# Bayesian optimization for cloud resource management through machine learning

Bruno Guindani, Danilo Ardagna and Alessandra Guglielmi

**Abstract** Optimal cloud configuration of recurring big data analytic jobs is a relevant and challenging task in the industry. To this end, Bayesian Optimization is a promising method for efficiently finding optimal or near-optimal configurations for such applications, which are often executed in the cloud. On the other hand, Machine Learning methods can provide useful knowledge about the application at hand thanks to the quality of their estimations. In this paper, we propose a hybrid algorithm that is based on Bayesian Optimization and integrates elements from Machine Learning techniques to tackle time-constrained optimization problems in a cloud computing setting. We consider a recurring job scenario, where unfeasible points are to be avoided by all means, as they are a waste of resources. In such a context, Machine Learning helps to convey valuable information about the violation of constraints. Experiments on big data applications have shown that our algorithm significantly reduces the amount of unfeasible executions with respect to a pure constrained BO approach.

**Key words:** acquisition function, cloud computing, Gaussian Process

## 1 Introduction

Big data analytics are employed in several industrial fields to allow organizations and companies to make better decisions. The most suitable execution environment of big data analytic applications is a cluster of virtual machines (VMs) which allows the adjustment of the allocated resources (CPU, memory, disk, network) to match the application current needs. Choosing the right cloud configuration to minimize

Bruno Guindani, e-mail: `bruno.guindani@polimi.it` · Danilo Ardagna, e-mail: `danilo.ardagna@polimi.it` · Alessandra Guglielmi, e-mail: `alessandra.guglielmi@polimi.it`
Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133 Milano (Italy)

execution times and reduce costs is essential to service quality and business competitiveness. This is especially crucial for recurring cloud jobs, i.e., applications that need to be executed multiple times, which constitute a significant part of the total amount of analytic jobs running in the cloud [2, 20]. However, variability in the progress and resource requirements of analytic jobs imply that choosing the best configuration for a broad spectrum of applications is a challenging process [2].

Bayesian Optimization (BO) has recently gained notoriety as a powerful tool to solve global optimization problems in which expensive black-box functions are involved; see the recent paper [15] or the popular tutorial paper [5]. BO is a sequential design strategy that requires few steps to get sufficiently close to the true optimum, while requiring no derivative information on the optimized function. Most commonly, it is initialized by choosing and evaluating a small handful of starting points, then fitting a Gaussian process (GP) on these points. This approach can be interpreted as assuming a prior distribution for the unknown infinite-dimensional parameter $f$, i.e., the function to be optimized. The posterior distribution of the fitted GP provides an estimate of both the function value at each point and the uncertainty around the estimate. BO then iteratively chooses new points at which to evaluate the function in a such a way to balance exploration (high uncertainty) and exploitation (best estimated function value); see, for instance, [11]. The topic of constrained BO has also received attention in the literature [8, 11].

The goal of this work is to integrate Bayesian Optimization algorithms with Machine Learning (ML) techniques in the context of cloud computing optimization for recurring jobs. The former techniques have proven to be successful [2, 16] in exploring and finding optimal or near-optimal cloud configurations after a small amount of exploratory runs. On the other hand, the latter can provide useful information to be incorporated into the BO mechanism in several ways to improve its performance, for instance in the form of cheap estimates of target quantities to guide the exploration process.

This work builds on previous results found in [2], in which the *CherryPick* system has been successfully applied to benchmark applications on cloud computing frameworks such as Apache Spark. This system exploits pure constrained BO to find optimal cloud configurations. Our work is motivated by the belief that ML models can lend their estimation capabilities to BO to further improve its effectiveness. This topic has been explored in [13], which examines the performance of several ML models in carrying out prediction of execution times of Spark cloud jobs with different types of workloads. The hybrid BO algorithm we propose here is promising since it shows the usefulness of ML in the context of cloud computing configuration.

The setup of this paper is as follows. Sect. 2 describes the mathematical formulation of the problem, Sect. 3 presents our proposal of a BO algorithm, while Sect. 4 collects a few experimental results. We conclude the paper with a short discussion in Sect. 5.

## 2 Background and mathematical formulation

The goal of Bayesian Optimization is to estimate, using an iterative method, the minimum of a given function $f$, called objective function, by using as few iterations as possible. We focus here on the minimum of the function $f$, though we could consider the maximum with similar arguments. Specifically, we want to find $\widehat{x}$ such that

$$\widehat{x} = \arg\min_{x\in\mathcal{A}} f(x).$$

Strong assumptions on $f$ or on the minimization domain $\mathcal{A}$ are not required, and BO algorithms are derivative-free, i.e., they do not require any knowledge about the derivatives of $f$. For these reasons, BO is often used to optimize expensive black-box objective functions (see [3]), that is, functions for which little to no information is available and whose evaluation has significant time, resource, and/or monetary costs.

We consider the mathematical formulation for our *constrained global optimization* problem similarly to [2]. Let $x \in \mathcal{A}$ denote the $d$-dimensional vector representing a configuration for the cloud job, including information such as the number of cores used for the job, with $\mathcal{A} \subset \mathbb{R}^d$ being the domain of all feasible configurations. The *objective function* to be minimized is the total cost $f(x) = P(x)T(x)$, where $T(x)$ is the unknown execution time and $P(x)$ is the price per unit (it is a known, deterministic function). We also assume the constraint that $T(x) \leq T_{max}$, where $T_{max}$ is a given threshold. By making this model explicit in $f$, we obtain:

$$\begin{aligned} &\min_{x\in\mathcal{A}} f(x) = P(x)T(x) \\ &\text{s.t. } \; f(x) \leq P(x)\, T_{max}. \end{aligned} \tag{1}$$

In this paper, we assume the deterministic price function $P(x)$ as being proportional to the number of virtual machines or cores used by the application job, which is always included in the cloud configuration vector $x$. Other choices of the price function are possible.

The key idea of BO comes from the Bayesian approach to statistics, in which values taken by $f$ are treated as random variables, and a *prior distribution* represents the a-priori information on the modeled phenomenon – in the case of BO, information on the location of the minimum. The prior distribution is then iteratively updated with information coming from the observed data, obtaining the *posterior distribution*. For the rest of the paper, we assume that observed data, i.e., the evaluations of $f$, are noise-free. This is justified by the analysis in [13] on the data considered for validation. In a more general scenario, data can be assumed to have independent, normally distributed additive noise with variance $\eta^2$. In this context, the parameter $\eta^2$ is usually estimated in a preliminary step.

In the BO framework, the Gaussian process (GP) is the preferred choice for the prior for $f$. This implies that for any $x \in \mathcal{A}$,

$$f(x) \sim \pi_x(\cdot) = \mathcal{N}(\mu_0(x), \sigma_0^2(x,x)).$$

The functions $\mu_0(\cdot)$ and $\sigma_0^2(\cdot,\cdot)$ are called mean and kernel functions, respectively, and are the GP model hyperparameters. These functions serve as the "initial guesses" on values of $f(\cdot)$ and on their uncertainty, and they will be iteratively updated with observed values. A constant mean function $\mu_0(\cdot) \equiv \mu_0$ is often adopted, whereas the choice of the kernel is more delicate, since it influences the smoothness of the process. Commonly used kernels include the squared exponential or Radial Basis Function and the Matérn kernel [19]. The former gives the GP an excessively large degree of smoothness, which is unrealistic in many practical scenarios. Therefore, in this work, we assume $\mu_0(\cdot) \equiv \mu_0$ and we use the Matérn kernel with smoothness parameter $\nu = 5/2$ (see [5]):

$$\sigma_0^2(x,x') := \frac{1}{2^{3/2}\Gamma(5/2)} \left(\sqrt{5}\|x-x'\|\right)^{5/2} K_{5/2}\left(\sqrt{5}\|x-x'\|\right). \tag{2}$$

In Eq. (2), $\|\cdot\|$ denotes the Euclidean norm, while $K$ is the modified Bessel function of the second type [1]. As usual, $\Gamma(\cdot)$ is the gamma function.

Having observed values $H_n = \{(x_1, f(x_1)), \ldots, (x_n, f(x_n))\}$ of the objective function, one computes the posterior distribution of $f(x)$, for any $x$. This distribution is Gaussian as well, with posterior mean $\mu_n(\cdot)$ and variance $\sigma_n^2(\cdot)$, i.e.

$$f(x)|H_n \sim \pi_x(\cdot|H_n) = \mathcal{N}(\mu_n(x), \sigma_n^2(x)),$$

and is computed by well-known properties of GPs (see [5]) as

$$\mu_n(x) = \mu_0(x) + \sigma_0^2(x, x_{1:n})^T\, \sigma_0^2(x_{1:n}, x_{1:n})^{-1}\big(f(x_{1:n}) - \mu_0(x_{1:n})\big), \tag{3}$$
$$\sigma_n^2(x) = \sigma_0^2(x,x) - \sigma_0^2(x, x_{1:n})^T\, \sigma_0^2(x_{1:n}, x_{1:n})^{-1}\sigma_0^2(x, x_{1:n}). \tag{4}$$

In Eqq. (3) and (4), $\sigma_0^2(x, x_{1:n})$ indicates the column vector of values of the $\sigma_0^2(\cdot,\cdot)$ function applied to pairs $(x, x_1), \ldots, (x, x_n)$, and similarly for $f(x_{1:n})$ and $\mu_0(x_{1:n})$. Analogously, $\sigma_0^2(x_{1:n}, x_{1:n})$ is the matrix of values of $\sigma_0^2(x_i, x_j)$ with $i, j = 1, \ldots, n$.

Bayesian Optimization is an iterative algorithm that obtains a new observation at each iteration by solving a proxy problem – the maximization of the *acquisition function* $g(x)$, which depends on the fitted GP model and measures the utility of evaluating the objective function at a given configuration $x$. This function is optimized at each round of the iterative algorithm, instead of directly optimizing the objective function itself, since it is available in closed form and inexpensive to evaluate. The acquisition function must strike a delicate balance – the exploration-exploitation trade-off. On the one hand, there are points to which large uncertainty is attached, for instance because they lie in a region of the domain which has not been explored yet. Choosing such points to evaluate the objective $f(\cdot)$ is appealing, especially early on in the optimization procedure, since this would allow a large decrease of the uncertainty on the position of the optimum. On the other hand, the algorithm does seek to find the optimum of the objective function, therefore it should also choose to evaluate points which most likely (according to the GP model) give small values of $f(\cdot)$. This is done by exploiting the information already available on the location of the optimum, especially in the late iterations of the algorithm. Convergence of the BO algorithm is guaranteed under mild conditions [3, 14].

The BO procedure is summarized in Algorithm 1. In step 1, a small number $n_0$

**Algorithm 1** Generic Bayesian Optimization algorithm

```
1: choose $n_0$ initial points
2: evaluate $f(\cdot)$ in the initial points, add evaluations to history $H$
3: for iterations $n = 1 : N$ do
4:     update the current posterior distribution of the GP model with data in $H$
5:     find point $x_{n+1}$ which maximizes the acquisition function $g(\cdot)$ under the current model
6:     evaluate $f(x_{n+1})$, add performed evaluation to $H$
7: end for
8: return estimated optimum $\widehat{x}$
```

of initial points (i.e., cloud configurations in our specific application) are selected, usually 3 to 10, in order to initialize the algorithm. These points should be chosen so as to cover the maximum domain area possible, for instance using a Latin hypercube design [10]. We then evaluate these points (in practical terms, this means executing the application using these configurations), and we record the points and their evaluations (step 2). After the initialization phase, we enter the algorithm loop, where we update the posterior distribution (step 4), choose the next point $x_{n+1}$ by maximizing the acquisition function (step 5), and evaluate it (step 6). The algorithm stops when the iteration budget $N$ runs out.

Fig. 1 shows how BO works. Specifically, in the top panel, the objective function

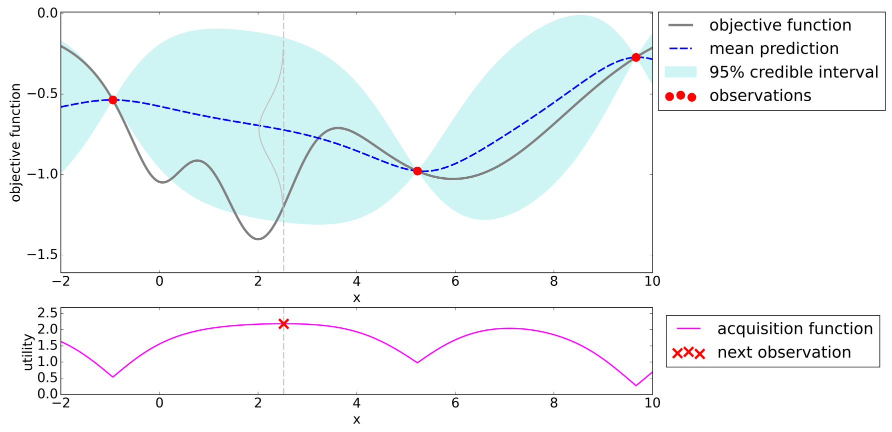

**Fig. 1** Bayesian Optimization after 3 iterations. Top panel: the objective function and its Bayesian estimate. Bottom panel: acquisition function.

to be minimized (the solid line) is estimated by the posterior mean function (the dashed line) and 95% credible interval (the highlighted area in the background) of the associated Gaussian process, after evaluating 3 points (the large dots). Both estimates are updated whenever a new observation is received, therefore getting more and more accurate over time. The vertical bell curve represents the Gaussian distribution which is assumed for each point $x$ of the optimization domain; we recall that both the mean and variance of this distribution depend on $x$. Instead, the bottom panel shows the acquisition function given the current posterior distribution. Its values are lower in points which have already been sampled, because the utility of repeating an observation is generally smaller than evaluating a brand new point. The crossed point indicates the maximum of the acquisition function, i.e., the next point which will be evaluated. Note that such point is at the center of a region which has both large expected value and large variance, making it appealing in the exploration-exploitation trade-off.

In this paper, we compare different acquisition functions. All of them are based on two popular acquisition functions, which have been shown to be effective in the literature [6, 7, 18]. The *Expected Improvement* (EI) over the best value $f_n^*$ found by the optimization process so far is defined as

$$EI_n(x) \coloneqq \mathbb{E}_{\pi_x(\cdot|H_n)}[\max(f_n^* - f(x), 0)] \quad \text{with} \;\; f_n^* = \min_{i \le n} f(x_i).$$

The expectation is taken under the current posterior distribution $\pi(\,\cdot\,|H_n)$ of $f(x)$, given history $H_n$. We consider a generalization of EI to the constrained optimization setting – the *Expected Improvement with Constraints* (EIC) acquisition function [10, 17], which accounts for the probability of a point of satisfying the constraints (see Eq. (1)):

$$EIC_n(x) \coloneqq EI_n(x) \cdot \mathbb{P}_{\pi_x(\cdot|H_n)}\big(f(x) \le P(x)\, T_{max}\big). \tag{5}$$

Note that the second factor in Eq. (5), which is equal to the posterior probability of respecting the constraint, is used as a correction factor for the basic Expected Improvement acquisition function.

## 3 Our hybrid algorithm

As mentioned in the Introduction, ML estimation techniques can provide useful information to be incorporated into the Bayesian Optimization mechanism to improve its performance, for instance in the form of cheap estimates of target quantities to guide the exploration process.

We first summarize the complete procedure in Algorithm 2. Our algorithm is based on pure BO, but it integrates elements coming from ML techniques. We use a first-in-first-out memory queue for discrete features to prevent exploration of already visited values, similarly to the taboo search meta-heuristic methods [4]. In this memory queue, we save the last $q$ points visited by the algorithm. Configurations currently in the queue are excluded from being selected again until they have shifted

**Algorithm 2** Proposed algorithm

```
 1: choose $n_0$ initial points
 2: evaluate $f(\cdot)$ in the initial points, add evaluations to history $H$
 3: for iterations $n = 1 : N$ do
 4:     update the current posterior distribution of the GP model with data in $H$
 5:     train model $\widehat{T}(\cdot)$ with data in $H$, to be used in $g(\cdot)$
 6:     find point $x_{n+1}$ which maximizes the acquisition function $g(\cdot)$ under the current model
 7:     evaluate $f(x_{n+1})$, add performed evaluation to $H$
 8:     update memory queue with $x_{n+1}$
 9:     if stopping criteria are met then
10:         terminate the algorithm
11:     end if
12: end for
13: return estimated optimum $\widehat{x}$
```

out of the queue, i.e., after $q$ iterations. At each round, we train an ML model $\widehat{T}(\cdot)$ (step 5) with all points evaluated so far by the algorithm. Then, similarly to regular BO, we maximize the acquisition function of choice, which now incorporates the trained ML model. Note that we have used ML language (i.e., *we train an ML model* $\widehat{T}(\cdot)$) here, but these *ML models* consist of statistical estimation methods which have become very popular in the ML community. After that, we evaluate the newly chosen configuration (step 7) as usual, and we update the aforementioned memory queue (step 8). The algorithm continues until the evaluated execution time at the current iteration is sufficiently close to the time threshold: $T(x_n) \in [\alpha T_{max}, T_{max}]$, with $\alpha \in (0, 1)$ (step 10). This is because our goal is to obtain a configuration that is compliant with the time threshold, but also uses as few resources as possible. Generally speaking, using more resources results in a lower execution time – meaning that a time which is just under the threshold likely consumes the least amount of resources for that configuration to be feasible. After termination of this algorithm, it is likely that we have found the true optimal configuration. Afterwards, we perform subsequent executions using such optimal or near-optimal configuration (recall that we are dealing with recurring jobs, whose periodic execution does not stop with the termination of the optimization procedure).

In the application of interest, our goal is ultimately to find optimal (or near-optimal) configurations which are also feasible, i.e., points $x$ s.t. $T(x) \leq T_{max}$. Unfeasible points are to be avoided by all means, since they represent a waste of resources in a recurring job setting, providing additional unnecessary costs. Integrating ML models into the acquisition function is crucial in assessing the feasibility of the points under consideration. Hence, ML methods can prove to be useful additions to the acquisition function in a setting where evaluating the objective function $f(\cdot)$ is expensive. Indeed, a cheap estimation of values of $f(\cdot)$ can compensate for the scarcity of direct information on them. In our case, ML models can be used to convey valuable information about the violation of constraints. Specifically, we use ML models to compute $\widehat{T}(\cdot)$ and use this estimate to correct the acquisition function $EIC(\cdot)$.

We therefore propose the following acquisition functions:

- $g_A(x) = EIC(x)$: the original Expected Improvement with Constraints acquisition function [10, 17], used by *CherryPick* [2], which we use as baseline;
- $g_B(x) = g_A(x) \cdot \exp(-k\,\widehat{T}(x))$, the latter term being a $[0, 1]$-valued weight for $g_A$. This correction factor is called nascent minima distribution function [12], and it serves the purpose of turning $\widehat{T}(x)$ into an acquisition-like function. More precisely, the correction factor takes on values close to 1 if the estimated execution time $\widehat{T}(x)$ is small, thus making $x$ a desirable point, while it is closer to 0 if such estimate is large;
- $g_C(x) = g_A(x) \cdot I_{\{\widehat{T}(x) \leq T_{max}\}}$, with $I$ being the indicator function: the search is prevented in areas where the current estimated execution time violates the threshold $T_{max}$. Here we are using the model $\widehat{T}(x)$ to estimate the current feasible domain;
- $g_D(x) = g_A(x) \cdot \exp(-k\,\widehat{T}(x)) \cdot I_{\{\widehat{T}(x) \leq T_{max}\}}$: the combination of cases B and C.

Note that the original definition of the nascent minima distribution functions (used in variants B and D) includes the normalization constant $1/C_k$ as a multiplicative factor, with $C_k = \int_{\mathcal{A}} \exp(-k\,\widehat{T}(w))\,\mathrm{d}w > 0$, as described in [12]. However, this value is independent of $x$, therefore we can omit it when maximizing the acquisition function in $x$.

Note that at each step of the algorithm, we *train* the ML regression model $\widehat{T}(\cdot)$, e.g., we fit the model with the data we have collected so far, obtaining a model which can estimate $T(x)$ for any $x \in \mathcal{A}$.

## 4 Experiments

We present experimental results using the techniques we have discussed in Sect. 3. We test variants B, C, and D of the algorithm, as well as pure constrained BO (represented by variant A), on two different big data applications, run with the Apache Spark analytics engine. The first is Query26 application from the TPC-DS industry benchmark, which we execute with input data size equal to 250 GB and time threshold equal to 150 seconds (s). The second is an application performing K-means clustering algorithm on a dataset with 15 million rows, under a time threshold equal to 330 s. In both cases, we optimize the total cost in Eq. (1) on the number of cores $x$, i.e., the minimization domain is uni-dimensional. The same three fixed initial points were used for all variants ($n_0 = 3$).

Fig. 2 shows the comparison of pure constrained BO (top row) with our three variants (other rows) at each algorithm iteration, when applied to Query26. In the left panel, we represent the number of cores chosen by the algorithm at each iteration. The solid horizontal line is the true optimum of the constrained optimization problem. The vertical dashed line indicates the execution at which the stopping criterion kicks in, and after which the recurring application at hand sticks to the best configuration found by the algorithm. In the center panel, each bar represents a single execution. The width of a bar represents the execution time of the job, while its height represents

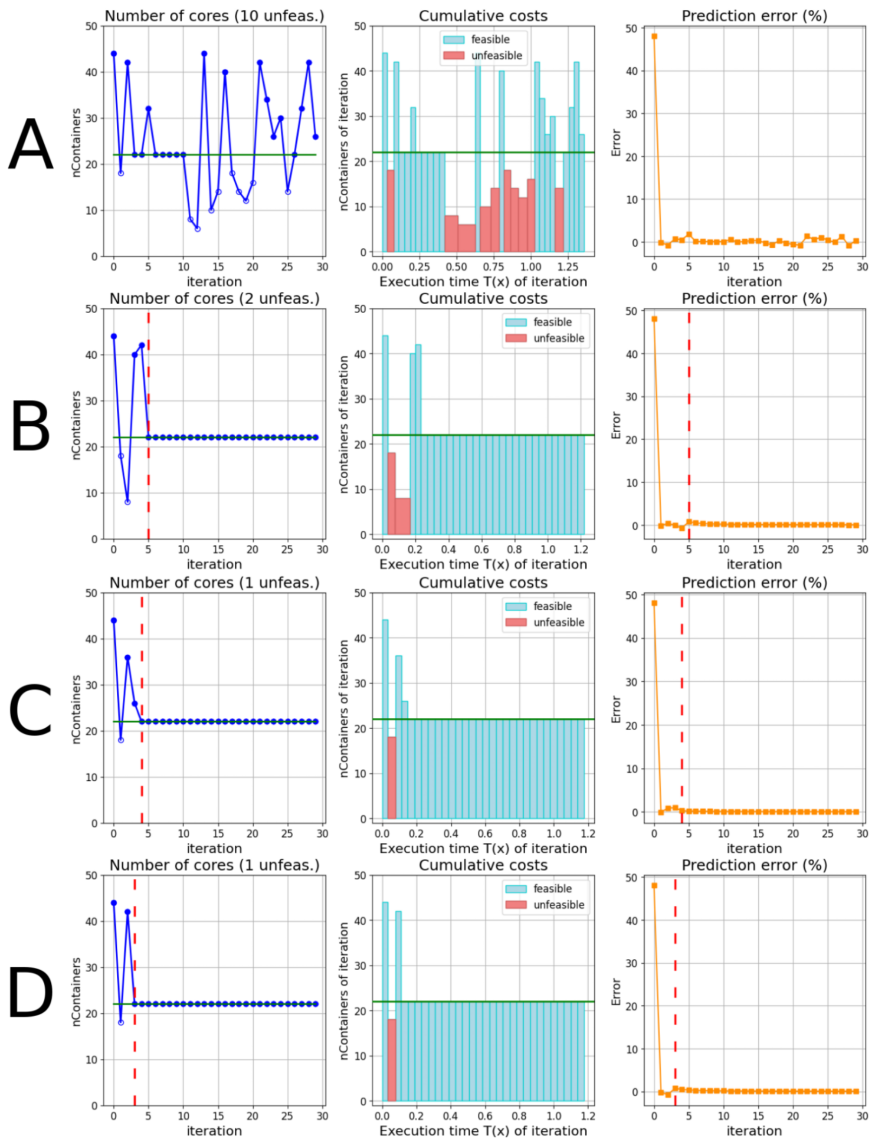

**Fig. 2** Comparison of pure constrained BO (top row) with variants B, C, and D at each algorithm iteration for Query26. Left panel: number of cores $x$ visited by the algorithm; center panel: cost of each configuration; right panel: percentage error of the ML model.

the number of cores used. Therefore, the area of a bar is proportional to the cost $f(x)$ for that particular execution (see Eq. (1)). We highlight the bars corresponding to feasible and unfeasible configurations in different shades. The signed percentage errors of the ML model for the execution time are displayed in the right panel. In

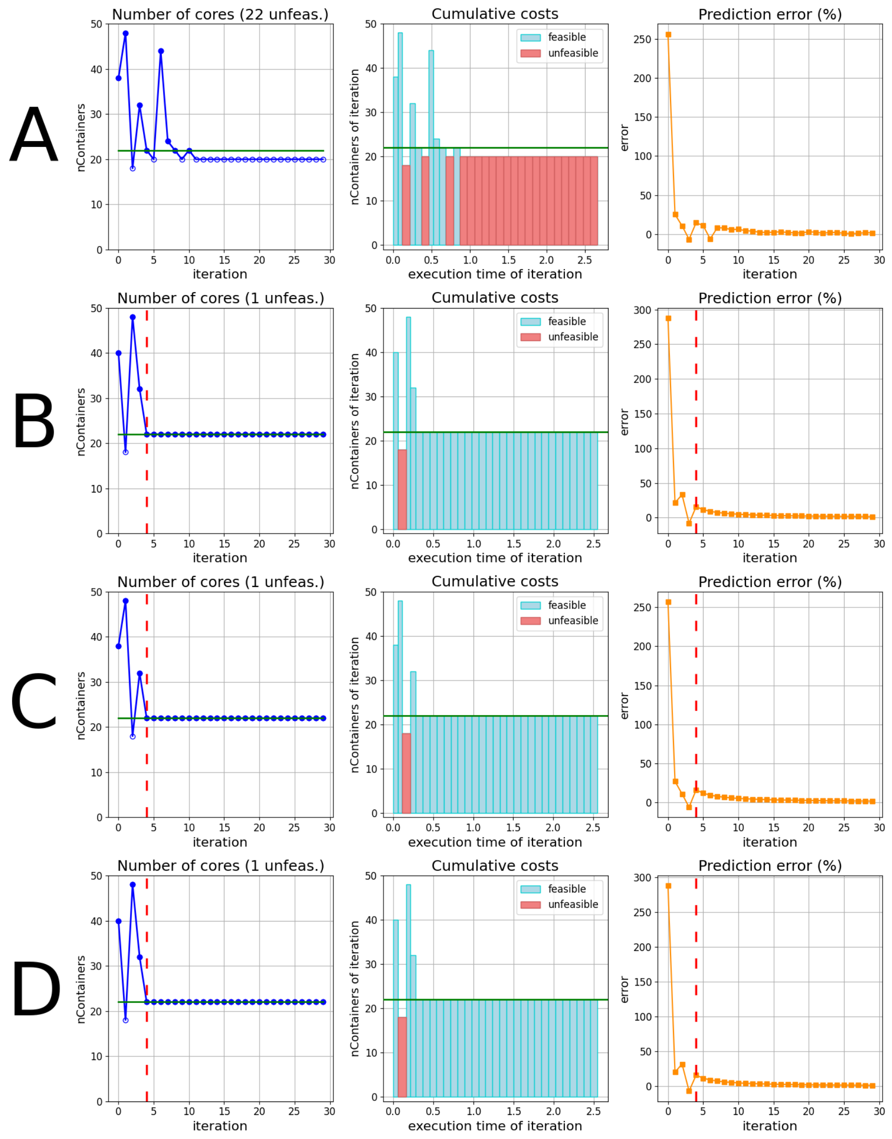

**Fig. 3** Comparison of pure constrained BO (top row) with variants B, C, and D at each algorithm iteration for K-means. Left panel: number of cores $x$ visited by the algorithm; center panel: cost of each configuration; right panel: percentage error of the ML model.

this case, we have used the Ridge regression method to estimate the execution time $T(x)$, since [13] show that it is the most accurate regression method in this context, with a mean absolute error smaller than 5%. In particular, $\widehat{T}(x) = x^\top \widehat{\beta}$, where $\widehat{\beta}$ is the estimate of the regression parameters under the Ridge regression method.

At each iteration, the model is trained with all data from previous iterations, and the error is evaluated on the new configuration chosen by the algorithm. We see in the right panel of Fig. 2 that after one initial iteration with a large error, our ML models quickly converge to errors very close to zero. This initial spike in the error can be explained by the large distance between the $n_0$ initialized points and the first point selected by the algorithm. Indeed, in this particular experiment, the initial points have a small number of cores. Therefore the algorithm, which is still in the exploration phase, selects a configuration with a large number of cores (as seen in the leftmost spike of the left panel), because it lies in a region of the domain that is still unexplored. For this reason, the ML model struggles to perform accurate estimations in the first iteration. After that, the ML model has accumulated enough information and consequently it is able to produce more accurate estimates, as shown in the following iterations in the right panel of Fig. 2. Finally, the termination criterion (see the vertical dotted line) correctly assesses the optimality of the configuration, and stops the exploration phase.

We show a similar plot for the K-means application in Fig. 3, for which we used the same ML model as Query26. Furthermore, Table 1 summarizes a few noteworthy average metrics from multiple runs of the algorithm, for both Query26 and K-means. It is clear that each of our algorithm variants (B-D) outperforms the constrained vanilla BO technique in [2] (variant A) with respect to multiple metrics. First of all, in Figg. 2 and 3, we can see that the number of "wasted" unfeasible executions, i.e., ones whose execution time is over the threshold $T_{max}$, decreases from 10-22 to just 1 or 2. In variants B-D, the ML model guides the exploration process towards feasible points, by excluding those that are likely to be unfeasible. These variants reach termination before the 6th iteration, whereas variant A does even struggle to reach convergence. On average (see Table 1), the total amount of unfeasible executions drops by two to three times. Furthermore, we measure the ratio of the total cost of unfeasible executions over the total cost of all executions (i.e., over the entire iteration budget), displayed as "ratio of unfeasible costs" in Table 1. With respect to variant A, this ratio also decreases by two to three times. Finally, the average cost of the feasible configuration (which we normalize over the cost of variant A in Table 1 for the sake of clarity) decreases by about 13% in Query26, while always remaining competitive with variant A in the K-means case.

**Table 1** Measured metrics for variants A-D of the algorithm

| scenario | var. A | **var. B** | **var. C** | **var. D** |
|---|---|---|---|---|
| Query26 | | | | |
| unfeasible executions | 11.23 | 4.34 | 4.11 | 4.31 |
| ratio of unfeasible costs | 0.36 | 0.14 | 0.13 | 0.14 |
| norm. mean feasible cost | 1.00 | 0.87 | 0.87 | 0.88 |
| K-means | | | | |
| unfeasible executions | 10.52 | 5.74 | 5.93 | 5.96 |
| ratio of unfeasible costs | 0.34 | 0.20 | 0.21 | 0.21 |
| norm. mean feasible cost | 1.00 | 1.00 | 1.01 | 1.00 |

## 5 Discussion

In this paper, we have presented a novel BO algorithm integrated with ML techniques, to find the optimal configuration for a recurring job running in the cloud, under a given time threshold. ML models help in the crucial task of recognizing configurations which lead to unfeasible executions, saving on unnecessary additional costs. Indeed, experiments on big data applications show that our algorithm significantly reduces the amount of unfeasible executions with respect to a pure constrained BO approach, as well as reducing the average cost of the configuration. Overall, each of our algorithm variants outperforms the state-of-the-art BO technique used as benchmark.

Convergence guarantees for BO shown in [3, 14] may no longer hold when introducing ML models in the expression of the acquisition function. However, we stress that in the context of cloud computing optimization, the reduction of exploration iterations is prioritized over the quality of the solution, which can also be near-optimal rather than optimal. Therefore, the issue of convergence is not relevant in this context. In such a setting, if the given iteration budget were limited enough, a traditional BO algorithm would not be able to reach convergence in the proper sense either. Besides this remark, note that our algorithm does have a sensible termination criterion, which assesses whether the current configuration is likely to be near-optimal, therefore taking the role of the former convergence guarantees.

A preliminary version of this manuscript has appeared as [9].

**Acknowledgements** The European Commission has partially funded this work under the Horizon 2020 Grant Agreement number 956137 LIGATE: LIgand Generator and portable drug discovery platform AT Exascale, as part of the EuroHPC Joint Undertaking.

## References

1. Abramowitz, M., Stegun, I.A.: Handbook of mathematical functions with formulas, graphs, and mathematical tables, vol. 55. US Government printing office (1964)
2. Alipourfard, O., Liu, H., Chen, J., Venkataraman, S., Yu, M., Zhang, M.: Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In: USENIX Symposium on Networked Systems Design and Implementation (2017)
3. Brochu, E., Cora, V.M., De Freitas, N.: A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. arXiv preprint arXiv:1012.2599 (2010)
4. Cvijović, D., Klinowski, J.: Taboo search: An approach to the multiple-minima problem for continuous functions. In: Handbook of global optimization, pp. 387–406. Springer (2002)
5. Frazier, P.I.: A tutorial on Bayesian optimization. arXiv preprint arXiv:1807.02811 (2018)
6. Gardner, J.R., Kusner, M.J., Xu, Z.E., Weinberger, K.Q., Cunningham, J.P.: Bayesian optimization with inequality constraints. In: Proceedings of the 31st International Conference on International Conference on Machine Learning, vol. 32, pp. 937–945 (2014)
7. Gelbart, M.A., Snoek, J., Adams, R.P.: Bayesian optimization with unknown constraints. In: Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence, pp. 250–259 (2014)

8. Gramacy, R.B., Lee, H.K.H.: Optimization under unknown constraints. In: Bayesian Statistics 9. Oxford University Press (2011)
9. Guindani, B., Ardagna, D., Guglielmi, A.: Bayesian optimization with machine learning for big data applications in the cloud. In: SIS2022 - Book of Short Papers, pp. 1479–1484. Pearson (2022)
10. Jones, D.R., Schonlau, M., Welch, W.J.: Efficient global optimization of expensive black-box functions. Journal of Global optimization **13**(4), 455–492 (1998)
11. Letham, B., Karrer, B., Ottoni, G., Bakshy, E.: Constrained Bayesian optimization with noisy experiments. Bayesian Analysis **14**, 495–519 (2019)
12. Luo, X.: Minima distribution for global optimization. arXiv preprint arXiv:1812.03457 (2018)
13. Maros, A., Murai, F., da Silva, A.P.C., Almeida, J.M., Lattuada, M., Gianniti, E., Hosseini, M., Ardagna, D.: Machine learning for performance prediction of spark cloud applications. In: IEEE International Conference on Cloud Computing, pp. 99–106 (2019)
14. Mockus, J.: Application of bayesian approach to numerical methods of global and stochastic optimization. Journal of Global Optimization **4**(4), 347–365 (1994)
15. Pourmohamad, T., Lee, H.K.: Bayesian optimization via barrier functions. Journal of Computational and Graphical Statistics **31**(1), 74–83 (2022)
16. Reagen, B., Hernández-Lobato, J.M., Adolf, R., Gelbart, M., Whatmough, P., Wei, G.Y., Brooks, D.: A case for efficient accelerator design space exploration via Bayesian optimization. In: IEEE/ACM International Symposium on Low Power Electronics and Design, pp. 1–6. IEEE (2017)
17. Schonlau, M., Welch, W.J., Jones, D.R.: Global versus local search in constrained optimization of computer models. IMS Lecture Notes-Monograph Series pp. 11–25 (1998)
18. Shahriari, B., Swersky, K., Wang, Z., Adams, R.P., De Freitas, N.: Taking the human out of the loop: A review of bayesian optimization. Proceedings of the IEEE **104**(1), 148–175 (2015)
19. Williams, C.K., Rasmussen, C.E.: Gaussian processes for machine learning, vol. 2. MIT press Cambridge, MA (2006)
20. Zhang, Y., Prekas, G., Fumarola, G.M., Fontoura, M., Goiri, I., Bianchini, R.: History-based harvesting of spare cycles and storage in large-scale datacenters. In: USENIX Symposium on Operating Systems Design and Implementation (2016)