A Fair Resource Allocation Algorithm for Peer-to-Peer Overlays

Yannis Drougas, Vana Kalogeraki
Department of Computer Science and Engineering
University of California, Riverside
Riverside, CA 92521
{drougas,vana}@cs.ucr.edu

Abstract— **Over the past few years, Peer-to-Peer (P2P) systems have become very popular for constructing overlay networks of many nodes (peers) that allow users geographically distributed to share data and resources. One non-trivial question is how to distribute the data in a fair and fully decentralized manner among the peers. This is important because it can improve resource usage, minimize network latencies and reduce the volume of unnecessary traffic incurred in large-scale P2P systems. In this paper we present a technique for fair resource allocation in unstructured Peer-to-Peer systems. Our technique uses the Fairness Index of a distribution as a measure of fairness and shows how to optimize the fairness of the distribution using only local decisions. Load balancing is achieved by replicating documents across multiple nodes in the system. Our experimental results demonstrate that our technique is scalable, has low overhead and achieves good load balance even under skewed demand.**

I. Introduction

In the recent years, the Peer-to-Peer (P2P) computing model is emerging as a powerful paradigm for developing large-scale distributed systems. This has found popular applications in content sharing [16], multicast [7], distributed object-location [20], [19], [21] and information retrieval [22]. P2P networks create virtual (logical) networks of many peers that allow users geographically distributed in wide-area networks to access large amounts of data, without the need for centralized control. Additional features include their ability for self-organization, resiliency to node failures and no need for dedicated servers. These features, however, bring new challenges. The challenge is *how can you implement a fair resource allocation scheme to distribute the data in an unstructured peer-to-peer system to balance the load, and achieve this in a fully decentralized manner?* The combination of large data repositories, the geographic distribution of the users and the dynamic and heterogeneous nature of the P2P systems demands careful distribution of the data across the system and orchestration of the utilization of the system resources.

The importance of these problems have been recognized by recent P2P systems [5] such as Pastry [19], Oceanstore [13], Chord [21] and CAN [20] (typically referred to as *Structured* Overlays) which are organized in such a way that objects are located at specific nodes in the network and nodes maintain some state information, to enable efficient retrieval of the objects. These sacrifice atomicity by mapping objects to particular nodes and assume that all nodes are equal in terms of resources, which can lead to bottlenecks and hot-spots. In *Unstructured* overlay networks [14], [10], on the other hand, objects can be located at random nodes, and nodes are able to join the system at random times and depart without a priori notification. Recent efforts have shown that a self-organizing unstructured overlay protocol maintains an efficient and connected topology when the underlying network fails, performance changes, or nodes join and leave the network dynamically [9]. Having a global view in unstructured overlays however, is impractical; thus choosing the right resource allocation protocol is non-trivial.

In this paper we present a fair resource allocation algorithm for unstructured P2P overlays. Our algorithm is based on the concept of fairness and uses the *Fairness Index* [8] of a distribution to measure ”how equally” the objects are allocated to the peers. The allocation of the objects is driven by decisions made by the individual peers based on information they have stored locally. An important consideration for our technique is to have a system that requires only local computation. Due to the large-scale of the P2P system, having global knowledge of the state of each node of such a network is practically impossible. Local computations improve efficiency, and allow the system to scale well and to react to changes quickly. Furthermore, the technique must handle (1) varying object loads, (2) different node capacities and (3) continuous insertion and deletion of the objects. In the paper we show how to optimize the fairness of the distribution in our system using only local decisions. We also present experimental results of our proposed technique. We investigate the fairness of the load distribution, the degree of replication and the reduction in the number of hops as a function of the popularity and the number of object requests in the system. The overhead of our approach is small because we piggy-back load measurement information in the messages. Thus, there is no need for generating additional load measurement messages in the system.

This paper is organized as follows. Section II discusses the problem we consider and our approach. Simulation results are presented in Section III. Section IV discusses related work, while Section V concludes the paper.

II. Approach Overview

We consider an unstructured P2P system of N nodes in which the nodes are typically user-machines. The communication link between two nodes p and q is characterized by the bandwidth available to it as $p_{band}(q)$. (Note that the incoming and outgoing bandwidth of the node may be different). This leads to a limit on the number of connections a peer can maintain. We denote this number of connections that a peer is maintaining by p_{conn}.

For a given peer u, let D_u be the set of documents that are stored in u. Without loss of generality, we assume that each document d is characterized by a sequence of keywords, and let $s(d)$ be the set of keywords in d. The query method to searching for documents in our system is K-random walks. K-random walks are shown to be a scalable solution, achieving linear increase in the number of messages propagated in the network. We note here, that although our system uses the K-random walks searching technique, other search algorithms could also be used in our system. We consider the load l_{p_i} of object i to be the total volume of data (in bytes) transmitted by p (to peers requesting i) in order to obtain object i. At any given time, we define the load l_p of peer p as the sum of the loads $\sum_i l_{p_i}$ due to all the objects stored on p at that time.

Each node p is characterized by its neighborhood p_{neigh} which is defined to be the set of nodes consisting of p as well as the nodes that are up to h hops away from p. For example, in Figure 1, for $h = 1$, the neighborhood p_{neigh} of node p is $\{p, q_1, q_2, q_3, q_4\}$. Node p keeps an *estimate* for the load $p_{q,l}$ of each peer q in its neighborhood. It collects load information about the nodes in its neighborhood only. These are obtained through resource utilization feedbacks propagated through queries and other system messages. Note that the accuracy of the load estimates (1) increases when the frequency of collecting the load information increases and (2) decreases as the number of hops between the peers increases. This information is later used by the load balancing algorithm. The architecture of our system, including resource utilization information obtained by the peers, is shown in Figure 1.

A. Fairness Index

We use the *Fairness Index* metric to measure "how equal" are the loads of the peers [8]. The fairness index of a load distribution $\bar{l}$ is defined to be:

$$\mathcal{F}(\bar{l}) = \frac{(\sum_{p \in N} l_p)^2}{|N| \cdot \sum_{p \in N} l_p^2} \tag{1}$$

where N is the set of nodes in the P2P system. This measurement allows us to compare the load distributions of the nodes by comparing their fairness indexes. The fairness index value ranges between values 0 and 1. Given the above equation, one can verify that the higher the value of the fairness index, the more uniform (fair) the distribution is. A totally fair system has a fairness index of 1, while a totally unfair system has an index value of 0. For example, when calculating the load fairness among a set of nodes N, a value of 0.1 indicates the

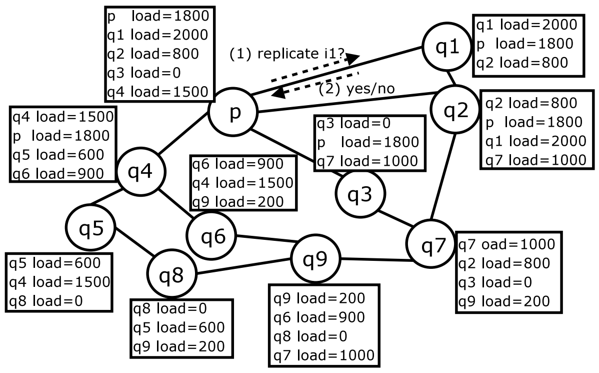

Fig. 1. *The model of our system*

system to be fair to only 10% of the nodes in N and unfair to 90% of them. The boundness of fairness index makes it ideal for the absolute evaluation of a load distribution's uniformity.

The Fairness Index has the important property that it is population size and metric independent, it can be applied to any number of nodes and the unit of measurement does not matter. An additional feature of the Fairness Index is that it is not necessary that it will monotonically increase of decrease when the load of a single node increases or decreases. Rather, it increases when the load approaches a specific value l_{best} and decreases when the load diverges from l_{best}. In addition to that, fairness index is a continuous quantity, reasonably modified when the loads of some of the nodes change. No matter how small a change of a load is, it is going to affect the fairness index of the load distribution. For example, the fairness of a distribution decreases when the load of a single element diverges from the value l_{best} that results in the maximum fairness index (assuming that the rest of the loads do not change), while it increases in the opposite case.

Finding the optimal distribution of the load to the peers is NP-complete. However, recent work shows that, in the centralized context (that is, when a node knows all the individual loads and the resource availability of all the nodes), a greedy algorithm can give good results [17].

B. Computing the Fairness for a Distributed Environment

The challenge in our scheme is to achieve a fair load distribution using only local information and making only local decisions at the nodes. To do that, nodes compute the load distribution $\mathcal{F}(\bar{l})$ and the average load $avg_{\bar{l}}$ in their neighborhoods using load measurements they received from the peers in their neighborhoods. These measurements are appended at the end of queries and other system messages, thus, eliminating the overhead of generating additional messages. The calculations are performed according to the following formulas:

$$\mathcal{F}(\bar{l}) = \frac{(\sum_{q \in p_{neigh}} l_q)^2}{|p_{neig}| \cdot \sum_{q \in p_{neigh}} l_q^2} \tag{2}$$

$$avg_{\bar{l}} = \frac{\sum_{q \in p_{neigh}} l_q}{|p_{neig}|} \tag{3}$$

Thus, node p uses the load estimates of each node q in p_{neigh} to compute the load fairness $\mathcal{F}_{\bar{l}}$ and the average load $avg_{\bar{l}}$ of its neighborhood p_{neigh}. If the neighborhood fairness is low (lower than a threshold τ_{neigh}) and node's p load is higher than the average load of the neighborhood, this indicates that p may be overloaded. Thus, node p tries to reduce its load by selecting an item i to replicate to another node. The idea behind this, is that p should now receive less requests for i. The item i is the one with the highest number of per node requests.

An important observation is that the item with the highest number of per node requests is not necessarily the most popular item of p. However, this is the item that will produce the largest reduction in the number of requests propagated to node p. For example, let us examine the case in Figure 1. Consider items i_1 and i_2, both stored in node p. Let us assume that p has received 5 requests for i_1 by neighbor q_1 and 1 by neighbor q_2. Also, that i_2 has been requested 3 times by neighbor q_3 and 4 times by neighbor q_4. The total number of requests for i_1 is $5 + 1 = 6$, while for i_2, it is $3 + 4 = 7$. However, the maximum number of per-neighbor requests is 5 for i_1 (the number of requests made for i_1 by q_1) and 4 for i_2 (the number of requests made for i_2 by q_4). Thus, among these two items, i_1 will be preferred to be replicated. This is because, when selecting i_1, we will select q_1 as the receiver of the replicated item (since q_1 has made the most requests for i_1). Doing this, p is expected to be relieved from k_1 requests (the requests that would be made from q_1 for i_1), in the future. On the other hand, if i_2 was chosen for replication, the best would be to replicate it to q_4. Then, p would be relieved from k_2 requests (the requests for i_2, made from q_4). But, the statistics make us estimate that $k_2 < k_1$. So, replicating i_1, rather than i_2 seems more promising.

Given the above observation, the item to be replicated is selected based on its popularity. The goal for node p is to reduce its load by replicating object i to one of its immediate peers. Thus, the best candidate to host the new replica is that immediate peer q that has propagated the maximum number of requests for object i to p. The replication attempt is made by p by sending a *replication request* to node q (message (1) in Figure 1).

To decide whether to accept or reject a replication request, node q estimates the effect in its load after the addition of the new object. The load l_q on node q will increase after replicating object i, since q will also have to reply to requests for i, in addition to the requests it is already receiving for the items it already stores. Furthermore, note that some of the already existing items of q may need to be deleted in order to make space for object i. The deleted items are the least popular items of q. In order for the user of a particular node to prevent any data loss, he must backup it. Thus, the disk space offered by each node is "dedicated" to the system, as a user cannot control the long term persistence of the data in that space. Eventually, unpopular data could be removed from the system in favor of popular ones. Node q evaluates its load against the high load threshold and the load l_p of p. If the allocation is acceptable, it is actually performed. However, if the request is not accepted by node q, node p will select the object with the second highest number of per node requests and a new node to host this object. Node q_1 informs p about its decision by sending it a reply message (message (2) in Figure 1).

One could argue that a heavy loaded node could migrate, rather than replicate the items that contribute most to its load. But, these items are also the most popular ones. By removing them, the number of hops to locate the items could increase. Furthermore, to avoid "ping-pong situations" in which popular items continuously migrate among the nodes in the network we associate a $num\_migration$ attribute with each object and (1) prefer to migrate a previously non-migrated object to an already migrated one and (2) restrict the maximum number of times an object can migrate.

III. Performance Evaluation

To evaluate the fairness of the system we ran experiments on the Neurogrid Simulator[1]. A P2P network was generated as a random graph of 1000 nodes. All links were assumed to have the same bandwidth. Each node had 6 connections on average. The average node capacity was $50KB$, while the sum of the size of all the items was about $20MB$. The threshold τ_{neigh} was set to 0.45. The neighborhood size h was set to 1. For the experiments we used the World Cup '98 dataset which consists of all the requests made to the 1998 World Cup Web site between April 26 - July 26, 1998 [1].

To simulate the queries run by the peers, we used $280,000$ GET requests of that dataset[2]. Each request is characterized by the ID of the node that issued it, the time it was sent and the size of the requested document. We selected a random set of 2500 documents taken from the set of documents appearing in these requests. This random set maintained the same statistical properties (*i.e.,* size and number of requests per document) as were in the original World Cup '98 dataset. It represents the set of documents of the peers. The documents were uniformly distributed throughout the nodes. Each node initially contained 3 or 4 randomly selected documents, and each document was replicated into two nodes maximum. The distribution of document requests throughout the experiment is shown in Figure 2. In this graph, the popularity (the total number of requests throughout the experiment) for each document is shown. The distribution is artificially generated in such a way that it keeps the properties of the original document requests. The figure shows that the document popularity is Zipf-like, with just a few documents receiving a large number of requests. Similar request distributions are also observed in popular file-sharing peer-to-peer systems [2].

[1]http://www.neurogrid.net/php/simulation.php

[2]The traces actually contained full logs of the communication between the web servers and clients. We just considered the HTTP GET requests, since GET requests represent user requests for files. $280,000$ is the number of the GET requests, not the size of the entire log file.

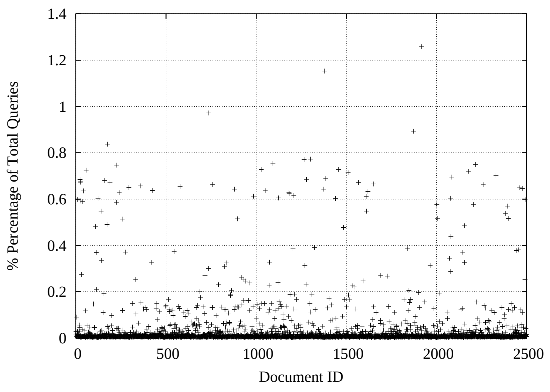

Fig. 2. *The popularity distribution of the documents*

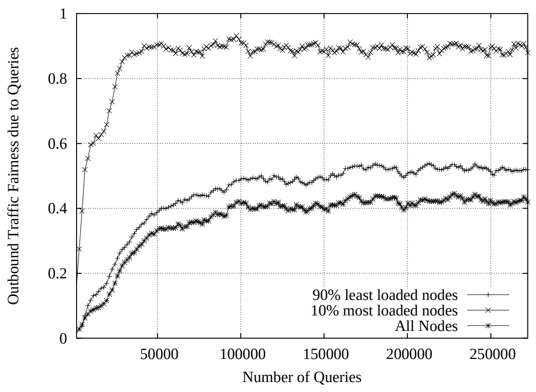

Fig. 3. *The fairness index of the load distribution of the nodes*

A. Simulation Results

Our goal in the fairness allocation algorithm is to make the load distribution among the nodes as fair as possible. We define the load on the peers as the amount of network bandwidth consumed by the documents on the peer. This is measured by the number of bytes transferred from the node as a response to queries originated from other nodes in the system. Replicating a document closer to the users requesting this document allows us to minimize the imbalance of the number of bytes sent by each node, to reduce the number of hops to find the document and thus, fairly distribute the query load on all the peers.

In computing the fairness index, one important observation is that a small number of overloaded nodes may cause an imbalance to the computation of the fairness index. To capture this fact in the computation of the fairness index we chose to plot separately the fairness index among the 10% most overloaded nodes and that among the remaining 90% of the nodes. Figure 3 shows the fairness index of the load distribution of the nodes as a function of the number of queries in the system, for the 10% overloaded peers, the 90% remaining peers and the total. The figure shows the same trend for all three graphs. The fairness index substantially improves as more user queries are submitted to the system. As the number of queries increases, more replicas are created on the nodes and thus the load is distributed across the peers. Note that the overall fairness does not approach a very high value, although it gradually increases. This happens because of these two different classes of nodes. However, the figure shows that the load distribution among these classes is fair.

Figure 4 shows the replication degree of the documents as a function of their popularity, as computed by the load balancing algorithm. The initial replication degree for all the documents was uniform and set to 2. The figure shows that as the popularity of the documents increases, their degree of replication increases. This indicates that requests for popular documents will be distributed across many nodes, rather than overloading only a few of the nodes in the system. Subsequently, this will increase the fairness index of the load of the system.

Our next observation is that the number of hops needed to answer a query for a document decreases. Figure 5 shows the average number of hops to the closest peer provider of the document. Our algorithm achieves incremental reduction in the hop-count and continues to improve as the documents get replicated in multiple peers. As a result, this reduces the bandwidth on the communication links (as less messages are propagated) and also the number of peers that process and propagate the messages.

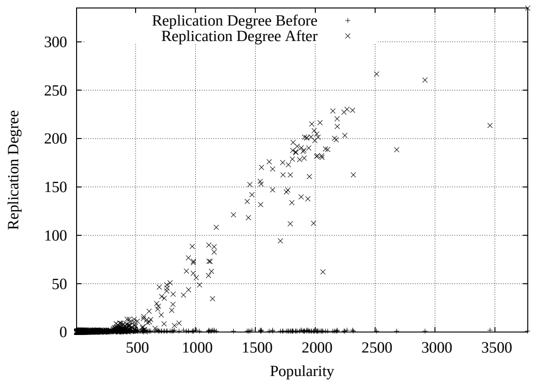

Fig. 4. *The replication degree vs. the popularity of each document*

IV. Related Work

In this section we briefly describe related work on replication strategies for load balancing in P2P overlays. The authors in [11], using offline heuristics, show that the right replica placement algorithm is greatly influenced by the applications and the user request patterns. Their algorithms however can not adapt to changing resource utilization workload. The comparison of various load balancing techniques in [18] indicates that, not careful load information propagation in P2P systems can result in large overheads and produce stale information. The authors suggest that dissemination of peer capacity (rather than load) information should be preferred in order to avoid instability. Peer capacity however is not enough to capture the changing resource availability at the peers as a result of new objects available at the nodes.

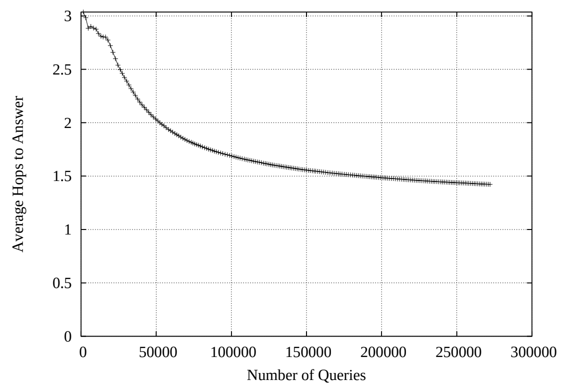

Fig. 5. *The average number of hops to answer queries*

Next, we discuss related work in load balancing in both structured and unstructured P2P systems.

a) Structured P2P systems: In [4], CFS, a distributed file system built on top of Chord [21], is proposed. In CFS, each server controls the replication degree of each of its items, and is responsible for its availability and performance. The RaDaR system [15] organizes network nodes in regions. Each region contains a *replicator* node, which disseminates information about items replicated in nodes of that area. Replicator nodes are organized in a tree structure which enables nodes located in different regions to communicate. However, frequent replicator nodes additions and deletions can affect the performance and robustness of the tree structure.

In [5] the authors propose a load balancing algorithm, applicable in dynamic structured P2P systems. The algorithm employs a set of directory servers. Their responsibility is to periodically schedule virtual server migrations between system nodes. The scheduling is based on load data, sent to each directory server by a random subset of nodes. The goal of the scheduling is to limit the aggregated load of all the nodes under a specific threshold. Their experiments illustrate that the algorithm is able to balance the node utilization, by moving a relatively small proportion of the load. The scheme concentrates on structured peer-to-peer systems and DHTs. The difference from our method is that our technique is completely decentralized and each node is not required to have global knowledge of the system. The method presented in [5] requires that each node has global knowledge of the system. The goal of our method is to balance the load using only local computations.

In [3], a peer-to-peer framework for the management of federated services is presented. The proposed framework implements a publish/subscribe system, where the goal is to offer services to clients that connect to a small and restricted set of servers. Each node requires global knowledge of the state of each of the other nodes in the system. This is impossible in the type of network we consider.

b) Unstructured P2P systems: In [14], various placement strategies are compared. The authors observe that a strategy that places replicas near most active users, performs better in terms of quality of service than a random or a greedy placement method. However all the proposed methods suffer performance degradation, when there is no complete knowledge of the nodes in the network.

In [6], a decentralized distributed system is considered. Apart from the object replicas, each node also caches limited information about items replicated in other nodes. This is for speeding-up the responses to the queries. Each node keeps track of its load and employs either an active or a passive replication strategy. The decision is based on whether the node is considered to be overloaded or not. However, the evaluation of the load does not take into account the states of the neighboring nodes. When all the neighbors of a loaded node are also loaded, trying to unload that node will not help and can result in oscillations.

In [10], authors examine various replica placement heuristics that are applied in Content Distribution Networks. Their results demonstrate that popularity based replication performs better even than a greedy algorithm with global knowledge of the system. The performance is improved by making the peers cooperate amongst them. However, if we assume a completely unstructured dynamic system, the overhead of co-operation prohibits its use.

The problem of replica placement in Content Distribution Networks is studied in [16]. Various topology informed heuristics are compared under different network topologies. The results suggest that topology information can help in reducing the user perceived latency. In particular, heuristics that promote replication in high degree nodes are shown to result in allocations with 20% higher latencies and network utilization, compared to centralized greedy approaches. However, the problem of load balancing is not considered.

Kazaa [12] is another widely used unstructured peer-to-peer file sharing system. Kazaa is used daily by millions of users and has very low latency. A two-tier architecture is used. Nodes in Kazaa are divided into *Ordinary nodes* and *Supernodes*. Supernodes are typically more powerfull and better connected than ordinary nodes. Each ordinary node connects to a Supernode. Supernodes on the other hand, form an unstructured peer-to-peer network among themselves. In contrary, our system is completely decentralized and all peers are equal; there is no set of “powerfull” nodes in the network.

V. Conclusions and Future Work

In this paper we discussed the problem of fair resource allocation in Peer-to-Peer systems and presented a technique that solves this problem in a decentralized manner. Our technique exploits load measurement information collected at the neighborhoods of the peers without introducing any significant overhead in the system; thus making the algorithm very appealing for large-scale systems. Our simulation results show that our technique achieves fairness in load balance distribution at the peers, reduces the number of hops to find the documents and highly replicates the popular documents in the system. The evaluation of our method was performed using requests taken from a web trace. In our future work we are planning to examine the behavior of our methods under different request arrival patterns. Specifically, we would like to investigate what happens when the requests are bursty over time and arriving in a limited subset of nodes. Furthermore, we are planning to make experiments with dynamic peer connections and disconnections.

References

[1] Martin F. Arlitt and Tai Jin. A workload characterization of the 1998 world cup web site. *IEEE Network*, 14(3):30–37, May/June 2000.

[2] Jacky Chu, Kevin Labonte, and Brian Neil Levine. Availability and popularity measurements of peer-to-peer file systems. In *ITCom: Scalability and Traffic Control in IP Networks II Conferences*, July 2002.

[3] Francisco Matias Cuenca-Acuna and Thu D. Nguyen. Self-managing federated services. In *The 23nd IEEE Symposium on Reliable Distributed Systems (SRDS-23)*, pages 240–250. IEEE Press, October 2004.

[4] Frank Dabek, Frank Mass Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, Chateau Lake Louise, Banff, Canada, October 2001.

[5] Brighten Godfrey, Karthik Lakshminarayanan, Sonesh Surana, Richard Karp, and Ion Stoica. Load balancing in dynamic structured p2p systems. In *IEEE INFOCOM*, 2004.

[6] Vijay Gopalakrishnan, Bujor Silaghi, Bobby Bhattacharjee, and Pete Keleher. Adaptive replication in peer-to-peer systems. In *The 24th International Conference on Distributed Computing Systems (ICDCS'04)*, pages 360–369, March 2004.

[7] Yang hua Chu, Sanjay G. Rao, and Hui Zhang. A case for end system multicast. In *SIGMETRICS '00: Proceedings of the 2000 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 1–12, Santa Clara, California, United States, 2000. ACM Press.

[8] Rajendra K. Jain, Dah-Ming W. Chiu, and William R. Have. A quantitive measure of fairness and discrimination for resource allocation in shared computer systems. Technical Report DEC-TR-301, Digital Institution Corporation, Hudson, MA 01749, September 26 1984.

[9] S. Jain, R. Mahajan, D. Wetherall, and G. Borriello. Scalable self-organizing overlays. Technical Report TR 02-06-04, UW-CSE, June 2002.

[10] J. Kangasharju, J. Roberts, and K. Ross. Object replication strategies in content distribution networks. In *WCW'01: Web Caching and Content Distribution Workshop*, Boston, MA, June 2001.

[11] Magnus Karlsson and Christos T. Karamanolis. Choosing replica placement heuristics for wide-area systems. In *The 24th International Conference on Distributed Computing Systems (ICDCS'04)*, pages 350–359, 2004.

[12] Kazaa website. http://www.kazaa.com.

[13] John Kubiatowicz, David Bindel, Yan Chen, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westly Weimer, Christopher Wells, and Ben Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ACM ASPLOS*. ACM, November 2000.

[14] Lili Qiu, Venkata N. Padmanabhan, and Geoffrey M. Voelker. On the placement of web server replicas. In *INFOCOM*, pages 1587–1596, April 22-26 2001.

[15] Michael Rabinovich and Amit Aggarwal. RaDaR: a scalable architecture for a global Web hosting service. *Computer Networks (Amsterdam, Netherlands)*, 31(11–16):1545–1561, 1999.

[16] P. Radoslavov, R. Govindan, and D. Estrin. Topology-informed internet replica placement. In *the Sixth International Workshop on Web Caching and Content Distribution (WCW'01): Web Caching and Content Distribution Workshop*, pages 229–238, Boston, MA, June 2001.

[17] Paraskevi Raftopoulou. Fair Resource Allocation in P2P systems: Theoretical and Experimental Results. Master's thesis, Department of Electronic and Computer Engineering, Technical University of Crete, Greece, 2003.

[18] Mema Roussopoulos and Mary Baker. Practical load balancing for content requests in peer-to-peer networks. Technical Report cs.NI/0209023, Stanford University, January 2003.

[19] Antony Rowstron and Peter Druschel. Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 188–201, Banff, Alberta, Canada, 2001. ACM Press.

[20] Ratnasamy S., Francis P., Handley M., Karp R., and Shenker S. A scalable content-addressable network. In *SIGCOMM'01*, 2001.

[21] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM '01 Conference*, San Diego, California, August 2001.

[22] Demetris Zeinalipour-Yazti, Vana Kalogeraki, and Dimitrios Gunopulos. Exploiting locality for scalable information retrieval in peer-to-peer systems. *Information Systems Journal*, 30(4):277–298, 2005.