Towards Multi-Site Metadata Management for Geographically Distributed Cloud Workflows

Luis Pineda-Morales
Microsoft Research - Inria Joint Centre
luis.pineda-morales@inria.fr

Alexandru Costan
IRISA / INSA Rennes
alexandru.costan@irisa.fr

Gabriel Antoniu
Inria Rennes - Bretagne Atlantique
gabriel.antoniu@inria.fr

***Abstract*—With their globally distributed datacenters, clouds now provide an opportunity to run complex large-scale applications on dynamically provisioned, networked and federated infrastructures. However, there is a lack of tools supporting data-intensive applications across geographically distributed sites. For instance, scientific workflows which handle many small files can easily saturate state-of-the-art distributed filesystems based on centralized metadata servers (e.g. HDFS, PVFS). In this paper, we explore several alternative design strategies to efficiently support the execution of existing workflow engines across multi-site clouds, by reducing the cost of metadata operations. These strategies leverage workflow semantics in a 2-level metadata partitioning hierarchy that combines distribution and replication. The system was validated on the Microsoft Azure cloud across 4 EU and US datacenters. The experiments were conducted on 128 nodes using synthetic benchmarks and real-life applications. We observe as much as 28% gain in execution time for a parallel, geo-distributed real-world application (Montage) and up to 50% for a metadata-intensive synthetic benchmark, compared to a baseline centralized configuration.**

***Keywords*—metadata management, multi-site clouds, scientific workflows.**

I. Introduction

As we move to the world of Big Data, single-site processing becomes insufficient: large scale scientific applications can no longer be accommodated within a single datacenter [1]. To enable such processing, a promising approach consists in simultaneously provisioning resources on *multiple datacenters* at different geographical locations [2]. This may have several benefits: resilience to failures, better locality (e.g., by moving computation close to data or viceversa), elastic scaling to support usage bursts, user proximity through content delivery networks, etc. In this context, sharing, disseminating and analyzing the data sets may result in frequent large-scale data movements across widely distributed sites. Studies show that the inter-datacenter traffic is expected to triple in the following years [3], [4].

Workflows are the perfect illustration of such data-driven applications [5]. They describe the relationship between individual computational tasks (usually standalone binaries) and their input and output data in a declarative way. Workflow tasks typically exchange data through temporary files stored on some shared storage system. In this setup, the workflow engines are basically schedulers that build and manage a task-dependency graph based on the tasks' input/output files [6]. Workflow data may be distributed on multiple sites for availability and reliability reasons. Hence, deploying workflows on multiple datacenters comes as a natural approach, allowing them to elastically balance performance and cost as they execute. This opens the possibility to globally optimize the performance of multiple workflows that share a common public cloud infrastructure. Among the notorious examples we recall the 40 PB/year data that is being generated by the CERN LHC. This volume overpasses single site or single institution capacity to store or process data, it requires workflows that span over multiple sites. This was the case for the Higgs boson discovery, for which the processing was extended to the Google cloud infrastructure [7].

Nevertheless, the lack of a *scalable metadata* service is becoming an important performance bottleneck for many cloud-based geographically distributed workflows, as it increases the gap between the workflow's I/O requirements and the storage performance. While extensive research efforts [8], [9] have been dedicated to cloud data management, there has been relatively less progress on optimizing *metadata* management for complex scientific workflows on cloud systems. This is the goal addressed by this paper. The underlying motivations are two-fold. First, most files in even the largest workflows are *small*, with median file sizes in the orders of kilo- or megabytes, yet generated in *large numbers* (exceeding millions). This means that metadata access has a high impact (sometimes being dominant) on the overall workflow I/O. Second, as cloud datacenters are interlinked through *high-latency wide-area networks*, remotely accessing metadata (i.e. when the metadata server is located in a distinct datacenter from the accessing process) may be costly and strongly impact the total makespan of the workflow. Conventional approaches to metadata management in such scenarios have a variety of limitations. Traditional distributed file systems [10], [11], [12], for instance, provide limited optimization for small files and metadata throughput. This is because most such file systems were designed for HPC clusters and not heterogeneous clouds; they have been optimized mainly for scaling the data path (i.e., providing high bandwidth parallel I/O to files that are gigabytes in size) and have limited metadata management scalability, typically relying on the decade-old design of a centralized metadata server.

In this paper, we tackle these problems by leveraging a hybrid distributed/replicated metadata architecture, uniform in-memory hashing, workflow provenance data and lazy updates, all of which collectively achieve the design goal: a *lightweight and scalable geographically distributed metadata service*. To this end, we analyze four strategies for improving concurrent metadata and workflow file I/O performance in multi-site clouds: centralized, replicated on each site, decentralized non-replicated and decentralized with local replication. Out of these four strategies, the first one has been previously demonstrated in separate workflow engine implementations [13] and we use it as a baseline. The remaining three are novel proposals in the context of distributed workflows. Our design is driven

by recent workflow workload studies on traces from several applications domains. We observe that workflows generate many small files and tend to have similar data access patterns, that can be exploited for efficient resolution of metadata. Therefore, we propose to keep all metadata in memory, in a uniform DHT based cache, distributed across cloud datacenters and following a 2-level hierarchy: metadata are first partitioned to the datacenters where they are likely to be used (leveraging information from the workflow engine) and then replicated to other datacenters.

To demonstrate the feasibility of our approach, we implemented a prototype middleware service that integrates all the above strategies. Existing workflow engines can benefit from it on any cloud platform, without requiring any modifications to the original system. We evaluated the prototype with synthetic benchmarks and real-life applications on the Azure cloud using up to 128 nodes. The results show promising scalability and performance: the hybrid strategy can scale almost linearly to 128 nodes, performs up to 1150 operations per second, and halfs the execution time with respect to the state-of-the-art baseline configuration as the number of servers scales in various metadata-intensive, widely distributed workloads.

In summary, this work makes the following contributions:

- A set of design principles for efficient metadata access on multi-site clouds, exploiting the workflows semantics (Section III);
- Four metadata partitioning strategies for distributed workflows: from centralized to distributed management, with and without replication (Section IV);
- A system prototype implementing these strategies, applying DHT-based techniques to manage distributed workflow metadata across geo-distant datacenters, integrated to a fully-fledged public cloud (Section V);
- An experimental evaluation of the benefits of these techniques on up to 128 nodes on 4 datacenters of the Microsoft Azure cloud both with synthetic benchmarks and real-life scientific scenarios (Section VI);
- An analysis of the best matching strategy for different workflow workloads (Section VII).

II. Context

This section describes the specific semantics of scientific workflows and the challenges raised by their execution in a multi-site environment. It explains the terminology used throughout the paper and states the problem we address.

Unlike tightly-coupled applications (e.g., MPI-based) communicating directly via the network, workflow tasks exchange data through files. An obvious solution for this consists in using the cloud's storage service (e.g., Amazon S3 [14]), which however results in high data access latency. Alternatively, recent proposals [8], [15] leverage some of the cloud virtual machines (VMs) allocated to the workflow to deploy a shared intermediate storage system dedicated to (and co-deployed with) the application. Such solutions suffer from an increased pressure on the metadata system, whose performance often cannot scale with the number of files. This phenomenon is accentuated when the workflow is deployed on several datacenters.

A. Core Workflow Features

In order to understand and address these challenges, we studied real-life workflows [16], [17] from several domains (bio-informatics, business, simulations etc.) and identified a set of common characteristics:

Many small files. Workflows generate and access a huge number of relatively small files [18]. The natural question is: *what is a small file?* Several scientific domains such as climatology, astronomy, and biology generate data sets that are most conveniently stored in small files: 20 million images hosted by the Sloan Digital Sky Survey with an average size of less than 1 MB [19], up to 30 million files averaging 190 KB generated by sequencing the human genome [20]. In our (cloud) context, a small file is any file for which it makes no sense to impose striping (e.g. no larger than the block size set to 64 MB in HDFS). Note however that the strategies presented in this paper are applicable to files of any size (even multi-gigabyte).

Common data access patterns. The most frequent data access models are: *pipeline, gather, scatter, reduce and broadcast*. Further studies [21] show that the workflow applications are typically a combination of these patterns.

Write once, read many times. During an experiment, scientific applications typically generate a large number of files, that are no longer updated, instead they are read many times by subsequent tasks or at the end of the workflow. Moreover, the batch jobs composing the workflows have well-defined data and metadata passing schemes for these read/write operations: the workflow engine queries the metadata service to retrieve the job input files, retrieves them, executes the job and stores the metadata and data of the final results.

These observations enable us to build an accurate model of the workflow execution and exploit its specificities when exploring the design space for our proposed strategies (cf. Section IV). We notice for instance that the file transfer times are dominated by the metadata access time in scenarios involving many small-files.

B. Problem Statement

From single- to multi-site workflows. The cloud *site (datacenter)* is the largest building block of the cloud. It contains a broad number of compute nodes, providing the computational power, available for rent. The datacenter is organized in a hierarchy of switches, which interconnect the compute nodes and the datacenter itself with the outside world, through the Tier 1 ISP. Multiple output endpoints (i.e., through Tier 2 switches) are used to connect the datacenter to the Internet. A cloud, particularly a public one, is made up of several sites (datacenters), geographically distributed across the globe. An application that has multiple instances running in several deployments across multiple cloud datacenters is referred to as a *multi-site application*.

Many workflows benefit from distribution across sites, as they aggregate resources beyond the limit of a single cloud datacenter. Besides the need for additional compute resources, workflows have to comply with several cloud providers requirements, which force them to be deployed on geographically distributed sites. For instance, in the Azure cloud, there is a limit of 300 cores per user *deployment* (i.e. set of VMs deployed at once) within a datacenter, for load balancing; any

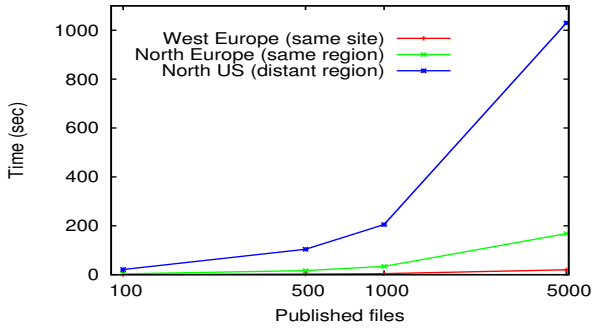

Fig. 1: Average time for file-posting metadata operations performed from the West Europe datacenter, when the metadata server is located within the same datacenter, the same geographical region and a remote region (log scale).

application requiring more compute power likely needs to be distributed across several sites.

Huge latency for remote metadata access. Conventional cluster file systems are optimized mainly for scaling the data path and lack support for the geographical distribution of the metadata. In such cases, users are left with the option of using a single metadata server or a federation of metadata servers within a single site, serving the whole multi-site application. Even VMs accessing data on their local datacenter may need to yield expensive remote calls to the datacenter where the metadata registries are located. Users have to set up their own tools to move metadata between deployments, through direct endpoint to endpoint communication (e.g. GridFTP, scp, etc.). This baseline option is relatively simple to set in place, using the public endpoint provided for each deployment.

The major drawback in this scenario is the high latency between sites. The numerous metadata requests have to traverse the slow WANs connecting the datacenters (which are the property of the ISPs, so out of the control of the cloud providers), limiting drastically the achieved throughput of the workflow. A simple experiment conducted on the Azure cloud and isolating the metadata access times for up to 5000 files (Figure 1) confirms that remote metadata operations take orders of magnitude more than local ones. This has a high impact on the overall workflow makespan, particularly for workflows handling many small files, when the metadata operations are dominant. Clearly, the paradigm shift towards multi-site workflow deployments calls for appropriate metadata management tools, that build on a consistent, global view of the entire distributed datacenter environment. This is precisely the goal targeted by this paper.

III. Design Principles

In this section we delve into the design space of our proposed strategies and audit the tradeoffs between different design decisions. In essence, our proposal implements a hierarchical metadata partitioning in order to hide the latency and reduce I/O through three simple strategies: full replication, full distribution and an intermediate, hybrid approach.

A. Hybrid Distributed/Replicated DHT Based Architecture

Most existing works on metadata partitioning simply consider the namespace partitioning and the distribution of shares to servers (as detailed in Section VIII). Such pure partitioning approaches may bring potential performance and scalability problems when used in widely distributed environments, like the multi-site clouds. For instance, it is difficult to apply updates, since these may incur costly communications among geographically distributed servers.

In this paper, we propose a hybrid approach mixing distribution and replication of metadata to address these problems. With this approach, updates can be applied by only updating shares in one datacenter and propagating them to other datacenters. In this setting and depending on the data structures used (e.g. trees, hashes), the time complexity for search operations can vary drastically. We argue that hashing is a good option for metadata scattering as distributed hash tables (DHT) have proved to be highly scalable in practice with a constant-time query cost. Also, a flat namespace implemented by hash tables exposes a simpler and less error-prone interface avoiding expensive operations required by others data structures (e.g. distributed trees). Thus, the hybrid approach in conjunction with the DHT can significantly reduce the user perceived response latency for update accesses.

B. Uniform In-Memory Caching

A key observation from the workflow traces is that traditional workflow metadata design incurs an excessive number of disk operations because of metadata lookups: the file's metadata must be read from disk into memory in order to find the file itself. While insignificant at a small scale, multiplied over millions of small files, using disk I/Os for metadata is a limiting factor for read throughput.

We therefore opted to keep all metadata in memory, which we make practical by reducing the per file metadata. That is we only store the information necessary to locate files and we don't keep additional POSIX type metadata, like permissions, since they are never used in a scientific workflow (i.e. during the workflow execution the files produced are used by the same user(s)). To store the metadata, we rely on a dedicated cache, uniformly distributed across all workflow's datacenters. As opposed to existing distributed caches (e.g. Memcached [22]) which aggregate memory resources from the actual deployment of the application, we argue in favor of a separate cache layer. This allows the data tier to scale independently and guarantees non-intrusiveness on the workflow execution. Our standard cache tier provides high availability by having a primary and a replica cache. If a failure occurs with the primary cache, the replica cache is automatically promoted to primary and a new replica is created and populated.

C. Leverage Workflow Metadata for Data Provisioning

Adequate *data provisioning* is crucial when scientific workflows run on geographically distributed clouds. A task might require a very large file stored at a distant location. If the data is not in place, idle execution times occur, impacting the workflow's makespan. It is therefore essential to know in advance *what* data would be needed, *when* and *where*.

Some workflow execution engines already leverage the workflow's metadata (i.e. data provenance, data dependencies between tasks) for smarter task scheduling strategies [13]. We claim that a similar approach can be adopted to optimize data provisioning. By efficiently querying the workflow's metadata, we can obtain information about data location and data dependencies which allow to proactively move data between nodes in distant datacenters before it is needed, keeping idle times as low as possible. In this paper we study the first step towards such optimizations: a reliable metadata management

service that ensures that metadata operations are also carried out efficiently across multi-site clouds.

D. Eventual Consistency for Geo-Distributed Metadata

In a system where metadata management instances are geographically distributed, metadata operations submitted by a node might take relatively long time to propagate. In order to maintain a fully consistent state of the system, all nodes would have to wait until the newest operations are acknowledged by all the instances. This is evidently inefficient considering the potentially long distances between instances and the large number of metadata operations usually performed.

Therefore we argue for a system where every metadata update is guaranteed to be *eventually* successful. Rather than using file-level eager metadata updates across datacenters, we favor the creation of batches of updates for multiple files. We denote this approach *lazy metadata updates*: it achieves low user-perceived response latency and high scalability in a widely distributed environment by asynchronously propagating metadata updates to all replicas after the updates are performed on one replica. Yet, this lazy approach only guarantees eventual consistency, meaning that if any other tasks try to access the distant metadata at the same time, the result will be undefined. However, eventual consistency is perfectly in line with the observations from the workflows traces. The typical data access pattern is write once/read many times, with readings occurring in two situations. For intermediate results, data is used as input for the next task(s) in the workflow, but in these cases the engine scheduler takes care to schedule the task close to the data production nodes (i.e. on the same node, in the same datacenter) so the metadata updates are instantly visible here. For final results, data might be accessed from remote locations, but typically this a posteriori analysis takes places long after the workflow execution has finished, leaving enough time for the lazy updates of the metadata to propagate. So, in both cases, the eventual consistency is not affecting the application performance or coherence.

IV. Strategies for Multi-Site Metadata Management

In the remainder of this paper we identify as *metadata registry* to the instance or set of distributed instances in charge of managing metadata entries (shown as diamonds in Figure 2). We denote as a *read* the action of querying the metadata registry for an entry, and as a *write* the publishing of a new entry. Note that since a metadata entry can be created by one node and subsequently updated by others, a *write* operation actually consists of a look-up read operation to verify whether the entry already exists, followed by the actual write.

We analyze the impact of high latencies as a consequence of the physical distance between an execution node and the corresponding metadata registry instance. In the following, we use the following terms to qualify physical distance:

a) *local* - the node and the metadata registry are in the same datacenter;
b) *same-region* - the node and the registry are in different datacenters of the same geographic region (e.g. Europe);
c) *geo-distant* - the datacenters are in different geographic regions (e.g. one in Europe, the other in the US).

Both *b)* and *c)* can also be referred to as *remote* scenarios. Our design accounts for several datacenters in various geographic regions in order to cover all these scenarios. We have focused on four metadata management strategies, detailed below and depicted in Figure 2. The dashed lines represent a very large physical distance between datacenters; the ones on the "same side" of the line fit the *same region* scenario, whereas datacenters on "different sides" are *geo-distant*.

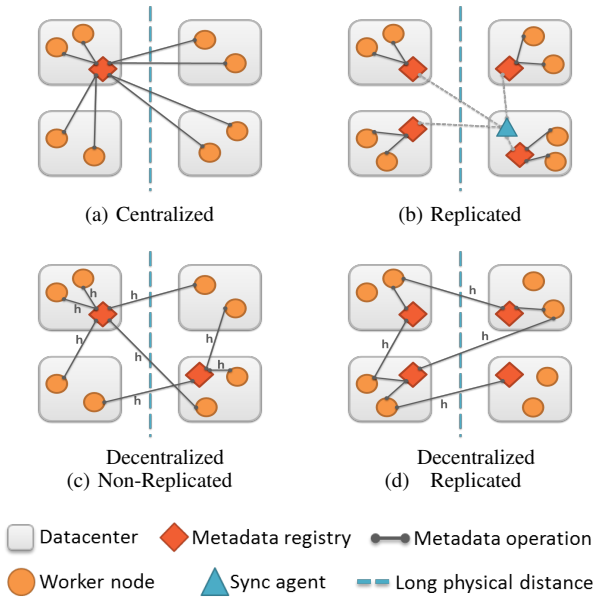

Fig. 2: Strategies for geographically distributed metadata management.

A. Centralized Metadata (Baseline)

As detailed in Section VIII, traditional metadata management relies on a centralized metadata server (e.g. HDFS). We therefore first consider a single-site, single-instance metadata registry, arbitrarily placed in any of the datacenters (Fig. 2a), which will serve as a *state-of-the-art baseline*. In this setup, the application processes are run on nodes which are distributed both locally and remotely with respect to the site of the metadata registry. In the case of non-local accesses to the centralized metadata, high-latency operations may occur.

B. Replicated Metadata (On Each Site)

Our second strategy builds on the assumption that local metadata operations are naturally faster than remote ones. Given a set of distributed datacenters, we place a local metadata registry instance in each of them, so that every node could locally perform its metadata operations. At this point, metadata information would be processed quickly, but it would only be known at local level. A *synchronization agent* iteratively queries all registry instances for updates, then synchronizes all metadata instances. The strategy is depicted in Figure 2b: the synchronization agent is presented as a triangle and can be placed in any of the sites. The dotted lines between the agent and the registry instances represent the synchronization communication. Given the previous design considerations, we claim that this strategy would perform at its best in a scenario where metadata operations are not so frequent within a task, for instance, a workflow which deals with few, very large files.

C. Decentralized, Non-Replicated Metadata

Even if a metadata registry instance is locally deployed in each datacenter, the previous approach is still centralized

(a) Without local repication (b) With local replication

Fig. 3: Decentralized metadata: local replicas avoid costly remote operations.

in that it relies on a single synchronization agent, which can become a potential bottleneck, particularly in the case of a metadata-intensive workflow. Taking this into consideration, our third strategy favors *decentralization*, based on a Distributed Hash Table (DHT) [23]. We maintain an instance of the metadata registry in each of the active sites. Every time a new entry is written to the metadata registry, we apply a hash function to a distinctive attribute of the entry (e.g. the file name) to determine the site where the entry should be stored. A similar procedure applies for read operations to identify the metadata registry instance in charge of a given entry. Note that in this case the metadata is partitioned across the registry instances, so the contents of these instances are no longer identical in this strategy: each instance stores a share of the global set of metadata entries.

This approach involves remote operations: on average only $1/n$ of the operations would be local (n = number of sites). However, as the registry is now distributed, metadata queries are now processed in parallel, potentially faster.

D. Decentralized Metadata With Local Replication

As observed in Figure 1, local metadata operations take negligible time in comparison with remote ones, especially when the total number of operations becomes large, which is the case of data-intensive scientific applications. Keeping this in mind, we propose to enhance the DHT-like approach with a local replica for each entry (Figure 2d).

Every time a new metadata entry is created, it is first stored in the local registry instance. Then, its hash value h is computed and the entry is stored in its corresponding remote site. When h corresponds to the local site, the metadata is not further replicated. For read operations we propose a two-step hierarchical procedure: the entry is first looked for in the local metadata registry instance; with local replication, assuming uniform metadata creation across the sites, we have twice the probability to find it locally than with the *non-replicated* approach. If the entry is not available locally, it is searched for in its remote location, determined by its hash value. Compared to the previous scheme (without replication), we expect that, overall, the gain (in terms of latency and bandwidth) due to an enhanced probability to succesfully look up metadata locally will be higher than the extra overhead added by local replication to the previous scheme.

To illustrate the benefits of local replication, we take the following scenario involving two nodes *n1* and *n2* running in the same site *s1*: *n1* writes an entry to the metadata registry, read by *n2*, the location of the entry being determined by a hash function. Assume that the hash value places the entry in a geo-distant site *s2*. In the *non-replicated* approach, both read and write operations would be remote and take up to 50x longer than a local operation (Figure 3a). With *local replication*, the write operation keeps a local copy and the subsequent read is performed locally, saving one costly remote operation and making reads up to 50x faster (Figure 3b).

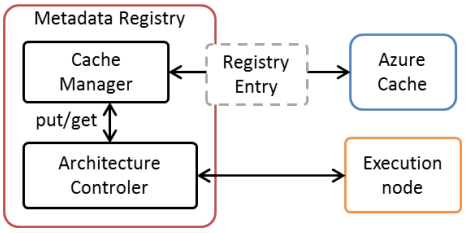

Fig. 4: Metadata Registry modules and interaction.

V. Implementation

Our proposed strategies are designed as a general multi-site metadata handling paradigm and not aimed to a specific cloud kit. For validation purposes, in this paper we use the Microsoft Azure Cloud [2] as a concrete example to demonstrate how to implement them in practice at Platform-as-a-Service (PaaS) level. We rely on the Azure SDK v2.5 for .NET which provides the necessary libraries for accessing and manipulating Azure features. The architectural overview of the metadata middleware is shown in Figure 4 and its components are discussed below.

The Metadata Registry stays at the core of our implementation, as it serves as communication channel and distributed synchronization manager between all nodes in the network. From previous observations on Azure, in-memory storage access outperforms regular database storage by a factor of 10 [24], thus we opted to implement the registry on top of the Azure Managed Cache service [25]. To manage concurrent access to the registry we leveraged the Optimistic Concurrency Model of Azure Cache, which does not pose locks on the registry object during a metadata operation (remember that often workflow data is written only once).

The Cache Manager is our interface with Azure Cache by means of an external .NET library. It exposes a set of internal methods to carry out (cache-managed) metadata operations. This independent module allows to explore different cache alternatives without affecting the application. As an alternative, in the short term we plan to also evaluate our design by replacing Azure Managed Cache with Redis Cache [26].

The Registry Entry is the fundamental metadata storage unit. The Cache Manager is able to *put* (write) and *get* (read) Registry Entries from the registry. An entry can contain any metadata provided it is serializable and includes a unique identifier. For our implementation we took the base case of a file uniquely identified by its name and containing a set of its locations within the network. The scope of the registry can be easily extended by defining different types of Registry Entries.

The Architecture Controller allows to switch between metadata management strategies. The desired strategy is provided as a parameter and can be dynamically modified as new jobs are executed. The modularity of our design allows us to add or remove strategies on the fly, using a simple plug-and-play approach, without altering the application flow.

The Execution Nodes are Virtual Machines running on the Microsoft Azure Cloud Service. We designed three types of instances based on Azure's PaaS-level abstractions (Web Roles and Worker Roles):

a) *The Worker Nodes* execute the application tasks and are implemented on top of Azure Worker Roles.

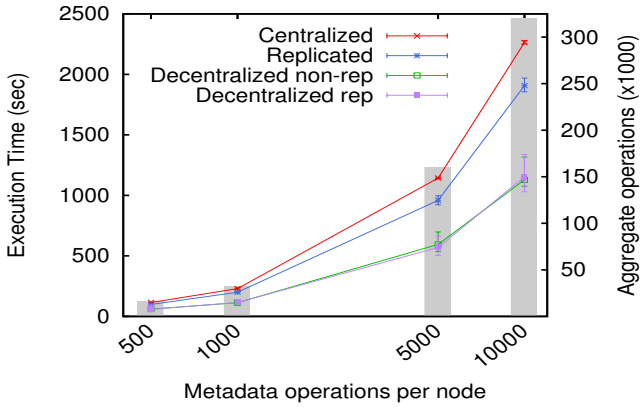

Fig. 5: Average execution time for a node performing metadata operations. The grey bars indicate the aggregated number of operations in one execution.

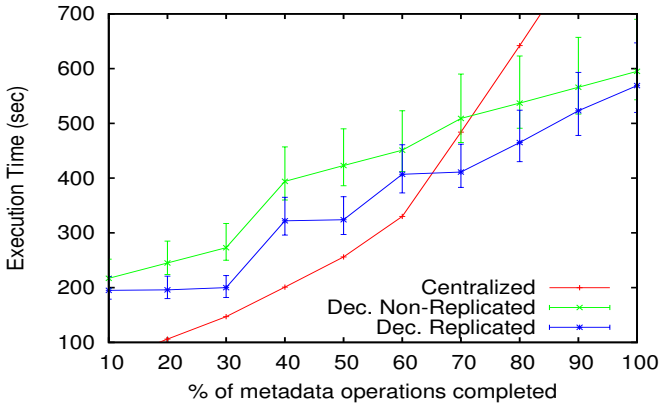

Fig. 6: Percentage of operations completed along time by each of the decentralized strategies: non-replicated (DN) and with local replication (DR).

b) *The Control Node* manages the application execution. Implemented as an Azure Web Role, it includes a simple web interface to provide configuration parameters. It launches the execution of the application by sending control messages to all the Worker Nodes.
c) *The Synchronization Agent* is a Worker Node in charge of synchronizing the metadata registry instances in the *replicated* strategy. It sequentially queries the instances for updates and propagates them to the rest of the set.

VI. Experimental Evaluation

In this section we answer the following questions: *what is the expected throughput and the time overhead of metadata decentralization?* (Section VI-B); *how do the proposed strategies support high concurrency and how are they performing under different scales?* (Section VI-C); *what is the performance improvement of real-life workflows?* (Section VI-D).

A. Experimental Setup

Our testbed consisted of four Azure datacenters: two in Europe - North (Ireland) and West (Netherlands) - and two in the US - South Central (Texas) and East (Virginia). We used up to 128 small VMs, each consisting of 1 core and 1.75 GB of memory. For the Metadata Registry we deployed one Basic 512 MB instance of Azure Managed Cache per datacenter. All experiments are repeated at least five times and the reported figures are the average of all runs. To hinder other factors such as caching effects and disk contention, the metadata entries posted to the registry (e.g. create, update or remove) correspond to empty files.

B. Impact of Metadata Decentralization on Makespan

We claim that the efficiency of our approaches becomes more evident in large-scale settings. The goal of the first experiment is to compare the performance of our implementation to the baseline centralized data management as the number of files to be processed increases. For this purpose, we keep a constant number of 32 nodes evenly distributed in our datacenters, while varying the number of entries to be written/read to/from the registry. To simulate concurrent operations on the metadata registry, half of the nodes act as writers and half as readers. Writers post a set of consecutive entries to the registry (e.g. *file1, file2, ...*) whereas readers get a random set of files (e.g. *file13, file201, ...*) from it. We measure the time required for a node to complete its execution, and obtain the average time for completion of all the nodes for each strategy. Figure 5 shows the results.

We observe that for a rather small number of processed entries our strategies do not significantly outperform the centralized baseline in terms of overall execution time, as they represent a gain of slightly more than 1 minute in the best case, which is rather low in our context. We infer that for small settings - up to 500 operations per node - a centralized approach remains an acceptable choice. However, as the number of operations grows, the improvement achieved particularly by the decentralized strategies becomes more evident, yielding up to 50% time gain (i.e. 18.5 minutes in a test with 320,000 operations).

Decentralized strategies: completion time vs. speedup. An interesting observation is that both decentralized approaches seem to overlap. The time for completion in each strategy depends on the time taken by the last active node to execute its very last operation. The curves in Figure 5 do not reflect then the progress of the nodes activity before their completion. For that reason we do not clearly see an advantage of one decentralized strategy over the other. Therefore, in Figure 6 we zoom on the internal execution of the two decentralized strategies (non-replicated and locally replicated), by analysing their progress towards completion. We also show as reference the average progression of the generic centralized approach.

We notice that during most of the execution, particularly between 20% and 70% progress, we get a speedup of at least 1.25 using local replication. This remark is crucial for data provisioning in a distributed scientific workflow: the time gain implies that data location information can be known by the whole network in anticipation, and represents the possibility to move the data between sites before it is needed.

The centralized approach has a fairly good start on average. However, as the execution advances, it slows down in a near-exponential behavior, reaching up to twice the time for completion compared to the decentralized ones. This delay is due to the increasing overload of operations on the centralized metadata registry, doubled by the accumulated latency of distant nodes performing remote operations.

Impact of the geographical location of a datacenter. Finally, we focus on the best and worst performance of the

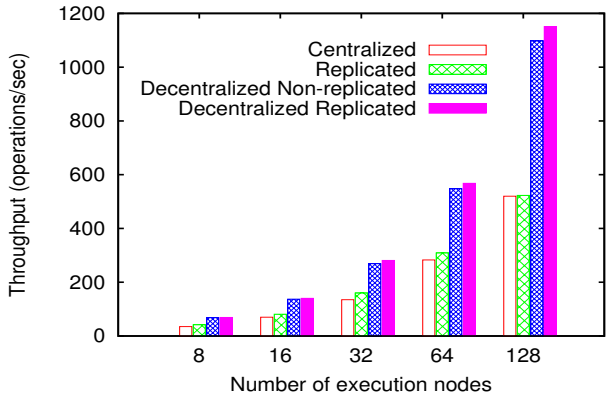

Fig. 7: Metadata throughput as the number of nodes grows.

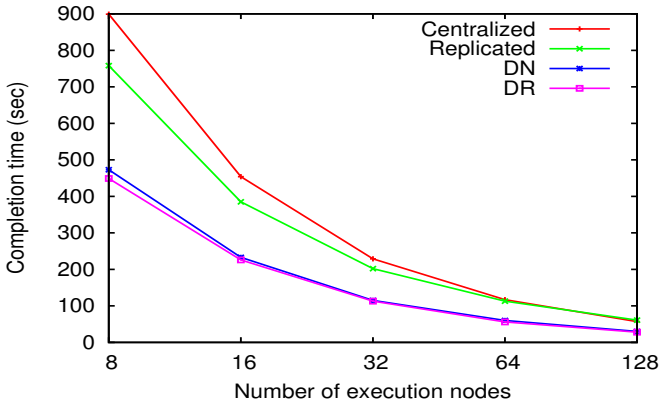

Fig. 8: Completion of 32,000 operations as the set size grows.

decentralized approaches shown in Figure 6. A careful analysis of the results made evident the impact of the physical distance between sites on the metadata performance. If we define a site's centrality as the average distance from it to the rest of the datacenters, the best performance in both decentralized cases corresponds to the nodes executed in the most centric datacenter - East US. Worst cases, on the other hand, correspond to the least centric datacenter, South Central US.

C. Scalability and Concurrency Sensitivity

In our next experiment, we evaluate the performance of our strategies when the number of metadata nodes increases. Note that as in our setup each node acts also as a client this scaling up translates into increased concurrency as well. First, we measure the metadata throughput when increasing the number of nodes from 8 up to 128, with a constant workload of 5,000 operations per node. In Figure 7 we observe that the decentralized implementations clearly win: they yield a linearly growing throughput, proportional to the number of active nodes. We only notice a performance degradation in the replicated scenario for more than 32 nodes. We assert that as the number of nodes grows, the single replication agent might become a performance bottleneck; however, in smaller settings of up to 32 nodes it still behaves efficiently.

To get a clearer perspective on the concurrency performance, we measured the time taken by each approach to complete a constant number of 32,000 metadata operations. Our results (Figure 8) were consistent with the previous experiment, showing a linear time gain for the centralized and decentralized approaches and only a degradation at larger scale for the replicated strategy.

D. Support for Real-Life Workflows

The final set of experiments focus on the benefits brought by our strategies to real-life scientific workflows with representative data access patterns. BuzzFlow [16] is a near-pipelined application that searches for trends and correlations in large scientific publications databases like DBLP or PubMed. Montage [27] is an astronomy application, in which mosaics of the sky are created based on user requests. It includes a split followed by a set of parallelized jobs and finally a merge operation (Figure 9). The workflow jobs were evenly distributed across 32 nodes. We simulated tasks internal computation by defining a *sleep* period for each node. We covered three scenarios: small scale, computation intensive and metadata intensive. Table I summarizes their settings.

Scenario	Small Scale	Comp. Int.	Metadata Int.
Operations / node	100	200	1,000
Computation time / node	1s	5s	1s
Total ops - BuzzFlow	7,200	14,400	72,000
Total ops - Montage	16,000	32,000	150,000

TABLE I: Settings for real-life workflow scenarios

In Figure 10 we compare the makespan of our strategies in the above scenarios. We firstly confirm that at small scale a decentralized approach actually adds overhead to the computation and hence centralized solutions are best for smaller settings, regardless of the workflow layout. We note as well that in computation intensive workflows the low metadata interaction benefits *centralized* replication while penalizing *distributed* replication, which is optimized for metadata intensive workloads. Overall, we assert that our decentralized solutions fit better to complex workflow execution environments, notably metadata intensive applications, where we achieved a 15% gain in a near-pipeline workflow (BuzzFlow) and 28% in a parallel, geo-distributed application (Montage) compared to the centralized baseline. Based on our findings, in the next section we present a best-match analysis of metadata strategies to workflow patterns.

VII. Discussion

In this section we try to answer the question: *which strategy fits what type of workflow on what kind of deployment?* We also examine the limitations of some design choices, their trade-offs and potential solutions.

A. Matching Metadata Strategies with Scientific Workflows

As shown by the previous experiments, the *centralized approach* remains the best option for *small scale workflows*: using few tens of nodes, managing at most 500 files each, running in a *single site*. Low latencies of intra-datacenter transfers coupled with the proximity of data and metadata servers enable a high throughput for data access and reduce the overall workflow makespan.

The *replicated metadata registry with a centralized synchronization agent* serves the needs of workflows manipulating *average sets of very large files* (i.e. tens or hundreds of MBs), where metadata operations are not so frequent. With tasks taking long enough time to process large files, the agent has sufficient time to synchronize the registry instances and to provide consistency guarantees that enable easy reasoning on concurrency at application level.

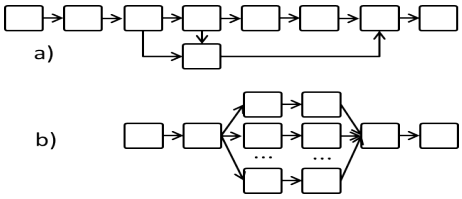

Fig. 9: Real-life workflows: a) BuzzFlow b) Montage

The *decentralized strategies* are expected to perform at their best with workflows managing a *large number of small files*. The *non-replicated approach* targets workflows with *high degree of parallelism* (e.g. following a scatter/gather pattern), where tasks and data are widely distributed across datacenters. With access to metadata remaining linear across sites, as shown previously, the scalability and the throughput performance of the workflow are preserved even for increased workloads. The *locally replicated registry* fits better for workflows with a larger proportion of *sequential jobs* (e.g. with pipeline patterns). We noticed that workflow execution engines schedule sequential jobs with tight data dependencies in the same site as to prevent unnecessary data movements. With our approach, when two consecutive tasks are scheduled in the same datacenter the metadata is available locally. Even when a task is scheduled in a remote site, it will still be able to access metadata in linear time via the hash value.

Our strategies enable metadata to be propagated as soon as the data is created. In a scientific workflow such metadata includes file location(s) information. During a multi-site workflow execution several tasks are scheduled simultaneously in different locations and require pieces of distributed data to start running. Thanks to our solution, tasks would learn about remote data location early enough and could request the data to be streamed as it is being generated, reducing the costly transfer-related idle time.

B. Low Overhead with Eventual Consistency

With respect to consistency semantics, our approach guarantees that each metadata operation applied by concurrent processes takes effect instantaneously in the local datacenter and is then propagated to the other datacenters by means of *lazy updates*. For read operations, we define the completion of the primitive to be the return of the queried metadata. For writes, the completion is the moment when the assigned cache entry is successfully generated in the local datacenter. Our approach has several advantages. On one hand, a writer can produce multiple updates asynchronously and does not have to wait in turn for the corresponding remote replicas to be synchronized. This feature enhances parallelism while still avoiding the need for complex synchronization mechanisms that are typically necessary in stronger consistency models. On the other hand, the lazy updates allow for efficient metadata handling when servers are added to or removed from the system, a common cloud scenario.

Note that our choice for this eventual consistent update model is in line with the data access patterns of most scientific workflows executed on clouds [17]. The intermediate files created by tasks on one site are generally needed only by tasks running within the same datacenter (the scheduler takes care of enforcing this locality policy), while the final files are used remotely long after the metadata propagation (the inconsistent window) finishes. While these semantics could improve insert performance and scalability, they might be inapplicable to those real-time applications where the time lapse until consistency is achieved exceeds the temporal difference between the creation and the use of remote metadata.

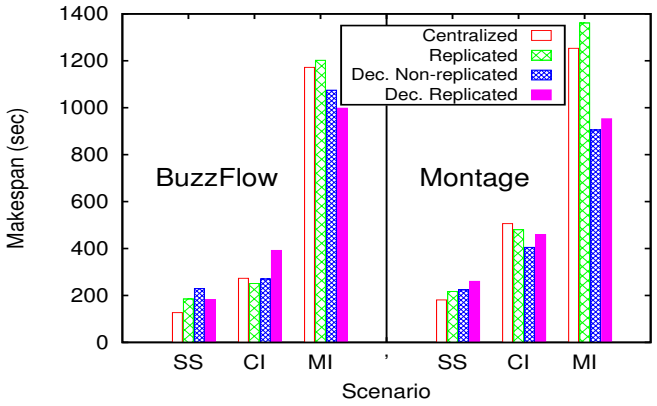

Fig. 10: Makespan for two real-life workflows. SS - Small Scale, CI - Computation Intensive, MI - Metadata Intensive

However, we argue that this application of CAP theorem to workflow metadata fits most of the geographically distributed cloud-based workflows (e.g. CERN ATLAS). Relaxing consistency enables scalability for bursty workloads and provides *fresh results where fresh results are needed* (i.e. the same datacenter), allowing some operations to see older metadata where time critical needs are not as important (i.e. remote datacenters). In our context, this tradeoff comes in preserving freshness for creation / update operations while at the same time allowing the read metadata to gain consistency over a longer period of time, that is, eventually.

VIII. Related Work

In this section, we discuss prior work related to metadata services in modern cluster file systems and optimized techniques for high-performance metadata. Although based on some techniques already known by the community of distributed data management (replication, distribution), to the best of our knowledge, our work is the first attempt to bridge the gap between single- and multi-site cloud metadata management. As opposed to previous work described below, we leverage both the workflow semantics (e.g. data-access patterns) and the practical tools available on today's public clouds (e.g. caching services for PaaS clouds) to propose multiple strategies for decentralized metadata management.

Centralized Metadata Traditional approaches create and maintain a consistent, uniform metadata registry on a *single server*, processing all the data access requests (e.g. Google FS [28], IBM GPFS [12]). However, the single metadata server can quickly become a performance bottleneck in case of increased client concurrency or high I/O pressure. Another straightforward option is to keep metadata in a *relational database* [13]. Similarly, this approach has proved to be too heavy for metadata-intensive workloads (e.g. overheads from transactions and locking) [29], [30]. A lightweight alternative to database is indexing the metadata. Although widely studied, most indexing techniques [31], [32] are designed for data rather than metadata. Even the dedicated index-based metadata schemes [33] use a centralized index and are not adequate for large-scale workflows.

Decentralized Metadata Distributing metadata across several servers mitigates the bottleneck of centralised management for large distributed storage systems. Currently, there are two major methods used to distribute the namespace and workload among metadata servers in existing distributed file systems [34]: partitioning and hashing.

Namespace Subtree Partitioning provides a natural way to partition the namespace among multiple servers according to directory subtrees. Each server manages one or more subtree(s) (also called file sets or volumes) of the hierarchy, analogous to directory mounting in NFS [35]. The strategy provides a good locality because using the subtree structure metadata servers can handle requests without communicating with other servers out of the local subtree. *Static partitioning* suffers from severe bottleneck problems when a single file, directory, or directory subtree becomes popular: the metadata requests may not be evenly partitioned among servers, which usually leads to workload imbalance (trading load balance for locality). In *dynamic partitioning*, metadata is automatically partitioned and distributed to servers according to the specific load-balancing policy of the filesystem. If some metadata severs become overloaded, some of its metadata subtrees could be migrated to other servers with light load. While providing better scalability this might cause slow metadata lookup. Server volatility, inherent in the cloud, causes the whole subdirectory structure to be recomputed in order to maintain the tree-based metadata hierarchy. Several state-of-the-art file systems rely on this partitioning technique. Ceph [36] dynamically scatters sets of directories based on server load. Giraffa [37] leverages HBase [38], a distributed key value store, to achieve load balancing on directory collections, suffering from the aforementioned hot entries issue. PVFS [39] uses a fine-grained namespace distribution by scattering different directories, even those in the same sub-tree, on different metadata servers. PanFS [40] is more coarse-grained: it assigns a subtree to each metadata server.

Hashing eliminates the problem of unbalanced workload among servers by assigning metadata based on a hash of the file identifier, file name or other related values. Although with this approach metadata can be distributed uniformly, the directory locality feature is lost, and if the path is renamed, some metadata have to migrate. Also, a directory hierarchy must still be maintained and traversed in order to support standard file naming semantics, tampering some of the apparent benefits. Giraffa [37] uses full pathnames as the key for file metadata in the underlying HBase store. Lustre [10] makes a hash on the tail of the filename and the identifier of the parent directory to map the metadata to a server.

Unfortunately, none of these schemes can well meet the practical requirements of workflows executed on clouds. Both solutions encounter difficulties when faced with a high volatility of metadata servers (i.e. adding or removing nodes from the deployment), which is the norm in the nowadays *elastic* clouds. In subtree partitioning this is due to whole subtrees being stored on each server, requiring the entire namespace to be repartitioned to keep the load balance. For hashing, the functions themselves may have to be changed (to produce outputs in new ranges). This results in tremendous metadata migrations - slow and costly operations in a multi-site cloud. Our strategies combine the best of both hierarchical directory subtrees and pure hashing. We keep locality by maintaining metadata within the same datacenter where it is likely to be used (by examining the workflow access pattern). The location is determined by a hash function which allows access to data in constant time. The problem of varying number of metadata servers is circumvented by relying on a uniform cache (e.g. Azure Cache, which deals transparently with nodes arrivals/departures) and by using some lazy update policies that allow for efficient, eventual consistent metadata updates when nodes are added/removed.

Support for Small Files Handling Several optimizations brought to PVFS leverage intelligent servers and collective communication to improve metadata latency [41]. The idea was to offload work from clients by allowing the servers to perform complex file system operations on their behalf. The servers used collective communication algorithms similar to those found in message-passing libraries in order to structure communication more efficiently. While the first idea is complementary to our work, the second one is hard to implement in a cloud environment. More recent research [18] proposed five techniques for improving concurrent metadata and small file I/O performance in parallel file systems: server-driven file precreaton, POSIX extensions, file stuffing, metadata commit coalescing, and eager data movement for reads and writes. In contrast, our strategies hide latency and reduce messages and I/O without using additional resources or imposing additional coordination requirements on clients. Similarly to us, CalvinFS [42] uses hash-partitioned key-value metadata across geo-distributed datacenters to handle small files, yet it does not account for workflow semantics.

An important difference to past work is our focus on a whole workflow application and its interaction with the cloud infrastructure. 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 seems very different from HPC: high latency networks connect the datacenters, a much lower degree of (per-object) concurrency, a more specialized storage interface provided to applications, and they are explicitly aware of the anticipated workloads and access patterns. 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 and put them in place leveraging the workflow semantics and the cloud services publicly available.

IX. Conclusion

In the context of fast growing volumes of data to be processed at larger and larger scales, geographically distributed workflows are emerging as a natural data processing paradigm. As of today, state-of-the-art public clouds do not provide adequate mechanisms for *efficient metadata management* across datacenters for scenarios involving masses of geographically distributed data that are stored and processed in multiple sites across the globe. In this work we investigate approaches to metadata management enabling an efficient execution of such *geographically distributed workflows running on multi-site clouds*. We focus on a common scenario where workflows generate and process a huge number of small files, which is particularly challenging with respect to metadata management. As such workloads generates a deluge of small and independent I/O operations, efficient metadata handling is critical.

To address this problem, we explored means to better hide

latency for metadata access as a way of improving the global performance. We propose specific techniques that implement this approach, combining distribution and replication for in-memory metadata partitioning. Although such techniques are already known by the community of distributed data management, to the best of our knowledge, our work is the first attempt to bridge the gap between single- and multi-site cloud metadata management. Our solution leverages both the workflow semantics (e.g. data-access patterns) and the practical tools available on today's public clouds (e.g. caching services for PaaS clouds) to propose several strategies for decentralized metadata management.

Encouraged by the results, we plan to further explore the closer integration between our metadata middleware service and the workflow engines [13]. In particular, we are interested in enabling mutual support between the two: while currently we only leverage workflow semantics to enhance metadata partitioning, we would like to make our metadata information also available to the engine so as to optimize scheduling for data-intensive tasks. Furthermore, an interesting direction to explore is how these strategies could help in handling streams of data in the cloud.

ACKNOWLEDGMENT

This work was supported by the Microsoft Research - Inria Joint Centre. The experiments presented in this paper were carried out using the Azure Cloud infrastructure provided by Microsoft in the framework of the Z-CloudFlow project.

REFERENCES

[1] "Azure Success Stories," http://azure.microsoft.com/en-us/case-studies/.

[2] "Microsoft Azure Cloud," http://www.windowsazure.com/en-us/.

[3] T. Kosar, E. Arslan, B. Ross, and B. Zhang, "Storkcloud: Data transfer scheduling and optimization as a service," in *Proc. of the 4th Workshop on Scientific Cloud Computing*, ser. Science Cloud '13, 2013, pp. 29–36.

[4] N. Laoutaris, M. Sirivianos, X. Yang, and P. Rodriguez, "Inter-datacenter bulk transfers with netstitcher," in *Proceedings of the ACM SIGCOMM 2011 Conference*, ser. SIGCOMM '11, 2011, pp. 74–85.

[5] S. Bharathi, A. Chervenak, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi, "Characterization of scientific workflows," in *Workflows in Support of Large-Scale Science, 3rd Workshop on*, 2008, pp. 1–10.

[6] Y. Zhao, M. Hategan, B. Clifford, I. Foster, G. von Laszewski, V. Nefedova, and I. Raicu, "Swift: Fast, reliable, loosely coupled parallel computation," in *Services, 2007 IEEE Congress on*, 2007, pp. 199–206.

[7] "Cloud computing and HEP physics: How ATLAS experiment at CERN uses Google compute engine in the search for new physics at LHC," https://developers.google.com/events/io/sessions/333315382.

[8] A. Costan, R. Tudoran, G. Antoniu, and G. Brasche, "Tomusblobs: scalable data-intensive processing on azure clouds," *Concurrency and Computation: Practice and Experience*, 2013.

[9] B. Allen, J. Bresnahan, L. Childers, I. Foster, G. Kandaswamy, R. Kettimuthu, J. Kordas, and M. Link, "Software as a service for data scientists," *Comm. of the ACM*, vol. 55, no. 2, pp. 81–88, 2012.

[10] "Lustre - OpenSFS," http://lustre.org/.

[11] "HDFS," http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.

[12] F. Schmuck and R. Haskin, "GPFS: A shared-disk file system for large computing clusters," in *Proc. of the 1st USENIX Conference on File and Storage Technologies*, ser. FAST '02, Berkeley, CA, USA, 2002.

[13] E. Ogasawara, J. Dias, V. Silva, F. 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*, vol. 25, no. 16, pp. 2327–2341, 2013.

[14] "Amazon S3," http://aws.amazon.com/fr/s3/.

[15] S. Al-Kiswany, A. Gharaibeh, and M. Ripeanu, "The case for a versatile storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, pp. 10–14, 2010.

[16] J. Dias, E. Ogasawara, D. De Oliveira, F. Porto, P. Valduriez, and M. Mattoso, "Algebraic dataflows for big data analysis," in *Big Data, 2013 IEEE Intl. Conf. on*, 2013, pp. 150–155.

[17] Y. Simmhan, C. van Ingen, G. Subramanian, and J. Li, "Bridging the gap between desktop and the cloud for escience applications," in *Cloud Computing (CLOUD), IEEE 3rd Intl. Conf. on*, July 2010, pp. 474–481.

[18] P. Carns, S. Lang, R. Ross, M. Vilayannur, J. Kunkel, and T. Ludwig, "Small-file access in parallel file systems," in *IEEE Int. Symp. on Parallel Distributed Processing*, ser. IPDPS '09, 2009, pp. 1–11.

[19] E. H. Neilsen Jr., "The sloan digital sky survey data archive server," *Computing in Science & Engineering*, vol. 10, no. 1, pp. 13–17, 2008.

[20] J. K. Bonfield and R. Staden, "Ztr: a new format for dna sequence trace data," *Bioinformatics*, vol. 18, no. 1, pp. 3–10, 2002.

[21] T. Shibata, S. Choi, and K. Taura, "File-access patterns of data-intensive workflow applications and their implications to distributed filesystems," in *Proc. of the 19th ACM Intl. Symposium on High Performance Distributed Computing*, ser. HPDC '10. ACM, 2010, pp. 746–755.

[22] B. Fitzpatrick, "Distributed caching with memcached," *Linux Journal*, vol. 2004, no. 124, pp. 5–, Aug. 2004.

[23] W. Galuba and S. Girdzijauskas, "Distributed hash table," in *Encyclopedia of Database Systems*. Springer US, 2009, pp. 903–904.

[24] R. Tudoran, A. Costan, R. R. Rad, G. Brasche, and G. Antoniu, "Adaptive file management for scientific workflows on the azure cloud," in *Big Data, 2013 IEEE Intl. Conference on*. IEEE, 2013, pp. 273–281.

[25] "MCache," https://msdn.microsoft.com/en-us/library/azure/dn386094.

[26] "Redis Cache," http://redis.io.

[27] "Montage," http://www.windowsazure.com/en-us/.

[28] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," *SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 29–43, Oct. 2003.

[29] M. Stonebraker, S. Madden, D. J. Abadi, and N. Hachem, "The end of an architectural era: Time for a complete rewrite," in *Proc. of the 33rd Intl. Conf. on Very Large Data Bases*, ser. VLDB '07, pp. 1150–1160.

[30] M. Stonebraker and U. Cetintemel, ""one size fits all": an idea whose time has come and gone," in *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, April 2005, pp. 2–11.

[31] J. Wang, S. Wu, H. Gao, J. Li, and B. C. Ooi, "Indexing multi-dimensional data in a cloud system," in *Proc. of the 2010 ACM SIGMOD Intl. Conf. on Management of Data*, 2010, pp. 591–602.

[32] S. Wu, D. Jiang, B. C. Ooi, and K.-L. Wu, "Efficient b-tree based indexing for cloud data processing," *Proc. VLDB Endow.*, vol. 3, no. 1-2, pp. 1207–1218, Sep. 2010.

[33] A. W. Leung, M. Shao, T. Bisson, S. Pasupathy, and E. L. Miller, "Spyglass: Fast, scalable metadata search for large-scale storage systems." in *FAST*, vol. 9, 2009, pp. 153–166.

[34] B. Nicolae, G. Antoniu, L. Bougé, and D. Moise, "Blobseer: Next-generation data management for large scale infrastructures," *J. of Parallel and Distributed Computing*, vol. 71, pp. 169 – 184, 2011.

[35] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, and B. Lyon, "Design and implementation of the sun network filesystem," in *Proceedings of the Summer USENIX conference*, 1985, pp. 119–130.

[36] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proc. of the 7th Symposium on Operating Systems Design and Implementationon Operating Systems Design and Implementation*, ser. OSDI '06. Berkeley, CA, USA: USENIX Association, 2006, pp. 307–320.

[37] "Giraffa," https://code.google.com/a/apache-extras.org/p/giraffa/.

[38] "Apache HBase," http://hbase.apache.org.

[39] P. Carns, W. B. Ligon, and R. B. Ross, "PVFS: a parallel file system for linux clusters," in *In Proc. of the 4th Linux Showcase & Conf.*, 2008.

[40] B. Welch, M. Unangst, Z. Abbasi, G. Gibson, B. Mueller, J. Small, J. Zelenka, and B. Zhou, "Scalable performance of the panasas parallel file system," in *Proc. of the 6th USENIX Conference on File and Storage Technologies*, ser. FAST'08, 2008, pp. 2:1–2:17.

[41] P. H. Carns, B. W. Settlemyer, and W. B. Ligon, III, "Using server-to-server communication in parallel file systems to simplify consistency and improve performance," in *Proc. of the 2008 ACM/IEEE Conference on Supercomputing*, ser. SC '08, 2008, pp. 6:1–6:8.

[42] A. Thomson and D. J. Abadi, "CalvinFS: consistent wan replication and scalable metadata management for distributed file systems," in *Proceedings of the 13th USENIX Conference on File and Storage Technologies*. USENIX Association, 2015, pp. 1–14.