Have We Seen These Data Before? A GRASP-Based Execution Strategy for Cloud-based Workflows With Memoization

Rodrigo A. P. Silva, Gaëtan Heidsieck, Esther Pacitti, Patrick Valduriez, Yuri Y. Frota, Daniel de Oliveira

Have We Seen these Data Before? A GRASP-based Execution Strategy for Cloud-based Workflows with Memoization

Paper Accepted on Concurrency and Computation: Practice and Experience

Rodrigo A. P. Silva (Universidade Federal Fluminense – Brazil)
Gaëtan Heidsieck (Georg-August University of Göttingen – Germany)
Esther Pacitti (Inria, University of Montpellier, CNRS, LIRMM – France)
Patrick Valduriez (Inria, University of Montpellier, CNRS, LIRMM – France)
Yuri Frota (Universidade Federal Fluminense – Brazil)
Daniel de Oliveira (Universidade Federal Fluminense – Brazil)

Abstract

Scientific workflows are used to model experiments that rely on computer simulations. Because these workflows are typically data-intensive, they commonly re quire execution in distributed environments. The cloud, with on-demand and elastic resources, has emerged as a cost-effective environment for workflow execution. However, the efficiency and cost-effectiveness of cloud workflow execution depend on how the workflow is executed on these resources. In particular, reusing cached data to avoid re-executing parts of the workflow is critical for performance but hard to make effective. This manuscript introduces `MemoirGRASP`, a workflow execution strategy based on the GRASP metaheuristic that optimizes execution while exploring memoization. The memoization technique involves caching intermediate results, reducing workflow execution redundancy, and enhancing efficiency across multiple workflow executions. The experimental evaluation of `MemoirGRASP` using synthetic workflows and two real-world workflows demonstrates the advantages and benefits of the `MemoirGRASP` strategy in improving workflow execution efficiency.

1 Introduction

Scientific workflows (or workflows for short in the rest of the manuscript) have been used successfully to model numerical experiments based on computer simulations [dOLP19, BDZ06, LAB+09]. Workflows are usually represented as graphs of activities (*i.e.*, programs, scripts or services invocations), in which nodes correspond to data processing activities and edges represent the dataflow among them [BDZ06]. In today's big data world, many workflows consume and produce large volumes of data, thus becoming data-intensive workflows [Suk21, dOLP19]. Considering that the activities of a data-intensive workflow usually process big data, each activity can be decomposed into several executable tasks, which we call *activations* [OdOV+11]. To reduce workflow execution time, each activation can be executed in parallel by consuming a different set of parameters or data partitions.

For complex workflows (both in terms of structure or volume of data to process), the cloud is a suitable environment since it provides on-demand and elastic resources (*e.g.*, VMs, storage, and more recently serverless functions [LCB+22]). In this manuscript, we consider the execution of data-intensive workflows in a multisite cloud, *i.e.*, a cloud with geo-distributed data centers (or sites). Today, all popular public clouds, *e.g.*, Microsoft Azure, Amazon AWS, and Google Cloud Platform, provide a multisite option that allows accessing multiple cloud sites with a single cloud account. The main reason for using multiple sites is that the application requirements may exceed the capabilities of a single site, either because the site imposes usage limits for fairness and security, or because the datasets are just too big.

Most existing workflow systems provide support for cloud-based execution. Some well-known examples are Parsl [BWL+19], eScience Central [WHW10], Pegasus [DdSV+21], OpenAlea [PFVB15] and SciCumulus [dOOBM12]. These workflow systems natively provide mechanisms to execute workflows efficiently, *e.g.*, (1) high-speed ingestion components, (2) data fragmentation algorithms, and (3) several types of scheduling algorithms that aim to minimize total execution time, reliability, financial cost, *etc.* Some of our previous work [NSPdO21, HdOP+21, LPP+19, SGdPJ+22] has dealt with these issues.

However, there is still much room for optimization. One important aspect of the scientific process is that users commonly execute the same workflow specification as many times as needed, varying a subset of the parameters and input data to confirm or refute a hypothesis. Such workflow executions may share *workflow fragments* (*i.e.*, sets of activations and their data dependencies) that consume and produce the same input and output data, respectively. If these intermediate data produced by previous workflow executions could be reused, we could avoid re-execution of some workflow fragments. Such caching mechanism uses the Memoization[1] [Mic68] technique, storing the result of *pure activations* defined by the user. A pure activation is conceptually similar to a "relaxed" pure function in programming languages, meaning it is deterministic and produces no side effects during workflow execution. In other words, pure activations in the context of this manuscript always return the same results when they are executed by consuming the same input parameters and input data [GE10].

By exploiting memoization, one can reuse computation across multiple and related workflows, thus reducing total execution time and financial cost (in public clouds). Another advantage is fostering data sharing with other users. Let us illustrate the advantages of memoization with the simple scenario in Figure 1. Figure 1(a) shows a workflow with nine activities. Activities three, five, and seven are data-intensive, *i.e.*, they require several hours to execute and produce large volumes of data. Although all activities can be re-executed every time the workflow is enacted, one can identify that activities three, five, and seven were already executed with the same parameters in a previous execution (Figure 1(b)), not necessarily from the same user. Then, previously cached intermediate data can be retrieved and sent to activities six and nine during the current workflow execution (Figure 1(c)).

However, to exploit memoization, the workflow scheduler must consider intermediate cached data. Some workflow systems, such as OpenAlea [HdOP+20] and Kepler [OPP+16] support data caching. In a previous work [HdOP+20], we proposed a method to determine the most suitable intermediate data to cache (by estimating the ratio between re-computation cost and storage cost) and an execution strategy that exploits memoization for executing workflows in a multisite cloud. The approach is based on a greedy algorithm, which may fail to produce the globally optimal solution because it is not aware of future choices that could be made.

In this manuscript, we propose a workflow scheduling strategy based on the GRASP (Greedy Randomized

[1]The Memoization technique is inspired by the "Memo" functions[Mic68] where the result of a function in a program or script is properly stored to be reused in the future, if and only if the function consumes the same parameter values of the original one.

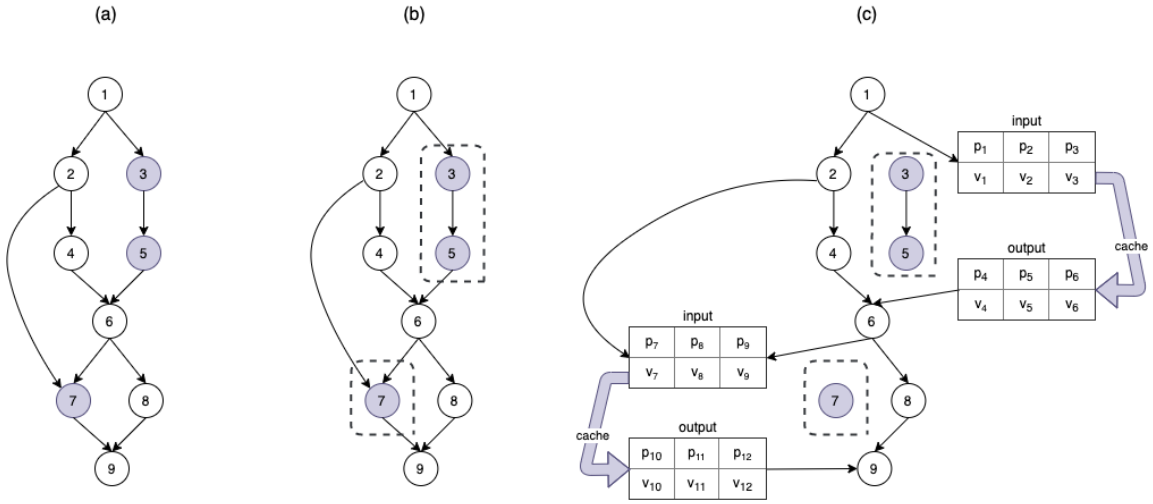


Figura 1: Exploring caching using the memoization technique during workflow execution

Adaptive Search Procedure) metaheuristic [RR03, FR18], named `MemoirGRASP`, to produce the optimal (or near-optimal) workflow activation scheduling plan while exploring memoization whenever possible. The main contributions of this manuscript are:

- The problem formulation of activation scheduling with memoization as a mixed integer programming problem;
- The design of the `MemoirGRASP` strategy for scheduling activations and data in a multisite cloud while benefiting from cached data;
- A thorough experimental evaluation using synthetic workflows and two real-world workflows, *i.e.*, Montage [S+09], for astronomical mosaic creation, and Phenomenal [ACC+19], for plant phenotyping, demonstrating the advantages and benefits of `MemoirGRASP`.

The rest of the manuscript is organized as follows. Section 2 discusses related work and provides a brief introduction to workflow memoization. In Section 3, we describe the mathematical formulation of the problem for workflow activation scheduling with memoization. In Section 4, we introduce `MemoirGRASP`, the execution strategy for solving the problem of workflow activation scheduling with memoization. Section 5 describes the experimental evaluation. Section 6 concludes.

2 Background and Related Work

In this section, we briefly discuss the concept of workflow memoization and review related work, highlighting similarities and differences with `MemoirGRASP`.

2.1 A Brief Introduction to Workflow Memoization

In many scientific domains, the execution of experiments aimed at confirming or refuting a hypothesis often involves several executions of identical or structurally similar activations or workflow fragments. As aforementioned, many of these workflows are both data- and compute-intensive, thus requiring several hours to complete execution even in distributed environments. To spare computational resources, reduce the makespan, and, in certain cases, minimize financial costs associated with execution, the concept of *memoization* has gained attention. This technique is designed as a strategy to improve efficiency by avoiding unnecessary recomputation of previously executed workflow fragments.

Although the concept of memoization is not new [Mic68], its use in workflows has emerged over the last decade and refers to the caching and subsequent reuse of intermediate results produced by workflow activations

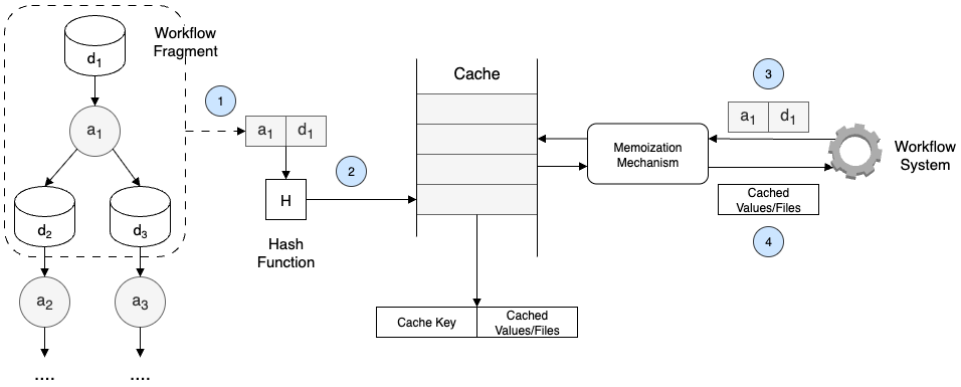


Figura 2: The workflow memoization process.

(Figure 2). Consider an activation a that executes a function $f_a : \mathcal{I}_a \rightarrow \mathcal{O}_a$, where $\mathcal{I}_a$ represents the set of input data consumed by the activation, and $\mathcal{O}_a$ denotes the corresponding output data it produces (step 1 in Figure 2). The idea behind memoization is that if the activation a is invoked again with the same input set $\mathcal{I}_a$ as in a prior execution of the workflow, the previously computed output $\mathcal{O}_a$ can be retrieved directly from a cache, thereby eliminating the need for redundant computation. The use of memoization is particularly valuable in workflows with: (i) Deterministic activations that produce identical outputs when provided with the same inputs and parameters; (ii) Compute- and data-intensive activations; and (iii) Frequent reuse of shared workflow fragments across executions.

To enable data caching and, consequently, memoization mechanisms, each activation execution is associated with a unique identifier, *i.e.*, a cache key, generated by a hash function $H(\mathcal{I}_a, a)$ (step 2 in Figure 2) that takes into account both the input data and the type of activation. This cache key serves as a reference to identify cached results in a persistent cache, enabling their reuse. In the context of this manuscript, a memoization mechanism can be defined as $\mathcal{H} \rightarrow \bigcup \mathcal{O}_a, where \mathcal{H} \subseteq \{H(\mathcal{I}_a, a)\}$ enabling a lookup operation $Cache(H(\mathcal{I}_a, a))$ (step 3 in Figure 2) that returns $\mathcal{O}_a$ or a cache miss (step 4 in Figure 2). Overall, workflow memoization plays an important role in scalable data reuse, reducing redundant computations.

2.2 Related Work

Over the last decade, a plethora of scheduling strategies were proposed as previously surveyed by Liu *et al.* [LPVM15]. Many of these solutions focus on efficiently executing workflows in the cloud, but they do not explore memoization techniques. Although some approaches focus on reusing cached data to optimize execution, there is no definitive solution. This way, the problem of efficient workflow scheduling with memoization remains an open, yet important, problem. In this section, we present the results of a simplified systematic mapping study, conducted following the guidelines proposed by Petersen *et al.* [PVK15]. Systematic mappings provide a high-level overview of a specific research domain. In this manuscript, the domain of interest is workflow scheduling with memoization (*i.e.*, caching). Table 1 lists each selected paper along with its citation, the venue in which it was published (journal or conference), and the year of publication. In the following paragraphs, we discuss these papers.

Well-known workflow systems such as Kepler, OpenAlea, and VisTrails [CFS+06] provide caching mechanisms. VisTrails implements memoization by storing the intermediate data produced by workflow activities in an internal directory of the user's machine. However, this type of "local" mechanism does not support large-scale data-intensive workflows such as Phenomenal or SciPhy [OdOO+11], which produce large amounts of data that do not fit the storage capacity of personal computers or even small clusters.

Differently from VisTrails, Kepler [CAWL14] exploits cloud-based storage to cache intermediate data. When a user starts the execution of a workflow, the engine checks whether there is data to be reused and then accesses the remote storage. Kepler reduces expensive data transfers, using Ant Colony Optimization technique to decide if data will be reused or not. However, this approach requires much time to converge to a local optimal plan with cached data (a known drawback of Ant Colony optimization) and is restricted to the storage of metadata only, not considering raw data files. Another extension to Kepler [OPP+17] encapsulates

Tabela 1: Published works on "*Workflow Scheduling with memoization*", their publishing vehicle, where C stands for conference and J for Journal.

Publication	Venue	Type	Year
Guo and Engler [GE10]	Workshop on the Theory and Practice of Provenance,	C	2010
Guo et al. [GE11]	International Symposium on Software Testing and Analysis	C	2011
Zohrevandi and Bazzi [ZB13]	IEEE International Symposium on Parallel and Distributed Processing	C	2013
Yuan *et al.* [YYL+13]	IEEE Transactions on Parallel and Distributed Systems	J	2013
Chen *et al.* [CAWL14]	IEEE World Congress on Services	C	2014
Tanaka et al. [TT14]	IEEE International Conference on Cluster Computing (CLUSTER)	C	2014
Moreno and Balch[MB16]	Concurrency and Computation: Practice and Experience	J	2016
Owsiak *et al.* [OPP+17]	Journal of Computational Science	J	2017
Casas *et al.* [CTR+17]	Future Generation Computer Systems	J	2017
Kunjir et al. [KFMB17]	ACM SIGMOD International Conference on Management of Data	C	2017
Qasha et al. [QWCW19]	Future Generation Computer Systems	J	2019
Heidsieck et al. [HdOP+21]	Future Generation Computer Systems	J	2021
Johnston et al. [JV22]	Workshop on Advanced tools, programming languages, and PLatforms for Implementing and Evaluating algorithms for Distributed systems	C	2022
Vassiliadis et al. [VJM22]	IEEE International Conference on Services Computing (SCC)	C	2022
Stojkovic et al. [SXFT23]	IEEE Symposium on High-Performance Computer Architecture	C	2023
Gaignard et al. [GSMB20]	Semantic Web: Interoperability, Usability, Applicability	J	2023

the workflow engine using Docker containers. Thus, the user can easily deploy the whole workflow environment (*i.e.*, the engine, programs, and data) in the cloud and exploit the cached data. Different from VisTrails and Kepler, OpenAlea [PFVB15] provides native caching mechanisms in both memory and disk. The memory caching mechanism is similar to that of Apache Spark, where only small portions of data can be cached. For larger datasets, cached data must be flushed to disk.

Vassiliadis *et al.* [VJM22] propose a taxonomy for characterizing workflow memoization strategies and introduce an algorithm that traverses graphs representing workflow specifications to define cache-keys. These keys determine what data should be cached and associate it with the corresponding workflow activity. The authors evaluated their approach using three real-world quantum chemistry workflows deployed across two geo-distributed OpenShift clusters. Although this approach marks a step forward, it does not perform workflow scheduling and is therefore complementary to `MemoirGRASP`.

Johnston *et al.* [JV22] argue that workflow systems should support approximate execution, which refers to when a specific activity of the workflow can be executed in multiple alternative ways, each with different characteristics in terms of performance, quality, and computational cost. Rather than following a fixed scheduling plan, the system should be capable of selecting the scheduling that provides the best trade-off, provided that the chosen schedule satisfies the constraints specified by the user. This approximation-aware capability can be enhanced through the use of cached data. When available, cached data that meets the constraints can be reused, thereby avoiding the need to re-execute certain activities.

Guo *et al.* [GE11] propose a system that memoizes the results of long-running pure function calls in scripts used to execute data science workflows. The system caches these results to disk and tracks dependencies between code and data. When a function is invoked again with the same inputs and no changes to its dependencies, the system reuses the cached result instead of re-executing the function. While this approach represents a step forward in reducing redundant computations, it does not address the problem of efficient workflow scheduling. As such, it is complementary to the approach presented in this manuscript, which focuses on optimizing the execution of memoized workflows.

Stojkovic *et al.* [SXFT23] propose an approach called Speculative Function-as-a-Service (SpecFaaS) to accelerate serverless workflows through speculative execution. In this model, function dependencies within a workflow are predicted, and data dependencies are speculatively satisfied using memoization, *i.e.*, by leveraging previously cached results. This strategy enables downstream functions to execute in parallel with upstream ones, overlapping computation and reducing the overall workflow execution time. Heidsieck *et al.* [HdOP+21]

propose a solution for cache-aware scheduling of scientific workflows in multisite cloud environments. Their approach includes a distributed, parallel architecture and novel algorithms for adaptive caching and cache site selection. The proposed system is implemented within the OpenAlea workflow system, integrating these cache-aware capabilities. In this manuscript, we reuse the mechanism that decides when to cache data in `MemoirGRASP`.

Qasha *et al.* [QWCW19] propose a framework that integrates version control, containerization, and automated deployment techniques for workflows. The framework also provides a set of optimization mechanisms that reduce the makespan by identifying tasks and data, combined with a caching system, which together enhance the reusability of workflows. Kunjir *et al.* [KFMB17] propose cache allocation strategies that aim to balance overall workload performance, *e.g.*, workflows, with fairness among tenants, taking into account the specific characteristics of each workload. Their approach enables transparent data sharing across multiple workloads in multi-tenant environments. Tanaka *et al.* [TT14] propose a scheduling strategy that optimizes the read performance of input files from cache, such as buffer or page caches in main memory. Their approach overlaps computation with I/O operations by leveraging cached data. The strategy is implemented in the Pwrake system, integrated with the Gfarm distributed file system.

Gaignard *et al.* [GSMB20] argue that intermediate data produced by scientific workflows should be treated as first-class entities. This approach not only reduces computational overhead by preventing redundant re-executions but also promotes data reuse by original researchers and peers for new hypotheses and experiments. The authors leverage provenance metadata automatically captured by the workflow system and domain-specific annotations from registries such as Bio.Tools. Yuan *et al.* [YYL+13] propose an approach that estimates the trade-off between the cost of reusing data and executing the workflow activations. They consider the specification of the workflow and the provenance data to check if certain data are consumed by many activities of the workflow, so to decide if a certain data product should be cached or not. However, the approach does not consider the impact of cached data on workflow scheduling, *i.e.*, the scheduling plan may be impacted by costly data transfers between activities that are executed in geo-distributed cloud sites.

Casas *et al.* [CTR+17] propose a scheduling algorithm, Balanced and file Reuse–Replication Scheduling (BaRRS), which fragments the workflow and identifies datasets that could be reused by each workflow fragment in future executions. Zohrevandi and Bazzi [ZB13] propose an approach to determine which intermediate data produced by a workflow activation should be stored in the cache for reuse by a different workflow. However, the approach considers only storage costs, without accounting for the activation schedule.

Guo and Engler [GE10] propose the IncPy approach for incremental recomputation of functions in Python scripts. IncPy stores the results of compute- and data-intensive functions in a local file that can be accessed to avoid recomputation. Although workflows can be modeled as scripts rather than using a workflow system, this approach only works for Python scripts and for workflows executed sequentially on a single computer. Memoizer [MB16] is a Python library designed to foster subproblem memoization. Memoizer is implemented using a decorator factory class, which allows users to annotate which functions they want to cache their output. One advantage of Memoizer is that it aims to identify pure functions in the code. If the function contains a random mechanism, Memoizer discards the cached data. However, it only works for Python scripts.

The main drawbacks of aforementioned approaches to workflow memoization and caching can be summarized as follows. First, several approaches rely on local caching mechanisms, which limit their applicability to large-scale, data-intensive workflows whose intermediate data cannot be stored in workstations or even in small clusters. Second, some approaches focus on caching metadata rather than raw data files, thereby reducing their effectiveness in scenarios where data movement and storage costs dominate execution time. Third, some caching strategies introduce an overhead due to long convergence times. Fourth, many approaches address memoization in isolation and do not integrate caching decisions with workflow scheduling, thereby ignoring the impact of cached data on task placement, execution order, and data transfers, particularly in geographically distributed settings. Finally, some methods consider storage costs or data reuse frequency but do not account for data locality and transfer costs, which can affect the workflow's makespan.

3 Preliminaries

In this manuscript, we strive to exploit the potential of multisite cloud and address the problem of Activation Scheduling with Memoization. In this section, we discuss the system model in Subsection 3.1 and the mathematical formulation of the problem defined in the Introduction as a mixed integer programming problem in

Subsection 3.2. This mathematical formulation will be the basis for producing optimal scheduling results to be compared with `MemoirGRASP`.

3.1 System Model

Our system model encompasses the formalization of workflows and the target cloud architecture. The problem we address is for a specific class of parallel workflows represented by a DAG (Directed Acyclic Graph) defined as $G = (V, A, a, \omega)$. Most related approaches represent data files as arcs (directed edges) in the DAG, *i.e.*, data dependencies. However, following the clean notation of Teylo *et al.* [TdPJF$^+$17], we represent data files as vertices, similar to workflow activations. Thus, the vertex set $V = N \cup D$ consists of activations $i \in N$ and data files $d \in D$; A stands for the set of arcs, which gives the precedence relation between activations and data files, a_i is the amount of work associated with an activation $i \in N$, and ω_{id} represents the cost associated with arc $(i, d) \in A$. The set of immediate predecessor activations of an activation $i \in N$ is defined as $pred(i) = \{j \in N \mid \exists d \in D, \text{ such that} (j, d) \in A \wedge (d, i) \in A\}$. Similarly, the set of immediate successors is given by $succ(i) = \{j \in N \mid \exists d \in D, \text{ such that} (i, d) \in A \wedge (d, j) \in A\}$. In G, an activation is always preceded and followed by a data file as illustrated in Figure 3, where $Activation_1$ reads data files $Data_1$ and $Data_2$ as needed for its execution and writes $Data_3$, which will be read later by $Activation_2$, which also writes the data file $Data_4$.

Figura 3: Activations and Data

The target architecture is multisite clouds. Let us define M as the set of all virtual machines (VMs) available for the execution of activations, storage, and caching data. It is important to highlight that VMs defined to cache intermediate data are not used for processing the activations. Each VM $j \in M$ has a storage capacity cm_j and computational slowdown index cs_j, which is inversely proportional to the computational power cp_j of the VM j [SBDP15]. Also, a VM j is deployed in a specific cloud site. The communication between two VMs must be considered as well. Let us define a communication delay index cd_l, which estimates the latency cost associated with each link l of the system. We use two types of communication delay in this manuscript, incurred for writing data (in the local disk or cache), cdw_l, and reading data (from local disk or from cache), cdr_l. Since we have to calculate the delay between two VMs independently if they are deployed in the same site or not, two communication delay matrices are built, each for a type of delay. By representing communication this way, we can model VMs deployed in multiple cloud sites since we can set different communication costs. Therefore, the execution time of an activation $i \in N$ on a VM $j \in M$ is given by $t_{ij} = a_i \times cs_j$. The communication time for an activation $i \in N$ executing on VM $j \in M$, to write data $d \in D$ in VM $p \in M$, where j and p are connected by link l, is given by $\overrightarrow{t}_{djp} = \omega_{id} \times cdw_l$. Similarly, the communication time for reading is given by $\overleftarrow{t}_{djp} = \omega_{di} \times cdr_l$. Figure 4 illustrates an example of the architectural model containing four VMs, two cloud sites (each with 2 VMs), and their characteristics where three of them are processing ones and one VM is dedicated to caching intermediate data.

3.2 Mathematical Formulation

The problem of Activation Scheduling with Memoization in multisite cloud can be defined as a mixed integer programming formulation. Let us define $M = M^{vm} \cup M^{st}$ as the set of all devices available for execution or storage, where M^{vm} is the set of VMs with processing power and local storage, and M^{st} is the set of storage devices (for storing intermediate data). Each device $j \in M$ has a storage limit of cm_j and a financial cost c_j^M (per quantum of time). The set of activations N includes (N_o), *i.e.*, activations that must be executed since there is no intermediate cached data to reuse, and (N_{no}), *i.e.*, activations whose execution can be avoided by reusing cached data. Let us also define $D = D_s \cup D_d$ 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.*, it is generated during workflow execution. For each $i \in N$, we consider a set of input data $\Delta_{in}(i) \subseteq D$ needed for

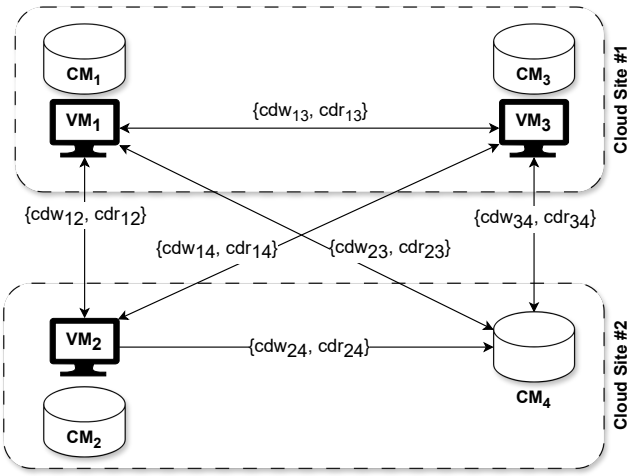


Figura 4: Architecture Model

their execution and a set of output data $\Delta_{out}(i) \subseteq D$ generated by activation i. An activation i is labeled as mandatory execution ($i \in N_o$) if and only if there is $d \in \Delta_{out}(i)$ such $d \in D_d$, *i.e.*, the activation produces data that is not available in the cache.

We define the user's requirements using c_{max} as the maximum estimated financial cost and t_{max} as the maximum execution time for the workflow, where $T = \{1, \ldots, t_{max}\}$ is denoted as the set of feasible periods of time. Furthermore, we define t_{ij} as the processing time of activation $i \in N$ on machine $j \in M$. Similarly, we define $\overleftarrow{t}_{djp}$ as the time that machine $j \in M$ spends reading the data $d \in D$ stored in a device $p \in M$, and $\overrightarrow{t}_{djp}$ as the time that machine $j \in M$ spends writing the data $d \in D$ on device $p \in M$ (assuming that a VM can write and read from a local disk or the cache). Table 2 summarizes the data used in the problem formulation.

Table 3 describes the binary and continuous decision variables of the problem formulation. The objective function (3.2) minimizes the workflow makespan and financial cost. The weights α_t and α_b define the relevance of the time and financial objectives, respectively. This allows for a fine-tuning of the proposed model, *i.e.,* users can choose if they want a "fast" execution or a "low-budget" execution. Note that both objectives are normalized using t_{max} and c_{max}, respectively.

min $\alpha_t \cdot (\frac{z^T}{t_{max}}) + \alpha_b \cdot (\frac{\sum_{j \in M_{vm}} c_j^M z_j^T}{c_{max}})$ Constraints (3.2) guarantee that every mandatory activation must be executed at least once (the same activation can be executed more than once if the workflow system has a backup/replicated activation for fault-tolerance [NNS21]). Constraints (3.2) and (3.2) rule that every read and write operation must be accomplished, for each (backup/replicated) activation.

$\sum_{j \in M_{vm}} \sum_{t \in T} x_{ijt} \geq 1, \forall i \in N_o$

$\sum_{j \in M_{vm} p \in M} \sum_{t \in T} \overleftarrow{x}_{idjpt} - \sum_{j \in M_{vm}} \sum_{t \in T} x_{ijt} = 0, \forall i \in N, \forall d \in \Delta_{in}(i)$

$\sum_{j \in M_{vm} p \in M} \sum_{t \in T} \overrightarrow{x}_{idjpt} - \sum_{j \in M_{vm}} \sum_{t \in T} x_{ijt} = 0, \forall i \in N, \forall d \in \Delta_{out}(i)$

Constraints (3.2), (3.2) and (3.2) rule that a VM can not repeat an action (*e.g.* execute the same activation or read the same data).

$\sum_{t \in T} x_{ijt} \leq 1, \forall i \in N, \forall j \in M_{vm}$

$\sum_{p \in M} \sum_{t \in T} \overleftarrow{x}_{idjpt} \leq 1, \forall i \in N, \forall d \in \Delta_{in}(i), \forall j \in M_{vm}$

$\sum_{p \in M} \sum_{t \in T} \overrightarrow{x}_{idjpt} \leq 1, \forall i \in N, \forall d \in \Delta_{out}(i), \forall j \in M_{vm}$

Inequalities (3.2) guarantee that data $d \in \Delta_{out}(i)$ can only be written, if an activation i was executed at the correct time. Furthermore, constraints (3.2) rule that data d can not be written before the processing time of the activation i (responsible for its writing). Note that both sets of constraints ((3.2) and (3.2)) work together to guarantee a feasible time for the writing process.

$\overrightarrow{x}_{idjpt} \leq \sum_{q=1}^{t-t_{ij}} x_{ijq}, \forall i \in N, \forall d \in \Delta_{out}(i), \forall j \in V_{vm}, \forall p \in M,$

$\forall t = (t_{ij} + 1) \cdots T_M$

$\overrightarrow{x}_{idjpt} = 0 \forall i \in N, \forall d \in \Delta_{out}(i), \forall j \in V_{vm}, \forall p \in M,$

Tabela 2: List of parameters

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 device of static data $d \in D_s$.
$W(d)$	Size of data $d \in D$.
N_o	Set of mandatory activations (*i.e.* activations that must be executed).
N_{no}	Set of non-mandatory activations (*i.e.* activations that could be executed).
$N = N_o \cup N_{no}$	Set of workflow activations
M_{vm}	Set of VMs (with processing power and storage service).
M_{st}	Set of storage devices (cache - with no processing power, only storage service).
$M = M_{vm} \cup M_{st}$	Set of machines (VMs and storage devices)
$\Delta_{in}(i) \subseteq D$	Set of data needed for the execution of an activation $i \in N$.
$\Delta_{out}(i) \subseteq D$	Set of data generated by an activation $i \in N$.
t_{max}	Maximum execution time for the workflow.
T	Set of feasible periods of time ($T = \{1 \dots t_{max}\}$).
t_{ij}	Processing time of an activation $i \in N$ in VM $j \in M$.
$\overleftarrow{t}_{djp}$	Time spent by VM $j \in M$ to read the data $d \in D$ stored in a device $p \in M$
$\overrightarrow{t}_{djp}$	Time spent by VM $j \in M$ to write the data $d \in \Delta_{out}(i)$ (such $i \in N$) stored in a device $p \in M$.
cm_j	Storage capacity of the device $j \in M$.
c_j^M	The financial cost of hiring a device $j \in M$ for one period of time.
c_{max}	The maximum financial cost requirement.
α_t and α_b	Time and budget weights that define the relevance of each objective ($\alpha_t + \alpha_b = 1$).

Tabela 3: List of variables of the problem formulation

Variables	Description
x_{ijt}	Binary variable which indicates whether activation $i \in N$ begins its execution in VM $j \in M_{vm}$ at period $t \in T$ or not.
$\overleftarrow{x}_{idjpt}$	Binary variable which indicates whether activation $i \in N$ executing in VM $j \in M_{vm}$ begins to read data $d \in \Delta_{in}(i)$ stored in device $p \in M$ at period $t \in T$ or not.
$\overrightarrow{x}_{idjpt}$	Binary variable which indicates whether activation $i \in N$ executing in VM $j \in M_{vm}$ begins to write data $d \in \Delta_{out}(i)$ stored in device $p \in M$ at period $t \in T$ or not.
y_{djt}	Binary variable which indicates whether data $d \in D$ is stored in device $j \in M$ at period $t \in T$ or not.
$\overline{y}_{dt}$	Binary variable which indicates whether data $d \in D$ is available in some machine (anyone) at the period of time $t \in T$ or not.
v_{jt}	Binary variable which indicates whether VM $j \in M_{vm}$ is used at time $t \in T$
z_j^T	Continuous variable which indicates the total time VM $j \in M_{vm}$ is used.
z^T	Continuous variable which indicates the total time to execute the workflow (makespan).

$1 \leq t \leq t_{ij}$

Constraints (3.2) rule that an activation can only be executed if all necessary reads were concluded in a feasible time.

$\mathrm{x}_{ijt} \leq \sum_{p \in M} \sum_{q=1}^{t-\overleftarrow{t}_{djp}} \overleftarrow{x}_{idjpq}, \forall i \in N, \forall d \in \Delta_{in}(i), \forall j \in M_{vm},$
$\forall t \in T,$ such $(t - \overleftarrow{t}_{djp}) \geq 1$

Inequalities (3.2) guarantees that only one active action (execute an activation, read or write data) 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). Note that if any action is performed in the VM j in a period of time t, variable v_{jt} will be set to 1.

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

Constraints (3.2) establish that there are no dynamic data available at the beginning of the workflow execution. On the other hand, constraints (3.2) guarantee that all static data are already stored on their origin devices (local disks on VMs and cache devices).

$\mathrm{y}_{dj1} = 0, \forall d \in D_d, \forall j \in M$
$y_{djt} = 1, \forall d \in D_s \mid j \in O(d), \forall t \in T$

Constraints (3.2) and (3.2) link the storage variable y with the write variable $\overrightarrow{x}$ and the read variable $\overleftarrow{x}$, guaranteeing a feasible write and read process, respectively. Constraint (3.2) ensures that data will only be stored in a device if it was previously stored or produced. On the other hand, constraints (3.2) ensure that data will only be read if previously stored in a device.

$\mathrm{y}_{dp(t+1)} \leq y_{dpt} +$
$\sum_{j \in M_{vm}} \overrightarrow{x}_{idjp(t-\overrightarrow{t}_{djp}+1)}, \forall d \in D, \forall p \in M, \forall t \in \{1 \ldots t_{max} - 1\},$
such $(t - \overrightarrow{t}_{djp} + 1) \geq 1$ and $d \in \Delta_{out}(i)$
$\sum_{j \in M_{vm}} \overleftarrow{x}_{idjpt} \leq |M|.y_{dpt}, \forall i \in N, \forall d \in \Delta_{in}(i), \forall p \in M, \forall t \in T$

The device's storage capacity is bounded by constraints (3.2). Constraints (3.2) relate the last write operation with the workflow total execution time (*i.e.*, makespan). Note that our application model assumes that an activation always writes (*i.e.*, creates) data. Constraints (3.2) link variables v_{jt} and z_j^T to establish

the correct allocation (paid) time of a VM j (*i.e.*, the last period that the VM j is used), while constraints (3.2) ensures that the financial costs do not exceed the maximum budget set by users.

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

$\overrightarrow{x}_{idjpt} \cdot (t + \overrightarrow{t}_{djp}) \leq z^T, \forall i \in N, \forall d \in \Delta_{out}(i),$

$\forall j \in M_{vm}, \forall p \in M, \forall t \in T$

$v_{jt} \cdot t \leq z_j^T, \forall j \in M_{vm}, \forall t \in T$

$\sum_{j \in M_{vm}} c_j^M z_j^T \leq c_{max}$

Constraints (3.2) link the storage variable y with the machine-independent storage variable $\overline{y}$. The following operational constraint (3.2) must be satisfied: an activation i can only begin any reading process if all data $d \in \Delta_{in}(i)$ is already available.

$\overline{y}_{dt} \leq \sum_{j \in M} y_{djt}, \forall d \in D, \forall t \in T$

$\sum_{p \in M} \overleftarrow{x}_{idjpt} \cdot |\Delta_{in}(i)| \leq \sum_{g \in \Delta_{in}(i)} \overline{y}_{gt}, \forall i \in N, \forall d \in \Delta_{in}(i),$

$\forall j \in M_{vm}, \forall t \in T$

4 A GRASP-based Execution Strategy for Workflow Scheduling with Memoization

This section introduces the proposed scheduling strategy for cloud-based workflows, employing the memoization technique named `MemoirGRASP`. We begin by explaining the pre-processing of the workflow, followed by an exposition of the scheduling algorithm.

4.1 Workflow Pre-processing

Prior to initiating the workflow scheduling algorithm, `MemoirGRASP` conducts a preprocessing step on the workflow specification to discover the available cached data for each workflow fragment (*i.e.*, a set of activations). In this manuscript, we use the algorithm proposed by Heidsieck *et al.* [HdOP+20] that defines the appropriate data to be cached by estimating the ratio between recomputation and storage costs. Furthermore, it takes into consideration the probability of a specific piece of data being reused.

When data are cached in a previous workflow execution, Heidsieck *et al.*'s method involves updating a catalog containing a cache key of all cached data and the corresponding machine where this data is stored. Consequently, `MemoirGRASP` performs a search in each potential workflow activation to determine if cached data exists in the catalog, as described in Algorithm 4.1. If the cached data is available, the preprocessing algorithm sets N_o and N_{no}. Activations that do not have cached data are marked as mandatory (N_o). On the other hand, those with cached data available are defined as non-mandatory (N_{no}), meaning they can be executed or not by the scheduling algorithm presented in the following Subsection.

texttt() The set of workflow activations N The sets N_o and N_{no} $N_o \leftarrow \emptyset$

$N_{no} \leftarrow \emptyset$

$i \in N$ $isCached \leftarrow$ is_Cached(i,$\Delta_{in}(i)$)

isCached $N_{no} \leftarrow N_{no} \cup i$ $N_o \leftarrow N_o \cup i$

$\{N_o, N_{no}\}$ Workflow Pre-processing

4.2 Scheduling Algorithm

The classic scheduling problem is classified as NP-Hard [EH02, LSV23]. Since our workflow execution strategy based on memoization is a variation of classic scheduling, it can be also classified as NP-Hard problem. For this class of problems, exact methods are often incapable of finding solutions in a reasonable time. A good alternative is to use metaheuristics [Tal09], *i.e.* methods that coordinate interaction between local improvement procedures and higher-level approaches to escape from local minima and perform a robust search in a solution space.

Greedy Randomized Adaptive Search Procedures (GRASP) is a simple and iterative metaheuristic that has been successfully applied to a large number of combinatorial optimization problems [FR09a, FR09b]. Each GRASP iteration is divided into two phases. First, a feasible solution (*i.e.*, a workflow schedule in the context of this manuscript) is built in a construction phase. Then, in the second stage, its neighborhood (*i.e.*,

variations in the workflow schedule) is explored by a local search procedure to find a better solution. The best solution found in all iterations is taken as a result (*i.e.*, final workflow schedule).

In this manuscript, we propose a GRASP-based heuristic, as part of the `MemoirGRASP` strategy, which employs a multi-start approach to explore larger areas of the search space. Our motivation is that such a multi-start approach is likely to find solutions whenever the exact method 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.

To present the proposed algorithm, we introduce some additional notation. Let $N_{pred}(i) \subseteq N$ denote all the immediate predecessors of activation $i \in N$, *i.e.*, activations that generate input data for i. We denote by $P(N_{pred}(i))$ the family of all subsets of $N_{pred}(i)$. Note that an element $N^{rx} \in P(N_{pred}(i))$ represents a set of activations that can optionally be (re)executed to generate data for activation i, *i.e.*, activations where input data is available in a device. Then we define a workflow (partial) scheduling $S = \{(i_1, j_1, N_1^{rx}, p_1), (i_2, j_2, N_2^{rx}, p_2), ...\}$ as the ordered set of 4-tuples (i, j, N^{rx}, p) representing that activations $i \in N_o$ and $N^{rx} \in P(N_{pred}(i))$ are executed in VM j and data $\Delta_{out}(i)$ is written in VM p. A scheduling is considered feasible if and only if all mandatory activations $i \in N_o$ are processed and the scheduling makespan and financial cost are limited by t_{max} (time deadline) and c_{max} (maximum financial cost), respectively. Similarly, a tuple (i, j, N^{rx}, p) is considered feasible for partial scheduling S if $S \cup (i, j, N^{rx}, p)$ does not exceed the time and financial cost limit.

The first phase of the proposed method is the construction phase, where a feasible scheduling is iteratively constructed, one element (tuple) at a time. At each construction iteration, the choice of the next tuple to be added is determined by ordering all candidate tuples in a candidate list (CL) concerning a cost function. Then, a restricted candidate list is considered, composed of the best candidates, using a threshold parameter $\beta \in [0, 1]$ to limit the candidate list, *i.e.*, tuple t will be considered only if its cost $c(t) \in [c^{min}, c^{min} + \beta(c^{max} - c^{min})]$, where c^{min} and c^{max} are the minimum and maximum costs of the CL, respectively. Finally, one element from the restricted CL is chosen at random to be part of the scheduling.

The constructive heuristic is described in Algorithm 4.2. Initially, a loop that runs until all mandatory activations are scheduled is executed. In this loop, a list of candidates (CL) is built by combining each mandatory activation $i \in N_o$, each subset of activations to be re-executed $N^{rx} \in P(N_{pred}(i))$, each possible VM for processing $j \in M_{vm}$, and each possible VM to write the output data files $p \in M$. Since some mandatory activations may require other mandatory activations to be executed (due to data dependencies), the candidate activations whose input data are not available, will not be considered to the CL (line 5).

The cost for each movement (tuple) is computed based on its insertion on the current partial scheduling S (line 11). After every possible permutation is considered, the CL is ordered increasingly by cost (line 13), and a random candidate is selected (restricted by parameter β) to be joined with the current solution S (lines 14 and 15).

The probabilistic component of the method is characterized by randomly choosing one of the best tuples in the list CL, but not necessarily the best. This approach allows for different solutions to be reached on each execution of the constructive heuristic.

texttt() size of the CL β Scheduling S $S \leftarrow \emptyset$
$\overline{N}_o \leftarrow N_o$
$\overline{N}_o \neq \emptyset$ $CL \leftarrow \emptyset$
$i \in \overline{N}_o$ where $\Delta_{in}(i)$ is available $j \in M_{vm}$ $N^{rx} \in P(N_{pred}(i))$ $p \in M$ (i, j, N^{rx}, p) is feasible
$S' = S \cup (i, j, N^{rx}, p)$
$cost \leftarrow$ Move_Cost(S')
$CL \leftarrow CL \cup \{(i, j, N^{rx}, p), cost\}$
$CL \leftarrow$ Sort(CL)
$(i, j, N^{rx}, p) \leftarrow$ Get_Candidate(CL, β)
$S \leftarrow S \cup (i, j, N^{rx}, p)$
$\overline{N}_o \leftarrow \overline{N}_o - \{i\}$ S Constructive_Heuristic

The cost of the movement is calculated in Algorithm 4.2. This cost is a composition of both makespan and financial cost of VMs (as depicted in Equation 3.2). Vectors T^M and T^N are auxiliary structures that are used to keep track of the times computed at every step of the method. The vector T^M is indexed by the VM index $j \in M_{vm}$ and contains the finishing time of the last activation executed in the corresponding VM. Each element of vector T^N corresponds to an activation $i \in N_o$ and contains the finishing time of the associated

activation.

For every tuple t in the ordered set (scheduling) S, the method first schedules the activations that will be re-executed (lines 5 to 9). The start time (*start*) of $i_{rx} \in N_t^{rx}$ is calculated as the maximum completion time among all its immediate predecessors (represented by *pred*) and the finishing time of the last activation executed in VM j_t (stored in $T^M[j_t]$), as presented in line 7. Then, in line 8, the finishing time of i_{rx} (*end*) is calculated by adding the following values: (1) the start time of i_{rx} (*start*); (2) the execution time of i_{rx} when executed in VM j_t ($t_{i_{rx}j_t}$); (3) the time to read all the data files required by i_{rx} from VM j_t; and (4) the time to write all the data files generated by i_{rx} in VM j_t. In line 9 the vector T^M is updated with the new finishing time for the VM j_t. The same update steps are performed for activation i_t in lines 10 to 14, except that in this case vector T^N is also updated with the finishing time of activation i_t.

Finally, in lines 15 and 16, the makespan (*mks*) and financial cost ($fcst$) are computed using vector T^M, and the cost of the scheduling is calculated in line 17.

The scheduling cost can be computed in $\mathcal{O}(|S|N)$ operations by Algorithm 4.2. However, it is an easy job to perform the calculations in linear time $\mathcal{O}(N)$ by just updating vectors T^M and T^N whenever a new tuple is added to the partial scheduling. Thus, the overall complexity of the constructive heuristic defined by Algorithm 4.2 is $\mathcal{O}(N^2M^2 2^{Max_p})$, where $Max_p = \max_{i \in N_o}\{|N_{pred}(i)|\}$ is defined as the maximum number of immediate predecessors for a mandatory activation. To avoid the combinatorial explosion of re-executions, a bound of $Max_p \leq 5$ was empirically determined and used to limit the number of analyzed subsets.

texttt() S cost of the scheduling $T^M \leftarrow 0$
$T^N \leftarrow 0$
tuple $t = 1...|S|$ Let tuple $t = (i_t, j_t, N_t^{rx}, p_t)$
/ re-execution loop */*
$i_{rx} \in N_t^{rx}$ $pred \leftarrow \text{Max}_T\text{\_Pred}(T^N, N_{pred}(i_{rx}))$
$start \leftarrow \max(pred, T^M[j_t])$
$end \leftarrow start + t_{i_{rx}j_t} + \text{Read}_T\ (i_{rx}, j_t) + \text{Write}_T\ (i_{rx}, j_t)$
$T^M[j_t] \leftarrow end$ $pred \leftarrow \text{Max}_T\text{\_Pred}(T^N, N_{pred}(i_t))$
$start \leftarrow \max(pred, T^M[j_t])$
$end \leftarrow start + t_{i_t j_t} + \text{Read}_T\ (i_t, p_t) + \text{Write}_T\ (i_t, p_t)$
$T^M[j_t] \leftarrow end$
$T^N[i_t] \leftarrow end$
$mks \leftarrow \text{Makespam}(T^M)$
$fcst \leftarrow \text{Financial\_Cost}(T^M)$
$cost \leftarrow \alpha_t \cdot (\frac{mks}{t_{max}}) + \alpha_b \cdot (\frac{fcst}{c_{max}})$
$cost$ Move_Cost

Right after the constructive heuristic, the GRASP method implements a sequence of local searches to improve the current solution. This intensification stage is composed of five neighborhood components that are analyzed following the Variable Neighborhood Descent (VND) method [GP10], applying each neighborhood in a deterministic way. Whenever an improvement movement is found, the method forces a restart in the local search, and the whole VND process starts over from the very first neighborhood. The VND process ends when no improvement is provided by all neighborhoods.

All five neighborhood components are executed in the following order [TdPF+17]: (1) Move_Act: Given tuple $(i, j, N^{rx}, p) \in S$, move activation i or activation $i_{rx} \in N^{rx}$ to be executed in a different machine $j' \in M_{vm} \backslash \{j\}$. (2) Move_Data: Given tuple $(i, j, N^{rx}, p) \in S$, move data $d \in \Delta_{out}(i)$ to be written in a different machine $p' \in M_{vm} \backslash \{p\}$. (3) Swap_Act: Given a pair of tuples $(i_1, j_1, N_1^{rx}, p_1), (i_2, j_2, N_2^{rx}, p_2)$ in S, swap the execution machines, thus reaching tuples $(i_1, j_2, N_1^{rx}, p_1)$ and $(i_2, j_1, N_2^{rx}, p_2)$. (4) Swap_Data: Given a pair of tuples $(i_1, j_1, N_1^{rx}, p_1), (i_2, j_2, N_2^{rx}, p_2)$ in S, swap the machines for the output data, thus reaching tuples $(i_1, j_1, N_1^{rx}, p_2)$ and $(i_2, j_2, N_2^{rx}, p_1)$. (5) Swap_Tuple: Given a pair of tuples t_1, t_2 in the scheduling, swap positions of t_1 and t_2 in the ordered set S.

Algorithm 4.2 presents our proposed GRASP method. Each iteration is composed of the construction of an initial solution (line 3) followed by a local search improvement (line 4). The best overall solution is updated in lines 5 to 6. Then, the whole procedure is repeated until the stop criterion is reached (*i.e.*, total number of GRASP iterations).

number of iterations $MaxIT$, size of the RCL β Scheduling S^* $S^* \leftarrow \emptyset$
$i = 1$ to $MaxIT$ $S \leftarrow \text{Constructive\_Heuristic}(\beta)$

$S' \leftarrow \text{VND}(S)$
$\text{Move\_Cost}(S') < \text{Move\_Cost}(S^*) \quad S^* = S \qquad S^*$ GRASP

5 Experimental Evaluation

To evaluate our `MemoirGRASP` strategy, we conducted a series of experiments using using both synthetic workflows and the Phenomenal workflow from plant phenotyping. First, we introduce these workflows and the experimental setup. Then, we introduce the experiments and discuss the results. All the data used in the experiments will be available on our GitHub repository[2].

5.1 Synthetic Workflows

We used five synthetic workflows. Each workflow has a size that allows the exact method to be solved within a reasonable time frame, as executing the exact method[3] in larger workflows would be impractical. Each workflow has a number of activations (3, 4, 5, or 8) and incorporates various constructs, including splits, merges, maps, and independent branches, following common workflow patterns [vdA18, JCD$^+$13].

Table 4 gives the main characteristics of each workflow: (1) Synthetic_7 has three activations and a split, where activations t_0 and t_2 consume the same input dataset and can be executed concurrently, (2) Synthetic_11 has two splits and one merge, with activation t_0 consuming d_2, d_3, d_4, and d_6, (3) Synthetic_12 has 2 splits and no merges, (4) Synthetic_13 has two splits and one merge, and (5) Synthetic_22 has two independent branches that are later merged by activation t_6.

For each workflow, we created three scenarios (A, B, and C) by varying the slowdown indexes and the cost of each device (as shown in Table 5), yielding a total of 15 synthetic workflow instances. We name the synthetic workflow instances following the convention *Synthetic_[Number of Activations + Data Files]_[Scenario]*, as shown in Table 4.

5.2 Real-World Workflows

In this subsection, we present a discussion of two illustrative, real-world workflows that were used in our experimental evaluation. The first is the Phenomenal workflow [ACC$^+$19], which is designed for applications in the field of plant phenotyping and plays a key role in the automated processing and analysis of large-scale phenotypic data. The second is the Montage [S$^+$09] workflow, from the domain of astronomy, which is widely employed for creating astronomical image mosaics from multiple observational datasets.

5.2.1 Phenomenal Workflow

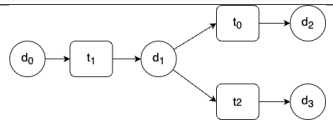
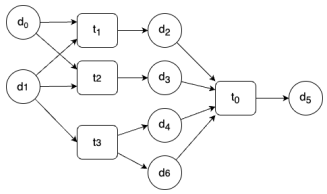
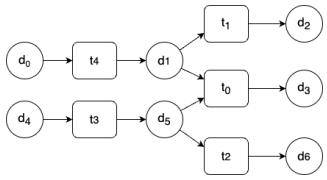
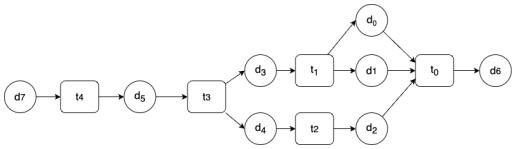
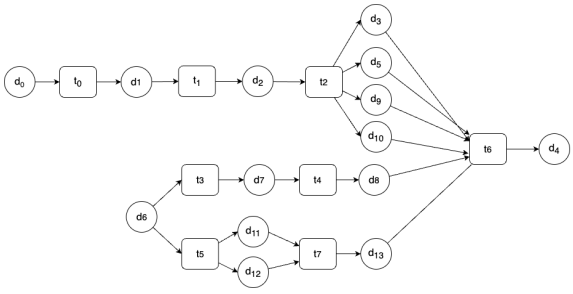
Over the past decade, high-throughput phenotyping platforms have fostered the acquisition of huge amounts of quantitative data from thousands of plants in controlled environments. These platforms enable users to gather heterogeneous data types, including images, sensor data, *etc.* For instance, the French Phenome project[4] generates approximately 200 TB of raw data each year. However, extracting meaningful insights from such big datasets is challenging [TCBPB17].

To analyze these datasets, a widely used solution is the Phenomenal workflow [APHC18] implemented in the OpenAlea workflow system [PFVB15]. Figure 5(a) gives a high-level overview of the Phenomenal workflow, where each circle represents an activity associated with a sub-workflow that may contain multiple steps (Figure 5(c)). The activities of Phenomenal are as follows: (1) Input Data Loading, (2) Binarization, (3) 3D Reconstruction, (4) Display Binaries, (5) Skeletonize, (6) Meshing, (7) Stem Detection, (8) Display Mesh, (9) Display Skeleton, (10) Output Data Storing, and (11) Organ Segmentation. Dark grey circles indicate I/O activities, while white circles represent display activities. In our experiments, we focus on data-intensive activities (Figure 5(b)) that could benefit from memoization techniques:

- The *Binarization* activity separates plant pixels from the background for each input image, creating a binary image from an RGB source;

[2] `https://github.com/UFFeScience/MemoirGRASP`
[3] the mathematical formulation in Section 3.2 was solved using IBM ILOG CPLEX version 22.1
[4] `https://www.phenome-emphasis.fr/`

Tabela 4: Specification of the Synthetic Workflows

Workflow ID	Workflow Structure	# of Activations	# of Data Files	Pattern
Synthetic_7		3	4	Map, Split
Synthetic_11		4	7	Map, Split, Merge
Synthetic_12		5	7	Map, Split, Merge
Synthetic_13		5	8	Map, Split, Merge
Synthetic_22		8	14	Independent Branches, Map, Split, Merge

- The *3D Reconstruction* activity generates a 3D volume using twelve sides and one top binary image of the plant;
- The *Skeletonize* activity calculates a skeleton within the reconstructed 3D volume;
- The *Meshing* activity creates a 3D mesh from the volume and decreases its size according to user-defined parameters;
- The *Stem Detection* activity identifies the main stem of cereal plants (*e.g.*, maize, wheat, sorghum) by calculating a primary path within the skeleton;
- The *Organ Segmentation* activity segments different organs on the skeleton after removing the main stem.

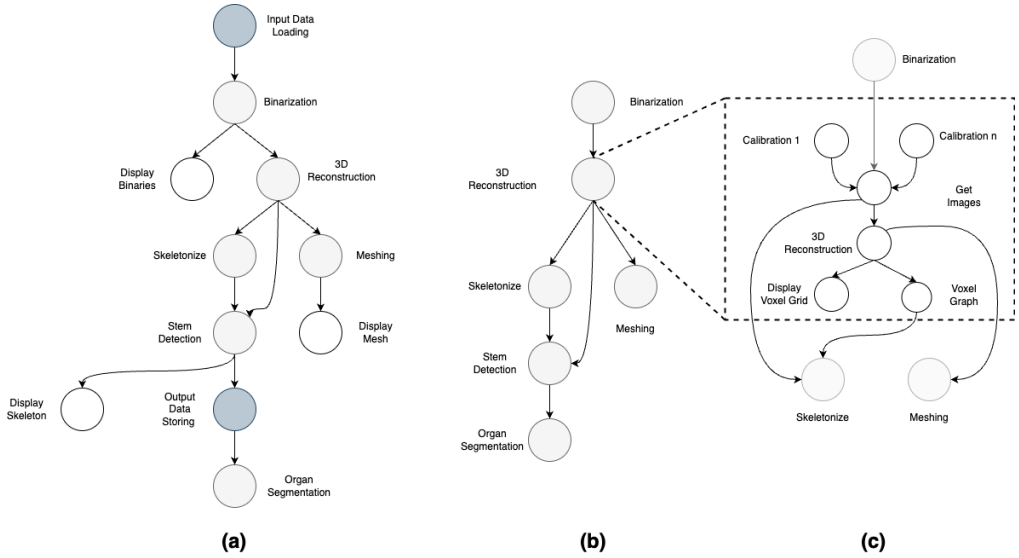


Figura 5: Phenomenal workflow (a) High-level specification (b) Data-intensive activities and (c) Steps executed within the 3D Reconstruction activity.

Executing Phenomenal in a single experiment can yield up to 11 TB of raw data, including images that may be several GB in size. For instance, the phenotyping facility in Montpellier[5] monitors more than 1,600 plants in a controlled environment (*e.g.*, temperature, humidity, irrigation) and automatically captures images over time. This facility encompasses approximately 80,000 time series of plants and over one million images.

5.2.2 Montage Workflow

The Montage workflow [S+09], presented in Figure 6, is a data-intensive workflow used in the field of astronomy. It creates astronomical image mosaics by composing multiple sky images. Montage achieves this by orchestrating a sequence of activations, each corresponding to an invocation of a specialized tool within the Montage *toolkit*[6]. The workflow is composed of the following seven main activities: (i) *mProject*: Projects input images onto a common coordinate system based on a specified scale; (ii) *mDiffFit*: Computes the differences between overlapping image pairs and adjusts them to minimize background discrepancies; (iii) *mConcatFit*: Concatenates multiple difference parameters into a single set for global adjustment; (iv) *mBgModel*: Models and corrects background variations across overlapping images to ensure seamless composition; (v) *mBackground*: Applies background corrections to individual images based on the computed model; (vi) *mImgtbl*: Extracts metadata from the image set for mosaic assembly; and (vii) *mAdd*: Performs the final co-addition of all processed images to create the output mosaic.

[5] `https://eng-lepse.montpellier.hub.inrae.fr/platforms-m3p/montpellier-plant-phenotyping-platforms-m3p/phenoarch`

[6] `http://montage.ipac.caltech.edu/`

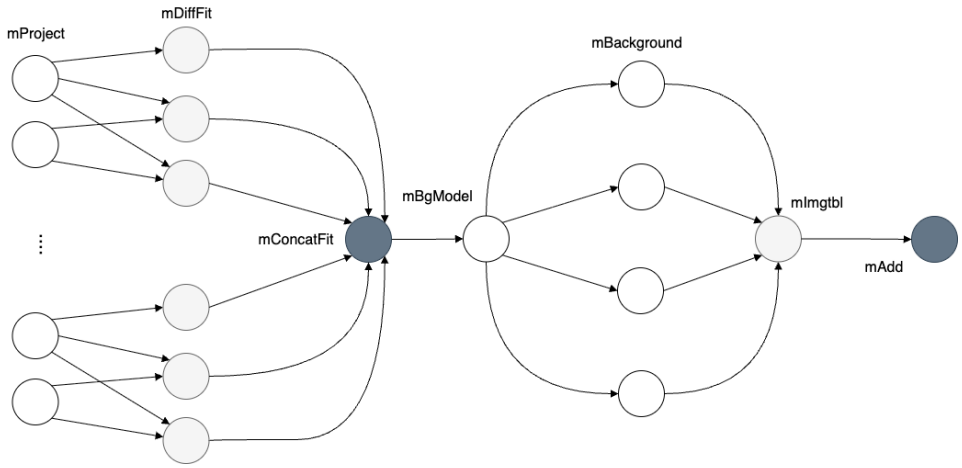


Figura 6: High-level specification of the Montage workflow.

The Montage workflow can be configured to process a user-defined region of the sky, with the target area expressed in square degrees. The total number of activations within the workflow depends directly on the size of this specified coverage area. In the experiments described in Subsection 5.5, we set the coverage area to 0.5 square degrees. For this configuration, the execution of Montage comprises a total of 54 activations, distributed as follows: 12 `mProject` activations, 18 `mDiffFit` activations, 3 `mConcatFit` activations, 3 `mBGModel` activations, 12 `mBackground` activations, 3 `mImgtbl` activations, and 3 `mAdd` activations.

5.3 Evaluation with the Synthetic Workflows through Simulation

We evaluated `MemoirGRASP` using the set of 15 workflow instances as outlined in Section 5.1. For each synthetic workflow (Synthetic_7, Synthetic_11, Synthetic_12, Synthetic_13, and Synthetic_22), three distinct instances were generated by varying the computing environments, including the number of VMs, their processing capacity, and bandwidth. The computational slowdown was set between 0.01 and 1. This evaluation explores seven scenarios: Scenario #1 with $\alpha_t = 1$ and $\alpha_b = 0$, emphasizing the reduction of makespan; Scenario #2 with $\alpha_t = 0$ and $\alpha_b = 1$, prioritizing the reduction of financial costs; and Scenario #3 with $\alpha_t = 0.5$ and $\alpha_b = 0.5$, seeking a balance between the objectives; Scenario #4 with $\alpha_t = 0.75$ and $\alpha_b = 0.25$, prioritizing reduction of makespan (but less than Scenario #1); Scenario #5 with $\alpha_t = 0.25$ and $\alpha_b = 0.75$, emphasizing the reduction of financial costs (but less than Scenario #2); Scenario #6 with $\alpha_t = 0.6$ and $\alpha_b = 0.4$, prioritizing makespan reduction (but less than Scenario #4); and Scenario #7 with $\alpha_t = 0.4$ and $\alpha_b = 0.6$, focusing on reducing financial costs (but less than Scenario #5).

Tables 6 through 12 summarize the results obtained with the exact approach and `MemoirGRASP`. In addition to comparing the results of `MemoirGRASP` with the exact solution, we include comparisons with our greedy scheduling approach [HdOP+19] and a multi-start scheduling algorithm, which uses the GRASP method without exploring the neighborhoods. In each table, the first column identifies each workflow instance. Then, we present the objective function (Function 3.2 in Section 3.2) value and the time required to solve the exact approach, the greedy algorithm, the multi-start algorithm, and `MemoirGRASP`. The values reported for `MemoirGRASP` represent averages from five executions.

From Tables 6 through 12, it can be observed that `MemoirGRASP` consistently produces solutions with small gaps, where the term "gap" refers to the difference between the objective function value of the obtained solution and that of the exact solution. Across all 105 executions evaluated (15 executions for each of the seven scenarios), `MemoirGRASP` achieved an objective function value greater than that of the exact solution in only 21 cases (20%), which are indicated in bold in the tables. In terms of computational efficiency, `MemoirGRASP` requires substantially less processing time to generate a schedule compared with the exact solution obtained using CPLEX. Specifically, the average computation times for the exact method are 46.47, 63.78, 28.32, 69.56, 61.58, 19.58, and 43.42 seconds for Scenarios #1 through #7, respectively. On the other hand, `MemoirGRASP` completes execution in less than 0.01 seconds for all scenarios. Furthermore, CPLEX is able to produce

a feasible solution within the one-hour time limit for all synthetic workflows tested. This is expected, as these workflows were intentionally designed with problem sizes that ensure the generation of both feasible and optimal solutions within the allotted time. However, for larger workflow instances, CPLEX is unable to guarantee the computation of the optimal schedule within the specified time limit, and its performance degrades accordingly.

Tabela 5: Slowdown and Financial Cost Specifications for the Synthetic Workflows

Synthetic Instances	cs_j	c_j^M
Synthetic_7_A	0.50, 0.40, 1.00	0.04, 0.05, 0.03
Synthetic_7_B	0.50, 0.40, 1.00	0.04, 0.05, 0.03
Synthetic_7_C	0.50, 0.40, 0.40	0.04, 0.05, 0.03
Synthetic_11_A	0.40, 0.53, 0.20, 0.20, 0.20	0.08, 0.01, 0.06, 0.09, 0.07
Synthetic_11_B	0.40, 0.53, 0.20, 0.20, 0.20	0.08, 0.01, 0.06, 0.09, 0.07
Synthetic_11_C	0.40, 0.53, 0.20, 0.20, 0.20	0.08, 0.01, 0.06, 0.09, 0.07
Synthetic_12_A	0.90, 0.40, 0.50	0.04, 0.03, 0.05
Synthetic_12_B	0.90, 0.40, 0.50	0.04, 0.03, 0.05
Synthetic_12_C	0.90, 0.40, 0.50	0.04, 0.03, 0.05
Synthetic_13_A	0.50, 0.40, 1.00	0.02, 0.01, 0.03
Synthetic_13_B	0.50, 0.40, 1.00	0.02, 0.01, 0.03
Synthetic_13_C	0.50, 0.40, 1.00	0.02, 0.01, 0.03
Synthetic_22_A	0.50, 0.40, 1.00	0.02, 0.01, 0.03
Synthetic_22_B	0.50, 0.40, 1.00	0.02, 0.01, 0.03
Synthetic_22_C	0.50, 0.40, 1.00	0.02, 0.01, 0.03

Tabela 6: Analysis of the Objective Function Value and Execution Time for Scenario #1 - $\alpha_t = 1$ and $\alpha_b = 0$

	Exact Solution		**Greedy**		**Multi-start**		`MemoirGRASP`	
Workflow	O.F	Time (s)	O.F	Time (s)	O.F	Time (s)	O.F	Time (s)
Synthetic_7_A	0.47	0.37	0.47	≤ 0.01	0.47	≤ 0.01	0.47	≤ 0.01
Synthetic_7_B	0.50	0.60	0.55	≤ 0.01	0.50	≤ 0.01	0.50	≤ 0.01
Synthetic_7_C	0.46	0.47	0.50	≤ 0.01	0.46	≤ 0.01	0.46	≤ 0.01
Synthetic_11_A	0.38	21.57	0.62	≤ 0.01	0.42	≤ 0.01	0.38	≤ 0.01
Synthetic_11_B	0.20	142.61	0.73	≤ 0.01	0.33	≤ 0.01	**0.22**	≤ 0.01
Synthetic_11_C	0.18	213.32	0.39	≤ 0.01	0.28	≤ 0.01	**0.19**	≤ 0.01
Synthetic_12_A	0.32	55.21	0.39	≤ 0.01	0.32	≤ 0.01	0.32	≤ 0.01
Synthetic_12_B	0.30	60.41	0.36	≤ 0.01	0.32	≤ 0.01	0.30	≤ 0.01
Synthetic_12_C	0.24	62.14	0.63	≤ 0.01	0.24	≤ 0.01	0.24	≤ 0.01
Synthetic_13_A	0.38	1.18	0.38	≤ 0.01	0.38	≤ 0.01	0.38	≤ 0.01
Synthetic_13_B	0.36	2.11	0.59	≤ 0.01	0.36	≤ 0.01	0.36	≤ 0.01
Synthetic_13_C	0.26	1.76	0.42	≤ 0.01	0.26	≤ 0.01	0.26	≤ 0.01
Synthetic_22_A	0.36	3.88	0.36	≤ 0.01	0.36	≤ 0.01	0.36	≤ 0.01
Synthetic_22_B	0.31	6.43	0.61	≤ 0.01	0.39	≤ 0.01	**0.39**	≤ 0.01
Synthetic_22_C	0.42	125.11	0.61	≤ 0.01	0.56	≤ 0.01	**0.56**	≤ 0.01

We evaluated the makespan for each workflow instance based on the schedules produced by four distinct approaches: (1) the exact solution, represented by the black bars; (2) the greedy algorithm, represented by the dark grey bars; (3) the multi-start approach, represented by the white bars; and (4) `MemoirGRASP`, represented by the light grey bars. The results are presented in Figure 7, which reports the makespan values for each workflow instance under the following parameter configurations: (a) $\alpha_t = 1$ and $\alpha_b = 0$; (b) $\alpha_t = 0$ and $\alpha_b = 1$; (c) $\alpha_t = 0.5$ and $\alpha_b = 0.5$; (d) $\alpha_t = 0.75$ and $\alpha_b = 0.25$; (e) $\alpha_t = 0.25$ and $\alpha_b = 0.75$; (f) $\alpha_t = 0.6$ and $\alpha_b = 0.4$; and (g) $\alpha_t = 0.4$ and $\alpha_b = 0.6$. The results show that `MemoirGRASP` consistently produces schedules with makespan values that are close to those of the optimal solution, both in scenarios where makespan is the main criterion (Scenarios #1, #4, and #6) and in scenarios where makespan is not the primary criterion (Scenarios #2, #3, #5, and #7).

We also evaluated the associated financial costs for each workflow instance, taking into account the schedules generated by the different approaches. Figure 8 presents the financial cost (in US dollars) for each workflow instance under the following parameter configurations: (a) $\alpha_t = 1$ and $\alpha_b = 0$; (b) $\alpha_t = 0$ and $\alpha_b = 1$; (c) $\alpha_t = 0.5$ and $\alpha_b = 0.5$; (d) $\alpha_t = 0.75$ and $\alpha_b = 0.25$; (e) $\alpha_t = 0.25$ and $\alpha_b = 0.75$; (f) $\alpha_t = 0.6$ and $\alpha_b = 0.4$; and (g) $\alpha_t = 0.4$ and $\alpha_b = 0.6$. Consistent with the observations from the makespan analysis, `MemoirGRASP` generates schedules whose financial costs are close to those of the optimal solution in scenarios where financial

Tabela 7: Analysis of the Objective Function Value and Execution Time for Scenario #2 - $\alpha_t = 0$ and $\alpha_b = 1$

	Exact Solution		**Greedy**		**Multi-start**		`MemoirGRASP`	
Workflow	O.F	Time (s)	O.F	Time (s)	O.F	Time (s)	O.F	Time (s)
Synthetic_7_A	0.16	0.55	0.16	≤ 0.01	0.16	≤ 0.01	0.16	≤ 0.01
Synthetic_7_B	0.17	0.68	0.21	≤ 0.01	0.17	≤ 0.01	0.17	≤ 0.01
Synthetic_7_C	0.16	0.20	0.20	≤ 0.01	0.16	≤ 0.01	0.16	≤ 0.01
Synthetic_11_A	0.01	3.44	0.02	≤ 0.01	0.01	≤ 0.01	0.01	≤ 0.01
Synthetic_11_B	0.01	197.44	0.02	≤ 0.01	0.01	≤ 0.01	0.01	≤ 0.01
Synthetic_11_C	0.01	6.22	0.01	≤ 0.01	0.01	≤ 0.01	0.01	≤ 0.01
Synthetic_12_A	0.12	9.42	0.12	≤ 0.01	0.12	≤ 0.01	0.12	≤ 0.01
Synthetic_12_B	0.11	80.67	0.12	≤ 0.01	0.11	≤ 0.01	0.11	≤ 0.01
Synthetic_12_C	0.08	176.33	0.26	≤ 0.01	0.08	≤ 0.01	0.08	≤ 0.01
Synthetic_13_A	0.04	0.77	0.04	≤ 0.01	0.04	≤ 0.01	0.04	≤ 0.01
Synthetic_13_B	0.06	1.92	0.08	≤ 0.01	0.06	≤ 0.01	0.06	≤ 0.01
Synthetic_13_C	0.05	377.03	0.09	≤ 0.01	0.05	≤ 0.01	0.05	≤ 0.01
Synthetic_22_A	0.04	1.64	0.04	≤ 0.01	0.04	≤ 0.01	0.04	≤ 0.01
Synthetic_22_B	0.04	3.13	0.07	≤ 0.01	0.04	≤ 0.01	0.04	≤ 0.01
Synthetic_22_C	0.06	97.31	0.07	≤ 0.01	0.06	≤ 0.01	0.06	≤ 0.01

Tabela 8: Analysis of the Objective Function Value and Execution Time for Scenario #3 - $\alpha_t = 0.5$ and $\alpha_b = 0.5$

	Exact Solution		**Greedy**		**Multi-start**		`MemoirGRASP`	
Workflow	O.F	Time (s)	O.F	Time (s)	O.F	Time (s)	O.F	Time (s)
Synthetic_7_A	0.32	0.64	0.32	≤ 0.01	0.32	≤ 0.01	0.32	≤ 0.01
Synthetic_7_B	0.33	1.37	0.38	≤ 0.01	0.33	≤ 0.01	0.33	≤ 0.01
Synthetic_7_C	0.35	0.76	0.38	≤ 0.01	0.35	≤ 0.01	0.35	≤ 0.01
Synthetic_11_A	0.22	27.43	0.33	≤ 0.01	0.24	≤ 0.01	0.22	≤ 0.01
Synthetic_11_B	0.13	59.45	0.39	≤ 0.01	0.19	≤ 0.01	**0.14**	≤ 0.01
Synthetic_11_C	0.11	56.60	0.22	≤ 0.01	0.16	≤ 0.01	**0.12**	≤ 0.01
Synthetic_12_A	0.24	22.02	0.27	≤ 0.01	0.24	≤ 0.01	0.24	≤ 0.01
Synthetic_12_B	0.23	36.98	0.26	≤ 0.01	0.24	≤ 0.01	0.23	≤ 0.01
Synthetic_12_C	0.18	92.37	0.43	≤ 0.01	0.18	≤ 0.01	0.18	≤ 0.01
Synthetic_13_A	0.21	3.77	0.21	≤ 0.01	0.21	≤ 0.01	0.21	≤ 0.01
Synthetic_13_B	0.22	2.50	0.40	≤ 0.01	0.22	≤ 0.01	0.22	≤ 0.01
Synthetic_13_C	0.18	3.45	0.30	≤ 0.01	0.18	≤ 0.01	0.18	≤ 0.01
Synthetic_22_A	0.20	3.79	0.20	≤ 0.01	0.20	≤ 0.01	0.20	≤ 0.01
Synthetic_22_B	0.18	5.92	0.34	≤ 0.01	0.22	≤ 0.01	**0.22**	≤ 0.01
Synthetic_22_C	0.26	107.80	0.34	≤ 0.01	0.31	≤ 0.01	**0.31**	≤ 0.01

cost is the primary objective (Scenarios #2, #5, and #7) as well as in scenarios where it is not the primary objective (Scenarios #1, #3, #4, and #6). These results indicate that `MemoirGRASP` produces schedules that are near-optimal with respect to both makespan and financial cost, demonstrating its effectiveness across different optimization priorities.

However, it is important to note that some weight configurations do not define a clearly "dominant" criterion for optimization. For instance, by analyzing both Figure 7 and Figure 8, one can observe that the configurations with $\alpha_t = 0.6$ and $\alpha_b = 0.4$ and $\alpha_t = 0.4$ and $\alpha_b = 0.6$ yield results that are very similar to those obtained with $\alpha_t = 0.5$ and $\alpha_b = 0.5$. A similar pattern can be observed for the configurations $\alpha_t = 0.75$ and $\alpha_b = 0.25$ and $\alpha_t = 0.25$ and $\alpha_b = 0.75$. These observations suggest that, depending on the structure of the workflow (both in terms of the number of activations and the complexity of their specification) configurations that are neither at the extremes nor exactly balanced may not offer clear benefits. In such cases, the value of the objective function tends to remain close to that of the first three scenarios, indicating limited sensitivity to these intermediate weight values. Based on these findings, in the next subsection we evaluate `MemoirGRASP` using the Phenomenal workflow, restricting the analysis to the following configurations: $\alpha_t = 1$ and $\alpha_b = 0$ (Scenario #1), $\alpha_t = 0$ and $\alpha_b = 1$ (Scenario #2), and $\alpha_t = 0.5$ and $\alpha_b = 0.5$ (Scenario #3).

5.4 Evaluation with the Phenomenal Workflow through Simulation

As discussed in Subsection 5.2.1, Phenomenal is a compute- and data-intensive workflow. In the experiments, we considered both single-site and multisite cloud environments. To define site heterogeneity in terms of storage and computing resources, we varied the slowdown index (cs_j) and the financial costs (c_j^M) of the VMs

Tabela 9: Analysis of the Objective Function Value and Execution Time for Scenario #4 - $\alpha_t = 0.75$ and $\alpha_b = 0.25$

	Exact Solution		**Greedy**		**Multi-start**		`MemoirGRASP`	
Workflow	O.F	Time (s)	O.F	Time (s)	O.F	Time (s)	O.F	Time (s)
Synthetic_7_A	0.39	0.48	0.39	≤ 0.01	0.39	≤ 0.01	0.39	≤ 0.01
Synthetic_7_B	0.42	1.12	0.46	≤ 0.01	0.42	≤ 0.01	0.42	≤ 0.01
Synthetic_7_C	0.41	0.79	0.44	≤ 0.01	0.41	≤ 0.01	0.41	≤ 0.01
Synthetic_11_A	0.30	595.33	0.48	≤ 0.01	0.33	≤ 0.01	0.30	≤ 0.01
Synthetic_11_B	0.17	330.76	0.57	≤ 0.01	0.26	≤ 0.01	**0.20**	≤ 0.01
Synthetic_11_C	0.15	12.62	0.30	≤ 0.01	0.22	≤ 0.01	**0.16**	≤ 0.01
Synthetic_12_A	0.28	10.52	0.33	≤ 0.01	0.28	≤ 0.01	0.28	≤ 0.01
Synthetic_12_B	0.27	36.93	0.31	≤ 0.01	0.28	≤ 0.01	0.27	≤ 0.01
Synthetic_12_C	0.21	16.35	0.53	≤ 0.01	0.21	≤ 0.01	0.21	≤ 0.01
Synthetic_13_A	0.30	1.59	0.30	≤ 0.01	0.30	≤ 0.01	0.30	≤ 0.01
Synthetic_13_B	0.29	3.45	0.50	≤ 0.01	0.29	≤ 0.01	0.29	≤ 0.01
Synthetic_13_C	0.22	4.07	0.36	≤ 0.01	0.22	≤ 0.01	0.22	≤ 0.01
Synthetic_22_A	0.28	3.89	0.28	≤ 0.01	0.28	≤ 0.01	0.28	≤ 0.01
Synthetic_22_B	0.24	4.35	0.48	≤ 0.01	0.30	≤ 0.01	**0.30**	≤ 0.01
Synthetic_22_C	0.34	21.29	0.48	≤ 0.01	0.43	≤ 0.01	**0.43**	≤ 0.01

Tabela 10: Analysis of the Objective Function Value and Execution Time for Scenario #5 - $\alpha_t = 0.25$ and $\alpha_b = 0.75$

	Exact Solution		**Greedy**		**Multi-start**		`MemoirGRASP`	
Workflow	O.F	Time (s)	O.F	Time (s)	O.F	Time (s)	O.F	Time (s)
Synthetic_7_A	0.24	0.60	0.24	≤ 0.01	0.24	≤ 0.01	0.24	≤ 0.01
Synthetic_7_B	0.25	1.28	0.30	≤ 0.01	0.25	≤ 0.01	0.25	≤ 0.01
Synthetic_7_C	0.30	0.98	0.33	≤ 0.01	0.30	≤ 0.01	0.30	≤ 0.01
Synthetic_11_A	0.13	17.58	0.17	≤ 0.01	0.13	≤ 0.01	0.13	≤ 0.01
Synthetic_11_B	0.09	99.05	0.20	≤ 0.01	0.10	≤ 0.01	**0.10**	≤ 0.01
Synthetic_11_C	0.08	134.30	0.13	≤ 0.01	0.10	≤ 0.01	0.08	≤ 0.01
Synthetic_12_A	0.21	36.64	0.22	≤ 0.01	0.22	≤ 0.01	0.21	≤ 0.01
Synthetic_12_B	0.19	239.11	0.22	≤ 0.01	0.19	≤ 0.01	0.19	≤ 0.01
Synthetic_12_C	0.14	202.82	0.33	≤ 0.01	0.14	≤ 0.01	0.14	≤ 0.01
Synthetic_13_A	0.13	1.19	0.13	≤ 0.01	0.13	≤ 0.01	0.13	≤ 0.01
Synthetic_13_B	0.16	4.41	0.24	≤ 0.01	0.16	≤ 0.01	0.16	≤ 0.01
Synthetic_13_C	0.14	19.58	0.24	≤ 0.01	0.14	≤ 0.01	0.14	≤ 0.01
Synthetic_22_A	0.12	3.68	0.12	≤ 0.01	0.12	≤ 0.01	0.12	≤ 0.01
Synthetic_22_B	0.12	20.10	0.20	≤ 0.01	0.13	≤ 0.01	**0.13**	≤ 0.01
Synthetic_22_C	0.18	142.41	0.20	≤ 0.01	0.19	≤ 0.01	**0.19**	≤ 0.01

involved in the execution. We considered as a baseline the m5a.8xlarge VM type in the AWS cloud to calculate the slowdown of all involved VMs. Based on the experience of previous work, we have chosen the m5a.8xlarge instance because it is a general-purpose VM type with 32 vCPUs, 128.0 GiB of memory, and a financial cost of $1.376 per hour ($0.0004 per second) where multiple tasks can execute in the same VM in parallel. Each storage has a capacity of 160 GB. The execution scenarios are given in Table 13.

The first scenario, *i.e.*, "*One Site no Memoization*", involves a single cloud site with 3 VMs with different processing capacities. In this scenario, Phenomenal is executed without accessing previously cached data, *i.e.*, does not exploit memoization techniques. The second scenario, *i.e.*, "*One Site with Memoization*" considers a single cloud site with 4 VMs, each with the same financial cost and processing capacity. In this scenario, memoization techniques are exploited, but only 27% of the data files are already cached. The third scenario, *i.e.*, "*Two Sites Homogeneous*", has 4 VMs, two at each site with different processing capacities, but all have the same financial cost. It has only one storage device to access cached data files. Finally, the fourth scenario, *i.e.*, "*Two Sites Heterogeneous*", has 2 cloud sites and a total of 7 VMs, with three at the first site and four at the second site. The financial cost and processing capacities are also heterogeneous, and it has two storage devices available to access cached data. In the fourth scenario, 85% of data files were already cached and available for consumption. For each of scenario, we scheduled Phenomenal by varying α_t and α_b as follows: (a) $\alpha_t = 1$ and $\alpha_b = 0$ (focus on reducing makespan), (b) $\alpha_t = 0$ and $\alpha_b = 1$ (focus on reducing financial costs), and (c) $\alpha_t = 0.5$ and $\alpha_b = 0.5$ (balanced objectives).

Unlike the analysis involving synthetic workflows, comparing the execution of Phenomenal with the exact

Tabela 11: Analysis of the Objective Function Value and Execution Time for Scenario #6 - $\alpha_t = 0.6$ and $\alpha_b = 0.4$

	Exact Solution		**Greedy**		**Multi-start**		`MemoirGRASP`	
Workflow	O.F	Time (s)	O.F	Time (s)	O.F	Time (s)	O.F	Time (s)
Synthetic_7_A	0.35	0.90	0.35	≤ 0.01	0.35	≤ 0.01	0.35	≤ 0.01
Synthetic_7_B	0.37	0.95	0.41	≤ 0.01	0.37	≤ 0.01	0.37	≤ 0.01
Synthetic_7_C	0.38	0.76	0.41	≤ 0.01	0.38	≤ 0.01	0.38	≤ 0.01
Synthetic_11_A	0.25	23.08	0.40	≤ 0.01	0.28	≤ 0.01	0.25	≤ 0.01
Synthetic_11_B	0.14	55.43	0.46	≤ 0.01	0.21	≤ 0.01	**0.18**	≤ 0.01
Synthetic_11_C	0.13	30.84	0.25	≤ 0.01	0.18	≤ 0.01	0.13	≤ 0.01
Synthetic_12_A	0.26	18.22	0.30	≤ 0.01	0.26	≤ 0.01	0.26	≤ 0.01
Synthetic_12_B	0.24	59.94	0.28	≤ 0.01	0.25	≤ 0.01	0.24	≤ 0.01
Synthetic_12_C	0.19	49.25	0.47	≤ 0.01	0.19	≤ 0.01	0.19	≤ 0.01
Synthetic_13_A	0.25	3.21	0.25	≤ 0.01	0.25	≤ 0.01	0.25	≤ 0.01
Synthetic_13_B	0.25	2.67	0.44	≤ 0.01	0.25	≤ 0.01	0.25	≤ 0.01
Synthetic_13_C	0.20	7.22	0.32	≤ 0.01	0.20	≤ 0.01	0.20	≤ 0.01
Synthetic_22_A	0.23	5.78	0.23	≤ 0.01	0.23	≤ 0.01	0.23	≤ 0.01
Synthetic_22_B	0.21	5.00	0.39	≤ 0.01	0.25	≤ 0.01	**0.25**	≤ 0.01
Synthetic_22_C	0.29	30.48	0.39	≤ 0.01	0.36	≤ 0.01	**0.36**	≤ 0.01

Tabela 12: Analysis of the Objective Function Value and Execution Time for Scenario #7 - $\alpha_t = 0.4$ and $\alpha_b = 0.6$

	Exact Solution		**Greedy**		**Multi-start**		`MemoirGRASP`	
Workflow	O.F	Time (s)	O.F	Time (s)	O.F	Time (s)	O.F	Time (s)
Synthetic_7_A	0.28	1.03	0.28	≤ 0.01	0.28	≤ 0.01	0.28	≤ 0.01
Synthetic_7_B	0.30	1.00	0.35	≤ 0.01	0.30	≤ 0.01	0.30	≤ 0.01
Synthetic_7_C	0.33	0.93	0.36	≤ 0.01	0.33	≤ 0.01	0.33	≤ 0.01
Synthetic_11_A	0.19	42.56	0.27	≤ 0.01	0.20	≤ 0.01	0.19	≤ 0.01
Synthetic_11_B	0.11	222.26	0.31	≤ 0.01	0.16	≤ 0.01	**0.13**	≤ 0.01
Synthetic_11_C	0.10	113.75	0.19	≤ 0.01	0.13	≤ 0.01	0.10	≤ 0.01
Synthetic_12_A	0.23	20.29	0.25	≤ 0.01	0.25	≤ 0.01	0.23	≤ 0.01
Synthetic_12_B	0.22	60.21	0.24	≤ 0.01	0.23	≤ 0.01	0.22	≤ 0.01
Synthetic_12_C	0.16	97.07	0.39	≤ 0.01	0.16	≤ 0.01	0.16	≤ 0.01
Synthetic_13_A	0.18	1.36	0.18	≤ 0.01	0.18	≤ 0.01	0.18	≤ 0.01
Synthetic_13_B	0.20	3.20	0.34	≤ 0.01	0.20	≤ 0.01	0.20	≤ 0.01
Synthetic_13_C	0.17	8.78	0.28	≤ 0.01	0.17	≤ 0.01	0.17	≤ 0.01
Synthetic_22_A	0.17	16.65	0.17	≤ 0.01	0.17	≤ 0.01	0.17	≤ 0.01
Synthetic_22_B	0.16	9.07	0.29	≤ 0.01	0.18	≤ 0.01	**0.18**	≤ 0.01
Synthetic_22_C	0.23	53.24	0.29	≤ 0.01	0.26	≤ 0.01	**0.26**	≤ 0.01

Tabela 13: Experimental Setup for the Phenomenal Workflow

Scenario Name	M_{vm}	cs_j	c_j^M	ω **(Mibps)**	M_{vm}	c
One Site no Memoization	3	1.75 (1), 1.0 (2)	(0.0001, 0.0005, 0.0005)	10	0	N
One Site with Memoization	4	1.0 (4)	(0.0003, 0.0003, 0.0003, 0.0003)	10	1	Y
Two sites Homogeneous	4	1.0 (1), 3.0 (3)	(0.0003, 0.0003, 0.0003, 0.0003)	10	1	Y
Two sites Heterogeneous	7	1.0 (3), 1.75 (1), 3.0 (3)	(0.0003, 0.0003, 0.0003, 0.0003, 0.001, 0.005, 0.005)	10	2	Y

solution is not practical, in terms of execution time. Thus, we evaluate the outcomes of the greedy scheduling, the multi-start approach, and `MemoirGRASP` as illustrated in Tables 14, 15, and 16. In these tables, the first column serves to identify each of the aforementioned scenarios. Subsequently, we present the objective function (O.F) value along with the time required to solve the greedy algorithm, the multi-start algorithm, and `MemoirGRASP`. The values reported for `MemoirGRASP` represent averages derived from five executions.

From Tables 14, 15, and 16, we observe that `MemoirGRASP` consistently yields the best solutions for all scenarios (in bold), *i.e.*, the value of the objective function, showcasing the same value of the objective function in only two cases. Furthermore, `MemoirGRASP` requires significantly more time to generate the schedule compared to the greedy and multi-start approaches. `MemoirGRASP` takes an average of 2.77, 2.3, and 2.61

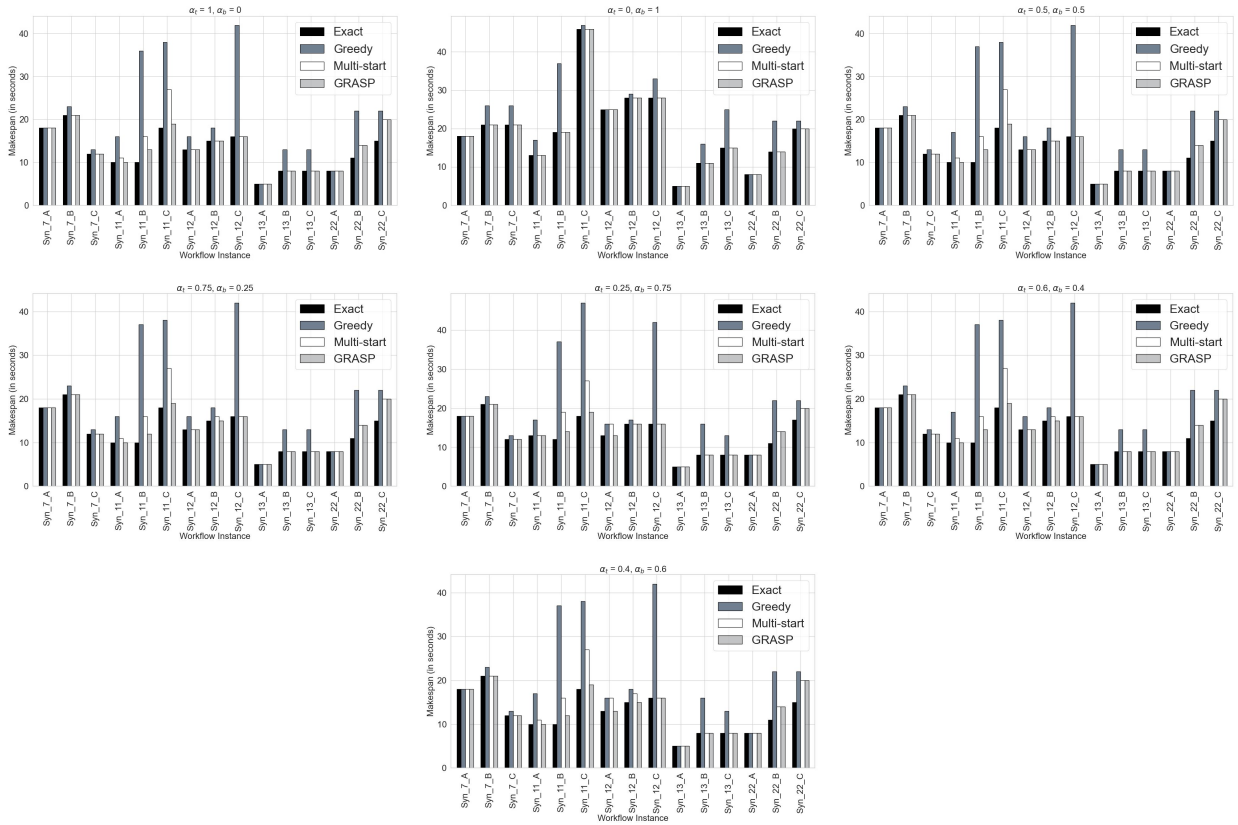


Figura 7: Analysis of Makespan of Synthetic Workflows - (a) $\alpha_t = 1$ and $\alpha_b = 0$, (b) $\alpha_t = 0$ and $\alpha_b = 1$ (c) $\alpha_t = 0.5$ and $\alpha_b = 0.5$ (d) $\alpha_t = 0.75$ and $\alpha_b = 0.25$ (e) $\alpha_t = 0.25$ and $\alpha_b = 0.75$ (f) $\alpha_t = 0.6$ and $\alpha_b = 0.4$ (g) $\alpha_t = 0.4$ and $\alpha_b = 0.6$.

Tabela 14: Analysis of the Objective Function Value and Time to Solution for the Phenomenal workflow with $\alpha_t = 1$ and $\alpha_b = 0$

	Greedy		**Multi-start**		`MemoirGRASP`	
Scenario	O.F	Time (s)	O.F	Time (s)	O.F	Time (s)
One Site no Memoization	0.435843	0.000055	0.424498	0.005260	**0.390478**	0.027763
One Site with Memoization	0.598625	0.000154	0.517396	0.009976	**0.503195**	0.027725
Two sites Homogeneous	0.485631	0.000070	0.390096	0.005302	**0.383098**	0.024794
Two sites Heterogeneous	0.233489	0.000076	0.233489	0.010428	**0.217183**	0.048949

Tabela 15: Analysis of the Objective Function Value and Time to Solution for the Phenomenal Workflow with - $\alpha_t = 0$ and $\alpha_b = 1$

	Greedy		**Multi-start**		`MemoirGRASP`	
Scenario	O.F	Time (s)	O.F	Time (s)	O.F	Time (s)
One Site no Memoization	**0.050217**	0.000052	**0.050217**	0.004717	**0.050217**	0.013404
One Site with Memoization	0.171327	0.000327	0.162048	0.005525	**0.159101**	0.024201
Two sites Homogeneous	**0.097592**	0.000089	**0.097592**	0.005993	**0.097592**	0.024132
Two sites Heterogeneous	**0.003883**	0.000258	**0.003883**	0.013432	**0.003883**	0.070640

seconds for the three configurations of α_t and α_b, respectively. In contrast, greedy scheduling is executed in less than 0.01 seconds for all configurations of α_t and α_b.

We evaluated the makespan and the financial costs associated with each combination of α_t and α_b, considering the schedules generated by various approaches, including (1) the greedy algorithm (depicted by the black bar), (2) the multi-start approach (represented by the dark grey bar), and (3) `MemoirGRASP` (indicated by the light grey bar). Figure 9(a) illustrates the makespan, while Figure 9(b) presents the financial cost for

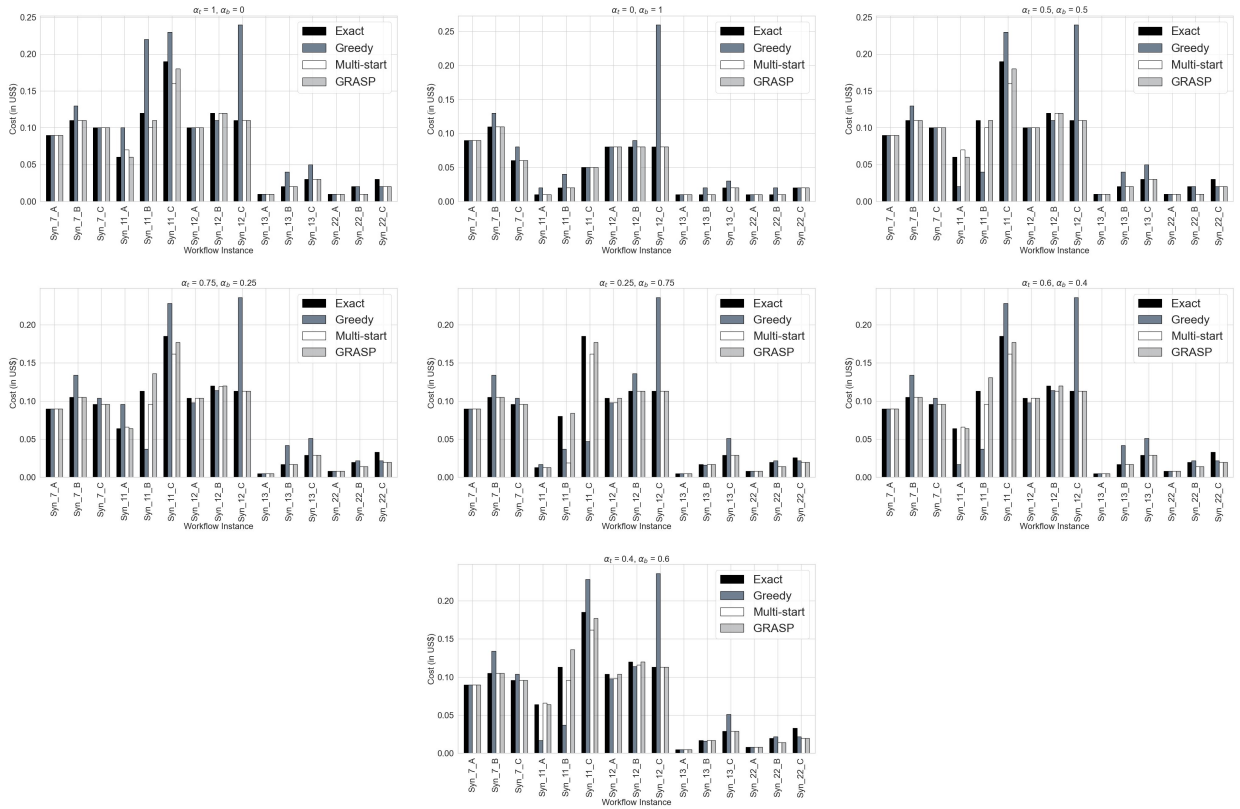


Figura 8: Analysis of Financial Cost of Synthetic Workflows - (a) $\alpha_t = 1$ and $\alpha_b = 0$, (b) $\alpha_t = 0$ and $\alpha_b = 1$ (c) $\alpha_t = 0.5$ and $\alpha_b = 0.5$ (d) $\alpha_t = 0.75$ and $\alpha_b = 0.25$ (e) $\alpha_t = 0.25$ and $\alpha_b = 0.75$ (f) $\alpha_t = 0.6$ and $\alpha_b = 0.4$ (g) $\alpha_t = 0.4$ and $\alpha_b = 0.6$.

Tabela 16: Analysis of the Objective Function Value and Time to Solution for the Phenomenal Workflow with - $\alpha_t = 0.5$ and $\alpha_b = 0.5$

	Greedy		**Multi-start**		`MemoirGRASP`	
Scenario	O.F	Time (s)	O.F	Time (s)	O.F	Time (s)
One Site no Memoization	0.304752	0.000056	0.297230	0.003975	**0.288231**	0.022641
One Site with Memoization	0.411257	0.000049	0.390021	0.005728	**0.379915**	0.028099
Two sites Homogeneous	0.291320	0.000063	0.276777	0.006529	**0.264659**	0.024692
Two sites Heterogeneous	0.137565	0.000133	0.137565	0.012887	**0.130440**	0.058412

each scheduling approach for the single site without using memoization. Intuitively, this scenario is the most challenging for `MemoirGRASP`, as there is no cached data available for reuse. Thus, `MemoirGRASP` is unable to discover better solutions compared with the other approaches. Figure 9(a) and 9(b) shows that there are no significant differences in terms of both makespan and financial costs among the approaches. Since this cloud environment is relatively small, it leaves limited room for exploring various possible schedules, thereby diminishing the utility of `MemoirGRASP`.

Figure 10 shows the results of makespan (Figure 10(a)) and financial cost (Figure 10(b)) for the scenario involving a single site with memoization for each scheduling algorithm and combination of α_t and α_b. In this scenario, `MemoirGRASP` is better when the user focuses on reducing makespan ($\alpha_t = 1$ and $\alpha_b = 0$) or minimizing financial costs ($\alpha_t = 0$ and $\alpha_b = 1$). In these cases, `MemoirGRASP` demonstrates the ability to generate faster schedules or schedules with equivalent speed while reducing the financial cost. The exception occurs when the weights are balanced ($\alpha_t = 0.5$ and $\alpha_b = 0.5$), where `MemoirGRASP` produces faster schedules but with higher financial cost. In this balanced scenario, each algorithm may exhibit a distinct behavior. While the greedy and multi-start algorithms choose to execute activations on more economical but slower VMs, resulting in lower costs but longer execution times, `MemoirGRASP` opts to execute activations on more

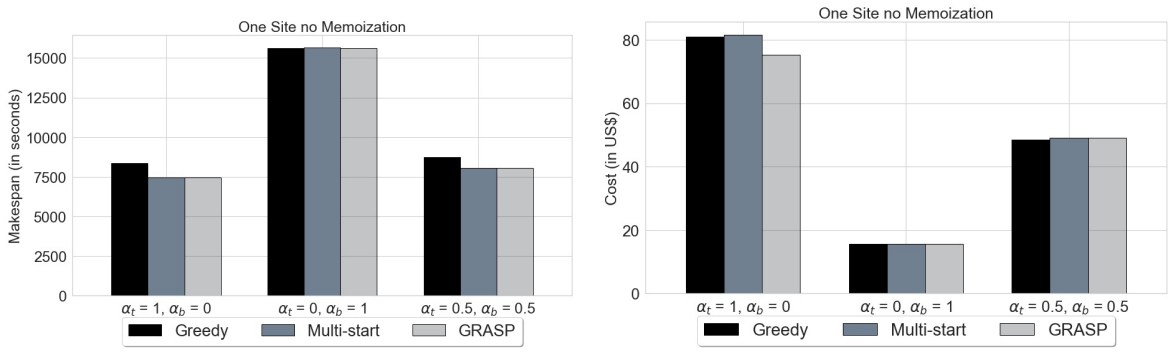


Figura 9: One Site no Memoization (a) Makespan (b) Financial Cost

powerful yet costlier VMs. This decision results in reduced execution times for Phenomenal but entails a 23% rise in financial cost. This trade-off is typical in optimization problems where weights are balanced among objectives. Given that the objective function's value serves as the "guide" for selecting and exploring possible solutions, the identical value of the objective function may represent both the decisions made by both the greedy algorithm and `MemoirGRASP`, despite their practical implementations being entirely distinct.

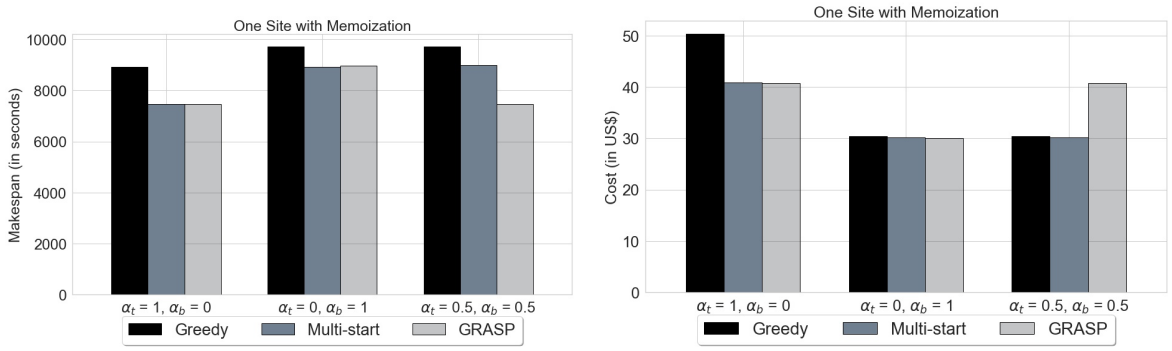


Figura 10: One Site with Memoization (a) Makespan (b) Financial Cost

Figure 11 shows the results of makespan (Figure 11(a)) and financial cost (Figure 11(b)) for the scenario involving a multisite cloud with memoization and two homogeneous sites for each scheduling algorithm and combination of α_t and α_b. In this scenario, `MemoirGRASP` consistently generates schedules that are both faster and more cost-effective, regardless of the chosen values of α_t and α_b. This makes `MemoirGRASP` suitable for rich computing environments with multiple VMs and sites, outperforming other approaches. As expected, the greedy algorithm produced slower and costlier results for all combinations of α_t and α_b. This is due to its inability to make short-term sacrifices for long-term benefits and its susceptibility to the order of activations for execution. Compared to the multi-start approach, `MemoirGRASP` still shows better results, in particular, in terms of financial costs.

Figure 12 presents the results of makespan (Figure 12(a)) and financial cost (Figure 12(b)) for the scenario involving a multisite with memoization and two heterogeneous sites for each scheduling algorithm and combination of α_t and α_b. Similar to the previous scenario, `MemoirGRASP` consistently generates schedules that are both faster and more cost-effective, regardless of the chosen values of α_t and α_b. However, since this scenario involves more VMs, `MemoirGRASP` is able to further widen the gap with the other approaches in terms of both makespan and financial cost. This shows that, while `MemoirGRASP` may require additional time to generate the schedule, it excels in producing faster and cost-effective schedules, in particular, in larger computing environments. This illustrated the advantages of using `MemoirGRASP` in rich computing environments where cached data can be reused. One interesting point in Figure 12(b) is that when $\alpha_t = 0$ and $\alpha_b = 1$, i.e., the focus is on reducing financial cost, all schedules consumed all possible cached files, which represent 85% of the data files consumed and produced by Phenomenal. This schedule reduces the financial costs but increases the makespan

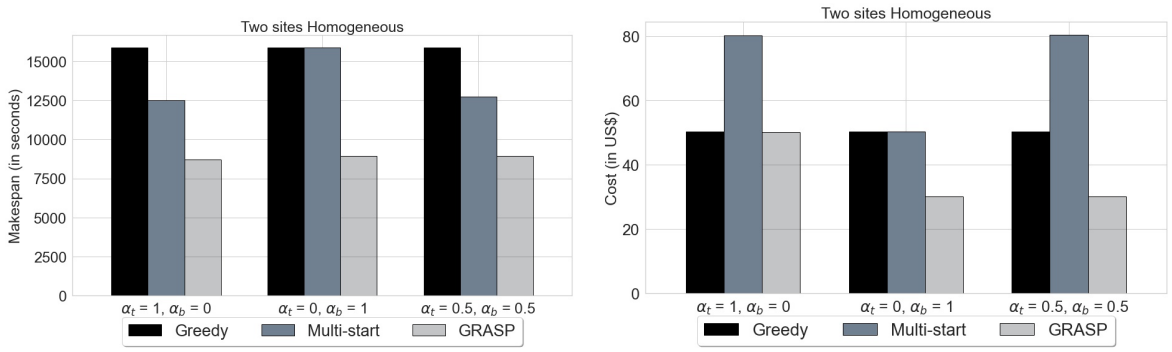


Figura 11: Two sites Homogeneous (a) Makespan (b) Financial Cost

at a peak level since the schedule chooses the cheapest (and lesser powerful) VM to transfer data.

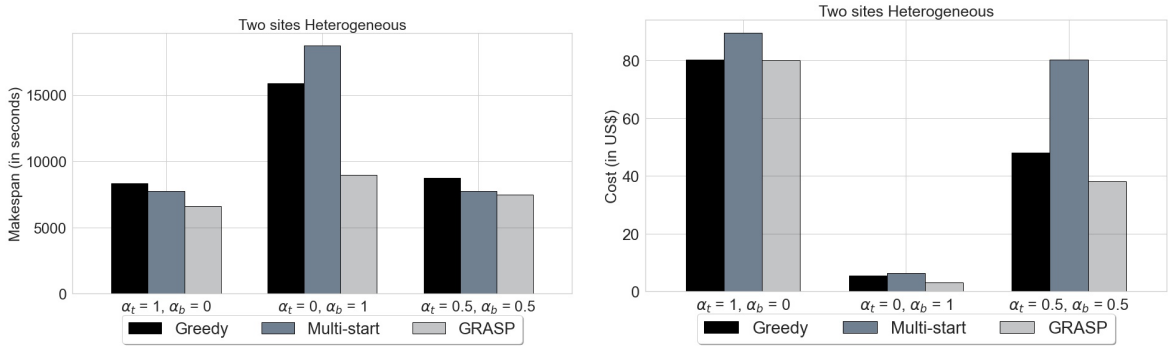


Figura 12: Two sites Heterogeneous (a) Makespan (b) Financial Cost

5.5 Evaluation with Montage in a Real Cloud Environment

Finally, we evaluated the performance of the schedules produced by `MemoirGRASP` in a real cloud computing environment. The Phenomenal workflow was not used in this experiment because, although its toolkit is open-source, the dataset employed in its experiments is under embargo and cannot be uploaded to public cloud platforms where the evaluation was conducted. Consequently, we selected the Montage workflow, configured to process a 0.5-degree region of the sky, for this set of experiments. We modeled and executed Montage using the `AkôFlow` middleware [FKP+24]. The execution environment consisted of a virtual cluster deployed on the Google Cloud Platform (GCP) and composed of three VMs: two instances of the `e2-highcpu-16` type and one instance of the `e2-highcpu-8` type. These VM types were selected because they are general-purpose configurations with a balanced ratio of processing power to cost. The `e2-highcpu-16` instances provide 16 vCPUs and 16 GiB RAM, at a cost of US$ 0.6332 per hour, while the `e2-highcpu-8` instance offers 8 vCPUs and 8 GiB RAM, at a cost of US$ 0.3166 per hour.

Tabela 17: Analysis of the Objective Function Value and Time to Solution for the Montage Workflow

	$\alpha_t = 1$ and $\alpha_b = 0$		$\alpha_t = 0$ and $\alpha_b = 1$		$\alpha_t = 0.5$ and $\alpha_b = 0.5$	
Scenario	O.F	Time (s)	O.F	Time (s)	O.F	Time (s)
Montage without Cached Data	0.11624	0.08658	0.03962	0.08181	0.06877	0.085034
Montage 25% cached data	0.05101	0.05914	0.00855	0.04351	0.03078	0.085034
Montage 50% cached data	0.03852	0.02387	0.00646	0.02012	0.02246	0.023545

Table 17 presents the results obtained with `MemoirGRASP` for three weight configurations: $\alpha_t = 1$ and $\alpha_b = 0$ (Scenario #1), $\alpha_t = 0$ and $\alpha_b = 1$ (Scenario #2), and $\alpha_t = 0.5$ and $\alpha_b = 0.5$ (Scenario #3). For each

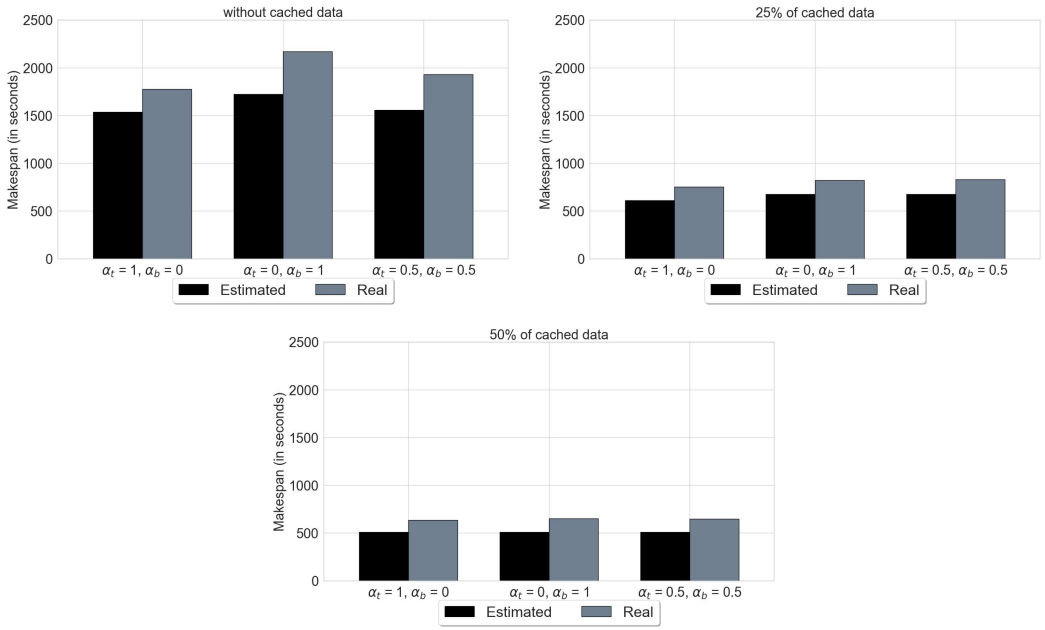


Figura 13: Execution of Montage Workflow (a) without cached data (b) 25% cached data (c) 50% cached data.

configuration, we evaluated three variants of the Montage workflow: (i) execution without cached data; (ii) execution with 25% of the data cached for reuse, specifically the files produced by the `mProject` activations; and (iii) execution with 50% of the data cached for reuse, specifically the files produced by `mProject`, `mConcatFit`, and `mBgModel`. In the table, the first column identifies the Montage variant. For each variant, we report the value of the objective function, as defined in Function 3.2 of Section 3.2, along with the time required by `MemoirGRASP` to generate the schedule for each weight configuration. All values in Table 17 correspond to the average of five independent executions. It is worth emphasizing that `MemoirGRASP` required no more than 0.09 seconds to generate the schedule. This execution time is negligible when compared to the makespan of the workflow, and therefore can be considered acceptable within the context of the scheduling process with memoization.

Figure 13 depicts the makespan results, whereas Figure 14 presents the financial costs, both evaluated for each variant of the Montage workflow and for every tested combination of weights α_t and α_b. Across all configurations, `MemoirGRASP` consistently produces schedules that achieve lower execution times and reduced financial costs, regardless of the specific values assigned to α_t and α_b. Two main points deserve analysis: (i) the discrepancies between the estimated makespan and financial cost values estimated by `MemoirGRASP` and their actual measured values; and (ii) the extent to which the presence of cached data influences the comparative advantage of using `MemoirGRASP`.

Regarding the first point, the difference between the estimated makespan and the actual observed makespan ranges from approximately 22% to 24%. This difference is mainly explained to characteristics of cloud environments that are not explicitly modeled within `MemoirGRASP`. Such characteristics include overcommitment strategies employed by cloud providers, where the number of VMs allocated temporarily exceeds the available physical resources, VM live migrations between physical servers during workflow execution, and the concurrent sharing of physical resources among multiple VMs. These factors are dynamic, difficult to estimate and model, and challenging to incorporate into heuristics such as `MemoirGRASP`. While alternative approaches, such as reinforcement learning–based scheduling [NSPdO21], may have the potential to adaptively handle these fluctuations, they often suffer from slower convergence times.

The second point concerns the proportion of workflow data already cached prior to execution. When comparing the case with no cached data against the scenario with 25% cached data, it becomes evident that `MemoirGRASP` produces more efficient schedules, both in terms of makespan reduction and financial cost savings. However, when the cached proportion reaches 50%, the performance gap narrows. This reduction in

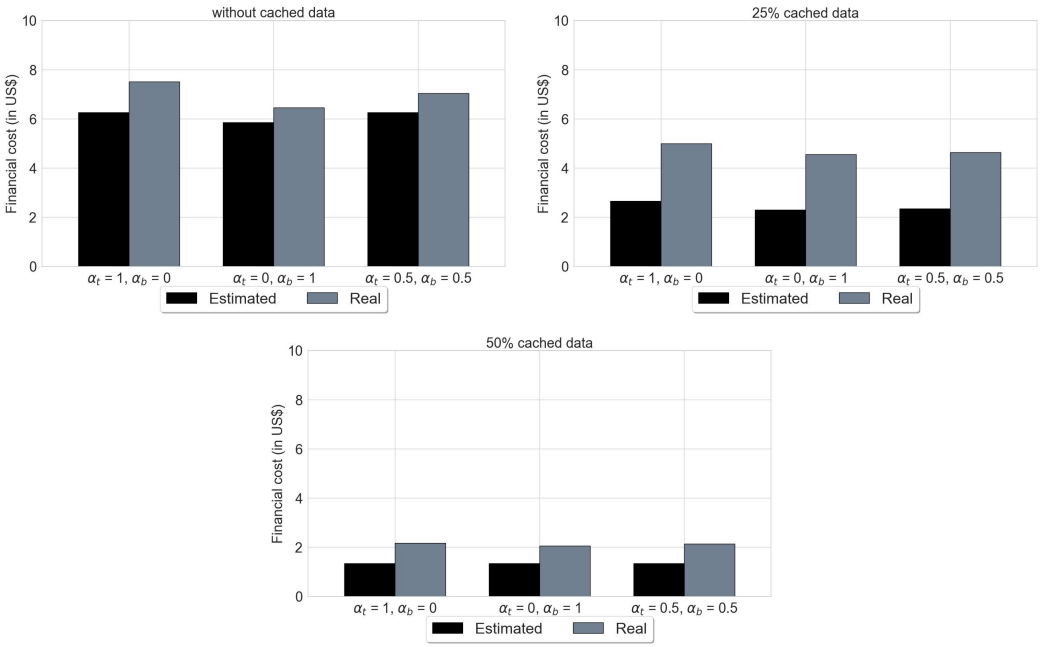


Figura 14: Financial Cost of Montage Workflow (a) without cached data (b) 25% cached data (c) 50% cached data.

benefit occurs because, with a large portion of the required data already available, the remaining activations of the workflow become straightforward to schedule, diminishing the advantage of employing a more complex scheduling strategy such as `MemoirGRASP`. In such cases, a simple greedy scheduler may produce and acceptable schedule. Specifically for the Montage workflow, when more than half of the data is cached, a straightforward greedy or multi-start heuristic may even outperform `MemoirGRASP` in terms of computational overhead versus achieved benefit.

6 Conclusion

A promising solution for efficient workflow execution is to use cached intermediate data, which avoids re-executing parts of the workflows. However, reusing cached data is hard to make effective. In this manuscript, we proposed `MemoirGRASP`, a GRASP-based strategy designed for the efficient execution of data-intensive workflows in a multisite cloud. `MemoirGRASP` strives to produce optimal or near-optimal workflow activation schedules while leveraging memoization when cached data is available. Building upon previous mechanisms developed by our research group that determine the optimal moments to cache data, `MemoirGRASP` represents a step forward.

We conducted a two-step evaluation of `MemoirGRASP`. In the first step, we used synthetic workflows to compare schedules with the exact solution. `MemoirGRASP` produced good results, generating schedules with makespan and financial costs close to optimal in more than 80% of executions. In the second step, we evaluated `MemoirGRASP` using both the Montage workflow, from astronomic domain, and Phenomenal workflow, from plant phenotyping domain. The comparisons with a greedy algorithm and a multi-start approach revealed that `MemoirGRASP` consistently produced faster and cost-effective schedules, particularly in multisite scenarios with more VMs. Experiments in a real cloud environment also showed the viability of `MemoirGRASP`, even with a difference in the actual performance of the workflow in comparison with the performance estimated by `MemoirGRASP`.

While our approach represents a step forward in experimental science, in particular, in scenarios involving repeated workflow executions with parameter sweep, there is room for improvement. Future work includes incorporating decision-making capabilities into `MemoirGRASP` for caching data during scheduling based on the potential future usage of cached data. Furthermore, addressing concerns related to storage bottlenecks in

multi-user environments and exploring containment issues [BRV14] for more informed caching decisions are areas for future investigation.

Data Accessibility Statement

The data used in this manuscript will be available at `https://github.com/UFFeScience/MemoirGRASP`

Conflict Of Interest

The authors declare that they have no competing interests.

Ethics Statement

The authors declare that the manuscript does not report on or involve the use of any animal or human data or tissue.

Authors Contribution Statement

All authors contributed substantially to this manuscript. All authors participated in conceiving the study, defining the research objectives, and designing the methodology. **Rodrigo A. P. Silva** and **Gaëtan Heidsieck** conducted the analysis and developed the implementation. **Daniel de Oliveira**, **Yuri Forta**, **Esther Pacitti**, and **Patrick Valduriez** drafted the manuscript, and all authors critically revised it. Daniel de Oliveira supervised the research activities. All authors reviewed and approved the final manuscript.

Referências

[ACC+19] Simon Artzet, Tsu-Wei Chen, Jérôme Chopard, Nicolas Brichet, Michael Mielewczik, Sarah Cohen-Boulakia, Llorenç Cabrera-Bosquet, François Tardieu, Christian Fournier, and Christophe Pradal. Phenomenal: An automatic open source library for 3d shoot architecture reconstruction and analysis for image-based plant phenotyping. *bioRxiv*, 2019.

[APHC18] Simon Artzet, Christophe Pradal, Heidsieck, and Jerome Chopard. openalea/phenomenal: Phenomenal release, September 2018.

[BDZ06] Adam Belloum, Ewa Deelman, and Zhiming Zhao. Scientific workflows. *Sci. Program.*, 14(3-4):171, 2006.

[BRV14] Pablo Barceló, Miguel Romero, and Moshe Y. Vardi. Does query evaluation tractability help query containment? In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '14, page 188–199, New York, NY, USA, 2014. Association for Computing Machinery.

[BWL+19] Yadu N. Babuji, Anna Woodard, Zhuozhao Li, Daniel S. Katz, Ben Clifford, Rohan Kumar, Lukasz Lacinski, Ryan Chard, Justin M. Wozniak, Ian T. Foster, Michael Wilde, and Kyle Chard. Parsl: Pervasive parallel programming in python. In Jon B. Weissman, Ali Raza Butt, and Evgenia Smirni, editors, *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing, HPDC 2019, Phoenix, AZ, USA, June 22-29, 2019*, pages 25–36. ACM, 2019.

[CAWL14] Wanghu Chen, Ilkay Altintas, Jianwu Wang, and Jing Li. Enhancing smart re-run of kepler scientific workflows based on near optimum provenance caching in cloud. In *2014 IEEE World Congress on Services, SERVICES 2014, Anchorage, AK, USA, June 27 - July 2, 2014*, pages 378–384. IEEE Computer Society, 2014.

[CFS+06] Steven P. Callahan, Juliana Freire, Emanuele Santos, Carlos Eduardo Scheidegger, Cláudio T. Silva, and Huy T. Vo. Vistrails: visualization meets data management. In Surajit Chaudhuri, Vagelis Hristidis, and Neoklis Polyzotis, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006*, pages 745–747. ACM, 2006.

[CTR+17] Israel Casas, Javid Taheri, Rajiv Ranjan, Lizhe Wang, and Albert Y. Zomaya. A balanced scheduler with data reuse and replication for scientific workflows in cloud computing systems. *Future Gener. Comput. Syst.*, 74:168–178, 2017.

[DdSV+21] Ewa Deelman, Rafael Ferreira da Silva, Karan Vahi, Mats Rynge, Rajiv Mayani, Ryan Tanaka, Wendy R. Whitcup, and Miron Livny. The pegasus workflow management system: Translational computer science in practice. *J. Comput. Sci.*, 52:101200, 2021.

[dOLP19] Daniel C. M. de Oliveira, Ji Liu, and Esther Pacitti. *Data-Intensive Workflow Management: For Clouds and Data-Intensive and Scalable Computing Environments*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2019.

[dOOBM12] Daniel de Oliveira, Kary A. C. S. Ocaña, Fernanda Araujo Baião, and Marta Mattoso. A provenance-based adaptive scheduling heuristic for parallel scientific workflows in clouds. *J. Grid Comput.*, 10(3):521–552, 2012.

[EH02] Thomas Erlebach and Alexander Hall. Np-hardness of broadcast scheduling and inapproximability of single-source unsplittable min-cost flow. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '02, page 194–202, USA, 2002. Society for Industrial and Applied Mathematics.

[FKP+24] Wesley Ferreira, Liliane Kunstmann, Aline Paes, Marcos Bedo, and Daniel de Oliveira. Akôflow: um middleware para execução de workflows científicos em múltiplos ambientes conteinerizados. In *Anais do XXXIX Simpósio Brasileiro de Bancos de Dados*, pages 27–39, Porto Alegre, RS, Brasil, 2024. SBC.

[FR09a] Paola Festa and Mauricio G. C. Resende. An annotated bibliography of grasp part i: Algorithms. *International Transactions in Operational Research*, 16:1–24, 2009.

[FR09b] Paola Festa and Mauricio G. C. Resende. An annotated bibliography of grasppart ii: Applications. *International Transactions in Operational Research*, 16:131–172, 2009.

[FR18] Paola Festa and Mauricio G. C. Resende. GRASP. In Rafael Martí, Panos M. Pardalos, and Mauricio G. C. Resende, editors, *Handbook of Heuristics*, pages 465–488. Springer, 2018.

[GE10] Philip J. Guo and Dawson R. Engler. Towards practical incremental recomputation for scientists: An implementation for the python language. In Margo I. Seltzer and Wang-Chiew Tan, editors, *2nd Workshop on the Theory and Practice of Provenance, TaPP'10, San Jose, CA, USA, February 22, 2010*. USENIX Association, 2010.

[GE11] Philip J. Guo and Dawson Engler. Using automatic persistent memoization to facilitate data analysis scripting. In *Proceedings of the 2011 International Symposium on Software Testing and Analysis*, ISSTA '11, page 287–297, New York, NY, USA, 2011. Association for Computing Machinery.

[GP10] Michel Gendreau and Jean-Yves Potvin. *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*. Springer, 2nd edition, 2010.

[GSMB20] Alban Gaignard, Hala Skaf-Molli, and Khalid Belhajjame. Findable and reusable workflow data products: A genomic workflow case study. *Semantic Web*, 11(5):751–763, 2020.

[HdOP+19] Gaëtan Heidsieck, Daniel de Oliveira, Esther Pacitti, Christophe Pradal, François Tardieu, and Patrick Valduriez. Adaptive caching for data-intensive scientific workflows in the cloud. In Sven Hartmann, Josef Küng, Sharma Chakravarthy, Gabriele Anderst-Kotsis, A Min Tjoa, and Ismail Khalil, editors, *Database and Expert Systems Applications - 30th International Conference, DEXA 2019, Linz, Austria, August 26-29, 2019, Proceedings, Part II*, volume 11707 of *Lecture Notes in Computer Science*, pages 452–466. Springer, 2019.

[HdOP+20] Gaëtan Heidsieck, Daniel de Oliveira, Esther Pacitti, Christophe Pradal, François Tardieu, and Patrick Valduriez. Efficient execution of scientific workflows in the cloud through adaptive caching. *Trans. Large Scale Data Knowl. Centered Syst.*, 44:41–66, 2020.

[HdOP+21] Gaëtan Heidsieck, Daniel de Oliveira, Esther Pacitti, Christophe Pradal, François Tardieu, and Patrick Valduriez. Cache-aware scheduling of scientific workflows in a multisite cloud. *Future Gener. Comput. Syst.*, 122:172–186, 2021.

[JCD+13] Gideon Juve, Ann L. Chervenak, Ewa Deelman, Shishir Bharathi, Gaurang Mehta, and Karan Vahi. Characterizing and profiling scientific workflows. *FGCS*, 29(3):682–692, 2013.

[JV22] Michael A. Johnston and Vassils Vassiliadis. Towards an approximation-aware computational workflow framework for accelerating large-scale discovery tasks: Invited paper. In *Proceedings of the 2022 Workshop on Advanced Tools, Programming Languages, and PLatforms for Implementing and Evaluating Algorithms for Distributed Systems*, ApPLIED '22, page 7–14, New York, NY, USA, 2022. Association for Computing Machinery.

[KFMB17] Mayuresh Kunjir, Brandon Fain, Kamesh Munagala, and Shivnath Babu. Robus: Fair cache allocation for data-parallel workloads. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, page 219–234, New York, NY, USA, 2017. Association for Computing Machinery.

[LAB+09] Bertram Ludäscher, Ilkay Altintas, Shawn Bowers, Julian Cummings, Terence Critchlow, Ewa Deelman, David De Roure, Juliana Freire, Carole A. Goble, Matthew B. Jones, Scott Klasky, Timothy M. McPhillips, Norbert Podhorszki, Cláudio T. Silva, Ian J. Taylor, and Mladen A. Vouk. Scientific process automation and workflow management. In Arie Shoshani and Doron Rotem, editors, *Scientific Data Management - Challenges, Technology, and Deployment*, Chapman and Hall / CRC computational science series. CRC Press / Taylor & Francis, 2009.

[LCB+22] Zhuozhao Li, Ryan Chard, Yadu N. Babuji, Ben Galewsky, Tyler J. Skluzacek, Kirill Nagaitsev, Anna Woodard, Ben Blaiszik, Josh Bryan, Daniel S. Katz, Ian T. Foster, and Kyle Chard. $f$funcx: Federated function as a service for science. *IEEE Trans. Parallel Distributed Syst.*, 33(12):4948–4963, 2022.

[LPP+19] Ji Liu, Luis Pineda-Morales, Esther Pacitti, Alexandru Costan, Patrick Valduriez, Gabriel Antoniu, and Marta Mattoso. Efficient scheduling of scientific workflows using hot metadata in a multisite cloud. *IEEE Trans. Knowl. Data Eng.*, 31(10):1940–1953, 2019.

[LPVM15] Ji Liu, Esther Pacitti, Patrick Valduriez, and Marta Mattoso. A survey of data-intensive scientific workflow management. *J. Grid Comput.*, 13:457–493, 2015.

[LSV23] Jan Karel Lenstra, Vitaly A. Strusevich, and Milan Vlach. A historical note on the complexity of scheduling problems. *Oper. Res. Lett.*, 51(1):1–2, 2023.

[MB16] Alexander Moreno and Tucker Balch. Improving financial computation speed with full and subproblem memoization. *Concurr. Comput. Pract. Exp.*, 28(3):905–915, 2016.

[Mic68] DONALD Michie. "memo" functions and machine learning. *Nature*, 218(5136):19–22, Apr 1968.

[NNS21] Samaneh Sadat Mousavi Nik, Mahmoud Naghibzadeh, and Yasser Sedaghat. Task replication to improve the reliability of running workflows on the cloud. *Clust. Comput.*, 24(1):343–359, 2021.

[NSPdO21] André Nascimento, Vítor Silva, Aline Paes, and Daniel de Oliveira. An incremental reinforcement learning scheduling strategy for data-intensive scientific workflows in the cloud. *Concurr. Comput. Pract. Exp.*, 33(11), 2021.

[OdOO+11] Kary A. C. S. Ocaña, Daniel de Oliveira, Eduardo S. Ogasawara, Alberto M. R. Dávila, Alexandre A. B. Lima, and Marta Mattoso. Sciphy: A cloud-based workflow for phylogenetic analysis of drug targets in protozoan genomes. In Osmar Norberto de Souza, Guilherme P. Telles, and Mathew J. Palakal, editors, *Advances in Bioinformatics and Computational Biology - 6th Brazilian Symposium on Bioinformatics, BSB 2011, Brasilia, Brazil, August 10-12, 2011. Proceedings*, volume 6832 of *Lecture Notes in Computer Science*, pages 66–70. Springer, 2011.

[OdOV+11] Eduardo S. Ogasawara, Daniel de Oliveira, Patrick Valduriez, Jonas Dias, Fábio Porto, and Marta Mattoso. An algebraic approach for data-centric scientific workflows. *Proc. VLDB Endow.*, 4(12):1328–1339, 2011.

[OPP+16] Michal Owsiak, Marcin Plóciennik, Bartek Palak, Tomasz Zok, Cedric Reux, Luc Di Gallo, Mireille Schneider, Thomas Johnson, and Denis Kalupin. Running simultaneous kepler sessions for the parallelization of parametric scans and optimization studies applied to complex workflows. In Michelle Connolly, editor, *International Conference on Computational Science 2016, ICCS 2016, 6-8 June 2016, San Diego, California, USA*, volume 80 of *Procedia Computer Science*, pages 690–699. Elsevier, 2016.

[OPP+17] Michal Owsiak, Marcin Plóciennik, Bartek Palak, Tomasz Zok, Cedric Reux, Luc Di Gallo, Denis Kalupin, Thomas Johnson, and Mireille Schneider. Running simultaneous kepler sessions for the parallelization of parametric scans and optimization studies applied to complex workflows. *J. Comput. Sci.*, 20:103–111, 2017.

[PFVB15] Christophe Pradal, Christian Fournier, Patrick Valduriez, and Sarah Cohen Boulakia. Openalea: scientific workflows combining data analysis and simulation. In Amarnath Gupta and Susan L. Rathbun, editors, *Proceedings of the 27th International Conference on Scientific and Statistical Database Management, SSDBM '15, La Jolla, CA, USA, June 29 - July 1, 2015*, pages 11:1–11:6. ACM, 2015.

[PVK15] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information & Software Technology*, 64:1–18, 2015.

[QWCW19] Rawaa Qasha, Zhenyu Wen, Jacek Cała, and Paul Watson. Sharing and performance optimization of reproducible workflows in the cloud. *Future Generation Computer Systems*, 98:487–502, 2019.

[RR03] Mauricio G. C. Resende and Celso C. Ribeiro. Greedy randomized adaptive search procedures. In Fred W. Glover and Gary A. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 219–249. Kluwer / Springer, 2003.

[S+09] Rizos Sakellariou et al. Mapping workflows on grid resources: Experiments with the montage workflow. In *ERCIM W. Group on Grids*, pages 119–132, 2009.

[SBDP15] Juliana M.N. Silva, Cristina Boeres, Lúcia M.A. Drummond, and Artur A. Pessoa. Memory aware load balance strategy on a parallel branch-and-bound application. *Concurr. Comput. : Pract. Exper.*, 27(5):1122–1144, April 2015.

[SGdPJ+22] Murilo B. Stockinger, Marcos A. Guerine, Ubiratam de Paula Junior, Filipe Santiago, Yuri Frota, Isabel Rosseti, Alexandre Plastino, and Daniel de Oliveira. A provenance-based execution strategy for variant gpu-accelerated scientific workflows in clouds. *J. Grid Comput.*, 20(4):36, 2022.

[Suk21] Oleg V. Sukhoroslov. Toward efficient execution of data-intensive workflows. *J. Supercomput.*, 77(8):7989–8012, 2021.

[SXFT23] Jovan Stojkovic, Tianyin Xu, Hubertus Franke, and Josep Torrellas. Specfaas: Accelerating serverless applications with speculative function execution. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 814–827, 2023.

[Tal09] El-Ghazali Talbi. *Metaheuristics - From Design to Implementation*. Wiley, 2009.

[TCBPB17] François Tardieu, Llorenç Cabrera-Bosquet, Tony Pridmore, and Malcolm Bennett. Plant phenomics, from sensors to knowledge. *Current Biology*, 27(15):R770–R783, 2017.

[TdPF+17] Luan Teylo, Ubiratam de Paula, Yuri Frota, Daniel de Oliveira, and Lúcia M.A.Drummond. A hybrid evolutionary algorithm for task scheduling and data assignment of data-intensive scientific workflows on clouds. *Future Generations Computer Systems*, 76:1–17, 2017.

[TdPJF+17] Luan Teylo, Ubiratam de Paula Junior, Yuri Frota, Daniel de Oliveira, and Lúcia M. A. Drummond. A hybrid evolutionary algorithm for task scheduling and data assignment of data-intensive scientific workflows on clouds. *Future Gener. Comput. Syst.*, 76:1–17, 2017.

[TT14] Masahiro Tanaka and Osamu Tatebe. Disk cache-aware task scheduling for data-intensive and many-task workflow. In *2014 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 167–175, 2014.

[vdA18] Wil M. P. van der Aalst. Workflow patterns. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems, Second Edition*. Springer, 2018.

[VJM22] Vassilis Vassiliadis, Michael A. Johnston, and James L. McDonagh. Fast, transparent, and high-fidelity memoization cache-keys for computational workflows. In *2022 IEEE International Conference on Services Computing (SCC)*, pages 174–184, 2022.

[WHW10] Paul Watson, Hugo Hiden, and Simon Woodman. e-science central for CARMEN: science as a service. *Concurr. Comput. Pract. Exp.*, 22(17):2369–2380, 2010.

[YYL+13] Dong Yuan, Yun Yang, Xiao Liu, Wenhao Li, Lizhen Cui, Meng Xu, and Jinjun Chen. A highly practical approach toward achieving minimum data sets storage cost in the cloud. *IEEE Transactions on Parallel and Distributed Systems*, 24(6):1234–1244, 2013.

[ZB13] Mohsen Zohrevandi and Rida A. Bazzi. The bounded data reuse problem in scientific workflows. In *27th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2013, Cambridge, MA, USA, May 20-24, 2013*, pages 1051–1062. IEEE Computer Society, 2013.