Predicting Cloud Performance Using Real-time VM-level Metrics

Jihua Tian*, Abdessalam Elhabbash||, Yehia Elkhatib*
* *School of Computing Science, University of Glasgow, United Kingdom*
|| *School of Computing and Communications, Lancaster University, United Kingdom*

***Abstract*—The vast range of cloud service offerings can easily overwhelm users and cause them to select ones that are unsuitable for their needs. As such, the literature has a number of proposals to predict application performance based on a history of executing a certain application or benchmark. However, this requires significant cost to pre-run the application on different service levels before identifying the most suitable one. We propose a machine learning model that enables a cloud user to select the optimal cloud service based on real-time execution without the need to do an exhaustive search. We develop and test this model using a popular benchmark suite on Microsoft Azure, a leading cloud provider. The key insight of this work is that *fluctuations in* rather than the absolute *amount of utilization levels* of CPU and memory can be strongly indicative of how well an application is executing.**

***Index Terms*—Cloud computing, Machine Learning, Service Level Objectives**

1. Introduction

Cloud Computing is now the de facto choice for obtaining highly available computing resources. Cloud providers install and manage massive computing infrastructure in their data centers and offer their computing resources, which can be hardware or software, to users as services. The cloud computing market is now a mature one with a number of high-profile providers who offer various services including computing, storage, networking, hosted services, lambda functions, and much more. The rapid flourishing of the cloud market has resulted in a wide range of services. See, for instance, the number of available Infrastructure as a Service (IaaS) offerings by the major cloud provider depicted in Fig 1. The number of Linux-based cloud instance types increased from 134 to 198 between 2015 and 2017 [1]. In 2022, there are more than 400 types of instances provided by Microsoft Azure[1].

On the face of it, such wide range of service provisions is a sign of a healthy and competitive market sector. However, it does have its own drawbacks. First, there is a significant risk of customers mis-selecting instances from the wide range of available options [2], resulting in either under-performing or over-provisioned

1. https://azure.microsoft.com/en-us/pricing/vm-selector/

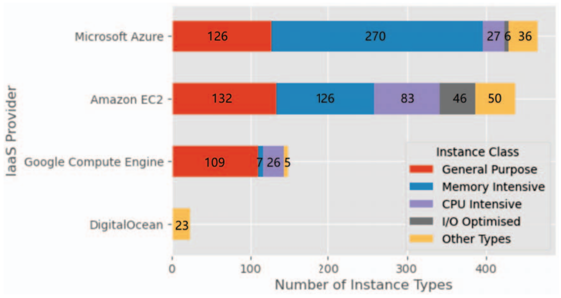

Figure 1: The number of Linux-based instance types offered by major IaaS vendors, as of January 2022.

applications. In fact, "a wrong choice can lead to a 20 times slowdown or an increase in cost by 10 times" [3]. Second, the number of alternative choices can be overwhelming even to the extent of avoiding IaaS completely [2], [4]. Third, any choice could eventually lead to vendor lock-in which can be significantly costly in the long-term [1], [5].

Cloud offerings are notoriously difficult to differentiate between [1], [4]. Typically, each cloud service provider (CSP) commits to a Service Level Agreement (SLA) that consists of a number of Service Level Objectives (SLOs). An SLO are commitments from a CSP regarding quantitative aspects of their service. SLOs include uptime, availability, throughput, etc. However, SLOs do not easily translate to the context of a specific customer application. In other words, it is difficult to ascertain the performance of an application given the SLOs of a certain provider.

As an illustration, consider a cloud customer who selects an instance to deploy their application on. The instance's SLA states that availability is guaranteed to be at least 99.999%. The customer needs each run of the application to be under 0.5 seconds. The customer could observe performance degradation, despite the CSP not violating their SLA. In essence, SLAs define service-specific SLOs that promise service performance levels that are somewhat independent from the real performance experienced by the customer application. The latter depends on several other factors including internal ones such as application design and implementation decisions.

A number of works aim to aid cloud customers by traversing or reducing the IaaS search space [1]. They support decision making prior to deployment and

979-8-3503-1993-4/22/$31.00 ©2022 IEEE
DOI 10.1109/HPCC-DSS-SmartCity-DependSys57074.2022.00184

are based on a given application 'silhouette'. As such, they cannot be used to react to variance in application execution, e.g. difference in input data. They also make decisions based on a relatively large number of data-points that form a generalization of the performance of a given instance type, but they cannot react in *realtime* to an under-performing instance.

In this paper, we present a novel mechanism to automate cloud service selection based on real-time assessment of low-level metrics (*e.g.,* CPU and memory utilization). We achieve this by identifying metrics that are most predictive of application performance on certain cloud instances. The metrics can then be used to anticipate whether or not the application SLOs will be met and to reason about whether to continue with the current selection of VMs or migrate to other types. In summary, this paper makes three contributions:

- We apply statistical analysis of collected application performance data to identify predictive metrics. To achieve this, the proposed framework evaluates the correlation between the collected metrics and the observed SLO values, and filters out metrics to include only those that have a strong correlation with the specified SLOs.
- We employ regression techniques to predict application SLO values in order to reason about the selection of a cloud service.
- We conduct extensive evaluation of the proposed solution to evaluate the accuracy of SLO prediction when deploying on a certain cloud service.

2. Related work

A number of works devised means to support making the decision of which cloud instance type to use. This is especially judicious considering that high specification VMs do not necessarily lead to lower application execution time [6], [7], [8], [9], [10].

Previously, several research addressed performance predictions for programs by modeling application characteristics such as architecture, inputs, and platforms or code characteristics [10], [11], [12], [13], [14], [15]. For example, Singh et al. [14] and Yang et al. [16] proposed methods to create robust models to predict execution time by capturing the interaction between software and hardware components. Although predictions with high accuracy have been delivered through these approaches, each application requires in-depth analysis to build an application-specific model of its behavior. This upfront cost is expensive as modern application architectures become increasingly complex, and their inherent characteristics are difficult to understand and develop into analytic models.

More recent efforts tend to focus on building machine learning models based solely on observed performance. Ernest [17] trains a model for applications with a small number of execution samples. Since its performance model is tightly bound to the particular structure of a given application, it does not work well for other applications. Based on Bayesian optimization, CherryPick [18] trains a Bayesian model to distinguish near-optimal cloud configuration from the rest. Nevertheless, it still demands one-time or regular (*e.g.,* daily) cost for modeling tasks. Paris [19] uses hybrid offline benchmarking to generate sufficient workload fingerprints to obtain a cost-performance trade-off. Baughman et al. [10] present similar work, predicting execution time based on knowledge from historical non-cloud executions and proactive profiling experiments. However, this approach of combining synthetically generated data with real-world data tends to result in reduced prediction accuracy [20]. MICKY [21] formulates the task of finding the right cloud instance types as a multi-armed bandit problem. This reduces the cost associated with modeling while still achieving near-optimal solutions.

Overall, this body of work still requires prior execution of an application either on cloud or off-cloud resources, and does not allow for real-time performance analysis. The closest work to ours is Arrow [3], which augments Bayesian optimization based on instance specifications with low-level performance indicators such as CPU and memory utilization. Nevertheless, the work is still dependent on pre-existing knowledge of how an application would execute on certain instance types and cannot be used in real-time like ours.

3. Methodology

3.1. Research questions and hypothesis

A cloud customer who needs to select a cloud service would reason their selection based on a number of criteria including the computational specifications and SLA of the the services. The specifications mainly include the number of virtual CPUs (vCPUs), memory and storage capacities, and data bandwidth.

As mentioned above, recent research results showed that higher computational specifications does not necessarily lead to improvements in application performance. This leads us to set the following research questions:

`RQ1:` Is it possible to predict application performance without relying on high-level SLO metrics (such as availability)?

`RQ2:` Is it possible to predict application performance by simply looking at basic resource utilization metrics at the VM level?

`RQ3:` Is there a combination of metrics, or a composite feature that would facilitate such prediction of application performance?

`RQ4:` Could such prediction be done in real-time?

In this work, we posit the following hypothesis: *The higher the stability of a cloud VM metric the higher the predictability of that metric of the application SLO.*

This postulates that if we monitor a certain metric of VM performance (*e.g.,* CPU utilization) and the performance of the application deployed on that VM, then the rate of fluctuation of that metric indicates whether the metric can be reliably used to predict the performance of the application.

3.2. Cloud benchmarks

In order to investigate the validity of this hypothesis, we conduct experiments to compare the values of application SLOs (in terms of completion time) and the fluctuation of monitored VM performance metrics when the application is running over on different VM types. To conduct our experiment, we employ an application from CloudSuite [22], a popular benchmark suite that is representative of a range of cloud applications. The selected application is the Graph Analytics benchmark [23], a CPU-intensive application, due to the popularity of this application profile. The benchmark utilizes the Spark framework to perform graph analytics on large-scale datasets. We execute it using a single-VM setup to avoid network contention issues.

3.3. Cloud VMs

We identified the VMs listed in Table 1 as our target infrastructures to perform the experimental comparison. The VMs were selected based on the following criteria:

- On-demand instance types charged at an hourly rate.
- Within the same region, namely UK South.
- At least one type from each category (*i.e.,* general purpose, CPU-optimized, etc.).
- Equivalent specifications as much as possible; particularly, they all have the same number of virtual cores as the application is a CPU-intensive benchmark.

Table 1: Computational specifications of the VMs used in the experiment.

Series	**VM type**	**vCPU**	**RAM** (GiB)	**Disk** (GiB)	**Price** ($/hr)
General purpose	B4ms	4	16	32	0.189
	A4v2	4	8	40	0.222
	D4v3	4	16	100	0.232
Compute optimized	F4sv2	4	8	32	0.202
	F4	4	8	64	0.237
Memory optimized	E4v3	4	32	100	0.312
	D12v2	4	28	200	0.469
	G2	4	56	768	0.878
Storage optimized	L4s	4	32	678	0.362

3.4. Data collection

As we ran benchmarks on the different VMs, we recorded resource utilization every ten seconds using `dstat`. We repeated benchmark execution on each VM ten times with five minutes delay between each pair of runs. All executions were performed on the same day in order to avoid any potential effect of VM performance variation between days of the week [7]. Our SLO (*i.e.,* response variable) was completion or execution time. The data is available here: https://www.dropbox.com/sh/5w9a6yaovm9tu4s/AADEvvKsWNwxfaeptlBNbs5ta?dl=0

4. Data Analysis

Before training regression models on the dataset, several aspects of the collected data need to be understood. Specifically, our objectives in this section are:

- Analyze the inherent characteristics of the dataset.
- Explore correlation between resource utilization indicators and completion time, if any.
- Perform feature engineering to transform raw data into features that are more beneficial for inducing a predictive model.

4.1. Measurement of resource utilization

We explore each monitoring component to figure out which features can effectively represent the ability to make use of system resources.

- We calculate the mean **CPU** utilization across cores (we observe balanced load across cores for all VMs in the dataset). We focus on CPU utilization by user-space and system processes. Other indicators (*e.g.,* idle CPU) are not indicative due to lack of sufficient variation.
- We calculate **memory** utilization as a percentage of the capacity (as specified by the provider – see Table 1).
- The amounts of **network traffic** sent and received are used as absolute values, as the network capacity is not specified by the provider.

4.2. Correlation between resource utilization and completion time

To discover additional information from the different executions, we explore the relationship between resource usage and execution time by analyzing statistical characteristics for each system component.

4.2.1. CPU utilization. Overall, there is a weak inversely proportional relationship between memory capacity of a VM and the completion time (Fig 2). Observed exceptions are A4v2 and E4v3, which are only able to use two of the available vCPUs. We also notice more uncertainty with these instance types as indicated by a large number of outliers (represented by circles) in each distribution. By contrast, instance types that are able to make use of all four vCPUs behave more stably with fewer outliers. An initial takeaway here is that such fluctuations in processor performance may be used to signify completion time, where a higher frequency of fluctuation corresponds to longer execution duration.

4.2.2. Memory utilization. In contrast, memory utilization exhibits a much weaker correlation with completion time (Fig 2). We notice similar completion times on VMs of dissimilar memory usage patterns. For example, D12v2 and F4 exhibit similar execution times, but the memory utilization of the latter is significantly higher. On the other hand, A4v2 and F4 have the same

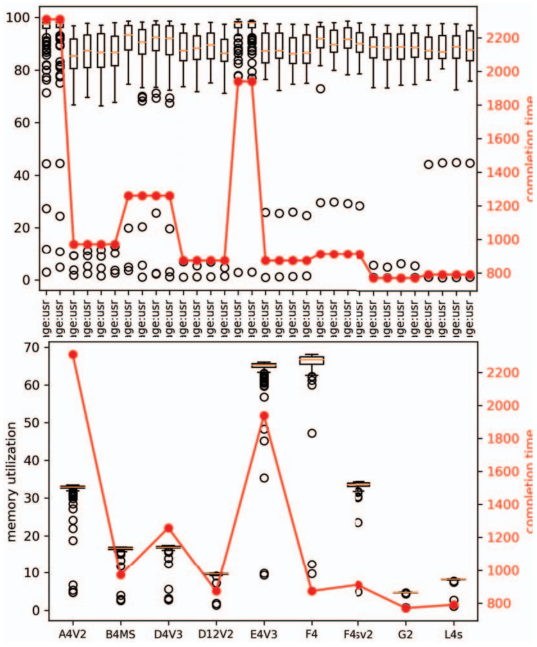
Figure 2: Percentage utilization levels of CPU (top) and memory (bottom), against completion times in run 1.

RAM capacity and similar memory utilization, but the execution time of the former is more than double as that of the latter.

Moreover, relying on outliers to predict completion times does not seem promising: *e.g.,* although B4ms and D4v3 completed their tasks in short times, there are a number of outliers in memory utilization. The reason why memory-related factors are not predictive for application performance is worth discussing. Memory utilization never really goes beyond 35%, even for low-spec instances such as A4v2 and F4. As such, memory is relatively underutilized and it would be difficult to detect additional information from monitoring memory usage.

4.2.3. Network usage. Although it is difficult to summarize the correlation between the fluctuation in network traffic and completion time, we observe that instances with lower execution times normally maintain greater traffic sizes, particularly sent data as is evident in Fig 3.

4.3. Median and frequency of fluctuation

Based on the analysis above, we can extract some indicators of performance to guide our prediction. For CPU utilization, we can extract the two features of the frequency of fluctuation and the median value, then add these new predictors to the dataset to create a more accurate model. The median value can be retrieved and computed directly, while some techniques need to be adopted to detect the points that fluctuate markedly.

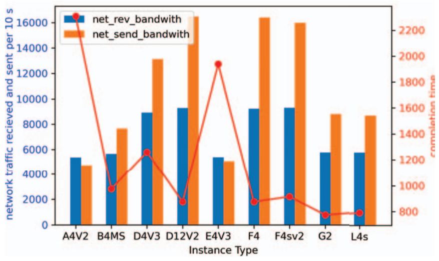

Figure 3: Averaged traffic usage in different instance

These outlier points are generally defined as points that remarkably deviate from their expected value; therefore, in a given time series, we can declare a point at time t as an outlier if the distance to its expected value exceeds a predefined threshold τ:

$$\left| x_t - \bar{x}_t \right| > \tau \tag{1}$$

where x_t is the observed data point, and $\bar{x}_t$ is its expected value [24]. A simple approach to estimate the expected value based on basic statistics is the Median Absolute Deviation (MAD) [25]; once x^t is assumed, outlier points can be determined directly using Equation 1. Generally, the sensitivity of the algorithm is controlled by the threshold value: the greater the τ, the more possible a point is considered as a local fluctuation. Fig 4 illustrates distinctive detection results for instance type A4v2 as the threshold value varies. As the threshold increases, the algorithm focuses more on capturing significantly varied points while ignoring those that are only slightly deviant.

In order to compare such fluctuations across different time series, frequency is considered as a metric to describe the number of occurrences of an anomaly during a unit of time. Since our data was measured once every ten seconds, we can calculate the number of fluctuations per minute using:

$$Freq_{fluct} = 6 * n_{fluct} / n_{execut} \tag{2}$$

where n_{fluct} is the number of outliers and n_{execut} is the total number of records.

4.4. Features after engineering

Based on the above analysis of system usage indicators, we select the following 11 features:

- CPU user utilization
- CPU system utilization
- CPU core numbers
- CPU fluctuation frequency
- CPU user median
- Memory utilization
- Memory capacity
- Network sent average
- Network received total
- Network sent total
- Network received average

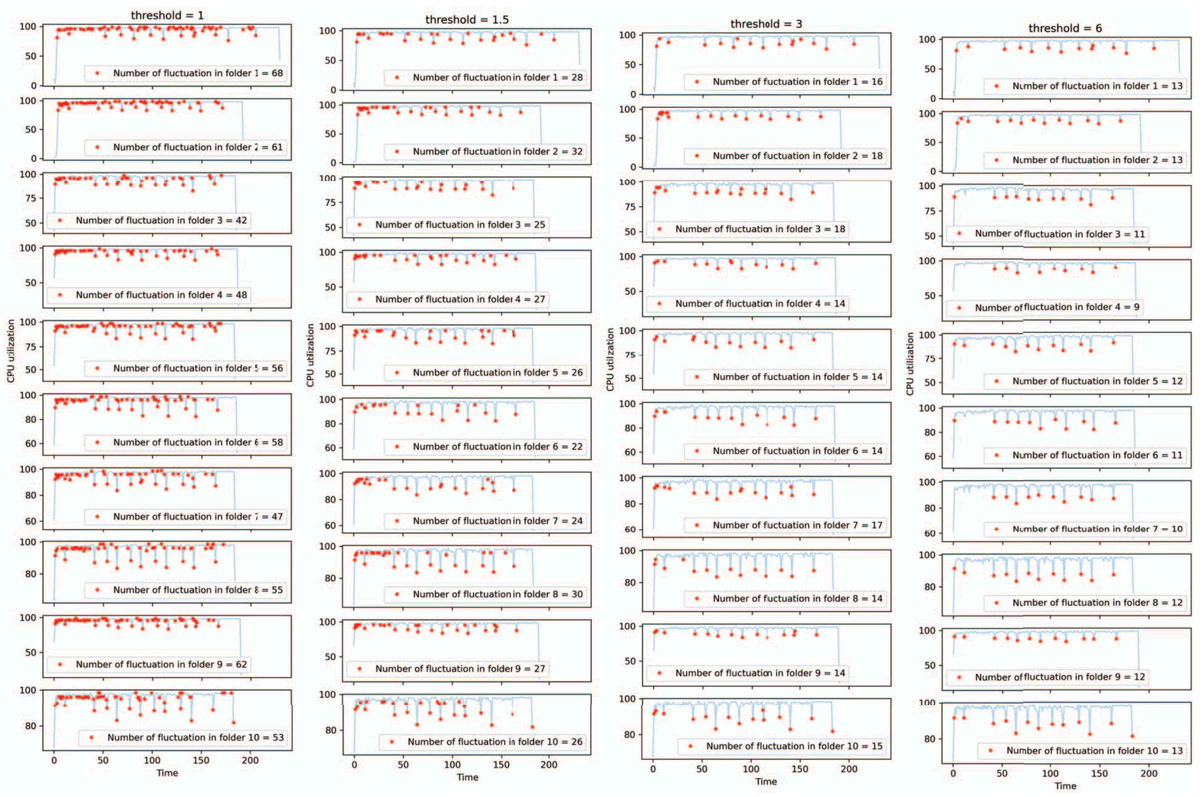

Figure 4: Outlier detection with different threshold values.

4.5. Dataset split

The last step of data pre-processing is the split of the dataset into training and testing sets. Since the original data was gathered from the tool which performed monitoring tasks on 10 different days and has already been stored in the corresponding runs, we can simply split the dataset according to run numbers. In this project, we take output from 9 runs as the training set, and the tenth run to be the test set.

5. Implementation

In this section, we introduce the models adopted for the prediction task and the means by which they were developed.

5.1. Regression models

Although the response variable (*i.e.,* completion time of a certain job executed by an application) is a fixed value over one specific project, we decide to employ regression methods to predict this target label. This is because such an attribute does not belong to any category we can pre-define. Meanwhile, it could be regarded as a continuous variable as we are concatenating tables when generating training and test sets.

5.1.1. Linear Regression. is a relatively simple model of the relationship between response variables and predictors. Generally, the simplest scenario of linear regression only involves one independent and one dependent variable. As there are multiple independent variables in our dataset, we use multi-variable linear regression to attempt to capture the correlations between these parameters and the response variable.

5.1.2. Support Vector Regression (SVR). models perform well in solving regression problems with a small sample, non-linear and highly-dimensional datasets [26]. Instead of trying to find a hyperplane that separates two classes in a classification problem that maximizes the width of the margin, SVR aims to look for decision boundaries that can fit as many data points between them and the hyperplane. Meanwhile, different kernels (such as RBF, Gaussian, and Polynomial) enable SVR to address non-linearly separable data, which is particularly helpful to our multidimensional dataset.

5.1.3. Decision Trees (DT). can work as either classification or regression model in a tree structure, breaking down a dataset into smaller and smaller subsets while simultaneously developing an associated precision tree. Its non-leaf nodes and branches represent truth and falsity values for a statement of a feature, and leaf nodes indicate decisions on the numerical response variables. An advantage of DT is that it can measure both categorical and numerical variables.

5.2. Procedure and implementation

We implemented the entire data processing pipeline in Python, using the libraries `NumPy`, `Pandas` and `Scikit-Learn`. The first two libraries are used to conduct data analysis and preparation, while the third is equipped with various classical machine learning algorithms (including regression models) as well as evaluation metrics.

In this project, the regression models learn the mapping from system resource usage (variables we measured in the previous section) to completion time for each record in the training set, then completion time of data in the testing set can be predicted based on that knowledge. However, in order to get robust models that can precisely predict the target values, we will investigate the impact of the threshold of outlier detection by setting different values and comparing their performance across *all* models and *all* instance types. We look for an optimal value that performs best in the prediction task. During this process, cross-validation will be adopted as a crucial technique to assess the goodness of the models on the training set whilst avoiding overfitting. After that, evaluation methods will be introduced to check the effectiveness of the models on the testing set.

The procedure is outlined as follows:

1) Split the training set into 9 folds by the number of runs.
2) Choose a threshold value from range 1-5, with an interval of 0.5.
3) Select a fold as the validation set (in turn) and others as the training set used to fit in the regression model, then compute the Mean Squared Error (MSE) between the prediction result and true value on the validation set.
4) Repeat step 3 and average the MSE for all iterations.
5) Compare the performance of the model with different settings of threshold across all instance types and choose the optimal one.
6) Repeat the steps above by applying three regression algorithms – namely, Linear Regression, SVR, and DT – and investigate whether they are able to achieve consistent conclusions in threshold selection. It is worth noting that for SVR, our exploration evaluates the above metrics on three different kernels so, in effect, we compare the performance of five different models.
7) Evaluate the effectiveness of different models on the testing set.

6. Evaluation Results

This section conducts sensitivity analysis and illustrates the experiment results, especially the outcomes of different threshold values and comparison of accuracy between the models we selected over the testing set. After that, we will discuss the best-performing models based on evaluation criteria as well as whether their predictive ability is influenced by instance types.

6.1. Threshold sensitivity analysis

We conduct sensitivity analysis to analyze the impact of changing the threshold value on the log loss in order to determine the appropriate threshold value at which performance fluctuations of the cloud VMs becomes effectively detectable. For this purpose, we plot the log loss against varied values of the CPU fluctuation threshold in Fig 5. In the cases of Linear Regression and Decision Tree Regression we observe that the threshold value has significant impact on the predicted performance as the log loss between the estimator and estimated value changes. This fluctuation is more obvious when the threshold varies in the range of 0 to 2. However, as the threshold gets larger, the logs loss value stabilizes, to a big extent, especially in the case of linear regression. This implies that the outliers detection algorithm is more sensitive to smaller values of the threshold, resulting in unpredictable loss. However, for larger values of the threshold, the frequency of outliers will no longer be significantly different.

In the case of SVR (Fig 5), the log loss does not vary against the change in the threshold values. This suggests that the feature of frequency of fluctuating points has no effect on the accuracy of the model.

Based on the above observations, we set the threshold value as 3.5 for detecting fluctuations which can result in relatively minimum loss for all the models. Another point worth noting is that the loss of B4ms is always significantly higher than the other instances, regardless of which model is used. This means that the application performance of this instance is difficult to predict, an interesting observation that motivates further investigation (see next subsection).

6.2. Model evaluation

To evaluate the accuracy of the models, we need to compare and check their prediction performances on the testing set. MSE and R^2 score are the evaluation metrics used in the experiments. The MSE measures the difference between the predicted value and expected output. The R^2 score measures the proportion of variability in the response variable that is explainable by the predictors [7], [15]. Table 2 illustrates a general overview of the prediction results of the models. The MSE values for both SVR models are considerably higher than that of the linear model, while MSE for DT fit is smaller than the linear method. Regarding other regression tests, the R^2 statistic always ranges between 0 and 1, which indicates the proportion of variability. The low R^2 values for SVR regressions indicate that this model fails to explain variability in terms of response variables. By contrast, more than 89% and 96%

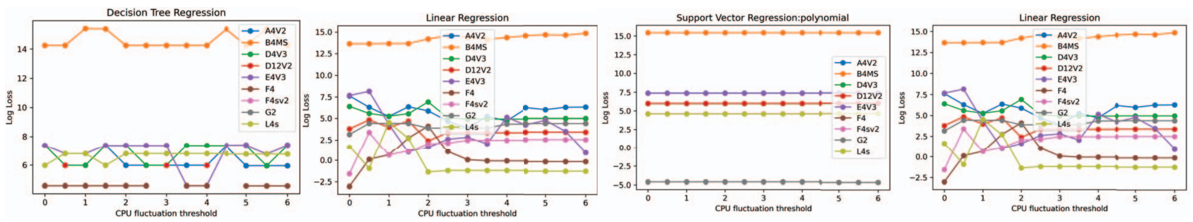

Figure 5: Sensitivity analysis of the CPU fluctuation threshold.

Table 2: MSE and R^2 scores for the different models.

Model	MSE	R^2
Linear Regression	44495.62	0.89
Decision Tree Regression	28579.28	0.96
Support Vector Regression (Kernel RBF)	583180.85	-50
Support Vector Regression (Kernel polynomial)	692471.04	-50.72

of data variability in the prediction of completion time are captured by Linear and DT respectively.

The prediction ability of the models can be visually checked by plotting the predicted and actual execution time from the testing set. In Fig 6, the blue lines indicate the real execution times of the different instances on the testing set, and the other lines represent the predicted values of the corresponding models. As we discussed earlier, the predictive ability of SVR is rather poor, especially for the model with a polynomial kernel, which can barely predict the response variable. In contrast, linear regression has a relatively good fit, although the predicted completion time on a particular instance varies slightly. The predicted values of DT are close to the true values, *i.e.,* it outperformed both linear and SVR models.

6.2.1. Effectiveness of models on different instances.

As mentioned above, in the cases of instance B4ms, the losses between prediction and actual responses are extremely high. This is also reflected in Fig 6, where the second blue line represents the actual execution time of B4ms, by which we can observe predictions of all models are relatively far away from the truth. Thus, to evaluate whether the predictive performance will vary

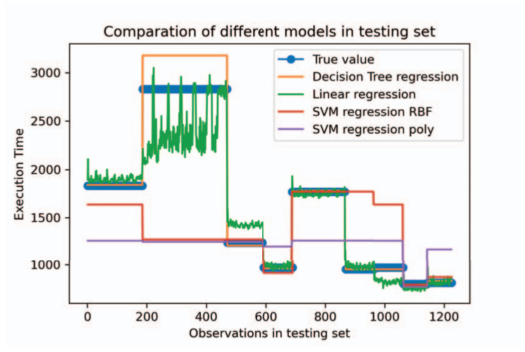

Figure 6: Predicted vs. real response on the testing set

Table 3: MSE of the DT and linear regression models.

Instance	Decision Tree	Linear Regression
A4v2	400	10.50
B4ms	122500	1903121
D4v3	900	30.09
D12v2	100	1.24
E4v3	2500	78.31
F4	0	18.69
F4sv2	100	29.47
G2	100	6.73
L4s	400	6.39

due to the instance type. Instead of fitting models with the whole data of the training set, each time we use data of different instances to train the regression model and assess the goodness of the model on respective testing data. To simplify, we select linear and DT regression in this step as the SVR did not fit the data well in previous results.

As is evident from Table 3, despite some variations, both models can effectively predict the application execution time except B4ms, with linear regression having an overall better performance. It also illustrates that the extremely large error of linear regression on B4ms is the reason for its worse predictive ability than the DT on the whole testing set. By referencing information provided by Azure, different from other VM instances which are configured with certain processors, B4ms sometimes need to burst to significantly higher CPU performance when the demand rises. Such variation in performance is likely the reason why the response values for this type of instance are not accurately predictable.

In summary, we found a suitable value for the threshold used in outlier detection through cross-validating the training set. Then, we assessed the predictive performance of linear, DT, and SVR models with the threshold value by using MSE and R^2 as evaluation metrics. Furthermore, we evaluated the performance of linear regression and DT on each instance type respectively, finding execution time can be effectively predicted in the majority of VM instances except B4ms.

7. Conclusion

In this paper, we investigated the correlation between cloud application performance and resource usage of the hosting VM. Our findings show that the robust regression models we built using linear regression and decision trees can be used to effectively predict service level objectives. The model is of immediate benefit to anyone running cloud applications, helping them to optimize deployment in a real-time fashion by observing basic VM-level metrics that are application-independent.

Threats to validity. Naturally, the data we used to train our model had its own limitations: it was collected from a limited set of VM types from a single provider, from a single availability zone, and from one weekday. However, literature shows that these do *not* affect the representativeness of the data. Specifically, none of the time of day, day of week (excluding weekends), or availability zone is a significant predictor of VM performance [7], [27], except for VM startup time which varies for some providers across availability zones [28]. In our study, we do not aim to establish any causal relationships hence threats to internal validity are minimal. Although we comment on the correlations we observe, we are cautious not to generalize in order to mitigate threats to external validity.

Future work. We will expand our models by exploring data from VMs across providers. We also plan to employ more than one application benchmark to gauge model effectiveness for different application profiles.

Acknowledgment

This work was partly supported by the UK EPSRC under grant number EP/R010889/2.

References

[1] A. Elhabbash, F. Samreen, J. Hadley, and Y. Elkhatib, “Cloud brokerage: A systematic survey,” *ACM Computing Surveys*, vol. 51, no. 6, pp. 119:1–119:28, Jan. 2019.

[2] C. Kilcioglu, J. M. Rao, A. Kannan, and R. P. McAfee, “Usage patterns and the economics of the public cloud,” in *International Conference on World Wide Web*, 2017, pp. 83–91.

[3] C.-J. Hsu, V. Nair, V. W. Freeh, and T. Menzies, “Arrow: Low-level augmented bayesian optimization for finding the best cloud vm,” in *International Conference on Distributed Computing Systems (ICDCS)*, 2018, pp. 660–670.

[4] Cloud Standards Coordination (Phase 2), “Cloud computing users needs - analysis, conclusions and recommendations from a public survey,” ETSI, Special Report 003 381 V2.1.1, 2016. [Online]. Available: http://csc.etsi.org/phase2/UserNeeds.html

[5] R. Chard, K. Chard, K. Bubendorfer, L. Lacinski, R. Madduri, and I. Foster, “Cost-aware cloud provisioning,” in *International Conference on e-Science*. IEEE, 2015, pp. 136–144.

[6] Z. Ou, H. Zhuang, J. K. Nurminen, A. Ylä-Jääski, and P. Hui, “Exploiting hardware heterogeneity within the same instance type of Amazon EC2,” in *Conference on Hot Topics in Cloud Computing (HotCloud)*. USENIX, 2012.

[7] F. Samreen, Y. Elkhatib, M. Rowe, and G. S. Blair, “Daleel: Simplifying cloud instance selection using machine learning,” in *IEEE/IFIP Network Operations and Management Symposium (NOMS)*. IEEE, 2016, pp. 557–563.

[8] K. Hwang, X. Bai, Y. Shi, M. Li, W. G. Chen, and Y. Wu, “Cloud performance modeling with benchmark evaluation of elastic scaling strategies,” *Trans. Parallel Distrib. Syst.*, vol. 27, no. 1, pp. 130–143, 2016.

[9] N. Ghrada, M. F. Zhani, and Y. Elkhatib, “Price and performance of cloud-hosted virtual network functions: Analysis and future challenges,” in *PVE-SDN*, 2018.

[10] M. Baughman, R. Chard, L. Ward, J. Pitt, K. Chard, and I. Foster, “Profiling and predicting application performance on the cloud,” in *IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*, 2018.

[11] D. J. Kerbyson, H. J. Alme, A. Hoisie, F. Petrini, H. J. Wasserman, and M. Gittings, “Predictive performance and scalability modeling of a large-scale application,” in *ACM/IEEE Conference on Supercomputing (SC)*, 2001.

[12] G. Marin and J. Mellor-Crummey, “Cross-architecture performance predictions for scientific applications using parameterized models,” in *Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS/Performance)*, 2004, pp. 2–13.

[13] L. Carrington, A. Snavely, and N. Wolter, “A performance prediction framework for scientific applications,” *Future Generation Computer Systems*, vol. 22, no. 3, pp. 336–346, 2006.

[14] K. Singh, E. Ipek, S. A. McKee, B. R. de Supinski, M. Schulz, and R. Caruana, “Predicting parallel application performance via machine learning approaches,” *Concurrency and Computation: Practice and Experience*, vol. 19, no. 17, pp. 2219–2235, 2007.

[15] A. Maricq, D. Duplyakin, I. Jimenez, C. Maltzahn, R. Stutsman, and R. Ricci, “Taming performance variability,” in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. Carlsbad, CA: USENIX Association, Oct. 2018, pp. 409–425. [Online]. Available: https://www.usenix.org/conference/osdi18/presentation/maricq

[16] L. T. Yang, X. Ma, and F. Mueller, “Cross-platform performance prediction of parallel applications using partial execution,” in *ACM/IEEE Conference on Supercomputing (SC)*, 2005.

[17] S. Venkataraman, Z. Yang, M. Franklin, B. Recht, and I. Stoica, “Ernest: Efficient performance prediction for large-scale advanced analytics,” in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2016.

[18] 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 *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017.

[19] 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 *ACM Symposium on Cloud Computing (SoCC)*, 2017.

[20] F. Samreen, G. S. Blair, and Y. Elkhatib, “Transferable knowledge for low-cost decision making in cloud environments,” *Transactions on Cloud Computing*, vol. 10, p. 3, Jul. 2022.

[21] C.-J. Hsu, V. Nair, T. Menzies, and V. Freeh, “Micky: A cheaper alternative for selecting cloud instances,” in *International Conference on Cloud Computing (CLOUD)*, 2018, pp. 409–416.

[22] “CloudSuite,” https://www.cloudsuite.ch/.

[23] “Graph analytics,” https://github.com/parsa-epfl/cloudsuite/blob/master/docs/benchmarks/graph-analytics.md.

[24] A. Blázquez-García, A. Conde, U. Mori, and J. A. Lozano, “A review on outlier/anomaly detection in time series data,” *ACM Computing Surveys (CSUR)*, vol. 54, no. 3, pp. 1–33, 2021.

[25] S. Mehrang, E. Helander, M. Pavel, A. Chieh, and I. Korhonen, “Outlier detection in weight time series of connected scales,” in *International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE, 2015, pp. 1489–1496.

[26] A. J. Smola and B. Schölkopf, “A tutorial on support vector regression,” *Statistics and computing*, vol. 14, no. 3, pp. 199–222, 2004.

[27] P. Leitner and J. Cito, “Patterns in the chaos—a study of performance variation and predictability in public IaaS clouds,” *ACM Trans. Internet Technol.*, vol. 16, no. 3, Apr. 2016.

[28] J. Hao, T. Jiang, W. Wang, and I. K. Kim, “An empirical analysis of VM startup times in public iaas clouds: An extended report,” *CoRR*, vol. abs/2107.03467, 2021. [Online]. Available: https://arxiv.org/abs/2107.03467