

Distributed Caching of Scientific Workflows in Multisite Cloud

Gaëtan Heidsieck, Daniel de Oliveira, Esther Pacitti, Christophe Pradal, Francois Tardieu, Patrick Valduriez

▶ **To cite this version:**

Gaëtan Heidsieck, Daniel de Oliveira, Esther Pacitti, Christophe Pradal, Francois Tardieu, et al.. Distributed Caching of Scientific Workflows in Multisite Cloud. DEXA 2020 - 31st International Conference on Database and Expert Systems Applications, Sep 2020, Bratislava, Slovakia. pp.51-65, 10.1007/978-3-030-59051-2_4 . hal-02962579

HAL Id: hal-02962579
https://hal.inrae.fr/hal-02962579

Submitted on 9 Oct 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed Caching of Scientific Workflows in Multisite Cloud

Gaëtan Heidsieck[1], Daniel de Oliveira[4], Esther Pacitti[1], Christophe Pradal[1,2], François Tardieu[3], and Patrick Valduriez[1]

[1] Inria & LIRMM, Univ. Montpellier, France
[2] CIRAD & AGAP, Univ. Montpellier, France
[3] INRAE & LEPSE, Univ. Montpellier, France
[4] UFF, Niteroi, Brazil

Abstract. Many scientific experiments are performed using scientific workflows, which are becoming more and more data-intensive. We consider the efficient execution of such workflows in the cloud, leveraging the heterogeneous resources available at multiple cloud sites (geo-distributed data centers). Since it is common for workflow users to reuse code or data from other workflows, a promising approach for efficient workflow execution is to cache intermediate data in order to avoid re-executing entire workflows. In this paper, we propose a solution for distributed caching of scientific workflows in a multisite cloud. We implemented our solution in the OpenAlea workflow system, together with cache-aware distributed scheduling algorithms. Our experimental evaluation on a three-site cloud with a data-intensive application in plant phenotyping shows that our solution can yield major performance gains, reducing total time up to 42% with 60% of same input data for each new execution.

Keywords: Multisite cloud · Distributed Caching · Scientific Workflow · Workflow System · Workflow Scheduling.

1 Introduction

In many scientific domains, *e.g.*, bio-science [7], complex numerical experiments typically require many processing or analysis steps over huge datasets. They can be represented as scientific workflows, or workflows, for short, which facilitate the modeling, management and execution of computational activities linked by data dependencies. As the size of the data processed and the complexity of the computation keep increasing, these workflows become data-intensive [7], thus requiring high-performance computing resources.

The cloud is a convenient infrastructure for handling workflows, as it allows leasing resources at a very large scale and relatively low cost. In this paper, we consider the execution of a large workflow in a multisite cloud, *i.e.*, a cloud with geo-distributed cloud data centers (sites). Note that a multisite option is now well supported by all popular public clouds, *e.g.*, Microsoft Azure, Amazon EC2, and Google Cloud, which provide the capability of using multiple sites with a

single cloud account, thus avoiding the burden of using multiple accounts. The main reasons for using multiple cloud sites for data-intensive workflows is that they often exceed the capabilities of a single site, either because the site imposes usage limits for fairness and security, or simply because the datasets are too large. In scientific applications, there can be much heterogeneity in the storage and computing capabilities of the different sites, *e.g.*, on premise servers, HPC platforms from research organizations or federated cloud sites at the national level [4]. As an example in plant phenotyping, greenhouse platforms generate terabytes of raw data from plants, which are typically stored at data centers geographically close to the greenhouse to minimize data transfers. However, the computation power of those data centers may be limited and fail to scale when the analyses become more complex, such as in plant modeling or 3D reconstruction. Other computation sites are then required.

Most Scientific Workflow Management Systems (workflow systems) can execute workflows in the cloud [12]. Some examples are Swift/T, Pegasus, SciCumulus, Kepler and OpenAlea [9]. Our work is based on OpenAlea [14], which is widely used in plant science for simulation and analysis. Most existing systems use naive or user-based approaches to distribute the tasks across sites. The problem of scheduling a workflow execution over a multisite cloud has started to be addressed in [11], using performance models to predict the execution time on different resources. In [10], we proposed a solution based on multi-objective scheduling and a single site virtual machine provisioning approach, assuming homogeneous sites, as in public cloud.

Since it is common for workflow users to reuse code or data from other workflows [5], a promising approach for efficient workflow execution is to cache intermediate data in order to avoid re-executing entire workflows. Furthermore, a user may need to re-execute a workflow many times with different sets of parameters and input data depending on the previous results generated. Fragments of the workflow, *i.e.* a subset of the workflow activities and dependencies, can often be reused. Another important benefit of caching intermediate data is to make it easy to share with other research teams, thus fostering new analyses at low cost.

Caching has been supported by some workflow systems, *e.g.*, Kepler, VisTrails and OpenAlea. Kepler [1] provides a persistent cache on the cloud, but at a single site, and does not support multisite. VisTrails [3] provides a persistent cache, but only for local execution on a personal desktop. In [6], we proposed an adaptive caching method for OpenAlea that automatically determines the most suited intermediate data to cache, taking into account workflow fragments, but only in the case of a single cloud site. Another interesting single site method, also exploiting workflow fragments, is to compute the ratio between re-computation cost and storage cost to determine what intermediate data should be stored [16]. All these methods are single site (centralized). The only distributed caching method for workflow execution in a multisite cloud we are aware of is restricted to hot metadata (frequently accessed metadata) [8], ignoring intermediate data.

Caching data in a multisite cloud with heterogeneous sites is much more complex. In addition to the trade-off between re-computation and storage cost at single sites, there is the problem of site selection for placing cached data. The problem is more difficult than data allocation in distributed databases [13], which deals only with well-defined base data, not intermediate data produced by tasks. Furthermore, the scheduling of workflow executions must be cache-aware, *i.e.*, exploit the knowledge of cached data to decide between reusing and transferring cached data versus re-executing the workflow fragments.

In this paper, we propose a distributed solution for caching of scientific workflows in a multisite cloud. Based on a distributed and parallel architecture [13], composed of heterogeneous sites (including on premise servers and shared-nothing clusters), we propose algorithms for adaptive caching, cache site selection and dynamic workflow scheduling. We implemented our caching solution in OpenAlea, together with a multisite scheduling algorithm. Based on a real data-intensive application in plant phenotyping, we provide an extensive experimental evaluation using a cloud with three heterogeneous sites.

This paper is organized as follows. Section 2 presents our real use case in plant phenotyping. Section 3 introduces our workflow system architecture in multisite cloud. Section 4 describes our cache management solution. Section 5 gives our experimental evaluation. Finally, Section 6 concludes.

2 Use Case in Plant Phenotyping

In this section, we introduce a real use case in plant phenotyping that will serve as motivation for the work and basis for the experimental evaluation. In the last decade, high-throughput phenotyping platforms have emerged to allow for the acquisition of quantitative data on thousands of plants in well-controlled environmental conditions. For instance, the seven facilities of the French Phenome project [5] produce each year 200 Terabytes of data, which are various (images, environmental conditions and sensor outputs), multiscale and originate from different sites. Analyzing such massive datasets is an open, yet important, problem for biologists [15].

The Phenomenal workflow ([2]) has been developed in OpenAlea to analyze and reconstruct the geometry and topology of thousands of plants through time in various conditions. It is composed of nine fragments such as image binarization, 3D volume reconstruction, organ segmentation or intercepted light simulation. Different users can conduct different biological analyses by reusing some workflow fragments on the same dataset to test different hypotheses [6]. To save both time and resources, they want to reuse the intermediate results that have already been computed rather than recompute them from scratch.

The raw data comes from the Phenoarch platform, which has a capacity of 1,680 plants within a controlled environment (*e.g.*, temperature, humidity, irrigation) and automatic imaging through time. The total size of the raw image

[5] `https://www.phenome-emphasis.fr/phenome_eng/`

dataset for one experiment is 11 Terabytes. To limit data movement, the raw data is stored at a server near to the experimental platform, with both data storage and computing resources. However, these computing resources are not enough to process a full experiment in a relatively short time. Thus, scientists who need to do a full experiment will execute the Phenomenal workflow at a more powerful site by transferring the raw data for each new analysis.

In this Phenomenal use case, the cloud is composed of heterogeneous sites, with both on premise servers close to the experimental platform and other more powerful cloud sites. The on premise server has high storage capacity and hosts the raw data. Other sites are used to computational intensive executions, with high-performance computing resources. On premise servers are used locally to execute some Phenomenal fragments that do not require powerful resources. In this case, one has to choose between transferring the raw data or some intermediate data to a powerful site or re-executing some fragments locally before transferring intermediate data. The trade-off between data re-computation and data transfer is complex in a multisite cloud with much heterogeneity. In particular, one needs to pay attention to cached data placement, so as to avoid bottlenecks on the most used intermediate data.

3 Multisite Cloud Workflow System Architecture

In this section, we present our workflow system architecture that integrates caching and reuse of intermediate data in a multisite cloud. We motivate our design decisions and describe our architecture in terms of nodes and components (see Figure 1), which are involved in the processing of workflows.

Our architecture capitalizes on the latest advances in distributed and parallel data management to offer performance and scalability [13]. We consider a distributed cloud architecture with on premise servers, where raw data is produced, *e.g.*, by a phenotyping experimental platform in our use case, and remote sites, where the workflow is executed. The remote sites (data centers) are shared-nothing clusters, *i.e.*, clusters of server machines, each with processor, memory and disk. We adopt shared-nothing as it is the most scalable and cost-effective architecture for big data analysis.

In the cloud, metadata management has a critical impact on the efficiency of workflow scheduling as it provides a global view of data location, *e.g.*, at which nodes some raw data is stored, and enables task tracking during execution [8]. We organize the metadata in three repositories: catalog, provenance database and cache index. The catalog contains all information about users (access rights, etc.), raw data location and workflows (code libraries, application code). The provenance database captures all information about workflow execution. The cache index contains information about tasks and cache data produced, as well as the location of files that store the cache data. Thus, the cache index itself is small (only file references) and the cached data can be managed using the underlying file system. A good solution for implementing these metadata repositories is a

key-value store, such as Cassandra[6], which provides efficient key-based access, scalability and fault-tolerance through replication in a shared-nothing cluster.

The raw data (files) are initially produced and stored at some cloud sites, *e.g.*, in our use case, at the phenotyping platform. During workflow execution, the intermediate data is generated and consumed at one site's node in memory. It gets written to disk when it must be transferred to another node (potentially at the same site), or when explicitly added to the cache. The cached data (files) can later be replicated at other sites to minimize data transfers.

We extend the workflow system architecture proposed in [9] for single site. It is composed of six modules: workflow manager, global scheduler, local scheduler, task manager, data manager and metadata manager, to support both execution and intermediate data caching in a multisite cloud. The workflow manager provides a user interface for workflow definition and processing. Before workflow execution, the user selects a number of virtual machines (VMs), given a set of possible instance formats, *i.e.*, the technical characteristics of the VMs, deployed on each site's nodes. When a workflow execution is started, the workflow manager simplifies the workflow by removing some workflow fragments and partitions depending on the raw input data and the cached data (see Section 4). The global scheduler uses the metadata (catalog, provenance database, and cache index) to schedule the workflow fragments of the simplified workflow. The VMs on each site are then initialized, *i.e.*, the programs required for the execution of the tasks are installed and all parameters are configured. The local scheduler schedules the workflow fragments received on its VMs.

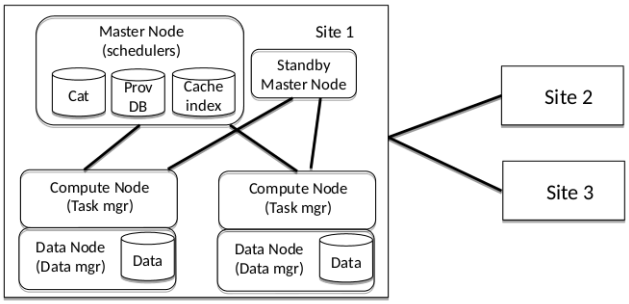


Fig. 1: Multisite Workflow System Architecture

The data manager module handles data transfers between sites during execution (for both newly generated intermediate data and cached data) and manages cache storage and replication. At a single site, data storage is distributed between nodes. Finally, the task manager (on each VM) manages the execution of fragments on the VMs at each site. It exploits the provenance metadata to decide whether or not the task's output data should be placed in the cache, based on

[6] `https://cassandra.apache.org`

the cache provisioning algorithm described in Section 4. Local scheduling and execution can be performed as in [6].

Figure 1 shows how these components are involved in workflow processing, using the traditional master-worker model. In this architecture, we consider two types of cloud sites, *i.e.*, coordinator and participant. The relationship between the site is also based on the master-worker model, the coordinator site, managing the participant sites. The workflow manager and the global scheduler modules are implemented on the coordinator site. The remaining modules are implemented on all sites.

At each site, there are three kinds of nodes: master, compute and data nodes, which are mapped to cluster nodes at configuration time, *e.g.* using a cluster manager like Yarn (`http://hadoop.apache.org`). There is one active master node per site. There is also a standby node to deal with master node failure. The master nodes are the only ones to communicate across sites. The local scheduler and metadata management modules are implemented on the master node, which manages communication, metadata and scheduling. The master nodes are responsible for transferring data between sites during execution.

4 Multisite Cache-aware Workflow Execution

In this section, we present in more details how the global scheduler performs multisite cache-aware workflow execution. In particular, the global scheduler must decide which data to cache (cache data selection) and where (cache site selection), and where to execute workflow fragments (execution site selection). Since these decisions are not independent, we propose a cost function to make a global decision, based on the cost components for individual decisions. We start by giving an overview of distributed workflow execution. Then, we present the methods and cost functions for cache data selection, cache site selection and execution site selection. Finally, we introduce our cost function for the global decision.

4.1 Distributed Workflow Execution Overview

We consider a multisite cloud with a set of sites $S=\{s_1, ..., s_n\}$. A workflow $W(A, D)$ is a a directed acyclic graph (DAG) of computational activities A and their data dependencies D. A task t is the instantiation of an activity during execution with specific associated input data. A fragment f of an instantiated workflow is a subset of tasks and their dependencies.

The execution of a workflow $W(A, D)$ in S starts at a coordinator site s_c and proceeds in three main steps:

1. The global scheduler at s_c simplifies and partitions the workflow into fragments. Simplification uses metadata to decide whether a task can be replaced by corresponding cached data references. Partitioning uses the dependencies in D to produce fragments.

2. For each fragment, the global scheduler at s_c computes a cost function to make a global decision on which data to cache where, and on which site to execute. Then, it triggers fragment execution and cache placement at the selected sites.
3. At each selected site, the local scheduler performs the execution of its received fragments using its task manager (to execute tasks) and data manager (to transfer the required input data). It also applies the decision of the global scheduler on storing new intermediate data into the cache.

We introduce basic cost functions to reflect data transfer and distributed execution. The time to transfer some data d from site s_i to site s_j, noted $T_{tr}(d, s_i, s_j)$, is defined by

$$T_{tr}(d, s_i, s_j) = \frac{Size(d)}{TrRate(s_i, s_j)} \tag{1}$$

where $TrRate(s_i, s_j)$ is the transfer rate between s_i and s_j.

The time to transfer input and cached data, $In(f)$ and $Cached(f)$ respectively, to execute a fragment f at site s_i is $T_{input}(f, s_i)$:

$$T_{input}(f, s_i) = \sum_{s_j}^{S} (T_{tr}(In(f), s_j, s_i) + T_{tr}(Cached(f), s_j, s_i)) \tag{2}$$

The time to compute a fragment f at site s, noted $T_{compute}(f, s)$, can be estimated using Amdahl's law [17]:

$$T_{compute}(f, s) = \frac{(\frac{\alpha}{n} + (1 - \alpha)) * W(f)}{CPU_{perf}(s)} \tag{3}$$

where $W(f)$ is the workload for the execution of f, $CPU_{perf}(s)$ is the average computing performance of the CPUs at site s and n is the number of CPUs at site s. We suppose that the local scheduler may parallelize task executions. Therefore, α represents the percentage of the workload that can be executed in parallel.

The expected waiting time to be able to execute a fragment at site s is noted $T_{wait}(s)$, which is the minimum expected time for s to finish executing the fragments in its queue.

The time to transfer the intermediate data generated by fragment f at site s_i to site s_j, noted $T_{write}(Output(f), s_i, s_j)$, is defined by:

$$T_{write}(Output(f), s_i, s_j) = T_{tr}(Output(f), s_i, s_j) \tag{4}$$

where $Output(f)$ is the data generated by the execution of f.

4.2 Cache Data Selection

To determine what new intermediate data to cache, we consider two different methods: greedy and adaptive. Greedy data selection simply adds all new data

to the cache. Adaptive data selection extends our method proposed in [6] to multisite cloud. It achieves a good trade-off between the cost saved by reusing cached data and the cost incurred to feed the cache.

To determine if it is worth adding some intermediate data $Output(f)$ at site s_j, we consider the trade-off between the cost of adding this data to the cache and the potential benefit if this data was reused. The cost of adding the data to site s_j is the time to transfer the data from the site where it was generated. The potential benefit is the time saved from loading the data from s_j to the site of computation instead of re-executing the fragment. We model this trade-off with the ratio between the cost and benefit of the cache, noted $p(f, s_i, s_j)$, which can be computed from equations 2, 3 and 4,

$$p(f, s_i, s_j) = \frac{T_{write}(Output(f), s_i, s_j)}{T_{input}(f, s_i) + T_{compute}(f, s_i) - T_{tr}(Output(f), s_j, s_i)} \tag{5}$$

In the case of multiple users, the probability that $Output(f)$ will be reused or the number of times fragment f will be re-executed is not known when the workflow is executed. Thus, we introduce a threshold $Threshold$ (computed by the user) as the limit value to decide whether a fragment output will be added to the cache. The decision on whether $Output(f)$ generated at site s_i is stored at site s_j can be expressed by

$$\epsilon_{i,j} = \begin{cases} 1, & \text{if } p(f, s_i, s_j) < Threshold. \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

4.3 Cache Site Selection

Cache site selection must take into account the data transfer cost and the heterogeneity of computing and storage resources. We propose two methods to balance either storage load (*bStorage*) or computation load (*bCompute*) between sites. The *bStorage* method allows preventing bottlenecks when loading cached data. To assess this method at any site s, we use a load indicator, noted $L_{bStorage}(s)$, which represents the relative storage load as the ratio between the storage used for the cached data ($Storage_{used}(s)$) and the total storage ($Storage_{total}(s)$).

$$L_{bStorage}(s) = \frac{Storage_{used}(s)}{Storage_{total}(s)} \tag{7}$$

The *bCompute* method balances the cached data between the most powerful sites, *i.e.*, with more CPUs, to prevent computing bottlenecks during execution. Using the knowledge on the sites' computing resources and usage, we use a load indicator for each site s, noted $L_{bCompute}(s)$, based on CPUs idleness ($CPU_{idle}(s)$) versus total CPU capacity ($CPU_{total}(s)$).

$$L_{bCompute}(s) = \frac{1 - CPU_{idle}(s)}{CPU_{total}(s)} \tag{8}$$

The load of a site s, depending on the method used, is represented by $L(s)$, ranging between 0 (empty load) and 1 (full). Given a fragment f executed at site s_i, and a set of sites s_j with enough storage for $Output(f)$, the best site s^* to add $Output(f)$ to its cache can be obtained using Equation 1 (to include transfer time) and Equation 6 (to consider multiple users),

$$s^*(f)_{s_i} = \underset{s_j}{\text{argmax}}(\epsilon_{i,j} * \frac{(1 - L(s_j))}{T_{write}(Output(f), s_i, s_j)}) \tag{9}$$

4.4 Execution Site Selection

To select an execution site s for a fragment f, we need to estimate the execution time for f as well as the time to feed the cache with the result of f. The execution time f at site s ($T_{execute}(f, s)$) is the sum of the time to transfer input and cached data to s, the time to get computing resources and the time to compute the fragment. It is obtained using Equations 2 and 3.

$$T_{execute}(f, s) = T_{input}(f, s) + T_{compute}(f, s) + T_{wait}(s) \tag{10}$$

Given a fragment f executed at site s_i and its intermediate data $Output(f)$, the time to write $Output(f)$ to the cache ($T_{feed_cache}(f, s_i, s_j)$) can be defined as:

$$T_{feed_cache}(f, s_i, s_j, \epsilon_{i,j}) = \epsilon_{i,j} * T_{write}(Output(f), s_i, s_j) \tag{11}$$

where s_j is given by Equation 9.

4.5 Global Decision

At Step 2 of workflow execution, for each fragment f, the global scheduler must decide on the best combination of individual decisions regarding cache data, cache site, and execution site. These individual decisions depend on each other. The decision on cache data depends on the site where the data is generated and the site where it will be stored. The decision on cache site depends on the site where the data is generated and the decision of whether or not the data will be cached. Finally, the decision on execution site depends on what data will be added to the cache and at which site. Using Equations 10 and 11, we can estimate the total time (T_{total}) for executing a fragment f at site s_i and adding its intermediate data to the cache at another site s_j:

$$T_{total}(f, s_i, s_j, \epsilon_{i,j}) = T_{execute}(f, s_i) + T_{feed_cache}(f, s_i, s_j, \epsilon_{i,j}) \tag{12}$$

Then, the global decision for cache data ($\epsilon(f)$), cache site (s^*_{cache}) and execution site (s^*_{exec}) is based on minimizing the following equation for the n^2 pairs of sites s_i and s_j

$$(s^*_{exec}, s^*_{cache}, \epsilon(f)) = \underset{s_i, s_j}{\text{argmin}}(T_{total}(f, s_i, s_j, \epsilon_{i,j})) \tag{13}$$

This decision is done by the coordinator site at before each fragment execution and only takes into account the cloud site's status at that time. Note that s^*_{exec}, s^*_{cache} can be the coordinator site and can be the same site.

5 Experimental Evaluation

In this section, we first present our experimental setup, which features a heterogeneous multisite cloud with multiple users who re-execute part of the workflow. Then, we compare the performance of our multisite cache scheduling method against two baseline methods. We end the section with concluding remarks.

5.1 Experimental Setup

Our experimental setup includes a multisite cloud, with three sites in France, a workflow implementation and an experimental dataset. *Site 1* in Montpellier is a server close to the Phenoarch phenotyping platform. It has the smallest number of CPUs and largest amount of storage among the sites. The raw data is stored at this site. *Site 2* is the coordinator site, located in Lille. *Site 3*, located in Lyon, has the largest number of CPUs and the smallest amount of storage.

To model site heterogeneity in terms of storage and CPU resources, we use heterogeneity factor H in three configurations: $H = 0$, $H = 0.3$ and $H = 0.7$. For the three sites altogether, the total number of CPUs is 96 and the total storage on disk for intermediate data is 180 GB (The raw data is stored on an additional node at Site 1). On each site, several nodes are instantiated for the executions, they have a determined number of CPUs from 1, 2, 4, 8 or 16 CPUs. The available disk size for each node is limited by implementation. With $H = 0$ (homogeneous configuration), each site has 32 CPUs (two 16 CPUs nodes) and 60 GB (30 GB each). With $H = 0.3$, we have 22 CPUs and 83 GB for Site 1, 30 CPUs and 57 GB for Site 2 and 44 CPUs and 40 GB for Site 3. With $H = 0.7$ (most heterogeneous configuration), we have 6 CPUs and 135 GB for Site 1, 23 CPUs and 35 GB for Site 2 and 67 CPUs and 10 GB for Site 3.

The input dataset for the Phenomenal workflow is produced by the Phenoarch platform (see Section 2). Each execution of the workflow is performed on a subset of the input dataset, *i.e.* 200 GB of raw data, which represents the execution of 15,000 tasks. For each user, 60% of the raw data is reused from previous executions. Thus each execution requires only 40% of new raw data. For the first execution, no data is available in the cache.

We implemented our cache-aware scheduling method, which we call *cacheA*, in OpenAlea and deployed it at each site using the Conda multi-OS package manager. The metadata distributed database is implemented using Cassandra. Communication between the sites is done using the protocol library ZeroMQ. Data transfer between sites is done through SSH. We have also implemented two baseline methods, *Sgreedy* and *Agreedy*, based on the *SiteGreedy* and *ActGreedy* methods described in [10], respectively. The *Sgreedy* method extends *SiteGreedy*, which schedules each workflow fragment at a site that is available

for execution, with our cache data/site selection methods. Similarly, the *Agreedy* method extends *ActGreedy*, which schedules each workflow fragment at a site that minimizes a cost function based on execution time and input data transfer time, with our cache data/site selection methods. These baseline methods perform execution site selection followed by cache data/site selection while *CacheA* makes a global decision.

5.2 Experiments

We compare *CacheA* with the two baseline methods in terms of execution time and amount of data transferred. We define total time as execution time plus transfer time. In experiment 1, we consider a workflow execution with caching or without. In Experiment 2, we consider multiple users who execute the same workflow on similar input data, where 60% of the data is the same. In Experiment 3, we consider different heterogeneous configurations for one workflow execution.

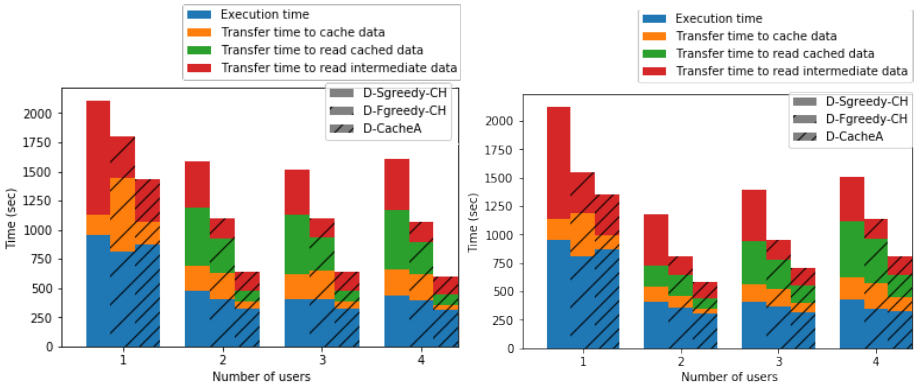
Experiment 1: with caching. In this basic experiment, we compare two workflow executions: with caching, using *CacheA* and *bStorage*; and without caching, using *ActGreedy*. We consider one re-execution of the workflow on different input datasets, from 0% to 60% of same reused data.

CacheA outperforms *ActGreedy* from 20% of reused data. Below 20%, the overhead of caching outweighs its benefit. For instance, with no reuse (0%), the total time with *CacheA* is 16% higher than with *ActGreedy*. But with 30%, it is 11% lower, and with 60%, it is 42% lower.

Experiment 2: multiple users. Figure 2 shows the total time of the workflow for the three scheduling methods, four users, $H = 0.7$ and our two cache site selection methods: (a) *bStorage*, and (b) *bCompute*.

Let us first analyze the results in Figure 2.a (*bStorage* method). For the first user execution, *CacheA* outperforms *Sgreedy* in terms of execution time by 8% and in terms of data and intermediate data transfer times by 51% and 63%, respectively. The reason *Sgreedy* is slower is that it schedules some compute-intensive fragments at Site 1, which has the lowest computing resources. Furthermore, it does not consider data placement and transfer time when scheduling fragments.

Again for the first user execution, *CacheA* outperforms *Agreedy* in terms of total time by 24%, when considering data transfer time to the cache. However, *CacheA* execution time is a bit slower (by 9%). The reason that *Agreedy* is slower in terms of total time is that it does not take into account the placement of the cached data, which leads to larger amounts (by 67%) of cache data to transfer. For other users' executions (when cached data exists), *CacheA* outperforms *Sgreedy* in terms of execution time by 29%, and for the fourth user execution, by 20%. This is because *CacheA* better selects the cache site in order to reduce the execution time of the future re-executions. In addition, *CacheA* balances the cached data and computations. It outperforms *Sgreedy* and *Agreedy* in terms of



(a) *bStorage method* (b) *bCompute method*

Fig. 2: Total times for multiple users (60% of same raw data per user) for three scheduling methods (*Sgreedy*, *Agreedy* and *CacheA*).

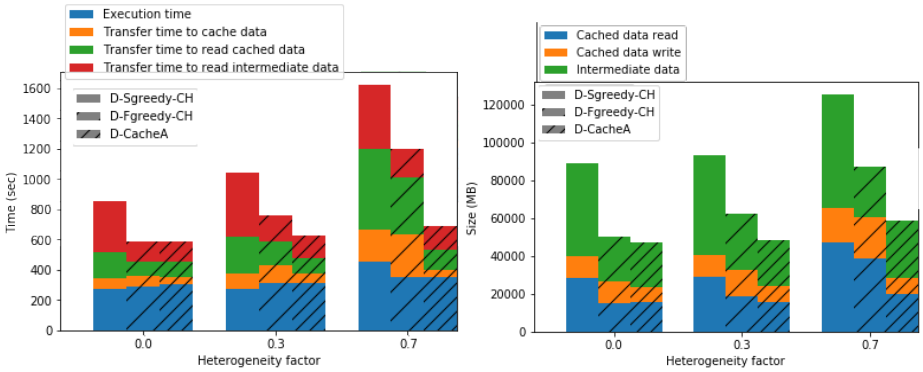
intermediate data transfer times (by 59% and 15%, respectively), and cache data transfer times (by 82% and 74%, respectively).

Overall, *CacheA* outperforms *Sgreedy* and *Agreedy* in terms of total times by 61% and 43%, respectively. The workflow fragments are not necessarily scheduled to the site with shortest execution time, but to the site that minimizes overall total time. Considering the multiuser perspective, *CacheA* outperforms baseline methods, reducing the total time for each new user (up to 6% faster for the fourth user compared to the second).

Let us now consider Figure 2.b (*bCompute* method). For the first user execution, *CacheA* outperforms *Sgreedy* and *Agreedy* in terms of total time by 36% and 10% respectively. *bCompute* stores the cache data on the site with most idle CPUs, which is often the site with the most CPUs. This leads the cached data to be stored close to where it is generated, thus reducing data transfers when adding data to the cache. For the second user, *CacheA* outperforms *Sgreedy* and *Agreedy* in terms of total time by 46% and 21% respectively. The cached data generated by the first user is stored on the sites with more available CPUs, which minimizes the intermediate and reused cached data transfers. From the third user, the storage at some site gets full, *i.e.* for the third user's execution, Site 3 storage is full and from the fourth user's execution, Site 2 storage is full. Thus, the performance of the three scheduling methods decreases due to higher cache data transfer times. Yet, *CacheA* still outperforms *Sgreedy* and *Agreedy* in terms of total time by 49% and 25% respectively.

Experiment 3: cloud site heterogeneity. We now compare the three methods in the case of heterogeneous sites by considering the amount of data transferred and execution time. In this experiment (see Figure 3), we consider only one user who executes the workflow and that previous executions with 60% of the

same raw data have generated some cached data. We use the *bStorage* method for cache site selection.



(a) Execution time (b) Amount of data transferred

Fig. 3: Execution times and amounts of data transferred for one user (60% of same raw data used), on heterogeneous sites with three scheduling methods (*Sgreedy*, *Agreedy* and *CacheA*).

Figure 3 shows the execution times and the amount of data transferred using the three scheduling methods in case of heterogeneous sites. With homogeneous sites ($H = 0$), the three methods have almost the same execution time. *CacheA* outperforms *Sgreedy* in terms of amount of intermediate data transferred and total time by 44% and 26%, respectively. *CacheA* has execution time similar to *Agreedy* (3.1% longer). The cached data is balanced as the three sites have same storage capacities. Thus, the total times of *CacheA* and *Agreedy* are almost the same.

With heterogeneous sites ($H > 0$), the sites with more CPUs have less available storage but can execute more tasks, which leads to a larger amount of intermediate and cached data transferred between the sites. For $H = 0.3$, *CacheA* outperforms *Sgreedy* and *Agreedy* in terms of total time (by 40% and 18%, respectively) and amount of data transferred (by 47% and 21%, respectively).

With $H = 0.7$, *CacheA* outperforms *Sgreedy* and *Agreedy* in terms of total time (by 58% and 42%, respectively) and in terms of amount of data transferred (by 55% and 31%, respectively). *CacheA* is faster because its scheduling leads to a smaller amount of cached data transferred when reused (48% smaller than *Agreedy*) and added to the cache (62% smaller than *Agreedy*).

5.3 Concluding Remarks

Our cache-aware scheduling method *CacheA* always outperforms the two baseline methods (which also benefit from our cache/data selection method), both in the case of multiple users and heterogeneous sites.

The first experiment (with caching) shows that storing and reusing cached data becomes beneficial when 20% or more of the input data is reused. The second experiment (multiple users) shows that *CacheA* outperforms *Sgreedy* and *Agreedy* in terms of total time by up to 61% and 43%, respectively. It also shows that, with increasing numbers of users, the performance of the three scheduling methods decreases due to higher cache data transfer times. The third experiment (heterogeneous sites) shows that *CacheA* adapts well to site heterogeneity, minimizing the amount of cached data transferred and thus reducing total time. It outperforms *Sgreedy* and *Agreedy* in terms of total time by up to 58% and 42% respectively.

Both cache site selection methods *bCompute* and *bStorage* have their own advantages. *bCompute* outperforms *bStorage* in terms of data transfer time by 13% for the first user and up to 17% for the second user. However, it does not scale with the number of users, and the limited storage capacities of Site 2 and 3 lead to a bottleneck. On the other hand, *bStorage* balances the cached data among sites and prevents the bottleneck when accessing the cached data, thus reducing re-execution times. In summary, *bCompute* is best suited for compute-intensive workflows that generate smaller intermediate datasets while *bStorage* is best suited for data-intensive workflows where executions can be performed at the site where the data is stored.

6 Conclusion

In this paper, we proposed a solution for distributed caching of scientific workflows in a cloud with heterogeneous sites (including on premise servers and shared-nothing clusters). Based on a distributed and parallel architecture, we proposed algorithms for adaptive caching, cache site selection and dynamic workflow scheduling. We implemented our solution in OpenAlea, together with a multisite scheduling algorithm. Using a real data-intensive application in plant phenotyping (Phenomenal), our extensive experimental evaluation using a cloud with three heterogeneous sites shows that our solution can yield major performance gains. In particular, it reduces much execution times and data transfers, compared to two baseline scheduling methods (which also use our cache/data selection method).

7 Acknowledgments

This work was supported by the #DigitAg French initiative (www.hdigitag.fr), the SciDISC and HPDaSc Inria associated teams with Brazil, the Phenome-Emphasis project (ANR-11-INBS-0012) and IFB (ANR-11-INBS-0013) from the Agence Nationale de la Recherche and the France Grille Scientific Interest Group.

References

1. Altintas, I., Barney, O., Jaeger-Frank, E.: Provenance collection support in the kepler scientific workflow system. In: International Provenance and Annotation Workshop. pp. 118–132 (2006)
2. Artzet, S., Brichet, N., Chopard, J., Mielewczik, M., Fournier, C., Pradal, C.: Openalea.phenomenal: A workflow for plant phenotyping (Sep 2018). https://doi.org/10.5281/zenodo.1436634
3. Callahan, S.P., Freire, J., Santos, E., Scheidegger, C.E., Silva, C.T., Vo, H.T.: Vistrails: visualization meets data management. In: ACM SIGMOD Int. Conf. on Management of Data (SIGMOD). pp. 745–747 (2006)
4. Crago, S., Dunn, K., Eads, P., Hochstein, L., Kang, D.I., Kang, M., Modium, D., Singh, K., Suh, J., Walters, J.P.: Heterogeneous cloud computing. In: 2011 IEEE International Conference on Cluster Computing. pp. 378–385. IEEE (2011)
5. Garijo, D., Alper, P., Belhajjame, K., Corcho, O., Gil, Y., Goble, C.: Common motifs in scientific workflows: An empirical analysis. Future Generation Computer Systems (FGCS) **36**, 338–351 (2014)
6. Heidsieck, G., de Oliveira, D., Pacitti, E., Pradal, C., Tardieu, F., Valduriez, P.: Adaptive caching for data-intensive scientific workflows in the cloud. In: Int. Conf. on Database and Expert Systems Applications (DEXA). pp. 452–466 (2019)
7. Kelling, S., Hochachka, W.M., Fink, D., Riedewald, M., Caruana, R., Ballard, G., Hooker, G.: Data-intensive science: a new paradigm for biodiversity studies. BioScience **59**(7), 613–620 (2009)
8. Liu, J., Morales, L.P., Pacitti, E., Costan, A., Valduriez, P., Antoniu, G., Mattoso, M.: Efficient scheduling of scientific workflows using hot metadata in a multisite cloud. IEEE Trans. on Knowledge and Data Engineering pp. 1–20 (2018)
9. Liu, J., Pacitti, E., Valduriez, P., Mattoso, M.: A survey of data-intensive scientific workflow management. Journal of Grid Computing **13**(4), 457–493 (2015)
10. Liu, J., Pacitti, E., Valduriez, P., de Oliveira, D., Mattoso, M.: Multi-objective scheduling of scientific workflows in multisite clouds. Future Generation Computer Systems(FGCS) **63**, 76–95 (2016)
11. Maheshwari, K., Jung, E., Meng, J., Vishwanath, V., Kettimuthu, R.: Improving multisite workflow performance using model-based scheduling. In: IEEE nt. Conf. on Parallel Processing (ICPP). pp. 131–140 (2014)
12. de Oliveira, D., Baião, F.A., Mattoso, M.: Towards a taxonomy for cloud computing from an e-science perspective. In: Cloud Computing. Computer Communications and Networks., pp. 47–62. Springer (2010)
13. Özsu, M.T., Valduriez, P.: Principles of Distributed Database Systems, Fourth Edition. Springer (2020)
14. Pradal, C., Fournier, C., Valduriez, P., Cohen-Boulakia, S.: Openalea: scientific workflows combining data analysis and simulation. In: Int. Conf. on Scientific and Statistical Database Management (SSDBM). pp. 11:1–11:6 (2015)
15. Tardieu, F., Cabrera-Bosquet, L., Pridmore, T., Bennett, M.: Plant phenomics, from sensors to knowledge. Current Biology **27**(15), R770–R783 (2017)
16. Yuan, D., Yang, Y., Liu, X., Li, W., Cui, L., Xu, M., Chen, J.: A highly practical approach toward achieving minimum data sets storage cost in the cloud. IEEE Trans. on Parallel and Distributed Systems **24**(6), 1234–1244 (2013)
17. Zhang, J., Luo, J., Dong, F.: Scheduling of scientific workflow in non-dedicated heterogeneous multicluster platform. Journal of Systems and Software **86**(7), 1806–1818 (2013)