Content Routing and Lookup Schemes using Global Bloom Filter for Content-Delivery-as-a-Service

Yichao Jin and Yonggang Wen
School of Computer Engineering
Nanyang Technological University
{yjin3, ygwen}@ntu.edu.sg

Abstract—Leveraging cloud computing technology, we have proposed content-delivery-as-a-service (CoDaaS) to distribute user generated content (UGC) in an efficient and economical fashion. However, due to the exponential increases of Internet traffic, traditional hashing-based content routing and lookup scheme suffers from high delay. This paper introduces a global compressed counting bloom filter (CCBF) into CoDaaS to address this issue. The global CCBF adds our system with the capability to early check the existence of any specific content among all the peering surrogates, before any local checking on each cache node. Using this global CCBF, we propose two content routing and lookup schemes (parallel and cut-through schemes) to reduce the delay for better user experience. These results provide a fundamental basis to understand the benefits of using bloom filter for content routing and lookup, and offer operational guidelines to optimize the design of CoDaaS for best possible user experience.

I. INTRODUCTION

The rapid growth of user-generated content (UGC) on the Internet stresses existing content distribution networks (CDNs), and its unique characteristics require novel schemes for the processing and distribution. UGC is currently one of the fastest growing forms of contents [1]. In 2008, 42.8% of Internet users in the U.S contributed to the UGC, and this ratio is expected to reach 51.8% by 2013. UGC, due to its long-tail nature and unique features (e.g., conversational media, social interaction, etc), poses additional challenges for its delivery. On the other hand, the leading CDNs service providers (e.g., Akamai, Limelight, etc), tailor their network architecture and operational structure towards popular contents. As a result, it is difficult to distribute heavily long-tailed contents over CDNs in a profitable way. We have previously proposed a novel architecture, Content-Delivery-as-a-Service (CoDaaS) [2], [3], by leveraging cloud computing technologies, to distribute UGCs in the most economic manner.

In addition to the cost objective, the proposed CoDaaS also aims to provide the best possible user experience to the content consumers. One of the quality-of-service (QoS) objectives is to minimize the request response time [4]. The response time basically depends on two processes in content distribution, including content route and content lookup. In traditional CDNs [5]–[7], these two processes are normally executed in sequential order, which directly translates into poor QoS in terms of high lookup latency and consequent long response time. In CoDaaS, we seek to reduce the request response time by parallelling these two processes.

In this research, we introduce a global bloom filter into CoDaaS to minimize the request response time. Specifically, the global bloom filter is maintained by all the surrogates in CoDaaS, to represent the existence of cached contents. It provides the capability to make an early decision on whether the requested content is cached in the CoDaaS before the request is routed to the designated server. Compared to traditional CDNs, our approach executes content route and lookup processes in parallel. As a result, the request response time can be reduced, providing an improved user experience.

Our contributions are multi-fold, including:

• We propose two content routing and lookup schemes (i.e., parallel and cut-through), using the global bloom filter to reduce the response time.
• We introduce a mathematical model to justify the potential gain of our proposed content routing and lookup schemes, and analyze their limitations.
• We quantify the performance gain and the overhead of our proposes schemes via a system-level simulation. Our numerical results suggest that, for light-loaded traffic (e.g., Twitter-like applications), the average request response time for our proposed content routing and lookup schemes can be reduced by up to 65.2%, compared to the sequential one. Our simulation results also suggest mechanisms to optimize the proposed schemes to reduce the impact of false positive inherent to BF, and false negative and the traffic overhead, resulting from the BF synchronization.

These results provide a fundamental basis to understand the benefits of using bloom filter for content routing and lookup, and offer operational guidelines to optimize the design of CoDaaS for best possible user experience.

The rest of the paper is organized as follows. In Section II, we show some related works. In Section III, we present the system architecture for CoDaaS and the proposed schemes by using bloom filter for content routing and lookup. In Section IV, we introduce a mathematical model to examine the potential gain and the overhead of the content routing and lookup schemes. In Section V, we outline the simulation environment and present the numerical results. In Section VI, we summarize the paper and point out potential future works.
II. RELATED WORK

Pervious researches on content routing design for CDNs mainly focused on the hashing-based approaches. The hashing-based method [5] was proposed to route the content request to desirable surrogate according to the hash value. Another semi-hashing-based scheme [6] was proposed to strike a balance between local hit ratio and cluster hit ratio. Other schemes were also proposed to improve the CDNs performance, including load and proximity-aware routing scheme [7], and decentralized replica-selection for cloud services [8].

Bloom filter (BF) [9] was also used intensively for network applications. Cache digest [10] uses BF to indicate the existence of local cached replications. L. Fan et al. introduced counting bloom filter (CBF) for web cache sharing to improve the Internet Cache Protocol (ICP) [11]. BF also was used to route content within Internet routers [12].

Our work differentiates from these pervious researches in several aspects. First, we use BF to jointly optimize the two processes of content routing and lookup for content delivery. Second, our design objective is to minimize the request response time, which can be translated into a better QoS. Finally, we introduce a mathematical framework to justify the validity of our proposed scheme and the performance gain is verified by a system-level simulation over CDNsim [13].

III. SYSTEM OVERVIEW

In this section, we present a schematic overview of the Content-Delivery-as-a-Service (CoDaaS) architecture, and then explain the concept of bloom filter. Finally, we propose two alternative schemes, based on the usage of a global bloom filter, to improve the content routing and lookup scheme.

A. CoDaaS Architecture

The CoDaaS architecture consists of a media cloud, content providers and consumers. The media cloud is made of a list of interconnected virtual machines (VMs), which are instantiated in geographically distributed data centers. The set of VMs can form a CDN overlay dynamically to cache and render contents. Content providers can publish their contents in origin servers with a specified service level agreement (SLA). The content consumers experience a better QoS with the controlled content distribution. The resources consumed by the CDN overlay can be scaled up and down to meet application demand, thus reducing the total cost of ownership. More details about CoDaaS can be found in [2], [3].

In Figure 1, we illustrate the joint process of content routing and lookup for CoDaaS. When the consumer requests a content, the request, after entering the CoDaaS system, will be routed from its ingress point to the designated server. The designated server will do a content lookup in its local storage to determine whether the content has been cached. In case of a cache hit, the content will be served from the designated server to the users. In case of a cache miss, the content will be retrieved from the origin server and then served to the users. In CoDaaS, we aim to optimize these two processes to reduce the response time, by introducing a global bloom filter.

![Fig. 1: Content routing and lookup scheme for CoDaaS](image)

B. Global Bloom Filter

1) Bloom Filter and Its Variation: In this paper, we use a compressed counting bloom filter, which is a variation of standard BF, for content routing and lookup in CoDaaS.

A standard bloom filter (BF) is a bit array of $m$ bits to represent a set of $n$ elements $S$, by using $k$ independent hash functions $h_1, h_2, \ldots, h_k$. For each element $s$ in $S$, it sets each $h_i(s), i = 1, 2, \ldots, k$ bit to 1 for insertion. To check the existence of any element $x$, it checks whether all the $h_i(s), i = 1, 2, \ldots, k$ bits are 1. Due to the hashing collision, a false positive [9] may occur with the probability as

$$f_{bf} \approx 1 - e^{-kn/m}.$$  \hspace{1cm} (1)

The counting bloom filter (CBF) is derived from the standard BF. It uses $m$ fixed size integers instead of $m$ single bits for presence. For an insertion or deletion operation, the corresponding counters increase or decrease by 1. Thus, CBF provides the deletion capability. In this work, we use four bits counters to construct our CBF, which has been shown to be sufficient for most network applications [11].

To reduce the transmission size of CBF as a message, compressed counting bloom filter (CCBF) is used. This work adopts the multi-layer compressed counting bloom filter (ML-CCBF) [14] that uses the run-length code to encode CBF messages. This encoding process can save up to 50% of the transmission traffic. Moreover, we also use a Delta compression scheme to further reduce the transmission size by only transferring the changes for each layer in our CCBF.

2) Synchronization Schemes: Synchronization is a process of establishing consistency of the global bloom filter among all the surrogates by exchanging sync messages. In this research, we consider two alternative trigger schemes for CCBF synchronization. One is the periodic update mode, in which each surrogate broadcasts its update in a regular interval. The other is the event-driven update mode, in which the broadcast is triggered by some predetermined events (e.g., the reception of a fixed number of requests). These two sectionalization modes will have different impact on the performance gain and the overhead, as verified in Section V.

C. Workflows for Content Routing and Lookup Schemes

In this paper, as illustrated in figure 2, we consider three alternative content routing and lookup schemes for CoDaaS.
Figure 2-(a) illustrates the workflow of the traditional scheme, in which content routing and lookup processes are executed sequentially. In a DHT-based scheme, the routing process takes on average \([\log_2 N]\) hops to reach the designated server, where \(N\) is the active number of the surrogates in CoDaaS. The content lookup process starts when the request is routed to the designated server. For the cache hit case, the designated server immediately replies to the user. Otherwise, it will retrieve the content from its origin server and then render it to the consumer.

The key idea in this research is to use a global CCBF to determine whether the request content has been cached in CoDaaS, at the ingress point of each request. This early decision offers a chance for CoDaaS to retrieve the content in parallel with the request routing. Using this insight, we propose two content routing and lookup schemes for CoDaaS, including a parallel scheme and a cut-through scheme. Figure 2-(b) illustrates the workflow of the parallel scheme. Upon a cache miss, the ingress redirects the request from the user directly to the origin server. At the same time, the request is routed to the designated server, which will retrieve a copy of the content directly from the original server for caching purpose. This scheme reduces the request response time significantly by cutting through the CoDaaS system; however, it increases the traffic load to the origin server.

In next two sections, we will compare the performance of these three schemes analytically and numerically.

### IV. Mathematical Formulation

In this section, we develop a mathematical model to compare the performance of the aforementioned three schemes for content routing and lookup in CoDaaS.

#### A. Latency for Content Routing and Lookup

1) System Stability: We first ensure the system is stable to keep every service queues are finite.

In CoDaaS, each surrogate may either be the ingress, relay or the designated server to serve the requests. When surrogate \(i\) works as an ingress or a relay, it checks the routing table to determine the next hop for current requested content. When surrogate \(i\) works as the designated server, it checks its own storage with the service rate \(\mu_i\) to determine whether there is a cache hit or miss, denoted as \(p_c\) (0 for cache miss and 1 for cache hit). We refer \(\mu_o\) as the serving rate at origin server, and \(C_o\) as the bandwidth capacity at origin server. The stability condition for the original server is given by,

\[
\sum_{c} \lambda_i^c < \min(\mu_o, C_o).
\]  

In next two sections, we will compare the performance of these three schemes analytically and numerically.

#### 2) Service Components for Content Routing and Lookup:

As indicated in figure 3, there are several service components involved in the content routing and lookup process. They serve as the foundations to calculate the expected request response time under a chosen scheme.
The routing process directs the requests from the ingress to their designated server. It can be viewed as a M/M/1 queue with the mean arrival rate $\lambda_i^c$ and the mean service rate $\mu_{ij}$ at surrogate $i$. Thus, we have the mean routing latency $T_r$ as,

$$E[T_r] = \frac{\log_2 N}{\sum_{j} \mu_{ij} - \sum_{c} \lambda_i^c}. \quad (4)$$

The lookup process occurs in the designated servers for each request. It can be modeled as a M/M/1 queue with the service rate $\mu_i$ at surrogate $i$. Assuming the hash functions are well designed, the contents must be evenly distributed among all the surrogates. Thus, the mean content lookup delay $T_l$ is,

$$E[T_l] = (\mu_i - \frac{1}{N} \sum_{j} \sum_{c} \lambda_i^c)^{-1}. \quad (5)$$

For the cache miss cases, the content retrieve process, which also constructs a M/M/1 queue, is required at the origin server. Assuming $\alpha$ is cache hit rate and $1 - \alpha$ is cache miss rate, we have the expectation of the content retrieve time $T_o$ as,

$$E[T_o] = (\mu_o - (1 - \alpha) \sum_{j} \sum_{c} \lambda_j^c)^{-1}. \quad (6)$$

Finally, there exists a M/M/1 queue when designated surrogates return the content to users only after all the previous requests are served. Thus, we have mean response time $R_c$ for traditional scheme as,

$$E[R_c] = \alpha E[T_o] + (1 - \alpha) E[T_r] + E[T_l] + E[T_o]. \quad (8)$$

The parallel scheme parallels the routing operation with the content routing and lookup process as shown in figure 4-(b). Hence, for a cache miss, $T_c$ could be saved to make the response. However, the overhead of this operation includes both false positive $f_p$, false negative $f_n$ and the extra sync messages. The mean response time $R_p$ for this scheme is,

$$E[R_p] = (\alpha + f_p - f_n)E[T_o] + \frac{f_n E[T_l]}{\alpha + f_p - f_n} + (1 - \alpha - f_p + f_n)E[T_o] + E[T_o]. \quad (9)$$

The cut-through scheme further cuts the content return time from designated surrogate by redirecting requests to the origin server for cache misses as indicated in figure 4-(c). Thus the mean response time $R_c$ for cut-through scheme is,

$$E[R_c] = (\alpha + f_p - f_n)(E[T_o] + E[T_l]) + \frac{f_n E[T_o]}{\alpha + f_p - f_n} \quad (10)$$

$$+ (1 - \alpha - f_p + f_n)E[T_o].$$

Nevertheless, the cut-through scheme doubles the traffic load at origin server for cache misses. Therefore, it may cause congestion, which can further lead to severe delay. Let $\eta$ be the corresponding congestion factor [15], and $S_c$ be the size of requested content. To avoid the congestion at the origin server, the following inequality must be satisfied,

$$2(1 - \alpha - f_p + f_n)E[S_c] \sum_{i} \sum_{c} \lambda_i^c < \eta S_c. \quad (11)$$

### B. Overhead of Importing Global CCBF

1) False Positive & False Negative: Two factors may lead to false positive, including the nature of CBF and the inconsistency problem when the cached content has been replaced. Since in CoDaaS, the cached content must be the most popular ones, and the requests follow zipf law [18], the false positive caused by inconsistency is negligible. As a result, the false positive $f_p$ is roughly equal with $f_{bf}$ as expressed by Eq. (1).

The false negative is only introduced by the consistency issues when each surrogate synchronize the global CCBF. Thus, it is affected only by the frequency of synchronization.

2) Synchronization Messages: Apart from the congestion that may occur at the origin server, the sync messages could also introduce jams inside the CoDaaS. For each surrogate $i$, its outcoming traffic consists of sync messages, content transfer and request forwarding. The overall traffic must not exceed its bandwidth capacity to avoid the traffic congestion.

For the parallel scheme, the content transfer occurs for both cache misses and hits. And for cache misses, this traffic will be doubled. Let $\lambda_i$ as the sync rate, $S_f$ as the size of CCBF, and $S_c$ as the size of content request. The following inequality should be satisfied to avoid congestion,

$$\lambda_i E[S_f] + ((2 - \alpha - f_p + f_n)E[S_c] + E[S_c]) \sum_{i} \sum_{c} \lambda_i^c < \eta \sum_{c} S_c. \quad (12)$$

Similarly, for cut-through scheme, the content transfer from surrogate to customers only happens for cache hits. Thus, the
congestion free condition for cut-through scheme is given by:

\[ \lambda_i \mathbb{E}[S_i] + ((\alpha + f_p - f_n)\mathbb{E}[S_e] + \mathbb{E}[S_i]) \sum_{\forall i} \lambda_i^c < \eta \sum_{\forall j} C_{ij}. \]  

(13)

V. SIMULATION AND RESULTS

This section provides the details on simulation methodology and settings. Then the analysis on the results are presented.

A. Simulation Methodology and Setup

1) Simulation Tool: We use CDNsim [13] to build our simulation environment. CDNsim is a discrete event simulation tool based on OMNeT++ library designed for CDNs. We modify some basic modules to fit our implementation and add the three content routing and lookup policies into this tool.

We simulate \( N = 50 \) homogenous VMs as geographically distributed surrogates to construct the media cloud inside the CoDaas. All the surrogates are coordinated by a DHT ring using Chord protocol [16]. The modulo function is used to hash the identification of both contents and surrogates. The cache capacity of each surrogate is 2% of the total size of the objects in the origin server. And the cache replacement policy is Least Recently Used (LRU). We use GT-ITM [17] to generate a real Internet topology model, Transit-Stub model with 1008 routers dispersed at different areas. There is only one origin server, which hosts 50000 unique content objects with the total size of approximately 5GB retrieved from a social website. The service capacity is 500reqs/s for each surrogate, and 1000reqs/s for the origin server. We assume the size of each request occupies 60 Bytes (i.e., \( S_{req} = 480\text{bits} \)).

Besides, we define the link capacity among all the nodes as \( C_o = 100\text{Mbps} \), and \( \sum_{ij} C_{ij} = 100\text{Mbps} \) for surrogate \( i \).

We generate requests from 100 client groups with the mean interval time at 0.01 second (i.e., \( \sum_{\forall i} \sum_{\forall c} \lambda_i^c = 100\text{reqs/s} \)). The distribution of the interval time of all the request follows exponential distribution, and distribution of requested content follows zipf law [18] to make the access pattern much closer to realistic ones. In this setting, the stability of our CDN system can be ensured by satisfying both Eq. (2) and (3).

The hash functions of the bloom filter are built by dividing a 128-bits MD5 signature for each URL into four 32-bits words, and the modulus of each word by the number of counters are the hash values. The maximum volume of the counter bloom filter is set at 20Kb (i.e., 5000 counters) which is a reasonable size that can keep the false positive rate below 1% for zipf distributed enquiries. We compress the global counting bloom filter using ML-CBHF [14]. And the delta compression algorithm is used to compress the sync message to further reduce the transmission size.

B. False Positive & False Negative

Figure 5 shows the relationship between synchronization frequency and false positives/negatives. We implement both periodical and event-driven sync schemes. The results indicate that the two schemes present almost the same trend. This can be attributed to the fact that for a given incoming pattern, the expectation for the inter-arrival time is fixed.

As the interval increases, the false negatives increase accordingly, while the false positives maintain at a certain level as discussed in section IV-B. When the sync frequency is extremely high (i.e., one sync for every single request), there is no false negatives at all. But this setting could generate intensive synchronization traffic. When the frequency is higher than once per 400 requests for event-driven scheme or once per 100 seconds for periodical scheme, the false negatives can be controlled below 1%. And in this case the traffic load could also be kept at a relatively low level.

C. Improvement on User Perceived Response Time

We simulate two scenarios to test our system. One is that the popularity of an object is inversely proportional to its size, that is, the smallest content is most popular. The other one is that the largest content has the highest popularity. Hence, the average size of requested content \( S_c \) for these two scenarios is around 32Kb and 170Kb respectively, to simulate different level of traffic load on data plane. The sync frequency for the global bloom filter is set at once per 400 requests using the event-driven scheme. Various cache hit rates are set by initializing the global bloom filter and the cached content in each surrogate. This simulation lasts for 1000 seconds. The results are presented by figure 6.

In the first scenario, the average response time of cut-through scheme is the lowest, while the traditional scheme costs the highest delay among all the three mechanisms, as expressed in Eq. (8) (9) (10). When there is no cache hit, the gap between our proposed schemes and the traditional one is the largest for all the cases. The parallel scheme and the cut-through scheme are able to reduce the mean response time by...
23.6% and 65.2% compared with that required by the sequential one. As the cache hit ratio goes up, the performances of the three schemes get closer, because the workflows are the same for cache hits. Finally, when all the requests are cache hit, the results from the three policies arrive at the same point.

The interesting observation in figure 6-(a) is the mean response time of both traditional sequential scheme and parallel scheme decreases, while the time required by cut-through scheme increase, as the cache hit ratio raises. It indicates that as long as there is no traffic congestion (i.e., Eq. (11) is satisfied), it is advantageous to redirect the missed requests to origin server in terms of minimizing the response time.

In the second scenario, the parallel scheme still needs less response time than that of the sequential scheme for all the cases, while the cut-through scheme suffers from the congestion when the true cache hit rate is lower than 30%. But the cut-through scheme performs most efficiently again when the cache hit rate passes the congestion threshold as derived from Eq. (11). It may motivate us to adopt an adaptive algorithm, which adopts the cut-through scheme when there is no congestion, and dynamically changes into the parallel scheme when the cache miss rate exceed the threshold.

D. Impact of Synchronization Traffic

Figure 7 presents the relationship between the sync frequency and the response time. To investigate the impact of sync traffic on the response time, we experiment on a set of sync frequency based on both event-driven and time-driven to generate different levels of sync traffic. This simulation is under the condition that the popularity of content is proportional to its size, and the cache hit rate is 50%.

Figure 7-(a) shows how the sync frequency of event-driven scheme affects the response time. Specifically, the response time is much longer when the frequency is higher than once per 5 requests for both schemes. The reason can be attributed to the traffic jam inside CoDaaS as discussed in Eq. (12) (13). As the frequency continues to increase, the sync traffic load becomes lighter, but on the other hand, the false negative increases. Such combined effects make the response time lowest at the point of approximately one sync per 400 requests.

Figure 7-(b) shows how the frequency of periodical scheme affects the response time. The response time is constrained by the congestion inside the CoDaaS when the frequency is high (i.e., higher than 10 syncs per second). In contrast, when the frequency is extremely low (i.e., lower than one sync per 1000 seconds), the incurred false negative causes more faking cache misses. This further introduces congestion at origin server, and thus increases the response time.

VI. CONCLUSION AND FUTURE WORKS

In this paper, we propose parallel and cut-through content routing and lookup schemes by using a global compressed counting bloom filter (CCBF). Mathematical models are presented to analyze both the user perceived response time and the overheads of importing this CCBF, including false positive/negative, traffic congestion, and synchronization cost. Simulations are conducted to prove the performance of those metrics. The simulation results match well with our analytic models, indicating our proposed schemes can save up to 65.2% mean response time for light traffic loads.

Our future work will aim to a congestion-aware adaptive algorithm to further improve the efficiency of our content routing and lookup scheme for all the traffic loads. We will also aim to fully make use of VMs in CoDaaS, where the service rate and storage capacity can be dynamically adjusted.

REFERENCES