Fast and Low-cost Search for Efficient Cloud Configurations for HPC Workloads

Vanderson M. do Rosario[1], **Thais A. Silva Camacho**[1], **Otávio O. Napoli**[1], **Edson Borin**[1]

[1]Institute of Computing (IC) – University of Campinas (UNICAMP)
Campinas – SP – Brazil

Abstract. *The wide variety of virtual machine types, network configurations, number of instances, among others configuration tweaks, in cloud computing, makes the finding of the best configuration a hard problem. Trying to reduce costs and resource underutilization while achieving acceptable performance can be a hard task even for specialists. Thus, many approaches to find these optimal or almost optimal configurations for a given program were proposed in the literature. Observing the performance of an application in the cloud takes time and money. Therefore, most of the approaches aim not only to find good solutions but also to reduce the search cost. One of those approaches relies on Bayesian Optimization, which analyzes fewer configurations, reducing the search cost while still finding good solutions. Another approach found in the literature is the use of a technique named Paramount Iteration, which enables users to reason about cloud configurations' cost and performance without executing the application to its completion (early-stopping) - this approach reduces the cost of each observation. In this work, we show that both techniques can be used together to do fewer and lower-cost observations. We demonstrate that such an approach can recommend solutions that are* $1.68\times$ *better on average than Random Searching and with a* $6\times$ *cheaper search.*

1. Introduction

The increasing availability of computing resources on public cloud service providers in the last years is allowing companies and researchers to migrate their infrastructure from on-site to on-demand external cloud off-site infrastructures. Among other benefits, the ease of accessing powerful resources on-demand and at any time has empowered many academics that did not have easy access to accelerators, clusters, and other HPC infrastructures.

Under the cloud computing model, users pay only for what they use; hence, users want to wisely use cloud resources, reducing their costs without compromising performance. For academics running HPC workloads, the difference between choosing one computing resource over another can mean more than a 20-fold difference in total computation cost. Not surprisingly, solving the problem of selecting the most efficient cloud configuration for any given application became a relevant research topic in the last few years [Alipourfard et al. 2017, Hsu et al. 2018b, Brunetta and Borin 2019].

Most of the first approaches to this problem consisted of creating or training performance models that describe the applications and the cloud resource performances. Thus, offline, beforehand, deciding which cloud configuration to use. However, this showed to be poor as it did not consider the dynamic performance of the cloud that is affected by the concurrent use and allocation of the physical resources, such as Virtual Machines, or VMs. Another possible solution is to dynamically try all possibilities and choose the best configuration; however, this approach can be expensive and hard to pay off. To reduce the cost of the dynamic search, tools such CherryPick [Alipourfard et al. 2017] and Arrow [Hsu et al. 2018b] treat this problem as a black-box function optimization problem and employ *Bayesian optimization* (BO) strategies to reduce the number of observations and find an efficient solution.

Orthogonal to the approach of reducing the number of observations, Brunetta and Borin [Brunetta and Borin 2019] proposed the use of Paramount Iterations (PI) to reduce the cost of each observation itself. PI early-stops the execution of HPC programs but still collecting sufficient information to estimate the relative performance of different cloud configurations.

In this work, we propose an approach that combines BO and PI to reduce the cost of searching for efficient cloud configurations even further. Combining both we can reduce the number of observations and the cost of each observation. We evaluated our approach searching for cost-efficient cloud configurations for 15 HPC workloads (5 kernels and 3 different input sizes) and showed a six-fold reduction in average search cost when compared to not using PI. The main contributions of this work are:

- We proposed a novel approach to reduce the cost of searching for efficient cloud configurations for HPC workloads by combining *Bayesian optimizations* (BO) and *Paramount Iterations* (PI) together;
- We evaluate our approach with different BO techniques and show that the best technique may depend on the workload, defined by the application and its input data set. Nonetheless, the results indicate that the BO techniques perform consistently better than random and grid search.
- We discuss how to organize the search space to reduce the number of dimensions and improve the performance of BO techniques.

The remaining text is organized as follows: Section 2 discusses the state-of-the-art and maps our approach position in the literature; Section 3 defines the problem and the BO details, mainly focusing on reducing the number of observations in the cloud VM instance configuration space; Section 4 further describes PI and shows how it can be used to reduce BO search cost even further; Sections 5 and 6 present the setup of our experiments and the experimental results; Finally, Section 7 lists our conclusions.

2. Related Work

Many public cloud providers exist such as Google Cloud Platform, Amazon AWS, and Microsoft Azure, each one delivering tons of possible Virtual Machines (VM) and cluster configurations to be instantiated. Each of these instantiations and providers result on different costs and system performances. Prior works report that there is not a one-size-fits-all VM type that is best for all workloads [Yadwadkar et al. 2017, Alipourfard et al. 2017, Herodotou et al. 2011, Hsu et al. 2018b]. Thus, finding and matching the best cloud provider and best VM configuration to cost-efficiently run a program became an important problem that has been approached by many authors [Hsu et al. 2018c, Hsu et al. 2018a, Hsu et al. 2018b, Wu et al. 2019, Brunetta and Borin 2019, Herodotou et al. 2011].

Moreover, the cloud infrastructure is dynamic [Li et al. 2010] and can have a high variation in performance [Ferguson et al. 2012], mostly because of the concurrent use of the physical resources from different VMs and users. Thus, the profiling collected from one run may not reflect a later one. Having that in mind, some authors proposed to dynamically explore the search space of instance's configurations. Techniques such as *random-search* or *grid-search* that extensively search the space of configurations could be used, but to decrease the number of configurations to be dynamically observed some works proposed the use of statistical methods such as *Bayesian optimization* using Random Forrest (RF) [Hsu et al. 2018b] or Gaussian Process (GP) [Alipourfard et al. 2017, Hsu et al. 2018a, Wu et al. 2019]. These approaches reduce the search cost by observing fewer configurations.

On the other hand, orthogonal to the approaches of reducing the number of observations to reduce the search cost, Brunetta and Borin [Brunetta and Borin 2019] proposed an approach to reduce the cost of each observation itself. They run a *grid-search* (test all possible configurations), but using a technique called Paramount Iteration to reason about the performance of HPC workloads with only a very small portion of their total execution. This early-stop technique showed to significantly reduce the number of resources needs to find the best configuration.

As far as we know, no work on the literature explore both early-stop and *Bayesian optimization* combined.

3. Cloud Configuration Search Problem

The problem of searching for a cloud configuration that minimizes the cost to run the experiments can be described as

$$\min cost(cfg) \tag{1}$$

where cfg is a cloud configuration and $cost$ is a function that describes the resources used to run a program p under the cloud configuration cfg. For our experimental purposes, we define cfg as:

$$cfg = (vm, n) \in VM \times \{1, 2, 4, 8, 16, 32\} \tag{2}$$

where $vm \in$ VM, VM is the set of all possible VM configurations in a specific cloud provider and n is the number of vm instances used to compose a computing cluster. The ordered pair (vm, n) is referred to in this paper as cloud configuration. So, for a given HPC workload p, we want to find the cloud configuration that minimizes the $cost$ of executing it.

This $cost$ is calculated from the time ($T(p)$) to execute a program p, the price ($price(vm)$) of each vm, n, and a random noise (ϵ) that comes from cloud provider context and the measurement mechanism.

$$p => cost(cfg) = T(p) \times price(vm) \times n + \epsilon \tag{3}$$

The random noise ϵ occurs because of virtualization technologies, as explained in previous work [Alipourfard et al. 2017, Brunetta and Borin 2019]. Finally, the search itself has a cost: the accumulated cost of all observations. Given a set of observations $O \subseteq VM \times \{1, 2, 4, 8, 16, 32\}$, the search cost is $\sum_{i=1}^{|O|} cost(O_i)$. A good search algorithm, therefore, solves (1) with the lowest possible search cost.

3.1. Approaches to the Cloud Configuration Search Problem

The optimization problem characterized before in Equation 1 can be classified as a Black-Box Function Optimization Problem (BBFOP). A BBFOP is such that it has an interior function that can only be understood through exterior observation and experimentation, being those observations expensive in some resources and the function is not possible to be derivative no matter the number of observations. Two straightforward ways to solve this optimization problem are the *grid-search* (test sequentially all possible inputs) and the *random-search* (randomly test possible inputs until some predefined stop criteria). A more sophisticated common way to approach this problem is to use a Sequential Model-Based Optimization (SMBO) that tends to do need fewer observations to find optimal or almost optimal solutions.

SMBO consists of a model for the function being optimized that is updated after every observation using a prior-posterior Bayesian approach. The Bayesian approach provides a posterior distribution of the black-box function and estimates the uncertainty that helps decides where to observe next, to find a maximum (or minimum). Frequent models used are Gaussian Process (GP) and Random Forrest (RF). An increasingly popular direction has been to use Gaussian Process (GP) to make smoothness assumptions on the function.

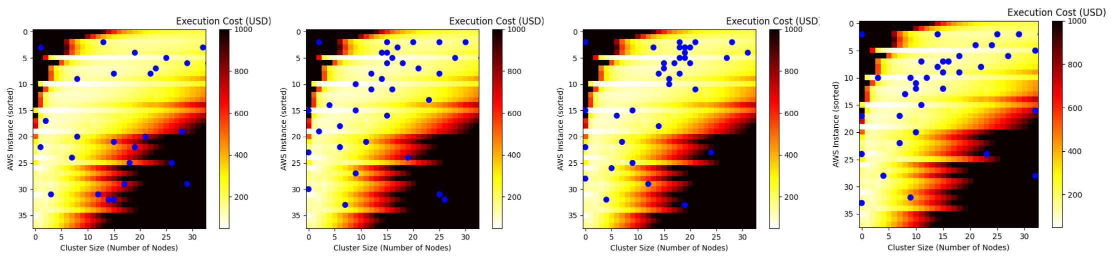
The observations used to update the model in SMBO are cherry-picking using information contained in the prior model. There are different strategies to pick the next point: to select the point that maximizes the probability of improvement (MPI) [Kushner 1964]; to select the point that maximizes the expected improvement (EI) [Močkus 1975]; or to select the point that has the upper confidence bound (UCB) [Srinivas et al. 2009] on the maximum function value. Each algorithm reduces the black-box function optimization problem to a series of optimization problems of known acquisition functions.

To illustrate how SMBOs model and acquire functions results and how they observe different points in the space, we model a fictional cloud space search using AWS VMs information and price. We created an application performance model inspired by Amdahl's Law and we ran 32 observations of *random-search*, Random-Forrest, GP-EI, and GP-MPI. Figure 1 shows the search space and the observations made by each strategy. Note that different from *random-search* RF and GP tend to focus on good configurations (it is able to learn through observations). Moreover, GP-MPI tends to more spatially explore the space while EI tends to maximize the best-found point by looking for nearby solutions. This is known as the "exploitation vs exploration" trade-off.

4. Reducing the Search Cost with Early-Stop

Sophisticated techniques, such as SMBO, reduces the search's cost by reducing the number of observations on the search space. In addition to this approach, one may want to reduce the cost of the observations themselves.

Brunetta and Borin [Brunetta and Borin 2019] showed that, in the context of HPC workloads on the cloud, it is possible to estimate the relative performance of different



(a) *Random-search*. (b) Random Forest. (c) GP-EI. (d) GP-MPI.

Figure 1. Distribution of 32 observations in a model-generated space for 4 SMBO strategies.

virtual machine types with very little execution time by only collecting information about few iterations of the main execution cycle of the application, or *Paramount Iteration* (PI), as suggested by the authors.

In experiments, the authors found that measuring the execution time of a few initial paramount iterations is enough to fully understand the performance of the whole application. They used it to accelerate a *grid-search* of configurations in the Microsoft Azure Cloud. Executing only the first iteration of an HPC application and stopping it, reduces drastically the cost of the observations.

In this work, we present a coupling between the two approaches to reduce the search cost (at least for HPC workloads). In other words, we reduce the number of observations using Bayesian Optimization, and use the *Paramount Iteration* to perform an early-stop and make each observation faster and cheaper.

5. Experimental Setup

5.1. Benchmark Suite

To evaluate our proposed approach, we execute the *Numerical Aerodynamic Simulation Parallel Benchmarks* (`NPB`) [Bailey 2011] on AWS virtual machines. NPB is a suite of parallel computer performance benchmarks that has five kernels and three simulated computational fluid dynamics (CFD) applications. Each benchmark comes with a set of inputs that are divided into the following classes: `S`, `W`, `A`, `B`, `C`, `D`, `E`, and `F`. Class `S` consists of small workloads for quick tests; class `W` is designed to be executed on workstations; classes `A`, `B`, and `C` contains standard workloads; and the classes `D`, `E`, and `F` are composed of large workloads.

In this work, we used `NPB` 3.4 with `MPI` support. As previous results [Brunetta and Borin 2019] showed that the best cloud configuration depends on application input and proved that 4 paramount iterations are a good estimate of application performance. We used the five kernels available with the input classes `C`, `D`, and `E`, a total of 15 workloads. Table 1 list the kernels used in our experiments.

We instrumented the kernels' code to report the execution time of each Paramount Iteration (PI) and to stop the execution after executing 4 paramount iterations. The modified benchmarks can be found in our lab repository[1].

[1]Modified ES-NPB benchmarks: to be available on publication

Table 1. Details of NPB's kernels used in this work.

Benchmark	Description	Language
EP	Embarrassingly Parallel	Fortran
FT	Discrete 3D fast Fourier Transform	Fortran
MG	Block Tri-diagonal solver	Fortran
IS	Integer Sort, random memory access	C
CG	Conjugate Gradient	Fortran

Table 2. VM Parameters and Price.

VM type	vCPU	MEM (GiB)	Network (Gbps)	Price (USD)	VM type	vCPU	MEM (GiB)	Network (Gbps)	Price (USD)
m5n.large	2	8	25	0.119	c5.12xlarge	48	96	12	2.04
m5n.xlarge	4	16	25	0.238	c5.18xlarge	72	144	25	3.06
m5n.2xlarge	8	32	25	0.476	c5.24xlarge	96	192	25	4.08
m5n.4xlarge	16	64	25	0.952	c5n.large	2	5.25	25	0.108
m5n.8xlarge	32	128	25	1.904	c5n.xlarge	4	10.5	25	0.216
m5n.12xlarge	48	192	50	2.856	c5n.2xlarge	8	21	25	0.432
m5n.24xlarge	96	384	100	5.712	c5n.4xlarge	16	42	25	0.864
m5dn.large	2	8	25	0.136	c5n.9xlarge	36	96	50	1.944
m5dn.xlarge	4	16	25	0.272	c5n.18xlarge	72	192	100	3.888
m5dn.2xlarge	8	32	25	0.544	r5.large	2	16	10	0.126
m5dn.24xlarge	96	384	100	6.528	r5.xlarge	4	32	10	0.252
c5.large	2	4	10	0.085	r5.2xlarge	8	64	10	0.504
c5.xlarge	4	8	10	0.17	r5.4xlarge	16	128	10	1.008
c5.2xlarge	8	16	10	0.34	r5.8xlarge	32	256	10	2.016
c5.4xlarge	16	32	10	0.68	r5.12xlarge	48	384	10	3.024
c5.9xlarge	36	72	10	1.53	r5.24xlarge	96	768	25	6.048

5.2. Cloud Provider and Configurations' Space

We chose the AWS cloud provider to execute our experiments. Nonetheless, it is worth mentioning that the techniques discussed here are not dependent on the cloud provider; hence, they can be easily used with other cloud providers.

AWS provides a wide variety of virtual machine (VM) types that one can instantiate. The list is organized into categories based on the characteristics of the physical machines that execute each VM type. Given the HPC context, we chose VM types from the compute-optimized (C5*), the general-purpose (m5*), and the memory-optimized (r5*) categories. Table 2 shows the characteristics of the 32 VM types selected for our experiments[2].

The cloud configurations search-space we explore in our experiments is composed of all pairs (vm, n) where vm indicates one of the 32 VM types in Table 2, and $n \in \{1, 2, 4, 8, 16, 32\}$, indicates the number of instances used on the configuration, also referred as the cluster size. Hence, there are a total of 192 possible configurations per program. In real-life scenarios, this number can become much larger if we consider different providers and more VM-type categories. Nonetheless, the search space selected for our experiments is already challenging, since executing all NPB kernels to their completion on all configurations would cost more than 600 USD.

[2]Information about the currently available AWS VM types can be found on `https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-types.html`

5.2.1. Using Fewer Dimensions

Other works [Alipourfard et al. 2017, Hsu et al. 2018b] use VM's characteristics such as the number of vCPUs, memory size, among others, to create the dimensions of the search space. However, as mentioned before, there is evidence that BO has a poor performance in high-dimensional spaces. Thus, we decide to use a 2-dimensional space to represent our problem. In contrast to using parameters of the VM types as dimensions, we use the VM type itself and the size of the cluster (*i.e.*, the number of VM instances on the configuration). BO also tends to work better when the search space is smooth. Therefore, to achieve that, we sorted the VMs' dimensions by their cost and amount of memory.

In most cases, this significantly improves the smoothness of the space by putting inexpensive machines that do not have the necessary amount of memory on one side and machines that are excessively expensive on the other side. The most cost-effective solutions are usually located between both, in a valley. In some configurations, for many reasons (lack of memory, vCPU number different than power of two, *etc.*), some workloads do not work. When the search observes these configurations we consider the cost of the observations but we neither use the configuration in the solution nor use it to update the Bayesian model. Despite this has never been mentioned in the literature about cloud configuration search, we discovered from our experiments that without these transformations BO may perform similar to *random-search* with low or no advantage over it.

5.3. Experimental Methodology

Having our search space and our benchmarks defined, we ran each of 5 NPB's kernels with each of the three input classes (C, D, and E) in each of the 192 cloud cluster configurations aforementioned. For each one of the 2880 executions (15 workloads times 192 configurations), we collect the execution time of the first four paramount iterations.

The kernels' performance on each of the 192 configurations is then used to compute the configurations cost using Equation 3. Finally, once we have all cloud-cost-space calculated, we use the Python library GPyOpt [3] to simulate an online search in this search space.

As the searches are stochastic, we executed them 50 times and collected their geometric mean. We tested the following search approaches: *random-search* and *Bayesian optimization* (GP and RF) with MPI, EI, and UCB. We executed all searches with 32 observations (one per search iteration). All *Bayesian optimization* strategies have their first eight observations randomly picked to initiate the model.

The source code and all data collected from our experiments will be available in our lab repository[4].

6. Experimental Results

6.1. Validation of Paramount Iteration for NPB Kernels

Before executing the cloud experiments, we first used our local cluster to validate the Paramount Iteration (PI) concept, making sure that the first paramount iterations' performance could be used to estimate the performance of the whole execution on a given

[3]https://sheffieldml.github.io/GPyOpt/

[4]to be available upon publication

configuration. We used our modified version of the NPB with support to monitor the PIs to collect the first four iterations' execution times. We did this for every kernel using the input class E and varying the number of MPI processes from 1 to 24 in a local machine with an Intel Xeon processor. We also collected the total execution time of these kernels using the original code, without the PI. Figure 2 shows the speedup foreseen by the PIs and the real speedup of the kernels.

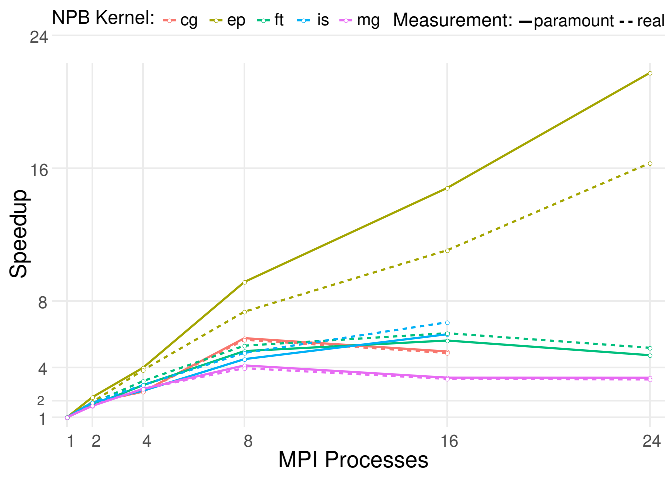


Figure 2. NPB Kernel's speedup measured with the average of 1 paramount iteration execution and with the real runtime for the whole execution. It shows less than 7.7% difference between the two measurements. IS and CG only executes with power-of-two threads.

Notice in the plot that the speedups estimated by the PIs follow the trend of the real ones. Moreover, the absolute value of the speedup is also close, less than 10% distant from the average. EP result highlights as being the one with a larger distance between both measurements. We discovered that this comes from the fact that EP only has communication in the initialization and the finalization, not in the PI. Thus, in the point of view of the PI measurement EP has an almost linear speedup, while in reality values have the communication overhead reducing its efficiency. However, despite that, both lines have the same behavior, thus PI could still be used to compare performance for EP in different configurations.

Thus, we have evidence that we can use the performance of the first PIs to compare the performance of different cloud configurations for the NPB kernels.

6.2. *Grid-search* in AWS's configurations

As discussed in the previous section, we ran the NPB kernels with three input classes for each one of the selected cloud configurations to create our search space. Figure 3 shows a heat map that indicates the cost of running each kernel with each input data set in each cloud configuration. It is possible to notice that when we increase the input data set size (C→E), increasing the demand for computation and memory, many configurations become unfeasible or too expensive (represented in black). Thus, the larger the input, the fewer configurations provide good cost-benefit, and the higher the average cost of the configurations. This difference can also be noticed when we go from one benchmark to another (*e.g.*, from MG-D to FT-D). Therefore, both the application and its input data set have a significant impact on the search space and, thus, implications on the search cost and results.

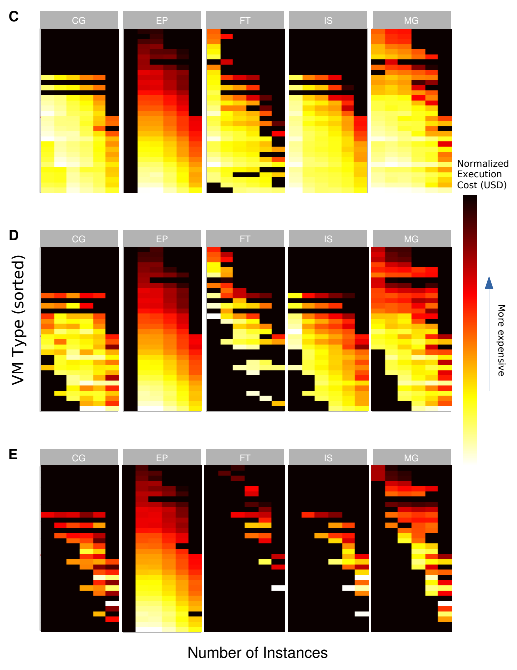


Figure 3. Real AWS cost space collected in our experiments for NPB kernels. The darker the color, the more expensive the configuration. AWS VM types and number of instances are sorted as in Figure 1.

6.3. SMBO Techniques Comparison: no one rules them all

We applied the SMBO techniques to search the space depicted in Figure 3 to try to find the less expensive configurations in each space. The results in Figure 4 show that no SMBO technique dominates in our experiments. However, although it is visible that *Bayesian optimization* approaches (BO) achieve better results than *random-search*, no SMBO model or acquisition function seems to be better for most cases. Moreover, the results indicate that the best technique may depend on the search space and, therefore, the program and input data set under evaluation. Nevertheless, when considering the average of all acquisition functions, RF has a slight advantage over GP. Also, apart from the CG/C case, RF does not perform worse than the Random Search. These conclusions about RF being better than GP as a model for this problem are also pointed out by Hsu *et al.* [Hsu et al. 2018b].

The Black dashed line in the plots from Figure 4 shows the best possible solution for each case. Notice that SMBO techniques can find solutions close to the space-best while only iterating 16% of the search space (32/192). In other words, all SMBO approaches tested find solutions that are, on average, less than 13% worse than the best solution that could be possibly found.

6.4. Search Cost

Figure 5-a shows the average normalized cost of the best configuration found by each search strategy after each observation. We initiated the prior model space for all SMBO

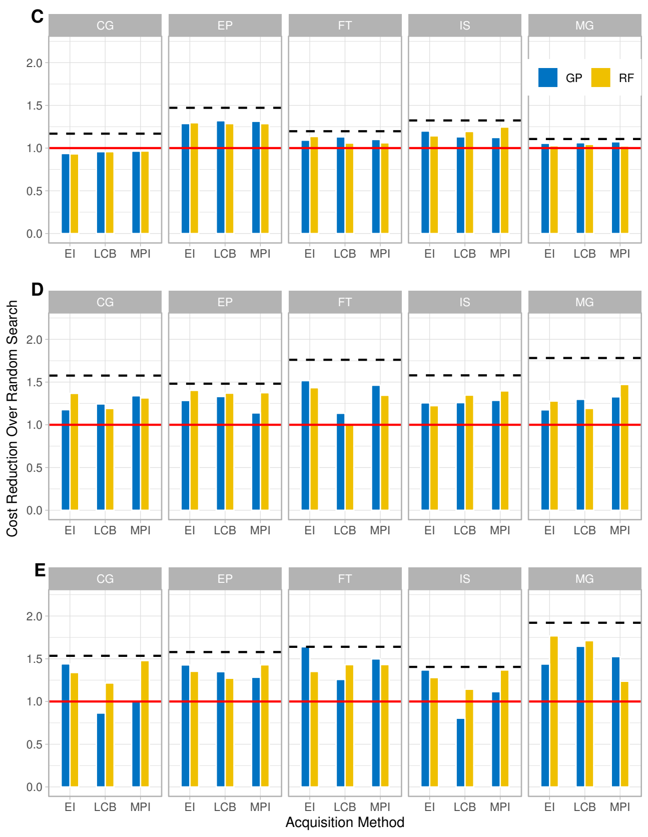


Figure 4. Average cost reduction achieved on 50 executions of each SMBO technique in the space from Figure 3. Higher is better. The cost is normalized by the cost achieved by the Random Search approach, which is depicted by a red-straight line. Black-dashed lines shows the cost of the best possible solution.

approaches with 8 random observations; hence, until the 8th observation, all SMBO approaches perform similar to the *random-search* approach. However, it is possible to see a performance improvement just after the 9th observation between SMBO and *random-search*. After the 15th observation, SMBO become significantly better than *random-search*. Moreover, after convergence, the best solution found by the SMBO techniques is $1.68\times$ cheaper than when using *random-search*.

Figure 5-b shows the average accumulated search cost after each observation for each search approach (the cost is normalized by the cheaper instance per kernel/class without PI). Notice that, after the 8^{th} observation, the accumulated cost for the SMBO approaches increases on a slower ratio. This happens because as SMBO learns it tends to explore points in the space with more confidence to be cheaper, as illustrated in Figure 1. Their search cost was 36% cheaper than *random-search* after 32 observations.

When we apply the *Paramount Iteration* (PI), we only need to execute four of the first paramount iterations of each application. This means a significant reduction in execution time (and cost) when estimating the configuration performance. On average, for the NAS kernels, we executed only 14% of the total execution time of the applications to predict the relative performance of each configuration (4.43% for CG, 17.33% for FT, 12% for MG, 40% for IS, and less than 1% for EP). The larger the total execution time

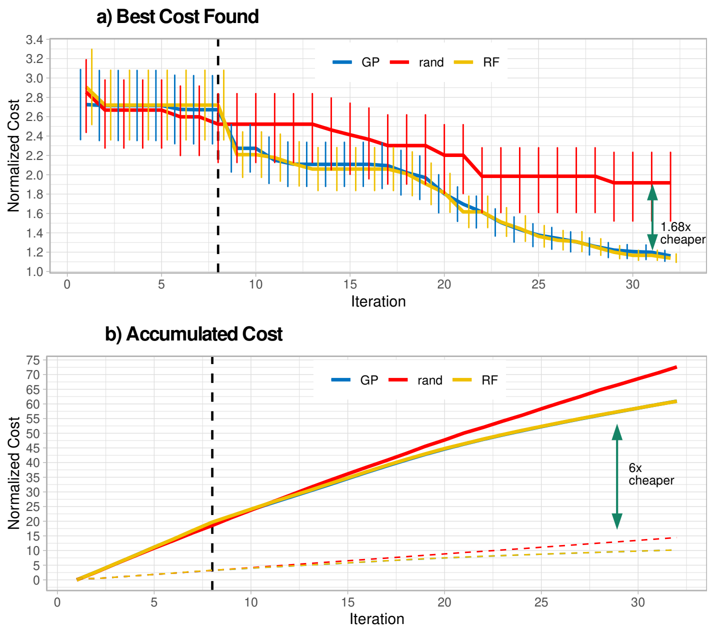


Figure 5. Average Best Cost and Average Accumulated Cost until each observation. Average values for all Kernels, Class and Acquisition. The dashed lines in Accumulated Cost represents the cost reduction when using PI.

and the total amount of PIs, the larger will be the difference between measuring only a first set of PIs and executing the whole program. In Figure 5-b, we have the accumulated search cost after each observation with PI represented by dashed lines. As indicated in the figure, by combining PI with SMBO, we can reduce search-cost by a factor of 6.

7. Conclusions

In this work, we propose a strategy that combines Sequential Model-Based Optimization (SMBO) techniques and the Paramount Iteration technique [Brunetta and Borin 2019] to search for efficient cloud configurations for HPC applications. We evaluate our approach using 192 different cloud configurations composed of AWS computing resources and 15 workloads defined by 5 kernels from the NPB benchmark and 3 different input datasets, and concluded that SMBO approaches are far more efficient than Random Search in finding the good configurations after 32 observations (1.68x better results). Further, we demonstrated that sorting the space for BO may be essential for its good performance. We also verified that different acquisition functions and models for SMBO have a small or no impact on its performance. Furthermore, we showed that the existing Paramount Iteration (PI) technique used to compare HPC workloads performance while early-stopping the application can be used to reduce search-cost without affecting its results. When using SMBO with PI, we achieved a 6-fold search-cost reduction.

References

[Alipourfard et al. 2017] Alipourfard, O., Liu, H. H., Chen, J., Venkataraman, S., Yu, M., and Zhang, M. (2017). Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In *14th USENIX NSDI 17)*, pages 469–482.

[Bailey 2011] Bailey, D. H. (2011). *NAS Parallel Benchmarks*, pages 1254–1259. Springer US, Boston, MA.

[Brunetta and Borin 2019] Brunetta, J. R. and Borin, E. (2019). Selecting efficient cloud resources for hpc workloads. In *12th IEEE/ACM ICUCC*, UCC'19, page 155–164, New York, NY, USA. Association for Computing Machinery.

[Ferguson et al. 2012] Ferguson, A. D., Bodik, P., Kandula, S., Boutin, E., and Fonseca, R. (2012). Jockey: Guaranteed job latency in data parallel clusters. In *7th ACM ECCS*, EuroSys '12, page 99–112, New York, NY, USA. Association for Computing Machinery.

[Herodotou et al. 2011] Herodotou, H., Dong, F., and Babu, S. (2011). No one (cluster) size fits all: Automatic cluster sizing for data-intensive analytics. In *2nd ACM SCC*, SOCC '11, New York, NY, USA. ACM.

[Hsu et al. 2018a] Hsu, C., Nair, V., Menzies, T., and Freeh, V. W. (2018a). Scout: An experienced guide to find the best cloud configuration. *CoRR*, abs/1803.01296.

[Hsu et al. 2018b] Hsu, C.-J., Nair, V., Freeh, V. W., and Menzies, T. (2018b). Arrow: Low-level augmented bayesian optimization for finding the best cloud vm. In *2018 IEEE 38th ICDCS*, pages 660–670. IEEE.

[Hsu et al. 2018c] Hsu, C.-J., Nair, V., Menzies, T., and Freeh, V. (2018c). Micky: A cheaper alternative for selecting cloud instances. In *2018 IEEE 11th CLOUD*, pages 409–416. IEEE.

[Kushner 1964] Kushner, H. J. (1964). A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 86(1):97–106.

[Li et al. 2010] Li, A., Yang, X., Kandula, S., and Zhang, M. (2010). Cloudcmp: comparing public cloud providers. In *10th ACM SIGCOMM*, pages 1–14.

[Močkus 1975] Močkus, J. (1975). On bayesian methods for seeking the extremum. In *Optimization techniques IFIP technical conference*, pages 400–404. Springer.

[Srinivas et al. 2009] Srinivas, N., Krause, A., Kakade, S. M., and Seeger, M. (2009). Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*.

[Wu et al. 2019] Wu, C., Summer, T., Li, Z., Woodard, A., Chard, R., Baughman, M., Babuji, Y., Chard, K., Pitt, J., and Foster, I. (2019). Paraopt: Automated application parameterization and optimization for the cloud. In *2019 IEEE CloudCom*, pages 255–262. IEEE.

[Yadwadkar et al. 2017] Yadwadkar, N. J., Hariharan, B., Gonzalez, J. E., Smith, B., and Katz, R. H. (2017). Selecting the best vm across multiple public clouds: A data-driven performance modeling approach. In *2017 SCC*, pages 452–465.