Efficient Scheduling of Scientific Workflows using Hot Metadata in a Multisite Cloud

Ji Liu, Luis Pineda, Esther Pacitti, Alexandru Costan, Patrick Valduriez, Gabriel Antoniu, Marta Mattoso

Efficient Scheduling of Scientific Workflows using Hot Metadata in a Multisite Cloud

Ji Liu[*], Luis Pineda[†], Esther Pacitti[‡], Alexandru Costan[§], Patrick Valduriez[¶], Gabriel Antoniu[‖] and Marta Mattoso[**]

Abstract

Large-scale, data-intensive scientific applications are often expressed as scientific workflows (SWfs). In this paper, we consider the problem of efficient scheduling of a large SWf in a multisite cloud, *i.e.* a cloud with geo-distributed cloud data centers (sites). The reasons for using multiple cloud sites to run a SWf are that data is already distributed, the necessary resources exceed the limits at a single site, or the monetary cost is lower. In a multisite cloud, metadata management has a critical impact on the efficiency of SWf scheduling as it provides a global view of data location and enables task tracking during execution. Thus, it should be readily available to the system at any given time. While it has been shown that efficient metadata handling plays a key role in performance, little research has targeted this issue in multisite cloud. In this paper, we propose to identify and exploit *hot metadata* (frequently accessed metadata) for efficient SWf scheduling in a multisite cloud, using a distributed approach. We implemented our approach within a scientific workflow management system, which shows that our approach reduces the execution time of highly parallel jobs up to 64% and that of the whole SWfs up to 55%.

Keywords: Metadata management, hot metadata, multisite cloud, scientific workflows, geo-distributed applications

[*]jiliuwork@gmail.com, Inria, LIRMM and University of Montpellier, France

[†]luis.pineda-morales@inria.fr, Inria, CNRS, IRISA/INSA and University of Rennes, France

[‡]esther.pacitti@lirmm.fr, LIRMM, Inria and University of Montpellier, France

[§]alexandru.costan@irisa.fr, IRISA/INSA Rennes, France

[¶]patrick.valduriez@inria.fr, Inria, LIRMM and University of Montpellier, France

[‖]gabriel.antoniu@inria.fr, Inria, CNRS, IRISA/INSA and University of Rennes, France

[**]marta@cos.ufrj.br, COPPE/UFRJ, Brazil

1 Introduction

Many scientific applications today process large amounts of data, in the order of Petabytes. As the size of the data increases, so do the requirements for computing resources. Such data-intensive applications can be expressed as Scientific Workflows (SWfs). A SWf is an assembly of scientific data processing jobs, such as loading input data, data processing, data analysis, and aggregating output data [26]. The application is modeled as a graph, in which vertices represent processing jobs, and edges their data dependencies. Since the input data of a job may be distributed or partitioned, a job can be further decomposed in a number of executable tasks. Such structuring of a SWf provides a clear view of the application flow and facilitates the execution of the application in a geo-distributed environment. Currently, many Scientific Workflow Management Systems (SWfMSs) are available, *e.g.* Pegasus [17] and Swift [43]. Some of them, *e.g.* Chiron [27], support distributed, multisite execution.

The cloud stands out as a convenient infrastructure for handling SWfs, as it offers the possibility to lease resources at a very large scale and relatively low cost. In this paper, we consider the execution of a large SWf in a multisite cloud, *i.e.* a cloud with geo-distributed cloud data centers (sites). However, let us point out that using multiple cloud sites is not necessarily better than a single cloud site. However, there are important cases where using a multisite cloud is a good option. This 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 multi-

ple sites with a single cloud account, thus avoiding the burden of using multiple accounts.

There are three main reasons for using multiple cloud sites: already distributed data, resource limits at a single cloud site and monetary cost. First, the data to be processed by the SWf may already be distributed at different sites, because it is sourced from different experiments, sensing devices or laboratories (*e.g.* the well-known ALICE LHC Collaboration spans over 37 countries [1]). In this case, it may be hard to move the data to a single site, *e.g.* because the data is too large or not allowed to leave the hosting site. Second, the resource requirements to execute the SWf may well exceed the limits imposed by the cloud provider at a single site. For instance, Microsoft Azure imposes a maximum number of virtual CPUs (20 by default) in the VMs per site for both standard and premium accounts. In addition, there are other limits on storage, network bandwidth and almost all the resources provided per site. There are similar limitations for other cloud providers, *e.g.* Amazon EC2 and Google Cloud. Third, it may be less expensive to use VMs at multiple sites [28]. Cloud providers, *e.g.* Amazon, Microsoft Azure and Google Cloud, typically have different VM prices at different sites. Furthermore, they charge for inter-site data transfer. Thus, using appropriate scheduling algorithms, *e.g.* DIM [27] and ActGreedy [28], we can use VMs at different sites and reduce data transfer so as to reduce the overall monetary cost of SWf execution.

Metadata has a critical impact on the efficiency of a SWfMS as it provides a global view of data location and enables task tracking during execution. Thus, it should be readily available to the system at any given time. While it has been shown that efficient metadata handling plays a key role in performance [41, 11], little research has targeted this issue in multisite cloud. Some SWf metadata even needs to be persisted as *provenance* data to allow traceability and reproducibility of the SWf's jobs. In particular, some metadata is more frequently accessed than others (*e.g.* the status of tasks in execution in a multisite SWf is queried more often than a job's creation date). We denote such metadata by *hot metadata* and argue that it should be dynamically identified and handled in a specific, more quickly accessible way than the rest of the metadata.

In a multisite infrastructure, inter-site network latency is much higher than intra-site latency. This consideration must stay at the core of the design of a multisite metadata management system. As we discuss in Section 4, several design principles must be taken into account. In most data processing systems (should they be distributed), metadata is typically stored, managed and queried at some centralized server (or set of servers) located at a specific site [17, 20, 38]. However, in a multisite setting, with high-latency inter-site networks and large amounts of concurrent metadata operations, a centralized strategy for hot metadata management is far from the best solution.

Furthermore, in a multisite cloud, the execution of the tasks of each job, which may involve distributed data, should be efficiently scheduled to a corresponding site. The multisite scheduling process should use scheduling algorithms to decide at which site to execute the tasks to achieve a given objective, *e.g.* reducing execution time. In the scheduling process, the metadata should also be provisioned to the scheduler to make smart decisions. A centralized strategy for hot metadata management is easy to implement and works well with a few concurrent queries. However, this strategy will be inefficient if the number of queries is high or the bandwidth is low. Furthermore, the input data of a SWf may be distributed at different sites so the tasks of one job may need to be executed at different sites. In this case, a centralized strategy will incur additional communication.

In this paper, we take a different strategy based on the distribution of hot metadata in order to increase the locality of access by the different computing nodes. Inspired by [35], we adapt two distributed metadata management strategies, *i.e.* Distributed Hash Table (DHT) and replication strategy (Rep), for hot metadata management. DHT stores the hot metadata at the site corresponding to its hash value and Rep stores the hot metadata at the sites where it is generated and the site corresponding to its hash value. We propose a local storage based hot metadata management strategy, *i.e.* LOC, which stores the hot metadata at the site where it is generated. We take the centralized strategy as a baseline to show the advantage of the distributed strategies.

A major contribution of this paper is to demonstrate the effectiveness of using hot metadata for SWf execution in a multisite cloud. In the context

of thousands of SWf tasks executed across multiple sites, separating hot metadata improves SWf execution time, yet at no additional cost. In addition, during SWf execution, some hot metadata may become cold or vice-versa. We show that metadata monitoring and dynamic identification of hot or cold metadata can address this dynamic variation of the "temperature" of metadata.

This paper is a major extension of [36], with more insights on hot metadata management strategies for SWf execution in a multisite cloud. In particular, we implemented and validated new techniques for identifying hot metadata dynamically and provisioning of hot metadata for scheduling algorithms. The paper makes the following contributions:

- Based on the notion of *hot metadata*, we introduce a distributed architecture in a multisite cloud with dynamic hot metadata identification. We adapt two strategies for hot metadata management and propose a local storage based strategy to optimize the access to hot metadata and ensure availability (see Section 5).
- We develop a prototype by implementing our proposed architecture and strategies within a state-of-the-art multisite SWfMS, namely Chiron [32], using a relational DBMS (RDBMS) to manage hot metadata (Section 6).
- We combine the hot metadata management strategies and three scheduling algorithms, *i.e.* OLB, MCT and DIM, by enabling hot metadata management provisioning for multisite task scheduling (see Section 6).
- We demonstrate that dynamic hot metadata identification and efficient management of hot metadata reduces the execution time of SWfs by enabling timely data provisioning and avoiding unnecessary *cold* metadata handling (see Section 7). We also show the advantages of decentralized hot metadata management strategies with different scheduling algorithms (Section 7).

This paper is organized as follows. Section 2 discusses related work. Section 3 presents our SWf model and discusses the challenges for hot metadata management. Section 4 introduces our design principles. Section 5 describes the SWfMS architecture with dynamic hot metadata identification and our hot metadata management strategies. Section 6 gives the implementation of our proposed strategies. Section 7 presents our experimental results. Finally, Section 8 concludes.

2 Comparison with Related Work

Most SWfMSs take advantage of the optimizations provided by the *data* management layer. For instance, data management techniques try to strategically place the data before or during the execution of a SWf. In multisite environments, these techniques favor starting the SWf only after gathering all the needed data in a shared-disk file system at one data center, which is time consuming. Less attention has been given to *metadata*, often considered a second class citizen in the SWf life cycle. Furthermore, although scheduling has been studied for SWf execution in a multisite cloud, little attention is paid to the combination of hot metadata management strategies and different scheduling algorithms.

SWf execution in a multisite cloud. The execution of SWfs in a multisite cloud is different from other execution environments. First, compared with traditional business workflows, *e.g.* purchase order processing, which are rather simple and manipulate small amounts of data, SWfs are data-intensive and can be very complex (in terms of number and combinations of jobs), thus making execution time a major issue [26]. Furthermore, they need to ensure result reproducibility, which requires registering and managing provenance data regarding data sets, intermediate data and job executions. Note that the recent big data analytics workflows share many similarities with SWfs. Second, the multisite cloud environment is different from the geographically distributed environment, which only needs to deal with multiple servers without considering the execution within each server. A multisite cloud environment is composed of multiple cloud sites, each of which contains distributed nodes (servers) connected by a fast local network, *e.g.* infiniband. Thus, both intra-site and inter-site execution must be considered, as well as the inter-

action between them. Furthermore, compared with data processing systems for cluster environments, *e.g.* Spark [45] or Hadoop [2], multisite SWf execution must handle data in multiple clusters and use existing programs at certain cloud sites, with limited network connections among different sites.

Centralized strategies. Metadata is usually handled by means of centralized registries implemented on top of relational databases, which only hold static information about data locations. SWfMSs like Pegasus [17], Swift [43] or Chiron [32] leverage such schemes, typically involving a single server that processes all the requests. In case of increased client concurrency or high I/O pressure, however, the single metadata server can quickly become a performance bottleneck. Also, the workloads involving many small files, which translate into heavy metadata accesses, are penalized by the overheads from transactions and locking [40, 39]. A lightweight alternative to databases is to index the metadata, although most indexing techniques [42, 44] are designed for data rather than metadata. Even the dedicated index-based metadata schemes [24] use a centralized index and are not adequate for large-scale SWfs, nor can they scale to multisite deployments.

Distributed strategies. As a first step towards complete geographical distribution, some SWfMSs rely on distributed file systems that partition the metadata and store it at each node (*e.g.* [19], [29]), in a shared-nothing architecture. Hashing is the most common technique for uniform partitioning: it consists of assigning metadata to nodes based on a hash of a file identifier. Giraffa [7] uses full pathnames as key in the underlying HBase [3] store. Lustre [9] hashes the tail of the filename and the ID of the parent directory. Similar hashing schemes are used by [14, 31, 13] with a low memory footprint, granting access to data in almost constant time. FusionFS [48] implements a distributed metadata management based on DHTs as well. Chiron itself has a version with distributed control using an in-memory distributed DBMS. All these systems are well suited for single-cluster deployments or SWfs that run on supercomputers. However, they are unable to meet the practical requirements of SWfs executed on clouds. Similarly to us, CalvinFS [41] uses hash-partitioned key-value metadata across geo-distributed sites to handle small files, yet it does not account for SWf semantics.

Hybrid strategies. Zhao *et al.* [47] proposed using both a distributed hash table (FusionFS [48]) and a centralized database (SPADE [46]) to manage the metadata. Similarly to us, their metadata model includes both file operations and provenance information. However, they do not make the distinction between hot and cold metadata, and they mainly target single site clusters.

Dynamic hot metadata identification. The "temperature" of metadata may vary during execution and trying to predict hot metadata before execution can be error prone. Thus, the solution is to dynamically identify hot metadata at runtime. Some frameworks assess the data "temperature" *offline*, *i.e.* they perform a later analysis on a frequency-of-access log to avoid overhead during the operation [25]. However, this approach is only useful when there are subsequent runs. More interestingly, *online* approaches maintain a rank on the frequency of access to the data alongside the execution, for example in adaptive replacement cache [30]. Then, we can dynamically identify the hot metadata based on the rank and frequency. The dynamic identification of hot and cold metadata is different from temporal locality [22, 37], which places a recently used object into memory or in a same node while assuming that this object will be accessed in the near future. The temporal locality is generally used to handle intermediate data. Our dynamic identification is focused on metadata and is based on the access frequency generated from a relatively long observation of the metadata. In addition, our different hot metadata management strategies handle the hot metadata at an inter-site level instead of multiple nodes (intra-site level).

Metadata in the cloud. Most of the previous work on metadata management comes from the HPC world, with solutions relying on low-latency networks for message passing and tiered cluster deployments that separate compute and storage nodes. On the other hand, cloud computing is different from HPC: high latency networks connect the sites, a much lower degree of (per-object) concurrency, a more specialized storage interface provided to applications, which are explicitly aware of the anticipated workloads and access patterns. An important difference with past work is our focus on a whole SWf application and its interaction with the cloud infrastructure. Because their target use-cases and interface semantics differ, parallel file

systems cannot be used out-of-the-box in the cloud and are often considered to be mutually inappropriate. Instead, we borrow ideas from the HPC community, leveraging the SWf semantics and the cloud services publicly available.

Multisite scheduling. There are many algorithms to schedule tasks in a distributed system. In this paper, we use three scheduling algorithms, *i.e.* OLB (Opportunistic Load Balancing), MCT (Minimum Completion Time) and DIM (Data-Intensive Multisite task scheduling) [27]. OLB and MCT are basic algorithms, which are widely used in diverse SWfMSs. DIM is the best in our context since it is designed with a centralized metadata management strategy, *i.e.* all the metadata is stored and managed at a single site, for SWf distributed execution. Although we propose decentralized hot metadata management strategies, the cold metadata is always stored in a centralized database, which fits well with the assumption of DIM. OLB randomly selects an available or an earliest available site for a task while MCT schedules a task to the site that can finish the execution first with the consideration of the time to transfer intermediate data. DIM first schedules the tasks to the site where the input data is located. Then, it manages the workload at each site in order to achieve load balancing and reduce the overall execution. In order to reduce the time to transfer intermediate data, both MCT and DIM are data location aware, *i.e.* they schedule many tasks to where the input data is. Since it may take much time to transfer intermediate data between sites, MCT generally has better performance than OLB. In addition, since it may also take much time to transfer metadata, DIM generally outperforms both MCT and OLB. In this paper, we also evaluate the performance of different hot metadata management strategies in combination with different scheduling algorithms.

3 Hot Metadata Based Approach

Metadata management significantly impacts the performance of computing systems that must deal with thousands or millions of individual files, as with SWfs. In this section, we introduce our SWf model, and discuss centralized metadata management and multisite cloud scalability. Then, we define hot metadata and the challenges for hot metadata management.

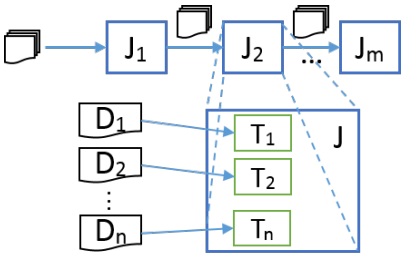


Figure 1: SWf diagram depicting the relation between *jobs* (J), *tasks* (T) and *data chunks* (D).

3.1 SWf Model

A SWf is modeled as a graph, in which vertices represent data processing jobs and edges represent dependencies between them (Figure 1). A *job* (J) is a description of a piece of work that forms a logical step within a SWf representation. Since SWf jobs may process multiple data chunks, one job can actually correspond to several executable tasks for different parts of input data during execution. A *task* (T) is the representation of a job within a one-time execution of this job, which processes a data chunk (D) [26], *i.e.* there is a task for each unit of data to be processed.

3.2 Centralized Metadata Management

SWfMSs handle more than file-specific metadata. Running the SWf itself generates a significant amount of execution-specific metadata, *e.g.* scheduling metadata (*i.e.* which task is executed where) and data-to-task mappings. Most of today's SWfMSs handle metadata in a centralized way. File-specific metadata is stored in a centralized server, either own-managed or through an underlying file system, while execution-specific metadata is normally kept in the execution's master entity.

Controlling and combining all these sorts of metadata translate into a critical workload as scientific datasets get larger. For instance, the CyberShake SWf [15] runs more than 800,000 tasks, handling an equal number of individual data pieces, processing and aggregating over 80,000 input files

(which translates into 200 TB of data read), and requiring all of these files to be tracked and annotated with metadata [23, 15]. Tasks' runtime is in the order of milliseconds, *e.g.*, in a Montage SWf of 0.5 degree (see Section 7.1), out of 80 tasks, 36 tasks execute under 500 milliseconds. With many tasks, the load of parallel metadata operations becomes very heavy, and handling it in a centralized fashion represents a serious performance bottleneck.

3.3 Multisite Cloud Scalability

Often enough, scientific data is so huge and widespread that it cannot be processed or stored at a single cloud site. On the one hand, the data size or the computing requirements might exceed the capacity of the site or the limits imposed by the cloud provider. On the other hand, data might be widely distributed, and due to their size, it is more efficient to process them closer to where they reside than to bring them together. For instance, the US Earthquake Hazard Program monitors more than 7,000 sensors systems across the country reporting to the minute [10]. In either case, multisite clouds are progressively being used for executing large-scale SWfs.

Managing metadata in a centralized way for such scenarios is not appropriate. On top of the congestion generated by concurrent metadata operations, remote inter-site operations cause severe delays in SWf execution. To address this issue, some approaches propose the use of decentralized metadata servers [41]. In our previous work [35], we also implemented a decentralized management architecture that proved to handle metadata up to twice as fast as a centralized solution.

In this paper we make one step further. Our focus is on the metadata access frequency, in particular on identifying fractions of metadata that do not require multiple updates. The goal is to enable a more efficient decentralized metadata management, reducing the number of inter-site metadata operations by favoring the operations on frequently accessed metadata, which we call *hot metadata*.

3.4 Hot Metadata

The term *hot data* refers to data that needs to be frequently accessed [25]. Hot data is usually critical for the application and must be placed in a fast and easy-to-query storage [21]. We apply this concept to the context of SWf management and define **hot metadata** as the metadata that is frequently accessed during the execution of a SWf. Conversely, less frequently accessed metadata will be denoted *cold metadata*. We distinguish two types of hot metadata: *task metadata* and *file metadata*.

Task metadata is the metadata for the execution of tasks, which is composed of the command, parameters, start time, end time, status and execution site. Hot job metadata enables SWfMSs to search and generate executable tasks. During execution, the status and the execution site of tasks are queried many times by each site to search for new tasks to execute and determine if a job is finished. In addition, the status of a task may be updated several times. As a result, it is important to get this metadata quickly.

File metadata, which we consider as "hot" for the SWf execution, is relative to the size, location and possible replicas of a given piece of data. Knowledge of file hot metadata allows the SWfMS to place the data close to the corresponding task, or vice-versa. This is especially relevant in multisite settings: timely availability of the file metadata enables moving data *before* it is needed, hence reducing the impact of low-speed inter-site networks. In general, other metadata such as file ownership is not critical for the execution and thus regarded as cold metadata.

3.5 Challenges for Hot Metadata Management

There are several implications in order to effectively apply the concept of *hot metadata* to real systems. At this stage of our research, we apply simple, yet efficient solutions to these challenges.

How to decide which metadata is hot? We have empirically chosen the aforementioned task and file metadata as *hot*, since they have statistically proven to be more frequently accessed by the SWfMS we use. A sample execution of a 1-degree Montage SWf (see Figure 2) as described in Section 7.1, running 820 jobs and 57K metadata operations reveals that in a centralized execution, 33% of them are file metadata operations (storeFile, getFile) and 32% are task metadata operations (loadTask,

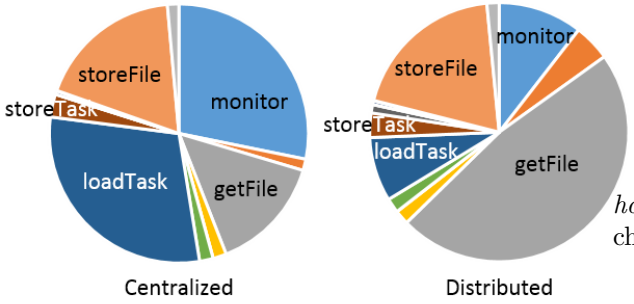


Figure 2: Relative frequency of metadata operations in Montage.

storeTask), whereas in a distributed run, up to 67% are file operations, and task operations represent 11%. The rest correspond mostly to monitoring and node/site related operations.

However, a particular SWf might actually use other metadata more often. Since SWfs are typically defined in structured formats (*e.g.* XML files), another way to account for user-defined hot metadata would be to add a property tag to each job definition where the user could specify which metadata to consider as *hot*. However, scientific applications evolve during execution. At a given point, some data might no longer be as relevant as it was initially; in other words, hot data becomes cold, or vice-versa. In the case of hot to cold data, SWfMSs might remove them from the fast-access storage or even delete them. Conversely, data that becomes relevant can be promoted to fast storage. In addition, a user may not have enough information about the "temperature" of the metadata and wrongly identify cold metadata as hot metadata. Thus, dynamic hot metadata identification is implemented in our system when using user's tags (see details in Section 5).

How to assess that the choice of hot metadata is right? Evaluating the efficiency of choosing hot metadata is not trivial. Metadata is much smaller than the application's data and handling it over networks with fluctuating throughput may produce inconsistent results in terms of execution time. Nevertheless, an indicator of the improvement brought by an adequate choice of hot metadata, which is not time-bounded, is the *number of metadata operations performed*. In our experimental evaluation (see Section 7) we present results in terms of both execution time and number of tasks performing such operations.

The next section describes how the concept of *hot metadata* translates into architectural design choices for efficient multisite SWf processing.

4 Design Principles

The foundation of our architecture is based on three key design choices: two-layer multisite SWf management, adaptive placement for hot metadata, and eventual consistency for high-latency communication.

Two-Layer Multisite SWf Management. We propose to use a two-layer multisite system (see details in Section 5): (1) The lower *intra-site* layer operates as current single-site SWfMS: a site composed of several computing nodes and a common file system, one of such nodes acts as master and coordinates communication and task execution. (2) An additional higher *inter-site* layer coordinates the interactions at the site-level through a master/slave architecture (one site being the master site). The master node in each site is in charge of synchronization and data transfers.

Adaptive Placement for Hot Metadata. Job dependencies in a SWf form common structures (*e.g.* pipeline, data distribution and data aggregation) [12]. SWfMSs usually take into account these dependencies to schedule the job execution in a convenient way to minimize data movements (*e.g.* job co-location). Accordingly, different SWfs will yield different scheduling patterns. In order to take advantage of these scheduling optimizations, we must adapt the SWf's metadata storage scheme. However, maintaining an updated version of all metadata across a multisite environment consumes a significant amount of communication time, incurring also monetary costs. To reduce this impact, we will evaluate different storage strategies for hot metadata during the SWf's execution, while keeping cold metadata stored locally and synchronizing such cold metadata only during the execution of the job.

Eventual Consistency for High-Latency Communication. While cloud sites are normally

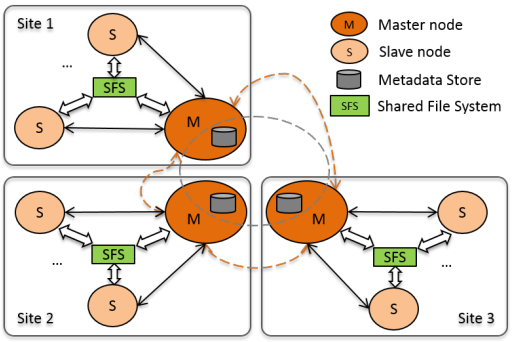


Figure 3: Multisite SWf execution architecture with decentralized metadata. Dotted lines represent inter-site interactions.

interconnected by high-speed networks, the latency is ultimately bounded by the physical distance between sites and communication time might reach the order of seconds [4]. Under these circumstances, it is unreasonable to aim for a system with a fully consistent state in all of its components at a given moment without strongly compromising the performance of the application. SWf semantics allows us the flexibility to opt for an eventually consistent system: a task processes one or several specific pieces of data; such task will begin its execution only when all the pieces it needs are available in the metadata storage; however, the rest of tasks continue executing independently. Thus, with a reasonable delay due to the higher latency propagation, the system is guaranteed to be eventually consistent (see [34] for details on eventual consistency).

5 Architecture

In our previous work [35], we explored different strategies for SWf-driven multisite metadata management, focusing on file metadata. Our study found that a hybrid strategy combining decentralized metadata and replication better suits the needs of large-scale multisite SWf execution. It also showed that the right strategy to apply depends on the SWf structure. In this section, we elaborate on top of such observations. First, we present an architecture for multisite cloud SWf processing which features decentralized metadata management. Second, we enrich this architecture with a component specifically dedicated to the management of *hot metadata* with dynamic hot metadata identification across multiple sites. Third, we adapt two metadata management strategies to hot metadata management and propose a local storage based hot metadata management strategy.

Two-level Multisite Architecture. Following our design principles, the basis for our SWfMS is a 2-level multisite architecture, as shown in Figure 3.

1. At the inter-site level, all communication and synchronization is handled through a set of master nodes (M), one per site. One site acts as a global coordinator (master site) and is in charge of scheduling jobs/tasks to each site. Every master node holds a *metadata store*, which is part of the global metadata storage and is directly accessible to all other master nodes.

2. At the intra-site level, our system preserves the typical master/slave scheme widely-used today on single-site SWfMS: the master node coordinates a group of slave nodes which execute the SWf tasks. All nodes within a site are connected to a shared file system to access data resources. *Metadata updates* are propagated to other sites through the master node, which classifies hot and cold metadata as explained below.

Dynamic Hot Metadata Identification. During execution, when there is no user's tag, we take the task and file metadata as hot metadata, as explained in Section 3.4. However, when the user puts the tags of hot metadata in the SWf definition, the user's tags are taken into consideration at the beginning of execution. The access frequency of all the metadata in each multi-task job, *i.e.* the job that consists of more than a single task, is monitored and ranked. The metadata that meets one of two following requirements is identified as the hot metadata. The first requirement is that its access frequency is bigger than 1 per task, which means that the metadata becomes more frequently accessed when there are more tasks. The second requirement is that if there is no metadata meeting the first requirement, the metadata of the highest frequency in the rank is identified as hot metadata. If the hot metadata (identified by the user) meets

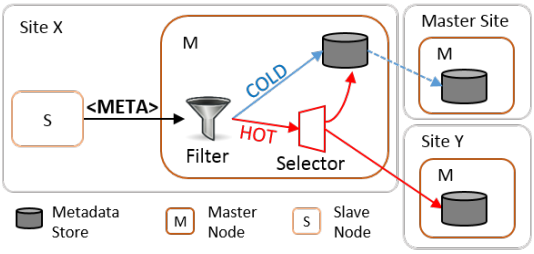


Figure 4: The hot metadata filtering component.

one of the two requirements, it is kept as hot. Otherwise, it is identified as cold data. The cold data (identified by the user) that meets one of the two requirements is identified as hot.

Separate Management of Hot and Cold Metadata. Following our characterization of *hot metadata* in Section 3.4, we incorporate an intermediate component that filters out cold metadata operations. This ensures that: hot metadata operations are managed with high priority over the network, and cold metadata updates are propagated only during periods of low network congestion.

The filter is located in the master node of each site (see Figure 4). It separates hot and cold metadata, favoring the propagation of hot metadata and thus alleviates congestion during metadata-intensive periods. The storage location of the hot metadata is then selected based on some metadata management strategies, as developed below.

Decentralized Hot Metadata Management Strategies. We consider two different alternative strategies for decentralized metadata management (previously explored in [35]). In this paper, we study their application to *hot* metadata. We also propose a local storage based hot metadata management strategy, *i.e.* LOC. These three strategies all include a metadata server in each of the sites where execution nodes are deployed. They differ in the way hot metadata is stored and replicated, as explained below.

Local without replication (LOC) Every new hot metadata entry is stored at the site where it has been created. For read operations, metadata is queried at each site and the site that stores the data will give the response. If no reply is received within a time threshold, the request is resent. This strategy will typically benefit pipeline-like SWf structures, where consecutive tasks are usually co-located at the same site.

Hashed without replication (DHT) Hot metadata is queried and updated following the principle of a distributed hash table (DHT). The site location of a metadata entry is determined by a simple hash function applied to its key attribute, *file-name* in case of file metadata, and *task-id* for task metadata. We assume that the impact of inter-site updates will be compensated by the linear complexity of read operations.

Hashed with local replication (REP) We combine the two previous strategies by keeping both a local record of the hot metadata and a hashed copy. Intuitively, this would reduce the number of inter-site reading requests. We expect this hybrid strategy to highlight the trade-offs between metadata locality and DHT linear operations.

6 DMM-Chiron SWfMS

In order to validate our architecture in a multisite cloud, we developed a SWfMS prototype, called Decentralized-Metadata Multisite Chiron (DMM-Chiron), which provides support for *decentralized* metadata management, with the dynamic identification of hot metadata. In this section, we first present the baseline system, *i.e.* multisite Chiron. Then, we describe DMM-Chiron, including its metadata model, the implementation of dynamic hot metadata identification and hot metadata management strategies, and provisioning hot metadata for multisite scheduling.

6.1 Baseline: Multisite Chiron

This work builds on Multisite Chiron [27], a SWfMS specifically designed for multisite clouds. Its layered architecture is presented in Figure 5; it is composed of nine modules. Multisite Chiron exploits a textual UI to interact with users. The SWf is analyzed by the *Job Manager* to identify executable jobs, *i.e.* unexecuted jobs, for which the input data is ready. The same module generates the executable tasks at the beginning of job execution at the master site.

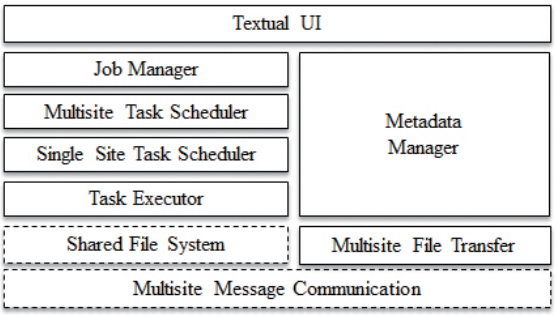


Figure 5: Layered architecture of Multisite Chiron [27].

Scheduling is done in two phases: the *Multisite Task Scheduler* at the master site schedules each task to a site, following one of three scheduling algorithms, *i.e.* OLB, MCT and DIM. The users of Multisite Chiron can choose the multisite scheduling algorithm. While the *Single Site Task Scheduler* applies the default dynamic FAF (First job First) approach used by Chiron [32] to schedule tasks to computing nodes. Some jobs (query jobs) can be expressed as queries, which exploit the DBMS to be executed. In order to synchronize execution, the query jobs are executed at a master site. It is worth to clarify that optimizations to the scheduling algorithms are out of the scope of this paper.

Then, the *Task Executor* at each computing node runs the tasks. Along the execution, metadata is handled by the *Metadata Manager* at the master site. Since the metadata structure is well defined, we use a relational database, namely PostgreSQL, to store it. All data (input, intermediate and output) is stored in a *Shared File System* at each site. The file transfer between two different sites is performed by the *Multisite File Transfer* module. The *Multisite Message Communication* module of the master node at each site is in charge of synchronization through a master/slave architecture while the *Multisite File Transfer* module exploits a peer-to-peer approach for data transfers.

6.2 Metadata Model

Figure 6 (see details in [27]) shows the metadata model for SWf execution in a multisite cloud. A SWf is related to multiple jobs. Each job is related to several relations, tasks and files and each task is related to a site, a VM, a relation and multiple files. A relation is a dataset to be processed by tasks or jobs. Each element in the model, *i.e.* job, task, site, file and VM, corresponds to a table in the relational database. We implement this model at each site in order to enable the execution at each site. The data stored in the task, file and job tables (for the experiment of dynamic hot metadata identification in Section 7.4) is different at each site according to different hot metadata management strategies used during SWf execution. The data stored in other tables is synchronized by a mediator implemented at each site at the beginning and the end of the execution of the SWf and each job. A mediator is a software module that exploits encoded knowledge about certain subsets of data at each site to create information at the site for SWf execution [34].

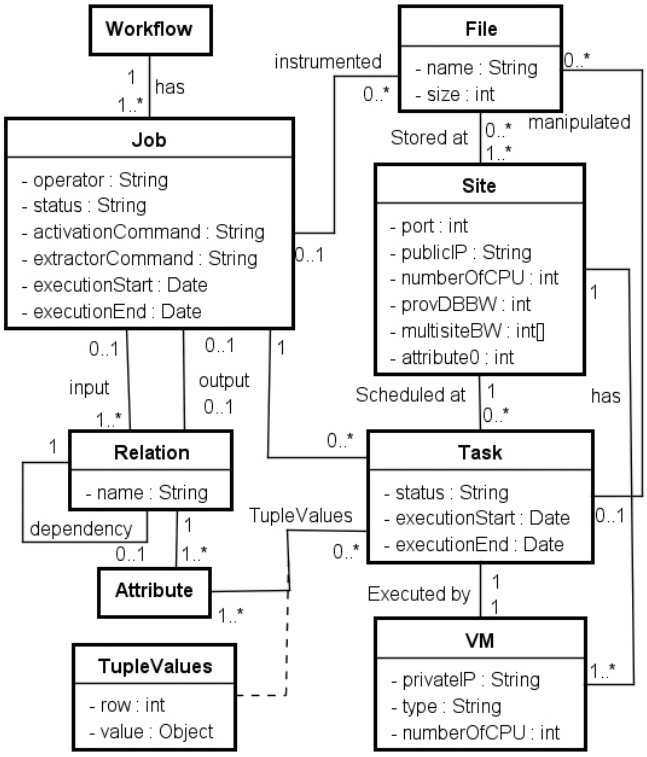


Figure 6: **Multisite Metadata Model [27].**

6.3 Dynamic Hot Metadata

Since the temperature of hot or cold metadata varies, we look into the two situations: promoting cold to hot metadata and downgrading hot to cold metadata.

Promoting Cold to Hot Metadata. User-defined hot metadata as discussed so far would not allow metadata to be dynamically promoted, since a SWf definition file is rather static. However, we build on this idea and integrate an online ranking: given a SWf defined through an XML file (or any other definition language), a list of metadata attributes is passed to the SWfMS in the same file; then, the SWfMS monitors the access frequency of each of such attributes and periodically produces a ranking to verify whether an attribute meets one of the requirements explained in Section 5, and thus *promotes* it to hot metadata. The maximum number of attributes allowed as *hot* metadata could be also dynamically calculated according to the aggregated size of the metadata stores.

Downgrading Hot to Cold Metadata. Less frequently accessed metadata could also be identified using the same approach as above. Upon identification, degrading some metadata to *cold* also incurs that metadata previously considered hot is removed from fast-access storage.

The process of dynamic identification of hot metadata occurs at the end of the execution of each multi-task job in the master node of the master site. The identification results are synchronized and broadcasted to all the sites for the next execution. To avoid interfering with the SWf scheduling and execution processes, these scenarios are implemented transparently within the metadata storage system.

6.4 Combining Multisite and Hot Metadata Management

To implement and evaluate our strategies to decentralized metadata management, we further extended Multisite Chiron by adding multisite metadata protocols. We mainly modified two modules as described in the next sections: the *Job Manager* and the *Metadata Manager*.

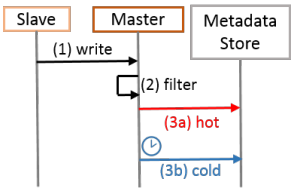
From Single- to Multisite Job Manager. The Job Manager handles the process that verifies if the execution of a job is finished, in order to launch the next jobs. Originally, this verification was done on the metadata stored at the master site. In DMM-Chiron we implement an optimization to each of the hot metadata management strategies (Section 5): for LOC, the local DMM-Chiron instance verifies only the tasks scheduled at that site and the master site confirms that the execution of a job is finished when all the sites finish their corresponding tasks. For DHT and REP, the master DMM-Chiron instance of the master site checks each task of the job.

RDBMS. Metadata obeys to a data model [33], which is equivalent to a data structure. Thus, it is convenient to store the metadata in a structured database. A data item represents a set of values corresponding to different attributes as defined in the metadata model. An RDBMS is very efficient to query structured data by comparing the value of different attributes of each data item in a structured database. We choose the popular PostgreSQL RDBMS to manage hot metadata.

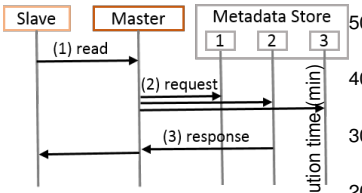
Protocols for Multisite Hot Metadata. The following protocols illustrate our system's *metadata operations*. We recall that metadata operations are triggered by the slave nodes at each site, which are the actual executors of the SWf tasks.

Metadata Write As shown in Figure 7a, a new metadata record is passed on from the slave to the master node at each site (1). Upon reception, the master filters the record as either hot or cold (2). The hot metadata is assigned by the master node to the metadata storage pool at the corresponding site(s) according to one metadata strategy. Created cold metadata is kept locally and propagated asynchronously to the master site during the execution of the job.

Metadata Read Each master node has access to the entire pool of metadata stores so that it can get hot metadata from any site. Figure 7b shows the process. When a read operation is requested by a slave (1), a master node sends a request to each metadata store (2) and it processes the response that comes first (3), provided such response is not an empty set. This mechanism ensures that the master node gets the required metadata in the shortest time. During execution, DMM-Chiron gathers all the task metadata stored at each site to verify whether the execution of a job is finished.



(a) Write



(b) Read

Figure 7: Metadata Protocols.

6.5 Hot Metadata Provisioning for Multisite Scheduling

In order to efficiently schedule the tasks at different sites, the scheduling process of MCT and DIM needs to get the information about where the input data of each task is located. This information is available in the hot metadata, *i.e.* file metadata. Thus, the hot metadata is provisioned to the multisite task scheduler. Since the job manager generates tasks for job execution at the master site, the file metadata is also available at the master site. Thus, we directly load the file metadata at the master site for multisite scheduling. In addition, we load the file metadata in the memory once and then use the data for the whole scheduling process in order to reduce scheduling as explained in [27].

7 Experimental Evaluation

In this section, we evaluate and compare our distributed hot metadata management strategies with the (state-of-the-art) centralized metadata management strategy, using our implementation of DMM-Chiron in a multisite cloud (using Azure). First, we present the experimental setup with two real-life SWf use cases. Second, we compare the performance of the different hot metadata management strategies, *i.e.* centralized, LOC, DHT and REP), against different SWf structures. Third, we study the impact of two different scheduling algorithms, *i.e.* MCT and DIM, on the performance of these strategies. Finally, we evaluate the performance of dynamic hot metadata identification.

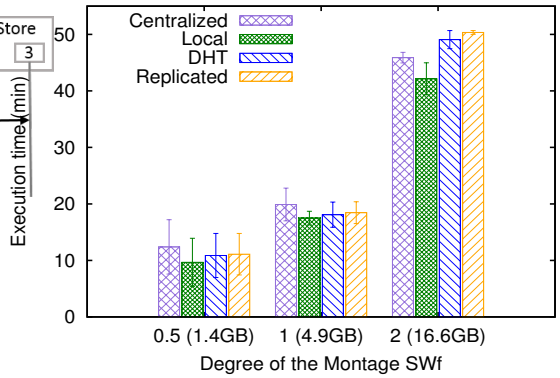


Figure 8: Montage execution time for different strategies and degrees. Avg. intermediate data shown in parenthesis.

7.1 Experimental Setup

DMM-Chiron was deployed on the Microsoft Azure cloudusing a total of 27 nodes of A4 standard VMs (8 cores, 14 GB memory). The VMs were evenly distributed among three sites: West Europe (WEU, Netherlands), North Europe (NEU, Ireland) and Central US (CUS, Iowa). Control messages between master nodes are delivered through the Azure Bus.

We consider two popular real-life SWfs, with which we have a long experience:

Montage is a toolkit created by the NASA/IPAC Infrared Science Archive and used to generate custom mosaics of the sky from a set of images [16]. Additional input for the SWf includes the desired region of the sky, as well as the size of the mosaic in terms of square degrees. We model the Montage SWf using the proposal of Juve et al. [23]. Montage contains 13 jobs, in which 4 jobs, *i.e.* mProjectPP (mPro), Prepare (Prep), mDiffFit (mDif) and mBackground (mBac), correspond to multiple tasks.

BuzzFlow is a big data SWf that searches for trends and measures correlations by analyzing the data in scientific publications [18]. It analyzes data collected from bibliography databases such as DBLP or PubMed. Buzz is composed of 13 jobs and 4 jobs, *i.e.* FileSplit, Buzz, BuzzHistory and HistogramCreator, correspond to multiple tasks.

7.2 Comparison of Hot Metadata Management Strategies

Our hypothesis is that no single decentralized strategy can well fit all SWf structures. For instance, a highly parallel task would exhibit different metadata access patterns than a concurrent data gathering task. Thus, the improvements brought to one type of SWf by either of the strategies might turn to be detrimental for another. To evaluate this hypothesis, we ran several combinations of our strategies with the featured SWfs and OLB while taking file and task metadata as hot metadata. The results are shown in Figures 8 - 9 and 11 - 12.

Figure 8 shows the average execution time for the Montage SWf generating 0.5-, 1-, and 2-degree mosaics of the sky, using in all the cases a 5.5 GB image database distributed across the three sites. With a larger degree, a larger volume of intermediate data is handled and a mosaic of higher resolution is produced. Table 1 summarizes the volumes of intermediate data generated per execution.

In the chart (Figure 8), we note in the first place a clear time gain of up to 28% by using LOC instead of a centralized strategy, for all degrees. This result was expected since the hot metadata is now managed in parallel by three instances instead of one, and only the cold metadata is forwarded to the master site for scheduling purposes.

We observe that for mosaics of Degree 1 and under, the use of distributed hashed storage also outperforms the centralized version. However, we note a performance degradation in DHT, starting at 1-degree and getting more evident at 2-degree. We attribute this to the fact that there is a larger number of long-distance hot metadata operations compared to the centralized strategy. With hashed strategies, 1 out of 3 operations are carried out on average between CUS and NEU. In the centralized strategy, NEU only performs operations in the WEU site, thus such long latency operations are reduced. We also associate this performance drop with the size of intermediate data being handled by the system: while we try to minimize inter-site data transfers, with larger volumes of data such transfers affect the execution time up to a certain degree and independently of the metadata management strategy. We conclude that while the DHT strategy might seem efficient due to linear read and write operations, it is not well suited for geo-distributed executions, which favor locality and penalize remote operations.

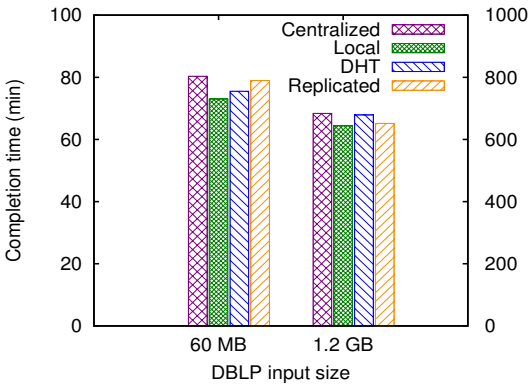


Figure 9: Buzz SWf execution time. Left Y-axis scale corresponds to 60 MB execution, right Y-axis to 1.2 GB.

In a similar experiment, we validated our strategies using the Buzz SWf, which is rather data intensive, with two DBLP database dumps of 60 MB and 1.2 GB. The results are shown in Figure 9. Note that the left and right Y-axes differ by one order of magnitude. We observe again that DMM-Chiron brings a general improvement in the execution time with respect to the centralized strategy: 10% for LOC in the 60 MB dataset and 6% for 1.2 GB, while for DHT and REP, the time improvement was of less than 5%.

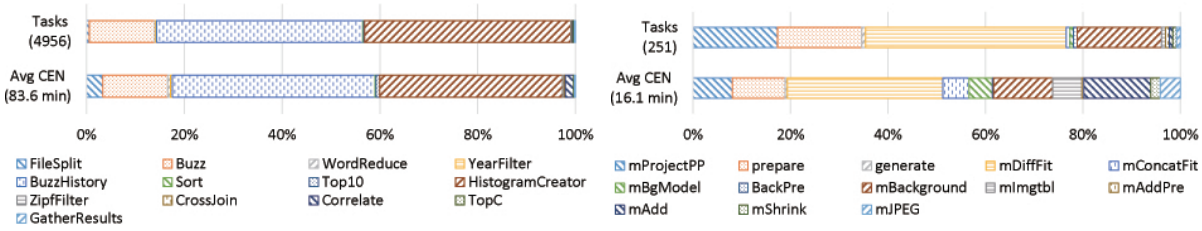
In order to better understand the performance improvements brought by DMM-Chiron, and also to identify the reason of the low runtime gain for the Buzz SWf, we evaluated Montage and Buzz in a per-job granularity. Although the time gains perceived in the experiments might not seem significant at first glance, two important aspects must be taken into consideration:

Optimization at no cost Our proposed solutions are implemented using exactly the same number of resources as their counterpart centralized strategies: the decentralized metadata stores are deployed within the master nodes of each site and the control messages are sent through the same existing channels. This means that such gains come at no additional cost for the user.

Actual monetary savings Our longest experiment (Buzz 1.2 GB) runs in the order of hun-

	0.5-degree	1-degree	2-degree
CEN	1.4 (0.5, 0.5, 0.4)	4.9 (1.7, 1.5, 1.7)	17.1 (6.0, 6.4, 4.7)
LOC	1.3 (0.7, 0.2, 0.4)	4.8 (2.1, 1.0, 1.7)	16.2 (8.4, 4.6, 3.2)
DHT	1.5 (0.6, 0.6, 0.4)	4.9 (1.9, 1.3, 1.8)	16.6 (5.4, 4.9, 6.2)
REP	1.4 (0.5, 0.5, 0.4)	4.9 (1.5, 1.9, 1.5)	16.8 (6.6, 3.8, 6.4)

Table 1: Intermediate data in GB for different degrees of Montage executions. Per-site breakdown is expressed as: Aggregated (size WEU, size NEU, size CUS).



(a) Buzz (b) Montage

Figure 10: SWf per-job breakdown. Very small jobs are enhanced for visibility.

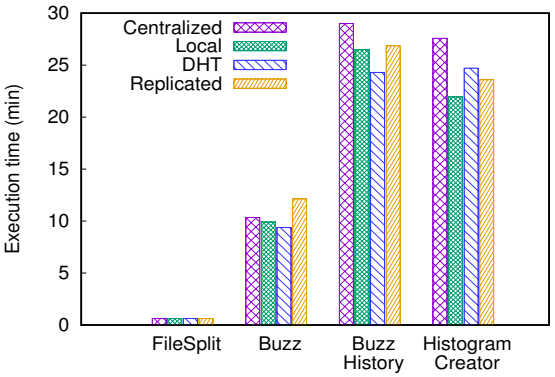


Figure 11: Execution time of multi-task jobs on the Buzz SWf with 60 MB input data.

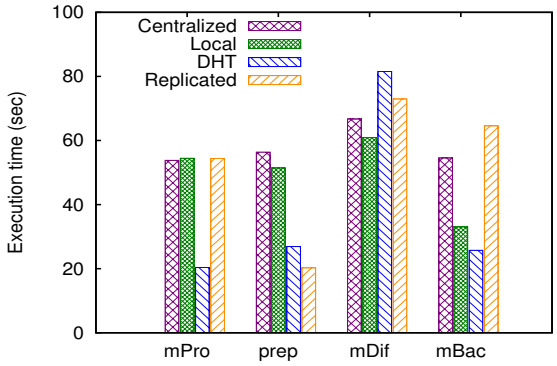


Figure 12: Execution time of multi-task jobs on the Montage SWf of 0.5 degree. Avg. intermediate data shown in parenthesis.

dreds of minutes. With today's scientific experiments running at this scale and beyond, a gain of 10% actually implies savings of hours of cloud computing resources.

In DMM-Chiron, the various tasks of multi-task jobs are evenly distributed to the available sites and thus can be executed in parallel. It is precisely with these kind of jobs that DMM-Chiron yields its best performance. Figure 10 shows a breakdown of Buzz and Montage SWfs with the proportional size of each of their jobs from two different perspectives: tasks count and average execution time. Our goal is to characterize the most *relevant* jobs in each SWf by number of tasks and confirm their relevance by looking at their relative execution time. In Buzz, we notice that both metrics are highly dominated by three jobs: Buzz (676 tasks), BuzzHistory (2134) and HistogramCreator (2134), while the others are so small that they are barely noticeable. FileSplit comes fourth in terms of execution time and it is indeed the only remaining multi-task job (3 tasks). Likewise, we identify for Montage the only four multi-task jobs: mProject (45 tasks), prepare (45), mDiff (107) and mBackground (45).

In Figures 11 and 12 we look into the execution

time of the multi-task jobs of Buzz and Montage, respectively. In Figure 11, we observe that except for one case, namely the Buzz job with REP, the decentralized strategies outperform considerably the baseline (up to 20% for LOC, 16% for DHT and 14% for REP). In the case of FileSplit, we argue that the execution time is too short and the number of tasks too small to reveal a clear improvement. However, the other three jobs confirm that DMM-Chiron performs better for highly parallel jobs. It is important to note that these gains are much larger than those of the overall execution time (Figure 9) since there are still a number of workloads executed sequentially, which have not been optimized by the current release of DMM-Chiron.

Figure 12 shows the execution of each multi-task job for the Montage SWf of 0.5 degree. The figure reveals that, on average, hot metadata distribution substantially improves centralized management in most cases (up to 40% for LOC, 53% for DHT and 64% for REP). However, we notice some unexpected peaks and drops when using DHT. There is a possibility that most tasks of some jobs are scheduled to the site corresponding to the hash code of the metadata. In this case, the DHT strategy has a good performance. Otherwise, its performance is poor. In addition, after a number of executions, such cases can also be due to common network latency variations of the cloud environment added to the fact that the job execution time is rather short (in the order of seconds).

7.3 Impact of Scheduling Algorithms

There is no single decentralized strategy that can fit all scheduling algorithms and some scheduling algorithms may be optimized for a certain metadata management strategy. Thus, the improvements brought to one scheduling algorithm by either of the strategies might turn to be not so good or even detrimental for another. To better understand this relationship between hot metadata management strategy and scheduling algorithm, we ran several combinations of our strategies with Montage and two other scheduling algorithms, *i.e.* MCT and DIM. We use the same configuration except for the scheduling algorithm as explained in Section 7.2.

Figure 13 shows that LOC is always better (up to 28%) than the centralized strategy in terms of

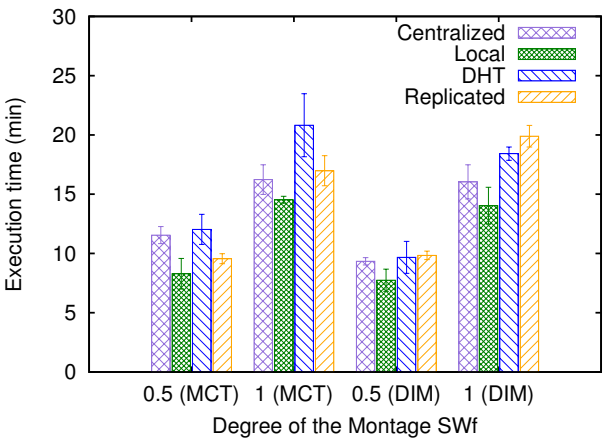


Figure 13: The execution time of the Montage SWf with MCT/DIM for different strategies and degrees. Avg. intermediate data shown in parenthesis.

overall SWf execution time with MCT and DIM. Since DIM is optimized for the centralized strategy, it is reasonable that the gain brought by LOC with DIM is less obvious compared with MCT. LOC only stores the hot metadata at the site where it is produced. The hot metadata is read many times by the same site when using MCT or DIM. Thus, LOC always reduces the time to read hot metadata and always outperforms the centralized strategy. In addition, figure 14 shows that LOC is generally better (up to 31% for MCT and 34% for DIM) than the centralized strategy while DHT and REP may be worse than the centralized strategy in terms of execution time of multi-task jobs.

However, the centralized strategy may outperform DHT and REP when using MCT and DIM. DMM-Chiron gathers the data to the master site (WEU) for query jobs. The input data of the jobs, which process the output of these query jobs, is centralized at the master site. In addition, MCT and DIM are data location aware and schedule most of the tasks where the input data is. In this case, the centralized strategy stores the hot metadata at the master site, which is read many times by the same site during execution, while DHT and REP distribute the hot metadata to other sites. As a result, the centralized strategy outperforms DHT and REP. However, in this case, similar to the centralized strategy, LOC also places the hot metadata at the site where the metadata is produced, *i.e.*

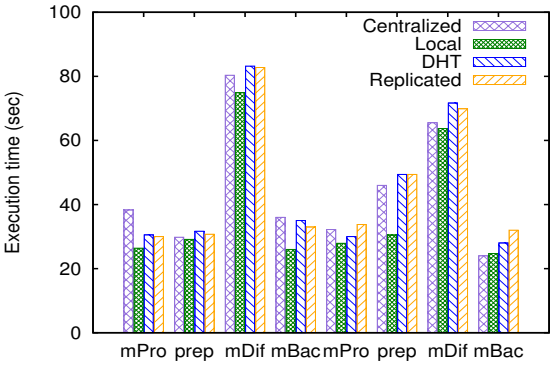


Figure 14: Execution time of multi-task jobs on the Montage SWf of 0.5 degree with MCT/DIM. The left four groups, *i.e.* left mPro, prep, mDif and mBac, represent the execution with MCT while the right four groups represent the execution with DIM.

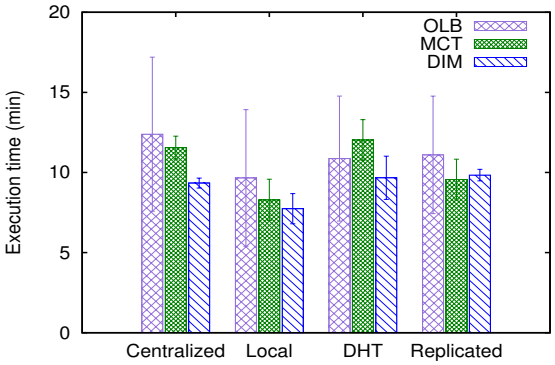


Figure 15: Execution time of the Montage SWf (0.5 degree) with different scheduling algorithms. Avg. intermediate data shown in parenthesis.

the master site. As a result, LOC can always outperform the centralized strategy in terms of overall SWf execution time.

Figure 15 shows the different performance of the scheduling algorithms. The performance of MCT is better than that of OLB and MCT when using the centralized or LOC strategies. The performance of MCT is the worst when using DHT. We attribute the reason to the wrong estimation of the time to transfer metadata. When using REP, the performance of DIM becomes slightly poor since DIM is based on a centralized metadata management strategy and it is difficult to estimate the time to transfer hot metadata with the distributed and replicated hot metadata. The combination of LOC and DIM reduces the overall SWf execution time up to 38%, which is much better than the best result (28%) presented in [36], of the execution time of Montage compared with the combination of the centralized strategy and OLB. In addition, the advantage of the combination of LOC and DIM is up to 55% (by simulation based on the real execution of Buzz) in terms of the execution time of Buzz with 60 MB input data (not shown in Figure 15).

7.4 Performance of Dynamic Hot Metadata Identification

We now evaluate the performance of dynamic hot metadata identification, by comparison with the static approach. We assume that the job metadata, *e.g.* job start time, job end time, is wrongly tagged by users as hot metadata. But in fact, the job metadata is cold while the file and task metadata is hot during execution. Thus, by using dynamic hot metadata identification, the job metadata is turned to cold metadata and the file and task metadata is identified as hot metadata. However, the static approach always takes job metadata as hot. In the experiment, we use LOC to manage the hot metadata.

Figure 16 shows that the advantage of dynamic hot metadata identification is up to 26% compared to the static identification in terms of the total SWf execution. The advantage is up to 40% in terms of multi-task job, *e.g.* mBackground. In addition, the combination of dynamic hot metadata identification and DIM is up to 37% better (in terms of total SWf execution) and 55% (in terms of the execution time of mBackground) compared with the combination of static identification and OLB.

8 Conclusion

Efficient metadata handling is critical for the performance of large-scale SWf execution in a multisite cloud. In this paper, we proposed to dynamically identify and exploit the hot metadata for efficient SWf scheduling in a multisite cloud, using a distributed approach. Our solution includes a distributed architecture with dynamic hot metadata identification and three hot metadata management strategies: two strategies adapted from related work (DHT and REP) and a new strategy (LOC) that stores the hot metadata at the site

where it is generated.

We implemented our approach within a multisite SWfMS, using an RDBMS to manage the metadata. We also adapted three scheduling algorithms, *i.e.* OLB, MCT and DIM, to perform multisite scheduling by provisioning hot metadata. Our approach provides efficient access to hot metadata, hiding inter-site network latencies and remaining non-intrusive and easy to deploy.

We validated our approach by deploying the three metadata management strategies and running real-life SWfs with different scheduling algorithms in a multisite cloud (using Azure). Our experimental results show an improvement of up to 55% for the whole SWf's execution time and 64% for specific highly-parallel jobs, compared to state-of-the-art centralized and OLB solutions, at no additional cost. In addition, the results show that dynamic hot metadata identification is up to 40% better (for highly-parallel jobs) compared with static hot metadata identification. Furthermore, our experiments show that, although no single decentralized strategy can well fit all SWf structures, our proposed hot metadata strategy, *i.e.* LOC, always outperforms other strategies in terms of overall SWf execution time.

For genericity, we validated our solution in a multisite cloud, which is the handiest way for scientists to execute large scale experiments. In this case, our solution is useful to reduce the monetary cost of using the multisite cloud while maximizing the use of available resources. However, when there is no budget for such cloud-based execution, our approach could still be used in large-scale experimental testbeds that federate multiple sites with free access to researchers and scientists, *e.g.* Grid5000 [8], FutureGrid [6] and PlanetLab [5]. All these platforms share the same vision: they are freely accessible to scientists, support high quality, reproducible experiments and allow a complete access to the nodes' hardware in exclusive mode - from one node to the whole infrastructure (i.e. Hardware-as-a Service). In these contexts, the initial assumptions of our solution (*e.g.* high latency inter-site, low latency intra-site, etc.) still hold and our proposed solution could well reduce the execution time of SWfs by more than 50%. This translates into better resource usage, making the underlying platforms (more) energy sustainable.

We end this conclusion by discussing the opportunity of hot metadata management for multisite SWfs. Note that we did not argue that handling metadata as a whole (*i.e.* without distinction between hot and cold) is inefficient. As discussed in Section 2, many of today's file systems do not make any distinction when processing metadata, and some have very good performance. However, these systems are general purpose and deployed at a single site. In this paper, we showed that in the context of multisite SWfs, where thousands of tasks are executed across sites, separating hot metadata reduces execution time, yet at no additional cost.

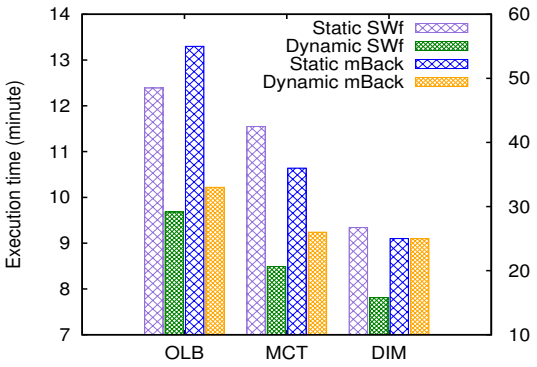


Figure 16: Execution time of the mBackground job and the whole Montage SWf (0.5 degree) with static/dynamic hot metadata identification. SWf represents the execution time of the whole SWf (left Y-axis) and mBack corresponds to the execution of mBackground Job (right Y-axis).

Our focus in this paper was on handling metadata in a distributed way in order to improve job/task execution time when processing a large number of data pieces. While our techniques show an improvement with respect to centralized management, we also notice that when the scale of the SWf and the size of data become larger, there is a degradation in the performance of DMM-Chiron (see Figure 8) due to the increase of intermediate data transfers. To mitigate this degradation, in-memory caching techniques can be used in order to reduce the time to read or write data in a multisite cloud environment.

Finally, we have only considered the case of a homogeneous multisite environment, where each site has the same amount of VMs of the same type. While this configuration is often utilized, in several cases multisite clouds are heterogeneous (different number of resources, different VMs sizes). A next

step in this path would be to account for these variations in order to balance the hot metadata load according to the site's computing capacity.

Acknowledgment

This work has been partially funded by the MSR - Inria Joint Centre (Z-CloudFlow project), the ANR OverFlow project, and the EU H2020 programme (HPC4e project), MCTI/RNP-Brazil, CNPq, FAPERJ, and Inria (MUSIC project). It has been performed (for the members of Inria and LIRMM) in the context of the Computational Biology Institute (http://ibc-montpellier.fr). The experiments were carried out using the Azure infrastructure provided by Microsoft in the context of the Z-CloudFlow project.

References

[1] Alice Collaboration. `http://aliceinfo.cern.ch/general/index.html`.

[2] Apache Hadoop. `http://hadoop.apache.org/`.

[3] Apache HBase. `http://hbase.apache.org`.

[4] Azure Speed Test. `http://www.azurespeed.com/`.

[5] Chameleon. `https://www.chameleoncloud.org`.

[6] FutureGrid. `http://futuregrid.org`.

[7] Giraffa. `https://code.google.com/a/apache-extras.org/p/giraffa/`.

[8] Grid5000. `https://www.grid5000.fr`.

[9] Lustre - OpenSFS. `http://lustre.org/`.

[10] USGS ANSS - Advanced National Seismic System. `http://earthquake.usgs.gov/monitoring/anss/`.

[11] S. R. Alam, H. N. El-Harake, K. Howard, N. Stringfellow, and F. Verzelloni. Parallel I/O and the metadata wall. In *Workshop on Parallel Data Storage (PDSW)*, pages 13–18, 2011.

[12] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M. Su, and K. Vahi. Characterization of scientific workflows. In *Workshop on WFs in Support of Large-Scale Science*, pages 1–10, 2008.

[13] S. A. Brandt, E. L. Miller, D. D. E. Long, and L. Xue. Efficient metadata management in large distributed storage systems. In *IEEE NASA Goddard Conf. on Mass Storage Systems and Technologies*, pages 290–298, 2003.

[14] P. F. Corbett and D. G. Feitelson. The vesta parallel file system. *ACM Trans. on Computer Systems (TOCS)*, 14(3):225–264, 1996.

[15] E. Deelman, S. Callaghan, E. Field, H. Francoeur, R. Graves, N. Gupta, V. Gupta, T. H. Jordan, C. Kesselman, P. Maechling, J. Mehringer, G. Mehta, D. Okaya, K. Vahi, and L. Zhao. Managing large-scale workflow execution from resource provisioning to provenance tracking: The cybershake example. In *IEEE Int. Conf. on e-Science and Grid Computing*, pages 14–14, 2006.

[16] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good. The cost of doing science on the cloud: The montage example. In *Int. Conf. for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1–12, 2008.

[17] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.

[18] J. Dias, E. Ogasawara, D. De Oliveira, F. Porto, P. Valduriez, and M. Mattoso. Algebraic dataflows for big data analysis. In *IEEE Int. Conf. on Big Data*, pages 150–155, 2013.

[19] A. Gehani, M. Kim, and T. Malik. Efficient querying of distributed provenance stores. In *ACM Int. Symp. on High Performance Distributed Computing (HPDC)*, pages 613–621, 2010.

[20] S. Ghemawat, H. Gobioff, and S. Leung. The Google File System. *SIGOPS Operating Systems Review*, 37(5):29–43, 2003.

[21] J. Hsieh, T. Kuo, and L. Chang. Efficient identification of hot data for flash memory storage systems. *ACM Trans. on Storage (TOS)*, 2(1):22–40, 2006.

[22] S. Jin and A. Bestavros. Greedydual* web caching algorithm: exploiting the two sources of temporal locality in web request streams. *Computer Communications*, 24(2):174–183, 2001.

[23] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi. Characterizing and profiling scientific workflows. *Future Generation Computer Systems (FGCS)*, 29(3):682 – 692, 2013.

[24] A. W. Leung, M. Shao, T. Bisson, S. Pasupathy, and E. L. Miller. Spyglass: Fast, scalable metadata search for large-scale storage systems. In *USENIX Conf. on File and Storage Technologies (FAST)*, pages 153–166, 2009.

[25] J. J Levandoski, P. Larson, and R. Stoica. Identifying hot and cold data in main-memory databases. In *Int. Conf. on Data Engineering (ICDE)*, pages 26–37, 2013.

[26] J. Liu, E. Pacitti, P. Valduriez, and M. Mattoso. A survey of data-intensive scientific workflow management. *Journal of Grid Computing*, 13(4):457–493, 2015.

[27] J. Liu, E. Pacitti, P. Valduriez, and M. Mattoso. Scientific workflow scheduling with provenance data in a multisite cloud. *Transactions on Large-Scale Data- and Knowledge-Centered Systems (TLDKS)*, 33:80–112, 2016.

[28] J. Liu, E. Pacitti, P. Valduriez, D. De Oliveira, and M. Mattoso. Multi-objective scheduling of scientific workflows in multisite clouds. *Future Generation Computer Systems (FGCS)*, 63:76–95, 2016.

[29] T. Malik, L. Nistor, and A. Gehani. Tracking and sketching distributed data provenance. In *Int. Conf. on e-Science*, pages 190–197, 2010.

[30] N. Megiddo and D. S. Modha. ARC: A self-tuning, low overhead replacement cache. In *USENIX Conf. on File and Storage Technologies (FAST)*, 2003.

[31] Ethan L. Miller and Randy H. Katz. RAMA: An easy-to-use, high-performance parallel file system. *Parallel Computing*, 23(4):419–446.

[32] E. Ogasawara, J. Dias, F. Porto, P. Valduriez, and M. Mattoso. An algebraic approach for data-centric scientific workflows. *Proceedings of the VLDB Endowment (PVLDB)*, 4(12):1328–1339, 2011.

[33] E. S. Ogasawara, J. Dias, V. Silva, F. S. Chirigati, D. de Oliveira, F. Porto, P. Valduriez, and M. Mattoso. Chiron: a parallel engine for algebraic scientific workflows. *Concurrency and Computation: Practice and Experience*, 25(16):2327–2341, 2013.

[34] M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems, Third Edition.* Springer, 2011.

[35] L. Pineda-Morales, A. Costan, and G. Antoniu. Towards multi-site metadata management for geographically distributed cloud workflows. In *IEEE Int. Conf. on Cluster Computing*, pages 294–303, 2015.

[36] L. Pineda-Morales, J. Liu, A. Costan, E. Pacitti, G. Antoniu, P. Valduriez, and M. Mattoso. Managing hot metadata for scientific workflows on multisite clouds. In *IEEE Int. Conf. on Big Data*, pages 390–397, 2016.

[37] D. Saha, A. Samanta, and S. R. Sarangi. Theoretical framework for eliminating redundancy in workflows. In *IEEE Int. Conf. on Services Computing*, pages 41–48, 2009.

[38] F. Schmuck and R. Haskin. GPFS: A shared-disk file system for large computing clusters. In *USENIX Conf. on File and Storage Technologies (FAST)*, pages 231–244, 2002.

[39] M. Stonebraker and U. Cetintemel. "one size fits all": an idea whose time has come and gone. In *Int. Conf. on Data Engineering (ICDE)*, pages 2–11, 2005.

[40] M. Stonebraker, S. Madden, D. J. Abadi, and N. Hachem. The end of an architectural era: Time for a complete rewrite. In *Int. Conf. on Very Large Data Bases (VLDB)*, pages 1150–1160.

[41] A. Thomson and D. J Abadi. CalvinFS: consistent wan replication and scalable metadata management for distributed file systems. In *USENIX Conf. on File and Storage Technologies (FAST)*, pages 1–14, 2015.

[42] J. Wang, S. Wu, H. Gao, J. Li, and B. C. Ooi. Indexing multi-dimensional data in a cloud system. In *ACM SIGMOD Int. Conf. on Management of Data (SIGMOD)*, pages 591–602, 2010.

[43] J. M. Wozniak, T. G. Armstrong, M. Wilde, D. S. Katz, E. L. Lusk, and I. T. Foster. Swift/t: scalable data flow programming for many-task applications. In *ACM SIGPLAN Symp. on Principles and Practice of Parallel Programming*, pages 309–310, 2013.

[44] S. Wu, D. Jiang, B. C. Ooi, and K. Wu. Efficient b-tree based indexing for cloud data processing. *Proceedings of the VLDB Endowment (PVLDB)*, 3(1-2):1207–1218, 2010.

[45] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, pages 10–10, 2010.

[46] M. J. Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1):31–60.

[47] D. Zhao, C. Shou, T. Maliky, and I. Raicu. Distributed data provenance for large-scale data-intensive computing. In *IEEE Int. Conf. on Cluster Computing (CLUSTER)*, pages 1–8, 2013.

[48] D. Zhao, Z. Zhang, X. Zhou, T. Li, K. Wang, D. Kimpe, P. Carns, R. Ross, and I. Raicu. Fusionfs: Toward supporting data-intensive scientific applications on extreme-scale high-performance computing systems. In *IEEE Int. Conf. on Big Data*, pages 61–70, 2014.