# TailCutter: Wisely Cutting Tail Latency in Cloud CDN under Cost Constraints

Zeqi Lai*, Yong Cui*, Minming Li†, Zhenhua Li‡, Ningwei Dai*, and Yuchi Chen*
* Department of Computer Science and Technology, Tsinghua University, China
† Department of Computer Science, City University of Hong Kong, China
‡ School of Software, TNLIST, and KLISS MoE, Tsinghua University, China
{*uestclzq, lemondnw, chenycmx*}*@gmail.com*, {*cuiyong, lizhenhua1983*}*@tsinghua.edu.cn, minmli@cs.cityu.edu.hk*

***Abstract*—Cloud computing platforms enable applications to offer low latency access to user data by offering storage services in several geographically distributed data centers. In this paper, we identify the *high tail latency problem* in cloud CDN via analyzing a large-scale dataset collected from 783,944 users in a major cloud CDN. We find that the data downloading latency in cloud CDN is highly variable, which may significantly degrade the user experience of applications. To address the problem, we present TailCutter, a workload scheduling mechanism that aims at optimizing the tail latency while meeting the cost constraint given by application providers. We further design the Maximum Tail Minimization Algorithm (MTMA) working in TailCutter mechanism to optimally solve the Tail Latency Minimization (TLM) problem in polynomial time. We implement TailCutter across data centers of Amazon S3 and Microsoft Azure. Our extensive evaluation using large-scale real world data traces shows that TailCutter can reduce up to 68% 99th percentile user-perceived latency in comparison with alternative solutions under cost constraints.**

## I. INTRODUCTION

Cloud storage services are gaining tremendous popularity in recent years by providing the appealing benefits of low maintenance, easy access and elasticity for geographical online data storage. A recent study [1] shows that the global cloud storage market is expected to reach $56.57 billion by 2019, with a compound annual growth rate of 33.1%. The recent emergence of cloud storage providers such as Amazon S3, Microsoft Azure and Google Cloud Storage (GCS) are notable examples. Cloud storage providers operate data centers that can offer Internet-based content storage and delivery capabilities with the assurance of service uptime and end user perceived service quality.

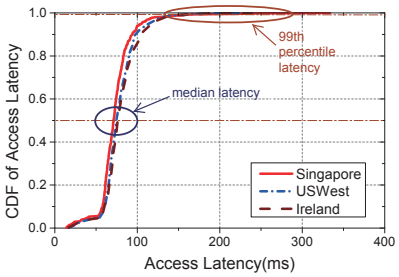
The emergence of cloud storage services provides new opportunities for application providers. On one hand, application providers can build a Content Delivery Network (CDN) serving users without the high cost of owning or operating geographically dispersed data centers. On the other hand, application providers are also able to easily build CDN to provide low-latency service to their users by leveraging the distributed data centers offered by cloud providers. This new breed of CDN based on cloud storage services is referred as "cloud CDN". As the user access latency is one of the most important QoS metrics, a large amount of previous efforts [2]–[13] have been focusing on how to place replicas in different clouds and optimize the latency performance in cloud CDN.

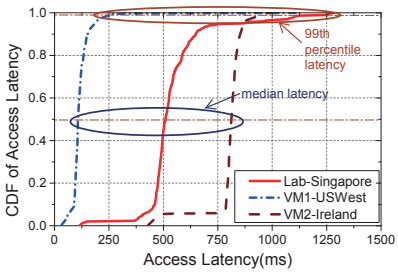
On many distributed systems, fetching content via a request from a single serving node is associated with *high latency variance*. This phenomena, often called tail latency, exists not only in many modern dedicated data centers [14], [15], but also in cloud CDN. The impact of high latency variance is problematic for popular applications where even 1% of traffic corresponds to a significant volume of user requests [16], and for applications where the user is required to download several objects (e.g., web page loading) and the user-perceived latency is constrained by the object downloaded last.

In this paper, we identify, formulate and address a very important but not yet solved problem: *high user-perceived tail latency* in cloud CDN. Our measurement shows that the high latency variance exists both within the cloud data center and over the Internet. Quantitatively, we analyze a recent large-scale dataset collected from a major cloud CDN named Xuanfeng [17], [18]. The dataset records the latency of 4,084,417 file downloads in a whole week, involving 783,944 users and 563,517 unique files. Surprisingly, we find that *the 99th-percentile download latency can be up to 49× of the median!* These variances are caused by many factors, including shared resources competition, equipment failure, etc. [19], and almost impossible to prevent. A feasible solution needs to reduce the serving latency while living with the high variability.

Since we observe that the high latency variance exists in a single cloud, we can leverage multiple cloud data centers to serve user requests and reduce the serving latency. An intuitive approach inspired by [7], [19] is to require a user to handle every data downloading process with a set of redundant requests to different cloud data centers with the assumption that data is replicated in each cloud, so that the earliest response can be used. As our measurement shows in Section II, leveraging multiple cloud data centers to download data is effective in reducing the user-perceived tail latency in cloud CDN. However, cloud storage providers charge their customers by their bandwidth usage, and application providers often have a budget for using cloud storage services. Issuing multiple redundant requests inevitably involves additional traffic overhead. Therefore, although the approach that issues a set of requests to different data centers is able to cut high tail latency, the concrete approach for a user to download data from different clouds, e.g. which data center a user should

978-1-4673-9953-1/16/$31.00 ©2016 IEEE




(a) Latency measured in different data centers. (b) Latency measured in different end-hosts.

Fig. 1. High latency variance within cloud data centers and over the Internet.

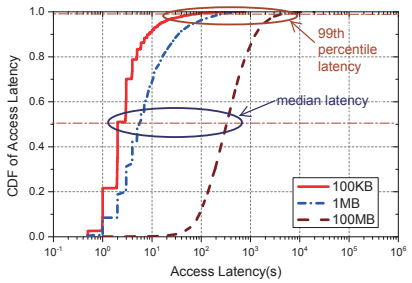



Fig. 2. CDF of access latency when fetching data from the Xuanfeng Cloud CDN.

send GET requests to, and in each request how much data should be downloaded, still needs to be carefully designed to satisfy the cost constraints of application providers.

To address the high tail latency problem in cloud CDN and enable application providers to avail of the low cost benefits provided by cloud services, we propose TailCutter, a novel workload scheduling mechanism lying between users and cloud storage data centers. TailCutter helps to schedule user requests to different data centers and optimize user-perceived latency while satisfying the cost constraint. In high level, TailCutter periodically measures the latency distribution and the workload from every cloud data center to different IP-prefixes. Then TailCutter schedules all user requests and decides the concrete download approach (e.g. how much data should be downloaded from each data center) for every user to download data, according to the latency distribution, the pricing policy of different clouds and the cost constraint of application providers. Finally the schedule results are delivered to each user and then users download the replica they needed accordingly.

More specifically, to efficiently solve the problem, we first formulate the Tail Latency Minimization (TLM) problem that aims at optimizing the tail latency while meeting application provider's cost constraint in cloud CDN. We further design the Maximum Tail Minimization Algorithm (MTMA) to optimally solve the TLM problem in polynomial time. We implement our TailCutter system across a set of data centers in Amazon S3 and Microsoft Azure. Extensive performance evaluation using a 2 Terabytes real-world data trace provided by a major ISP shows that our TailCutter can effectively reduce up to 68% of the 99th percentile user-perceived latency without exceeding the budget of application providers.

In summary, this paper makes the following contributions.

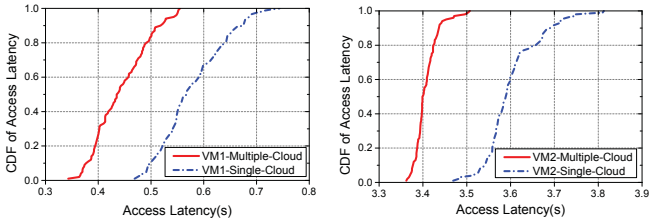
- By measuring and analyzing the latency performance in cloud CDN, we identify the *high user-perceived tail latency problem*, which may significantly degrade user experience but has not been addressed in cloud CDN (§II).
- We formulate the Tail Latency Minimization problem in cloud CDN. To our best knowledge, we are the first to optimize the tail latency under the cost constraints in cloud CDN (§III).
- We propose TailCutter, a novel workload scheduling mechanism involving the Maximum Tail Minimization Algorithm (MTMA) to optimally solve the Tail Latency Minimization problem in polynomial time (§IV).
- We implement TailCutter over multiple cloud data centers of different providers, and extensive evaluation using a 2 TB real-world data trace collected from a major ISP demonstrates the effectiveness of TailCutter on reducing user-perceived tail latency (§V).

## II. Measurement and Motivation

We begin with a measurement study to motivate our work. In this section, we 1) quantify the *high tail latency* in cloud CDN, and 2) introduce and analyze the basic approach that leverages multiple clouds to reduce tail latency. We then highlight the importance of considering cost constraint when optimizing tail latency in cloud CDN.

**Quantifying and analyzing the high tail latency.** We first conduct a lab-scale measurement to quantify the user-perceived access latency, which is defined as the time for a user to download a certain file from a cloud. We select three Amazon S3 data centers in different locations and then measure the access latency 1) from an EC2 VM to the storage in the same data center, 2) from our lab to the nearest S3 data center, and 3) from two end-host VMs out of Amazon in North America and Europe to their nearest S3 data center. In each case, we issue a single GET request to download a 100 KB file in every 10 seconds in one day. We use tcpdump to record the packet-level network traces and calculate the user-perceived access latency. The cumulative distribution function (CDF) of access latency is plotted in Figure 1. We can see that the access latency indeed exhibits a wide spread within the cloud data centers and over the Internet, although the file size is the same across all experimentations. Specifically, the 99th percentile latency is at least 2× and up to 4× of the median. The latency measured in an end-host is higher than the latency measured in a VM inside the cloud data center. This is because the end-to-end access latency is also impacted by the intermediate link over the Internet.

We further collect a large-scale dataset from 783,944 Xuanfeng [17] cloud CDN users. We extract the downloading time and the fetched file size from the dataset, and plot the CDF of access latency when downloading files in around 100KB, 1MB and 100MB respectively. As shown in Figure 2, surprisingly the 99th percentile latency of the 1MB file is 49× of the median latency. The user-perceived tail latency is much




(a) Latency variance in VM1 (b) Latency variance in VM2

Fig. 3. High variance in user-perceived end-to-end latency.

higher in a large-scale commercial cloud CDN as compared to our lab-scale experiment, because these latency variances are caused by many complex factors, including shared resources competition, equipment failure, etc., and are almost impossible to prevent.

High latency variance can significantly degrade the application performance. Worst-case performance matters much more to applications like the Web that requires excellent user experience [14]. The *high user-perceived tail latency problem* we observed in cloud CDN is very problematic because for popular applications only 1% of their traffic corresponds to a significant volume of requests [16]. Besides, for applications where a single request issued by a user requires the application to fetch several objects (e.g., web page loading), operation completion time is constrained by the object fetched last. *Therefore, we need a latency optimization solution in cloud CDN that can address the latency variance over the Internet and within the cloud service's data center network.*

**Leveraging multiple clouds to reduce latency.** Since the latency of a single cloud data center suffers from significant variance, a straightforward way to reduce tail latency is to issue multiple requests to different data centers to download data. To identify the effectiveness of using multiple clouds, we place the same 1 MB replica in two Amazon S3 data centers, and set two nearby VMs as clients that need to download the replica. Each VM uses two approaches to download the replica: 1) send a single GET request to the nearest data center to get the entire replica, and 2) concurrently send two redundant requests to different data centers, and the first received response will be used. We repeat the two approaches 100 times and plot the CDF in Figure 3. *The measurement results show that leveraging multiple cloud data centers is indeed an effective approach that can reduce the latency variance and decrease the user-perceived latency.*

**Cost-Effectiveness considerations.** To use cloud storage services, application providers pay money to cloud providers according to the bandwidth and the storage usage of their applications. Generally the bandwidth cost is much higher than the storage cost. Therefore application providers may have a cost constraint for using cloud storage services. Cloud data center with lower serving latency may be more expensive. *Although we can leverage multiple clouds to reduce tail latency in cloud CDN, the concrete approach for all users to download data should be well considered to meet the cost constraint of application providers.*

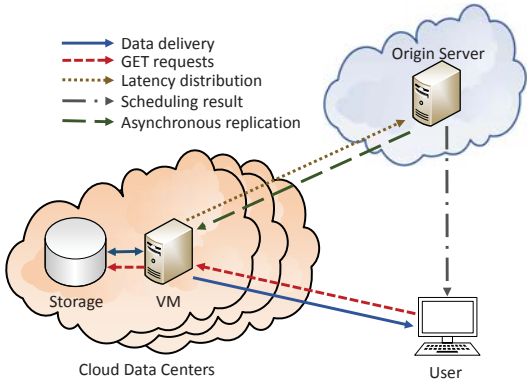



Fig. 4. Problem scenario.

## III. Problem Formulation

In this section we introduce the problem scenario and formulate the Tail Latency Minimazation (TLM) problem across multiple clouds. As shown in Figure 4, the application provider owns a origin server to store the original data. Replicas of the original data are pushed to different cloud data centers asynchronously. In each data center a VM is placed in front of the storage and can be used to measure the cloud latency distribution and the workload in a cloud data center. To download data the user can issue a set of GET requests to multiple cloud data centers.

In reality, a cloud provider owns and operates multiple data centers, and each data center belongs to only one provider. We assume that there are $M$ cloud data centers $C = \{C_1, C_2, ..., C_M\}$ indexed by $i$, and $N$ end users $U = \{U_1, U_2, ..., U_N\}$ indexed by $j$. Since Internet latencies to any particular data center are similar from all end-hosts in a same prefix [20], we define the term "user" here as a set of clients from the same IP prefix.

For ease of exposition, we assume that time is slotted and each slot lasts for $\tau$ seconds. There are $K$ slots in a scheduling period and each one is indexed by $k$. As we have mentioned previously, the serving latency of each cloud data center inevitably suffers from high variance and the available bandwidth changes overtime. Let $u_{ik}$ denote the total available bandwidth of cloud $i$ in slot $k$. Besides, the available end-to-end bandwidth between a user and a cloud is also limited by other factors, e.g. the physical distance or RTT. Therefore we use $d_{ijk}$ to denote the bandwidth limit between cloud $i$ and user $j$ in slot $k$. Assume that the size of replica is $W$ MB. A replica is split into multiple $\omega$ MB small data units, and a GET request downloads a certain number of data units. Let $x_{ijk}$ denote the amount of data unit that user $j$ downloads from cloud $i$ in slot $k$. User $j$ downloads $\sum_{i=1}^{M} x_{ijk}$ data units in slot $k$ and $\sum_{i=1}^{M}\sum_{k=1}^{K} x_{ijk}$ data units in total. The amount of data units delivered by cloud $i$ in slot $k$ is $\sum_{j=1}^{N} x_{ijk}$. The goal of the Tail Latency Minimization problem is to determine how to

assign user requests to different cloud data centers to download data and minimize the maximum user-perceived latency while meeting the given cost constraint.

Cloud providers will charge for their outgoing traffic. For delivering content to end users via GET requests, cloud $C_i$ charges unit price of $G_i$ per data unit for outgoing traffic. In practice, some cloud providers (e.g. Amazon) charge the bandwidth cost according to both the number of GET requests and the amount of delivered traffic size. In our model, we assume that each download operation to a cloud data center is operated in a single GET request. Therefore, the total cost overhead in a scheduling peroid is $\sum_{i=1}^{M}\sum_{j=1}^{N}\sum_{k=1}^{K} G_i \cdot x_{ijk}$. In our problem, we focus on the bandwidth cost because it is dominant in the overall cost of building a cloud CDN.

Let the binary variable $y_{jk}$ denote the transmission state of user $j$ in slot $k$. $y_{jk}$ is 1 if and only if user $j$ has not finished downloading data in slot $k$. Note that user $j$ may pause transmission and not download any data in slot $k$ but $y_{jk}$ is still 1. Because all users start to download data in the first slot, $y_{jk}$ is a non-increasing value over time. Therefore, the user perceived latency of user $j$ can be formulated as $\sum_{k=1}^{K} y_{jk}$.

In summary, assume that in a scheduling period the total cost constraint given by the application provider is $f$ and the access frequency from user $j$, which is defined as the total times that user $j$ requests to the same file in all $K$ slots, is $R_j$. Our TLM problem can be formulated as follows:

$$\textbf{min}\{\textbf{max}\{\sum_{k=1}^{K} y_{jk}\}\} \tag{1}$$

subject to:

$$\sum_{i=1}^{M}\sum_{k=1}^{K} x_{ijk} = \frac{R_j \cdot W}{\omega}, \forall j \in U \tag{2}$$

$$\sum_{i=1}^{M}\sum_{j=1}^{N}\sum_{k=1}^{K} G_i \cdot x_{ijk} \leq f \tag{3}$$

$$\sum_{j=1}^{N} \omega \cdot x_{ijk} \leq u_{ik} \cdot \tau, \forall i \in C, k \in K \tag{4}$$

$$\omega \cdot x_{ijk} \leq d_{ijk} \cdot \tau, \forall i \in C, j \in U, k \in K \tag{5}$$

$$\sum_{i=1}^{M} x_{ijk} \leq \frac{R_j \cdot W}{\omega} \cdot y_{jk}, \forall j \in U, k \in K \tag{6}$$

$$\sum_{i=1}^{M} x_{ijk} > y_{jk} - y_{j(k+1)} - 1, \forall j \in U, k \in K \tag{7}$$

$$y_{jk} \geq y_{j(k+1)}, y_{jk} \in \{0,1\}, x_{ijk} \in N, \forall j \in U, k \in K \tag{8}$$

The goal of TLM is to find a proper assignment for all requests that minimizes the maximum user-perceived latency among all users. Constraint (2) guarantees that each user downloads enough data from all clouds in a scheduling period. Constraint (3) indicates that the total bandwidth cost should not exceed the cost limit $f$. Constraint (4) and (5) require that in every time slot, the total amount of data delivered by cloud $i$ should not exceed the cloud capacity, and the available bandwidth between a user and a cloud is limited by the end-to-end bandwidth constraint. The specially designed constraints (6) and (7) indicate that a user can download data units from cloud data centers if and only if the transmission does not complete. Constraint (8) guarantees binary value $y_{jk}$ is non-increasing overtime, and $x_{ijk}$ is natural number. The notations used in this paper are summarized in Table I

Our TLM problem is essentially an instance of the Integer Linear Programming (ILP) problem. We implement the ILP formulation and solve it with CPLEX. The typical running time for the solver ranges from minutes to hours which is too long to be practical. This also motivates the need for a more efficient solution.

## IV. TailCutter Mechanism

In this section, we present TailCutter, a novel workload scheduling mechanism that leverages the workload and latency distribution in different cloud centers to wisely schedule requests of users and optimize the tail latency under the cost constraints. Next we first study on a simplified scenario with only two clouds in a single slot, and then address the TLM problem in the general scenario.

### A. Feasibility checking for two clouds in a singe slot

We first solve our problem by considering a simplified version with two clouds in a single slot ($M = 2$, $K = 1$). Since time is slotted and there is only one slot, any feasible solution obtains the minimal latency 1. Therefore our goal in this scenario is to find a feasible solution that has the minimal cost. Our basic idea to find a feasible solution is to greedily let a user download data from a cheaper cloud until the cheaper cloud is fully loaded. Here we assume that for any user $j \in U$ the needed data size $R_j * W$ is lower than the total available bandwidth of user $j$, i.e. $R_j * W \leq (d_{1j} + d_{2j})$. Otherwise there is no feasible solution.

TABLE I
Summary of Notations.

| Term | Definition |
|---|---|
| $C_i$ | Cloud site i |
| $U_j$ | User j |
| $G_i$ | Cost per data unit of $C_i$ |
| $x_{ijk}$ | The amount of data unit downloaded by of $U_j$ from $C_i$ in slot $k$ |
| $\tau$ | Time period of a slot |
| $u_{ik}$ | Bandwidth of cloud site $i$ in slot $k$ |
| $d_{ijk}$ | End-to-end bandwidth between cloud site $i$ and user $j$ in slot $k$ |
| $\omega$ | The size of every data unit |
| $R_j$ | Replica's access frequency of $U_j$ |
| $f$ | The total storage cost constraint |

**Algorithm 1** Greedy Algorithm

**Inputs:** Cloud sites $C$, Users $U$, Cloud capacity $u_i$, End-to-end capacity $d_{ij}$, Request frequency $R_j$, Cost per unit $G_i$
**Outputs:** $x_{ij}$

1: //Without loss of generality we assume that $G_1 \leq G_2$
2: **for all** $j \in U$ **do**
3: $x_{1j} \leftarrow \frac{min\{R \cdot W, d_{1j}\}}{\omega}$, $x_{2j} \leftarrow \frac{R \cdot W - x_{1j} \cdot w}{\omega}$
4: **end for**
5: **if** $\sum_{j=1}^{N} x_{2j} > u_2$ **then**
6: //Cannot find a feasible solution.
7: No feasible solution, return.
8: **else if** $\sum_{j=1}^{N} x_{1j} < u_1$ and $\sum_{j=1}^{N} x_{2j} < u_2$ **then**
9: //A feasible solution with minimal cost is found.
10: return all $x_{ij}$.
11: **else**
12: //$\sum_{j=1}^{N} x_{1j} > u_1$ and $\sum_{j=1}^{N} x_{2j} < u_2$
13: $u_1 \leftarrow u_1 - \sum_{j=1}^{N} x_{1j}$
14: **for** $j = 1 : N$ **do**
15: //Move workload to $C_2$
16: $\delta \leftarrow min\{-u_1, (d_{1j})\}$
17: $x_{1j} \leftarrow (x_{1j} - \delta)$, $x_{2j} \leftarrow (x_{2j} + \delta)$, $u_1 \leftarrow (u_1 + \delta)$
18: **if** $u_1 \geq 0$ and $\sum_{j=1}^{N} x_{2j} \leq u_2$ **then**
19: return all $x_{ij}$.
20: **end if**
21: **end for**
22: No feasible solution, return.
23: **end if**

Algorithm 1 shows the detail of our greedy algorithm. Without loss of generality we assume that the cost of cloud satisfies $G_1 \leq G_2$. To find a feasible solution, we first allocate as much as possible bandwidth of $C_1$ to all users (line 2-4). In this way, the bandwidth allocated to each user is limited by both needed data size ($R_j \cdot W$) and the end-to-end bandwidth constraint $d_{ij}$. After allocation, if $C_2$ is overused now, no feasible solution can be found (line 5-7) because we cannot allocate more bandwidth from the cheaper cloud $C_1$. A feasible solution with the minimal cost is found if both clouds are not fully loaded (line 8-10). However, doing this may cause the workload of $C_1$ to exceed its overall available bandwidth. We then consider the scenario where $C_1$ is fully loaded but $C_2$ is still available We consider to move some workloads from $C_1$ to $C_2$. We update $u_1$ in line 13. Since $u_1$ is the available bandwidth of $C_1$, a negative value indicates overloading. We gradually move some workloads from $C_1$ to $C_2$ (line 14-21) while making sure that the cheaper cloud is fully used. During this process, we are able to find a feasible solution when $C_1$ is exactly full and $C_2$ is not overload (line 18-19). Otherwise, we cannot find a feasible solution. Algorithm 1 can output the optimal solution in $O(N)$ in the two-cloud scenario, where $N$ is the amount of users. However, it is difficult to extend Algorithm 1 to solve the problem where more clouds and time slots are considered.

### *B. Maximum Tail Minimization Algorithm*

We further study the more complex scenario and design a polynomial time optimal algorithm, named *Maximum Tail Minimization Algorithm (MTMA)* to solve the problem with more clouds in multiple slots. The basic approach of our algorithm is to search the optimal solution by repeating the following two steps: 1) construct a flow network graph $G$ according to all inputs, and transform TLM to the MCMF problem; 2) use binary search to find a latency $D$ so that MCMF algorithm on the corresponding graph has a solution while the corresponding cost is exactly less than constraint $f$.

**Transforming to MCMF problem:** We first construct a network flow graph based on the TLM problem and transform our problem to the Minimum Cost Maximal Flow (MCMF) problem. Assume that there is only one single slot. We build a graph $G(V, E)$ as shown in Figure 5, where the vertex set $V$ contains all elements in the cloud site set $C$ and user set $U$, together with a source vertex $s$ and a sink vertex $t$ in $V$. Edges between cloud $i$ and user $j$ have two attributes: 1) the link capacity of the edge, which is set to the end-to-end bandwidth constraint $d_{ij}$, and 2) the cost for every data unit delivered in this edge, which is set to zero. For each edge between $s$ and user $j$, the link capacity is set to $\frac{R_j \cdot W}{\omega}$, and the cost is set to zero. We set the capacity and cost of the edge between cloud $i$ and $t$ to be $u_i$ and $G_i$ respectively. Thus we obtain the network flow graph shown in Figure 5.

Note that in the single slot scenario, our problem is to find a feasible solution with the minimum cost. Our TLM problem is then transformed to the Minimum Cost Maximal Flow problem in Figure 5: given a network graph $G(V, E)$ where each edge has a cost for delivered data units and a capacity, how to assign the number of delivered data unit in every edge to find a feasible solution while minimizing the total cost.

Next we consider the TLM problem in multiple slots. Assume the number of slot is $D$. Following the method introduced above, we construct the network flow graph for the multiple-slot scenario by making copies of cloud data centers for different slots. As shown in Figure 6, we set the capacity and cost of edges linked to these new vertexes following the way in single slot scenario. Let $G_D(V, E)$ denote the graph constructed for $D$ slots, our TLM problem in multiple slots scenario can be transformed to the problem of finding the minimum $D$ guaranteeing that $G_D(V, E)$ has a feasible MCMF solution.

**Searching the optimal solution:** We do a binary search on $D$ to find the minimum $D$ and the corresponding data unit assignment in each edge. Specificaly, we start with a sufficiently large $D_{right}$ as the latency allowed (for example, the maximum latency when we let every user download data from his closest cloud) and then use binary search to find a latency $D$ so that MCMF algorithm on the corresponding

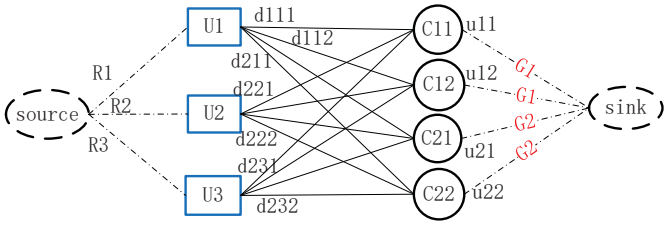



Fig. 5. Problem transformation: single slot.

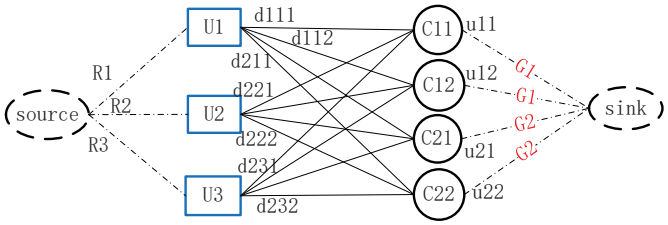



Fig. 6. Problem transformation: multiple slots.

graph has a solution with cost at most $f$ but generates a solution with cost larger than $f$ for $D-1$ copies of cloud data centers.

Algorithm 2 shows how we search the minimum latency that meets the cost overhead constraint. To improve efficiency, we conduct binary search on latency $D$. We set the search scope from zero ($D_{left}$) to the latency of the approach that every user sends request to their nearest data center ($D_{nearest}$). We initialize the latency (line 1-2) and then we construct the flow graph using $D$ as we described above (line 4). We use binary search on $D$ and in every iteration we solve the MCMF problem in the constructed network flow graph (line 3-21). The optimal solution $D$ is found if $cost(D) \leq f$ and $cost(D-1) > f$ (line 11-13). Our MTMA can optimally solve the problem in polynomial time. MTMA conducts a binary search to find the optimal feasible solution and in each iteration we solve the MCMF problem to find a feasible solution.

**Algorithm 2** Maximum Tail Minimization Algorithm

**Inputs:** Cloud sites $C$, Users $U$, Cloud capacity $u_{ik}$, End-to-end capacity $d_{ijk}$, Request frequency $R_j$, Cost per unit $c_i$, Cost constraint $f$

**Outputs:** $D$, $x_{ijk}$

1: $D_{left} \leftarrow 0, D_{right} \leftarrow D_{nearest}$
2: $D \leftarrow \frac{D_{left}+D_{right}}{2}$
3: **while** TRUE **do**
4: Construct the graph $G_D(V,E)$ according to $D$, $C$, $U$, $u_{ik}$, $d_{ijk}$, $R_j$, $c_i$, $f$.
5: Solve $G_D(V,E)$ by MCMF to get the solution $X$.
6: **if** $X$ does not exist **then**
7: $D = \frac{D+D_{right}}{2}$
8: **else**
9: //Find a feasible solution.
10: $cost(D) \leftarrow \sum_{i=1}^{M}\sum_{j=1}^{N}\sum_{k=1}^{K} x_{ijk} \cdot G_i$
11: **if** $cost(D) \leq f$ **then**
12: **if** $cost(D-1) > f$ **then**
13: break, return $x_{ijk}$ and $D$.
14: **else**
15: $D = \frac{D+D_{left}}{2}$
16: **end if**
17: **else**
18: $D = \frac{D+D_{right}}{2}$
19: **end if**
20: **end if**
21: **end while**

## V. Performance Evaluation

Using real world data traces, in this section we evaluate the effectiveness of TailCutter prototype implemented on data centers of several cloud providers. We also conduct extensive simulation to evaluate the performance of TailCutter at scale.

### A. Prototyping

We implement TailCutter server over five data centers across Amazon S3 and Microsoft Azure. To download an object, the user issues a set of requests to multiple cloud data centers following the decision made by TailCutter. In each cloud data center we leverage a VM to call the cloud API given by cloud providers to fetch data from the local storage. Content in the storage of a data center is delivered to the user relying on the VM. Communications between the user and the VM are based on HTTP. We extract the time stamp from network traces captured by `tcpdump` to analyze the latency distribution. At runtime, the VM also measures the access frequency of each replica. Both the latency distribution and workload information are stored in a SQLite database in the VM, and are used for scheduling in every schedule period. We rent 12 VMs in different geographic locations as our users, and implement the TailCutter client on them.

We run our scheduling algorithm in the central origin server according to the measured workload and bandwidth distribution. Before the first scheduling period, replicas are delivered to each cloud data center via the cloud API given by cloud providers. The access tokens of all clouds are stored in a SQLite database in the origin server.

### B. Experimentation setup

In our experiment, the scheduling period $T$ is set to 1 hour and each slot lasts 1 min. We set the cost overhead of each cloud according to the corresponding pricing policy. We use a large scale real-world data trace to evaluate our prototype. Our large traffic flow trace is generated from 99 collection points by a local major ISP on January 10, 2013. The trace data captures about 821 million flow records (about 2 Terabytes). Each record corresponds to the information of one flow which contains the user IP, server IP, flow time stamps, downloading data size but without any personal data.

Next, we first use the real-world data trace to evaluate our TailCutter prototype from three perspectives: 1) its cost-

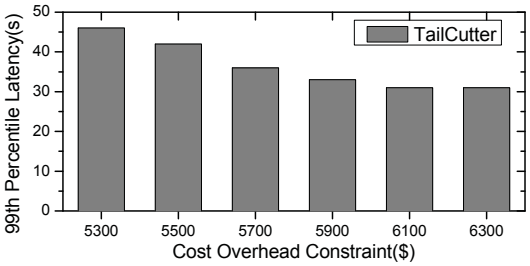
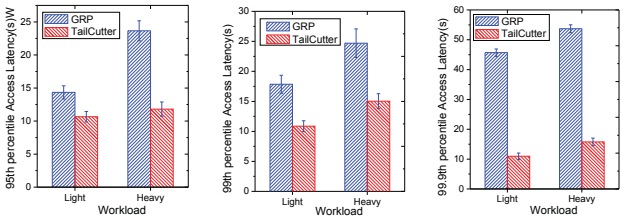



Fig. 7. Cost-efficiency evaluation in satisfying cost constraint.




(a) 95th percentile (b) 99th percentile (c) 99.9th percentile

Fig. 8. Tail latency reduction of light and heavy workload.

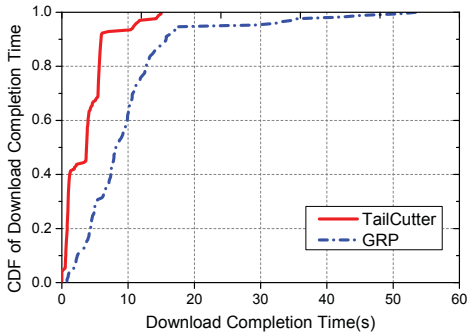



Fig. 9. Verification of TailCutter's ability to reduce latency variance.

effectiveness in optimizing user-perceived latency, 2) its ability in cutting tail latency, 3) its efficiency executing our request scheduling algorithm. We further conduct large scale simulation to demonstrate the effectiveness of TailCutter at scale.

### C. Cost-effectiveness

An application that uses TailCutter incurs additional costs for issuing multiple requests to fetch data from different cloud data centers. Intuitively, given a higher budget, TailCutter fetches more data from the cloud with lower latency but higher cost to optimize the latency performance. We change the overall cost overhead constraint to quantify how TailCutter reduces the maximal user-perceived latency under different budgets. In this case, we randomly pick workload from 12 IPv4 subnetworks (/24) as the input of TailCutter.

Figure 7 depicts the maximal user-perceived latency as a function of the cost overhead constraint $f$. At the higher end of the examined range of budget $f$, TailCutter significantly reduces the tail latency by making more users fetch data from a faster but expensive cloud to speed up data transmission. As the budget $f$ decreases, latency increases because TailCutter tries to find the cheaper clouds to serve users or selects less cloud data centers to process user requests. Because of the lower available bandwidth and resource competition, the user-perceived latency is increased.

### D. Ability to reduce serving latency

TailCutter is able to meet cost overhead constraint and optimize user-perceived latency by properly scheduling user requests to different cloud data centers. We randomly select workload from 12 subnetworks twice, and for each selection the average fetched data size is below or above 512KB to generate light and heavy workload respectively. We then run TailCutter under different scale of workloads and compare the result with GRP [2]. GRP is a user request assignment algorithm that optimizes QoS in cloud CDN. However, GRP ignores the high latency variance within the cloud data centers and over the Internet.

Figure 8 shows the tail latency of TailCutter and GRP with light and heavy workload. We observe that TailCutter's tail latency profile, even at the 99.9th percentile degrades proportionally to the increase in system load. Furthermore, for all 95th, 99th and 99.9th percentile of latency, TailCutter outperforms GRP up to 48%. This is because GRP does not consider the high latency variance within cloud data centers and over the Internet, and statically assign user requests to a single cloud. However, using a single cloud to serve users when the available capacity of a cloud and the end-to-end bandwidth significantly change over time is unable to limit the maximum user-perceived latency among all users. As a result GRP is hard to guarantee low latency especially when the system workload increases.

To evaluate the variance of each algorithm in different workloads, we also plot the CDF of the latency when we use GRP and TailCutter to download data with light and heavy workload in Figure 9. We observe that TailCutter can effectively decrease the user-perceived latency and also reduce the latency variance as compared to GRP.

### E. Running time in practice

We leverage three different algorithms, straightforward solving the ILP problem, TailCutter and GRP to make scheduling decision for all the requests from 12 subnetworks for a scheduling period. We compute the average running time per subnetwork to evaluate their running time in practice. We repeat this for cost constraint ranging from $5000 to $7000, and the result is presented in Figure 10. As shown in Figure 10, solving the original integer linear programming problem by CPLEX to schedule requests of a period consumes more than half hour, which is not feasible in a real system. However, even though TailCutter costs more computation time than GRP, consuming 100s to obtain the scheduling results for the next 1 hour is feasible for a real world workload monitor system.

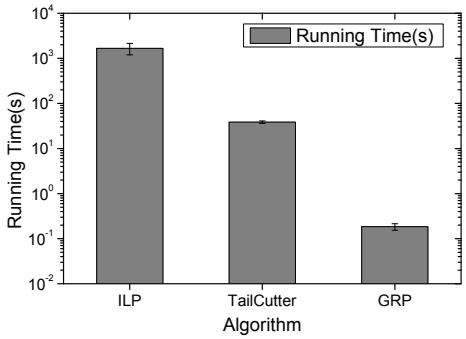



Fig. 10. Efficiency comparison of different request scheduling algorithms.

### F. Simulation at scale

In the above experiments we evaluate our TailCutter prototype and demonstrate the effectiveness to reduce user-perceived tail latency while satisfying the cost constraint. We finally conduct a large scale simulation to evaluate TailCutter at scale. We set 20 cloud data centers and select 4 workloads differing in the amount of subnetworks from our trace data. These 4 workloads generate data respectively. The TailCutter simulator is built using Matlab in around 1000 lines code. We set the bandwidth distribution and the pricing policy following the similar way in our prototype experimentation. Also we compare the results of TailCutter with the GRP algorithm.

Figure 11 plots CDF of the completion time of all requests in different workload scale. The user-perceived latency increases as the scale of workload increases because of more resource competitions. We find that the proportion of low latency request in TailCutter is slightly lower than that in GRP. But TailCutter reduces up to 68% tail latency compared with GRP. This result indicates that TailCutter can wisely schedule user requests to different cloud data centers and avoid long user-perceived tail latency in the network condition where high latency variance happens.

## VI. Related Work

A considerable amount of research has been done on optimizing the performance of cloud storage services.

**Optimizing QoS in cloud CDN.** A lot of research has been done for optimizing the QoS in cloud CDN [2]–[9], [21]–[24]. Authors of [10] and [11] proposed algorithms to optimize total storage and update cost. They used the assumption that requests can be issued from any node, but ignored retrieval cost. Rodolakis et al. [12] added server capacity limitation to the formulation while optimizing storage and retrieval cost. Broberg et al. [13] proposed MetaCDN and designed mechanisms to place content in different cloud storage provider networks and redirect user requests to appropriate replicas. However no replica placement and request redirection algorithm is given. All solutions described above are static in nature in that they simply assume that the link quality between the client and the cloud is constant. They ignore the latency variance within cloud data center or over the Internet.

**Improving cloud data centers.** Several recent studies redesign storage systems and data centers to offer bandwidth guarantees to tenants [25], or to ensure predictable completion times for TCP flows [14], [26], [27]. However, all of their studies require modifications to a cloud service's infrastructure. It is unknown whether and when cloud storage providers will improve their infrastructure to these more complex designs. TailCutter instead satisfies cost constraint for applications deployed on the cloud and optimizes user-perceived latency without having to wait for any modifications to cloud services.

**Reducing variance of cloud services.** The approach of issuing multiple requests to different clouds to reduce tail latency has been considered previously [7], [19], but the focus has primarily been on understanding the implications of redundancy on system load. In contrast, our work is the first to formulate and address the problem of how to issue multiple requests to different cloud data centers to reduce use-perceived latency under the cost constraints.

**Cloud measurement studies.** Previous studies have compared the performance offered by different cloud providers [28] and studied application deployments on the cloud [29]. Moreover, Bodik et al. [30] focused on characterizing and modeling spikes in application workloads. All these previous work complement our work. Furthermore, our measurement in this paper demonstrates that the tail latency problem inside traditional cloud data centers also exists in cloud CDN, and we further highlight that the high tail latency is caused by inevitable reasons within cloud data centers and over the Internet.

## VII. Conclusion

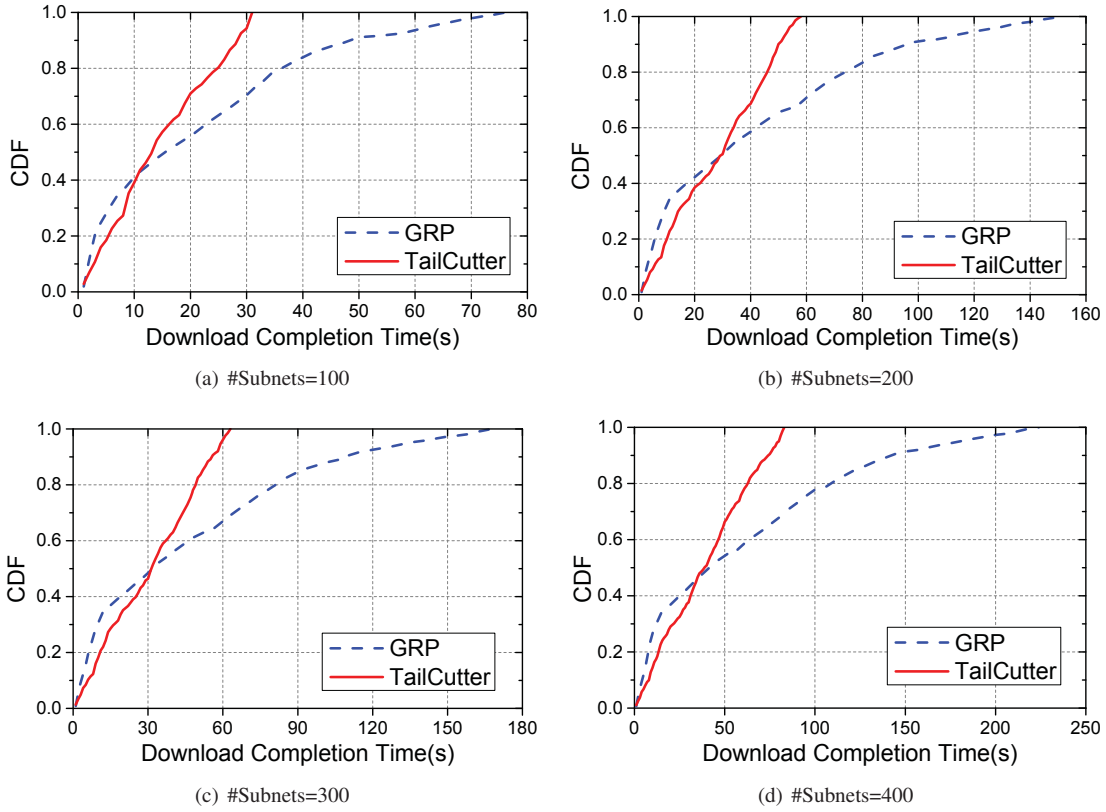
In this paper, we identify, formulate and address the high user-perceived latency problem in cloud CDN. Specifically, we measure and analyze the latency performance of cloud storage services, finding that the high tail latency problem indeed exists in cloud CDN. We then formulate the problem of how to minimize the tail latency in a network condition where high latency variance is inevitable. To address this problem we propose and implement TailCutter, a request scheduling mechanism that leverages multiple requests to different cloud data centers to reduce tail latency. We implement TailCutter in modern cloud providers' data centers and extensive evaluations using real world data traces show that TailCutter is able to cut up to 68% tail latency in cloud CDN.

## Acknowledgment

This research is supported by NSFC under grant 61120106008, 61422206, 61471217 and 61432002, Tsinghua National Laboratory for Information Science and Technology (TNList), and the Research Grants Council of the HK SAR China under grant CityU117913.

## References

[1] "Public/private cloud storage market," http://www.marketsandmarkets.com/PressReleases/cloud-storage.asp.
[2] F. Chen, K. Guo, J. Lin, and T. La Porta, "Intra-cloud lightning: Building cdns in the cloud," in *INFOCOM*. IEEE, 2012.




(a) #Subnets=100 (b) #Subnets=200

(c) #Subnets=300 (d) #Subnets=400

Fig. 11. CDF of user-perceived latency with different system workloads.


[3] Z. Wu, C. Yu, H. V. Madhyastha, and U. Riverside, "Costlo: Cost-effective redundancy for lower latency variance on cloud storage services," in *NSDI*. USENIX, 2015.

[4] G. Joshi, Y. Liu, and E. Soljanin, "On the delay-storage trade-off in content download from coded distributed storage systems," *JSAC*, 2014.

[5] Y. Wu, C. Wu, B. Li, L. Zhang, Z. Li, and F. C. Lau, "Scaling social media applications into geo-distributed clouds," in *INFOCOM*. IEEE, 2012.

[6] D. Wang, G. Joshi, and G. Wornell, "Efficient task replication for fast response times in parallel computation," in *SIGMETRICS*. ACM, 2014.

[7] A. Vulimiri, P. B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker, "Low latency via redundancy," in *CoNEXT*. ACM, 2013.

[8] A. Vulimiri, O. Michel, P. Godfrey, and S. Shenker, "More is less: reducing latency via redundancy," in *HotNets*. ACM, 2012.

[9] K. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter, E. Hyytiä, and A. Scheller-Wolf, "Reducing latency via redundant requests: Exact analysis," in *SIGMETRICS*. ACM, 2015.

[10] X. Tang and J. Xu, "Qos-aware replica placement for content distribution," *TPDS*, 2005.

[11] H. Wang, P. Liu, and J.-j. Wu, "A qos-aware heuristic algorithm for replica placement," in *Proceedings of the 7th IEEE/ACM international conference on grid computing*. IEEE Computer Society, 2006.

[12] G. Rodolakis, S. Siachalou, and L. Georgiadis, "Replicated server placement with qos constraints," *TPDS*, 2006.

[13] J. Broberg, R. Buyya, and Z. Tari, "Metacdn: Harnessing storage clouds for high performance content delivery," *Journal of Network and Computer Applications*, 2009.

[14] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, "Detail: reducing the flow completion time tail in datacenter networks," *SIGCOMM*, 2012.

[15] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," *SIGCOMM*, 2011.

[16] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," in *SIGOPS*. ACM, 2007.

[17] "Xuanfeng offline downloading system," http://xf.qq.com.

[18] Z. Li, C. Wilson, T. Xu, Y. Liu, Z. Lu, and Y. Wang, "Offline downloading in china: A comparative study," in *IMC*. ACM, 2015.

[19] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, 2013.

[20] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "iplane: An information plane for distributed services," in *OSDI*. USENIX, 2006.

[21] Y. Huang, Z. Li, G. Liu, and Y. Dai, "Cloud download: using cloud utilities to achieve high-quality content distribution for unpopular videos," in *MM*. ACM, 2011.

[22] Z. Li, Y. Huang, G. Liu, F. Wang, Z.-L. Zhang, and Y. Dai, "Cloud transcoder: Bridging the format and resolution gap between internet videos and mobile devices," in *NOSSDAV*. ACM, 2012.

[23] Z. Li, G. Liu, Z. Ji, and R. Zimmermann, "Towards Cost-Effective Cloud Downloading with Tencent Big Data," *Journal of Computer Science and Technology*, 2015.

[24] Y. Cui, Z. Lai, X. Wang, N. Dai, and C. Miao, "Quicksync: Improving synchronization efficiency for mobile cloud storage services," in *MobiCom*. ACM, 2015.

[25] H. Ballani, K. Jang, T. Karagiannis, C. Kim, D. Gunawardena, and G. O'Shea, "Chatty tenants and the cloud network sharing problem." in *NSDI*. USENIX, 2013.

[26] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," in *SIGCOMM*. ACM, 2011.

[27] C.-Y. Hong, M. Caesar, and P. Godfrey, "Finishing flows quickly with preemptive scheduling," *SIGCOMM*, 2012.

[28] A. Li, X. Yang, S. Kandula, and M. Zhang, "Cloudcmp: comparing public cloud providers," in *IMC*. ACM, 2010.

[29] K. He, A. Fisher, L. Wang, A. Gember, A. Akella, and T. Ristenpart, "Next stop, the cloud: Understanding modern web service deployment in ec2 and azure," in *IMC*. ACM, 2013.

[30] P. Bodik, A. Fox, M. J. Franklin, M. I. Jordan, and D. A. Patterson, "Characterizing, modeling, and generating workload spikes for stateful services," in *SoCC*. ACM, 2010.