Noname manuscript No.
(will be inserted by the editor)

Data Access Management System in Azure Blob Storage and AWS S3 Multi-Cloud Storage Environments

Yaser Mansouri · Rajkumar Buyya

Received: date / Accepted: date

Abstract Multi-cloud storage offers better Quality of Service(QoS) such as availability, durability, and users perceived latency. The exploitation of price differences across cloud-based storage services is a motivate example of storing data in different Geo-graphically data stores, where data migration is also a choice to achieve more cost optimization. However, this requires to migrate data in tolerable time from the perspective of users. This paper first proposes a comprehensive review on different classes of data stores inspiring data migration within and across data stores. Then, it presents the design of a system prototype spanned across storage services of Amazon Web Services (AWS) and Microsoft Azure employing their RESTful APIs to store, retrieve, delete, and migrate data. Finally, the experimental results show that the data migration can be conducted in a few seconds for data with a magnitude of Megabytes.

Keywords Cloud Storage, Amazon Web Services (AWS), Microsoft Azure Storage, Latency, Data Migration

1 Introduction

Cloud computing has gained significant attention form the academic and industry communities in recent years. It provides the vision that encompasses the movement of computing elements, storage and software delivery away from personal computer and local servers into the next generation computing infrastructure hosted by large companies such as Amazon Web Service (AWS), Microsoft Azure, and Google. Cloud computing has three distinct characteristics that differentiate it from its traditional counterparts: pay-as-you-go model, on-demand provisioning of infinite resources, and elasticity [1].

Cloud computing offers three types of resources delivery models to users [2]: (i) Infrastructure as a Service (IaaS) which offers computing, network, and storage resources, (ii) Platform as a Service (PaaS) which provides users tools that facilitate the deployment of cloud applications, and (iii) Software as a Service (SaaS) which enables users to run the provider's software on the cloud infrastructure.

One of the main components of IaaS offering by cloud computing is Storage as Services (StaaS). StaaS provides an elastic, scalable, highly available, and pay-as-you-go model, which renders it attractive for data outsourcing, both for the users to manipulate data independent of the location and time and for firms to avoid expensive upfront investments of infrastructures. The

Yaser Mansouri
Cloud Computing and Distributed Systems (CLOUDS) Lab, School of Computing and Information System, The University of Melbourne, Australia
E-mail: yase@student.unimelb.edu.au

Rajkumar Buyya
Cloud Computing and Distributed Systems (CLOUDS) Lab, School of Computing and Information System, The University of Melbourne, Australia
E-mail: rbuyya@unimelb.edu.au

well-known Cloud Storage Providers (CSPs)–AWS, Microsoft Azure, and Google– offer StaaS for several storage classes which differ in price and performance metrics such as availability, durability, the latency required to retrieve the first byte of data, the minimum time needed to store data in the storage, etc.

The data generated by online social networks, e-commerce, and other data sources is doubling every two years and is expected to augment to a 10-fold increase between 2013 and 2020-from 4.4 ZB to 44 ZB.[1] The network traffic, generated from these data, from data centers (DCs) to users and between DCs was 0.7 ZB in 2013 and is predicated to reach 3.48 ZB by 2020.[2] The management of such data in the size of several exabytes or zettabytes requires capital-intensive investment; the deployment of cloud-based data stores (data stores for short) is a promising solution.

Moving the data generated by data-intensive applications into the data stores guarantees users the required performance Service Level Agreement (SLA) to some extent, but it causes concern for monetary cost spent in the storage services. Several factors contribute substantially to the monetary cost. First, the monetary cost depends on the size of the data volume that is stored, retrieved, updated, and potentially migrated from one storage class to another one in the same/different data stores. Second, it is subject to the required performance SLA (e.g., availability,[3] durability, the latency needed to retrieve the first byte of data) as the main distinguishing feature of storage classes. As the performance guarantee is higher, the price of storage classes is more. Third, the monetary cost can be affected by the need of data stores to be in a specific geographical location in order to deliver data to users within their specified latency. To alleviate this concern (i.e., monetary cost spending on storage services) from the perspective of application providers/users, it is required to replicate data in an appropriate selection of storage classes offered by different CSPs during the lifetime of the object regarding to the satisfaction of latency for Put (write), Get(read), and data migration from the users perspective.

The use of multiple CSPs offering several storage classes with different prices and performance metrics brings a substantial benefit to users who seek the reduction of monetary cost in storage services, while respecting their Quality of Service (QoS) in terms of availability and network latency [3]. In this direction, we designed algorithms that take advantage of price differences across CSPs with several storage classes to reduce monetary cost on storage services for time-varying workloads [4] [5]. This mandates data migration across data stores. As a concern for users, it is important to migrate data in tolerable time. This paper shed a light on this gap through the following contribution:

- We provide a taxonomy of the cloud-based data stores offered by the well-known cloud providers to bring the attention of researchers for future research directions.
- We provide modules using RESTful API of AWS and Microsoft Azure to store, retrieve, delete, and migrate data for AWS' and Microsoft Azure' data stores.
- We profile data migration time between a pair of data stores within and across regions. We also show that this time is reasonable for transferring users' data, which reaches to a magnitude of Megabytes (MBs).

The reminder of this paper is organized as follows. Section 2 discusses the background of the well-known and commercial cloud-based data stores and compares them in the key QoS criteria such as availability, durability etc. Section 3 presents the benefit of spreading data across data stores and the concerns in this respect. In Section 4, we discuss the system design and the modules which are implemented for AWS and Microsoft Azure storage services. Finally, Section 5 presents the evaluation of our system and Section 6 concludes this paper.

[1] International Data Corporation (IDC). `https://www.emc.com/leadership/digital-universe/2014iview/index.htm`.

[2] The Zettabyte Era—Trends and analysis. `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html`.

[3] Availability and durability are described in terms of *nines.*

2 Background

Well-known cloud providers such as AWS, Azure, and Google offer resources as Infrastructure as a Service (IaaS), where Storage as a Service (StaaS) is one of its main components. StaaS supports several classes of storage, which are differentiated in price and performance metrics.[4] These metrics are (i) durability, (ii) availability SLA, (iii) minimum object size: an object smaller than s kilobytes in size is charged for s kilobytes of storage, (iv) minimum storage duration: an object deleted less than d days after storing in storage incurs a minimum $d-$day charge, (v) retrieval first byte latency: the time taken to retrieve the first byte of an object. The storage classes supported by AWS, Azure, and Google are as follows.

AWS provides four classes of storage[5]: (i) Simple Storage Service (S3) is a highly reliable, available, and secure storage service for data frequently accessed; (ii) Reduced Redundancy Storage (RRS) offers users storage resources with lower cost at the expense of lower levels of redundancy as compared to S3. RSS is suitable for data which require less durability as compared to those stored in S3; (iii) Standard-Infrequent Access (S-IA) is optimized for data accessed less frequently, but needs a low retrieval time (e.g., backup data); (iv) Glacier storage is the cheapest AWS storage which is suited to data with very low access rates and without the need for rapid access.

Microsoft Azure supports four classes of storage services,[6] which are mainly distinguished based on the number of replicas of an object that are stored in a single or multiple DCs. These classes are: (i) Locally Redundant Storage (LRS) stores 3 synchronous replicas within a single DC; (ii) Zone Redundant Storage (ZRS) stores 3 asynchronous replicas across multiple DCs within or across regions; (iii) Geographical Redundant Storage (GRS) is the same as LRS, in addition to storing 3 asynchronous replicas in a secondary DC that is far away from the primary DC; (iv) Read-access GRS (RA-GRS) is the same as GRS with the added functionality of allowing users to access data in the secondary DC. All classes of Azure storage support five types of storage: Blob, Table, Queue, File, and Disk. Each type of storage is used for a specific purpose. Blob storage is specialized for unstructured object data, Table storage for structured data, Queue storage for reliable messaging between different components of cloud services, File storage for sharing data across the components of an application, and Disk (premium) storage for supporting data-intensive workload running on Azure virtual machines (VMs). Besides these classes and types of storage, Azure also provides Blob storage with two access tiers. These are *hot* and *cold* access tiers which are supported in three classes of storage: LRS, GRS, and RA-GRS. Hot (resp. cold) access tier is used for data that are frequently (resp. rarely) accessed. Hot tier access is more expensive than the cold one, and this allows users to save cost when they switch between these access tiers based on a change in the usage pattern of the object. This switch incurs additional charges, and thus users are required to select each access tier in the appropriate time during the lifetime of the object.

Google supports five storage classes[7]: (i) Multi-regional storage is appropriate for frequently accessed data. This class is Geo-redundant storage service that maintains an object in at least two regions; (ii) Regional storage enables users to store data within a single region. This class is suitable for Google Compute Engine (GCE) instances; (iii) Durable Reduced Availability (DRA) has a lower availability SLA with the same cost (apart from the cost of operations) compared to Regional storage; (iv) Nearline storage is suitable for data that are accessed on average once a month or less. Thus, this class is a good choice for data backup, archival storage, and disaster recovery; (v) Coldline storage is the cheapest Google storage, and it is a suitable option for data accessed at most once a year.

Based on the offered storage classes, we classify them into five tiers. (i) Very hot tier provides the highest levels of redundancy across multiple regions and allows users to access the data in the secondary DC as the primary DC faces faults. (ii) Hot tier stores data in multiple regions, but the redundancy level is lower than the first tier. (iii) Warm tier is the same as hot tier in

[4] Key features of storage classes. `https://aws.amazon.com/s3/storage-classes/`.

[5] AWS storage classes. `https://aws.amazon.com/s3/storage-classes/`

[6] Microsoft Azure storage classes. `https://azure.microsoft.com/en-us/pricing/details/storage/blobs/`

[7] Google storage classes. `https://cloud.google.com/storage/`

redundancy level, but less durable. (iv) Cold tier provides lower availability and durability as compared to the first three tiers and imposes restriction on metrics like minimum object size and minimum storage duration. (v) Very cold tier has the same durability and availability levels compared to the cold tier, but it has more minimum storage duration. The last two tiers impose retrieval cost and they are more expensive than the first three tiers (i.e., very hot, hot, and warm) in operations cost. Table 1 summarizes the characteristics of the storage tiers offered storage services by AWS, Azure, and Google based on the discussed tiers.

Although these tiers are the same in functionality, their performance is directly proportional to price. For example, AWS offers S3 (belongs to hot tier) and RRS (belongs to warm tier) as online storage services, but RRS compromises redundancy for lower cost. Moreover, the price of storage resources across cloud providers is different. Thus, given these differences, many cost-based decisions can be made. These decisions will become complicated especially for applications with time-varying workloads and different QoS requirements such as availability, durability, response time, and consistency level. To make satisfactory decisions for application providers, a joint optimization problem of resources cost and the required QoS should be characterized. Resources cost consists of: (i) *storage cost* calculated based on the duration and size of storage the application provider uses, (ii) *network cost* computed according to the size of data the application provider transfers out (reads) and in (writes) to data stores (typically data transfer into data stores is free), and (iii) *computing cost* calculated according to duration of renting a VM by the application provider. The first two costs relate to the cost of data storage management.

In respect to the above discussion, we can define many optimization problems in conjunction with QoS such as availability, latency, and consistency. One of these is monetary cost reduction by exploiting price differences among CSPs, which mandates data migration across data stores. Although data migration may reduce the monetary cost of data management across data stores, this is a concern about the migration time of data between data stores within and across across regions. We measure this time and show the feasibility of data transfer required a reasonable time. It is worth to mention that this time is only measured between a pair of data stores without relaying to datacenter(s) as a mediator. This is because that we do not make any optimization on the time of data migration across data stores.

Table 1: The characteristics of different storage classes for the well-known Cloud providers: Amazon Web Service (AWS), Azure(AZ), and Google(GO)

Storage Class	**Durability**	**Availability**	**MOS**	**MSD**	**Retrieval Fee**	**First Byte Latency**	**Applicability**	**Examples of Cloud Providers**		
								AWS	Azure	Googlec
Very Hot	NA	6 Replicas	NA	NA	NA	milliseconds	Data frequently accessed	N/A	GRS, RA-GRS	N/A
Hot	11 nines	3-3.5 nines	NA	NA	NA	milliseconds	Data frequently accessed	S3	ZRS	Regional, Multi-Regional
Warm	NA	4 nines (AWS) 2 nines (GO)	NA	NA	NA	milliseconds	Data frequently accessed	RRS	LRS	DRA
Cold	11 nines	2 nines	128 KB (AM) NA (GO)	30 days	Per GB	milliseconds	Data accessed at most once a month	Standard - IA	GRS (Cool tier)	Nearline
Very Cold	11 nines	NA (AWS), 2 nines (GO)	NA	90 days	Per GB	hours (AWS) milliseconds (GO)	Data accessed at most once a year	AWS Glacier	LRS (Cool tier)	Coldline

† The abbreviation used in this table are:
MOS: Minimum Object Size
MSD: Minimum Storage Duration
GRS: Geographically Redundant Storage, RA-GRS: Read-Access Geographically Redundant Storage
ZRS: Zone Redundant Storage, LRS: Locally Redundant Storage
S3: Simple Storage Service, RRS: Reduced Redundancy Storage
Standard-IA: Standard- Infrequent Access, DRA: Durable Reduced Availability

3 Related Works and Motivation

Migrating data into a single data store facilitates users performance SLA/QoS to some extent, but faces them with several limitations. Data store unavailability can confront users with inaccessible data if the data is stored in a single data store. This is counted as one of the top ten obstacles for cloud adoption [6]. Reliance on a single data store makes it difficult to migrate data from a data store to another in the face of price increment by the cloud provider, the emergence of a new data store with lower price, the mobility of users, and changes in workload that demands data migration. This is recognized as data lock-in and is listed as another main obstacle in regard to cloud services. Storing data in a single data store faces the fact that the read (Get) and write (Put[8]) requests are not served with adequate responsiveness. This is because the requests are issued by users who are located worldwide and, consequently users experience more network latency to retrieve/store data from/into a data store. Furthermore, the use of a single data store deprives users from the opportunity to exploit the pricing differences across CSPs. Therefore, storing data within a single data store can be inefficient in both performance SLA and monetary cost.

These factors make inevitable the use of multiple CPSs which improve availability, durability, and data mobility. The deployment of multiple CSPs also brings another benefits to users. (i) If the outage of a data store happens then the requests issued by users are directed to another data store. (ii) Users can have a wider selection of data stores, which results in reducing user-perceived latency as experimentally confirmed [7]. (iii) This deployment also allows application providers to select storage classes across CSPs based on the workload on data and the QoS specified by users to reduce monetary cost of storage and network resources.

The workload on data can be a determining factor for the selection of a storage class. Some data-intensive applications generate a time-varying workload in which as the time passes the rate of read and write requests on the data changes. In fact, there is a strong correlation between the age of data stored in a data store and data workload. For example, in Online Social Network (OSN) data initially receive many read and write requests and gradually these requests reduce [8]. Based on this change of requests rate, we define two statuses for data: *hot-spot* and *cold-spot*. Hot-spot data receive many read and write requests, while cold-spot data receive a few.

This demands for a suitable selection of storage classes throughout the lifetime of an object. Each storage class provided by the well-known CSPs can be suited for data with specific requirements. For instance, one class may be suitable for data that is frequently accessed. Another class may be designed to host data that is rarely accessed and required for a lower availability and durability.

Therefore, CSPs with a variety of storage classes with different prices and performance SLAs and data-intensive applications with time-varying workloads give us an incentive to design novel algorithms to optimize monetary cost [4] [5]. These algorithms work for any data to which workload transits from hot-spot to cold-spot and vice versa, as observed in OSN applications [8]. Significant studies have been done in the area of cost optimization of OSN applications across CPSs, as conducted in SPANStore [9] and Cosplay [10]. Neither leverage different storage classes owned by different CSPs nor consider the object with hot- and cold-spot status, which result in migration of data across different data stores or movement of data between storage classes within a data store [14].

Fig. 1 illustrates a simple data placement across data stores (owned by different providers) to clarify data placemen across data stores. Among all available data stores, application providers, for example OSNs, select a subset of data stores to replicate data to serve their users. OSN users are typically assigned to the closest DCs and have a set of friends and followers who make network connections with them. These connections are between users who are assigned to the same DC as represented by a graph in rectangles (see Fig. 1) or different DCs (e.g., user connections UC1 and UC2). In this model, for example, a user in user group UG1 puts data in AWS DC named replica R1. To make this data available to his/her friends and followers, the data is also replicated in Azure DC as replica R2. These replicas stay in DCs until they receive many read and write requests. As time passes, one replica probably migrates to another DC or moves between storage

[8] Read and Write are respectively interchangeable with Get and Put in this paper.

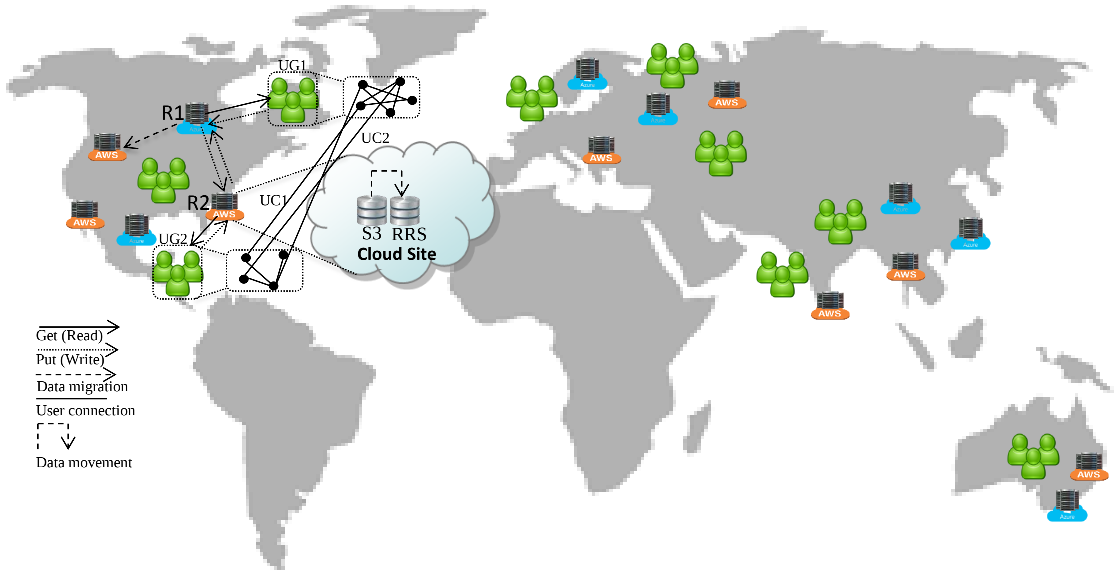

Fig. 1: A simple system model used in the paper

classes (e.g., Simple Service Storage (S3) and Reduced Redundancy Storage (RRS) in AWS) within a DC as backup data. This data migration reduces the monetary cost though it arises a concern about the of data migration for users. We demonstrate this is not of concern because the data migrated between data stores locating within and across regions within several seconds for a magnitude of Megabytes (MBs).

4 System Design

In this section, we introduce a modular architecture for object placement across cloud-based data stores. It enables users/application providers to store, retrieve, delete, list, and migrate objects across data stores based on the desired user's QoS (i.e., latency), the specification of objects and DCs. As shown in Fig. 2, the architecture contains several main components:

(1) Users/Application Provider: this component is the entry point to the system and allows users to store, retrieve, delete, list, migrate objects across data stores.

(2) Object Information: this component includes objects' metadata like object ID/name and size.

(3) Datacenters (DC) information: this is available to the system and consists of DC region/ID, and the price of storage and bandwidth.

(4) Object Placement Decision: this component is the core of the system and is responsible to find appropriate locations for objects based on the objective function (i.e., cost optimization) and the users perceived latency (i.e, migration, read, and write latency). It consists of three modules: (i) Cost Calculation acts based on the proposed algorithm in [14] and calculates the management cost of objects based on the price of storage and bandwidth, and the size of objects. (ii) Location Determination: this makes a decision based on the previous module to store objects in the corresponding data store. (iii) Storing Objects's Metadata: metadata can be stored in VMs deployed across DCs or in Replica Placement Migration (RPM) Manager discussed in the next section.

(5) Cloud Provider API: Amazon S3 and Microsoft Azure Storage provide REST (Representational State Transfer) APIs to Get(read), Put(write), delete, and monitor data in data stores around the world. We use these APIs offering for Java programming to manage data across these two cloud providers.

(6) Cloud Resources: This includes two classes of AWS storage services (i.e., S3 and RSS) and two Azure storage services (LRS and ZRS). More details on these storage services discussed in Section 2.

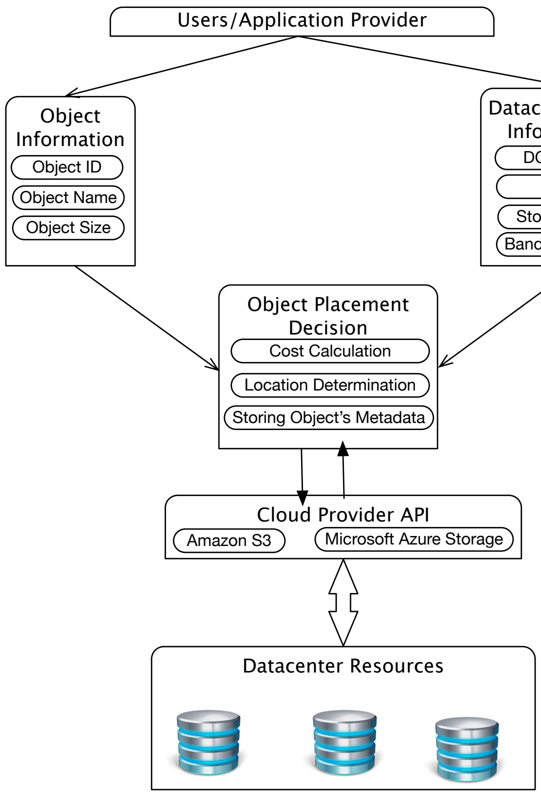

Fig. 2: Key architectural components of object placemnt across cloud-based data stores

4.1 Data Access Management Modules

To implement the architecture depicted in Fig. 2, we implemented two packages using Java programming language:

(1) Data Placement: as depicted in Fig. 3, this is an extension of CloudSim [11] in which we extended `Datacenter` and `File` classes according to the properties and methods required for data placement across data stores. Moreover, we implemented `MigrationObject` class for objects migrated within or across data stores.

(2) Datacenter Connection: as shown in Fig. 4, this consists of two classes for each cloud provider. One class is related to the securing information and makes secure connections to the desired DC, and another class represents methods to manage data across and within data stores. To provide these methods, we implemented a prototype system across Amazon Web Service (AWS) and Microsoft Azure cloud providers. For this purpose, we use JAVA-based AWS S3[9] and Microsoft Azure[10] storage REST APIs. With this prototype, an individual end-user can (i) manage data across two well-known cloud providers, and (measure) the perceived latency for operations conducted on the data.

Our prototype system provides a set of modules that facilities users to store, retrieve, delete, migrate, list data across AWS and Microsoft Azure data stores. Tables 2 - 5 show the list of main web services that is used in the prototype system for data access across AWS and Microsoft

[9] Amazon S3 REST API `http://docs.aws.amazon.com/AmazonS3/latest/API/Welcome.html`

[10] Azure storage REST API `https://docs.microsoft.com/en-us/rest/api/storageservices/fileservices/azure-storage-services-rest-api-reference`

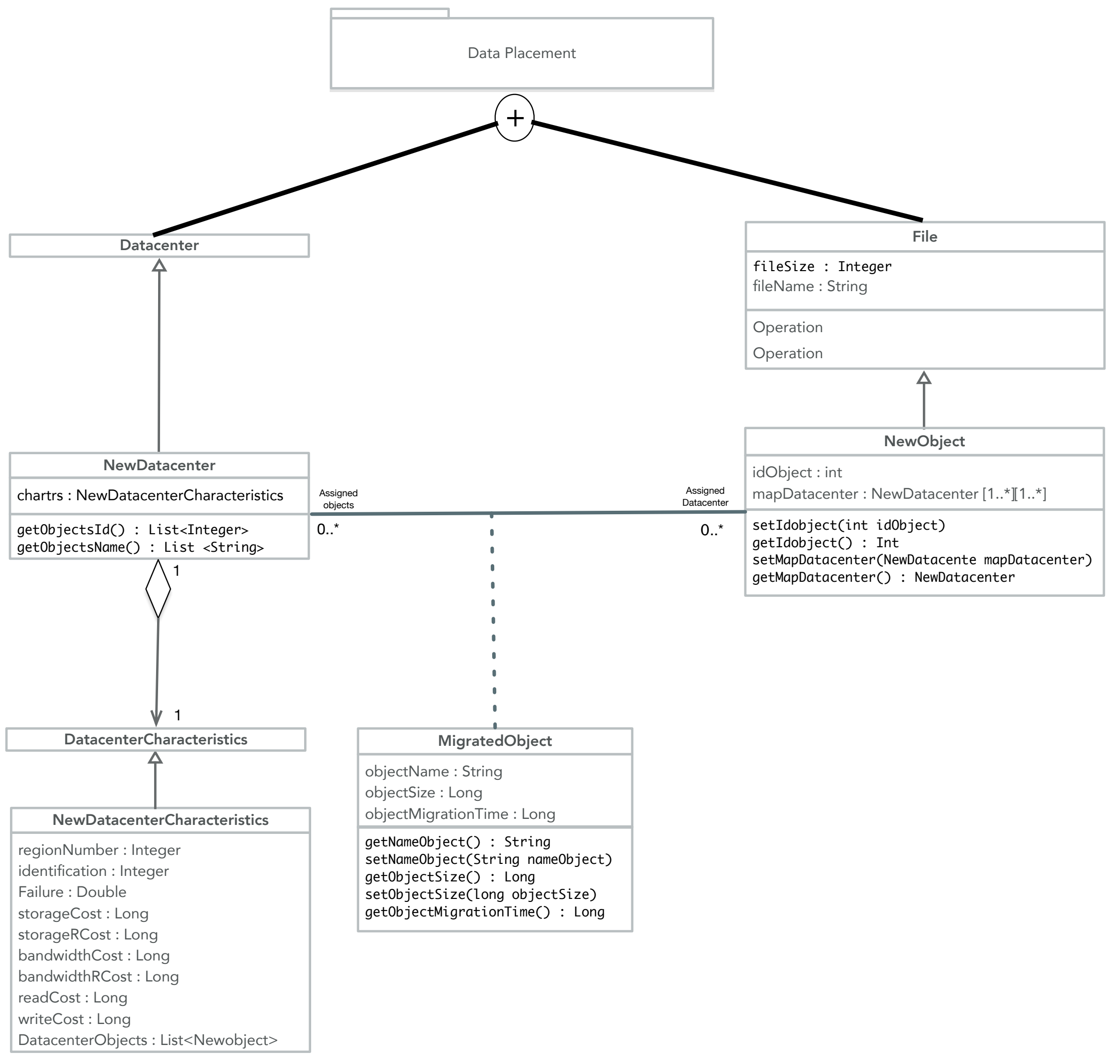
Data Placement
Datacenter
File
fileSize : Integer
fileName : String
Operation
Operation
NewDatacenter
chartrs : NewDatacenterCharacteristics
getObjectsId() : List<Integer>
getObjectsName() : List <String>
Assigned objects
0..*
Assigned Datacenter
0..*
NewObject
idObject : int
mapDatacenter : NewDatacenter [1..*][1..*]
setIdobject(int idObject)
getIdobject() : Int
setMapDatacenter(NewDatacente mapDatacenter)
getMapDatacenter() : NewDatacenter
1
1
DatacenterCharacteristics
MigratedObject
objectName : String
objectSize : Long
objectMigrationTime : Long
getNameObject() : String
setNameObject(String nameObject)
getObjectSize() : Long
setObjectSize(long objectSize)
getObjectMigrationTime() : Long
NewDatacenterCharacteristics
regionNumber : Integer
identification : Integer
Failure : Double
storageCost : Long
storageRCost : Long
bandwidthCost : Long
bandwidthRCost : Long
readCost : Long
writeCost : Long
DatacenterObjects : List<Newobject>

Fig. 3: Data Placement Package

Azure clouds. All these services are RESTful web services that utilize AWS S3 and Microsoft Azure storage APIs in Java. They produce response in the JavaScript Object Notations (JSON) format in successful cases and error message in the error cases. We use JSON format because it is a lightweight data-interchange format and easy to understand. In the following we discuss the provided web services in more details.

Table 2 shows the list of main modules that is provided by the prototype system for data management in AWS data stores. These modules are as follows:

- `amazonCreateS3Client`: This module provides users to create a client with the type of AmazonS3client for accessing the Amazon S3 web services. It also allows user to set a region for the created AmazonS3 client account.
- `createBucket`: This module creates a bucket in the AmazonS3 client specified by users. It also allows users to determine the Access Control List (ACL) in terms of private, publicRead, and PublicReadWrite.
- `amazonCreateFolder`: This module creates a folder in a bucket and allows users to determine the storage class of objects stored in a folder.
- `amazonUploadObject`: This module facilitates users to store objects in the specified directory which has a scheme like /AmazonS3 client/bucket/folder/.
- `amazonDownloadObject`: This module allows users to retrieve objects from AWS data stores in the specified directory with a scheme like /AamzonS3 client/bucket/folder/.
- `amazonDeleteFolder/Objeject`: As its name implies, it deletes folders or objects.

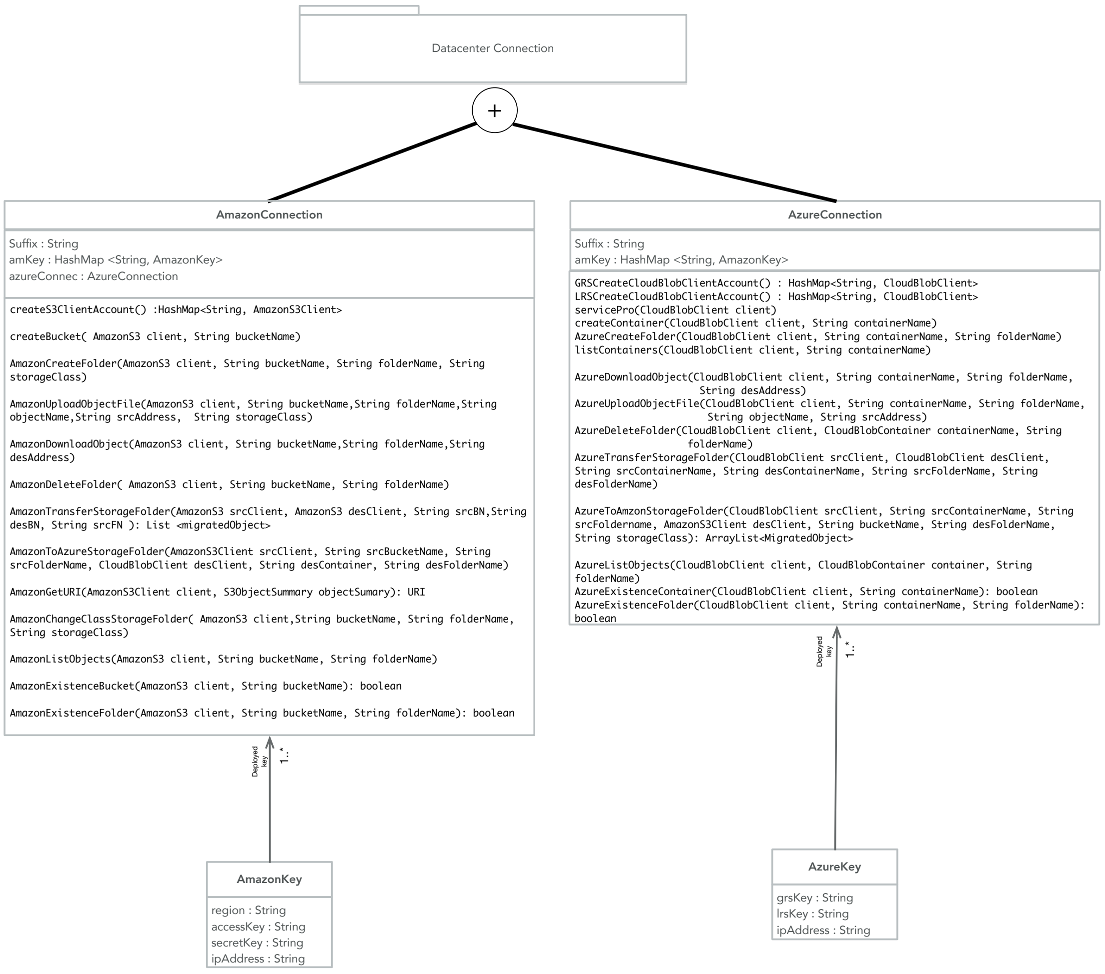
Datacenter Connection
+
AmazonConnection
Suffix : String
amKey : HashMap <String, AmazonKey>
azureConnec : AzureConnection
createS3ClientAccount() :HashMap<String, AmazonS3Client>
createBucket(AmazonS3 client, String bucketName)
AmazonCreateFolder(AmazonS3 client, String bucketName, String folderName, String storageClass)
AmazonUploadObjectFile(AmazonS3 client, String bucketName,String folderName,String objectName,String srcAddress, String storageClass)
AmazonDownloadObject(AmazonS3 client, String bucketName,String folderName,String desAddress)
AmazonDeleteFolder(AmazonS3 client, String bucketName, String folderName)
AmazonTransferStorageFolder(AmazonS3 srcClient, AmazonS3 desClient, String srcBN,String desBN, String srcFN): List <migratedObject>
AmazonToAzureStorageFolder(AmazonS3Client srcClient, String srcBucketName, String srcFolderName, CloudBlobClient desClient, String desContainer, String desFolderName)
AmazonGetURI(AmazonS3Client client, S3ObjectSummary objectSumary): URI
AmazonChangeClassStorageFolder(AmazonS3 client,String bucketName, String folderName, String storageClass)
AmazonListObjects(AmazonS3 client, String bucketName, String folderName)
AmazonExistenceBucket(AmazonS3 client, String bucketName): boolean
AmazonExistenceFolder(AmazonS3 client, String bucketName, String folderName): boolean
AzureConnection
Suffix : String
amKey : HashMap <String, AmazonKey>
GRSCreateCloudBlobClientAccount() : HashMap<String, CloudBlobClient>
LRSCreateCloudBlobClientAccount() : HashMap<String, CloudBlobClient>
servicePro(CloudBlobClient client)
createContainer(CloudBlobClient client, String containerName)
AzureCreateFolder(CloudBlobClient client, String containerName, String folderName)
listContainers(CloudBlobClient client, String containerName)
AzureDownloadObject(CloudBlobClient client, String containerName, String folderName, String desAddress)
AzureUploadObjectFile(CloudBlobClient client, String containerName, String folderName, String objectName, String srcAddress)
AzureDeleteFolder(CloudBlobClient client, CloudBlobContainer containerName, String folderName)
AzureTransferStorageFolder(CloudBlobClient srcClient, CloudBlobClient desClient, String srcContainerName, String desContainerName, String srcFolderName, String desFolderName)
AzureToAmzonStorageFolder(CloudBlobClient srcClient, String srcContainerName, String srcFoldername, AmazonS3Client desClient, String bucketName, String desFolderName, String storageClass): ArrayList<MigratedObject>
AzureListObjects(CloudBlobClient client, CloudBlobContainer container, String folderName)
AzureExistenceContainer(CloudBlobClient client, String containerName): boolean
AzureExistenceFolder(CloudBlobClient client, String containerName, String folderName): boolean
Deployed key
1..*
Deployed key
1..*
AmazonKey
region : String
accessKey : String
secretKey : String
ipAddress : String
AzureKey
grsKey : String
lrsKey : String
ipAddress : String

Fig. 4: Datacenter Conection package for connecting to AWS and Azure datacenters

– `amazonListObjects`: This module lists folders in a bucket as well as the objects stored in a folder.
– `amazonChangeClassStorageFolder`: This module changes the storage class for objects stored in a folder.
– `amazonTransferFolder`: This module allows users to transfer objects (stored in a folder) from an AWS data store to another one.
– `amazonToAzureTransferFolder`: This module facilitates users to migrate objects from an AWS data store to Azure data store.

Table 3 summarizes the type and description of the main input parameters used in the above modules. It is worth nothing that the `storageClass` parameter can be one of the four constant values: `STANDARD`, `REDUCED_REDUNDANCY`, `STANDARD_IA` (IA for infrequent access), and `GLACIER`. In our prototype, we use the first two storage classes as discussed in Section 2. Some input parameters used in the above modules are the same in the type and description while they are different in the name. We excluded these parameters in Table 3 and give their details here. All `(src/des)Client`, `(src/des)BucketName`, and `(src/des)FolderNmae` parameters are respectively the same with the `client`, `bucketName`, and `folderName` parameters in the type and descriptions. The `srcClient`, `srcBucketName`, and `srcFolderNmae` parameters represent that from which client, bucket and folder data are transferred. The `desClient`, `desBucketName`, and `desFolderNmae` parameters indicate the location to that data are transferred.

Table 2: The modules used for data access management in AWS.

Module Name	Input Parameters	Output
amazonCreateS3Client	accesskey,secretkey	AmazonS3Client
createBucket	client,bucketName	Bucket
amazonCreateFolder	client,bucketName,folderName,storageClass	Folder
amazonUploadObject	client,bucketName,folderName,objectname, path,storageClass	put Object
amazonDownloadObject	client,bucketName,folderName,desPath	Get object(s)
amazonDeleteFolder/Objects	client,bucketName,folderName	Delete folder/objects
amazonListObjects	client,bucketName,folderName	Objects list
amazonChangeClassStorageFolder	client,bucketName,folderName,storageClass	Change storage class
amazonTransferFlder	srcClient,srcBucketName,srcFoldername,desClient, desBucketName	Transfer objects
amazonToAzureTransferFolder	srcClient,bucketName,srcFolderName,desClient, containerName,desFolderName	Transfer objects

Table 3: The input parameters used in Modules of AWS.

Input Parameter	Type	Description
accessKey	String	This key is uniquely assigned to the owner of AWS S3 account.
secretKey	String	This key is the password for the owner of AWS S3 account.
client	AmazonS3Client	This parameter allows users to invoke the service methods on AWS S3.
bucketName	String	This refers to the name of the bucket that contains folders.
folderName	String	This refers to the name of the folder containing objects.
storageClass	String	This specifies constants that include four storage classes of AWS S3.
objectName	String	This parameter specifies the name of object generated by users.
srcPath	String	This represents the path from which the data can be transferred.
desPath	String	This indicates the path to which the data can be transferred.

Similarly, we provide a set of modules to manage data across Microsoft Azure data stores. As listed in Table 5, these modules are mostly similar to the discussed ones in the functionality. They are summarized as follow:

- `azureCreateCloudBlobClient`: This module creates an Azure cloud storage account in a region to access Azure cloud storage web services.
- `createContainer`: It creates a container in the the Azure storage account specified by users. This module can determine the type of ACL in the forms of Container, Blob, and OFF (i.e., no blob neither container). It is worth noting that container in Azure data stores and bucket in AWS data stores are the same in the concept.
- `azureTransferFolder`: It facilitates users to transfer objects (stored in a folder) from an Azure data store to another one.
- `azureToAmazonTrasnferFolder`: This allows users to transfer object from Azure data stores to Amazon data stores.
- `azureCreateFolder`, `azureUploadObject`, `azureDownloadObject`, `azureDelete-Folder/Objeject`, `azureListObjects` modules respectively allow users to create folder, store objects, download object, and list objects in Azure data stores.

The modules of Microsoft Azure require input parameters which are similar to those of AWS data stores to the large extent. Table 4 gives a list of those that are only used in Microsoft Azure Modules. The `accessKey` parameter is a 512-bit storage access key which is generated when users create their storage accounts. The `client` parameter allows users to create containers in different Azure regions. The `containerName` parameter is an instance of the "CloudBlob-Container" class and its name is a string value used in Microsoft Azure modules. Note that the `desClient` parameter in the `amazonToAzureTransferFolder` module is an instance of the "cloudBlobClient" class; likewise this parameter in the `azureToAmazonTransferFolder` module is a reference variable of the "AmazonS3Client" class.

5 Performance Evaluation

We design a simple prototype as shown in Fig. 5. The way in which the deployed virtual machines (VMs) should serve Puts and Gets for each object is dictated by a central replica placement

Table 4: The input parameters used in Modules of Microsoft Azure.

Input Parameters	Type	Description
accessKey	String	This key is used to authenticate when the storage account is accessed.
client	cloudBlobClient	This parameter allows users to invoke the service methods on blob storage.
containerName	String	This refers to the name of the container that contains the folders.

Table 5: The modules used for data access management in Microsoft Azure.

Module Name	Input Parameter(s)	Output
azureCreateCloudBlobClient	accessKey	CloudBlobClient
createContainer	client,containerName	Container
azureCreateFolder	client,containerName,folderName	Folder
azureUploadObject	client,containerName,folderName,objectname, path	put Object
azureDownloadObject	client,containerName,folderName,desPath	Get object(s)
azureDeleteFolder/Objects	client,containerName,folderName	Delete folder/objects
azureListObjects	client,containerName	Objects list
azureTransferFlder	srcClient,srcContainerName,srcFolderName,desClient, desContainerName, desFolderName	Transfer object(s)
azureToAmazonTransferFolder	srcClient,srcContainerName,srcFolderName,desClient, BucketName,desFolderName	Transfer object(s)

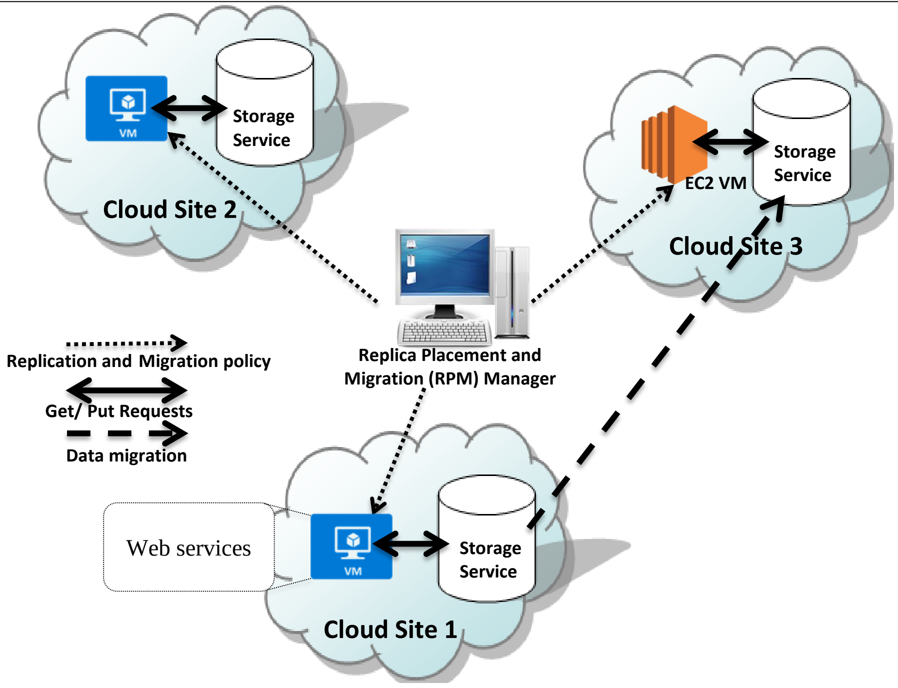

Fig. 5: An overview of prototype

and migration (RPM) manager. The RPM manager makes decision on replica placement and migration across data stores based on the proposed heuristic solution [14]. The RPM issues Http requests (REST call) to the VMs deployed in cloud sites and receives Http responses (JSON objects). The VMs process the received requests via the deployed web services that are implemented based on Spring Model-View-Controller (MVC) framework [13] as illustrated in Fig. 6.

We measured the latency for Gets across 18 data stores spanned around the world. The object size is between 1 KB and 10 KB for 100 users (in the Twitter traces) [4] who tweeted on their Feed. As shown in Tables 6 and 7, a subset of data stores are appropriate to store users' data according to the required QoS, i.e., latency.

To measure the time spent on data migration across DCs, we utilized the federation of cloud sites from Microsoft Azure and Amazon in our prototype. We spanned our prototype across 3 Microsoft Azure cloud sites in `Japan West`, `North Europe`, and `South Central US` regions and 3 Amazon cloud sites in `US East (North Virginia)`, `US West (Oregon)`, and `US West (North California)` regions. In each Azure cloud site, we created a `Container` and

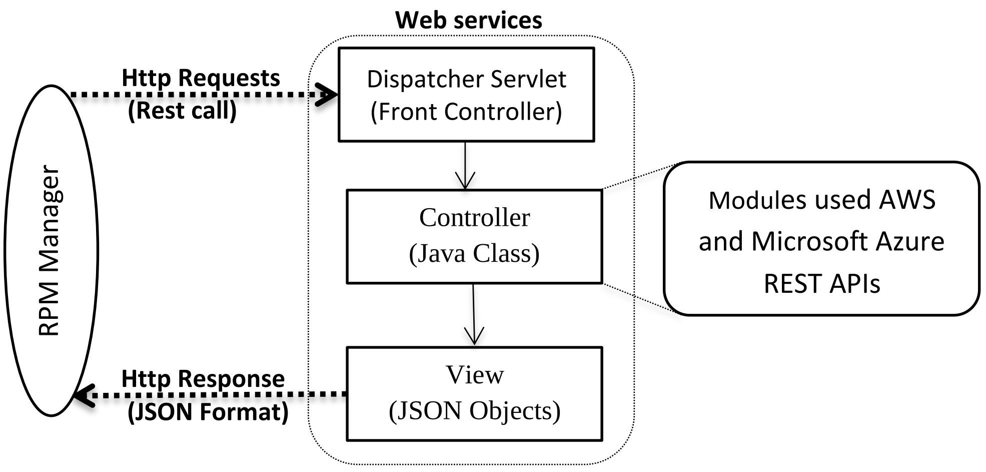

Fig. 6: Web services components used in the prototype

Table 6: Latency (in milliseconds) between Amazon' and Azure's data stores to Azure' data stores (Abbreviations: AZ: azure, AM: Amazon, SUS: South USA, CUS: Central USA, EA: East Asia, JW:Japan West, AUE: Australia East)

Datacenter Name	AZ-SUS	AZ-CU	AZ-Sao Paulo	AZ-Dublin	AZ-Amsterdam	AZ-EA	AZ-JW	AZ-Singapore	AZ-AUE
AZ-SUS	1	25	146	115	127	176	135	210	190
AZ-CUS	25	1	156	105	117	185	135	212	210
AZ-Sao Paulo	146	156	1	196	195	322	280	341	313
AZ-Dublin	115	105	196	1	24	292	241	313	305
AZ-Amsterdam	127	117	195	24	1	290	250	302	281
AZ-EA	176	185	322	290	292	1	60	38	118
AZ-JW	135	135	280	241	250	60	1	85	13
AZ-Singapore	210	212	341	313	302	38	85	1	150
AZ-AUE	190	210	313	305	281	118	113	150	1
AM-Oregon	56	48	209	146	158	187	184	185	177
AM-Virginia	38	40	130	87	86	231	237	253	214
AM-California	37	42	176	153	156	179	182	178	152
AM-Sao Paulo	146	155	5	218	216	394	368	384	408
AM-Dublin	123	106	207	3	24	259	252	254	305
AM-Frankfurt	121	123	213	29	10	262	254	243	299
AM-Tokyo	159	169	270	268	256	54	11	70	173
AM-Singapore	216	212	348	337	316	38	140	3	130
AM-Sydney	169	189	317	330	325	149	158	159	5

Table 7: Latency between different Amazon's data stores

Datacenter Name	AM-Oregon	AM-Virginia	AM-California	AM-Sao	AM-Dublin	AM-Frankfurt	AM-Tokyo	AM-Singapore	AM-Syd
AM-Oregon	1	72.4	21.2	181.2	133.6	146.2	90.7	162.3	177
AM-Virginia	73.6	1	80.4	120.9	76.3	88.9	157.2	231.4	238.2
AM-California	21.3	74.8	1	194.6	153.2	169.2	107.2	177	159.5
AM-Sao Paulo	181.2	120.6	194.6	1	192.4	199.6	275.9	349.6	313.6
AM-Dublin	144.7	76.4	150.1	192.5	1	20	226.4	267.4	312.4
AM-Frankfurt	146.3	88.9	169.2	199.6	19.9	1	240.2	250.7	325.5
AM-Tokyo	91.5	160.6	107.7	281.7	215.8	242.2	1	76.6	106
AM-Singapore	170.7	229.4	177	349.7	267.1	250.7	76.6	1	177.1
AM-Sydney	177	235.4	156.4	313.5	333.1	325.5	106	177	1

deploy a `DS3_V2 Standard` VM instance. In each Amazon cloud site, we created a `Bucket` and deployed a `t2.medium` VM instance. All VM instances used in the prototype run Ubuntu 16.04 LTS as operating system.

After the set-up, we run the heuristic algorithm [14] for 100 users (in the Twitter traces) who are assigned to the aforementioned cloud sites. The data of each user in data stores is integrated in a folder (analogous to bucket in Spanner [12]) for each user. Based on the heuristic algorithm, data migration happens when the cost of storing data in the source data store is more than the summation of (i) the cost of storing data in the destination data store, and (ii) the migration cost between the source and the destination data stores. In the occurrence of data migration, we recorded the time of data transfer from source cloud site to the destination cloud site.

Fig. **??** shows the CDFs of data migration time observed for 100 buckets (each user is associated to a bucket), each of which with the size of about 47.35 MB in average. Fig. **??** depicts that data migration can be transmitted in several seconds across regions. About 60% of buckets are transmitted in 2.5 seconds from Azure DC in Japan west (`AZ-JAW`) to Amazon DC in US

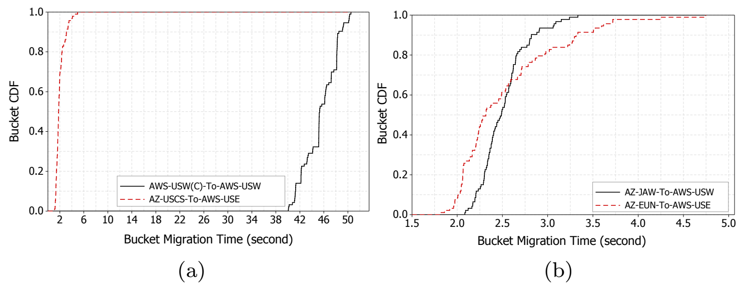

Fig. 7: CDF of data migration time (a) from Azure DC in Japan west to Amazon DC US west and from Azure DC in Europe north to Amazon US east, and (b) Amazon DC in US west (California) to Amazon DC in US west and Azure DC in US Center south to Amazon US east.

west (`AWS-USW`) as well as from Azure DC in Europe north (`AZ-EUN`) to Amazon DC in US east (`AWS-USE`). Also, all buckets are transmitted in 3.5 seconds from Asia region to US region and likewise 4.5 seconds from Europe region to US region. Fig. **??** illustrates the data migration time within US region. About 80% of buckets are migrated from Azure DC in US center south (`AZ-USCS`) to Amazon US east (`AWS-USE`) below 2 seconds. In contrast, bucket migration time between Amazon DC in US west (`North California)` (`AWS-USW(C)`) to another DC in US west (`Oregon`) (`AWS-USW`) is between 40-48 seconds for about 80% of buckets. From the results, we conclude that the duration of buckets migration is considerably low. In the case of a large number of buckets, we can transfer data in bulk across DCs with the help of services such as AWS Snowball[11] and Microsoft Migration Accelerator[12].

6 Conclusions

This paper presented a comprehensive review of the well-known and commercial cloud-based storage resources and compared them in the main QoS criteria such as availability, durability, first byte latency, etc. Then, it discussed the motivation of using the cloud-based storage resources owned by different cloud providers and indicated the latency for Get (read), Put (write), and data migration as one of the main concerns from the users perspective in the respect of replicating data across Geo-graphically distributed data stores. We presented the design of a system prototype spanned across storage services of Amazon Web Services (AWS) and Microsoft Azure. We deployed the RESTful APIs to store, retrieve, delete, migrate and list objects across data stores. We also measured the latency of Get and data migration within and across regions, which shows it is tolerable to migrate a data bucket (the integration of objects belonging to the specific user) as its size is a magnitude of Megabytes (MBs).

References

1. Rajkumar Buyya, Yeo Chee Shin, Srikumar Venugopal, James Broberg, Ivona Brandic,Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility, Future Generation Computing System, 25, 599–616, 2009.
2. Peter M. Mell and Timothy Grance, SP 800-145. The NIST Definition of Cloud Computing, Gaithersburg, MD, United States, National Institute of Standards & Technology, 2011.
3. Yaser Mansouri, Adel Nadjaran Toosi and Rajkumar Buyya, Data Storage Management in Cloud Environments: Taxonomy, Survey, and Future Directions, ACM Computing Survey (ACM CSUR), 50, PP, 2017.
4. Yaser Mansouri and Rajkumar Buyya, To move or not to move: Cost optimization in a dual cloud-based storage architecture, Journal of Network and Computer Applications, 75, 223 - 235, 2016.
5. Yaser Mansouri, Adel Nadjaran Toosi and Rajkumar Buyya, Cost Optimization for Dynamic Replication and Migration of Data in Cloud Data Centers, IEEE Transactions on Cloud Computing, 2017.

[11] AWS Snowball. `https://aws.amazon.com/snowball/`.

[12] Microsoft Migration Accelerator. `https://azure.microsoft.com/en-au/blog/introducing-microsoft-migration-accelerator/`.

6. Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia, Above the clouds: A Berkeley view of cloud computing, Department Electrical Eng. and Computing Sciences, University of California, Berkeley, Rep. UCB/EECS, 28, PP, 2009.
7. Zhe Wu, Harsha V. Madhyastha,Understanding the Latency Benefits of Multi-cloud Webservice Deployments, ACM SIGCOMM Computer Communication Review, 43, 13-20, 2013.
8. Subramanian Muralidhar, Wyatt Lloyd, Sabyasachi Roy, Cory Hill, Ernest Lin, Weiwen Liu, Satadru Pan, Shiva Shankar, Viswanath Sivakumar, Linpeng Tang and Sanjeev Kumar, f4: Facebook's Warm BLOB Storage System, Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14), 383–398, 2014.
9. Zhe Wu, Michael Butkiewicz, Dorian Perkins, Ethan Katz-Bassett, Harsha V. Madhyastha, SPANStore: Cost-effective Geo-replicated Storage Spanning Multiple Cloud Services, ACM SIGCOMM 2013 Conference, SIGCOMM'13, 545–546, 2013.
10. Lei Jiao, Jun Li, Tianyin Xu, Wei Du, and Xiaoming Fu, Optimizing Cost for Online Social Networks on Geo-distributed Clouds, IEEE/ACM Transactions on Networking, 24, 99–112, 2016.
11. Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, Cesar A. F. De Rose, and Rajkumar Buyya, CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms, Software Practice and Experience, 41, 23-50
12. James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford, Spanner: Google's Globally Distributed Database, ACM Transactions on Computer Systems, 31, PP, 2013.
13. Rod Johnson, and Juergen Hoeller, Alef Arendsen, Thomas Risberg, and Dmitriy Kopylenko, Professional Java Development with the Spring Framework, John Wiley and Sons, Chapter 12, 2005.
14. Yaser Mansouri, Brokering algorithms for data replication and migration across cloud-based data stores, PhD Thesis, The University of Melbourne, Australia, 2017.