Space4time: Optimization Latency-Sensitive Content Service in Cloud

Lingfang Zeng *, Bharadwaj Veeravalli †, Qingsong Wei §

* *Wuhan National Laboratory for Optoelectronics*
School of Computer, Huazhong University of Science and Technology, China, 430074.
† *Department of Electrical and Computer Engineering, National University of Singapore, Singapore 117576.*
§ *Data Storage Institute, Singapore, 117608.*
E-mail: lfzeng@hust.edu.cn;elebv@nus.edu.sg;wei_qingsong@dsi.a-star.edu.sg

Abstract

Nowadays, as Cloud service is increasingly a commodity, some Cloud Service Providers (CSPs) make profit by providing content services, and play the more important role of delivering content to users. Providing content services presents new challenges for coordination between storage systems and network infrastructures, specifically for latency-sensitive applications such as voice, video, and terminal services.

However, prior research has not applied collaboration techniques of storage and network inline to the request path for latency sensitive applications. In this paper, we propose a latency-sensitive content distribution mechanism, *Space4time*, for a real world system. We observe that operation is largely affected by the collaboration among end users, storage and networks in Cloud. Meanwhile, dynamic request routing within the cloud is strongly coupled with content placement decisions when designing the mechanism. Based on blocking probability, we propose content distribution and request routing strategies that take as input storage and network traffic information. Our strategies enable us to balance storage capacity savings and network traffic for performance, as demonstrated in our YouTube trace-based simulation. Our evaluation shows that Space4time outperforms *StaticScheme* on average 22.3% for access latency improvement. With more sites, Space4time has a better performance due to reduce traffic.

Keywords: Cloud, Storage, Content distribution, Latency sensitive, Blocking probability

1. Introduction

Cloud computing for content distribution services is becoming a well accepted technique or concept. As Cloud services are increasingly a commodity, Cloud Service Providers (CSPs) make profit by providing computing and storage services, and play the more important role of delivering content to users. The services are in general hard to achieve with the traditional technologies due to the key characteristics of cloud environments as well as the provided services.

CSPs may use the on-demand scaling feature of the Cloud and easily adjust its requirement for the cost-effective computation powers and storage capacities. Although the content distribution network (CDN) problem has been widely studied in the literature [1, 2, 3], there are some unique challenges when considering this problem in a Cloud environment. On one hand, a CSP can construct an arbitrary network based on the demands to facilitate the accesses. This overlay network may have different topologies with respects to the underlying physical networks provisioned from different ISPs (Infrastructure Service Providers). As a consequence, the content service is becoming a joint problem, requesting for both access serving and content distributing in a Cloud. On the other hand, latency-sensitive content services in Cloud bring a great impact on CSPs, traditional network model and the business model is difficult to meet the needs. Some of these characteristics include: (1) The rapid changes of service requests: the service requests are in general highly time and location varying. They are continuously changing with respect to the time and location of the users, e. g. mobile users; (2) The availability of unreliable computation, storage and network resources: the unreliability is caused by multiple factors, including dynamism that introduces unpredictable and changing behaviors, failures that have an increasing probability of occurrences as system/application scales increase, and instability of system states that is intrinsic to Cloud environments. (3) The heterogeneous traffic characteristics of Cloud applications have posed many technical challenges. For instance, some latency-sensitive applications such as voice, video, and terminal services begin to constitute an ever-increasing fraction of Internet traffic.

In Cloud environment, latency-sensitive content services need high bandwidth requirements and stable bandwidth guarantee, e.g. IPTV generally

requires 500Kbit/s-1Mbit/s, due to two-way asymmetric flow requirements and concurrent service. Latency-sensitive content services require a lot of software and hardware resources, the application servers and storage servers need support large concurrent services / requests, e.g. generally a server need support $1,000$ concurrent video streams accessed. The traffic behavior is constant bit rate, or bursty traffic with silence suppression. They are very sensitive to delay, jitter and loss. High quality interactive voice and video applications can tolerate little delay variation and packet loss. Clearly, providing Cloud services without considering these factors may significantly increase the access delays and much worse impose a large amount of access traffics on the system which might course a service disruption.

Another, to truly fulfill the promise of cloud computing, a rising trend is to federate different Cloud services (in separate data centers). However, in this paradigm, if CSPs and content providers are independent entities, e.g. CSPs only provide storage, computation and connectivity, CSPs find their profit margin shrinking. This motivates CSPs to host and distribute content to their customers, e.g. IPTV or online game services. When CSPs provide cloud computing and content service dependently, they can optimize their performance with cooperation [4].

So, there are many factors that affect the latency-sensitive applications such as the load on the application servers [5], the request routing mechanism [6], traffic engineering [7] and data access bottlenecks [8], etc. One dominating factor that affects latency-sensitive applications is the access latencies from application servers to storage servers. For most of CSPs, its primary role is to deploy infrastructure, manage storage, computation and connectivity inside its infrastructure, which solves the availability problem, i.e., providing fault-tolerant strategy. To offer its own content service, a CSP replicates content over a number of storage servers and directs requests to different servers. The CSP should solve the content distribution and server selection problem, i.e., determining which storage servers should reserve content replication or which server should deliver content to each end user. The goal is to meet the CSP and user demand, minimize the operation monetary cost, minimize network latency to reduce user waiting time, and balance server load to increase throughput.

In this paper, we study several approaches a CSP could take in managing content distribution and request routing, ranging from running the tree systems (application server, storage server and network) independently to designing a joint system. Based on heterogeneous request patterns at differ-

ent locations of the system and asymmetric settings of storage capacities, we design our content placement strategy and the corresponding request routing rules. This paper makes the following main contributions:

- We proposed a novel content distribution and request routing solution, *Space4time* (space for time). Based on blocking probability, Space4time effectively exploits the storage and network capacity for latency-sensitive applications.
- From an architecture point of view, Space4time provides a space-time tradeoff. Space4time enable us to balance storage capacity savings and network traffic for performance.
- We conducted comprehensive trace-based simulation. Our evaluation shows that Space4time achieves on average 22.3% for access latency improvement.

The remainder of the paper is organized as follows. In Sec. 2, we discuss our motivation with comparisons to related works. In Sec. 3, we present characteristics of Space4time, along with a description of our problems at hand. Sec. 4 introduces the design and implementation of Space4time. Sec. 5 evaluates the performance of Space4time. Finally, we conclude the paper in Sec. 6.

2. Related Work

Previous efforts in content distribution service typically fall into two categories: 1) those that make little use of different ISP topologies, preferring to build an efficient and cost-effective storage system [9, 10, 11], thereby ignoring content replication concerns and potentially facing communication cost limitations and, 2) those that focus on the *traffic engineering* by adjusting the routing configuration to the prevailing traffic [12, 4, 7] and the *server selection* by determining which servers should deliver content to each end user [6, 8]. Zheng et al. [8] present storage migration scheduling algorithm that can greatly improve storage I/O performance during wide-area migration. These efforts can optimize latency-sensitive content service by minimizing network latency, reducing user waiting time, and balancing server load to increase throughput. However, most of these methods were proposed to address one or two topics optimization problems of content storage, server selection and

communication (traffic engineering). Two topics, if considered, were usually being converted to either a weighted one of content storage, server selection and communication problem or modeled as a constrained one topic problem.

Recently, we have witnessed an escalating interest in the research towards content distribution service in *Cloud environment*. Many classical optimization methods, such as hierarchical cache placement strategies [13, 14, 15] distributed caching algorithms [16], data staging algorithms [17, 18, 19, 20], and optimizing data access latencies by intelligent virtual machine placement [21, 22, 23, 24, 25, 5]. Also, a considerable amount of research has been done for content placement and *monetary cost* in CDNs for *Cloud environment*. These systems provide mechanisms to place content in different storage cloud provider networks and redirect user requests to appropriate replicas. In Clouds the charge model for uploading and downloading the content replicas is often asymmetric with different prices, which implies that the content replication directions are usually needed to take into account in the distribution decisions [26]. The monetary cost model has evolved to include network (download/upload), storage, computation and power cost [27, 28]. MetaCDN by Broberg et al. [29] is a low cost CDN using storage cloud resources. Li et al. [30] took the storage and bandwidth costs into consideration and reduced the operating expenses for supporting IPTV services. Fast provisioning of contents and VM instances has significant impacts on the overall system performance and elasticity requirement. Peng et al. [31] took advantage of the hierarchical network topology of data centers to reduce the VM instance provisioning time and also to minimize the overhead of maintaining chunk location information. SpotCloud by Wang et al. [32] is a customer-provided cloud platform and enables the customers to find the best tradeoff between benefit and cost. By exploiting social influences among users, Wu et al. [33] proposed efficient proactive algorithms for dynamic, optimal scaling of a social media application in a geo-distributed cloud. Dai et al. [34] discussed the collaborative hierarchical caching with dynamic request routing for massive content distribution.

According to our observations, these solutions focus on content distribution and network topology and ignore to discuss the cooperation between the application servers, storage servers, and network. Also, in Cloud environment, the overall topology is different from the traditional hierarchical structure that is widely applied in web caching systems, e.g. [26] and VOD caching systems. In these existing works, requests are simply forwarded to the upper-layer parent server when the content is not locally available. To of-

fer both cloud service infrastructure and content delivery, a CSP is faced with coupled content placement and request routing problems. Content placement and request routing interact because content placement affects the operation monetary cost, and request routing affects the offered load seen by the server.

Actually, the content placement controls cost matrix, which is the constant parameter in the budget problem, while the request routing controls routing matrix, which is the constant parameter in the profit problem. In most of existing work, requests are simply forwarded to the upper-layer parent servers when the content is not locally available. Or request-redirection occurs over distributed sets of servers, to minimize redirection latency [6]. This adaptation may include dynamically shipping the request as well as the requested data set to some vantage locations in the cloud that are close to the users so that the total access latency of the request sequence and migration costs of the request and data are minimum.

However, prior research has not applied collaboration techniques of storage and network inline to the request path for latency-sensitive workloads in cloud sites. In this paper, we propose a latency-conscious content distribution mechanism based on blocking probability for a real-world system. We observe that access latency is largely affected by the collaboration among network and storage servers (SSs) in Cloud. Meanwhile, dynamic request routing within the cloud is strongly coupled with content placement decisions when designing the mechanism which takes as input storage capacity, network traffic and topology, and work load information such as user location and request rates. Our work is similar to Dai *et al.* [34] and Chen *et al.* [26]– dynamic request routing is designed jointly with content placement strategies – and focuses on evaluating the new features of adaptation to the changes of concurrent access patterns to efficiently satisfy the user requests in a cost effective way. However, different from [26] and [34], Space4time focuses on that contents are distributed in a CSP and collaborative *flat storage*. In our cooperative environment, requests are directed to sets of storage servers deployed across multiple sites belonging to a single CSP.

Another Space4time is exclusively focused on optimization latency-sensitive content service by reducing blocking probability (Sub-section 3.2). In [9], they showed that allocating the movies to disks such that each disk has a uniform probability of being accessed can minimize the session blocking probability. Different form [9] and [3], system allocate contents according to the hit rate distribution due to the phenomenon that many contents will only be demanded with low hit rates, whereas others may be simultaneously re-

quested with high hit rates), Space4time minimizes blocking probability by finding an optimal *traffic intensity vector*.

3. Problem Formulation

In this section, we first present our system model, based on the requirement of a real-world IPTV system [1]. And then we analyze the potential of storage and network traffic collaboration and propose our challenges and a practical decomposition of our problem.

3.1. System Model

We primarily follow the model presented in [26] to define the problem of dynamic content replication. We consider a substrate n-site Cloud G = (V,E) with an arbitrary shape provided by a CSP/ISP (infrastructure service provider). Let G be a directed graph, V denotes a set site with a large amount of servers, $(v_i, v_j) \in E$ indicates a feasible provisioning or replication path from site v_i to site v_j. Clearly, given above assumption, the shortest path between any pair of sites in G can be measured by the total number of hops along the path from the source to the destination.

We consider a geo-distributed cloud infrastructure which consists of multiple disparate data centers distributed in different geographical locations, and owned by one CSP. Each data center contains a collection of interconnected servers. There are two categories of servers, storage servers (SSs) to store data files and application servers (ASs) to support the running and provisioning of virtual machines (VMs); all application and storage servers inside a data center are inter-connected with high speed switches and LAN buses. Different data centers are connected over a WAN. For ease of understanding, we summarize the notations and their meanings used throughout of this paper in Table 1.

Each data center is composed of Γ independent heterogeneous SSs. All SSs in the cloud store a total of Z different files b_1,b_2,$\cdots$,b_Z, where $Z \gg \Gamma$. A file b_j $(1 \leq j \leq Z)$ is modeled as attribute tuple $b_j = (p_j, s_j, \gamma_j, d_j)$, where p_j, s_j, γ_j and d_j are popularity (i.e. file b_j be requested with probability p_j.), size, replica number, and access latency requirement of file b_j respectively.

[1] This system is a commercial deployment of China Telecom Guangzhou which provides IPTV service to millions of users in a metropolitan network. Please refer to [34] for more details.

Table 1: Major notations used in this paper.

Symbol	Definition
G	A substrate n-site Cloud
V	A set site with a large amount of servers
v_i	Cloud site
SS	Storage server
AS	Application server
ACM	Admission control manager
Γ	Number of SS
b	File
Z	Number of files
p_j	Probability of file b_j be requested
s_j	Size of file b_j
γ_j	Replication number of file b_j
d_j	Access latency requirement of file b_j
λ_y	Request arrive rate of storage server SS_y
τ_y	Average service time of storage server SS_y
ξ_y	Failure probability of storage server SS_y
bw_y	Network bandwidth of storage server SS_y
h_y	Average hit ratio of storage server SS_y
σ	A sequence of batch requests
R_i	A set recording indices corresponding to the requests to be redirected in time window of t_i
r_{iu}^g	The gth request in SS_u (in time window of t_i)
$I_{u,g}$	Traffic intensity for the gth request in SS_u
$\mathbf{I}$	Traffic intensity vector
Q	Assignment matrix
$F(Q)$	Blocking probability function for Q
X	Total number of requests
Y	Total number of storage servers (SSs) in Cloud
N_y	Total number of sessions in the SS_y
ϕ	Blocking probability
P_k	Steady-state probability of kth state
$\lambda_{x,y}$	Average arrival rate of request r_{iy}^x
Λ_y	Average arrival rate of SS_y
$\mho_y$	Service rate of SS_y
$\rho_{x,y}$	Average utilization of r_{iy}^x
$S_{v_j,rep}$	Size of the replicated files' of site v_j
$S_{v_j,used}$	Used storage space in site v_j
$S_{v_j,total}$	Total storage capacity of site v_j
SC_{ss}	Storage capacity of a SS
f	Difference of traffic intensity
Ψ	Vector distance
T_i	Time window at time t_i
γ_{max}	Maximal replica number
θ, ε	System constant

We model storage server SS_y $(1 \leq y \leq Y)$ as $SS_y = (\lambda_y, \tau_y, \xi_y, bw_y, h_y)$, where λ_y, τ_y, ξ_y,and bw_y are request arrive rate, average service time, failure probability, network bandwidth, and average hit ratio of storage server SS_y respectively.

When retrieving a file b_j from storage server SS_y with performance requirement (i.e. access latency less than d_j.), bandwidth s_j/d_j should be assigned to this session to guarantee performance. Obviously, in SS_y, the total bandwidth used to serve different requests should be no more than bw_y at any time.

$$bw_y \geq \sum_{j=1}^{N_y} \frac{s_j}{d_j} \tag{1}$$

where N_y is maximal network sessions storage server SS_j can serve concurrently, and can be calculated from Eq. (1).

3.2. Blocking Probability

Suppose assignment matrix is $Q = [q_{x,y}]_{X \times Y}$, with X as the total number of requests, Y as the total number of storage servers (SSs) in Cloud. If a request r_{iy}^x is assigned to SS_y at time t_i, then $q_{x,y} = 1$, or else, $q_{x,y} = 0$. The target of routing policy is to find the optimal assignment matrix Q that minimizes the blocking probability function $F(Q)$, and satisfy the following model:

$$\begin{aligned} &Minimize: \quad F(Q),\ Q = [q_{x,y}]_{X \times Y}, q_{x,y} \in \{0, 1\}, \\ &0 \leq x \leq X - 1, 0 \leq y \leq Y - 1 \end{aligned} \tag{2}$$

An admission control manager (ACM) manages a global request assignment matrix: when a request is arrived, it updates the assignment matrix; when a request is assigned, by accessing the assignment matrix the system gets the ID of the SS which the request belongs to.

Due to a limited bandwidth of each SS, only few sessions can be supported simultaneously. Also, note that, different SSs have different service capacities. We suppose an SS_y $(0 \leq y \leq Y - 1)$ can simultaneously support a maximum N_y (calculated from Eq. (1)) sessions.

Assume that the request arrival from clients to an SS_y is a Poisson process with an average arrival rate Λ_y, and $\mho_y$ is the service rate of the SS_y. Then the traffic intensity (or average utilization) of SS_y is $I_y = \frac{\Lambda_y}{\mho_y}$.

Suppose $\lambda_{x,y}$ is the average arrival rate of request r_{iy}^x in an SS_y at time $t_i (0 \leq x \leq X - 1$, where, X is the number of requests in the $SS_y)$, $\mu_{x,y}$ is the

service rate of r_{iy}^{x} in the SS_y, the average utilization of r_{iy}^{x} is $\rho_{x,y} = \frac{\lambda_{x,y}}{\mu_{x,y}} = \lambda_{x,y} \cdot s_{x,y}$, then, we have the following relationships:

$$\Lambda_y = \sum_{x=0}^{X-1} q_{x,y} \cdot \lambda_{x,y} \tag{3}$$

$$(\frac{1}{\mho_y}) = \frac{\sum_{x=0}^{X-1} q_{x,y} \cdot \lambda_{x,y}}{\Lambda_y \cdot \mu_{x,y}} = \frac{1}{\mu_{x,y}} \tag{4}$$

So, we can obtain:

$$I_y = \sum_{x=0}^{X-1} q_{x,y} \cdot \rho_{x,y} = \sum_{x=0}^{X-1} q_{x,y} \cdot \frac{\lambda_{x,y}}{\mu_{x,y}} \tag{5}$$

The evaluation of the number of accessed SSs under admission control can be modeled by a continuous time finite state Markov chain where the system state is represented by the number of SSs (Γ) being used. We will adopt our assumption that the request arrival rate (while the system is in state k) is Λ_k. The steady-state probability of kth state is given by P_k:

$$\begin{cases} \Lambda_y \cdot P_0 & = & \mho_y \cdot P_1 \\ (\Lambda_y + k\mho_y) \cdot P_k & = & \Lambda_y \cdot P_{k-1} + (k+1) \cdot \mho_y \cdot P_{k+1} \\ N_y \cdot \mho_y \cdot P_{N_y} & = & \Lambda_y \cdot P_{N_y - 1} \end{cases} \tag{6}$$

and,

$$\sum_{k=0}^{N_y} P_k = 1 \tag{7}$$

Using Equations (6) and (7), we obtain:

$$P_k = \frac{I_y^k}{k!} (\sum_{m=0}^{N_y} \frac{I_y^m}{m!})^{-1}, \ (0 \leq k \leq N_y) \tag{8}$$

When the SS_y is in the state N_y (meaning that the SS is confronted with the maximal access streams), then those requests for SS_y after the state will be redirected. So, the blocking probability (ϕ) of SS_y is $\phi_y = P_{N_y}$, given as follows:

$$\phi_y = \frac{I_y^{N_y}}{N_y!} (\sum_{m=0}^{N_y} \frac{I_y^m}{m!})^{-1}, \ (0 \leq y \leq N_y) \tag{9}$$

For the assignment matrix Q, the blocking probability function for the SSs can be obtained using Erlang formula:

$$F(Q) = \phi = \sum_{y=0}^{Y-1} \frac{I_y}{I} \phi_y = \frac{1}{I} \sum_{y=0}^{Y-1} \frac{I_y^{N_y+1}}{N_y!} (\sum_{m=0}^{N_y} \frac{I_y^m}{m!})^{-1} \tag{10}$$

where,

$$I = \sum_{y=0}^{Y-1} I_y \tag{11}$$

Note that, I is the traffic intensity of the SSs. From Equations (9) and (10), we observe that ϕ_y and ϕ is determined by I_y, that is to say, popularity and mean service time of the requests have a great impact on the load of SSs. In order to clearly describe, $\mathbf{I} = (I_0, I_1, I_2, \cdots, I_{Y-1})$, a vector (referred to as a traffic intensity vector) is introduced, denotes the distribution of SSs' traffic intensity. Eq. (10) is the access latency function of our scheme. An object should be reasonably placed so as to make the least value of the Eq. (10). Eq. (10) shows that the ϕ is the function of I, so, an optimal value of I_y $(0 \leq y \leq Y-1)$ should be determined to support the Eq. (7) as well as to minimize ϕ. We aim to find an optimal traffic intensity vector $\mathbf{I}_{opt} = (I_0^{opt}, I_1^{opt}, \cdots, I_{Y-1}^{opt})$ for the minimization of ϕ (ϕ_{min}). A procedure with gradient search algorithm is designed to find I^{opt}. The floor of the SS' blocking probability is given by:

$$\phi_{min} = \frac{1}{I} \sum_{y=0}^{Y-1} \frac{(I_y^{opt})^{N_y+1}}{N_y!} (\sum_{m=0}^{N_y} \frac{(I_y^{opt})^m}{m!})^{-1} \tag{12}$$

From Eq. (10), we can obtain the following product form:

$$(\frac{\partial \phi}{\partial I_y}) = \phi_y [N_y + 1 - I_y(1 - \phi_y)], \ (0 \leq y \leq Y-1) \tag{13}$$

and finally, the gradient of ϕ is given by:

$$\frac{\partial \phi}{\partial \mathbf{I}} = (\frac{\partial \phi}{\partial I_0}, ..., \frac{\partial \phi}{\partial I_{Y-1}}) \tag{14}$$

3.3. Remarks

As discussed, the *blocking probability* represents the probability of request not is serviced for a group of identical parallel storage servers. It can represent a probability in a queuing system where with a number of storage servers but no queuing space for incoming requests to wait for a free storage server. Eg.(12) implies under the condition that an unsuccessful request, because the storage is busy, is not retried or queued, but instead really not serviced forever. It is assumed that request attempts arrive following a Poisson process, so requests are independent. Blocking probability is an essential issue in latency-sensitive content service. Users expect to get the service immediately after sending request. However, the request may be rejected, or blocked by the service system. This is also true in Cloud storage system, where the capabilities of storage servers are diverse and limited, so each storage sever can only admit a restricted number of requests.

4. Design and Implementation of Space4time

4.1. Admission Control Manager (ACM)

Because hundreds of SSs might coexist in data centers, sorting and searching blocking probability of all SSs is challenging. An attractive choice for managing index and list structures is by using B+ tree implementation because it maintains records in sorted order and scale efficiently in both time and space. The ACM uses B+ tree to sort SSs in descending order using their blocking probability ϕ_i as the key. When we want to find a candidate SS to service request, the ACM quickly searches the B+ tree and return a SS with lowest blocking probability. Blocking probability of each SS is calculated locally in SS side and updated to the ACM periodically, which can reduce ACM management workload. With updated value of blocking probability, ACM quickly rebuilds the B+ tree and makes decision of request assignment.

Assume ϱ_{max} represents a percentage of a data center's total storage space, each data center will still have some storage available to facilitate the file replication. In the case that ϱ_{max} is set to 100%, additional temporary storage space may need to be acquired to serve as a buffer before the file replication process can be completed. We use $DEL\_THRESHOLD = 85\%$ as the replica deletion threshold to reclaim storage space and $REC\_THRESHOLD = 20\%$ as the reclaim space threshold for each reclaim execution. In our system, for every data center, we use ϱ_{max} - ($DEL\_THRESHOLD$ - $REC\_THRESHOLD$)=

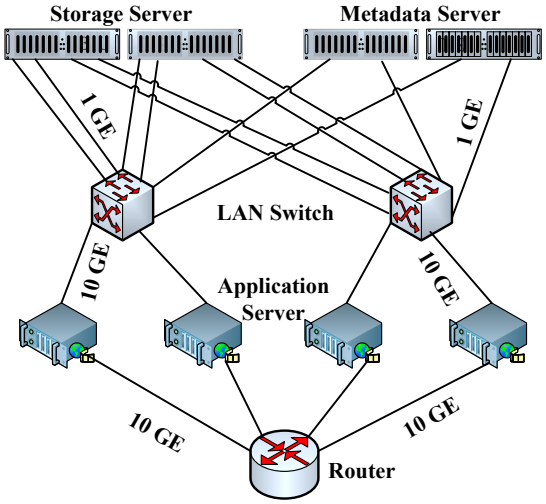


Figure 1: Real-world system overview in a data center.

35%[2] to reserve certain amount of runtime storage for future file replication. We use $\varrho_{ts} = 95\%$ as the maximal utilization limit for a data center's storage space. So, a successful file replication should satisfy the following equation:

$$S_{v_j,rep} + S_{v_j,used} < S_{v_j,total} \times \varrho_{ts}, v_j \in V \tag{15}$$

where, $S_{v_j,rep}$ is the size of the replicated files of site v_j, $S_{v_j,used}$ is the used storage space in site v_j, and $S_{v_j,total}$ is the total storage capacity of site v_j.

In the situations when $\gamma_i = 1$ and the files are distributed such that each SS has a uniform probability of being accessed, the bounds for the number of SSs required for the Cloud can satisfy the system requirement **only if** the following equations can be satisfied:

$$max\{\frac{\sum_{i=1}^{Z} s_i}{SC_{ss}}, \lceil \frac{X}{N_y} \sum_{i=1}^{Z} (1 - h_i \cdot p_i) \rceil\} \leq \Gamma \leq \frac{X}{N_y} \tag{16}$$

where, SC_{ss} denotes the storage capacity of a SS.

For a real world system (as shown in Fig. 1 and Table 2), in specific, the root router connects four ASs and each AS connects a LAN switch,

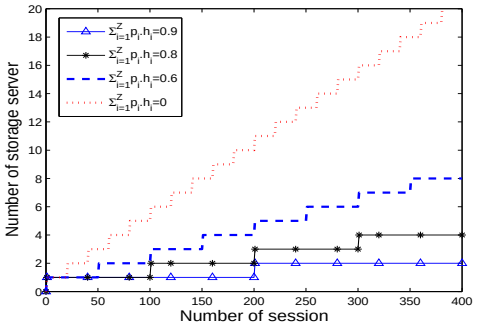
[2]Mark Levin [35] shows that the percentage of storage managed by SANs is nearly the same for UNIX (65%) and Windows environments (63.6%).

Table 2: Example system characteristics.

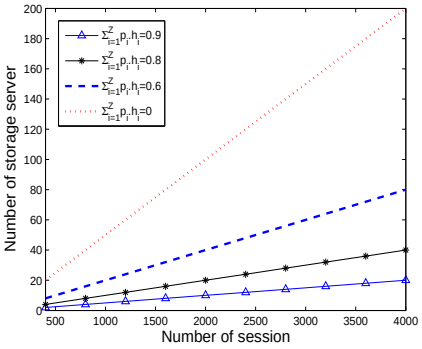
Data Center	20 Storage Server (SS) 2 Metadata Server (MDS) 4 Application Server (AS) 2 LAN Switch 1 Router
SS & MDS	$6 \times 1GE$ $12 \times 1TB\ SATA\ 7200rpm$
AS	$2 \times 10GE$ $12 \times 1TB\ SATA\ 7200rpm$
LAN Switch	$48 \times 1GE + 2 \times 10GE$
Router	$8 \times 10GE$

each consisting of 48 (1Gbps) and 2 (10Gbps) ports. We assign 10Gbps, 10Gbps/1Gbps and 1Gbps for router, AS, switch, and SSs links, respectively. Since the lower bound is obtained from the constraints imposed by the probability of access and hit ratio for the most popular file, and the upper bound is derived from the extreme case when $\sum_{i=1}^{Z} h_i \cdot p_i = 0$. These bounds are illustrated in Fig. 2 for $N_y = 20$ (the data size of each session is 5MB/s), and various values of sessions. Fig. 2(a) demonstrates the bounds for a small number of sessions, while Fig. 2(b),(c) illustrate the bounds for a large number of sessions.

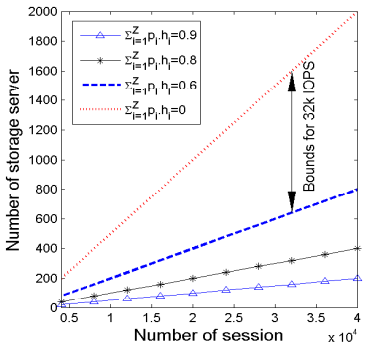
To address the sensitivity of the scheme to a content replacement and request routing, we performed two simulations. Although, Space4time provides an efficient scheme for the content service in a heterogeneous system paradigm, we set all the SSs are alike in order to simplify the simulation experiment. The first simulated a cloud with 100 SSs and the second simulated a cloud site with 100 SSs. Each SS contains 78000 files and supports a maximum of 20 concurrent sessions. Each session is assumed to have an exponentially/uniform distributed viewing time with a mean of 120 minutes. Fig. 3(a) illustrates that the exponential request arrival rate results in some files which were excessively popular, while others were very unpopular. Fig. 3(b) shows the probability of new session blocking versus request arrival rate. The results indicate that a "uniform" content placement to different SSs always yields the highest session availability.



(a) 0-400 sessions



(b) 400-4,000 sessions



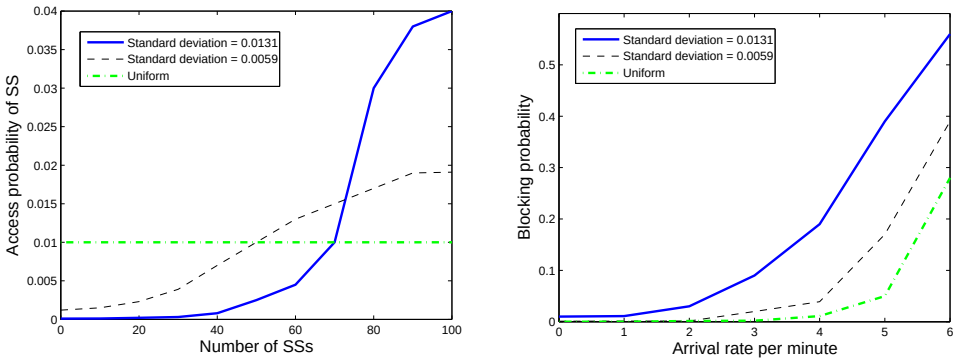
(c) 4,000-40,000 sessions

Figure 2: Bounds of Γ for SS. Note that, the low bound of total storage capacity is larger than $\sum_{i=1}^{Z} s_i$.

4.2. Content Placement and Request Routing Strategies

The problem we investigate in this paper is a joint problem of content placement and dynamic request routing scheme. Storage, servicing and routing decisions are made based on user request patterns, heterogenous storage sizes, link capacities and the specific system topology. Our objective is to explore the capacity of the existing system infrastructure by maximizing the amount of supported requests and minimizing the access latency of requests.

Intuitively, shorter paths of data packets result in less traffic in the network backbone. To maximize the amount of supported requests, it is more favorable to replicate popular contents at each of the edge layer servers. However, Space4time is questionable when we introduce the cooperation among content servers through dynamic request routing in the application hierarchy.



(a) SS access probability (b) Blocking probability

Figure 3: Simulation results for storage servers.

Moreover, the problem becomes even more complicated when we consider the bandwidth consumed during the delivery of massive contents in systems, e.g. IPTV or file downloading, which is especially important when we deal with heterogeneous link capacities and user demands in real-world systems. It has been shown in previous work [36] that the content placement problem in the cache hierarchy is NP-hard even without considering dynamic routing schemes and link capacity constraints.

4.2.1. Content Placement

A global replication server (GRS) contacts metadata server (MDS) with availability setting and file number. The MDS inserts the file name into the file system hierarchy, sets the maximal replica number γ_{max} (defined by CSP) and quickly searches the storage server B+ tree to obtain a list of storage servers for each file. The MDS responds to the AS I/O request with a list of storage servers for each file, the destination files and replication factor. Then the master storage server flushes each file of data from the local temporary file to the specified storage server and its replicas to selected storage servers in pipelining way.

In real application, workloads may change frequently. To adaptively satisfy availability and load balance according to dynamic environment in terms of node failure and access pattern changes, we develop a dynamic replica placement policy (referred to as DRPP) running on storage server, as illustrated in Algorithm (1). DRPP divides the entire files into hot files and cold files by simply using least-recently-used (LRU) scheme. For each file,

we initially assign 3, 4 or 5 replicas.

By monitoring the workload changes and evaluating the efficiency of control in current time window T_i, Algorithm (1) dynamically adjusts replica number and replica location in next time window T_i+1 with updated parameters input from storage servers. DRPP maintains reasonable replica number in the system, and adjusts replica location according to workload changes. In case of SS unreachable and current replica number less than maximal replica number γ_{max}, a new replica will be added to a SS with lowest blocking probability to guarantee the availability requirement. If the storage utilization is larger than the replica deletion threshold $DEL\_THRESHOLD$, the *Reclaim()* function will issued to reclaim $REC\_THRESHOLD$ storage space in the SS with highest blocking probability. The time complexity of Algorithm (1) is $O(|T| \times Z)$, where $|T|$ is the number of time window and Z represents the number of files.

4.2.2. Request Routing

We can obtain ϕ_{min} and I^{opt} using gradient search method. However, a common gradient search method uses step model and it may not converge when this model approaches the extreme point. In the following proposed gradient search algorithm (shown in Algorithm (2)), a small system constant θ is used to control the precision of the result, ε is a system constant and it is used to tune the speed of gradient search, and the step ε is variable. A searching oscillation results to withdraw a step and to make for another direction searching. When all directions show oscillation, the length of step is halved again and a different direction is selected. In *Step 3*, the gradient of ϕ can be computed by Eq. (13). Initially, ε is set to a large value, and the procedure approaches the extreme point in coarse-grained step. Once the algorithm detects the step is too big (in *Step 6* $\sim$ *14*), it reduces ε to make a short step. Eventually, the procedure will approach the extreme point with fine-grained step. This makes the procedure avoid fluctuating around the extreme point and obtains more precise results. The time complexity of Algorithm (2) is $O(2Y \times \log \varepsilon)$, where Y is the number of storage servers.

When get I^{opt}, the access latency function of object layout can be transferred to a *vector distance*:

$$\Psi = ||\mathbf{I} - \mathbf{I}^{opt}||^2 = \sum_{y=0}^{Y-1} f_y^2 \tag{17}$$

```
1  γ_max=5;
2  DEL_THRESHOLD = 85%;
3  REC_THRESHOLD = 20%;
4  foreach time window T do
5      if Storage utilization ≥ DEL_THRESHOLD then
6          Reclaim();
7      end
8      Get φ_i reported by GRS;
9      Reorder storage server B+ tree using φ_i as key;
10     for i = 0; i < Z; i++ do
11         γ_i=current replica number of file b_i;
12         if γ_i < γ_max and Eq. (15) is satisfied then
13             Add (γ_max − γ_i) replicas to (γ_max − γ_i) storage servers with
               lowest φ;
14         end
15     end
16 end
17 Function Reclaim() is
18     while Free space < REC_THRESHOLD do
19         for i = 0; i < Z; i++ do
20             Get φ_i reported by GRS;
21             Reorder storage server B+ tree using φ_i as key;
22             if γ_i > 1 then
23                 Find the SS with highest φ;
24                 Delete file's (b_i) replica with lowest LRU;
25                 Increase the Free space;
26             end
27         end
28     end
29 end
```

Algorithm 1: Dynamic replica placement policy running on GRS

1 Randomly generate the initialized traffic intensity vector $\mathbf{I}^{init}$ which support Eq. (11);
2 $\phi^* = 0$;$\mathbf{I}^A = \mathbf{I}^{init}$; $\varepsilon = \varepsilon_{init}$;
3 $\mathbf{I}^B = \mathbf{I}^C = \mathbf{I}^A - \varepsilon \frac{\partial \phi}{\partial \mathbf{I}}|_{\mathbf{I}=\mathbf{I}^A}$; $E = \{0, 1, \cdots, Y-1\}$;
4 Randomly select a number from set E, denoted as k, adjust I_k^B to satisfy Eq. (11): $I_k^B = I - \sum_{y \neq k, 0 \leq k \leq Y-1} I_y^B$;
5 Compute ϕ (in Eq. (10)) using $\mathbf{I}^B$;
6 **if** $\phi \geq \phi^*$ **then**
7 | /* Drawback a step and adjust direction.*/
8 | $E = E - \{k\}$;
9 | **if** $E \neq \Phi$ **then**
10 | | $\mathbf{I}^B = \mathbf{I}^C$; Go to Step 4;
11 | **end**
12 | /*The step is halved and restarts before the surge point.*/
13 | $\varepsilon = \frac{\varepsilon}{2}$; Go to Step 3;
14 **end**
15 **if** $|\phi - \phi^*| > \theta$ **then**
16 | $\phi^* = \phi$; $\mathbf{I}^A = \mathbf{I}^B$; Go to Step 3;
17 **end**
18 **else**
19 | /*Find the optimal traffic intensity vector. */
20 | $\mathbf{I}^{opt} = \mathbf{I}^B$; Stop;
21 **end**

Algorithm 2: Gradient search for getting I^{opt}.

where $f_y = I_y - I_y^{opt}$ is the difference of traffic intensity. This is, the target of object layout is to minimize f_y which is the sum of ideal traffic intensity difference and each traffic intensity of SS. The ideal vector distance is 0.

The gist of our request routing algorithm is given in the following Theorem 4.1.

Theorem 4.1. *Suppose a request is assigned in a SS, the optimal assignment is that the request is assigned in an SS_i subject to $f_i \leq f_y$ $(0 \leq y \leq Y-1)$.*

Proof It is only required to prove the vector distance is the least for a request assigned in an SS_i. Set the access traffic intensity is ρ_x, the request

is assigned in the SS_i. Suppose f_i has the least traffic intensity, and so, the vector distance (Ψ) is:

$$\Psi = ||\mathbf{I} - \mathbf{I}^{opt}||^2 = \sum_{y=0, y\neq i}^{Y-1} f_y^2 + (f_i + \rho_x)^2 \tag{18}$$

If the request is placed in any other $SS_k (k \neq i \text{ and } 0 \leq y \leq Y-1)$, then, the vector distance is:

$$\Psi' = \sum_{y=0, y\neq k}^{Y-1} f_y^2 + (f_k + \rho_x)^2 \tag{19}$$

To prove Theorem 4.1, it is sufficient to show that $\Psi \leq \Psi'$. We prove this inequality as follows:

$$\Psi - \Psi' = (f_i + \rho_x)^2 - (f_k + \rho_x)^2 + f_k^2 - f_i^2 = 2\rho_x(f_i - f_k) \tag{20}$$

Since f_i has the least traffic intensity, $f_i \leq f_y$ for all $0 \leq y \leq Y-1$. This means that Inequation (20) is less than or equal to zero. Therefore, $\Psi - \Psi' \leq 0$ or $\Psi' - \Psi \geq 0$.

When a request arrives, according to the **Theorem 4.1**, the request can be assigned a SS whose traffic intensity difference f_y is the smallest. According to the Eq. (10) and Eq. (11), I^{opt} is related each access traffic intensity $\rho_x = \frac{\lambda_x}{\mu_x}$, thus it affects the traffic intensity difference of each SS, and moreover, it affects the request assignment too. When the system is running, it is very difficult to capture ahead the workload characteristic, as traffic intensity may change along with respect to time.

For an online request adjustment algorithm, although, the ACM need not to know the average arrival rate and the average length of request, the workload characteristics of each request in a SS can be obtained by self-learning. When the workload status is changed, a SS can apperceive the changing and adjust the routing policy of sequential request. The traffic intensify and the utility rate of an SS_y is I_y, and $\frac{I_y}{B_y}$, respectively. An SS avoids allocating a request to a SS whose utility rate approaches 1. When allocating a SS for a request, we should assure $I_y < 1$ or $\frac{I_y}{B_y} < 1$.

Our request routing algorithm is to choose/redirect a request to a SS by reducing the vector distance. Suppose a request r_{iu}^{x1} in an SS_u at time t_i (its

traffic intensity is ρ_{x1}) is interchanged with another request r_{iv}^{x2} in a SS_v at time t_i (its traffic intensity is ρ_{x2}). Then the variance of vector distance is:

$$\begin{aligned} \Psi' - \Psi &= (f_u - \rho_{x1} + \rho_{x2})^2 + (f_v - \rho_{x2} + \rho_{x1})^2 - f_u^2 - f_v^2 \\ &= 2(\rho_{x1} - \rho_{x2})[(\rho_{x1} - \rho_{x2}) - (f_u - f_v)] \end{aligned} \quad (21)$$

So, when $\rho_{x1} - \rho_{x2} < f_u - f_v$, there is $\Psi' - \Psi < 0$, the variance of vector distance caused by the interchange (redirect) is change to small value. Specially, when the value of $\rho_{x1} - \rho_{x2}$ approaches $\frac{(f_u - f_v)}{2}$, the $\Psi' - \Psi$ can obtain the least value.

At the beginning of the routing process, the procedure tries to assigns requests that have large traffic intensities (larger than a threshold $I_{threshold}$) among SSs. Then comes several rounds of request redirecting. In each round, the procedure tries to move the requests from the SS with maximal f_y to the SS with minimal f_y(may be a minus), here $(0 \leq y \leq Y - 1)$. When selecting a request to redirect, the procedure attempts to choose the request with traffic intensity as large as possible. The procedure is depicted as request routing algorithm (Algorithm 3). The time complexity of Algorithm (3) is $O(|R_i| \times (1 + Y \log_2 Y)$, where $|R_i|$ represents the number of request set in t_i's time window and Y is the number of storage servers.

The adjustment procedure can save the vector I^{opt} in a cache table, and avoid using gradient search procedure to compute the vector every time. In the static environment where no SS is added or removed and the N_y of SS_y remains unchanged, because I^{opt} is only determined by the total traffic intensities I, the cache table can be indexed by I. For a given value of I, an item indexed by I^* in the table is considered to be matched with I if $|I - I^*|$ is less than a given threshold. If the search fails, the adjustment algorithm calculates the optimal vector with total traffic I. When the environment changes, the cache table should be invalidated and be re-built dynamically.

5. Performance Evaluations

In this section, we evaluate Space4time using system settings derived from a real-world Cloud system, conduct simulation studies driven by real traces.

5.1. Simulation Setup

We use Java to implement a simulator that constructs the system topology under several ISP topologies (AT&T, Level3, Sprint and Ebone). The raw

$R_i = \Phi$; /* R_i is the set recording indices corresponding to the requests to be redirected in time window of t_i, r_{iu}^g is the gth request in SS_u, and $I_{u,g}$ is its traffic intensity */
foreach *request* r_{iu}^g *in the system which satisfies* $I_{u,g} > I_{threshold}$ **do**
    $p = min\{\lceil \frac{I_{u,g}}{I_{threshold}} \rceil, Y\}$;
    Sort $(f_0, f_1, \ldots, f_{u-1}, f_{u+1}, \ldots, f_{Y-1})$ in descending order, denoted as $(f_{v_0}, f_{v_1}, \ldots, f_{v_{p-1}})$;
    $R_i = R_i \bigcup \{< u, v_0, g >, < u, v_1, g >, \ldots, < u, v_{p-1}, g >\}$;
    $f_u = f_u - I_{u,g} \frac{(p-1)}{p}$;
    $I_u^{org} = I_u^{org} - I_{u,g} \frac{p-1}{p}$;
    $f_x = f_x + \frac{I_{u,g}}{p}$ (for $x = v_0, v_1, \ldots, v_{p-1}$);
    $I_x^{org} = I_x^{org} + \frac{I_{u,g}}{p}$ (for $x = v_0, v_1, \ldots, v_{p-1}$);
end
$f_u = max(f_0, \cdots, f_{Y-1})$; /*$0 \leq u \leq Y-1$ */
$f_v = min(f_0, \cdots, f_{Y-1})$; /* $0 \leq v \leq Y-1$ */
if $f_v \geq 0$ **then**
    foreach 3-*tuple* $< u, v, g >$ *in* R_i **do**
        Migrate request r_g^{iu} from SS_u to SS_v;
    end
    $I_u^{org} = I_u^{org} - I_{u,g}$;
    $I_v^{org} = I_v^{org} - I_{u,g}$;
    Stop;
end
$f_m = min(|f_u|, |f_v|)$; /* m may be set to u or v */
while $f_m > 0$ **do**
    Find request r_{iu}^g in SS_u, such that $I_{u,g}$ has the maximal value while satisfying condition $I_{u,g} \leq f_m$;
    $R_i = R_i \bigcup \{< u, v, g >\}$; $f_u = f_u - I_{u,g}$; $f_v = f_v + I_{u,g}$;
    $f_m = f_m - I_{u,g}$;
end
Go to Step 11;

Algorithm 3: Request routing.

data was obtained from Rocketfuel [37] and released at [38]. For each problem instance we evaluate, we randomly pick one site which has the origin dataset

and a subset of sites as cloud provides. We then place the group of chosen sites including the origin dataset and the CSPs onto a metric field using the geographical information of each site. To place a server, we first look up its IP address in the GeoLite city database [39] and convert the IP address to its corresponding (latitude, longitude) pair. We consider the case where latency and resource utilization are the primary performance metrics. We choose a small number of providers and vary the number from 10 to 40. We show such a small number because it reflects the fact that there are few CSPs today.

YouTube is the most popular video serving website. We extract end user request patterns from YouTube trace [40]. The traces used in our experiments are collected from a campus network with a total of 6 different traces. Since our main design goal is to significantly benefit IPTV applications in content distribution and performance, we select 1 of the 6 traces, that have the larger number of overall requests for use in the main portion of our experiments and sensitivity studies to show how different design parameters may impact our strategies.

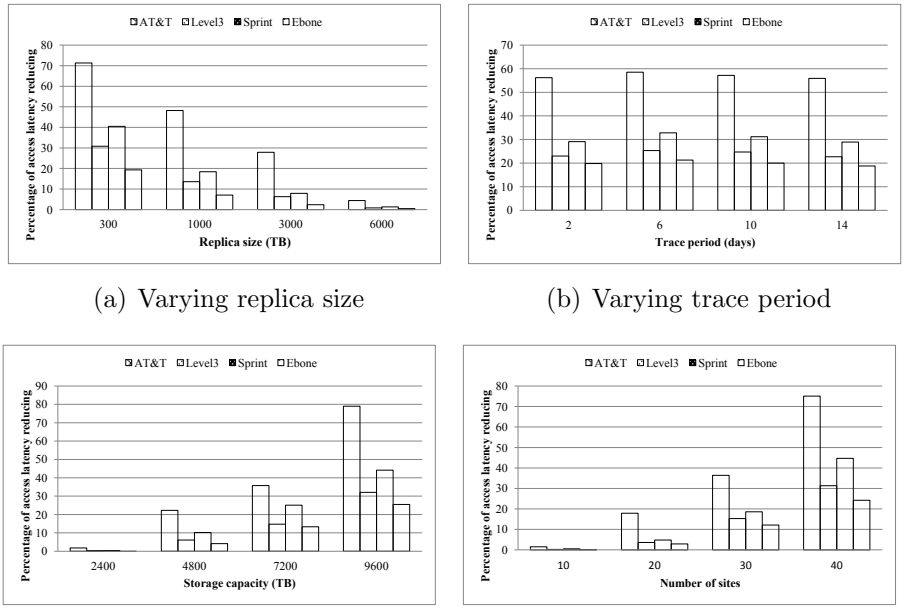
The traces chosen for our experimental study have different length, number of unique clients, videos and requests, to represent multiple types of workloads in real enterprise-level data centers. The summary of the characteristics of the trace (T5) is listed in Table 3.

Table 3: YouTube trace characteristics.

Trace	Length (h)	# of clients	# of videos	# of req.
T5	336	16336	303331	611968

5.2. Performance

We compare the performance between the Space4time scheme (denoted as *Space4time*) and the static content distribution and request routing (denoted as *StaticScheme*). For the StaticScheme configuration, for each file, we set a steady replication number randomly selected from 3, 4 or 5 and the contents are stored at a random location. For the sake of fairness, the same initializing settings apply to Space4time. We evaluate both Space4time and StaticScheme schemes on different ISP topologies which have the topological properties of different graphs. For instance, Ebone is the simplest graph. The backbones of AT&T has a hub-and-spoke structure with some shortcuts between nodes pairs. The topology of Level3 is almost a complete mesh, while Sprint is in between these two kinds.



(a) Varying replica size

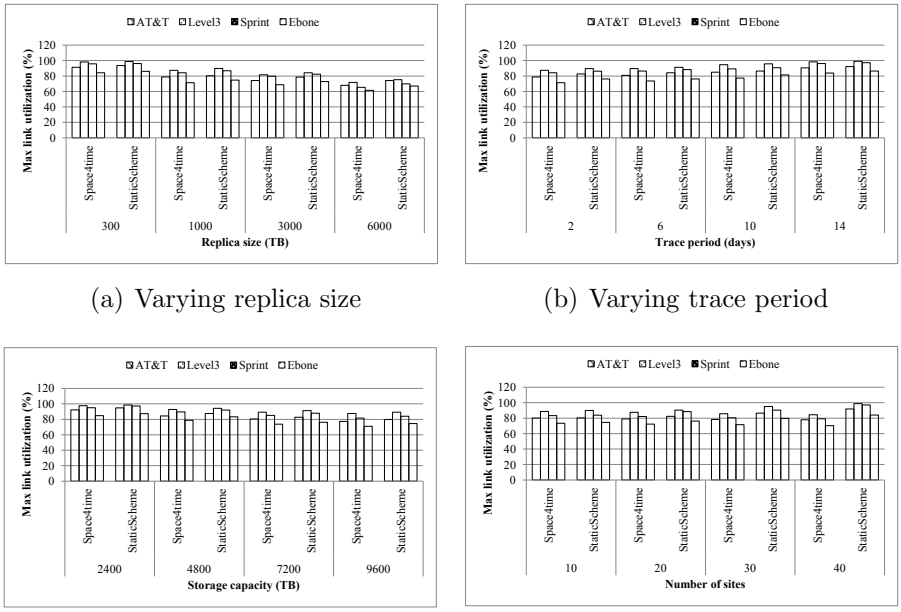
(b) Varying trace period

(c) Varying storage capacity

(d) Varying number of sites

Figure 4: Access latency evaluation over different ISP topologies.

Fig. 4 shows the performance improvement of *Space4time* over *StaticScheme* on average 22.3%. The experimental detail and the effect of different parameters on Space4time performance are as follows. We first vary the replica size from 300 TB to 6,000 TB as shown in Fig. 4(a). Space4time outperforms StaticScheme by 19.3%-71.3% (on average 40.5%), 7.1%-48.2% (on average 21.8%), 2.4%-27.9% (on average 11.1%), 0.5%-4.4% (on average 1.75%) at the replica size as 300 TB, 1000 TB, 3000 TB, 6,000 TB respectively. We then increase the trace period from 2 to 14 days as shown in Fig. 4(b). Space4time outperforms StaticScheme by 19.8%-56.2% (on average 32.0%), 21.3%-58.5% (on average 34.5%), 20.0%-57.2% (on average 33.3%), 18.8%-55.9% (on average 31.6%) at the trace period as 2 days, 6 days, 10 days, 14 days respectively. In the third test shown in Fig. 4(c), we vary storage capacity from 2.4PB to 9.6PB. Space4time outperforms StaticScheme by 0.12%-1.8% (on average 0.63%), 4.14%-22.3% (on average 10.7%), 13.3%-35.8% (on average 22.2%), 25.5%-79.0% (on average 45.2%) at the storage capacity as 2.4 PB, 4.8 PB, 7.2 PB, 9.6 PB respectively. In the forth test in Fig. 4(d), we vary the number of sites from 10 to 40. Space4time out-



(a) Varying replica size

(b) Varying trace period

(c) Varying storage capacity

(d) Varying number of sites

Figure 5: Link utilization evaluation over different ISP topologies.

performs StaticScheme by 0.14%-1.5% (on average 0.6%), 2.9%-17.9% (on average 7.3%), 12.1%-36.4% (on average 20.6%), 24.2%-75.1% (on average 43.8%) at the site number as 10, 20, 30, 40 respectively. In Fig. 4(a), we observe that the performance of Space4time over StaticScheme is improved dramatically when the replica size is small scale. And with the replica size increasing, the performance improvement is not substantial. This is because when the total storage capacity is fixed, the larger replica size means the lease storage capacity for file replication thus leaving little space for improvement. The same pattern is found among in Fig. 4(c) (with larger storage space) and in Fig. 4(d) (with more sites). These two tests also illustrate an important factor that affects performance: the size of the storage space. When we increase the storage capacity, or increase the total number of sites, files have more replications and a user has more choices for site selection. Fig. 4(b) shows that the performance improvement looks stable under varying period. This demonstrates that Space4time scheme works well even with large-scale burst requests.

To better understand the results, we then analyze the effect on traffic with

different schemes in Fig. 5. We plot the maximal link utilization to illustrate the level of congestion in the network. The utilization of link capacities varies with the network traffic load. The maximum link utilization is to illustrate the level of congestion in the network. Higher storage capacity and more sites show more space for potential improvement. We found that their utilization of link capacities made little difference for the two schemes. In Fig. 5(a), we observe that the utilization of link capacities increases with the replica size increasing from 300 TB to 6,000 TB for both schemes. The same pattern is found among in Fig. 5(c) with the storage space increasing. In Fig. 5(b), the utilization of link capacities is on the rise for both schemes, but in Fig. 5(d), the utilization of link capacities shows a downward trend for Space4time. So with more sites, Space4time has a better performance due to reduce traffic.

In summary, Space4time achieves improvement solutions in various problem instances. Most of cases, Space4time improves both access latency and link utilization. Instead of aggregating all requests from each user into one, Space4time processes requests in the order they arrive and assigns requests to a site solely based on blocking probability. This request redirecting policy results in both more storage servers being used and users being assigned to sites with low load. At the same time, our file replication policy results in high hit ration of user requests which leads to lower access latency.

6. Conclusion

The current practice with content distribution in Cloud environment still exhibits room for improvement: content storage and request routing are scheduled separately without joint optimization, which can cause starvation and unfavorable data locality. To this end, we design a novel content distribution and request routing solution, *Space4time*. Based on blocking probability, with full consolidation of storage space, Space4time effectively exploits the storage and network capacity for latency-sensitive applications. From an architecture point of view, Space4time couples a space-time tradeoff and enable us to optimize the use of storage capacity and network traffic to alleviate starvation, and jointly optimizes the data locality for r latency-sensitive applications. Extensive experiments with Space4time demonstrate significant improvements in access latency. Although promising, its realization presents challenges on how to efficiently store and replicate contents / route requests among different CSPs, and to distribute user requests to the appropriate CSPs for timely responses. These challenges escalate when we consider the

persistently increasing contents and volatile user behaviors in mobile Internet applications. By exploiting "space for time", this paper proposes efficient proactive strategies for dynamic scaling of cloud applications.

Acknowledgment

We thank the anonymous reviewers whose comments noticeably improve the quality of this paper. This work is supported in part by the National 973 Program of China under grant $2011CB302301$ and A*STAR SERC, Singapore, under grant $102 - 158 - 0036$.

References

[1] S. V. Rompaey, K. Spaey, C. Blondia, Bandwidth versus storage tradeoff in a content distribution network and a single server system, in: Proc. of the 7th International Conference on Telecommunications (ConTEL), 2003.

[2] H. Ebara, Y. Abe, D. Ikeda, T. Tsutsui, K. Sakai, A. Nakaniwa, H. Okada, A costeffective dynamic content migration method in CDNs, IEICE Transactions on Communications E88-B (12) (2005) 4598–4604.

[3] M. Li, C.-H. Wu, A cost-effective resource allocation and management scheme for content networks supporting IPTV services, Computer Communications 33 (1) (2010) 83–91.

[4] W. Jiang, R. Zhang-Shen, J. Rexford, M. Chiang, Cooperative content distribution and traffic engineering in an ISP network, in: Proc. of SIGMETRICS/Performance, 2009, pp. 239–250.

[5] M. Alicherry, T. Lakshman, Optimizing data access latencies in cloud systems by intelligent virtual machine placement, in: Proc. IEEE INFOCOM 2013, 2013, pp. 647–655.

[6] M. Pathan, R. Buyya, Resource discovery and request-redirection for dynamic load sharing in multi-provider peering content delivery networks, Journal of Network and Computer Applications 32 (5) (2009) 976–990.

[7] D. DiPalantino, R. Johari, Traffic engineering versus content distribution: A game theoretic perspective, in: Proc. of IEEE INFOCOM, 2009, pp. 540–548.

[8] J. Zheng, T. S. E. Ng, K. Sripanidkulchai, Workload-aware live storage migration for clouds, in: Proc. of the 7th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, 2011, pp. 133–144.

[9] T. D. C. Little, D. Venkatesh, Probabilistic assignment of movies to storage devices in a video-on-demand system, in: Proc. of the Fourth International Workshop on Network and Operating System Support for Digital Audio and Video, 1994, pp. 204–215.

[10] C. S. Kim, Y. H. Bak, S. M. Woo, W. J. Lee, O. G. Min, H. Y. Kim, Design and implementation of a storage management method for content distribution, in: Proc. of the 9-th International Conference on Advance Communication Technology (ICACT), 2006, pp. 1143–1147.

[11] J. A. Chandy, Storage allocation in unreliable peer-to-peer systems, in: Proc. of International Conference on Dependable Systems and Networks (DSN), 2006.

[12] M. Roughan, M. Thorup, Y. Zhan, Performance of estimated traffic matrices in traffic engineering, ACM SIGMETRICS Performance Evaluation Review 31 (1) (2003) 326–327.

[13] R. Wartel, T. Cass, B. Moreira, E. Roche, M. Guijarro, S. Goasguen, U. Schwickerath, Image distribution mechanisms in large scale cloud providers, in: Proc. of IEEE Second International Conference on Cloud Computing Technology and Science, 2010, pp. 112–117.

[14] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, I. Stoica, Managing data transfers in computer clusters with orchestra, in: Proc. of the ACM SIGCOMM, 2011, pp. 98–109.

[15] W. Li, E. Chan, G. Feng, D. Chen, S. Lu, Analysis and performance study for coordinated hierarchical cache placement strategies, Computer Communications 33 (15) (2010) 1834–1842.

[16] S. Borst, V. Gupta, A. Walid, Distributed caching algorithms for content distribution networks, in: Proc. of IEEE INFOCOM, 2010.

[17] A. Epstein, D. H. Lorenz, E. Silvera, I. Shapira, Virtual appliance content distribution for a global infrastructure cloud service, in: Proc. of IEEE INFOCOM, 2010, pp. 1–9.

[18] Y. Seung, T. Lam, L. E. Li, T. Woo, Seamless scaling of enterprise applications into the cloud, in: Proc. of IEEE INFOCOM, 2011.

[19] M. Hajjat, X. Sun, Y.-W. E. Sung, D. A. Maltz, S. Rao, K. Sripanidkulchai, M. Tawarmalani, Cloudward bound: Planning for beneficial migration of enterprise applications to the cloud, in: Proc. of ACM SIGCOMM, 2010, pp. 243–254.

[20] Y. Wang, B. Veeravalli, C.-K. Tham, On data staging algorithms for shared data accesses in clouds, IEEE Trans. Parallel Distrib. Syst. 24 (4) (2013) 825–838.

[21] S. Chaisiri, B.-S. Lee, D. Niyato, Optimal virtual machine placement across multiple cloud providers, in: Proc. of IEEE Asia-Pacific Services Computing Conference, 2009, pp. 103–110.

[22] X. Meng, V. Pappas, L. Zhang, Improving the scalability of data center networks with traffic-aware virtual machine placement, in: Proc. of IEEE INFOCOM, 2010, pp. 1–9.

[23] M. Bj¨rkqvist, L. Y. Chen, M. Vukolic, X. Zhang, Minimizing retrieval latency for content cloud, in: Proc. of IEEE INFOCOM, 2011.

[24] J. Zhu, Z. Jiang, Z. Xiao, Twinkle: A fast resource provisioning mechanism for internet services, in: Proc. of IEEE INFOCOM, 2011, pp. 802–810.

[25] Y. Wu, C. Wu, B. Li, X. Qiu, F. C. Lau, CloudMedia: When cloud on demand meets video on demand, in: Proc. of the 31st International Conference on Distributed Computing Systems, 2011, pp. 268–277.

[26] F. Chen, K. Guo, J. Lin, T. L. Porta, Intra-cloud lightning: Building CDNs in the cloud, in: Proc. of IEEE INFOCOM, 2012, pp. 433–441.

[27] U. Sharma, P. Shenoy, S. Sahu, A. Shaikh, A cost-aware elasticity provisioning system for the cloud, in: Proc. of the 31st International Conference on Distributed Computing Systems, 2011, pp. 559–570.

[28] G. Dán, Cache-to-Cache: Could ISPs cooperate to decrease Peer-to-Peer content distribution costs?, IEEE Transactions on Parallel and Distributed Systems 22 (9) (2011) 1469–1482.

[29] J. Broberg, R. Buyya, Z. Tari, MetaCDN: Harnessing storage clouds' for high performance content delivery, Journal of Network and Computer Applications 32 (5) (2009) 1012–1022.

[30] M. Li, C.-H. Wu, A cost-effective resource allocation and management scheme for content networks supporting IPTV, Computer Communications 33 (1) (2010) 83–91.

[31] C. Peng, M. Kim, Z. Zhang, H. Lei, Vdn: Virtual machine image distribution network for cloud data centers, in: Proc. of IEEE INFOCOM, 2012, pp. 181–189.

[32] H. Wang, F. Wang, J. Liu, J. Groen, Measurement and utilization of customer-provided resources for cloud computing, in: Proc. of IEEE INFOCOM, 2012, pp. 442–450.

[33] Y. Wu, C. Wu, B. Li, L. Zhang, Z. Li, F. C. Lau, Scaling social media applications into geo-distributed clouds, in: Proc. of IEEE INFOCOM, 2012, pp. 684–692.

[34] J. Dai, Z. Hu, B. Li, J. Liu, B. Li, Collaborative hierarchical caching with dynamic request routing for massive content distribution, in: Proc. of IEEE INFOCOM, 2012, pp. 2444–2452.

[35] M. Levin, Storage management disciplines are declining (June 2006).
URL `http://www.computereconomics.com/article.cfm?id=1129`

[36] I. Baev, R. Rajaraman, C. Swamy, Approximation algorithms for data placement problems, SIAM Journal on Computing 38 (4) (2008) 1411–1429.

[37] N. Spring, R. Mahajan, D. Wetherall, T. Anderson, Measuring ISP topologies with Rocketfuel, IEEE/ACM Transactions on Networking 12 (1) (2004) 2–6.

[38] Rocketfuel maps and data.
URL `http://www.cs.washington.edu/research/networking/rocketfuel/`

[39] Geolite city database by maxmind.
URL http://www.maxmind.com/

[40] M. Zink, K. Suh, Y. Gua, J. Kurose, Characteristics of YouTube network traffic at a campus network - measurements, models, and implications, The International Journal of Computer and Telecommunications Networking 53 (4) (2009) 501–514.