A Parallel Execution Strategy for GPU-accelerated Cloud-based Scientific Workflows

Murilo B. Stockinger, Filipe Santiago
Instituto de Computação
Universidade Federal Fluminense
Niterói, RJ, Brazil
{murilobruggerstockinger,filipe_santiago}@id.uff.br

Marcos A. Guerine
Instituto Federal de Educação, Ciência e Tecnologia do Rio de Janeiro
Arraial do Cabo, RJ, Brazil
mguerine@id.uff.br

Yuri Frota, Isabel Rosseti
Instituto de Computação
Universidade Federal Fluminense
Niterói, RJ, Brazil
{yuri,rosseti}@ic.uff.br

Ubiratam de Paula
Departamento de Ciência da Computação
Universidade Federal Rural do Rio de Janeiro
Nova Iguaçu, RJ, Brazil
upaula@ufrrj.br

Kary Ocaña
Laboratório Nacional de Computação Científica - LNCC
Petrópolis, RJ, Brazil
karyann@lncc.br

Alexandre Plastino, Daniel de Oliveira
Instituto de Computação
Universidade Federal Fluminense
Niterói, RJ, Brazil
{plastino,danielcmo}@ic.uff.br

***Resumo*—Several complex scientific simulations process large amounts of distributed and heterogeneous data. These simulations are commonly modeled as scientific workflows and require HPC environments to produce results timely. Although scientists already benefit from clusters and clouds, new hardware, such as General Purpose Graphical Processing Units (GPGPUs), can be used to speedup the execution. Clouds also provide virtual machines (VMs) with GPU capabilities that can also be used, thus becoming hybrid clouds. This way, many workflows can be modeled considering programs that run in GPUs and/or in CPUs. A problem that arises is how to schedule workflows in this hybrid environment. Although existing workflow systems (WfMS) can execute in GPGPUs and clouds independently, they do not provide mechanisms for scheduling the workflow in this hybrid environment. In fact, reducing the makespan and the financial cost in hybrid clouds is a difficult task. In this paper, we present a scheduling strategy for cloud-based and GPU-accelerated workflows, named PROFOUND, which schedules activations (atomic tasks) to a set of CPU and GPU/CPU VMs. PROFOUND is based on a combination of a mathematical formulation and a heuristic, and aims at minimizing not only the makespan, but also the financial cost. To evaluate PROFOUND, we used a set of benchmark instances based on synthetic and real scenarios gathered from different workflows traces. The experiments show that PROFOUND is able to solve the referred scheduling problem.**

***Index Terms*—hybrid clouds, GPGPU, workflows, scientific experiments, scheduling**

I. Introduction

The effective need to process heterogeneous, distributed, and large volumes of data has increased dramatically over the last decade [1]. These data are generated by several types of applications, especially complex scientific simulations from many domains of science (*e.g.,* biology, astronomy, chemistry, physics, and engineering). These simulations are commonly part of data- and compute-intensive *in silico* experiments [1], [2].

An *in silico* scientific experiment may be implemented in many ways, *e.g.*, using Python scripts, makefiles and scientific workflows [2]. Scientific workflows are abstractions used to model a coherent chaining of scientific programs (henceforth named only as programs), where the outcome of a specific program is the input of another program. This way, a workflow can be formally defined as a directed acyclic graph $W(A, Dep)$, where the set of nodes $A = \{a_1, a_2, ..., a_n\}$ represents the activities and the set of edges Dep represents the data dependencies among the activities in A [3].

Let us define D as the set of all data consumed and produced by W. For each $a_i \in A$, we consider a set of input data $\Delta_{in}(a_i) \subseteq D$ needed for its execution and a set of output data $\Delta_{out}(a_i) \subseteq D$ generated by it. We say that activity a_j depends on activity a_i, represented as $dep(a_i, a_j)$, if and only if $\exists d \in \Delta_{in}(a_j) | d \in \Delta_{out}(a_i)$.

Let us also define as *activation* [2] the smallest unit of work that can be processed in parallel and consumes a specific data chunk [4]. Let us consider N as the set of activations of the workflow W. Each activation $i \in N$ is associated with a specific activity $a_i \in A$ that is represented as $act(i) = a_i$, *i.e.,* the same activity a_i may be associated with several different activations. Thus, activations also present data dependencies, *i.e.,* considering activations i and j, we say that $dep(i, j) \leftrightarrow \exists d \in \Delta_{in}(j) | d \in \Delta_{out}(i)$. Figure 1 presents the different abstraction levels that we consider in this paper. The Activity Level represents the higher level of abstraction where the scientist defines the conceptual activities of the workflow (*e.g.,* Multiple Sequence Analysis and Format Conversion). Each activity can be implemented by one or many programs at the Program Level (*e.g.,* MAFFT and CSMA for the first activity and ReadSeq for the second). At the Activation Level,

Authors would like to thank CNPq and FAPERJ. This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001. This research made use of Montage, which is funded by the National Science Foundation under Grant Number ACI-1440620, and was previously funded by the NASA's Earth Science Technology Office, Computation Technologies Project, under Cooperative Agreement Number NCC5-626 between NASA and the C.I.T.

we can visualize the several executions of an activity (*e.g.,* activations i_1 to i_6), where each activation consumes a specific portion of data (data chunk).

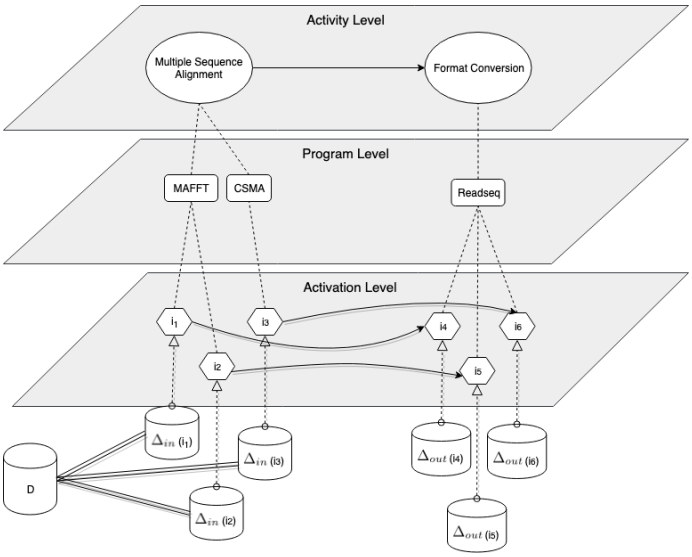

Figura 1. Different levels of abstraction in the SciPhy [5] workflow

Such workflows are commonly managed by complex engines named Workflow Management Systems (WfMS), which are used to orchestrate, dispatch, execute, monitor and analyze data-intensive experiments. Existing WfMS are Swift/T [6], Pegasus [7], and SciCumulus [8]. As the complexity of the scientific workflows grows in terms of exploration of thousands of large datasets (several GBs or even TBs) and hundreds of parameters (*i.e.,* parameter sweep), the performance requirements for such workflows have been pushing the envelope on the capacity of sequential systems (*e.g.,* computers with a few of processors) for a while already [9]. Due to the intensive computation of large scale workflows, parallel hardware is required to produce results timely. Distributed environments such as clusters, supercomputers, grids, and clouds are commonly used to execute workflows in parallel.

However, over the last years, the several improvements in the programmability of General Purpose Graphical Processing Units (GPGPUs) have made this type of hardware an attractive platform for computing- and data-intensive applications, such as many scientific workflows. The idea is to use GPGPUs as co-processors to speed up data parallel processing. GPGPUs are massively parallel devices that can execute SIMD (Simple Instruction Multiple Data) programs very efficiently. Thus, GPGPUs have become a new computing paradigm and have been largely used in HPC systems. Today's hybrid clusters (with CPUs and GPUs) play a fundamental role in HPC. Using GPGPUs, higher degrees of parallelism can be achieved by executing complex programs since each GPGPU is composed of hundreds of cores that can execute thousands of threads simultaneously, in contrast to CPUs that can handle only a few threads at a time. These GPU-accelerated programs are several times faster than those CPU-based ones. In fact, the evolution of CUDA [10] programming model allowed the development of GPU-accelerated programs in several domains of science [11]–[13].

However, acquiring GPGPUs may be quite expensive for many scientists (especially the ones in underdeveloped countries). For example, a single NVIDIA Tesla P100 SXM2 with 16GB may cost approximately U$9,428.00[1] and a NVIDIA DGX GPU cluster may easily cost more than US$100,000.00. This way, several cloud providers also attach graphics acceleration to existing VM types, such as Amazon AWS and Microsoft Azure, thus becoming what we call in this paper a "hybrid cloud". By deploying GPU-based VMs, scientists are able to acquire resources on-demand, based on the pay-per-use pricing model, *e.g.,* scientists pay per quantum of time used (*e.g.,* minutes, hours, days *etc.*). These VMs are suited for any workload that needs an amount of graphics acceleration such as those mentioned above as GPU-accelerated scientific programs.

Although such VMs represents a step forward, they introduce many challenges for executing scientific workflows. One of these challenges is how to schedule workflows in a hybrid virtual cluster, with CPU and GPU VMs, considering the makespan and financial issues. In addition, the scientific workflow may also be "hybrid". For instance, a specific activity of the workflow may be implemented by different versions of the same program (what we call a variability in the activity) (*e.g.,* one implemented in C++ and other in CUDA as presented in Figure 1, where MAFFT is implemented in C++ and CSMA in CUDA for GPUs) and, depending on the status of the workflow execution, one version can be more suitable for executing in a specific VM than the other, *i.e.,* all versions of the same program are interchangeable during execution.

This way, to enable workflow execution in a hybrid cloud, the execution of each activation should be scheduled to a corresponding VM. Then, the scheduling problem is to decide where to execute the activations. In general, map the execution of activations to distributed computing resources is an NP-hard problem. The objectives can be to reduce makespan and/or financial cost, to maximize performance, reliability *etc*. Thus, to execute a workflow in a hybrid cloud, all activations have to be scheduled and executed in parallel on a set of VMs $M = \{m_1, m_2, \ldots, m_m\}$, which may present heterogeneous characteristics (*i.e.,* CPU or CPU/GPU - that we call dual machines). To the best of authors' knowledge, only Swift/T and Pegasus WfMS can execute workflows in GPGPUs. However, none of them is able to schedule workflows on hybrid clouds neither to consider workflow variabilities during scheduling.

In this paper, we present a cloud-based GPU-accelerated workflow scheduling strategy named PROFOUND (g**P**u acce**l**e**R**ated w**O**rk**F**l**O**w sched**U**ling i**N** clou**D**s). PROFOUND is based on a combination of a mathematical model and a heuristic, and schedules workflow activations to a set of VMs in hybrid clouds iteratively. At each iteration, PROFOUND estimates the execution times and the necessary portion of data

[1]https://www.microway.com/hpc-tech-tips/nvidia-tesla-p100-price-analysis/

of each activation for specific VMs. An activation is assigned to a VM that is expected to complete it at the earliest time. PROFOUND can be implemented in the scheduler component of a WfMS.

It is worth mentioning that in this paper, we assume that all VMs are deployed before the workflow execution, and we only use on-demand VMs. Although clouds traditionally provide on-demand VMs, most of the providers also have reserved VMs to be used. Commonly, reserved VMs present a financial discount and capacity reservation. It is also important to mention that PROFOUND aims at scheduling program executions for a specific GPU, *i.e.,* it does not consider kernel (portion of CUDA code) scheduling in different cores of the GPU. In this way, PROFOUND can be coupled to existing approaches such as StarPU [14]. Thus, the main contributions of this paper are: (i) The formulation of the activation scheduling problem as a mixed integer programming problem; (ii) The design of an exact and heuristic methods for scheduling workflow activations in the hybrid cloud; and (iii) An extensive experimental evaluation, based on real and synthetic workflow traces that shows the advantages and benefits of PROFOUND.

In order to be evaluated, PROFOUND is implemented within SciCumulus, which is a cloud workflow engine that supports parallel execution of scientific workflows in clouds with provenance support. Differently from the current mainstream, SciCumulus automatically adapts the workflow execution according to the current state of the environment, thus benefiting from cloud elasticity. In addition, SciCumulus captures and provides runtime provenance data, *i.e.,* registers information about the workflow being executed and makes this data available for queries at runtime. Provenance data can be used as input for PROFOUND to provide a better scheduling plan.

The remainder of this paper is organized as follows. Sections II presents the background on GPGPUs. Section III brings related work. Section IV presents the problem definition and the mathematical formulation. Section V presents the proposed heuristic approach. Experimental results are presented in Section VI, and, finally, Section VII concludes this paper and presents future work.

II. General Purpose Graphical Processing Units

GPGPUs are programmable co-processors initially developed for graphic processing. Each GPGPU is composed of multiple Streaming Multiprocessors (SMs). Each SM can execute thousands of threads concurrently and contains a specific number of cores each. GPGPUs can be programmed using particular languages such as CUDA (for NVIDIA GPGPUS) and OpenCL. In a CUDA program, each fragment of code is named kernel, and may be executed by hundreds of threads depending on the GPGPU processing capacity. In GPGPUs, threads are organized in blocks, which, in turn, are grouped in grids. All threads within the same block can access the shared memory. This helps synchronize threads within the same block. For example, for NVIDIA Fermi and Kepler GPGPUs, one block can have 1024 threads maximum. Threads in a block are split into warps to execute in the same SM. A warp may be defined as a set of threads that share the same code and follow the same execution path. There is a standard pipeline for executing programs on GPUs. In general, users pre-process the data on CPU (data augmentation, cleaning *etc.*), then load small portions of pre-processed data on the GPU, since all data may not fit in GPU memory or transfer overhead is non-negligible compared to total execution time. Commonly raw data is split into several chunks, and each chunk is loaded to GPU to be processed. This way, several outputs are generated and have to be transferred from GPU to CPU to produce the final result.

III. Related Work

Over the last decade, a growing number of scheduling strategies and algorithms were proposed. Many of these algorithms are focused on clouds but restricted to CPU-based VMs, as surveyed by Liu *et al.* [15], which makes them not directly applicable to hybrid clouds. Although there are some approaches to schedule workflows in hybrid clouds [16]–[19], to the best of authors' knowledge, none of the existing approaches consider all characteristics that are considered in this paper. This way, the problem of efficient workflow scheduling algorithms in hybrid clouds is still an open, yet important problem.

Shweta *et al.* [16] propose an automated workflow tool to perform AMBER GPGPUs simulations. The proposed workflow was developed on top of Kepler WfMs. The proposed approach eases the access to GPU clusters but does not propose a scheduling strategy for execution workflows in GPGPUs. In fact, it uses the standard Kepler's schedulers. In addition, the authors do not consider hybrid clouds neither variabilities in the workflow. Blattner *et al.* [17] propose the Hybrid Task Graph Scheduler (HTGS) for scheduling workflows in multi-core and multi-GPU systems. The proposed approach aims at keeping multiple GPUs occupied, and uses all available computing resources. Although the authors consider GPGPUs, they do not take into account workflow variabilities neither financial issues that are fundamental for cloud-based workflows.

Jiang *et al.* [18] propose an approach to schedule MapReduce-based applications in multi-GPU environments. Although this approach provides sophisticated mechanisms that benefit from GPU features such as streams and Hyper-Q as well as hard disk utilization, it is decoupled from the scientific workflow concept. In addition, it does not deal with the scenario where GPUs are virtualized. Shirahata *et al.* [19] propose a hybrid scheduling algorithm for GPU-based applications, which aims at minimizing the makespan and queue time. Authors only consider workflows that can be modeled as Map and Reduce tasks, which is quite restrictive. Although authors achieved good results in their experiments, they do not consider the scenario where GPUs are virtualized neither variabilities in the workflow specification.

IV. Mathematical Formulation

The problem of scheduling workflows with variabilities in a hybrid cloud can be formulated as a mixed integer programming problem, as follows. Let us redefine $M = M^{cpu} \cup M^{dual}$ as the set of all VMs available for execution or storage, where M^{cpu} (M^{dual} - with CPU/GPU) is the set of VMs with a CPU (dual) processor. Each VM $j \in M$ has a storage capacity cm_j and a financial cost c_j (per quantum of time). The set of activations N is composed by activations that are only executed in CPU processors (N^{cpu}), activations that are only executed in GPU processors (N^{gpu}) and activations that can be executed in both CPU and GPU processors (N^{dual}). In order to simplify the notation, we denote $M_i \subseteq M$ as the set of VMs that can execute activation $i \in N$. Note that an activation from the set N^{gpu} can only be executed by VMs from the set M^{dual}. On the other hand, an activation from the set $N^{cpu} \cup N^{dual}$ can be executed in all VMs.

Let us also redefine $D = D_s \cup D_d$ as the set of all data, where each data $d \in D$ has a size $W(d)$ and can be either static (D_s), with an origin machine $O(d) \in M$, or dynamic (D_d), *i.e.*, generated during the workflow execution. We also define the user's requirements C_M as the maximum estimated financial cost, and T_M as the maximum execution time for the workflow, where $T = \{1, \ldots, T_M\}$ is denoted as the set of feasible periods of time. Moreover, we define t_{ij}^{cpu} (t_{ij}^{gpu}) as the processing time of activation $i \in N$ executed on a CPU (GPU) processor, on machine $j \in M$. Similarly, we define $\overrightarrow{t}_{djp}$ as the time spent by machine $j \in M$ to read the data $d \in D$ stored in machine $p \in M$, and $\overleftarrow{t}_{djp}$ as the time of machine $j \in M$ to write data $d \in D_d$ in machine $p \in M$. Table I summarizes the data used in the proposed mathematical formulation while Table II describes the binary and continuous decision variables of the model.

The objective function (1) pursues both the minimization of execution time (makespan) and financial costs. The parameter φ defines the weight of each objective (defined by the user). Note that both objectives are normalized using T_M and C_M respectively.

$$\min \varphi\left(\frac{z_T}{T_M}\right) + (1-\varphi)\left(\frac{\sum_{j \in M} \sum_{t \in T} z_{jt} \cdot c_j}{C_M}\right) \tag{1}$$

Constraints (2-4) guarantee that every activation must be executed. Constraints (5) and (6) rule that every read and write operations must be accomplished, respectively.

$$\sum_{j \in M} \sum_{t \in T} x_{ijt}^{cpu} = 1, \qquad \forall i \in N^{cpu} \tag{2}$$

$$\sum_{j \in M^{dual}} \sum_{t \in T} x_{ijt}^{gpu} = 1, \qquad \forall i \in N^{gpu} \tag{3}$$

$$\sum_{j \in M} \sum_{t \in T} x_{ijt}^{cpu} + \sum_{j \in M^{dual}} \sum_{t \in T} x_{ijt}^{gpu} = 1, \qquad \forall i \in N^{dual} \tag{4}$$

$$\sum_{j \in M_i} \sum_{p \in M} \sum_{t \in T} \overrightarrow{x}_{idjpt} = 1, \qquad \forall i \in N, \forall d \in \Delta_{in}(i) \tag{5}$$

$$\sum_{j \in M_i} \sum_{p \in M} \sum_{t \in T} \overleftarrow{x}_{djpt} = 1, \qquad \forall d \in D_d, \text{ such } d \in \Delta_{out}(i) \text{ and } i \in N \tag{6}$$

Inequalities (7-10) guarantee that data $d \in \Delta_{out}(i)$ can only be written if activation i was executed in the correct time.

$$\overleftarrow{x}_{djpt} \leq \sum_{q=1}^{t-t_{ij}^{cpu}} x_{ijq}^{cpu}, \quad \forall d \in D_d, \forall j \in M^{cpu}, \forall p \in M, \forall t = (t_{ij}^{cpu}+1) \cdots T_M, \text{ such } d \in \Delta_{out}(i) \text{ and } i \in (N^{cpu} \cup N^{dual}) \tag{7}$$

$$\overleftarrow{x}_{djpt} \leq \sum_{q=1}^{t-t_{ij}^{gpu}} x_{ijq}^{gpu}, \quad \forall d \in D_d, \forall j \in M^{dual}, \forall p \in M, \forall t = (t_{ij}^{gpu}+1) \cdots T_M, \text{ such } d \in \Delta_{out}(i) \text{ and } i \in N^{gpu} \tag{8}$$

$$\overleftarrow{x}_{djpt} \leq \sum_{q=1}^{t-t_{ij}^{cpu}} x_{ijq}^{cpu}, \quad \forall d \in D_d, \forall j \in M^{dual}, \forall p \in M, \forall t = (t_{ij}^{cpu}+1) \cdots T_M, \text{ such } d \in \Delta_{out}(i) \text{ and } i \in N^{cpu} \tag{9}$$

$$\overleftarrow{x}_{djpt} \leq \sum_{q=1}^{t-t_{ij}^{cpu}} x_{ijq}^{cpu} + \sum_{q=1}^{t-t_{ij}^{gpu}} x_{ijq}^{gpu}, \qquad \forall d \in D_d, \forall j \in M^{dual}, \forall p \in M, \forall t = (t^{min}+1) \cdots T_M, \text{ such } t^{min} = min\{t_{ij}^{cpu}, t_{ij}^{gpu}\}, d \in \Delta_{out}(i) \text{ and } i \in N^{dual}, \text{ where } (t - t_{ij}^{cpu}) \geq 1 \text{ and } (t - t_{ij}^{gpu}) \geq 1 \tag{10}$$

Furthermore, constraints (11-14) define that data d cannot be written before the processing time of the activation i (responsible for its writing). Note that constraints (7-14) work together to guarantee a feasibly time for the writing process.

$$\overleftarrow{x}_{djpt} = 0, \qquad \forall d \in D_d, \forall j \in M^{cpu}, \forall p \in M, 1 \leq t \leq t_{ij}^{cpu}, \text{ such } d \in \Delta_{out}(i) \text{ and } i \in (N^{cpu} \cup N^{dual}) \tag{11}$$

$$\overleftarrow{x}_{djpt} = 0, \qquad \forall d \in D_d, \forall j \in M^{dual}, \forall p \in M, 1 \leq t \leq t_{ij}^{gpu}, \text{ such } d \in \Delta_{out}(i) \text{ and } i \in N^{gpu} \tag{12}$$

$$\overleftarrow{x}_{djpt} = 0, \qquad \forall d \in D_d, \forall j \in M^{dual}, \forall p \in M, 1 \leq t \leq t_{ij}^{cpu}, \text{ such } d \in \Delta_{out}(i) \text{ and } i \in N^{cpu} \tag{13}$$

Tabela I
LIST OF PARAMETERS AND THEIR DESCRIPTIONS.

Data	*Description*
D_s	Set of static data.
D_d	Set of dynamic data.
$D = D_s \cup D_d$	Set of data.
$O(d)$	Origin machine of static data $d \in D_s$.
$W(d)$	Size of data $d \in D$.
N^{cpu}	Set of activations that can be only executed in a CPU processor.
N^{gpu}	Set of activations that can be only executed in a GPU processor.
N^{dual}	Set of activations that can be executed in both processors type (CPU and GPU).
$N = N^{cpu} \cup N^{gpu} \cup N^{dual}$	Set of activations.
M^{cpu}	Set of VMs with a CPU processor.
M^{dual}	Set of VMs with CPU and GPU processors.
$M = M^{cpu} \cup M^{dual}$	Set of VMs.
$T = \{1...T_M\}$	Set of feasible periods of time, where T_M is the maximum execution time for the workflow.
c_j	Financial cost of purchasing VM $j \in M$ for one period of time.
C_M	Maximum estimated financial cost to execute the workflow.
t_{ij}^{cpu}	Processing time of activation $i \in (N^{cpu} \cup N^{dual})$ in VM $j \in M$ on a CPU processor.
t_{ij}^{gpu}	Processing time of activation $i \in (N^{gpu} \cup N^{dual})$ in VM $j \in M^{dual}$ on a GPU processor.
$\overrightarrow{t}_{djp}$	Time spent by VM $j \in M$ to read data $d \in D$ stored in VM $p \in M$.
$\overleftarrow{t}_{djp}$	Time spent by VM $j \in M$ to write data $d \in D_d$ stored in VM $p \in M$.
$\Delta_{in}(i) \subseteq D$	Set of data needed for the execution of activation $i \in N$.
$\Delta_{out}(i) \subseteq D_d$	Set of data generated by activation $i \in N$.
cm_j	Storage capacity of the VM j.

Tabela II
LIST OF VARIABLES AND THEIR DESCRIPTIONS.

Variables	*Description*
x_{ijt}^{cpu}	Binary variable which indicates whether activation $i \in N^{cpu} \cup N^{dual}$ begins its execution in VM $j \in M$ on a CPU processor at period $t \in T$ or not.
x_{ijt}^{gpu}	Binary variable which indicates whether activation $i \in N^{gpu} \cup N^{dual}$ begins its execution in VM $j \in M^{dual}$ on a GPU processor at period $t \in T$ or not.
$\overrightarrow{x}_{idjpt}$	Binary variable which indicates whether activation $i \in N$ executing in VM $j \in M_i$ begins to read data $d \in \Delta_{in}(i)$ stored in VM $p \in M$ at period $t \in T$ or not.
$\overleftarrow{x}_{djpt}$	Binary variable which indicates whether data $d \in D_d$ begins to be written from VM $j \in M_i$ (where $d \in \Delta_{out}(i)$) to VM $p \in M$ at period $t \in T$ or not.
y_{djt}	Binary variable which indicates whether data $d \in D$ is stored in VM $j \in M$ at period $t \in T$ or not.
z_{jt}	Binary variable which indicates whether VM $j \in M$ was allocated (hired) at period $t \in T$ or not.
z_T	Continuous variable which indicates the total time to execute the workflow (makespan).

$$\overleftarrow{x}_{djpt} = 0, \quad \forall d \in D_d, \forall j \in M^{dual}, \forall p \in M, 1 \leq t \leq min\{t_{ij}^{cpu}, t_{ij}^{gpu}\} \text{ such } d \in \Delta_{out}(i) \text{ and } i \in N^{dual} \tag{14}$$

$$x_{ijt}^{gpu} \leq \sum_{p \in M} \sum_{q=1}^{t-\overrightarrow{t}_{djp}} \overrightarrow{x}_{idjpq}, \quad \forall i \in N^{gpu}, \forall d \in \Delta_{in}(i), \forall j \in M^{dual}, \forall t \in T \text{ such } (t - \overrightarrow{t}_{djp}) \geq 1 \tag{16}$$

Constraints (15-18) rule that an activation can only be executed if all necessary reads were concluded in a feasible time.

$$x_{ijt}^{cpu} \leq \sum_{p \in M} \sum_{q=1}^{t-\overrightarrow{t}_{djp}} \overrightarrow{x}_{idjpq}, \quad \forall i \in (N^{cpu} \cup N^{dual}), \forall d \in \Delta_{in}(i), \forall j \in M^{cpu}, \forall t \in T \text{ such } (t - \overrightarrow{t}_{djp}) \geq 1 \tag{15}$$

$$x_{ijt}^{cpu} \leq \sum_{p \in M} \sum_{q=1}^{t-\overrightarrow{t}_{djp}} \overrightarrow{x}_{idjpq}, \quad \forall i \in N^{cpu}, \forall d \in \Delta_{in}(i), \forall j \in M^{dual}, \forall t \in T \text{ such } (t - \overrightarrow{t}_{djp}) \geq 1 \tag{17}$$

$$x_{ijt}^{cpu} + x_{ijt}^{gpu} \leq \sum_{p \in M} \sum_{q=1}^{t-\overrightarrow{t}_{djp}} \overrightarrow{x}_{idjpq}, \qquad \forall i \in N^{dual}, \forall d \in \Delta_{in}(i), \forall j \in M^{dual}, \forall t \in T, \text{ such } (t - \overrightarrow{t}_{djp}) \geq 1 \tag{18}$$

Inequalities (19) and (20) guarantee that only one action (execution, reading or writing) can be accomplished at each period of time in each VM (*e.g.,* a VM cannot execute an activation and write data at the same time).

$$\sum_{i \in (N^{cpu} \cup N^{dual})} \sum_{q=\max(1, t-t_{ij}^{cpu}+1)}^{t} x_{ijq}^{cpu} + \sum_{i \in (N^{cpu} \cup N^{dual})} \sum_{d \in \Delta_{out}(i)} \sum_{p \in M} \sum_{r=\max(1, t-\overleftarrow{t}_{djp}+1)}^{t} \overleftarrow{x}_{djpr} + \sum_{i \in (N^{cpu} \cup N^{dual})} \sum_{d \in \Delta_{in}(i)} \sum_{p \in M} \sum_{r=\max(1, t-\overrightarrow{t}_{djp}+1)}^{t} \overrightarrow{x}_{idjpr} \leq z_{jt}, \forall j \in M^{cpu}, \forall t \in T \tag{19}$$

$$\sum_{i \in (N^{cpu} \cup N^{dual})} \sum_{q=\max(1, t-t_{ij}^{cpu}+1)}^{t} x_{ijq}^{cpu} + \sum_{i \in (N^{gpu} \cup N^{dual})} \sum_{r=\max(1, t-t_{ij}^{gpu}+1)}^{t} x_{ijr}^{gpu} + \sum_{i \in N} \sum_{d \in \Delta_{out}(i)} \sum_{p \in M} \sum_{r=\max(1, t-\overleftarrow{t}_{djp}+1)}^{t} \overleftarrow{x}_{djpr} + \sum_{i \in N} \sum_{d \in \Delta_{in}(i)} \sum_{p \in M} \sum_{r=\max(1, t-\overrightarrow{t}_{djp}+1)}^{t} \overrightarrow{x}_{idjpr} \leq z_{jt}, \forall j \in M^{dual}, \forall t \in T \tag{20}$$

Moreover, constraints (21) and (22) together with constraints (19) and (20) guarantee the correct interpretation of variables z_{jt}. Note that constraints (19) and (20) ensure that a VM is allocated during an active action (execute, read or write). On the other hand, constraints (21) guarantee a feasible VM allocation during a passive action (to be read or to be written). Finally, constraints (22) ensure continuous hiring periods (*i.e.*, if a VM is hired at time $t+1$, then it must also be hired at time t).

$$\sum_{i \in N} \sum_{d \in \Delta_{in}(i)} \sum_{p \in M_i} \sum_{r=\max(1, t-\overrightarrow{t}_{dpj}+1)}^{t} \overrightarrow{x}_{idpjr} + \sum_{i \in N} \sum_{d \in \Delta_{out}(i)} \sum_{p \in M_i} \sum_{r=\max(1, t-\overleftarrow{t}_{dpj}+1)}^{t} \overleftarrow{x}_{dpjr} \leq z_{jt}.(|N|.|D|), \forall j \in M, \forall t \in T \tag{21}$$

$$z_{j(t+1)} \leq z_{jt}, \qquad \forall j \in M, \forall t \in \{1...T_M - 1\} \tag{22}$$

Constraints (23) establish that there are no dynamic data at starting time. On the other hand, constraints (24) guarantee that all static data are already stored on their origin machines.

$$y_{dj1} = 0, \qquad \forall d \in D_d, \forall j \in M \tag{23}$$

$$y_{djt} = 1, \qquad \forall d \in D_s \mid j \in O(d), \forall t \in T \tag{24}$$

Constraints (25-27) link the storage variable y with the write variable $\overleftarrow{x}$ and the read variable $\overrightarrow{x}$, guaranteeing a feasible write and read process, respectively.

$$y_{dp(t+1)} \leq y_{dpt}, \qquad \forall d \in D_s, \forall p \in M, \forall t \in \{1...T_M - 1\} \tag{25}$$

$$y_{dp(t+1)} \leq y_{dpt} + \sum_{j \in M_i} \overleftarrow{x}_{djp(t-\overleftarrow{t}_{djp}+1)}, \quad \forall d \in D_d, \forall p \in M, \forall t \in \{1...T_M - 1\}, \text{ such } d \in \Delta_{out}(i) \text{ and } (t - \overleftarrow{t}_{djp} + 1) \geq 1 \tag{26}$$

$$\sum_{j \in M_i} \overrightarrow{x}_{idjpt} \leq |M_i|.y_{dpt}, \qquad \forall i \in N, \forall d \in \Delta_{in}(i), \forall p \in M, \forall t \in T \tag{27}$$

In detail, constraints (27) ensure that data will only be read if previously stored in a VM, and constraint (25) and (26) ensures that data will only be stored in a VM if it has already been produced (written).

The VMs storage capacity is bounded by constraints (28). Constraints (29) relate the last write operation with the application execution time (makespan). Note that in our application model an activation always creates data.

$$\sum_{d \in D} y_{djt} W(d) \leq cm_j, \qquad \forall j \in M, \forall t \in T \tag{28}$$

$$\overleftarrow{x}_{djpt} \cdot (t + \overleftarrow{t}_{djp}) \leq z_T, \qquad \forall d \in D_d, \forall j \in M_i, \forall p \in M, \forall t \in T, \text{ such } d \in \Delta_{out}(i) \tag{29}$$

Furthermore, the following operational constraints (30) must be satisfied: an activation i can only begin any reading process if all data $d \in \Delta_{in}(i)$ is already available (*i.e.*, if all data $d \in (\Delta_{in}(i) \cap D_d)$ is written). Finally, the remaining constraints are the integrality and non-negativity constraints.

$$\overrightarrow{x}_{idjpt} \cdot |\Delta_{in}(i) \cap D_d| \leq \sum_{\substack{g \in \{\Delta_{in}(i) \cap D_d\} \\ such \ g \in \Delta_{out}(h)}} \sum_{l \in M_h} \sum_{o \in M} \sum_{u=1}^{t-\overleftarrow{t}_{glo}} \overleftarrow{x}_{glou}, \forall i \in N, \forall d \in \Delta_{in}(i), \forall j \in M_i, \forall p \in M, \forall t \in T \tag{30}$$

V. Heuristic Approach

Due to the complex characteristics and size of instances (*i.e.,* workflow traces) for the aforementioned problem, the effective usage of exact methods seems to be impractical. Although small-size instances can be solved by such models using appropriate solvers, larger instances are usually time and memory consuming, demanding a fast and more effective alternative approach. In addition, even when the solver finds a feasible solution, it cannot take much time to do that, because otherwise it could add a non-negligible time to the workflow execution, *i.e.,* the scheduling time cannot be higher than the execution time of the workflow.

A good alternative is the use of metaheuristics. As stated by Gendreau and Potvin [20], metaheuristics are methods that orchestrate an interaction between local improvement procedures and higher level strategies to create a process capable of escaping from local optima and performing a robust search in a solution space.

Greedy Randomized Adaptive Search Procedures [21] (GRASP for short) is a simple and iterative metaheuristic that has been successfully applied to a large class of optimization problems. Each GRASP iteration is divided into two stages. First, a feasible solution is built in a construction phase. Then, in a second phase, its neighborhood is explored by a local search procedure in order to find a better solution. The best solution found in all the iterations is taken as a result.

GRASP's construction phase is a well-known heuristic that seeks to balance greediness and randomness choices during the construction of a solution. At each iteration of the construction procedure, it considers a set of candidate elements that can be added to the partial solution under construction. After evaluating all candidates according to a greedy function (which usually computes the incremental cost of that element in the current solution), the list of candidates is created and sorted. Then, a restricted candidate list (RCL) is defined, composed of the best candidates, using a threshold parameter $\alpha \in [0, 1]$ to truncate the candidate list, *i.e.,* element e will be part of the RCL only if its cost $c(e) \in [c^{min}, c^{min} + \alpha(c^{max} - c^{min})]$ (Considering a minimization problem). Finally, one element from RCL is chosen at random to be part of the solution.

Thus, in this paper, we also propose a greedy randomized constructive heuristic, as a part of PROFOUND strategy, to solve the workflow scheduling in hybrid clouds problem, based on the constructive phase of the GRASP metaheuristic. Our main motivation is that such constructive method is suitable to find solutions whenever the former mathematical model is not able to find an integer feasible solution to the problem, either reaching a predetermined time limit or running out of memory due to a huge number of restrictions.

The constructive algorithm is described in Algorithm 1. Initially, a loop that runs until all activations are executed in a VM is performed. In this loop, a list of candidates (LC) is built by combining each activation, each possible VM for processing, and each possible VM to write the output data files on – lines 5 to 12. Since some activations may require another activations to be processed beforehand (due to data dependencies), the candidates whose activations have unprocessed precedent activations will not be added to the LC – line 8. The cost for each candidate is calculated based on its insertion on the current solution – line 9. After every possible permutation of activation, processing machine and output machine is computed, the LC is ordered increasingly by cost – line 13. At the end of the loop, a restricted candidate list (RCL) will be created according to the α parameter and, finally, a random candidate selected from RCL will be added to the current solution – lines 14 to 16.

Given the randomized characteristics of the constructive algorithm, a solution may choose different candidates on each execution of the constructive procedure. To maximize the search for better solutions, the algorithm is executed multiple times and all generated solutions are stored. The best overall solution is reported.

The objective function is calculated in Algorithm 2. This function is a composition of both makespan and financial cost of VM (exactly as shown in Equation 1). The calculated values are normalized by precomputed values called C_M and T_M, presented in Table III. Either time or money can be prioritized on the objective function by setting the multiplier $\varphi \in [0, 1]$, a user defined parameter. By default it is set to 0.5.

Algorithm 1: Greedy Randomized Constructive Heuristic

```
   Input: α, C_M, T_M
   Output: A solution for workflow scheduling in hybrid clouds
           problem
1  s ← ∅;
2  ExecutedActivations ← 0;
3  while ExecutedActivations < |N| do
4      LC ← ∅;
5      foreach activation j ∈ N do
6          foreach VM v ∈ M do
7              foreach OutputMachine o ∈ M do
8                  if j can be executed then
9                      OFcost ← calculateOF(s, v, j, o, C_M,
                         T_M);
10                     LC ← Candidate(v, j, o, OFcost);
11                 else
12                     try next activation;
13     LC ← orderByOFCost(LC);
14     RCL ← createRCL(LC, α);
15     c ← getRandomCandidate(RCL);
16     s ← doMovement(c);
17     ExecutedActivations ← ExecutedActivations + 1;
18 return s;
```

Algorithm 2: calculateOF

```
   Input: s, v, j, o, C_M, T_M
   Output: The Value of Objective Function
1  φ ← 0.5 ;
2  OFValue ← 0 ;
3  mkspn ← getMakespan(s) ;
4  cost ← getCost(s) ;
5  OFValue ← φ*(mkspn / T_M) + (1 - φ)*(cost / C_M) ;
6  return OFValue;
```

VI. Experimental Evaluation

To evaluate PROFOUND, a series of experiments was conducted. The main results achieved so far are presented and discussed in this section. Thus, we first provide information on how the experimentation environment and the experiments were set up. Then, we discuss the experimental evaluation of PROFOUND. All data used in the experiments presented in this section is available for download in our group repository in GitHub[2].

We have performed four experiments. In the first experiment, we set the values of C_M and T_M for the objective function. In the second experiment, we compared the exact results with the ones generated by the heuristic using 'toy instances". In the third experiment, we executed the heuristic for larger workflow traces (*i.e.,* instances that cannot be solved by the mathematical model). All traces were obtained in the Workflow Generator[3] web site. Finally, in the fourth experiment, we executed the Phylogenetic workflow SciPhy [5] in SciCumulus WfMS [8] in AmazAWS cloud and compared the makespan of a greedy scheduling and the scheduling plan generated by PROFOUND.

The mathematical formulation explained in Section IV were implemented with compiler gcc version 4.8.5 and solved using the IBM ILOG CPLEX version 12.5.1. The experiments with CPLEX were carried out on a personal computer Intel®Xeon(R) CPU E5-2620 v3 @ 2.40GHz, with 64GB RAM running under CentOS 7.6.1810 OS. The real executions of SciPhy used Amazon AWS. Each deployed VM for running SciPhy presented in this paper uses Linux Cent OS 5.5. To execute SciPhy, each input file is aligned using MAFFT version 6.857 (CPU) or CMSA (https://github.com/wangvsa/CMSA - GPU), this way, the first activity presents a variability. Each alignment is used as input to the program ModelGenerator version 0.85. Then, the alignment and evolutionary model are used as input to construct phylogenetic trees using RAxML-7.2.8-ALPHA. The proposed heuristic was executed 10 times, each execution with a different random seed. The total number of constructions was set to 100. The value of input parameters α, defined in preliminary experiments, was set to 0.2. Heuristic experiments were carried out on a personal computer with Intel®Core$^{\mathrm{TM}}$ i7-8700K CPU @ 3.70GHz with 16GB RAM running under operating system Linux Ubuntu version 18.04 LTS with parallel processing features disabled.

As stated before, the first experiment was executed to set the values of C_M and T_M. The objective function requires both C_M and T_M values to be calculated for each instance and inserted as parameters for both the constructive heuristic and the formulation. The initial approach to estimate those parameters was to predict the most expensive solution and the longest makespan for each instance, and use those values as C_M and T_M. However, since the formulation size grows rapidly with the number of periods of time, we have used the aforementioned initial estimation to find, for each instance, rough solutions using the proposed constructive heuristic. This way, we obtained new values of makespan and cost, that should be doubled in order to compute larger estimations and, finally, used as C_M and T_M for each instance. Table III presents, for each instance, the values of C_M and T_M used in the experiments, as well as the number of activations and number of data files.

In the second experiment, we created a series of toy workflow traces to compare the results of the exact and the heuristic methods. Table IV reports the computational experiments of the heuristic and the CPLEX execution of the mathematical formulation. The first column identifies the instance. Second column reports, for each instance, the value of the best solution's objective function found over 10 executions of the heuristic. Third and fourth columns present, respectively, the best makespan and financial cost values achieved by the proposed heuristic. The fifth column shows the average execution time of the heuristic. In a similar way, the remaining columns present information regarding the CPLEX execution of the mathematical model. The results indicate that the mathematical model was able to report the optimal solutions for all instances. On the other hand, the greedy randomized heuristic was able to yield good solutions with small gap, in average 5.5%, but was not able to find the optimal solution for 7 instances (5_toy_10_A, 5_toy_10_B, 5_toy_10_C, 3_toy_15_A, 5_toy_15_A, 5_toy_15_B, 5_toy_15_C). It is worth noticing, however, that the execution times for the heuristic are drastically lower (always less than a hundredth of a second) than the ones reported by the CPLEX execution.

In the third experiment, we executed the heuristic for larger instances. Since we did not have access for GPU-accelerated workflows different from SciPhy, the benchmark instances were artificially generated using real information from existing workflows' executions. We have obtained traces from Montage, Cybershake, Inspiral and Epigenomics. Since these traces do not consider activities that execute in GPUs, we have artificially modified the traces to add GPU variabilities for each activity of the workflow. In this experiment, the execution time of an activity in GPUs is calculated as $rand(cpu_{time} * 0.3, cpu_{time} * 0.7)$, where $rand$ randomly selects the time reduction factor.

Table V shows the computational experiments for the larger instances (based on real workflow traces) by the greedy randomized heuristic. The columns of this table corresponds to the ones of Table IV, except for the *Avg.* column, which reports the average of the objective function values obtained by the heuristic. The instances are much larger than the ones shown on Table IV. For that reason, the CPLEX model could not report any integer feasible solutions since the formulation grew too large and too fast to fit RAM memory. For these larger instances, the proposed heuristic was also able to find solutions very fast, proving its importance.

In the last experiment, we performed a real execution of one instance of SciPhy (SciPhy_200) using SciCumulus. We have compared the makespan produced by the execution of the standard scheduling plan of SciCumulus (6.52 hours in

[2]https://github.com/UFFeScience/Wf-GPU

[3]https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator

Tabela III
CHARACTERISTICS OF THE INSTANCES

Instances	***Activations***	***Data***	T_M	C_M
3_toy_5_A	2	3	10	120.00
5_toy_5_A	2	3	10	120.00
3_toy_5_B	2	3	35	120.00
5_toy_5_B	2	3	16	108.00
3_toy_5_C	1	4	6	80.00
5_toy_5_C	1	4	14	96.00
3_toy_10_A	4	6	46	168.00
5_toy_10_A	4	6	82	268.00
3_toy_10_B	3	7	16	198.00
5_toy_10_B	3	7	22	300.00
3_toy_10_C	4	6	42	440.00
5_toy_10_C	4	6	68	450.00
3_toy_15_A	5	10	36	154.00
5_toy_15_A	5	10	38	88.00
3_toy_15_B	5	10	52	54.00
5_toy_15_B	5	10	44	46.00
3_toy_15_C	5	10	54	136.00
5_toy_15_C	5	10	54	56.00
CyberShake_30	30	49	2,820	282.20
CyberShake_50	50	79	2,332	1,758.04
CyberShake_100	100	154	4,604	5,353.16
Epigenomics_24	24	38	6,788	9,888.78
Epigenomics_46	46	71	21,252	3,377.80
Epigenomics_100	100	152	106,878	197,180.00
Montage_25	25	38	340	454.36
Montage_50	50	53	816	138.62
Montage_100	100	93	1,018	1,792.26
Inspiral_30	30	47	2,456	772.28
Inspiral_50	50	77	3,006	2,044.60
Inspiral_100	100	151	6,140	3,008.92
Sipht_30	29	963	5,888	669.40
Sipht_60	58	1,049	8,774	4,422.60
Sipht_100	97	1,121	15,160	2,256.00
SciPhy_200	155	310	39,798	3,980,520.00

average) and the scheduling plan provided by PROFOUND (5.99 hours in average). Although this is an initial result and more experiments are needed, we can state that the integration of PROFOUND using with SciCumulus is promising since we can state a difference of 8.01% on the overall performance of the execution following PROFOUND scheduling plan.

VII. CONCLUSIONS AND FUTURE WORK

The use of clouds for executing workflows has been growing in the last few years, but the usage of those assets may become expensive if not planned in an optimal way. For this reason, a better scheduling plan for a given workflow is essential for keeping costs as low as possible while still having access to the high-processing power VMs available on hybrid cloud services. In this paper, we address the problem of scheduling workflow activations in hybrid clouds, *i.e.,* that are composed of CPU and GPU VMs. The problem consists in scheduling the execution of activations into a set of CPU/GPU VMs. Besides, each activation may consume or produce data files, which can be stored in the same VM or any other available. We seek to minimize not only the makespan but also the financial costs of this scheduling. Hence, we propose PROFOUND, a combination of mathematical formulation and a heuristic procedure to solve this problem. To evaluate our proposal, we introduced a set of benchmark instances that are based on both artificial and real workflow traces. The proposed approach aims at creating a scheduling plan that respects their dependencies, aiming at the lower financial cost and shorter makespan solution.

The mathematical formulation was solved by CPLEX, a high-performance mathematical programming solver. The formulation yielded optimal solutions when dealing with the small workflow traces, acquiring solutions for every instance. On the other hand, the solver was not able to cope with larger instances, which are based on real scenarios, since the model became too large to fit RAM memory and, therefore, could not achieve feasible solutions. The same benchmark instances were used to evaluate the proposed heuristic. For the set of small instances, the heuristic was able to find 11 out of 18 optimal solutions found by the exact method. For the set of larger instances artificially created based on the traces provided by the Workflow Generator web site, the heuristic reported feasible solutions for all instances. Moreover, the proposed heuristic found solutions within a small computational time, always less than a few seconds, showing its importance as a good alternative for the mathematical model. We also evaluated the scheduling plan provided by PROFOUND with the workflow SciPhy in a real execution. The scheduling plan provided by PROFOUND produced an execution 8.01% faster than the standard SciCumulus WfMs scheduler.

As a future work, we plan to extend the study on GPU-accelerated cloud-based scientific workflows by implementing local and global searches procedures in order to enhance solution quality of the proposed heuristic. Another interesting future work is the combination of the heuristic with data mining techniques, in which patterns extracted from good solutions can be used to guide the search in the solution space [22].

REFERÊNCIAS

[1] T. Hey, S. Tansley, K. M. Tolle *et al.*, *The fourth paradigm: data-intensive scientific discovery*. Microsoft research Redmond, WA, 2009, vol. 1.

[2] D. C. M. de Oliveira, J. Liu, and E. Pacitti, *Data-Intensive Workflow Management: For Clouds and Data-Intensive and Scalable Computing Environments*, ser. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2019.

[3] D. de Oliveira, K. A. C. S. Ocaña, F. A. Baião, and M. Mattoso, "A provenance-based adaptive scheduling heuristic for parallel scientific workflows in clouds," *J. Grid Comput.*, vol. 10, pp. 521–552, 2012.

[4] J. Liu, E. Pacitti, P. Valduriez, D. de Oliveira, and M. Mattoso, "Multi-objective scheduling of scientific workflows in multisite clouds," *Future Generation Comp. Syst.*, vol. 63, pp. 76–95, 2016.

[5] K. Ocaña, D. de Oliveira, E. S. Ogasawara, A. M. R. Dávila, A. A. B. Lima, and M. Mattoso, "SciPhy: A Cloud-Based Workflow for Phylogenetic Analysis of Drug Targets in Protozoan Genomes." in *BSB*, 2011, pp. 66–70.

[6] J. M. Wozniak, T. G. Armstrong, M. Wilde, D. S. Katz, E. Lusk, and I. T. Foster, "Swift/t: Large-scale application composition via distributed-memory dataflow processing," in *13th CCGrid*, 2013, pp. 95–102.

Tabela IV
COMPUTATIONAL RESULTS OF GREEDY RANDOMIZED HEURISTIC AND MATHEMATICAL MODEL ON SMALL INSTANCES

	Heuristic				Model			
Instances	**Best**	**Makespan**	**Cost**	**Time (s)**	**Opt**	**Makespan**	**Cost**	**Time (s)**
3_toy_5_A	0.60	7	60	< 0.01	0.60	7	60	0.15
3_toy_5_B	0.37	10	54	< 0.01	0.37	10	54	1.14
3_toy_5_C	0.96	7	60	< 0.01	0.96	7	60	0.04
5_toy_5_A	0.60	7	60	< 0.01	0.60	7	60	0.19
5_toy_5_B	0.56	10	54	< 0.01	0.56	10	54	0.42
5_toy_5_C	0.71	11	60	< 0.01	0.71	11	60	0.20
3_toy_10_A	0.48	26	65	< 0.01	0.48	26	65	78.23
3_toy_10_B	0.56	10	98	< 0.01	0.56	10	98	1.94
3_toy_10_C	0.45	20	190	< 0.01	0.45	20	190	11.51
5_toy_10_A	0.49	46	114	< 0.01	0.42	40	96	2,087.65
5_toy_10_B	0.53	11	169	< 0.01	0.49	11	143	6.04
5_toy_10_C	0.54	40	218	< 0.01	0.44	32	186	225.29
3_toy_15_A	0.39	23	22	< 0.01	0.37	22	21	68.50
3_toy_15_B	0.42	23	22	< 0.01	0.42	23	22	37.10
3_toy_15_C	0.35	27	26	< 0.01	0.35	27	26	310.53
5_toy_15_A	0.46	21	32	< 0.01	0.42	20	28	359.32
5_toy_15_B	0.54	25	24	< 0.01	0.46	21	20	312.84
5_toy_15_C	0.55	31	30	< 0.01	0.46	26	25	5,896.59

Tabela V
COMPUTATIONAL RESULTS OF GREEDY RANDOMIZED HEURISTIC ON LARGER INSTANCES

Instances	**Best**	**Avg.**	**Makespan**	**Cost**	**Time (s)**
CyberShake_30	0.53	0.59	1367	161.39	< 0.01
CyberShake_50	0.26	0.34	678	418.84	< 0.01
CyberShake_100	0.26	0.40	1,342	1,275.72	0.01
Epigenomics_24	0.44	0.50	2,890	4,557.87	< 0.01
Epigenomics_46	0.45	0.46	11,614	1,209.05	< 0.01
Epigenomics_100	0.50	0.51	49,151	106,856.00	0.01
Montage_25	0.41	0.43	124	206.56	< 0.01
Montage_50	0.39	0.41	355	48.90	< 0.01
Montage_100	0.48	0.52	458	924.13	0.02
Inspiral_30	0.44	0.46	1,583	183.14	< 0.01
Inspiral_50	0.48	0.50	1,955	642.73	< 0.01
Inspiral_100	0.46	0.47	3,844	903.30	0.01
Sipht_30	0.47	0.48	2,741	321.50	0.01
Sipht_60	0.38	0.41	5,173	750.90	0.02
Sipht_100	0.47	0.48	7,057	1,079.50	0.05
SciPhy_200	0.65	0.69	21,056	3,060,680.00	1.81

[7] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. C. Laity, J. C. Jacob, and D. S. Katz, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, vol. 13, pp. 219–237, 2005.

[8] D. de Oliveira, E. Ogasawara, F. Baião, and M. Mattoso, "Scicumulus: A Lightweight Cloud Middleware to Explore Many Task Computing Paradigm in Scientific Workflows," in *3rd International Conference on Cloud Computing*, 2010, pp. 378–385.

[9] C. Hsu, G. C. Fox, G. Min, and S. Sharma, "Advances in big data programming, system software and HPC convergence," *The Journal of Supercomputing*, vol. 75, pp. 489–493, 2019.

[10] A. B. Hayes, F. Hua, J. Huang, Y. Chen, and E. Z. Zhang, "Decoding CUDA binary," in *IEEE/ACM International Symposium on Code Generation and Optimization, CGO 2019, Washington, DC, USA, February 16-20, 2019*, 2019, pp. 229–241.

[11] V. Mallawaarachchi, A. Wickramarachchi, A. Welivita, I. Perera, and D. Meedeniya, "Efficient bioinformatics computations through gpu accelerated web services," in *Proc. of ICACS '18*. New York, NY, USA: ACM, 2018, pp. 94–98.

[12] C. V. Staggs, Z. Shi, and D. Lee, "Biocloud: Using gpu architecture for bioinformatics tools: Extended abstract," in *Proceedings of ACM SE '17*. New York, NY, USA: ACM, 2017, pp. 259–261.

[13] K. van der Veldt, R. van Nieuwpoort, A. L. Varbanescu, and C. Jesshope, "A polyphase filter for gpus and multi-core processors," in *Proceedings of the 2012 Workshop on High-Performance Computing for Astronomy Date*, ser. Astro-HPC '12. New York, NY, USA: ACM, 2012, pp. 33–40.

[14] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier, "Starpu: a unified platform for task scheduling on heterogeneous multicore architectures," *Concurrency and Computation: Practice and Experience*, vol. 23, pp. 187–198, 2011.

[15] J. Liu, E. Pacitti, P. Valduriez, and M. Mattoso, "A survey of data-intensive scientific workflow management," *J. Grid Comput.*, vol. 13, pp. 457–493, 2015.

[16] S. Purawat, P. U. Ieong, R. D. Malmstrom, G. J. Chan, A. K. Yeung, R. C. Walker, I. Altintas, and R. E. Amaro, "A kepler workflow tool for reproducible amber gpu molecular dynamics," *Biophysical Journal*, vol. 112, pp. 2469 – 2474, 2017.

[17] T. Blattner, W. Keyrouz, S. S. Bhattacharyya, M. Halem, and M. Brady, "A hybrid task graph scheduler for high performance image processing workflows," *Signal Processing Systems*, vol. 89, pp. 457–467, 2017.

[18] H. Jiang, Y. Chen, Z. Qiao, T. Weng, and K. Li, "Scaling up mapreduce-based big data processing on multi-gpu systems," *Cluster Computing*, vol. 18, pp. 369–383, 2015.

[19] K. Shirahata, H. Sato, and S. Matsuoka, "Hybrid map task scheduling for gpu-based heterogeneous clusters," in *2010 IEEE International Conference on Cloud Computing Technology and Science*, Nov 2010, pp. 733–740.

[20] M. Gendreau and J.-Y. Potvin, *Handbook of Metaheuristics*, 2nd ed., ser. International Series in Operations Research & Management Science. Springer, 2010, vol. 146.

[21] T. A. Feo and M. G. C. Resende, "Greedy randomized adaptive search procedures," *Journal of Global Optimization*, vol. 6, pp. 109–133, 1995.

[22] S. L. Martins, I. Rosseti, and A. Plastino, *Data Mining in Stochastic Local Search*. Cham: Springer International Publishing, 2018, pp. 39–87.