_PER_CHECK	monitor frequency as in Figure 7.1.1
	OFFLINE_SEARCH	use "FULL"[1] as in Section 7.1.1
	REMOTE	use remote[2] server for optimization
	GUROBI	formalize optimization problem in Gurobi[3]
	MISSION_LOG	log the re-configuration activity
	DEBUG	verbose information[4]
	RAPID_M	use RAPIDS_M server
	RUSH_TO_END	use RUSH_TO_END method as in Section 5.5.1

1. "FULL" is the black box approach and it requires the ".cost" if selected
2. REMOTE is a MUST in Rapids-M *and optional in* Rapids
3. Rapids *also supports the LP_SOLVE [119] problem format but LP_SOLVE typically has much higher overhead than GUROBI*
4. verbose information includes overhead per re-configuration, working progress, etc

```
{
  "basic": {
    "app_name": "APP_NAME",
    "defaultXML": "PATH/TO/KDG/WITH/DEFAULT/
       QUALITY"
  },
  "mission": {
    "budget": YOUR_BUDGET,
    "UNIT_PER_CHECK": 10,
    "OFFLINE_SEARCH": false,
    "REMOTE": true,
    "GUROBI": true,
    "RAPID_M": true,
    "MISSION_LOG": true,
    "DEBUG": true,
    "RUSH_TO_END": false,
    "POWER_SAVING": false
  },
  "appMet": "PATH/TO/PYTHON/APP_METHODS.py",
  "rapidScript": "PATH/TO/rapid.py",
  "preferences": [pref\_to\_sub\_metrics]
}
```

Figure B.1: Application Runtime Configuration File