Bayesian Optimization with Machine Learning for Big Data Applications in the Cloud

Ottimizzazione Bayesiana Integrata col Machine Learning per Applicazioni Big Data su Cloud

Bruno Guindani, Danilo Ardagna and Alessandra Guglielmi

Abstract Bayesian Optimization is a promising method for efficiently finding optimal cloud computing configurations for big data applications. Machine Learning methods can provide useful knowledge about the application at hand thanks to their predicting capabilities. 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.
Abstract *L'ottimizzazione bayesiana è un metodo promettente per trovare configurazioni ottimali di applicazioni big data eseguite su cloud. I metodi di machine learning possono fornire informazioni utili sull'applicazione in oggetto grazie alle loro capacità predittive. In questo articolo, proponiamo un algoritmo ibrido basato sull'ottimizzazione bayesiana che integra tecniche di machine learning per risolvere problemi di ottimizzazione con vincoli di tempo in sistemi di cloud computing.*

Key words: acquisition function, cloud computing, Gaussian Process

1 Introduction

Big data analytics are employed in several industries 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 execution times and reduce costs is essential to service quality and business competitiveness. However, due to the diverse behavior and resource requirements of analytic jobs, choosing the best configuration for a broad spectrum of applications is a challenging process [1].

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 [6] or the popular tutorial paper [3]. BO is a sequential design strategy that requires few steps to get sufficiently close to the true optimum,

Bruno Guindani[1], Danilo Ardagna[1] and Alessandra Guglielmi[2]
[1] Department of Electronics, Information and Bioengineering, Politecnico di Milano, Milano, Italy
[2] Department of Mathematics, Politecnico di Milano, Milano, Italy
e-mail: {bruno.guindani, danilo.ardagna, alessandra.guglielmi}@polimi.it

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) using these points. 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), as explained in [4], for instance.

The goal of this work is to integrate Bayesian optimization algorithms with Machine Learning (ML) techniques in the context of cloud computing optimization. The former have proven to be successful [1, 7] 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 [1], in which the *CherryPick* system is successfully applied to benchmark applications on cloud computing frameworks such as Apache Spark. This system exploits pure BO to find optimal cloud configurations. Our work is motivated by the belief that ML can lend their predicting capabilities to BO to further improve its effectiveness. This topic has been explored in [5], which examines the performance of several ML models in carrying out predictions 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. Section 2 describes the mathematical formulation of the problem, Section 3 presents our proposal of a BO algorithm, while Section 4 collects some preliminary experimental results.

2 Background and mathematical formulation

The goal of BO is to approximate, e.g., the minimum of a given function f, called objective function, by using as few iterations as possible. Namely, we want to find $\widehat{x}$ where $\widehat{x} = \arg\min_{x \in \mathscr{A}} f(x)$. Strong assumptions on f or on the minimization domain $\mathscr{A}$ 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 [2]), 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 [1]. Let $x \in \mathscr{A}$ 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 $\mathscr{A} \subset \mathbb{R}^d$ 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. Hence the problem is to find the minimum of f,

$$\min_{x\in\mathscr{A}} f(x) = P(x)T(x) \quad \text{s.t.} \;\; f(x) \le P(x)\,T_{max}. \tag{1}$$

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 [5] on the data considered for validation. In a more general context, data can be assumed to have independent, normally distributed additive noise with variance η^2.

In this context, the Gaussian process (GP) is the preferred choice for the prior for f. This means that for any $x \in \mathscr{A}$,

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

Functions $\mu_0(\cdot)$ and $\sigma_0^2(\cdot,\cdot)$ are called mean and kernel functions, respectively, and are the GP model hyperparameters. In this work, we assume $\mu_0(\cdot) \equiv \mu_0$ and we use the Matérn kernel with smoothness parameter $\nu = 5/2$ (see [3]):

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

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 each $f(x)$, which is also Gaussian and it is characterized by the posterior mean $\mu_n(\cdot)$ and variance $\sigma_n^2(\cdot)$, i.e.

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

which can be computed by well-known properties of GPs (see, for instance, [3]).

BO 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. See Figure 1 for a summary of how BO works. Specifically, in the top panel, the objective function to be minimized (in gray) is approximated by the Gaussian Process, in the form of its posterior mean function (in dashed blue) and 95% credible interval (in light blue) after evaluating 3 points (red dots). The bottom panel shows the acquisition function given the current posterior distribution. The red cross indicates the maximum of the acquisition function, i.e. the next point which will be evaluated.

In this paper, we compare different acquisition functions. The *Expected Improvement* (EI) over the best value f_n^* found by the optimization process so far is:

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

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 (see [8]), which accounts for the probability of a point of respecting the constraints:

$$EIC_n(x) := EI_n(x) \cdot \mathbb{P}_{\pi_x(\cdot|H_n)}\big(f(x) \leq P(x)\, T_{max}\big). \tag{2}$$

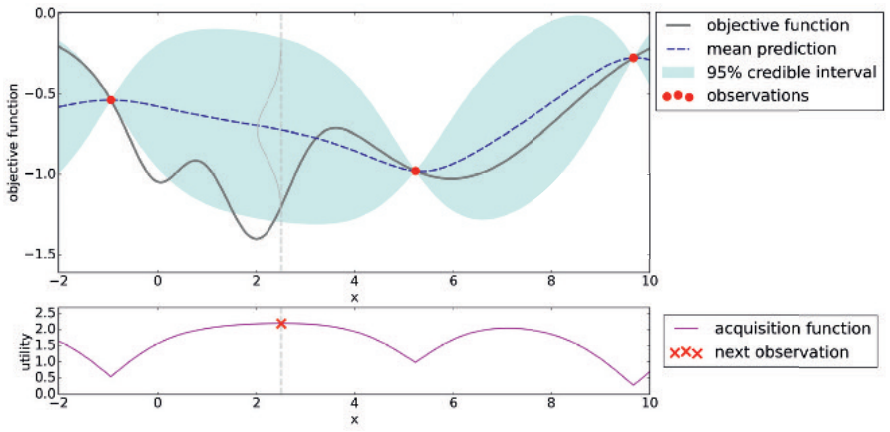

Fig. 1: Bayesian optimization after 3 iterations. Top panel: the objective function and its Bayesian approximation. Bottom panel: acquisition function.

3 Our hybrid algorithm

Our algorithm is based on pure BO, but it integrates elements coming from ML techniques. We use a memory queue for discrete features to prevent exploration of already visited values. The algorithm continues until the evaluated running time at the current iteration is sufficiently close to the time threshold: $T(x_n) \in [0.9\,T_{max},\ T_{max}]$. 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 consumes the least amount of resources for that configuration to be feasible. After termination, it is likely that we have found the true optimal configuration because of the convergence properties of the BO algorithm. Afterwards, we execute subsequent runs using such optimal or near-optimal configuration.

As far as the acquisition function is concerned, in this paper we use two variations of EIC (see (2)), which both integrate EIC with information coming from ML.

In particular, at each iteration, a ML model $\widehat{T}(\cdot)$ is trained on the current history H_n to accurately predict execution times $T(x)$ for any configuration x. Then, the EIC acquisition function in (2) is either multiplied by an exponential factor of $\widehat{T}(x)$ (variant B), or is set to zero for values in which $\widehat{T}(x)$ is larger than the threshold T_{max} (variant C). The former encourages the search process towards configurations which respect the time constraint, while the latter prevents the search outright in areas which are predicted to violate such constraint.

4 Experiments

We present preliminary results using the techniques we have discussed in Section 3. We use variants B and C of the algorithm, as well as pure BO, on the Query26 application from the TPC-DS industry benchmark run on Apache Spark with input data size equal to 250 GB and time threshold equal to 150 s. In this case, we optimize the total cost in (1) on the number of cores x. The same three fixed initial points were used for all variants. Figure 2 shows the comparison of pure BO (top row) with variants B (middle row) and C (bottom row) at each algorithm iteration. The left panels display the number of cores x selected by the algorithm, while the middle panels show the cumulative costs of the selected configurations. The percentage errors between the actual execution time and the one predicted by the ML model are displayed on the right column. We apply Ridge regression since [5] shows that it is the most accurate in this context, with a mean absolute error smaller than 5%.

Using our new algorithm, we are able to reduce the number of iterations which produce unfeasible configurations from 15 to 1 or 2. Similarly, cumulative costs associated to unfeasible runs are reduced from 11.67 to 0.80 and 1.51. At the very first iteration, all variants explore high values of the number of cores since there is no enough information on that part of the domain. This also explains the large prediction errors at the same iteration. After that, our ML model is able to predict the execution time of subsequent iterations with very good accuracy. Finally, the termination criterion (see the vertical dotted line in Figure 2) correctly assesses the optimality of the configuration with $x = 22$ cores, and stops the exploration phase.

Acknowledgments

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. O. Alipourfard, H. Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang. Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In *NSDI USENIX*, 2017.
2. E. Brochu, V. M. Cora, and N. De Freitas. 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.

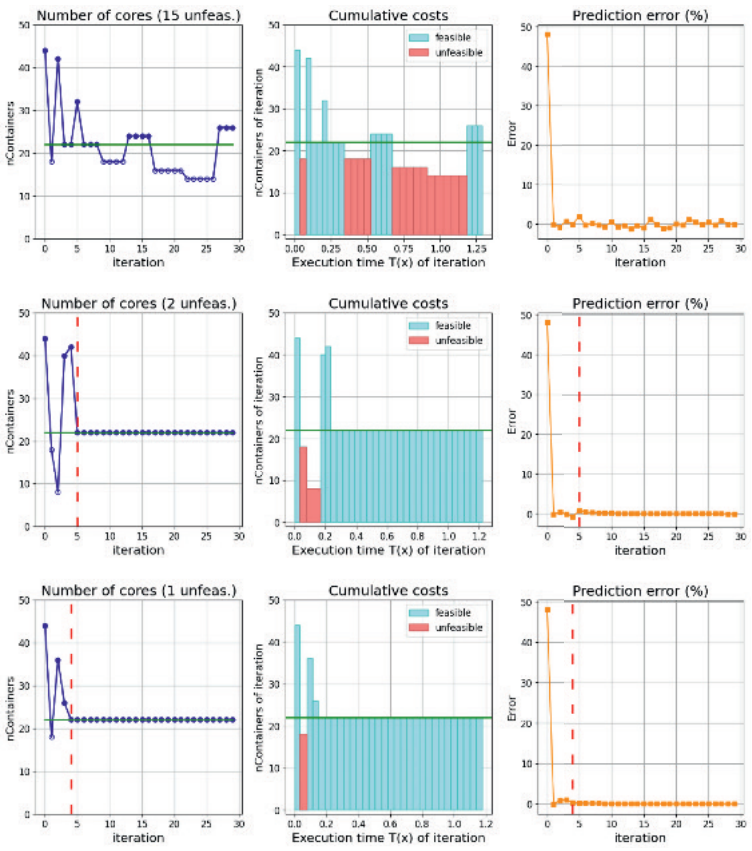

Fig. 2: Comparison of pure BO (top row) with variants B (middle row) and C (bottom row) at each algorithm iteration. Left panel: chosen numbers of cores x; middle panel: cumulative costs of the chosen configuration; right panel: percentage error. The true optimal number of cores is denoted by the green horizontal line.

3. P. I. Frazier. A tutorial on Bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.
4. B. Letham, B. Karrer, G. Ottoni, and E. Bakshy. Constrained Bayesian optimization with noisy experiments. *Bayesian Analysis*, 14:495–519, 2019.
5. A. Maros, F. Murai, A. P. C. da Silva, J. M. Almeida, M. Lattuada, E. Gianniti, M. Hosseini, and D. Ardagna. Machine learning for performance prediction of spark cloud applications. In *IEEE 12th International Conference on Cloud Computing (CLOUD)*, pages 99–106, 2019.
6. T. Pourmohamad and H. K. Lee. Bayesian optimization via barrier functions. *Journal of Computational and Graphical Statistics*, (just-accepted):1–23, 2021.
7. B. Reagen, J. M. Hernández-Lobato, R. Adolf, M. Gelbart, P. Whatmough, G.-Y. Wei, and D. Brooks. A case for efficient accelerator design space exploration via Bayesian optimization. In *2017 IEEE/ACM ISLPED*, pages 1–6. IEEE, 2017.
8. M. Schonlau, W. J. Welch, and D. R. Jones. Global versus local search in constrained optimization of computer models. *IMS Lecture Notes-Monograph Series*, pages 11–25, 1998.