Accordia: Adaptive Cloud Configuration Optimization for Recurring Data-Intensive Applications

ABSTRACT

Recognizing the diversity of big data analytic jobs, cloud providers offer a wide range of virtual machine (VM) instances or even clusters to cater for different use cases. The choice of cloud instance configurations can have a significant impact on the response time and running cost of data-intensive, production batch-jobs, which need to be re-run regularly using cloud-scale resources. However, identifying the best cloud configuration with a low search cost is quite challenging due to i) the large and high-dimensional configuration-parameters space, ii) the dynamically varying price of some instance types (e.g. spot-price ones), iii) job execution-time variation even given the same configuration, and iv) gradual drifts / unexpected changes of the characteristics of a recurring job. To tackle these challenges, we have designed and implemented Accordia, a system that enables Adaptive Cloud Configuration Optimization for Recurring Data-Intensive Applications. By leveraging recent algorithmic advances in Gaussian Process UCB (Upper Confidence Bound) techniques, the design of Accordia can handle time-varying instance pricing while providing a performance guarantee of *sub-linearly increasing regret* when comparing with the static, offline optimal solution. Using extensive trace-driven simulations and empirical measurements of our Kubernetes-based implementation, we demonstrate that Accordia can dynamically learn a near-cost-optimal cloud configuration (i.e. within 10% of the optimum) after fewer than 20 runs from over 7000 candidate choices within a 5-dimension search space, which translates to a 2X-speedup and a 20.9% cost-savings, when comparing to CherryPick.

CCS CONCEPTS

• **Computer systems organization → Cloud computing**; • **Mathematics of computing** → *Stochastic processes*; *Mathematical optimization.*

KEYWORDS

Big data analytics, Cloud configuration, Gaussian-Process UCB, Kubernetes

1 INTRODUCTION

Nowadays, the number of big data applications running in cloud environments, such as MapReduce [34], Spark [1], SQL queries [2, 3] and Deep learning [5] are increasing rapidly. The job profiles are also becoming more diverse and the number of VM instances required by different jobs can vary substantially. For example, users can specify the number of mappers and reducers themselves when running MapReduce applications in a cloud. Meanwhile, the resource configuration for each mapper or reducer task, such as CPU, RAM, disk-space and network bandwidth can differ a lot across applications. To better support the diverse nature of the computation needs of their users, public cloud providers [6, 7] have offered a large number of types of VM instances or even clusters to cater for different use cases. As such, cloud-service customers can have great flexibility to select the cloud configuration for each specific application. The choice of cloud configurations can have a significant impact on the response time and the overall running cost of big data analytics jobs. As reported in [33], choosing configurations poorly can substantially degrade the response time by 5x-6x on average, and may increase the cost by 10x in the worst-case. It is particularly important for recurring data-intensive jobs to select a matching cloud configuration as they need to be re-run many times on a regular basis, e.g., daily log parsing as well as hyper-parameter-tuning for Deep Learning jobs. In this paper, we focus on the effective search of the best cloud configuration for recurring batch jobs to minimize the job running cost while meeting its completion deadline. The running cost here is defined as the sum of the costs charged by a cloud for holding all the instances until a job completes. Besides this, we also want to keep the job execution time to be shorter than a given threshold.

An efficient searching scheme will yield a small number of runs before identifying the best configuration for a recurring job. However, it is quite challenging to achieve this goal without careful design. Firstly, the running time of a big data analytics job is affected by the amount of configured resources in an unpredictable way. Increasing the amount of resources by selecting a more powerful type of VMs may only result in a marginal performance gain. In general, the relationship between resource-cost and job performance is non-trivial (e.g. non-linear and multi-modal) and difficult to characterize using a simple well-formed function. Secondly, since the computing resources in a cloud are often shared by multiple users/ jobs with non-perfect separation/ isolation, stragglers can happen easily [10]. As such, the running time of the same job/ application under the same cloud configuration may still suffer from a large variance, which makes performance prediction much more complex. Thirdly, public cloud service providers, such as AWS, have offered the so-called of Spot Instances Pricing [14] with typical savings of 70-90% over On-Demand instance types to make their service/ pricing more competitive. As such, there is a high incentive and essential to leverage the spot-instance model by considering and predicting the time-varying behavior of different configuration options. Besides choosing the instance types provided in a cloud, users have another flexibility to configure the number of instances for holding different parts of a job (e.g., the Map and Reduce phase). As a result, the configuration space can consist of thousands of choices even for a simple big data analytics job. Exhaustively searching for the best one without relying on an accurate prediction mechanism can lead to an unacceptable overhead. If not designed properly, one may need hundreds of test runs to identify a suitable cloud configuration, which is quite time-consuming and costly for big data jobs as each run would require a long time to finish.

In this paper, we discuss the design and implementation of Accordia, a system which enables Adaptive Cloud Configuration Optimization for Recurring Data-Intensive Applications. Accordia adopts the Gaussian-Process based (GP) approach to model the cloud configuration optimization problem. In particular, it relies on the Gaussian-Process model to estimate the mean and variance of job completion time for the not-yet observed configurations and calculate the first-order increment between two successive prices in the last two runs to predict the timely price of VM instances. Based on the estimated job completion time and VM instances price, Accordia uses the Upper Confidence Bound (UCB) algorithm to select a valuable configuration from the candidate set for the next run. To further reduce the overhead caused by those less-valuable runs, Accordia supports an early abort mechanism, under which the system will abort a partially-run job and seek another alternative cloud configuration if the predicted job completion cost is much larger than the best solution observed so far. Even though other state-of-the-art schemes, such as CherryPick[9] and Arrow[18], have also taken the Gaussian-Process based approach, Accordia goes further by leveraging and extending the recent advances in Gaussian-Process Upper Confidence Bound (GP-UCB) techniques [23, 30] which enable Accordia to handle time-varying instance pricing and provide a performance guarantee of *sub-linearly increasing regret* when comparing with the static, offline optimal solution.

To evaluate the performance of Accordia, we have conducted extensive trace-driven simulations and empirical measurements on our actual Kubernetes-based implementation of Accordia under the AWS cloud with fixed and time-varying instance prices. For the simulation study, we consider the 16 different real-world application traces collected by [18–20], which are mostly large-scale data-intensive applications, and use the same candidate set of cloud configurations as that in CherryPick. The experimental results show that Accordia can find a near-cost-optimal configuration (i.e. within 10% of the optimal cost) within 6 test runs for most cases, whereas CherryPick needs around 10 runs to do so. For the empirical evaluation of our actual implementation, we have run different mixes of recurring Spark jobs over the Google public cloud. In our experiments, Accordia dynamically learns the best cloud configuration from over 7000 candidate choices within a 5-dimensional parameter space, covering the number of executors, as well as the number of CPU cores and memory (RAM) allocation for the driver and the executor pods. Empirical measurements show that Accordia can find a near-cost-optimal configuration for a recurring job with fewer than 20 runs, which translates to a 2X-speedup and a 20.9% cost-savings when comparing to CherryPick. To highlight Accordia's capability to handle abrupt/unexpected changes of the characteristics of a recurring job, we even dynamically switch the type of a recurring job (without notifying Accordia) over exponentially-distributed time-intervals. Under such cases, Accordia can still obtain up to 18.6% cost-savings comparing to CherryPick.

The rest of the paper is organized as follows. After Section 2 introduces the literature work related to Accordia, Section 3 presents the optimization framework of Accordia and Section 4 discusses the solution approach. Then Section 5 evaluates the performance of Accordia by trace-driven simulations. Accordia is also implemented on top of Kubernetes and evaluated by empirical measurements in Section 6 and Section 7. Finally, we conclude the paper in Section 8.

2 RELATED WORK

In recent years, many related efforts try to handle the cloud configuration selection problem. Some heuristic dynamic allocation algorithms, such as dynamic allocation [4] in Apache Spark and DS2 [21] in Apache Flink, have been implemented in distributed platforms. The dynamic allocation of Apache Spark allows to automatically remove the idle executors and exponentially increase the number of executors for pending tasks in the application. And DS2 [21] in Apache Flink is an automatic scaling controller which can maximize the executors' processing and output rate based on the knowledge of computational dependencies. These kinds of dynamic allocation algorithms are based on heuristic arguments and lack of modeling or analytic effort.

Supervised machine learning strategies also have been applied to predict the performance of the candidate cloud configurations. For example, Paris [32] and Wrangler [31] collect and run a set of benchmarking jobs to train the random forest and SVM model, then use the representation task of a given application to predict the running time and cost for a set of candidate cloud configurations. However, the accuracy of supervised learning algorithms heavily depends on the quality and amount of training data, which is hard to collect and leads to heavy overhead in the offline benchmarking phase. It is even worse when the supervised learning models like Paris and Wrangle need to re-train every 10 hours.

Selecta [22] uses latent factor collaborative filtering to predict the performance of a given application for a set of candidate cloud configurations, based on sparse data collected in the offline benchmarking phase. Latent factor collaborative filtering is a kind of recommendation system algorithm, which can capture the performance correlation across the applications and cloud configuration. It can suffer from less amount of overhead than the supervised machine learning algorithms. However, this kind of overhead can be avoided by an online learning based approach like ours.

LinkedIn [24] and Saboori A et. al [28] use genetic and evolutionary algorithms to search the optimal cloud configurations for Hadoop applications and three-tier web systems. Genetic and evolutionary algorithms are online algorithms without the offline overhead and commonly used to search the optimum for black-box optimization problems. Under genetic and evolution algorithms, cloud configurations with better performance will have a higher probability to survive and evolute in the following selection. However, in practice, measuring performance is often expensive. It may take hours or days to complete the recurring big data analytics applications. This discards the genetic and evolutionary algorithms, as they usually require hundreds or thousands of evaluations to converge.

Bayesian optimization [11] [29] is also a popular online framework to search the optimum for black-box optimization problems. It models objective function drawn from a statistic process and keeps updating the posterior distribution to compute the confidence interval, the range of values that the objective function is most likely to fall into. CherryPick [9], Arrow [18] and BOAT [16] assume the cost of candidate cloud configurations following the Gaussian Process and use the EI algorithm, selecting the next round cloud configuration with the maximum expected improvement, to handle the Bayesian optimization problem. CherryPick directly uses the

cloud configuration as the parameter to unearth the cheapest cloud configuration to complete the application. Arrow combines cloud configuration and low-level running time information (e.g., CPU utilization and memory utilization) to accelerate the search. And BOAT leverages the contextual information, the human understanding of the application's behavior, to handle the high dimensions selection case. However, the performance of the traditional Bayesian optimization based algorithms, e.g. EI algorithm and PI algorithm, heavily depends on the choice of initial samples and may suffer from a long search with bad initial sampling.

Cao Z. et. al [13] compare the performance among different selection algorithms, including simulated annealing, genetic algorithm, Deep Q-Networks and Bayesian optimization EI algorithm. In most of the cases, the Bayesian optimization based algorithm is the fastest algorithm to find the optimal cloud configuration. It is also the most similar method of our work. However, all of the above algorithms assume the per-unit price of cloud configuration is time-invariant and can not provide a performance guarantee. In our system model, we go further to consider the time-varying price case and achieve sub-linearly increasing regret compared to the static optimal solution.

3 ANALYTICAL MODEL FOR ACCORDIA

Accordia focuses on recurring data-intensive applications. The design goal of Accordia is to identify the optimal or a near-optimal cloud configuration that satisfies the job completion time requirement and in the meanwhile, minimizes the total execution cost. To overcome the offline overhead, we adopt the online optimization framework for the Accordia system as shown in Figure 1. Accordia starts with an initial cloud configuration, runs it, and feeds the cloud configuration details and job completion time into the performance model, which can base on the existing observation to estimate the confidence interval of job completion time for candidate cloud configurations. Then Accordia sequentially selects the next-run cloud configuration based on the current understanding. In this section, we carefully present this online optimization framework under Accordia in both the fixed and time-varying price cases.

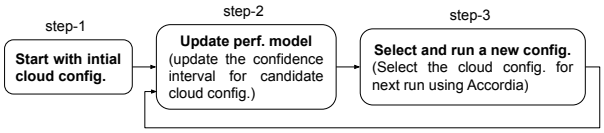


Figure 1: Accordia Workflow

3.1 Online Selection Problem under Fixed Pricing of the Cloud Service

Consider a public computing cloud that consists of hundreds of instance types. The price of each instance type is fixed at any time. Users can request resources for any type, so as to fit their requirements. Besides configuring the cloud resource types, users have certain flexibility to configure the number of instances running different parts of a big data analytics job. For instance, users may want to specify the number of instances running for Map phase and Reduce phase individually.

Formally, let $g \mid \boldsymbol{x} \rightarrow \mathbb{R}$ define the job completion time function for a given recurring big data application. The job completion time depends on the cloud configuration vector $\boldsymbol{x}$, which includes the number of instances for each portion of the job, number of CPU cores, memory size, network interface, network bandwidth and other resource configurations for each instance. All the candidate cloud configurations form a search space $\mathcal{X}$. Moreover, denote by $p \mid \boldsymbol{x} \rightarrow \mathbb{R}$ the function which captures the per unit time cost for all instances under cloud configuration $\boldsymbol{x}$.

Accordia aims to find the best cloud configuration that minimizes the total running cost, which yields the following optimization problem (P1):

$$\begin{aligned} \min_{\boldsymbol{x}} \ & g(\boldsymbol{x}) \cdot p(\boldsymbol{x}) \\ s.t. \ & g(\boldsymbol{x}) \leq T_{\max}, \\ & \boldsymbol{x} \in \mathcal{X}, \end{aligned} \tag{1}$$

where $g(\boldsymbol{x}) \cdot p(\boldsymbol{x})$ characterizes the total cost for running a job under the cloud configuration $\boldsymbol{x}$. And $T_{\max}$ is the maximum running time that can be tolerable (i.e., the deadline). In other words, P1 is essential to obtain the most economic cloud configuration $\boldsymbol{x}^*$ as long as as the running time of $\boldsymbol{x}^*$ is upper bounded by $T_{\max}$. Mathematically, $\boldsymbol{x}^*$ is given by:

$$\boldsymbol{x}^* = \arg \min_{x \in \mathcal{X}} g(\boldsymbol{x}) \cdot p(\boldsymbol{x}), \ \ s.t.\ g(\boldsymbol{x}) \leq T_{\max}. \tag{2}$$

3.1.1 Multi-armed bandit framework. Since g is not known in advance, we need to run multiple test runs before we identify $\boldsymbol{x}^*$. However, exhaustively searching for $\boldsymbol{x}^*$ is not scalable since $\mathcal{X}$ may consist of a huge number of candidate configurations, especially when $\boldsymbol{x}^*$ is a high dimension vector. To avoid this, we need to design a searching scheme that yields a small number of runs for identifying $\boldsymbol{x}^*$.

To design an efficient searching algorithm, we reformulate P1 using the Multi-armed Bandit (MAB) framework where each configuration is treated as one arm [12]. To be specific, for a given recurring job, we sequentially select cloud configurations to optimize the objective function. In each round t, we select a cloud configuration $\boldsymbol{x}_t$ and run the recurring job as a test. The job completion time is observed only after a job completes. And the observation is perturbed by a random noise: $g_t = g(\boldsymbol{x}_t) + \epsilon_t$, where $\epsilon_t \sim N(0, \sigma^2)$ follows a Gaussian distribution. Based on the current observation and understanding, we select the next round cloud configuration $\boldsymbol{x}_{t+1}$ and deploy it in the cloud until reaching the stop condition. The stop condition aims to prevent Accordia from struggling to make small improvements in the fixed price case. The pseudo-code of the MAB framework is exhibited in Algorithm 1.

3.1.2 Reformulation of P1. Regarding the performance of a searching algorithm, we adopt static regret [26] as the evaluation metric. Static regret is a commonly-used metric in the MAB literature. In our case, it captures the cumulative difference between the cost of all runs achieved under Accordia and that under the optimal configuration $\boldsymbol{x}^*$. The existence of regret is due to the lack of knowledge about $g(\boldsymbol{x})$ beforehand.

To better fit into the conventional definition of regret, we redefine a new function $f \mid \boldsymbol{x} \rightarrow \mathbb{R}$ to denote the job completion ratio, i.e., the multiplicative inverse of job completion time where $f = 1/g$. Under this new definition, P1 transforms to the following maximization

Algorithm 1 Online Searching Framework under the fixed-price case

Input: $p(\boldsymbol{x})$: price function;
$\boldsymbol{x}_1$: initial cloud configuration;
T_{max}: maximum tolerated running time;
$\mathcal{X}$: search space;
Output: optimal cloud configuration $\boldsymbol{x}^*$

1: Use the initial cloud configuration $\boldsymbol{x}_1$ to run the recurring job;
2: Observe the perturbed job completion time $g_1 = g(\boldsymbol{x}_1) + \epsilon_1$;
3: Store $\boldsymbol{x}_1, g_1$;
4: **repeat** for round $t = 2, 3, \cdots$
5: Select the current time-slot cloud configuration $\boldsymbol{x}_t$ based on the current observation and understanding;
6: Use the cloud configuration $\boldsymbol{x}_t$ to run the job;
7: Observe the perturbed job completion ratio value g_t;
8: **until** Reach the stop condition;

problem (P2):

$$\begin{aligned} \max_{\boldsymbol{x}} \ & f(\boldsymbol{x})/p(\boldsymbol{x}) \\ s.t. \ & f(\boldsymbol{x}) \geq 1/T_{\max}. \\ & \boldsymbol{x} \in \mathcal{X}. \end{aligned} \tag{3}$$

Then, the static regret Reg_T^s is defined as follows:

$$\text{Reg}_T^s = \sum_{t=1}^{T} f(\boldsymbol{x}^*)/p(\boldsymbol{x}^*) - \sum_{t=1}^{T} f(\boldsymbol{x}_t)/p(\boldsymbol{x}_t), \tag{4}$$

where T is the total number of test runs. A selection algorithm which can achieve a small sub-linearly increasing regret with respect to time T is preferable. For instance, a regret bound of $O(\log T)$ means the difference between $f(\boldsymbol{x}_t)$ and $f(\boldsymbol{x}^*)$ is only $\log T/T$, which converges quickly to zero when T increases. For this case, we can identify the optimal configuration, i.e., $\boldsymbol{x}^* = \arg\max_{\boldsymbol{x}} f(\boldsymbol{x})/p(\boldsymbol{x})$, with a very few number of runs.

Remark 1. *With the per dollar job completion ratio as the performance metric, we do not need to guarantee that the recurring job has been completed at each run, as long as we can get a good estimation for a given cloud configuration. It means that we may not need to waste money on some obviously poor cloud configuration. In our implementation, Accordia supports an early abort mechanism, under which the system will abort (i.e., kill) a partially-run job if its per dollar job completion ratio is 30% less than the largest one observed so far.*

3.2 Online Selection Problem under Time-varying Pricing of the Cloud Service

When considering a dynamic cloud where the price of an instance changes over time (e.g., Amazon EC2 spot instance), we use $p_t \mid \boldsymbol{x} \rightarrow \mathbb{R}$ to capture the price per-unit time for the t-th round. Paralleling P1, in each round t, we need to solve the following online optimization problem (P2):

$$\begin{aligned} \max_{\boldsymbol{x}} \ & f(\boldsymbol{x})/p_t(\boldsymbol{x}) \\ s.t. \ & f(\boldsymbol{x}) \geq 1/T_{max}, \\ & \boldsymbol{x} \in \mathcal{X}. \end{aligned} \tag{5}$$

Since the price of cloud instances is not fixed, the optimal solution to P2 also changes over time and may not converge. We need to remove the stop condition and keep searching. It is essential to trace the most economic cloud configuration $\boldsymbol{x}_t^*$, as characterized in the below.

$$\boldsymbol{x}_t^* = \arg\max_{\boldsymbol{x}} f(\boldsymbol{x})/p_t(\boldsymbol{x}), \ \ s.t. \ f(\boldsymbol{x}) \geq 1/T_{\max}. \tag{6}$$

In order to dynamically choose the optimal cloud configuration, we need to predict both the computing ratio and the time-varying price of the candidate cloud configurations. Both of them are unknown before making decisions. As such, we extend the static regret in Eq.(4) to the dynamic one for evaluating the performance of an online selection method [25]. In particular, the dynamic regret in our case is given by:

$$\text{Reg}_T^d = \sum_{t=1}^{T} f(\boldsymbol{x}_t^*)/p_t(\boldsymbol{x}_t^*) - \sum_{t=1}^{T} f(\boldsymbol{x}_t)/p_t(\boldsymbol{x}_t). \tag{7}$$

Remark 2. *The price function $p_t(\boldsymbol{x})$ for all the candidate cloud configurations can be observed one time-slot later, which is the online optimization setting. However, the job completion ratio $f(\boldsymbol{x})$ only can be observed for already tested cloud configuration, which is the bandit setting. Our online selection problem under the time-varying price case is the hybrid online optimization and multi-armed bandit problem.*

4 DESIGN DETAILS OF ACCORDIA

In this section, we present the design of Accordia. At a high level, Accordia mainly includes two parts, namely, how to pick the next-round configuration and when to stop the searching. For the first part, we shall make use of the Gaussian-Process UCB method to solve the multi-armed bandit online selection problem, in both the fixed and time-varying price cases. For the second part, we adopt machine learning algorithms to decide when to early stop the searching.

4.1 Insights of the Gaussian-Process UCB Approach

Towards this end, at each round, our algorithm needs to pick the next-round cloud configuration and balance exploitation (i.e., regions with a high probability of containing optimum) and exploration (i.e., regions with high uncertainty of containing optimum). We use the Upper Confidence Bound(UCB) [12] algorithm to handle the exploitation v.s. exploration dilemma, which is the state-of-the-art algorithm to deal with the multi-armed bandit problem and can be analyzed clearly. The basic idea behind the UCB algorithm is that we should explore the arms more often if they are promising or not well-explored. Specifically, inadequate exploration reduces the chance of moving away from a local optimum, and inadequate exploitation impacts the efficiency of identifying the global optimum.

For the recurring big data analytics jobs, job completion time with similar cloud configurations is highly correlated. We should not waste too many resources to explore similar cloud configurations. In this case, We add the smoothness assumption on the job completion ratio function $f(\boldsymbol{x})$, that $f(\boldsymbol{x})$ is sampled from a

Gaussian process [27]. In this assumption, the job completion ratios have a high probability to be close, if they have similar cloud configurations.

The Gaussian-Process UCB algorithm [23], combining the UCB algorithm and Gaussian process model, can capture the high correlation across the cloud configurations and balance exploitation v.s. exploration. In our work, we extend the Gaussian-Process UCB algorithm to handle the online selection problem for both the fixed and time-varying price cases.

4.2 Gaussian-Process UCB Algorithm for the Fixed Price Case

In our system, we assume the job completion ratio function $f(\boldsymbol{x})$ sampled from a Gaussian process (GP): a collection of dependent random variables, one for each $x \in \mathcal{X}$, every finite subset of which is multivariate Gaussian distributed in an overall consistent way [27]. A $GP(\mu(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x'}))$ is specified by its mean function $\mu(\boldsymbol{x}) = \mathbb{E}[f(\boldsymbol{x})]$ and covariance (or kernel) function $k(\boldsymbol{x}, \boldsymbol{x'}) = \mathbb{E}[(f(\boldsymbol{x}) - u(\boldsymbol{x}))(f(\boldsymbol{x'}) - u(\boldsymbol{x'}))]$.

A major advantage of working with GP is the existence of simple analytic formula for mean and covariance of the posterior distribution. For noisy samples $\boldsymbol{y}_t = \{y_1, y_2, \cdots, y_t\}$ at points $\boldsymbol{A}_t = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_t\}$, the posterior over f is a GP distribution again, with mean $u_t(\boldsymbol{x})$ and variance $\sigma_t^2(\boldsymbol{x})$:

$$\begin{aligned} \mu_t(\boldsymbol{x}) &= \boldsymbol{k}_t(\boldsymbol{x})^T(\boldsymbol{K}_t + \sigma^2 I)^{-1}\boldsymbol{y}_t \\ k_t(\boldsymbol{x}, \boldsymbol{x'}) &= k(\boldsymbol{x}, \boldsymbol{x'}) - \boldsymbol{k}_t(\boldsymbol{x})^T(\boldsymbol{K}_t + \sigma^2 I)^{-1}\boldsymbol{k}_t(\boldsymbol{x'}) \\ \sigma_t^2(\boldsymbol{x}) &= k_t(\boldsymbol{x}, \boldsymbol{x}) \end{aligned} \tag{8}$$

where $\boldsymbol{k}_t(\boldsymbol{x}) = [k(\boldsymbol{x}_1, \boldsymbol{x}), k(\boldsymbol{x}_2, \boldsymbol{x}), \cdots, k(\boldsymbol{x}_t, \boldsymbol{x})]^T$ and $\boldsymbol{K}_t$ is the positive definite kernel matrix $[k(\boldsymbol{x}, \boldsymbol{x'})]_{\boldsymbol{x}, \boldsymbol{x'} \in \boldsymbol{A}_t}$. There are a range of commonly used kernel functions, such as squared exponential kernel, linear kernel and Matern kernel. In Accordia, we adopt the squared exponential kernel since it is easy to compute when solving optimization problems. In addition, the computing ratio with similar cloud configurations are also highly correlated.

In round t, we have observed the noisy job completion ratio $\boldsymbol{y}_{t-1} = \{y_1, y_2, \cdots, y_{t-1}\}$ under the cloud configuration $\boldsymbol{A}_{t-1} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_{t-1}\}$. We can use the above Equations.(8) to predict the mean and covariance of the job completion ratio for all the candidate cloud configurations. Then we extend the Gaussian-Process UCB algorithm to select the current round cloud configuration. In round t, we choose the cloud configuration as the optimal solution to the following optimization problem (P3):

$$\begin{aligned} &\max_{\boldsymbol{x}_t \in \mathcal{X}} \mu_{t-1}(\boldsymbol{x}_t)/p(\boldsymbol{x}_t) + \beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x}_t)/p(\boldsymbol{x}_t) \\ &s.t.\ \mu_{t-1}(\boldsymbol{x}_t) - \beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x}_t) \geq 1/T_{max}, \end{aligned} \tag{9}$$

where $\beta_t = 2log(|\mathcal{X}|t^2\pi^2\delta/6)$ and $\delta \in (1, \infty)$. The above constraint in optimization problem P3(9) guarantees that the job completion ratio under the selected cloud configuration has $\geq 1 - 1/\delta$ probability to satisfy the deadline requirement in the optimization problem P1. The pseudo-code of our algorithm is provided in Algorithm 2.

To solve P3, we do not need to enumerate all the candidate configurations to search the optimal solution. Instead, we only apply the convex optimization approach, i.e., the dual approach to directly handle P3. Specifically, we first relax all the values in each configuration dimension to be continuous numbers. We then solve P3 in its dual space and obtain a relaxed solution. Finally, we round back all the fractional solutions to integers.

Algorithm 2 The Modified Gaussian-Process Bandit Algorithm under Time-invariant Price Case

Input: $p(x)$: price function;
$\boldsymbol{x}_1$: initial cloud configuration;
T_{max}: maximum tolerated running time;
$\mathcal{X}$: search space;
Output: optimal cloud configuration $\boldsymbol{x}^*$

1: Use the initial cloud configuration $\boldsymbol{x}_1$ to run the recurring job;
2: Observe the perturbed computing ratio value $y_1 = f(\boldsymbol{x}_1) + \epsilon_1$;
3: Add y_1 into $\boldsymbol{y}_1$ and add x_1 into $\boldsymbol{A}_1$;
4: **repeat** for round $t = 2, 3, \cdots$
5: Use $\boldsymbol{A}_{t-1}, \boldsymbol{y}_{t-1}$ to update the posterior distribution of the objective function to get $\mu_{t-1}(\boldsymbol{x})$ and $\sigma_{t-1}^2(\boldsymbol{x})$ as Eq.8;
6: Select the current round cloud configuration $\boldsymbol{x}_t$ based on $\mu_{t-1}(\boldsymbol{x})$ and $\sigma_{t-1}^2(\boldsymbol{x})$ as Eq.9;
7: Use the cloud configuration $\boldsymbol{x}_t$ to run the job;
8: Observe the perturbed computing ratio value y_t;
9: Add y_t into $\boldsymbol{y}_t$ and add x_t into $\boldsymbol{A}_t$;
10: **until** Reaching the stop condition;

Remark 3. *Comparing to the original Gaussian-Process UCB algorithm [23], our modified algorithm adds the constraint $\mu_{t-1}(\boldsymbol{x}_i) + \beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x}_t) \geq 1/T_{max}$, which can guarantee the deadline-constraint $f(\boldsymbol{x}) > 1/T_{max}$ has $\geq 1 - 1/\delta$ probability to be satisfied. At the same time, the objective function of the original Gaussian-Process UCB algorithm is sampled from a Gaussian process. However, the objective function of our model is divided by another known function $p(\boldsymbol{x})$.*

The performance of our algorithm can be characterized by static regret as follows:

Theorem 4.1. *Let $x \in \mathcal{X} \subset \mathbb{R}^d$ and $\delta \in (1, \infty)$. Running our Accordia algorithm with $\beta_t = 2log(|\mathcal{X}|t^2\pi^2\delta/6)$, we can obtain a regret bound of $O(\sqrt{(d+2)log|\mathcal{X}|TlogT}$ with high probability. Precisely,*

$$Pr\{Reg_T^s \leq O(\sqrt{\log(\delta|\mathcal{X}|)T(logT)^{d+1}})\} \geq 1 - \frac{1}{\delta}. \tag{10}$$

while the deadline constraint $\{f(\boldsymbol{x_t}) > 1/T_{max}, \forall t\}$ is also satisfied with probability at least $(1 - \frac{1}{\delta})$.

Proof. First of all, we want to show that $\{|f(\boldsymbol{x}) - \mu_{t-1}(\boldsymbol{x})| \leq \beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x}),\ \forall x \in \mathcal{X}, \forall t > 1\}$ holds with probability at least $(1 - \frac{1}{\delta})$.

If $r \sim N(0, 1)$ is drawn from a Gaussian distribution, one can show that:

$$Pr\{r \geq c\} \leq (1/2)e^{-c^2/2}. \tag{11}$$

In the Gaussian Process model as (8), $f(\boldsymbol{x}) \sim N(\mu_{t-1}(\boldsymbol{x}), \sigma_{t-1}^2(\boldsymbol{x}))$ is also drawn from a Gaussian distribution. We set $r = (f(\boldsymbol{x}) - \mu_{t-1}(\boldsymbol{x}))/\sigma_{t-1}(\boldsymbol{x})$ and $c = \beta_t^{1/2}$, we can get:

$$Pr\{|f(\boldsymbol{x}) - \mu_{t-1}(\boldsymbol{x})| \geq \beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x})\} \leq exp(-\beta_t/2). \tag{12}$$

With $\beta_t = 2log(|\mathcal{X}|t^2\pi^2\delta/6)$, we can modify the above (12) as:

$$Pr\{|f(\boldsymbol{x}) - \mu_{t-1}(\boldsymbol{x})| \geq \beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x})\} \leq 6/(\pi^2t^2\delta|\mathcal{X}|). \tag{13}$$

Due to $\sum(1/t^2) = 6/\pi^2$ and $\boldsymbol{x} \in \mathcal{X}$, we can get $\{|f(\boldsymbol{x})-\mu_{t-1}(\boldsymbol{x})| \leq \beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x}), \forall x \in \mathcal{X}, \forall t > 1\}$ holds with probability $\geq 1 - \frac{1}{\delta}$.

Secondly, we shall show that the difference in $f(\boldsymbol{x})/p(\boldsymbol{x})$ between the optimal configuration $\boldsymbol{x}^*$ and $\boldsymbol{x_t}$ produced by the optimization problem has an upper bound.

Since $\boldsymbol{x}^*$ is the optimal cloud configuration satisfied $f(\boldsymbol{x}^*) \geq 1/T_{max}$ in the optimization problem P1. If $\{|f(\boldsymbol{x}) - \mu_{t-1}(\boldsymbol{x})| \leq \beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x}), \forall x \in \mathcal{X}, \forall t > 1\}$ holds, we have:

$$\mu_{t-1}(\boldsymbol{x}^*) + \beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x}^*) \geq f(\boldsymbol{x}^*) \geq 1/T_{max}. \tag{14}$$

In this sense, $\boldsymbol{x}^*$ is a feasible solution of the optimization problem P3 (9) and $\boldsymbol{x_t}$ is the selected cloud configuration at this round, we have:

$$f(\boldsymbol{x_t}) \geq \mu_{t-1}(\boldsymbol{x}^*) - \beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x}^*) \geq 1/T_{max}. \tag{15}$$

In this sense, the deadline constraint $\{f(\boldsymbol{x_t}) \geq 1/T_{max}\}$ is satisfied. At the same time, consider $\boldsymbol{x_t}$ is the optimal solution of the optimization problem P3 and $\boldsymbol{x}^*$ is a feasible solution, we have:

$$\begin{aligned} &\mu_{t-1}(\boldsymbol{x_t})/p(\boldsymbol{x_t}) + \beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x_t})/p(\boldsymbol{x_t}) \\ &\geq \mu_{t-1}(\boldsymbol{x}^*)/p(\boldsymbol{x}^*) + \beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x}^*)/p(\boldsymbol{x}^*) \\ &\geq f(\boldsymbol{x}^*)/p(\boldsymbol{x}^*). \end{aligned} \tag{16}$$

Therefore, the $f(\boldsymbol{x})/p(\boldsymbol{x})$ difference between the optimal cloud configuration $\boldsymbol{x}^*$ and our algorithm $\boldsymbol{x_t}$ is bounded with probability $\geq 1 - \epsilon$, as follows:

$$\begin{aligned} &f(\boldsymbol{x}^*)/p(\boldsymbol{x}^*) - f(\boldsymbol{x_t})/p(\boldsymbol{x_t}) \\ &\leq \mu_{t-1}(\boldsymbol{x_t})/p(\boldsymbol{x_t}) + \beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x_t})/p(\boldsymbol{x_t}) - f(\boldsymbol{x_t})/p(\boldsymbol{x_t}) \\ &\leq 2\beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x_t})/p(\boldsymbol{x_t}). \end{aligned} \tag{17}$$

Thirdly, by Cauchy-Schwarz inequality, we can bound the static regret in Eq.4 as:

$$\begin{aligned} (Reg_s^T)^2 &= (\sum_{t=1}^{T} f(\boldsymbol{x}^*)/p(\boldsymbol{x}^*) - f(\boldsymbol{x_t})/p(\boldsymbol{x_t}))^2 \\ &\leq T\sum_{t=1}^{T}(f(\boldsymbol{x}^*)/p(\boldsymbol{x}^*) - f(\boldsymbol{x_t})/p(\boldsymbol{x_t}))^2 \\ &\leq \frac{4T\beta_T}{p^2}\sum_{t=1}^{T}\sigma_{t-1}^2(\boldsymbol{x_t}). \end{aligned} \tag{18}$$

where $p = \min_{\boldsymbol{x}\in\mathcal{X}} p(\boldsymbol{x})$ is the minimal price of the candidate cloud configurations and β_t is monotone increasing with t.

Finally, we want to proof $\sum_{t=1}^{T}\sigma_{t-1}^2(\boldsymbol{x_t})$ is bounded via analyzing the *maximum information gain* [15]. *Maximum information gain* comes from Bayesian Experimental Design, where the informativeness of a set of sampling points $A \subset \mathcal{X}$ for f is measured by the information gain, which is defined as the mutual information between f and observations $\boldsymbol{y}_A = \boldsymbol{f}_A + \epsilon_A$ at these points:

$$I(\boldsymbol{y}_A; f) = H(\boldsymbol{y}_A) - H(\boldsymbol{y}_A|f). \tag{19}$$

It measures the reduction in uncertainty about f from observing $\boldsymbol{y}_A$. The problem of finding the maximum information gain $\max_{|A|\leq T} I(\boldsymbol{y}_A; f)$ is NP-hard. If the covariance function $k(\boldsymbol{x}, \boldsymbol{x'})$ follows the squared exponential kernel, it can be bounded by $\gamma_T = O((logT)^{d+1})$ [17].

In our setting, we observe the function $f(\boldsymbol{x})$ at points $\boldsymbol{A}_T$ after T rounds. Following the same method as [23], by analyzing the information gain of $f(\boldsymbol{x})$, we can prove that:

$$\sum_{t=1}^{T}\sigma_{t-1}^2(\boldsymbol{x_t}) \leq 2\gamma_T/log(1+\sigma^{-2}). \tag{20}$$

We leave the detail proof of Eq.(11) and (20) to the Appendix.

Reformulate Eq.(20) and combine Eq.(18), we can bound the static regret as:

$$(Reg_s^T)^2 \leq \frac{8\beta_T\gamma_T T}{log(1+\sigma^{-2})p^2}. \tag{21}$$

With $\gamma_T = O((logT)^{d+1})$ and $\beta_T = 2log(|\mathcal{X}|T^2\pi^2\delta/6)$, we can get:

$$Pr\{Reg_s^T \leq O(\sqrt{\beta_T\gamma_T T})\} \geq 1 - \frac{1}{\delta}. \tag{22}$$

This completes the proof of Theorem 4.1. □

4.3 Extending Gaussian-Process UCB Algorithm for the Time-varying Price Case

In the time-varying price case, the price of spot instances can change gradually based on long-term trends in supply and demand. Due to the unknown time-varying price, we need to predict the price of cloud configurations. The predicted price is a function of all the past prices observed. Formally, let $\overline{p}_t$ denote the predicted price in the tth test run, $\overline{p}_t$ is given by:

$$\overline{p}_t = h(p_1, p_2, \cdots, p_{t-1}), \tag{23}$$

where h is the prediction function.

To get an accurate prediction, we can apply deep learning methods such as RNN and LSTM. Nevertheless, the actual price of spot instance in Amazon EC2 is based on long-term trends in supply and demand in a cloud, the change of spot prices in practice is relatively gradual (e.g. overall significant changes often take days and weeks) and highly predictable. The first-order increment estimation is already good enough for common big data analytic jobs which can complete within several hours. Due to this, Accordia adopts the following formula to update the time-varying price:

$$\overline{p}_t(\boldsymbol{x_t}) = p_{t-1}(\boldsymbol{x_t}) + \frac{(p_{t-1}(\boldsymbol{x_t}) - p_{t-2}(\boldsymbol{x_t}))}{2\mu_{t-1}(\boldsymbol{x_t})}. \tag{24}$$

With the predicted time-varying price for all the cloud configurations, we extend the Gaussian Process Bandit approach to dynamically select the current cloud configuration, via solving the following optimization problem (P4):

$$\begin{aligned} &\arg\max_{\boldsymbol{x_t}} \mu_{t-1}(\boldsymbol{x_t})/\overline{p}_t(\boldsymbol{x_t}) + \beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x_t})/\overline{p}_t(\boldsymbol{x_t}) \\ &s.t.\ \mu_{t-1}(\boldsymbol{x_t}) - \beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x_t}) \geq 1/T_{max}. \end{aligned} \tag{25}$$

The pseudo-code of the modified Gaussian-Process bandit algorithm is provided in Algorithm 3.

Paralleling Theorem 4.1, we show in the following theorem that, if the prediction in Eq.(23) is accurate enough, then the dynamic regret defined in Eq.(7) can be bounded with a high probability.

Theorem 4.2. *Let $\boldsymbol{x} \in \mathcal{X} \subset \mathbb{R}^d$ and $\delta \in (0,1)$. Running Algorithm 3 with $\beta_t = 2log(|\mathcal{X}|t^2\pi^2\delta/6)$, when*

$$\left|\overline{p}_t(\boldsymbol{x_t}) - p_t(\boldsymbol{x_t})\right| = o(\frac{1}{\sqrt{T}}),\ \ \forall \boldsymbol{x_t} \in \mathcal{X}.$$

Algorithm 3 The Modified Gaussian-Process Bandit Algorithm under Time-varying Price Case

Input: $\boldsymbol{x}_1$: initial cloud configuration;
T_{max}: maximum tolerated running time;
$\mathcal{X}$: search space;
Output: optimal cloud configuration $\boldsymbol{x}^*$

Use the initial cloud configuration $\boldsymbol{x}_1$ to run the recurring job;
Observe the perturbed computing ratio value y_1 and the price p_1;
Add y_1 into $\boldsymbol{y}_1$, add x_1 into $\boldsymbol{A}_1$ and record the price $p_1(\boldsymbol{x})$;
repeat for round $t = 2, 3, \cdots$
    Use $\boldsymbol{A}_{t-1}$ and $\boldsymbol{y}_{t-1}$ to update the posterior distribution of the objective function to get $\mu_{t-1}(\boldsymbol{x})$ and $\sigma^2_{t-1}(\boldsymbol{x})$ as Eq.8;
    Use $\mu_{t-1}(\boldsymbol{x})$ to predict the time-varying price $\overline{p}_t(\boldsymbol{x})$ as Eq.24;
    Select the current round cloud configuration $\boldsymbol{x}_t$ based on $\mu_{t-1}(\boldsymbol{x})$, $\sigma^2_{t-1}(\boldsymbol{x})$ and $\overline{p}_t(\boldsymbol{x})$ as Eq.9;
    Use the cloud configuration $\boldsymbol{x}_t$ to run the job;
    Observe the perturbed computing ratio value y_t and the price p_t;
    Add y_t into $\boldsymbol{y}_t$, add x_t into $\boldsymbol{A}_t$ and record the price $p_t(\boldsymbol{x})$;
until Stop by user;

we can obtain a dynamic regret bound of $O(\sqrt{(d+2)log|\mathcal{X}|TlogT}$ *with high probability. Precisely,*

$$Pr\{Reg_T^d \leq O(\sqrt{\log(\delta|\mathcal{X}|)T(logT)^{d+1}})\} \geq 1 - \frac{1}{\delta}. \quad (26)$$

while the deadline constraint $\{f(\boldsymbol{x_t}) > 1/T_{max}, \forall t\}$ *is also satisfied probability at least* $(1 - \frac{1}{\delta})$.

We leave the detail proof of THEOREM 4.2 to the Appendix.

4.4 Initial Cloud Configuration and Stopping Condition of Accordia

In this part, we present more details of Accordia. We shall first introduce the choices of the initial cloud configurations. In the sequel, we describe how to apply machine learning methods to early stop not useful test runs.

4.4.1 *Starting Point.*

We use four c4.xlarge instances as the initial cloud configuration in our algorithm. The choice of initial cloud configuration can give Gaussian Process model a starting point to estimate the shape of the search space. For that, Bayesian optimization based methods, e.g. EI and PI algorithms [11, 29] always need to randomly sample a few points (e.g. three) from the sample space to provide an estimation about the shape of the job completion cost model. They need to make sure that the selected initial cloud configurations are as diverse as possible and can cover the sample space uniformly. It helps the prior function to avoid making wrong assumptions about the sample space. The performance of these Bayesian optimization methods heavily depends on the initial samples. They may suffer long-time trials from bad initial samples.

In our trace-driven study, it is shown that the choice of initial cloud configuration does not influence the performance of UCB-based algorithms. We can just use four c4.xlarge instances as the initial cloud configuration. Each c4.xlarge instance has 4 CPU cores and 8 GB memory in Amazon EC2. It is suitable for most of the big data analysis applications. Abandoning the unnecessary first three random samplings, our algorithm can quickly move into the search phase.

4.4.2 *Stopping Condition.*

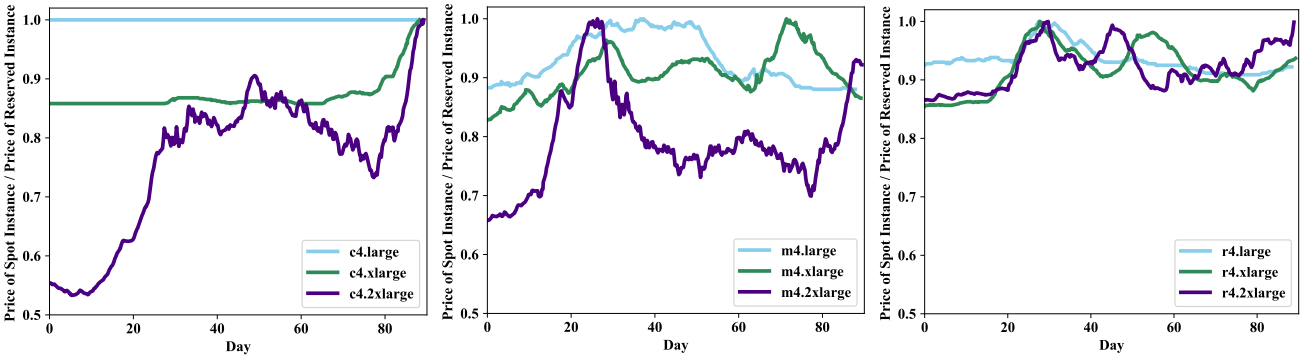
We use stopping condition to prevent Accordia from struggling to make small improvements in the fixed price case. For that, other methods always [9] [18] define the stopping condition based on predicted variance $\sigma(\boldsymbol{x})$ in Eq.8 and the number of runs. They will stop their algorithm when the variance $\sigma(\boldsymbol{x})$ of the selected cloud configuration is less than a threshold (e.g. 10% of the initial value) and at least N (e.g. $N = 10$) cloud configurations have been observed. However, this kind of stopping condition may suffer from testing the same cloud configuration many times, until the variance $\sigma(\boldsymbol{x})$ decreases below the threshold.

Table 1: 11-dimension Low-level Running Time Information

Feature	Description
cpu.%usr	Percentage of CPU utilization.
task.proc/s	Total number of tasks created per second.
memory.%usr	Percentage of used memory.
swap.%used	Percentage of used swap space.
swap.%cad	Percentage of cached swap memory in relation to the amount of used swap space.
disk.rd/s	Number of sectors read from the device.
disk.wt/s	Number of sectors written to the device.
disk.%util	Bandwidth utilization for the device.
network.rxpck/s	Total number of packets received per second.
network.txpck/s	Total number of packets transmitted per second.
network.%util	Bandwidth utilization for the network.

In the fixed price case, we define the stopping condition based on the 11-dimension low-level running-time performance metrics as Table 1. Our algorithm aims to achieve the cloud configuration with minimal job completion cost, which may benefit from high CPU utilization, memory utilization and etc. The low-level running-time performance metrics should be highly correlated to the job completion cost. In this sense, we use the large-scale performance dataset [18] [19] [20] to train a logistic regression model to predict whether a big data analytic job obtains its optimal cloud configuration under the given low-level running-time information. The dataset includes 17 workloads, thousands of big data analytics jobs and its low-level running-time information. When we train the logistic regression model, we ignore the workload name and use the jobs across all the workloads.

The stop condition of Accordia is defined as the cloud configuration achieving the current minimum job completion cost and logistic regression result larger than 0.8. In the experiment, the cross entropy of the logistic regression is 0.76 and the logistic regression result is highly correlated to the job completion cost. Accordia can quickly stop as long as obtaining the optimal cloud configuration. When we consider the time-varying price and dynamic cloud environment, the optimal cloud configuration is time-varying and may



(a) Compute optimized instance family (b) General purpose instance family (c) Memory optimized instance family

Figure 2: Time-Varying Price of AWS Instances Collected in Three Months

not converge. We need to keep searching and never stop. In this case, we remove the stop condition.

5 TRACE-DRIVEN SIMULATIONS

In this section, we evaluate the performance of Accordia via conducting extensive numerical simulations based on the large-scale performance dataset [18–20].

5.1 Experimental Setup

The simulations are driven by the large-scale performance dataset [18–20] containing Hadoop and Spark applications running on AWS EC2 instances and AWS time-varying price history collected in a three-month period.

Application: The large-scale dataset has 16 workloads, including supervised learning such as regression and classification application, unsupervised learning such as K-means clustering, and statistic tools such as WordCount and PCA. For each application, the dataset collects its job completion time and its low-level running-time information using [sar] [8].

Cloud Configuration: There are three instance families {c4, m4, r4 } and three instance size {large, xlarge, 2xlarge } (available in AWS EC2) in the large-scale performance dataset. An instance type and the number of the same instances comprise the totally 66 candidate cloud configurations in the dataset.

AWS Time-varying Price: Based on the price, there are two kinds of instances in Amazon EC2: On-demand instances and spot instances. The price of On-Demand instances is fixed, while AWS can adjust the price of spot instances gradually based on long-term trends, up to 90% off discount. We collect a 3-month AWS time-varying price history as depicted in Figure 2. We both consider the fixed price case for AWS On-demand instances and the time-varying price case for AWS spot instances in the experiment.

Encode Cloud Configuration: Similar to CherryPick, in our simulation, each cloud configuration is encoded using three feature dimensions: the number of instances, number of CPU cores and memory (RAM) allocation for each instance. We also test the performance under other feature combinations and choose the best one.

5.2 Evaluation results in the fixed price case

In this part, we set the price of instances the same as AWS On-Demand instances and evaluate our Accordia algorithm in the fixed price case.

Figure 3 presents the sequence of chosen cloud configurations in the Hadoop WordCount example. The search space is projected into two dimensions. The x-axis is the total number of CPU cores and the y-axis is the number of instances under the cloud configuration. The 66 points represent the set of candidate cloud configurations and the points with brighter color can achieve smaller job completion costs. In Accordia, 4-c4.xlarge-instances is the initial cloud configuration in the left lower corner of the search space in Figure 3. Then, Accordia keeps jumping dramatically among the search space to unearth the optimal cloud configuration until it finds out a near-optimal cloud configuration at 5th runs and starts to do the local search. Comparing to the other Bayesian Optimization algorithm, Accordia does not need to use a fixed number of runs (e.g. 3 or 6) to do random sampling first. It can automatically sample the first several runs as diverse as possible and cleverly decide when to start the local search.

Figure 4 presents the performance of the stop condition under the Hadoop WordCount example. The point represents the cloud configuration associated with its job completion cost in the x-axis and logistic regression result in the y-axis which shows the logistic regression result decreases with the increasing job completion cost. The cloud configuration with lower completion cost has a high probability to achieve higher logistic regression result. As the Accordia stop condition is defined as the cloud configuration achieving the minimum job completion cost observed so far and its logistic regression result larger than 0.8, it is highly possible to stop at the near-optimal cloud configuration (i.e., within 10% of the optimum).

Figure 5 shows the percentage of workloads achieving the near-optimal cloud configurations with the number of runs increasing. The x-axis in Figure 5 is the number of runs observed so far and the y-axis is the percentage of workloads achieving the near-optimal cloud configuration at current run. The curve of Accordia is above that of CherryPick within the first eight runs and both of them converge after 12 runs. It states that about 80% of workloads can find

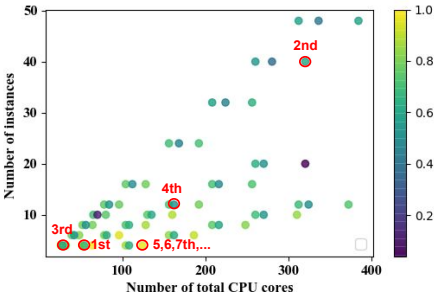


Figure 3: Sequence of Cloud Configurations Chosen by Accordia running Hadoop WordCount Example.

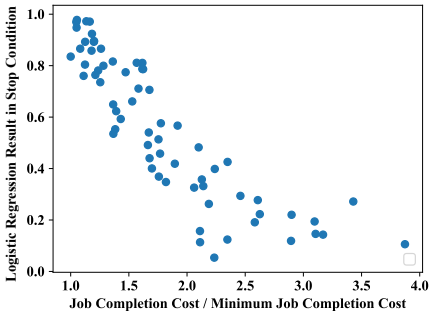


Figure 4: Relation between the Job Completion Cost and Logistic Regression Results with Early Stop.

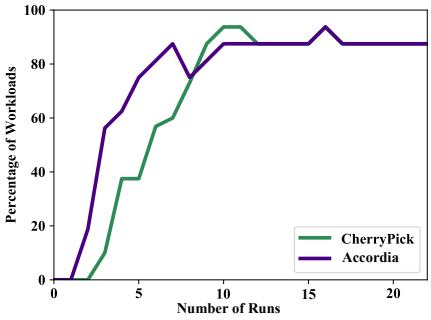
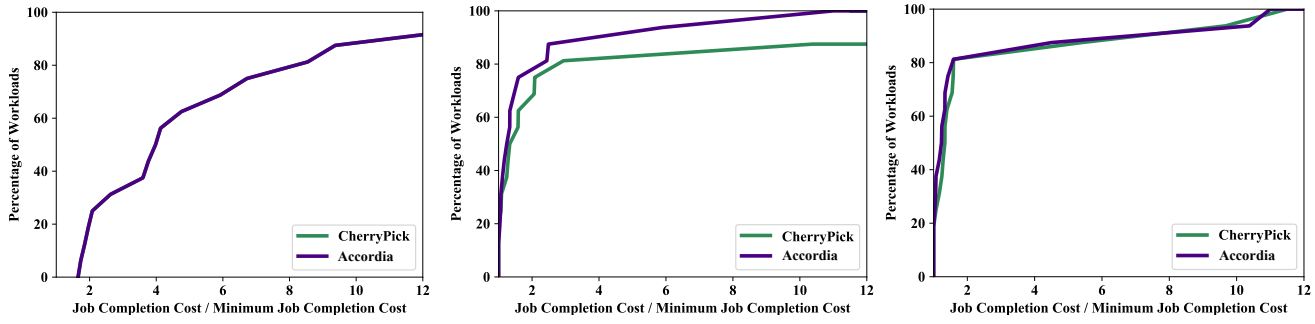


Figure 5: Percentage of Workloads Achieving the Near-optimal Cloud Configuration.



(a) After the first run. **(b) After 5 runs** **(c) After 10 runs**

Figure 6: How Far the Selected Cloud Configuration is from the Optimal Cloud Configuration.

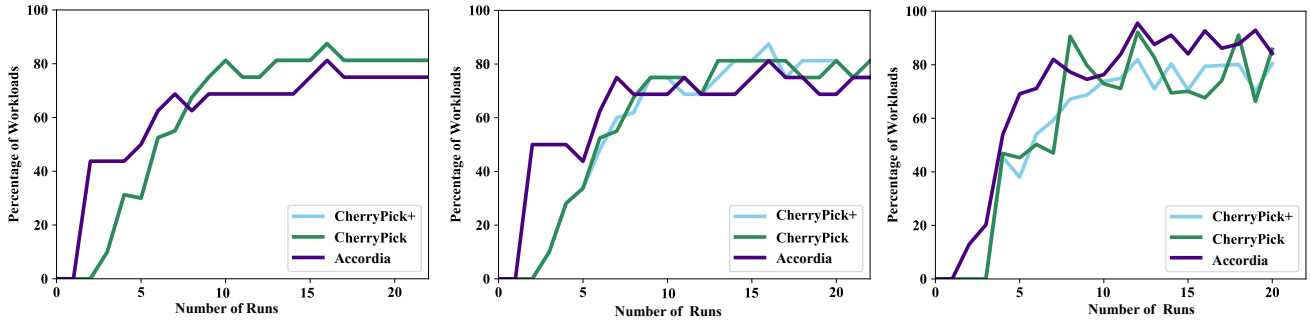
out the near-optimal configuration after 6 runs using Accordia while CherryPick requires 10 runs. It implies that Accordia can obtain the optimal cloud configuration much faster than CherryPick.

Figure 6 shows how far the selected cloud configuration from the optimal one with the number of runs increasing. The x-axis is the normalized job completion cost and the y-axis is the percentage of workloads. The curve represents the percentage of workloads that can complete under the given budget. Figure 6(a), Figure 6(b) and Figure 6(c) show the curves obtained by Accordia and Cherrypick after the first run, after five runs, and after ten runs. They present, with the number of runs increasing, an increasing percentage of workloads can complete under the given budget. Both Accordia and CherryPick use the same initial cloud configuration at the first run and converge to the near-optimal cloud configuration after ten runs, so the curves of Accordia and CherryPick are overlapped in Figure 6(a) and Figure 6(c). However, the curve of Accordia is totally above that of CherryPick after five runs in Figure 6(b). It states Accordia can help more workloads to achieve lower job completion costs than CherryPick after five runs. Accordia can save more job completion costs than CherryPick.

5.3 Evaluation for the time-varying price case

In this part, we evaluate Accordia using the time-varying price of AWS EC2 spot instances as shown in Figure 2. We assume the arriving of recurring big data analytic jobs follows the Poisson Process and set different arriving rate to control the variation of price. It has following three cases: i) fixed price case; ii) arriving rate is one job per three hours and the price changes within 5%; iii) arriving rate is one job per day and the price change may exceed 10%.

Accordia separately predicts the job completion time and the per unit-time price to unearth the cloud configuration with minimum job completion cost. To be fair, we add this feature to CherryPick as CherryPick+ and show the percentage of workloads achieving the near-optimal cloud configuration in Figure 7. In the fixed price case, the curves of CherryPick and CherryPick+ are overlapped as shown in Figure 7(a). Figure 7 shows the curves of Accordia always exceed that of CherryPick and CherryPick+ at first 10 runs, which states the efficiency of Accordia to find out the time-varying near-optimal cloud configuration. Also, in Figure 7(a) and Figure 7(b) when the price of cloud configuration changes within 5%, the methods separately considered the job completion time and per unit-time price get the similar performance of the methods considered job completion cost as a whole. However, when we decrease the arriving rate and the price changes dramatically, Accordia performs much better than CherryPick and CherryPick+ which states Accordia can handle the time-varying price case. It is due to i) Accordia separately predicts the job completion time and the per unit-time price; ii) Accordia can quickly find out the near-optimal cloud configuration and chase the time-varying price more closely.



(a) Fixed price case. (b) One job per three hours (c) One job per day.

Figure 7: Percentage of Workloads Achieving Near-optimal Cloud Configuration in the Time-varying Price Case.

6 SYSTEM IMPLEMENTATION OF ACCORDIA UNDER KUBERNETES

In this section, we present the system implementation of Accordia. To be specific, we build the Accordia system to unearth the optimal cloud configuration for Spark applications on top of Kubernetes in the public clouds.

6.1 Spark in Kubernetes

Spark 2.3 or its above can run on the Kubernetes platform and be managed by the native Kubernetes scheduler. When a Spark job is submitted, a Spark driver and multiple Spark executors are deployed as containers, each within its own Kubernetes pod. Accordia then dynamically adjusts the resource types/ allocation for the pods from over 7000 candidate choices. They form a 5-dimensional search space, covering from the number of executors, as well as the number of CPU cores and memory (RAM) allocation for the driver and the executor pods. Unlike CherryPick, considering the cloud configuration as the number and resource type of instances, Accordia manages a higher dimensional search space and provides a more fine-grained resource allocation for Spark jobs in the public clouds.

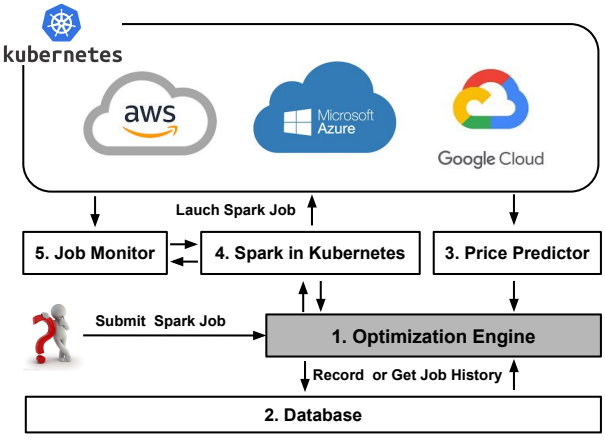


Figure 8: System Architecture of Accordia Under Kubernetes

6.2 System Architecture

Figure 8 depicts the system architecture of our implementation of Accordia for Spark applications on top of Kubernetes. It has the following four components.

Database: Database has the list of candidate cloud configurations and stores all the job history information, including job name, timestamp, cloud configuration, job completion time, cost and etc. It is responsible for maintaining and sending the job history information to Optimization Engine. There is a sliding window (e.g., N = 20) to limit the size of job history information sending to Optimization Engine. It helps Optimization Engine to focus on the recent 20 recurring jobs and reduces the influence of the dynamic cloud noise.

Price Predictor: Price Predictor can collect the real-time per unit-time price for all the candidate cloud configurations and predict their future price for the Optimization Engine. In our implementation, we just use the first order increment method as Eq.(24) to predict the future price. Since the time-varying price of Amazon EC2 is based on long-term trends in supply and demand, the first order increment method is already accurate enough to predict the price for common big data analytic jobs which can complete within several hours.

Optimization Engine: Optimization Engine is an implementation of the Accordia algorithm in Python. Based on the Database's job history information, it can predict the job completion time for all the candidate cloud configurations. Then combine the predicted price from Price Predictor to select and launch the cloud configuration to minimize the job completion cost.

Job Monitor: As long as a Spark job is submitted to the Kubernetes platform, Job Monitor starts to monitor its running time information. It will use the 10% and 20% workload's completion time to predict the job completion cost. The running Spark job will be aborted as soon as possible if its predicted job completion cost is 30% larger than the best one observed so for. The aborted job and it's predicted job completion time will also be recorded in Database as the job history information.

7 EXPERIMENTAL EVALUATION OF ACCORDIA IMPLEMENTATION UNDER KUBERNETES

In this section, we present the experimental results of Accordia on Kubernetes implementation depicted in Section 6.

7.1 The Effect of Abort Mechanism

To avoid the waste of resources for the obvious poor-performance cloud configuration, Job Monitor is designed to abort the job, as long as its estimated job completion cost is 30% larger than the best solution observed so far. We repeatedly submit the SparkPi jobs to the Accordia system and record the job completion cost in Figure 9 to show the performance of Accordia and the abort mechanism.

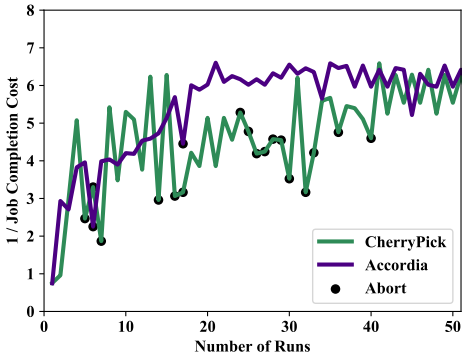


Figure 9: Abort Mechanism in SparkPi Example.

In Figure 9, the y-axis is the inverse of job completion cost and the black dot represents the job is aborted because of large estimated job completion cost. It shows that Accordia keeps searching for the optimal cloud configuration and converges after fewer than 20 runs, which translates to a 2X-speedup and a 20.9% cost-savings comparing to CherryPick. The curve of Accordia also increases more smoothly and has a lower number of aborted jobs than that of CherryPick. It states Accordia can efficiently unearth the optimal cloud configuration in high-dimension large search space.

7.2 Tracking Changes in Job Characteristics

Since computing resources in a cloud are often shared by multiple users/ jobs with non-perfect separation/ isolation, stragglers can happen easily. The completion time of the same job/ application under the same cloud configuration may still suffer from a large variance. As such, there is a sliding window to make Optimization Engine only focusing on the recent job history to reduce the influence of the dynamic cloud noise in Accordia implementation.

To highlight Accordia's capability to handle the dynamic cloud environment and abrupt/ unexpected changes of the characteristics of a recurring job, we submit one recurring spark job per day and dynamic switch the job type (i.e., from SparkPi to PageRank to WordCount) without notifying Accordia every 30 days. The performance of Accordia under different window sizes is shown in Figure 10. The y-axis is the inverse of job completion cost and the x-axis is the number of runs observed so far. The blue curve represents Accordia under the window size equaling to 40 which is similar with no sliding window (i.e., remembering all the job history information). With the misleading history information, it may take another 20 runs to find out the new optimal cloud configuration. The violet curve and green curve represent Accordia under the window size equaling to 10 and 20, which is enough for Accordia to find out the new optimal cloud configuration and ignore the misleading history information. They can find out the new optimal cloud configuration within 15 rounds and beat the Accordia without sliding window.

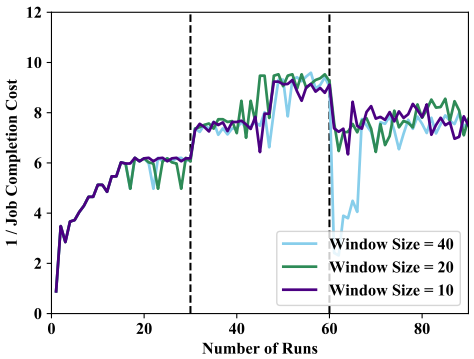


Figure 10: Different Window Size under Dynamic switching the Job Type.

Dynamic switching the job type is the extreme case of dynamic clouds. It can also simulate the misuse of Accordia, e.g., the user replacing the job without notifying the system. We also compare the cost-saving of Accordia with that of CherryPick under dynamic switching the job type over exponentially-distributed time-intervals (i.e., Poisson distribution with $\lambda = 20$) as shown in Table 2. Accordia still can obtain up to 18.6% cost-savings comparing to CherryPick.

Table 2: Accordia Cost-saving under Changing Job Types

Window Size	10	20	40	60
Cost-saving	18.4%	18.6%	9.9%	10.2%

Accordia under the window size equaling to 20 can achieve the best cost-saving, which is able to quickly find out the (near) optimal cloud configuration and can ignore the misleading history information. We even adjust the job arriving rate as one job per week to simulate the dramatical changes in time-varying price. Accordia still can obtain 18.2% cost-savings under the window size equaling to 20.

8 CONCLUSION

In this paper, we identify the online selection problem to find out the most economical cloud configuration for recurring big data analytics jobs. We propose the Accordia system, which separately predicts the job completion time and per unit-time price to select the optimal cloud configuration to minimize job completion cost. Accordia is based on the solid mathematical foundation and implemented in the Kubernetes platform for Spark applications. Accordia is also evaluated in both numerical simulation and Kubernetes implementation, which can find out the (near) optimal cloud configuration after fewer than 20 runs from over 7000 candidate choices, which translates to a 2X-speedup and a 20.9% costing-savings comparing to the state-of-art algorithm, CherryPick.

REFERENCES

[1] 2014. Apache Spark. http://spark.apache.org.
[2] 2015. Apache Hive. http://hive.apache.org.
[3] 2015. Apache Pig. http://pig.apache.org.
[4] 2015. Spark Dynamic Allocation. https://jaceklaskowski.gitbooks.io/mastering-apache-spark/spark-dynamic-allocation.html.
[5] 2017. Tensorflow. http://www.tensorflow.org.
[6] 2018. Amazon Web Services. http://aws.amazon.com/ec2.
[7] 2018. Google Cloud. http://cloud.google.com/compute.
[8] 2019. sar - Linux man page. https://linux.die.net/man/1/sar.
[9] Omid Alipourfard, Hongqiang Harry Liu, Jianshu Chen, Shivaram Venkataraman, Minlan Yu, and Ming Zhang. 2017. CherryPick: Adaptively Unearthing the Best Cloud Configurations for Big Data Analytics.. In *NSDI*, Vol. 2. 4–2.
[10] Ganesh Ananthanarayanan, Michael Chien-Chun Hung, Xiaoqi Ren, and Ion Stoica. 2014. GRASS: Trimming Stragglers in Approximation Analytics. In *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation*.
[11] Eric Brochu, Vlad M Cora, and Nando De Freitas. 2010. 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).
[12] Sébastien Bubeck, Nicolo Cesa-Bianchi, et al. 2012. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends® in Machine Learning* 5, 1 (2012), 1–122.
[13] Zhen Cao, Vasily Tarasov, Sachin Tiwari, and Erez Zadok. 2018. Towards better understanding of black-box auto-tuning: a comparative analysis for storage systems. In *2018 USENIX Annual Technical Conference(USENIX ATC)*. 893–907.
[14] Deepthi Chelupati and Roshni Pary. 2018. New Amazon EC2 Spot pricing model: Simplified purchasing without bidding and fewer interruptions. https://aws.amazon.com/tw/blogs/compute/new-amazon-ec2-spot-pricing.
[15] TM Cover and JA Thomas. 1991. Elements of Information Theory,(pp 33-36) John Wiley and Sons. *Inc, NY* (1991).
[16] Valentin Dalibard, Michael Schaarschmidt, and Eiko Yoneki. 2017. BOAT: Building auto-tuners with structured Bayesian optimization. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 479–488.
[17] Varsha Dani, Thomas P Hayes, and Sham M Kakade. 2008. Stochastic linear optimization under bandit feedback. (2008).
[18] Chin-Jung Hsu, Vivek Nair, Vincent W Freeh, and Tim Menzies. 2018. Arrow: Low-Level Augmented Bayesian Optimization for Finding the Best Cloud VM. In *the 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS 2018)*.
[19] Chin-Jung Hsu, Vivek Nair, Tim Menzies, and Vincent Freeh. 2018. Micky: A Cheaper Alternative for Selecting Cloud Instances. In *the IEEE International Conference on Cloud Computing (IEEE CLOUD 2018)*.
[20] Chin-Jung Hsu, Vivek Nair, Tim Menzies, and Vincent Freeh. 2018. Scout: An Experienced Guide to Find the Best Cloud Configuration. *arXiv preprint arXiv:1803.01296* (2018).
[21] Vasiliki Kalavri, John Liagouris, Moritz Hoffmann, Desislava Dimitrova, Matthew Forshaw, and Timothy Roscoe. 2018. Three steps is all you need: fast, accurate, automatic scaling decisions for distributed streaming dataflows. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 783–798.
[22] Ana Klimovic, Heiner Litz, and Christos Kozyrakis. 2018. Selecta: heterogeneous cloud storage configuration for data analytics. In *2018 USENIX Annual Technical Conference (USENIX ATC)*. 759–773.
[23] Andreas Krause and Cheng S Ong. 2011. Contextual gaussian process bandit optimization. In *Advances in Neural Information Processing Systems*. 2447–2455.
[24] Linkedin. 2016. Open Sourcing Dr. Elephant: Self-Serve Performance Tunning for Hadoop and Spark. https://github.com/linkedin/dr-elephant.
[25] Mahdavi M, Jin R, and Yang T. 2012. Trading regret for efficiency: online convex optimization with long term constraints[J]. In *Journal of Machine Learning Research*. 2503–2538.
[26] Zinkevich M. 2003. Online convex programming and generalized infinitesimal gradient ascent[C]. In *ICML*. 928–936.
[27] Carl Edward Rasmussen. 2004. Gaussian processes in machine learning. In *Advanced lectures on machine learning*. Springer, 63–71.
[28] Anooshiravan Saboori, Guofei Jiang, and Haifeng Chen. 2008. Autotuning configurations in distributed systems for performance improvements using evolutionary strategies. In *2008 The 28th International Conference on Distributed Computing Systems*. IEEE, 769–776.
[29] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*. 2951–2959.
[30] Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias Seeger. 2010. Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design. In *Proceedings of the 27th International Conference on Machine Learning*.
[31] Neeraja J Yadwadkar, Ganesh Ananthanarayanan, and Randy Katz. 2014. Wrangler: Predictable and faster jobs using fewer resources. In *Proceedings of the ACM Symposium on Cloud Computing*. ACM, 1–14.
[32] Neeraja J Yadwadkar, Bharath Hariharan, Joseph E Gonzalez, Burton Smith, and Randy H Katz. 2017. Selecting the best vm across multiple public clouds: A data-driven performance modeling approach. In *Proceedings of the 2017 Symposium on Cloud Computing*. ACM, 452–465.
[33] Zhibin Yu, Zhendong Bei, and Xuehai Qian. 2018. Datasize-Aware High Dimensional Configurations Auto-Tuning of In-Memory Cluster Computing. In *Proceedings of the 23rd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.
[34] M. Zaharia, D. Borthakur, J. S. Sarma, K. Elmeleegy, S. Shenker, and I. Stoica. 2009. Job scheduling for multi-user mapreduce clusters. In *tech. rep.* University of California, Berkeley.

A PROOF OF EQ.(11)

Proof. If $r \sim N(0,1)$ is drawn from a Gaussian distribution, one upper bound of $Pr\{r \geq c\}$ is given by:

$$\begin{aligned} & Pr\{r \geq c\} \\ &= (2\pi)^{-1/2} \int_c^{+\infty} exp(-r^2/2)dr \\ &= e^{-c^2/2}(2\pi)^{-1/2} \int_c^{+\infty} exp(-(r-c)^2/2 - c(r-c))dr \\ &\leq e^{-c^2/2}(2\pi)^{-1/2} \int_c^{+\infty} exp(-(r-c)^2/2)dr \\ &\leq e^{-c^2/2}(2\pi)^{-1/2} \int_0^{+\infty} exp(-r^2/2)dr \\ &\leq (1/2)e^{-c^2/2}. \end{aligned} \tag{27}$$

This completes the proof. □

B PROOF OF EQ.(20)

Proof. For a Gaussian distribution, $H(N(\boldsymbol{\mu}, \Sigma)) = \frac{log|2\pi e\Sigma|}{2}$ and $I(\boldsymbol{y}_A; f) = log|I + \sigma^{-2}\boldsymbol{K}_A|/2$, where $\boldsymbol{K}_A = [k(\boldsymbol{x}, \boldsymbol{x'})]_{\boldsymbol{x},\boldsymbol{x'} \in A}$, $I(\boldsymbol{y}_A; f)$ is bounded by:

$$I(\boldsymbol{y}_A; f) \leq \max_{|A| \leq T} I(\boldsymbol{y}_A; f). \tag{28}$$

The problem of finding the maximum information gain in Eq.(28) is NP-hard. It can be bounded by $\gamma_T = O((logT)^{d+1})$ [17], if the covariance function $k(\boldsymbol{x}, \boldsymbol{x'})$ follows the squared exponential kernel.

In our setting, we observe the computing ratio function f at points $\boldsymbol{A}_T$ after T rounds, as a consequence, we have:

$$\begin{aligned} \gamma_T &\geq I(\boldsymbol{y}_T; f) \\ &\geq H(\boldsymbol{y}_T) - H(\boldsymbol{y}_T|f) \\ &\geq H(\boldsymbol{y}_T) - log|2\pi e\sigma^2 I|/2 \\ &\geq H(\boldsymbol{y}_{T-1}) + H(y_T|\boldsymbol{y}_{T-1}) - log|2\pi e\sigma^2 I|/2 \\ &\geq H(\boldsymbol{y}_{T-1}) + log(2\pi e(\sigma^2 + \sigma^2_{T-1}(\boldsymbol{x}_T)))/2 \\ &\quad - log|2\pi e\sigma^2 I|/2 \\ &\geq \sum_{t=1}^{T} log((1 + \sigma^{-2}\sigma^2_{t-1}(\boldsymbol{x}_t)))/2 \\ &\geq \frac{log(1+\sigma^{-2})}{2} \sum_{t=1}^{T} \sigma^2_{t-1}(\boldsymbol{x}_t). \end{aligned} \tag{29}$$

This completes the proof. □

C PROOF OF THEOREM 4.2

Proof. Following the same method as THEOREM 4.1, $\boldsymbol{x_t}$ is the optimal solution of the optimal problem P4 and satisfied the deadline constraint with probability $\geq 1-\frac{1}{\delta}$, while $\boldsymbol{x}_t^*$ is a feasible solution. Then,

$$\begin{aligned}
&\mu_{t-1}(\boldsymbol{x_t})/\overline{p}_t(\boldsymbol{x_t}) + \beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x_t})/\overline{p}_t(\boldsymbol{x_t}) \\
&\geq \mu_{t-1}(\boldsymbol{x}_t^*)/\overline{p}_t(\boldsymbol{x}_t^*) + \beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x}_t^*)/\overline{p}_t(\boldsymbol{x}_t^*) \\
&\geq f(\boldsymbol{x}_t^*)/\overline{p}_t(\boldsymbol{x}_t^*) \\
&\geq (f(\boldsymbol{x}_t^*)/p_t(\boldsymbol{x}_t^*)) \cdot (p_t(\boldsymbol{x}_t^*)/\overline{p}_t(\boldsymbol{x}_t^*)).
\end{aligned} \tag{30}$$

Therefore, the $f(\boldsymbol{x})/p_t(\boldsymbol{x})$ difference between the optimal cloud configuration $\boldsymbol{x}_t^*$ and our algorithm $\boldsymbol{x_t}$ is bounded with probability $\geq 1-\epsilon$, as follows:

$$\begin{aligned}
&f(\boldsymbol{x}_t^*)/p_t(\boldsymbol{x}_t^*) - f(\boldsymbol{x_t})/p_t(\boldsymbol{x_t}) \\
&\leq [\overline{p}_t(\boldsymbol{x}_t^*)/p_t(\boldsymbol{x}_t^*)] \cdot \left(\mu_{t-1}(\boldsymbol{x_t})/\overline{p}_t(\boldsymbol{x_t}) + \beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x_t})/\overline{p}_t(\boldsymbol{x_t})\right) \\
&\quad - f(\boldsymbol{x_t})/p_t(\boldsymbol{x_t}) \\
&\leq [\overline{p}_t(\boldsymbol{x}_t^*)/\left(p_t(\boldsymbol{x}_t^*)\overline{p}_t(\boldsymbol{x_t})\right)] \cdot \left(\mu_{t-1}(\boldsymbol{x_t}) + \beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x_t})\right) \\
&\quad - f(\boldsymbol{x_t})/p_t(\boldsymbol{x_t}) \\
&\leq [\overline{p}_t(\boldsymbol{x}_t^*)/\left(p_t(\boldsymbol{x}_t^*)\overline{p}_t(\boldsymbol{x_t})\right)] \cdot \left(\mu_{t-1}(\boldsymbol{x_t}) + \beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x_t})\right) \\
&\quad - [\overline{p}_t(\boldsymbol{x}_t^*)/\left(p_t(\boldsymbol{x}_t^*)\overline{p}_t(\boldsymbol{x_t})\right) - \overline{p}_t(\boldsymbol{x}_t^*)/\left(p_t(\boldsymbol{x}_t^*)\overline{p}_t(\boldsymbol{x_t})\right)] \cdot f(\boldsymbol{x_t}) \\
&\quad - f(\boldsymbol{x_t})/p_t(\boldsymbol{x_t}) \\
&\leq [\overline{p}_t(\boldsymbol{x}_t^*)/\left(p_t(\boldsymbol{x}_t^*)\overline{p}_t(\boldsymbol{x_t})\right)] \left(\mu_{t-1}(\boldsymbol{x_t}) + \beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x_t}) - f(\boldsymbol{x_t})\right) \\
&\quad + [\overline{p}_t(\boldsymbol{x}_t^*)/\left(p_t(\boldsymbol{x}_t^*)\overline{p}_t(\boldsymbol{x_t})\right) - 1/p_t(\boldsymbol{x_t})] \cdot f(\boldsymbol{x_t}) \\
&\leq 2[\overline{p}_t(\boldsymbol{x}_t^*)/\left(p_t(\boldsymbol{x}_t^*)\overline{p}_t(\boldsymbol{x_t})\right)]\beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x_t}) \\
&\quad + \left[\left(\overline{p}_t(\boldsymbol{x}_t^*)p_t(\boldsymbol{x_t}) - p_t(\boldsymbol{x}_t^*)\overline{p}_t(\boldsymbol{x_t})\right)/\left(p_t(\boldsymbol{x}_t^*)\overline{p}_t(\boldsymbol{x_t})p_t(\boldsymbol{x_t})\right)\right] \cdot f(\boldsymbol{x_t}) \\
&\leq O(\beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x_t})) + o(1/\sqrt{T}).
\end{aligned} \tag{31}$$

The $o(1/\sqrt{t})$ in Eq.(31) is due to the differences between $p_t(\boldsymbol{x_t})$ and $\overline{p}_t(\boldsymbol{x_t})$, captured by $\left|\overline{p}_t(\boldsymbol{x_t}) - p_t(\boldsymbol{x_t})\right| = O(\frac{1}{\sqrt{T}})$, as follows:

$$\begin{aligned}
&\overline{p}_t(\boldsymbol{x}_t^*)p_t(\boldsymbol{x_t}) - p_t(\boldsymbol{x}_t^*)\overline{p}_t(\boldsymbol{x_t}) \\
&= (\overline{p}_t(\boldsymbol{x}_t^*) - p_t(\boldsymbol{x}_t^*))(\overline{p}_t(\boldsymbol{x_t}) + p_t(\boldsymbol{x_t}))/2 \\
&\quad - (\overline{p}_t(\boldsymbol{x}_t^*) + p_t(\boldsymbol{x}_t^*))(\overline{p}_t(\boldsymbol{x_t}) - p_t(\boldsymbol{x_t}))/2 \\
&= o(1/\sqrt{T})(\overline{p}_t(\boldsymbol{x_t}) + p_t(\boldsymbol{x_t})) + o(1/\sqrt{T})(\overline{p}_t(\boldsymbol{x}_t^*) + p_t(\boldsymbol{x}_t^*)) \\
&= o(1/\sqrt{T}).
\end{aligned} \tag{32}$$

By Cauchy-Schwarz inequality, we can bound the dynamic regret in Eq.7 as:

$$\begin{aligned}
(Reg_d^T)^2 &= (\sum_{t=1}^{T} f(\boldsymbol{x}_t^*)/p_t(\boldsymbol{x}_t^*) - f(\boldsymbol{x_t})/p_t(\boldsymbol{x_t}))^2 \\
&\leq T\sum_{t=1}^{T}(f(\boldsymbol{x}_t^*)/p_t(\boldsymbol{x}_t^*) - f(\boldsymbol{x_t})/p_t(\boldsymbol{x_t}))^2 \\
&\leq T\sum_{t=1}^{T}(O(\beta_t^{1/2}\sigma_{t-1}(\boldsymbol{x_t})) + o(1/\sqrt{T}))^2 \\
&\leq O(T\beta_T\sum_{t=1}^{T}\sigma_{t-1}^2(\boldsymbol{x_t})) + o(1) \\
&\leq O(T\beta_T\sum_{t=1}^{T}\sigma_{t-1}^2(\boldsymbol{x_t})).
\end{aligned} \tag{33}$$

We can bound the dynamic regret as the same order of static regret in THEOREM 4.1. Precisely,

$$Pr\{Reg_T^d \leq O(\sqrt{\log(\delta|\mathcal{X}|)T(logT)^{d+1}})\} \geq 1-\frac{1}{\delta}. \tag{34}$$

while the deadline constraint $\{f(\boldsymbol{x_t}) > 1/T_{max}, \forall t\}$ is also satisfied probability at least $(1-\frac{1}{\delta})$.

□