Network Load-Aware Content Distribution in Overlay Networks

G.Suresh[1]
[1]Department of Computer Science Engineering
[1]Saveetha School of Engineering, Saveetha University Chennai-602105, India

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.

Keywords: 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 itis 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 videoconferencing 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 Bit Torrent [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.

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.

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], Coop Net [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 (Bullet0 [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 he 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].

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 The overlay network we consider is a hypercube with Nnodes, numbered 0; 1; : : : ;N ¡1. Suppose N = 2m, 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 the bit. Consider the example where N = 25, the source node is 10110 and the destination node is 00000. The route consists of the following sequence of nodes: 10110 ! 10100 ! 10000 ! 0000.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. Wearer not aware of any prior node-selection algorithms on the hypercube that is 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 networks special instances of the Plax ton-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 6= 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 T = Bk. 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.

III. 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 = fj1; : : : ; jng be a set of independent jobs. Each job ji requires a processing time ti and one unit of resources for its completion. Let P = fp1; : : : ; pmg be a set of identical non-preemptive processors. There are n units of resources available. Each processor is allocated up to d n me units of resources. In other words, each processor can accommodate up to d n me jobs due to the limitation of the resources. Let D be a deadline for the jobs. Let ®max be the latest completion time of any job, i.e., ®max = max1·i·m Xk:jk assigned to pi tk : 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 ®max · 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 t1; : : : ; tn are all distinct integers, is still NP complete. The proof is by reducing the general version of the problem to the restricted version5

B. Single-Server Partition Problem (SSPP)

Given a single server and a set C = fc1; : : : ; cng of clients in T , suppose the server can serve a maximum of d nme 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, C1; : : : ;Cm, and serves each group sequentially. Let D be a targeted DOI cost. Suppose ¯max is the worst cost of all sessions defined by max = max1·i·md(Ci):

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

Theorem 6: The single-server partition problem is NPcomplete.

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 = fj1; : : : ; jng; T = ft1; : : : ; tng; P =fp1; : : : ; pmg;D) of the resource constrained scheduling problem. The output of the reduction is an instance, (C =
fc1; : : : ; cn+mg; m;D) of the SSPP. Without loss of generality, we make the assumption that t1 < t2 < : : : < tn. The reduction generates the set C = fc1; : : : ; cn; cn+1; : : : ; cn+mgof clients, where I(ci) < I(ci+1) for all i, and8><>:d(ci; ci+1) = ti; if 1 · i · n
d(ci; ci+1) ¸ ©; if n < i < n + m
d(ci; cj) = ti; if 1 · i · n, n + 1 · j · n + m; (2)
where we define © =Pn i=1 ti and assume © > D. The idea is to force the cj 's, n + 1 · j · n + m, to be assigned in different subsets because they have much larger DOIs.
A reduction example is,
c1 =
t1 z }| {
1 : : : 1
tn¡t1 z }| {
0 : : : 0
©¡tn+m z }| {
0 : : : 0
c2 =
t2 z }| {
1 : : : 1
tn¡t2 z }| {
0 : : : 0
©¡tn+m z }| {
0 : : : 0
: : :
cn =
tn z }| {
1 : : : : : : 1
©¡tn+m z }| {
0 : : : 0 ; and
cn+1 =
© z }| {
1 : : : : : : 1
m z }| {
0 : : : 01
cn+2 =
© z }| {
1 : : : : : : 1
m z }| {
0 : : : 10
: : :
cn+m =
© z }| {
1 : : : : : : 1
m z }| {
1 : : : 00 :

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 © + m bits. We now show that ®max · D if and only if ¯max · D. First, suppose ®max · D. Then, we can generate m disjoint subsets of C, C1; : : : ;Cm, where, for 1 · i · m, Ci = fckj jk is assigned to pi; 1 · k · ng [fcn+ig:Let ®i and ¯i be the completion time of processor pi and the DOI of Ci, respectively. Then, the completion time of processor pi is equal to the DOI of Ci. By Lemma 4 and (2),
®i =X
k:jk assigned to pi
tk = d(Ci) = ¯i: (3)
Hence,
®max = max1·i·m
®i = max1·i·m
¯i = ¯max: (4)
Thus, if ®max · D, then ¯max · D.

Conversely, suppose ¯max · D. Take any such partition C1; : : : ;Cm. None of these subsets can contain more than on eci, n+1 · i · n+m, since the DOI of the subset would then be greater than D. Let Ji be the set of jobs corresponding to Ci ¡fcn+ig, and assign Ji to pi, 1 · i · m. J1; : : : ; J will be a valid assignment for the resource constrained scheduling problem satisfying ®max · 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 =fs1; : : : ; smg of servers and a set L = fl1; : : : ; lng of clients in the hypercube, each server selects d nme clients to serve with no clients left unnerved. Let us first define the DOI cost of asset of clients with respect to a particular server, say s.

Definition 4: Consider a root node s and a set of nodes Q = fq1; : : : ; qng. Let © represent the bit-wise XOR operator. ds(Q) = d(s © q1; : : : ; s © qn):Let Li be the set of clients selected by server si, 1 · i ·m; Li µ L, and let D be a targeted DOI cost. Let ±max be the worst cost defined by,±max = max1·i·mdsi (Li):The MSPP is to find a partition of the client set so as to minimize ±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 ±max · D?

IV. 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, S1 and S2, 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 S1 and S2, 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)

```
–   input: S = fs1; : : : ; sng, a client set with increasing
    order of IDs; an integer k · n
–   if k = 1 then
–   return
–   end if
–   S1 Ã ;
–   S2 Ã ;
–   while S 6= ; do
–   Calculate the DOI of each client in S with its
    predecessor S
–   Find si in S who has the maximum DOI with its
    Predecessor, sj, in S
–   Add sj to set S1 and si to set S2
–   Remove si, sj from S
–   end while
–   SSP(S1, k2 )
–   SSP(S2, k2 )
–   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(n2) time. Thus, the recurrence for the running time is, T(n; k) =½1 if k = 1; T(n2 ; k2) + O(n2) otherwise: 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(n2).

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 = d n me 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 · 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)

```
–   input: S = fs1; : : : ; sng,a client set;an integer k · n
–   output: G, an optimal set with k clients w.r.t. the
    DOI cost
–   G Ã ;
–   Sort S in increasing order of the node (client) ID
–   d[1] = 0
–   for i = 2 to n do
–   d[i] = d(si¡1; si)
–   end for
–   for i = 1 to k do
–   Find sj in S whose d[j] is the minimum
–   Add sj to the client set G
–   Remove sj from S
–   end for
–   return G
```

2) Multi-server partition (MSP) algorithm:

Let S = fs1; : : : ; smg be the set of servers and L = fl1; : : : ; lng be the set of clients. The algorithm consists of two phases.

a) 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.

b) 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.

7

3) Running time and message complexity:

Phase 1 requires O (mnlog 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 coordinators end 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(mnlog 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(m2 logm) total running time. (Note that, in general, m ¿ 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 mostO(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 client's can start the download as early as the beginning of

phase1. Since the servers execute the algorithm sequentially in phase 1, in the worst case, some clients have to wait until the end of phase1 and the waiting duration is O(mnlog 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.

V. CONCLUSION

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 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.

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 hypercube," 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