Profiling and Predicting Application Performance on the Cloud

Matt Baughman*, Ryan Chard†, Logan Ward‡, Jason J. Pitt¶,
Kyle Chard§, and Ian T. Foster†‡§,
*Minerva Schools at KGI, San Francisco, CA, USA
†Data Science and Learning Division, Argonne National Laboratory, Argonne, IL, USA
‡Department of Computer Science, University of Chicago, Chicago, IL, USA
§Globus, University of Chicago, Chicago, IL, USA
¶Cancer Science Institute of Singapore, National University of Singapore, Singapore

***Abstract*—Cloud providers continue to expand and diversify their collection of leasable resources to meet the needs of an increasingly wide range of applications. While this flexibility is a key benefit of the cloud, it also creates a complex landscape in which users are faced with many resource choices for a given application. Suboptimal selections can both degrade performance and increase costs. Given the rapidly evolving pool of resources, it is infeasible for users alone to select instance types; instead, automated methods are needed to simplify and guide resource provisioning. Here we present a method for the automatic prediction of application performance on arbitrary cloud instances. We combine offline and online profiling approaches, using historical data gathered from non-cloud environments and targeted profiling runs on cloud environments to create a composite application model that can predict run times on a given cloud instance type for a given input data size. We demonstrate average error of 17.2% across nine applications used in production bioinformatics workflows. Finally, we evaluate an experiment design approach to explore the trade-off between the cost of profiling and the accuracy of our models. Using this approach, with no prior knowledge, we show that using 4 selectively chosen experiments we can achieve performance within 30% of a model trained using all instance types.**

Keywords*—*Cloud profiling, performance prediction, instance selection, transfer learning

I. Introduction

The pervasiveness of on-demand, pay-as-you-go cyber-infrastructure now makes it possible to acquire appropriate resources for most applications at the click of a button. The enormous and ever-growing selection of resources provided by cloud platforms is a key reason for this success. For example, Amazon Web Services (AWS) alone now provides over 100 distinct *instance types*, each a virtual computer with specific resource capabilities (e.g., CPU, memory, disk, and network). However, without expert knowledge of both the inner workings of an application and the capabilities of all available instance types, it is difficult to select the *best* (from the perspectives, for example, of cost, performance, or reliability) instance type for a given application. Moreover, selecting the wrong instance type can greatly reduce application performance or be unnecessarily expensive [1], [2]. Thus, the need for expert knowledge is a significant obstacle to effective cloud computing.

While prior work has focused on the related problem of predicting application performance in homogeneous [3], [4] and heterogeneous [5], [6] environments, there has been relatively little prior research focused on the extremely heterogeneous cloud computing environments. Given the enormous search space—a near infinite combination of applications, configurations, and instance types—it is infeasible for users to estimate how an application might perform on a given instance type or with a particular configuration. Instead, new automated methods are needed to predict performance given features describing the application and available instance types.

Here we focus on predicting the execution performance of various genomics applications when deployed on arbitrary cloud instance types and with different input datasets. We explore a set of nine genomics applications used in two production workflows to elucidate a range of different execution characteristics. We explore and then combine two different approaches for measuring and predicting performance: the first without any prior knowledge of an application and the second when prior execution history is available from a non-cloud environment. To address the needs of the first case, we present an automated, parallel profiling system that can adaptively conduct experiments to explore the search space. In the second, we develop a composite function to merge two distinct models and apply a transfer learning approach to calibrate this model in a cloud environment. This reuse of external histories allows us to bootstrap our prediction models without requiring extensive experimentation on cloud instances. Collectively, our models obtain a mean absolute percent error (MAPE) across 9 genomics applications of 17.2%. Finally, given the potentially high cost of exhaustive profiling experiments, we explore an experiment design-based approach to selectively choose instance types on which to profile performance. Our results show that with as few as 4 experiments we can achieve within 30% MAPE of the best accuracy when conducting experiments on every instance type.

The remainder of this paper is organized as follows. In Section II we motivate our work by describing two real-world bioinformatics workflows and a set of nine bioinformatics applications. We outline our profiling approach in Section III. In Section IV we summarize our profiling datasets before presenting and evaluating our predictive models in Section V. In Section VI we explore active experiment design to investigate the trade-off between profiling accuracy and profiling costs. Finally, we discuss related work in Section VII and summarize our contributions in Section VIII.

II. Background

The last several years have seen remarkable advances in bioinformatics and genomics, with data-intensive and computational methods leading to new discoveries [7]. As a by-product of these new approaches, combined with increasing sample sizes, modern bioinformatics is no longer possible on a personal laptop. Instead, significant computing resources, such as those offered by compute clusters and cloud providers, are required for most analyses. However, the dynamic and murky bioinformatics software landscape can lead some researchers to make ill-advised decisions concerning scientific objectives, computational needs, and in turn, budget. Beyond simply selecting the right application for a task, many applications perform quite differently on heterogeneous resources due to inherent bottlenecks and complex configuration options.

Computational genomics research is often conducted using scientific workflows that comprise many distinct applications. Each of these applications may perform widely different tasks, from computationally intensive variant calling to memory intensive sorting, and for each task there are myriad comparable applications, each with different performance, accuracy, and associated costs. The performance of an application is primarily related to its configuration and input dataset combined with its ability to make use of available resources. In many cases the time required to execute a workflow can vary by several orders of magnitude. For example, the same genotyping workflow when used to analyze a relatively small exome (tens of GBs) on modest resources (16 CPUs, 32 GB RAM) takes approximately 4 hours, however when applied to an entire human genome (hundreds of GBs) this time increases to more than 24 hours.

We focus on genomics workflows, as they are commonly executed on cloud platforms and the individual applications exhibit vastly different characteristics. Specifically, we seek to model application performance on instance types with vastly different capabilities and when using various input datasets. The aim of our work is to predict performance such that it can be used to optimize and automate resource selection as well as understand trade-offs between execution time and cost. To explore this space, we focus on two production genotyping workflows that analyze NGS Illumina sequencing data through the SwiftSeq framework [8].

A. SwiftSeq

SwiftSeq is a computational framework for developing highly parallel and efficient processing workflows of DNA sequencing data. It is implemented as a polyglot program in Python and uses either the parallel scripting language Swift [9] or the parallel scripting library Parsl [10] for execution. SwiftSeq is designed for usability with a focus on sleek interfaces that expose bioinformatics processes while abstracting the complexities of parallel programming. SwiftSeq allows researchers to compose arbitrary workflows using a suite of best practice applications. It codifies these workflows in a simple JSON format which can be easily modified to tune performance, switch algorithms, and change parameters. SwiftSeq provides a library of best practice workflows. These are highly optimized workflow definitions for both germline and tumor-normal pair analyses. SwiftSeq's parallel engines mean that there is no difference to the user between analyzing one or one thousand samples. SwiftSeq tasks are both reliable and robust, providing built-in fault tolerance through fine-grained failure detection and the ability to restart workflows. These advantages are some of the many reason why SwiftSeq is rapidly becoming a common tool in bioinformatics. It has been used to analyze many thousands of genomes across dozens of sites. In one example, SwiftSeq was used on Beagle2, a supercomputer at the University of Chicago, to uniformly align and call genetic variants on over 11,000 exome and 500 genome sequencing samples, a once Herculean undertaking that required large consortia [11]. These harmonized datasets have allowed researchers to better understand clinical disparities in Nigerian breast cancer patients [12] as well as to show that the high burden of inherited, deleterious mutations is associated with earlier cancer diagnoses [13].

B. Genotyping workflows

A common use of SwiftSeq is for genotyping, the process of discovering genetic differences between individuals and reference sequences. Standard genotyping workflows include multiple applications with quite different resource requirements, from long-lived, memory intensive applications to others that are short and CPU intensive. SwiftSeq is designed to be flexible and therefore different genotyping workflows can be easily defined by swapping alternative implementations of the same function. For example, researchers often use different, or multiple, variant callers, such as Platypus and HaplotypeCaller. These workflows can be used to analyze both exomes and genomes. In the remainder of the paper we primarily focus on the analysis of a single $\sim$10 GB exome dataset, using two common variants of the genotyping workflow (Platypus and HaplotypeCaller). The genotyping workflows are comprised of the following nine applications with each implementing distinct bioinformatics tools.

- BwaMem [14]: Uses BWA MEM to map low-divergent sequences against large reference genomes.
- RgMergeSort: Groups of aligned reads are combined via mergesort (Sambamba [15]) and output is split into contig (chromosomal) BAM files with BamUtil.
- PicardMarkDuplicates [16]: Uses Picard Tool's MarkDuplicates utility to remove duplicate reads from each contig BAM file.
- IndexBam: Indexes a BAM file using Samtools [17] to improve search efficiency.
- PlatypusGerm [18]: Calls germline variants with Platypus.
- HaplotypeCaller [19]: Calls germline variants with HaplotypeCaller.
- ContigMergeSort: Performs a Sambamba mergesort to combine contigs.
- SamtoolsFlagStat: Extracts alignment metrics via Samtools [17] Flatstat.
- ConcatVcf: Concatenates multiple VCF (Variant Call File) files into a single output VCF file.

III. Profiling

The ability to create, and tune, machine learning models is dependent on sufficient training data. To obtain this training data we explore two possible approaches. The first relies on exploiting information obtained from executions on external resources (e.g., a local cluster or supercomputer); the second uses an automated profiling approach to conduct experiments to measure performance on different instance types, with different configurations, and different input data.

A. Profiling using external histories

Scientific workflows are often developed and executed in a variety of environments before transitioning to the cloud. This prior execution information can be used to develop predictive models that can later be mapped to cloud environments using *transfer learning*. In the case of SwiftSeq we have obtained nearly one million task executions on Beagle2, a Cray XE6 supercomputer. Specifically, we have collected detailed execution traces and input metadata for variant calling workflows used to process tumor-normal pairs for over 2,100 exome sequencing samples. The input exome datasets range in size between 2 GB and 40 GB and have between 1 and 12 distinct readgroups. This dataset provides a basis to investigate how the applications behave when processing different input data. This workflow is subtly different from the genotyping workflows presented in Section II-A as it replaces the two variant callers, HaplotypeCaller and Platypus, with two somatic variant callers, specific to tumors, called Mutect and Strelker. However, we ignore these applications when building and evaluating our models so as to retain models only on the applications that are consistent across both sets of data.

B. Profiling using automated experiments

In the absence of prior executions and to gather cloud-specific execution data we also explore proactive experiments to collect execution information. To do so, we have developed a parallel profiling system that is able to provision arbitrary cloud instances and then parameterize, execute, and monitor executions on those instances. We build upon our prior work creating an automated profiling service [20], [21] to develop a simplified parallel profiling system.

Our parallel profiling system is built using Parsl, a parallel scripting library for Python. Parsl allows for the simple construction of parallel scripts that are comprised of both Python and external command line applications. Parsl manages the execution of the script on arbitrary execution environments including laptops, supercomputers, and clouds. We use Parsl to construct simple scripts to execute a given SwiftSeq application on a set of cloud instances using various input datasets and settings. Using Parsl's AWS executor, we define the set of instance types and a preconfigured Amazon Machine Image (AMI) with all applications preinstalled. Parsl uses these settings to provision cloud instances from the spot market. When the spot request is granted, Parsl deploys an IPyParallel engine on the instance. This engine connects back to the Parsl executor and begins accepting profiling jobs. Our profiling system then dispatches jobs in the form of Python scripts to the instance. Each Python script, dynamically configures the execution environment by first downloading the input data from S3, installing the Python Resmon tool to record resource usage, configuring the execution environment, and customizing the SwiftSeq execution script. This process enables us to dynamically set execution parameters, such as the memory allocated to a specific application or the number of SwiftSeq processes to deploy. We customize these values for each instance type to reflect its capabilities and available resources.

Immediately prior to invoking the SwiftSeq workflow the profiling script installs Resmon to capture resource statistics and utilization counters for CPU, memory, disk, and network utilization. Resmon is configured to run as a daemon, collecting data every second and appending it to a local file. Once the SwiftSeq workflow completes the profiling script renames both the SwiftSeq and Resmon logs and publishes them to an S3 bucket.

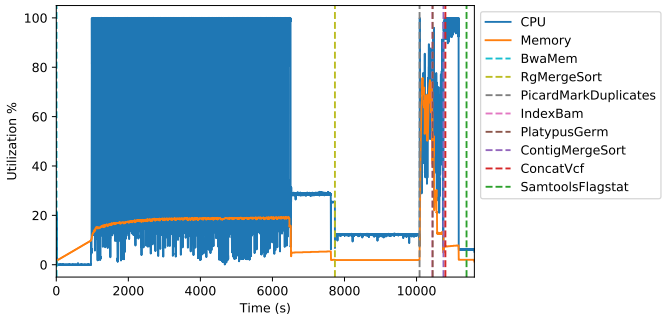


Fig. 1: The execution profile of a Platypus genotyping workflow executed on an `m4.4xlarge` instance. The starting time of each application is denoted by a vertical dotted line.

We have used the profiling system to execute the genotyping workflows on 14 distinct AWS EC2 instance types with varying resource capabilities. We focus specifically on the M4, R4, and C5 instance families, ranging from 2 to 72 vCPUs and from 2 GB to 488 GB of memory. The CPU and Memory utilization from one execution of the Platypus genotyping workflow is shown in Fig. 1. The figure highlights the distinct applications that comprise the workflow and exemplifies their different execution characteristics.

IV. Measuring application performance

The performance of a bioinformatics application can depend on a wide range of features, from input data size to the application's ability to take advantage of multiple cores. In this section we briefly review the performance of our nine genomics applications when executed on the Beagle2 supercomputer and on EC2.

A. Application performance on cloud instances

We used the profiling system described in Section III-B to deploy and monitor the two genotyping workflows across 14 different instance types from three instance families. Each experiment was repeated at least three times. The profiling system captures both Resmon resource utilization counter values and, via the SwiftSeq execution log, the start and end time for each process executed by a workflow. As some applications are parallelized and start multiple processes, we

aggregate processes by name to compute the total execution time of a given application.

Fig. 2 shows the range of execution times for each application across the 14 instance types. Execution times range from less than five seconds (ConcatVcf) to more than nine hours (HaplotypeCaller). We see that some applications are more sensitive to instance type than others. For example, IndexBam has a maximum-to-minimum execution time ratio of 19.4 (i.e., the longest execution time was 19.4 times more than the shortest), whereas SamtoolsFlagstat's ratio is only 1.23.

Fig. 3 illustrates the relationship between execution time, CPU, and memory for several applications, highlighting the different instance families. We note several observations. First, while some applications are CPU and/or memory bound (e.g, BwaMem, ConcatVcf), some display constant performance irrespective of resources (e.g., RgMergeSort), and others seem to be unrelated to available resources (e.g., PlatypusGerm). Second, scaling relationships are most obvious within an instance family and performance is often proportional to the price of the instance (e.g., as shown in Fig. 4 for the HaplotypeCaller genotyping workflow). And third, there are significant performance differences between instance families as evidenced by different execution times when using instance types with relatively similar resources.

Fig. 5 shows how the total workflow performance differs across instance types and instance families as a result of the cumulative application performance in the workflow, in this case Platypus. We see that performance typically increases within an instance family up until a point at which additional resources do not improve performance. Note also that the workflow performs best on the compute-optimized (C5) family of instances. It is these relationships that we aim to capture in the models presented in Section V.

B. Application performance on a supercomputer

We have analyzed the SwiftSeq execution logs from over 2,100 exome tumor-normal variant calling workflows, consisting of over 900,000 successfully executed tasks on homogenous resources. We have aggregated the execution data to extract each application's execution history and combined these with the workflow metadata to associate information regarding the input data size.

Fig. 6 shows the relationship between execution time and input data size for several applications. These figures highlight the highly variable performance irrespective of input data size. While we expected to see some differences, the observed results differ by several orders of magnitude in some cases. It is possible that resource contention affected performance. For example, execution of thousands of concurrent applications, many of which that have not been designed for parallel execution and that access shared reference genomes or inefficiently access files stored on a shared file system is one potential reason for this contention. Furthermore, Beagle2 is a production, multi-tenant supercomputer, and therefore execution may be affected not only from contention among SwiftSeq workflows, but also from external, unknown usage.

V. Predictive models

Based on the data collected in the previous section we now develop models to predict execution time given instance type and input data size. We present three distinct models: first, using data generated by the profiling system we create a model to predict application execution time on a given instance type; second, we use execution traces from the Beagle2 supercomputer to model the execution time of an application given different input data size; finally, we combine these two models to construct a composite model to predict execution time given arbitrary input data size and instance type. The respective model parameters are reinitialized and then fitted with unique values in every run to ensure independence between evaluations. We evaluate these models by using the deviation of the actual execution time from the predicted value from the respective regressions. We then calculate the mean absolute percent error (MAPE) for each model respectively.

A. Instance type model

We see in Fig. 3 that some applications scale well with respect to resources, while others do not. Of those that do scale well, they tend to scale well within instance families, where the amount of memory per core is constant between instance types. Based on the observations in the previous section, we developed a regression-based model to predict execution time for each application and instance type.

To profile each of these applications, we implemented a regression using SciPy's curvefit function which uses nonlinear least squares. Based on the observed nature of how these applications scale, we selected the formula below to capture and fit scaling from both available memory and vCPUs.

$$f(vCPUs, RAM) = \frac{1}{a * vCPUs + b * RAM} + c$$

Regressing this equation using parameters a, b, and c over our data, we can represent the portion and magnitude of the scaling that is dependent on vCPUs and memory, respectively. The structure of this regression also allows us to create composite functions, as discussed in Section V-C. Finally, fitting only three parameters allows initial fitting with limited quantities of data, as well as a fitting process that can be performed in milliseconds on a commodity laptop.

Evaluation of our model on the two genotyping workflows, containing nine individual applications, we were able to achieve a fit with a mean absolute percent error (MAPE) of 17.2%. Individual accuracy for each of the applications is reported in Tables I and II for the Platypus and HaplotypeCaller workflows, respectively. In these tables, we also provide a baseline MAPE that is calculated in reference to the application's mean performance. We use this as a baseline as it functions as a random selection analogue given that the expected performance of a randomly selected instance will be the mean performance. An example of how our model fits the BwaMem application is shown in Fig. 7. Most applications achieve a mean validation error of less than 20% and those that do not achieve such accuracy exhibit the highest dependence on instance family qualities, like compute optimization and memory per core. Such a fit shows that the execution of these workflows on AWS is both dependable and predictable and that our model is suitable for providing accurate performance

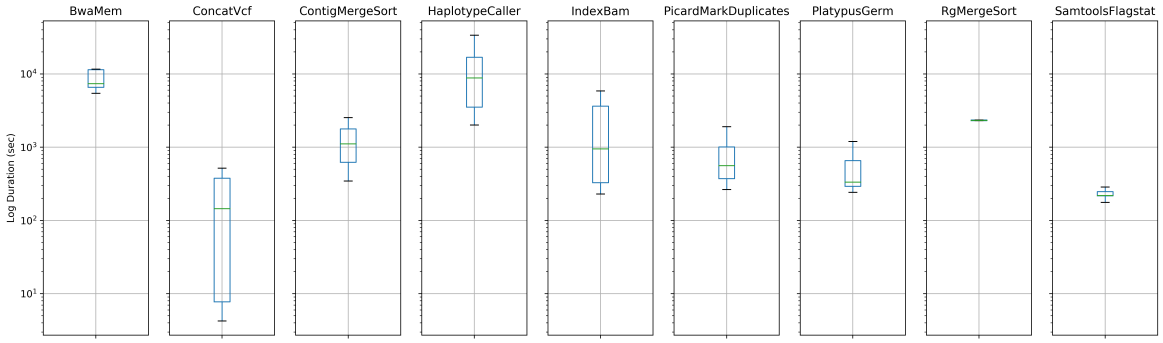
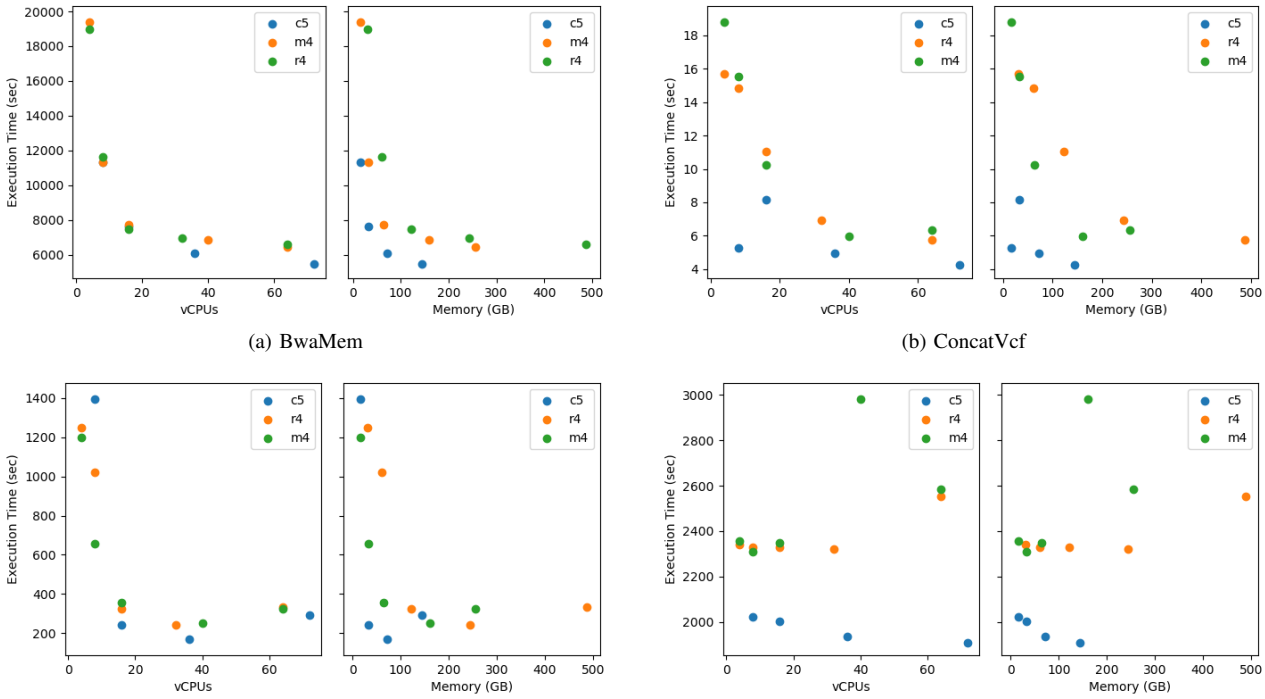


Fig. 2: Average application performance across instance types.



(a) BwaMem

(b) ConcatVcf

(c) PlatypusGerm

(d) RgMergeSort

Fig. 3: Execution time for four genomics applications with identical input data, on instance types with varying numbers of vCPUs and varying amounts of memory.

TABLE I: Model accuracy for Platypus workflow

Application	MAPE	SD	Baseline MAPE
BwaMem	8.0	6.8	33.7
RgMergeSort	10.6	6.3	10.6
PicardMarkDuplicates	10.3	7.9	68.2
IndexBam	34.7	21.3	57.5
PlatypusGerm	33.3	26.2	76.0
ContigMergeSort	13.5	16.1	56.5
ConcatVcf	30.5	34.9	51.2
SamtoolsFlagStat	8.8	3.2	9.4

TABLE II: Model accuracy for HaplotypeCaller workflow

Application	MAPE	SD	Baseline MAPE
BwaMem	12.0	16.1	40.8
RgMergeSort	8.4	6.6	8.4
PicardMarkDuplicates	11.8	9.9	64.9
IndexBam	44.0	50.0	142.4
ContigMergeSort	26.1	21.9	31.0
ConcatVcf	8.2	5.8	10.4
SamtoolsFlagStat	9.8	8.5	14.5
HaplotypeCaller	5.7	4.8	144.1

predictions across a diverse range of applications and instance types.

B. Data size model

Given the varied performance measurements from execution on Beagle2 we choose to model execution behavior

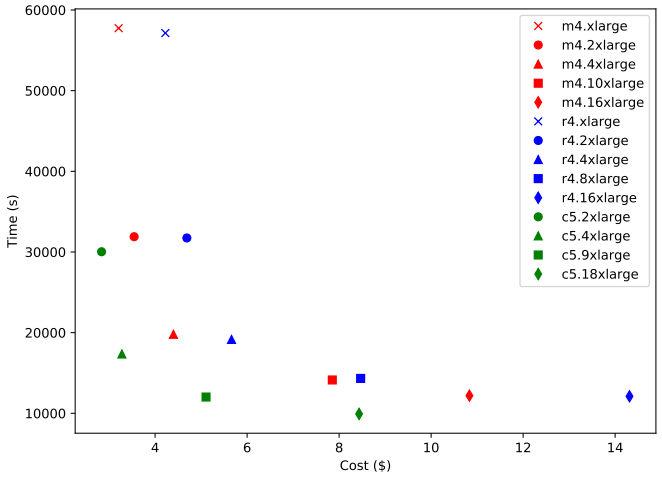


Fig. 4: The performance and cost of performing the entire HaplotypeCaller genotyping workflow when using different AWS instance types.

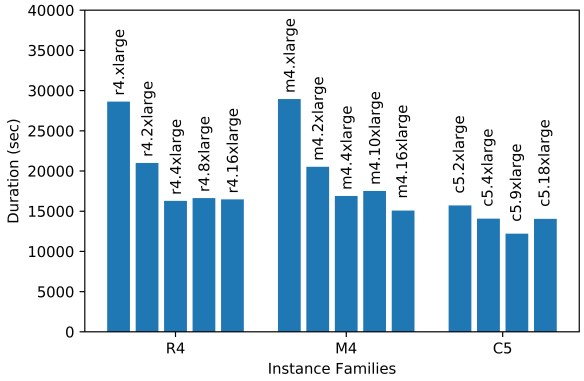


Fig. 5: Execution time for the Platypus workflow across instance type, grouped by instance family.

based on only the highest performance executions. This is appropriate as the highest performance successful executions can be assumed to be those with the least contention and thus the most representative of an execution performed on a dedicated resource. We thus fitted a lower bound to capture the "best case" runs (as shown in Fig. 6).

We use a similar regression model to that defined above, but with an additional variable exponent, *l*, in the denominator because larger input data sizes typically take longer to run (i.e., this is a growth model) and different applications scale with different computational complexities:

$$\begin{aligned} g(size) &= \frac{k}{size^{l}} + m \\ &= k \times size^{-l} + m \end{aligned}$$

To fit the lower bound, we binned the data and retained the smallest values within each bin. The exact number of bins and values retained were calculated dynamically on an application by application basis to maximize accuracy. To avoid overfitting, a condition was added that we needed to retain at least twice the number of observations as we have parameters in our model. This quantity was selected as it forced binning optimization away from selecting increasingly larger bins while being low enough to allow the data binning process to still include data from each bin, thereby ensuring a level of representativeness. Due to the much larger quantity of data and the increased complexity of the regression equation, the mean fitting time for this model was 0.032 seconds running on a commodity laptop.

TABLE III: Model accuracy for lower bound fit

Application	MAPE	SD
BwaMem	5.4	5.1
RgMergeSort	1.4	1.0
PicardMarkDuplicates	8.0	5.4
IndexBam	14.5	8.7
ContigMergeSort	4.3	4.0
SamtoolsFlagStat	12.6	10.0

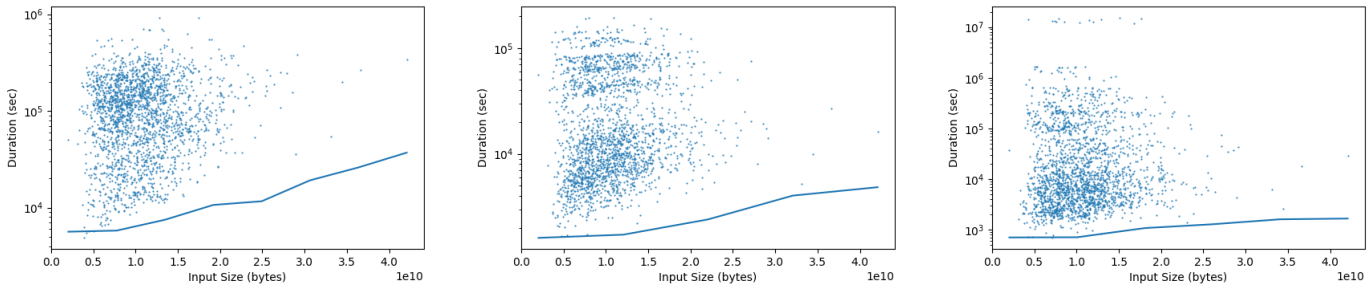
Using this model, we were able to achieve relatively good lower bound fits. Table III shows the accuracy for each of the applications. Note: we include only applications executed before the variant callers as these workflows used alternative variant calling algorithms. In each case, the mean absolute percent error is below 15% and in two cases (RgMergeSort and ContigMergeSort) is below 5%. We believe this represents an adequate lower bound fit and, as such, allows for fair representation of optimal runtime.

C. Combining performance models

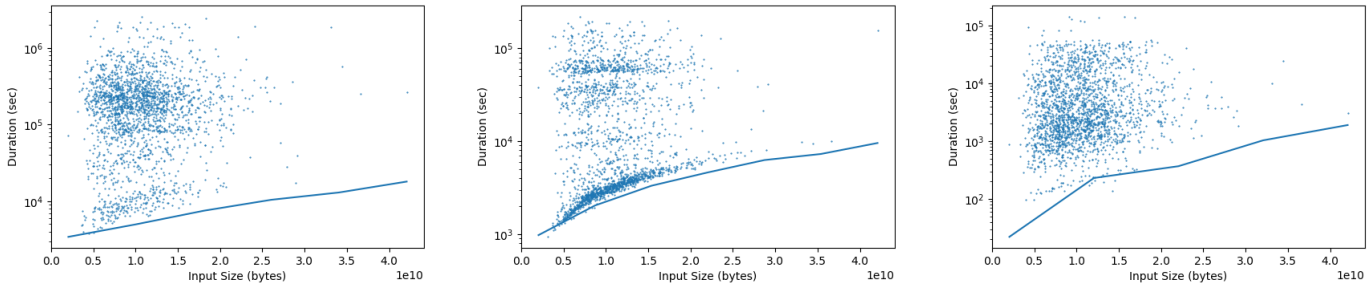
We developed the two models f and g defined above by using manual profiling on cloud instances and historical data from Beagle2, respectively. We now seek to combine these models so that we can use them to predict performance on arbitrary instance types given arbitrary input data sizes. To do so, we apply a transfer learning approach in which we first define a composite function such that the input for one model is the output from another and then transfer the data size model to the cloud by training on a single observation (i.e., instance type). While this approach requires the addition of an internal regularization constant (within the exterior function to allow us to fit the models at the point of known overlap), the composite function allows us to easily relate scaling factors resonating from both of the respective models:

$$G(size, vCPUs, RAM) = f(vCPUs, RAM) \times g(size) \times r$$

To create this three-dimensional model, we retain the crucial scaling parameters from both models and solve for *r*, which is our new regularization factor. This is possible as we used a single 10 GB input file to train our function *f()*. The new regularization factor can be trained using only one additional runtime observation. We selected one instance at a time from our validation set and trained for *r* using that observation while keeping all other parameters constant. Iterating through all possible instance types from our validation set allows us to find the expected value for model error (MAPE) had we selected a single instance type from the set randomly. We validated our model using the BwaMem application and found a mean MAPE of 38% and a minimum of 22.9%. Our combined model performs best for small and compute optimized instance



(a) BwaMem (b) ContigMergeSort (c) IndexBam



(d) PicardMarkDuplicates (e) RgMergeSort (f) SamtoolsFlagstat

Fig. 6: Execution time vs. input data size for six common applications executed on Beagle2. Each dot indicates an execution of the application. The line shows a fitted lower bound.

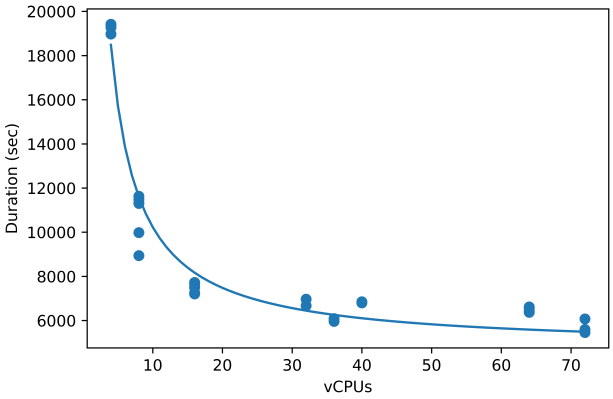


Fig. 7: The fit for the BwaMem application given different resource capabilities.

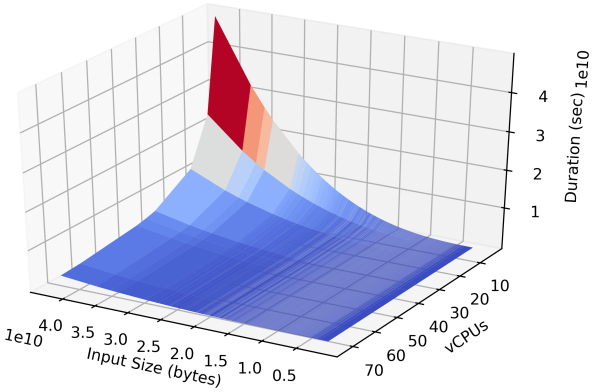


Fig. 8: Combining both the predictive models.

types, with only `16xlarge` or larger instance types having a MAPE greater than 32%. This shows that we can successfully use the data type model (trained on executions in a non-cloud environment) by adding a single observation from cloud execution to construct a multi-dimensional model with useful accuracy. An example of the combined model for the BwaMem application is shown in Fig. 8.

VI. Experiment Design

Our profiling approaches to this point assumes significant knowledge of application performance. However, gaining this knowledge can be expensive in the case of an unknown application. Ideally, we would like to obtain just a few measurements for a new application, for example by considering only some representative instance types. Such information could then be used to guide additional profiling, or potentially as part of an online scheduling tool that could explore the search space by selectively assigning tasks to instance types and learning from execution performance. To minimize the number of instance types required to generate an application profile we explore an active experiment design approach [22].

In order to intelligently select the "best" instance types (i.e., those that would provide the highest marginal reduction in error across the model search space), we have implemented a simple approach that calculates where the search space has the

least knowledge—where *density* of observations is the least. By selecting computational resources strategically to increase our knowledge of the solution space, we can then achieve a better model fit not only for that space with little prior information but also for the rest of the model as this new observation can act as a new "anchor" to help validate and improve the model.

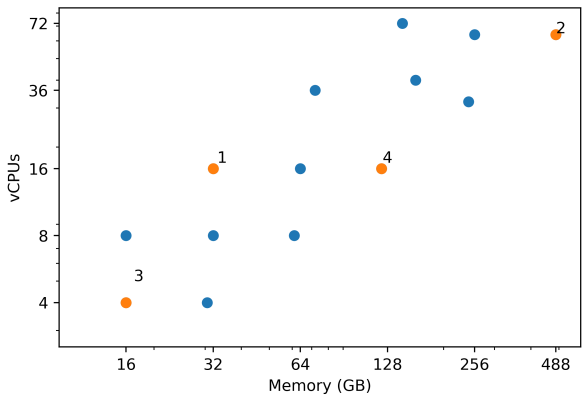


Fig. 9: An example of the experiment selection algorithm at work, showing the first four instance types selected as it explores the search space.

Figure 9 shows how instance types are selected for a new application. The algorithm is simple—it first selects the most central instance type and then progressively selects instance types in the most *remote* part of the search space (calculated using a distance measure). More specifically, given a set of existing data points (given by prior history), we will then launch the application on an instance type that is the most remote in terms of instance CPU and memory. As there are few (relatively speaking) instance types, we can compute relative remoteness by calculating the distance between each instance type for which we have data and for which we do not. Though this is inefficient, the small scale of this problem makes it relatively inconsequential. Other methods such as dynamic programming and prior state retention could be explored to improve performance.

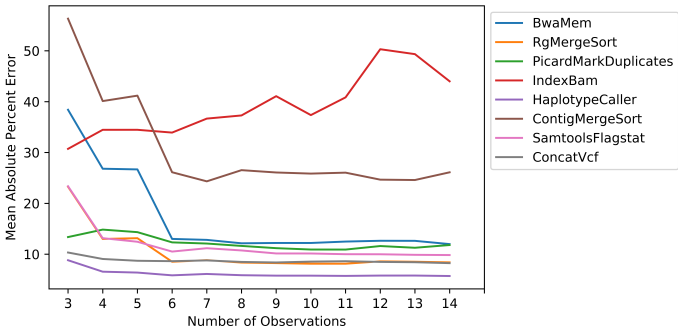


Fig. 10: Validation of the active experiment design approach.

We have applied our experiment design approach to model each of the applications in the Haplotype genotyping workflow. We assume no prior knowledge of the applications and therefore begin by selecting an instance type from the center of the search space, `c5.4xlarge`. As depicted in Figure 9, we select the second observation as a `r4.16xlarge` instance type. Subsequent observations then select the most distant instance type before gradually filling in the search space.

TABLE IV: Difference in MAPE for models trained with varying numbers of observations from the MAPE obtained by the model trained with all observations.

Observations	Difference
3	79.3
4	29.9
5	19.8
6	2.3
7	1.9
8	0.9
9	0.6
10	0.1
11	1.9
12	1.8
13	1.6
14	0.0

Figure 10 shows the MAPE achieved for each application as the number of observations increases. We see that accuracy generally increases with additional observations, except in the case of IndexBam, where accuracy declines as more observations are made. This is due to the tool being tightly CPU bound; it performs roughly 25% better on the compute-optimized C5 family of instances than on any other family. Thus, the first instance type selected, `c5.4xlarge`, is in fact an accurate representation of performance on the C5 family. Subsequent observations on other instance types only increase the prediction error. As the experiment selection approach encourages the diverse selection of instance families, the scaling properties of the model are not efficiently found between instance families. We aim to address this problem in later work by extending our experiment selection techniques to explore scaling both within and across instance families to further optimize the selection of observation points.

Table IV shows the difference between the average MAPE at each observation and the final MAPE as we select and add instance type information to our model. In other words, when we have three observations on which to train our model, we achieve a validation MAPE that deviates by 79.3% from our final MAPE value once experiments are conducted on all 14 instance types. The table also shows that our approach allows accurate prediction to within 30% of the final MAPE on average with only four observations and within 2.3% after six observations. We conclude that there is little benefit to be gained by conducting more than seven experiments.

VII. Related Work

The problem of selecting suitable resources for a given application is ubiquitous across cloud providers. Google now provides a *VM sizing recommendation* service [23] that monitors system utilization and recommends appropriately sized virtual machines. Although such a service relies on understanding prior execution information, it does not enable forecasting for variable usage requirements (e.g., in terms of applications, data sizes, settings), or let users explore trade-offs between application performance and cost.

There has been considerable work to determine the execution requirements of applications, most often in high performance and distributed computing environments [6], [3], [24], [4], [25], [26]. Recently, researchers have begun to explore predictive modeling as a solution to the instance selection problem. PARIS [27] is a data-driven system that aims to select the best instance type given minimal data collection. PARIS uses a hybrid online and offline data collection and modeling system to accurately predict workload performance. However, unlike our work, PARIS focuses on video-encoding workloads and serving latency. Our work targets scientific uses cases that differ substantially from many business-oriented cloud uses. For example, the genotyping workflow that we evaluate here consists of extremely diverse applications with a wide range of computational requirements. CherryPick [28] uses Bayesian Optimization to model application performance to determine the best, or close-to-best, instance type with minimal experiments. Ernest [29] is able to predict the execution time of distributed analytics jobs based on cluster size. However, unlike our approach, Ernest cannot forecast performance without previously deploying an application on a given resource. Paragon [30] and Quasar [31] employ historic execution traces to rapidly classify new applications. In combination with a small amount of online profiling, Quasar is able to manage resource allocations and guide job packing. However, these traces do not incorporate fine-grained resource utilization statistics. To incorporate these, the Arrow framework [32] incorporates low-level performance statistics during its iterative optimization. This framework uses these statistics along with the standard runtime and cost statistics to inform a Bayesian optimization process to reduce the fragility of the optimization system and to more efficiently reduce "areas" of uncertainty in the Bayesian process. They have seen promising results that have overcome many of the previously observed shortcoming found in Bayesian optimization when applied to workflow profiling.

Micro-benchmarks have also shown great promise to aid in predicting the performance of a given workload in a foreign computing environment [33]. This is accomplished by using a suite of benchmarks to evaluate the unknown compute infrastructure and then use the execution characteristics of those benchmarks on the known infrastructure (i.e., where the execution characteristics of the workflow are known) to predict the workflow's execution characteristics in the new environment. Others have also sought to learn application performance models from data obtained from production environments [34].

VIII. Conclusion

Users of cloud platforms face the challenging task of optimally selecting from a huge collection of instance types those that are most appropriate for their workflows. To address this challenge we have presented an approach for selecting instance types based on a combination of proactive profiling experiments and transfer learning from prior non-cloud executions. Our models are able to accurately predict execution time for two real-world genome analysis workflows when considering arbitrary instance types with MAPE of 17.2% and arbitrary data size with MAPE less than 14.5%. To reduce the cost of running profiling experiments we explored an experiment design approach in which we intelligently explored the instance type search space. Our results show that with only four experiments we can achieve accuracy within 30% of that achieved when running experiments on every instance type.

In future work we aim to integrate these models and profiling techniques into a platform that learns to optimize genome analyses across cloud instances. We also plan to explore self-optimization through adaptive parameter tuning to automatically optimize parametrization based on input data and workflow composition. In addition, we intend to explore the ability to trade-off between accuracy, performance, and cost, such that users can increase accuracy by, for example, employing multiple variant callers, or decrease cost by electing smaller, less powerful resources or by integrating optimal instance type with predicted spot pricing [35]. Finally, we will continue to explore and optimize our adaptive learning approaches to better incorporate and predict scaling properties across instance types.

Acknowledgments

This material is based upon work supported by the RAMSES project and the National Science Foundation under Grant Nos. 1550588 and 1816611. We also acknowledge research credits provided by the AWS Cloud Credits for Research program.

References

[1] R. Chard, K. Chard, K. Bubendorfer, L. Lacinski, R. Madduri, and I. Foster, "Cost-aware cloud provisioning," in *11th International Conference on e-Science (e-Science)*. IEEE, 2015, pp. 136–144.

[2] ——, "Cost-aware elastic cloud provisioning for scientific workloads," in *8th International Conference on Cloud Computing (CLOUD)*. IEEE, 2015, pp. 971–974.

[3] W. Smith, I. Foster, and V. Taylor, "Predicting application run times with historical information," *Journal of Parallel and Distributed Computing*, vol. 64, no. 9, pp. 1007–1016, 2004.

[4] W. Tang, N. Desai, D. Buettner, and Z. Lan, "Job scheduling with adjusted runtime estimates on production supercomputers," *Journal of Parallel and Distributed Computing*, vol. 73, no. 7, pp. 926–938, 2013.

[5] L. T. Yang, X. Ma, and F. Mueller, "Cross-platform performance prediction of parallel applications using partial execution," in *ACM/IEEE Conference on Supercomputing*. IEEE Computer Society, 2005, p. 40.

[6] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D. Reed, L. Torczon, and R. Wolski, "The GrADS project: Software support for high-level grid application development," *International Journal of High Performance Computing Applications*, vol. 15, no. 4, pp. 327–344, 2001.

[7] A. W. Toga, I. Foster, C. Kesselman, R. Madduri, K. Chard, E. W. Deutsch, N. D. Price, G. Glusman, B. D. Heavner, I. D. Dinov, J. Ames, J. Van Horn, R. Kramer, and L. Hood, "Big biomedical data as the key resource for discovery science," *Journal of the American Medical Informatics Association*, vol. 22, no. 6, pp. 1126–1131, 2015. [Online]. Available: http://dx.doi.org/10.1093/jamia/ocv077

[8] J. J. Pitt, "Deciphering cancer development and progression through large-scale computational analyses of germline and somatic genomes," 2017.

[9] M. Wilde, M. Hategan, J. Wozniak, B. Clifford, D. Katz, and I. Foster, "Swift: A language for distributed parallel scripting," *Parallel Computing*, vol. 37, no. 9, pp. 633–652, 2011.

[10] Y. Babuji, K. Chard, I. Foster, D. S. Katz, M. Wilde, A. Woodard, and J. Wozniak, "Introducing Parsl: Parallel scripting in python," in *10th International Workshop on Science Gateways*, 2018.

[11] J. N. Weinstein, E. A. Collisson, G. B. Mills, K. R. M. Shaw, B. A. Ozenberger, K. Ellrott, I. Shmulevich, C. Sander, J. M. Stuart, C. G. A. R. Network *et al.*, "The cancer genome atlas pan-cancer analysis project," *Nature genetics*, vol. 45, no. 10, p. 1113, 2013.

[12] J. Pitt, M. Riester, Y. Zheng, T. Yoshimatsu, A. Sanni, O. Oluwasola, A. Veloso, E. Labrot, S. Wang, A. Odetunde *et al.*, "Characterization of Nigerian breast cancer reveals prevalent homologous recombination deficiency and aggressive molecular features," *Nature Communications*, 2018. [Online]. Available: https://doi.org/10.1038/s41467-018-06616-0

[13] J. Pitt, M. Bolt, D. Fitzgerald, L. Pesce, P. Van Loo, and K. White, "Aggregate allelic burden for cancer risk genes associates with age at diagnosis," in *Submitted to Cancer Research*. American Association for Cancer Research, 2018.

[14] H. Li and R. Durbin, "Fast and accurate short read alignment with Burrows–Wheeler transform," *bioinformatics*, vol. 25, no. 14, pp. 1754–1760, 2009.

[15] A. Tarasov, A. J. Vilella, E. Cuppen, I. J. Nijman, and P. Prins, "Sambamba: Fast processing of NGS alignment formats," *Bioinformatics*, vol. 31, no. 12, pp. 2032–2034, 2015.

[16] "PICARD mark duplicates," http://broadinstitute.github.io/picard/, accessed: 2018-08-15.

[17] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, and R. Durbin, "The sequence alignment/map format and SAMtools," *Bioinformatics*, vol. 25, no. 16, pp. 2078–2079, 2009.

[18] A. Rimmer, H. Phan, I. Mathieson, Z. Iqbal, S. R. Twigg, A. O. Wilkie, G. McVean, G. Lunter, W. Consortium *et al.*, "Integrating mapping-, assembly-and haplotype-based approaches for calling variants in clinical sequencing applications," *Nature genetics*, vol. 46, no. 8, p. 912, 2014.

[19] "Genome analysis toolkit," https://software.broadinstitute.org/gatk/, accessed: 2018-08-15.

[20] R. Chard, K. Chard, B. Ng, K. Bubendorfer, A. Rodriguez, R. Madduri, and I. Foster, "An automated tool profiling service for the cloud," in *16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 2016, pp. 223–232.

[21] R. Chard, K. Chard, R. Wolski, R. Madduri, B. Ng, K. Bubendorfer, and I. Foster, "Cost-aware cloud profiling, prediction, and provisioning as a service," *IEEE Cloud Computing*, vol. 4, no. 4, pp. 48–59, July 2017.

[22] B. Settles, "Active learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 6, no. 1, pp. 1–114, 2012.

[23] "Google cloud sizing recommendations," https://cloud.google.com/compute/docs/instances/apply-sizing-recommendations-for-instances, accessed: 2018-08-15.

[24] A. Matsunaga and J. A. Fortes, "On the use of machine learning to predict the time and resources consumed by applications," in *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*. IEEE Computer Society, 2010, pp. 495–504.

[25] A. V. Do, J. Chen, C. Wang, Y. C. Lee, A. Zomaya, and B. B. Zhou, "Profiling applications for virtual machine placement in clouds," in *IEEE International Conference on Cloud Computing (CLOUD)*, July 2011, pp. 660–667.

[26] T. Wood, L. Cherkasova, K. Ozonat, and P. Shenoy, "Profiling and modeling resource usage of virtualized applications," in *9th ACM/IFIP/USENIX International Conference on Middleware*, 2008, pp. 366–387.

[27] N. J. Yadwadkar, B. Hariharan, J. E. Gonzalez, B. Smith, and R. H. Katz, "Selecting the best VM across multiple public clouds: A data-driven performance modeling approach," in *Symposium on Cloud Computing*. ACM, 2017, pp. 452–465.

[28] O. Alipourfard, H. H. Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang, "CherryPick: Adaptively unearthing the best cloud configurations for big data analytics," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. Boston, MA: USENIX Association, 2017, pp. 469–482. [Online]. Available: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/alipourfard

[29] S. Venkataraman, Z. Yang, M. J. Franklin, B. Recht, and I. Stoica, "Ernest: Efficient performance prediction for large-scale advanced analytics," in *NSDI*, 2016, pp. 363–378.

[30] C. Delimitrou and C. Kozyrakis, "QoS-aware scheduling in heterogeneous datacenters with paragon," *ACM Transactions on Computer Systems*, vol. 31, no. 4, p. 12, 2013.

[31] ——, "Quasar: Resource-efficient and QoS-aware cluster management," *ACM SIGPLAN Notices*, vol. 49, no. 4, pp. 127–144, 2014.

[32] C. Hsu, V. Nair, V. W. Freeh, and T. Menzies, "Arrow: Low-level augmented bayesian optimization for finding the best cloud vm," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, vol. 00, Jul 2018, pp. 660–670. [Online]. Available: doi.ieeecomputersociety.org/10.1109/ICDCS.2018.00070

[33] J. Scheuner and P. Leitner, "Estimating cloud application performance based on micro-benchmark profiling," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, July 2018, pp. 90–97.

[34] P. Bodik, R. Griffith, C. Sutton, A. Fox, M. I. Jordan, and D. A. Patterson, "Automatic exploration of datacenter performance regimes," in *1st Workshop on Automated Control for Datacenters and Clouds*. ACM, 2009, pp. 1–6.

[35] M. Baughman, C. Haas, R. Wolski, I. Foster, and K. Chard, "Predicting amazon spot prices with lstm networks," in *Proceedings of the 9th Workshop on Scientific Cloud Computing*, ser. ScienceCloud'18. New York, NY, USA: ACM, 2018, pp. 1:1–1:7. [Online]. Available: http://doi.acm.org/10.1145/3217880.3217881