Global Cost/Quality Management across Multiple Applications

Liu Liu
Rutgers University
New Brunswick, New Jersey, USA

Sibren Isaacman
Loyola University Maryland
Baltimore, Maryland, USA

Ulrich Kremer
Rutgers University
New Brunswick, New Jersey, USA

ABSTRACT

Approximation is a technique that optimizes the balance between application outcome quality and its resource usage. Trading quality for performance has been investigated for single application scenarios, but not for environments where multiple approximate applications may run concurrently on the same machine, interfering with each other by sharing machine resources. Applying existing, single application techniques to this multi-programming environment may lead to configuration space size explosion, or result in poor overall application quality outcomes.

Our new RAPID-M system is the first cross-application configuration management framework. It reduces the problem size by clustering configurations of individual applications into local "similarity buckets". The global cross-applications configuration selection is based on these local bucket spaces. RAPID-M dynamically assigns buckets to applications such that overall quality is maximized while respecting individual application cost budgets. Once assigned a bucket, reconfigurations within buckets may be performed locally with minimal impact on global selections. Experimental results using six configurable applications show that even large configuration spaces of complex applications can be clustered into a small number of buckets, resulting in search space size reductions of up to 9 orders of magnitude for our six applications. RAPID-M constructs performance cost models with an average prediction error of ≤3%. For our application execution traces, RAPID-M dynamically selects configurations that lower the budget violation rate by 33.9% with an average budget exceeding rate of 6.6% as compared to other possible approaches. RAPID-M successfully finishes 22.75% more executions which translates to a 1.52× global output quality increase under high system loads. The overhead of RAPID-M is within ≤1% of application execution times.

CCS CONCEPTS

• **Software and its engineering → Development frameworks and environments**; **Software configuration management and version control systems**.

KEYWORDS

Approximate Computing, Global Configuration Management, Performance Prediction, Multi-Programming

ESEC/FSE '20, November 8–13, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-7043-1/20/11…$15.00
https://doi.org/10.1145/3368089.3409721

ACM Reference Format:
Liu Liu, Sibren Isaacman, and Ulrich Kremer. 2020. Global Cost/Quality Management across Multiple Applications. In *Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '20), November 8–13, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3368089.3409721

1 INTRODUCTION

A significant number of important applications can be configured / approximated to trade off application outcome quality for execution time reduction. This property can be crucial when applications are executed on resource constrained devices under soft execution deadlines, including edge devices and mobile phones. Since multiple applications execute on the same hardware platform, they interfere with each other through resource sharing (e.g., memory hierarchy, CPU cores, buses, communication networks). Further, different configurations may have different resource footprints and different quality outcomes, making optimal or near optimal selection of configurations across all applications challenging. In previous work, single application performance/cost models are constructed by applying machine learning strategies to 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 single application input dependencies or runtime noise, but not interference from other applications.

This paper discusses the design and evaluation of RAPID-M **R**econfiguration, **A**pproximation, **P**references, **I**mplementation, and **D**ependencies for **M**ulti-Programming, a new cross-application configuration management framework which uses a novel local-global-local approach that allows a systematic exploration of the combined configuration search spaces of all active applications. To the best of our knowledge, our work is the first to address this problem. The main steps of RAPID-M are:

local (first) - The problem space size is reduced by clustering individual application configuration spaces 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 (p-model), and 2) the common resource demand by configurations in the bucket;

global - Across applications, a machine model (m-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.

local (second) - Finally, a selection within each bucket is 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.

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). Together, m-model and p-models allow the prediction of system and configuration behaviors, i.e., assessing the mutual interference and benefits of configuration selections. 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.

The target applications of RAPID-M have to be adaptive, i.e., must have configurable components/parameters that can be manipulated during runtime, resulting in different cost/quality trade-offs. Six applications were selected from different domains (financial analysis, image analyses, machine learning), with three applications used in multiple related works (e.g., CALOREE [23], PowerDial [16], ESP [24]) on single-application adaptations. The applications differ in their configuration space sizes and cost/quality trade-offs.

Experimental results on the six applications and different execution traces show the effectiveness of RAPID-M and its implementation against five other heuristics. Application configurations can be partitioned into a small number of buckets allowing the system to produce global configurations of high quality. The runtime overhead includes the 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 RAPID-M's offline overhead. Compared to existing approaches in which each application adapts itself individually and application interference is treated as noise, on a 4-core machine, RAPID-M achieves 3.4% higher success rate when the system is not busy (≤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, RAPID-M achieves such improvement with an average of 40% fewer performed reconfigurations.

2 RAPID-M FRAMEWORK OVERVIEW

RAPID-M uses the "standard" notion of a configuration as defined by most existing adaptive configuration management approaches [16, 23, 32]. An application's configuration is a set of discrete "knob" values, where knobs are entities (e.g., program variables) that impact the quality and cost of the application's outcome. RAPID-M is a framework that manages the configurations of concurrent applications with the goal of choosing individual configurations such that all individual resource constraints are satisfied while maximizing the combined quality of active applications. In other words, RAPID-M solves the Global Configuration Problem defined below.

The particular cost metric used throughout this paper is execution time, though one might use, for instance, energy consumption as an alternative possible metric.The quality of an application execution is defined by an application-specific quality metric applied to its final output/outcome, e.g., the accuracy of a computed numeric value, or the precision/recall of the set of identified faces in a face-detection application. If an application fails to successfully terminate, its execution does not have any measurable quality. Further, to enable dynamic reconfiguration, applications are broken into a collection of work units, with configurations dynamically assigned to work units at runtime. Work units are also used to keep track of an application's progress towards successful termination.

Global Configuration Problem: At system-defined points in time, t_x, determine a global configuration vector $[c_1, c_2, \ldots c_n]$ 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.

A single application's configuration space is the Cartesian product of all knob value ranges, where each knob has finitely many value settings. Assuming that a single application has k knobs with discrete ranges of m values each, the resulting configuration space is $O(m^k)$. Existing approaches construct performance models to rank all configurations in the space according to some optimization objective. A "brute-force" approach towards the global configuration problem is to treat all active applications as a single, meta application. A configuration in this meta application is then a combination of all configurations in each component application. The size of the resulting configuration space is $O((m^k)^n)$ for n applications. It is exponential in n, which may be infeasible large to effectively explore. However, configuration space exploration is necessary because of the interdependence of the individual application configurations due to shared target system resources. To the best of our knowledge, the RAPID-M framework is the first to address configuration management of multiple active applications.

A key design feature of RAPID-M is to model cross-application configuration interference not at the configuration space level, but on the system footprints associated with each configuration. A *system footprint* is a vector of hardware counters that characterize the use of different system resources by an executing application, i.e., are used to represent a configuration's resource demands. In addition, footprints can represent resource demands of groups of active applications, including entire system workloads. Since resource sharing and contention happens at the machine level, 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 execution time (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. This stresser workload is introduced by running either another application or an instance of "stress" tool, e.g., the Linux Stresser [34].

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 combination of application configuration buckets that maximize the overall quality under the changed resource availability. The selected buckets together with their predicted slowdown are assigned to their respective applications. The local controller relies on optimization strategies used in single-application scenarios for configuration selection within the assigned bucket.Therefore, we will concentrate our discussion in this paper on RAPID-M's offline component and the online global configuration manager.

3 RAPID-M OFFLINE PHASE

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

3.1 Resource Usage Prediction: M

The performance of an application can significantly degrade when the overall system resource utilization is high. For example, one of our applications, Bodytrack, has a 15× slow-down in execution time under heavy system loads. Predicting the overall system load, i.e, the system environment is crucial before estimating the performance degradation of an application under multi-programming environments. The key questions to answer is how a system workload will change when a new application starts or an existing application terminates in the context of other active applications.

RAPID-M trains the m-model with a set of experiments, each with a pair of running "instances" involved. Such instance can be a realistic application (e.g., one of our benchmark application), or a Linux "stress" tool [34] that introduces arbitrary workloads to the system, including I/O, CPU utilization, and hard disk access. The system footprint generated from the two instances can vary in different experiments by using different applications or different configurations. We use Intel's Performance Counter Monitor (PCM) [17] to measure the footprint and represent it by a vector V where each entry corresponds to a particular system metric/feature considered by RAPID-M. RAPID-M, uses [EXEC, FREQ, INST, INSTnom, IPC, L2MISS, L2MPI, L3HIT, L3MISS, L3MPI, PhysIPC, MEM] [17].

For each experiment (training data point), RAPID-M measures and collects the footprints when each instance executes in isolation (v_1, v_2) and together $(v_{1,2})$, all with the same length m where m is the number of features to observe (in our case, $m = 12$). RAPID-M then constructs a separate model for each feature using the standard regression method. Equation 1 shows the model construction for the k-th feature. X is a matrix of size $N * 2m$, where N is the number of experiments. The first m columns of X are list of v_1's and the last m columns are list of v_2's, i.e., each row of X is a concatenated vector $[v_1, v_2]$. The goal is to locate β_k that minimizes the error ϵ.

$$\left[v^1_{1,2}[k], v^2_{1,2}[k], ...v^N_{1,2}[k]\right]^T = X\beta_k + \epsilon \quad (1)$$

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

$$V = M \otimes ...(M \otimes (M \otimes (V_1, V_2), V_3)...V_n) \quad (2)$$

3.2 Performance Prediction: P

RAPID-M predicts the performance degradation for each application under different environments by a performance regression model p-model. p-model is trained by collecting the application slow-down under different configurations and workload environments. During training, RAPID-M first records the execution time for an application under configuration c when running alone. It then runs the configuration under different environments induced by adding extra workloads from different "stressers". Under each environment, RAPID-M records the overall system footprint V and the execution slow-down (α). The regression model minimizes the error between prediction $\bar{\alpha}$ and observed slow-down α. A unique p-model is constructed for each bucket. During construction, configurations that are not part of the bucket are not considered.

Different types of models (e.g., Linear-Regression [27], Bayesian-Ridge [27]) may be a better fit for a particular footprint feature or slowdown (predicted by m-model and p-model respectively), and different candidate models may produce accurate predictions based only on a subset of vector features. The latter issue is addressed in ESP [25] by first filtering out insignificant features, then training a higher-order model with the remaining features. However, the drawback is that it uses a single, linear model approach for all applications, and does not distinguish between different configurations. Also, to support the slowdown prediction for 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 on single applications. To address the first issue, RAPID-M maintains a model-pool with multiple models, including Regular Linear-Regression (LR) regression [27], Elastic-Net (EN) regression with cross-validation [27], Lasso (LS) regression with

cross-validation [27], Bayesian-Ridge (BR) regression [27], and a neural network with a single hidden layer with 50 and 'relu' activation function (NN) [7]. When constructing the models, RAPID-M trains all models in the pool and picks the one with highest accuracy. For each of these candidate models except NN, RAPID-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.

3.3 Bucket Determination

An application's configurations may have significantly different behaviors in terms of the system footprint they introduce, and their performance degradation under different system workloads. However, these configurations can be clustered into a limited number of groups with configurations within each group sharing similar behavior, i.e., introduce similar workloads to the system and suffer from similar slow-down given a particular environment. This is a key observation that allows RAPID-M to summarize configuration spaces with only limited information loss.

Figure 1 shows the dendrogram of hierarchically clustered configurations [9] in one of our benchmark applications (Ferret) by system footprints using Wards minimum variance method [33]. The X-axis shows the index of all 700 configurations. For simplicity, we 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 distance between clusters at certain levels. The height of each "sub-tree" reveals the distance between the inter sub-cluster (e.g., the distance between the two yellow dots is the height of the black dot). Thus, configurations can be clustered into buckets with certain granularity. Configurations within each bucket have higher similarity in terms of system footprints.

The number of buckets can be determined by the distance threshold (closeness of configurations in a cluster). In Figure 1, 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 1, 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 for Ferret by two orders of magnitude (5 buckets instead of 700 configurations).

The bucket design captures two aspects of similarity, namely 1) same system footprint: switching between configurations in to the same bucket will not change the contribution of the application to the overall system workload,and 2) same slow-down: all configurations within a bucket suffer from the same performance degradation under a given environment. The number of buckets could range from 1 (similar footprint for all configurations) to N (all configurations have a unique footprint). Having more buckets results in higher configuration similarity, but increases the problem size, while fewer buckets could hurt the accuracy of m-model and the p-models. RAPID-M implements a variant of Hierarchical Clustering[9] as described in Algorithm 1. First, a standard Hierarchical Clustering procedure generates the initial buckets, satisfying the distance threshold T_{dis} (first cut). Then, we 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} threshold is not satisfied, we iteratively refine the bucket that have the worst p-model accuracy until the threshold is satisfied. Lower T_{dis} and T_{acc} may result in more buckets or a more accurate p-model but will increase the size of the problem. In RAPID-M, we use T_{dis}=4 and T_{acc}=6%. The particular choice of these numbers was based on our experiences with our sample applications.

Input: $all\_configs$, T_{dis}, T_{acc}
Result: buckets
buckets = [all_configs];
buckets = h_cluster(buckets, criterion='dis', T_{dis}) // first cut
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

4 ONLINE CONFIGURATION MANAGER

The goal of RAPID-M is to find a global optimal configuration for all active applications. By grouping configurations into buckets, the size of the global search space is reduced from all combinations of application configurations to all combinations of application buckets, reducing the 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. It has the property that if the predicted slow-downs (p-models) and the predicted configuration interactions (m-model) are correct (accurate within an error threshold), then 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 configuration manager is invoked each time an application starts, 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 by local configuration managers, i.e., a local controller if due to system uncertainties no feasible configuration in the assigned $fCgs$ set meets the application's execution time constraint.

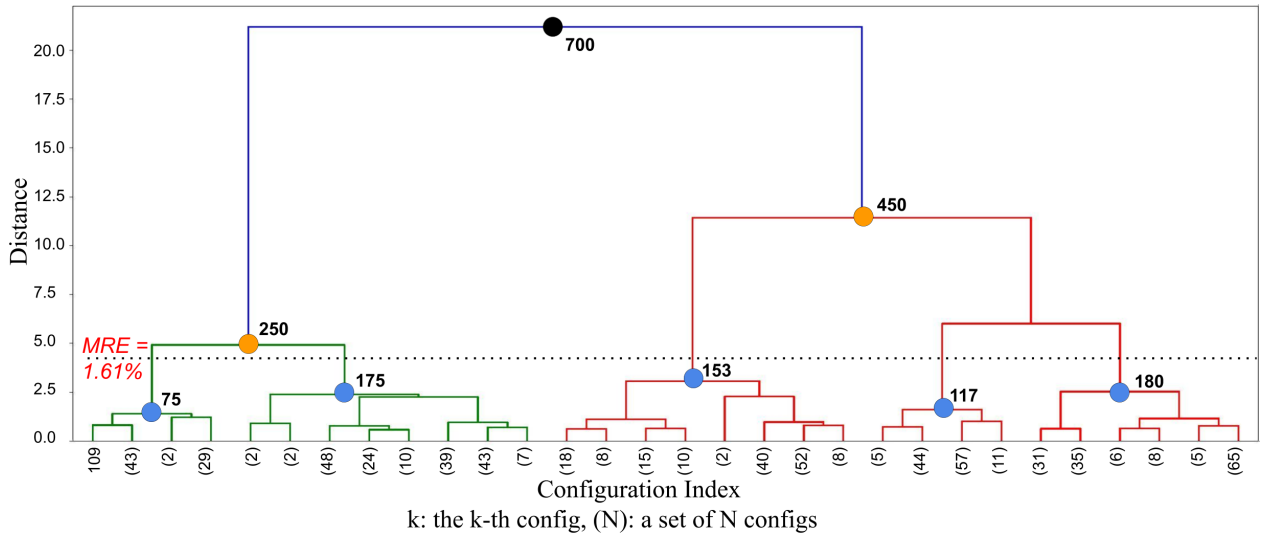


Figure 1: Dendrogram of Clustering Configurations into Buckets; Vertical Bar Shorter ⇒ Higher Inner-Cluster Similarity; Dots with Numbers: Number of Configurations in a Cluster (Sub-tree); Percentage Number: P-Model Prediction Error

5 SAMPLE APPLICATIONS

To assess the practical, end-to-end effectiveness of RAPID-M, we implemented and evaluated a prototype system. The evaluation uses six sample applications which have been used in multiple related works [16, 22, 25] in approximation: Swaptions[3], Bodytrack[3], Ferret[3], FaceDetection[5] SVM[27], and NeuralNet (NN)[27].

Swaptions – financial analysis application. Computation is based on iterative simulation. Output quality is defined as the average accuracy loss across all swaptions calculated. It has *10* configurations.

Bodytrack – computer vision application that tracks a set of human body components from a video frame by frame. The output quality is calculated by evaluating the position accuracy loss per-component in all frames. Bodytrack has **Two** Knobs: number of annealing layers from [1, 5], number of particles to track sampled within [100, 4000] with *50* configurations in total.

Ferret – image similarity application that returns the top-K images in a database for a query image ranked by content-similarity using a Multi-Probe LSH [21] algorithm. We use a common ranking score function shown in Eq 3:

$$err = 2 * (k - z)(k + 1) + \sum_{i \in Z} |rank_1(i) - rank_2(i)| - \sum_{i \in S} rank_1(i) - \sum_{i \in T} rank_2(i) \quad (3)$$

with $Q = 1 - err/k(k + 1)$ 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 $list_2$. k is the size of set S or T, and z is the size of Z. $rank_1$ is the rank of an image in $list_1$, and $rank_2$ the rank in $list_2$. In our case, we use the execution output as $list_1$ and the default output as $list_2$. Ferret has three knobs: number of probe bucket sampled within [2, 20], number of hash tables {2, 4, 8}, and number of iterations [10, 500] to compute Earth Mover's Distance [30]. Ferret has emph700 configurations in total.

FaceDetection – object detector that detects human faces from a series of input images based on Haar Cascade[5]. We adopt the standard measurement of recognition performance, the F-score [6]. FaceDetection has three knobs: pyramid levels sampled within [5,25], neighbors pixels to examine {0,4,8}, and the minimum number of eyes {0,1,2}, with *90* configurations in total.

SVM and **NN** – two supervised image classifiers. They run 1000 iterations on a set of labeled data and construct a Support Vector Machine and a Neural Network model. Classification accuracy is used as the quality. Both SVM and NN have three knobs: learning rate sampled within [1e-7,1e-5], batch size {64, 128, 256, 512, 1024}, regularization rate [5000, 25000], resulting in *250* configurations.

All our applications are input dependent, i.e., the cost/execution time for each work unit can vary for different inputs. Ferret has the most significant deviation of cost per work unit (mean=87.85ms, std = 80). () has the least deviation (mean=2191.22ms, std=25). Such dependency shows the necessity of adjusting the configuration dynamically (re-configuration) even when the application is running alone. The problem gets more complex in multi-programming environments with cross-application interference.

6 RAPID-M IMPLEMENTATION

The RAPID-M framework is implemented as a set of offline and online modules. During new application development or adapting an existing application for execution within RAPID-M, the application developer has to provide information to RAPID-M's offline local module via provided, simple APIs. Such information includes (1) the arguments needed to run each application, (2) quality notions to evaluate application outcomes through profiling, and (3) configuration space specification. Items (1) and (2) are implemented as a Python module, and item (3) is a separate configuration file. This information enables the offline application profiler to automatically

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), ... 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

collect profiling data for an application and is necessary for any system where offline modeling is required (e.g., OpenTuner [2]). The information 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 [34].

In contrast to other approaches (e.g., ESP[24]), RAPID-M collects each application's profile data, and constructs the models independently of other applications, making RAPID-M easily scalable. The offline generated information is represented in an application profile and stored on the RAPID-M server, which also hosts the online global configuration manager. 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 budgets. It then determines bucket assignments for each application using Algorithm 2. Application start and termination

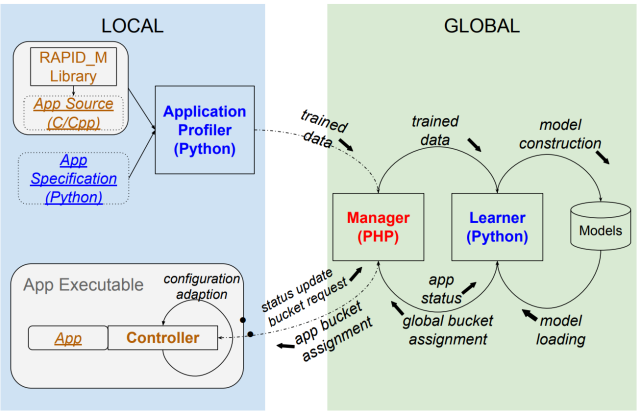


Figure 2: RAPID-M **Overview: Solid Border Boxes Provided by** RAPID-M**; Dashed border Required from Developers.**

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 2 shows the RAPID-M framework overview. The RAPID-M profiler and learner (for model construction and prediction) are implemented in Python. The local runtime controller is a C++ library with API's to be integrated into source code.During runtime, it communicates with the online global configuration manager (in PHP). The manager communicates with the learner via sockets.

Local Application Profile: It is designed as a data collector for the developers to train the application for RAPID-M required models. The profiler requires the developer to specify individual knob settings, the output quality evaluation, and application execution instructions. Table 1 shows the profiling strategy of RAPID-M. For each configuration C_i, RAPID-M's profiler runs $1+K$ tests. The first run is a *base-run* where applications execute alone on the target system using C_i, measuring the Base Cost: c, Output Quality: q, and System Footprint: v. Then the profiler runs the application K times each with a different "stresser". For each run, RAPID-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. The value of K can be set by developers with a default value of 10. Larger K will collect more slow-down observations resulting in higher profiling overhead.

Global Learner: After the profiler(s) collect the training data, the data is sent over to a global server for model construction. During the training, the learner:

- Constructs / Updates the m-model for the machine: uses all observed system footprint from all applications, $X = [v_s, v_a]$ $Y = [v_{as}]$

Table 1: Data Collected by RAPID-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

- Computes the bucket for the application: uses the observed system footprint of all configurations: $[v]$
- Constructs the p-model for the bucket: uses overall system footprint and 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 target machine. There are four cases when running applications need to contact the server:

(1) 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.
(2) Re-configuration: The application actively requests a new bucket assignment when no configuration in the current bucket can satisfy the budget constraint.
(3) Routine Check: The application periodically checks with the server for updated bucket assignments. This is needed since the assignment can be changed when new applications start on the same machine. By default, RAPID-M performs a routine check after each 10% of the total work units.
(4) 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.

7 MODEL VALIDATION

RAPID-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. RAPID-M constructs the p / m models from a set of publicly available models as discussed in Section 3: 1) Neural Net **(NN)**[7] with 1 hidden layer and 50 neurons, using 'relu' as the activation function; 2) Linear regression **(LR)**[27]; 3) Lasso **(LS)**[27] regression with cross-validation; 4) Elastic net **(EN)**[27] regression with cross-validation; and 5) BayesianRidge **(BR)**[27] regression. The RAPID-M framework uses a modular design and allows developers to add more models to the model pool.

Table 2: Selected Model and Features.*Poly*: polynomial features used; *MRE*: slowdown prediction error

	# 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%

Bucketing with p-model: The purpose of bucketing configurations is to reduce the search space size from the number of all configuration combinations to the number of combinations of buckets. However, this approach requires a good prediction of both system environments and bucket slowdowns. Therefore, besides the footprint similarity, the number of buckets is determined by the accuracy of the per-bucket slowdown model (p-model).

In Table 2, the column named "# configs & # buckets" shows the number of buckets constructed from the total number of configurations. Column "Model" and "Poly" report the type of model and whether the model use polynomial features or not. Column "MRE" reports the per-bucket **M**ean **R**elative **E**rror of slow-down prediction. The different models types and features selected by RAPID-M show the need for model customization. Comparing to ESP [25] where Elastic Net is used across multiple applications, RAPID-M locates the best model on an application bucket level. RAPID-M reduces the overall problem size by clustering large configuration spaces into a few buckets while still providing an accurate slowdown prediction with an error below 3%, and 1.5% on average, which justifies the feasibility of this approach.

System Profile Prediction with m-model: m-model validation uses system footprints consisting of 12 features. For space reasons we do not list per-feature prediction accuracy. On average, RAPID-M has an R2 score of 0.93 for all features with an MRE of 14.8%.

Optimality Validation: Though the prediction accuracy of the p-model is high (see Table 2), m-model predicted overall system footprint is used when applying the p-model during runtime since the future behavior of the system cannot be measured. Errors due to m-model prediction errors can impact slow-down prediction accuracy, which negatively effects bucket and configuration selection. We conduct another simulation to investigate how the error propagation from m-model to p-model would affect selection optimality.

The experiment first runs an application with a stresser and records (1) the footprint of the application, fp_a, (2) the footprint

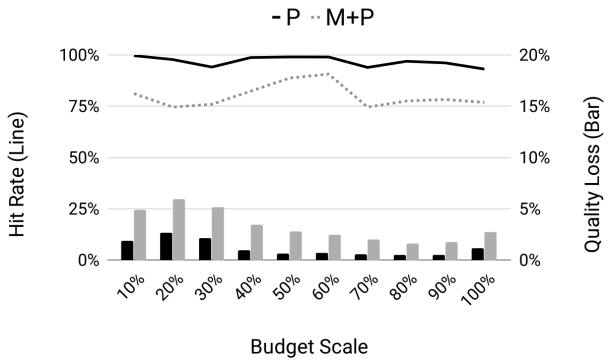


Figure 3: Impact of Error Propagation on Optimality (Hit Rate and Quality Loss) under Different Budget Scales.

of the stresser fp_s, (3) the overall environment env, and (4) the application slowdown sd. With the performance profile of the application, the optimal configuration c_{opt} can be computed using sd. We then compute the selected, "optimal" configuration c_p by using the slowdown predicted by applying the p-model to env. Finally, we 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. We evaluate such difference in terms of (1) hit rate: whether the selection is identical, and (2) quality loss: 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, respectively. 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. We run the simulation covering the whole range of possible budgets for each application from 10% to 100% of $MAX - MIN$, where MAX and MIN represent the highest and lowest cost of all configurations when running alone.

Figure 3 shows the optimality evaluation result of the simulation. The X-axis represents the available budget percentages. The lines following the left Y-axis report the average prediction correctness of the configuration selection across all applications, e.g, 100% means that the optimal configuration from RAPID-M gives the exact same configuration as the oracle. The bars following the right Y-axis show the relative loss of quality. The loss of quality is computed by comparing the quality outcome with optimal configuration under the budget (as determined by offline measurement) against the configuration computed by p-model with or without the m-model. Note that "P" strategy cannot be used in the real deployment of RAPID-M but is only intended to show the impact of prediction errors caused by the p-model alone. Bucket selection is performed at runtime using the m-model, however we assume the same bucket selection for both, the P and M-P models in this simulation. As shown in the graph, if the environment is known, the error in p-model contributes to ≤10% of selection difference as the optimal. The errors of combining m-model and p-model result in different configuration selections on average in 19.9% of the cases. However, the wrongly selected configuration is close to the optimal configuration, i.e., suffers only up to a 6% output quality loss with an average of 3.3%. This indicates that the embedded errors in m-model and p-model affect the output quality to a limited extent.

8 RAPID-M EXPERIMENTAL EVALUATION

We conduct several experiments to show the effectiveness of key components of our approach. These experiments concentrate on showing the ability of RAPID-M to produce configuration selections of high quality. The evaluation methodology compares possible configuration selection strategies with RAPID-M on different sets of concurrently active applications. We define the following alternative configuration selection strategies which are constructed either as extensions to existing single application approaches or multi-application strategies for other problems (e.g., ESP [24] for scheduling). The strategies differ in what information they use to determine each application's configuration.

ContextOblivious (CO): Applications are not aware of each other. Slowdown caused by other applications is treated as "noise."

AwareShare (AS): This strategy is partially inspired by ESP[24]. It measures the overall environment of different combinations of benchmark applications offline using their default configurations (ignoring buckets). Then it applies the p-model on the recorded environment to select optimal configurations. This strategy is only used as an oracle in static evaluation where the application combination is known before execution.

RAPID-M **(RM):** This strategy utilizes the full power of RAPID-M: configurations are clustered into buckets, the m-model predicts the system environment, and p-models predict slowdown.

RAPID-M **with Rush-To-End (RM_Rush):** This strategy extends RAPID-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 is designed as an insurance policy that tries to avoid failing an execution after most of the work has been done.

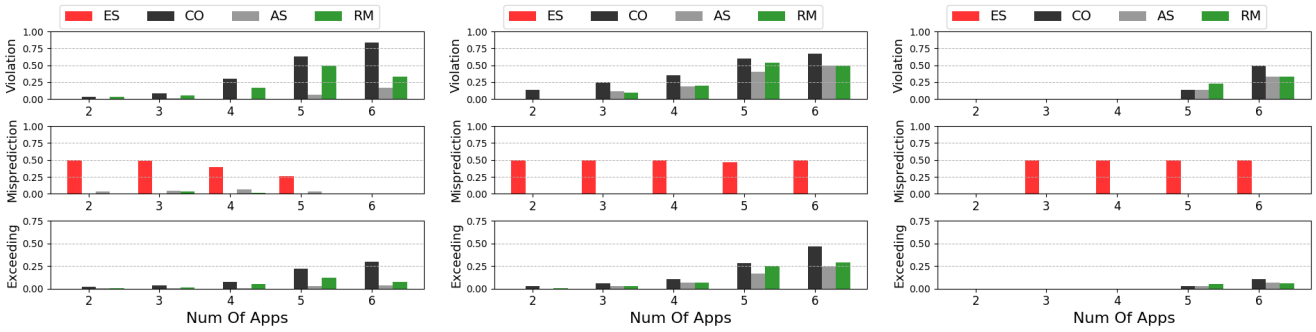
EqualShare (ES): Each application divides its assigned resource budget by the number of concurrently active applications. This reduced budget is used to determine the application's configuration.

Always Low(LOW): Use lowest cost configuration for execution.

Table 3: 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 3 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



(a) low budget (scale=80%) (b) medium budget (scale=100%) (c) high budget (scale=150%)

Figure 4: Static Configuration Selection Comparison under Different Budgets Scales.

different budget constraint levels (high, medium, low), we measured the overall combined quality outcomes of the applications.

8.1 Global Reconfigurations - Static

The first set of experiments assesses 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 ≥ provided budget.
- *Misprediction*: No feasible configuration can be found by the strategy though there exists at least one.
- *Exceeding Rate*: Fraction the execution time exceeds the budget; E.g.: rate=1.0 indicates twice the budget.

Figures 4a, 4b, and 4c show the performance of the computed global configurations by the different selection strategies for low (80%), medium (100%), and high (150%) resource budgets (time deadlines) for each application. The scales are defined the same as in Figure 3. Note that even if an application gets 150% of budget scale, it still may not be able to run with the highest configuration because of the overall system workload. The results show that: 12.7% of executions using *CO violate* the budget when the system is not busy (≤4 active apps), and 56.1% when busy. RAPID-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% due to either errors in the p-model, input dependencies, or minor system effects since no reconfiguration is performed. For executions that *violates* the budget, RAPID-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 dynamic reconfiguration. This experiment shows that RAPID-M lowers the *violation* rate by 33.9% compared to *CO* with a small exceeding rate of 6.6%.

8.2 Global Reconfigurations - Dynamic

The second set of experiments shows the overall performance of RAPID-M with local and global reconfigurations. We evaluate the performance of RAPID-M by dispatching different applications starting at random times (to mimic the unpredictable real-world execution pattern) each with a given budget. Each application gets to reconfigure during the execution. We 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, i.e., best configuration producing the highest possible output quality (*Mission* in Figure 5). For each generated trace (mission), we repeat the trace with shrinking budgets to force reconfiguration using different strategies. Different strategies are evaluated on four aspects:

- *Rejection*: Failing to find a feasible configuration at application start time. Reported as a rate.
- *Success*: Finishing the execution within budget Reported as rate.
- *Reconfiguration*: Number of reconfigurations in response to changes in performance. More frequent reconfiguration is usually caused by the mismatch between the real and predicted execution.
- *Output Quality*: Normalized overall output quality from 0 to 1.0 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.

To demonstrate the performance in different scenarios, we run the experiment with a threshold N such that the simulator will stop dispatching new applications when there are N active applications. Figure 5 shows an example of the experimental results for executions using *CO* and RAPID-M with a 10-minute-execution trace where N=4. As shown in the graph, executions using *CO* are more likely to *fail* (shown as dashed lines followed by crosses) during the middle of execution due to underestimating the slowdown. However, only two runs failed using RAPID-M excluding the two runs that got *rejected* (shown as crosses). We omitted the

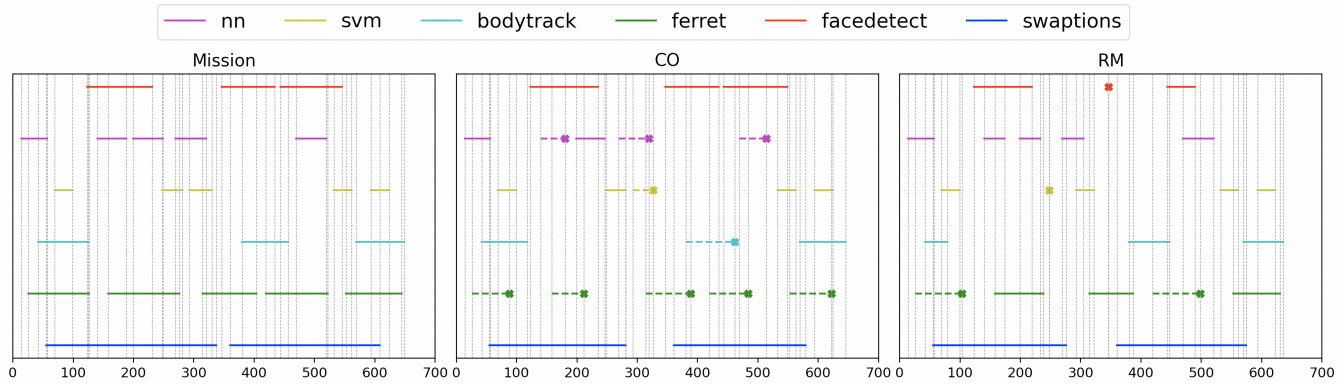


Figure 5: Sample Execution using Different Strategies for a Mission Trace (top left): Length=10 min, N=4, Budget_Scale=1.0.

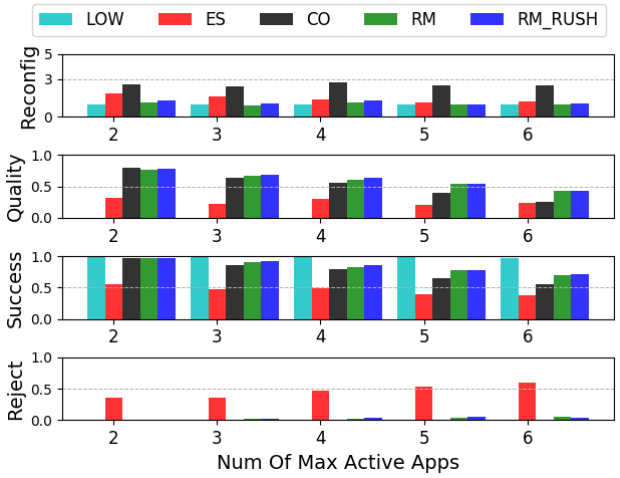


Figure 6: Averages from Dynamic Configuration Selection on 18 Mission Traces and Low/Medium/High Cost Budgets.

trace for other strategies due to space limitation. *LOW* finishes all executions indicating that there exists at least one solution that successfully executes the trace. *ES* rejects most executions due to overestimating the slowdown. As for *RM_RUSH*, one of the failed runs is "rescued" by adding the 'rush_to_end' strategy.

Figure 6 summarizes the results of 18 traces of 10 minutes, each repeated with multiple budget settings using different strategies. Quality is calculated as the average per-run quality across all traces. Quality per application run is calculated by the raw quality value normalized by its quality range as discussed above. (0.0 and 1.0 indicate the lowest and the highest quality). Different quality notions are application-specific and discussed in Section 5. To summarize, *ES* predicts the slowdown so aggressively that it rejects most executions and has 54.4% of RAPID-M's success rate. On a 4-core machine, RAPID-M achieves a 3.4% higher success rate than existing approaches (modeled by *CO*) when the system is not busy (≤4 active apps), and 22.75% higher when busy. This translates to 2.6% and 52.99% higher output quality. Furthermore, RAPID-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.

8.3 Off-line and Runtime Overheads

We evaluate RAPID-M on both off-line profiling/model training, and runtime reconfiguration on a single-socket machine with 4 cores at 3.7GHz, and 16GB of RAM at 2666MHz.

Off-line Overhead: RAPID-M and existing approaches for single application configuration selections occur overheads due to training of different models. These off-line training times can be substantial, sometimes in the order of days. If fact, Meantime[10] reported training times of up to two weeks for a single application. The overhead is determined by the configuration space and the length of per-configuration training needed for collecting meaningful results.

Table 4 summarizes the offline overheads of using RAPID-M. The second column reports the number of Lines Of Code changes required from the developer to integrate RAPID-M library calls into the original source code. The column "# configs" reports the total number of configurations for each application. The column "profiling" shows the time required to collect the performance data. 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. However, the data collecting phase can be time-consuming with an overhead proportional to the complexity of the application.

Table 4: RAPID-M **Implementation with Offline Overheads**

	# LOC Changes	Offline Training Overhead		
		# configs	profiling	bucket and 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

Runtime Overhead: The dynamic overhead is measured as the average time an application has to wait for the server's answer and

the local reconfiguration time if the bucket assignment changed. Overall, the average turnaround time of requesting a bucket assignment is ~ 201ms using an off-the-shelf system. The major part of the overhead comes from the online application of the m-model and p-models. Therefore, the overhead can be further reduced by selecting a more powerful server and/or a network with lower latency. The average execution time overhead experienced by each application was less than 5% of their execution time.

Challenges and Limitations: RAPID-M uses the system footprint as a key abstraction to represent configurations. However, this assumes the availability of hardware counters/metrics for cost model construction. In our experience, the model accuracy dramatically relies on the training coverage (the "stresser"). With growing diversity of the target applications, we expect RAPID-M to need more training to refine its models. Since we predict the combined/global execution environment through iteratively applying the m-model, the error may propagate, resulting in decreased prediction accuracy.

9 RELATED WORK

Exploiting application interference has been explored by several research groups in the context of non-configurable applications. In addition, using approximation and reconfiguration as an optimization opportunity has been discussed in multiple works recently.

Configuration Management: Several control theoretical approaches [10, 16, 37] aim to deal with runtime disturbance. JouleGuard [15] and Caloree [22] combine machine learning with control theory to overcome the shortcomings in each strategies when used separately. Probabilistic and approximate programming use probability variables and their distributions [4, 11–13, 26, 28, 29]. This research focuses on how to represent such distributions and operations on these distributions induced by operations on their associated probabilistic approximate variables. In the database community, approximation has been used to provide statistical error bounds on queries [1, 36] and more recently in the context of Hadoop (Map-Reduce) applications [14, 19]. These works target single-user environments and interference between multiple active applications is ignored. These approaches could be deployed in RAPID-M as the local controller selection strategy.

Performance Prediction: Significant research focuses on constructing cost models. Learning based models [8, 18, 25, 31, 35] predict performance through either input or execution features, with model accuracy bound by the richness of the data set, and introducing runtime overheads. Our work uses a similar approach to predict the slowdown for each configuration rather than the base performance in the single-application scenario. Control theoretical approaches [10, 15, 16, 37] aim to deal with runtime disturbance. However, to directly extend these approaches to multi-programming environments, the model has to be built on the entire search space which is infeasible due to the size. Even getting the profile for a single large application may take weeks ([10]).

Interference Prediction: Optimizing the behaviors of groups of applications in a multi-programming environment has been the goal of different research efforts. Models for predicting application interference have to deal with the non-linear impact of resource sharing on individually observed application performance / slowdowns. D-Factor [20] 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 of current system footprints to perform the prediction. The ESP system [25] is similar to our approach, since it measures specific system footprints for different applications. 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 requires the actual training process of k application running simultaneously to support the prediction of slow-down among up to k applications. In contrast, our approach is based on large configuration spaces for each application. Each application is trained individually, making our approach more flexible since groups of applications do not have to be known and trained in advance which makes RAPID-M's approach much more scalable.

10 CONCLUSION

To the best of our knowledge, RAPID-M is the first framework that enables effective and efficient approximation / configuration management across reconfigurable applications that execute concurrently on the same system. RAPID-M dynamically adapts multiple applications together at the same time, allowing the selection of configurations across all active applications that together result in the highest overall quality while respecting each application's resource budget (here execution time). Without the global optimization, applications with only local adaptation may encounter a higher failure rate and lower overall quality.

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. RAPID-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, global search space. For each of such bucket, RAPID-M constructs a specialized performance model with average prediction error of ≤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 RAPID-M scalable. Experimental results using six applications and different concurrent traces of application start and exit events show that RAPID-M is able to select globally optimal configurations. Compared to other possible approaches, RAPID-M makes configuration selections that lower the average budget violation rate by 33.9% with an average exceeding rate of 6.6%. At runtime, RAPID-M successfully finishes 22.75% more executions which results in a 1.52× improvement of output quality under high system loads. For our benchmark applications, the overhead of RAPID-M is within ≤1% of the application's execution time.

All code and raw data sets are available on github at https://github.com/niuye8911/rapid_m.

Acknowledgements

The authors would like to thank the reviewers for their insightful comments. This work has been supported by NSF grant CSR:EDS #1617551, and partially funded by NSF grant CSR:II-EN #1730043.

REFERENCES

[1] Swarup Acharya, Phillip B Gibbons, Viswanath Poosala, and Sridhar Ramaswamy. 1999. The Aqua approximate query answering system. In *ACM Sigmod Record*, Vol. 28. ACM, 574–576.

[2] Jason Ansel, Shoaib Kamil, Kalyan Veeramachaneni, Jonathan Ragan-Kelley, Jeffrey Bosboom, Una-May O'Reilly, and Saman Amarasinghe. 2014. Opentuner: An extensible framework for program autotuning. In *Proceedings of the 23rd international conference on Parallel architectures and compilation*. 303–316.

[3] Christian Bienia. 2011. *Benchmarking Modern Multiprocessors*. Ph.D. Dissertation. Princeton University.

[4] J. Borgstrom, A.D. Gordon, M. Greenberg, J. Margetson, and J. Van Gael. 2011. Meaure Transformer Semantics for Bayesian Machine Learning. In *European Symposium on Programming (ESOP)* (Saarbrucken, Germany).

[5] G. Bradski. 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools* (2000).

[6] Nancy Chinchor. 1992. MUC-4 evaluation metrics. In *Proceedings of the 4th conference on Message understanding*. Association for Computational Linguistics, 22–29.

[7] François Chollet. 2015. keras. https://github.com/fchollet/keras.

[8] Valentin Dalibard, Michael Schaarschmidt, and Eiko Yoneki. 2017. 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, 479–488.

[9] Richard C Dubes and Anil K Jain. 1988. Algorithms for clustering data.

[10] Anne Farrell and Henry Hoffmann. 2016. MEANTIME: Achieving Both Minimal Energy and Timeliness with Approximate Computing. In *2016 USENIX Annual Technical Conference (USENIX ATC 16)*. Denver, CO, 421–435.

[11] W.R. Gilks, A. Thomas, and D.J. Spiegelhalter. 1994. A Language and Program for Complex Bayesian Modelling. *Journal of the Royal Statistical Society, Series D (The Statistician)* 43, 1 (1994), 169–177.

[12] N.D. Goodman, V.K. Mansinghka, D.M. Roy, K. Bonawitz, and J.B. Tenenbaum. 2008. Church: A Language for Generative Models. In *Conference in Uncertainty in Artificial Intelligence (UAI)* (Helsinki, Finland). 220–229.

[13] A. Gordon, T. Henzinger, A.V. Nori, and S.K. Rajamani. 2014. Probabilistic Programming. In *International Conference on Software Engineering (ICSE)* (Hyderabad, India).

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

[15] H. Hoffmann. 2015. JouleGuard: Energy Guarantees for Appoximate Applications. In *Symposium on Operating Systems Principles (SOSP)* (Monterey, CA).

[16] Henry Hoffmann, Stelios Sidiroglou, Michael Carbin, Sasa Misailovic, Anant Agarwal, and Martin Rinard. 2011. Dynamic Knobs for Responsive Power-Aware Computing. In *ASPLOS '11*. Newport Beach, California, USA.

[17] Intel. 2012. *Intel Performance Counter Monitor - A better way to measure CPU utilization*. www.intel.com/software/pcm

[18] Pooyan Jamshidi, Miguel Velez, Christian Kästner, Norbert Siegmund, and Prasad Kawthekar. 2017. 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, 31–41.

[19] N. Laptev, K. Zeng, and C. Zaniolo. 2012. Early Accurate Results for Advanced Analytics on MapReduce. *Proceedings of the VLDB Endowment* 5, 10 (2012), 1028–1039.

[20] Seung-Hwan Lim, Jae-Seok Huh, Youngjae Kim, Galen M Shipman, and Chita R Das. 2012. D-factor: a quantitative model of application slow-down in multi-resource shared systems. *ACM SIGMETRICS Performance Evaluation Review* 40, 1 (2012), 271–282.

[21] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. 2007. Multi-probe LSH: efficient indexing for high-dimensional similarity search. In *Proceedings of the 33rd international conference on Very large data bases*. VLDB Endowment, 950–961.

[22] Nikita Mishra, Connor Imes, John D Lafferty, and Henry Hoffmann. 2018. CALOREE: learning control for predictable latency and low energy. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 184–198.

[23] Nikita Mishra, John D. Laferty, and Henry Hofmann. 2018. CALOREE: Learning Control for Predictable Latency and Low Energy. In *ASPLOS '18*. Williamsburg, Virginia, USA.

[24] Nikita Mishra, John D Lafferty, and Henry Hoffmann. 2017. Esp: A machine learning approach to predicting application interference. In *2017 IEEE International Conference on Autonomic Computing (ICAC)*. IEEE, 125–134.

[25] Nikita Mishra, John D Lafferty, and Henry Hoffmann. 2017. ESP: A Machine Learning Approach to Predicting Application Interference. In *Proceedings of the International Conference on Autonomic Computing* (Columbus, Ohio, USA) *(ICAC)*.

[26] S. Park, F. Pfenning, and S. Thrun. 2005. A Probabilistic Language Based on Sampling Functions. In *ACM Symposium on Principles of Programming Languages (POPL)* (Long Beach, CA). 171–182.

[27] 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. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[28] A. Pfeffer. 2001. IBAL: A Probabilistic Rational Programming Language. In *International Joint Conference on Artificial Intelligence (IJCAI)* (Seattle, WA). 733–740.

[29] N. Ramsey and A. Pfeffer. 2002. Stochastic Lambda Calculus and Monads of Probability Distributions. In *ACM Symposium on Principles of Programming Languages (POPL)* (Portland, OR). 154–165.

[30] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. 2000. The earth mover's distance as a metric for image retrieval. *International journal of computer vision* 40, 2 (2000), 99–121.

[31] Xin Sui, Andrew Lenharth, Donald S. Fussell, and Keshav Pingali. 2016. Proactive Control of Approximate Programs. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems* (Atlanta, Georgia, USA) *(ASPLOS '16)*. ACM, New York, NY, USA, 607–621.

[32] Shu Wang, Chi Li, Henry Hoffmann, Shan Lu, William Sentosa, and Achmad Imam Kistijantoro. 2018. 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, 154–168.

[33] Joe H Ward Jr. 1963. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association* 58, 301 (1963), 236–244.

[34] Amos Waterland. 2002. *The stress workload generator*. https://people.seas.harvard.edu/~apw/stress/

[35] Ran Xu, Jinkyu Koo, Rakesh Kumar, Peter Bai, Subrata Mitra, Sasa Misailovic, and Saurabh Bagchi. 2018. Videochef: efficient approximation for streaming video processing pipelines. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*. 43–56.

[36] K. Zeng, Shi Gao, B. Mozafari, and C. Zaniolo. 2014. The Analytical Bootstrap: A New Method for Fast Error Estimation in Approximate Query Processing. In *ACM SIGMOD'14* (Snowbird, UT). 277–288.

[37] Huazhe Zhang and Henry Hoffmann. 2016. Maximizing Performance Under a Power Cap: A Comparison of Hardware, Software, and Hybrid Techniques. In *ASPLOS'16*. Atlanta, Georgia, USA.