Multi-Phase Task-Based HPC Applications: Quickly Learning how to Run Fast

Lucas Nesi, Lucas Mello Schnorr, Arnaud Legrand

Multi-Phase Task-Based HPC Applications: Quickly Learning how to Run Fast

Lucas Leandro Nesi
Institute of Informatics, PPGC/UFRGS
Porto Alegre, Brazil
Univ. Grenoble Alpes
Grenoble, France
lucas.nesi@inf.ufrgs.br

Lucas Mello Schnorr
Institute of Informatics
PPGC/UFRGS
Porto Alegre, Brazil
schnorr@inf.ufrgs.br

Arnaud Legrand
Univ. Grenoble Alpes, CNRS,
Inria, Grenoble INP, LIG
F-38000 Grenoble, France
arnaud.legrand@imag.fr

***Abstract*—Parallel applications performance strongly depends on the number of resources. Although adding new nodes usually reduces execution time, excessive amounts are often detrimental as they incur substantial communication overhead, which is difficult to anticipate. Characteristics like network contention, data distribution methods, synchronizations, and how communications and computations overlap generally impact the performance. Finding the correct number of resources can thus be particularly tricky for multi-phase applications as each phase may have very different needs, and the popularization of hybrid (CPU+GPU) machines and heterogeneous partitions makes it even more difficult. In this paper, we study and propose, in the context of a task-based GeoStatistic application, strategies for the application to actively learn and adapt to the best set of heterogeneous nodes it has access to. We propose strategies that use the Gaussian Process method with trends, bound mechanisms for reducing the search space, and heterogeneous behavior modeling. We compare these methods with traditional exploration strategies in 16 different machines scenarios. In the end, the proposed strategies are able to gain up to ≈51% compared to the standard case of using all the nodes while having low overhead.**

***Index Terms*—HPC, Heterogeneous, Task-Based, Distribution, Load Balancing**

I. Introduction

Resource heterogeneity is a reality in many HPC computational resource providers. It is present intra-node, with the combination of CPU and accelerators, and also at a system level, where different computational nodes exist and can be used together. The Swiss Piz Daint[1], the French Jean Zay[2], and the Brazilian SDumont[3] are some examples of the Top500 list [1] supercomputers equipped with multiple heterogeneous nodes, grouped into two or more partitions. While supercomputers do have system-level heterogeneity, it is even more present in the Cloud [2]. Efficiently exploiting such heterogeneous sets of resources is unfortunately very challenging for HPC applications.

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, the National Council for Scientific and Technological Development (CNPq), grant no 141971/2020-7 to the first author, and the projects: FAPERGS (Data Science – 19/711-6), CAPES/Cofecub (04/2017).

[1]https://www.top500.org/site/50422/

[2]https://www.top500.org/site/50403/

[3]https://www.top500.org/site/50576/

Traditionally, HPC applications are developed directly with relatively low-level APIs like MPI and CUDA, which makes it very hard to obtain good and portable performance. Indeed, HPC applications often comprise several computational phases that may have very different resource requirements. Using the same rigid block-cyclic distributions across all application phases often incur spurious communication overheads (e.g., when the number of nodes is a prime number) and often complicate load balancing. It is thus common to use different distributions [3], [4] and try to overlap the phases as much as possible to improve the overall application performance, but this remains very difficult without resorting to a higher-level programming paradigm. This complexity has motivated the adoption of the task-based programming paradigm in which the application is structured as a Directed Acyclic Graph (DAG) of tasks. A runtime then schedules the tasks dynamically to computational resources based on performance models and the system state. This separation of concerns allows to easily implement flexible data distribution [5] and generally leads to better performance.

Although the task-based paradigm largely mitigates communication overhead, unforeseen effects (e.g., network contention or complex inter-node synchronizations) remain possible and particularly hard to model, especially when trying to exploit heterogeneous sets of nodes. In this context, finding an adequate number of computational nodes to use for each phase can be particularly challenging to anticipate. Thus, methods that dynamically learn and adapt to such complex scenarios and improve performance over time are desirable.

In this paper, we study and propose, in the context of a task-based iterative multi-phase GeoStatistic application, strategies for the application to actively learn and adapt to the best set of heterogeneous nodes it has access to. We model the application with a Gaussian-Process (GP), which provides us with a surrogate to guide the exploration and optimization. During the execution, the application can query the surrogate to determine which action to take to adapt itself to the available computing resources. Experimental results are gathered using real and simulation environments with heterogeneous machines setups.

The main contributions of this paper are the following. (i) We analyze this problem's main characteristics (structure

and noise) and explain why generic optimization and learning techniques are likely to fail. This analysis motivates the design of specific variations of a reinforcement learning technique based on Gaussian-Process. To quickly learn, we also take particular care in the initialization (the exploration in the first iterations) and propose a bound mechanism to discard bad cases and reduce the search space. (ii) We conduct a comprehensive performance evaluation with 16 different heterogeneous machines and workloads that compare the proposed solutions with other generic optimization methods (Brent, Bandits, GP-UCB). Among these various methods, we show that our carefully designed GP-based optimization is the only robust and parsimonious method to quickly reach the optimal configuration in a wide variety of scenarios. This contribution includes an external interactive tool to illustrate how off-the-shelf strategies behave for a particular setup and iteration[4]. (iii) A real implementation of the method to enable the application to adapt during execution, demonstrating the low overhead of the methods.

The rest of the paper is structured as follows. Section II describes the structure and complexity of the selected application and the used runtime. Section III presents the challenges of selecting the best number of heterogeneous nodes with real data. Section IV presents standard exploration strategies and the different strategies based on the GP-UCB mechanism but tailored to the problem. Section V presents the experimental methodology, including real experiments, simulations, and the computation of the results. Section VI presents the experimental evaluation of the proposed strategies, including a detailed step-by-step analysis, an overview of the results in all 16 different scenarios, and the strategy cost overhead evaluation. Section VII presents related work in searching for the best types of machines or resources to optimize an application. Finally, Section VIII concludes this paper with a discussion and further considerations of investigation. A public companion[5] contains the software modifications, data, and instructions to reproduce our results.

II. ExaGeoStat: A Multi-Phase Task-Based Applic.

A Task-Based application is described as a directed acyclic graph of tasks and their dependencies. They are constructed in a declarative manner [6], with no assignment of where and when the tasks or communications (dependencies) will execute. A runtime schedules the tasks dynamically based on the system state. Out of the many available runtimes, this work uses StarPU [7] because it allows transparent data redistribution by simply informing the new data locations. All data blocks used by tasks need to be registered using the StarPU API and defining a node that owns it.

StarPU can work using Sequential Task Flow (STF) [8], where tasks are submitted sequentially, and a DAG is generated dynamically on execution time. A task will execute on the node that owns the data blocks they write. StarPU can schedule tasks using performance models that assume a similar duration for a given task type input size. Also, outlier tasks (that may present abnormal duration in relation to the task type mean) are handled by the StarPU scheduler on each node. During the submission of tasks, it is possible to inform the runtime about data movement, causing the following submitted tasks to change their execution node accordingly. These movements can reflect new distributions, application-tailored, and can use more or fewer nodes. The StarPU runtime will move all the data to the right place asynchronously overlapping with computation. Consequently, calling the communication library or creating a synchronous point for such data movement is unnecessary. Similarly, the task-based paradigm enables the asynchronous execution of different application phases.

ExaGeoStat [9] is a StarPU-enabled application that allows the prediction of missing observations. The goal is to model spatial data (X, Z), where X are locations and Z observations. This process requires the iterative optimization of the covariance kernel hyper-parameter θ by maximizing the log-likelihood. At each main loop iteration, the application goes through the following five phases: (i) Generation of the Σ_θ covariance matrix (ii) Cholesky Factorization of Σ_θ using the Chameleon library [10], (iii) Solve, (iv) Determinant, and (v) Dot product. However, the two most significant phases are Generation and Factorization. Figure 1 (Generated by StarVZ [11], [12]) depicts three iterations of ExaGeoStat where the x-axis is the time, and the y-axis has the aggregated resource type utilization per node. The different colors correspond to different phases: the yellow ones are the generation, while the green ones are the tasks the factorization, a small number of tasks in gray correspond to the other three phases. The phases can use different distributions and can overlap as any spurious synchronization is removed thanks to the task-based programming and other optimizations [4].

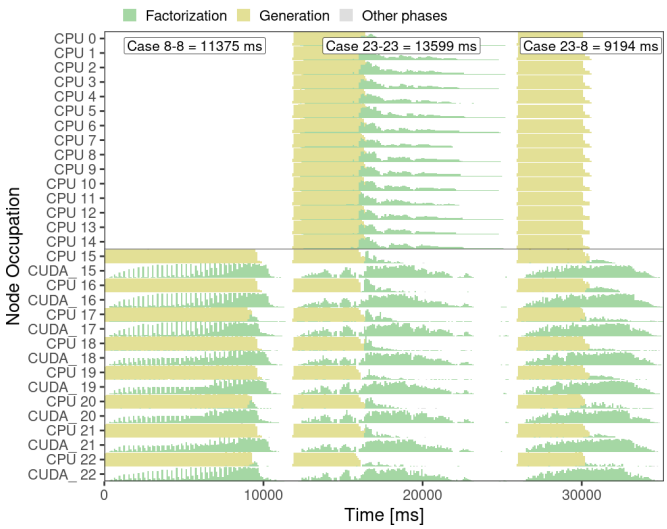

Fig. 1. Three iterations of ExaGeoStat: the first using a small amount of homogeneous nodes for both phases. The second increasing the number of nodes (with CPU-only nodes) and using all for both generation and factorization, the third restricting the factorization to the eight fast nodes.

[4]Publicly available at: https://adaphetnodes.shinyapps.io/shiny/

[5]https://gitlab.com/lnesi/ipdps22

ExaGeoStat's iteration phases have very different computational requirements and resource affinities. While generation only runs on CPUs, the Cholesky factorization can exploit GPUs to accelerate the application. Moreover, the factorization cost is stationary across iterations, as it is only based on the matrix size, and the generation cost is mostly constant between iterations when using adequate parameters. However, the distribution of multiple phases for a given number of resources is not trivial. A recent work [4] demonstrates that resource heterogeneity can improve performance by using nodes more suitable to each phase. This strategy is depicted in Figure 1, where the generation phase ends earlier on the second iteration, but excessive communication or critical-path problems slow down factorization. However, iteration three, using all nodes for generation and only the eight faster nodes for the factorization, presents the best makespan. The work also relies on a linear program (LP) to determine the ideal number of tasks each node should receive, considering a specific amount of nodes to use. This LP also serves as a makespan lower bound and is used in this work. The final distribution for each phase is based on heterogeneous methods [13], [14]. For iterative multi-phase applications such as ExaGeoStat, it is interesting to have nodes that will only be used for some phases, like the generation, and not for the other ones.

However, using a different number of nodes for interleaving phases may cause unforeseen network contention and heterogeneous distributions problems (see Section III for further details). It is also unsatisfactory and costly to manually discover how such a complex application would behave for a certain number of nodes, hardware, and workload. Consequently, it is desirable that this application adapts to any HPC system without extensive analysis or complete executions of all possible configurations. Automatic adaptation for these setups is key to achieving portable performance. Therefore, our goal is to design a method that learns the best set of nodes to use for each application phase during runtime. The method could exploit the application's iterations to explore different configurations and find the best number of nodes to use per application phase, considering a regular iteration cost on the same number of resources. Many applications have this structure of stable iterations (stationary workload) but whose total duration is difficult to anticipate in some setups, as some phases scale well while others do not. For the usage of other runtimes, a feature of moving data arbitrarily during execution time based on the dynamic choice of the strategy is required.

III. Varying heterogeneous nodes per phase

Estimating the ideal number of nodes to use per phase is a complex process [4], and not only because adding infinite nodes is not the ideal situation. A perfect duration modeling would require anticipating the stochastic behavior of the scheduler, the network conditions, and the distribution's issues. It seems unfeasible to anticipate every possible condition. Figure 2 provides three representative examples of the ExaGeoStat iteration duration (Y-axis) depending on the number of factorization nodes it uses (X-axis). The application uses all the nodes in the generation step for all scenarios as this phase is embarrassingly parallel. In the figure, the nodes in the X-axis are sorted by computational power, meaning that we always use the fastest nodes in the set. The nodes are organized in three categories, Large (L), Medium (M), and Small (S), depending on the computing capability of each category (Further described in Table II and Section VI-A), as depicted by the vertical black lines showing when the categories change. The dark blue line corresponds to the lower bound provided by the LP. The yellow and green vertical bars represent the duration of the asynchronous generation and factorization phases. In all cases, the granularity of the workloads is large enough to exploit parallelism in all nodes and present a similar resource usage behavior as in Figure 1.

It is possible to observe that these are complex scenarios. In all of these cases, using all nodes for all phases is sub-optimal. The main behavior observed is that the addition of new nodes usually forms convex-like shapes. In the beginning, adding new nodes is beneficial by having more processing power. However, there is a point where adding new nodes is no longer useful, and there are some scenarios where the network could get overwhelmed. In these setups with a limited network, the performance starts to deteriorate. Furthermore, there are scenarios with significant breaks, usually related to the heterogeneity and the distribution shape. Sometimes, adding a slow node (especially CPU-only ones), like in scenario (p), creates a critical path that may degrade the overall performance. The observation noise is generally the same for all number of nodes, with few outliers. Some scenarios like (i) have small breaks related to the distribution. Adding new nodes may cause the reorganization of the partition structure, creating more communications and synchronizations.

IV. Exploration Strategies

In the case of the studied application, ExaGeoStat, although computation phases can partially overlap thanks to the fine-grain dependencies expressed in StarPU, an iteration (the evaluation of the likelihood of a given value of θ) cannot start before the previous one is completely finished. At each iteration, the application may thus select a different subset of nodes and use the iteration duration as our minimization target. Instead of exploring all possible nodes permutations, it can choose n between 1 and N nodes to use and pick the n fastest nodes since trading a slow node for a fast one is always detrimental. Therefore, our search space consists of the number of nodes per phase. Although this search space is discrete, it is visible from Figure 2 that there is an underlying continuous structure. This section presents common methods to explore and optimize such search space.

A. Naive Heuristics

For comparison purposes, we implement two simple naive heuristics. First, a simple divide and conquer dichotomy (**DC**) to carry out a recursive binary search over the space. At each step of the exploration, the search space is divided in two, and

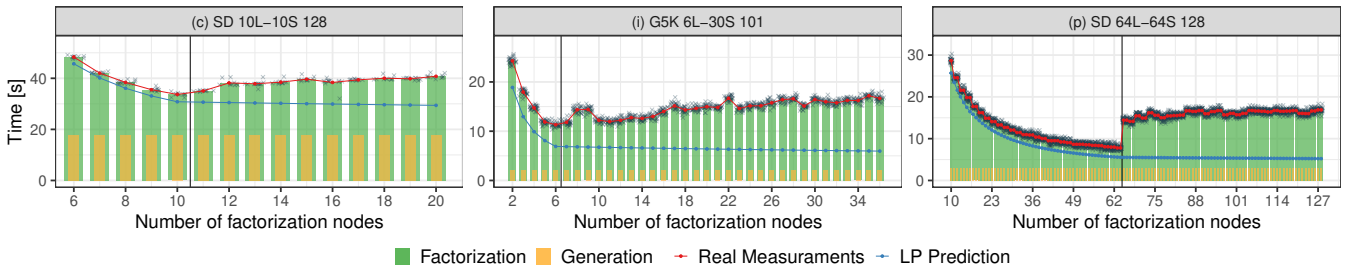

Fig. 2. Behavior using different heterogeneous nodes setups (Table II) by varying the number of factorization nodes.

the middle point of each division is measured. The method selects the division that has a lower makespan as the new search space. This simple heuristic will converge quickly and pick the right point in simple low variance curves. Second, a heuristic that assumes that the best candidate is to use all the machines. The heuristic walks from the rightmost option to the left, while the left point presents a lower measurement (**Right-left**). The method works when the curve is also well-behaved (without huge discontinuities), and the excessive number of resources creates extra overhead on the rightmost options.

B. Classical continuous minimization approaches

Another subset of strategies is the ones for classical continuous optimization. The simplest algorithms use gradient, which is unavailable. Yet, the problem's search space may have local minima. Since there is only one dimension in our context, a sensible choice is the **Brent** algorithm, which combines the bisection method, the secant method, and inverse quadratic interpolation. It is also available on R's `optim` [15]. We also tried multi-dimension algorithms like Nelder-Mead and BFGS with no better results. We also investigated Stochastic Approximation [16] and Simulated Annealing (SANN from optim), but they achieved bad results because they are not parsimonious, so we refrain from reporting them.

C. Multi-armed bandits

Multi-armed bandits are a Reinforcement Learning framework that models K possible unrelated choices [17]. Each choice has its distribution and variance. The goal is to maximize its total reward, $\sum_{t=1}^{T} y(n_t)$, where $y(n)$ is the (stochastic) reward of step t (in our case, the opposite of the duration of an iteration when using n_t nodes). The difference between the cumulative reward from choosing the best action upfront is the regret. To minimize regret, one should balance exploration (to discover the best action) and exploitation (selecting the best action to improve the overall reward). A no-regret strategy (i.e., whose regret is $O(\log(T))$, the optimal bound) consists in using the Upper-Confidence-Bound (**UCB**) algorithm [18] that selects the action which maximizes the mean empirical reward plus an upper bound that increases over time:

$$x_{t+1} = \underset{x \in A}{\operatorname{argmax}}\ \mu_t(x) + c\sqrt{\ln t / N_t(x)} \qquad (1)$$

$\mu_t(x)$ is the mean reward measured so far for action x, c is a adjustment constant, $N_t(x)$ is the number of times action x was selected. Best actions are often selected while less promising actions are not taken frequently but have their upper confidence bound component increased so that they are eventually selected again after some time.

In our case, each action corresponds to a number of nodes. However, the fact that a similar number of nodes lead to similar performance is not exploited, making large setups have large search spaces that will require a long time to explore. To speed up the exploration, it is natural to consider a restricted version (**UCB-struct**) that will only look at multiple complete groups of homogeneous nodes. For example, in a setup with five machines of group A, five of B, and five of C, the only options are 5, 10, or 15 nodes. If the best action is outside these choices, it will never reach the optimal configuration.

D. Gaussian Process

The Gaussian Process [19] or Kriging is a surrogate model that assumes a form of smoothness over the data. It uses an Multivariate Normal Distribution (MVN) to model possible realizations over observations. The assumption is we're trying to optimize a function f, which is unknown, and which is assumed to have been drawn at random from the set of smooth functions, $f \sim GP$. Furthermore, this is a noisy scenario so $f(x)$ cannot be directly observed, all we have is noisy observations $y(x) = f(x) + \epsilon$ with $\epsilon \sim \mathcal{N}(0, \sigma_N^2)$. Then, based on a set of t observations $D_t = \{x_i, y_i\}$ and given the smoothness of the function which stems from the GP assumption, it is possible to compute $\mu_t(x) = E[f(x)|D_t]$ and $\sigma_t(x) = \sqrt{Var[f(x)|D_t]}$ with standard kriging R libraries.

After the generation of the surrogate model, it can be used to select an action to take. This decision needs to consider the predicted mean of the location, its confidence interval, and the exploration and exploitation trade-off. A possible approach with the same kind of no-regret properties as UCB is to use GP-UCB [20] where the following equations (with β growing logarithmic with iterations) will give the choice:

$$x_{t+1} = \underset{x \in A}{\operatorname{argmax}}\ \mu_t(x) + \beta_t^{1/2}\sigma_t(x) \qquad (2)$$

Figure 3 illustrates such a process. The red line is the *cos* function representing a true unknown function that GP wants to model. The black points are real random measurements taken in the function, which will be used as the GP's input. The black line is the predictive mean provided by the GP,

while the gray area inside the dashed black lines represents the 95% confidence interval. The mean prediction is very close to the real function in the neighborhood of measured points. Coordinates that do not have measurements have more uncertainty, but the true *cos* function indeed lies in the 95% confidence region. The red cross is the next point to be evaluated, as it represents the most promising point while considering the current uncertainty.

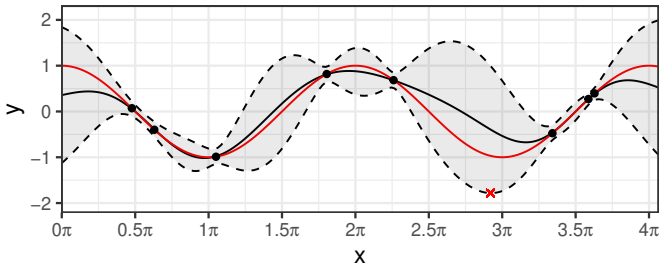

Fig. 3. An example of the GP fit with eight measurements over *cos* function.

The GP puts a probability distribution over smooth functions through a covariance function (typically the exp function), which is commonly parameterized as:

$$\Sigma(x, x') = \alpha \; exp\left\{\frac{-||x - x'||}{\theta}\right\} \tag{3}$$

It has parameters α, and θ that GP-UCB assumes are known. In practice, they are often estimated from the data with an ML approach (as in ExaGeoStat), but with little data, this can be a problem as with bad luck, the algorithm may be overconfident. Likewise, σ_N is generally estimated from the data.

In practice, when using GP and building surrogates, it is common to initialize the process with a uniform quasi-random design (e.g., LHS, maximin), where the x_i are uniformly spread over the space as it allows a reasonable estimation of the hyper-parameters. However, this would be too costly in our case, and we need a more parsimonious approach. Therefore, the model selects the actions of the first iterations following a simple procedure. The first iteration will always choose the action with N nodes, which is the application's default behavior. Using all the available nodes would provide the best performance in ideal circumstances. Then, to obtain as much information as possible, standard Bayesian optimization approaches would select the left-most point, i.e., the configuration which uses the minimal number of nodes, which may not be a good idea from an application makespan perspective. Finally, it selects the middle of these two points for the subsequent two iterations.

Any new measurement will bring some information about f but also about α, θ, and σ_N. Measuring the same location several times provides a lot of information about σ_N. In our case, for example, σ_N is estimated as follows. We consider $S = \{x \in D | n(x) > 1\}$, then σ_N^2 can be estimated by $(\sum_{x \in S} \sum_{y(x) \in D} (y(x) - \bar{y}(x))^2) / (\sum_{x \in S} n(x) - 1)$. Conversely, measuring two points next to each other provides mostly information about α and θ.

The version so far described is **GP-UCB**, and it is reasonable when we do not know a priori the shape and the noise is roughly the same regardless of x. However, the f function may have discontinuities (see Figure 2(p)), and the functions are not entirely random as they have a general shape, more or less decreasing than increasing (see Figure 2).

The standard GP has no particular trend, which explains why in Figure 3 it naturally reverts to 0. If we know something about the shape of f, it should be put in the trend $\mu(x) = \sum_i \gamma_i g_i(x)$ where g_i is a basis function and γ_i the coefficients. But again, the more unknown parameters, the more complex the model and the more measurements we will have to do to reduce uncertainty and explore instead of exploit. In our case, measuring two points far away from each other gives a lot of information about γ if we have a linear model.

So we exploit the properties of our scenario. First, the results indicate that the trend of the observations follows a pattern of $1/x + x$. This follows the intuitive view that the makespan will decrease every time a new node is added. However, adding a new node may cause more communications, and therefore there is an overhead associated with it. However, the $1/x$ parameters are not very useful as this part is already well estimated in the LP. So another approach is to use the LP as a baseline and use the GP to model the overhead with respect to the LP. In this sense, the trend will be linear x, as the LP captures the $1/x$ behavior. Second, bound the exploration with the LP results. Considering that the first iteration uses all the machines, some configurations with very few resources cannot provide better results when comparing a theoretical lower bound (LP) to the actual first iteration makespan. In G5K 6L-15S or SD 64L, the most left points are inevitably higher than using all nodes as predicted by the linear program. To find the most adequated left-point, the method uses the linear programming bound and find the lower n_l that satisfy $\mathrm{LP}(n_l) < f(N)$. The bound given by the linear program is optimistic and does not consider communications nor critical path. The approach excludes all the points where the linear programming bound is higher than using all the nodes. This will limit the search space and avoid bad actions.

Finally, the model so far is still imperfect because they are too smooth. Indeed, some scenarios present a discontinuity of the performance when a new group of machines is used. Although the GP-UCB will eventually explore all x after a very long time, a strong smoothness assumption may prevent finding optimal configuration shortly. This situation is sometimes caused by abrupt changes in the partition distribution or because of the critical path. The new nodes may be so less powerful than the others that the few data blocks that they receive may cause a synchronous critical path. Many other tasks may have to wait for it, causing a global slowdown. In the case of Cholesky, giving a block of coordinate (i, j) to a node will cause a synchronous sequence of $\min(i, j)$ `dgemm` tasks on that particular node. This can take some time if the node does not have GPUs.

To model such discontinuities, we introduce dummy variables [21] to the trend model for each group of homoge-

neous machines. In this way, the trend will be given by $x + \sum_{g \in G} d_g(x)$ where $d_g(x)$ will be 1 if the node x is present on group g and 0 otherwise. The dummy variable allows us to indicate where discontinuities may appear and generally allow to pool, which is good to estimate parameters. However, this model may get overconfident when the number of measurements is low, given a bad estimate for θ. To overcome this, we set θ to 1 and α to the sample variance. Because this model adds new variables, it requires more initial points for the first fit. Then, each group (after the leftmost point) will have its last point measured once. If this point is already measured, we choose to evaluate the next point. The last group (using all the machines) is already measured and skipped. This is **GP-discontinuous**. All of the GP versions are implemented using the R package DiceKriging [22].

E. Summary of Strategies

All presented strategies will behave differently in this particular problem. Table I presents a list of all discussed algorithms with expected behavior. Our expectations, considering the technical aspects of the algorithms, are that **DC**, **Right-left**, and **Brent** were not resilient to noise and so can be misled by the measurements in this problem. **UCB** would require a full exploration of the search space, which can be bad in applications with few iterations, and because some configurations are particularly bad. **UCB-struct** will only search a fraction of the possibilities. In this way, if the best case is not one of them, it will never discover it. **GP-UCB** does not use the problem particularities and insights to improve its model, which leads to the exploration of bad configurations and lower confidence when finding discontinuities. **GP-discontinuous** is our proposed version that uses knowledge of the problem to solve some of the earlier issues discussed.

TABLE I
SUMMARY OF EXPLORATION STRATEGIES AND EXPECTED BEHAVIOR

Algorithm	Resilient to noise	Optimal	Fast
DC			×
Right-left			×
Brent			×
UCB	×	×	
UCB-struct	×	(limited exploration)	×
GP-UCB	×	×	
GP-discontinuous	×	×	×

V. METHODOLOGY

Two performance evaluation methods have been used in this work. First, we use real executions in real environments, running ExaGeoStat to solve the used workload entirely and forcing the exploration of all possible number of nodes sets for the factorization phase. These measures enable the estimation of the true variability of the application and the system. Second, we rely on simulations of ExaGeoStat with StarPU-SimGrid, whose accuracy has been shown in [5], [23] but which we also consistently checked.

Using simulation allows us to quickly and accurately estimate the response of ExaGeoStat even for platforms that are not directly available or would be very time and resource consuming. However, with the default StarPU-Simgrid configuration, the time for a given iteration and number of nodes is deterministic. Therefore, the simulation evaluation of each configuration is augmented 30 times, assuming a normal distribution with a standard deviation of 0.5s (computed from the real experiments). Real experiments and simulations are marked in the title of each configuration in Figure 5.

The evaluation of the exploration strategies proposed in Section IV also has been conducted using two complementary techniques. First, to obtain a fast and fair comparison, we used all the iteration durations obtained through real experiments or simulation and resampled them in R every time an action was chosen. This way, all exploration strategies are compared with the exact same iteration durations (see the Shiny app[4]), and their comparison can be made reliable from a statistical point of view. Finally, we also implemented the GP strategies directly in ExaGeoStat, which lets it control the number of nodes it uses. All the possible distributions were precomputed and are accessed when an action is chosen. With this second approach, it is possible to measure the computational cost overhead of the methods, as discussed in Section VI-E.

VI. EXPERIMENTAL EVALUATION

A. Hardware and Software Description

The computational nodes used in the experiments are from Grid5000 (*G5K*) and the Santos Dumont supercomputer (*SD*), as shown in Table II with CPUs and GPUs. In Grid5000, the Chetemi and Chifflet network is a 10Gb/s Ethernet, while the Chifflot interconnection is a 25Gb/s Ethernet. A 2x100Gb/s Ethernet network connects both partitions. In Santos Dumont, an Infiniband FDR 56Gb/s network interconnects all the nodes.

TABLE II
COMPUTATIONAL NODES USED IN THE PERFORMANCE EVALUATION

	Site	Machine	CPU	GPU
S	G5K	Chetemi	2x Xeon E5-2630 v4	–
M	G5K	Chifflet	2x Xeon E5-2680 v4	2x GTX 1080
L	G5K	Chifflot	2x Xeon Gold 6126	2x Tesla P100
S	SD	B715	2 x Xeon E5-2695v2	–
M	SD	B715-GPU[6]	2 x Xeon E5-2695v2	1xK40
L	SD	B715-GPU	2 x Xeon E5-2695v2	2xK40

The experiments use ExaGeoStat on commit `9518886` of its public repository, with other software dependencies present as submodules of the same commit. The StarPU version is the master branch commit `015357bd` with the NewMadeleine library [24] commit `d6542d72` and backend. ExaGeoStat and Chameleon have modifications to accept custom distributions, asynchronous behavior, and the GP method[7]. The experiments use the 96100 (101×101 blocks) and the 122880 (128×128 blocks) matrices from the ExaGeoStat samples.

[6]Artificial machine to increase heterogeneity by only using one GPU.
[7]Available in the paper's companion: https://gitlab.com/lnesi/ipdps22

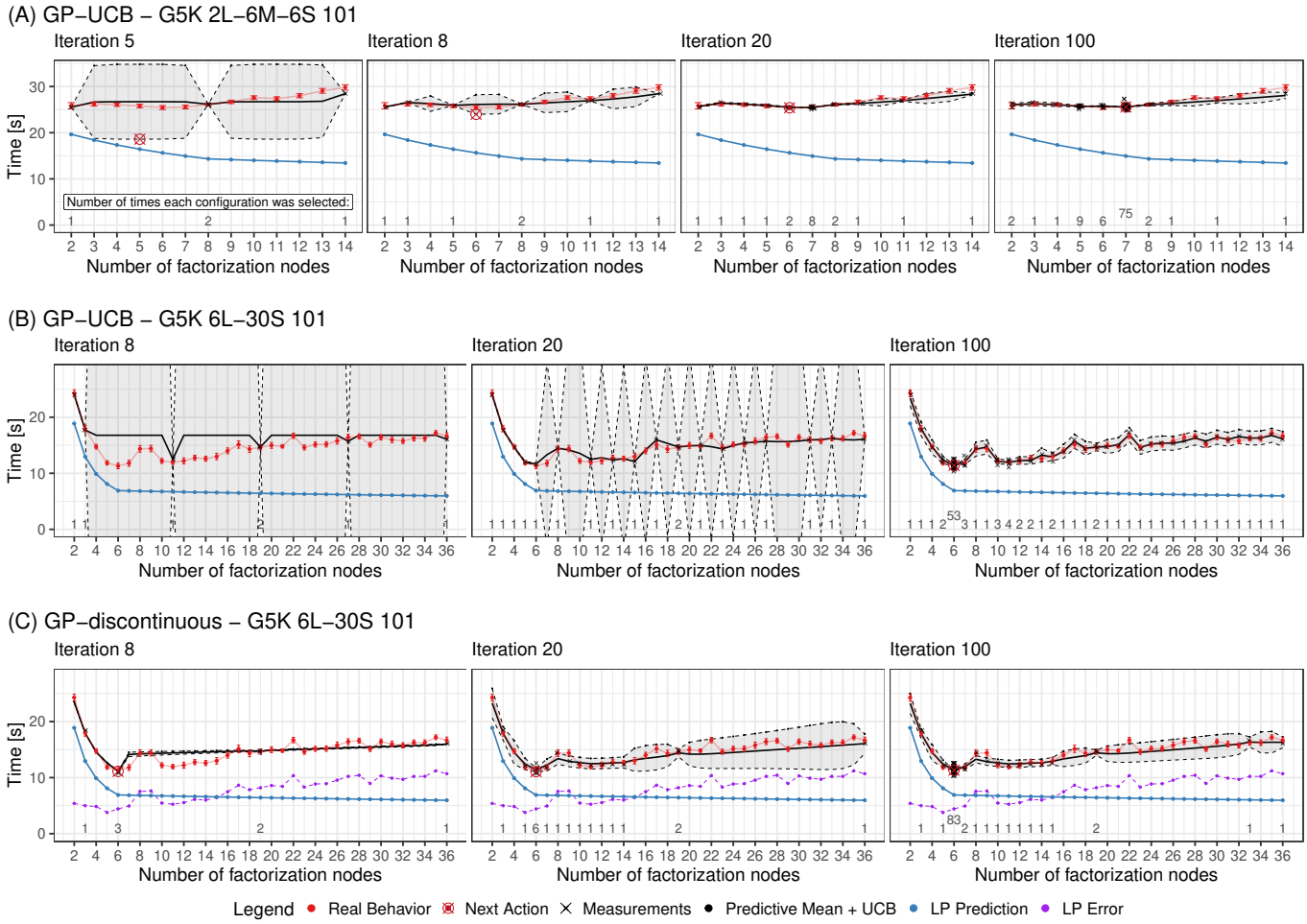

Fig. 4. Step-by-Step of (A) **GP-UCB** in G5K 2L-6M-6S 101, (B) **GP-UCB** in G5K 6L-30S 101, and (C) **GP-discontinuous** in G5K 6L-30S 101. The numbers indicated in the bottom of each graph indicates the number of times a given configuration has been selected so far.

B. Depicting the GP exploration/exploitation step-by-step

Figure 4 (A) illustrates the step-by-step process of the **GP-UCB** on the G5K 2L-6M-6S 101 scenario. The graphs show different iterations and the corresponding GP estimation. For each iteration, the solid red line with error bars shows the real data for each action with a 99% confidence interval. The blue lines show the LP estimation. The black line is the GP mean prediction, and the black dashed lines are the UCB component. The large red cross is the next action to take based on the current situation. The bottom of each graph shows the number of measurements taken with that number of nodes.

At Iteration 5 in Figure 4 (A), because of the experimental design's first points, there are two areas without measurements with a lot of uncertainty in the surrogate model. This area is further reduced, as shown in Iteration 8 when middle points were evaluated. At Iteration 20, there is already a convergence on finding the best points (action six or seven, which are very similar). This behavior is maintained until Iteration 100. However, it is interesting to note that GP-UCB keeps exploring as actions 5 and 6 continue to be evaluated sometimes but that some actions (10, 12, 13) are not even tried as they would clearly provide bad performance. In this simple scenario, the **GP-UCB** is enough to find the best configuration without assessing all the possible search space.

However, the **GP-UCB** does not behave so well in the scenario G5K 6L-30S 101 as shown in Figure 4 (B). There is a lot of uncertainty on Iterations 8 and 20 because the measurements scale is large, and the curve has some discontinuities that mislead the smooth GP and overestimate the scale parameters α and θ. Although it does find the best option by Iteration 100, it has explored all the points. Applying the most elaborate version, **GP-discontinuous**, solves these problems as shown in Figure 4 (C). One addition to the plot is the purple line which represents the difference between the real data and the LP. At Iteration 8, it is possible to check how this GP version models the discontinuities with the dummy variables. The model presents two distinctive curves, one until action six, and another after it, which is a very flat line on this iteration. On Iteration 20, the local minima zone from actions $10 - 14$ is already evaluated, improving the model accuracy. At this iteration, the surrogate model includes all the real values on its UCB component. On Iteration 100, it does find the best action

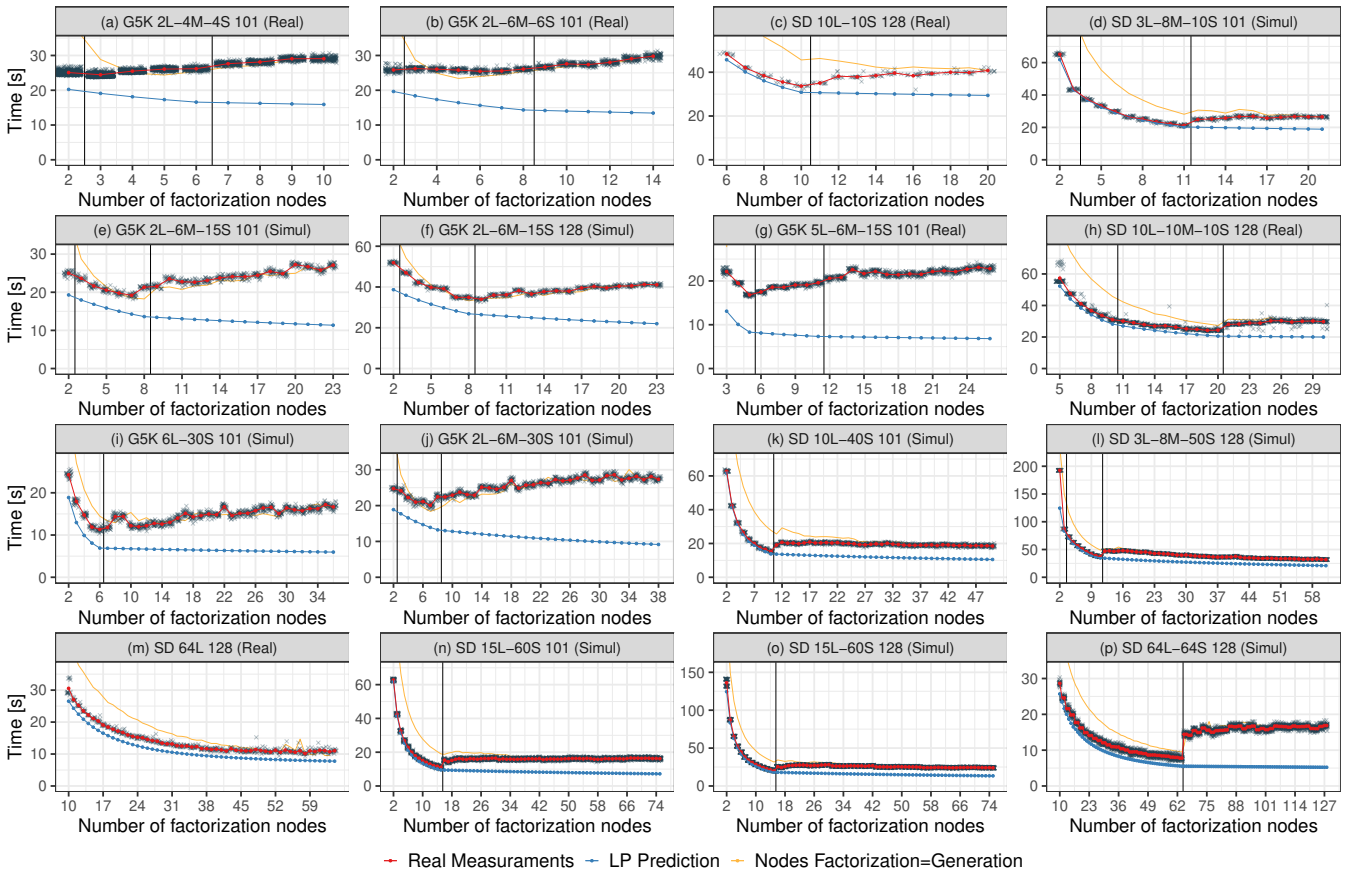

Fig. 5. Behavior using different heterogeneous nodes setups (Table II) by varying the number of factorization nodes.

without needing to measure all the points, skipping most of the right zone after action 20.

C. Behavior on different setups

The behavior of the problem can be further studied in setups formulated by varying the machines of Table II. Figure 5 presents the behavior (performance when adding nodes) of 16 setups in a similar way of Figure 2. One difference is the addition of a yellow line that shows the rigid situation where the same number of nodes is used for both the generation and the factorization steps. These setups present all the shapes we found even in situations not reported. Some scenarios present a very smooth behavior, like cases (a), (b), (e), (f), (m). Most of them are scenarios on G5K that have a limited network compared to SD. Another behavior is the presence of discontinuities when a new group of machines is introduced, like cases (d), (g), (h), (k), (l), (n), (o), (p). In these cases, adding a very slow node (CPU-only) caused problems on the critical path. Finally, some situations had small breaks related to the heterogeneous partition of that specific number of nodes (and occurring inside groups of the same types of machines), like cases (c), (e), (f), (g), (i), (j), (p).

D. Results Overview: GP and existing Exploration Strategies

Figure 6 depicts a performance evaluation overview of all studied exploration strategies with all scenarios. Each colored line is a group of strategies where the points are the makespan mean of 30 executions after 127 iterations. The top dashed horizontal line describes the performance when using all nodes, and the bottom one is the best option when knowing the best configuration upfront. The percentage for each strategy is the acceleration with respect to using all the machines. In what follows, we discuss the scenarios referencing Figures 5 and 6.

The **UCB** and **Right-Left** perform poorly in more than half of the scenarios. **UCB** requires a full exploration that degrades the overall performance. Typically bad configurations for this strategy have a large number of nodes or should use most nodes. In scenario (o), it is worse than using all the nodes (**UCB** point is above the top dashed line) because of inevitable bad points exploration. Moreover, **Right-Left** fails because it does not explore enough and may get stuck in local minima. For example, in Figure 5 (p), using all 128 factorization nodes is better than using 127, so it never explores the best action, which is 64. In (a), variability plays a role, and some **Right-Left** executions presented better results than others. The algorithm often stops too soon by bad luck because it is not resilient to variability.

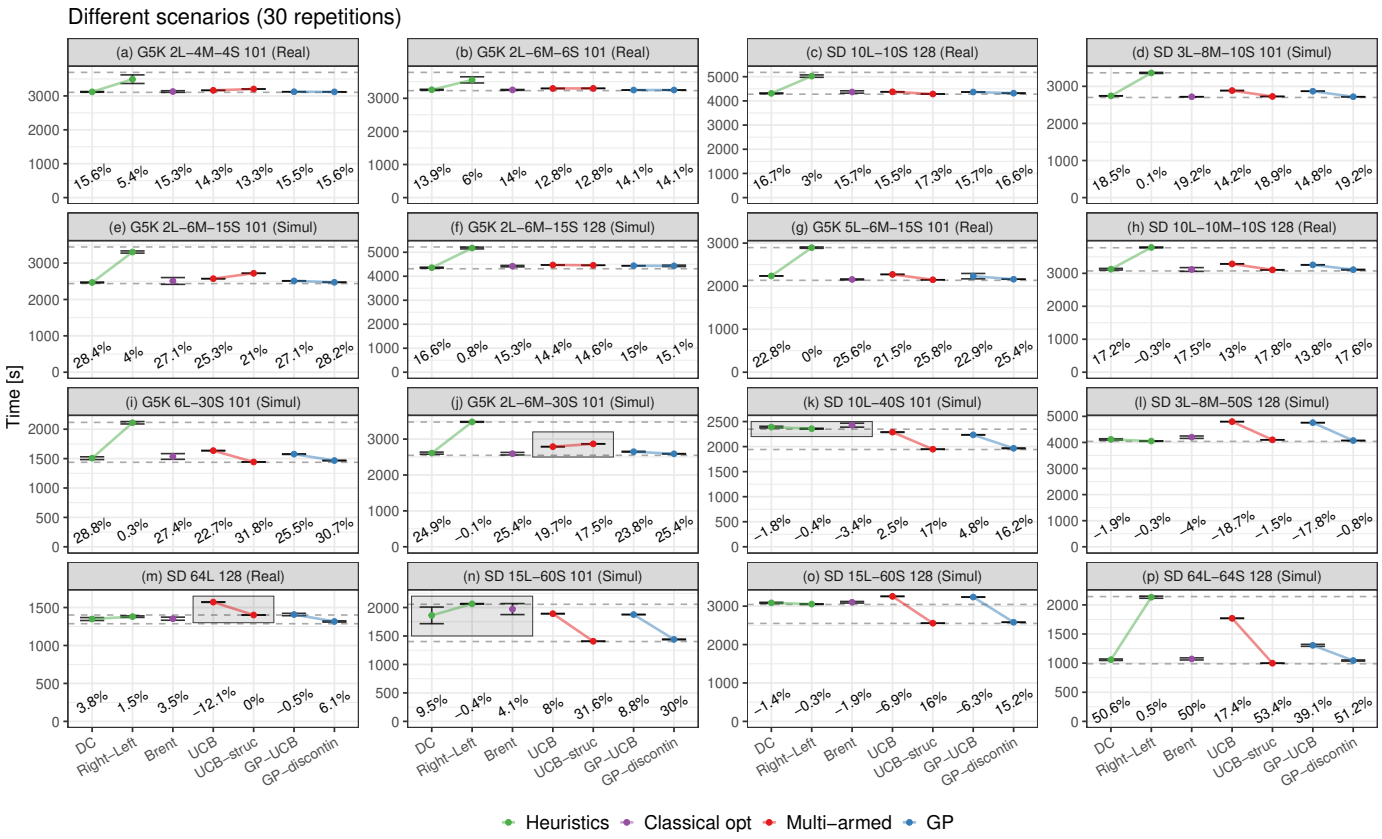

Fig. 6. Comparison of different methods in 16 scenarios. The Y-axis is the meantime of 30 executions after 127 iterations. The X-axis is the strategies, and each color is a different group of strategies. The number above the X-axis inside the plot is the percentage of gain compared with the standard approach of using all nodes (The top dashed line).

The simple divide and conquer (**DC**) algorithm does perform very well sometimes, finding the best option. Indeed, using this simple heuristic in most of the Figure 5 curves will work correctly. However, the variability of the measurements may trick the decisions, and in some scenarios like that of Figure 6 (n), it may perform very well or very badly depending on the observations. In these scenarios, making a wrong decision in any division moment may be enough to never find the best number of nodes. As shown in Figure 5 (n), the best point is in the left, but the method will never explore it if the right side is chosen in the first division.

The **Brent** strategy also does perform very well in a lot of scenarios, but in ones with discontinuity like Figure 5 (k), (n), and (o), it may not explore enough and get tricked into a local minimum. In such scenarios, because of the scale of the x-axis, it may only try points after the addition of one group of homogeneous machines. The Figure 5 (n) does have a large plateau corresponding to the 60 nodes of that group. Also, it is subject to the variability of the measurements, like in Figure 6 (e), (i), and (l). In these scenarios, it is possible to check that not all executions found the best number of nodes (the confidence interval is larger), resulting in a larger makespan in some executions.

The **GP-UCB** approach works well in more or less half of the scenarios. It does not always find the best option or requires a full exploration of the search space, as discussed in Section VI-B. It presents good acceleration in Figure 6 (a), (b), (c), (e), (f) and (j), which do not present discontinuities and have a small search space. However, scenarios like (k), (l), (m), (n), and (o) indicate that the GP requires extra modeling to handle more complex situations.

However, the **GP-discontinuous** performs very well in all scenarios. The introduction of trend, search space limits, and dummy variables allows quickly and dynamically reaching the optimal configuration with a very limited overhead compared to the lower bound obtained with a static clairvoyant choice. An example is Figure 4 (C), that presents the state of the GP-discontinuous strategy in (i). After the 20 iterations, the method already identified the optimal number of nodes, and that the surrogate model has a very good estimate of the response of the whole system as it comprises most of the (true) red line without having ever explored large regions of the space.

UCB-struct also performs very well in almost all scenarios. It has a minimal number of possible actions to check, and its algorithm is resilient to variability. Also, usually selecting few multiple groups of homogeneous machines is, or close to, the best option to take. However, there are scenarios where the best action to take is far from those points, and because this strategy would never explore or know these points, it cannot

find them. Examples are Figure 6 (a), (e), and (j). In these scenarios, the optimal choice (Figure 5) does not correspond to any of the vertical lines that mark the groups' transitions.

In short, the **GP-discontinuous** present good results in all scenarios, up to 51.2% speedup as shown in Figure 6 (p). In scenarios where using all the nodes is the best option (e.g., (l)), the final performance compared to using all notes is superior to -1%, meaning that the exploration overhead is low. All these results corroborate with the Table I expectations.

E. GP Computation Overhead Evaluation

The performance gains of using an exploration strategy should be smaller than the overhead of its computation. Figure 7 shows the overhead of the **GP-discontinuous** strategy on the scenario (b) G5K 2L-6M-6S, with ten repetitions of a real experiment running the GP online. Each black point represents the average overhead for that iteration. The overhead per iteration is almost constant in this model. The first iteration is longer than the others, and the successive four iterations are less expensive, as they do not perform the GP's computations. In the sixth and subsequent iterations, the DiceKriging package is called, resulting in a constant duration. The final overhead per iteration is negligible (0.04s – 0.06s) compared to the typical iteration total duration (10s – 30s). The cost is thus not of great concern compared to the risk of missing a learning opportunity.

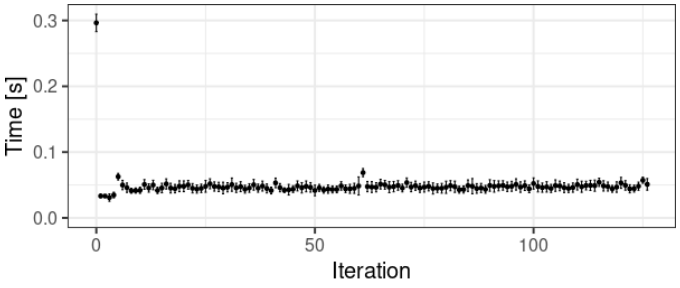

Fig. 7. Overhead of GP in function of iterations.

VII. Related Work

Some works study the problem of optimizing the best number of nodes for an HPC application and also describe related issues similar to the ones in Section III that corroborates to the problem. Applications such as Deep Neural Networks may present many challenges when strong scaling it on homogeneous nodes [25]. The issues range from communication overhead to the number of parallel operations. The number of nodes choice can also play an essential role in the performance and energy considerations. In some applications, it is possible to use more homogeneous nodes with slower frequencies and different voltages and improve performance while reducing energy consumption [26].

A study for HPC checkpoint fault-tolerant environments [27] also shows convex-like curves for makespan as a function of the number of homogeneous processes. They use Newton's Method to find the best number of processes to use. In another work [28], the authors used the GP to search for the best homogeneous cloud instance for a given application considering performance and cost. They restrict to very specific (powers of 2 from 1 to 32) subsets of homogeneous nodes. In all of these works, the focus was on using homogeneous resources, while this paper focuses on using heterogeneous resources together and considering iterative multi-phase applications such as ExaGeoStat.

VIII. Discussion and Conclusion

This paper proposes and analyzes different exploration methods to find the best collection of heterogeneous nodes for an HPC application phase. The ExaGeoStat application we consider, with partial phases overlapping and multiple specialized distributions, had already been very optimized for heterogeneous resources [4]. However, determining the ideal number of nodes per phase was an open and challenging problem. Indeed, the network and the distribution of data over heterogeneous nodes may cause unpredictable and unexpected behavior during the execution of the application on a particular number of nodes. The application should thus explore various configurations and adapt online to discover the optimal subset of nodes for a given scenario.

Our results show that an informed Gaussian-Process-based reinforcement learning strategy can quickly find the best configuration of the main phase (factorization). The superior approach, **GP-discontinuous**, uses bounding mechanisms to filter the search space, model the difference of the makespan to a lower bound (and already have some knowledge of the scenario), and use dummy variables and a linear trend to model discontinuities caused by the distributions. This method provided the consistent best results on all 16 studied scenarios. Another (simpler) method that performs particularly well is **UCB-struct** which only considers specific points. However, this method will not always find the best configuration, as it is constrained not to search all the space but only the multiple complete homogeneous groups. In the end, using all the nodes for the generation phase and only a learned subset for the factorization provided up to 51.2% speedup compared to using all the nodes for both phases.

For most scenarios, using all the machines for the first (generation) phase is the best option. However, one of the investigated scenarios shows that using too many nodes for the generation can also result in a slowdown. Figure 8 presents the iteration duration (as a colored gradient) when varying both the number of generation nodes and the number of factorization nodes for (f) G5K 2L-6M-15S 128. In this scenario, using 10 generation nodes and 8 factorization nodes is better than using all 23 generation nodes (23/9) by 1.1 seconds ($\approx$3% less). In this type of situation, the problem should then be explored in both dimensions. Although the GP exploration should gracefully extend to more dimensions, we believe the gain will be limited in practice.

Future work includes the modeling of the 2D space considering both phases, as there are some scenarios that using all the nodes for the generation also degrades performance

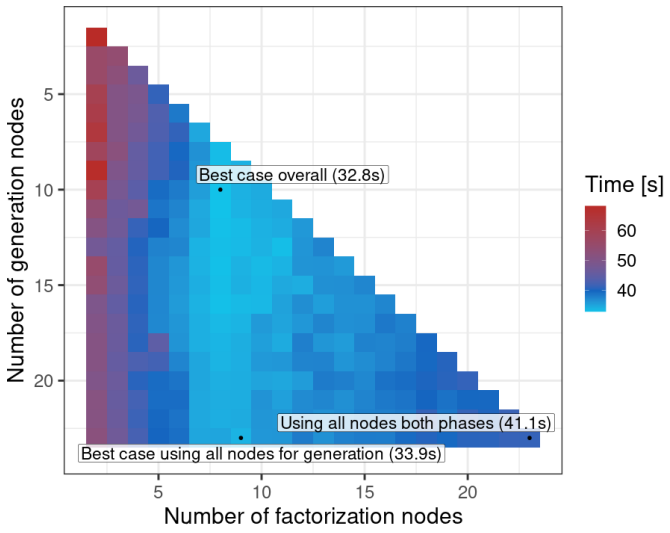

Fig. 8. Iteration makespan with different number of generation and factorization nodes.

(as shown in Figure 8). In this situation, the number of possible actions increases. Moreover, further investigation is required to propose or adapt the GP strategies to non-stationary scenarios. Also, dynamically adapting methods can play a role in other applications' parameters. For example, ExaGeoStat can run the factorization with mixed precision blocks. The application could dynamically adjust the number of diagonals that use each precision in a trade-off between accuracy and performance. Finally, we intend to evaluate other iterative multi-phase applications.

Acknowledgements

Some experiments were carried out using Grid'5000, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (https://www.grid5000.fr). The authors acknowledge the National Laboratory for Scientific Computing (LNCC/MCTI, Brazil) for providing HPC resources of the SDumont supercomputer, which have contributed to the research results reported in this paper (http://sdumont.lncc.br).

References

[1] J. J. Dongarra, H. W. Meuer, E. Strohmaier *et al.*, "Top500 supercomputer sites," *Supercomputer*, vol. 13, pp. 89–111, 1997.

[2] E. Roloff, M. Diener, L. Gaspary, and P. Navaux, "Exploring Instance Heterogeneity in Public Cloud Providers for HPC Applications," in *Proceedings of the 9th International Conference on Cloud Computing and Services Science - CLOSER*. SciTePress, 2019, pp. 210–222.

[3] J. Herrmann *et al.*, "Assessing the cost of redistribution followed by a computational kernel: Complexity and performance results," *Parallel Computing*, vol. 52, 2016.

[4] L. L. Nesi, A. Legrand, and L. M. Schnorr, "Exploiting system level heterogeneity to improve the performance of a geostatistics multi-phase task-based application," in *50th International Conference on Parallel Processing*, ser. ICPP. New York, NY, USA: ACM, 2021.

[5] L. L. Nesi, L. M. Schnorr, and A. Legrand, "Communication-aware load balancing of the LU factorization over heterogeneous clusters," in *26th IEEE International Conference on Parallel and Distributed Systems, ICPADS 2020*. Hong Kong: IEEE, 2020, pp. 54–63.

[6] J. J. Dongarra *et al.*, "With extreme computing, the rules have changed," *Computing in Science & Engineering*, vol. 19, no. 3, p. 52, 2017.

[7] C. Augonnet *et al.*, "StarPU: A unified platform for task scheduling on heterogeneous multicore architectures," *Concurrency and Computation: Practice Experience, SI: EuroPar'09*, vol. 23, pp. 187–198, 2011.

[8] E. Agullo, A. Buttari, A. Guermouche, and F. Lopez, "Implementing multifrontal sparse solvers for multicore architectures with sequential task flow runtime systems," *ACM Tr. Math. Softw.*, vol. 43, no. 2, 2016.

[9] S. Abdulah, H. Ltaief, Y. Sun, M. G. Genton, and D. E. Keyes, "Exageostat: A high performance unified software for geostatistics on manycore systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 12, pp. 2771–2784, 2018.

[10] E. Agullo *et al.*, "Faster, cheaper, better – a hybridization methodology to develop linear algebra software for GPUs," in *GPU Computing Gems*, W. mei W. Hwu, Ed. Morgan Kaufmann, Sep. 2010, vol. 2.

[11] V. Garcia Pinto *et al.*, "A visual performance analysis framework for task-based parallel applications running on hybrid clusters," *Concurrency and Computation: Practice and Experience*, vol. 30, no. 18, p. e4472, 2018.

[12] L. L. Nesi, S. Thibault, L. Stanisic, and L. M. Schnorr, "Visual performance analysis of memory behavior in a task-based runtime on hybrid platforms," in *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 2019, pp. 142–151.

[13] O. Beaumont, V. Boudet, F. Rastello, and Y. Robert, "Matrix multiplication on heterogeneous platforms," *IEEE Trans. Parallel Distributed Systems*, vol. 12, no. 10, pp. 1033–1051, 2001.

[14] O. Beaumont, A. Legrand, F. Rastello, and Y. Robert, "Static LU decomposition on heterogeneous platforms," *Int. Journal of High Performance Computing Applications*, vol. 15, pp. 310–323, 2001.

[15] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2021. [Online]. Available: https://www.R-project.org/

[16] H.-F. Chen, *Stochastic approximation and its applications*. Springer Science & Business Media, 2006, vol. 64.

[17] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[18] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine Learning*, vol. 47, pp. 235–256, 2002.

[19] R. B. Gramacy, *Surrogates: Gaussian Process Modeling, Design, and Optimization for the Applied Sciences*, ser. Chapman & Hall/CRC Texts in Statistical Science. United States: CRC Press, 2020.

[20] N. Srinivas, A. Krause, S. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ser. ICML'10. Madison, WI, USA: Omnipress, 2010, p. 1015–1022.

[21] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An introduction to statistical learning*. Springer, 2013, vol. 112.

[22] O. Roustant, D. Ginsbourger, and Y. Deville, "DiceKriging, DiceOptim: Two R packages for the analysis of computer experiments by kriging-based metamodeling and optimization," *Journal of Statistical Software*, vol. 51, no. 1, pp. 1–55, 2012.

[23] L. Stanisic *et al.*, "Faithful performance prediction of a dynamic task-based runtime system for heterogeneous multi-core architectures," *Conc. Comp.: Pract. Exp.*, vol. 27, no. 16, pp. 4075–4090, 2015.

[24] A. Denis, "Scalability of the NewMadeleine Communication Library for Large Numbers of MPI Point-to-Point Requests," in *19th Int. Symp. in Cluster, Cloud, and Grid Comp.* IEEE, 2019, pp. 371–380.

[25] J. Keuper and F.-J. Preundt, "Distributed training of deep neural networks: Theoretical and practical limits of parallel scalability," in *2nd WS on Machine Learning in HPC Environments*, 2016, pp. 19–26.

[26] V. W. Freeh, D. K. Lowenthal, F. Pan, N. Kappiah, R. Springer, B. L. Rountree, and M. E. Femal, "Analyzing the energy-time trade-off in high-performance computing applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 6, pp. 835–848, 2007.

[27] H. Jin, Y. Chen, H. Zhu, and X.-H. Sun, "Optimizing HPC Fault-Tolerant Environment: An Analytical Approach," in *2010 39th International Conference on Parallel Processing*, 2010, pp. 525–534.

[28] V. Rosario, T. Camacho, O. Napoli, and E. Borin, "Fast and low-cost search for efficient cloud configurations for hpc workloads," in *Anais do XXII Simpósio em Sistemas Computacionais de Alto Desempenho*. Porto Alegre, RS, Brasil: SBC, 2021, pp. 144–155.