International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2018 IJSRCSEIT | Volume 3 | Issue 4 | ISSN : 2456-3307

Multiple Cloud Service Providers with an efficient and SLO Guaranteed cloud storage

D. G. Prasad
MCA Sri Padmathi College of Computer Sciences And Technology Tiruchanoor, Andhra Pradesh, India

ABSTRACT

Due to the increase of the advantages of the cloud computing many enterprisers started adopting the cloud servers. There are so many cloud service providers in the cloud system. The enterprisers will choose the cloud platform based on their satisafaction.the satisfaction of the customer mainly depends on the two things firstly cloud configuration affects the quality of service which is an important factor affecting customer satisfaction. Secondly customer satisfaction affects the request arrival rate of a cloud service provider. it is necessary the clouds service brokers should provide multi cloud storage service to reduce their cost payments to the cloud servers. . In this paper, we propose a multi-cloud Economical and SLO-guaranteed Storage Service (ES3), which determines data allocation and resource reservation schedules with payment cost minimization and SLO guarantee. ES3 incorporates (1) a coordinated data allocation and resource reservation method, which allocates each data item to a datacenter and determines the resource reservation amount on datacenters by calculating all the pricing policies; (2) a genetic algorithm based data allocation adjustment method, to maximize the data reservation benefit in all data centers.. We also propose several algorithms to enhance the cost efficient and SLO guarantee performance of ES3 including i) dynamic request redirection, ii) grouped Gets for cost reduction, iii) lazy update for cost-efficient Puts, and iv) concurrent requests for rigid Get SLO guarantee.

Keywords: Cloud Storage, SLO, Payment Cost.

I. INTRODUCTION

Cloud providers, such as Amazon S3, Google Cloud Storage (GCS) and Windows Azure offer storage as a service. It is important for cloud providers to reduce Service Level Agreement (SLA) violations to provide high quality of service and reduce the associated penalties High data durability is usually required to meet SLAs. Durability means the data objects that an application has stored into the system are not lost due to machine failures (e.g., disk failure).

Data loss caused by machine failures typically affects data durability. Machine failures usually can be categorized into correlated machine failures and non-correlated machine failures. Correlated machine failures refer to the events in which multiple nodes (i.e., servers) fail concurrently due to the common failure causes (e.g., cluster power outages, Denial-of-Service attacks), and this type of failures often occur in large-scale storage systems Significant data loss is caused by correlated machine failures which has been documented by Yahoo LinkedIn and Facebook. Non-correlated machine failures refer to the events in which nodes fail individually (e.g., individual disk failure). Usually, non-correlated machine failures are caused by factors such as different hardware/software compositions and configurations and varying network access abilities.

To enhance data durability, data replication is commonly used in cloud storage systems. Due to

highly skewed data popularity distributions, popular data with considerably higher request frequency (referred to as hot data) [18] could generate heavy load on some nodes, which may result in data unavailability at a time. Availability means that the requested data objects will be able to be returned to users. Actually, much of the data stored in a cloud system is rarely read (commonly referred to as cold data). Replicas of cold data waste the storage resource and generate considerable storage cost and bandwidth cost (for data updates, data requests and failure recovery) that outweigh their effectiveness on enhancing data durability. Thus, it is important to compress and deduplicate unpopular data and store them in low-cost storage medium.

Random replication has been widely used in cloud storage systems Cloud storage systems, such as Hadoop Distributed File System (HDFS) [14], Google File System (GFS) and Windows Azure use random replication to replicate their data in three servers randomly selected from different racks to prevent data loss in a single cluster. However, the three-way random replication cannot well handle correlated machine failures because data loss occurs if any combination of three nodes fail simultaneously. To handle this problem, Copy set Replication and Tiered Replication have been proposed. However, both methods do not try to leverage data popularity to substantially reduce storage cost or bandwidth cost caused by replication.

To address the above issues, we aim to design a cost effective replication scheme that can achieve high data durability and availability while reducing storage cost and bandwidth cost caused by replication. To achieve our goal, we propose a popularity-aware multi-failure resilient and cost-effective replication scheme (PMCR), which has advantages over the previous proposed replication schemes. We summarize the contributions of this work below.

• PMCR replicates the first two replicas of each data chunk in primary tier, and replicates the third replica in remote backup tier. The three replicas of each data chunk are stored in one Copy set, which can handle correlated failures. As a result, PMCR can handle both correlated and independent failures.

• PMCR classifies data into hot data, warm data and cold data based on data popularity. It compresses the third replicas of warm data and cold data in the backup tier. For read-intensive data, PMCR uses the Similar Compression (SC), which lever-ages the similarities among replica chunks and removes redundant replica chunks; for write-intensive data, PMCR uses the Delta Compression (DC), which records the differences of similar data objects and between sequential data updates. As a result, PMCR significantly reduces the storage cost and bandwidth cost caused by replication without compromising data durability and availability, as well as data request delay greatly.

• To further reduce the storage and bandwidth costs caused by replication, PMCR enhances SC by eliminating the redundant chunks between different data objects (rather than only within one data object) and enhances DC by recording the differences between different data objects (rather than only the difference between sequential updates).

• We have conducted extensive trace-driven experiments to compare PMCR with other state-of-the-art replication schemes. The results show PMCR achieves high data durability, low data loss probability, storage and bandwidth cost.

II. PROPOSED SYSTEM

We propose a heuristic solution, called coordinated data allocation and reservation method It determines the data allocation first (that proactively increase the reservation benefit) and then determines the resource reservation schedule based upon the data allocation schedule. To maximize the reservation

benefit, as shown in Figure 1, ES3 can use its GA-based data allocation adjustment method to improve the data allocation schedule before determining the resource reservation schedule.

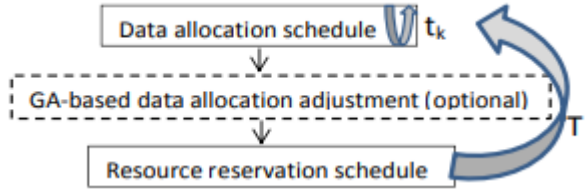

Figure 1. Sequence of scheduling.

Coordinated Data Allocation and Resource Reservation:-

First, we introduce how to find the optimal reservation amount on each datacenter that maximizes the reservation benefit given a data allocation schedule.

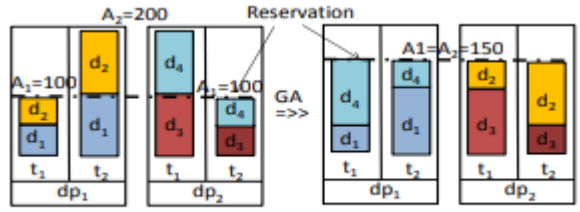

Figure 2. Unbalanced and optimal data allocation.

GA-based Data Allocation Adjustment:-

If the allocated Get/Put rates vary over time largely (i.e., the rates exceed and drop below the reserved rates frequently), then the reservation saving is small according to Equation. For example, Figure shows a data allocation schedule. Then, both R g dpj = 100 and R g dpj = 200 reduce reservation benefit at a billing period. We propose the GA-based data allocation adjustment method to make the reserved amount approximately equal to the actual usage as shown in Figure 3(b).

As shown in Figure 3, this method regards each data allocation schedule, represented by (dl ∈ D), as a genome string, where {dp1, ..., dpβ} (denoted by Gdl) is the set of datacenters that store dl . it generates the data allocation schedule with the lowest total cost (named as global optimal schedule). It also generates the data allocation schedules with the lowest Storage cost, lowest Get cost and lowest Put cost (named as local optimal schedules) by assuming all data items as Storage-, Get- and Put-intensive, respectively.

Figure 3. GA-based data allocation adjustment.

COST EFFICIENT AND SLO GUARANTEE ENHANCEMENTS:

1) Dynamic Request Redirection:

ES3 master predicts the Get load of each storage datacenter dpj at the initial time of tk (Atk), which is used to calculate the data allocation schedule. If the actual number of Gets is larger or smaller than Atk , then the schedule may not reach the goal of SLO guarantee and minimum cost. There may be a request burst due to a big event, which leads to an expensive resource usage under current request allocation among storage datacenters. Sudden request silence may lead to a waste of reserved usage. The Get operation only needs to be resolved by one of β replicas. Therefore, we can redirect the burst Gets on a datacenter that uses up its reservation to a replica in a datacenter whose reservation is underutilized in order to save cost. This redirection can also be conducted whenever a datacenter overload or failure is detected.

2) Grouped Gets for Cost Minimization:

To fetch all data objects of a webpage, many Get requests are generated; each get fetching one data object. In cloud storage, each Get has a size limitation (denoted by ug) such as the 4kB specified in Amazon DynamoDBFor a Get gi from a user, the actual number of Gets considered by the cloud provider in cost calculation is equal to dsgi /uge, where sgi is the requested data size of Get gi . That is, if the Get size is larger than the size limitation, the

Get is considered as multiple Gets by the cloud provider when deciding the charging amount.

Algorithm 2: Coefficient-based data grouping algorithm.

Input: List L with all data objects in an aggregated data item
Output: List L' with all grouped data objects
1 Sort data objects in list L in descending order of their levels in the dependency tree of the data item;
2 **for** *each* o_i *in list* L **do**
3 Find $o_j \in L'$ with the largest grouping benefit with o_i, B_{o_i,o_j};
4 **if** $B_{o_i,o_j} > 0$ **then**
5 o_i is grouped into o_j;
6 **else**
7 o_i is inserted into list L';

The detailed procedure of coefficient-based data grouping method is shown in Algorithm 2. To group data objects, we sort all single data objects in the descending order of their levels in the dependency tree into a list L (Line 1). We loop all data objects to combine each object into an existing grouped data object or form an individual grouped data object (Lines 2-7). For each data object oi ∈ L (Line 2), we loop each of all data objects inside another list L 0 , which initially is empty, and calculate the grouping benefit. For the data object oj with the largest grouping benefit Boi,oj (Line 3), if Boi,oj > 0, we group oi into grouped data object oj (Lines 4-5); otherwise, we directly insert oi into L 0 as a grouped data object with a single object (Lines 6-7). After looping all data objects inside L, L 0 includes the grouped data objects that can save Get cost. For newly added data objects, we first insert them into L and insert all current grouped data objects into L 0 , and then each new data object is grouped into an existing grouped data object or form a new grouped data object according to the procedure in Lines 2-7. This algorithm is conducted before the real data allocation conduction, so that the objects in a grouped data object are stored as a file unit for Get/Puts.

3) Lazy Update for Cost-Efficient Puts:-

a) Put Aggregation:-

Eventual consistency means that if no new updates are made to a given data item, eventually all accesses to that data item will return the last updated value. Each Put of a data item needs to be propagated to all of its replicas for consistency maintenance. We notice that for eventual consistency, the propagation of updates on rarely used replicas can be postponed, which can be leveraged to save Put cost. For example, adding an advertisement to a webpage only needs eventual consistency and it does not need an instant update. Similar to reading a grouped data object, we can aggregate sequential writes into one Put to propagate to all rarely used replicas. Recall that for data item dl of customer datacenter dci , a storage datacenter dpj storing dl always serves Gets from dci targeting dl and β replicas of dl are stored in other storage datacenters for data availability. We call dl in datacenter dpj the master replica of data dl for customer datacenter dci and call other replicas slave replicas of data dl for dci .

b) Replica Deactivation:-

Recall that slave replicas of customer datacenter dci usually do not serve its Get requests and they are created mainly to increase the data availability. The slave replicas introduce Storage cost and Put cost. Only when the Get workload from dci is high and the storage datacenters of master replicas cannot provide SLO-guaranteed service, the Get requests will be forwarded to datacenters hosting the slave replicas of the requested data. Therefore, when the request rate of Gets from dci towards data item dl drops below a threshold Tr (i.e., when the slave replicas are unlikely to be used), in order to save Put cost on the slave replicas, we can deactivate the slave replicas of dl from storage datacenters. When the request rate of Gets towards dl from dci increases beyond Tr, the slave replicas are dynamically created by transferring the updated replicas of dl from the datacenters containing them.

4) Concurrent Requests for Rigid Get SLO Guarantee:-

Within each billing period, the data allocation of a customer is stable. However, the customer may

require a more rigid Get SLO (low tail latency SLO) during this billing period with a smaller (dl) or L g dl . If the Get SLO of dl is too rigid for the storage datacenter of the main replica to handle, we can concurrently submit multiple Get requests to different replicas including the master and slave replicas. This way, although some of the datacenters cannot supply a Get SLO guarantee service, there can be a datacenter among them responding the request with the rigid SLO guarantee. Though this method introduces additional Get cost due to more Get requests, it avoids the need to conduct data reallocation again, so that it saves the replica Transfer cost and does not waste the reserved Get/Put cost for datacenters currently storing dl .

Intuitively, if we transmit a Get request targeting data dl to all β datacenters with its replica, we can get low response latency with a high probability. However, it may introduce unnecessary Get costs, since a combination of part of the datacenters may already supply a Get SLO guaranteed service. The problem to find such a combination with Get SLO guarantee and Get cost minimization can be easily reduced to the knapsack problem, which is NP-hard [16]. Since β is usually small, we can enumerate all combinations that satisfy the rigid Get SLO and find the one with the minimum cost. To efficiently find the combination, we introduce a greedy heuristic algorithm. Unlike the master replicas of dl , the Gets towards its slave replicas are not considered in deciding the Get reservation of their datacenters. Then, the Get cost is calculated based on the pay-as-you-go policy Thus, to minimize the additional Get cost introduced by concurrent requests, we sort all slave replica datacenters of dl in ascending order of the Get unit cost. Then, we sequentially check each slave replica datacenter dpj in the list. If the additional Get workload on dpj does not make its total Get workload exceed its Get capacity, we add dpj into the combination C. This process continues until the combination can satisfy the rigid Get SLO guarantee, that is,

$$\prod_{dp_j \in \mathbb{C}} (1 - F^g_{dc_i,dp_j}(L^g(d_l))) \leq \epsilon^g(d_l). \quad (6)$$

Therefore, the probability that all storage datacenters cannot respond a Get within the deadline, $\prod_{dp_j \in \mathbb{C}}(1 - F^g_{dc_i,dp_j}(L^g(d_l)))$, is no larger than the allowed percentage $\epsilon^g(d_l)$, so that the rigid SLO is satisfied.

:

III. CONCLUSION

In this paper, we tend to propose a multi-cloud Economical and SLO-guaranteed cloud Storage Service (ES3) for a cloud broker over multiple CSPs that gives SLO guarantee and value reduction even beneath the Get rate variation. ES3ismore advantageous than previous strategies therein it fully utilizes completely different rating policies and considers request rate variance in minimizing the payment value. ES3 incorporates a knowledge allocation and reservation technique and a GA-based knowledge allocation adjustment technique to ensure the SLO and minimize the payment value. ES3also incorporates many methods to reinforce its value economical and SLO guarantee performance. Our trace-driven experiments on a supercomputing cluster and real completely different CSPs show the superior performance of ES3in providing SLO guarantee and value minimization compared with previous systems. The Transfer value incorporates a layer rating model and becomes additional complex, and CSP give different unit costs from a supply storage datacenter to alternative datacenters happiness to CSPs or at different locations. In our future work, we'll study the price reduction down side of transferring replicas of knowledge things to completely different storage data centers whenever a replacement knowledge allocation schedule is generated.

IV. REFERENCES

[1]. J. Howard "Scale and performance in a distributed file system" ACM Trans. Computer Systems, 1988.

[2]. P. Hunt, M. Konar, F. Junqueira, and B. Reed. "Zookeeper: Waitfree coordination for internet-scale services," In USENIX ATC, 2010.

[3]. A. Bessani, E. P. Alchieri, M. Correia, and J. S. Fraga "DepSpace:A Byzantine fault-tolerant coordination service," in EuroSys, 2008.

[4]. StorSimple. StorSimple. http://www.storsimple.com/.

[5]. TwinStrata. TwinStrata. http://www.twinstrata.com/.

[6]. Alysson Bessani, Miguel Correia, Bruno Quaresma, Fernando Andre, and Paulo Sousa "DEPSKY: Dependable and Secure Storage in a Cloud-of-clouds," EuroSys 11-April- 2011.

[7]. Ricardo Mendes, Tiago Oliveira, Vinicius Cogo, Alysson Bessani "The CHARON file system,"

[8]. Idilio Drago, Marco Mellia, Maurizio M. Munafò, Anna Sperotto and Aiko Pras "Inside Drop box: Understanding Personal Cloud Storage Services," in Proceeding of IMC -12 of ACM conference on internet measurement conference,2012 PP.481-494.

[9]. http://www.techrepublic.com/blog/five-apps/keep-your-data-safewith-one-of-these-five-cloud-backup-tools/

[10]. http://www.cloudwards.net/spideroak-or-wuala-which-is-moresecure/

[11]. Kailas Pophale, Priyanka Patil, Rahul Shelake, Swapnil Sapkal "Seed Block Algorithm: Remote Smart Data- Backup Technique for Cloud Computing," International Journal of Advanced Research in Computer and Communication Engineering, Vol. 4, Issue 3, March 2015.

[12]. Lili Sun, Jianwei An, Yang, and Ming Zeng "Recovery Strategies for Service Composition in Dynamic Network," International Conference on Cloud and Service Computing, 2011

[13]. Giuseppe Pirro, Paolo Trunfio, Domenico Talia, Paolo Missier and Carole Goble "ERGOT: A Semantic-based System for Service Discovery in Distributed Infrastructures," 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, 2010.

[14]. Xi Zhou, Junshuai Shi, YingxiaoXu, Yinsheng Li and Weiwei Sun "A backup restoration algorithm of service composition in MANETs," Communication Technology ICCT 11th IEEE International Conference, 2008, pp. 588-591.

[15]. Ms.KrutiSharma, and Prof K.R. Singh "Online data Backup and Disaster Recovery techniques in cloud computing: A review", JEIT, Vol.2, Issue 5, 2012.

[16]. Eleni Palkopoulouy, Dominic A. Schupke, Thomas Bauscherty "Recovery Time Analysis for the Shared Backup Router Resources (SBRR) Architecture", IEEE ICC, 2011.

[17]. http://searchcloudstorage.techtarget.com/definition/cloud-storage.

[18]. M. Rosenblum and J. K. Ousterhout."The Design and Implementation of a Log-Structured File System," ACM Transactions on Computer Systems (TOCS), 1992.

[19]. Alysson Bessani, Ricardo Mendes, Tiago Oliveira, Nuno Neves, Miguel Correia, Marcelo Pasin, and Paulo Verissimo "SCFS: A Shared Cloud-backed File System, " in Proceedings of the USENIX ATC on USENIX Annual Technical Conference 19&20-June -2014.

[20]. P. Mahajan, S. Setty, S. Lee, A. Clement, L. Alvisi, M. Dahlin, and M. Walfish "Depot: Cloud Storage with Minimal Trust," In Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI), Oct. 2010.