A Geometric Approach to Server Selection for Interactive Video Streaming

Yaochen Hu, Di Niu
Department of Electrical and
Computer Engineering
University of Alberta
{*yaochen, dniu*}*@ualberta.ca*
Zongpeng Li
Department of Computer Science
University of Calgary
zongpeng@ucalgary.ca

***Abstract*—Many distributed interactive multimedia applications, such as live video conferencing and video sharing, require each participating client to transmit its captured video stream to other clients via relay servers. We consider connecting multiple clients through a multiple relay servers and study the server selection problem from a dense pool of CDN edge locations and datacenters to reduce the end-to-end delays between clients. To achieve scalability in the presence of a large number of candidate servers, we formulate server selection as a geometric problem in a delay space instead of in a graph, which turns out to be an extension of the well known Euclidean k-median problem. We propose practical approximation schemes when using only one or two servers with theoretical worst-case guarantees as well as fast heuristics when using k servers. We demonstrate the benefit of our optimized multi-server selection schemes through extensive evaluation based on real-world traces collected from the PlanetLab and Seattle platforms, containing personal mobile devices, as well as real network experiments based on a prototype implementation.**

***Index Terms*—Interactive video streaming; Server selection; End-to-end delay; Geometric optimization.**

I. Introduction

In many interactive multimedia streaming applications, such as live video/audio conferencing and interactive video gaming, each of the geographically distributed clients needs to receive a video/audio stream from all other clients in real time. A common objective of these applications is to reduce the end-to-end delays between clients to the minimum, while maintaining a sufficiently high data throughput, especially when users are spread across different regions [1]–[3]. In production systems like FaceTime and Google+, relay servers [1], [2] are adopted to collect data from all the participating users and distribute a mixed stream to every user. As compared to the earlier peer-to-peer (P2P) solutions, such a server-based solution has several benefits. *First*, each user can upload its data stream at the full rate to the server and does not need to split its uplink bandwidth to serve all other users like in a P2P architecture. *Second*, major application providers such as Apple, Google, and Microsoft usually have their own large server clouds composed of point of presence (POP) locations in their content delivery networks (CDNs) and datacenters, connected in well-provisioned and high-bandwidth backbone networks. *Third*, a large part of mixing and processing jobs can be done by servers, relieving the computational burden of users.

In this paper, we ask the questions—how can the choices of server locations affect end-to-end delays between clients in interactive video/audio streaming applications? Is a single server sufficient, or can we reduce end-to-end delays by using multiple relay servers spread at different locations? And where should these relay servers be placed? As a toy example, if 3 clients in Paris are in a meeting with 3 other clients in Sao Paulo, having two inter-connected servers placed in the two cities, respectively, clearly achieves a lower mean end-to-end network distance than placing a single server at some place in between the two cities. However, how this network-distance phenomenon can affect delay performance in general is yet to be investigated.

Formally speaking, we aim to minimize the mean end-to-end delay in an interactive video streaming session between N clients, by optimally choosing k servers from a dense cloud of CDN nodes (POP locations and datacenters). This problem has been studied in [4] on a graph formed by all the clients and candidate servers, assuming the latency between every pair of nodes is known. It is shown that this new problem is different from the well-known k-median and k-means problems, and is NP-complete for any $k < N$. A greedy algorithm performed on the graph is given in [4] with an approximation ratio of 2 if triangle inequalities are assumed. However, the complexity of this graph-based solution critically depends on the number of candidate servers, which can increase dramatically as the candidate server pool grows to a size of hundreds, thousands or even more. For example, Akamai has deployed a pervasive, highly distributed CDN with over $175,000$ servers in more than 100 countries within over $1,300$ networks [5].

In this paper, we propose a novel geometric approach to multi-server selection in order to achieve scalability in the presence of a large and dense server cloud consisting of CDN nodes and datacenters. By leveraging network coordinate systems [6], each host can be mapped onto a point in a delay space. Using geometric optimization, we compute the *ideal* locations of k servers for N clients in the delay space. We

finally map the computed ideal server locations back to the closest physical servers to form server selection decisions. Unlike graph-based solutions, the complexity of the proposed geometric optimization in the delay space is independent of the number of candidate servers. Moreover, we do not need to measure the latencies from every client to all candidate servers; it suffices to probe a few reference servers to estimate the network coordinate of each client in the delay space, which can be efficiently computed or even pre-computed by many existing network coordinate systems.

It turns out that the proposed geometric problem in the delay space is an extension of the k-median and facility location problems, for which no polynomial-time solution is known [7]. We provide simple approximation schemes for this new geometric problem, and prove that the one-server optimal solution incurs a mean delay of at most $2 - 2/N$ times the best possible value of using N servers forming a full mesh. If two servers can be used, we provide a 2.5-approximation scheme based on centroids, and a one-or-two-server scheme with an approximation ratio of *strictly* less than $2-2/N$. When k ($k > 2$) servers are adopted, we further propose efficient heuristic algorithms to choose k servers based on k-means partitioning and convex optimization.

To evaluate our server selection algorithms under various random errors, including network coordinate errors and mapping errors, we perform extensive simulations driven by latency traces collected from 490 PlanetLab nodes over a 15-day period as well as traces collected from the Seattle platform [8] which include latency measurements from personal and mobile devices. We observe that despite network coordinate errors, triangle inequality violations (TIV), as well as mapping errors from ideal to true server locations, the benefit of our optimized server selection in the delay space can largely offset these errors. Moreover, the proposed server selection procedure is efficient and can finish all the tasks within 1 second for a server pool of close to 500 nodes. To further verify real packet delays including system processing and queueing delays, we have implemented a prototype interactive video streaming system leveraging multiple servers and deployed it on the PlanetLab. We observe that when server locations are optimally determined, not only can multiple servers reduce mean end-to-end delays, but they also help to distribute CPU workloads as the video bit rate increases.

II. Related Work

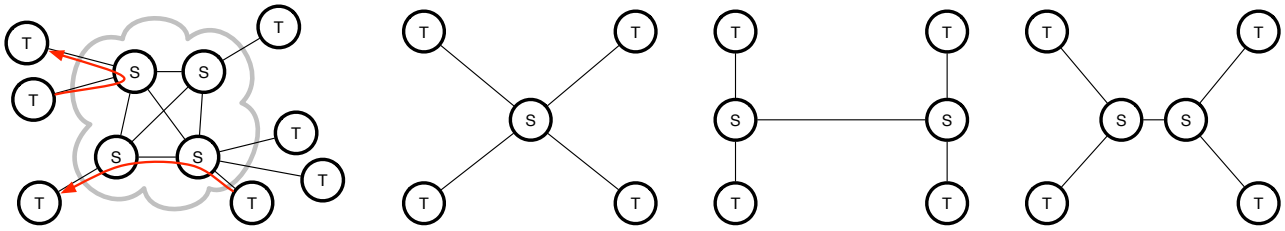
Current production interactive video streaming systems adopt different architectures according to the measurement study in [1]. iChat does not use servers and adopts a P2P star topology. Skype uses centralized servers to relay video traffic, yet with all servers found to be placed in the same location [1]. Google+ adopts multiple servers distributed all over the globe to relay all the data in a session. But its server selection strategy is proprietary and unknown to the public. Other measurement studies have analyzed interactive video streaming with respect to latency, bandwidth and video quality. In [9], the responsiveness of Skype video calls to bandwidth variations is measured. In [10] an extensive measurement of Skype two-party video calls is presented under different network conditions. [11] shows that people will easily get impatient when they face long end-to-end delay. In addition, [12], [13] analyze the overlay architecture, P2P protocol, and VoIP traffic of Skype. In this paper, we provide an in-depth study on server location optimization specifically to minimize delays in interactive video streaming, based on a multi-server mesh topology.

Sever placement and selection have been extensively studied for a variety of applications and topics. Various placement strategies for Web server replicas have been proposed in [14] to improve CDN performance. [15] presents a distributed algorithm that selects game servers for a group of clients in order to minimize the server resource usage with real-time delay constraints. [16] proposes a server selection scheme which can achieve high availability while maintaining low delays and low cost. [17] considers the server allocation problem with dense servers and clients, and has developed an algorithm based on the high-rate vector quantization theory. In contrast, in this paper we specifically focus on selecting the optimal server locations in real-time interactive video streaming applications.

The design of interactive video streaming systems, especially live conferencing, has been extensively studied in the context of P2P networks [18], [19] within a utility maximization framework, in order to optimize the streaming rates of the clients subject to network bandwidth constraints. Recent research has used cloud computing and datacenter networks to enhance the performance of interactive video streaming. Airlift [20] leverages inter-datacenter networks to relay traffic and process data streams for video conferences. It maximizes the total throughput in multiple conference sessions by choosing the optimal routes to deliver and relay packets in the inter-datacenter network, subject to end-to-end delay constraints.

In contrast, in this paper we consider a server pool that is much larger than a few datacenters and may consist of a large number of CDN POP locations. A piece of closely related work [21] also adopts a cloud of servers, called the *Virtual Mixer*, to enhance delay performance in a video conference. In particular, it tries to minimize either the average or the maximum end-to-end delay using a heuristic based on Steiner tree optimization performed on a graph of servers and clients. However, this heuristic has no performance guarantee. Neither is it scalable to a graph formed by a large number of CDN servers.

A similar server selection problem for distributed interactive applications is studied in [4] to select k servers from a graph formed by all candidate servers and clients. It is proved [4] that the graph version of this problem is NP-complete when $k < N$ or under some other conditions. A greedy approximation scheme has been proposed to solve the problem in graph with a complexity that grows fast as the size of the server pool increases. Our work is similar to this work in that we also use a server cloud to improve delay performance of interactive multimedia applications. However, we propose a novel geometric optimization procedure conducted in a geometric delay space, achieving better scalability as the number of utilizable servers increases.



(a) A 4-server solution for 7 clients (b) 1-server optimal (c) 2-server using centroids (d) 2-server optimal

Fig. 1. The illustration of our multi-server topology, where multiple servers are chosen from the cloud to serve each session. "T" is a client and "S" is a server. The arrows illustrate the data flow paths between clients.

III. Problem Formulation

There are several media distribution models for interactive video streaming, including architectures based on centralized servers, end-system mixing and peer-to-peer (P2P). A centralized-server solution [1], [22] collects user media using a relay server, performs signal processing (e.g., silent suppression) on the data streams and mixes them to be sent out to different clients. End-system mixing places the mixer on one of the clients. However, in many sessions, there may be no particularly well-endowed peer. A P2P solution adopts a full-mesh topology, in which each client sends its data to all the other clients directly. However, a P2P solution is apparently not scalable, especially for mobile clients, since as the number of clients N grows, each client has to split its limited upload bandwidth capacity to serve $N-1$ other clients, leading to vanishing pairwise throughput [2].

A. The Multi-Server Mesh Topology

In this paper, we consider using a "multi-server mesh" to serve the interactive video streaming session, which combines the advantages of both centralized servers and P2P architectures, as illustrated in Fig. 1(a).

Definition 1. *(***Multi-Server Mesh***) Every client is connected to only one server and no other host. The servers form a full mesh. In this topology, each client sends its own data to its server. For each server S, if it receives data from a client T, the data is forwarded to all the other hosts connected to this server, including servers and clients, excluding T. If server S receives data from other servers, the data is forwarded only to the clients directly connected to S.*

According to the above topology and protocol, a client connected to S transmits a packet to another client connected to S in two hops via S, and transmits a packet to another client connected to another server S' in three hops via S and S'. There are two benefits of the above multi-server topology, namely throughput advantage and latency improvement.

From a throughput perspective, since each client only needs to upload one copy of its data to one server, it does not suffer from the upload bottleneck as in the P2P case. As servers are connected in well-provisioned backbone infrastructures, transfers between servers are also free of bottleneck. Unlike the multi-server topology in [21] which allows a client to relay traffic for other clients, we require each client to only talk to its assigned server. If a certain client i is not connected to any server and is connected to some other client j, client j will relay traffic for client i, leading to an upload bottleneck at client j. More formally, suppose that a client i is not connected to any server, while the other clients are connected in an arbitrary way with or without servers. In this case, we can easily give an example where the bandwidth capacity of this network is not fully utilized as follows:

Consider the case that the download capacity D_i of each node is greater than the sum of upload capacities of other nodes, i.e., $D_i > \sum_{j\neq i} U_j$, and the servers have high bandwidth connections. Then the total throughput supportable by the network is $\sum_i \sum_{j\neq i} U_j$, since each node can at most download at the rate $\min\{D_i, \sum_{j\neq i} U_j\}$. Let us show that the maximum supportable total throughput $\sum_i \sum_{j\neq i} U_j$ can not be achieved. To achieve the maximum throughput, client i must be able to download at the rate $\sum_{j\neq i} U_j$. Since it is not connected to any server, it must use up all the upload capacities of other clients. As a result, other clients have no spare capacity to upload anything to any server. Since no server plays a role in this case at all, the usable upload capacity is only $\sum_i U_i$, which cannot support a total download rate of $\sum_i \sum_{j\neq i} U_j$.

From the latency perspective, the servers form a full mesh to minimize the transfer delays between any pair of servers. By adjusting the choice of server locations, we can effectively reduce the end-to-end delays between all clients in the formed multi-server topology that supports relatively high throughput. In addition, the servers can be extended to process data individually or collaboratively. Since our focus is on the network aspect, we do not discuss the details of signal processing and mixing functions.

B. Server Placement as Delay-Space Geometric Optimization

Since an interactive video streaming solution may utilize a large pool of servers from datacenters, CDN nodes, and dedicated servers, we can assume the utilizable servers are *geographically densely distributed*. It would be computationally expensive for any graph algorithms such as packing spanning trees or Steiner trees to select servers in the existing dense graph of a large server pool.

Given a set of N clients as input, this paper considers a completely different geometric approach to compute *where the servers should be placed* in a delay space, where each host has a coordinate, whether it be a server or client. The distance between two hosts in the delay space can predict their latency on the Internet. We can employ network coordinate systems (NCSs) [23] to compute the coordinates for a set of hosts given their pairwise ping data. For example, Vivaldi [6] is a

representative distributed NCS, and is deployed in many well-known Internet systems, e.g., Bamboo DHT [24] and Azureus BitTorrent [25].

Let vector $x_i = (x_i^1, ..., x_i^D) \in \mathbb{R}^D$ denote the coordinate of client i in the formed delay space. Each utilizable server has a coordinate as well. In Vivaldi, a new node only needs to collect the latency information from a few other existing nodes to compute its own coordinate. In other words, Vivaldi embeds the hosts into a delay space $\mathbb{R}^D$ based on a relatively sparse matrix of latencies between the hosts. Although it is believed that a Euclidean space of low dimension (e.g., $D = 2, 3, 5$) may embed the hosts with errors, due to triangle inequality violation [6], [25], in Sec. V through large-scale measurement data, we will show that despite coordinate mapping errors, our proposed server selection algorithms can still reduce latency in interactive video streaming sessions.

Suppose that $X = \{x_1, \ldots, x_N\}$ is a given set of coordinates of N clients in the delay space. Under the topology in Definition 1, our problem is to find a partition $P = \{C_1, C_2, ..., C_k\}$ of the client set X, together with a set of vectors $Y = \{y_1, y_2, ..., y_k\}$, where y_j $(j = 1, \ldots, k)$ denotes the server location for class C_j, such that the sum (or the mean) of end-to-end delays between all the clients is minimized. Given P and Y, there is a mapping $y : X \to Y$, where $y(x_i)$ is the server location of the class to which client x_i belongs. The problem is formally stated as

$$\min_{P,Y} \frac{1}{2} \sum_{i \neq j} \left(\|x_i - y(x_i)\| + \|y(x_i) - y(x_j)\| + \|x_j - y(x_j)\| \right) \tag{1}$$

$$\text{subject to} \quad |Y| \leq k,$$

where k is the maximum number of servers to be used.

When $k = N$, a trivial solution is to place a server just beside (arbitrarily close to) each client and connect each client to its server. Since a full mesh is formed among the N servers that are directly connected to each other, the sum of end-to-end delays between all clients reaches the minimum. However, it is costly to engage so many servers. Since deploying each server is associated with a cost of launching a VM instance on a certain physical machine in the cloud, we aim to provide a solution under a server number constraint $k < N$. Note that it is proved [4] that the graph version of Problem (1), in which a number of candidate servers and clients are connected in a graph with pairwise distances known, is NP-complete for any $2 \leq k < N$.

Once the ideal server locations are computed in the delay space, we can choose the nearest (in terms of delay) real servers in the physical network, and connect them to the clients according to the topology in the optimal solution. Note that the entire procedure is light-weight as long as the geometric Problem (1) can be efficiently solved. Since server coordinates are usually stable and can be routinely maintained prior to the session, we only need to compute the coordinates of the few participating clients using Vivaldi when they join the session. This procedure is efficient, since for each client, we only need to measure its RTTs to a few reference servers to obtain its coordinate in a network coordinate system. Moreover, unlike the graph version, the complexity of solving Problem (1) in space is independent of the number of available candidate servers. Even though the network coordinate system has errors, e.g., due to the violation of triangle inequality on the Internet and inaccuracy of network embedding, we will show the capability of our proposed optimization procedure in terms of reducing overall latencies, despite network coordinate errors, through extensive performance evaluation.

C. Relationships to Euclidean k-Median and Facility Location

For $2 \leq k < N$, as the decision variables include both the partition P and server locations Y, Problem (1) is a non-convex combinatorial problem in general. Note that Problem (1) appears to be similar to the famous *k-median* problem and *facility location* problem [7], yet is even more complex than them. In fact, if we ignore the delays between servers in (1), Problem (1) is reduced to the famous *Euclidean k-median clustering* problem, which is proved to be NP-hard as well as being hard to approximate to within arbitrary constant factor [7]. On the other hand, if we fix the partition P, (1) becomes the *Euclidean multi-facility location* problem [26], which is a convex problem. However, our problem is even harder because the partition P is also a decision variable. To the best of our knowledge, there is no known efficient solutions or even approximation schemes to the proposed Problem (1) in space.

IV. Server Placement in the Delay Space

We propose a number of algorithms to compute the ideal server locations in the embedded delay space, from one-server algorithms to two-server algorithms, with theoretical performance guarantees. We further propose practical heuristics that can place more than two servers efficiently. Given the set of clients, all the proposed schemes compute the ideal server locations in the delay space by approximately solving Problem (1), and then map each ideal server location to the nearest real physical server.

A. One-Server Algorithms

The current common solution to interactive video streaming in practice uses a single server location. A simple solution is to set the server location as the centroid of all clients:

Algorithm 1. *(**One-Server Centroid**) Set the server location to be* $y = \sum_{x_i \in X} x_i / N$.

A direct improvement is to solve Problem (1) with $k = 1$. When $k = 1$, Problem (1) becomes

$$\min_y \frac{1}{2} \sum_{x_i, x_j \in X, i \neq j} (\|x_i - y\| + \|x_j - y\|), \tag{2}$$

where y is the location of the single server, or equivalently,

$$\min_y (N-1) \sum_{x_i \in X} \|x_i - y\|, \tag{3}$$

which is a *convex program* to find the *geometric median* of the client set X. Since y is the median if and only if $\sum_{i=1}^{N} (x_i -$

$y)/\|x_i - y\| = 0$, we can perform a fixed-point iteration on the above equation to compute y, which is sometimes referred to as Weiszfeld's algorithm [27]:

Algorithm 2. *(**One-Server Median/One-Server Optimal**) Use the following iteration to get the server location y:*

$$y := \left(\sum_{i=1}^{N} \frac{x_i}{\|x_i - y\|}\right) \Big/ \left(\sum_{i=1}^{N} \frac{1}{\|x_i - y\|}\right) \tag{4}$$

Let $D_{1_{\mathsf{opt}}}$ and D_1 denote the sum of end-to-end delays achieved by One-Server Median and One-Server Centroid, respectively. Let D_N denote the sum of end-to-end delays in the full-mesh topology where *all the clients are directly connected*. D_N can be regarded as the minimum value of (1) if N servers are used, i.e., $k = N$. D_N is achieved if a server is placed arbitrarily close to each client and all the servers form a full mesh directly connecting each other. Thus, D_N is the minimum possible sum of end-to-end delays among all cases, since no server placement can beat connecting all client pairs directly in terms of delay. We will compare $D_{1_{\mathsf{opt}}}$ and D_1 against the best delay D_N.

Proposition 1. *Given any N clients $x_1, \ldots, x_N$, we have*

$$\frac{D_{1_{\mathsf{opt}}}}{D_N} \leq \frac{D_1}{D_N} \leq 2 - \frac{2}{N}, \tag{5}$$

where the left inequality achieve equality if and only if the centroid coincides with the median, while the right inequality achieves equality when all the clients are distributed on two points.

Proof: For the right inequality, using triangle inequalities, we have

$$\begin{aligned} D_1 &= (N-1) \sum_{x_i \in X} \left\| x_i - \frac{\sum_{x_j \in X} x_j}{N} \right\| \\ &= \frac{2(N-1)}{N} \cdot \frac{1}{2} \sum_{x_i \in X} \Big\| \sum_{x_j \in X} (x_i - x_j) \Big\| \\ &\leq \frac{2(N-1)}{N} \cdot \frac{1}{2} \sum_{x_i, x_j \in X} \|(x_i - x_j)\| = \frac{2(N-1)}{N} D_N, \end{aligned}$$

where the equality is achieved if and only if vector $x_i - x_j$ and vector $x_i - x_k$, $\forall i, j, k$, have the same direction or either of them is zero, which is equivalent to that all the clients are distributed on two points. The left inequality is obvious. ∎

Remarks: Proposition 1 shows that both One-Server Median and One-Server Centroid are $(2 - 2/N)$-approximation schemes to Problem (1) for any k. Moreover, One-Server Median, although being optimal using one server, has the same worst-case performance as One-Server Centroid. Finally, it is worth noting that even if we use N servers each serving one client to achieve the best possible delay, the delay reduction as compared to using one server is no more than $2\times$.

B. Two-Server Algorithms

One-server algorithms may perform poorly when the clients tend to distribute in separable clusters. If we slightly increase the budget and use two servers, it is not hard to check that Problem (1) becomes

$$\begin{aligned} \min_{\{C_1, C_2, y_1, y_2\}} (N-1) \Bigg(\sum_{x_i \in C_1} \|x_i - y_1\| + \sum_{x_i \in C_2} \|x_i - y_2\| \Bigg) \\ + mn \|y_1 - y_2\|, \end{aligned} \tag{6}$$

where $m := |C_1|$ and $n := |C_2|$ are the numbers of clients in classes C_1 and C_2, respectively. Problem (6) is still a hard combinatorial problem, in which server locations and the way we partition clients will both affect the sum of end-to-end delays. We provide a centroid-based approximate solution and bound its performance. To simplify notations, we set

$$A = \frac{1}{2} \sum_{x_i, x_j \in C_1} \|x_i - x_j\|, \; B = \frac{1}{2} \sum_{x_i, x_j \in C_2} \|x_i - x_j\|$$
$$C = \sum_{x_i \in C_1, x_j \in C_2} \|x_i - x_j\|.$$

Clearly, we have $D_N = A + B + C$.

To judge how good a partition is, we define *separability* as

$$\beta(C_1, C_2) := \frac{C}{2mn} \Big/ \left(\frac{A}{m^2} + \frac{B}{n^2}\right), \tag{7}$$

which represents the ratio between (normalized) cross-cluster distances and in-cluster distances. Intuitively, the greater the $\beta(C_1, C_2)$, the further apart the clusters C_1 and C_2. We describe our two-server algorithm in Algorithm 3.

Algorithm 3. *(**Two-Server Centroids**) First, choose the partition $\{C_1, C_2\}$ to maximize $\beta(C_1, C_2)$. Then choose the centroid as the server location in each class, i.e., set $y_1 = \sum_{x_i \in C_1} x_i/m$ and $y_2 = \sum_{x_i \in C_2} x_i/n$.*

Since the number of clients in a session (e.g., a live conference) is usually no more than a few, the first step is easy to perform, for example, simply by enumerating all the partitions $\{C_1, C_2\}$ to find the one with the maximum β. For example, even with 10 clients, there are only 512 different partitions to go through. In Sec. V, we will show that our proposed algorithms always finish within 1 second for up to 12 clients.

Let D_2 denote the sum of end-to-end delays achieved by Algorithm 3. Again, we analyze D_2 as compared to D_N.

Proposition 2. *For any client set X, we have $\frac{D_2}{D_N} < 2.5$.*

Please refer to the appendix for the proof.

Remarks: Proposition 2 shows that Two-Server Centroids is at least a 2.5-approximation scheme to Problem (1) for any $k \geq 2$. However, there is really no information about whether Two-Server Centroids is better than one-server algorithms or not. According to the proof of Proposition 2, Two-Server Centroids may not outperform one-server algorithms in some cases. The underlying reason is that although the two-server optimal solution is certainly no worse than the one-server optimal solution, Problem (1) with $|Y| = 2$ is an extension of the 2-median problem, while k-median is NP-hard [7]. Without any known efficient algorithm to compute the 2-server optimal solution, we cannot conclude if the *centroids* of two servers are better than one-server optimal solution: if the clients are

distributed in two clusters, two servers are better; if they are more mingled, even forming multiple (more than 2) clusters, one server is better. We will see this effect in Sec. V through simulations.

We now present a simple algorithm that guarantees to beat Algorithm 2 in theory.

Algorithm 4. *(**One-Or-Two-Server**) Use either Algorithm 2 or Algorithm 3, whichever produces a smaller sum of end-to-end delays.*

Denote $D_{12} = \min\{D_{1\mathsf{opt}}, D_2\}$ as the sum of end-to-end delays achieved by Algorithm 4. Then we have:

Proposition 3. *For any client set X, we have*

$$\frac{D_{12}}{D_N} = \min\left\{\frac{D_{1\mathsf{opt}}}{D_N}, \frac{D_2}{D_N}\right\} < 2 - \frac{2}{N}, \tag{8}$$

Remarks: The worst-case performance of Algorithm 4 is now *strictly* less than $2 - 2/N$, because when $D_{1\mathsf{opt}}/D_N$ achieves its maximum value of $2-2/N$, all clients must reside on two points, and in this case, it is easy to check that D_2/D_N is smaller than $2 - 2/N$.

Although a tighter bound on the worst-case D_{12}/D_N is hard to derive, we can characterize the improvement of Algorithm 4 over Algorithm 2 (One-Server Median). Since $D_{12} = \min\{D_{1\mathsf{opt}}, D_2\}$, we only need to bound $D_{1\mathsf{opt}}/D_2$ from above. If $D_{1\mathsf{opt}}/D_2 \le \alpha$, we can say Algorithm 4 can reduce delay by a factor of at most α.

Suppose $\{C_1, C_2\}$ is the partition of clients found by Algorithm 3. Let y represent the centroid of X, and let y_1 and y_2 represent the centroids of C_1 and C_2, respectively. Let $m = |C_1|$ and $n = |C_2|$ be the numbers of clients in C_1 and C_2, respectively. To simplify notation, define D, E and F as

$$D := \sum_{x_i \in C_1} \|x_i - y_1\|,\ E := \sum_{x_i \in C_2} \|x_i - y_2\|,\ F := \|y_1 - y_2\|$$

Similar to β, we define another separability measure

$$\beta'(C_1, C_2) := \frac{F}{D/m + E/n}, \tag{9}$$

which will be used to derive the improvement ratio of Algorithm 4 over one-server algorithms. Due to the hardness of the problem, we are able to provide results for up to $N = 6$ clients:

Proposition 4. *For a given client set X, run Algorithm 3. If $1 = m < n \le 3$, we have*

$$\frac{D_1}{D_2} \le \frac{1}{n + \beta'} \cdot \left(\sqrt{n^2 + \left(\frac{2}{n+1}\right)^2 \beta'^2} + \left(\frac{2n-2}{n+1}\right)\beta'\right), \tag{10}$$

and the bound is the best possible; if $1 < m \le n \le 3$, then

$$\frac{D_1}{D_2} \le \frac{m+n-1}{(m+n-1)m + mn\beta'} \cdot \left(\sqrt{m^2 + \left(\frac{2n}{m+n}\right)^2 \beta'^2} + \left(\frac{2mn-2n}{m+n}\right)\beta'\right), \tag{11}$$

and the bound is the best possible.

Please refer to the appendix for the proof of Proposition 4. Since $D_{1\mathsf{opt}} \le D_1$, the right hand sides of (10) and (11) also upper-bound $D_{1\mathsf{opt}}/D_2$, which is the performance improvement of Algorithm 4 over One-Server Median. We can get the following corollary immediately:

Corollary 5. *For a given client set X, run Algorithm 3. If $1 \le m \le n \le 3$ with $m + n = N$ and $\beta' \to \infty$, we have*

$$\frac{D_{1\mathsf{opt}}}{D_2} \le \frac{D_1}{D_2} \le 2 - \frac{2}{N}. \tag{12}$$

Proof Sketch: By analyzing the derivatives, it can be shown that the right hand sides of both (10) and (11) reach the same maximum value of $2 - 2/N$ when $\beta' \to \infty$. ■

Remarks: Corollary 5 implies that Algorithm 4 suffices to achieve the best improvement factor of $2 - 2/N$ over One-Server Optimal. The best improvement is achieved when $\beta' \to \infty$. In this case, compared to the distance between two clusters, the clients are almost distributed on two points, and thus using 2 servers is equivalent to using N servers. According to Proposition 1, the best improvement of using N servers is also achieved in this extreme case. However, since in reality β' is usually finite, the improvement factor of Algorithm 4 over One-Server Optimal will not be as significant as using more servers.

C. Multi-Server Placement

When we need to place more than two servers, we may use the k-means clustering heuristic to quickly partition the clients into k classes, although k-means aims to find a partition to minimize the sum of delays from the clients to their corresponding class centroids, which is different from our objective of minimizing the sum of end-to-end delays. Once the partition is obtained, it is not hard to check that Problem (1) using k servers is reduced to the convex program of finding the server locations $Y = \{y_1, \ldots, y_k\}$:

$$\min_{\{y_1,\ldots,y_k\}} (N-1) \sum_{j=1}^{k} \sum_{x_i \in C_j} \|x_i - y_j\| + \sum_{i=2}^{k} \sum_{j=1}^{i-1} |C_i| \cdot |C_j| \cdot \|y_i - y_j\| \tag{13}$$

Clearly, once the partition $P = \{C_1, \ldots, C_k\}$ is fixed, the objective of (13) is a convex function of y_i. Problem (13) can be solved efficiently using standard convex problem solvers, with the initial server locations set as the class centroids. Now we have obtained the following multi-server placement scheme:

Algorithm 5. *(**k-Server Optimization Heuristic**) First, partition the clients into k classes via k-means heuristic. Then solve the convex problem (13) to get the server locations.*

As has been analyzed in Sec. III-C, the k-means problem is a simplified version of our problem with the weights between servers removed. Thus, it can be a good heuristic to partition the clients into any k classes. In our k-Server Optimization Heuristic, we utilize the similarity between k-means and our

problem to help obtain a sub-optimal partition and further reduce the errors of ignoring the weights between servers by fine-tuning the locations of the servers under the found k-means partition.

D. Ideal-to-Real Server Mapping and Algorithm Complexity

Note that all the proposed algorithms aim to find (ideal) server locations in the delay space. As a common final step of all the proposed algorithms, we map each computed ideal server location to the nearest real physical server in terms of delay, which is fast. We will see in Sec. V that our server selection procedure can indeed reduce latency even if the real server mapping phase can introduce errors.

Let us have a glimpse on the complexity of different algorithms. In fact, the running time of an algorithm is mainly dictated by the time to compute ideal server locations in the delay space, which depends only on the number of clients N and the number of servers to be adopted k, but not on the total number of candidate servers M. In contrast, selecting k servers on a graph of all M candidate servers and N clients [4] has a complexity that dramatically increases with M, which can be quite large in today's large-scale server cloud. The complexity of the proposed one-server algorithms in space is linear in N. For Two-Server Centroids and One-Or-Two-Server, the complexity is $O(2^N)$. However, the speed is still satisfactory when the number of clients N is limited to a few closely related people (which is common in today's multi-party live conferencing and other interactive video streaming applications). Even if N is large, the simple k-means partition can achieve a satisfactory performance in polynomial time with complexity $O(k^c)$, c being a constant.

V. Trace-Driven Simulations

We conduct simulation evaluation based on two network latency datasets we have collected, one containing round-trip times (RTTs) in the PlanetLab, the other containing RTTs from the Seattle network [8] to PlanetLab nodes. The PlanetLab dataset contains the RTTs between 490 PlanetLab nodes (including 51 nodes in Asia, 222 nodes in Europe, 192 nodes in North America, and 25 nodes in other regions), that we collected in 2014 over a 15-day period, with the geographic distribution of the nodes shown in Fig.(2). We have also collected RTT measurements from 99 nodes in *Seattle*, an open peer-to-peer computing platform [8] that consists of laptops, servers, and phones donated by users and institutions for research purposes, to the 490 PlanetLab nodes. While PlanetLab nodes are mostly stable and located in university networks, the Seattle-PlanetLab RTTs are used to model the longer delays from personal devices to servers in Sec. V-D.

We use the median latencies between nodes as the input to our server placement algorithm. In real-world applications, network coordinates of the severs may be computed and stored *a priori*. When a client joins the session, its network coordinate can be computed by a local decentralized Vivaldi algorithm with `pings` to only a few random servers. In our experiments based on PlanetLab data, we directly embed all the nodes into a delay space using an local iterative Vivaldi algorithm. In each test run, we randomly choose a certain number of nodes as clients, while the remaining nodes act as the potential server pool. This way, the clients in each of our experiments are essentially randomly distributed over the entire world across multiple regions. In experiments involving the Seattle data, we take the 490 PlanetLab nodes as the potential server pool and randomly select nodes from the 99 Seattle nodes as clients.

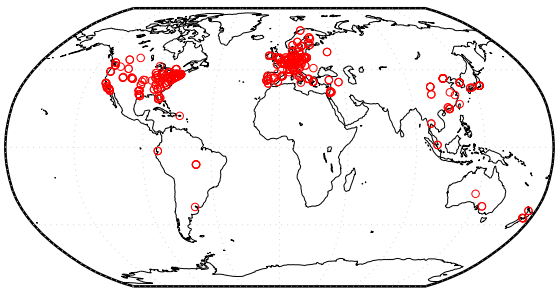
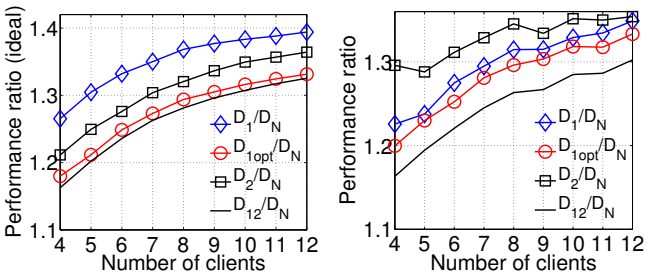


Fig. 2. The locations of the 490 PlanetLab nodes.



(a) Theoretical (in the delay space) (b) After real server mapping

Fig. 3. The performance of different algorithms in the 490-node simulation.

A. Algorithms with No More than Two Servers

In this subsection and the next one, we study the delay performance based on the 490-node PlanetLab dataset, as the number of clients N ranges from 4 to 12. In each run, different algorithms are applied, and their corresponding server locations and client partitions are computed in the delay space. Then we map these solutions to the nearest real servers in the dataset and build up the network. Finally, we evaluate the delay performance of these networks according to the real end-to-end delays of the clients in the RTT traces. For every parameter setting, we repeat the simulation for 1000 times. For each server selection solution produced by a certain algorithm, we record the sum of end-to-end delays between all clients D (in the real traces), and normalize it by D_N, which is the sum of pairwise distances of all the clients forming a *full mesh*. We finally use the ratio D/D_N to evaluate the server selection.

Fig. 3 shows the performance of our proposed algorithms on the PlanetLab data. Fig. 3(a) shows the average (normalized) sum of end-to-end delay in each session for the ideal server locations computed in the delay space, whereas Fig 3(b) shows such value after mapping the computed ideal server locations to real servers. In both figures, the performance ratios are much smaller than the theoretical worst-case bounds. And the ratios

TABLE I
THE PERFORMANCE OF DIFFERENT ALGORITHMS IN THE 490-NODE SIMULATION WITH 8 RANDOM CLIENTS REPEATED FOR 1000 TIMES.

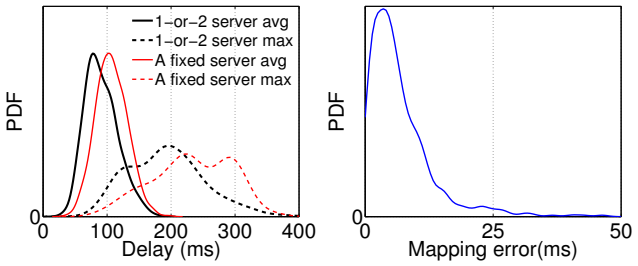
		Average	Worst	Best	Variance
Theorectical	D_1/D_n	1.388	1.617	1.187	0.003259
	D_{1opt}/D_n	1.324	1.529	1.125	0.004288
	D_2/D_n	1.356	1.487	1.162	0.002175
	D_{12}/D_n	1.317	1.450	1.125	0.003227
Real	D_1/D_n	1.334	2.089	0.979	0.02355
	D_{1opt}/D_n	1.317	1.994	0.977	0.01539
	D_{1opt}/D_n	1.350	2.153	0.981	0.02621
	D_{12}/D_n	1.286	1.855	0.977	0.01375

monotonically increase when the number of clients increases. The Two-Server algorithm performs worse in the real case due to the mapping error. However, the proposed One-Or-Two-Server algorithm is obviously better than One-Server-Optimal. The reason is that we are choosing a better one from two solutions, and despite network coordinate errors and mapping errors, the algorithm can pick the one with a smaller error or take advantage of random changes.

Specifically, Table I shows more details for the average, best, worst-case and variance of the performance of 1000 tests on 8 random clients. The best and worst-case performance follows a similar trend as the average performance. The real ratio has a larger variance due to other errors like network coordinate errors and mapping errors. Note that the real best performance can even be slightly better than 1 since the triangle inequality in real networks does not hold strictly.

Fig. 4(a) plots the distribution of the mean and maximum end-to-end delays achieved by One-Or-Two-Server, compared to a *fixed single server location* (similar to what Skype adopts [1]) for 8 clients randomly chosen from the 490-node dataset, repeated 1000 times. We can see that the average end-to-end delays of most sessions with our simple One-Or-Two-Server algorithm are around 90 ms, and the maximum end-to-end delay is mostly less than 200 ms, which is obviously better than the single fixed server solution. Furthermore, Fig. 4(b) plots the real server mapping error in the same experiment, that is the gap between the real end-to-end delay and the estimated end-to-end delay in the delay space (without mapping to real servers). Most errors are less than 15 ms. Compared to the average end-to-end delay of about 90 ms, the real delay is close to the estimated delay in the delay space.

Through the above performance comparisons, we have the following observations. *First*, One-Or-Two-Server, which is a combination of One-Server Optimal and Two-Server-Centroids, outperforms each of them. Two-Server Centroids can successfully complement One-Server Optimal, reducing delay in the case when One-Server Optimal performs poorly. With the One-Or-Two-Server algorithm, we can effectively reduce the delay over one-server algorithms, including a single fixed server and one-server optimal algorithms, and achieve a reasonably good solution when the number of clients does not exceed 12. *Second*, although One-Server Optimal outperforms One-Server Centroid, yet One-Server Centroid has the shortest execution time, making it a valuable choice in applications that just need one server for extremely quick launching.



(a) Avg/max end-to-end delays (b) Gap of computed and mapped

Fig. 4. (a) The distributions of the mean and maximum end-to-end delays with One-or-Two-Server or a single fixed server; (b) The gap between the computed mean end-to-end delay and that after mapping to real servers. Tests are performed on 8 random clients for 1000 times.

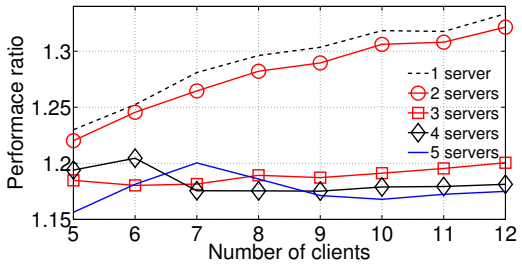


Fig. 5. Delay performance (normalized by D_N) of the k-Server Optimization Heuristic in the 490-node simulation.

B. Performance of k-Server Heuristics

Fig. 5 shows the average performance of the k-Server optimization heuristic. The performance generally increases when more servers are engaged. There is a relatively large improvement from the 2-server to 3-server solutions. When using more than 3 servers, the improvement becomes marginal and not stable. It seems that the 3-server solution is the best and sufficient to handle the diversity in the geographic distribution of 12 clients, thus leading to a large performance increase over the 2-server solution, while adopting more than 3 servers appears to be unnecessary for 12 clients. We can also see that as the number of clients increases, the performance of more servers degrades at a slower pace than 1 or 2 servers. Another observation is that having more servers might experience bad performance when the number of clients is slightly larger than the number of servers. Therefore, the k-Server Optimization Heuristic with 3 servers are generally the most cost-effective solution.

We have also measured the execution times of our algorithms, and present them in Table II for all the proposed algorithms under different numbers of clients. According to the discussions at the end of Sec. IV, we estimate the running time for each algorithm by keeping track of the computation time as well as the mapping time, and summing them up. The running times of the two-server algorithms involving exhaustive partition search increases dramatically as the number of clients grows. However, they are still practical when there are no more than 12 clients—all algorithms finish in only 1 second. The running times of the other algorithms are consistently low regardless of the number of clients.

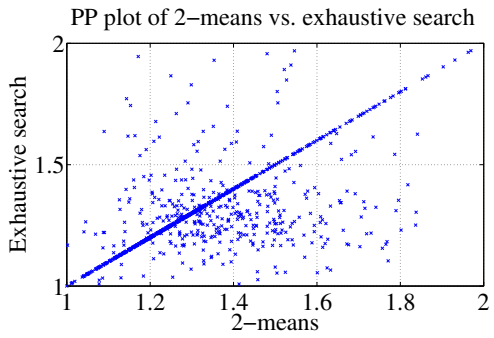
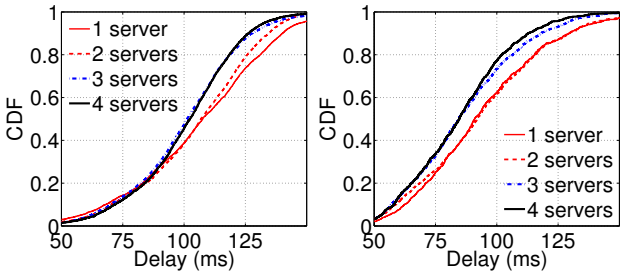


Fig. 6. The performance of Two-Server Centroid: 2-means partition vs. max-β partition.



(a) Seattle and PlanetLab (b) PlanetLab

Fig. 7. The CDF of mean end-to-end delays for the cross-network simulation and the 490-node simulation for 12 clients.

C. k-Means Partition vs. Max-β Partition

In our solutions with two servers, the performance critically depends on the partition. As we have shown, the more separate the two classes are, the better the two-server solution will perform. We have proposed to exhaustively search for the best partition to maximize the separability β, which leads to excessive computational overhead when N is greater than a few dozens (although a rare case nowadays).

We propose to reduce such complexity using the k-means algorithm. By (9), separability is related to the sum of the delays from the clients to the class centroid in each class, and we know that the k-means algorithm is an efficient heuristic for finding the solution minimizing that sum. Consequently, we can conduct partition in Two-Server Centroid and One-Or-Two-Server algorithm using the 2-means algorithm instead of an exhaustive search.

For a typical $N = 6$, we have performed thousands of trials with on the 490-node dataset, and record the normalized delay performance D_2/D_N of Two-Server Centroid with 2-means partition versus with the max-β partition, and plot the results in Fig. 6. We observe that a large group of the points are near the line $y = x$, which implies that the 2-means algorithm is an excellent alternative to exhaustive partition search. Due to network coordinate projection errors and server mapping errors, sometimes 2-means has even better performance. This efficient heuristic has nearly the same average performance as max-β partition in general, and can easily scale to a large number of clients.

TABLE II
THE ALGORITHM RUNNING TIMES (MS)

Number of Clients	4	6	8	10	12
One-Server Centroid	3.71	3.66	3.40	3.50	3.46
One-Server Optimal	5.85	6.67	6.99	8.30	8.81
Two-Server Centroid	8.22	10.78	22.37	87.76	393.35
One-Or-Two-Server	14.06	11.45	29.36	96.06	402.16
k-Server Heuristic (k=2)	69.62	66.64	62.05	66.84	66.67

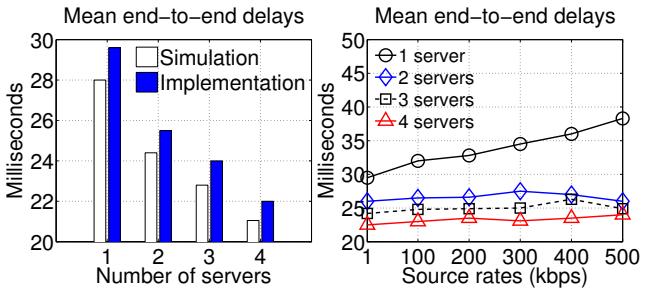
D. Simulations across Seattle and PlanetLab Networks

To simulate the possibly different latencies between inter-server connections and client-to-server connections, we make the 490 PlanetLab nodes the server pool and in each run, randomly choose 12 nodes from the 99 Seattle nodes to serve as the clients and deploy the k-Server heuristics to select server(s). Note that the Seattle nodes have relatively worse network conditions than the PlanetLab nodes since the Seattle nodes come from a more diverse environment. Fig. 7(a) shows the average end-to-end delay between all client pairs achieved in 1000 runs and Fig. 7(b) shows the corresponding result in the PlantLab-only simulations (both servers and clients are from the 490 PlanetLab nodes). We can see that the multi-server solution has a larger benefit over a single server in the cross-network simulation especially for the 2-server heuristic, since the better network conditions among servers reduces the delay between servers, which reduces the cost of engaging one more transmission hop with respect to the one-server solutions. The better the network conditions between servers, the more the multi-server solutions will help.

VI. Prototype Implementation

To verify the real-world performance of the proposed methods, we have developed an *asynchronous* multi-threaded packet communication module, with the *Apache Thrift* framework and the Boost library, in $2,000$ lines of C++ code. And Thrift generated about $5,000$ lines of code. We deployed this prototype interactive streaming system on PlanetLab nodes. In our experiments, each client sends packets to its designated server at a frequency of 300 packets/second using TCP, where the source rate is controlled by tuning the packet size: $\mathsf{sourcerate} = \mathsf{packetsize} \times \mathsf{frequency}$. We aim to measure the real *packet-level* end-to-end delays achieved at different source rates considering both the network distance effect and system load.

Since it is hard to synchronize the clocks on different computers, the delay between clients (on the order of ms) cannot be measured by simply recording the sending time on the sender and the receiving time on the receiver. We propose an indirect method to measure the end-to-end delay of each packet. Suppose client A is sending packets to another client B via some servers. At the very moment before A sends out a packet, it starts a timer. When the packet eventually reaches B, B will send a ping packet *directly* to the sender A. When A receives the ping packet, it stops its timer and record the time span T_{circle}, which is the end-to-end delay of the packet from A to B plus the one-way ping time from B to A. When B gets the reply of the ping from A, it records the round trip



(a) Implementation v.s. simulation (b) Impact of source rates

Fig. 8. The mean end-to-end delay in implementation at various source rates for 6 clients.

time RTT_{AB} . Therefore, the end-to-end delay from A to B can be evaluated by $T_{\text{circle}} - \text{RTT}_{AB}/2$. In our implementation, each client measures the end-to-end delay to all other clients once every 300 packets.

The goal of the prototype-based experiments is to verify if the delays calculated by summing up the ping values really conform to the packet-level delays in a real system where CPU and other resource usage may also affect end-to-end delays. Therefore, we have selected a group of 6 clients, for which the multi-server heuristic can indeed improve delay performance in the simulation and we want to see whether such benefits still exist in the implementation.

Fig. 8(a) shows the performance comparison between simulation and implementation. To eliminate the influence of source rates on latency , here we deliberately set the source rate to be 1 kbps. Note that as the number of servers increases from 1 to 4, the real delay (ms) in the implementation decreases at a similar pace to that in the simulation (which estimates delays by summing up RTTs). The real delay is only slightly worse than the simulated result due to the existence of queuing delays and processing (CPU) delays.

Fig. 8(b) illustrates the change of the mean end-to-end delays as the source rates of clients increase. When the number of servers is 2, 3 and 4, the mean end-to-end delays only increases slightly as the source sending rate increases from 1 kbps all the way to 500 kbps at each source (which can support sufficiently high video quality). However, in the one-server solution, as the source rate increases, the mean end-to-end delays has a dramatic increase, which surges from 29.5 ms to 38.3 ms as the source rate changes from 1 kbps to 500 kbps. The reason is that the server uploading burden increases with fewer servers, since now each server needs to upload data to more terminals.

VII. Concluding Remarks

In this paper, we study server selection and server location optimization with a k-server mesh topology in distributed interactive video streaming applications, aiming at minimizing the summation (or mean) of end-to-end delays between clients. We formulate the problem in a delay space instead of on a graph, and propose a number of conceptually simple one-server or two-server approximate solutions with theoretical worst-case performance guarantees. We further propose practical optimization heuristics to enable rapid selection of three or more servers. We have performed extensive trace-driven simulations based on large amounts of measurement data collected from the PlanetLab and Seattle platforms and implemented a prototype system to verify our proposed server selection schemes in real networks. Despite the errors in network embedding and real server mapping, our proposed algorithms can efficiently select servers for interactive video streaming, achieving low end-to-end delays.

In conclusion, by using a simple One-or-Two-Server scheme, both the mean and maximum end-to-end delays in interactive video streaming sessions can be greatly reduced, as compared to always using servers at a single location. With the proposed k-server selection heuristic, mean end-to-end delays can be further reduced as more servers are used. However, for the cost-effectiveness in sessions composed of no more than 12 clients, a practical size of today's interactive video streaming sessions, using more than 3 servers is mostly unnecessary. Moreover, with any of the proposed algorithms, it takes less than 1 second on a commodity personal computer to optimally select the locations of multiple servers for 12-client sessions, given a pool of close to 500 candidate servers.

References

[1] Y. Xu, C. Yu, J. Li, and Y. Liu, "Video telephony for end-consumers:measurement study of google+, ichat, and skype," in *Proc. of ACM conference on Internet measurement conference*, 2012.

[2] Y. Lu, Y. Zhao, F. Kuipers, and P. V. Mieghem, "Measurement study of multi-party video conferencing," in *Proc. of 9th International IFIP TC 6 Networking Conference*, 2010.

[3] C. Yu, Y. Xu, B. Liu, and Y. Liu, ""Can you SEE me now?" a measurement study of mobile video calls," in *INFOCOM*, 2014, pp. 1456–1464.

[4] H.Zheng and X. Tang, "On server provisioning for distributed interactive applications," in *the 33rd IEEE International Conference on Distributed Computing Systems (ICDCS)*, Philadelphia, PA, July 2013, pp. 500 – 509.

[5] Akamai, http://www.akamai.com.

[6] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A decentralized network coordinate system," in *Proc. of ACM SIGCOMM*, 2004.

[7] V. V. Vazirani, *Approximation Algorithms*. Springer-Verlag New York, 2001.

[8] J. Cappos, I. Beschastnikh, A. Krishnamurthy, and T. Anderson, "Seattle: a platform for educational cloud computing," in *ACM SIGCSE Bulletin*, vol. 41, no. 1. ACM, 2009, pp. 111–115.

[9] L. D. Cicco, S. Mascolo, and V. Palmisano, "Skype video congestion control: an experimental investigation," *Computer Networks*, vol. 55, no. 3, pp. 558–571, Feb 2011.

[10] X. Zhang, Y. Xu, H. Hu, Y. Liu, Z. Guo, and Y. Wang, "Profiling skype video calls: Rate control and video quality," in *Proc. of IEEE INFOCOM*, Mar 2012, pp. 621–629.

[11] A. Arefin, Z. Huang, R. Rivas, S. Shi, P. Xia, K. Nahrstedt, W. Wu, G. Kurillo, and R. Bajcsy, "Classification and analysis of 3d tele-immersive activities," 2012.

[12] S. A. Baset and H. G. Schulzrinne, "An analysis of the skype peer-to-peer internet telephony protocol," in *INFOCOM*, 2006, pp. 1–11.

[13] D. Bonfiglio, M. Mellia, N. R. M. Meo, , and D. Rossi, "Tracking down skype traffic," in *INFOCOM*, April 2008, pp. 261–265.

[14] L. Qiu, V. N. Padmanabhan, and G. M. Voelker, "On the placement of web server replicas," in *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 3. IEEE, 2001, pp. 1587–1596.

[15] K.-W. Lee, B.-J. Ko, and S. Calo, "Adaptive server selection for large scale interactive online games," *Computer Networks*, vol. 49, no. 1, pp. 84–102, 2005.

[16] C. Ding, Y. Chen, T. Xu, and X. Fu, "Cloudgps: a scalable and isp-friendly server selection scheme in cloud computing environments," in *Proceedings of the 2012 IEEE 20th International Workshop on Quality of Service*. IEEE Press, 2012, p. 5.

[17] C. W. Cameron, S. H. Low, and D. X. Wei, "High-density model for server allocation and placement," *ACM SIGMETRICS Performance Evaluation Review*, vol. 30, no. 1, pp. 152–159, 2002.

[18] X. Chen, M. Chen, B. Li, Y. Zhao, Y. Wu, and J. Li, "Celerity: A low-delay multi-party conferencing solution," in *Proc. of ACM Multimedia*, 2011.

[19] C. Liang, M. Zhao, and Y. Liu, "Optimal bandwidth sharing in multi-swarm multiparty p2p video-conferencing systems," *IEEE/ACM Trans. Networking*, vol. 19, no. 6, pp. 1704–1716, 2011.

[20] Y. Feng, B. Li, and B. Li, "Airlift: Video conferencing as a cloud service using inter-datacenter networks," in *Proc. of IEEE ICNP*, 2012.

[21] J. Liao, C. Yuan, W. Zhu, and P. A. Chou, "Virtual mixer: Real-time audio mixing across clients and cloud for multi-party conferencing," in *Proc. of IEEE ICASSP*, 2012.

[22] K. Singh, G. Nair, and H. Schulzrinne, "Centralized conferencing using sip," *Internet Telephony Workshop*, vol. 7, 2001.

[23] B. Donnet, B. Gueye, and M. A. Kaafar, "A survey on network coordinates systems, design, and security," *Communications Surveys & Tutorials, IEEE*, vol. 12, no. 4, pp. 488–503, 2010.

[24] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, "Handling Churn in a DHT," in *the USENIX Annual Technical Conference*, 2004.

[25] J. Ledlie, P. Gardner, and M. Seltzer, "Network coordinates in the wild," in *Proc. of NSDI*, 2007.

[26] J. B. Rosen and G. L. Xue, "On the convergence of miehle's algorithm for the euclidean multifacility location problem," *Operations Research*, vol. 40, no. 1, pp. 188–191, 1992.

[27] R. Chandrasekaran and A. Tamir, "Open questions concerning weiszfeld's algorithm for the fermat-weber location problem," *Mathematical Programming, Series A*, vol. Volume 44, no. 1-3, pp. 293–295, May 1989.

APPENDIX

Proof of Proposition 2: We first present several lemmas.

Lemma 6. *Given any N clients $x_1, \ldots, x_N$, we have*

$$\frac{D_2}{D_N} < 2 - \frac{4(\beta-1)}{\frac{m}{n}+\frac{n}{m}+4\beta}. \tag{14}$$

Similar to (6) in the proof of Proposition 1, using triangle inequalities, we can get

$$\sum_{x_i \in C_1} \|x_i - y_1\| \le \frac{2}{m} \cdot \frac{1}{2} \sum_{x_i, x_j \in C_1} \|x_i - x_j\| = \frac{2}{m} \cdot A$$
$$\sum_{x_i \in C_2} \|x_i - y_1\| \le \frac{2}{n} \cdot \frac{1}{2} \sum_{x_i, x_j \in C_2} \|x_i - x_j\| = \frac{2}{n} \cdot B,$$

where each equality is achieved when the clients in that class are distributed on two points. Using triangle inequalities with some careful edge counting, we can get

$$mn\|y_1 - y_2\| = mn\left\|\frac{\sum_{x_i \in C_1} x_i}{m} - \frac{\sum_{x_j \in C_2} x_j}{n}\right\|$$
$$= \left\|\sum_{x_i \in C_1, x_j \in C_2} (x_i - x_j)\right\| \le \sum_{x_i \in C_1, x_j \in C_2} \|x_i - x_j\| = C,$$

where the equality is achieved when all the clients are distributed on a line and there exists a point on the line which separates the clients from the two classes. Therefore,

$$\begin{aligned}\frac{D_2}{D_N} &\le \frac{(N-1)\left(\frac{2}{m}A + \frac{2}{n}B\right) + C}{A+B+C}. \\ &= 2 - \frac{C - \left(\frac{2n}{m}A + \frac{2m}{n}B\right) + \left(\frac{2}{m}A + \frac{2}{n}B\right)}{A+B+C} \\ &< 2 - \frac{C - \left(\frac{2n}{m}A + \frac{2m}{n}B\right)}{A+B+C},\end{aligned}$$

Eliminating C by(7), we have

$$\frac{D_2}{D_N} < 2 - \frac{2(\beta-1)\left(\frac{n}{m}A + \frac{m}{n}B\right)}{A+B+2\beta\left(\frac{n}{m}A + \frac{m}{n}B\right)}. \tag{15}$$

Since

$$\frac{n}{m}A + \frac{m}{n}B \ge 2\sqrt{AB}, \tag{16}$$

where the equality is achieved when

$$A/B = m^2/n^2. \tag{17}$$

Once this equality holds, we get (14).

Lemma 7. *Given m, n with $m + n = N$, there exists a partition $\{C_1, C_2\}$, such that $|C_1| = m$, $|C_2| = n$, and $\beta(C_1, C_2) \ge 1/(2 - \frac{1}{m} - \frac{1}{n})$.*

We exhaust all partitions $\{C_1, C_2\}$ with $|C_1| = m$ and $|C_2| = n$ and inspect the ratio

$$\beta_{\mathsf{sum}} = \frac{\sum_{P:|C_1|=m,|C_2|=n} C/(2mn)}{\sum_{P:|C_1|=m,|C_2|=n} \left(A/m^2 + B/n^2\right)}.$$

Due to the exhausting procedure, both the numerator and the denominator of β_{sum} evenly cover all the pairwise delays of the clients. Therefore, we can evaluate their ratio by counting how many pairs of delays they cover. Since there are totally $\binom{m+n}{m}$ different kinds of partitions, after cancelling the same factor, we have

$$\beta_{\mathsf{sum}} = \frac{\frac{1}{2mn} \cdot \binom{m+n}{m} \cdot mn}{\binom{m+n}{m} \cdot \left(\frac{1}{m^2} \cdot \binom{m}{2} + \frac{1}{n^2} \cdot \binom{n}{2}\right)} = \frac{1}{2 - \frac{1}{m} - \frac{1}{n}}.$$

Hence, among all the different $\beta(C_1, C_2)$ with $|C_1| = m$ and $|C_2| = n$, there must exist one such that $\beta(C_1, C_2) \ge 1/(2 - \frac{1}{m} - \frac{1}{n})$.

According to Lemma 7, for the partition that maximizes β, we have $\beta \ge 0.5$. Therefore, by (14), we have

$$\frac{D_2}{D_N} < 1 + \frac{\frac{m}{n}+\frac{n}{m}+4}{\frac{m}{n}+\frac{n}{m}+4\beta} \le 1 + \frac{\frac{m}{n}+\frac{n}{m}+4}{\frac{m}{n}+\frac{n}{m}+2} \le 2.5.$$

Proof of Proposition 4: When proving Propositions 1 and 2, we have fixed D_N and found upper bounds on D_1 and D_2. In the following, we take a different approach to fix D_2 and find the maximum value $D_{1\mathsf{max}}$ of D_1, that is the delay produced by One-Server Centroid. Since $D_{1\mathsf{opt}} \le D_1$, $D_{1\mathsf{max}}$ is also an upper bound on $D_{1\mathsf{opt}}$. The next lemma gives $D_{1\mathsf{max}}$ for fixed D, E, F:

Lemma 8. *Given D, E, and F, and the numbers of clients in the two classes m, n, with $m \le n \le 3$, we have*

$$D_{1\mathsf{max}} = (m+n-1)\left(\sqrt{D^2 + \left(\frac{2n}{m+n}F\right)^2} + \sqrt{E^2 + \left(\frac{2m}{m+n}F\right)^2} + \left(\frac{2mn}{m+n} - 2\right)F\right).$$

Proof Sketch: When F is given, the delays between the centroids of the two classes and the centroid of all the clients are fixed, i.e., $\|y_1 - y\|$ and $\|y_2 - y\|$ are both fixed, since

$$\frac{N}{n}\|y_1 - y\| = \frac{N}{m}\|y_2 - y\| = F. \tag{18}$$

Therefore we only need to consider how the clients in each class are distributed to maximize the sum of their delays to

the main centroid. That is equivalent to solving

$$\begin{aligned} \max_{x_i \in C_k} \quad & \sum_{x_i} \|x_i - y\| \qquad (19) \\ \text{s.t.} \quad & \sum_{x_i \in C_k} x_i = 0 \quad \text{and} \quad \sum_{x_i \in C_k} \|x_i\| = T, \end{aligned}$$

where T is some constant. We consider when C_k has 1,2 or 3 clients. When C_k has only 1 client, (19) is constant and trivially maximized. When C_k has 2 clients, the two clients x_1 and x_2 are symmetric to each other and it is easy to find that (19) achieves its maximum when $x_1 - x_2$ is orthogonal to y. When C_k has 3 clients, assuming x_3 has the shortest norm so that $\|x_3\| \le T/3$, we rewrite (19) as

$$\max_{x_3} \quad f(x_3) = \|x_3 - y\| + \max_{x_1, x_2} \sum_{i=1}^{2} \|x_i - y\| \quad (20)$$

$$\text{s.t.} \quad x_1 + x_2 = -x_3 \qquad (21)$$

$$\|x_1\| + \|x_2\| = T - \|x_3\|. \qquad (22)$$

Now we calculate $\max_{x_1,x_2} \sum_{i=1}^{2} \|x_i - y\|$. We have

$$\|x_1 - y\| + \|x_2 - y\| \le 2\sqrt{\frac{1}{2}\left(\|x_1 - y\|^2 + \|x_2 - y\|^2\right)}, \quad (23)$$

where the equality is achieved when $\|x_1 - y\| = \|x_2 - y\|$. We also have

$$x_1 - y = \left(x_1 - \frac{x_1 + x_2}{2}\right) + \left(\frac{x_1 + x_2}{2} - y\right), \qquad (24)$$

which also holds for x_2. Hence, by (21) and (24), after some derivation, (23) can be rewritten as

$$\begin{aligned} &\|x_1 - y\| + \|x_2 - y\| \\ &\le \sqrt{2\left(\left\|x_1 + \frac{x_3}{2}\right\|^2 + \left\|x_2 + \frac{x_3}{2}\right\|^2 + 2\left\|\frac{x_3}{2} + y\right\|^2\right)}. \end{aligned}$$

If we fix x_3, $-x_3/2$ is the geometric median of x_1 and x_2, and thus

$$\left\|x_1 + \frac{x_3}{2}\right\| + \left\|x_2 + \frac{x_3}{2}\right\| \le \sum_{i=1}^{2} \|x_i\| = T - \|x_3\|, \quad (25)$$

where the equality is achieved when $x_3 = 0$. Also, since

$$x_1 + \frac{x_3}{2} + x_2 + \frac{x_3}{2} = 0,$$

$\|x_1 + x_3/2\|^2 + \|x_2 + x_3/2\|^2$ achieves the maximum when $\|x_1 + x_3/2\| = \|x_2 + x_3/2\|$.

Therefore, by (23) to (25), we can get $\max_{x_1,x_2}(\|x_1 - y\| + \|x_2 - y\|)$, so (20) can be calculated as

$$f(x_3) \le 2\sqrt{\left\|y - \frac{x_3}{2}\right\|^2 + \left(\frac{T - \|x_3\|}{2}\right)^2} + \|x_3 - y\|, \quad (26)$$

the rhs of which reaches its maximum value $2\sqrt{\|y\|^2 + T^2/4} + \|y\|$ when $x_3 = 0$, which can be shown by analyzing the derivatives of rhs of (26) for all x_3. When $x_3 = 0$, all the above inequalities can achieve equality. Therefore, when C_k has three clients, we can calculate $D_{1\max}$. Finally, using the same analysis as above, we can calculate $D_{1\max}$ for all m and n no more than 3. They all have the same form as (18), completing the proof. ■

Now we can vary D, E, F to find the maximum values of $D_{1\mathsf{max}}/D_2$ under different cases. Since $D_{1\mathsf{opt}} \le D_1 \le D_{1\mathsf{max}}$, we only need to find the maximum of $D_{1\mathsf{max}}/D_2$. Since $D_2 = (m+n-1)(D+E) + mnF$, by Lemma 8, we can express $D_{1\mathsf{max}}/D_2$ as a function of D,E and F. By dividing both the numerator and denominator by F, and denoting D/F as λ, E/F as μ, we get

$$\begin{aligned} \frac{D_{1\mathsf{max}}}{D_2} = \frac{m+n-1}{(m+n-1)(\lambda+\mu) + mn} \cdot \Bigg(& \sqrt{\lambda^2 + \left(\frac{2n}{m+n}\right)^2} \\ & + \sqrt{\mu^2 + \left(\frac{2m}{m+n}\right)^2} + \frac{2mn}{m+n} - 2 \Bigg). \end{aligned} \quad (27)$$

From (9), we have $\frac{\lambda}{m} + \frac{\mu}{n} = \frac{1}{\beta'}$. We maximize $D_{1\mathsf{max}}/D_2$ as a function of λ and μ subject to $\frac{\lambda}{m} + \frac{\mu}{n} = \frac{1}{\beta'}$, and can find that for both $1 = m < n \le 3$ and $1 < m \le n \le 3$, $D_{1\mathsf{max}}/D_2$ given by (27) achieves its maximum value when $\mu = 0$. Substituting $\mu = 0$ into (27) for different m, n proves Proposition 4. ■

Yaochen Hu received his B.Engr. degree from the Department of Electronics and Information Engineering, Huazhong University of Science and Technology, China. Since January 2013, he has been with the Department of Electrical and Computer Engineering at the University of Alberta, where he is pursuing his Ph.D. degree. He is interested in the fields of cloud computing, machine learning, algorithm analysis.

Di Niu received the B.Engr. degree from the Department of Electronics and Communications Engineering, Sun Yat-sen University, China, in 2005 and the M.A.Sc. and Ph.D. degrees from the Department of Electrical and Computer Engineering, University of Toronto, Toronto, Canada, in 2009 and 2013. Since 2012, he has been with the Department of Electrical and Computer Engineering at the University of Alberta, where he is currently an Assistant Professor. His research interests span the areas of cloud computing and storage, multimedia delivery systems, data mining and statistical machine learning for social and economic computing, distributed and parallel computing, and network coding. He is a member of IEEE and ACM.

Zongpeng Li received his B.E. in Computer Science and Technology from Tsinghua University in 1999, and his MSc in Computer Science and PhD in Electrical and Computer Engineering from University of Toronto in 2001 and 2005, respectively. Since 2005, he has been with the Department of Computer Science, University of Calgary, where he is now Professor. Zongpengs research interests include computer networks, network coding, and cloud computing.