Distribution Data Across Multiple Cloud Storage using Reinforcement Learning Method

Abdullah Algarni and Daniel Kudenko
Department of Computer Science, University of York, Deramore Lane, Heslington, YO10 5GH, York, U.K.

Keywords: Machine Learning, Reinforcement Learning, Multiple Cloud Computing Storage, File Access Pattern.

Abstract: Storing data on a single cloud storage service may cause several potential problems for the data owner such as service continuity, availability, performance, security, and the risk of vendor lock-in. A promising solution to tackle some of these issues is to distribute the data across multiple cloud storage services (MCSS). However, the distinguishing characteristics of different cloud providers, in terms of pricing schemes and service performance, make it difficult to optimise the cost and the performance concurrently on MCSS. This paper proposes a framework for automatically tuning the data distribution policies across MCSS from the client side based on file access patterns. The aim of this work is to optimise the average cost and the average service performance (mainly latency time) on MCSS. To achieve this goal, two different machine learning algorithms are used in this work: (1) supervised learning to predict file access patterns, and (2) reinforcement learning to control data distribution parameters based on the prediction of file access pattern. The framework was tested on a cloud storage emulator, where its was set to act like several common cloud storage services. The result of testing this framework shows a significant improvement in the cost and performance of storing data in multiple clouds, as compared to the commonly used uniform file distribution.

1 INTRODUCTION

Cloud storage allows users to store data on a remote storage, where it can be accessed through the Internet. It offers many advantages for users such as increased work efficiency and reduced operational costs in the long-run. One of the major benefits of cloud storage is the scalability and elasticity, allowing organisations to improve the management of growth of storage capacity demand in their physical storage, which are increasing dramatically over time (Rebello, 2012). Despite these advantages, there are some underlying concerns about cloud storage:

- **Cloud performance:** This usually refers to network latency, which is the time a packet takes to travel from the client-side until it is completely stored on the cloud provider's servers. Typically, the latency is influenced by the distance between locations of client and cloud service, besides network throughput (Solomon et al., 2014).
- **Cloud vendor lock-in:** Every cloud provider has specific API (Application Programming Interface) requirements that allow users' applications to interact with their services. This means that cloud users must build their applications based on these requirements. The divergence between API requirements poses the risk of locking a users' data to a particular cloud provider (Mu et al., 2012). Several unified APIs have been implemented for interacting with many of the popular cloud service providers such as Libcloud (Libclouds, 2016) and Jcloud (Jclouds, 2016), which allow users to move from one cloud to another without changing their applications and also allow them to adopt multiple cloud services.
- **Service continuity:** cloud services may suffer outages or even go out of business at any time (see (Marshall, 2013), (Brinkmann, 2016), (Armbrust et al., 2010), (Tsidulko, 2015), (Bort, 2016) for prominent examples).

Some researchers have addressed the above issues and suggested to use the RAID principle (Redundant Array of Independent Disks) to distribute data across multiple cloud storage instead of relying on a single one. Typically, RAID has been used in traditional storage to avoid problems with a single hard-disk by

Algarni A. and Kudenko D.
Distribution Data Across Multiple Cloud Storage using Reinforcement Learning Method.
DOI: 10.5220/0006124804310438
In *Proceedings of the 9th International Conference on Agents and Artificial Intelligence (ICAART 2017)*, pages 431-438
ISBN: 978-989-758-220-2

distributing data on multiple hard disks. The main objectives of using RAID: (1) improve the performance of reading from and writing onto hard-disks and (2) provide some level of fault tolerance in case one of the hard disks fails. However, the case with multiple cloud storage is different, because each cloud service has different characteristics that affect the connectivity between client and cloud services. Furthermore, the file access patterns along with file size play a key role in the cost of cloud services. More specifically, the cost of cloud storage is computed mainly from the number of operations (typically: read, write, and delete), network usage, and the lifetime and size of files. Moreover, the file size directly affects the latency time.

Optimising cost and latency on different cloud services are made difficult because of the differences in service performance and pricing schemes of each cloud storage provider. Besides, the prices scheme and performance in any cloud provider are not stationary, i.e. the cost may change automatically based on how much data is stored or transferred through the network. Thus, the optimisation should ideally account for long-term cost and performance, rather than just optimising the current state. Accordingly, it is important to find a dynamic solution that is capable of optimising cost and latency time and adapt to changes in the states of the different cloud storage services.

This paper demonstrates that reinforcement learning is a suitable technique to deal with these challenges because of its ability to maximise the expected long-term utilities. Furthermore, we show how supervised learning can be used to generate predictive models of file access pattern within large enterprise workflows.

The overall contribution of this paper is an intelligent framework for optimising the cost and latency on MCSS. The framework has a novel design to overcome the problem of service continuity, availability and vendor lock-in. The development of this framework combines two machine learning methodologies: (1) a new method of reinforcement learning to determine the percentage of file data that should be located in each cloud storage based on file access patterns and (2) supervised learning to predict these access patterns for each file.

2 CLOUD COMPUTING OVERVIEW

Cloud Computing has become a significant business trend recently. In its broadest sense cloud computing represents computing resources delivered as services over the Internet. One of these services is *cloud storage service*, which is getting much attention as a result of the increasing of data capacity volume every year in organisation data centres.

Broadly, there are two categories of ”*on-line*” storage, both often referred to as cloud storage. The first category is designed to be a personal on-line storage for private consumers. This category usually offers limited free storage and a fixed price per month for any extra space. Examples of this kind of storage including GoogleDrive and OneDirve. The second category is designed to work as utility services, mainly for enterprises. This type is called Cloud Storage Service. It provides unlimited storage space and charges users via a pay-as-you-go method. This category is intended to be accessed primarily through an API. Cloud storage services are more attractive to businesses since they offer many benefits, such as flexibility, scalability, and a reduction in spending on technology infrastructure. (Furht, 2010; Thakur and Lead, 2010). Examples of this type include Google Cloud Storage , Amazon S3 , Microsoft Azure Storage, and Rackspace File Cloud. Usually, users of this type are charged based on three factors: (1) the amount of data stored, (2) the amount of network bandwidth used, and (3) the number of operations such as read, write, and delete.

This paper focus on the second type only (cloud storage services), which is most relevant to our target domain of data-intensive enterprises.

3 RELATED WORK

Several researchers have addressed the issue of depending on a single cloud storage service. An useful approach for tackling the problems associated with a dependence on a single cloud provider is adopting MCSS. Although the distribution of data among MCSS increases availability, and reduces the probability of losing data, it may increase the cost and not guarantee to enhance the performance because it is restricted to the lowest performance of cloud services.

Scalia (Papaioannou et al., 2012) presented a cloud brokerage solution that continuously adapts the placement of data, based on files access statistics among several cloud storage services to minimise the storage cost, improve the data availability, and eliminate vendor lock-in risk. However, the work does not evaluate the impact of the system on the latency time. **HAIL** (Bowers et al., 2009) used the principle of RAID to distribute files across a collection of cloud storage to enhance the availability of data and remotely manage data security risks in the cloud by

Table 1: A comparison between our framework LOMCSS and other work, where S.Cost is Storage Cost, N.Cost is Network Cost (i.e Transaction cost), O.Cost is Operation Cost (read, write, and delete).

Framework	**the goal is to optimise**				**Tackling problems of**		**# clouds**
	S.Cost	N.Cost	O.Cost	Latency time	Vendor lock-in	Availability	Multiple
HAIL	✓	x	x	x	✓	✓	✓
Scalia	✓	x	x	x	✓	✓	✓
MCDB	x	x	x	x	✓	✓	✓
μLibCloud	x	x	x	✓	✓	✓	✓
DepSky	x	x	x	✓	✓	✓	✓
Deco	✓	x	x	✓	x	x	x
LOMCSS	✓	✓	✓	✓	✓	✓	✓

employing the Proofs of Retrievability (PORs) system. Although this work shows a reduction of storage cost, they do not consider the effect of access patterns on the network cost. Besides, they assume that the file is static, such as backup files and archives. **MCDB** (AlZain et al., 2011) is a multi-cloud database module, which employs Shamers secret sharing algorithm (Shamir, 1979) to reduce the data integrity, data intrusion, and service availability. (Mu et al., 2012) have designed a prototype system called **μLibCloud** that leverages RAID to improve the availability, read and write performance, global access experience of clouds, along with fault tolerance of cloud storage services. However, their work is subject to reasonable extra costs. In other research, (Bessani et al., 2011) presented a system called **DepSky** that employs a cryptographic secret sharing scheme with erasure codes to avoid vendor lock-in and enhance the availability and efficiency of distributed data. Although DepSky shows an improvement of performance, the cost doubles on average compared to a single cloud storage.All above mentioned solutions do not consider the differences in cloud services network characteristics that have an impact on the connectivity between client and cloud storage services. Each cloud storage possesses a different network architecture and access policies that affect the performance (latency time) of reading and writing the data. Moreover, no one, to the best of our knowledge, has proposed a solution to optimise the cost and performance together in the distribution of files across MCSS. However, it is worth mentioning here that there is one work by (Zhou et al., 2015) who proposed a system called "**Deco**" to optimise cost while keeping performance at reasonable levels . But, they designed their system for distributing processes tasks across multiple instances (not storage) in a single cloud for workload purpose. Besides, they did not take account of network and operational cost in their system. Furthermore, the efficiency of their solution will be restricted by the slowest cloud service used, if they implement it on MCSS. In this paper, we propose a framework (we call it Learning to Optimise Multiple Cloud Storage Services, for short LOMCSS) to optimise the performance of multiple cloud storage, while keeping the overall cloud storage cost low. Table (1) summarises the differences between the various solutions mentioned above compared to LOMCSS.

4 FRAMEWORK ARCHITECTURE

Figure (1) demonstrates an overview of the proposed framework. It comprises two machine learning methodologies: (1) Reinforcement Learning that decides how to distribute files across multi-clouds based on the variance between cloud storage services and the prediction of file access patterns; (2) supervised learning that predicts the access pattern for each file, which we call it "Access Pattern Prediction Model" (APPM). Before discussing these algorithms, we will first give a brief description of the RAID technology.

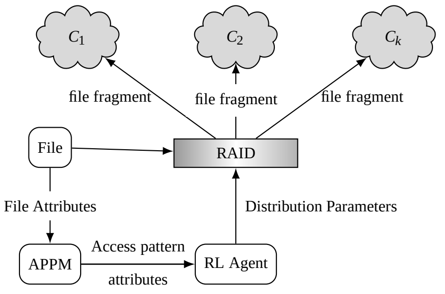

Figure 1: A high level of the Distribution Framework Structure, where APPM is estimating file access pattern; the RL decide how much proportion of each file size will be located in each cloud; RAID is the distribution manager; C_1,C_2,and C_K are cloud storages.

4.1 RAID Controller

RAID stands for Redundant Array of Independent Disks, which is a technology that combines several hard disks into a single logical drive. It was developed to overcome the performance and availability issues in traditional data centre hard disks. There are several ways of using RAID to spread data across multiple hard drives, which have been standardised into various levels (Buyya et al., 2002). In this work, we use the principle of RAID level 5 because it is the most common RAID configuration for business servers (LYNN, 2014) and it offers appropriate performance and fault-tolerance.

Although employing RAID for MCSS can improve the reliability, it is hard to optimise overall performance while reading and writing to different cloud storage services concurrently. Many factors impact the performance such as the distance, the cloud network structure and policies, and throughput. Consequently, the overall performance is restricted by the slower cloud services. In this work, we address this challenge.

4.2 Access Pattern Predictive Model

The goal of this module is to predict the lifetime of the file (lifespan) and how many times a file will be read and updated in the future. Prediction is performed using numeric linear regression , which expresses the inputs as a linear combination of attributes with predetermined parameters (usually called weights). As the name implies , this technique use regression analysis to optimise the parameters over training.

In LOMCSS, file attributes are used as an input vector for three linear regression models, each of which predicts one of the following access pattern characteristics: (1) Lifetime which indicates to how many days the file will be active.; (2) Number of read operations which shows how ; (3) Number of write operations which is the number of times the file will be updated during its lifetime. For our experiments, we trained the algorithm on a file access log, based on the business workflow of a large organisation. The training set is composed of six input patterns and three desired outputs.

4.3 Reinforcement Learning Agent

Reinforcement Learning (RL) is a machine learning method that is used to tackle sequential decision-making problems through a trial-and-error technique (Sutton and Barto, 1998). In the broadest sense, RL

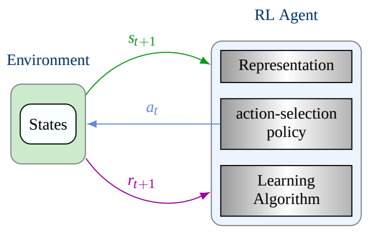

Figure 2: The reinforcement learning paradigm consists of an agent interacting with an environment. At each discrete time t, the agent observes s_t and performs a_t in order to transition to state subsequent s_{t+1}, and receives a reward r(t+1).

agent interacts with a single environment by observing the state of its environment, selecting an action, and receiving a scalar reward for that action, as depicted in figure 2. The reward from the environment provides an indication of the utility of the action in a given state. More specifically, the environment is characterised by a set of states S in which every state is constructed from a vector of state features, and the agent decides which action(s) to be performed from a set of admissible actions A based on an *action-selection policy*. Over time, an RL agent can learn how to behave in its environment through the search for the optimal policy (denoted π^*) of choosing an action in a respective state. Many methodologies have been used to learn an optimal policy in RL problems. These methodologies can be categorised into three groups (Konda and Tsitsiklis, 2003): actor-only, critic-only , and actor-critic methods. *Critic* refers the state-value function, where *actor* refers to the policy structure. Most of the RL applications use methods that estimate the value function (critic-only), which determines the value of the agent to be in a given state $V : S \to \mathbb{R}$ or the value of performing an action in a particular state $Q : S \times A \to \mathbb{R}$, which called Q-value. Broadly, the value of each state is computed as the sum of discounted rewards accumulated when starting from state s, acting under a policy π for a horizon of t time steps, which takes the following form:

$$Q^{\pi}(s_t, a_t) = E_{\pi}\left[\sum_{t=0}^{\infty} \gamma r_t | s_t = s, a_t = a\right] \quad (1)$$

Where $0 \leq \gamma < 1$ is a factor used to discount the future reward, and E_{π} is the expectation assuming the RL agent follows policy π. The goal of the agent is to learn the optimal policy π^* that maximises the cumulative rewards starting a given state s.

The best way to represent the Q-value is via a parametrised function approximation such as a *linear regression* and *neural network*. In this work we use neural network with back-propagation that to optimise the network parameters with *Temporal Difference error* (TD error) function that evaluate the Q-value function, which takes the below form:

$$TD = \left[r_t + \gamma Q(s_{t+1}, a_{t+1})\right] - Q(s_t, a_t) \qquad (2)$$

Despite the wide use of the critic-only method, employing this method in a continuous state and action spaces can be non-trivial and time-consuming (Hasselt, 2012). In such case, actor-critics methods have been demonstrated to be more successful (Hasselt, 2012). Actor-critics methods rely on two functions: (1) the actor function which is used to implement a stochastic policy that maps states to actions, and (2) the critic or the value function as described above. This method , also, uses TD-error function (2) to evaluation and update critic value and adjusting the action probabilities.

Recall that the action of the RL agent, in this work, is to decide on the proportion of each file to be stored in each cloud storage service to optimise the cost and latency time. This implies that the RL agent must execute multiple continuous actions simultaneously, where each action is dependent on other action values since they all have to sum up to 100 percent. Moreover, in this work, the RL agent interacts with multiple independent environments (the cloud storage providers) and receives various reward values, which means we have multiple states values at each time step t.

The reward function incorporates two factors: (1) latency time and (2) total cost of each cloud service including storage, network, and operation cost. These factors are subject to change at any time which means that the reward is *non-stationary*. With all these challenges, we developed a new algorithm specially designed for this problem.

The new algorithm is an on-policy algorithm because it is based on the state value function that is evaluated based on the TD-error. Unlike critic-only algorithms, it does not have a discrete action space. Also, in contrast to actor-critic algorithms, it does not have an independent actor function. However, actions are generated directly from the state value function. This algorithm can be thought of as a value and actions approximation based on a neural network. More specifically, the neural network produces a number of output values equal to the number of cloud services and then transforms these value to actions, i.e. a list of percentage values corresponding to the file proportion to be stored on each cloud service provider respectively. Also, unlike other applications of neural networks in RL, in this work, the number of output values in the neural network is not fixed because the number of cloud services available can change over time. This means that the number of output values can be increased or decreased at any time, which is another contribution in this work. More details about our algorithm are provided in the following section.

4.3.1 Learning to Optimise Multiple Cloud Storage Services (LOMCSS)

Our framework (LOMCSS) is based on neural networks, which is a powerful mathematical method capable of representing complex linear and non-linear functions (Whiteson, 2012). It has been used widely in machine learning applications. The basic structure of a neural network is a feed-forward network. Usually, neural network applications use the back-propagation (BP) algorithm to adjust the connection weights using the gradient descent method. In most RL applications, there are two methods to approximate $V(s)$ or $Q(s,a)$ using a neural network. The first is that the neural network takes the state vector s as input, and output a single number that represents $V(s)$. The second method is to produce multiple outputs, where each output is interpreted as $Q(s,a)$ denoting the value of action a in the given state s. Then the RL agent chooses a single action that satisfies the policy $\pi(s,a) = \arg max(Q(s,a))$. Afterwards, the agent computes the TD-error for each output value separately and applies back-propagation to update the weights, ignoring the rest of the output nodes. This method requires a fixed number of output nodes, which does not fulfil our requirement in this work. For this reason, we design a new method for the neural network that fulfills our target, as illustrated in figure 3. The goal of this new design is to allow an RL agent to interact with each cloud independently and produce multiple output values, where each one of them represents the value of the state in the different environment. For example, suppose we have three cloud storages, each of which has its state space S. The neural network takes all the state features from APPM as the input vector and produces three output values, each one corresponding to a single cloud. Each one of these values can be interpreted as $V(s)$ for the state of the file on that cloud. As shown in figure 3, each output value will be transformed into a single numerical action, considering the value of other actions, as shown in Algorithm **??** (line #8). After executing all actions, the RL agent receives a reward from each environment, which is used to compute the multiple TD-error functions. Each of these errors is used to evaluate the state value for each cloud within the BP algorithm. However, it is worth mentioning that the cost is an accumulative function based on usage rate, which means it rises over time within the billing period. Thus, optimising the cost of cloud storage after each action is tricky. On top of that, as aforementioned, the reward function is non-stationary. Thus,

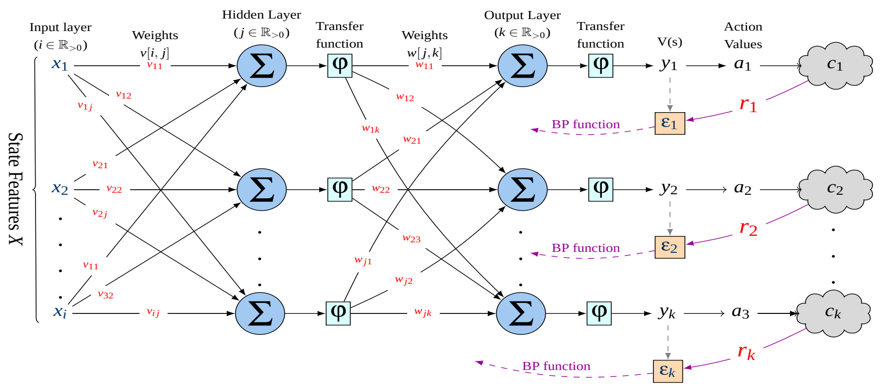
Input layer ($i \in \mathbb{R}_{>0}$)
Weights $v[i,j]$
Hidden Layer ($j \in \mathbb{R}_{>0}$)
Transfer function
Weights $w[j,k]$
Output Layer ($k \in \mathbb{R}_{>0}$)
Transfer function
V(s)
Action Values
State Features X
BP function

Figure 3: At each discrete time t, the agent receives file access pattern arbitrates (from APPM) that represent the state features X, which will be passed through neural network to produce different outputs. Each output will be transformed to proportion of the file size that will be stored in each cloud a_k. Afterwards, the agent receives rewards from each cloud r_k, which will be used by the TD-error (TD_K). Then, the back-propagation function (BP) updates the connection weights.

we compute the cost with respect to the amount used of data as expressed below:

$$cost = \sum_{i=0}^{n} \left(\frac{sc_i}{su_i} + \frac{nc_i}{nu_i} + \frac{otc_i}{otu_i}\right) \quad (3)$$

Where sc is the cost of storing data (su) into cloud i per US dollar, su is the amount of data stored in cloud i per GB, nc is the network usage cost, nu the amount of transferring data to cloud, otc is the cost of performing a number of operations (otu) on cloud provider i. n denotes to the number of cloud providers available. In short, by given the network the predicted attributes of each file (as discussed in section 4.2) along with the file size, the network produces multiple output values corresponding to the number of cloud storage services available at a time t. The output values represent the value of storing the file on each cloud. Based on that, the RL agent decides how much proportions of the file size is going to be sent to each cloud. Afterwards, the decisions will be evaluated and tuned using TD-error and back-propagation. One of the strengths of the proposed framework is that the number of outputs of the network is flexible, corresponding to the number of cloud storage providers available at each time step. This feature gives the framework the ability to overcome the issue of service availability and continuity. For example, if one of the cloud providers is not available for any reason, or the data owner has added new cloud storage, the number of output nodes can be changed accordingly, without the need to reset the agent.

5 EXPERIMENT SET-UP

In this section, we show how the proposed framework was evaluated. First of all, we have designed and implemented a cloud storage emulator, especially for this work to emulate performance and costs. The emulator is flexible and capable of emulating any number of cloud storages, simultaneously. The performances of the providers in this emulator have been set up to act as a number of cloud providers: Google Cloud storage services, Amazon S3, Microsoft Azure Storage, and Rackspace Cloud File. The performance of these providers has been measured using the *performance analysis services* of "cloudharmony.com". Then, APPM has been trained based on an access log file for several department systems of a large organisation (SFD, www.sfd.gov.sa), which has been generated based on the file workflow of the systems for vacations, payroll, financial reporting, and budget. Recall, that the goal of APPM is to predict access pattern attributes for each file which includes expected lifetime, the number of read operations, and a number of write operations. After the APMM has trained, we used the following statistical methods to evaluate the overall quality of the APMM(1) Correlation coefficient (denoted as r), and (2) Root Mean Squared Error (as *RMSE*).(3) Mean Absolute Error (denoted as *MAE*).

The settings for the RL agent and the neural network are shown in table 2. Most of these settings have been chosen based on (Gatti, 2015). Finally, the whole framework has been evaluated by distributing 2874 files with a total size 261 GB.

In order to test the flexibility of the framework (i.e. how the RL agent adapts with a dynamically changing of number of cloud providers), we defined three sce-

Table 2: Neural network and RL parameters sittings.

parameters	values
# input nodes	4
# hidden nodes	16
# output nodes	= # cloud storages
Transfer function	$\frac{1}{1+e^{-x}}$
β	0.0007
η	0.5
α	0.001
λ	0.7
γ	0.75

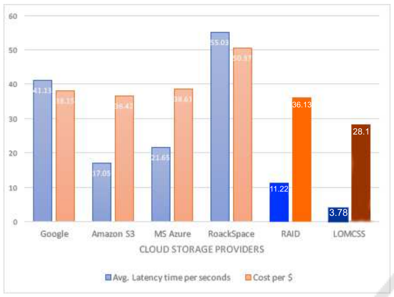

Figure 4: comparison between the cost and latency time in each cloud individually and between the Standard RAID.

narios: **Scenario #1:** the number of cloud providers are fixed and does not change during the learning process. **Scenario #2:** one of the cloud providers is removed out of the cloud storage array in the middle of the learning process. **Scenario #3:** a new cloud provider is added to the storage array in the midst of the learning process.

6 EXPERIMENT RESULTS

The results of the training data that performed to train the APPM are shown in Table 3. Also, To evaluate our approach, we first measured latency time and the total cost for each cloud provider individually by sending the whole files to each one (i.e. without distribution) . Therefore we distributed the entire data into each cloud providers individually. Then we re-distribute them using the standard principle of RAID. When using RAID, the average latency time was 11.22 seconds for all files and the total cost was $36.13. After that, we tested LOMCSS on the same data and the same cloud storage provider settings. The bar chart in figure 4 illustrates the differences in total cost and average latency times in four different clouds. In addition, the graph shows the cost and average latency of distributing files using RAID and LOMCSS.

The result indicates that, by using LOMCSS, the total cost decreased by % 22 which amounted to

Table 3: APPM, measurement evaluation.

	Lifetime	# of read	# of write
r	0.9761	0.9228	0.9889
MAE	0.9621	0.5648	0.0418
RMSE	1.3025	1.143	0.2119

$28.1. and the average latency time was significantly reduced by more than %66 (to 3.78 seconds), comparing to RAID. In fact, The RL agent learned how to optimise both latency and cost by applying different weight to cost and latency based on the importance of the file which is extracted from the file access pattern attributes. Additionally, the experiments show that the framework is flexible and adaptable when changing the number of clouds or the behaviour of a cloud. Figures 5 6 demonstrate how the agent's learning curve changes when the number of available cloud providers is reduced increased based on the scenarios that we described in the previous section.

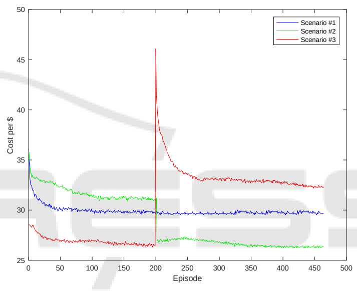

Figure 5: Cost Scenarios, cost slightly decreases when a one of cloud was removed, and raised up as one cloud was removed , then start gone down over learning.

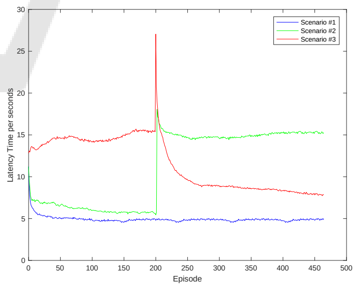

Figure 6: Latency Scenario , cost raised up as one cloud was removed , then start gone down over learning.

7 CONCLUSION

This paper presented a framework for automatically tuning distribution parameters over multi-cloud storage services to optimise long-term cost and latency time. The framework combines two machine learning methods: supervised learning for predicting the access patterns for each file and RL for tuning the distribution parameters based on the predicted access patterns. We empirically demonstrated the benefits of the framework by performing experiments on a cloud storage emulator. The main challenges that were tackled are how to interact with multiple environments, execute a non-fixed number of actions simultaneously, and deal with non-stationary multiple rewards signal. Therefore, the learning algorithm in RL has been designed in a unique way to satisfy the goal of this work. The empirical evaluation showed that the proposed framework is capable to significantly reduce both the cost and average latency time in MCSS.

REFERENCES

AlZain, M., Soh, B., and Pardede, E. (2011). Mcdb: Using multi-clouds to ensure security in cloud computing. In *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*, pages 784–791. IEEE.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. (2010). A view of cloud computing. *Commun. ACM*, 53:50–58.

Bessani, A., Correia, M., Quaresma, B., Andr, F., and Sousa, P. (2011). Depsky: Dependable and secure storage in a cloud-of-clouds. In *Proceedings of the Sixth Conference on Computer Systems*, EuroSys ’11, pages 31–46, New York, NY, USA. ACM.

Bort, J. (2016). Google apologizes for cloud outage that one person describes as a comedy of errors.

Bowers, K. D., Juels, A., and Oprea, A. (2009). Hail: A high-availability and integrity layer for cloud storage. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, pages 187–198, New York, NY, USA. ACM.

Brinkmann, M. (2016). Copy cloud storage services life ends on may 1, 2016. http://www.ghacks.net/2016/02/02/copy-cloud-storage-services-life-ends-on-may-1-201. Online; accessed 19-10-2016.

Buyya, R., Cortes, T., and Jin, H. (2002). *A Case for Redundant Arrays of Inexpensive Disks (RAID)*, pages 2–14. Wiley-IEEE Press.

Furht, B. (2010). *Cloud Computing Fundamentals*, chapter 1. Springer.

Gatti, C. (2015). *Design of Experiments for Reinforcement Learning*. Springer, Heidelberg New York Dordrecht London.

Hasselt, H. (2012). *Reinforcement Learning in Continuous State And action Spaces*, pages 207–251. Springer, Heidelberg New York Dordrecht London.

Jclouds, A. (2016). The java multi-cloud toolkit. http://jclouds.apache.org. Online; accessed 08-07-2014.

Konda, V. R. and Tsitsiklis, J. N. (2003). On actor-critic algorithms. *SIAM J. Control Optim.*, 42(4):1143–1166.

Libclouds, A. (2016). One interface to rule them all. https://libcloud.apache.org. Online; accessed 08-07-2014.

LYNN, S. (2014). Raid levels explained.

Marshall, D. (2013). Cloud storage provider nirvanix is closing its doors. http://www.infoworld.com/article/2612299/cloud-storage/cloud- storage-provider-nirvanix-is-closing-its-doors.html. Online; accessed 19-10-2016.

Mu, S., Chen, K., Gao, P., Ye, F., Wu, Y., and Zheng, W. (2012). libcloud: Providing high available and uniform accessing to multiple cloud storages. In *Grid Computing (GRID), 2012 ACM/IEEE 13th International Conference on*, pages 201–208, Beijing. IEEE.

Papaioannou, T., Bonvin, N., and Aberer, K. (2012). Scalia: An adaptive scheme for efficient multi-cloud storage. In *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference*, pages 1–10. ACM.

Rebello, J. (2012). Subscriptions to cloud storage services to reach half-billion level this year.

Shamir, A. (1979). How to share a secret. *Commun. ACM*, 22:612–613.

Solomon, M. G., Kim, D., and Carrell, J. L. (2014). *Fundamentals Of Communications And Networking*. Jones and Bartlett Publishers, Inc., USA, 2nd edition.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*. MIT press.

Thakur, N. and Lead, Q. (2010). Performance testing in cloud. Citeseer A pragmatic approach.

Tsidulko, J. (2015). Overnight aws outage reminds world how important aws stability really is. CRN. Online; accessed 01-06-2016.

Whiteson, S. (2012). *Evolutionary Computation for Reinforcement Learning*, pages 325–355. Springer, Heidelberg New York Dordrecht London.

Zhou, A. C., He, B., Cheng, X., and Lau, C. T. (2015). A declarative optimization engine for resource provisioning of scientific workflows in iaas clouds. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC ’15, pages 223–234, New York, NY, USA. ACM.