A Cache-based Data Movement Infrastructure for On-demand Scientific Cloud Computing

David Abramson[1], Jake Carroll[1], Chao Jin[1], Michael Mallon[1], Zane van Iperen[1], Hoang Nguyen[1], Allan McRae[1], and Liang Ming[2]

[1] The University of Queensland, St Lucia QLD 4072, Australia
[2] Huawei Technologies Co., Ltd., Shenzhen, China
{david.abramson, jake.carroll, c.jin, m.mallon, z.vaniperen, h.nguyen30, a.mcrae}@uq.edu.au and l.ming@huawei.com

Abstract. As cloud computing has become the de facto standard for big data processing, there is interest in using a multi-cloud environment that combines public cloud resources with private on-premise infrastructure. However, by decentralizing the infrastructure, a uniform storage solution is required to provide data movement between different clouds to assist on-demand computing. This paper presents a solution based on our earlier work, the MeDiCI (Metropolitan Data Caching Infrastructure) architecture. Specially, we extend MeDiCI to simplify the movement of data between different clouds and a centralized storage site. It uses a hierarchical caching system and supports most popular infrastructure-as-a-service (IaaS) interfaces, including Amazon AWS and OpenStack. As a result, our system allows the existing parallel data intensive application to be offloaded into IaaS clouds directly. The solution is illustrated using a large bioinformatics application, a Genome Wide Association Study (GWAS), with Amazons AWS, HUAWEI Cloud, and a private centralized storage system. The system is evaluated on Amazon AWS and the Australian national cloud.

Keywords: Cloud, Big Data, Caching, Data Migration.

1 Introduction

Presently, many big data workloads operate across isolated data stores that are distributed geographically and manipulated by different clouds. For example, the typical scientific data processing pipeline [27][41] consists of multiple stages that are frequently conducted by different research organizations with varied computing demands. Accordingly, accelerating data analysis for each stage may require computing facilities that are located in different clouds. Between different stages of the geographical data pipeline, moving a large amount of data across clouds is common [5][38]. This type of multi-cloud environment can consist of resources from multiple public cloud vendors, such as Amazon AWS [1] and Microsoft Azure [30], and private data centers. Multi-cloud is used for many reasons, such as best-fit performance, increased fault tolerance, lower cost, reduced risk of vendor lock-in, privacy, security, and legal restrictions. Different from hybrid-cloud, however, data silos in multi-cloud are isolated by varied storage mechanisms of different vendors. This complicates

applying on-demand computing for scientific research across clouds. Although computation offloading into clouds is standardized with virtual machines, a typical data processing pipeline faces multiple challenges in moving data between clouds. First, a uniform way of managing and moving data is required across different clouds. Second, the network connections for inter-clouds and intra-cloud are typically different in terms of bandwidth and security. Moving a large amount of data between centers must utilize critical resources such as network bandwidth efficiently, and resolve the difficulties of latency and stability issues associated with long-haul networks. Third, users have to maintain the consistency of duplicated copies between silos with different storage mechanisms. Fourth, data migration between the stages of a pipeline needs to cooperate efficiently with computing tasks scheduling.

In this work, we propose a hierarchical global caching architecture across geographical data centers of different clouds. Such a system supports automatic data migration to cooperate on-demand Cloud computing. It unifies distant data silos using a file system interface (POSIX) and provides a global namespace across different clouds, while hiding the technical difficulties from users. Data movement between distant data centers is made automatic using caching. Our high performance design supports massive data migration required by scientific computing. Our previous work, called MeDiCI [10], has been shown to work well in an environment consisting of private data centers dispersed across the metropolitan area. In this paper, we extend MeDiCI into the multi-cloud environment that consists of most popular infrastructure-as-a-service (IaaS) cloud platforms, including Amazon AWS and OpenStack-based public clouds, and Australian data centers of NeCTAR (The National eResearch Collaboration Tools and Resources). Existing parallel data intensive applications are allowed to run in the virtualized resources directly without significant modifications.

This paper mainly discusses the following innovations:

- A global caching architecture that moves data across clouds in accordance with on-demand compute and storage resource acquirement;
- A demonstration of the proposed architecture using parallel file system components;
- A platform independent mechanism that manages the system across different IaaS clouds, including Amazon AWS and OpenStack-based clouds;
- Demonstrating our solution using a Genome Wide Association Study with resources from Amazon Sydney and a centralized storage site in Brisbane.

The rest of the paper is organized as follows. Section 2 provides an overview of related work and our motivation. Section 3 introduces our proposed global caching architecture. Sections 4 describe the realization of our storage architecture. Section 5 provides a detailed case study in Amazon EC2 and HUAWEI Cloud with according performance evaluation. Our conclusions follow in Section 6.

2 Background

Massive data analysis in the scientific domain [27][31] needs to process data that is generated from a rich variety of sources, including scientific simulations, instruments,

sensors, and Internet services. Many data intensive applications are embarrassingly parallel and can be accelerated using the high throughput model of cloud computing. Therefore, complementing private data centers with on-demand resources drawn from multiple (public) clouds is frequently used to tolerate compute demand increases. However, offloading computation into clouds not only requires acquiring compute resources dynamically, but also moving target data into the allocated virtual machines. Most existing storage solutions are not designed for a multi-cloud environment. In particular, they often require users to move data between processing steps of a geographical data processing pipeline explicitly. In addition, many existing methods do not directly support parallel IO to improve the performance of scalable data analysis. This section reviews the existing methods and motivates our solution.

"Data diffusion" [19][20], which can acquire compute and storage resources dynamically, replicate data in response to demand, and schedule computations close to data, has been proposed for Grid computing. In the cloud era, the idea has been extended to scientific workflows that schedule compute tasks [41] and move required data across the global deployment of cloud centers. Both data diffusion and cloud workflows rely on a centralized site that provides data-aware compute tasks scheduling and supports an index service to locate data sets dispersed globally. In contrast, our model suits a loosely coupled working environment in which no central service of task scheduling and data index is required. Actually, each department controls its own compute resources and the collaboration between departments relies on shared data sets that are stored in a central site for long-term use. Our previous solution, MeDiCI [10], works well on dedicated resources. In this paper, we extend MeDiCI to a multi-cloud environment with dynamic resources and varied storage mechanisms.

A substantial portion of our work needs to move data across different clouds efficiently. Cloud storage systems, such as Amazon S3 [1] and Microsoft Azure [30], provide specific methods to exchange data across centers within a single cloud, and mechanisms are available to assist users to migrate data into cloud data centers. For instance, the Microsoft Azure Data Factory and the AWS Data Pipeline support data-driven workflows in the cloud to orchestrate and to automate data movement and data transformation. These cloud specific solutions mainly handle data in the format of cloud objects and database. Some other workflow projects combine cloud storage, such Amazon S3, with local parallel file systems to provide a hybrid solution. For example, a staging site [26] is introduced for Pegasus Workflow Management System to convert between data objects and files and supports both Cloud and Grid facilities. In comparison, some cloud backup solutions, such as Dropbox [9], NextCloud [34], and SPANStore [46], provide seamless data access to different clouds. However, most of these cloud storage solutions do not directly support parallel IO that is favored by embarrassing parallel data intensive applications.

Recent projects support directly transferring files between sites to improve overall system efficiency [39]. For example, OverFlow [40][38] provides a uniform storage management system for multi-site workflows that utilize the local disks associated with virtual machine instances. It extends replication service to handle data transfer for inter-site and intra-site traffic using different protocols and mechanisms. This type

of customized storage solution is designed to cooperate with the target workflow scheduler using a set of special storage APIs.

The distributed file system provides a general storage interface widely used by almost all parallel applications. How to support a distributed file system in the global environment has been investigated extensively [24][42][12][33][14][23][6][28]. Typically, the tradeoff between performance, consistency, and data availability must be compromised appropriately to address the targeted data access patterns. Most general distributed file systems designed for the global environment focus on consistency at the expense of performance. The Andrew File System (AFS) [25] federates a set of trusted servers to provide a consistent and location independent global namespace to all of its clients. The AFS caching mechanism allows accessing files over a network as if they were on a local disk. OpenAFS [35] is an open source software project implementing the AFS protocol. Exposing clustered file systems, such as GPFS and Lustre, to personal computers using OpenAFS has been investigated [18]. Frequently, AFS caching suffers from performance issues due to overrated consistency and a complex caching protocol [29]. Overall, AFS was not designed to support large-scale data movement required by on-demand scientific computing. The similar idea of using a global caching system to transfer data in a wide area was also investigated by Content Delivery Networks (CDN) [12]. CDN caches site content at the edge of the Internet, close to end users, in order to improves website performance. In comparison, BAD-FS [22] and Panache [32] improve data movement onto remote computing clusters distributed across the wide area, in order to assist dynamic computing resource allocation. BAD-FS supports batch-aware data transfer between remote clusters in a wide area by exposing the control of storage policies such as caching, replication, and consistency and enabling explicit and workload-specific management. Panache is a scalable, high-performance, clustered file system cache that supports wide area file access while tolerating WAN (Wide Area Network) latencies. It transfers remote files in parallel using the NFS protocol, instead of other batch mode data movement solutions, such as GridFTP [43] and GlobusOnline [7]. Panache maintains the consistency of both meta-data and files.

Our previous work, MeDiCI [10], builds on AFM [17], which is a commercial version of Panache. MeDiCI constructs a hierarchical caching system using AFM and parallel file systems. MeDiCI exploits temporal and spatial locality to move data on demand in an automated manner across our private data centers that spans the metropolitan area. With this paper, we extend MeDiCI to 1) unify the varied storage mechanisms across clouds using the POSIX interface; and 2) provide a data movement infrastructure to assist on-demand scientific computing in a multi-cloud environment.

3 Design

A geographical data processing pipeline may consist of multiple stages and each stage could be executed in different data centers that have appropriate computing facilities. Each stage needs to process both local data and remote files, which require moving data from a remote center to the local site. After each stage is finished, the migrated

data can be deleted according to the specific request, while the generated output may be collected. Frequently, a central storage site keeps long-term use data for pipelines. The common data access patterns of these pipelines include data dissemination, collection, and aggregation [38]. In addition, concurrent data write operations across different phases are very rare.

Our global caching infrastructure aims to support this type of data pipeline that are performed using compute resources allocated dynamically in IaaS clouds. Our system provides a POSIX interface and supports parallel IO to the data intensive applications running in a virtual cluster. With this storage infrastructure, applications do not have to be modified and can be offloaded into clouds directly.

In particular, this global caching architecture accommodates on-demand data movement across different clouds to meet the following requirements: 1) a unified storage solution for multi-cloud; 2) automatic on-demand data movement to fetch data from a remote site; 3) facilitating parallel IO for high performance computing directly; 4) supporting data access patterns commonly used; and 5) efficiently utilizing the network bandwidth to transfer a large amount of data over distant centers.

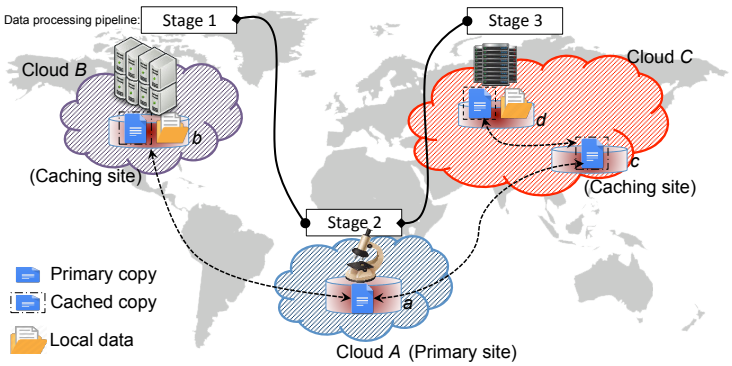

Fig. 1. The hierarchical caching architecture across different clouds.

Our design principle builds on the following key factors.

A global namespace with a POSIX interface: most high performance computing applications rely on a traditional file interface, instead of the cloud objects. Exposing a file interface can save the extra effort of converting between files and objects and it works with many existing scientific applications seamlessly. Furthermore, the global namespace across different clouds allow multiple research organizations share the same set of data without concerning its exact location.

A hierarchical caching architecture: the caching architecture aims to migrate remote data to locate sites in an automated manner without users' direct involvement. In addition, it takes advantage of data location to save unnecessary data transfer.

Data consistency model for the target data access patterns: appropriate data consistency model is critical for the global performance and latency perceived by applications. Our consistency model supports common data access patterns, such as data dissemination and data collections.

Network optimization for distant connections: our system should support optimized global network path with multiple hops, and improve the usage of limited network bandwidth.

The expense of acquiring virtual clusters is out of the scope of this paper. In particular, we expect that users should be aware of whether the advantage of using a remote virtual cluster offsets the network costs caused by significant inter-site data transfer.

3.1 Hierarchical Global Caching Architecture

Many distributed storage systems use caching to improve performance by reducing remote data access. Different from other work, our global caching architecture uses caching to automate data migration across distant centers. The proposed caching model assumes that each data set in the global domain has a primary site and can be cached across multiple remote centers using a hierarchical structure, as exemplified in **Fig. 1**, in which the primary site is the central storage center for keeping long-term use data. Each data set has a primary copy maintained by its primary site and multiple cached copies maintained by caching sites. The primary copy and cached copies form a tree structure, in which the primary copy is the root. As illustrated in **Fig. 1**, the primary copy is maintained by Cloud *A* on site *a* and it is cached in Cloud *B* and *C* respectively. In Cloud *C*, the data is cached on two sites, *c* and *d*.

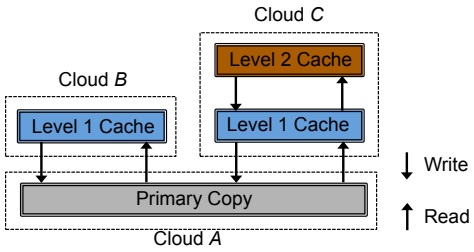

Fig. 2. Data movement path.

Each file in this system can have multiple replicas across data centers that are identified using the same logical name. Within a single data site, a replica is controlled by the local storage solution, typically a parallel file system, which may use data duplication to improve performance. All the copies in a single center are taken as a single logical copy. Across data centers, duplications of logical replicas and their consistency are managed by the global caching architecture. Users are not aware of the exact location for each data set. However, data is actually moved between geographically distributed caching sites according to data processing requirements, in order to lower the latency to frequently accessed data. The migrated data set typically stays locally for the term of use, instead of permanently. The basic data migration operations are supported: 1) fetching remote files to the local site; and 2) sending local updates to a remote site. These two basic operations can be composed to support the common data access patterns, such as data dissemination, data aggregation and data collection.

Data movement is triggered on-demand, although pre-fetch can be used to hide the latency according to the exact data access patterns. Cache capacity on each site is configurable, and an appropriate data eviction algorithm is used to maximize data reuse. In the hierarchical caching architecture, data movement only occurs between neighbor layers. **Fig. 2** illustrates the data movement path for the caching architecture of **Fig. 1**. The top tier caching serves any requested data from the next layer of cache in line. When a cache miss occurs, the request will be forwarded to the next tier of the hierarchical cache, until the primary site is reached. Typically, a read operation moves data from the primary site to the targeted caching site layer by layer, while writes are committed in a reverse order to the primary site.

The hierarchical structure enables moving data through intermediate sites. This layered caching architecture can be adopted in two scenarios: 1) improving the usage of local data copies while decreasing remote data access; and 2) data movement adapted to the global topology of network connections. The exact path to transfer data from the source site to the destination center should be optimized, because the direct network path between two sites may not be the fastest one. In particular, all of the available global network paths should be examined to take advantage path and file splitting [15]. Sometime, adding an intermediate hop between source and destination sites performs better than the direct link. This feature can be achieved by using the hierarchical caching structure naturally. Transferring data via an intermediate site only need to add the cached directory for the target data set, as described in section 3.2. For example, in **Fig. 1**, assume site *a* has poor direct network connection with site *d*, but site *c* connects both *a* and *d* with high bandwidth network. Therefore, site *a* can move data to *d* using site *c* as an intermediate hop with the layered caching structure.

3.2 Global Namespace and POSIX File Interface

With a geographical data pipeline, we envisage that different cloud facilities can take part in the collaboration and exit dynamically. Distant collaborative sites should be loosely coupled. Accordingly, we need a flexible method to construct the storage system across different clouds. In order to allow data to be exchanged transparently across different clouds, a consistent and uniform namespace needs to span a set of distant centers. In addition, different from many other systems, our caching architecture does not maintain a global membership service that monitors whether a data center is online or offline. This saves the overhead of keeping the location of each piece of data in multi-cloud.

The global namespace is provided using the POSIX file interface, and is constructed by linking the remote data set to a directory in the local file system. In other words, a local directory is specified to hold the cache for the remote data set. Multiple remote data sets, which may originate from different data centers, can be stored in different directories on the same site. Therefore, a POSIX file interface unifies storage access for both local and remote data. The cached remote directory has no difference from other local directories, except its files are copied remotely whenever necessary.

3.3 Storage Organization of a Caching Site

In each site, a local parallel file system is used to maintain both cached remote data and local files accessed by the parallel applications running in the virtual cluster. The local parallel file system can be installed on dedicated storage resources to work as a shared storage service, or located on storage devices associated with the same set of compute resources allocated for the data analysis job. The first option maintains cached data for long-term usage, while the second option suits short-term data maintenance, because data storage is normally discarded after computing is finished. The storage media used in each site can be multi-tiered, using varied storage devices such as SSD and hard disk drives. How to organize the storage media to host the parallel file system is out of the scope of this paper.

3.4 Data Consistency

To accommodate common data access patterns used in typical data analysis pipelines, we adopt a consistency model to prioritize data access performance while providing acceptable consistency. In particular, data consistency within a single site is guaranteed by the local parallel file system. Across distant sites, a weak consistency semantic is supported across shared files and a lazy synchronization protocol is adopted to save unnecessary remote data movement. Remote files are copied only when they are accessed. However, a prefetching policy can be specified to hide the latency of moving data, such as copying neighbor files when one file in a directory is accessed. The validity of cached files is actively maintained by each caching site. The validity is verified both periodically and when directories and files are accessed. In addition, users can select an appropriate policy for output files, such as write-through or write-back, to optimize performance and resilience.

The updates on large files are synchronized using a lazy consistency policy, while meta-data is synchronized using a prompt policy. Assuming each caching site verifies its validation every f seconds, for an n level caching hierarchy, the protocol guarantees that the whole system reaches consistency on updated meta-data within $2n \bullet f$ seconds. This consistency model supports data dissemination and collections very well across distant sites on huge files, according to our experience.

3.5 Component Interaction

Fig. 3 illustrates the major components that realize the global caching architecture across distant sites. A POSIX file interface spans different clouds to provide a uniform storage access interface. In each site, different native parallel file system can be used and a file system adaptor provides a general POSIX-compliant interface. The Global Caching module maintains the connections between each cached data set and its parent copy. It coordinates with its peer on the remote site to move requested files according to user requests and to synchronize updates. The Global Caching module builds on top of the local native parallel file system by organizing the local storage space to maintain the duplicated copies of remote files. It intercepts local file requests

and moves remote data transparently in case the requested file is not available locally. Therefore, it exposes the same POSIX-compliant file interface to applications. Accordingly, the global namespace is provided using the tree-based directory structure and seamlessly integrates into the namespace of local file system. The Consistency module coordinates the data synchronization according to user specified configurations.

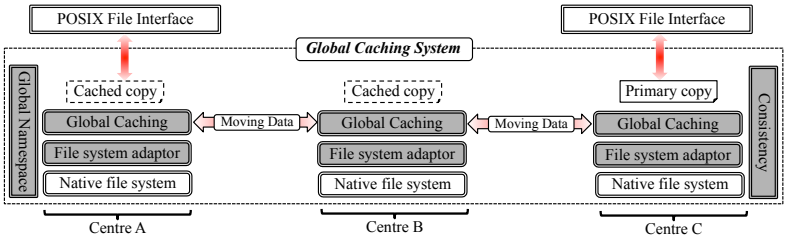

Fig. 3. Major components in the global caching architecture.

4 System Demonstrations

We are currently building on a prototype of the global caching architecture for testing and evaluation purpose. We reuse existing file system components as much as possible to minimize the implementation effort. The caching system builds on GPFS [13], Active File Management (AFM) [17], and the NFS protocol [37]. We realized a platform-independent method to allocate, instantiate and release the caching instance with the target compute cluster across different IaaS clouds in an on-demand manner.

4.1 Existing Components

GPFS is a parallel file system designed for clusters, but behaves likes a general-purpose POSIX file system running on a single machine. GPFS uses the shared-disk architecture to achieve extreme scalability, and supports fully parallel access both to file data and metadata. Files are striped across all disks, while distributed locking synchronizes accesses to shared disks. Our caching system uses the GPFS product, (also known as IBM Spectrum Scale [16]), to hold both local and remote data sets.

As a component of IBM Spectrum Scale, AFM is a scalable, high-performance, clustered file system cache across a WAN. It provides a POSIX-compliant interface with disconnected operations, persistence across failures, and consistency management. AFM transfers data from remote file systems and tolerates latency across long haul networks using the NFS protocol. Parallel data transfer is supported with concurrent NFS connections.

Fig. 4 illustrates an instance of the global caching site. The GPFS cluster provides data service to the compute cluster. Each GPFS cluster is equipped with an AFM component. Each server is attached multiple network shared disks. A configuration of mirror redundancy is shown in **Fig. 4**. The number of servers and associated disks

depends on the total storage capacity needed. The number of gateway nodes determines the aggregated bandwidth that can be achieved to transfer data in and out from the cluster. Quorum managers maintain data consistency in the failure cases. The compute cluster consists of multiple workers and a job scheduler with a login node.

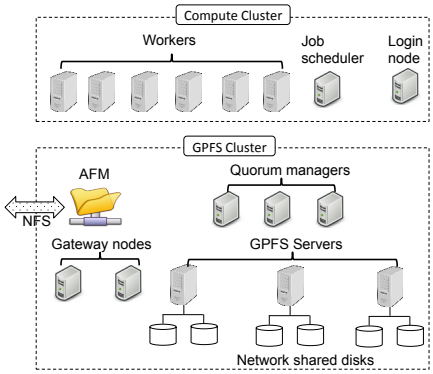

Fig. 4. An instance of the global caching site.

4.2 Platform-independent System Resource Management

The global caching system aims to support different IaaS cloud systems and provides a platform-independent way of managing resource usage, including compute and storage resource allocation, instantiation and release. Our primitive implementation handles Amazon EC2, HUAWEI Public Cloud and OpenStack-based clouds. We realized a tool that supports different cloud orchestration methods, such as CloudFormation in EC2 and Heat in OpenStack and HUAWEI Cloud, to automate the allocation, release, and deployment of both compute and storage resources for building the caching site. To configure each virtual node and storage resources in an automated manner, we use Ansible [2] scripts. Both CloudFormation and Heat support Resource Tags to identify and categorize cloud resources. Our automation tool utilizes this feature to generate Ansible inventory and variables programmatically for system installation and configuration.

Table 1. Cloud resources in Amazon EC2, HUAWEI Cloud, and OpenStack.

Resources	Amazon EC2	HUAWEI Cloud	OpenStack
Virtual machine	Instance	Elastic Compute Server	Nova Instance
OS Images	AMI	Glance	Glance
Block Storage	EBS	Elastic Volume Service	Cinder
Private Network	VPC	VPC	Neutron network
Public IP	Public IP	Elastic IP	Floating IP
AAA	SSH key pairs	SSH key pairs	SSH key pairs

In order to accommodate a consistent caching system deployment over different clouds, according network resources, Authentication, access and authorization (AAA), virtual machines, and storage instances must be supported. **Table 1** lists supported resources in Amazon EC2, HUAWEI Cloud, and OpenStack.

Fig. 5 illustrates the typical deployment in both HUAWEI and Amazon clouds. In HUAWEI Cloud, shown in the left half of **Fig. 5**, both the compute and GPFS clusters are instantiated using Elastic Compute Server (ECS). Each GPFS server is attached with two Elastic Volume Service (EVS) disks. All of the ECS servers are connected with a Virtual Private Cloud (VPC) that communicates with the Internet via a NAT gateway. Each GPFS gateway node is equipped with an Elastic IP (EIP) to access the Internet directly. In EC2, different resources are used to provide the similar configuration, as illustrated in the right half of **Fig. 5**. Our automation tool allows for and accommodates configurable parameters for the type and number of instances as well as block devices attached, operating system image and other tunable parameters.

4.3 Data Transfer Optimization

Achieving high performance data transfer in a WAN requires tuning the components associated with the distant path, including storage devices and hosts in both source and destination sites and network connections [4][8][45]. Critical system parameters such as the TCP buffer size and the number of parallel data transfers in AFM must be optimized. In most cases, it is necessary to transfer the data with multiple Socket connections in order to utilize the bandwidth of the physical link efficiently. Besides moving multiple files concurrently, parallel data transfer must support file split to transfer a single large file. With AFM, parallel data transfer can be achieved at different levels: 1) multiple threads on a single gateway node; 2) multiple gateway nodes. Each option suits for different scenario. For example, in case the Network Interface Card (NIC) on a single gateway node provides enough bandwidth, the first option is enough. In case there is restriction on the NIC bandwidth, multiple gateway nodes can be used. AFM supports parallel data transfer with configurable parameters to adjust the number of concurrent read/write threads, and the size of chunk to split a huge file.

With Amazon EC2, we use a single gateway node with multiple threads to parallelize data transfer. However, with HUAWEI Cloud, each EIP has a bandwidth limitation. To maximize the performance of copying remote files, multiple gateway nodes are used in the cache site. Accordingly, in the home site the same number of NFS servers are deployed. The mapping between them is configured explicitly with AFM.

4.4 Data Consistency

The following data consistency modes are provided to coordinate concurrent data access across distant centers with the assistance of AFM:

- Read Only: each caching site can only read the cached copies, but cannot update them.
- Single Writer: only a single data site updates the cached file set, and the updates are synchronized to other caching sites automatically.

- Concurrent Writer: multiple writers update the same file with application layer coordination. The updates are synchronized without users interference.

AFM allows data to be cached at the block level, while data consistency is maintained per file. When reading or writing a file, only the accessed blocks are fetched into a local cache. When the file is closed, the dirty blocks are written to the local file system and synchronized remotely to ensure a consistent state of the file.

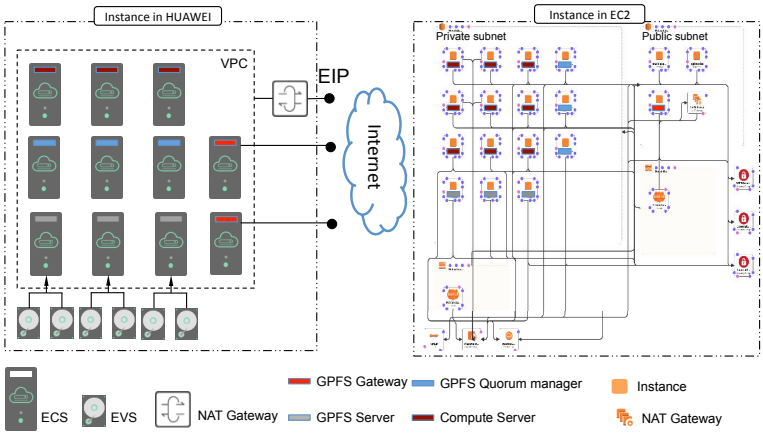

Fig. 5. The deployment of a caching instance in HUAWEI Cloud and Amazon EC2.

5 Case Study and Performance Evaluation

We use a Genome Wide Association Study (GWAS) as an application driver to show how to use our global caching architecture to assist on-demand scientific computing across different clouds. GWAS are hypothesis-free methods to identify associations between regions of the genome and complex traits and disease. This analysis was performed on data from the Systems Genomics of Parkinson's Disease consortium, which has collected DNA methylation data on about 2,000 individuals. This study aimed to test how genetic variation alters DNA methylation, an epigenetic modification that controls how genes are expressed, while the results are being used to understand the biological pathways through which genetic variation affects disease risk.

The work totally conducts 3.3 x 10^{12} statistical tests using the PLINK software [36]. The workload is essentially embarrassingly parallel and does not require high performance communication across virtual machines within the cloud. The data to be analyzed, around 40GB in total, is stored in NeCTAR's data collection storage site located in the campus of the University of Queensland (UQ) at Brisbane. The input data is moved to the virtualized clusters, acquired in Amazon EC2 and HUAWEI Cloud, as requested. In addition, we can control the size of the cloud resource, for both the compute and GPFS clusters, according to our testing requirements.

5.1 System Deployment

As shown in **Fig. 6**, virtualized compute clusters acquired from EC2 and HUAWEI clouds respectively process input data stored in Brisbane. Each instance, including both compute and GPFS clusters, is created using the automation tool described previously. The size of each instance was selected to make the best usage of our available credit. The EC2 instance is created with a single VPC located in Sydney availability zone. The HUAWEI instance consists of two layers. The first layer is a caching only site located in Beijing and the second layer consists of both caching and compute clusters located in Shanghai. The EC2 and HUAWEI resources connected to the central storage site in Brisbane via the global caching architecture. The data transferred between these data sites is achieved using NFS connections and AFM caching. Typically, AFM NSD protocol outperforms NFS. However, due to the security concern occurred in the public network, we could only use NFS. The global caching system provided local access to data even though it was actually stored in NeCTAR, and the necessary files were fetched transparently on demand. Likewise, output files were written back to NeCTAR without the user being aware. Therefore, the application was identical to as if it was executed on a local cluster without any modification.

In the EC2 cluster, the Nimrod [11] job scheduler was used to execute 500,000 PLINK tasks, spreading the load across the compute nodes and completing the work in three days. Overall, approximately 60 TBs of data were generated by the experiment and sent back to Brisbane for long-term storage and post-processing.

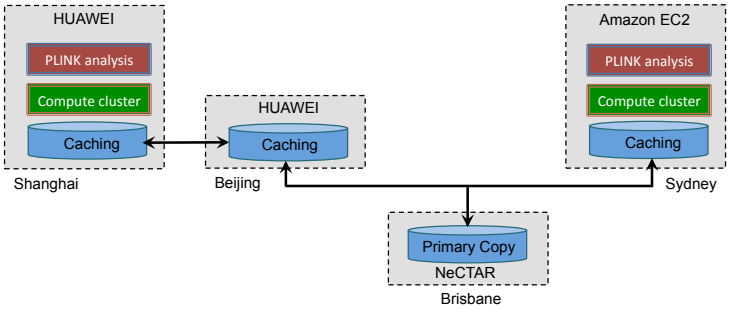

Fig. 6. The deployment of the global caching architecture for GWAS case study.

With the hierarchical caching structure in HUAWEI cloud, the input data was moved from Australia to Beijing first and then copied to Shanghai center. Due to credit limitation, no significant compute jobs were executed in HUAWEI Cloud.

5.2 AWS EC2 Instance Selection

We investigated the appropriate AWS instance for our EC2 experiment. Due to the constraints of time and cost, we could not exhaustively explore all the available instances. We used a holistic approach to identify which instance types provide optimal

performance for different roles. Briefly, with the option of network-attached storage, instance types, such as *m4*, could not provide sufficient EBS bandwidth for GPFS Servers. Therefore, we examined the instances associated with the ephemeral storage of local block devices. However, the *d2* series, namely the *d2.8xlarge* type, experienced hardware and underlying infrastructure reliability issues. Finally, we used the *i3* instance types, *i3.16xlarge*, for the GPFS cluster that provided 25 Gbit/sec network with the ephemeral NVMe class storage, and had no reliability issues. For the compute cluster, we selected the compute-optimized flavours, *c5.9xlarge*. It had a dedicated 10 Gbit/sec bandwidth with Intel Xeon Skylake CPUs.

Table 2. Configurations of Amazon EC2 Testing.

Type of nodes	Instances	Count	Details
Nimrod worker	c5.9xlarge	25	750 Xeon Skylake cores in total.
AFM Gateway	i3.16xlarge	2	Each instance is equipped with
GPFS Quorum	i3.16xlarge	3	25 Gbit/sec network bandwidth
GPFS Server	i3.16xlarge	10	and 8 x 1.9TB NVME.

To determine instance counts, we matched aggregated worker bandwidth to GPFS Server bandwidth to satisfy a fully balanced IO path. Further, we tested a small scale of the PLINK workload to estimate run time per job and then sized our virtual cluster to execute the full workload within ~3 days. The final configuration is listed in **Table 2**. Totally, 750 Nimrod worker threads were launched on the compute cluster.

5.3 Network Transfer Optimization

The network between AWS Sydney and UQ is 10Gbps with around 18.5ms latency. The connection is under a peering arrangement between the national research network provider, AARNET, and Amazon. The network is shared with other academic and research sector AARNET partners. Therefore, our configuration aims to maximize the effective bandwidth. For this case, a single active gateway node was used with 32 AFM read/write threads at the cache site. In comparison, the default option is just one read/write thread. TCP buffers were tuned to improve performance at both source and destination sites. The home NFS server currently serves production workloads and requires 4,096 NFS daemons to service this workload. With these optimizations in place, we achieved about 2 Gbps, which is 20% of the peak bandwidth on the shared public link. The total amount of data moved from UQ to Amazon Sydney was 40GB, but the amount of data moved back to our datacenter (home) was 60TB in total.

5.4 Performance Evaluation

During the 3 days of experiment, system utilization was on average about 85~92% on each node, with the I/O peaking at about 420,000 input and 25,000 output operations

per second (IOPS). The total 500,000 tasks were launched in 5 batches sequentially. This allowed us to optimize the system configuration while monitoring the progress of computing and expense used. Actually, the system was tuned in the first batch. Therefore, we only present the performance statistics for the last 4 batches. We used the EC2 CloudWatch tools to monitor the performance. In particular, we captured CPU utilization, network traffic and IOPS for each instance.

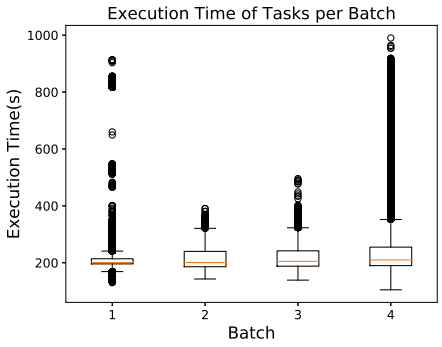

Fig. 7. Performance variation of PLINK tasks.

Although each PLINK task consists of similar computational complexity with almost same size of input data, we observed significant performance variation, as illustrated in **Fig. 7**. The averaged execution time is 200 seconds with a long tail of outliers, and some special cases could take up to 1,000 seconds. Commonly, performance variability exists in a large scale of distributed system. Shared resources and system and network instability can lead to huge performance variation [3]. For our case, we observed significant variations of IO access for PLINK tasks.

Fig. 8. Disk read operations per second.

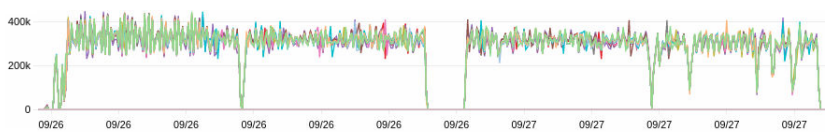

Fig. 9. Disk write operations per second.

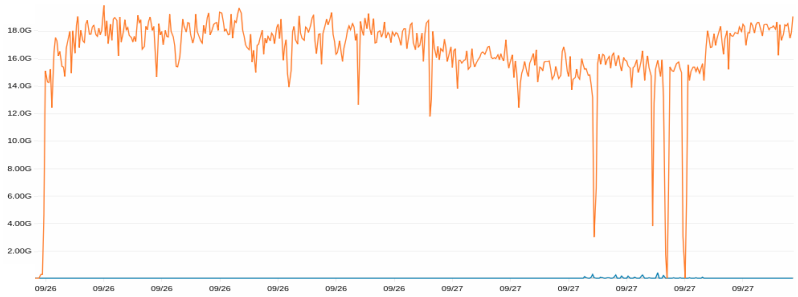

Fig. 10. Outbound network traffic of AFM gateway nodes.

Fig. 8 and **Fig. 9** show the disk read and write statistics for compute servers, in which each line with different color represents metrics for a single instance. Because of the PLINK workload, write IO is an order of magnitude higher than read performance. The metrics of different instances are correlated very well and it means the workload on each instance is pretty similar. In particular, the write performance was comparatively stable within the range of 200K and 400K. We believe this is because the updates were first committed to local NVMe devices before being transferred to the home site through AFM gateway. In comparison, averaged read operations changes from around 22K to less 15K. This may be caused by unreliable long-haul network.

Fig. 10 presents the network traffic from Amazon to UQ through GPFS gateway at the caching site, in which the orange line represents the operative gateway node and the blue one is for the fail-over backup node. We can see that most remote data traffics were managed by the operative gateway node. There are significant drops in the last day of experiment. We assume they were caused by shared bandwidth competition from other public users. This resource contention also impacts the PLINK execution time at the last day, especially the performance of read IO.

6 Conclusions

Geographically distributed data processing pipelines are becoming common. The stages of data intensive analysis can be accelerated using cloud computing with the high throughput model and on-demand resource allocation. It is desired that existing parallel applications can be offloaded into a multi-cloud environment without significant modifications. To achieve this goal, this paper presents a global caching architecture that provides a uniform storage solution to migrate data sets across different clouds transparently. In particular, on-demand data movement is provided by taking advantage of both temporal and spatial locality in geographical data pipelines. Cooperating with the dynamic resource allocation, our system can improve the efficiency of large-scale data pipelines in multi-clouds. Our architecture provides a hierarchical caching framework with a tree structure and the global namespace using the POSIX file interface. The system is demonstrated by combining existing storage software, including GPFS, AFM, and NFS. Parallel IO is supported directly to improve the performance of scalable data analysis applications. Both block-based caching and file-

based data consistency are supported in the global domain. A platform independent method is realized to allocate, instantiate and release the caching site with both compute and storage clusters across different clouds. The case study of GWAS demonstrates that our system can organize public resources from IaaS clouds, such as both Amazon EC2 and HUAWEI Cloud, in a uniform way to accelerate massive bioinformatics data analysis. In particular, the PLINK analysis was offloaded into the multi-cloud environment without any modification and worked as if it was executed on a local cluster. The performance evaluation demonstrates that our global caching architecture has successfully addressed its design goals.

ACKNOWLEDGMENTS

We thank Amazon and HUAWEI for contributing cloud resources to this research project.

References

1. Amazon S3 Homepage, https://aws.amazon.com/s3/, last accessed 2018/11/30
2. Ansible Homepage, https://www.ansible.com/, last accessed 2018/11/30
3. Dean J. and Barroso L.: The Tail at Scale. Communications of the ACM, vol. 56, pp. 74-80, 2013.
4. Kumar A., et al.: BwE: Flexible, Hierarchical Bandwidth Allocation for WAN Distributed Computing. In: Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM'15), London, 2015.
5. Rajendran A., et al.: Optimizing Large Data Transfers over 100Gbps Wide Area Network. In: Proceedings of 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGrid'13), Delft, 2013.
6. Thomson A., and J.Abadi D.: CalvinFS: Consistent WAN Replication and Scalable Metadata Management for Distributed File Systems. In: Proceeding of the 13th USENIX Conference on File and Storage Techniques (FAST'15), CA, 2015.
7. Allen B., et al.: Globus Online: Radical Simplification of Data Movement via SaaS. The University of Chicago, Technical Report. 2011.
8. Settlemyer B., et al.: A Technique for Moving Large Data Sets over High-Performance Long Distance Networks. In: Proceedings of IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST'11), Denver, 2011.
9. Dropbox Homepage, https://www.dropbox.com, last accessed 2018/11/30
10. Abramson D., Carroll J., Jin C., and Mallon M.: A Metropolitan Area Infrastructure for Data Intensive Science. In: Proceedings of IEEE 13th International Conference on e-Science (e-Science), Auckland, 2017.
11. Abramson D., Sosic R., Giddy J., and Hall B.: Nimrod: a tool for performing parametrised simulations using distributed workstations. In: Proceedings of the 4th IEEE International Symposium on High Performance Distributed Computing, 1995.
12. Nygren E., Sitaraman R., and Sun J.: The Akamai Network: A Platform for High-Performance Internet Applications, ACM SIGOPS Operating Systems Review archive, Vol. 44(3), pp. 2-19, 2010.

13. Schmuck F., and Haskin R.: Gpfs: A shared-disk file system for large computing clusters. In: Proceedings of the 1st USENIX Conference on File and Storage Techniques (FAST), 2002.
14. Hupfeld F., et al.: The xtreemfs architecture: a case for object-based file systems in grids, Journal of Concurrency and Computation, Vol. 20(17), pp. 2049-2060, 2008.
15. Khanna G. et al.: Using Overlays For Efficient Data Transfer Over Shared Wide-Area Networks. In: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing (SC'08), Austin, 2008.
16. IBM Spectrum Scale Homepage, https://www.ibm.com/support/knowledgecenter/en/STXKQY_4.2.0, 2018/11/30
17. IBM, Active file management (AFM) Homepage, https://www.ibm.com/support/knowledgecenter/en/STXKQY_4.2.0/com.ibm.spectrum.scale.v4r2.adv.doc/bl1adv_afm.htm, 2018/11/30
18. Reuter H.: Direct Client Access to Vice Partitions. AFS & Kerberos Best Practice Workshop 2009, CA, 2009.
19. Raicu I., Foster I., Zhao Y., Little P., Moretti C., Chaudhary A., and Thain D.: The Quest for Scalable Support of Data-Intensive Workloads in Distributed Systems. In: Proceedings of the 18th ACM International Symposium on High performance Distributed Computing (HPDC'09), Munich, 2009.
20. Raicu I., Zhao Y., Foster I., and Szalay A.: Accelerating Large-scale Data Exploration through Data Diffusion. In: IEEE International Workshop on Data-Aware Distributed Computing (DADC'08), 2008.
21. IOR Homepage, https://github.com/LLNL/ior, 2018/11/30
22. Bent J., et al.: Explicit Control in a Batch-Aware Distributed File System. In: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation (NSDI'04), CA, 2004.
23. Corbett J., et al.: Spanner: Google's globally distributed database. ACM Transactions on Computer Systems (TOCS), Vol. 31(3-8), pp. 1–22, 2013.
24. Kubiatowicz J., et al.: OceanStore: An Architecture for Global-Scale Persistent Storage. In: Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2000.
25. Morris J., et al.: Andrew: a distributed personal computing environment. Communications of the ACM - The MIT Press scientific computation series, Vol. 29(3), pp. 184–201,1986.
26. Vahi K., et al.: Rethinking Data Management for Big Data Scientific Workflows. In: Proceedings of 2013 IEEE International Conference on Big Data, Silicon Valley, 2013.
27. Biven L.: Big Data at the Department of Energy's Office of Science, 2nd NIST Big Data Public Working Group Workshop, 2017.
28. Pacheco L., et al.: GlobalFS: A Strongly Consistent Multi-Site File System. In: Proceedings of IEEE 35th Symposium on Reliable Distributed Systems (SRDS), Budapest, 2016.
29. Vitale M.: OpenAFS Cache Manager Performance. AFS & Kerberos Best Practice Workshop 2015, PA, 2015.
30. Microsoft Azure Homepage, https://azure.microsoft.com/en-us/, 2018/11/30
31. Hey T., Tansley S., and Tolle K.: The Fourth Paradigm: Data-Intensive Scientific Discovery. Microsoft Corporation, 2012.
32. Eshel M., Haskin R., Hildebrand D., Naik M., Schmuck F., and Tewari R.: Panache: A Parallel File System Cache for Global File Access. In: Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST'10), California, 2010.

33. Ardekani M. and Terry D.: A self-configurable geo-replicated cloud storage system. In: Proceedings of the 11th USENIX conference on Operating Systems Design and Implementation (OSDI'14), 2014.
34. Nextcloud Homepage, https://nextcloud.com, 2018/11/30
35. OpenAFS Homepage, https://www.openafs.org/, 2019/01/31
36. PLINK Homepage, http://zzz.bwh.harvard.edu/plink/, 2018/11/30
37. Sandberg R., Goldberg D., Kleiman S., Walsh D., and Lyon B.: Design and Implementation of the Sun Network File System, Summer USENIX Proc., 1985.
38. Tudoran R., Costan A., and Antoniu G.: OverFlow: Multi-Site Aware Big Data Management for Scientific Workflows on Clouds. IEEE Transactions on Cloud Computing. Vol. 4-1, 2016.
39. Tudoran R., Costan A., Rad R., Brasche G., and Antoniu G.: Adaptive File Management for Scientific Workflows on the Azure Cloud. In: Proceedings of 2013 IEEE International Conference on Big Data, Silicon Valley, 2013.
40. R. Tudoran, A. Costan, R. Wang, L. Bouge, and G. Antoniu. Bridging Data in the Clouds: An Environment-Aware System for Geographically Distributed Data Transfers. In: Proceedings of 14th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing (CCGrid'14), Delft, 2013.
41. Dolev S., Florissi P., Gudes E., Sharma S., and Singer I.: A Survey on Geographically Distributed Big-Data Processing using MapReduce. IEEE Transactions on Big Data, 2017.
42. Rhea S., et al.: Pond: the OceanStore Prototype. In: Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST'03), 2003.
43. Allcock W.: GridFTP: Protocol Extensions to FTP for the Grid. Global Grid ForumGFD-R-P.020, 2003.
44. Allcock W., et al.: The Globus Striped GridFTP Framework and Server. In: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing (SC'05), 2005.
45. Kim Y., Atchley S., Vallee G., and Shipman G.: LADS: Optimizing Data Transfers using Layout-Aware Data Scheduling. In: Proceedings of the 13th USENIX Conference on File and Storage Technologies (FAST'15), Santa Clara, 2015.
46. Wu Z., et al.: SPANStore: Cost-effective Geo-replicated Storage Spanning Multiple Cloud Services. In: Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP'2013), 2013.