WCSim: A Cloud Computing Simulator with Support for Bag of Tasks Workflows

Maicon Ança dos Santos
PPGC/UFPel
Pelotas, Brazil
madsantos@inf.ufpel.edu.br

Gabriel J. A. Grabher
LIG/Université Grenoble-Alpes
Grenoble, France
gabriel.job-antunes-grabher@univ-grenoble-alpes.fr

Matheus F. Kovaleski
II/UFRGS
Porto Alegre, Brazil
mfkovaleski@inf.ufrgs.br

Cláudio F. R. Geyer
II/UFRGS
Porto Alegre, Brazil
geyer@inf.ufrgs.br

Gerson Geraldo H. Cavalheiro
PPGC/UFPel
Pelotas, Brazil
gerson.cavalheiro@inf.ufpel.edu.br

***Abstract*—In this paper, we present WCSim, Workflow Cloud Simulator. Firstly, we argue that this cloud simulation tool offers a high level of accessibility by allowing the description of various components, such as users, infrastructures, and workload, of a given scenario simply by providing parameters at launch time, without requiring the extension of the simulator code. Then, we explain how we conceived the components for the simulation models and provide a detailed description of the implemented software. Additionally, we compare the results of a small scenario obtained from two other simulation tools with those provided by WCSim. Finally, we present a case study that illustrates the usage of WCSim. The paper also introduces the a abstraction to model workflows as a Direct Acyclic Graph of Bag of Tasks.**

***Index Terms*—Cloud Simulators, Simulation, Cloud Computing, Programming Models, Performance Evaluation**

I. Introduction

Simulators are computer programs developed to programmatically replicate the behavior of a real system. In various fields of knowledge, simulators present themselves as an attractive alternative for system evaluation due to the level of accuracy they can offer, which depends on the level of detail in the simulation model used and the abstractions of reality considered. In the field of Computer Science, simulators are widely popular, particularly for the study, design, and sizing of computer network environments and distributed systems. In this context, one of the oldest and perhaps the most well-known simulator is NS [1], used for simulating protocols in computer networks. In recent years, several simulators for grid and cloud computing have also been proposed. Several surveys in the literature, such as [2]–[5], evaluate, compare, and discuss the limitations of such tools. Indeed, the identification of challenges in cloud computing simulators represents a significant motivation for their construction, as highlighted in [6]. This aspect represents one of the key obstacles to overcome in the field of modeling and simulation. Among these various tools, two stand out for their popularity, as expressed by the size of the community involved and the number of published works, the SimGrid [7], and the CloudSim [8], [9]. These two simulators are widely used to evaluate different scenarios in grid and cloud computing.

In this paper, we present WCSim, the Workflow Cloud Simulator, as an another grid/cloud simulator. WCSim was conceived with the same goal as SimGrid and CloudSim, which is simulating the execution of diverse workloads on various infrastructure configurations, considering different scheduling policies. Presented in this way, the space to be occupied by WCSim may seem the same as that of these two tools, but its implementation takes into account certain premises, positioning itself with its own distinctive approach. Those main premises are: provide an accessible simulation tool, and support naively support both workflow and bag-of-tasks (BoT) programming models. We believe we have fulfilled these prerequisites by developing a simulation tool that enables the description of study cases as parameters to the simulator, instead of requiring programming as in other tools. Additionally, our tool provides a model for application description in the form of a workflow of BoTs. The current version of WCSim was built upon the simulator presented in [10], where the concepts of separating the description of the simulated scenario (infrastructure and workload) from the simulation engine and the handling of BoT applications were adopted. WCSim has provided valuable data to support the thesis presented in [11].

The main contribution of this paper is introducing WCSim. WCSim is designed and implemented with an object-oriented style using C++, without delving into more complex aspects of the language in order to enhance the accessibility of the tool to a wider audience. The proof of concept is provided by two case studies. In the first, we compare the results obtained from WCSim to those provided by SimGrid and CloudSim. In the second, we evaluate different scenarios in WCSim. Furthermore, we offer comprehensive insights into the initial assumptions made during the development of the simulation tool, considering the knowledge documented in the literature on the prerequisites for designing and implementing an alternative solution [12]–[14] by other authors. Finally, we would like to highlight the intriguing concept of representing applications as a workflow of Bag of Tasks, which is a novel idea introduced in this article. Nonetheless, we believe that this concept warrants further exploration as a separate line of investigation.

The remaining part of this text is organized as follows. Section II provides an overview of simulation tools in the literature, with a focus on SimGrid and CloudSim. Section III presents the main elements considered in the design of the simulation model. Section IV describes the software structure, and Section V provides details on how case studies are submitted to the simulator and the generated outputs. Section VI presents the two experiments conducted as proof of concepts. Finally, Section VII concludes the paper.

II. Simulators in the literature

Various simulators for cloud computing have been documented in the literature [2], [15]. These simulators serve as vital tools for developing, configuring, and evaluating different cloud architecture configurations and application behaviors. However, studies show that none of them are complete or ideal for all analyses. Among these simulators, this work considers DISSECT-CF-WMS, SimGrid, and CloudSim.

DISSECT-CF[1] [16] IaaS simulation framework was implemented to offer easy extensibility for introducing models, supporting energy evaluation of IaaS, and enabling rapid assessment of various scheduling strategies. An extension to this simulator, DISSECT-CF-WMS [17], where the authors evaluate scenarios involving workflows under virtualized infrastructures.

SimGrid[2] is a C-based tool for simulating distributed environments available by a GNU-Linux package. Its interface provides extensions to C++, Java, and Python. The scheduling core is based on discrete events and [18] arguments that this feature affects scalability as the simulation model complexity increases. SimGrid offers the *actors model* to enable users to define simulation behavior for tasks, communications, and disk usage. The simulation core predicts execution time and orchestrates actor activation. Although originally presented as a generic simulation environment for distributed systems focusing on evaluating parallel applications, particularly those developed in MPI (*e.g.* [19], [20], [21], [22]), and even in OpenMP [23], SimGrid is currently being used in a much wider range of cloud applications, such as IaaS clouds, volunteer computing, and fog computing. A functionality added to SimGrid through an extension [24] is the ability to specify programatically a directed acyclic graph (DAG) of tasks.

CloudSim was originally developed in Java, and in this work, we consider the fork of the project called CloudSim Plus[3] which brings several performance improvements compared to the original version [9]. CloudSim itself was introduced as a derivative project of GridSim [25]. The installation of CloudSim Plus is available through a software repository and requires manual installation, utilizing the provided source code packages. The core simulation of CloudSim Plus extends the functionalities of the SimJava package [26] and also employs a discrete simulation model. Being a newer project compared to SimGrid, CloudSim focuses more on cloud concepts, likely due to their wider acceptance in the community in last years. Virtual machine and cloudlet concepts are explored. The simulator is widely used for evaluating scheduling strategies for clouds, including workload scheduling between virtual machines and virtual machine placement on host machines [27], [28], [29]. CloudSim has also been applied in other computing models derived from Cloud Computing, such as Fog Computing [30] and Edge Computing [31] and [32]. While CloudSim does not provide a native abstraction for constructing DAGs, achieving it programmatically is possible. WorkflowSim [33], an extension of this tool, follows the Pegasus model for describing workflows, offering this level of abstraction.

It is worth noting that both the SimGrid and CloudSim projects have a large community and several derived projects. According to [2], CloudSim is likely the most popular. It is important to highlight that while both projects can simulate cloud computing models, SimGrid primarily serves as a simulator for grid computing, focusing on fine-grained task applications [34]. In contrast, CloudSim is specifically designed to simulate virtual machine abstraction and virtual machine scheduling, which extends the capabilities of SimGrid [35].

Both SimGrid and CloudSim allow the construction of task workflows. SimGrid provides this capability through an extension called SimDAG [24], examples in [36], [37] and [38]. In CloudSim, workflows are created by adding actions to the completion event of a cloudlet, generating subsequent cloudlets or even expliting the WorkflowSim extension. These tools are also used for evaluating scheduling strategies considering cloud usage cost, examples in [39], [40] and [41]. Another aspect observed in these tools, relevant in this work, is that neither of them provides a native abstraction for a user entity with cloud access. This entity is responsible for job submissions and has ownership rights over virtual machines, as introduced in the basic assumptions of the simulator.

SimGrid and CloudSim Plus are simulation tools that offer various features for modeling case studies in cloud computing and should remain as references in many usage contexts. These tools have been selected for our case studies based on their popularity and the active involvement of community members in their development. However, we understand that the advancement of cloud computing environments requires greater dynamism in the evaluation processes of application behavior over infrastructure projects. We still consider that the study of application behavior described as a workflow of bag of tasks[4] is not adequately addressed in these tools.

III. Initial Assumptions

In cloud computing, a *physical infrastructure* supports the execution of virtual machines provided by a *virtualization layer*. *Users* own virtual machines and launch their *applications* in the form of a workflow of bag of tasks. A two-level *scheduling* mechanism distributes the workload of the applications across

[1]https://github.com/kecskemeti/dissect-cf, Ago 14, 2023.

[2]https://simgrid.org, May 22, 2023.

[3]https://cloudsimplus.org, May 22, 2023.

[4]The concept of *workflow of bag of tasks* is introduced in Section III-D.

the user's virtual machines and the processing load of the virtual machines across the physical infrastructure, potentially applying a load distribution policy. Finally, the *utilization and accounting* of resources must be effectively monitored and managed to provide information that can be considered by a scheduling policy and ensure accurate billing.

All components of a cloud computing environment, in our simulation model, are described by a set of attributes, except for scheduling, which must be described by an algorithm. The attributes are called *static* when they describe the immutable physical characteristics of the component, or *dynamic* when they record a value representing a specific state during the simulation process. The number of cores in a processing server is an example of a static attribute, while the current number of virtual machines hosted on a processing server exemplifies a dynamic attribute. In our model, time evolves discretely, second by second. This time unit is convenient in our model since the processing requirements (the length) of the tasks are given in millions of instructions, the processing power of the cores is given in MIPS, and the network bandwidth is given in Mbps.

A. The Physical Infrastructure

The physical infrastructure is composed by *datacenters* (DC), each DC composed by a set of *processing servers*. A two-layered network interconnects all of these elements, with one layer connecting servers within a DC and another layer connecting multiple DCs.

1) The processing server: A processing server (PS) represents bare metal that provides processing resources such as cores (CPUs), RAM, and storage, described by static attributes. The processing power of a PS is given in terms of the number of cores and MIPS (Million Instructions Per Second) per core, while RAM and storage are measured in gigabytes (GB). Each PS has a unique identifier and bootstrapping and shutting down data. Optionally, a PS can host one or more GPUs. Also, as a static attribute, each PS is instantiated with a cost per MIPS, which corresponds to the cost of executing a million instructions. The dynamic attributes of a given PS describes its status during the simulation process, such as: `isOnline`, `nVM`, `nPinnedCores`, and `fRAM` identify whether the PS is active, the number of virtual machines hosted, the number of cores requested by the hosted virtual machines, and the amount of memory free for allocation. For the sake of utilization billing, a set of cost attributes is associated with a PS: the cost of executing a million instructions, the cost of a GB of RAM or storage, and the cost of a GPU.

Derived from the static and dynamic attributes, it is possible to obtain additional information about the PS. The utilization rate, which represents the degree of performance degradation of the PS is an example. The degradation of a PS is given in terms of the decrease in the delivery of the nominal amount of MIPS from its cores. This degradation occurs when the demand for virtual machines exceeds the capacity of the cores offered by the bare metal.

As dynamic attributes, a PS also has an *invoice*. This attribute identifies how many resources, such as the number of executed MIPS, RAM, storage, and GPU it provided to each hosted VM. Additionally, the invoice attribute records the actual utilization of the PS. The utilization is represented in terms of server occupancy. There are 8 ranges considered, and the portion of time, from the total simulation time, in which the PS remained with processing load in the respective range, is accumulated. The time recorded in the first range indicates the portion of time that the server remained inactive. The time in the last range indicates the portion of time that the server had an occupancy higher than 200% of its nominal capacity. The remaining ranges correspond to the occupancy intervals: $(0, < 25\%], (25\%, 50\%], (50\%, 75\%] \dots (175\%, 200\%]$.

2) The datacenter: A datacenter (DC) is composed by a set of (heterogeneous or homogeneous) PSs. Each DC has a unique identifier, a name[5], and bootstrapping and shutting down data, as static attributes. A local network connects the PS belonging to a DC, and a wide area network connects the DCs. The model doesn't distinguish between federated DCs and public DCs, but a hybrid cloud configuration can be achieved by selecting the appropriate configurations (e.g., PS configurations and network speeds). In this case, the private part of this cloud is composed of a set of federated DCs, each belonging to different institutions. The public part is a DC hosted by a public provider, whose resources are provisioned following a cloud bursting strategy [42].

3) The network: Bidirectional links allow all PS to send/receive messages to/from all other PS. The current implementation of the communication model assumes no reduction of bandwidth due to contention and does not handle network faults. The bandwidth of each link is individually provided in Mbps as a static attribute. We model local and wide area networks by assigning different values to connections between PS belonging to the same or different DCs.

B. The Virtualization Layer

The virtualization layer provides virtual machines to the users. A virtual machine (VM) is owned by a user and has the same static attributes as a PS, including a unique identifier, bootstrapping and shutting down data, number of virtual cores, and size of virtual RAM. Each VM also has its owner's unique identification. In terms of dynamic attributes, a VM also keeps information about the number of running tasks, the number of virtual cores in use, and allocated virtual memory. However, unlike PSs that can be online or offline, a VM can also be in a suspended or migrating state. Another dynamic attribute is billing, which records the amount of MIPS executed for each PS for billing purposes.

C. The User

The user is an entity that submits computational demands (or applications) to the cloud. Each user has, as static attributes, a unique identifier, a login date, and a home, representing the DC to which they have access in the cloud. The list of VMs belonging to a user is also a static attribute, as well as its list

[5]We assign a *name* to the DCs to provide a user-friendly identification.

of jobs (*cf*. Section III-D). The group and priority attributes are also static attributes for a user and can be used by scheduling policies to make placement/mapping decisions. The attribute group represents the different levels of user access to cloud resources: group 0 indicates that the user's VMs can only run in their home DC, group 1 indicates that the VMs can migrate between other DCs in the infrastructure, and group 2 indicates that the user's VMs can be selected for cloud bursting. The priority indicates the execution priority of the user's VMs over the resources they have access to.

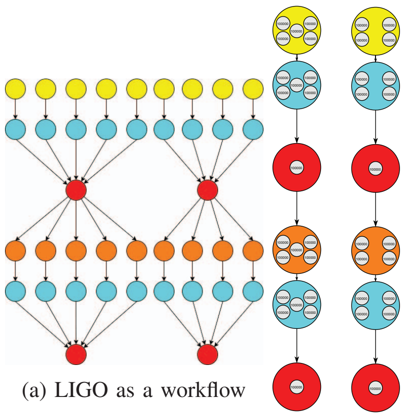
As dynamic attributes, a user has a *wallet*. The wallet informs the credits (financial resources) the user has to consume launching its application over the cloud.

D. The Applications

The literature presents BoT (bag of tasks) [43] and workflow models [44] as the most common in application submissions for grid and cloud computing. The fundamental difference between these models lies in the dependency relationships between tasks. BoT applications are characterized by a set of independent tasks. According to [43], it is somewhat difficult to identify a BoT in a grid trace. However, if a group of tasks is submitted by the same user within a small time interval, it can be assumed that they belong to a BoT [43]. Workflow applications involve defining tasks and the dependencies of data among them, typically represented by a Directed Acyclic Graph (DAG). Upon closer examination of workflow applications, we observe that the workflow tasks may represent Bags of Tasks. In fact, [45], [46] identify that some workflows can describe coarse grain tasks describing a set of independent finer tasks. In such cases, the application models a workflow of Bags of Tasks. This application style represents our application model, and we refer to it as DoB, which stands for DAG of Bags of Tasks. One advantage of DoB, as model of application, is to mitigate the drawback of using a BoTs is that users need to handle data dependencies between tasks before introducing a new task to the bag [47]. Another one is to allow to describe communication dependent tasks, such as in MPI programs.

In the context of the Pegasus project [48], applications for grid and cloud computing are represented graphically as workflows where tasks are represented as circles and dependence among tasks as directed arrows. Figure 1a reproduces the original representation [48] for LIGO workflow. The different colors represent the different operations achieved by the tasks. In Figure 1b the same application is presented as a DoB. Large circles represent workflow tasks, while the small ones still representing the same operations achieved by tasks in the original representation. The DoB abstraction is represented by two elements in the simulation model: job and task. A *job* is a container of a computational demand (the larger circles). A *task* describes a unit of work.

Both jobs and tasks are entities in the simulation model that represent computational demands. A job is a container of tasks and has a unique identifier, as well as the identification of its owner. A job may or may not have dependencies. Also as static attributes, a job has a list of jobs which depends on it, together



(a) LIGO as a workflow

(b) LIGO as a DoB

Fig. 1: Contrast between the representation of the original workflow of the LIGO application in the Pegasus project and the proposed DoB form.

with the number and a list of tasks it holds. If a job doesn't have any dependencies, it has a static attribute that informs its launch time. As dynamic attributes, each job has a status that informs its execution state (waiting, finished, executing, etc.) and the number of dependencies to be satisfied before it can be executed. The static attributes of tasks describe the computation itself. The static attributes of each task, in addition to its unique identifier, are its length (in MIPS), memory and storage requirements, and the identifier of the job to which it belongs. The dynamic attributes record the status of tasks during their life cycle (waiting, finished, executing, etc.), the identifier of the VM on which the task is being executed, and the number of instructions (MIPS) already completed.

E. The Scheduler

The scheduler is modeled in four layers. First, at the local level of a PS, we observe the sharing of resources (e.g., cores) provided by a PS among the VMs it hosts. Second, at the internal level of a DC, the VMs are managed among the PSs belonging to a DC. Third, at the global level of the federated cloud, the load produced by VMs can be migrated between PSs belonging to different DCs. Fourth, for cloud bursting to a public provider, the load produced by VMs can be bursted to a public provider. The basic scheduling operations involve mapping processing requirements over computing resources and sharing computing resources among processing requirements. Additionally, the scheduling may support a load distribution policy to optimize some performance index. It is also a scheduling decision to launch or cancel the execution of a task for a user whose credits have run out.

The scheduling in the simulation process is reactive to the evolution of applications during execution and to activities observed in the infrastructure. In these cases, the simulator maps tasks onto virtual machines or maps virtual machines onto processing servers. Scheduling is activated whenever: (*i*) a

```
DATE & EVENT & ID & RELATED & NOTE
...
1000 & 10  & 32   & 0   & PS bootstrapping
1000 & 10  & 33   & 1   & PS bootstrapping
...
1001 & 20  & 4    & 0   & User login
1001 & 20  & 5    & 1   & User login
...
1001 & 30  & 12   & 4   & DoB launching
1001 & 30  & 13   & 5   & DoB launching
1001 & 330 & 145  & 12  & Job ready
1001 & 330 & 146  & 12  & Job ready
1001 & 3330 & 809 & 145 & Task ready
1001 & 3330 & 810 & 145 & Task ready
1001 & 3331 & 809 & 145 & Task launch
1001 & 3331 & 810 & 145 & Task launch
...
1011 & 3332 & 809 & 145 & Task finish
1011 & 3332 & 810 & 145 & Task finish
1011 & 331  & 145 & 12  & Job finish
...
```

Fig. 2: Fragment of log file.

job becomes ready to execute, (*ii*) a task needs to be launched, (*iii*) a task finishes, (*iv*) a new VM is spawned, (*v*) a new PS is integrated into the cloud, or (*vi*) a PS becomes inactive. Each time any of these situations occurs, the scheduler is triggered, receiving the corresponding information as a parameter. For example, it receives a job in situation (*i*), a task in situations (*ii*) and (*iii*), and so on. Each scheduling intervention generates a log of information. This log provides the output of the simulation, enabling analysis of utilization and accounting.

F. Utilization and Accounting

Our user model includes the attribute *wallet* which tracks the credits that users have for consuming cloud resources. Credits are deducted when using resources at the PS level, such as executing a million instructions and provisioning RAM, storage, or GPU to a VM. As a user's tasks are performed, their credits are progressively consumed. In our model, costs are associated with the bare metal infrastructure rather than the VMs. This is because we have observed limitations in resource elasticity within a federated cloud, and we cannot guarantee the consistent performance of a VM family.

The utilization is recorded at each intervention of the scheduler in a log (CSV type) file. Each log entry contains five or six fields, depending on the kind of the event. Ordinary events have five fields: the timestamp of the event; a class code that identifies the event that generated the log; two identifiers: one for the simulation component that triggered the log and another for the component to which that component is related; and a textual description of the event. Figure 2 shows a fragment of a log file. In this fragment, we identify the bootstrapping of two PS, PS:32 on DC:0 and PS:33 on DC:1, and the logon of two users, User:4 and 5, on DC:0 and 1, respectively. At time 1001, each user launches a DoB: User:4 launches DoB:12, and User:5 launches DoB:13. Additionally, two jobs and two tasks become ready at time 1001: Job:145 and 146, both belonging to DoB:12, and Task:809 and 810, both belonging to Job:145.

The combination of resource utilization information, recorded by the invoice attribute in the PSs, with the accounting of credits used by users, recorded in the "wallet" attributes, enables the analysis of effective cloud utilization, evaluating its performance in meeting the presented demands.

IV. General Simulator Structure

The scheduling mechanisms are presented in the form of methods belonging to an abstract class called *Scheduling*. Each method corresponds to each of the situations that trigger a scheduling operation (cf. Section III-E). A default behavior for scheduling is provided.

A. The Software Structure

Figure 3 presents the simplified class diagram of the developed software structure. We highlight the *Component* class, from which all event-driven simulation model elements are specialized. This class provide the abstract method *execute*, to promote the execution of an event. The *Component* constructor is responsible for inserting a new event in the event list and the destructor for generating log information. The diagram also provides examples of events supported by each component.

The cloud (class *Cloud*) is not a component because it does not generate events by itself. It consists of a utility class that holds the selected scheduling policy.

B. The Simulation Kernel

The simulation kernel handles a global clock and an event queue. An event describes a simulation activity, representing something that happens in the cloud and is associated with a specific timestap. Events in the queue are sorted by timestamp. If multiple events occur simultaneously, there is a defined priority order among components: PS (higher priority), User, VM, Job, and Task (lower priority). During its execution, an event may schedule new events. Initially, the event queue is populated with the events programmed in the input files provided to the simulator. The scheduler kernel runs the Algorithm 1.

As shown in the Algorithm 1, the time evolves discretelly adding `INC` time units to the global clock. The default `INC` value is 1 (meaning one second). Increasing this value can speed up the simulation, but it may lead to a loss of accuracy in the results. At the end of each step, the scheduling operations are

Algorithm 1 Simulator kernel

```
INC ← 1; Reads input files;
Creates the cloud infrastructure;
Populates eventQueue;
GlobalClock ← 0;
while eventQueue.size() > 0 do
    while eventQueue.first().date() <= GlobalClock do
        event ← eventQueue.first();
        eventQueue.removeFirst();
        event.execute();
    end
    Cloud::localSchedule();
    Cloud::datacenterSchedule();
    Cloud::cloudSchedule();
    Cloud::burstingSchedule();
    GlobalClock ← GlobalClock + INC;
end
Generates outputs;
```

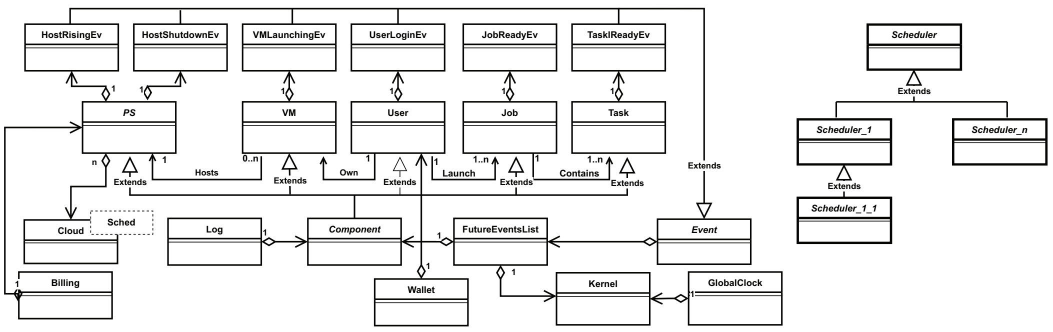


Fig. 3: Simplified UML class diagram for WCSim.

activated to advance the execution of tasks and, if taken into account, promote load balancing.

C. Class Scheduler

The virtual class *Scheduler* offers those virtual methods:

- `load_t getVMLoad(VM* vm)`: returns the load of a `VM`.
- `load_t getPSLoad(PS* ps)`: returns the load of a `PS`.
- `void placeTask(Task* t)`: task `t` belongs to a `Job` that was launched by a `User` which own a set of `VM`s; `t` is placed on one of the `VM`s owned by the `User`.
- `void placeVM(VM* vm, DC* dc)`: places a `VM` in a `PS` belonging to the datacenter `dc`.
- `VM* selectVM2Migrate(PS* ps)`: returns a `VM` selected in `ps` to migrate.
- `void localSchedule(PS* ps)`: visits all `PS`s, simulating the execution of `INC` × millions of instructions.
- `void datacenterSchedule(DC* dc)`: promotes the load balancing between `PS`s belogin to `dc`.
- `void cloudSchedule(DC* dc)`: activates load balancing between `DC`s from `dc`.
- `void burstingSchedule(DC* dc)`: enables `dc` to achieve cloud bursting.

The scheduler methods are invoked in a callback manner: when a specific scheduling situation is required, the corresponding method is called. A default behavior is provided for all methods, however, the default for `cloudSchedule`, `datacenterSchedule`, and `burstingSchedule` is to do nothing. New scheduling strategies can be elaborated by extending the `Schedule` class. For a given simulation, the chosen strategy must be provided as a *template* to the `Cloud` class.

D. The components

A *component* is a representation in the simulation model of a dynamic entity from the real model that evolves in the domain of time [49]. During a simulation, each component evolves its lifecycle, performing activities related to its purpose. Components are reactive to events that occur during the simulation process and, in turn, they can themselves cause new events, which may trigger a reaction within themselves or in other components.

E. The events

An *event* is an object that contains data to be triggered and identifies the component that should undergo the action during event processing. Events can be classified as exogenous or endogenous. An exogenous event originates from an external action in the simulation context. In our simulation model, these events are provided as input when the simulator is launched. Handling such events may lead to modifications in simulation elements, potentially interfering with programmed actions. Examples of exogenous events include booting a new PS and submitting a new job without predecessors. An endogenous event is generated during the simulation process as a result of processing another event. Examples of endogenous events include suspending or resuming the execution of a VM after migration and changing the job status to *ready to execute* once all dependencies are satisfied.

V. Input and Output Files

WCSim requires three files as inputs and provides four files as outputs. The input files consist of a descriptions of workflows to be submitted to the cloud, user identification, and a description of the cloud infrastructure. These inputs are provided as files with the extensions `.dob` (Figure 4a), `.pas` (Figure 4b), and `.inf` (Figure 4c), respectively. In those files, any content following the # character on a line is ignored.

The description of the workflows submitted to the cloud is presented in terms of dependence between jobs, as in Figure 4a. Each line represents the description of a job within the workflow. The first column provides the job ID, and the second column provides the ID of the user responsible for submitting it. The thirty column inform how many other jobs must be completed before be ready to execute (the number of predecessors). The fourth column informs the timestamp (in time units) the job is launched. Important to notice: if a job has predecessors, the arrival date is unconsidered. The fifth column informs how many tasks are contained in the job and the next two columns identify the length (in millions of instructions) and the memory requirement (in GB) of each task.

In the users description file (Figure 4b), we can observe, in the first three columns, that each user has a *name*, a datacenter as its *home*, and the identification of the group to which they belong. Following that, each entry specifies how many VMs will be launched when the user logs on, along with the *VM family*, identifying the type for those instances. The line that describes each user is taken as its ID.

An example of infrastructure is given in Figure 4c. In this file, each line represents a PC belonging to a specific DC: there are three DCs, each with four PCs. Also the number of lines are taken as PS IDs. The second column indicates the booting timestamp for each PC, while the third column specifies the PC's bare metal architecture. In this case, all PCs are active at simulation time 0. UFPel and IFSul have a bare metal family of 0, while UFRGS has a family of 1. The subsequent columns provide information about the communication link speeds between the PCs (Gbps). As the simulation model assumes bidirectional channels with equal bandwidth in both directions, the speeds are given in one direction only. The first speed value corresponds to the link between the described server and the next server in the list. The second speed corresponds to the link between the described PC and the second PC in the list, and so on. The last server in the list does not have an explicitly stated communication speed as its links to the other servers have already been mentioned in previous lines. In the provided example, PCs within the same DC communicate at 10 Gbps, and between different DCs at 1 Gbps.

In addition to the log file (Figure 2), the simulator produces the files `performance`, `trace`, `wallet`. The last column of each line in performance file (Figure 5) indicates the timestamp of the last task generated by each user. The users are identified by their names and IDs. The trace file presents the accumulated time (in time units) that each PS remained at a specific workload level. There are a total of nine workload levels, including the PS's inactive state. The workload levels are presented in intervals of 25% utilization. Finally, the billing file indicates the number of millions of instructions each user received from their respective PS. User identification is in the first two columns, and the corresponding PS is in the second-to-last column.

VI. Case Studies

The proposed simulator is fully operational and publicly available in a repository for community collaboration. The experiments conducted for this article are also included in the repository for reproducibility and as usage examples. These case studies serve to demonstrate the availability and potential of the tool. First we compare the results presented by WCSim, SimGrid, and CloudSim Plus for a very simple cenario. Then, we exemplify an evaluation of scheduling strategies, presenting the results of an analysis on the economic impact of utilizing cloud bursting in a federated cloud environment.

A. Comparative experiment

This section presents a case study comparing WCSim's results to those of SimGrid and CloudSim Plus. These results relate to the simulation time of submitting a set of tasks to a cloud environment. As additional information, the simulation runtime is also provided. The objective of this case study is not to compare the performance of the simulators based on their response time, nor to identify the accuracy of these tools. The goal is purely to position WCSim comparatively with these two tools in qualitative terms by interpreting their model abstractions and simulation results for a simple scenario. The collected performance are shown in Table I.

The case study involves submitting three sets of workloads to an infrastructure consisting of four processing servers, each with four processors, interconnected by an unbounded network. The workloads comprise 100, 1,000, and 10,000 identical tasks, each requiring the execution of 1 billion instructions – 1,000 MI. All tasks are launched at simulation time 0 (zero) with all PSs already booted and no task migration occurs. Only the processing cost attributed to the instructions executed by the tasks is taken into consideration, while other expenses such as memory and storage are disregarded.

a) WCSim simulation model: At time 0, four PSs with four 1,000 MIPS cores each are started in separate DCs. Also, at this time, four users, one from each DC, log into the cloud and launch a VM configured with four vcores, each offering 100 or 1,000 vMIPS according to the case evaluated. Each VM is mapped to the PS belonging to the DC home of its owner. Then, each user triggers a job that represents 1/4 of the total workload considered in the experiment.

TABLE I: Comparing performances and behaviors.

Simulator	100 Tasks		1,000 Tasks		10,000 Tasks	
	Sim	Exec	Sim	Exec	Sim	Exec
	PS: 1,000 MIPS/core, VM: 100 vMIPS/vcore					
WCSim	61s	<0.01s	621s	0.41s	6250s	34.6s
SimGrid	62.5s	<0.02	625s	0.24s	6250s	3.3s
CloudSim Plus	65.1s	2.58s	650.1s	5.27s	6,500.1s	18.6s
	PS: 1,000 MIPS/core, VM: 1,000 vMIPS/vcore					
WCSim	7s	<0.01s	63s	<0.1s	625s	8.2s
SimGrid	6.3s	<0.05s	62.7s	<0.3s	626.9s	3.7s
CloudSim Plus	6.5s	2.5s	65.2s	5.1s	652.2s	15.1s

b) SimGrid simulation model: The environment platform is defined in an XML file, identifying the number of available servers, their computing speed (FLOPS), their number of cores, and the network bandwidth between them. As performance is given in FLOPS, we assume 5 instructions are executed per floating-point operation. At the start of the simulation, 5 servers are started with 4 cores, providing a processing speed of either 80 MFLOPS or 800 MFLOPS, depending on the experiment, to represent the proposed scenario where each core provides 100 or 1,000 MIPS. Simultaneously, 4 worker and 1 scheduler

```
#Job owner nDep arrival ntasks MI RAM [id_dep] # Comments
0  0 0 0    4 216000 50       # Blast ...
1  0 1 0    1 432000 50 0     # SRNA
2  0 1 0    1 432000 50 1     # FFN_parse
3  0 1 0    3 216000 50 1     # Blast_candidate ...
4  0 1 0    1 432000 50 2     # Blast_syntese
5  0 3 0    1 216000 50 4 3 8 # SRNA_annotation
6  0 1 0    1 432000 50 5     # Sendemail
7  0 0 0    19 108000 60      # Patser
8  0 1 0    1 432000 50 7     # Patser_con
9  1 0 3600 4 216000 50       # Blast ...
10 1 1 3600 1 432000 50 9     # SRNA
11 1 1 3600 1 432000 50 10    # FFN_parse
12 1 1 3600 3 216000 50 10    # Blast_candidate ...
13 1 1 3600 1 432000 50 11    # Blast_syntese
...
```

(a) Workload file: sipht.dob

```
#User DC Date Grp nVMs VMFam
user00 UFPel 0       0 4 0
user01 UFRGS 0       0 2 0
user02 IFSul 0       0 4 1
user03 UFPel 0       0 2 0
user04 UFRGS 0       0 1 0
user05 IFSul 2000    0 3 1
user06 UFPel 3000    0 2 0
user07 UFRGS 4000    0 4 0
user08 IFSul 1000000 0 4 1
```

(b) Login file: users.pas

```
#Datacenter Date PSFam CommCosts
UFPel 0 0 10 10 10  1  1  1  1  1  1  1  1
UFPel 0 0 10 10  1  1  1  1  1  1  1  1
UFPel 0 0 10  1  1  1  1  1  1  1  1
UFPel 0 0  1  1  1  1  1  1  1  1
UFRGS 0 1 10 10 10  1  1  1  1
UFRGS 0 1 10 10  1  1  1  1
UFRGS 0 1 10  1  1  1  1
UFRGS 0 1  1  1  1  1
IFSul 0 0 10 10 10
IFSul 0 0 10 10
IFSul 0 0 10
IFSul 0 0
```

(c) Infrastructure file: thecloud.inf

Fig. 4: WCSim input files: parametrizing the simulation.

```
#User,UserId,ExecutionTime
user00,0,2594
user01,1,2593
user02,2,2593
user03,3,2593
user04,4,0
...
```

(a) Performance file: performance.csv

```
#Host,HostId,Idle,25%,50%,75%,100%,125%,150%,175%,>200%
...
UFPelH2,2,16336,1000,237,364,5210,506,6442,2517,54150
UFPelH3,3,7171,2000,0,0,182,0,1202,0,76207
UFRGSH4,4,39396,0,294,0,852,2311,2054,7445,34410
UFRGSH5,5,36497,4855,3481,1314,1581,3556,4253,5228,25997
...
```

(b) Trace file: trace.csv

```
#User,UserId,Host,HostId,nbInst
user00,0,UFPel,0,3079728
user00,0,UFPel,1,5726270
user00,0,UFPel,2,5022350
user00,0,UFPel,3,3998172
user01,1,UFRGS,4,4863310
user01,1,UFRGS,5,4105350
user01,1,UFRGS,6,3350820
user01,1,UFRGS,7,5508702
user02,2,UFPel,0,4428528
user02,2,UFPel,1,3349488
...
```

(c) Billing file: bill.csv

Fig. 5: WCSim output files: simulation performance results.

processes are started on 5 different PS, where each process runs in one VM using all the resources of one of the servers – in this sense, the VM has the same processing power than the server. The scheduler then distributes 100, 1,000, and 10,000 tasks of 200 MFlops each in a circular manner between the workers. The workers stay in a loop of receiving and executing tasks until a finishing signal is sent by the scheduler, indicating that the experiment is over.

c) CloudSim Plus simulation model: All elements belonging to the simulation model are described as classes extending the framework API. Tasks are described by cloudlets of 1000 MI in size, which are executed by VMs. In this model, a VM is launched on each processing server, and the cloudlets (100, 1,000 or 10,000) are distributed in a circular manner, by a *broker*, performing scheduling operations, among the VMs. The processing servers run at 1,000 MIPS, while the VMs were configured to run at 100 and 1,000 MIPS.

d) Analysis of the results: All models were first simulated applying the default configurations of the tools than we promote changes in these configurations to evaluate the results. The best performance obtained are in Table I. In our case study, the optimal execution times for 100, 1000, and 10000 tasks are 62.5, 625, and 6250 seconds and 6.25, 62.5, and 625 seconds on 4 quadcore PSs at 100 and 1000 MIPS, respectively, in a network without bandwidth limitation. The closest results are those presented by SimGrid in this very simple scenario. It is important to note that during the calibration of the experiment, we set the constant `MIN_TIME_BETWEEN_EVENTS` (related to the sensitivity to events) to 1ns in CloudSim. With this parameter, CloudSim yielded the best results. Changes in the default parameters of SimGrid, `cpu-threshold` and `maxmin` in particular, didn't affect significantly the results or the execution time. We made changes in parameters. In WCSim, all scheduling operations are triggered by events during the simulation and the minimum time step is 1s. Therefore, as the number of tasks being handled increases, more events occur, leading to more precise results. Conversely, the granularity of a 1s time step proved to be inappropriate for considering small pieces of work. This is represented in the experiment by the distribution of "6.25" tasks among the cores – in practice, since tasks cannot be split, three cores of each PS handle 6 tasks while one core handles 7. The results presented by WCSim become more accurate as the number of tasks increases and the size of the time step becomes less important.

e) Simulators performance: Table I presents the execution time required by each simulator to process each scenario in the same hardware. These results are shown to illustrate the performance of the simulators themselves and do not represent an exhaustive performance assessment. However, we observe very similar performances, with the execution time increasing as the number of tasks grows and decreasing as the processing power of the virtual machine increases.

f) Overall analyses: All three simulators offer abstractions to model the proposed scenario. WCSim differs from the others by offering abstractions for users (who are responsible for generating work demands). SimGrid abstraction of VMs seems more limited when compared to the ones made available by WCSim and CloudSim. WCSim, in other sense, provides low sensibility due to apply a very large time step to evolve the simulated model.

B. Evaluating a scenario

This case study involves the assessment of the impact of adopting cloud bursting [42] for provisioning processing in a federated cloud environment by a public provider. We consider three levels of workload (low, average, and high) and four bare metal families (thin, medium, large, and huge). Memory and storage requirements are not taken into consideration. The federated cloud consists of three datacenters (DCs), each equipped with four processing servers (PSs). The federated cloud is homogeneous, with a communication bandwidth of 10 Gbps within each DC and 1 Gbps between PSs belonging to different DCs. We are evaluating two scenarios: in one scenario, only the three federated DCs provide processing resources, while in the other a public cloud provides processing resources when the federated cloud is detected as overloaded. Huge bare metals are provided only by the public cloud – the bandwidth between a PS in the federated cloud and any PS in the public cloud is 1 Gbps. Our case study limits the number of PSs provisioned at a time to four.

Three users, one on each DC, launch four virtual machines (VMs), each one with four virtual cores, and 1/3 of the entire workload at simulation time 0 (zero). The workload is described by DoBs reproducing the behavior of three Pegasus [48] applications: Spith, LIGO, and Galactic. Each user launch four applications, resulting in 12 DoBs submissions. The average level of workload is represented by each user launching 16 applications per DC (48 total), and the high level by the launching of 64 applications per DC (192 total). The families of bare metal differs among them according to the number of cores: 4, 12, 24, and 48, respectively. In all cases, the nominal speed of each core is 100,000 MIPS. There is a cost associated with core usage, which is \$0.03, \$0.07, \$0.08, and \$0.03, respectively thin to huge, per core.

Regarding the scheduling we consider two strategies. *SchBasic*: At launch time, each user's VMs are distributed in a circular manner among the PSs belonging to their home DC. As the jobs become ready to execute, their tasks are randomly distributed among the user's VMs. During execution, if a PS becomes overloaded, a VM hosted on it is (randomly) selected for migration (sender initiated policy) and a PS, belonging or not to the same DC, is (also randomly) selected to receive the VM. The VM migration is achieved over the network. *SchBurst*: This scheduling strategy has the same behavior as the previous scheduling, but it provides cloud bursting. The scheduler decides to *burst* a VM to a public provider when the PS receiving a migrating VM is also overloaded. The VM to be bursted is (randomly) selected among the VMs hosted by the receiving PS, including the migrated one.

We performs the simulation of both scenarios, with and without support of a public provider. The Table II presents the average execution time for each workflow (4, 16, and 64 of each one, depending on the processing demand) and also the costs of executing over the federated processing resources and the cost of executing in a public provider. The total cost is the sum of both. This cost considers only the consumed instructions, as identified in the performance file (Figure 5b presents a fragment of the actual file).

Table II shows that when the computational demand is low, the total execution time of the applications significantly decreases when the *thin* PSs are replaced by *medium* PSs, roughly by 50%, even though the cost is approximately 3.5 times higher. On the other hand, maintaining a *thin* infrastructure and adopting cloud bursting as support for processing peaks also reduces the average execution times of workflows, although at a lower rate compared to simply switching bare metal families, but with a marginal increase in operational costs. A possible conclusion is: it may be recommended to provision extra resources on-demand in a public cloud instead of investing in upgrades of PSs in the federated cloud. When the processing demand is classified as average, Table II shows that increasing the processing power of the PSs and/or adopting cloud bursting promotes performance improvement. In this situation, a thin architecture may not be interesting due to the lower performance it provides, while a large architecture may not be profitable if the processing demand is not constant in the cloud. In the last case, when the processing demand is high, we notice that cloud bursting significantly increases the performance, considering both the use of *medium* or *large* PSs. In this case, a possible conclusion is that the use of *medium* configurations for the PSs provides acceptable performance levels when provisioned, during processing peaks, by a public provider.

TABLE II: Average execution times of workflows and their costs. **BM**: Bare Metal; **FC**: Federated Cloud cost; **PP**: Public Provider cost; time in seconds; cost in \$.

BM	Sched.	Sipht	LIGO	Galactic	FC	PP	Total Cost
			Low computational demand				
Thin	SchBasic	73386	51772	15437	34.70	-	34.70
Medium	SchBasic	39102	25622	9906	126.93	-	126.93
Thin	SchBurst	34222	32042	12384	15.20	26.75	41.95
			Average computational demand				
Thin	SchBasic	1610784	1096387	385192	746.67	-	746.67
Medium	SchBasic	128353	102779	44195	461.09	-	461.09
Thin	SchBurst	165145	149767	54769	102.31	341.39	433.70
Medium	SchBurst	66547	36619	14103	265.05	133.00	398.04
Large	SchBasic	40487	31099	13152	342.52	-	342.52
			High computational demand				
Medium	SchBasic	1954715	2072887	740944	6,160.07	-	6,160.07
Medium	SchBurst	988912	648929	78300	4,031.96	2,142.64	6,164.60
Large	SchBasic	1310498	1031711	379766	12,159.96	-	12,159.96
Large	SchBurst	712883	361170	41562	7,493.27	1,640.18	9,133.45

The Table III complements the analysis by providing more information to a cloud designer to specify an infrastructure and the necessity of provisioning resources in a public cloud. In Table II, the performance corresponds to the execution of the total workload just once. The Table III presents an estimation of cloud usage, assuming that the workload will remain constant throughout a year. That is, once the workload is completed, an equal amount of work is launched again. In this table, **TAT** corresponds to the time (in hours) required to execute an entire workload and **Total** represents the cost of executing this workload continuously throughout a year. **Total** is the sum **AFC**, representing the cost of the consumed resources in the federated cloud, and **APP**, the cost of resources provisioned by a public provider (which are limited to four PSs at a time). All costs correspond to the effective utilization of the cores, as shown in the fragment of the trace file output presented in Figure 5b. The column **AFC available** indicates the cost of

TABLE III: Annual cost estimation for accommodating different demand projections. **AFC max**: Annual Federated Cloud maximum cost; **TAT**: Turn Around Time (hours); **AFC**: Annual Federated Cloud cost; **APP**: Annual Public Provider cost; **Total=AFC+APP**; **Diff.=Total-AFC**; Costs in $.

Bare metal	AFC max	Sched.	TAT	AFC	APP	Total	Diff.
Low computational demand							
Thin	12,614.40	SchBasic	24.66	12,324.30	-	12,324.30	290.10
Medium	88,300.80	SchBasic	12.59	88,316.66	-	88,316.66	-15.86
Thin	12,614.40	SchBurst	10.56	12,609.09	22,193.84	34,803.74	5.31
Average computational demand							
Thin	12,614.60	SchBasic	518.52	12,614.42	-	12,614.42	0.08
Medium	88,300.80	SchBasic	45.74	88,306.70	-	88,306.70	-5.90
Thin	12,614.60	SchBurst	71.05	12,614.15	42,091.81	54,705.96	0.25
Medium	88,300.80	SchBurst	26.29	88,316.30	44,308.98	132,625.28	-6.30
Large	201,830.30	SchBasic	14.87	201,780.44	-	201,829.46	49.86
High computational demand							
Medium	88,300.80	SchBasic	611.12	88,300.52	-	88,300.52	0.28
Large	201,830.30	SchBasic	527.78	201,828.88	-	201,828.88	1.41
Medium	88,300.80	SchBurst	400.00	88,299.92	46,924.30	135,224.22	0.88
Large	201,830.30	SchBurst	325.23	201.829.61	44,178.05	246,007.66	0.67

the federated infrastructure if all cores are active, processing instructions, 100% of the time.

The forth column in the table indicates that the low computational demand requires approximately 24.66 hours (TAT) to be computed by a federated cloud equipped with the weakest bare metal option. This means that, approximately, an entire workload is completed within a day over the course of a year. The cost of nearly 355 submissions, estimated during a year of execution, is $12,324.30 (AFC). As the total available processing corresponds to $12,614.40 (AFC max), there is $290.10 (Diff.) in change left over – or 9,670 hours of usage from one core, or yet 11 days of processing from the entire federated infrastructure. Upgrading the federated infrastructure to a medium bare metal will reduce the turnaround time by 50%, but it will cost slightly over $88K per year to compute, which is very close to the annual nominal capacity. On the other hand, by maintaining the thin infrastructure and provisioning the federated cloud bursting VMs, the investment of $22K (APP) is lower than what is required to compensate for the previous alternative (upgrade to medium bare metal), which amounts to nearly $78K ($88.3K - $12.6K).

Considering the scenario where the load is greater, representing an *average* computational demand, we observe poor performance provided by the thin bare metal. While the thin architecture allows completing 17 workload submissions, the medium completes 192 executions in a year. Large PSs provide even better performance, achieving 590 runs. Considering this demand level, a possible conclusion is that adopting cloud bursting instead of upgrading the infrastructure (from thin to medium or medium to large) results in significant improvement with lower investment.

Finally, the data obtained from simulating a high computational demand in the proposed cloud configuration allow us to conclude that a significant change in our cloud infrastructure must be considered. The architecture achieving the best performance completes 27 executions per year, requiring an investment of $246K ($201K in the federated cloud and $44K for a public provider).

VII. Conclusion

This paper presents WCSim (Workflow Cloud Simulator) tool for simulating scenarios in cloud computing. In contrast to the popular cloud simulation tools, SimGrid and CloudSim, its high accessibility potential was argued as a positive aspect. By allowing different scenarios to be described as input parameters to the simulator, users are relieved from the need to develop code to represent the elements of the simulated model. However, scheduling strategies still require intervention in the code to be described. Nevertheless, the fact that they are implemented as methods invoked in a callback approach simplifies the introduction of new scheduling strategies. As a proof of concept for the simulator's usage, two case studies were presented. One compared the results of WCSim with those presented by SimGrid and CloudSim Plus for a simple simulation scenario, while the other discussed the interest in adopting cloud bursting within a federated network.

Although fully functional and being an extension of an existing simulator [10], there are still some functionalities that need improvement. High level abstractions for file handling (storage and migration) are not yet available, and we understand that they are essential for a comprehensive performance analysis of workflow applications. Another aspect that needs improvement is the inclusion of a more realistic communication network model, as well as the introduction of an adaptive time step to enhance simulation sensitivity. Nevertheless, we believe that the capability to depicting entities like Users, Virtual Machines, and the concept of a workflow consisting of BoTs, seamlessly integrated into the developed simulation model, holds significant value for the community engaged in assessing diverse scenarios within a network environment.

The concept of DoB (Direct Acyclic Graph of Bag of Tasks) representing the workflow of BoTs associated to a simulator framework, is another important contribution introduced in this work. We have conceived a workflow structure as a program model for cloud computing, where units of work, called jobs, have precedence relationships among themselves and internally define a set of independent tasks, described as a bag of tasks. Refining this model and extending it to accommodate dependent tasks, thus allowing the simulation of MPI-like programs, will be the subject of future work.

As immediate extensions of the current work, we will propose more complex scenarios for comparing the results of WCSim with those presented by SimGrid and CloudSim Plus. Also it is projected to evaluate different scenarios of applications, infrastructures, and scheduling strategies in the context of cloud computing in order to help cloud designers to quantify the wastage of resources in specific use cases, such as HPC [50] or to specify platforms to support workflows conceived in the context of the Pegasus project [48].

Acknowledgment

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001.

References

[1] K. Fall and K. Varadhan, *The NS Manual*, 2nd ed., UC Berkeley: The VINT Project, 2011.

[2] N. Mansouri, R. Ghafari, and B. M. H. Zade, "Cloud computing simulators: A comprehensive review," *Simulation Modelling Practice and Theory*, vol. 104, p. 102144, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1569190X20300836

[3] A. Ahmed and A. S. Sabyasachi, "Cloud computing simulators: A detailed survey and future direction," in *2014 IEEE international advance computing conference (IACC)*. IEEE, 2014, pp. 866–872.

[4] K. Gupta, R. Beri, V. Behal, K. Gupta, R. Beri, and V. Behal, "Cloud computing: a survey on cloud simulation tools," *International Journal for Innovative Research in Science & Technology (IJIRST)*, vol. 2, no. 11, 2016.

[5] O. O. Oladimeji, D. Oyeyiola, O. Oladimeji, and P. Oyeyiola, "A comprehensive survey on cloud computing simulators," *Scientific journal of informatics*, vol. 8, no. 1, pp. 51–59, 2021.

[6] S. J. E. Taylor, A. Khan, K. L. Morse, A. Tolk, L. Yilmaz, and J. Zander, "Grand challenges on the theory of modeling and simulation," in *Proceedings of the Symposium on Theory of Modeling and Simulation - DEVS Integrative M&S Symposium*, ser. DEVS 13. San Diego, CA, USA: Society for Computer Simulation International, 2013.

[7] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Versatile, scalable, and accurate simulation of distributed applications and platforms," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2899–2917, Jun. 2014. [Online]. Available: http://hal.inria.fr/hal-01017319

[8] R. N. Calheiros, R. Ranjan, C. A. F. D. Rose, and R. Buyya, "Cloudsim: A novel framework for modeling and simulation of cloud computing infrastructures and services," *CoRR*, vol. abs/0903.2525, 2009. [Online]. Available: http://arxiv.org/abs/0903.2525

[9] M. C. Silva Filho, R. L. Oliveira, C. C. Monteiro, P. R. M. Inácio, and M. M. Freire, "Cloudsim plus: A cloud computing simulation framework pursuing software engineering principles for improved modularity, extensibility and correctness," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. Lisbon, Portugal: IEEE, 2017, pp. 400–406.

[10] N. Nononono, "Nonononono nono no nonon n oonon non nonon," in *9999 Nonononono nono no nonon n oonon non nonon*. Nono nono: Nononono, 9999, p. 99.

[11] ——, "Non nononono nono nnonon noonon non nonon," in *XIXI Non Nononono Nono Nnonon Noonon Non Nonon*. Nono nono: Nononono, XIXI, p. 99.

[12] B. S. Onggo, S. Taylor, and A. Tulegenov, "The need for cloud-based simulation from the perspective of simulation practitioners," in *Proceedings of the Operational Research Society simulation workshop*, 2014, pp. 103–112.

[13] A. Nunez, J. L. Vazquez-Poletti, A. C. Caminero, J. Carretero, and I. M. Llorente, "Design of a new cloud computing simulation platform," in *Computational Science and Its Applications-ICCSA 2011: International Conference, Santander, Spain, June 20-23, 2011. Proceedings, Part III 11*. Springer, 2011, pp. 582–593.

[14] J. J. Padilla, S. Y. Diallo, A. Barraco, C. J. Lynch, and H. Kavak, "Cloud-based simulators: Making simulations accessible to non-experts and experts alike," in *Proceedings of the Winter Simulation Conference 2014*. IEEE, 2014, pp. 3630–3639.

[15] S. E. Chafi, Y. Balboul, S. Mazer, M. Fattah, M. E. Bekkali, and B. Bernoussi, "Cloud computing services, models and simulation tools," *International Journal of Cloud Computing*, vol. 10, no. 5-6, pp. 533–547, 2021.

[16] G. Kecskemeti, "Dissect-cf: A simulator to foster energy-aware scheduling in infrastructure clouds," *Simulation Modelling Practice and Theory*, vol. 58P2, pp. 188–218, 11 2015.

[17] G. K. A. Al-Haboobi, "Developing a workflow management system simulation for capturing internal iaas behavioural knowledge," *Journal of Grid Computing*, vol. 21, no. 2, 2023.

[18] W. Depoorter, N. De Moor, K. Vanmechelen, and J. Broeckhove, "Scalability of grid simulators: An evaluation," in *Euro-Par 2008 – Parallel Processing*, E. Luque, T. Margalef, and D. Benítez, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 544–553.

[19] C. E. Ramamonjisoa, L. Z. Khodja, D. Laiymani, A. Giersch, and R. Couturier, "Simulation of asynchronous iterative algorithms using simgrid," in *2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC, CSS, ICESS)*, IEEE. Paris, France: IEEE, 2014, pp. 890–895.

[20] A. Pham, T. Jéron, and M. Quinson, "Verifying mpi applications with simgridmc," in *Proceedings of the First International Workshop on Software Correctness for HPC Applications*. Denver, CO, USA: Association for Computing Machinery, 2017, pp. 28–33.

[21] A. B. M. Fanfakh, "Predicting the performance of mpi applications over different grid architectures," *Journal of University of Babylon for Pure and Applied Sciences*, vol. 27, no. 1, pp. 468–477, 2019.

[22] H. Casanova, A. Legrand, M. Quinson, and F. Suter, "Smpi courseware: Teaching distributed-memory computing with mpi in simulation," in *2018 IEEE/ACM Workshop on Education for High-Performance Computing (EduHPC)*, IEEE. Dallas, USA: IEEE, 2018, pp. 21–30.

[23] I. Daoudi, P. Virouleau, T. Gautier, S. Thibault, and O. Aumage, "somp: Simulating openmp task-based applications with numa effects," in *International Workshop on OpenMP*. Lyon, France: INRIA, 09 2020, pp. 197–211.

[24] A. Mohammed, A. Eleliemy, and F. M. Ciorba, "Towards the reproduction of selected dynamic loop scheduling experiments using simgrid-simdag," in *2017 IEEE 19th International Conference on High Performance Computing and Communications; IEEE 15th International Conference on Smart City; IEEE 3rd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. Bangkok, Thailand: IEEE, 2017, pp. 623–626.

[25] R. Buyya and M. M. Murshed, "Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing," *Concurr. Comput. Pract. Exp.*, vol. 14, no. 13-15, pp. 1175–1220, 2002. [Online]. Available: https://doi.org/10.1002/cpe.710

[26] F. Howell and R. McNab, "Simjava: A discrete event simulation library for java," *Simulation Series*, vol. 30, pp. 51–56, 1998.

[27] M. Bendechache, S. Svorobej, P. T. Endo, M. N. Mario, M. E. Ares, J. Byrne, and T. Lynn, "Modelling and simulation of elasticsearch using cloudsim," in *2019 IEEE/ACM 23rd International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, IEEE. Cosenza, Italy: IEEE, 2019, pp. 1–8.

[28] S. Narang, P. Goswami, and A. Jain, "Statistical analysis of cloud based scheduling heuristics," in *International Conference on Information, Communication and Computing Technology*, Springer. Singapore: Springer Singapore, 2019, pp. 98–112.

[29] ——, "A comprehensive review of load balancing techniques in cloud computing and their simulation with cloudsim plus," *Recent Advances in Computer Science and Communications (Formerly: Recent Patents on Computer Science)*, vol. 14, no. 6, pp. 1684–1694, 2021.

[30] D. I. Hatti and A. V. Sutagundar, "Resource provisioning in fog-based iot," in *Inventive Computation and Information Technologies*. Coimbatore, India: Springer, 2022, pp. 433–447.

[31] D. Li, C. Asikaburu, B. Dong, H. Zhou, and S. Azizi, "Towards optimal system deployment for edge computing: a preliminary study," in *2020 29th International Conference on Computer Communications and Networks (ICCCN)*, IEEE. Honolulu, HI, USA: IEEE, 2020, pp. 1–6.

[32] D. Li, C. Asikaburu, J. Shang, and N. Wang, "Numerical and simulation verification for optimal server allocation in edge computing," in *2021 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS)*, IEEE. Toronto, ON, Canada: IEEE, 2021, pp. 1–7.

[33] W. Chen and E. Deelman, "Workflowsim: A toolkit for simulating scientific workflows in distributed environments," in *2012 IEEE 8th International Conference on E-Science*, 2012, pp. 1–8.

[34] F. Mastenbroek, G. Andreadis, S. Jounaid, W. Lai, J. Burley, J. Bosch, E. van Eyk, L. Versluis, V. van Beek, and A. Iosup, "Opendc 2.0: Convenient modeling and simulation of emerging technologies in cloud datacenters," in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. Melbourne, Australia: IEEE, 2021, pp. 455–464.

[35] T. Hirofuchi, A. lèbre, and L. Pouilloux, "Simgrid vm: Virtual machine support for a simulation framework of distributed systems," *IEEE Transactions on Cloud Computing*, vol. PP, 01 2016.

[36] J. Arabnejad H., Barbosa, "A budget constrained scheduling algorithm for workflow applications," *J Grid Computing*, vol. 12, p. 665–679, 2014.

[37] R. Buyya, M. M. Murshed, D. Abramson, and S. Venugopal, "Scheduling parameter sweep applications on global grids: a deadline and budget constrained cost–time optimization algorithm," *Software: Practice and Experience*, vol. 35, 2005.

[38] H. Mehta, P. Kanungo, and M. Chandwani, “Ecogrid: a dynamically configurable object oriented simulation environment for economy-based grid scheduling algorithms,” in *Proceedings of the Fourth Annual ACM Bangalore Conference*. Bangalore, India: Association for Computing Machinery, 2011, pp. 1–8.

[39] V. Arabnejad, K. Bubendorfer, and B. Ng, “Budget and deadline aware e-science workflow scheduling in clouds,” *IEEE Transactions on Parallel and Distributed systems*, vol. 30, no. 1, pp. 29–44, 2018.

[40] A. A. Nasr, N. A. El-Bahnasawy, G. Attiya, and A. El-Sayed, “Cost-effective algorithm for workflow scheduling in cloud computing under deadline constraint,” *Arabian Journal for Science and Engineering*, vol. 44, no. 4, pp. 3765–3780, 2019.

[41] S. Gupta, R. S. Singh, U. D. Vasant, and V. Saxena, “User defined weight based budget and deadline constrained workflow scheduling in cloud,” *Concurrency and Computation: Practice and Experience*, vol. 33, no. 24, p. e6454, 2021.

[42] M. Mattess, C. Vecchiola, S. K. Garg, and R. Buyya, “Cloud bursting: Managing peak loads by leasing public cloud services,” *Cloud Computing: Methodology, Systems, and Applications*, pp. 343–367, 2011.

[43] A. Iosup and D. Epema, “Grid Computing Workloads,” *IEEE Internet Computing*, vol. 15, no. 2, pp. 19–26, Mar. 2011. [Online]. Available: http://ieeexplore.ieee.org/document/5620891/

[44] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, “Characterizing and profiling scientific workflows,” *Future Generation Computer Systems*, vol. 29, no. 3, pp. 682–692, Mar. 2013.

[45] J. Yu and R. Buyya, “A taxonomy of scientific workflow systems for grid computing,” *SIGMOD Rec.*, vol. 34, no. 3, p. 44–49, sep 2005. [Online]. Available: https://doi.org/10.1145/1084805.1084814

[46] A. Wijewickrama, R. Wijayawardana, K. Ranasinghe, D. N. Ranasinghe, K. P. M. K. Silva, and K. Karunanayaka, “Sciflow: A composable framework for developing scientific workflows on hpc clusters,” in *2020 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, 2020, pp. 275–284.

[47] C. Ramon-Cortes, P. Alvarez, F. Lordan, J. Alvarez, J. Ejarque, and R. M. Badia, “A survey on the distributed computing stack,” *Computer Science Review*, vol. 42, p. 100422, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574013721000629

[48] Pegasus, “Pegasus WMS – Automate, recover and debug scientific computations,” <https://pegasus.isi.edu/>. Access in 22 May, 2023., 2022.

[49] G. Wainer and P. Mosterman, *Discrete-Event Modeling and Simulation: Theory and Applications*, ser. Computational Analysis, Synthesis, and Design of Dynamic Systems. CRC Press, 2018. [Online]. Available: https://books.google.com.br/books?id=WQvzk7ZnwHkC

[50] W. F. C. Tavares, M. Roberto Miranda Assis, and E. Borin, “Quantifying and detecting hpc resource wastage in cloud environments,” in *2021 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW)*, 2021, pp. 41–46.