CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2016; **28**:4485–4506
Published online 12 February 2016 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.3795

A cost-efficient resource provisioning algorithm for DHT-based cloud storage systems‡

Jingya Zhou*,†, Jianxi Fan and Juncheng Jia

School of Computer Science and Technology, Soochow University, Suzhou 215006, China

SUMMARY

Personal cloud storage provides users with convenient data access services. Service providers build distributed storage systems by utilizing cloud resources with distributed hash table (DHT), so as to enhance system scalability. Efficient resource provisioning could not only guarantee service performance, but help providers to save cost. However, the interactions among servers in a DHT-based cloud storage system depend on the routing process, which makes its execution logic more complicated than traditional multi-tier applications. In addition, production data centers often comprise heterogeneous machines with different capacities. Few studies have fully considered the heterogeneity of cloud resources, which brings new challenges to resource provisioning. To address these challenges, this paper presents a novel resource provisioning model for service providers. The model utilizes queuing network for analysis of both service performance and cost estimation. Then, the problem is defined as a cost optimization with performance constraints. We propose a cost-efficient algorithm to decompose the original problem into a sub-optimization one. Furthermore, we implement a prototype system on top of an infrastructure platform built with OpenStack. It has been deployed in our campus network. Based on real-world traces collected from our system and Dropbox, we validate the efficiency of our proposed algorithms by extensive experiments. Copyright © 2016 John Wiley & Sons, Ltd.

Received 2 June 2015; Revised 10 January 2016; Accepted 16 January 2016

KEY WORDS: cloud storage; data access services; resource provisioning; queuing network

1. INTRODUCTION

Personal cloud storage provides users with many attractive functions, such as data storage, data sharing, data access, and management. It utilizes cloud technologies to build storage systems based on IT resources located in data centers. With the popularity of cloud computing, cloud storage services have gained more and more attention in both academia and industry [1–4]. According to an IHS report, personal cloud storage subscriptions have reached 500 million in 2012 for major providers such as Dropbox and iCloud [5]. The cloud storage market is estimated to grow at a rate of 33.1% and reach $56.57bn by 2019 [6]. One of the most attractive features of personal cloud storage is the ability to provide users with convenient data access services without worrying about data loss. When users use services, they are mainly concerned about data availability and response delay. The former represents the probability that users can successfully access the target data, and the latter refers to the time required for the system to respond to requests. Both of them directly affect the service level that users experienced and become the preferred performance metrics discussed in this paper.

In general, cloud storage providers build their storage systems based on the infrastructure rented from infrastructure as a service providers. For example, Dropbox chooses IT resources that come

*Correspondence to: Jingya Zhou, School of Computer Science and Technology, Soochow University, Suzhou 215006, China.
‡A previous version of this paper has appeared in NPC'14 conference.
†E-mail: jy_zhou@suda.edu.cn

from Amazon as its servers to store data and deal with requests [7]. Hence, cloud storage providers have to face an important problem — how many resources are required to build the system as well as to provide satisfactory services. The objective is to minimize the resource cost as well as guarantee the agreed performance level. However, it is not easy to achieve this objective because some new observations pose great challenges to the problem. In this paper, we explore the resource provisioning problem from the cloud storage provider's point of view.

First, currently, cloud storage systems often face a high amount of concurrent access requests. More than 1 billion files are accessed in Dropbox everyday. A cloud storage system consists of two kinds of servers: *index server*, which stores meta data about users' files, including the list of files, their sizes and attributes, and pointers to where the files were stored; *storage server*, which stores file data uploaded by users. As shown in Figure 1, access requests from users' client reach the index server at first for retrieving the corresponding meta data of files and then the client interacts with storage server directly based on meta data. Single index server architecture (*e.g.*, GFS [8]) cannot support such a high amount of concurrent access requests. Distributed hash table (DHT) and Zookeeper are two well-known mechanisms for maintaining multiple servers, and both of them can be used to build multi-index server architecture. DHT focuses on the scalability of system by building on a highly flexible structure, while Zookeeper is a centralized coordination service that ensures strong consistency by following rigorous distributed algorithms. Nowadays, many cloud storage providers begin to choose DHT (*e.g.*, Dynamo [1], Cassandra [2]). DHT requires that users' requests need to be matched and forwarded among index servers after they arrive at the system. Its execution logic is more complicated than both single index server architecture and multi-tier web applications because the interactions among servers depend on the routing process. While in single index server architecture there is no direct interaction among servers, in multi-tier web applications, interactions occur sequentially layer by layer.

Second, resource heterogeneity has become very common in today's production data centers [9]. Modern data centers often comprise different types of machines from more than one generation. Those machines have heterogeneous processor architectures, clock frequencies, memories, and disks. Even if the system is running on virtual machines, we cannot neglect the fact that different types of virtual machines have varied processing capacities. As a result, cost optimization is never easily equivalent to the minimum number of servers. In particular, modern storage technology such as solid-state drives (SSD) have greatly improved I/O performance and have been used in many storage systems. Nevertheless, the price per unit capacity of SSD is much higher than that of

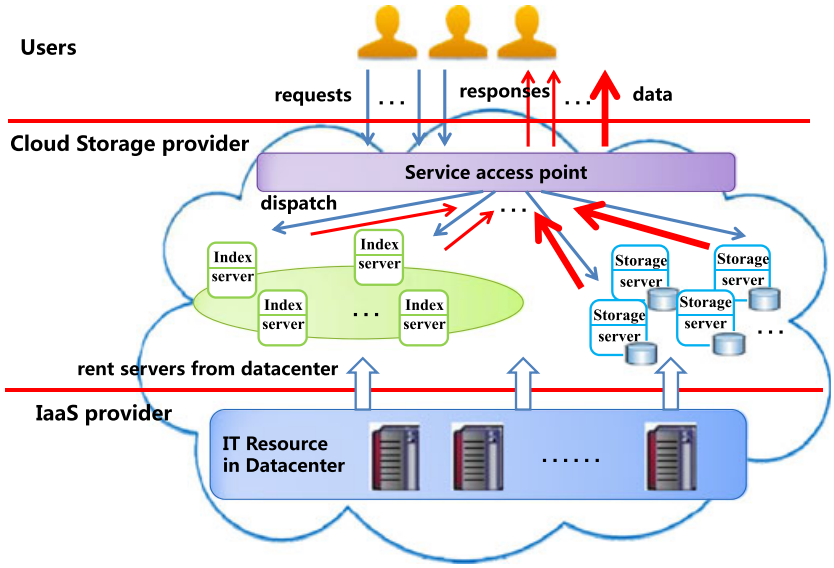

Figure 1. Cloud storage system overview.

DOI: 10.1002/cpe

hard disk drives (HDD). Hence, there exists a trade-off between performance and cost in realizing a fine-grained resource provisioning.

To address these challenges, we present a cost-efficient resource provisioning scheme. First, we propose a queuing network model that takes the heterogeneity of both the servers capacity and its pricing into account. In the model, each server is regarded as an M/G/1/k queue with its own processing capacity, where users' requests arrive randomly; the processing time of requests to be executed on servers is a distribution rather than a fixed value, and each queue has a length limit, which meet the practical facts. Different types of servers are charged varied prices. More importantly, we use this model to analyze the complex interactions by calculating the forwarded requests. It captures the specific correlation between performance metrics and the allocated resources. Second, we propose a fine-grained resource provisioning algorithm to avoid additional cost due to resource waste. Our algorithm is based on the principle of renting just enough resources to meet the performance objective. It answers not only how many index servers and storage servers are needed to rent, but also what types of servers and which type of storage media are required for index server and storage server, respectively. Third, we propose an implementation of parallel algorithm based on MapReduce paradigm to deal with the time-consuming problem of algorithm when applied in large-scale systems.

Our scheme is able to reap the benefits of fine-grained provisioning, resulting in significantly lower cost. It calculates each server's capacity by solving a nonlinear programming and chooses the proper types of server according to capacity values, so as to minimize the total cost. Trace-driven experiment results illustrate that our scheme can reduce cost by at least 40% compared with a state-of-the-art scheme without incurring significant degradation of performance.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 presents the modeling of resource provisioning, including performance analysis and problem description. In Section 4 we provide a solution and implement a parallel algorithm in Section 5. The experimental results are analyzed in Section 6. Finally, Section 7 draws on some important conclusions along with suggestions for future work.

2. RELATED WORK

Resource provisioning on demand is a promising approach for reducing cost by dynamically adjusting the number of servers to match resource demands. Xiong *et al*. [10] presented a prediction model of service performance for web applications and attempted to answer the question of how many resources are required to guarantee the performance for a given number of users. Similarly, Shi *et al*. [11] analyzed the resource utilization log by a linear predicting model and proposed a flat period reservation-reduced method to achieve better response delay. However, the model of these studies was designed by using an M/M/1 queuing system. An M/M/1 queuing system indicates that service times follow exponential distribution uniformly. It is a strong assumption that only applies to few practical scenarios. In production environments, service times on an arbitrary server should be independent and follow a general distribution. Zhu *et al*. [12] created a resource provisioning model by employing an M/G/1 queuing system, which is an extension of the M/M/1 queue, and developed meta-heuristic solutions based on the mixed tabu-search optimization algorithm to solve the provisioning problem. However, the proposed model only takes response delay into consideration and focuses on the maximization of infrastructure as a service provider's profit, which is different from the goal of this paper.

Resource demands not only depend on user requests but also have a strong correlation with specific execution logic. Zhang [13] presented a resource management algorithm for cloud storage systems. The proposed algorithm aims to achieve load balancing by using two types of operation, that is, merge operation and split operation. However, such an algorithm does not consider server interactions during the execution of services and only considers load balancing as performance metrics. Jing *et al*. [14] focused on how to minimize cost while satisfying response delay constraint for multi-tier applications. It employs a flexible hybrid queuing model that consists of one M/M/c queue and multiple M/M/1queues to determine the number of servers at each tier. Different from layer-by-layer research ideas, Lama *et al*. [15] suggested employing fuzzy theory to guide server provisioning

DOI: 10.1002/cpe

and designed a model-independent fuzzy controller, so as to minimize servers while guaranteeing end-to-end response delay. For multi-tier applications, the interactions among servers are executed layer by layer, so the execution logic is simple. For cloud storage services, users' requests should usually be matched and forwarded among many servers after they arrive at systems. The interactions among servers depend on the routing process and are not to be executed in accordance with the fixed order. Hence, the interactions that occurred in cloud storage systems are complicated and lack of an effective resource provisioning model for characterization.

The works in [14, 15] are based on the assumption that servers are identical, which is contradictory to the reality of the situation in production data centers. Furthermore, failure to consider machine heterogeneity results in coarse-grained resource provisioning and limited cost optimization. More recently, Zhang *et al*. [9] studied the heterogeneity-aware resource provisioning problem in the cloud data center. Different from our work, they dynamically adjusted the number of servers to strike a balance between energy savings and scheduling delay. Neither interactions among servers nor resource cost have been considered in their work. Our previous work [16] provided a novel resource provisioning model by characterizing the execution logic of cloud storage services. However, the resource heterogeneity has not been fully considered in [16], and this paper acts as an extension. Specifically, we updated the resource provisioning model, provided the implementation of parallel algorithm to improve efficiency, and evaluated the effectiveness of resource provisioning.

3. RESOURCE PROVISIONING MODEL

To tackle the previous problem, we have to establish a resource provisioning model. As we know, cloud storage is a carefully designed distributed storage system. A ring structure is used to describe the mapping from file name to its meta data. The ring is divided into many partitions that correspond to index servers. Each index server stores the meta data that belong to the corresponding partition and maintains a routing table at the same time. When the system receives requests, it will dispatch them to index servers firstly. To accelerate operations, the meta data is stored in memory. If requests match, index server will return results directly. Otherwise it will forward requests to another one by checking its routing table. The information contained in a routing table depends on the system scale. For a system with dozens of servers, the routing table could contain full information about all partitions (index servers) through gossiping each server's routing table. The meta data can be found within one hop in this case. However, the overhead of maintaining routing table increases with the system size. In a large-scale storage system, each index server's routing table contains information about only a small number of other partitions (index servers). As a consequence, the requests will be forwarded to the next server according to the DHT scheme and may be forwarded several hops before matching.

Index servers interact with each other through forwarding requests. To better describe this type of interaction, we propose a resource provisioning model based on queuing network. As shown in Figure 2, the system consists of N index servers and M storage servers, and each is modeled as an M/G/1/k queue with independent general execution time distribution. We use M/G/1/k queue based on the following considerations. On one hand, users' requests arrive randomly and could be

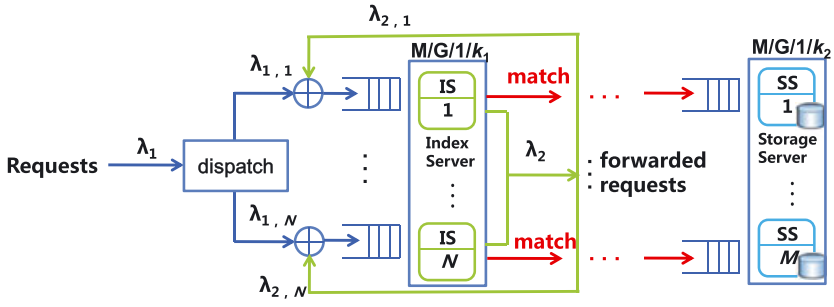

Figure 2. Queuing network model for resource provisioning.

DOI: 10.1002/cpe

described as a Poisson distribution with rate λ_1 . On the other hand, because of the limit of server capacity, the server cannot simultaneously receive and handle an unlimited number of requests. As requests increase, the length of queue becomes larger, which results in a higher response delay or even blocks servers. To avoid server overload, we set up size limits (k_1, k_2) for the queues of index server and storage server, respectively. When the length of queue reaches the limits, the server workload will be saturated and then the new arrived requests will be denied. Once a request is denied, the user's access will fail, and as a consequence, the data availability will decrease. In our model, index servers are classified and charged by their processing capacities. Server's capacity mainly depends on its own processor and memory, for example, the processing capacity of index server i is represented by μ_i $(\mu_{\min} \leqslant \mu_i \leqslant \mu_{\max})$, where $\mu_{\max}$ and $\mu_{\min}$ are the upper bound and lower bound, respectively. Storage servers are classified and charged by their I/O capacities that mainly depend on disk, for example, the I/O capacity of storage server i is represented by v_i $(v_{\min} \leqslant v_i \leqslant v_{\max})$. The cost of both index server and storage server are represented by two functions $f_I(\mu_i)$ and $f_S(v_i)$, respectively.

3.1. Performance modeling

When users' requests arrive at the system, they are dispatched to servers and will be processed according to DHT mechanism. It is assumed that the data have been stored in the system, and then the primary performance metrics concerned by users should be data availability and response delay. The former is denoted by P_{ava}, which represents the probability that users can successfully access the target data, and the latter is denoted by R, which represents the time required for the system to respond to requests. This paper strives to research on resource provisioning from the cloud storage provider's point of view. Considering users' requirements, we should focus on performance guarantee; while considering the rented servers, we should focus on cost minimization. Therefore, we combine both of them together and describe the problem as follows:

Data availability and response delay are regarded as performance metrics, while server cost is regarded as economic metrics. Hence, our problem is how to generate a resource provisioning demand for providers according to the current users' requests, so that it can meet performance metrics while optimizing economic metrics. The resource provisioning demand consists of five parameters, that is, the number of index and storage servers, the processing and I/O capacity of each server, and cost.

3.1.1. Performance analysis for index servers.. In a cloud storage system, users' requests can be satisfied within O($\log N$) hops forwarding according to DHT rules, so that the mean match rate at each hop is at least $1/(\log N + 1)$. Assuming that the mean rejection rate of index server is P_{rej}, then the meta data availability should be

$$P_{meta} = \sum_{j=0}^{\log N} A(j)B(j)\frac{j+1}{\log N + 1}, \tag{1}$$

where $A(j) = (1 - P_{rej})^{j+1}$ represents the probability that the request arrives at the $(j + 1)$th server after it finishes j hops forwarding without being denied, while $B(j) = \prod_{m=0}^{j}(1 - \frac{m}{\log N+1})$ represents the probability that the request has not been matched at previous j servers. The probability of being matched successfully at the $(j + 1)$th server is $\frac{j+1}{\log N+1}$. Assuming that the request stops at the $(j + 1)$th server, then there exists three cases, as shown in Figure 3:

(i) The request is not matched at the $(j + 1)$th server. Then the request is forwarded to the $(j + 2)$th server and is denied by the server. The probability of such case should be $B(j + 1)P_{rej}$.
(ii) The request is matched successfully at the $(j + 1)$th server. The probability of such case should be $B(j)\frac{j+1}{\log N+1}$.
(iii) The request has arrived at the last hop, that is, $j = \log N$. The probability of such case should be $A(\log N)B(\log N)$.

DOI: 10.1002/cpe

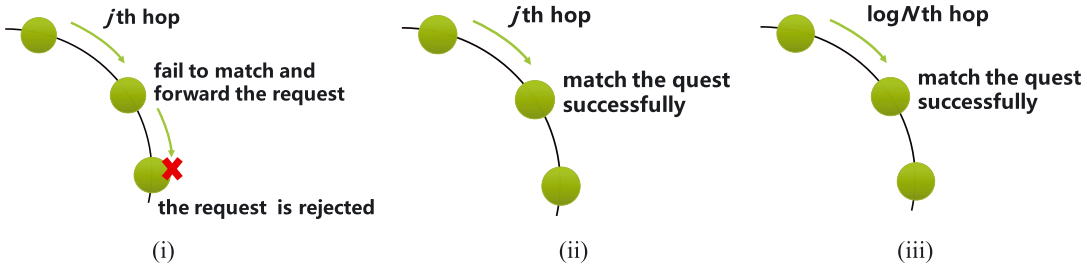

Figure 3. An example of request forwarding.

Combining the aforementioned cases, we conclude that the mean hop counts of request can be represented by

$$H = \sum_{j=0}^{\log N-1} \left(A(j)B(j) \left(\left(1 - \frac{j+1}{\log N+1} \right) P_{rej} + \frac{j+1}{\log N+1} \right) j \right) + A(\log N)\mathrm{B}(\log N)\log N \tag{2}$$

We can deduce the mean number of forwarded messages in the same way:

$$FM = \sum_{j=0}^{\log N-1} \left(A(j)B(j) \left(\left(1 - \frac{j+1}{\log N+1} \right) P_{rej}(j+1) + \frac{j+1}{\log N+1} j \right) \right) + A(\log N)B(\log N)\log N \tag{3}$$

As described by Figure 2, the arrival rate of index servers consists of both the requests λ_1 issued from users and the forwarded requests λ_2, where $\lambda_2 = \lambda_1 FM$. So the mean arrival rate from users can be calculated by λ_1/N. It is noted that the probability of receiving requests depends on the access frequency of meta data stored on server. $Q_i (0 < Q_i < 1)$ is used to represent the access frequency of server i, and then the arrival rate of forwarded requests at server i should be $\lambda_{2,i} = Q_i \lambda_2$. For server i, the arrival rate can be represented by

$$\lambda(i) = \lambda_{1,i} + \lambda_{2,i} = \lambda_1 (1/N + Q_i FM). \tag{4}$$

Response delay consists of two parts, that is, the query time T_q and I/O latency T_{io}, and T_q also includes the mean time required to forward the request, denoted by T_f, and the mean sojourn time at index server, denoted by T_s. Thus, the mean response delay should be

$$R = T_f \cdot H + T_s \cdot (H+1) + T_{io}. \tag{5}$$

We should deduce the sojourn time by analyzing M/G/1/k queuing system. There are a number of possible approaches to achieve an exact analysis: embedded Markov chains [17], regenerative processes [18], and mean busy period approaches [19], yet these approaches tend to be very complex and only appropriate for small value of k. We choose to use two-moment approximation approach [20] that is based on diffusion theory. The key idea of this approach is concluded that the discrete queuing process is approximated to a continuous diffusion process. The rejection rate of server i can be represented by a function of ρ_i, denoted by $f_{rej}(\rho_i)$, which equals the probability of having k_1 requests in the queue, that is,

$$P_{k_1,i} = f_{rej}(\rho_i) = \frac{\rho_i^{(\Phi_i + 2k_1)/(2+\Phi_i)}(\rho_i - 1)}{\rho_i^{2(\Phi_i + k_1 + 1)/(2+\Phi_i)} - 1} \tag{6}$$

where $\Phi_i = \sqrt{\rho_i e^{-s_i^2}}\, s_i^2 - \sqrt{\rho_i e^{-s_i^2}}$, $\rho_i = \lambda(i)/\mu_i$ represents the service intensity of server i and s_i represents the coefficient of variation of the service process. The mean rejection rate is calculated by

DOI: 10.1002/cpe

$$P_{rej} = \frac{1}{N}\sum_{i=1}^{N} P_{k_1,i}. \tag{7}$$

Because servers may deny the incoming requests, the effective arrival rate at server i should be less than $\lambda(i)$, and can be represented by $\lambda_e(i) = \lambda(i)(1 - P_{k_1,i})$. Thus, the probability of empty workload at server i is given by

$$P_{0,i} = 1 - \frac{\lambda_e(i)}{\mu_i} = \frac{(\rho_i - 1)}{\rho_i^{2(\Phi_i + k_1 + 1)/(2+\Phi_i)} - 1}. \tag{8}$$

The probability that there are j requests waiting in the queue of server i is $\rho_i^j P_{0,i}$, and then the mean number of requests waiting in the queue of server i should be

$$L_i = \sum_{j=0}^{k_1-1} j\rho_i^j P_{0,i} + k_1 P_{k_1,i}. \tag{9}$$

Based on Little's Formula [17], the sojourn time at server i is represented by $T_{si} = L_i/\lambda_e(i)$. Therefore, the mean sojourn time is given by

$$T_s = \frac{1}{N}\sum_{i=1}^{N} T_{si}. \tag{10}$$

3.1.2. Performance analysis for storage servers. Once the meta data has been obtained, the next step is to access files stored on storage servers. We use a queuing model to estimate the I/O latency in terms of Noop I/O scheduling mechanism. As illustrated by Figure 4, initially, the requests arrive at the cache queue. After being served by the cache, two alternatives are possible for each request: Either the request is completed and leaves with probability p_c, or the request arrives at the disk queue with probability $1 - p_c$. Merging is often used to enhance I/O performance by minimizing seeking time, and it occurs when a request is issued to an identical or adjacent region of the disk. Merging requires the kernel to sort and merge a set of requests. However, it is not suitable for truly random-access devices such as SSD, because SSD has no overhead associated with seeking. In addition, Noop I/O scheduler performs merging without sorting and merely maintains the request queue in near-first in first out order. Requests from cloud storage users are highly random. Considering these issues, we think merging has little influence on I/O performance in our system, and we do not consider request merging in this paper. Specifically, the I/O latency at a storage server has a strong correlation with the storage media. In a hybrid storage system, SSD is usually used as a secondary storage to improve I/O performance [21, 22]. The latency can be calculated by

$$\begin{cases} T_{ssd} = p_c(W_{cache} + t_{cache}) + (1 - p_c)(W_{ssd} + W_{cache} + t_{ssd}), \\ T_{hdd} = p_c(W_{cache} + t_{cache}) + (1 - p_c)(W_{hdd} + W_{cache} + t_{hdd}), \end{cases} \tag{11}$$

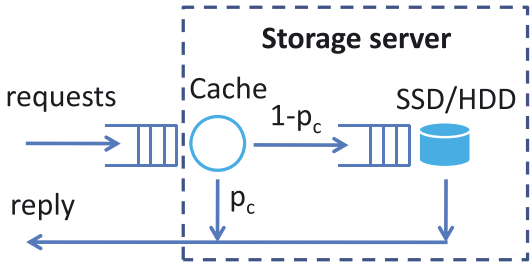

Figure 4. A queuing model for storage server.

DOI: 10.1002/cpe

where t_{cache}, t_{ssd}, and t_{hdd} represent the time required to read a block of data from cache, SSD and HDD, respectively, W_{cache}, W_{ssd}, and W_{hdd} represent the waiting time spent on queuing and can be calculated by the same approach as discussed in Section 3.1.1. Assuming that the percentage of requests replied by SSD is p_{ssd}, then the mean I/O latency is given by

$$T_{io} = p_{ssd}T_{ssd} + (1 - p_{ssd})T_{hdd}. \tag{12}$$

Before calculating the data availability, P_{ava}, we have to obtain data availability at cache, SSD and HDD, denoted by p_{cache}, p_{ssd}, and p_{hdd}, respectively. p_{cache}, p_{ssd}, and p_{hdd} could be calculated by $(1-P_{cache\_rej})$, $(1-P_{ssd\_rej})$, and $(1-P_{hdd\_rej})$, and the calculation of rejection rate is the same as analyzed in Section 3.1.1.

$$\begin{cases} P_{cache\_rej} = f_{rej}(\rho_{cache}), \\ P_{ssd\_rej} = f_{rej}(\rho_{ssd}), \\ P_{hdd\_rej} = f_{rej}(\rho_{hdd}), \end{cases} \tag{13}$$

$$P_{ava} = P_{meta}\left(p_{ssd}(p_c p_{cache} + (1 - p_c)p_{ssd}) + p_{hdd}(p_c p_{cache} + (1 - p_c)p_{hdd})\right). \tag{14}$$

3.2. Resource provisioning problem formulation

The total cost paid by a storage provider mainly includes index server cost, storage server cost, and traffic cost. The index server cost is the function of each server's capacity, and the storage server cost is the function of each server's I/O capacity. In particular, for simplicity, storage server's I/O capacity only depends on the storage media, for example, v_{ssd} *or* v_{hdd}, and we assumes that servers with the same storage media have the same disk capacity. Traffic cost caused by retrieving data can be reduced by data compression techniques, while traffic cost caused by geo-replication depends on the specific replication scheme. Our work focused on resource provisioning for a single data center; both of them are out of the scope of this paper. Another type of traffic cost, which is caused by data movement between two continuous resource provisionings, has not been considered yet, because it belongs to intra-data center traffic and is free for storage providers. Therefore, the cost discussed here mainly refers to server cost, including cost of both index server and storage server. The final objective of resource provisioning is to generate the server level resource demands to minimize cost while satisfying performance requirements. The definition of the resource provisioning problem is formalized by Equation (14).

Given the constraints (1)–(6), the objective is to find a resource provisioning demand that is able to minimize the overall cost. Constraints (1)–(2) are the thresholds of performance metrics, for example, data availability P^*_{ava} and response delay R^*. Constraints (3)–(5) are the thresholds of rejection rate of both index server P^*_{rej} and storage server, for example, SSD server P_{ssd_rej} and HDD server P_{hdd_rej}. Constraint (6) represents the range of μ_i. The index server cost is $f_I(\mu_i)$ that is a non-decreasing function of μ_i, and the request arrival rate is λ_1. N represents the number of servers and μ represents the vector of server's processing capacities. M_{ssd} or M_{hdd} is the number of storage servers with SSD or HDD, and its cost, denoted by $f_S(v_{ssd})$ or $f_S(v_{hdd})$, is also a non-decreasing function of I/O capacity. Cost is the sum of all server cost. The resource provisioning demand is represented by $(N, \mu, M_{ssd}, M_{hdd})$.

$$\begin{aligned} \text{Min} \quad & Cost = \sum_{i=1}^{N} f_I(\mu_i) + M_{ssd} f_S(v_{ssd}) + M_{hdd} f_S(v_{hdd}), \\ \text{s.t.} \quad & (1)\ P_{ava} \geqslant P^*_{ava}, \\ & (2)\ R \leqslant R^*, \\ & (3)\ P_{rej} \leqslant P^*_{rej}, \\ & (4)\ P_{ssd\_rej} \leqslant P^*_{ssd\_rej}, \\ & (5)\ P_{hdd\_rej} \leqslant P^*_{hdd\_rej}, \\ & (6)\ \mu_{\min} \leqslant \mu_i \leqslant \mu_{\max}. \end{aligned} \tag{15}$$

DOI: 10.1002/cpe

4. RESOURCE PROVISIONING ALGORITHM

Recall that the cloud storage provider's greatest concern is to maximize profit (*e.g.*, by minimizing cost) while providing high quality service (*e.g.*, by guaranteeing data availability and response delay). The resource provisioning is defined as a nonlinear cost optimization problem with performance constraints from the cloud storage provider's point of view. We decompose our resource provisioning problem into two subproblems: *index server provisioning* and *storage server provisioning*. For index servers, we are not only trying to answer how many servers need to rent, but also answer what the capacities vector of these servers is. μ is used to represent the vector of server capacities, while N is the number of rented servers and is also the dimension of capacities vector. For storage servers, we attempt to give the optimal number of both SSD servers and HDD servers. Assuming that the total amount of data is a function of time, denoted by $D(t)$, and the disk capacity is c_{hdd}, then the number of HDD servers M_{hdd} can easily be obtained by $D(t)/c_{hdd}$. We should determine the feasible range of N and M_{ssd} at the first step.

Considering constraints (4) and (5), substitute $P^*_{ssd\_rej}$ and $P^*_{hdd\_rej}$ into Equation (13), we can obtain the value of M_{ssd}, denoted by M'_{ssd}. The maximum value of M_{ssd}, denoted by M''_{ssd} is $D(t)/c_{ssd}$, where c_{ssd} is the disk capacity of each SSD server. Therefore, the feasible range of M_{ssd} is represented by $[M'_{ssd}, M''_{ssd}]$. In addition, we can calculate the possible range of T_{io} by substituting $[M'_{ssd}, M''_{ssd}]$ into Equation (12) and further obtain the range of T_s denoted by $[T'_s, T''_s]$ by substituting T_{io} into Equation (5). In the same way, substitute $P^*_{ssd\_rej}$, $P^*_{hdd\_rej}$, $[M'_{ssd}, M''_{ssd}]$, and P^*_{ava} into Equation (14), we could obtain the possible range of P_{meta}, denoted by $[P'_{meta}, P''_{meta}]$.

Substitute $[P'_{meta}, P''_{meta}]$ and P^*_{rej} into Equation (2). Then we can obtain the range of N that satisfies constraints (1) and (3), denoted by $[N_1, N_2]$. In the same way, we can also obtain another range of N, denoted by $[N_3, N_4]$ that satisfies constraints (2) and (3) by substituting $[T'_s, T''_s]$ and P^*_{rej} into Equation (6). It is noted that the server rejection rate is a non-increasing function of server capacity. Then substituting $\mu_{\max}$ or $\mu_{\min}$ in constraint (6) and $P_{k_1,i} = P^*_{rej}$ into Equation (7), we can deduce the maximal or minimal arrival rate $\lambda_{\max}$ or $\lambda_{\min}$. Combining Equations (4) and (5) together, we find that $\lambda(i)$ is related to total users' requests arrival rate λ_1, rejection rate P_{rej}, and the number of servers N. By substituting λ_1, P^*_{rej} and $\lambda_{\max}$ or $\lambda_{\min}$ into Equation (5), we can achieve the feasible range of N , denoted by $[N', N'']$ that satisfies the threshold of rejection rate. In order to satisfy all constraints, the feasible range should be trimmed by N_1, N_2, N_3, and N_4. Theorem 1 shows us the proof of feasible range of N.

Theorem 1

In the server provisioning problem, the feasible range of number of index servers that satisfies all constraints is $[\max(N', N_1, N_2, N_4), \max(N'', N_1, N_2)]$.

Proof

P_{rej} is a non-increasing function of N by analyzing Equations (7) and (8). If $N_2 \leqslant N'$, the lower bound of N, denoted by $N_{\min}$, takes the value of N' for satisfying constraint (3). If $N' \leqslant N_1$, $N_{\min}$ takes the value of N_1 for satisfying constraint (1). If $N_1 \leqslant N' \leqslant N_2$, $N_{\min}$ takes the value of N_2. In addition, R is a non-increasing function of N by analyzing Equation (5) (T_f is much smaller when compared with T_s). In order to satisfy constraint (2), N should belong to the range of $[N_3, N_4]$, that is, $N_{\min} = \max(N', N_1, N_2, N_4)$. Assume the optimal value $N^* < N_{\min}$, and then it will result in that one constraint or all of the constraints cannot be satisfied. Therefore, N_{min} should take the value of $\max(N', N_1, N_2, N_4)$.

In the same way, if $N_2 \leqslant N''$, the upper bound of N, denoted by $N_{\max}$, takes the value of N'' for satisfying constraint (3). If $N'' \leqslant N_1$, $N_{\max}$ takes the value of N_1 for satisfying constraint (1). If $N_1 \leqslant N'' \leqslant N_2$, $N_{\max}$ takes the value of N_2, that is, $N_{\max} = \max(N'', N_1, N_2)$. Assume the optimal value $N^* > N_{\max}$, and the corresponding optimal cost is $Cost^* = \sum_{i=1}^{N^*} f_I(\mu_i) + M_{ssd} f_S(v_{ssd}) + M_{hdd} f_S(v_{hdd})$, $(\mu_{\min} \leqslant \mu_i \leqslant \mu_{\max})$, then all constraints can be satisfied, and N_2 is located in the feasible range. The corresponding cost $Cost\_N_2 = N_2 f_I(\mu_{\min}) +$

DOI: 10.1002/cpe

Algorithm Resource_provisioning

Input:

$D(t)$: Total amount of data at time t;
λ_1: the total users' requests arrival rate;
$\mu_{\max}$: the upper bound of processing capacity;
$\mu_{\min}$: the lower bound of processing capacity;
Q_i: the access frequency of server i;
T_f: the mean time required to forward the request;
c_{ssd}: the disk capacity of each SSD server;
c_{hdd}: the disk capacity of each HDD server;

Output:

Opt_solution$(N, \mu, M_{ssd}, M_{hdd}, Cost)$: the optimal resource demands;

1. $M_{hdd} \leftarrow D(t)/c_{hdd}$;
2. Calculate M'_{ssd} by subjecting $P^*_{ssd\_rej}$ and $P^*_{hdd\_rej}$ to equation (13);
3. $M''_{ssd} \leftarrow D(t)/c_{ssd}$;
4. Calculate the range of T_{io} by subjecting M'_{ssd}, M''_{ssd} into equation (12);
5. Calculate T'_s, T''_s by subjecting the range of T_{io} into equation (5);
6. Calculate P'_{meta}, P''_{meta} by subjecting M'_{ssd}, M''_{ssd} and P^*_{ava} into equation (14);
7. Calculate N_1, N_2 by subjecting P'_{meta}, P''_{meta} and P^*_{rej} to equation (2);
8. Calculate N_3, N_4 by subjecting T'_s, T''_s and P^*_{rej} to equation (6);
9. Calculate $\lambda_{\max}(\lambda_{\min})$ by subjecting $\mu_{\max}\,(\mu_{\min})$ and $P_{k_1,i} = P^*_{rej}$ to equation (7);
10. Calculate $N'\,(N'')$ by subjecting $\mu_{\max}\,(\mu_{\min})$, P^*_{rej} and λ_1 to equation (5);
11. $N_{\min} \leftarrow \max(N', N_1, N_2, N_4)$, $N_{\max} \leftarrow \max(N'', N_1, N_2)$;
12. Opt_solution $(N, \mu, M_{ssd}, Cost) \leftarrow \mathbf{NLP\_OPT}\,(N_{\min}, M'_{ssd})$;
13. **if** $N_{\min} \neq N_{\max}$
14. **for** $N \leftarrow N_{\min} + 1$ to $N_{\max}$
15. **for** $M_{ssd} \leftarrow M'_{ssd} + 1$ to M''_{ssd}
16. Calculate P_{ava}, R, P_{rej} by subjecting $N, M_{ssd}, \mu = (\mu_{\max}, ..., \mu_{\max})$ to equations (14),(5),(7);
17. **if** P_{ava}, R, P_{rej} can meet constraints (1),(2),(3)
18. solution $(N, \mu, M_{ssd}, Cost) \leftarrow \mathbf{NLP\_OPT}\,(N, M_{ssd})$;
19. **if** solution $(N, \mu, M_{ssd}, Cost)$ is better than Opt_solution $(N, \mu, M_{ssd}, Cost)$
20. Opt_solution $(N, \mu, M_{ssd}, Cost) \leftarrow$ solution $(N, \mu, M_{ssd}, Cost)$;
21. **end if**
22. **end if**
23. **end for**
24. **end for**
25. **end if**
26. **return** Opt_solution $(N, \mu, M_{ssd}, M_{hdd}, Cost)$;

$M_{ssd}\, f_S(v_{ssd}) + M_{hdd}\, f_S(v_{hdd})$, but $Cost\_N_2 < Cost^*$, which conflicts with the assumption. Therefore, $N_{\max}$ should take the value of max (N'', N_1, N_2). □

The optimization problem is a nonlinear programming problem that generally belongs to NP-hard problems. To solve the problem, a novel algorithm is proposed, called **Resource_provisioning**. In the algorithm, lines 1–3 and 4–11 are used to compute the feasible range of both M_{ssd} and N, and then for each N and M_{ssd} in the feasible ranges, lines 18–20 use **NLP_OPT**(N, M_{ssd}) to solve the sub-optimization problem with the fixed value of both N and M_{ssd}. Before calling **NLP_OPT**(N, M_{ssd}), we will filter out some (N, M_{ssd}) that does not meet the constraints.

The sub-optimization problem is also a non-linear programming problem, which can be formalized as follows:

DOI: 10.1002/cpe

Algorithm NLP_OPT(N, M_{ssd})

Input:

N: the number of index servers;
M_{ssd}: the number of storage servers;
$\gamma^{(1)}$: the initial multiplier vector;
c: penalty factor;
ε: control error, $\varepsilon > 0$;
a: amplification factor, $a > 1$;
b: $0 < b < 1$;
input of **Resource_provisioning**;

Output:

solution$(N, \mu, M_{ssd}, Cost)$: the optimal resource demands;

1. Initialize $\mu^{(0)}$ with $(\mu_{\min}, \ldots, \mu_{\min})$;
2. $\tau \leftarrow 1$;
3. **while** $\Delta(\tau) \geqslant \varepsilon$
4. Solve unconstrained optimization $\min F(\mu, \gamma^{(\tau)}, c^{(\tau)})$ started with $\mu^{(\tau-1)}$ and find $\mu^{(\tau)}$;
5. **if** $\frac{\Delta(\tau)}{\Delta(\tau-1)} \geqslant b$
6. $c^{(\tau+1)} \leftarrow ac^{(\tau)}$;
7. **end if**
8. Calculate $\gamma_j^{(\tau+1)}$ with equation (19);
9. $\tau \leftarrow \tau + 1$;
10. **end while**
11. $\mu \leftarrow \mu^{(\tau)}$;
12. **return** solution$(N, \mu, M_{ssd}, Cost)$;

$$\begin{aligned} \text{Min} \quad & Cost = \sum_{i=1}^{N} f_I(\mu_i) + M_{ssd} f_S(v_{ssd}) + M_{hdd} f_S(v_{hdd}), \\ \text{s.t.} \quad & (1)\ g_1(\mu) = P^*_{ava} - P_{ava} \leqslant 0, \\ & (2)\ g_2(\mu) = R - R^* \leqslant 0, \\ & (3)\ g_3(\mu) = P^*_{rej} - P_{rej} \leqslant 0, \\ & (4)\ g_4(\mu) = \mu_{\min} - \mu_i \leqslant 0, \\ & (5)\ g_5(\mu) = \mu_i - \mu_{\max} \leqslant 0. \end{aligned} \tag{16}$$

In this paper, we use the augmented Lagrangian approach to solve the problem. By introducing slack variable z_j, the inequality constraints become equality constraints, that is, $g_j(\mu) - z_j^2 = 0$, $j = 1, 2, ..., 5$. We design the augmented Lagrangian function as follows:

$$F(\mu, \gamma, c) = Cost + \frac{1}{2c} \sum_{j=1}^{5} \left\{ \left[\max\{0, \gamma_j + c g_j(\mu)\} \right]^2 - \gamma_j^2 \right\}, \tag{17}$$

where γ is multiplier vector, c is penalty factor, and $z_j^2 = \frac{1}{c} \max\left\{0, \gamma_j + c g_j(\mu)\right\}$. Thus, the problem is transformed into a simple unconstrained optimization problem, that is, Min $F(\mu, \gamma, c)$. The solution of non-linear programming can be obtained by iteratively solving unconstrained optimization problem. Iteration rules are as follows:

$$c^{(\tau+1)} = ac^{(\tau)}, \tag{18}$$

$$\gamma_j^{(\tau+1)} = \max\{0, \gamma_j^{(\tau)} + c g_j(\mu^{(\tau)})\},\ j = 1, 2, ..., 5, \tag{19}$$

where a ($a > 1$) is the amplification factor. The condition for iteration termination is given by

DOI: 10.1002/cpe

$$\Delta(\tau) = \sum_{j=1}^{5} \left\{ \max\{g_j(\mu^{(\tau)}), \frac{\gamma_j^{(\tau)}}{c^{(\tau)}}\} \right\}^2 < \varepsilon, \tag{20}$$

where ε is the control error. Algorithm **NLP_OPT**(N, M_{ssd}) is described previously.

5. THE IMPLEMENTATION OF PARALLEL ALGORITHM

The optimization problem will become time consuming when the system scale is large. Because the feasible range of N and M_{ssd} becomes larger as the system scale increases, the program loops also increases. In this section, we design an implementation of parallel algorithm based on MapReduce paradigm labeled **MRRP** to solve the problem.

The MapReduce model can split a large problem space into small pieces and automatically parallelize the execution of small tasks on the smaller space. We deploy MapReduce to parallelize the solution of sub-optimization problems that are the most time consuming. The whole execution model consists of two phases, and the architecture is shown by Figure 5. In the 1st phase, for each sub-optimization problem, we deploy a set of map and reduce operations to find the sub-optimal solution. After that, a global selection is required, so in the 2nd phase, we deploy a reduce operation in order to achieve the optimal solution.

1) The 1st phase: The solution algorithm for sub-optimization consists of an iterative process. However, the standard MapReduce lacks built-in support for iterative programs. We design our model based on HaLoop, an efficient iterative data processing framework [23]. The initial partition is constructed according to the feasible range of N and M_{ssd}, and the record can be represented by (Ni, $M_{ssd}i$). Assuming that the sizes of feasible ranges of N and M_{ssd} are n and m, respectively. Then the parallel degree can be up to $n * m$. The map operation reads a record, solves unconstrained optimization min $F(\mu, \gamma, c)$, and then submits the result as an intermediate output, as shown in function **mapper**. The reduce operation is illustrated in function **reducer**, which receives the results generated by the map operation and updates the parameters in each iteration. HaLoop provides a caching mechanism for accelerating processing. Inspired by this mechanism, we design a reducer output cache. The cache stores the most recent local output and can be further used to evaluate the termination condition.

2) The 2nd phase: The 2nd phase consists of only one reduce operation that is called at the end of process. The final reducer collects the intermediate results generated in the 1st phase and produces the final optimal solution.

As illustrated in Figure 5, the runtime system also needs a master to coordinate the parallel execution of map and reduce tasks. In function master, partitions are built based on the feasible range

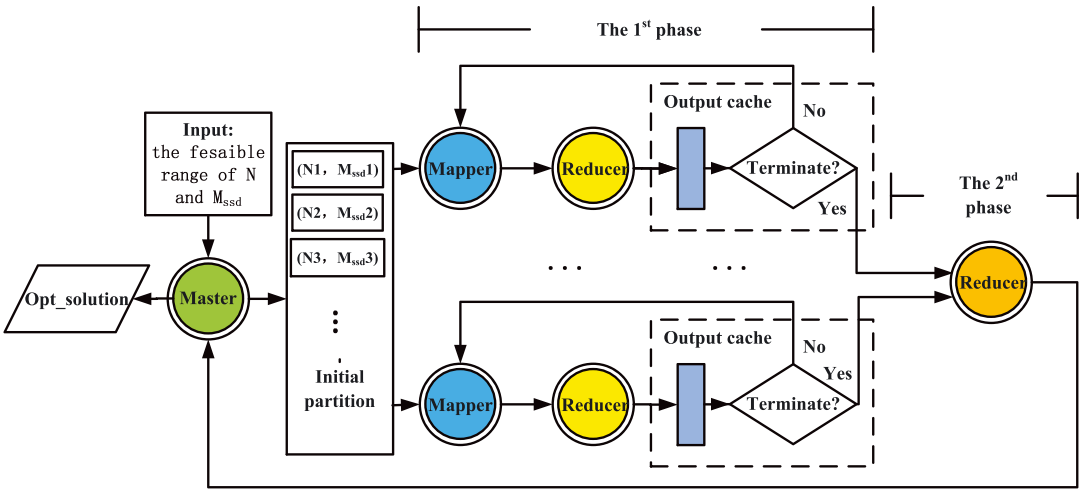

Figure 5. MapReduce based architecture for parallel implementation of algorithm.

DOI: 10.1002/cpe

Function mapper

Input:
Ni: the number of index servers;
$M_{ssd}i$: the number of storage servers;
Output:
$\mu, Cost$;
1. **if** μ is **null**
2. $\mu \leftarrow (\mu_{\min}, \ldots, \mu_{\min})$;
3. **end if**
4. **if** γ is **null**
5. $\gamma \leftarrow \gamma^{(1)}$;
6. **end if**
7. solve $\min F(\mu, \gamma, c)$ started with μ and find a new μ and the corresponding $Cost$;
8. submit($\mu, Cost$);

Function reducer

Input:
output of **mapper**;
input of **NLP_OPT**;
Output:
solution($N, \mu, M_{ssd}, Cost$);
1. read the value of both current $\Delta(\tau)$ and previous $\Delta(\tau - 1)$;
2. **if** $\frac{\Delta(\tau)}{\Delta(\tau-1)} \geqslant b$
3. $c^{(\tau+1)} \leftarrow ac^{(\tau)}$;
4. **end if**
5. $\tau \leftarrow \tau + 1$;
6. Calculate $\gamma_j^{(\tau)}$ with equation (19);
7. Calculate $\Delta(\tau)$ with equation (20);
8. **if** $\Delta(\tau) \geqslant \varepsilon$
9. continue loop;
10. **end if**
11. submit solution($N, \mu, M_{ssd}, Cost$);

of both N and M_{ssd}. For each record in partitions, if constraints (1),(2), and (3) of Equation (16) can be satisfied, a new job will be launched to solve the sub-optimization problem in parallel. After all parallel jobs finish, the master collects results from the final reducer and outputs the optimal solution.

In order to guarantee system performance, we set two trigger mechanisms together with the proposed approach:

(1) Active mechanism. We set an adjustable time window, and the proposed approach is triggered to generate the latest resource demands during each time window and adjust the resource provisioning accordingly. The window size could be altered according to the frequency of load variation. For example, if the load varies quickly, the size will become small, while if the load varies slowly, the size will become large so as to avoid the frequent changes in resources.
(2) Passive mechanism. The proposed approach is triggered in case the system performance becomes lower than the threshold values. The mechanism is used as a complement to the active one to ensure that the proposed approach could well adapt to the scenario of dynamic load.

6. EXPERIMENTAL EVALUATION

In this section, we implement a prototype of DHT-based cloud storage system. We first introduce the system design and deployment environment. Then we describe experiment setup followed by

DOI: 10.1002/cpe

Function final_reducer

Input:
solution list generated in the 1^{st} phase;
Output:
Opt_solution $(N, \mu, M_{ssd}, M_{hdd}, Cost)$;
1. Opt_solution ← **null**;
2. **for** each solution in solution list
3. **if** solution is better than Opt_solution
4. Opt_solution ← solution;
5. **end if**
6. **end for**
7. submit Opt_solution $(N, \mu, M_{ssd}, M_{hdd}, Cost)$ to master;

Function master

Input:
the feasible range of $N, and M_{ssd}$;
input of **Resource_provisioning**
Output:
Opt_solution $(N, \mu, M_{ssd}, M_{hdd}, Cost)$;
1. **for** each N in the range of N
2. **for** each M_{ssd} in the range of M_{ssd}
3. Calculate P_{ava}, R, P_{rej} by subjecting N, M_{ssd}, $\mu = (\mu_{\max}, ..., \mu_{\max})$ to equations (14),(5),(7);
4. **if** P_{ava}, R, P_{rej} can meet constraints (1),(2),(3)
5. start job;
6. **end if**
7. **end for**
8. **end for**
9. Collecting Opt_solution $(N, \mu, M_{ssd}, M_{hdd}, Cost)$ from **final_reducer**;
10. **return** Opt_solution $(N, \mu, M_{ssd}, M_{hdd}, Cost)$;

the analysis and discussion of the evaluation results of several groups of experiments based on real-world traces collected from our system and Dropbox.

6.1. System overview

Our system is implemented on top of project Voldemort [24], which is an open source implementation of Dynamo. The modular architecture is shown in Figure 6. It consists of three layers: (1) the underlying resources are managed by an infrastructure platform built with OpenStack. Through mapping virtual machines (VMs) onto physical machines, VM placement module provides server instances for the system; (2) system layer provides the core functionalities for the system operation. For example, data needs to be partitioned across multiple nodes through data partition module. The replication level is controlled by the data replication module, while the indexing module is responsible for matching of meta data of required data. The proposed provisioning scheme is implemented in resource provisioning module. This module is responsible for generating cost effective resource demands according to the current performance; (3) service layer provides some user oriented services, for example, data store, data sharing, and data query.

The infrastructure platform is built on top of a cluster of 14 IBM HS22 blade servers (2.66 GHz X5650 Six-core 2C/6*4 GB VLP DDR3/1*146 GB 6 Gbps SAS), and all of them are connected in a 1 Gbps LAN. Servers in our system can be classified as control servers, index servers, and storage servers. Control servers are in charge of dispatching requests to index servers, recording run-time log, and performance statistics. Index servers are in charge of providing the required meta data for incoming requests, and storage servers are designed to respond to I/O requests. In addition, we selected six blade servers and mounted an SSD (60 GB) on each of them.

DOI: 10.1002/cpe

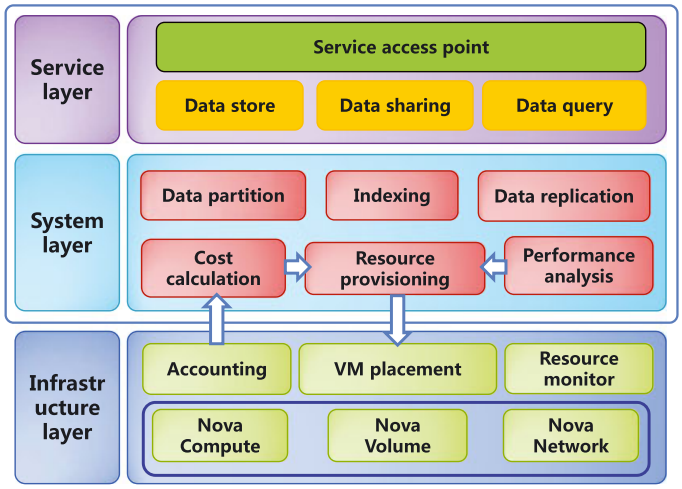

Figure 6. The modular architecture of our system.

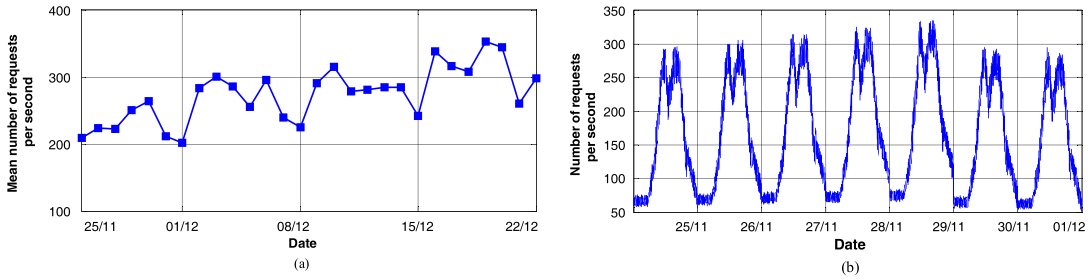

Figure 7. The arrival rate of requests received by the system.

6.2. *Trace-driven evaluation*

The system consists of a blade server as control server and one small VM and one medium VM as index servers, and two blade servers as storage servers. We set up three types of VMs: small (1VCPU 2 GB RAM, $0.06 per hour), medium (2VCPU 4 GB RAM, $0.12 per hour), and large (4VCPU 8 GB RAM, $0.24 per hour). The storage server with HDD and SSD are charged $0.0002 per gigabyte per hour, $0.001 per gigabyte per hour. Our pricing is based on Amazon EC2 and Aliyun. The system has been open to campus users since October 2013, attracted more than 500 users to date, and the total amount of storage data is 223 GB (the maximum storage size for each user is 1 GB). Files larger than 4 MB are split into several chunks. We collected real-world traces labeled *DCS* from November 25, 2013 to December 22, 2013. Figure 7(a) reports the requests received by the system during the period. The mean number of requests per second became larger as the growth of users scale from 337 to 512. It reflects a weekly pattern that the amount of concurrent visits is lower on weekends. A daily pattern is reflected by Figure 7(b). There are two peaks that appeared in the morning and afternoon separately, and the trough appears at noon and midnight. Note that request for files larger than 4 MB will be split into several requests, so the actual arrival rate of requests will be higher. Figure 8 reports separately the CDF of data availability and response delay. The performance remained stable during our measurements, meaning that the underlying resources were sufficient for supporting high level services. The little differences in response delay are related to the requests load.

6.2.1. Evaluation with traces DCS. The amount of concurrent visits was too low to evaluate our provisioning scheme. We reprocessed the traces by adding the last three weeks dataset to the first week. Then we used LoadRunner [25] to test our system. LoadRunner is a powerful performance load testing tool that can generate actual visit load based on the reconstructed traces. The

DOI: 10.1002/cpe

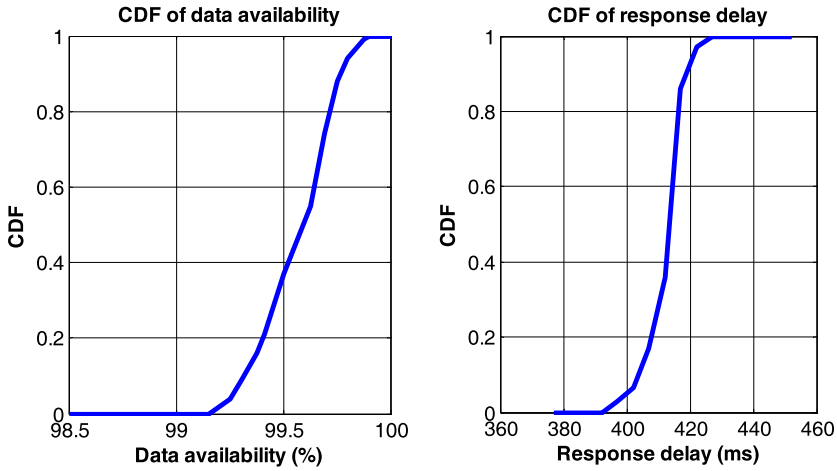

Figure 8. Distribution of service performance levels.

thresholds of P^*_{ava}, R^*, P^*_{rej}, $P^*_{ssd\_rej}$, and $P^*_{hdd\_rej}$ are set as 99%, 500ms, 0.3%, 0.1%, and 0.1%, respectively.

We conducted a group of experiments to verify the feasibility of M/G/1/k. Firstly, we deployed all types of VM and storage server with HDD and SSD in our system, and used LoadRunner to replay requests against system for measurement. The capacity values of three types of VM are 232, 525, and 1292, respectively, and the I/O capacities of storage server with HDD and SSD are 316 and 472, respectively. For an index server, its capacity is measured by the number of requests processed per second, while for a storage server, its capacity is measured by the number of chunks transmitted per second. Then we substituted capacity values into equations to calculate the sojourn time and compared with the measured values. We made comparisons under varied visit loads, and the comparison results showed that the differences between both types of values fall in the range from 2 ms to 11 ms. The differences are small enough to be acceptable for our evaluation.

Besides our scheme, we implement two resource provisioning schemes as follows for comparison:

(1) UoP: Utilization-oriented Principle (UoP) [26] is a simple and widely used resource provisioning approach and has become state-of-the-art. It tries to reduce cost by improving resource utilization (*i.e.*, equals ρ) to a predetermined range. UoP approach does not care about the specific execution logic, and the only thing it cares about is the resource utilization of every queue. In order to make a fair comparison, UoP approach is implemented based on M/G/1/k queue. The ranges are set as [60%, 70%], [70%, 80%], and [80%, 90%], respectively.
(2) DPA: A Dynamic Provisioning Approach (DPA) [14] is proposed for multi-tier applications that employ a hybrid queuing model. It assumes that the server's capacities are identical, and the objective is to minimize the total number of servers. We use the modeling approach in [14] to model the storage system as a combination of one M/M/c queue and multiple M/M/1 queues. M/M/c queue is used to model the indexing subsystem consisting of c index servers and M/M/1 queues are used to model the storage subsystem consisting of multiple storage servers. Then we implement the provisioning approach denoted by DPA by following the same solving method as [14] and set the type of server used is medium type.

Because of the limited system scale, each index server could maintain full information about all partitions (index servers) through gossiping each servers routing table. Then meta data can be found within one hop for each request. Figure 9 shows separately the CDF of P_{ava}/P^*_{ava} and R/R^*. It is concluded from Equations (7) and (8) that the rejection rate has a positive correlation with utilization rate. For UoP [60%, 70%] and [70%, 80%] approach concerned, the mean utilization rate are restricted in a low level without large variations, which results in a low level of rejection rate without large variations. When the rejection rate is low, the data availability is so high that

DOI: 10.1002/cpe

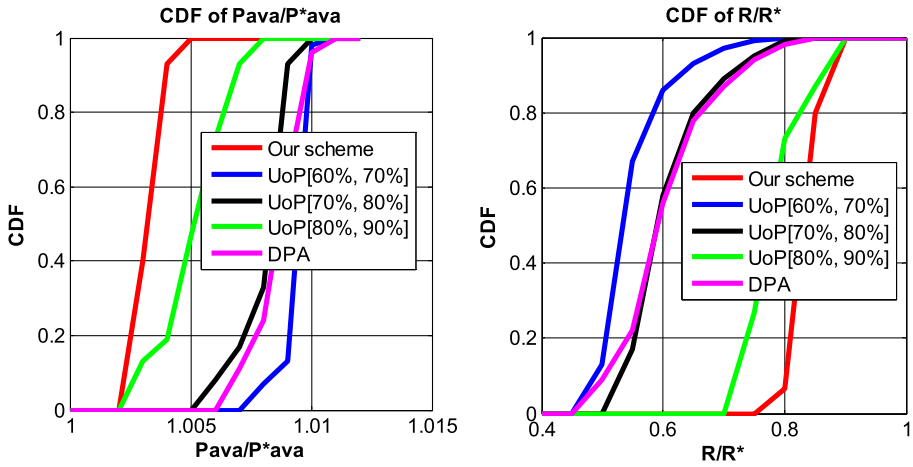

Figure 9. Comparison of distribution of service performance levels.

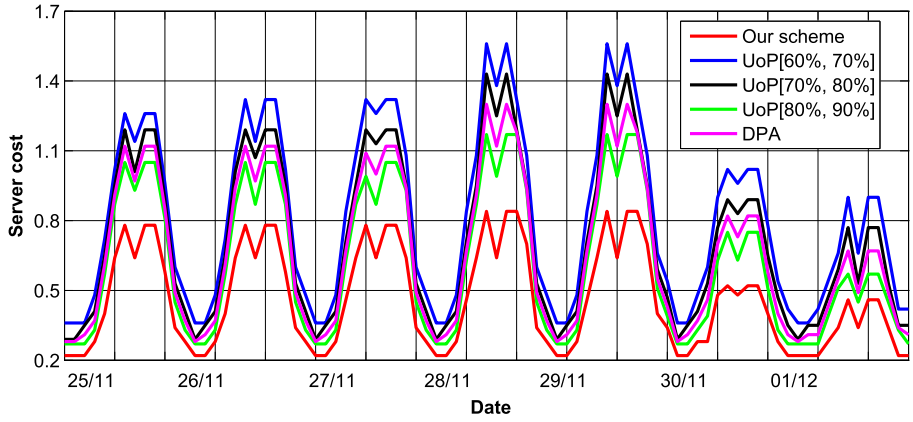

Figure 10. Comparison of cost.

we can neglect the influence of other factors on data availability. For UoP [80%, 90%] approach concerned, if the arrival rate is low, the system could maintain a high level of both utilization rate and rejection rate. As the arrival rate increases, in order to satisfy performance constraints, both utilization rate and rejection rate intend to decrease. DPA models the system as a combination of one M/M/c queue and multiple M/M/1 queues, which cannot accurately describe the execution logic of storage services because it ignores the interactions among index servers. The performance levels of DPA are between UoP [60%, 70%] and [70%, 80%]. Note that compared with the threshold, our scheme can achieve much closer data availability and response delay.

Figure 10 describes the comparison of hourly server cost by using different provisioning approaches. UoP approach is sensitive to ρ^*. It appears to be relatively conservative when ρ^* is in the interval [60%, 70%] and [70%, 80%]. Then excessive provisioning of resources results in a much higher performance level than the threshold level. Furthermore, the UoP approach pays 72.9% higher cost than our scheme, that is, the higher cost in exchange of the higher performance level. When ρ^* is in the interval [80%, 90%], it pays 31.8% higher cost than our scheme. Hence, the cost can be reduced by increasing ρ^*. However, excessive increase in ρ^* will greatly increase rejection rate. As a result, the data availability decreases and becomes lower than the threshold. The provisioning solution provided by DPA is counted by the number of servers, and its granularity is too coarse. The results show that DPA pays 47.6% higher cost than our scheme and it is even worse than UoP when the utilization rate becomes high.

DOI: 10.1002/cpe

Table I. Datasets overview.

Dataset No.	1	2	3	4
IP addresses	13723	18785	2528	400
Concurrent visit rate	[8481, 2552]	[10498, 3123]	[1474, 735]	[291, 109]
Total volume of data	$3.01*10^4$ GB	$5.1*10^4$ GB	$5.51*10^3$ GB	$5.32*10^2$ GB

In addition, we made a comparison on execution time between algorithms **Resource_provisioning** and **MRRP**. The former runs on a single blade server, and the latter runs on a Hadoop cluster consisting of six IBM HS22 blade servers. The Hadoop cluster is a public parallel computing platform, and we have installed Haloop framework upon it for supporting the iterative structure of **MRRP**. **MRRP** was actually scheduled to two servers. The execution time of both **Resource_provisioning** and **MRRP** are 11.25 s and 21.02 s, respectively. Interestingly, parallel algorithm **MRRP** runs longer than **Resource_provisioning**. The fixed number of types of VMs indicates the number of processing capacities available for selection is small, and the system scale is small as well, which results in the practical execution time of algorithm **Resource_provisioning** is not long. However, for parallel algorithm **MRRP**, although caching mechanisms of HaLoop framework can help reducing execution time by 1.85 compared with Hadoop, the processes such as job initialization and shuffle still occupy a certain percentage of execution time. Especially when the problem size is not large, time consumed by these processes may dominate the total execution time.

6.2.2. Evaluation with traces Dropbox. We consider a more realistic large-scale scenario and use Dropbox traces labeled *Dropbox* [27] collected in a European country from March 24, 2012 to May 5, 2012 for further evaluation. The traces consist of four datasets, and the overview is summarized in Table I, where we can see the unique user IP addresses, concurrent visit rate, and total volume of data observed during the whole period. The data we need are the set of visit records including access filename, file size, and access time. In order to evaluate the effectiveness of our scheme in a large-scale scenario, we extracted those data from four datasets separately and combined them together as the source of requests to do simulation. Figure 11 reports the mean request rate that appears a weekly pattern as well. In this scenario, each index server maintains only partial information, namely, the information about a limited number of others, and so indexing often requires multi-hop forwarding of requests. We compared the parallel algorithm **MRRP** with **Resource_provisioning**. The former runs on an IBM fat node (2.4 GHz X3850 eight-core 2C/32 GB VLP DDR3/1 TB 6 Gbps SAS), and **MRRP** was scheduled to five servers in Hadoop cluster for execution. In addition, we have adjusted some parameters in the cluster to optimize job executions. The execution time of both **Resource_provisioning** and **MRRP** are 5.6 h and 91.7 s, respectively. Therefore, the proposed algorithm **MRRP** can deal with resource provisioning problem efficiently in large-scale cloud storage systems.

Figure 12 shows the comparison of distribution of service performance levels. Our scheme remains achieving much closer data availability and response delay. Compared with the results reflected by Figure 9, the differences between provisioning approaches did not become larger with the growth of system scale. Meanwhile the system using UoP as well as DPA should pay much higher cost than our scheme (57.2%, 43.1%, 30.8%, and 34.4% higher), as shown in Table II.

6.2.3. Evaluation under failure scenarios.. In a production environment comprised of multiple servers, server failure may happen at any given time. Redundancy is the only way to avoid data loss incurred by server failures. Replication and erasure code are two types of commonly used redundancy approaches to enhance access performance. To evaluate the proposed scheme in a more practical environment with failure scenarios, we implement both types of approaches for comparison.

(1) Replication: In practice, the distribution of data access frequency is often nonuniform, and then there exist both hot data and frozen data. Both of them should not be replicated based on the same factor. The popularity of data stored in our system was depicted by Figure 13, where we can see

DOI: 10.1002/cpe

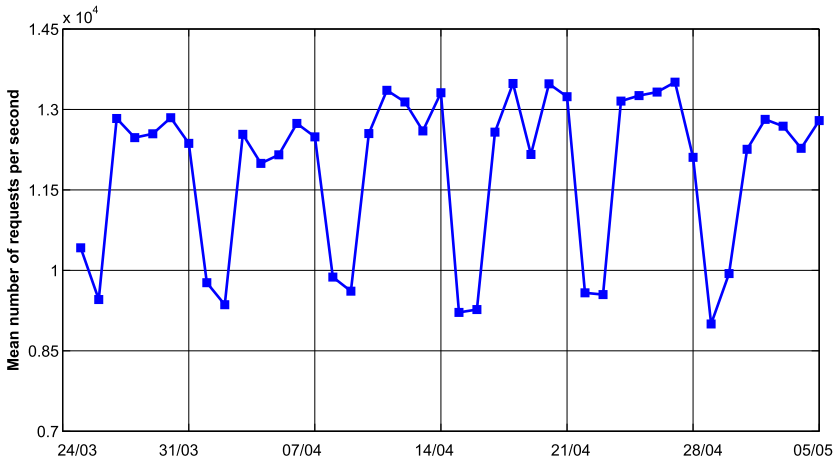

Figure 11. The arrival rate of requests in *Dropbox*.

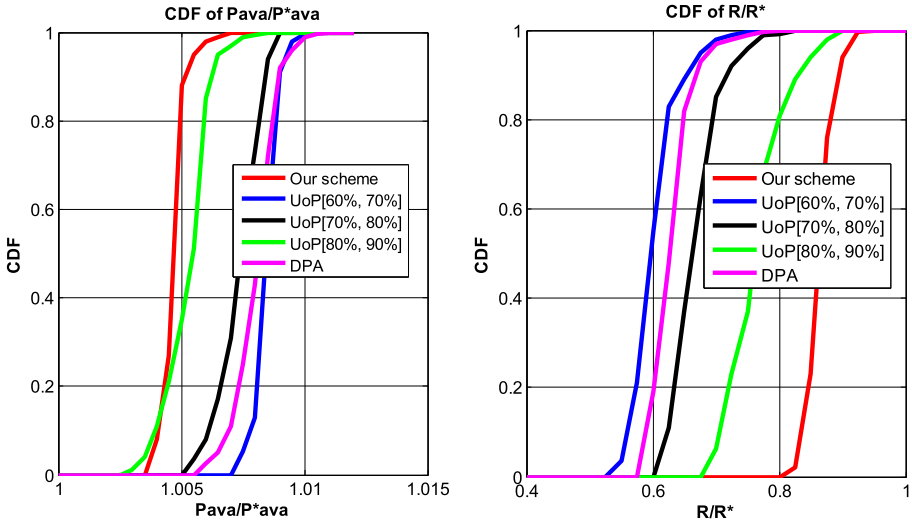

Figure 12. Comparison of distribution of service performance levels.

that files that occupy the first 20% of data size were accessed with frequency as high as 70%. We choose two replication approaches for comparison: proportional and square-root approach [28]. The former requires that each file is replicated based on a factor proportional to file frequency, while the latter requires the replication factor is proportional to the square-root of file frequency.

(2) Erasure code: In general, the erasure code can be represented by a tuple (x, y) for simplicity, where x is the total number of chunks after coding, and y is the number of chunks before coding. We have implemented an erasure coding scheme based on Reed Solomon code in our system. The reason why we choose Reed Solomon code [29] is that it has strong fault tolerance as well as high scalability. Currently, the implementation of Reed Solomon coding has been integrated in many storage systems such as Google Colossus [30], Amazon S3 and Hadoop.

We make the comparison under the scenario that servers fail at the probability of 10%, and the other settings remain the same as Section 6.2.1. From the results depicted by Figure 14, we have observed that both data availability and response delay are improved along with the increase of redundancy level (equivalent to cost increase). The proportional approach tends to cause excessive replication for hot data, and extra replicas of hot data could help to enhance data availability under failure scenarios. Hence, compared with square-root approach, proportional approach achieves a little bit higher improvement. Meanwhile, more replicas imply that most requests for hot data can get

DOI: 10.1002/cpe

Table II. Cost comparison with traces *Dropbox*.

	Server cost ($)				
Date	Our scheme	UoP [60%, 70%]	UoP [70%, 80%]	UoP [80%, 90%]	DPA
Week 1	5278.8	8116.1	7503.8	6896.1	7076
Week 2	5131.2	7825.4	7213.1	6666.2	6813.8
Week 3	5419.4	8572	7916.5	7122.7	7331.2
Week 4	5369.2	8302	7721.6	7028.9	7237.1
Week 5	5506.5	8930.9	7980.5	7270	7483.4
Week 6	5203.1	8400.2	7328.5	6767.8	6931.8
Total	31908.2	50146.6	45664	41751.7	42873.3

UoP, Utilization-oriented Principle; DPA, Dynamic Provisioning Approach.

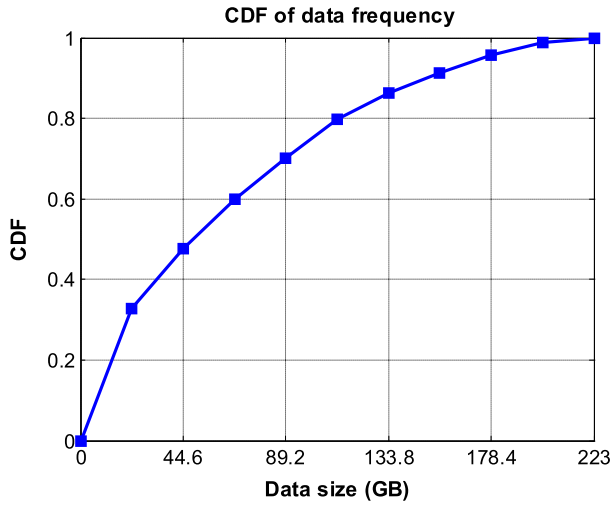

Figure 13. Distribution of data frequency in our system.

a faster reply, and the mean response delay for all data decreases as well. Compared with replication approaches, erasure coding could achieve the same level of data availability with much lower redundancy level. Interestingly, the response delay of erasure coding is much higher than others, although it can achieve a relative high data availability. It is because the system needs more time to recover the data on the failed server, and the recovery process increases the load of other servers. We also make the comparisons after reprovisioning resources based on our scheme (denoted by #-RSP). In order to preserve the threshold of data availability, replication approaches require about 40% of cost increment, while erasure coding requires only 20% of cost increment. However, erasure coding requires 27% higher cost than replication approaches to keep response delay under the threshold. Obviously there is a trade-off between performance and cost, and the best approach depends on the performance preferences, which may be used to explain the coexistence of both types of redundancy approaches in many cloud storage systems.

In addition, we run a group of experiments to further evaluate the influence of different approaches under different percentages of server failure. During the experiments, each server, including index server and storage server, is assumed to fail at a predefined probability. Table III shows the comparison results. The system has a poor fault tolerance when no redundancy. Redundancy approaches can help system achieve a good performance on fault tolerance. Erasure coding performs best in terms of both cost and data availability that remains above 80% even if 40% of servers fail. However, its response delay increases much higher as the increment of failed servers. In production environments, service provider is suggested to make a trade-off between cost and fault tolerance according to budget and quality of service.

DOI: 10.1002/cpe

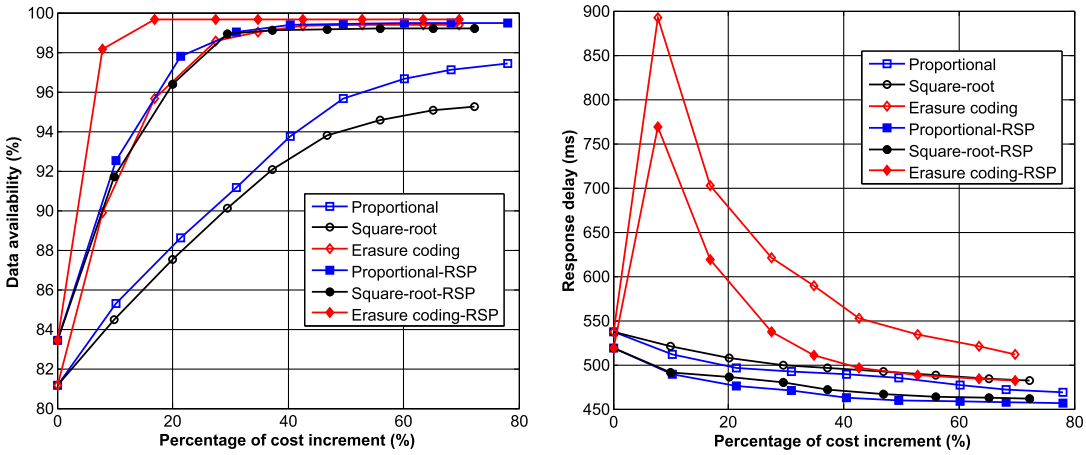

Figure 14. Performance comparison with different redundancy approaches.

Table III. Cost and performance comparison under different percentages of server failure.

Redundancy approach	Cost	Performance	Percentage of server failure 10%	20%	40%
No redundancy	68.62	Data availability (%)	81.16	58.95	37.11
		Response delay (ms)	537	749	1022
Proportional	97.68	Data availability (%)	95.64	82.91	71.03
		Response delay (ms)	489	587	703
Square-root	92.76	Data availability (%)	93.78	78.86	69.53
		Response delay (ms)	496	602	726
Erasure coding	80.92	Data availability (%)	98.16	90.48	83.06
		Response delay (ms)	569	986	1687

7. CONCLUSIONS AND FUTURE WORK

In this paper, we explore the resource provisioning from personal cloud storage provider's point of view and propose a novel resource provisioning model. The model considers the complex interactions among servers during system running by using queuing network and captures the correlation between performance metrics and the allocated resources. Then based on the model, the resource provisioning problem is defined as a cost optimization with performance constraints. We put forward solution algorithms for solving the optimization problem and design a parallel implementation of algorithm based on MapReduce paradigm for large-scale scenarios. We have built a DHT-based storage system based on real-world traces collected from system and Dropbox. The experimental results demonstrate that the proposed scheme can reduce cost while guaranteeing both data availability and response delay.

As a future work, the provisioning model will be extended to support the cloud storage systems that are built in multi-data center environments. Moreover, we intend to study the influence of data scheduling across various storage layers including memory, SSD and HDD, and energy saving will be considered as well.

ACKNOWLEDGEMENTS

This work is supported by the National Natural Science Foundation of China (No. 61502328, No. 61572337, No. 61201212), the Joint Innovation Funding of Jiangsu Province (No. BY2014059-02), the Natural Science Foundation of the Higher Education Institutions of Jiangsu Province (No. 15KJB520032), the Research

DOI: 10.1002/cpe

Starting Funding of Soochow University (No. Q411800314), the Open Project Funding of Soochow University (No. S811800114), and the Collaborative Innovation Center of Novel Software Technology and Industrialization.

REFERENCES

1. DeCandia G, Hastorun D, Jampani M, Kakulapati G, Lakshman A, Pilchin A, Sivasubramanian S, Vosshall P, Vogels W. Dynamo: Amazons highly available key-value store. *Acm Symposium on Operating Systems Principles (SOSP)*, Stevenson, WA, USA, 2007; 205–220.
2. Lakshman A, Malik P. Cassandra: a decentralized structured storage system. *Operating Systems Review* 2010; **44**(2):35–40.
3. Chen F, Mesnier MP, Hahn S. Client-aware cloud storage. *The 30th Symposium on Mass Storage Systems and Technologies (MSST)*, Santa Clara, CA, USA, 2014; 1–12.
4. Wang H, Shea R, Wang F, Liu J. On the impact of virtualization on dropbox-like cloud file storage/synchronization services. *IEEE/ACM International Symposium on Quality and Service (IWQOS)*, Coimbra, Portugal, 2012; 1–9.
5. Butler B. Personal cloud subscriptions expected to reach half a billion this year, Network World, 2012. (Available from: http://www.networkworld.com/article/2159722/cloud-computing/personal-cloud-subscriptions-expected-to-reach-half-a-billion-this-year.html) [Accessed on 7 September 2012].
6. Marketsandmarkets. *Public/private cloud storage market by solution, by software - worldwide forecasts & analysis*, 2014. (Available from: http://www.marketsandmarkets.com/Market-Reports/cloud-storage-market-902.html) [Accessed on 15 April 2014].
7. Drago I, Mellia M, Munaf MM, Sperotto A, Sadre R, Pras A. Inside dropbox: understanding personal cloud storage services. *Acm Internet Measurement Conference (IMC)*, Boston, MA, USA, 2012; 481–494.
8. Ghemawat S, Gobioff H, Leung S-T. The google file system. *ACM Symposium on Operating Systems Principles (SOSP)*, Bolton Landing, NY, USA, 2003; 29–43.
9. Zhang Q, Zhani MF, Boutaba R, Hellerstein JL. Dynamic heterogeneity-aware resource provisioning in the cloud. *IEEE Transactions on Cloud Computing* 2014; **2**(1):14–28.
10. Xiong K, Perros HG. Service performance and analysis in cloud computing. *SERVICES I*, Los Angeles, CA, USA, 2009; 693–700.
11. Shi Y, Jiang X, Ye K. An energy-efficient scheme for cloud resource provisioning based on cloudsim. *IEEE International Conference on Cluster Computing (CLUSTER)*, Austin, TX, USA, 2011; 595–599.
12. Zhu Z, Bi J, Yuan H, Chen Y. Sla based dynamic virtualized resources provisioning for shared cloud data centers. *The IEEE International Conference on Cloud Computing (CLOUD)*, Washington, DC, USA, 2011; 630–637.
13. Zhang C, peng Chen H, Gao S. Alarm: autonomic load-aware resource management for p2p key-value stores in cloud. *The 9th IEEE International Conference on Dependable,Autonomic and Secure Computing (DASC)*, Sydney, Australia, 2011; 404–410.
14. Bi J, Zhu Z, Tian R, Wang Q. Dynamic provisioning modeling for virtualized multi-tier applications in cloud data center. *The IEEE International Conference on Cloud Computing (CLOUD)*, Miami, FL, USA, 2010; 370–377.
15. Lama P, Zhou X. Efficient server provisioning with control for end-to-end response time guarantee on multitier clusters. *IEEE Transactions on Parallel and Distributed Systems* 2012; **23**(1):78–86.
16. Zhou J, He W. A novel resource provisioning model for dht-based cloud storage systems. *The 11th IFIP International Conference on Network and Parallel Computing (NPC)*, Ilan, Taiwan, 2014; 257–268.
17. Gross D, Shortle JF, Thompson JM, Harris CM. *Fundamentals of Queueing Theory* (4th edn). Wiley: New York, NY, USA, 2008.
18. Tijms HC. *Stochastic Modelling and Analysis: A Computational Approach*. Wiley: New York, NY, USA, 1986.
19. Cooper RB. *Introduction to Queueing Theory* (2nd edn). Elsevier North Holland: New York, NY, USA, 1981.
20. Smith JM. Properties and performance modelling of finite buffer m/g/1/k networks. *Computers & Operations Research* 2011; **38**(4):740–754.
21. Yang Q, Ren J. I-cash: Intelligently coupled array of ssd and hdd. *The 17th International Conference on High Performance Computer Architecture (HPCA)*, San Antonio, Texas, USA, 2011; 278–289.
22. Chen F, Koufaty DA, Zhang X. Hystor: making the best use of solid state drives in high performance storage systems. *The 25th International Conference on Super Computing (ICS)*, Tucson, AZ, USA, 2011; 22–32.
23. Bu Y, Howe B, Balazinska M, Ernst MD. Haloop: efficient iterative data processing on large clusters. *Proceedings of the VLDB Endowment* 2010; **3**(1):285–296.
24. Sumbaly R, Kreps J, Gao L, Feinberg A, Soman C, Shah S. Serving large-scale batch computed data with project voldemort. *The 10th USENIX Conference on File and Storage Technologies (FAST)*, San Jose, CA, USA, 2012; 1–18.
25. *Hp loadrunner tutorial*, 2010.
26. *Aws elastic beanstalk*. (Available from: http://aws.amazon.com/elasticbeanstalk/) [Accessed on 2015].
27. *Dropbox traces*. (Available from: http://traces.simpleweb.org/dropbox/) [Accessed on 2015].
28. Tewari S, Kleinrock L. Proportional replication in peer-to-peer networks. *The 25th IEEE International Conference on Computer Communications(INFOCOM)*, Barcelona, Catalunya, Spain, 2006.
29. Plank JS, Luo J, Schuman CD, Xu L, Wilcox-O'Hearn Z. A performance evaluation and examination of open-source erasure coding libraries for storage. *The 7th USENIX Conference on File and Storage Technologies (FAST)*, San Francisco, CA, USA, 2009; 253–265.
30. McKusick K, Quinlan S. GFS: evolution on fast-forward. *Commun. ACM* 2010; **53**(3):42–49.

DOI: 10.1002/cpe