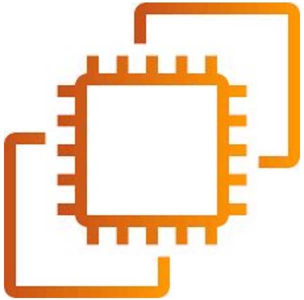
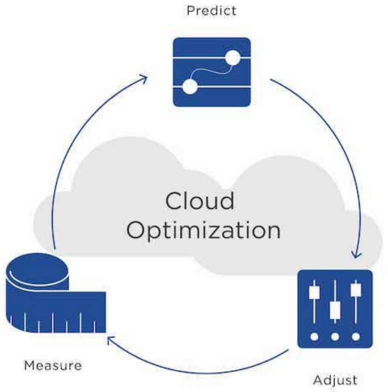
On the Use of ML for Blackbox System Performance Prediction

Silvery Fu[1], Saurabh Gupta[2], Radhika Mittal[2], Sylvia Ratnasamy[1]
(1: UC Berkeley, 2: UIUC)

Performance prediction is increasingly important!

- Optimization, capacity planning, SLO-aware scheduling





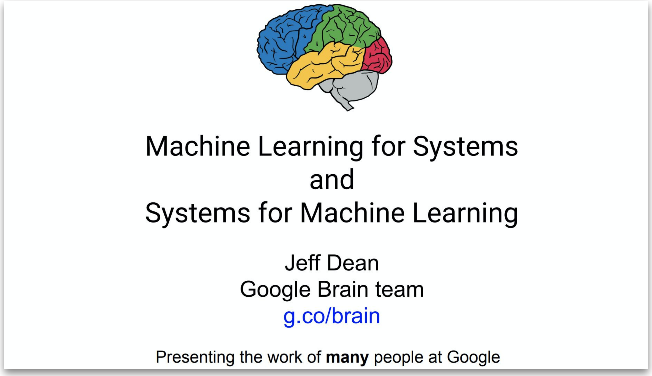
https://opsani.com/resources/cloud-optimization/

F(parameters) → performance

E.g., how many workers, size of input, machine configurations → JCT, query latency

Challenges

- Accurate
  - precise predictions
- Simple/easy-to-use
  - in-depth understanding of the systems not required
- General
  - works across a spectrum of workloads and applications



Can ML provide an *accurate*, *general*, and *simple* performance predictor?

ML for system perf. prediction?

This paper: a *systematic* and *broad* study on performance prediction!

Selecting the *Best* VM across Multiple Public Clouds: A Data-Driven Performance Modeling Approach

Neeraja J. Yadwadkar, University of California, Berkeley, neerajay@eecs.berkeley.edu
Bharath Hariharan, Facebook AI Research, bharathh@fb.com
Joseph E. Gonzalez, University of California, Berkeley, jegonzal@eecs.berkeley.edu
Burton Smith, Microsoft Research, burtons@microsoft.com
Randy H. Katz, University of California, Berkeley, randy@cs.berkeley.edu

ABSTRACT

Users of cloud services are presented with a bewildering choice of VM types and the choice of VM can have significant implications on performance and cost. In this paper we address the fundamental problem of *accurately* and *economically* choosing the *best VM* for a given *workload* and *user goals*. To address the problem of optimal VM selection, we present PARIS, a data-driven system that uses a novel *hybrid* offline and online data collection and modeling framework to provide *accurate* performance estimates with *minimal* data collection. PARIS is able to predict workload performance for different user-specified metrics, and resulting costs for a wide range of VM types and workloads across multiple cloud providers. When compared to sophisticated baselines, including collaborative filtering and a linear interpolation model using measured workload performance on two VM types, PARIS produces significantly better estimates of performance. For instance, it reduces runtime prediction error by a factor of 4 for some workloads on both AWS and Azure. The increased accuracy translates into a 45% reduction in user cost while maintaining performance.

CCS CONCEPTS

• **Computer systems organization → Cloud computing;** • **General and reference → Performance; Estimation;** • **Social and professional topics →** *Pricing and resource allocation;*

KEYWORDS

Cloud Computing, Resource Allocation, Performance Prediction, Data-Driven Modeling

ACM Reference Format:
Neeraja J. Yadwadkar, Bharath Hariharan, Joseph E. Gonzalez, Burton Smith, and Randy H. Katz. 2017. Selecting the *Best* VM across Multiple Public Clouds: A Data-Driven Performance Modeling Approach. In *Proceedings of SoCC '17, Santa Clara, CA, USA, September 24–27, 2017,* 14 pages. https://doi.org/10.1145/3127479.3131614

1 INTRODUCTION

As companies of all sizes migrate to cloud environments, increasingly diverse workloads are being run in the Cloud — each with different performance requirements and cost trade-offs [59]. Recognizing this diversity, cloud providers offer a wide range of Virtual Machine (VM) types. For instance, at the time of writing, Amazon [2], Google [7], and Azure [42] offered a combined total of over 100 instance types with varying system and network configurations.

In this paper we address the fundamental problem of *accurately* and *economically* choosing the *best VM* for a given *workload* and *user goals*. This choice is critical because of its impact on performance metrics such as runtime, latency, throughput, cost, and availability. Yet determining or even defining the *"best"* VM depends heavily on the users' goals which may involve diverse, application-specific performance metrics, and span tradeoffs between price and performance objectives.

For example, Figure 1 plots the runtimes and resulting costs of running a video encoding task on several AWS VM types. A typical user wanting to deploy a workload might choose the cheapest VM type (m1.large) and paradoxically end up not just with poor performance but also high total costs. Alternatively, overprovisioning by picking the most expensive VM type (m2.4xlarge) might only offer marginally better runtimes than much cheaper alternatives like c3.2xlarge. Thus, to choose the right VM for her performance goals and budget, the user needs accurate performance estimates.

Recent attempts to help users select VM types have either focused on optimization techniques to efficiently search for the best performing VM type [12], or extensive experimental evaluation to model the performance cost trade-off [69]. Simply optimizing for the best VM type for a particular goal (as in CherryPick [12]) assumes that this goal is *fixed*; however, different users might prefer different points along the performance-cost trade-off curve. For example, a user might be willing to tolerate mild reductions in performance for substantial cost savings. In such cases, the user might want to know precisely how switching to another VM type affects performance and cost.

The alternative, directly modeling the performance-cost trade-off, can be challenging. The published VM characteristics (e.g., memory and virtual cores) have hard-to-predict performance implications for any given workload [24, 39, 72]. Furthermore, the performance often depends on workload characteristics that are difficult to specify [15, 28, 39]. Finally, variability in the choice of host hardware, placement policies, and resource contention [59] can result in performance variability [29, 35, 54] that is not captured

Ernest: Efficient Performance Prediction for Large-Scale Advanced Analytics

Shivaram Venkataraman, Zongheng Yang, Michael Franklin, Benjamin Recht, Ion Stoica
University of California, Berkeley

Abstract

Recent workload trends indicate rapid growth in the deployment of machine learning, genomics and scientific workloads on cloud computing infrastructure. However, efficiently running these applications on shared infrastructure is challenging and we find that choosing the right hardware configuration can significantly improve performance and cost. The key to address the above challenge is having the ability to predict performance of applications under various resource configurations so that we can automatically choose the optimal configuration.

Our insight is that a number of jobs have predictable structure in terms of computation and communication. Thus we can build performance models based on the behavior of the job on small samples of data and then predict its performance on larger datasets and cluster sizes. To minimize the time and resources spent in building a model, we use optimal experiment design, a statistical technique that allows us to collect as few training points as required. We have built Ernest, a performance prediction framework for large scale analytics and our evaluation on Amazon EC2 using several workloads shows that our prediction error is low while having a training overhead of less than 5% for long-running jobs.

1 Introduction

In the past decade we have seen a rapid growth of large-scale *advanced* analytics that implement complex algorithms in areas like distributed natural language processing [24, 74], deep learning for image recognition [34], genome analysis [72, 61], astronomy [17] and particle accelerator data processing [19]. These applications differ from traditional analytics workloads (e.g., SQL queries) in that they are not only data-intensive but also computation-intensive, and typically run for a long time (and hence are expensive). Along with new workloads, we have seen widespread adoption of cloud computing with large data sets being hosted [7, 1], and the emergence of sophisticated analytics services, such as machine learning, being offered by cloud providers [9, 6].

With cloud computing environments such as Amazon EC2, users typically have a large number of choices in terms of the instance types and number of instances they can run their jobs on. Not surprisingly, the amount of memory per core, storage media, and the number of instances are crucial choices that determine the running time and thus indirectly the cost of running a given job. Using common machine learning kernels we show in §2.2 that choosing the right configuration can improve performance by up to 1.9x at the same cost.

In this paper, we address the challenge of choosing the configuration to run large advanced analytics applications in heterogeneous multi-tenant environments. The choice of configuration depends on the user's goals which typically includes either minimizing the running time given a budget or meeting a deadline while minimizing the cost. The key to address this challenge is developing a performance prediction framework that can accurately predict the running time on a specified hardware configuration, given a job and its input.

One approach to address this challenge is to predict the performance of a job based on monitoring the job's previous runs [39, 44]. While simple, this approach assumes the job runs repeatedly on the same or "similar" data sets. However, this assumption does not always hold. First, even when a job runs periodically it typically runs on data sets that can be widely different in both size and content. For example, a prediction algorithm may run on data sets corresponding to different days or time granularities. Second, workloads such as interactive machine learning [9, 55] and parameter tuning generate unique jobs for which we have little or no relevant history. Another approach to predict job performance is to build a detailed parametric model for the job. Along these lines, several techniques have been recently proposed in the context of MapReduce-like frameworks [77, 52]. These techniques have been aided by the inherent simplicity of the two-stage MapReduce model. However, the recent increase in the popularity of more complex parallel computation engines such as Dryad [51] and Spark [83] make these parametric techniques much more difficult to apply.

In this paper, we propose a new approach that can accurately predict the performance of a given analytics job. The main idea is to run a set of instances of the entire job on samples of the input, and use the data from these training runs to create a performance model. This approach has low overhead, as in general it takes much less time and resources to run the training jobs than running the job itself. Despite the fact that this is a black-box approach (i.e., requires no knowledge about the internals of

Selecta: Heterogeneous Cloud Storage Configuration for Data Analytics

Ana Klimovic, *Stanford University*
Heiner Litz, *UC Santa Cruz*
Christos Kozyrakis, *Stanford University*

Abstract

Data analytics are an important class of data-intensive workloads on public cloud services. However, selecting the right compute and storage configuration for these applications is difficult as the space of available options is large and the interactions between options are complex. Moreover, the different data streams accessed by analytics workloads have distinct characteristics that may be better served by different types of storage devices.

We present Selecta, a tool that recommends near-optimal configurations of cloud compute and storage resources for data analytics workloads. Selecta uses latent factor collaborative filtering to predict how an application will perform across different configurations, based on sparse data collected by profiling training workloads. We evaluate Selecta with over one hundred Spark SQL and ML applications, showing that Selecta chooses a near-optimal performance configuration (within 10% of optimal) with 94% probability and a near-optimal cost configuration with 80% probability. We also use Selecta to draw significant insights about cloud storage systems, including the performance-cost efficiency of NVMe Flash devices, the need for cloud storage with support for fine-grain capacity and bandwidth allocation, and the motivation for end-to-end storage optimizations.

1 Introduction

The public cloud market is experiencing unprecedented growth, as companies move their workloads onto platforms such as Amazon AWS, Google Cloud Platform and Microsoft Azure. In addition to offering high elasticity, public clouds promise to reduce the total cost of ownership as resources can be shared among tenants. However, achieving performance and cost efficiency requires choosing a suitable configuration for each given application. Unfortunately, the large number of instance types and configuration options available make selecting the right resources for an application difficult.

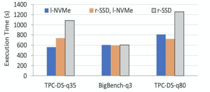


Figure 1: Performance of three applications on eight i3.xl instances with different storage configurations.

The choice of storage is often essential, particularly for cloud deployments of data-intensive analytics. Cloud vendors offer a wide variety of storage options including object, file and block storage. Block storage can consist of hard disks (*HDD*), solid-state drives (*SSD*), or high bandwidth, low-latency NVMe Flash devices (*NVMe*). The devices may be local (*l*) to the cloud instances running the application or remote (*r*). These options alone lead to storage configuration options that can differ by orders of magnitude in terms of throughput, latency, and cost per bit. The cloud storage landscape is only becoming more diverse as emerging technologies based on 3D X-point become available [35, 16].

Selecting the right cloud storage configuration is critical for both performance and cost. Consider the example of a Spark SQL equijoin query on two 128 GB tables [53]. We find the query takes 8.7× longer when instances in an 8-node EC2 cluster access *r*-HDD compared to *l*-NVMe storage. This is in contrast to a recent study, conducted with a prior version of Spark, which found that faster storage can only improve the median job execution time by at most 19% [50]. The performance benefits of *l*-NVMe lead to 8× lower execution cost for this query, even though NVMe storage has higher cost per unit time. If we also consider a few options for the number of cores and memory per instance, the performance gap between the best and worst performing VM-storage configurations is over 30×.

ML for system perf. prediction?

Start with the *best-case* scenario!

The **Best-Case** (BC) Test

- Given parameters P_1, P_2, P_3, …, P_k, want to learn F(P) → Perf. (e.g. JCT)
  - Dataset: data points of <P=X, JCT=Y>; split into training and testing sets

- ML assumptions:
  - *One-feature-at-a-time:* e.g., vary P_2, keeping P_1, P_3, … P_k fixed
  - *Seen-configuration:* e.g., points where P_2=1GB appear in training and testing-sets
- Systems assumptions:
  - *No-contention:* dedicated EC2 instances, isolated experiments;
  - *Identical-inputs:* same input data for a given input dataset size;

Applications and Models

Framework	Application/Description	Input Workload	Input Parameter	App. Config. Parameter	Infra. Parameter	Metric
Memcached [12]	Disributed in-memory k-v store	Mutilate [13]	value size	# servers	inst. type	mean query lat.
Nginx [9]	Web server, LB, Reverse Proxy	Wrk2 [5]	req. rate	# servers	inst. type	median req. lat.
Influxdb [15]	Open source time series database	Inch [10]	# points per timeseries	# servers	inst. type	mean query lat.
Go-fasthttp [7]	Fast HTTP package for Go	wrk2 [5]	# conn.	# servers	inst. type	median req. lat.
Spark [3]	*TeraSort:* sorting records	TeraGen	# records	# executors	inst. type	JCT
	PageRank: graph computation	GraphX SynthBenchmark [8]	# vertices			
	LR1: logistic regression	MLLib examples	# examples			
	LR2: logistic regression	Databricks Perf Test [16]				
	KMeans: clustering					
	Word2vec: feature extraction					
	FPGrowth: data mining					
	ALS: recommendation					
TensorFlow [17], Kubernetes [11]	*TFS:* Tensorflow model serving	Resnet examples [18]	# conn.	# servers	inst. type	requests/sec

ML models:

Nearest-neighbors,
Linear-regression,
Random forest,
SVM, SVM-kernelized,
Neural networks

Metrics and Predictors

- Accuracy metric: $\sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{Y_i - f(X_i)}{Y_i}\right)^2}$
  - rMSRE
- ML predictors → Best-of-Model/BoM-err
  - rMSRE of the most accurate model
- Oracle predictor → **O-err**

$$f_{\text{oracle}}(X) = \left(\sum_{i=1}^{n} \frac{\delta(X_i,X)}{Y_i}\right) / \left(\sum_{i=1}^{n} \frac{\delta(X_i,X)}{Y_i^2}\right),$$

$$\delta(a,b) = 1 \text{ if } a \text{ is equal to } b\text{, and } 0 \text{ otherwise.}$$

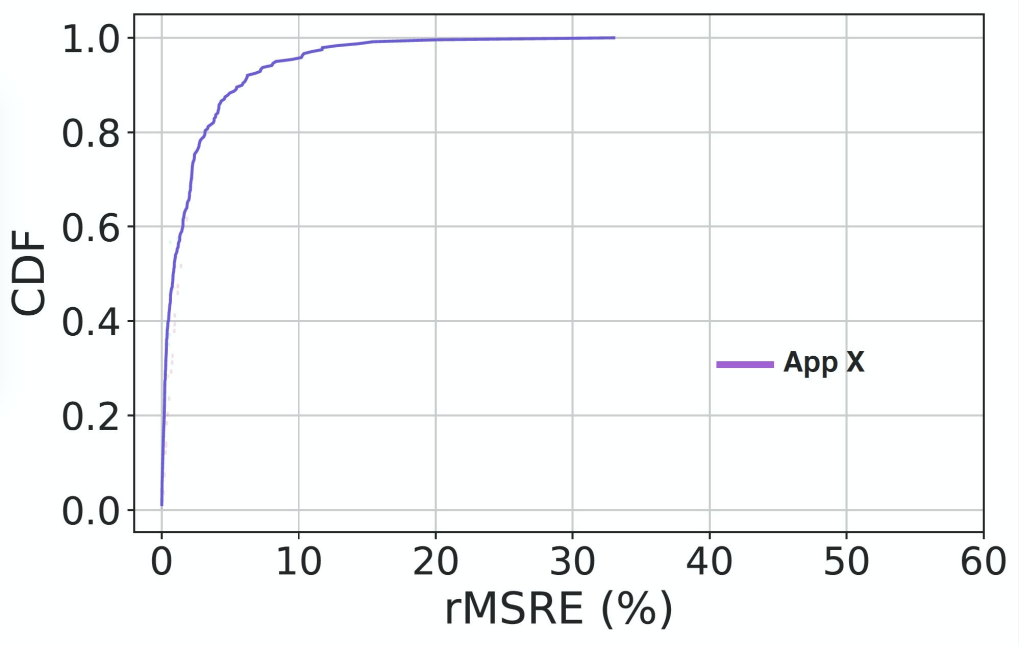
Y_i: true value

$f(X_i)$: predicted value

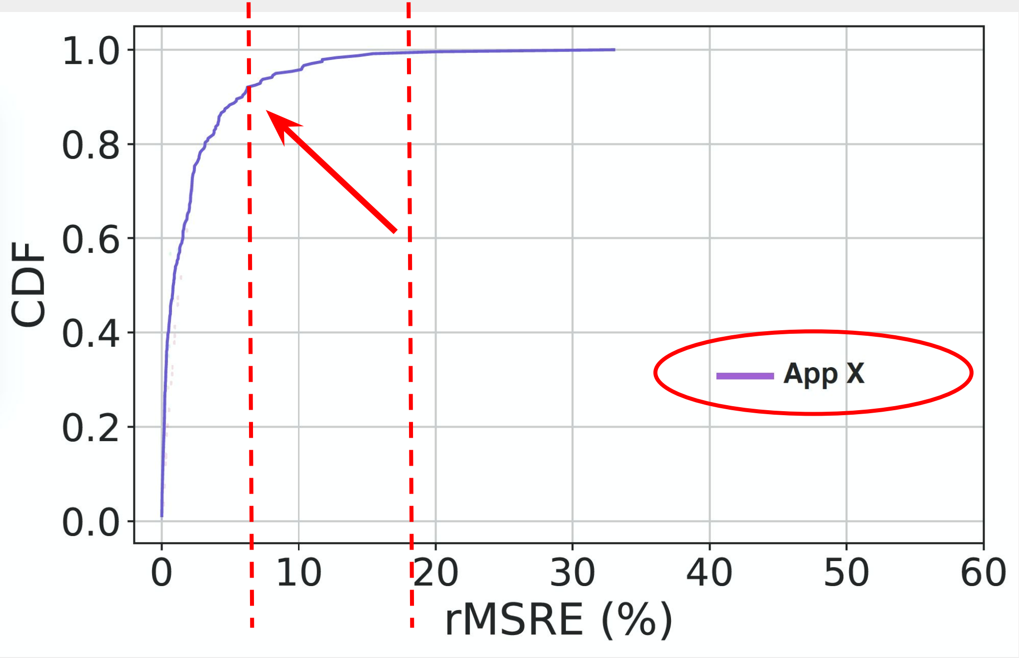
To obtain O-err:

- Allow Oracle to peek at both the error function and test data!
- BoM-err ≥ O-err

Best Case Test Results



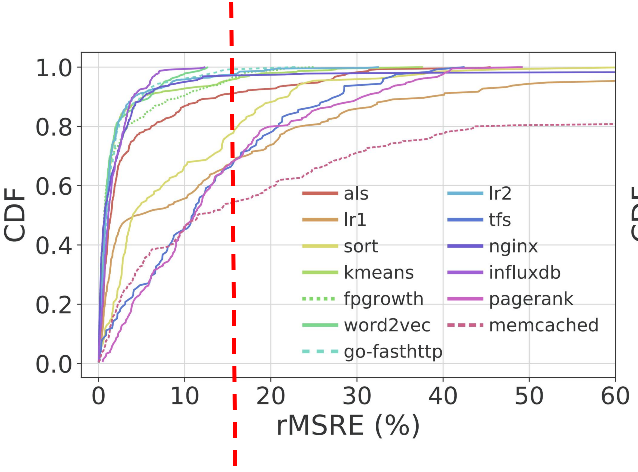
Best Case Test Results



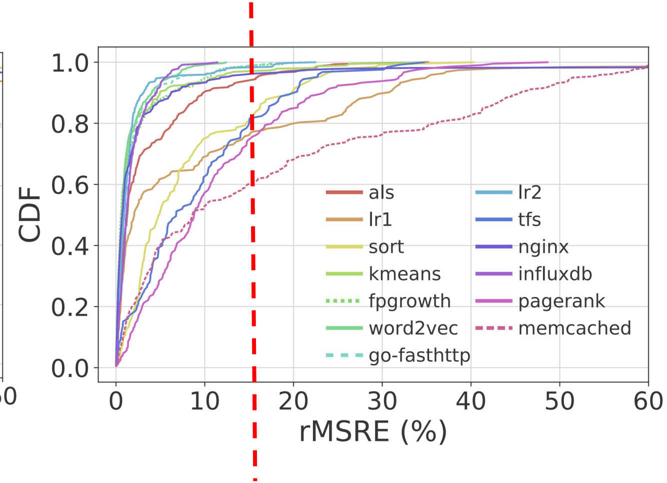
Error < 5% for 90% of predictions!

Error < 15% for ~99% predictions!

Best Case Test Results



(b) CDF of BoM-err in the BC test



(a) CDF of O-err in the BC test

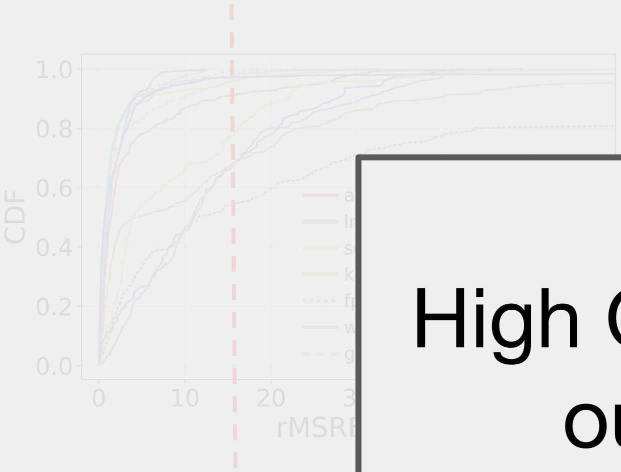
BoM-err: rMSRE from the most accurate model

O-err: rMSRE from the Oracle

Observations:

- Despite best-case assumptions, the BoM often fails to achieve high accuracy.
- Oracle errors (the lower bound) are high.

Best Case Test Results



(b) CDF of BoM-err

SRE from the
e model

E from the Oracle

High Oracle error even under our best-case setup!

(Accurate ❌)

Observations:

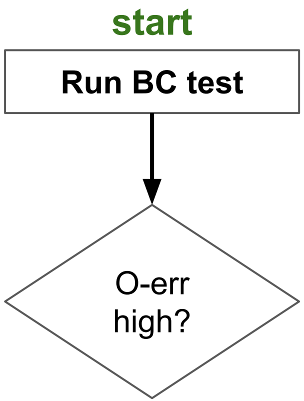
- Despite bes ccuracy.
- Oracle errors (the lower bound) are high.

Methodology

start

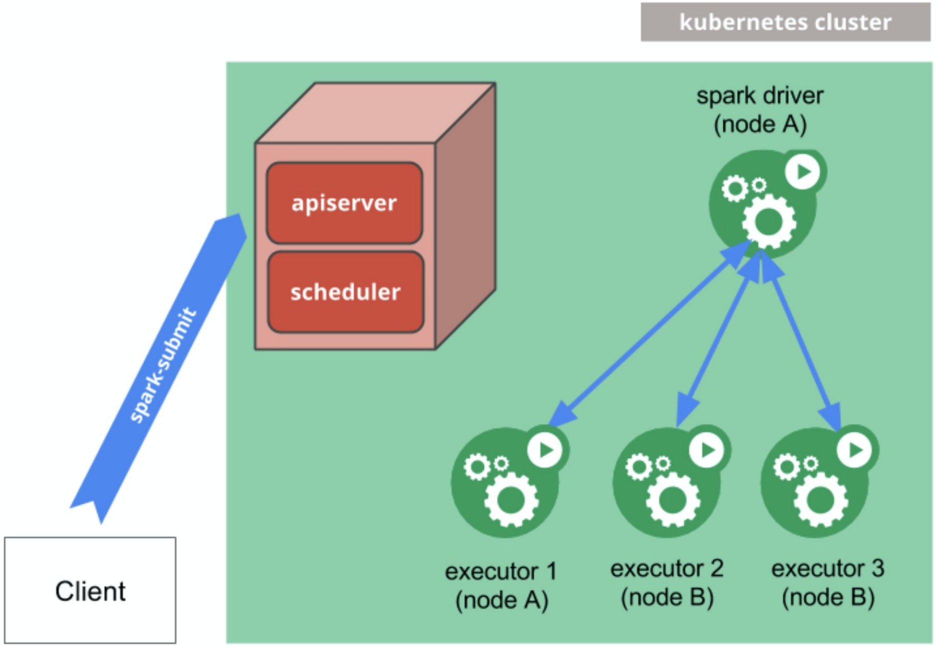
Run BC test

Methodology



Root Cause	Applications Impacted
Spark's "start when 80% of workers are ready" optimization	Terasort
Multi-mode optimization in JVM Garbage Collector	LR1
Non-determinism in Spark sched.	PageRank
HTTP redirects and DNS caching in S3's name resolution	KMeans, LR2, FPGrowth, ALS
Imperfect load-balancing at high load	TensorFlow serving
Variability in implementations of Cloud APIs (EC2)	memcached, Nginx

E.g., Spark worker readiness



https://spark.apache.org/docs/latest/running-on-kubernetes.html

- Spark launches a job once at least 80% of target workers are ready

Root-causes

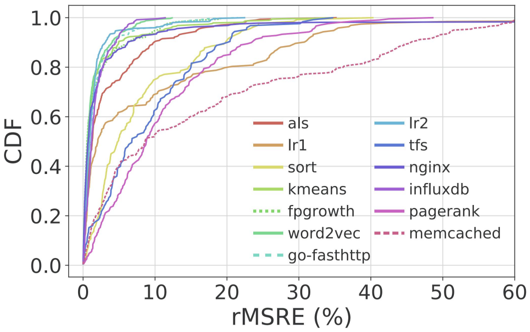
Fix?

Root Cause	Applications Impacted
Spark's "start when 80% of workers are ready" optimization	Terasort
Multi-mode optimization in JVM Garbage Collector	LR1
Non-determinism in Spark sched.	PageRank
HTTP redirects and DNS caching in S3's name resolution	KMeans, LR2, FPGrowth, ALS
Imperfect load-balancing at high load	TensorFlow serving
Variability in implementations of Cloud APIs (EC2)	memcached, Nginx

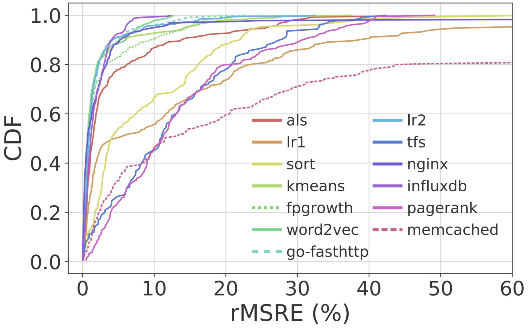
Root-causes Fix?

Root Cause	Applications Impacted	Modification
Spark's "start when 80% of workers are ready" optimization	Terasort	Disable optimization
Multi-mode optimization in JVM Garbage Collector	LR1	Avoid triggering, or disable, optimization
Non-determinism in Spark sched.	PageRank	Use deterministic data structure
HTTP redirects and DNS caching in S3's name resolution	KMeans, LR2, FPGrowth, ALS	Client-side caching of HTTP redirects (OR always redirect)
Imperfect load-balancing at high load	TensorFlow serving	Modified load-balancing policy to always favor local workers
Variability in implementations of Cloud APIs (EC2)	memcached, Nginx	Use AWS placement APIs / include inter-node RTTs as ML feature

With system modifications

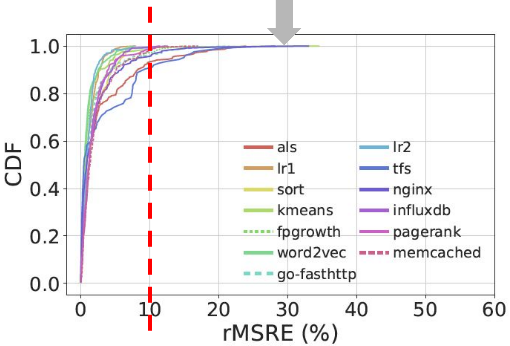


(a) CDF of O-err in the BC test

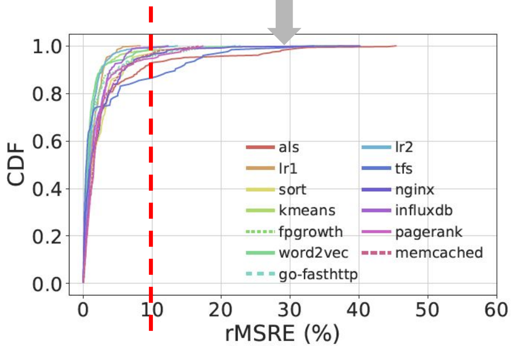


(b) CDF of BoM-err in the BC test

Before



(a) CDF of O-err in the BC test



(b) CDF of BoM-err in the BC test

After

- For all applications, Oracle error is now well within 10%!
- Best-of-Model error likewise!

All root-causes Fixes

Root Cause	Applications Impacted	Modification	Trade-off
Spark's "start when 80% of workers are ready" optimization	Terasort	Disable optimization	Decreased resilience to stragglers and worker failure
Multi-mode optimization in JVM Garbage Collector	LR1	Avoid triggering, or disable, optimization	Slower garbage collection
Non-determinism in Spark sched.	PageRank	Use deterministic data structure	None
HTTP redirects and DNS caching in S3's name resolution	KMeans, LR2, FPGrowth, ALS	Client-side caching of HTTP redirects (OR always redirect)	Decreased flexibility[6] (OR slower name resolutions)
Imperfect load-balancing at high load	TensorFlow serving	Modified load-balancing policy to always favor local workers	Load imbalance when each server has different numbers of workers
Variability in implementations of Cloud APIs (EC2)	memcached, Nginx	Use AWS placement APIs / include inter-node RTTs as ML feature	Cloud APIs expose more information (less flexibility)

- Trade-off between predictability and other design goals!
- E.g., disabling an optimization can lead to higher prediction accuracy but degraded performance

All root-causes

Fixes

Root Cause	Applications Impacted	Modification	Trade-off
Spark's "start when 8 are ready" opti			resilience to stragglers worker failure
Multi-mode optimiz Garbage Col			garbage collection
Non-determinism in			None
HTTP redirects and D S3's name res			flexibility[6] (OR slower ne resolutions)
Imperfect load-bala load			ance when each server nt numbers of workers
Variability in impler Cloud APIs (			APIs expose more tion (less flexibility)

- Trade-off b
- E.g., disabling an optimization can lead to higher prediction accuracy but degraded performance

These "fixes" require in-depth understanding of the app. and reasoning about trade-offs!

(Easy-to-use ❌)

Embrace variability: probabilistic predictions

- Idea: predicting a mixture distribution instead of a single value;
- Then, use the "modes" of each distribution as the "top-k" prediction value
  - E.g., id
- ML: Mixtu

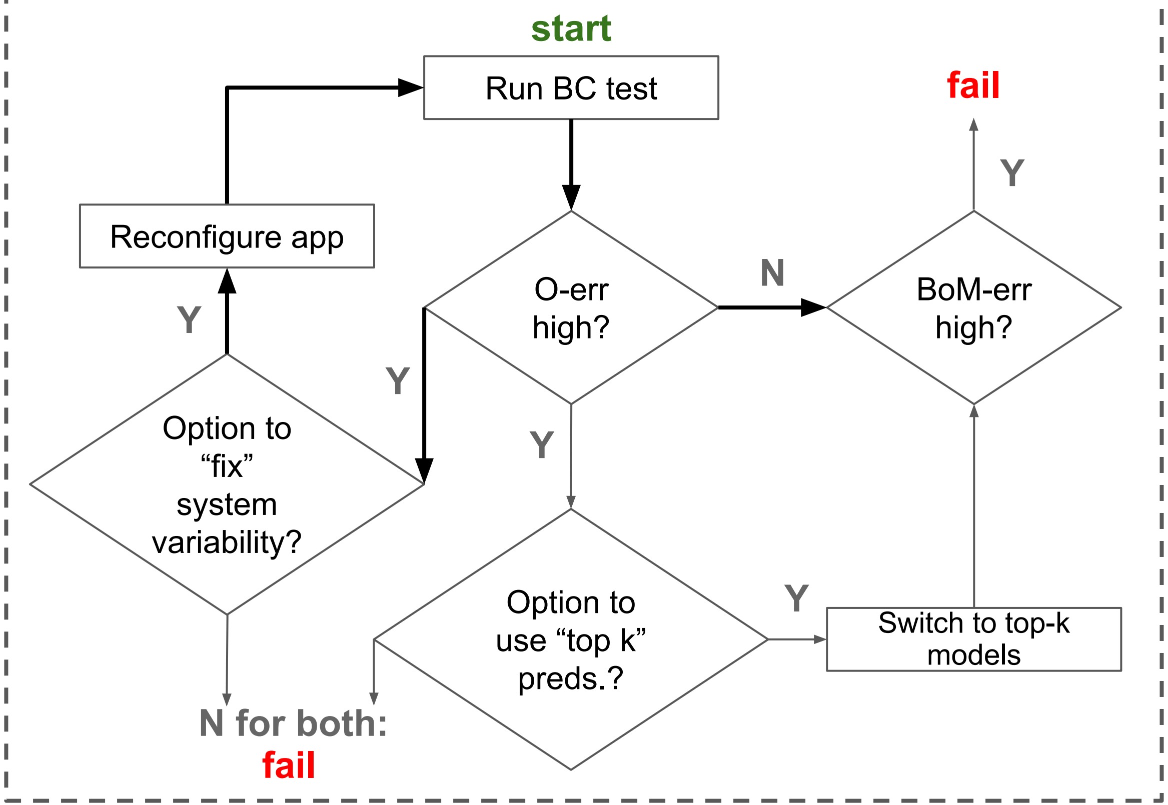
Significant decrease in BoM-err with top-3 (k=3) predictions!

But top-k predictions may not be useful to all cases!

(General ❌)

doi: https://doi.org/10.1371/journal.pcbi.1006869.g001

Methodology



So far, *best-case* setup only!

- one-feature-at-a-time
- seen-configuration
- no-contention
- identical-inputs

What if we go "beyond the best case"?

- Relaxing the **one-feature-at-a-time** assumption:
  - vary all parameters!
- Relaxing the **seen-configuration** assumption:
  - configuration-to-predict is never seen during model training!
- Relaxing the **no-contention** assumption:
  - use default/shared EC2 instances!
- Relaxing the **identical-inputs** assumption:
  - varied datasets (e.g., different random seeds in data generation)

Run on modified systems with the fixes!

What if we go "beyond the best case"?

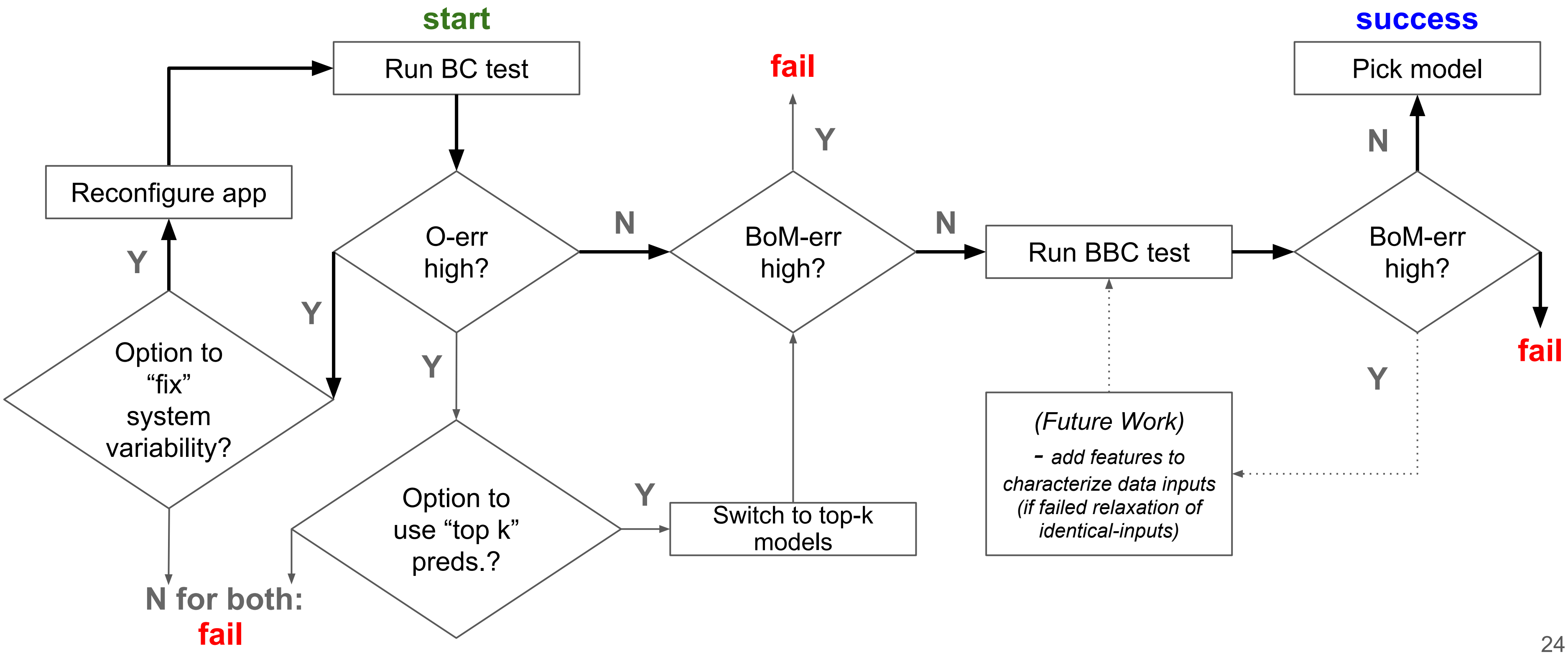
- Relaxing the **one-feature-at-a-time** assumption:
  - vary all parameters!
- Relaxing the
  - configura
- Relaxing the
  - use defau
- Relaxing the
  - varied datasets (e.g., different random seeds in data generation)

on modified
ems with the
fixes!

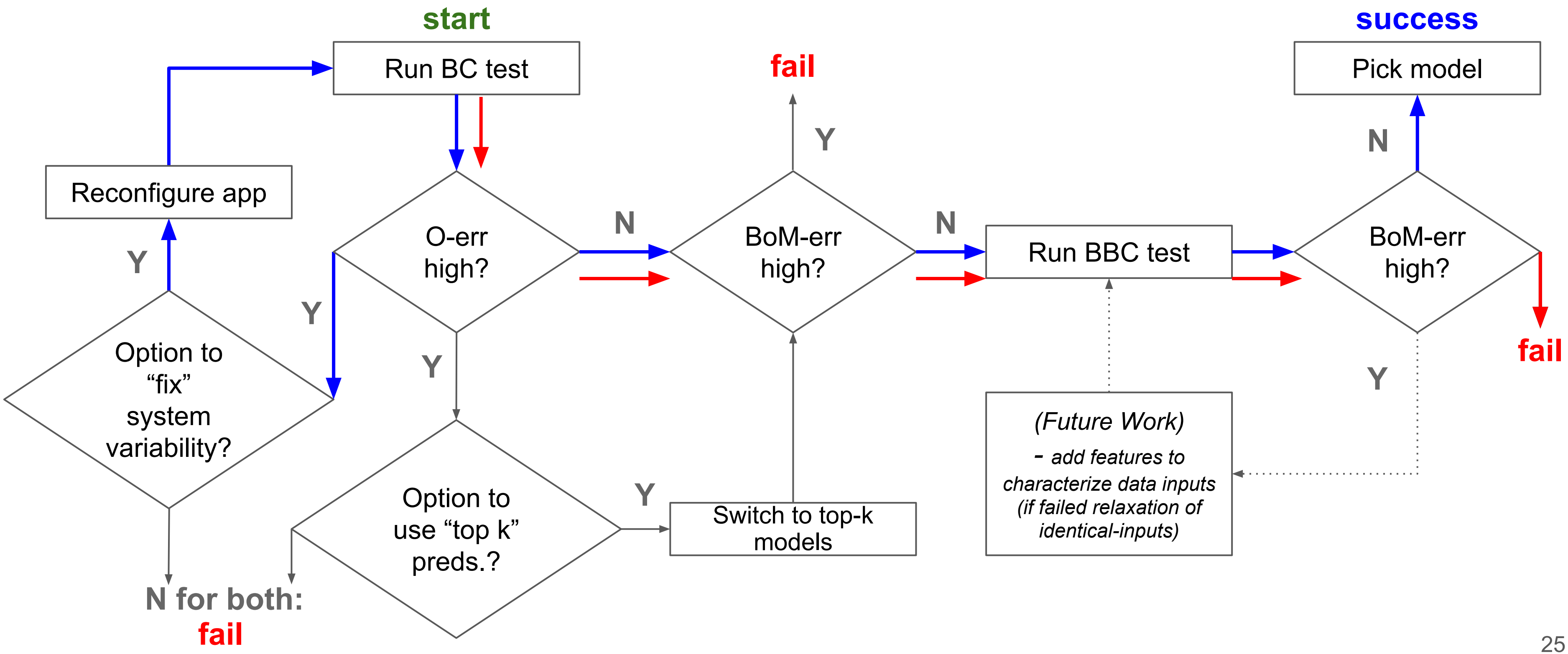
Prediction errors can remain high if the underlying performance trend is difficult to learn!

(General ❌)

Methodology Blueprint



Methodology Blueprint



Conclusion:

- Taken "out of the box", many apps exhibit a surprisingly *high degree of irreducible error*

- We *can* significantly improve the accuracy if we accept the loss of simplicity and/or generality:
  - modify applications
  - modify predictions
  - ..but they don't work in all cases

- Need a more nuanced methodology for applying ML

Conclusion:

- Accurate
  - precise predictions
- Simple/easy-to-use
  - in-depth understanding of the systems not required
- General
  - works across a spectrum of workloads and applications

Can ML provide an accurate, general, and simple performance predictor?

No.

Thanks!

Datasets: https://s3.console.aws.amazon.com/s3/buckets/perfd-data

Tools: https://github.com/perfd/perfd.git

Contact: silvery@eecs.berkeley.edu