FIFE: an Infrastructure-as-Code Based Framework for Evaluating VM Instances from Multiple Clouds

Yuhui Lin*
University of St Andrews
St Andrews, UK
Email: yl205@st-andrews.ac.uk

Jack Briggs*
University of St Andrews
St Andrews, UK
Email: jb260@st-andrews.ac.uk

Adam Barker
University of St Andrews
St Andrews, UK
Email: adam.barker@st-andrews.ac.uk

***Abstract*—To choose an optimal VM, Cloud users often need to step a process of evaluating the performance of VMs by benchmarking or running a black-box search technique such as Bayesian optimisation. To facilitate the process, we develop a generic and highly configurable Framework with Infrastructure-as-Code (IaC) support For VM Evaluation (FIFE). FIFE abstract the process as a *searcher*, *selector*, *deployer* and *interpreter*. It allows users to specify the target VM sets and evaluation objectives with JSON to automate the process. We demonstrate the use of the framework by setting up of a *Bayesian optimization* VM searching system. We evaluate the system with various experimental setups, i.e. different combinations of cloud provider numbers and parallel search. The results show that the search efficiency remains the same for the case when the search space is consist of VM from multiple cloud providers, and the parallel search can significantly reduce search time when the number of parallelisation is set properly.**

***Keywords*-Cloud Computing, Infrastructure-as-Code, VM Evaluation Framework, Bayesian Optimization**

I. Introduction

Choosing optimal Virtual Machine (VM) instances for an application is a complex, time consuming and often costly task. The performance of different types of applications have very different relationships with the computational resources available from cloud providers. The diversity in services provided by different cloud providers creates a large search space, e.g. AWS EC2 alone has over 200 predefined instance models.

Even when an application runs on identical cloud configurations from the same provider, the same application can have significantly different performance due to hardware heterogeneity and 'noisy neighbours' [1]. Evidence shows that cloud providers have been developing methods of minimising these problems [2, 3]. Nonetheless, such randomness must be considered, when optimising cloud configurations, to rules out assumptions of deterministic application performance.

Many previous studies have addressed the challenge of VM instance selection, presenting a range of possible solutions [4, 5, 6, 7, 8, 9]. These approaches are typically based upon benchmarking, search or a combination of both. However, their solutions are usually limited in scope, either to a single provider or to specific application types. Moreover, their solutions do not focus on facilitating end-users [10] to easily adapt their technologies, and there is little support for automated Infrastructure-as-Code (IaC) based cloud management which treats cloud resources as software and manages them with a machine-readable script.

In this paper, we propose a user-oriented framework designed for automated VM instance evaluation using IaC, called *FIFE* (**F**ramework with **I**aC support **F**or VM **E**valuation). FIFE is generic and highly configurable. It automates the VM evaluation process for an application, thereby reducing the human effort involved. Users can define a search strategy with different objectives, e.g. cost-value or performance, and configure candidate VM options from different cloud providers.

In FIFE, we provide two search strategies: an exhaustive method, which enumerates all possible candidates; and *Bayesian optimization* [11], which is an optimization method specifically designed for situations where the objective functions itself is unknown or expensive to evaluate. FIFE uses the Bayesian methodology to iterate optimization process by incorporating a prior model on the space of possible target. The source code of FIFE and all experiment results are publicly available at a GitHub repository [12].

The contributions of this paper are:

- The development of FIFE, a generic VM evaluation framework with IaC support.
- The development of a Bayesian optimization VM searching system within FIFE which allows applying parallel Bayesian optimization to decrease search time when finding optimal VMs from multiple providers.
- An evaluation of search performance of Bayesian optimization system between single providers, i.e. Google Cloud Platform (GCP), and both AWS EC2 and GCP.
- An evaluation of both the single process and parallel versions of the Bayesian optimization system, which demonstrates that the parallel version has the potential to significantly reduce search time.

The rest of the paper is organised as follows. §II introduces IaC and Bayesian optimization. In §III, we present an overview of FIFE, followed by a case study of setting up a parallel Bayesian optimization search system in §IV. The evaluation is presented in §V, and related work and future work is discussed in §VI. §VII concludes this paper.

II. Background

A. IaC and Terraform

Infrastructure-as-Code (IaC) is an approach for automating infrastructure management with the ideas from software practice [13]. By treating infrastructure as software, as opposed to physical entities, the practices from software development, e.g. version control, can then be applied to automate management. This approach is, in particular, suitable for cloud computing, because cloud virtualisation has provided an abstraction over the underlying physical hardware. Machine-readable scripts, typically, in a declarative style, are used to improve automation of machine provisioning. One popular IaC tool is *Terraform* [14]. It provides an additional layer of abstraction to describe cloud resources from different cloud providers and uses a JSON based high-level configuration language to automate the configuration of cloud resource. Figure 1 shows an example of AWS EC2 configuration in Terraform, where `region` specifies the location of a desirable AWS datacenter, `instance_type` is the VM type and *ami* is the system image to be run on the VM. By applying `terraform apply` in the command line, an EC2 VM instance will be automatically provisioned from AWS.

```
provider "aws" {
  profile = "default"
  region = "eu-west-2"
}
resource "aws_instance" "example" {
  instance_type = "m5.large"
  ami = "ami-09ead922c1dad67e4"
}
```

Figure 1. An example of AWS EC2 Terraform script

B. Bayesian optimization

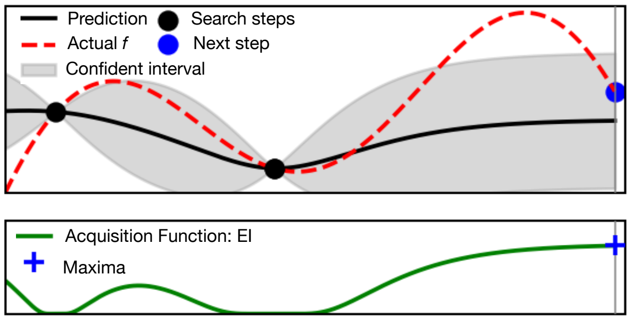
Bayesian optimization takes a Bayesian reasoning approach to approximate parameters to minimise or maximise a black-box objective function that is expensive to evaluate. With a set of observations, e.g.

$$\mathcal{D} = \{(x_1, y_1), ...(x_n, y_n)\}$$

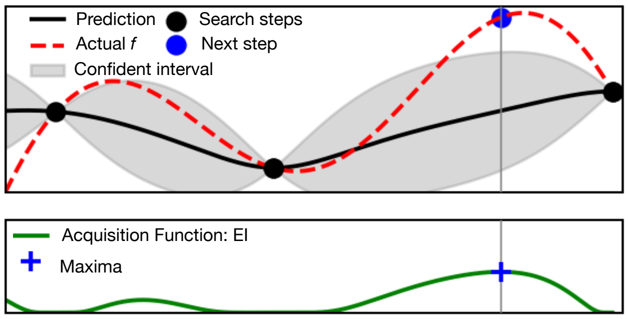
and a prior over the function, the posterior can be obtained:

$$P(f|\mathcal{D}) \propto P(\mathcal{D}|f)P(f). \tag{1}$$

where f is the target function. The two main configurations for Bayesian optimization are a *prior* model and an *acquisition function*. A prior model specifies the assumption about the objective function and the model is updated when a new observation is obtained; an acquisition function is used to decide the choice of the next search step based on its estimation of search space.



(a) t = 2



(b) t = 3

Figure 2. An example of Bayesian optimization's working process to approximate the target black-box function where t is the number of search steps has been taken. Each sub-figure shows that, with previous t search steps, Bayesian optimization attempts to choose the next search step are the values that maximise the *acquisition function*. The solid line and the shadow area illustrate the estimated function with confident interval.

One popular setting is using *Gaussian Process* (GP) as a prior and *Expected Improvement* (EI) for an acquisition function. GP is a stochastic process where the random variables follow a Gaussian distribution. Assuming the objective function is a function mapping from D dimentional data to $\mathbb{R}$, i.e.,

$$f \in \mathbb{R}^D \longrightarrow \mathbb{R}$$

GP assumes that $f(x_1)$ and $f(x_2)$ are close to each other if x_1 and x_2 are close to each other. GP is fully specified by:

$$\mu \in \mathbb{R}^D \longrightarrow \mathbb{R}, \qquad \mathcal{K} : \mathbb{R}^D \times \mathbb{R}^D \longrightarrow \mathbb{R}$$

where μ is a mean function and $\mathcal{K}$ is a covariance function. With a set of N observations, e.g. $\{(x_n, y_n)\}_{n=1}^N$, GP constrains f that $y_n \sim \mathcal{N}(f(x_n), v)$ where v is the variance of the noise. With the prior and the set of observations, a posterior can be obtained from (1). Use posterior to determine the next point to be evaluated according to an *acquisition function*. Some popular acquisition functions are *Probability of Improvement(PI)* [15], *Expected Improvement(EI)* [16] and *Thompson Sampling* [17].

A diagram[1] illustrating this process is shown in Fig-

[1]This diagram is generated based on the source code from [18].

ure [2]. Through this process, Bayesian optimization can find optimal or near-optimal solutions for any non-parametric problem in a relatively small number of samples compared to other optimization methods. In addition, whereas other methods, such as exhaustive search, may handle uncertainty through sampling results from the same inputs multiple times, Bayesian optimization can incorporate uncertainty (non-deterministic), into its estimates, further reducing the number of samples needed.

III. IaC based VM Evaluating Framework

In this section, we present FIFE, a generic framework for evaluating VMs from multiple cloud providers. We aim to design a framework to support the two main VM evaluation cases: searching for an optimal VM with a given target application and user-defined measurements and collecting VM performance data by running benchmarking tools. To achieve this, we abstract a generalised formalisation for both cases.

The parameters for choices of VM instances are typically categorical data variables, e.g. instance type (*c5* in *AWS EC2*) and vCPU numbers (2, 4, 8). The set of all possible VM parameters $\mathcal{T}$ is a subset of D dimension categorical variable, i.e.

$$\mathcal{T} \subset \mathcal{N}^D$$

The performance of a VM regarding a target application can be represented using the following objective function:

$$f : \mathcal{T} \longrightarrow \mathbb{R}$$

The actual measures of the 'performance' depend on the service-level-objective from users, e.g. latency or throughout. For both two VM evaluation scenarios, the process is to find the set of VM performance with given VM sets, i.e.

$$\mathcal{F}_{proc} : \mathcal{T} \text{ set} \longrightarrow (\mathcal{T} * \mathbb{R}) \text{ set}$$

where $(\_ * \_)$ is the symbol for a pair. Figure [3] illustrates the main functionality in $\mathcal{F}_{proc}$. $\mathcal{F}_{srch}$ is a search control to decide the next VM candidate to be evaluated, i.e.

$$\mathcal{F}_{srch} : (\mathcal{T} * \mathbb{R}) \longrightarrow \mathcal{T} \text{ option}$$

where `option` denotes that return value can be none. That is no need to continue evaluation.

Selector $\mathcal{F}_{slct}$ and deployer $\mathcal{F}_{deploy}$ provide IaC support to automate the process of VM provisioning, running application and pulling performance measurement data. Interpreter $\mathcal{F}_{intr}$ then obtains performance measurements by processing the data, and then feeds the measurements back to $\mathcal{F}_{srch}$.

Based on the abstraction, we designed our framework as a process with four main sub-processes, which are *search control*, *instance selection*, *deployment* and *interpretation*. Each sub-process is controlled by a component. In *search control*, a *searcher* holds the logic of search, e.g. which

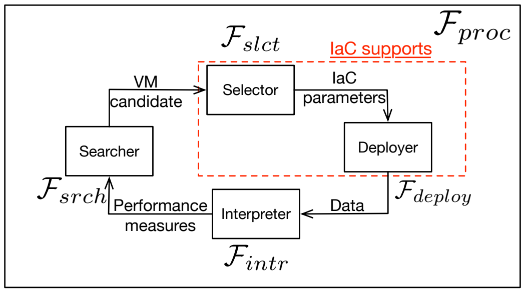


Figure 3. An abstract process of evaluating VMs with IaC supports.

VM to evaluate and when to stop; in *instance selection*, a *selector* keeps a list of VM candidates as a search space, and it translates the next search candidate as variables to be updated in the IaC script; in *deployment*, a *deployer* then updates the value of variables in IaC script, and then provisions the VM and runs the application accordingly; and in *interpretation*, an interpreter pulls log data to parse and calculates the 'unit of interests', i.e. objective measures.

One of the main design objectives of FIFE is to reduce the human effort involved in evaluating VMs. Once users set up a system with the framework, the rest of the evaluation process can be fully automated. As illustrated in Figure [4], there are two main stages, *setup stage* and *evaluation stage*. The process in the setup stage is where most of the human interactions are required. With the given target application and VM search space, users only need to configure it once. Four inputs are required from users:

- A search strategy: The *searcher* component applies a search strategy to evaluate VM instances for a user-defined objective measurement. In FIFE, we provide two implementations: Bayesian optimization and exhaustive search. Users can provide their implementations of search strategies, and then configure the framework to set up a system accordingly.
- VM search space: We take a list of VM data from different cloud providers in CSV format. The data should include the key attributes of each VM, e.g. vCPU cores, and the instance type. Users can reuse the list among different settings of the framework, e.g. single providers, multiple providers, a full list of all VM instances, or a subset of VMs.
- IaC scripts and cloud credentials: A IaC script together with cloud credentials are required to deploy the target application in the cloud. Currently our implementation supports *Terraform* scripts [19].
- A log/metrics parser: Users need to provide the functions to calculate the measure of interests from the log or monitoring metrics. Examples of measures are latency, completion time and benchmark scores.

For the purposes of reusability, we save each configuration as a JSON profile. More details are presented with a Bayesian optimization system case study in §IV.

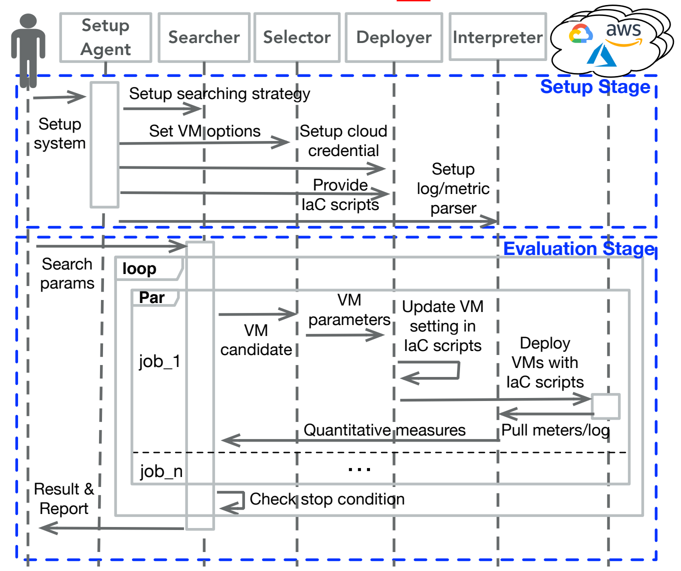


Figure 4. A workflow showing the processes involved in the framework.

Once set up, users can automate the VM evaluation process by providing search parameters, e.g number of parallel search. Figure 4 depicts the details of the whole process: at the evaluation stage, the *searcher* component drives forward the evaluation process by iterating through chosen VM candidates in the VM search space until a user-defined condition is met. Searcher chooses a VM candidate and starts a *job* to provision the target application and then pulls the log/metrics for quantitative measurement calculation before the VM is terminated. Note that jobs can be executed in parallel depending on the search parameters. A report containing the result of the desirable VM, as well as the whole search history, will be available for users to review.

IV. Case Study: Bayesian Optimization VM Searching System

In this section, we illustrate details of setting up a system with FIFE through a case study of a Bayesian optimization VM searching system. The scenario is that users want to find the most cost-efficient VM for a video decoding application from a set of AWS EC2 and GCP VMs shown in Table I. A system will be set up to deploy a benchmarking tool, called *vbench* [20], to get benchmark scores for each VM, to find the optimal instance type by considering both performance and cost.

The use of Bayesian optimization in VM searching was originally presented in *Cherrypick* [4]. Here, we extend this idea with parallel Bayesian search and IaC support. As with Cherrypick, our system is implemented using an existing Python Bayesian optimization library, called *spearmint* [21]. Parallel Bayesian search was developed by Snoek et al. [21]. It allows one Bayesian optimization procedure to evaluate multiple points at the same time, and Bayesian optimization can decide the next evaluation point even when a set of points are being evaluated. The parallel Bayesian search is implemented by calling the interface of *spearmint* with proper parameters.

Table I
A List of VM options from GCE and AWS EC2

Instance Type	Provider	Category	vCPU	Price ($/hour)
n1-standard-2	GCE	General	2	0.095
n1-standard-4	GCE	General	4	0.19
n1-standard-8	GCE	General	8	0.38
n1-highcpu-2	GCE	CPU	2	0.0709
n1-highcpu-4	GCE	CPU	4	0.1418
n1-highcpu-8	GCE	CPU	8	0.2836
n1-highmem-2	GCE	Memory	2	0.1184
n1-highmem-4	GCE	Memory	4	0.2368
n1-highmem-8	GCE	Memory	8	0.4736
m5.large	AWS	General	2	0.096
m5.xlarge	AWS	General	4	0.192
m5.2xlarge	AWS	General	8	0.384
c5.large	AWS	CPU	2	0.085
c5.xlarge	AWS	CPU	4	0.17
c5.2xlarge	AWS	CPU	8	0.34
r5.large	AWS	Memory	2	0.126
r5.xlarge	AWS	Memory	4	0.252
r5.2xlarge	AWS	Memory	8	0.504

```
{
  searcher : {
    name : "bo_searcher",
    concurrent : 2
    eistop : 0.1,
    jobstop : 6,
    search_dimension : "cpu_prd_typ"
  },
  selector : {
    vm_options : "vm_eval_set"
  },
  interpreter : "vbench_by_cost",
  deployer {
    name : "vm_docker_deployer",
    docker_image : "briggsby/vbench",
    docker_cmd : "vod house"
  }
}
```

Figure 5. A JSON configuration for a Bayesian optimization system

Figure 5 shows a JSON configuration file for the system setup process. We will walk through this JSON file in the rest of this section.

A Bayesian optimization search strategy: To set up a search strategy, users need to provide the name of their search function with the `name` field. All the fields below are search parameters. The Setup Agent dynamically links the function with the search API in the framework, with the search parameters as the arguments for the function. Users only need to specify the name of the searcher functions.

The Bayesian optimization searcher takes the default Gaussian process from the spearmint library as a prior over the objective function, and uses Expected Improvement (EI) for the acquisition function. For the search parameters, `concurrent` is a general parameter to specify the numbers of parallel search; `eistop` and `jobstop` are Bayesian optimization specific parameters to specify the termination condition, which are EI thresholds and a minimum number of search steps, respectively.

Apart from the parameters for search control, the searcher also needs to understand the dimensions of the search space. This is specified with a search space dimension configuration JSON file in `search_dimension`. In this example, the search space is constructed as a three-dimensional space, which is composed of a cloud provider, vCPU numbers and an instance category, as shown in Figure 6. Users can choose to reduce or increase the dimension of the search space by adding or remove a `variable` definition.

```
variable {
 name: "CPU"
 type: ENUM
 options: 2
 options: 4
 options: 8
}
```

```
variable {
 name: "Provider"
 type: ENUM
 options: "aws"
 options: "gcp"
}
```

```
variable {
 name: "Category"
 type: ENUM
 options:"General"
 options: "Memory"
 options: "CPU"
}
```

Figure 6. Configure of dimensions of search space

Encoding the search space with IaC: To make the selector understand the VM candidate specified by the searcher, users need to set the list of all VM options, e.g. Table I, in `vm_options` under the block of `selector`. For each VM, the list should include the data that is used to define the search dimensions, the data that are not part of search dimensions but are necessary for provisioning, e.g. *region* and *ami*, and the data for calculating the objective measures in Interpreter, e.g. price.

Automating IaC provisioning: Here users only need to provide the relative pathname of the Terraform scripts to be deployed. Note that, for the case of multiple cloud providers, the script for each provider should be supplied. When provisioning, the system will automatically update the related Terraform variables in the script to change the VM setting, e.g. *instance_type* for AWS.

In the configuration file, field `name` under the `deployer` block specifies the function which Deployer calls to provision Terraform scripts. We have provided two implementations for the provisioning function, `vm_deployer` and `vm_docker_deployer`. The only difference is that `vm_docker_deployer` allows additional parameters to specify the docker image and docker command.

Objective measurement calculation: For the Interpreter, users need to provide a function for calculating a quantitative objective based on logs or metrics. The measure is used to provide Searcher feedback in order to decide on the next search step. In this case study, we implement a function to parse vbench log files for the benchmark score, and then to calculate the value of VM by dividing the score by the cost ($ per hour).

The requirement of resource demands can also be reflected using the quantitative objective measure in the configuration of the interpreter. For example, insufficient resources, such as the number of CPU cores and memory size, would result in bad performance or exception.

Once the Bayesian optimization system has been set up, users can automate the rest of the VM search process by ‘pushing-a-button’, and a JSON configuration file can be saved for quick reconfiguration, e.g., changing the value of `concurrent` to 1 for single process search. In the next section, we will show the results of running the system.

V. Evaluation

In the rest of this section, we will show the results of running experiments with both the Bayesian optimization system and an exhaustive sampling system for different purposes. The objectives of the experiments are two-fold: firstly, to assess the effectiveness of the Bayesian optimization system, and secondly, to compare the impact of parallel search between the case when VMs are from single or multiple providers.

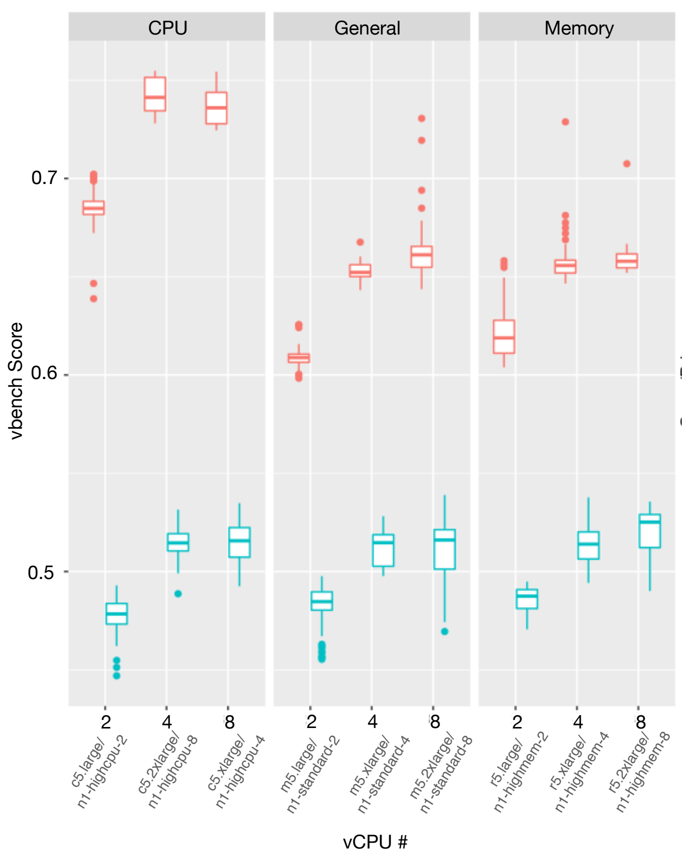
```
{
  searcher : {
    name : "exhaustive_searcher",
    concurrent : 2
    iteration: 20
  },
  selector : {
    ...
  },
  interpreter : "vbench_scores",
  deployer : {
    ...
  }
}
```

Figure 7. A JSON configuration for an exhaustive sampling system

FIFE can also be used to instantiate a VM benchmarking system to collect/sample data by running benchmarking tools in all VM candidates. To illustrate, to configure an exhaustive sampling system with vbench, users only need to edit the JSON configuration file as shown in Figure 7, where `...` means the settings remain the same as those in Figure 5. `exhaustive_searcher` is a pre-set searcher to enumerate all VM options N iteration where N is 20 in this example. `vbench_scores` parses vbench score from a log file.

A. Experimental setup

vbench measures an instance type’s relative rate of transcoding of a single 5-second 1920x1080 video file, returning a score of 0 if the quality was below a given threshold. We use the value obtained by diving *vbench* score by cost as an objective measure to rank VM candidate. The VM candidates, which are configured by a user-defined input for search space setup, are shown in Table I. Note that a VM instance can be uniquely determined by the provider, vCPU number and the category, e.g. *c5.large* (AWS EC2, 2 vCPU and CPU). To evaluate the Bayesian optimization searching



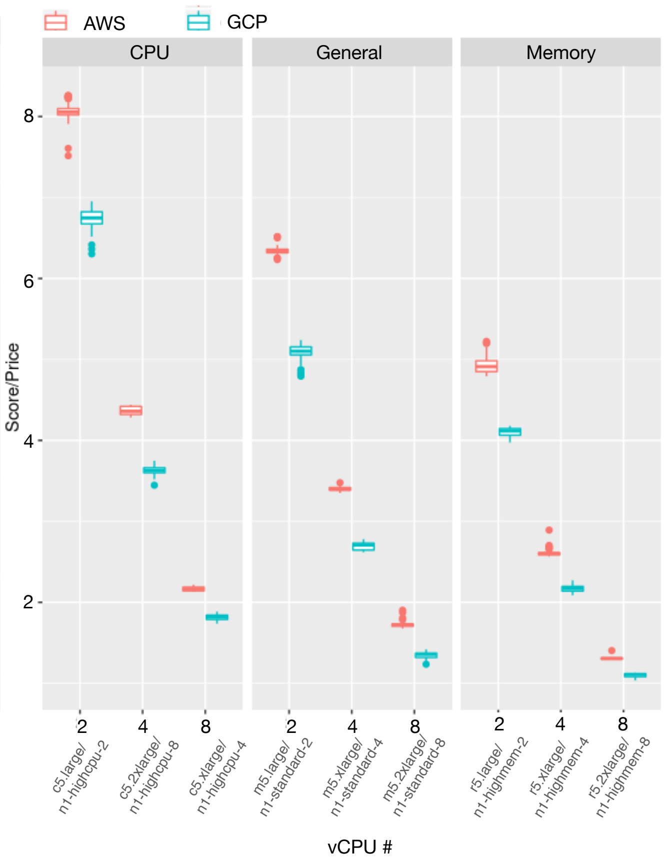


Figure 8. Boxplots showing the distribution of vbench scores (left) and objective measure values (right) for each VM in the search space. The distributions are derived from the results of 20 evaluations per VM by running the exhaustive search system. Each VM instance is specified by cloud provider (colour), instance category (columns), and number of vCPUs (x-axis). The higher the y-axis value is, the better ranking the VM gets. The plots are used to justify the recommended VM from the Bayesian search system.

system, we need to compare search results with a VM ranking. This is done by running the exhaustive sampling system to get the performance of all the VMs. Note that, to address the dynamics of VM performance, each VMs is sampled 20 times. We then run the Bayesian optimization VM searching system with various experiment settings, i.e. when VMs are either from a single provider (GCP only) or multiple providers (AWS & GCP). Searches were stopped at $N > 6$ and $EI < 10\%$. A summary of the settings are given in Table II, where **Parallelism** means parallelism for Bayesian search.

Table II
VARIOUS EXPERIMENT SETUP: PARALLEL CONFIGURATIONS OF BAYESIAN OPTIMIZATION WITH SINGLE/MULTIPLE PROVIDER(S).

Parallelism / Providers	1	2	3
GCP only	✓		✓
AWS & GCP	✓	✓	✓

B. Exhaustively Sampling the Search Space

The results of the exhaustive search are shown in Figure 8, where Figure 8 (left) shows the distribution of the raw scores of performance, and value (performance / cost) for Figure 8 (right). For the y-axis of both figures, a higher value means a better VM ranking, and a shorter range box indicates less variance. In general, relative standard deviations for all scores measured were low, the highest being $\pm 3.23\%$ for the n1-standard-8 instance type. No relationship with this variance was found for any of the three search variables.

The analysis was performed in RStudio [14]. Statistical results are quoting P-values which look at the probability that the means of two groups are significantly different. More details about statistical testing methods and results can be found in [22]. Full logs and scripts used for analysis are available on [12].

The raw scores show a significant difference between 2 vCPUs and either 4 or 8 vCPUs, and between 4 and 8 vCPUs, where the score increases with vCPU number, when accounting for differences between provider and machine category. However, the effect size is dramatically diminished between the 4 to 8 categories compared to that between the 2 to 4 categories.

However, once the scores are divided by that machine's hourly costs, to give the objective measure value, there is much less overlap. The c5.large. machine was significantly better than the next best option, the n1-highcpu-2, as well as all other options. AWS EC2's machine types consistently outperformed GCP's equivalents of the same category and vCPU number at both raw score and value for money. While provider was the most important determining factor as to whether a cloud configuration gave a higher raw score than others, it was the least important determining factor as to whether a cloud configuration gave a higher objective measure value. For example, the n1-highcpu-2 still gave significantly better values than the m5.large, or the c5.xlarge making it the second most cost-efficient option.

C. Parallel Bayesian optimization

Table III
Top 3 rankings of VM by Scores/Price for single (left) & multiple (right) provider(s)

Ranking	GCP VMs Only	Ranking	GCP & AWS VMs
1	n1-highcpu-2	1	c5.large
2	n1-standard-2	2	n1-highcpu-2
3	ni-highmem-2	3	m5.large

Table IV
Summary of recommended VMs from various settings of Bayesian optimization.

Expriment Settings:	Search Result	Remarks	Counts (Total: 20)
GCP, Parallel 1	n1-highcpu-2	optimal VM	17
	n1-standard-2	2nd best VM	3
GCP, Parallel 3	n1-highcpu-2	optimal VM	15
	n1-standard-2	2nd best VM	5
GCP&AWS, Parallel 1	c5.large	optimal VM	18
	m5.large	2nd best VM	2
GCP&AWS, Parallel 2	c5.large:	optimal VM	18
	n1-highcpu-2	2nd best VM	1
	m5.large	3rd best VM	1
GCP&AWS, Parallel 3	c5.large	optimal VM	11
	n1-highcpu-2	2nd best VM	2
	m5.large	3rd best VM	7

For each experiment configuration shown in Table II, we ran the Bayesian optimization searching system 20 times because the search results are non-deterministic. We would like to compare different settings from the following three different perspectives: *search accuracy*, *search time* and *search cost*.

The search results from these experiments are shown in Table IV. For ease of presentation, a summary of the rankings of VMs generated from Figure 8 (right) are shown in Table III.

For the case of a single provider with a single process job (GCP only with non-parallel), in 17 of 20 experiments, Bayesian optimization was able to reach the same conclusions as exhaustive search in only 6-12 samples, with a mean performance that was approximately 96.9% of that from the optimal possible configuration. It shows a significant improvement in reducing the search time compared to exhaustive search, as exhaustive search needs to sample each VM instance type multiple times due to the nature of performance variances. That is 20 samples for each of 9 VM instance types.

For the case of multiple providers with a single process, the correct optimal instance was predicted in 18 out of 20 evaluations, in only 6-11 samples (97.9% relative mean performance). For our evaluation, despite doubling the search space, Bayesian optimization was no less effective.

Each diagram compares the performance of various parallel configurations of Bayesian optimization in term of

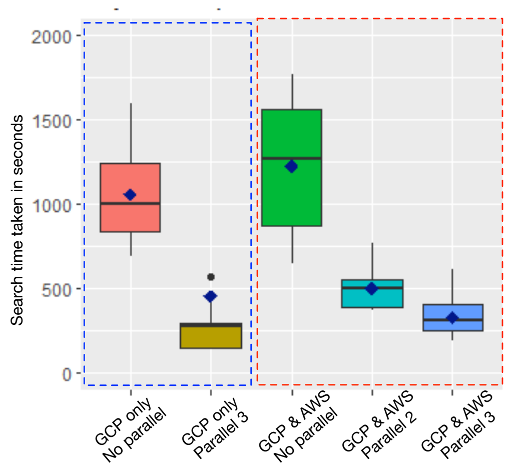


Figure 9. Comparison of search time between different search settings. Each boxplot shows the distribution of search time for each setting. The dot frames group the comparable settings, e.g. search space of single provider or multiple providers. Results show that enabling parallel in Bayesian optimisation in search of the optimal VM from multiple cloud providers can significantly improve reduce search time.

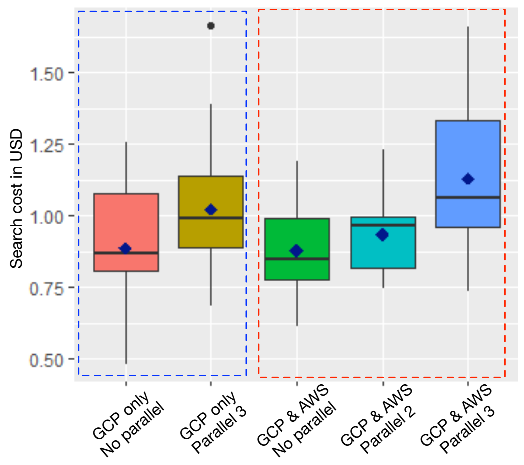


Figure 10. Comparing of search cost in US dollar between different search settings. Each boxplot shows the distribution of actual search cost in dollar for each setting. The dot frames group the comparable settings, e.g. search space of single provider or multiple providers. Results show that, with a proper setting, e.g. 2, the search cost of parallel Bayesian optimisation would not be increased comparing to the single process version.

different measurements.

We also compare the search time and the search cost between settings, which are shown in Figure 9 and Figure 10. Search time was calculated from the difference in time between the last and first completed job. This time does not include initial setup or time taken to terminate any unfinished jobs once the search was stopped. Relative search cost was calculated by summing the hourly cost of instances samples, including repetitions. This assumes that either every job took the same amount of time regardless of instance, or that the applications were short enough (<1 minute) to accrue only the minimum possible cost for provisioning each instance type. Even on the slowest instance type tested, our vbench benchmark completed in approximately 35s, so we are confident that this second assumption holds.

We are particularly interested in the results of multiple providers with parallel search, i.e. parallel > 1. By allowing 2 parallel jobs to be run at any one time, the time taken for the search was reduced to less than half that of before ($P < .001$). Along with this dramatic reduction in time taken, there was no rise in search cost ($P = .95$). The number of completed jobs performed was still between 6 and 10 in all experiments. Even more impressively, there was also no loss in the optimal value returned (as compared in Table IV), with it still choosing the correct instance in 18 out of 20 repetitions.

However, increasing the number of parallel jobs higher, to 3, caused a dramatic reduction in effectiveness. For multiple providers, the experiment returned the optimal instance in only 11 out of 20 repetitions. The increased maximum allowed job concurrency also increased the search cost compared to 2 parallel jobs ($P = .03$), and did not lead to any significant further decrease in search time ($P = .81$). To ensure this held true even in a reduced search space, the experiment was repeated with 3 parallel jobs but only a single provider, to similar results.

Bayesian optimization was shown to be very resilient to the increased search space introduced from adding a cloud provider, with no loss of effectiveness or efficiency. A difference in search effectiveness may have been seen if the performance difference between providers was small rather than large, or if the relationship was more complex and dependent on other variables.

When 2 parallel jobs were running, the effectiveness and cost-efficiency of the search were unaffected, but the time taken to perform it was dramatically reduced. Past 2 parallel jobs, allowing further parallel reduced the search's effectiveness without any further benefit. This suggests that it is beneficial for users to allow 2 parallel jobs, but not to allow any more, at least for our example use-case.

1) Additional case studies: Apart from the *vbench* [20] presented in the last section, More case studies with different applications, e.g. *sysbench* [23], *Cloudsuit3* [24] and a Ping server which is a naive web server to response ping request. are available at [22].

VI. Related work & Future work

To help users to select the right VM, cloud providers offer high-level guides based on basic metrics such as vCPUs and memory [25, 26]. They also provide recommendations on VM types based on the monitoring data collected from the applications after deployment.

Current research addresses the challenge mainly from three approaches: benchmarking, data-driven model fitting and VM searching. For the benchmarking approach, it investigates the resource utilisation patterns for a specific application and then develop a tailored benchmark for performance estimates, e.g. the *YCSB* suite [27] for database systems. There are also research aiming at facilitating automation of benchmarking process, e.g. *Google PerfKit Benchmarker* [28] and *Cloud WorkBench*[2]. Systems like *Ernest* [6] and *Paris* [5] take a data-driven approach to fit a model from the a profile of VM benchmark data. An example of the search approach is *Cherrypick* [4]. It considers VM selection as a black-box searching problem and takes a Bayesian optimization approach to iteratively sample the VM configuration space to search for an optimal VM.

Comparing to the existing work above, our framework generalises the scenarios of sampling VM performance data with benchmark tools and selecting optimal VM instances. In particular, our framework emphasises on the integration of IaC tool, *Terraform* and *Docker*, in the process.

The framework does not compete with the current solutions of VM benchmark tools, e.g. DocLite [9], and optimal VM selection, e.g. Ernest [6] and *CherryPick* [4]. FIFE does not preclude these solutions, and their designs can be incorporated into it. Even PARIS [5], with its markedly different approach to VM selection compared to other solutions, could be replicated in our framework by implementing it as a composition of a benchmark procedure system and a random forest model based searching system.

For our Bayesian optimization system, it follows the same idea of applying Bayesian optimization in VM selection from *CherryPick* [4]. We expanded on their findings to show that parallel jobs with proper setting can reduce search time and cost, while too many parallel jobs can reduce accuracy can still reduce search time but resulting in increased cost. As we only ran the Bayesian parallel search on one experiment setting, the data is limited for investigating the factors that might have impact on the configurations of the parallel search. In future, we plan to run more settings for such investigation.

In our case study, we only consider a simple scenario where users concern about the value of VM, i.e. ($performance/cost$). In future, we would like to investigate a more complicated case where benchmarking is significantly expensive. One possible solution is to put the cost of benchmarking as a penalty factor in the stop searching condition to balance the overhead of VM searching and the advantages of a better VM. We also would like to evaluate multiple-tiers applications that require more than one VM instance and the target performance measure is defined at application level rather than at application component level.

As stated throughout the design and implementation of our framework and system, there are numerous avenues for extension of our system through the development of different component modules. In particular, we present three suggestions for further research:

GPU Instances: Our results do not tackle set up of drivers necessary to make use of GPU instances. GPU instances are well suited for live-streaming media servers [20]. A Deployer would have to be developed that can install necessary drivers and ensure an application can make use of a VM's available GPU.

Serverless services: Serverless services are particularly opaque as to their performance. Once APIs are available for Terraform or other IaC tools for services such as Google Cloud Run our framework could illuminate performance differences between available serverless providers.

Expanded search space: In general, our study was limited in its search space. Our provided Selector dataset spans the instances from GCP and AWS EC2 that do not require additional requested quotas. Expanding this to larger machines or other providers, such as Microsoft Azure, is a straightforward way to improve its scope.

VII. Conclusion

We have presented a generic IaC based framework, called FIFE, to automate the process of VM evaluation. FIFE has a modular design. It is highly configurable to allow users to set up systems depending on the target application and their objectives, e.g. searching for an optimal VM, and sampling VMs by benchmarking. We have also shown a Bayesian optimization system to illustrate the process of setting up a system from FIFE. An evaluation was carried out to assess the search performance of the system with respect to the search space with multiple cloud providers and parallel search. The results showed that in the case of two providers, when 2 parallel jobs were allowed, the effectiveness and cost-efficiency of the search were unaffected, but the time taken to perform it was dramatically reduced.

Acknowledgement

This work is a part of the ABC project (Adaptive Brokerage for the Cloud) funded by EPSRC EP/R010528/1.

References

[1] P. Leitner and J. Cito, "Patterns in the chaos - a study of performance variation and predictability in public iaas clouds," *TOIT 14*, vol. 16, no. 3, pp. 1–23, 2014.

[2] J. Scheuner and P. Leitner, "Estimating cloud application performance based on micro-benchmark profiling," in *2018 IEEE CLOUD*, 2018, pp. 90–97.

[3] C. Laaber, J. Scheuner, and P. Leitner, "Software microbenchmarking in the cloud. How bad is it really?" *Empirical Software Engineering*, pp. 1–40, apr 2019.

[4] 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 *NSDI 14*, 2017, pp. 469–482.

[5] 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 *SoCC 17*, 2017, pp. 452–465.

[6] S. Venkataraman, Z. Yang, M. Franklin, B. Recht, and I. Stoica, "Ernest: efficient performance prediction for large-scale advanced analytics," in *NSDI 16*, 2016, pp. 363–378.

[7] N. Ekwe-Ekwe and A. Barker, "Location, location, location: Exploring Amazon EC2 spot instance pricing across geographical regions," in *CCGRID 2018*, 2018, pp. 370–373.

[8] B. Varghese, O. Akgun, I. Miguel, L. Thai, and A. Barker, "Cloud benchmarking for maximising performance of scientific applications," *IEEE TCC*, vol. 7, no. 1, pp. 170–182, 2016.

[9] B. Varghese, L. T. Subba, L. Thai, and A. Barker, "Container-based cloud virtual machine benchmarking," in *IC2E 2016*. IEEE, 2016, pp. 192–201.

[10] A. Barker, B. Varghese, J. S. Ward, and I. Sommerville, "Academic cloud computing research: Five pitfalls and five opportunities," in *HotCloud 14*, 2014.

[11] J. Mockus, *Bayesian approach to global optimization: theory and applications*. Springer Science & Business Media, 2012, vol. 37.

[12] Paper Resource Webpage. [Online]. Available: https://github.com/lyhlbyl/AutomatedBayesCloudSelection

[13] K. Morris, *Infrastructure as code: managing servers in the cloud*. O'Reilly, 2016.

[14] R Core Team, "R: A Language and Environment for Statistical Computing," Vienna, Austria, 2018.

[15] H. J. Kushner, "A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise," 1964.

[16] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions," *Journal of Global optimization*, vol. 13, no. 4, pp. 455–492, 1998.

[17] W. R. Thompson, "On the likelihood that one unknown probability exceeds another in view of the evidence of two samples," *Biometrika*, vol. 25, no. 3/4, pp. 285–294, 1933.

[18] A. Agnihotri and N. Batra, "Exploring bayesian optimization," *Distill*, 2020, https://distill.pub/2020/bayesian-optimization.

[19] Y. Brikman, *Terraform: Up & Running: Writing Infrastructure as Code*. O'Reilly Media, 2019.

[20] A. Lottarini, A. Ramirez, J. Coburn, M. A. Kim, P. Ranganathan, D. Stodolsky, M. Wachsler, A. Lottarini, A. Ramirez, J. Coburn, M. A. Kim, P. Ranganathan, D. Stodolsky, and M. Wach, "vbench : Benchmarking video transcoding in the cloud," *ASPLOS 18*, pp. 797–809, 2018.

[21] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," *NIPS 2012*, pp. 2951–2959, 2012.

[22] J. Briggs, “Automated Bayesian optimization for cloud configurations across multiple providers using a generalizable framework,” Master’s thesis, 2019. [Online]. Available: https://github.com/Briggsby/AutomatedBayesCloudSelection/blob/master/Report/Report2.pdf

[23] A. Kopytov, “Sysbench manual,” *MySQL AB*, pp. 2–3, 2012.

[24] T. Palit, Y. Shen, and M. Ferdman, “Demystifying cloud benchmarking,” in *ISPASS 2016*. IEEE, 2016, pp. 122–132.

[25] C. the Right EC2 Instance Type for Your Application. [Online]. Available: https://aws.amazon.com/blogs/aws/choosing-the-right-ec2-instance-type-for-your-application/

[26] A. sizing recommendations for VM instances. [Online]. Available: ApplyingsizingrecommendationsforVMinstances

[27] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, “Benchmarking cloud serving systems with YCSB,” in *SoCC 2010*, pp. 143–154.

[28] Google PerfKit Benchmarker. [Online]. Available: http://googlecloudplatform.github.io/PerfKitBenchmarker/