A Framework for Executing Long Simulation Jobs Cheaply in the Cloud

Alan Nunes, Daniel Sodré, Cristina Boeres, José Viterbo, Lúcia Drummond,
Vinod Rebello, Luan Teylo, Felipe Portella, Paulo Estrela, Renzo Malini

A Framework for Executing Long Simulation Jobs Cheaply in the Cloud

Alan L. Nunes, Daniel B. Sodré, Cristina Boeres, José Viterbo, Lúcia M. A. Drummond, and Vinod E. F. Rebello
Instituto de Computação, Universidade Federal Fluminense, Niterói, RJ - Brazil.
E-mail: {alan_lira, danielbougleux}@id.uff.br, {boeres, viterbo, lucia, vinod}@ic.uff.br
Luan Teylo
Inria Centre at the University of Bordeaux, Bordeaux - France.
E-mail: luan.teylo@inria.fr
Felipe A. Portella, Paulo J. B. Estrela, and Renzo Q. Malini
Petróleo Brasileiro S.A. (Petrobras), Rio de Janeiro, RJ - Brazil.
E-mail: {felipeportella, paulo.estrela, renzo}@petrobras.com.br

***Abstract*—This paper presents the framework SIM@CLOUD that optimizes cost-related resource allocation decisions for simulation jobs in cloud environments. SIM@CLOUD offers comprehensive management of simulations throughout their execution life-cycle in the cloud, including the selection of Virtual Machine (VM) types across different regions and markets. By leveraging Spot VMs and application checkpointing, the framework transparently reduces the monetary costs associated with the execution without client intervention. Historical data analysis enables the prediction of simulation execution times, which is refined further by a dynamic predictor for adaptive VM selection. SIM@CLOUD is being deployed in an industrial setting and employs a cache-based storage solution to improve access latency to in-house data by VMs located in geographically distinct regions. An evaluation carried out on AWS EC2, using real oil reservoir simulations, demonstrates the effectiveness of the framework.**

***Index Terms*—Cloud computing, Spot instances, High performance computing, Scientific simulations, Resource and cost management.**

I. INTRODUCTION

Current simulation tools require substantial amounts of computing power to handle the massive amounts of data and calculations required to analyze complex phenomena accurately and quickly [1]. Simulations are widely employed in various fields, such as science, engineering, finance, and healthcare, to perform predictions and optimize the respective scenarios in ways that would otherwise be challenging, costly, or impossible to achieve through alternative methods [2] [3].

With the ever-growing size and complexity of simulation models and unpredictable computational demands, on-premise infrastructures may occasionally fail to meet the necessary processing needs [4]. Setting up new infrastructures for large projects is seldom straightforward, as it can be costly and may result in investments being tied up in local computing resources that remain idle once the project is completed. In contrast, cloud platforms offer almost limitless computing resources and storage space that average users can rent as necessary without the need for ownership and constant maintenance of the equipment [5]. By leveraging resources provided by cloud service providers, researchers and engineers can potentially run complex simulations with enhanced efficiency, cost-effectiveness, and less effort.

The myriad of computing options on offer from cloud providers makes choosing the resources necessary to incur the lowest cost a daunting task for users, especially given how the relative cost of each cloud service or VM type can significantly impact the decision [6] [7]. To keep cloud expenditure in check, especially for heavy cloud users, organizations must look closely at the different pricing options offered by cloud providers like On-Demand, Spot, Reserved Instances, and Saving Plans. Each option has advantages and considerations, and choosing the right one depends on the organization's specific needs and budget [8] [9].

In this context, this paper presents SIM@CLOUD, a framework whose goals are: to make wise resource allocation decisions that may be too complex for users or impractical for the resource provider to determine; to reduce monetary costs, and; to manage the overall performance of simulation jobs effectively in the cloud. SIM@CLOUD is designed to manage the entire life cycle of simulations running in the cloud, including being responsible for selecting the most appropriate VM type across different cloud regions and markets (On-Demand and Spot). This is achieved transparently without the need for any intervention from the client. To make this possible, the framework can request that the running simulation record one or more checkpoints. These checkpoints can be used to migrate the simulation to an alternate VM instance either out of necessity (when the current Spot is about to be revoked) or when it would be financially beneficial, for example, when a spot instance type becomes available, or its price falls sufficiently.

SIM@CLOUD also uses historical data from previous job executions to identify and analyze execution patterns and trends in resource usage. As a result, resource demands under different conditions can be understood better in order to predict simulation execution times. However, due to the computational nature of the simulation jobs themselves and the fact that applications are subject to interference when using shared cloud

resources, the execution time of even the same simulation may vary, possibly quite significantly. Therefore, a dynamic predictor is also employed to constantly monitor the progress and estimate the remaining execution time of the simulation. While the motivation for these predictions is to facilitate the identification of the most apt instance, its location, and in which market, this is not their primary purpose. The principal reasons why this information is necessary are the following: HPC jobs are typically submitted to job schedulers that apply priorities to jobs or allocate them to specific job queues depending on their estimated execution times, while future work aims to investigate the correlation between the likelihood of a Spot instance being revoked, the relationship between its price and the price of the equivalent On-Demand instance, and the (remaining) execution time of the simulation.

As a case study to evaluate the effectiveness of the proposed framework, we consider realistic oil reservoir simulations in an production environment. Production oil reservoir simulation studies involve three essential elements: (1) defining a reservoir exploitation strategy consisting of the number, positions, and types of wells, injection flows, *etc.*, (2) matching historical performance to calibrate the flow model, including parameters such as porosity and permeability, and (3) making predictions about future production rates (oil, gas, and water flows), pressure and saturation inside the reservoir, the composition of the fluids produced, *etc.* These studies allow engineers to evaluate different exploitation strategies using only a small fraction of the cost required to implement them in the real world [10]. SIM@CLOUD was evaluated on resources available in AWS EC2 using commercial reservoir simulation software with a semi-synthetic reservoir simulation model, known as Pre-salt 100, as the workload. This model represents the complex pre-salt reservoirs off the Brazilian coast, currently being explored by Petrobras, a globally recognized energy company and multi-million dollar public cloud user. Two versions are considered: a 20-year simulation (called SHORT-PRE-SALT) and a full 50-year simulation (called PRE-SALT).

The rest of this paper is organized as follows: Section II summarizes some of the existing work on moving simulations to the cloud. Section III introduces SIM@CLOUD, with its main components being detailed in Sections IV to VI. An evaluation of running reservoir simulation experiments on AWS using SIM@CLOUD under different scenarios is presented in Section VII, highlighting the benefits of the framework in terms of monetary costs. Finally, we close with a discussion in Section VIII and final remarks in Section IX.

II. RELATED WORK

Cloud platforms have become increasingly popular for scientific simulations due to advancements in specialized infrastructure and services tailored for scientific computing [11]. Eldred, Good, and Adams [12] presented a case study where a 3D Geocellular modeling simulation was moved from an on-premise cluster to a public cloud. This study aimed to understand the challenges and practicalities of transitioning simulations to the cloud and evaluating changes in application provisioning models. Temelkovski, Kiss, and Terstyanszky [13] explored extending domain-specific desktop applications for scientific simulations onto diverse cloud platforms. They utilized distributed heterogeneous clouds for molecular docking experiments, extending a specific tool to leverage various cloud computing resources through the CloudBroker Platform. Their experiments showcased the capability to execute simulations across different clouds and harness the on-demand scalability offered by cloud computing. Subsequently, Kiss [14] introduced the CloudSME Simulation Platform, aiming to streamline the development and execution of large-scale industrial and scientific simulations across heterogeneous cloud environments, ensuring the scalability and elasticity demanded by such applications.

Reservoir simulation is crucial in the Oil & Gas industry for decision-making, enhancing drilling efficiency, and reducing ecological impacts. Eldred *et al.* [10] introduced High-Performance Cloud Computing (HPCC) for efficient reservoir simulations, utilizing AWS and Eclipse software. Their framework included a cost model based on HPC resource use, showcasing benefits like flexibility, accessibility, and cost savings. Noor *et al.* [15] explored the transition from traditional to cloud-based reservoir simulation, emphasizing benefits like process simplification and reduced hardware costs with Software-as-a-Service. Their approach integrates cloud-based simulation services with containerization, Kubernetes optimization, and data analysis tools for efficient uncertainty analysis and history matching. Flister and Hopstaken [16] investigated reservoir simulations in a public cloud, focusing on Azure's HPC setup. They employed CycleCloud to automate HPC cluster deployment, dynamically adjusting resources based on workload demands. This approach optimized resource usage for running simulations using tNavigator and Eclipse reservoir simulators. They implemented job queues based on VM types to differentiate between memory and CPU power but did not incorporate spot instances or automated submission to job queues.

Portella and Souza [17] offered an overview of reservoir engineering and its simulation applications, focusing on a project involving Petrobras. They described moving a portion of the daily reservoir simulation demand to third-party cloud providers, addressing challenges like Network Attached Storage (NAS) synchronization and balancing between on-premises and cloud provisioning. Computer Modelling Group Ltd. (CMGL) offers CMG Cloud [18], a solution tailored for reservoir simulations in cloud environments. CMG Cloud aims to alleviate resource and budget constraints while ensuring productivity. However, it focuses primarily on providing a user-friendly interface for data submission, job monitoring, and result retrieval. Notably, it lacks features to optimize costs and does not consider the performance and pricing variability of different VM instances, including Spot and On-Demand, across multiple regions.

Our research aims to leverage Spot VMs effectively in an attempt to minimize the monetary cost of an application's

execution. These VMs offer computing resources at a notably lower cost than their On-Demand counterparts but can be terminated by the cloud provider at any moment. Portella *et al.* [19] introduced the MScheduler framework to reduce the costs of running long reservoir simulations on Spot VMs while ensuring job completion in the event of Spot revocation. However, that work does not consider the impact of data transfer times, given that running reservoir simulation models in the cloud often involves the use of significant amounts of in-house data.

In this paper, we have incorporated the data transfer time into the instance selection process in order to harness multiple geographical regions. In addition, we advanced the SIM@CLOUD framework in three key aspects. First, we have employed a machine learning model to predict job execution times, enhancing cost optimization — a novel approach not previously utilized in cloud reservoir simulations. Second, we developed a tool to estimate the remaining run time of a simulation job, which helps in selecting an appropriate instance type when the simulation needs to be reallocated. Finally, when making its selection, SIM@CLOUD considers the performance of the simulation on each available instance type, and the instance's current price.

III. SIM@CLOUD ARCHITECTURE

SIM@CLOUD is a framework designed to manage the life cycle of simulations running in the cloud. As shown in Figure 1, the framework is comprised of two main components: the LAUNCHER and the EXECUTION MANAGER, both home to distinct modules responsible for specific management procedures during a simulation's execution. To illustrate the behavior of the framework, Figure 1 presents a step-by-step example of operation.

First, a client submits their simulation request to the cluster job scheduler, in this case, SLURM, setting execution-related parameters such as the number of cores necessary to run the simulation, the batch file, and the directory (*work_dir*) where the output files should be written. With these parameters, SLURM starts the LAUNCHER on the head node, which invokes the ML-PREDICTOR (Section V), a machine learning component that estimates the execution time of the simulation. The LAUNCHER then invokes the INSTANCE-SELECTOR module (Section IV), which uses this estimated time along with other application and environment characteristics (*e.g.*, the overhead to perform checkpoints) to decide the VM instance type, the AWS region, and market where the simulation job should be executed. With this decision, the simulation job is now submitted to SLURM to allocate the chosen VM instance and initiate the execution of the job. During the execution, the EXECUTION MANAGER monitors the progress of the simulation via the DYNAMIC-PREDICTOR module (Section VI), which, when requested by the EXECUTION MANAGER, predicts the remaining execution time. This can be used to select a new VM instance on which to resume the simulation if the Spot VM being used is being revoked or a cheaper alternative to the current Spot or On-Demand instance becomes available.

When running on a Spot VM, the EXECUTION MANAGER uses the CHECKPOINT-RECORDER module to instigate the simulation to record checkpoints at regular intervals. Additionally, this module manages the multiple checkpoint files – maintaining a couple of recent files as a fail-safe measure – and, when necessary, restarts the application from the latest checkpoint available. The EXECUTION MANAGER is also responsible for receiving the interruption metadata alert provided by AWS that is sent two minutes before the Spot VM's lease is revoked [20]. Upon receiving the notification, the CHECKPOINT-RECORDER initiates a final checkpoint to preserve the simulation's current progress.

Should the use of the Spot VM instance be revoked (either by the LAUNCHER in an attempt to find a cheaper instance after a price update or by AWS in the case of a Spot interruption), the INSTANCE-SELECTOR uses the remaining time predicted by the DYNAMIC-PREDICTOR to help determine the best instance on which to resume the simulation. This revocation and restart process can repeat until the simulation finally finishes. At this point, the LAUNCHER saves the execution information in the HISTORY-DATABASE and notifies the client. The HISTORY-DATABASE stores all information related to the simulation execution, such as the region, market, type, price history of the selected VMs, and the simulation's execution time. This database enables posterior analysis of the performance and monetary cost of executions. The number of revocations a job can be submitted to is limited by SIM@CLOUD, and once reached, the job will be restarted on the best On-Demand instance available. The limit can be adjusted in relation to the estimated execution time of a job, but for this paper, it is fixed at five.

Note that SIM@CLOUD employs a storage solution composed of the following components: one shared file system located in the on-premise cluster and a cache located in each different region. This is an AWS-endorsed NetApp cache [21] solution that links each cloud region eligible to run the simulation, in particular for our experiments, the AWS regions *SA-EAST-1* and *US-EAST-1*. This cache setup ensures that data written into one region is replicated in another, enhancing availability and providing relatively fast access to all data related to the simulation's execution. It may also enable organizations to comply with data sovereignty requirements when the public cloud is the only computing environment that hosts the data (*e.g.*, the Brazilian government's regulations on minimum security requirements for the use of cloud services [22]).

IV. THE INSTANCE-SELECTOR

The selection of a VM instance by SIM@CLOUD does not require the intervention of the user or the system administrator. The INSTANCE-SELECTOR module selects an instance based on its current price, the estimated execution time of the simulation, and the amount of data that each simulation requires to be transferred from the on-premise environment to

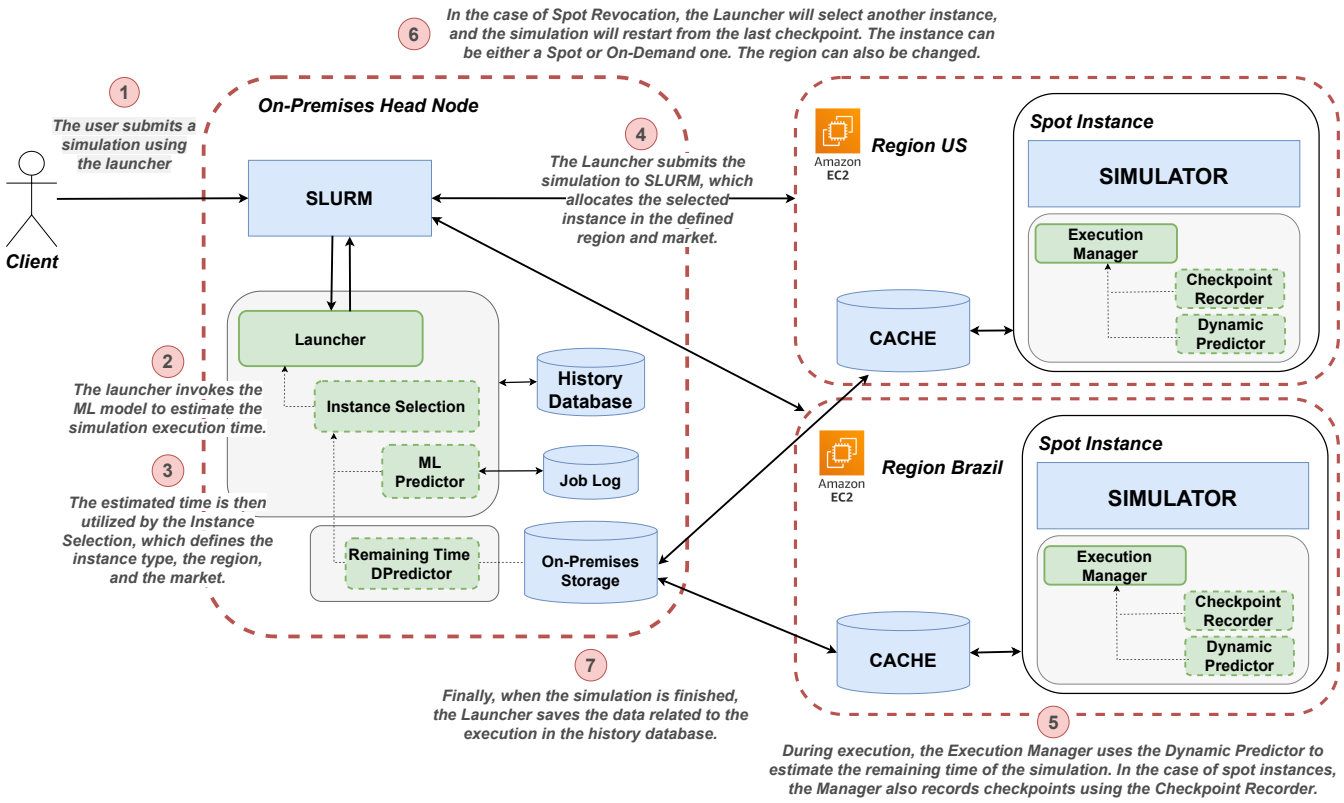


Fig. 1: A general overview of the SIM@CLOUD architecture. The green-colored components are those developed and presented in detail in this work, while the blue ones are off-the-shelf components integrated into the framework.

the cloud region in question. In addition, several static cloud performance parameters, obtained through prior benchmarking, are taken into consideration, such as the simulation's estimated performance on each usable instance type, the network bandwidth from the on-premise environment to each cloud region, the corresponding relative overhead of writing data back, and the average delay to record a checkpoint.

An overview of the instance selection strategy is outlined in Algorithm 1, which also takes into account several parameters, including a predefined list of suitable VM types (VMSET). From this list, the On-Demand VM with a performance most similar to nodes of the on-premise cluster, in the region closest to the on-premise storage, with enough CPU cores and memory to execute the simulation, is chosen as the baseline or reference VM instance (VMREF). The strategy also considers the data transfer bandwidth from the on-premise environment to the cloud infrastructure of each region R (BW_R). This value is used to provide an indication of the delay the simulation is likely to experience waiting for the I/O operations at initialization to complete. Of course, the bandwidth will be significantly higher if the same data is used subsequently in the same region, because of the cache. The CP-OHD_R represents the slowdown when an instance in the region R writes a checkpoint, relative to doing the same in *SA-EAST-1*, the closest region to the on-premise cluster (thus $\text{CP-OHD}_{\textit{SA-EAST-1}}$ is normalized to one).

Although Algorithm 1 is based on the work in [19], there are three clear distinctions. Whereas in [19], the application's execution time was a parameter provided by the user; here, this is estimated by the ML-PREDICTOR and the DYNAMIC-PREDICTOR. Algorithm 1 considers the I/O time to migrate the data from the on-premise environment to the cloud; and even in this case, the volume of data is not required from the user as SIM@CLOUD automatically computes this $tsize$ by scanning the simulation job's file directory and summing the respective file sizes. Finally, the algorithm considers the simulation's distinct performances on different instance types.

Table I explains the notation used in Algorithm 1. The algorithm requires three parameters: $tsize$, $restarted$, and VMREF, and uses a set of predefined constants, such as the checkpoint interval (INTERVAL), the checkpoint delay, CP-LAT, and the previously discussed CP-OHD_R and BW_R.

Firstly, Algorithm 1 obtains the simulation time from either the ML-PREDICTOR or the DYNAMIC-PREDICTOR, in which the estimate, made on a previous instance type, is normalized to the performance of VMREF. If a Spot VM is chosen and revoked by the provider, the algorithm is notified via $restarted$ and, for all subsequent selections, the value provided by the DYNAMIC-PREDICTOR is used. Secondly, VMREF is chosen as the initial candidate for vm_{best}, and its execution cost is calculated. The algorithm then iterates through all the available VM types in VMSET. The I/O

TABLE I: Notation used in Algorithm 1.

Predefined Values	
VMSET	The set of instance types available, with their corresponding region, market, price, and performance factor PF.
$PF(vm_i)$	A measure of the performance of the simulation model on vm_i relative to VMREF.
INTERVAL	The simulation time that should elapse between consecutive checkpoints.
CP-LAT	The average simulation time lost while recording a checkpoint.
CP-OHD$_R$	Slowdown factor to write a checkpoint from a VM in region R relative to the region of VMREF.
BW$_R$	Estimated I/O bandwidth between the on-premise cluster and region R.
Input Parameters and Variables	
$tsize$	The total volume of simulation data to be transferred from the on-premise environment to the cloud cache.
$restarted$	A boolean variable that indicates whether or not the simulation is being restarted.
VMREF	The baseline reference On-Demand VM instance.
$makespan$	The estimated execution time provided by the ML-PREDICTOR or the remaining execution time estimated by the DYNAMIC-PREDICTOR.
n_ckp	Estimated total number of checkpoints likely to be recorded.
$time\_ckp$	Total time required to record n_ckp checkpoints.
$price(vm_i)$	Price of instance vm_i per hour.
vm_{best}	The VM on which the simulation is estimated to incur the lowest cost.
COST$_{best}$	The estimated cost to execute the remainder of the simulation on vm_{best}.

Algorithm 1: Makespan prediction, performance, price and I/O overhead based VM Selection

Input: $tsize, restarted$, VMREF
1: {*** Step 1: Makespan Estimation ***}
2: **if NOT** $restarted$ **then**
3: $makespan \leftarrow$ ML-PREDICTOR()
4: **else**
5: $makespan \leftarrow \frac{PF(\text{VMREF})}{PF(vm_{prev})} \times$ DYNAMIC-PREDICTOR()
6: **end if**
7: {*** Step 2: VM Selection ***}
8: $vm_{\text{best}} \leftarrow$ VMREF
9: COST$_{best} \leftarrow price(vm_{\text{best}}) \times makespan$
10: **for** $vm_i \in$ VMSET **do**
11: $R \leftarrow Region(vm_i)$
12: MAKESPAN$' \leftarrow PF(vm_i) \times makespan + (\frac{tsize}{\text{BW}_R})$
13: **if** vm_i is Spot **then**
14: $n\_ckp \leftarrow \lceil PF(vm_i) \times makespan/\text{INTERVAL} \rceil - 1$
15: $time\_ckp \leftarrow n\_ckp \times$ (CP-LAT $\times$ CP-OHD$_R$)
16: MAKESPAN$' \leftarrow time_ckp +$ MAKESPAN$'$
17: **end if**
18: **if** COST$_{best} > (price(vm_i) \times$ MAKESPAN$')$ **then**
19: $vm_{\text{best}} \leftarrow vm_i$
20: COST$_{best} \leftarrow price(vm_i) \times$ MAKESPAN$'$
21: **end if**
22: **end for**
23: **return** vm_{best}

time required to move the simulation's data to the respective region and the estimated simulation time on that instance is computed. In the case of a Spot VM, the checkpoint overhead is considered by predicting the number of checkpoints and their duration from the parameters INTERVAL, $makespan$, CP-LAT, and CP-OHD$_R$. If a lower cost is found, vm_{best} and COST$_{best}$ are updated. Finally, after evaluating all of the instances in VMSET, vm_{best}, the instance that will incur the lowest estimated cost, assuming revocations do not occur, is returned.

V. THE ML-PREDICTOR

Employing a machine learning-based (ML) predictor was driven by two reasons: (i) users may not always provide precise estimations for their application's execution time, and (ii) the need for SIM@CLOUD to be transparent and user-friendly by requiring minimal user input. The ML-PREDICTOR was built using *Weka* [23], an open-source software that implements traditional ML algorithms for data mining tasks.

The *Job Log* (the output log history of a job scheduler) contains the raw dataset used to build and retrain the ML-PREDICTOR at predetermined intervals. The analysis here is based on a recorded set of 5,070,674 jobs submitted from April 9, 2021, to December 31, 2023, to the on-premise cluster of Petrobras through SLURM [24], an open-source workload manager widely employed in clusters and in data centers worldwide. The job records were gathered via SLURM's *scontrol* command, which allows the streaming of extensive information about each job to a separate text file. Of the jobs submitted, 4,027 were purposefully relocated to Amazon EC2 and executed in VMs with computing resources comparable to the on-premise nodes. A filtering technique was applied that takes into consideration the following rules: (i) discard jobs with an error status, (ii) discard test jobs, and (iii) discard jobs not related to specific scripts. The raw dataset was reduced from 5,070,674 to 2,834,851 job records after filtering, representing a reduction of 44.09%. Since each job record comprised 46 attributes of varying types and values, three main actions defined the most relevant attributes for the training and testing datasets: *Attributes Analysis*, *Attributes Transformation*, and *Attributes Subset Selection*.

The *Attributes Analysis* determines which attributes should be discarded or transformed. Some free-form attributes, in which users could enter unstructured text, required strategic transformation to allow the extraction of relevant information. The timestamp attributes were transmuted into categories, so the hour of the day (ranging from 0 to 23) and day of the week (ranging from 0 to 6) could be extracted. Some attributes were discarded, as they lacked helpful information with regard to the goal of estimating the execution time. Some correlated attributes, *i.e.*, attributes that can be derived from each other, were also removed.

The *Attributes Transformation* involves pre-processing attributes that are not feasible for training an ML model due to the variety of values. Among them, the *nodes*, *script*, and *work_dir* attributes were transformed as follows. The *nodes* attribute contains the list of nodes that executed a job. Its values were truncated, retaining only their first four characters. The resulting attribute was named *nodes_prefix*. The *script* attribute contains the string representing the batch script submitted by a job. Its values were transformed through regular expression patterns to obtain, when available, the name and version of the reservoir simulator used by the job. The resulting attribute was named *employed_simulator*. Finally, the

work_dir attribute contains the string representing the job's working directory (logical path). Its values were transformed into common prefix paths, as it was assumed that near-identical paths are indicative of jobs with similar durations, reducing their data dimensionality without a substantial decline in prediction performance. The resulting attribute was named *work_dir_common_prefix*.

The *Attributes Subset Selection* extracts an appropriate subset of attributes regarding the ML problem to be solved. The supervised machine learning paradigm is routinely adopted for the job duration prediction task. In this paradigm, an algorithm receives a set of labeled training data, *i.e.*, whose values are known, and makes predictions for data not seen during the training [25]. The regression method is widely used to predict the duration of the job, and is of a numerical type. On the other hand, the classification method can be used to categorize and predict the duration interval associated with a job. The latter approach was adopted in this work, as it has previously indicated a higher prediction performance in comparison to some regression models [26]. In this sense, a nominal attribute representing the duration of jobs had to be set as the target attribute. Therefore, the values of the *elapsed* attribute, which stores the job's duration in seconds, are transformed into classes (or bins) of job duration, with lower and upper bounds that describe the duration range of the jobs, as shown in Table II. The 15 classes of duration intervals were defined considering three spans: 30 minutes for short-term jobs (classes 0 and 1), one hour for medium-term jobs (class 2), and steps of two hours for long-running jobs (class 3 onwards). The chosen granularities have practical utility for the VM selection stage as they were chosen to address the unbalanced distribution of job durations, the majority being predominantly short-term jobs. The resulting attribute was named *job_duration_class* and designated as the target attribute. Finally, the subset of nontarget attributes was selected through the *Weka*'s *InfoGainAttributeEval* evaluator, which assessed the worth of each attribute regarding the target one for a random sample containing 1% (50,707 jobs) of the raw dataset. Table III summarizes the nontarget attributes, selected assuming a minimum worth score of 0.011.

TABLE II: Classes of the target attribute *job_duration_class*.

Class	Duration Interval (in seconds)	Class	Duration Interval (in seconds)
0	[0, 1800)	8	[43200, 50400)
1	[1800, 3600)	9	[50400, 57600)
2	[3600, 7200)	10	[57600, 64800)
3	[7200, 14400)	11	[64800, 72000)
4	[14400, 21600)	12	[72000, 79200)
5	[21600, 28800)	13	[79200, 86400)
6	[28800, 36000)	14	[86400, ∞)
7	[36000, 43200)	—	—

The *J48* model, available on Weka and often referred to as a statistical classifier, was used to train and evaluate the ML-PREDICTOR, as it provided better results than some regression models in previous tests [26]. It implements the C4.5 [27] algorithm, one of the most renowned decision tree algorithms, which uses the concept of information entropy. At each node of the tree, the attribute that most effectively divides the set of samples into subsets enriched in one class or another is selected, given that the division criterion is the normalized information gain, *i.e.*, the entropy difference. The filtered dataset was randomly split into two parts: 80% (2,267,881 jobs) as the training dataset and the remaining 20% (566,970 jobs) as the test dataset. The trained predictor correctly classified the job duration interval of 76.01% of the test instances.

TABLE III: Subset of selected non-target attributes.

Attribute	Description
work_dir_common_prefix	Common prefix of the job's working directory (logical path).
username	User who submitted the job.
groupname	Group name associated with the job.
account	Account associated with the job.
employed_simulator	Name and version of the reservoir simulator.
tres_req	Trackable resources requested for the job.
ntasks	Number of parallel processes executed by the job.
partition	Partitions (queues) in which the job was submitted for.
time_limit	Timelimit in seconds for the job execution.
nodes_prefix	First four characters of nodes that executed the job.
qos	Scheduling priority defined for the job execution.
queue_wait	Time in seconds in which the job was queueing until started.
job_submit_hour_of_day_class	Class associated with the hour of the day in which the job was submitted.
job_eligible_hour_of_day_class	Class associated with the hour of the day in which the job was eligible to start.
job_start_hour_of_day_class	Class associated with the hour of the day in which the job started.
alloc_node	Nodes allowed to execute the job.

When predicting the duration interval of jobs executed outside the data collection period, specifically for 45,007 jobs executed in the first ten days of January 2024, the accuracy was 57.79%. As stated in recent work [28], there are two main reasons for this drop in prediction performance: (i) accuracy degrades over time due to the non-stationary factor of workloads (changing job profiles), which is typical across most job records; and (ii) SLURM's records, although extensive concerning the overall characteristics of the jobs, lacks fine-grained information about the executed application. Both causes are being investigated and addressed, the first through a data drift policy that detects and tracks changes in data distribution, properties, or behavior over time, and the second through an extended dataset that reunites the information from both SLURM's records and the parameters of the reservoir simulation, such as the physical system (e.g., geology and fluids data) and the production strategy (e.g., number and location of wells, water flows injection). Finally, the ML-PREDICTOR module supports more complex machine learning models, such as Ensemble algorithms that combine the predictions of multiple models. Although complex models

can capture intricate patterns and nonlinear relationships in the data, they are much more challenging to interpret and are prone to overfitting, noise, or bias. Therefore, using interpreting models, such as *J48*, is a prudent choice for several reasons, including allowing data scientists to comprehend how the model makes predictions and identify any potential biases or errors. In addition, the interpretability of models can provide valuable insights into the underlying structure of the data and help in feature selection and model optimization.

VI. The Dynamic-Predictor

The behavior of reservoir simulations can be affected by a number of factors, including the choice of fluid modeling (*e.g.*, black oil, compositional); the sparse matrix solver, and numerical parameters adopted (*e.g.*, the maximum number of simulation days per time step); the size and resolution of the geological model; and the number, positioning, and scheduling (*i.e.*, openings and closings) of the wells. Simulation tools iterate through models one step at a time, with each "time step" potentially having a different duration for efficiency and accuracy [29]. Larger time steps reduce the total number of iterations and tend to shorten the *Total Execution Time* (TET) of the simulation. However, large time steps may sacrifice precision, necessitating recalculations with smaller time steps. Additionally, simulation execution times are influenced by the hardware architecture, CPU cores, memory, and I/O bandwidth due to parallelization. The Dynamic-Predictor estimates the remaining run time of a reservoir simulation based on the application's log generated during its execution. This log file provides the user with key information about the state of the simulation model after each completed time step.

To evaluate the progress rate of a simulation and predict the TET, the Dynamic-Predictor adopts the concept of a sliding time window to represent a sample temporal period of the simulation. Associated with each window of approximately N simulation days is its corresponding execution time. Note that the number of time steps may vary between different windows. To derive the execution time for a window, the last time step to be completed and the last modification date of the log file are monitored at regular intervals that define a *Monitoring Interval*. Associated with each Monitoring Interval i is the *Simulation Period* $SP(i)$ elapsed since the start of the simulation, which is calculated by scanning the output log file and aggregating the number of simulation days processed during each completed time step. The *Current Execution Time* (in seconds) $CET(i)$ is the difference between the time at which the log file was created and the time at which it was last updated. At the end of each Monitoring Interval i, the Dynamic-Predictor computes: the average progress rate $PR(i)$ during the last time window between an earlier interval j and i as $PR(i) = (CET(i) - CET(j))/(SP(i) - SP(j))$

The *Estimated Remaining Time*, $ERT(i)$, depends on the number of remaining simulation days, so the Dynamic-Predictor presumes that $ERT(i) = PR(i) \times (TSP - SP(i))$ will be an acceptable approximation, given that the simulator obtains TSP from the simulation's stop condition, be it the last simulation date, a maximum number of time steps, or a maximum wall clock time.

In practice, simulations do not exhibit constant rates of progress during their execution, and therefore this approach is highly susceptible to time-varying, non-negligible prediction errors. To help address this, N is sufficiently large so that there is enough historical data to identify cyclic events and sustained progress rates but short enough to adjust its estimate quickly without extenuating fluctuations in its predictions. Furthermore, instead of providing a single ERT value, several variations of this linear extrapolation approach are applied simultaneously to return an interval for the estimate. The smaller the interval, the higher the confidence one might have in the tool's prediction.

Dynamic-Predictor currently employs four variations of the sliding-time window concept, while others can be easily added later:

- *Naive Window* where $PR(i)$ is determined over the total period of the simulation completed so far. If $PR(i)$ slows over time, this prediction can be used as a lower bound;
- *Fixed Sliding Window* calculates $PR(i)$ based on the behavior over the last N simulation days;
- *Adaptive Sliding Windows* - Before calculating $PR(i)$ for a given window, the window's interval and duration may be adjusted, should temporary but significant slowdowns be detected. To detect such changes in the simulation's behavior, the tool analyzes the progress rate between successive time steps. The following variants are currently utilized: *The Optimist* reduces the interval of the window by ignoring periods where slowdown events occur; *The Pessimist* reduces the time interval of the window so that it begins at the time step at which the slowdown in the progress rate was first detected.

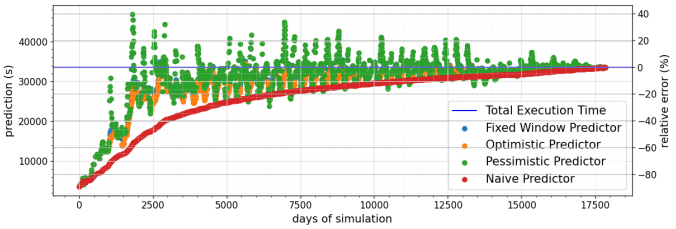


Fig. 2: Changing estimates of the total execution time by the four predictors during the simulation's execution.

Figure 2 presents the behavior of the four predictors for Pre-salt 100×100 (from the year 2000 to 2048), during its execution of $\approx 33,575.88$ seconds. Each dot represents an individual estimate of TET by a single predictor; the closer to the blue line (the actual total execution time), the better. The further the simulation progresses (*i.e.*, moves to the right), the more accurate each of the predictors becomes. As expected, Figure 2 shows that the predictions during the first $\approx 2,500$ simulation days are not as precise as the others since, during this period, the simulation exhibits higher progress rates than during the rest of the simulation, which leads to the predictor

underestimating the TET. Unfortunately, these progress rates can vary over time as they depend on the characteristics of the simulation, the model, and the data. To avoid having to design an overly complex generic dynamic predictor for different types of simulation, SIM@CLOUD opts to rely more on the ML-PREDICTOR during the initial phase of the simulation's execution.

Experimental evaluations with a variety of distinct models have provided positive indications of the predictor's potential. For example, the DYNAMIC-PREDICTOR predictions are within 10% of the total execution time of the Pre Sal 100×100 49-year simulation for 82% of its duration.

VII. EXPERIMENTAL EVALUATION

This section experimentally demonstrates the automated management of SIM@CLOUD and gives some indication of the gains in terms of monetary costs. Table IV presents the values of the predefined parameters obtained during a prior phase to benchmark cloud performance in different regions. Due to space limitations, the tests conducted are not fully reported here, but instead are detailed in the Web Supplementary Material at https://gitlab.com/u693/simCloud.

TABLE IV: Predefined parameters used in the experiments.

Parameters	**Values**
VMREF	c5a.24xlarge
VMSET	c5.24xlarge, c5a.24xlarge, c6a.24xlarge, c6i.32xlarge
INTERVAL	1800 seconds
CP-LAT	13 seconds
CP-OHD$_{US\text{-}EAST\text{-}1}$	2.25
CP-OHD$_{SA\text{-}EAST\text{-}1}$	1.0
BW$_{US\text{-}EAST\text{-}1}$	55 MB/s
BW$_{SA\text{-}EAST\text{-}1}$	250 MB/s

In order to evaluate the impact of Spot revocations on the cost and makespan of simulations under SIM@CLOUD, it is essential to have some control over when AWS reclaims the Spot instance. Here, the process is emulated by adhering to the AWS recommended guidelines for evaluating Spot instances [30]. The method involves using the Amazon EC2 Metadata Mock (AEMM) tool to replicate the two-minute warning issued prior to Spot revocation. Using AEMM, another tool was developed to mimic Spot revocations by: 1) calculating the VM's revocation time using a Poisson distribution; 2) updating the VM's metadata to include a termination alert; and 3) shutting down the Spot VM. The discrete probability distribution is well suited to predict the occurrence of events within a specific time frame, including the timing of Spot VM terminations, as supported by several studies [31]–[33]. With a parameter λ representing the average number of revocations per hour, the termination time is determined by sampling the Poisson distribution and multiplying the result by 3,600 seconds, thus defining τ as the duration for which the VM will be available. The tool tracks the VM's uptime and generates a termination alert at $\tau - 120$ seconds. For the evaluation, four termination rates were considered (λ): 0.1, 0.5, 0.8, and 1.0. As λ increases, the probability of terminating a Spot VM sooner increases, with $\lambda = 1$ representing executions with a high likelihood that the Spot is reclaimed within the first hour.

A. The influence of Spot market price variability

The effect of varying Spot market prices and availability over time is illustrated by Figure 3. The same set of experiments was executed at two different times, in this case, an interval of one week between each set. Each experiment comprises a series of four executions, one for each value of λ, of the same SHORT-PRE-SALT simulation. This series is executed three times to define the set, with the average execution times and costs being presented in the figure.

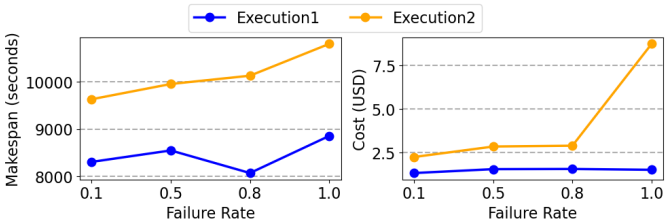


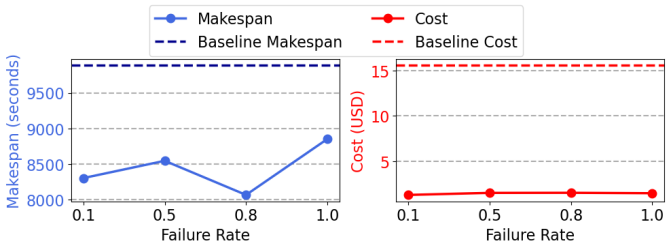
Fig. 3: Comparison between two time shifted series of executions, *Execution1* and *Execution2*, of the SHORT-PRE-SALT model, one week apart.

In *Execution1* (the blue lines), Spot market conditions meant that SIM@CLOUD generally explored several newer generation Spot instances (c6a and c6i) to obtain shorter runtimes. In contrast, in *Execution2* (the orange lines), the majority of executions relied on older generations of instances (c5 and c5a). For the experiment of the second week, *Execution2*, the prices of all Spot instances were significantly higher (almost the double in some cases) and varied during the execution to a greater degree than during the experiment of the week before. The differences were large enough to change the SIM@CLOUD's preferred instance types. Furthermore, with higher prices being a reflection of demand, this also means that SIM@CLOUD's first choice instance may not have been available, and a less optimal choice might have had to be selected. On the other hand, in the case of $\lambda = 0.8$ of *Execution1*, the availability of a c6i Spot instance at an unusually low price allowed one execution to execute quicker without significantly increasing the average cost. However, one of the *Execution2* experiments with $\lambda = 1.0$ suffered five revocations and was finalized on an On-Demand c6i instance, causing this execution to be four times more expensive than the others with the same λ value. Thus, no single instance type always provides the cheapest execution. So, while *Execution1* exhibited better performance and lower costs compared to *Execution2*, the variability in price and availability underscores the need for an instance selection algorithm that checks the prices of instances before every job submission.

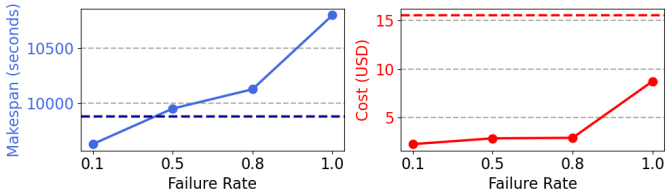
B. Reducing costs with Spot instances

Figures 4a, 4b and 4c present the average makespans and monetary costs of the SIM@CLOUD Instance selection

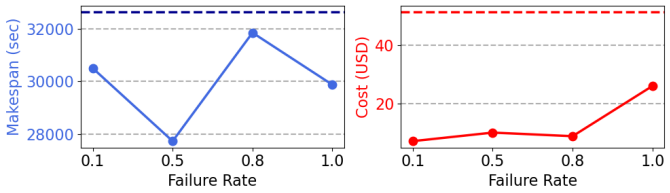
scheme, for three executions per failure rate, for the SHORT-PRE-SALT and PRE-SALT simulation models, respectively. As can be seen in the three figures, compared to the baseline, VMREF, *i.e.*, the reference On-Demand VM (and one of the cheapest instances in VMSET at the time) in the home region (*SA-EAST-1*), SIM@CLOUD was able to reduce the monetary cost significantly in all scenarios, while also shortening the execution time in some.



(a) SHORT-PRE-SALT *Execution1*



(b) SHORT-PRE-SALT *Execution2*



(c) PRE-SALT

Fig. 4: Costs and Makespans for (a) SHORT-PRE-SALT and (b) PRE-SALT. The dashed lines represent the baseline execution (using the cheapest On-Demand VM in the home region)

Even in the worst-case scenario ($\lambda = 1.0$), the savings in terms of monetary costs were as high as 43.87% for SHORT-PRE-SALT *Execution2*, and up to 90.39% for SHORT-PRE-SALT *Execution1*. While in the case of the former, this was achieved at the expense of the makespan, which increased 9.28% in this exact scenario as a consequence of the need for checkpointing when exploiting Spot VMs, SHORT-PRE-SALT *Execution1* obtained an average reduction of 10.42% in the makespan. In addition, revocation and restart overheads are incurred whenever a Spot VM is reclaimed, during which the time to select a new VM, boot it up, and restart the simulation are included. Thus, when the failure rate increases, an increase in the makespan may be observed, as shown quite clearly in Figure 4b. For $\lambda = 0.5$ and $\lambda = 0.8$, the monetary reduction for the SHORT-PRE-SALT *Execution2* was 81.78% and 81.46%, respectively, against an increase in the makespan of 0.70%, and 2.48%. For *Execution1*, cost reduction was slightly better, 90.13% and 90.07%, respectively, while the makespans also decreased by 13.52% and 18.35%.

In particular, significant gains can be seen in both metrics for the case where $\lambda = 0.1$: **reductions** of 91.58% (*Execution1*) and 85.74% (*Execution2*) in the monetary costs and 15.97% and 3.92%, respectively, in terms of the makespans. In this scenario, the selection heuristic acquired a Spot VM with better performance than the baseline VM at a lower price. Furthermore, the relatively short execution time of SHORT-PRE-SALT tends to mean that no Spot revocations occur, and therefore, no additional overheads were incurred to migrate the simulation. This case demonstrates the ability of the selection heuristic to explore cheaper resources in the Spot market.

For the PRE-SALT model, the results shown in Figure 4c are somewhat analogous to those of the SHORT-PRE-SALT model; however, here, average reductions in both the monetary costs and makespans, in relation to the baseline, in all scenarios were obtained. For the scenario where λ was 0.1, SIM@CLOUD achieved an average reduction in the cost of 85.88% and in the makespan of 6.50%. For cases where $\lambda = 0.5$, 0.8, and 1.0, SIM@CLOUD was able to lower costs by 80.19%, 82.59%, and 49.11%, respectively. Meanwhile, the makespans were shorter by 14.96%, 2.38%, and 8.42%, for the respective scenarios. Note that the execution times are also greatly affected by the availability of Spot instances in a given region at the time of job submission. While Spot instance prices can change during execution, the principal reason for the higher costs for both models when $\lambda = 1.0$ is due to SIM@CLOUD selecting an On-Demand instance to terminate the respective simulations, given that the execution reached the limit of allowed revocations.

These results demonstrate the effectiveness of SIM@CLOUD in reducing the overall monetary cost independently of the simulation's execution time. Furthermore, for longer simulation models, even under high failure rates, smaller makespans can be achieved with cost reductions in relation to the On-Demand reference VM.

To better understand the results obtained by SIM@CLOUD, let's analyze two distinct scenarios for the PRE-SALT execution (Fig. 4c): the *best-case scenario*, in terms of the number of VM revocations, when $\lambda = 0.1$ but **no** Spot revocations occurred, and; the *worst-case scenario* when $\lambda = 1.0$, where **several** revocations occurred during execution. Table V details the scheduling breakdown of the execution of each scenario, while the results in Figure 4c presented the average values of three executions.

In the *best-case revocation scenario* ($\lambda = 0.1$), SIM@CLOUD selected a Spot VM of type c6a.24xlarge in *SA-EAST-1* (the region closest to the on-premise cluster) that did not suffer revocation and was therefore used for the entire execution. At the moment of job submission, this instance offered the best cost-performance. During the execution, the price of the Spot instance varied between US$0.76 and US$0.81 per hour, so that the total execution cost reached a total of US$6.93. Since there was no need to restart the simulation, no request was made to the DYNAMIC-PREDICTOR for an ERT. Given that the ERT depends on the instance that was used, the value provided by the DYNAMIC-PREDICTOR is adjusted to

TABLE V: Individual executions of the PRE-SALT model for $\lambda = 0.1$ (best-case scenario) and $\lambda = 1.0$ (worst-case scenario).

λ	Instantiation	Selected VM	Region	Market	Exec. Time (s)*	ERT_{ref} (s)	ML-PREDICTOR	Cost**
0.1	t_0	c6a.24xlarge	*SA-EAST-1*	Spot	31,456		36,000 (s)	US$6.93
1.0	t_0	c6a.24xlarge	*SA-EAST-1*	Spot	3,718	32,627	36,000 (s)	
	t_1	c5.24xlarge	*SA-EAST-1*	Spot	10,921	27,429		
	t_2	c6a.24xlarge	*SA-EAST-1*	Spot	263	-		
	t_3	c6i.32xlarge	*SA-EAST-1*	Spot	3,718	20,506		
	t_4	c6a.24xlarge	*SA-EAST-1*	Spot	263	-		
	t_5	c6i.32xlarge	*US-EAST-1*	On-Demand	11,864			
					Total:30,747			US$22.69

* "Exec. Time" is the wall clock time from job submission to termination, and thus, in addition to executing the simulation, this also includes the time required to allocate the VMs, the overhead of the rollback/restart procedure, and any additional recomputation by the simulator.

** If an On-Demand VMREF in the region *SA-EAST-1* were used, the cost to simulate the PRE-SALT model would have been US$51.32, with an execution time of 32,620 seconds. Alternatively, with the faster On-Demand c6i.32xlarge instance in the region *US-EAST-1*, it would have taken 28,248 seconds and incurred a cost of US$42.69.

represent the remaining time on the reference VM (VMREF). Note that the ML-PREDICTOR classified the simulation as a class 7 job with an estimated running time between 36,000 and 43,200 seconds, if executed on the VMREF, a c5a.24xlarge instance. Using the lower bound of this range, and the instance performance factors (PF), SIM@CLOUD can estimate that the execution will require at least 30,787 seconds on the selected c6a.24xlarge instance.

For the *worst-case revocation scenario* ($\lambda = 1.0$), the execution faced five Spot revocations that resulted in a total monetary cost of US$22.69 and a total execution time of 30,747 seconds. As indicated in Table V, except for the revocation at the end of t_4, the framework migrated the simulation to another Spot VM four times. Moreover, the simulation was migrated to different VM types and across various regions: the final On-Demand VM selected being a c6i.32xlarge in *US-EAST-1*. While On-Demand instance prices in *SA-EAST-1* are approximately 50% higher than their corresponding instances in *US-EAST-1*, the same instances in the Spot market in *SA-EAST-1*, at the moment of execution, were 22% to 37% cheaper, the only exception for the VMs in VMSET, being c5a.24xlarge, our VMREF, which was 40% more expensive.

Having again chosen the best Spot option, that VM was revoked after a little more than an hour. After a revocation, to avoid selecting the same instance type in the same region immediately in sequence, SIM@CLOUD places this option in quarantine for a predefined period of time. The motivation is the high likelihood of that instance being revoked again by AWS in the short term future. The framework thus utilized the estimate provided by the DYNAMIC-PREDICTOR, normalized to the VMREF instance (ERT_{ref}), to choose from the remaining choices in VMSET, the Spot VM c5.24xlarge again in *SA-EAST-1*. This second instance was also revoked after three hours, time enough for the first instance to be freed from quarantine and available for selection at the end of t_1.

The instances selected for steps t_2 and t_4 were revoked within 5 minutes of being requested. As this time also includes instantiating and booting the VM and the overheads (associated with I/O and rollback recovery) to restart the simulation, this meant that the simulation could not advance sufficiently for the DYNAMIC-PREDICTOR to provide a revised ERT, and forced SIM@CLOUD to use the previous ERT value (or, should one not be available, use the difference between the ML-PREDICTOR's estimate and the sum of the execution times of the previous steps). After a given number of revocations (in this case, five), SIM@CLOUD opts to move the execution to On-Demand market in order to avoid further delays. The number of revocations can be adjusted to investigate trade-offs between cost and execution time.

These executions highlight SIM@CLOUD's ability to reduce monetary costs by leveraging the Spot market, even in scenarios with several revocations. For the worst-case scenario, more than 60% of the simulation was executed on Spot VMs (18,883s on Spot versus 11,864s on On-Demand), directly impacting the monetary cost. As seen in the baseline case, presented in Figure 4c, using the VMREF, an On-Demand c5a.24xlarge VM in *SA-EAST-1*, for the entire simulation would result in a much higher cost (more than twice as much, US$51.32) and a slightly longer execution time (32,620s). Note, too, that VMSET included an oversized c6i.32xlarge instance rather than the cheaper c6i.24xlarge version to show that even expensive instances can be viable options in both the Spot and On-Demand markets.

VIII. DISCUSSION

One of our goals was to compare the SIM@CLOUD framework with the solution provided by AWS. However, after further investigation, it turns out that such a comparison would be unfair since the tools that AWS currently provides their clients with to request Spot instances suffer from a couple of limitations. First, AWS suggests that users do not request specific VM types, but instead specify a list of minimum capacity requirements. Second, the user should choose one of three allocation strategies: Lowest Price (which chooses the cheapest Spot VM, but has a higher chance of being revoked); Capacity Optimized (which will try to choose an instance type with the lowest chance of being revoked), and; Price-Capacity Optimized (that looks for the lowest priced instance with high availability). None of these strategies guarantee, or are even

likely, to choose the instance that provides the lowest cost. Secondly, the scope of these strategies is limited to a single region. SIM@CLOUD overcomes both of these limitations.

Therefore, instead of directly comparing the allocation strategies of AWS with our approach, we briefly analyze the impact imprecision in makespan predictions has on the selection of the VM by SIM@CLOUD. To exemplify this analysis in a more intuitive way, we will consider the scenario of choosing between the same instance type in the Spot and On-Demand markets of the same region. This same analysis can be extended in a similar fashion to the more general case of heterogeneous instance types and different regions, as considered by SIM@CLOUD. Let x_s be the total execution time (makespan) of the simulation, p_s, the current hourly price of the Spot instance under consideration (here, let us assume that this price does not change during the execution), and y_s be the duration (portion) of the makespan that characterizes the overhead for checkpoints. In turn, x_o and p_o are, respectively, the makespan on the On-Demand instance and its hourly price. The instance selected as the cheapest option is thus determined by comparing the costs of the two options, the Spot instance being selected over the On-Demand one if the following condition is true:

$$x_s.p_s < x_o.p_o \implies \frac{x_o + y_s}{x_o} < \frac{p_o}{p_s} \implies \frac{y_s}{x_o} < \frac{p_o - p_s}{p_s} \quad (1)$$

The left-hand side of these equations shows that the makespan plays a role in determining the cheapest market. To illustrate this numerically, consider the following hypothetical case. Let the Spot instance price be US$3.00 per hour, the On-Demand instance price be US$3.10 per hour, and the checkpoint INTERVAL be 30 minutes. Consider the following two scenarios: In Scenario 1, the ML or dynamic predictor provides an accurate execution time x_{o1} equal to 59 minutes, with an additional time, y_{s1}, of 1.5 minutes required to record the single checkpoint if a Spot VM is chosen. In this case, the Spot option would be chosen since the monetary cost of the On-Demand one would be US$182.9/60 against a cost of US$181.5/60 for the Spot instance. In Scenario 2, suppose that an imprecisely estimated execution time $x_{o2} > x_{o1}$ equal to 61 (x_{o2} was provided, implying an additional time y_{s2} of 3 minutes to record two checkpoints if a Spot VM is chosen. Now, in this case, the On-Demand instance would be chosen, since its estimated cost would be US$189.10/60 versus US$192.00/60 for the Spot one. Hence, although the only difference in the metrics was a 3.4% overestimation of the execution time, the framework would choose the more expensive market, incurring an additional expense of US$1.4/60.

This might appear to the reader as an obstacle to embracing the proposal, especially given the extreme difficulty of predicting the execution time of applications running in the cloud. Furthermore, one might expect that the excess expenditure would continue to increase with longer executions. This is not true, and erroneous instance selections by SIM@CLOUD are not as common as one might expect. Although our expectations have always been that the makespan predictions would be imprecise, the principal motivation for adopting this approach is the simplicity and speed of the selection algorithm. Secondly, the inequality of Equation 1 can be approximated by Equation 2 which is not affected by the precision of the makespan prediction:

$$\frac{T_1 ckpt}{\text{INTERVAL}} < \frac{p_o - p_s}{p_s} \quad (2)$$

Where $T_1 ckpt$ = CP-LAT $\times$ CP-OHD$_R$ is the delay due to a single checkpoint. Note that the makespan does not appear in Equation 2 and that the variables p_o, p_s, and $T_1 ckpt$ are known beforehand and all depend on the cloud infrastructure being used. This equation implies that the decision of which instance to choose depends on the relationship between the checkpointing overhead and the price difference. This also means we can predict when a misprediction may cause the algorithm to make the wrong choice. For the hypothetical example above, the Spot instance should always be chosen if its price is lower than 95.238% of its On-Demand price p_o. Notice that for Scenario 2, if the Spot price was US$2.94 (i.e., less than 95% of the On-Demand price), the Spot would now be chosen as the estimated total cost would be US$188.16/60. For the experiments in Section VII, as $T_1 ckpt$ has a lower value than our hypothetical example, this percentage threshold tends to be even higher. In practice, it is unlikely that the spot price would be so close to that of the On-Demand one for most instances, or at least for any significant period of time. If it were, by choosing the On-Demand instance, the absolute value of the overspend, if the selection were incorrect, is bounded and small, as it does not increase with longer execution times. In the case of the hypothetical example, independently of the actual execution time, the most one could overpay by is only US$0.07377.

The above example aimed to illustrate the robustness of SIM@CLOUD to imprecise makespan predictions, in the case of identical On-demand and Spot instances in the same region. Depending on the relative prices of the instances, there is a small window where the execution time can influence the instance selection. This analysis can be extended analogously to the more general case of Algorithm 1, where the makespan considers not only the estimated simulation time and checkpointing costs but also the data transfer overhead, all of which vary according to the instance type and region being used.

IX. CONCLUSION AND FINAL REMARKS

SIM@CLOUD is a framework tailored for executing simulations in the cloud while aiming to reduce the monetary costs associated with leasing instances. While solutions for executing reservoir simulations in the cloud exist (*e.g.*, those offered by CMGL), the VM selection approach proposed here has the advantage of exploiting the price differences of Spot and On-Demand VMs across multiple EC2 regions to reduce a company's cloud expenditure. The strategies deployed in SIM@CLOUD prioritize monetary cost mitigation while considering the potential time implications of VMs accessing data from non-local regions. In an effort to enhance its decisions,

SIM@CLOUD employs techniques to predict the makespan of the simulation. Using a real-world SLURM job submission log to build a reservoir simulation runtime predictor, the ML-PREDICTOR incorporates relevant attributes that distinguish and classify jobs. Given the dynamic nature of the simulations and the computing environment, the DYNAMIC-PREDICTOR iteratively estimates the remaining time of a simulation already in progress. The results obtained in a real cloud environment demonstrate SIM@CLOUD's practicality. While the work in this paper has focused on AWS EC2 (for commercial reasons), the SIM@CLOUD framework is cloud provider agnostic and is being extended to support other cloud providers and multi-cloud environments.

With regard to performance modeling, work continues to study different scheduling trade-offs for reservoir simulations in terms of resource utilization, energy consumption, performance, and monetary costs when using various instance families. Additional studies on the correlation between Spot instance availability and the probability of its revocation, the performance variability of cloud instances, and the impact of I/O operations on performance are being investigated.

ACKNOWLEDGMENTS

This research was supported financially by Petróleo Brasileiro S.A. – Petrobras, by the Brazilian Federal Funding Agency CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) through grants CNPq/AWS number 421828/2022-6 and Project Universal/CNPq number 404087/2021-3, and by the Rio de Janeiro State Funding Agency (FAPERJ) through grant CNE/FAPERJ number E-26/201.012/2022(271103).

REFERENCES

[1] F. Zhu, Y. Yao, W. Tang *et al.*, "A high performance framework for modeling and simulation of large-scale complex systems," *Future Generation Computer Systems*, vol. 51, pp. 132–141, 2015, Special Section: A Note on New Trends in Data-Aware Scheduling and Resource Provisioning in Modern HPC Systems.

[2] E. Winsberg, *Science in the Age of Computer Simulation*. University of Chicago Press, 2019.

[3] ——, "Computer Simulations in Science," in *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, 2022.

[4] S. J. E. Taylor, A. Anagnostou, N. T. Abubakar *et al.*, "Innovations in Simulation: Experiences With Cloud-Based Simulation Experimentation," in *Winter Simulation Conference*, 2020, pp. 3164–3175.

[5] M. A. S. Netto, R. N. Calheiros, E. R. Rodrigues *et al.*, "HPC Cloud for Scientific and Business Applications: Taxonomy, Vision, and Research Challenges," *ACM Computing Surveys*, vol. 51, no. 1, 2018.

[6] D. Ma and J. Huang, "The pricing model of cloud computing services," in *Proceedings of the 14th Annual International Conference on Electronic Commerce*. Association for Computing Machinery, 2012, pp. 263–269.

[7] N. Dimitri, "Pricing cloud IaaS computing services," *Journal of Cloud Computing*, vol. 9, no. 1, p. 14, 2020.

[8] I. Menache, O. Shamir, and N. Jain, "On-demand, Spot, or Both: Dynamic Resource Allocation for Executing Batch Jobs in the Cloud," in *11th International Conference on Autonomic Computing*. USENIX Association, 2014, pp. 177–187.

[9] L. Teylo, L. Arantes, P. Sens, and L. M. d. A. Drummond, "A Bag-of-Tasks Scheduler Tolerant to Temporal Failures in Clouds," in *2019 31st International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2019, pp. 144–151.

[10] M. E. Eldred, A. Orangi, A. A. Al-Emadi *et al.*, "Reservoir Simulations in a High Performance Cloud Computing Environment," in *SPE Intelligent Energy International Conference and Exhibition*, 2014.

[11] D. Salomoni, I. Campos, L. Gaido *et al.*, "INDIGO-DataCloud: A platform to facilitate seamless access to e-infrastructures," *Journal of Grid Computing*, vol. 16, pp. 381–408, 2018.

[12] M. Eldred, A. Good, and C. Adams, "Simulation as a service - a case study of provisioning scientific simulation software via a cloud service," in *2nd Int. Workshop on Emerging Soft. as a Service and Analytics*, 2015.

[13] D. Temelkovski, T. Kiss, and G. Terstyanszky, "Molecular docking with Raccoon2 on clouds: extending desktop applications with cloud computing," in *CEUR Workshop Proceedings*, 2017.

[14] T. Kiss, "Scalable multi-cloud platform to support industry and scientific applications," in *41st Int. Convention on Information and Communication Technology, Electronics and Microelectronics*, 2018, pp. 150–154.

[15] Z. Noor, Q. Wang, N. Govindaraju *et al.*, "Transitioning a Legacy Reservoir Simulator to Cloud Native Services," in *Int. Petroleum Tech. Conf.*, 2020.

[16] M. S. Flister and K. Hopstaken, "Running Reservoir Simulations in the public cloud; A case study of a cost-controlled method, running tNavigator and Eclipse in an Azure HPC environment," in *EAGE/AAPG Digital Subsurface for Asia Pacific Conference*, 2020.

[17] F. A. Portella and F. M. de Souza, "Reservoir Simulation in the Cloud," in *High Performance Computing in Clouds: Moving HPC Applications to a Scalable and Cost-Effective Environment*. Springer, 2023, pp. 265–282.

[18] Computer Modelling Group, "CMG Cloud: Your Guide," Online, 2021, Available at https://www.cmgl.ca/solutions/cloud/. Last visited on March 7th, 2024.

[19] F. A. Portella, P. Estrela, R. Malini, L. Teylo, J. Berral, and L. M. de A. Drummond, "MScheduler: Leveraging Spot Instances for High-Performance Reservoir Simulation in the Cloud," in *14th IEEE International Conference on Cloud Computing Technology and Science*. Napoli, Italy: IEEE, 2023.

[20] AWS, "Spot instance interruption notices," Mar 2024. [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/spot-instance-termination-notices

[21] NetApp, "Netapp global file cache," Mar 2024. [Online]. Available: https://www.netapp.com/data-management/global-file-cache/

[22] A. H. R. Pereira, "Instrução Normativa nº 5, de 30 de agosto de 2021," Aug 2021. [Online]. Available: https://www.in.gov.br/en/web/dou/-/instrucao-normativa-n-5-de-30-de-agosto-de-2021-341649684

[23] M. Hall, E. Frank, G. Holmes *et al.*, "The WEKA Data Mining Software: An Update," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 10–18, 2009.

[24] A. B. Yoo, M. A. Jette, and M. Grondona, "SLURM: Simple Linux Utility for Resource Management," in *Job Scheduling Strategies for Parallel Processing*. Springer, 2003, pp. 44–60.

[25] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning*. MIT Press, 2018.

[26] A. L. Nunes, F. Portela, P. Estrela *et al.*, "Prediction of Reservoir Simulation Jobs Times Using a Real-World SLURM Log," in *XXIV Symp. on High Performance Computing Syst.* SBC, 2023, pp. 49–60.

[27] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, 1993.

[28] M. Kuchnik, J. W. Park, C. Cranor, E. Moore, N. DeBardeleben, and G. Amvrosiadis, "This is why ML-driven cluster scheduling remains widely impractical," Carnegie Mellon University, Tech. Rep., 05 2019.

[29] L. Gasparini, J. R. Rodrigues *et al.*, "Hybrid parallel iterative sparse linear solver framework for reservoir geomechanical and flow simulation," *Journal of Computational Science*, vol. 51, p. 101330, 2021.

[30] AWS, "Best practices for handling ec2 spot instance interruptions," Feb 2024. [Online]. Available: https://aws.amazon.com/fr/blogs/compute/best-practices-for-handling-ec2-spot-instance-interruptions/

[31] L. Teylo, L. Arantes, P. Sens, and L. M. de A. Drummond, "Scheduling Bag-of-Tasks in Clouds Using Spot and Burstable Virtual Machines," *IEEE Transactions on Cloud Computing*, 2023.

[32] L. Teylo, A. L. Nunes, A. C. Melo, C. Boeres, L. M. d. A. Drummond, and N. F. Martins, "Comparing SARS-CoV-2 Sequences using a Commercial Cloud with a Spot Instance Based Dynamic Scheduler," in *The 21st IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGrid 2021)*, 2021, pp. 247–256.

[33] R. C. Brum, W. P. Sousa, A. C. M. A. Melo, C. Bentes, M. C. S. de Castro, and L. M. d. A. Drummond, "A Fault Tolerant and Deadline Constrained Sequence Alignment Application on Cloud-Based Spot GPU Instances," in *The 27th International European Conference on Parallel and Distributed Computing (Euro-Par 2021)*. Springer, 2021, pp. 317–333.