Network Load-Aware Content Distribution in Overlay Networks

Seung Chul Han and Ye Xia

***Abstract*—Massive content distribution on overlay networks stresses both the server and the network resources because of the large volumes of data, relatively high bandwidth requirement, and many concurrent clients. While the server limitation can be circumvented by replicating the data at more nodes, the network limitation is far less easy to cope with, due to the difficulty in determining the cause and location of congestion and in provisioning extra resources. In this paper, we present novel schemes for massive content distribution, that assign the clients to appropriate servers, so that the *network load* is reduced and also well balanced, and the network resource consumption is low. Our schemes allow scaling to very large system because the algorithms are very efficient and do not require network measurement, or topology or routing information. The core problems are formulated as partitioning the clients into disjoint subsets according to the degree of interference criterion, which reflects network resource usage and the interference among the concurrent connections. We prove that these problems are NP-complete, and present heuristic algorithms for them. Through simulation, we show that the algorithms are simple yet effective in achieving the design goals.**

***Index Terms*—Node Selection, Server Selection, Overlay Networks, Hypercube, Content Distribution Networks, Peer-to-Peer Networks**

I. Introduction

One of the distinct trends related to the Internet is that it is being applied to the transfer of more and more massive content. This can be long and high-quality streaming content – including high-definition DVD movies sold or rented online, live events, recorded TV programming and long-running video conferencing sessions – mountains of scientific data and all other automatically collected data such as consumer, market or economic data. Most of the recently proposed or operational distribution systems, to which this paper is relevant, rely on the overlay-network approach. These include the web content distribution networks (CDN) (e.g., Akamai [1]) and web caches, various peer-to-peer (P2P) file-sharing networks such as BitTorrent [2], and tree-based end-system multicast (e.g., [3], [4]). Recently, it is increasingly recognized that overlay networks are more than attractive platforms for deploying network applications and services. They can also be used as the substrates of virtual networks for deploying new network architectures [5] or other specialty networks such as those for E-science or grid computing. They also allow decoupling of the operator of the physical network from the service providers. For advanced services, the service providers may compose their own application-level networks with required connectivity, route capacity and reliability.

A significant feature of this type of application-level overlay networks is that data (as apposed to only queries for data) is routed on the overlay network instead of the underlay network. Since the leased overlay network capacity can be costly, prudent use of the network resources is important. On the other hand, massive content distribution over such networks poses some challenges because of the large volumes of data and relatively high bandwidth requirement. Careless design can lead to very inefficient use of the network resources. Furthermore, if the application involves streaming media, the quality experienced by a user is not only affected by the server's capacity but also by the *network load*, which is likely to be contributed by the concurrent streams.

In this paper, we consider the server/client selection problem in content distribution on such overlay networks with the objective of reducing the network load and resource usage. One can imagine that there exist a set of nodes (servers) containing the data and a large number of nodes (clients) requesting all or portions of the data.[1] Due to the capacity limitation, each server can service a subset of all the clients at a time. We call a server and the associated clients serviced by the server a *session*. The problem addressed in this paper is how to assign the clients to appropriate servers subject to the server capacity constraint so that the *network load* is reduced for each session and also well balanced across the sessions, and that the total network resource usage is also reduced. This node selection problem is fundamental in content distribution systems and deserves a systematic investigation. Performance can vary dramatically for different assignments. While node selection under server capacity limitation alone has been widely investigated [6], [4], much less attention has been paid to the network load and resource usage due to the difficulty of the problem.

While the overlay network may in general have an arbitrary topology, in this paper, we consider one of the most popular structured networks, the hypercube, because of its simplicity and versatility. It is well known ([7], [8], [9], [10], [11], [12]) that the structural regularity of the hypercube often makes parallel or distributed algorithms extremely simple and efficient. The same is expected for many networking algorithms on the hypercube. For instance, routing can be performed based on a pre-specified convention without the need of a routing protocol. The chance is high that the hypercube will be the topology of choice for many applications that employ virtual

The authors are with the Computer and Information Science and Engineering Department, University of Florida, 301 CSE Building, P. O. Box 116120, Gainesville, FL 32611-6120. Email: {schan, yx1}@cise.ufl.edu. Ye Xia is the corresponding author.

[1]It is not required that the clients request identical data, e.g., the same file.

networks. To further support its importance, the hypercube network with prefix-based routing as specified in this paper is related to other important structured networks that are based on the distributed hash table (DHT): It is a special case of Plaxton-type networks [13] and a relative to Pastry [14] and Tapestry [15].

Our server selection algorithms on the hypercube can be done efficiently without the knowledge of the current network condition, not even the topology or routing information, hence, avoiding the expensive overhead of network measurement or passing control messages. The only requirement is that each server can obtain the IDs of the clients and other servers through the overlay network. This is usually done by searching a DHT-based distributed directory [16], [17], [15], possibly augmented by some gossip protocol [2], [18] when the network is very dynamic. The efficiency and simplicity of the algorithms allow scaling the network to very large size, capable of handling massive content distribution.

To summarize this paper, the core problems are formulated as optimally partitioning the clients into disjoint subsets (sessions) according to the *degree of interference* (DOI) criterion, which reflects network resource usage, degree of congestion and the interference among the concurrent connections. We prove that these problems are NP-complete by reducing a well-known NP-complete problem, the resource constrained scheduling problem, to the single-server partition problem (SSPP), and then reducing the SSPP to the multi-server partition problem (MSPP). Then, we present novel, heuristic algorithms for solving the problems. This work extends our earlier work on node selection [19], which minimizes the DOI of a single session corresponding to a single server. This paper deals with minimizing the worst DOI of multiple sessions and is naturally applicable to the more general multiple-server situation. Using simulation, we show that the proposed algorithms are simple yet effective in achieving the design goals on both the hypercube and the underlay Internet-like network.

This paper is organized as follows. In Section II, we review previous work on node selection in overlay or P2P networks and introduce the hypercube network. In Section III, we describe two performance measures and some related definitions and facts. In Section IV, we state the two partition problems and prove that they are NP-complete. In Section V, we present novel heuristic algorithms for the problems. We evaluate these algorithms through simulation in Section VI. Finally, the conclusions are drawn in Section VII.

II. Background

A. Previous Work on Server Selection

The literature on node selection is vast. We will mainly review the most relevant studies in overlay content distribution systems, which handle node selection in a variety of (usually ad-hoc) ways.[2] We can roughly classify the overlay content distribution systems into three categories, which are likely to continue their coexistence. In the first category, the infrastructure-based content distribution networks (CDN) (e.g., Akamai [1]) and web caches generally assign the closest server to each client. The second category is tree-based end-system multicast, (e.g., Bayeux [3], CoopNet [4], Narada [20], NICE [21]), where all clients are typically served by the common tree root. But, the node selection problem still occurs during the construction of the multicast tree when some nodes are assigned as the children of existing nodes on the tree. This process is often carried out incrementally as nodes explicitly join the tree, and the assignment is essentially determined by the unicast routing.

The third category is mesh-based P2P systems, which typically employ the techniques of file striping and collaborative download (e.g., BitTorrent [2], PROMISE [22], Splitstream [23], FastReplica [24], Bullet [18]). Their node selection algorithms vary a lot. In SplitStream [23], and FastReplica [24], server selection is essentially done randomly. Other systems employ a server or peer ranking function. A node favors those peers with high ranking. The ranking function may be the nodal load (CoBlitz [25]), the round-trip time (RTT) (ChunkCast [26]), the sending and/or receiving bandwidth to and from each peer (Bullet′ [27], Slurpie [28] and BitTorrent [2]), and the degree of content overlap between the receiver and the server candidate (Bullet [18]). One common practice is that a node initially selects some random peers, but gradually probes other peers and dynamically switches to those with better ranking over the course of downloading. Julia [29] assumes a structured but locality-aware P2P network, where each peer exchanges file chunks with direct neighbors in differential amount, more with closer neighbors. This reduces the total *work*, which is defined as the weighted sum of the total network traffic during the entire distribution process, where the weights are the distances traveled by each bit. In [30], several continuous optimization problems are formulated for peer selection in P2P file download or streaming.

In more traditional server selection literature, [31] presents a dynamic server selection scheme based on instantaneous measurement of the RTT and available bandwidth. Similar research work is also reported in [32], [33].

Only a small subset of these earlier studies are concerned with the bandwidth bottleneck and congestion created by the concurrent connections. But, they do not deal with this issue systematically. (See also [3], [34], [22].) Many others ignore the issue. Very few consider the total network resource usage. [22] proposes a selection scheme based on the traffic condition and available bandwidth of network segments, as well as the performance information of the servers. Unlike ours, this scheme requires the network topology and routing information, and network measurement; the selection objective is to optimize the aggregate rate at the receiver. Our paper is fairly unique in emphasizing both the network-related and user-related performance metrics.

B. The Hypercube Network

In this section, we introduce the definition of the hypercube network and some of its properties that make node selection based on the DOI criterion efficient and easy.

[2]Not all systems frame or handle the node selection problem explicitly. But all should have at least an implicit selection algorithm.

The overlay network we consider is a hypercube with N nodes, numbered $0, 1, \ldots, N-1$. Suppose $N = 2^m$, where m is a positive integer. The node IDs can be expressed as binary strings, and two nodes are linked with an edge if and only if their binary strings differ in precisely one bit. The routing rule is as follows. At each node s and for the destination d, let l be the first bit position, counting from the right, that s and d have different values. Then, the next hop on the route to d is the neighbor of s that differs with s in the l^{th} bit. Consider the example where $N = 2^5$, the source node is 10110 and the destination node is 00000. The route consists of the following sequence of nodes: $10110 \rightarrow 10100 \rightarrow 10000 \rightarrow 00000$.

The hypercube is among the most popular networks studied by researchers due to its useful structural regularity. Many have studied its topological properties and communication capability for parallel and distributed computing applications. (See [7], [8], [9], [10], [11] for a small sample and [12] for a comprehensive textbook treatment.) In the area of communication networks, [35] investigates how the hypercube affects the spread of worms and viruses. [36] considers a multiple access protocol in wireless ad-hoc networks with the hypercube topology. [6] provides analysis of a hypercube-based distributed hash table (DHT) that achieves asymptotically optimal load balance. [37] proposes a failure recovery protocol for structured P2P networks that use hypercube routing. We are not aware of any prior node-selection algorithms on the hypercube that are similar to ours.

In most earlier work, especially in the parallel computing community, the definition of a hypercube only specifies how nodes are connected. In this paper, as in many structured overlay networks, the routing rule is an important part of the definition. With the above routing rule, the hypercube network is a special instance of the Plaxton-type networks [13], which are broad enough to also include Pastry [14] and Tapestry [15], and are fundamentally related to Chord [17] and CAN [16]. We also note that the hypercube clearly requires that every position of the name space is occupied by a node, which is our current assumption. (In Section VI-C, we consider the situation where the name space of the overlay network is not fully populated by nodes.)

A helpful device for visualizing the hypercube and its routing is the embedded tree consisting of all valid routes allowed by the routing rule from node 0 to all nodes, as shown in Fig. 1. It is known that such a tree is a binomial tree [38]. The binomial tree embedded in the hypercube network is a labeled tree, where the label of each node is the node's ID. By the symmetry of the hypercube, there is an embedded binomial tree rooted at every node where the tree paths are the valid routes to the root node. Given the labeled binomial tree rooted at node 0, an easy way to derive the labels for the binomial tree rooted at node $d \neq 0$ is to XOR the node labels of the former tree with d.

Throughout the paper, let us denote the k-level binomial tree rooted at node 0 by $\mathcal{T} = B_k$. Let us denote the ID of node u by $I(u)$. When there is no confusion, we will use u and $I(u)$ interchangeably to denote node u. With the above XOR transformation, the root node 0 is understood as a server.

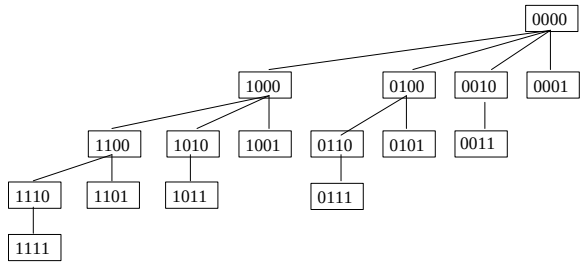

Fig. 1. A labeled 4-level binomial tree B_4

III. Network Load Metrics

We introduce two performance metrics, the worst link stress (WLS) and the degree of interference (DOI), that can capture *network load*.

Definition 1: Suppose $S = \{s_1, \ldots, s_n\}$ is a set of nodes in $\mathcal{T}$ and $E = \{e_1, \ldots, e_l\}$ is the set of edges used by the paths from the root to the nodes in S, called the **S-paths**. The **link stress** of an edge e in $\mathcal{T}$, denoted by $LS(e)$, is the number of S-paths via e. Let E be the set of all edges in $\mathcal{T}$. The **worst link stress** (WLS) is defined as $\max_{e \in E} LS(e)$.

The WLS is the largest number of downloading streams on any link. Assuming the links have roughly identical bandwidth, the WLS is directly related to the worst congestion level in the network, an important performance concern for network and service providers. The minimum WLS corresponds to the best-balanced network load. A balanced use of the network allows it to accommodate more traffic, i.e., to improve the overall throughput. The issue of reducing link stress has also been considered in several other overlay networks [34], [3], [20], [39]. Furthermore, applications such as multimedia streaming are very sensitive to bandwidth availability. Lower WLS implies better performance for these applications. In the case of elastic data transfer, the inverse of the WLS provides an alternative performance perspective: It yields the maximum achievable throughput of the parallel download sessions, which determines the shortest download time.

Definition 2: Suppose $S = \{s_1, \ldots, s_n\}$ is a set of nodes in $\mathcal{T}$. The **degree of interference** (DOI) of nodes $s_1, \ldots, s_n$, denoted by $d(S)$ or $d(s_1, \ldots, s_n)$, is

$$\begin{aligned} d(s_1, \ldots, s_n) &= \sum_{i=1}^{l} (\text{number of } S\text{-paths via } e_i - 1) \\ &= n(\text{average path length}) - l. \end{aligned} \quad (1)$$

For example in Fig. 1, if $u = 1100$, $v = 1101$ and $w = 1010$, then $d(u, v, w) = 3$ because the number of S-paths on edge (1100, 1000) is 2 and on edge (1000, 0000) is 3.

The DOI will be the performance measure to be minimized. It is closely related to the network resource usage and the degree of congestion in the network, both of which are among the most important performance metrics to be minimized. The following gives a justification. Suppose the nodes in S are clients, each receiving a data stream from the server (root). Let the base case for comparison be that every edge involved in the data transmission session sees exactly one stream. The DOI measures the difference between the total number of streams

seen by all edges and the base case. From a slightly different viewpoint, suppose there is one unit of cost associated with a packet traversing an edge, and suppose every client receives one packet from the server. The DOI is the difference between the actual total cost and the cost of the base case where each edge sees exactly one packet.

Note that the total network bandwidth consumed by the base case is less than any other distribution method without network coding. If all clients request the same data, the base case is realized by the tree-based multicast. Then, the comparison is between the total network bandwidth consumed by a unicast-based distribution method and that by a multicast-based method. Therefore, the DOI is useful to measure the total network bandwidth usage by the session beyond the absolute minimum. It can also be interpreted as the degree of overall congestion throughout the network, as apposed to the worst-case link congestion.

Also note that minimizing the DOI is not the same as minimizing the average path length, which is an easier problem. But, the DOI tends to be small when the average path length, hence the average path delay, is small. Since the aggregate bandwidth used by all connections in a session is proportional to the average path length from the clients (See (1).), this is another way to see that DOI measures the total network resource usage, including the total bandwidth and the number of links used by the session. Finally, as will be shown by our experimental results in Section VI, the DOI tends to be correlated with the WLS. Hence, minimizing the DOI usually also reduces the worst-case link congestion. The reason is that the minimum DOI usually occurs when none of the links have many data streams on them, that is, none of the links are highly congested.

Let us denote the *longest common prefix* (LCP) of a set of m-digit binary labels, $R = \{b^1, \ldots, b^k\}$, by $LCP(b^1, \ldots, b^k)$ or $LCP(R)$, and denote the m-digit label equal to the concatenation of $LCP(R)$ with an appropriate number of 0's by $\lambda(R)$ or $\lambda(b^1, \ldots, b^k)$. For instance, $LCP(01110, 01011) = 01$, and $\lambda(01110, 01011) = 01000$. For a set of nodes $S = \{s_1, ..., s_n\}$, $LCP(S)$ or $LCP(s_1, \ldots, s_n)$ are the simplified notations for $LCP(I(s_1), \ldots, I(s_n))$, and $\lambda(S)$ or $\lambda(s_1, \ldots, s_n)$ are the simplified notations for $\lambda(I(s_1), \ldots, I(s_n))$. The following results proven in [19] will be used later.

Definition 3: The **lowest common ancestor** (LCA) of a set of nodes $S = \{s_1, \ldots, s_n\}$, where $n \geq 2$, in a rooted tree is the deepest node in the tree that is a common ancestor of all nodes in S. It is denoted by $LCA(S)$ or $LCA(s_1, \ldots, s_n)$.

Lemma 1: Suppose $S = \{s_1, \ldots, s_n\}$, where $n \geq 2$, is a set of nodes in $\mathcal{T}$. Then, the node ID of $LCA(S)$ is $\lambda(s_1, \ldots, s_n)$.

Lemma 2: Suppose S is a set of two nodes in $\mathcal{T}$. The DOI, $d(S)$, is the depth of $LCA(S)$.

For each node u in $\mathcal{T}$, denote the depth of u by $l(u)$. By convention, the root has depth 0.

Lemma 3: Suppose $S = (s_1, \ldots, s_n)$ is a sorted list of distinct nodes in $\mathcal{T}$ by their IDs, for some $1 < n \leq 2^m$. Then,

$$d(S) = \sum_{i=1}^{n-1} l(LCA(s_i, s_{i+1})).$$

Lemma 4: Suppose $(s_1, \ldots, s_n)$ is a sorted list of distinct nodes in $\mathcal{T}$ by their IDs, for some $1 < n \leq 2^m$. Then,

$$d(s_1, \ldots, s_n) = d(s_1, s_2) + d(s_2, s_3) + \ldots + d(s_{n-1}, s_n).$$

IV. Two Server Selection Problems and Their Hardness

Suppose that there exist one or more servers containing the data and a large number of clients requesting the data. Due to the server capacity limitation (e.g., computational power, outbound bandwidth), each server can service a subset of all the clients at a time. The problem is how to partition the clients into disjoint sessions subject to the server capacity constraint so that the DOI is reduced for each session and also well-balanced across the sessions. We consider two versions of the problem, the single-server partition problem (SSPP) and the multi-server partition problem (MSPP). In the SSPP, the complete client set is partitioned into disjoint subsets (or sessions) and the only server in the system services each session *sequentially*. In the MSPP, the client set is partitioned into disjoint sessions and each session is assigned to a different server to be serviced *in parallel*. These two are the most frequently encountered problems in distribution systems and are fundamental in content distribution networks. They deserve a systematic investigation. In this section, we will formally describe the SSPP and the MSPP. We prove that both problems are NP-complete by reducing a well-known NP-complete problem, the resource constrained scheduling problem, to the SSPP, and then reducing the SSPP to the MSPP.

A. Resource Constrained Scheduling Problem

We consider the following resource constrained scheduling problem. Let $J = \{j_1, \ldots, j_n\}$ be a set of independent jobs. Each job j_i requires a processing time t_i and one unit of resources for its completion. Let $P = \{p_1, \ldots, p_m\}$ be a set of identical non-preemptive processors. There are n units of resources available. Each processor is allocated up to $\lceil \frac{n}{m} \rceil$ units of resources. In other words, each processor can accommodate up to $\lceil \frac{n}{m} \rceil$ jobs due to the limitation of the resources. Let D be a deadline for the jobs. Let α_{max} be the latest completion time of any job, i.e.,

$$\alpha_{max} = \max_{1 \leq i \leq m} \sum_{k: j_k \text{ assigned to } p_i} t_k \ .$$

The resource constrained scheduling problem asks the following question: Is there a schedule (assignment) of all the jobs in J on the m processors in P such that $\alpha_{max} \leq D$?

Lemma 5: The resource constrained scheduling problem is NP-Complete [40].

It can be shown easily that a restricted version of the problem, where $t_1, \ldots, t_n$ are all distinct integers, is still NP-complete. The proof is by reducing the general version of the problem to the restricted version.

B. Single-Server Partition Problem (SSPP)

Given a single server and a set $C = \{c_1, \ldots, c_n\}$ of clients in $\mathcal{T}$, suppose the server can serve a maximum of $\lceil \frac{n}{m} \rceil$ clients simultaneously because of its capacity limitation (e.g, bandwidth, computation power). In order to efficiently serve all the clients, the server partitions the clients into m disjoint groups, $C_1, \ldots, C_m$, and serves each group sequentially. Let D be a targeted DOI cost. Suppose β_{max} is the worst cost of all sessions defined by

$$\beta_{max} = \max_{1 \le i \le m} d(C_i).$$

The SSPP is to find a partition of the client set so as to minimize β_{max}. For convenience, in the following, we will consider the decision version of the SSPP, which asks: Is there a partition of C into m disjoint subsets such that $\beta_{max} \le D$?

Therorem 6: The single-server partition problem is NP-complete.

Proof: The problem is clearly in NP: The verification is just by calculating the DOI of each subset and comparing it to D. The calculation of the DOI of a session can be done in $O(n)$ time by Lemma 1, 2, and 3.

We prove that the single-server partition problem is NP-hard by showing that the resource constrained scheduling problem is polynomial-time reducible to it. The reduction takes as input an instance, $(J = \{j_1, \ldots, j_n\}, T = \{t_1, \ldots, t_n\}, P = \{p_1, \ldots, p_m\}, D)$ of the resource constrained scheduling problem. The output of the reduction is an instance, $(C = \{c_1, \ldots, c_{n+m}\}, m, D)$ of the SSPP. Without loss of generality, we make the assumption that $t_1 < t_2 < \ldots < t_n$. The reduction generates the set $C = \{c_1, \ldots, c_n, c_{n+1}, \ldots, c_{n+m}\}$ of clients, where $I(c_i) < I(c_{i+1})$ for all i, and

$$\begin{cases} d(c_i, c_{i+1}) &= t_i, \quad \text{if } 1 \le i \le n \\ d(c_i, c_{i+1}) &\ge \Phi, \quad \text{if } n < i < n+m \\ d(c_i, c_j) &= t_i, \quad \text{if } 1 \le i \le n,\ n+1 \le j \le n+m, \end{cases} \tag{2}$$

where we define $\Phi = \sum_{i=1}^{n} t_i$ and assume $\Phi > D$. The idea is to force the c_j's, $n+1 \le j \le n+m$, to be assigned in different subsets because they have much larger DOIs.

A reduction example is,

$$\begin{aligned}
c_1 &= \overbrace{1\ldots1}^{t_1}\ \overbrace{0\ \ldots\ 0}^{t_n-t_1}\ \overbrace{0\ldots0}^{\Phi-t_n+m} \\
c_2 &= \overbrace{1\ \ldots\ 1}^{t_2}\ \overbrace{0\ldots0}^{t_n-t_2}\ \overbrace{0\ldots0}^{\Phi-t_n+m} \\
&\ldots \\
c_n &= \overbrace{1\quad\ldots\quad\ldots\quad1}^{t_n}\ \overbrace{0\ldots0}^{\Phi-t_n+m}, \text{ and} \\
c_{n+1} &= \overbrace{1\quad\ldots\quad\ldots\quad1}^{\Phi}\ \overbrace{0\ldots01}^{m} \\
c_{n+2} &= \overbrace{1\quad\ldots\quad\ldots\quad1}^{\Phi}\ \overbrace{0\ldots10}^{m} \\
&\ldots \\
c_{n+m} &= \overbrace{1\quad\ldots\quad\ldots\quad1}^{\Phi}\ \overbrace{1\ldots00}^{m}.
\end{aligned}$$

It is easy to see that the reduction can be performed in polynomial time. The set C contains $n+m$ elements, each of which has $\Phi + m$ bits.

We now show that $\alpha_{max} \le D$ if and only if $\beta_{max} \le D$. First, suppose $\alpha_{max} \le D$. Then, we can generate m disjoint subsets of C, $C_1, \ldots, C_m$, where, for $1 \le i \le m$,

$$C_i = \{c_k |\ j_k \text{ is assigned to } p_i, 1 \le k \le n\} \ \cup\ \{c_{n+i}\}.$$

Let α_i and β_i be the completion time of processor p_i and the DOI of C_i, respectively. Then, the completion time of processor p_i is equal to the DOI of C_i. By Lemma 4 and (2),

$$\alpha_i = \sum_{k:j_k \text{ assigned to } p_i} t_k = d(C_i) = \beta_i. \tag{3}$$

Hence,

$$\alpha_{max} = \max_{1 \le i \le m} \alpha_i = \max_{1 \le i \le m} \beta_i = \beta_{max}. \tag{4}$$

Thus, if $\alpha_{max} \le D$, then $\beta_{max} \le D$.

Conversely, suppose $\beta_{max} \le D$. Take any such partition $C_1, \ldots, C_m$. None of these subsets can contain more than one c_i, $n+1 \le i \le n+m$, since the DOI of the subset would then be greater than D. Let J_i be the set of jobs corresponding to $C_i - \{c_{n+i}\}$, and assign J_i to p_i, $1 \le i \le m$. $J_1, \ldots, J_m$ will be a valid assignment for the resource constrained scheduling problem satisfying $\alpha_{max} \le D$, because (3) and (4) still hold. ■

C. Multi-Server Partition Problem (MSPP)

The problem addressed in this section is about m servers and n clients, where $1 < m < n$. Given a set $S = \{s_1, \ldots, s_m\}$ of servers and a set $L = \{l_1, \ldots, l_n\}$ of clients in the hypercube, each server selects $\lceil \frac{n}{m} \rceil$ clients to serve with no clients left unserved. Let us first define the DOI cost of a set of clients with respect to a particular server, say s.

Definition 4: Consider a root node s and a set of nodes $Q = \{q_1, \ldots, q_n\}$. Let $\oplus$ represent the bit-wise XOR operator.

$$d_s(Q) = d(s \oplus q_1, \ldots, s \oplus q_n).$$

Let L_i be the set of clients selected by server s_i, $1 \le i \le m$, $L_i \subseteq L$, and let D be a targeted DOI cost. Let δ_{max} be the worst cost defined by,

$$\delta_{max} = \max_{1 \le i \le m} d_{s_i}(L_i).$$

The MSPP is to find a partition of the client set so as to minimize δ_{max}. Again for convenience, we will consider the decision version of the MSPP, which asks: Is there a partition of L into m disjoint subsets (sessions), each served by a different server such that $\delta_{max} \le D$?

Therorem 7: The multi-server partition problem is NP-complete.

Proof: See Appendix A. ■

V. Partition Algorithms

A. Single-Server Partition Algorithm

1) Recursive algorithm for the power-of-two case: Algorithm 1 can be used to find sub-optimal solutions for the SSPP in the special case where the total number of clients, n, and the number of partitions, k, are each a power of two. It takes two arguments as input, a set of clients sorted in increasing order of their IDs and the number of partitions to be created. Inside the loop between line 7 and 12, the server calculates the DOI of each client with its predecessor in S,[3] selects a pair of adjacent clients whose DOI is the maximum, splits them into two new sets, S_1 and S_2, and removes the two clients from the set S. The procedure repeats until the set S becomes empty. In line 13 and 14, the same procedure described above is applied recursively to the sets S_1 and S_2, with the number of partitions equal to $k/2$ for each set. The intuition is that, by Lemma 4, the DOI of a set is equal to the sum of the pair-wise DOIs of adjacent clients, assuming the client list is sorted. In order to reduce the DOI of each of the two subsets, we may greedily split adjacent client-pairs that contribute large DOIs into the separate subsets. For the power-of-two case, our simulation results show that Algorithm 1 almost always achieves the optimal WLS. We suspect that it also achieves near optimal DOI. But, it is difficult to verify this with experiments since there is no known algorithm for finding the optimal DOI without enumerating all possibilities.

Algorithm 1 SSP(S, k)

1: **input**: $S = \{s_1, \ldots, s_n\}$, a client set with increasing order of IDs; an integer $k \leq n$
2: **if** $k = 1$ **then**
3: return
4: **end if**
5: $S_1 \leftarrow \emptyset$
6: $S_2 \leftarrow \emptyset$
7: **while** $S \neq \emptyset$ **do**
8: Calculate the DOI of each client in S with its predecessor in S
9: Find s_i in S who has the maximum DOI with its predecessor, s_j, in S
10: Add s_j to set S_1 and s_i to set S_2
11: Remove s_i, s_j from S
12: **end while**
13: SSP(S_1, $\frac{k}{2}$)
14: SSP(S_2, $\frac{k}{2}$)
15: **return**

2) Running time of Algorithm 1: The running time of Algorithm 1 is as follows. Inside the *while* loop between line 5 and 12, calculating the DOI of each client with its predecessor and finding the largest one take $O(n)$ by Lemma 1 and 2. Hence, the entire *while* loop takes $O(n^2)$ time. Thus, the recurrence for the running time is,

$$T(n,k) = \begin{cases} 1 & \text{if } k = 1; \\ 2T(\frac{n}{2}, \frac{k}{2}) + O(n^2) & \text{otherwise.} \end{cases}$$

[3]By convention, the first client's DOI with its predecessor is 0.

By expanding the recursion, the running time of the algorithm can be shown to be $O(n^2)$.

For the general case of arbitrary n and k, we do not have an algorithm yet. But, we do have one for the more important multi-server problem, which is shown next.

B. Multi-Server Partition Algorithm

Suppose there are m servers and n clients, where $m < n$. Each server is to select $k = \lceil \frac{n}{m} \rceil$ clients to serve.

1) The building block - a single-server selection algorithm: Our algorithm will be built upon Algorithm 2, given in [19], which addresses the problem of how a single server selects a subset of the clients to serve from a larger candidate set. In Algorithm 2, node 0 is assumed to be the server. More precisely, given a set of n clients and a positive integer k, $k \leq n$, Algorithm 2 allows the server to select an optimal set of k clients that has the minimum DOI. The running time of the algorithm is $O(n \log n)$.

Algorithm 2 CLIENT-SELECTION-DOI(S, k)

1: **input**:$S = \{s_1, \ldots, s_n\}$,a client set;an integer $k \leq n$
2: **output**: G, an optimal set with k clients w.r.t. the DOI cost
3: $G \leftarrow \emptyset$
4: Sort S in increasing order of the node (client) ID
5: $d[1] = 0$
6: **for** $i = 2$ to n **do**
7: $d[i] = d(s_{i-1}, s_i)$
8: **end for**
9: **for** $i = 1$ to k **do**
10: Find s_j in S whose $d[j]$ is the minimum
11: Add s_j to the client set G
12: Remove s_j from S
13: **end for**
14: **return** G

2) Multi-server partition (MSP) algorithm: Let $S = \{s_1, \ldots, s_m\}$ be the set of servers and $L = \{l_1, \ldots, l_n\}$ be the set of clients. The algorithm consists of two phases.

- **PHASE 1**: The servers sequentially select clients in an arbitrary order. At its turn, a server selects k clients from the current client (candidate) pool using Algorithm 2. After the server makes its selection, the selected clients are removed from the client pool. At the end of this phase, every server has its own set of clients.
- **PHASE 2**: A threshold value is determined based on the DOI values of all sessions. In this paper, we only consider a simple case: The average DOI across the sessions is computed, and the threshold value is set at twice the average DOI. All sessions whose DOI is above the threshold value are identified. For each such session, the server makes a fresh selection of clients using Algorithm 2, from the client set L. For each selected client that is not already in the server's client set from the first phase, the server exchanges an unwanted client with another server who has that wanted client.

3) Running time and message complexity: Phase 1 requires $O(mn \log n)$ running time. However, in practice, the servers will run this phase separately for the most part, with the help of a distributed protocol for coordination. In the simplest implementation, one of the servers is elected as the *coordinator* using any typical leader-election algorithm. The coordinator is responsible for serializing the client-selection operations taken by the servers and for maintaining the server-to-client assignment information temporarily. It can achieve the serialization by granting permission to each of the servers according to an ordered list; this takes only several messages per server. After a server selects its clients, it reports the selected clients back to the coordinator. Then, the coordinator sends messages to the subsequent servers in the ordered list for them to remove those clients from their candidate sets. The last server in the list receives m such messages, which is the worst-case message complexity experienced by any ordinary server. However, the coordinator needs to transmit a total of $m(m+1)/2$ such messages. Implemented this way, the running time at each server is $O(n \log n)$. But, the entire phase 1 takes $O(mn \log n)$ running time.

In phase 2, each of the servers with a high DOI value selects clients again from the original candidate set, independently and in parallel. This takes $O(n \log n)$ running time at each of these servers. Then, each of these servers reports the selected clients back to the coordinator. For each such server, the coordinator checks if any of its clients has already been taken by another server as the result of phase 1. In a straightforward implementation, this step takes $O(m^2 \log m)$ total running time. (Note that, in general, $m \ll n$.) If any of the clients is assigned to two servers, the coordinator requests the two servers to initiate a client exchange operation. Overall, there are no more than $O(n)$ such operations, which take at most $O(n)$ messages.

To make the presentation easier, we assume the algorithm is carried out before actual content distribution. The worst-case waiting time before a client can download the content is given by the running time of the algorithm. However, in actual operation, a client can be served immediately once it is assigned to some server during the execution of the algorithm, even if the assignment may be temporary. Some clients can start the download as early as the beginning of phase 1. Since the servers execute the algorithm sequentially in phase 1, in the worst case, some clients have to wait until the end of phase 1 and the waiting duration is $O(mn \log n)$. In phase 2, it can happen that a client is reassigned to another server. This only causes temporary service interruption. The service is resumed as soon as the client makes a connection to the new server.

VI. EVALUATION

In this section, we present simulation results demonstrating the benefits of the proposed schemes. We compare our algorithms with (1) the *random* scheme where, for each session, the server chooses k clients uniformly at random from the client pool, (2) the *closest* scheme, where the server chooses k clients with the shortest round-trip times (RTTs) from the client pool. These two schemes are the most typical strategies used in related studies [30], [22], [32], [33].

The running times are as follows. In the random scheme, the running time is $O(k)$ for each server, if it selects $k = n/m$ clients. In the closest scheme, it takes $O(n)$ time for each server to calculate the distances from the server to the clients, and $O(n \log n)$ time to sort them. Hence, the running time of the closest scheme is $O(n \log n)$ for each server. These should be compared to the running times of our algorithms presented in Section V. The worst-case waiting time by a client before it can download the content is determined by the algorithm running time. In the random scheme, the waiting time is $O(k)$; in the closest scheme, it is $O(n \log n)$.

Two metrics that can capture the performance objective are the WLS and the DOI of each session. We report the *worst* WLS and DOI across all sessions.

A. Single-Server Partition

In this experiment, we show the goodness of the client partition scheme for the single-server case. Here, Algorithm 1 is performed on the overlay hypercube network, where each overlay link is an end-to-end IP path. We will show by simulation that, despite being implemented at the overlay level, Algorithm 1 leads to reduced link stress and DOI for both the overlay links and the underlying physical links.

1) Performance on the overlay network: First, we show the simulation results at the overlay hypercube network. The size of the overlay network is 4096 nodes, with 1024 clients uniformly distributed over the hypercube. We vary the size of the each partition from 8 to 128 clients. The delays on the links are assumed to be the same constant. The simulator counts the number of data streams on each overlay link. We run each experiment with different seeds for the random number generator and present the average of the results obtained.

Fig. 2 shows the distribution of the WLS of the sessions when the size of each partition is 32 clients. For each simulation run, we sort the measured WLS for the sessions in increasing order. Fig. 2 reveals that Algorithm 1 yields results very close to the ideal one. It brings the WLS of the sessions very close to each other, leading to very well balanced network load across the sessions. The WLS of the worst session is also much smaller than the worst WLS in either the random scheme or the closest scheme.

Fig. 3 shows the distribution of the DOI of the sessions when the size of each partition is 32 clients. For each simulation run, we sort the measured DOI for the sessions in increasing order. Algorithm 1 yields the most uniformly distributed DOI. Moreover, the average DOI of all sessions is much lower than the other two schemes. The saving in network bandwidth usage is substantial. We also conducted simulations for different partition sizes and observed the same trend.

Fig. 4 shows the WLS of the worst session, for different partition sizes under the three schemes. In all cases, Algorithm 1 generates the best results, and the closest scheme performs the worst. Algorithm 1 is effective in reducing the bottleneck stress. Comparing with the closest scheme, improvement of up to 43% is observed.

Fig. 5 shows the DOI of the overlay links for the worst session. When compared with the random scheme or the

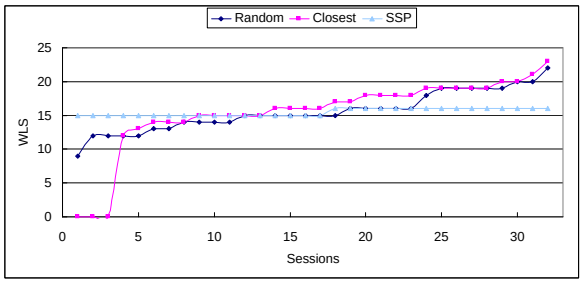

Fig. 2. Distribution of the WLS at the *overlay* network for the single-server case. The size of each partition is 32 clients.

Fig. 3. Distribution of the DOI at the *overlay* network for the single-server case. The size of each partition is 32 clients.

closest scheme, Algorithm 1 clearly produces better results over all the partition sizes. For example, when the partition size is 16, the worst DOI can be reduced from 58 to 20, if we use Algorithm 1 rather than the closest scheme.

2) Performance on the physical network: In this case, the overlay hypercube network is built on top a physical network represented by the transit-stub network model [41]. The transit-stub model uses a 2-level hierarchy of routing domains with the transit domains interconnecting the lower-level stub domains. The model is often used to represent part of the physical Internet. The simulations run on network topologies consisting of 4200 nodes split into 10 Autonomous Systems (AS). The average diameter of the network is 10. The size of the overlay network is 4096 nodes, with 1024 clients uniformly distributed over the name space. We show the simulation results for the same performance metrics.

Fig. 6 shows the number of streams on the most stressed physical link, i.e., the WLS of the physical link, of the worst session. In all cases, Algorithm 1 generates the best results.

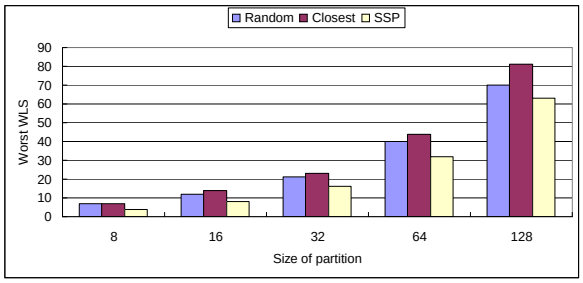

Fig. 4. The worst WLS of the *overlay* link for the single-server case

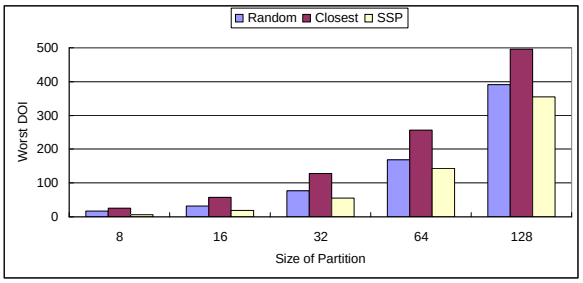

Fig. 5. The worst DOI of the *overlay* links for the single-server case

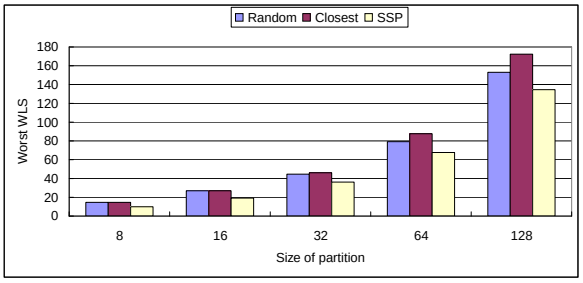

Fig. 6. The worst WLS of the *physical* link for the single-server case

Improvement of up to 35% is observed. Fig. 7 shows the DOI of the physical links for the worst session. The trend is the same as in Fig. 5, which is for the overlay links.

We also observed that the distributions of the WLS or DOI on the physical links across the sessions show the same trends as the overlay-link cases. Overall, these results have demonstrated the effectiveness of Algorithm 1 in reducing the link stress and DOI at the physical links as well as at the overlay links.

B. Multi-Server Partition

1) Performance on the overlay network: In this experiment, the overlay network has 4096 nodes, a quarter of which are clients. We vary the number of servers from 8 to 128. The servers are ordered arbitrarily in the client-selection process. The clients and servers are both uniformly distributed over the 4096 nodes.

Fig. 8 depicts the WLS of the overlay link for the worst session, for different numbers of servers. It shows that our multi-server partition algorithm (labeled MSP) is effective in

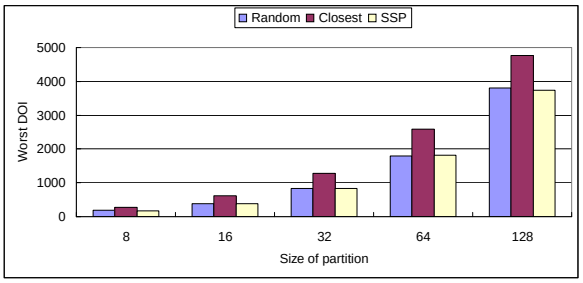

Fig. 7. The worst DOI of the *physical* links for the single-server case

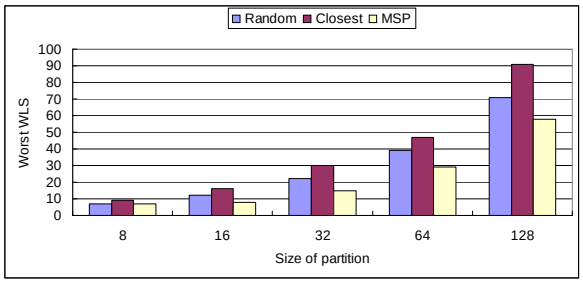

Fig. 8. The worst WLS of the *overlay* link for the multi-server case

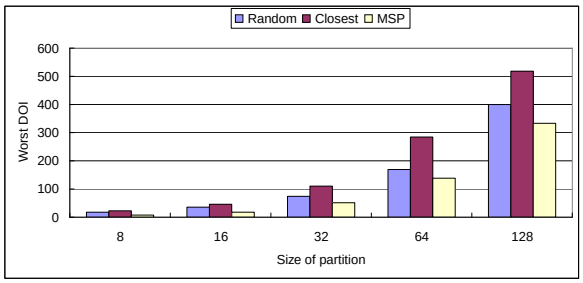

Fig. 9. The worst DOI of the *overlay* links for the multi-server case

reducing the stress at the bottleneck link. For example, when the number of servers is 32, the worst WLS can be reduced from 30 to 15, if we use our algorithm rather than the closest scheme. Fig. 9 shows the DOI of the overlay links for the worst session.

Fig. 10 and 11 show the distribution of the WLS and DOI across the sessions when the size of each partition is 32 clients. One can observe that our MSP algorithm can reduce the average WLS or DOI by as much as a half.

2) Performance on the physical network: The simulation results on the transit-stub model are plotted in Fig. 12 and 13. Fig. 12 depicts the WLS of the physical link for the worst session. Fig. 13 shows the DOI of the physical links for the worst session. One can observe that the general trend of the distributions of the WLS and DOI on the physical links are the same as the overlay cases.

C. Name Space Not Fully Occupied

In this subsection, we consider the situation where the name space is not fully populated by nodes. This is the most likely

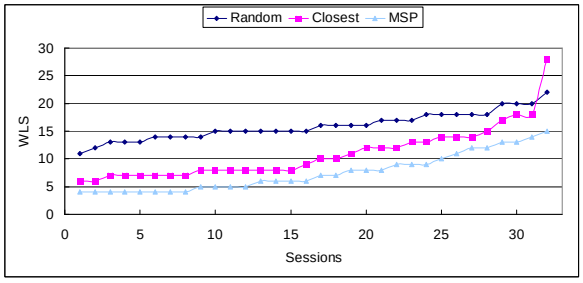

Fig. 10. Distribution of the WLS across the sessions at the *overlay* network for the multi-server case. The size of each partition is 32 clients.

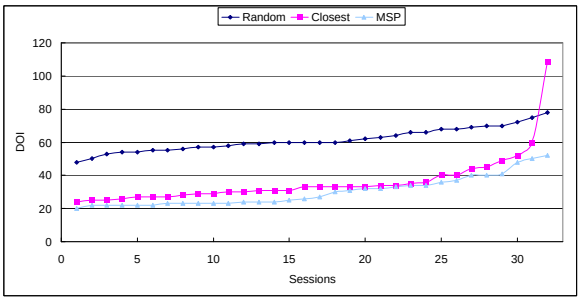

Fig. 11. Distribution of the DOI across the sessions at the *overlay* network for the multi-server case. The size of each partition is 32 clients.

Fig. 12. The worst WLS of the *physical* link for the multi-server case

scenario for an overlay network where nodes join and leave frequently. We modify the routing as follows. When a message for destination d arrives at a node, the next node will be the first available node on the hypercube path to d. In this experiment, the physical network is again modeled as a transit-stub network with the same parameters as in Section VI-A2 and VI-B2. The overlay network has a name space size 4096, but only contains 2048 actual nodes, uniformly distributed throughout the name space.

The performance results for the single-server case are shown in Fig. 14 and 15. The performance results for the multi-server case are shown in Fig. 16 and 17. It can be observed that, compared with the results for the fully occupied networks in Section VI-A2 and VI-B2, the performance gains by our algorithms hold up.

VII. Conclusions

In this paper, we make an in-depth investigation on the issue of client/node selection, which is a fundamental problem in massive content distribution on overlay networks. We

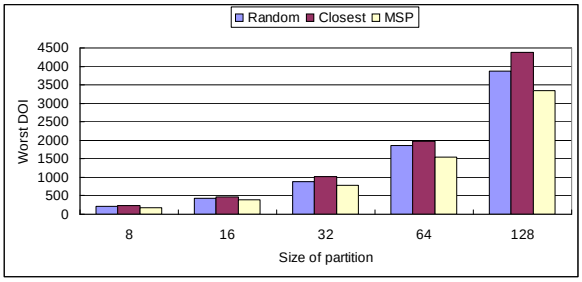

Fig. 13. The worst DOI of the *physical* links for the multi-server case

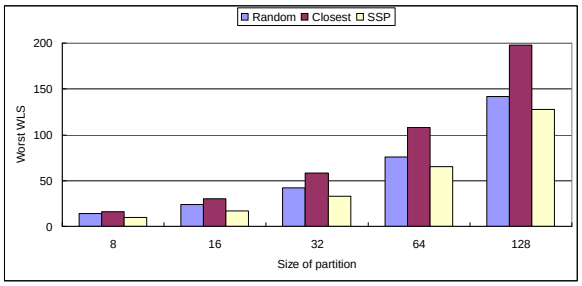

Fig. 14. The worst WLS of the *physical* link for the single-server case. The overlay network has a name space size 4096 and contains 2048 nodes.

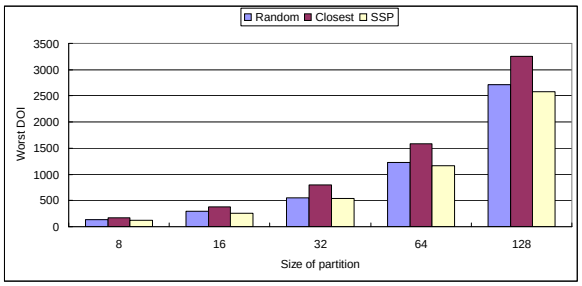

Fig. 15. The worst DOI of the *physical* links for the single-server case. The overlay network has a name space size 4096 and contains 2048 nodes.

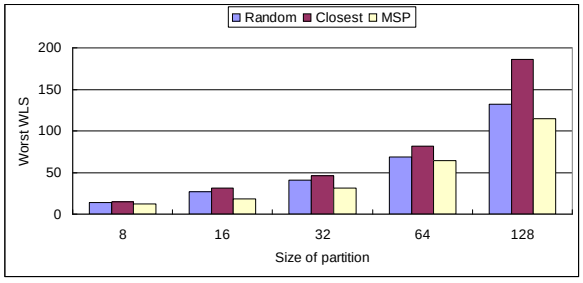

Fig. 16. The worst WLS of the *physical* link for the multi-server case. The overlay network has a name space size 4096 and contains 2048 nodes.

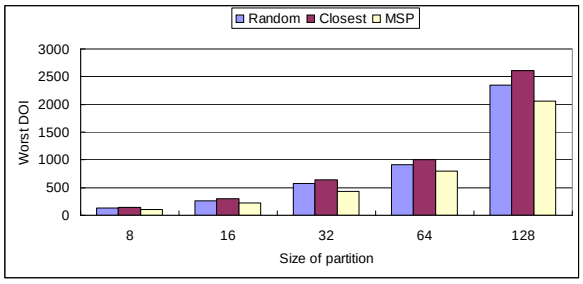

Fig. 17. The worst DOI of the *physical* links for the multi-server case. The overlay network has a name space size 4096 and contains 2048 nodes.

envision a hypercube as the overlay network and give novel server/client selection schemes. As a result of the schemes, the *network load* of each session is reduced and also well balanced across the sessions and the network resource consumption is low. Our schemes do not require measurement of some network performance metrics or the network topology or routing information. The assumption is that each server can obtain the IDs of the clients and other servers through the overlay network. Being free from network measurement and having low implementation complexity make the algorithms scalable.

In the paper, the core problems are formulated as partitioning the clients into disjoint subsets according to the degree of interference criterion, which reflects network resource usage and the interference among the concurrent connections. We prove that these problems are NP-complete and present heuristic algorithms for them. Using simulation, we show that the algorithms are simple yet effective in achieving the design goals, particularly in reducing the worst-case link stress and the network bandwidth usage. Moreover, lower WLS implies less network congestion created by concurrent streams, hence, better quality of for streaming applications.

Due to the close relationship between the hypercube and the popular Tapestry, Pastry and Chord networks, the problem formulation, algorithms, and performance metrics in this paper are relevant and may be extended to those networks. The methodologies proposed in this paper can have many applications, including P2P file downloading or media streaming, multicast group member selection, and edge-mapping in content distribution networks.

REFERENCES

[1] Akamai Website, http://www.akamai.com.
[2] BitTorrent Website, http://www.bittorrent.com/.
[3] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. Kubiatowicz, "Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination," in *Proceedings of the 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)* , June 2001.
[4] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchar, "Distributing streaming media content using cooperative networking," in *IEEE/ACM NOSSDAV*, Miami, FL, May 2002.
[5] S. Shenker, L. Peterson, and J. Turner, "Overcoming the internet impasse through virtualization," in *ACM Hot Nets*, 2004.
[6] M. Adler, E. Halperin, V. Vazirani, and R. M. Karp, "A stochastic process on the hypercube with applications to peer-to-peer networks," in *ACM Symposium on Theory of Computing*, San Diego, CA, June 2003.
[7] S. L. Johnsson and C.-T. Ho, "Optimum broadcasting and personalized communications in hypercubes," *IEEE Transactions on Computers*, vol. 38, no. 9, 1989.
[8] S. S. Gupta, D. Das, and B. P. Sinha, "The generalized hypercube-connected-cycle: An efficient network topology," in *Third International Conference on High-Performance Computing (HiPC '96)*, Trivandrum, India, December 1996.
[9] J.-H. Park, H.-C. Kim, and H.-S. Lim, "Fault-hamiltonicity of hypercube-like interconnection networks," in *19th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, April 2005.
[10] O. Beaumont, L. Marchal, and Y. Robert, "Broadcast trees for heterogeneous platforms," in *19th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, Denver, Colorado, April 2005.
[11] L.-J. Fan, C.-B. Yang, and S.-H. Shiau, "Routing algorithms on the bus-based hypercube network," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, 2005.
[12] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann, 1991.

[13] C. Plaxton, R. Rajaraman, and A. Richa, "Accessing nearby copies of replicated objects in a distributed environment," in *Proceedings of the 9th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA '97)*, Newport, Rhod Island, June 1997, pp. 311–320.

[14] A. Rowstron and P. Druschel, "Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems," in *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware '01)*, Heidelberg, Germany, November 2001.

[15] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: a resilient global-scale overlay for service deployment," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, pp. 41–53, Jan. 2004.

[16] S. Ratnasamy, P. Francis, M. Hanley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proc. ACM SIGCOMM '2001*, San Diego, CA, August 2001, pp. 161–172.

[17] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for Internet applications," in *Proc. ACM SIGCOMM '2001*, San Diego, CA, August 2001, pp. 149–160.

[18] D. Kostić, A. Rodriguez, J. Albrecht, and A. Vahdat, "Bullet: high bandwidth data dissemination using an overlay mesh," in *Proceedings of 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, October 2003.

[19] S. C. Han and Y. Xia, "Constructing an optimal server set in structured peer-to-peer network," *IEEE Journal on Selected Areas in Communications*, vol. 25, no. 1, pp. 170–178, Jan. 2007.

[20] Y.-H. Chu, S. G. Rao, and H. Zhang, "A case for end system multicast," in *Proceedings of ACM SIGMETRICS 2000*, June 2000.

[21] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," in *Proceedings of ACM SIGCOMM 2002*, August 2002.

[22] M. Hefeeda, A. Habib, B. Boyan, D. Xu, and B. Bhargava, "Promise: Peer-to-peer media streaming using CollectCast," in *Proc. of ACM Multimedia*, Berkeley, CA, November 2003.

[23] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh, "Splitstream: High-bandwidth multicast in cooperative environments," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, October 2003.

[24] L. Cherkasova and J. Lee, "Fastreplica: Efficient large file distribution within content delivery networks," in *Proceedings of the 4th USITS*, Seattle, WA, March 2003.

[25] K. Park and V. S. Pai, "Scale and performance in the coblitz large-file distribution service," in *Proceedings of the 3rd USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI 06)*, San Jose, CA, May 2006.

[26] B. Chun, P. Wu, H. Weatherspoon, and J. Kubiatowicz, "Chunkcast: An anycast service for large content distribution," in *Proceedings of the Internaltional Workshop on Peer-to-Peer Systems (IPTPS)*, February 2006.

[27] D. Kostić, R. Braud, C. Killian, E. Vandekieft, J. W. Anderson, A. C. Snoeren, and A. Vahdat, "Maintaining high bandwidth under dynamic network conditions," in *Proceedings of USENIX Annual Technical Conference*, 2005.

[28] R. Sherwood, R. Braud, and B. Bhattacharjee, "Slurpie: A cooperative bulk data transfer protocol," in *Proceedings of IEEE Infocom*, Hong Kong, March 2004.

[29] D. Bickson, D. Malkhi, and D. Rabinowitz, "Efficient large scale content distribution," in *Proceedings of the 6th Workshop on Distributed Data and Structures (WDAS'2004)*, Lausanne, Switzerland, July 2004.

[30] M. Adler, R. Kumar, K. Ross, D. Rubenstein, T. Suel, and D. Yao, "Optimal peer selection for P2P downloading and streaming," in *Proceedings of IEEE Infocom*, Miami, FL, March 2005.

[31] R. L. Carter and M. Crovella, "Server selection using dynamic path characterization in wide-area networks," in *Proceedings of IEEE Infocom*, 1997, pp. 1014–1021.

[32] P. Rodriguez, A. Kirpal, and E. Biersack, "Parallel-access for mirror sites in the Internet," in *Proceedings of IEEE Infocom*, Tel Aviv, Israel, March 2000.

[33] A. Shaikh, R. Tewari, and M. Agrawal, "On the effectiveness of DNS-based server selection," in *Proceedings of IEEE Infocom*, Anchorage, AK 2001.

[34] M. C. Castro, M. B. Jones, A.-M. Kermarrec, A. Rowstron, M. Theimer, H. Wang, and A. Wolman, "An evaluation of scalable application-level multicast built using peer-to-peer overlays," in *Proceedings of IEEE Infocom*, San Francisco, April 2003.

[35] A. Ganesh, L. Massoulie, and D. Towsley, "The effect of network topology on the spread of epidemics," in *Proceedings of IEEE Infocom*, Miami, FL, March 2005.

[36] Z. Tang and J. J. Garcia-Luna-Aceves, "Hop-reservation multiple access (HRMA) for ad-hoc networks," in *Proceedings of IEEE Infocom*, New York, NY, March 1999.

[37] S. S. Lan and H. Liu, "Failure recovery for structured P2P networks: protocol design and performance evaluation," in *ACM SIGMETRICS/Performance 2004*, New York, June 2004.

[38] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. The MIT Press, 2001.

[39] M. Zhang, L. Zhao, Y. Tang, J.-G. Luo, and S.-Q. Yang, "Large-scale live media streaming over peer-to-peer networks through global Internet," in *Proceedings of the ACM Workshop on Advances in Peer-to-Peer Multimedia Streaming*, 2005.

[40] M. R. Garey and R. L. Graham, "Bound for multiprocessor scheduling with resource constraints," *SIAM Journal on Computing*, vol. 4, no. 2, pp. 187–200, June 1975.

[41] K. Calvert, M. Doar, and E. W. Zegura, "Modeling Internet topology," *IEEE Communications Magazine*, June 1997.

APPENDIX A
PROOF OF THEROREM 7

Therorem 7: The multi-server partition problem is NP-complete.

Proof: This problem is obviously in NP, since it is easy to verify whether $d_{s_i}(L_i) \leq D$, $1 \leq i \leq m$.

We prove that the multi-server partition problem is NP-hard by showing that the SSPP is polynomial-time reducible to it. The reduction takes as input an arbitrary instance of the SSPP, $(C = \{c_1, \ldots, c_n\}, m, D)$. The output of the reduction is an instance of the MSPP, $(S = \{s_1, \ldots, s_m\}, L = \{l_1, \ldots, l_n\}, D)$. Suppose the name space is r bits. The reduction generates a set $S = \{s_1, \ldots, s_m\}$ of servers, where

$$
\begin{aligned}
s_1 &= \overbrace{0\ldots0}^{r}\ \overbrace{0\ \ldots\ \ 0}^{m}\\
s_2 &= \overbrace{0\ldots0}^{r}\ \overbrace{0\ \ldots\ \ 01}^{m}\\
s_3 &= \overbrace{0\ldots0}^{r}\ \overbrace{0\ \ldots\ \ 10}^{m}\\
&\ldots\\
s_m &= \overbrace{0\ldots0}^{r}\ \overbrace{01\ \ldots\ \ 0}^{m},
\end{aligned}
$$

and it generates a set $L = \{l_1, \ldots, l_n\}$ of clients, where

$$
\begin{aligned}
l_1 &= I(c_1)\overbrace{0\ \ldots\ \ 0}^{m}\\
l_2 &= I(c_2)\overbrace{0\ \ldots\ \ 0}^{m}\\
l_3 &= I(c_3)\overbrace{0\ \ldots\ \ 0}^{m}\\
&\ldots\\
l_n &= I(c_n)\overbrace{0\ \ldots\ \ 0}^{m}.
\end{aligned}
$$

The reduction can be performed in polynomial time.

We now show that $\beta_{max} \leq D$ if and only if $\delta_{max} \leq D$. First, suppose $(C = \{c_1, \ldots, c_n\}, m, D)$ is a "yes" instance of the SSPP and the subsets $C_1, \ldots, C_m$ are the partition. We

can partition L into the corresponding m subsets, $L_1, \ldots, L_m$, where

$$L_i = \{l_k | c_k \in C_i, 1 \leq k \leq n\},\ 1 \leq i \leq m.$$

We assign L_i to server s_i, $1 \leq i \leq m$. Then,

$$d_{s_i}(L_i) = d(C_i).$$

Thus, $L_1, \ldots, L_m$ will work in the MSPP by the assumption.

Conversely, suppose $(S = \{s_1, \ldots, s_m\}, L = \{l_1, \ldots, l_n\}, D)$ is a "yes" instance of the MSPP, and $L_1, \ldots, L_m \subset L$ are the corresponding assignments of the clients to the servers. Take any server s_i and the associated client set L_i. We have,

$$d_{s_i}(L_i) = d(L_i),$$

where $d(L_i)$ is the DOI of the clients in L_i with respect to node 0. Thus, $L_1, \ldots, L_m$ form a "yes" instance for the SSPP.

■