2019

Data compression based cost optimization for a multi-cloud data storage system

Abdullah
Iowa State University

Data compression based cost optimization for a multi-cloud data storage system

by

Abdullah

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:
Wensheng Zhang, Major Professor
Johnny Wong
Guong Song

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this thesis. The Graduate College will ensure this thesis is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2019

DEDICATION

I would like to dedicate this thesis to my parents and my wife without whose support I would not have been able to complete this work. I would also like to thank my friends and family for their loving guidance and assistance during the writing of this work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this thesis. First and foremost, Dr. Wensheng Zhang for his guidance, patience and support throughout this research and the writing of this thesis. His insights and words of encouragement have often inspired me and renewed my hopes for completing my graduate education. I would also like to thank my committee members for their efforts and contributions to this work: Dr. Johnny Wong and Dr. Guang Song.

ABSTRACT

A user wants to store data on cloud but given multiple heterogeneous cloud service providers available in the market, it is very difficult for the user to make the best choice. The cloud service providers differ in the services provided, cost of services, security, availability, latency, bandwidth etc. The number of cloud service providers in the market is already very large. Further, one cloud service provider like Amazon offers about 70 different cloud services for the users, which makes it very tedious for customers, especially small and medium size businesses to search through all the available cloud providers and make an informed decision. Cloud broker can facilitate users by selecting the best cloud service provider for their data meeting their service level agreement (SLA) requirements. The cloud broker keeps up-to date information about the cloud providers in the market, like the services provided, price of services, QoS, latency, datacenter regions, availability and security etc. The questions of concern are: how to select the datacenter which meets user's SLA requirements with minimum cost; and whether data compression can be helpful in saving storage space and bandwidth use. We develop the datacenter selection problem as an optimization problem with an objective function and constraints. To solve the optimization problem, we develop heuristic solutions which include three algorithms namely Simple Greedy Algorithm, Partly Compressed Algorithm and Fully Compressed Algorithm. First algorithm stores the data in plain text with no compression, while the cloud broker explores effects of data compression to save storage space and bandwidth in the next two algorithms. Simulation experiments are conducted to evaluate the optimal datacenter selection results from three algorithms. Data compression results in reducing the cost while still meeting the SLA requirements.

CHAPTER 1. OVERVIEW

Cloud computing has experienced phenomenal growth in recent years owing to the advancements in technology and improvements in internet speed and availability. Cloud service providers are offering new and improved services at competitive prices for personal, organizational and industrial users. More and more organizations are adapting cloud services for their business needs. Organizations and businesses ranging from small to huge, are choosing cloud services over setting up their own infrastructure for compute, storage, communication etc. needs. Cloud computing offers compute, storage, platform and other services to the users with minimal upfront cost. The cloud solutions are more scalable, easy to use and cost effective compared to having on premise infrastructure for these business needs. It frees organizations from the worries of buying and setting up hardware resources for hosting their own storage and compute infrastructure, and then recruiting and maintaining an employee team to use these resources. Cloud users can scale as their needs grow or shrink, owing to the pay as you go model of cloud services, which would be difficult and costly in case of on-premise infrastructure. Cloud storage is one of the widely used cloud services for storing user data on remote datacenters in huge scale. Cloud storage providers, such as Amazon, Rackspace, Google, etc. offer mass storage online to cloud users at competitive terms and costs. Users can face problems or may have to compromise with a lower level of service if they use services from only one cloud service provider. Vendor lock-in is a major problem when using only one cloud service provider. The users are locked in to the provider and may be subject to increase in prices, degradation of availability of data items, and suffer in case of cloud provider's closure of services, for example the provider

may become bankrupt. There may emerge new cloud service providers offering better prices and/or better quality of services, but the user is locked in with the current provider, who might charge a huge amount if the user wants to migrate to a new cloud provider, especially users with huge amount of data that will incur huge migration cost and will be time consuming. Further, the data might not be available to the user, if that cloud provider is experiencing outages. These concerns have led the users to store their data at more than one cloud service provider, or in multi-cloud storage. Users purchase storage services from more than one cloud provider to avoid problems like vendor lock-in and increase the availability of data. Availability of data is very important for most of the users, which can be increased in a multi-cloud storage system. Different cloud storage providers offer different availability warranties, but they still experience outages due to software bugs, user errors, administrator errors, malicious insiders, and natural catastrophes [14]. Data access latency can also be improved by storing data in a multi-cloud storage system. A solution to achieve the desired availability and reduced data access latency is to store replicas of the data in multiple datacenters, although this solution results in raised storage cost when the number of replicas increases. The cloud service providers are heterogeneous with respect to the services provided, cost of services, security, availability, latency, bandwidth etc. Given multiple heterogenous cloud service providers available in the market, it is very difficult for the user to make the best choice. The number of cloud service providers in the market is already very large and increasing with time. Further, each cloud service provider offers numerous different cloud services for the users, and even for the same service, for example storage, cloud service provider for example Amazon AWS, offers many different solutions which vary in cost, availability etc. The cloud service providers have their own set of interfaces, ways of accessing their services like consoles and graphical user interfaces. All this makes it very time consuming and tedious for customers, especially small and medium size businesses to search through all the available cloud providers, the services offered by each of them and make an informed decision. Also, for the user, accessing the services of more than one cloud

service provider, each in a different window for example with a different console or graphical user interface, will be quite cumbersome. The role of cloud broker arises as a result, which will be responsible for managing data on behalf of the customers. Cloud broker facilitates a user by providing a unified interface to manage all the user's accounts which may span over different cloud service providers, so that the user just opens one interface provided by the cloud broker and easily uses the services of multiple cloud providers. To this end the cloud broker creates a layer of abstraction between users and the cloud service providers. Cloud brokers provide more choices for the users and enable them in making opportunistic choices and be flexible with the cloud services. This leads to increased competitiveness between the cloud service providers for attracting a larger share of customers and then retaining the customers by offering innovative services at better prices. Further, the cloud broker can detect failure of any cloud provider's services and in a seamless manner to the user, takes appropriate steps to mitigate the effects of that failure for the end user, so the user might not even notice that failure and continue to receive the cloud services without interruption because of the cloud broker. Cloud broker selects the best cloud service datacenter for the user's data according to the user's requirements, i.e. the service level agreement (SLA) requirements. The user sends information about the data items to be stored on cloud storage. A cloud storage Service Level Agreement (SLA) articulates detailed user expectations of the cloud storage services such as availability, Get latency of the data items stored on cloud storage. In the competitive cloud service market, different cloud storage providers offer varying services with different levels of SLAs. In general, the more reliable the SLA requirements, the more the user must pay. Further, with the mass size of user data items to be stored in the cloud storage, the storage cost for users can be enormous. Thus, optimal selection of cloud storage datacenters under the SLA requirements with minimum cost is a critical decision for users. We need to develop algorithms to find the optimal datacenter allocation for user's data items that minimizes overall cost under the SLA requirements. These algorithms can be used by the cloud broker. The cloud broker collects all the in-

formation from the users about their data items including data access latency SLA. On the other hand, cloud broker keeps up to date information of all the features, services offered and cost of the services from all the cloud service providers. Next the cloud broker compares user's SLA requirements with the cloud service providers and selects the optimal datacenter for all the data items from all the users with the objective of minimizing overall cost. Cloud broker also leverages the benefits of data compression. By compressing data items, cloud broker can reduce the total storage space used at all storage datacenters. The requirement for the broker is to still meet the user's SLA requirements for compressed data items. Compression and decompression of data involves additional time needed for the compression and decompression operations, which add to the network latency for that data item. Cloud broker can only compress a data item if, even after adding the compression and decompression time to the network latency of the compressed data item, still meets user's latency SLA requirements. In this thesis, we propose three algorithms to be used by the cloud broker. The datacenter allocation problem is developed as an optimization problem. Two critical problems for the cloud broker are;

1. optimal placement of data items to minimize the storage cost while SLA requirements are met, and

2. using data compression to reduce storage cost and bandwidth usage under the given SLA requirements.

The main contributions of this work are:

1. A mathematical model for the data center selection problem

2. Three algorithms for data placement to minimize the storage cost for data items when the SLA requirements met,

3. Two different algorithms for evaluating the effects of data compression on cost optimization under SLA constraints

CHAPTER 2. PROBLEM STATEMENT AND PROPOSED SOLUTION

A customer wants to store data on multiple cloud datacenters, the customer sends a data storage request to cloud broker with information about the size of data, cost and service level requirements such as latency, bandwidth, availability requirements etc. Cloud broker compares the user's service level agreement (SLA) requirements with the cloud service provider's upto date information and selects the best datacenter for storing that data i.e. the datacenter which meets the SLA requirements and has minimum cost. We use $\overline{P}$ to denote the set of cloud service providers with storage datacenters, n_p the total number of cloud service providers and $P_i \in \overline{P}$, the ith datacenter. For each cloud storage datacenter P_i, the broker keeps upto date information; storage capacity $S(P_i)$, get request capacity $G(P_i)$, put request capacity $P(P_i)$, cost of storing a data item $C_s(P_i)$, cost per get request $C_G(P_i)$ and cost per put request $C_P(P_i)$. n_c denotes the total number of broker's customers. D_i is the set of all data items to be stored on cloud datacenters from user i. $d_{(i,k)} \in D_i$ is the kth data item from user i. The SLA requirements include the maximum tolerable get and put round trip latency for the data item denoted by $l_G(d_{i,k})$ and $l_P(d_{i,k})$ respectively. The broker can store the data item in compressed or uncompressed form at the cloud storage datacenter. When a data item is compressed, l_C and l_D denote the latency to compress and latency to decompress the data item respectively, which is added to the Data Access Latency for compressed data item. The datacenter selected for storing data item $d_{(i,k)}$ is denoted by $p(d_{i,k})$. A complete set of notations used is presented in the next section.

The problems/questions of concern are:

How to select the datacenter which meets user's SLA requirements with minimum cost?
Can data compression be helpful in saving storage space and bandwidth use?
Cloud broker maintains β number of replicas of user's data items. One datacenter can not have multiple replicas of the same data item. First replica is on the lowest cost datacenter possible and is used to serve the requests made for that data item, while other replicas ensure data availability and avoid problems like vendor lock-in.
We develop three different algorithms namely Simple Greedy Algorithm, Partly Compressed Algorithm and Fully Compressed Algorithm to make the selection of best datacenter for the storage of user's data. For each algorithm the input is an ordered sequence of data items sorted by the Get latency. The algorithms pick data items from the sequence one by one and determine the best datacenter for β replicas of that data item. The storage cost for storing a data item for one billing period is calculated for each datacenter and sorted in ascending order. Simple greedy algorithm stores the data in plain-text i.e. with no compression, it checks all the datacenters that meet the user's SLA requirements, starting with the lowest cost datacenter and assigns the data to the datacenter with lowest cost. Cloud broker explores the effects of data compression to save storage space and bandwidth in the next two algorithms. Partly compressed algorithm starts by storing the data in plain-text first, but as the datacenter with lowest storage cost gets filling up, it selects the least recently used data items at the datacenter and compresses them to create space for more data. Whereas, the Fully compressed algorithm tries to explore the benefits of fully compressing all the data to save storage space and bandwidth use.

2.1 Assumptions

- Data is low Storage intensive, or in other words stored once and accessed multiple times later.
- Data is GET intensive, i.e. the data is frequently accessed.

2.2 Notations

We introduce the following notations:

- n_p : The number of Cloud Service Providers (CSPs)
- $\overline{P} = \{P_1, P_2, ..., P_{n_p}\}$: The Set of CSPs
- For each $P_i \in \overline{P}$:
  - $S(P_i)$: the storage capacity of CSP P_i
  - $G(P_i)$: the GET capacity of CSP P_i
  - $P(P_i)$: the PUT capacity of CSP P_i
  - $C_s(P_i)$: the cost for storing a data item at P_i
  - $C_G(P_i)$: the cost for Getting a data item from P_i
  - $C_P(P_i)$: the cost of Putting a data item at P_i
- n_c : the number of users
- $D_i = \{d_{i,1}, d_{i,2}, ..., d_{i,m}\}$ the data set of user i.
- $\mid D_i \mid$: the size of data set of user i
- For each $d_{i,k} \in D_i$:
  - $s(d_{i,k})$: the size of $d_{i,k}$
  - $s(dc_{i,k}) = \{s(d_{i,k}) - 0.01\alpha.s(d_{i,k})\}$: the size of $d_{i,k}$ after compression
  - $f_G(d_{i,k})$: the GET frequency of $d_{i,k}$
  - $f_P(d_{i,k})$: the PUT frequency of $d_{i,k}$
  - $l_G(d_{i,k})$: the maximum tolerable GET latency of $d_{i,k}$
  - $l_P(d_{i,k})$: the maximum tolerable PUT latency of $d_{i,k}$
- $p(d_{i,k}) \in \overline{P}$: the CSP that stores $d_{i,k}$

- β : the required number of replicas for each data item
- $\forall P_i \in \overline{P}$: $\hat{D}_i = \{d_{j,k} | p(d_{j,k}) = P_i, j = 1, ..., n_c, k = 1, ..., |D_i|\}$
- For each user $i \in D_i$ and each $P_j \in \overline{P}$
  - $l_N(i, j)$: the round trip latency between user i and P_j
  - l_C: the latency for compressing a data item
  - l_D: the latency for decompressing a data item
  - C_C: the cost of compressing a data item
  - C_D: the cost of decompressing a data item
  - α: the compression ratio
- C_i: The total cost for storing and accessing data items of user i, consisting of the storage cost, Get cost and Put cost for data items from user i. Storage cost is calculated by multiplying unit storage price with the size of the data stored in that datacenter. Get and Put cost is calculated as a product of number of get and put requests and the per get and put request price of the datacenter respectively.

$$C_i = \sum_{j=1}^{|D_i|} \{C_s(p(d_{i,j})) + f_G(d_{i,j}).C_G(p(d_{i,j})) + f_P(d_{i,j}).C_P(p(d_{i,j}))\}$$

For Compressed Data item:

C_i: The total cost for storing and accessing compressed data items of user i, which in addition to the cost of uncompressed data item, includes the cost of compressing and decompressing data items from user i. C_i is

$$\sum_{j=1}^{|D_i|} \{C_s(p(dc_{i,j})) + f_G(dc_{i,j}).C_G(p(dc_{i,j})) + f_P(dc_{i,j}).C_P(p(dc_{i,j})) + C_C(dc_{i,j}) + C_D(dc_{i,j})\}$$

2.3 Problem Objective and Constraints

We formulate the optimal datcenter selection problem as an optimization problem with an objective function and constraints. The objective is a minimization function to minimize broker's total cost C, which is the sum of the costs of storing and accessing data items from all users. The set of constraints consists of datacenter's storage capacity, Get request capacity, Put request capacity and user's SLA requirements.

Cost Minimization Objective: We aim to minimize the total cost of the broker for storing all data items from all users including the storage cost, Get cost, Put cost, compression and decompression cost. Broker's total cost C is calculated as the sum of all the individual user costs denoted by C_i introduced in notations section.

Broker's total cost: $$C = \sum_{i=1}^{n_c} C_i$$

Constraints: The cost minimization function is subject to the following constraints.

- Storage Capacity: The first constraint is to ensure storing data items at a datacenter up to the maximum storage capacity of the datacenter. The data items can be stored in compressed or uncompressed form. The constraint (1a) considers the reduced size of compressed data items.

  Sum of size of data items from all users stored at P_j should be less than max storage capacity of P_j

  $$\sum_{d \in \hat{D}_i} s(d) \leq S(P_j) \qquad (1)$$

  $$\sum_{dc \in \hat{D}_i} s(dc) \leq S(P_j) \qquad (1a)$$

- GET/PUT Capacity: Each datacenter has a maximum capacity of serving Get and Put requests. The second constraint ensures that the maximum number of Get and Put requests sent from all users to the datacenter are within the datacenter's Get and Put capacity respectively. The constraints (2a) consider the reduced size compressed data items.

  Sum of Get/Put requests from all users for data items at Provider j should be less than max GET/PUT capacity of Provider j

$$\left.\begin{array}{l} \sum\limits_{d\in\hat{D}_i} f_G(d) \leq G(P_j) \\ \\ \sum\limits_{d\in\hat{D}_i} f_P(d) \leq P(P_j) \end{array}\right\} (2)$$

$$\left.\begin{array}{l} \sum\limits_{dc\in\hat{D}_i} f_G(dc) \leq G(P_j) \\ \sum\limits_{dc\in\hat{D}_i} f_P(dc) \leq P(P_j) \end{array}\right\} (2a)$$

- Data SLA: Get and Put latency requirements are set by the user in the service level agreement (SLA), which must be satisfied for the data item to be stored at a datacenter. For uncompressed data items, it is the round trip latency between the user and the datacenter storing that data item. In case of compressed data items represented by constraints (3a), the latency to compress and decompress the data item is accommodated by adding it to the Data Access Latency.

  Data item $d_{i,k}$ be stored on Provider P_j, Only if P_j satisfies both Get and Put SLA for $d_{i,k}$;

$$\forall d_{i,j} \in \bigcup_{i=1}^{n_c} : \left\{\begin{array}{l} l_G(d_{i,j}) \geq l_N(i, p(d_{i,j})) \\ \\ l_P(d_{i,j}) \geq l_N(i, p(d_{i,j})) \end{array}\right\} \quad (3)$$

$$\forall dc_{(i,j)} \in \bigcup_{i=1}^{n_c} : \left\{\begin{array}{l} l_G(dc_{(i,j)}) \geq \{l_N(i, p(d_{(i,j)})) + l_C(d_{(i,j)}) + l_D(d_{(i,j)})\} \\ \\ l_P(dc_{(i,j)}) \geq \{l_N(i, p(d_{(i,j)})) + l_C(d_{(i,j)}) + l_D(d_{(i,j)})\} \end{array}\right\} \quad (3a)$$

The final optimization problem is to determine

Optimal datacenter $p(d_{i,k})$

for each data item

$$d_{i,j} \in \bigcup_{i=1}^{n_c} D_i$$

that minimizes the broker's total cost

$$C = \sum_{i=1}^{n_c} C_i$$

Subject to

Constraints (1), (2), (3) for uncompressed data items and

Constraints (1a), (2a), (3a) for compressed data items.

It can be proved that this problem is NP-hard by reducing it from the generalized assignment problem [13].

2.4 Proposed Solution

To solve the optimization problem formulated above, we develop heuristic solutions which include three algorithms to make the optimal datacenter allocations . The broker receives data storage requests from different users and adds the data items to a list of data items to be stored at cloud storage datacenters. The data items in this list are sorted in ascending order of Get latency and provided as input to the optimal datacenter selection algorithms.

Thus for each algorithm, the input is an ordered sequence of data items, denoted as;

$\overline{D} = \{< \overline{d}_1, \overline{d}_2, ..., \overline{d}_n >\}$

such that:

1. $\{\overline{d}_1, \overline{d}_2, ..., \overline{d}_n\} = \bigcup_{i=1}^{n_c} D_i$
2. $l_G(\overline{d}_j) \geq l_G(\overline{d}_j + 1)$ for $j = \{1, 2, ..., n-1\}$

The algorithms pick data items from this sequence one by one starting from the head of the list and select the optimal datacenter for each of the data items until all data items are allocated datacenters. The algorithms also make use of the user to datacenter cost list, which contains the cost of storing a data item from the user to each of the cloud storage datacenter. The cost list is sorted from lowest cost to the highest cost datacenter for the user's data item. After the allocations are done, broker calculates the total cost for a billing period. The three algorithms are described in detail as follows;

2.4.1 Heuristic 1: Simple Greedy Algorithm

The Simple Greedy Algorithm, as the name suggests behaves in a greedy manner and stores the data in plain-text i.e. with no compression. It picks up the first data item from the ordered input sequence and the lowest cost datacenter from the cost list for that data item's user. First the algorithm checks for the SLA requirements by comparing the Get latency SLA of the data item to the round trip data access latency, if the data item is allocated this datacenter. If the round trip data access latency is less than or equal to the

Algorithm 1 Simple Greedy Algorithm

1: **for** $i = 1$ to n **do**
2: Let $\overline{d}_i = d_{j,k}$
3: **while** No of Replicas of $d_{j,k} \leq \beta$ **do**
4: determine $p_{j,k} \in \overline{P}$ that minimize:
5: $\{C_s(p(d_{i,j})) + f_G(d_{i,j}).C_G(p(d_{i,j})) + f_P(d_{i,j}).C_P(p(d_{i,j}))\}$
6: Subject to:
7: constraints (1), (2) and (3) are satisfied.
8: **end while**
9: **end for**

Get latency, the algorithm checks if the datacenter has enough storage capacity and Get/Put capacity available for the data item next. If storage space and/or Get/Put capacity is not available, pick the next higher cost datacenter in the user to datacenter cost list and repeat the above steps. Otherwise, the data item is allocated to this datacenter and move to the next data item in the data item sequence. In this manner the algorithm checks the datacenters one by one for the data item and assigns the data item to the first datacenter that meets SLA requirements and has enough storage space available without compressing data. The broker keeps β replicas of the data item, so the algorithm repeats the above steps for selecting up to β datacenters for the same data item. Each replica is on a different datacenter to maximize data availability. Algorithm 1 shows the simple greedy algorithm.

2.4.2 Heuristic 2: Partly Compressed Algorithm

In this heuristic solution, the data items are stored in plain-text at start in the cloud storage datacenter. After certain number of allocations, the storage space at the datacenter with lowest storage cost will become full. But the cloud broker would still want to store a

Algorithm 2 Partly Compressed Algorithm

Compute Cost: C_x {Compute cost for all Datacenter Sorted in Ascending Order}

$= \{C_s(p(dc_{i,j})) + f_G(dc_{i,j}).C_G(p(dc_{i,j})) + f_P(dc_{i,j}).C_P(p(dc_{i,j})) + C_C(dc_{i,j}) + C_D(dc_{i,j})\}$

while No of Replicas of $d_{j,k}$ or $dc_{j,k} \leq \beta$ **do**

  determine $p_{j,k} \in \overline{P}$ that minimize:

  $\{C_s(p(dc_{i,j})) + f_G(dc_{i,j}).C_G(p(dc_{i,j})) + f_P(dc_{i,j}).C_P(p(dc_{i,j})) + C_C(dc_{i,j}) + C_D(dc_{i,j})\}$

  Subject to:

    (1), (2) and (3) are satisfied.

  for $x = 1$ to n_p **do**

    Assume $\overline{d}_i = d_{j,k}$

    if Constraint (2) and (3) are satisfied **then**

      if Constraint (1) is satisfied **then**

        y = 0;

      else

        Find y Least Recently Used data items at datacenter x such that

        Compress (y) results in Constraint (1a),(2a),(3a) satisfied for all

      end if

    end if

    if y data items found **then**

      Let $\overline{d}_i = d_{j,k}$

      Compress the y data items

      break;

    else

      $x \leftarrow x + 1$

    end if

  end for

end while

few more data items at this datacenter to reduce cost. One way can be to reserve more storage space at this datacenter, but that will incur additional storage cost. An alternative is to compress some of the data items that are stored on this datacenter and have not been accessed for the longest time. In this way the broker can save cost by storing more data items at this lower cost datacenter without reserving more storage space. But the SLA requirements for the data items still need to be satisfied, meaning that data items at a datacenter will only be compressed if the SLA requirements are still satisfied for all data items after compressing the least frequently data items. Otherwise the data items already stored at the datacenter are not compressed and the new data item is stored at a higher cost datacenter. The algorithm starts by picking up the first data item from the sorted input sequence and the lowest cost datacenter from the user to datacenter cost list for that data item's user. First the algorithm checks for the SLA requirements by comparing the Get latency SLA of the data item to the round trip data access latency if the data item is stored at this datacenter. If the data access latency is less than or equal to the Get SLA; the algorithm checks if the datacenter has enough Get/Put capacity available for the data item next, if both above conditions are satisfied, the algorithm checks if the datacenter has enough storage capacity to store this data item, if storage capacity is available there is no need to compress some of the already stored data items and this datacenter is allocated to the data item. Otherwise, to create storage space for this new data item, the algorithm calculates y, the number of data items needed to be compressed, which depends on the compression ratio α and size of the new data item. Next it tries to find y least recently used data items stored at this datacenter which if compressed, will still meet the SLA requirements. If such y data items are found, they are compressed to free storage space and the new data item is stored on this datacenter. Otherwise, the new data item is not stored at this datacenter and the algorithm moves to the next higher cost datacenter in the user to datacenter cost list. The broker keeps β replicas of the data item, so the algorithm repeats the above steps for selecting up to β datacenters for the same data item. Each

replica is on a different datacenter to maximize data availability. Algorithm 2 shows the partly compressed algorithm.

2.4.3 Heuristic 3: Fully Compressed Algorithm

In this heuristic solution, the benefits of compression are leveraged and data items are compressed whenever possible. The latency requirement will still need to be satisfied after

Algorithm 3 Fully Compressed Algorithm

1: **for** $i = 1$ to n **do**
2: Let $\overline{d}_i = d_{j,k}$
3: **while** No of Replicas of $d_{j,k}$ or $dc_{j,k} \leq \beta$ **do**
4: determine $p_{j,k} \in \overline{P}$ that minimize:
5: $\{C_s(p(dc_{i,j}))+f_G(dc_{i,j}).C_G(p(dc_{i,j}))+f_P(dc_{i,j}).C_P(p(dc_{i,j}))+C_C(dc_{i,j})+C_D(dc_{i,j})\}$
6: Subject to:
(1a), (2a) and (3a) are satisfied.
7: **end while**
8: **end for**

adding the time needed for compression and decompression of data items. If the SLA requirements are still satisfied after adding the compression and decompression latency, the data item is compressed. Otherwise data item is not compressed and stored at a higher cost. The algorithm works similar to the simple greedy algorithm, except that the data items are compressed. It picks up the first data item from the sorted input sequence and the lowest cost datacenter from the user to datacenter cost list for that data item's user. First the algorithm checks for the SLA requirements by comparing the Get latency SLA of the compressed data item to the round trip data access latency if the compressed data item is stored at this datacenter. If the data access latency is less than or equal to the Get SLA; the algorithm checks if the datacenter has enough storage capacity and Get/Put capacity

available for the compressed data item next. If storage space and/or Get/Put capacity is not available, pick the next higher cost datacenter in the user to datacenter cost list and repeat the above steps. Otherwise, the compressed data item is allocated to this datacenter and move to the next data item in the data item sequence. In this manner the algorithm checks the datacenters one by one for the data item and assigns the data item to the first datacenter that meets SLA requirements and has enough storage space available for the compressed data item. The broker keeps β replicas of the data item, so the algorithm repeats the above steps for selecting up to β datacenters for the same data item. Each replica is on a different datacenter to maximize data availability.Algorithm 3 shows the fully compressed algorithm.

CHAPTER 3. REVIEW OF LITERATURE

In [1] Liu et al propose ES3 which aims to minimize cloud broker's payment cost to the cloud service providers while still meeting the SLO guarantees. ES3 finds the optimal data allocation and resource reservation schedules to minimize the cost and meeting SLO guarantee. The authors also propose a genetic algorithm based data allocation adjustment algorithm, which reduces the Get/put rates in each datacenter which in turn results in maximizing the reservation benefit. Further algorithms are proposed to reduce the cost by dynamic request redirection, grouping the Get requests to minimize the Get cost and delaying the Put requests for cost efficiency. The authors claim improved performance in terms of cost minimization with SLO guarantees by performing trace based and real world cloud experiments. In [2] Mansouri et al. develop the optimal datacenter selection with an objective function and constraints. The authors propose two different algorithms which can be used by a cloud broker to minimize the cost. First algorithm selects a set of datacenters for data items to minimize the storage cost under the given availability requirements. While the second algorithm splits the data items into fixed number of chunks, each chunk has a fixed number of replicas, the algorithm selects optimal datacenter allocation for all the chunks, so that the availability is maximized under a given budget. In [3] Zhang et al. propose CHARM which is a cost-efficient multi-cloud storage system. This scheme selects the minimum cost storage datacenter from the set of all available cloud storage datacenters with high availability and to avoid problem of vendor lock-in. CHARM makes use of replication and erasure coding for redundancy. It uses data item's access history to determine which of the two redundancy mechanisms incur lower storage cost. It provides a cloud-based redun-

dant storage system which monitors data item's access pattern and dynamically optimizes the placement of the data objects in a cost-efficient way while taking into account SLOs. In [4] Chang et al. develop objective function for selecting the best cloud storage provider and make use of probability in the algorithms for the selection of datacenter. They store the data items on multiple cloud storage datacenters by replicating the data items. In [5] Libdeh et al. propose RACS (Redundant Array of Cloud Storage) which transparently uses erasure coding and splits the data items into several chunks and then stores these chunks on multiple cloud datacenters. The main aim is to increase data availability even in the case of failure or outage of some cloud service provider and avoid vendor lock-in problems. RACS follows the idea similar to that of RAID which is well known technique for file systems and storage disks. The authors concluded that by using RACS the cost of migrating to another cloud service provider can be reduced seven times. But RACS does not monitor the usage of the data items and hence can not use it make cost efficient data placement. In [6] Le et al. introduce CloudCmp, which enables users in selecting a cloud service provider according to their requirements and budget. It measures elastic computing, persistent storage and network services by comparing four well known cloud service providers. These services have a direct effect on user application's performance. In [7] Ford et al. used Markov chain modelling for their analytical measurements and predicted the data item availability in a datacenter based on these measurements. For their experiments they examined object availability in Google's mainstream storage datacenters. The authors focused on analyzing the availability of different components like machines, racks, multi-racks in Google storage clusters. In [8] Bonvin et al. design the problem on the concept of a virtual economy. In their work each of the object partitions run as an individual optimizer. Each optimizer calculates the net profit maximization based on the utility of this partition and this partition's storage and maintenance. After that based on this, it chooses one of three decisions about itself from replicating, migrating or removing itself. In [9] Placek et al. propose the concept of a Storage exchange. The authors present the idea of a broker market for

cloud storage services where the owners of cloud storage services are allowed to exchange the storage. In [10] Sripanidkulchai et al. worked on achieving cloud services with focus on high availability and proposed three solutions to achieving that. The main requirements for cloud computing studied by the authors were high availability, mass scale deployment and problem resolution. First solution proposed was to extend the architecture across different cloud service providers, second being developing new virtualization technologies and the third to observe the technical differences between different sites that led to the observed difference in service availability. In [11] Wu et al. propose SPANStore which is a key-value data storage system. It stores the data items over multiple cloud storage datacenters to minimize cost and guarantee SLOs. The authors formulate the datacenter selection problem as an integer programming problem which is NP-hard as it fails to consider the capacity limitation of datacenters. Further the authors did not consider reserving resources to minimize the cost. In [12] Bermbach et al. propose MetaStorage which uses full replication to store data items on multiple cloud storage providers targeting to achieve high data availability. MetaStorage is highly scalable as it uses a distributed hashtable to allocate data items to the available cloud storage datacenters. This scheme does not optimize the data placement to the minimum cost datacenters, and on the other hand all data items are fully replicated on different storage datacenters, which raises the network traffic and storage space, and as a result the overall cost is increased.

CHAPTER 4. PERFORMANCE EVALUATION

We conducted simulation based experiments to evaluate the performance of the three optimal datacneter allocation algorithms described in previous section and compare their performance. First we briefly introduce CloudSim Plus simulator that was used in this work to simulate the cloud storage scenarios and conducting the experiments.

4.1 CloudSim Plus Simulator

CloudSim Plus is popular, state of the art simulator for modeling the cloud service providers, services offered by these providers like storage datacenters, compute resources in the form of compute instances etc. It offers extensive features for modeling simulation experiments very close to real world cloud service providers, and make it convenient to perform extendable cloud service scenarios. CloudSim Plus implements all the low-level programs for a cloud service provider resembling real world cloud service providers, which enables users to spend more time on the designing part of experiments, scenarios for the investigations to be done, instead of worrying about the implementation of for example cloud storage datacenter or a compute instance. Cloud computing has emerged as the leader for providing off-premise, reliable, secure, low upfront cost services. These services offer higher levels of fault tolerance and are scalable as the needs grow. As the amount of research in cloud computing has increased tremendously in recent times, there is need for reliable evaluation tools for testing new cloud policies and applications. These tools help in evaluating the performance of these methodologies by performing repeatable and

controllable experiments. Computer based simulation has gained importance as real testbed experiments incur cost, limit the size of experiment to the real world testbed. CloudSim Plus is an impressive tool because it is suitable to design and evaluate different simulation scenarios and run them efficiently on a personal computer.

4.2 Experimental Setup

Next we introduce the experiment settings.

We simulated cloud datacenters distributed geographically in six different regions. Each region has one datacenter. The Get/Put latency distribution depends on the distance between the user and the datacenter. The price of unit storage, Put and Get operations follow the real storage, Put and Get prices of Amazon AWS listed online. The minimum number of replicas of each data item β is set to 2. The Get latency SLA for data items range from 13 ms to 17 ms, averaging around 15 ms. We experimented with the data access latency for a data item ranging close to the Get SLA and very close to the Get SLA. For compressed data items, we experimented with three different values of α, the compression ratio with which the compressed data item's size is reduced, at 30%, 25% and 20% of the original data item size. The number of data items from each user are chosen randomly. Each data item has the same size. The experiments were run with different total number of data items from all users, for example, in one experiment equal to 10000 data items. The Get frequency is chosen randomly from 10000 to 15000 Get requests per billing period. As per assumptions for this work, data is stored once and accessed multiple times later, hence the Put frequency is small, ranging from 1 to 5 for a data item. In the simulation, billing period is set to 30 calendar days. Broker's total cost is calculated for the billing period by summing up all users costs. For each user, Get latency from user to all datacenters is calculated and stored in a list. Further, for each user, the storage cost of storing user's data item at each datacenter is calculated and stored in a list, sorted from lowest cost to the highest cost as shown in Figure 4.1. We ran each experiment 20 times and report the

average performance of each of the datacenter allocation algorithms. We show the results from some representative experiments to evaluate the performance of the three algorithms here.

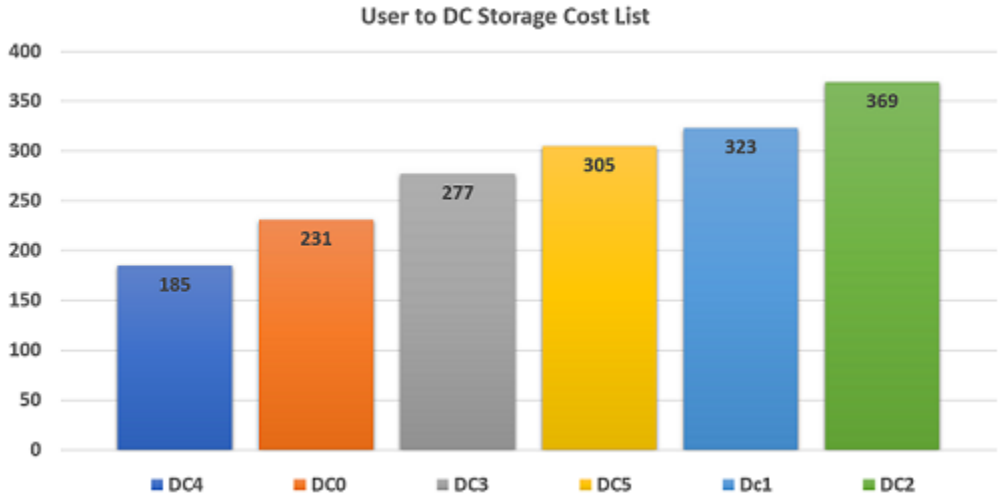

Figure 4.1 User to Datcenter Sorted Cost List

In Experiment 1, the parameters for the optimal datacenter selection problem are as follows; The latency SLA requirements are ranging from 13 ms to 17 ms and averaging around 15 ms. The data access latency ranges from 10 ms to 12 ms, averaging around 11 ms for uncompressed data items, while for the compressed data items in the same experiment the data access latency ranged from 12 ms to 14 ms, averaging around 13 ms after adding the compression and decompression time. Total number of data items from all users is 10000. Table 4.1, 4.2 and 4.3 show Experiment 1 with compression ratio α at 30%, 25% and 20% respectively.

Table 4.1 Datacenter Selection Parameters-Experiment 1a

Parameter	**Value**
User Latency SLA	Average 15 (Range 13-17)
Data Access Latency	Average 11 (Range 10-12)
Data Access Latency with Compression	Average 13 (Range 12-14)
Compression Ratio	30%
No of Data items	10000

Table 4.2 Datacenter Selection Parameters-Experiment 1b

Parameter	**Value**
User Latency SLA	Average 15 (Range 13-17)
Data Access Latency	Average 11 (Range 10-12)
Data Access Latency with Compression	Average 13 (Range 12-14)
Compression Ratio	25%
No of Data items	10000

Table 4.3 Datacenter Selection Parameters-Experiment 1c

Parameter	**Value**
User Latency SLA	Average 15 (Range 13-17)
Data Access Latency	Average 11 (Range 10-12)
Data Access Latency with Compression	Average 13 (Range 12-14)
Compression Ratio	20%
No of Data items	10000

Figure 4.2 shows broker's total cost for the above experiments. We can see that the simple greedy algorithm incurs highest cost and has the same cost at different compression ratios because of storing data items in plain-text. Fully compressed algorithm outperforms the other two algorithms in all 3 compression ratios, and higher the compression ratio, better the algorithm performs. Partly compressed algorithm ranges between the simple greedy and fully compressed algorithms. Compression ratio effects the performance of partly compressed algorithm, as do the number of the data items which get compressed by

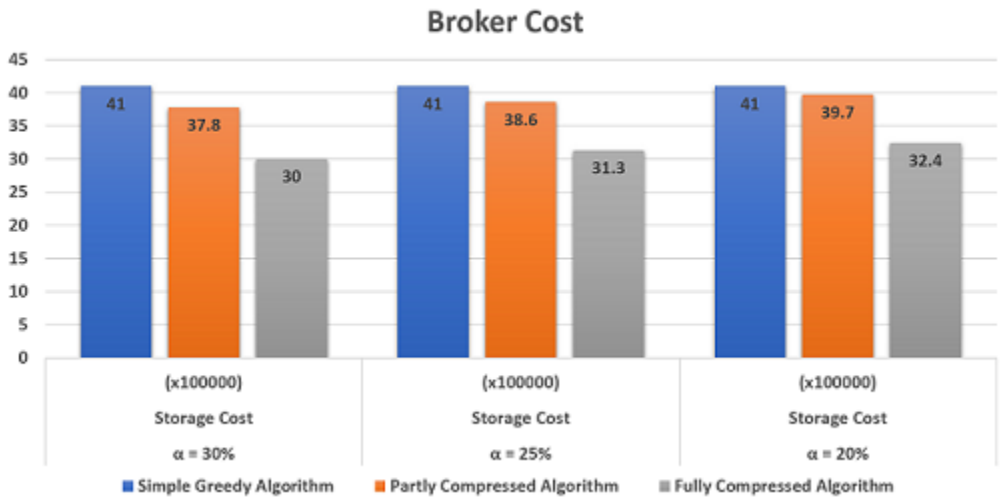

Figure 4.2 Broker Cost-Experiment 1a, 1b, 1c

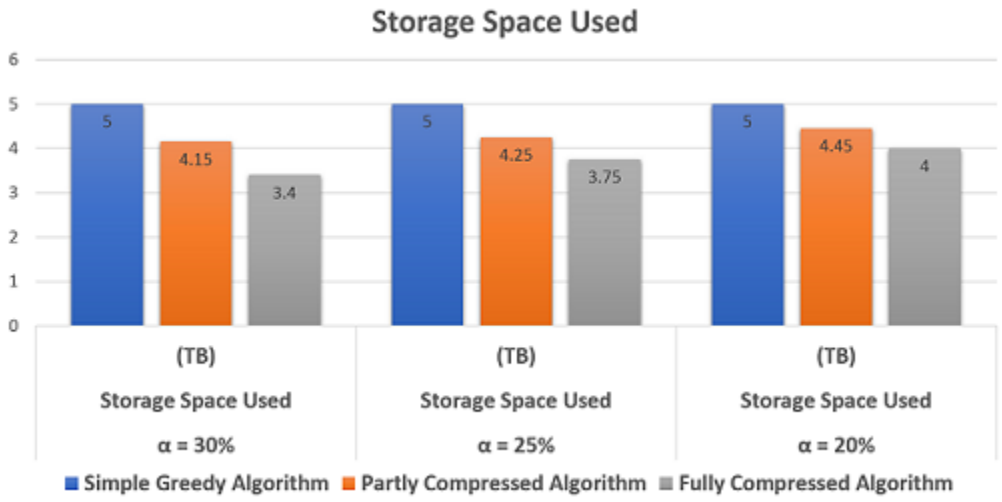

Figure 4.3 Storage Space Used-Experiment 1a, 1b, 1c

the partly compressed algorithm. The higher the number of data items compressed by the partly compressed algorithm, the closer it gets to the performance of fully compressed algorithm. Figure 4.3 shows the storage space reserved at the cloud datacenters by each of the three algorithms in the above experiments. Naturally simple greedy algorithm uses the most and fully compressed algorithm uses the least amount of storage, while partly

compressed algorithm reserves storage space between the other two algorithms.

In Experiment 2, the parameters for the optimal datacenter selection problem are as follows; The latency SLA requirements are ranging from 13 ms to 17 ms and averaging around 15 ms, the data access latency for a data item ranged from 12 ms to 14 ms, averaging around 13 ms for uncompressed data items, while for the compressed data items in the same experiment the data access latency ranged from 14 ms to 16 ms, averaging around 15 ms after adding the compression and decompression time. Total number of data items from all users is 10000. Table 4.4, 4.5 and 4.6 show Experiment 2 with compression ratio α at 30%, 25% and 20% respectively.

Table 4.4 Datacenter Selection Parameters-Experiment 2a

Parameter	**Value**
User Latency SLA	Average 15 (Range 13-17)
Data Access Latency	Average 13 (Range 12-14)
Data Access Latency with Compression	Average 15 (Range 14-16)
Compression Ratio	30%
No of Data items	10000

Table 4.5 Datacenter Selection Parameters-Experiment 2b

Parameter	**Value**
User Latency SLA	Average 15 (Range 13-17)
Data Access Latency	Average 13 (Range 12-14)
Data Access Latency with Compression	Average 15 (Range 14-16)
Compression Ratio	25%
No of Data items	10000

Table 4.6 Datacenter Selection Parameters-Experiment 2c

Parameter	**Value**
User Latency SLA	Average 15 (Range 13-17)
Data Access Latency	Average 13 (Range 12-14)
Data Access Latency with Compression	Average 15 (Range 14-16)
Compression Ratio	20%
No of Data items	10000

Figure 4.4 shows broker's total cost for the above experiments. As in experiment 1,we can see that the simple greedy algorithm incurs highest cost and has the same cost at different compression ratios because of storing data items in plain-text. Simple greedy algorithm is able to allocate all the data items in the input sequence. In terms of cost, fully compressed algorithm outperforms the other two algorithms in all 3 compression ratios, and higher the compression ratio, better the algorithm performs. But as the data access latency plus compression and decompression time is very close to the Get SLA, some of the data items, when compressed, fail to meet the SLA requirements and hence can not be compressed. As a result fully compressed algorithm was unable to allocate all data items in compressed form. Experiment results showed that fully compressed algorithm was only able to allocate between 80% to 85% of the data items in the input sequence. The rest of the data items would still need to be stored in uncompressed form, which increases the cost for fully compressed algorithm as compared to experiment 1, as can be seen in figure 4.3 also. Partly compressed algorithm is able to allocate datacenters for all the data items in the input sequence. As for the cost, partly compressed algorithm ranges between the simple greedy and fully compressed algorithms. Compression ratio effects the performance of partly compressed algorithm, as do the number of the data items which get compressed by the partly compressed algorithm. The higher the number of data items compressed by the partly compressed algorithm, the closer it gets to the performance of fully compressed

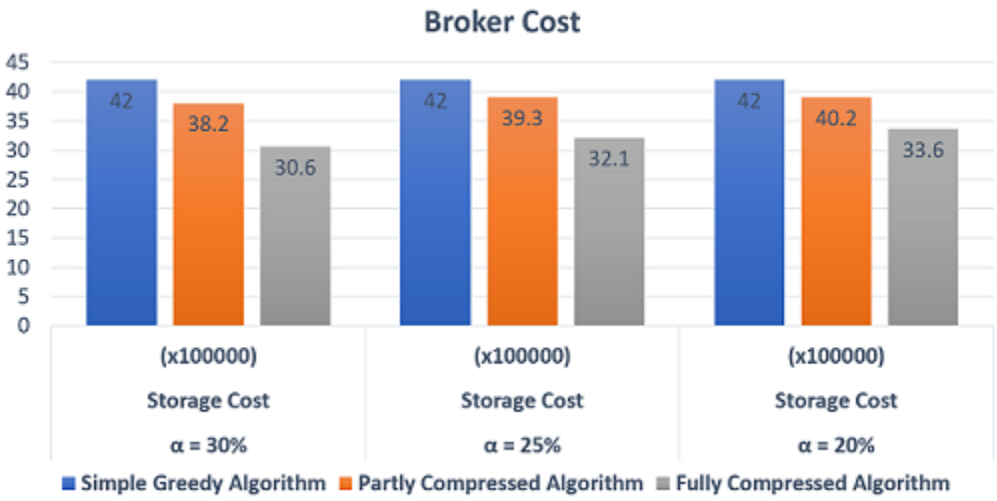

Figure 4.4 Broker Cost-Experiment 2a, 2b, 2c

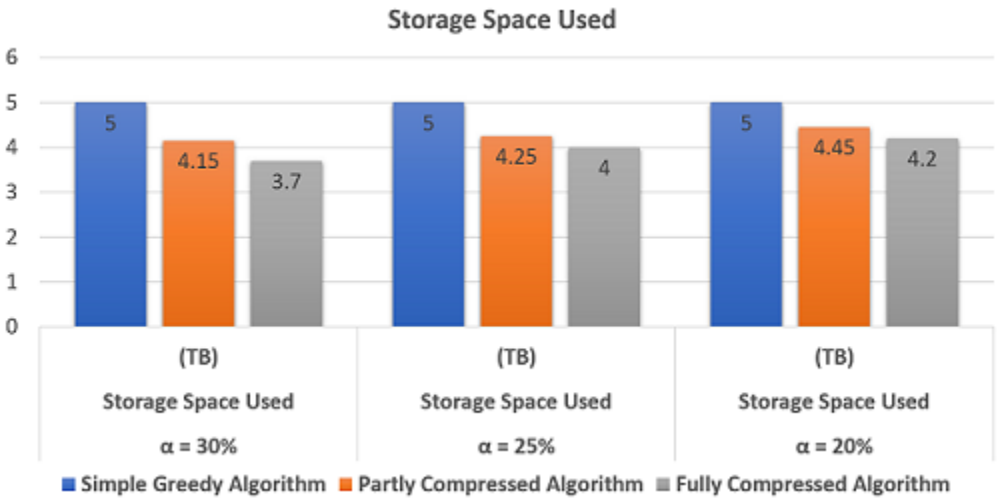

Figure 4.5 Storage Space Used-Experiment 2a, 2b, 2c

algorithm. Figure 4.5 shows the storage space reserved at the cloud datacenters by each of the three algorithms in the above experiments. Naturally simple greedy algorithm uses the most and fully compressed algorithm uses the least amount of storage, while partly compressed algorithm reserves storage space between the other two algorithms. As compared to

experiment 1, the storage space used by fully compressed and partly compressed algorithms is increased, because less number of data items can now be compressed owing

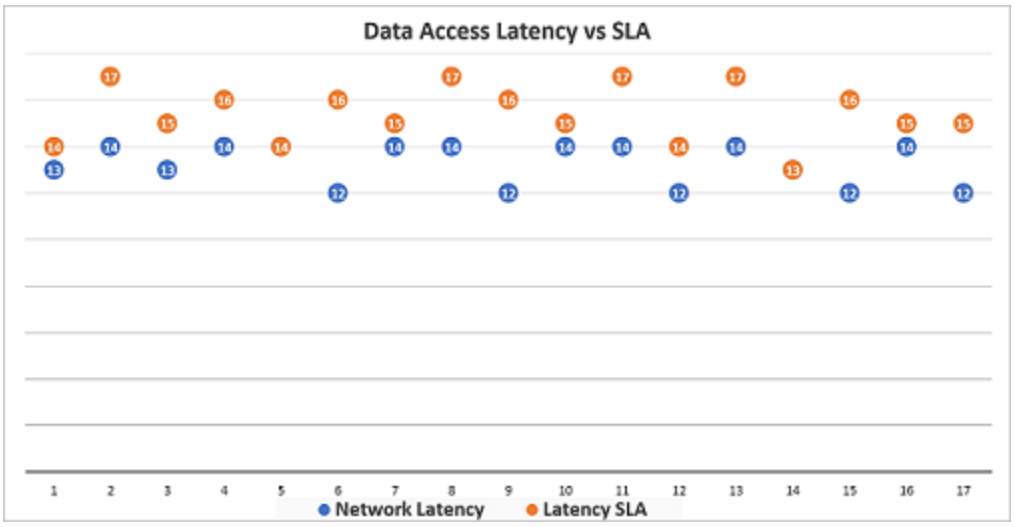

Figure 4.6 Data Access Latency vs SLA

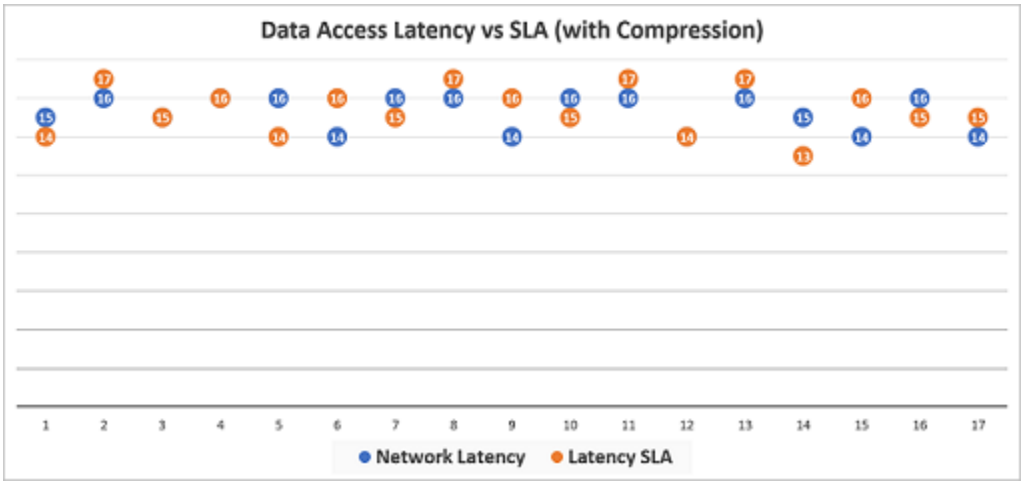

Figure 4.7 Data Access Latency vs SLA-With Compression

to narrow difference between data access latency and Get SLA in case of compressed data items. Figure 4.6 and 4.7 also make this scenario clear, figure 4.6 shows the Get SLA and data access latency for uncompressed data items, and it can be seen that for almost all the data items data access latency is below or equal to Get SLA and hence, these data items

can be allocated to the datacenters. Figure 4.7 depicts the Get SLA and data access latency plus compression and decompression time. It can be seen that after adding the compression and decompression time, data access latency for some data items becomes greater than the Get SLA and now these data items can not be allocated to the datacenters. These data items must be stored in uncompressed form at a higher cost.

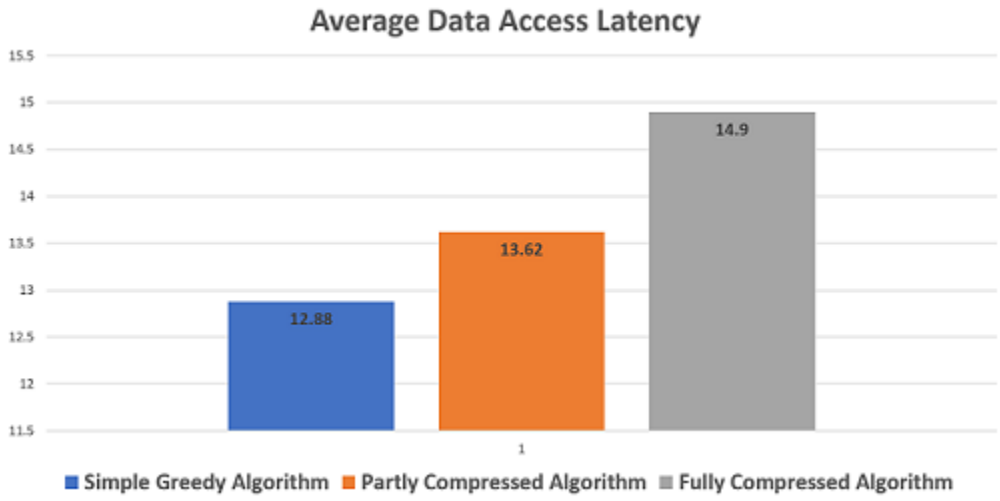

Figure 4.8 Average Data Access Latency Comparison

Figure 4.8 shows average data access latency for the three algorithms and thus a tradeoff scenario between the partly compressed and fully compressed algorithms. We can see that partly compressed algorithm perform better than fully compressed algorithm in terms of average data access latency while still making use of compression to save cost and performs better than simple greedy algorithm in terms of cost. So if the user wants better average data access latency, partly compressed algorithm proves to be the optimal algorithm.

CHAPTER 5. SUMMARY AND CONCLUSION

In this thesis, we formulated a mathematical model for the optimal datacenter selection problem for data items to be stored on cloud storage datacenters with the objective of minimizing the overall cost and defined the constraints to meet the SLA requirements. We have also proposed three heuristic algorithms for the optimal datacenter selection problem namely Simple Greedy Algorithm, Partly Compressed Algorithm and Fully Compressed Algorithm. These algorithms can be used by cloud service broker to select the optimal datacenters for user's data items and thus minimize the total cost. We used the idea of using data compression to reduce the storage space and bandwidth use to save the storage cost. Fully compressed algorithm compresses all the data items whenever possible within the constraints, whereas partly compressed algorithm attempts to make the optimal use of the lowest cost datacenters by compressing the data items already stored on that datacenter and have not been accessed recently. Simulation experiments were conducted to evaluate the performance of the three algorithms under different settings of the parameters. Simple greedy algorithm incurs highest cost but minimum average data access latency. Fully compressed algorithm incurs lowest cost but highest average data access latency and in the case where SLA latency is very close to data access latency, it fails to allocate datacenters for all data items in compressed form. Partly compressed algorithm incurs less cost than simple greedy algorithm and succeeds in allocating all data items under different data access latencies.

BIBLIOGRAPHY

[1] G. Liu, H. Shen, H. Wang, "An Economical and SLO-Guaranteed Cloud Storage Service Across Multiple Cloud Service Providers", IEEE Transactions on Parallel and Distributed Systems, vol. 28, no. 9, pp. 2440-2453, 2017.

[2] Y. Mansouri, A. N. Toosi, and R. Buyya, "Brokering algorithms for optimizing the availability and cost of cloud storage services," in 5th Int. Conf. on Cloud Comp. Tech. and Science, 2013, pp. 581–589.

[3] Q. Zhang, S. Li, Z. Li, Y. Xing, Z. Yang, and Y. Dai, "CHARM: A Cost- Efficient Multi-Cloud Data Hosting Scheme with High Availability," IEEE Trans. on Cloud Computing, vol. 3, no. 3, pp. 372–386, 2015.

[4] C.-W. Chang, P. Liu, and J.-J. Wu, "Probability-based cloud storage providers selection algorithms with maximum availability," 2012 41st International Conference on Parallel Processing, vol. 0, pp. 199–208,2012.

[5] H. Abu-Libdeh, L. Princehouse, and H. Weatherspoon, "Racs: a case for cloud storage diversity," in Proceedings of the 1st ACM symposium on Cloud computing, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 229–240.

[6] A. Li, X. Yang, S. Kandula, and M. Zhang, "Cloudcmp: comparing public cloud providers," in Proceedings of the 10th ACM SIGCOMM conference on Internet measurement, ser. IMC '10. New York, NY, USA: ACM, 2010, pp. 1–14.

[7] D. Ford, F. Labelle, F. I. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan, "Availability in globally distributed storage systems," in Proceedings of the 9th USENIX conference on Operating systems design and implementation. USENIX Association, 2010, pp. 1–7.

[8] N. Bonvin, T. G. Papaioannou, and K. Aberer, "A self-organized, fault-tolerant and scalable replication scheme for cloud storage," in Proceedings of the 1st ACM symposium on Cloud computing, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 205–216.

[9] M. Placek and R. Buyya, "Storage exchange: a global trading platform for storage services," in Proceedings of the 12th international conference on Parallel Processing, ser. Euro-Par'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 425–436.

[10] K. Sripanidkulchai, S. Sahu, Y. Ruan, A. Shaikh, and C. Dorai, "Are clouds ready for large distributed applications?" SIGOPS Oper. Syst. Rev., vol. 44, no. 2, pp. 18–23, Apr. 2010.

[11] Z. Wu, M. Butkiewicz, D. Perkins, E. Katz-Bassett, H. V. Madhyastha, "SPANStore: Cost-Effective Geo-Replicated Storage Spanning Multiple Cloud Services", SOSP, 2013.

[12] D. Bermbach, M. Klems, S. Tai, and M. Menzel, Metastorage: A federated cloud storage system to manage consistency-latency tradeoffs, in IEEE International Conference on Cloud Computing (CLOUD), 2011, pp.452-459.

[13] M. R. Garey, and D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness. New York, NY, USA: W.H. Freeman, 1979.

[14] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, " Above the clouds: A berkeley view of cloud computing," Comm. of the ACM, vol. 53, no. 4, pp. 50–58, 2010.