A Topology-Aware Load Balancing Algorithm for P2P Systems

Seyed Iman Mirrezaei[1] Javad Shahparian[1]
Mohammad Ghodsi[1,2*]
Computer Engineering Department
[1]Sharif University of Technology, Tehran, Iran
[2]IPM School of Computer Science, Tehran, Iran
{mirrezaei, shahparin}@ce.sharif.edu, ,{ghodsi}@sharif.edu

Abstract

One of the challenges of P2P systems is to perform load balancing efficiently. A distributed hash table (DHT) abstraction, heterogeneous nodes, and non uniform distribution of objects cause load imbalance in structured P2P overlay networks. Several solutions are suggested to solve this problem but they have some restrictions. They assume the homogeneous capabilities of nodes, unawareness of the link latency during transferring load and imposing logical structures to collect and reassign load. This paper presents a distributed load balancing algorithm with topology awareness using the concept of virtual servers. In our proposed approach, each node collects neighborhood load information from physically close nodes and reassigns virtual servers to overlay nodes according to topology of underlying network. Consequently, it provides rapid convergence on load balance and reduces the load transfer cost. Moreover, our parametric algorithm increases the quality of load balancing among close nodes of overlay and also provides a new tradeoff between the quality of load balancing and load transfer cost among all overlay nodes. Our simulations show that our approach reduces the load transfer cost and saves a great network bandwidth.

1 Introduction

Structured P2P overlay networks [3, 6, 7] provide DHT abstraction for object storage and retrieval. In these overlays, each object and node is identified by a unique identifier. The search space is partitioned among overlay nodes and each node is responsible for storage and retrieval of objects in its region. These systems assume that resources such as network bandwidth, capacity and storage are uniformly distributed among all participants of network.

In these structured systems, a distributed hash function chooses identifier of nodes and objects. So, it causes an $O(\log N)$ imbalance factor in the number of stored objects at a node. Moreover, if identifier of nodes are no longer uniformly distributed, the imbalance factor becomes worse. This could happen in database applications because all data items (tuples) of a relation are kept with regard to their primary key values(identifiers). In addition, load imbalance becomes worse when there exist many nodes with different capabilities (storage, bandwidth, CPU, etc.). Resulted load imbalance deteriorates the functionality of overlay networks.

Several solutions are offered to solve the load balancing problem [7, 8, 9, 11, 10, 15, 16, 17]. But these solutions have some restrictions. Firstly, they may assume that nodes have similar capabilities. Secondly, they ignore the link latency between nodes and extra load of a node may traverse the high link latency, thereby increasing the bandwidth consumption, increasing traffic in underlying network and delaying the convergence of load balancing. Thirdly, they may use some logical fixed nodes to collect load information and plan new reassignments. However, these solutions reduce the load balancing problem to a centralized problem, therefore, the single point of failure problem and limited scalability are matter of concern.

This paper presents a distributed load balancing algorithm with topology awareness in which those restrictions do not hold. Our algorithm also uses the concept of *virtual servers* formerly suggested in Chord [7]. In our approach, each node collects neighborhood load information from *close* nodes according to topology of underlying network and reassigns load. Then virtual servers are transferred between physically *close* nodes. Consequently, it provides rapid convergence on load balance, quick reply to load imbalance, reduce the load transfer cost and improvement on load balancing traffic. This approach does not impose any logical structure or overhead to overlay network,

[*] This work has been partially supported by IPM School of CS (contract: CS1385-2-01)

978-1-4244-4254-6/09/$25.00 ©2009 IEEE

While collecting load information of nodes. Moreover, it does well in term of scalability. Our parametric algorithm increases the quality of load balancing among *close* nodes of overlay and also provides a different kind of tradeoff between the quality of load balancing and load transfer cost across overlay nodes. In addition, each node or group of nodes can perform the proposed load balancing algorithm based on its desired network distance. We perform our load balancing algorithm on the RAQNet [1] overlay network. In RAQNet overlay network, each node has a practical internet coordinate(*PIC*) for estimating internet network distances between nodes by the *PIC* mechanism [4].

This solution can be performed in other structured P2P overlay networks if each overlay node knows its practical internet coordinate [4]. The rest of this paper is organized as follows. Section 2 provides a survey of related work. A brief overview of RAQNet overlay network are presented in section 3. Section 4 describes the topology aware load balancing algorithm. The experimental evaluations are shown in section 5 and section 6 concludes the paper.

2 Related work

Most Structured P2P systems [3, 6, 7] suppose that object IDs are distributed by the uniform hash function. Additionally, they suppose that all nodes have similar capacities and load. Even so, the resulted load balance is not completely perfect and they have an $O(\log n)$ imbalance load.

Many load balancing methods have been suggested to handle this problem in P2P systems. The first work has been done by Chord [7]. They diminish load of overlay nodes by using the concept of virtual servers. They allocate $\log N$ virtual servers per physical node and suppose that all overlay nodes are similar. However, their approach does not practically resolve the load balancing problem.

CFS [15] does not ignore the heterogeneity of nodes. In CFS, virtual servers are allocated to nodes according to their capacities. Also, they use a simple solution to transfer extra load from heavy nodes, but their method may cause other nodes become overloaded.

Triantafillou et al. [16] introduce the novel design to perform fair load distribution in the context of content and resource management in unstructured P2P systems. They collect load objects by the meta-data and after that they compute a reassignment of objects by using that information.

Karger and Ruhl [17] present dynamic load balancing algorithms without using virtual servers. In their algorithms, lightly loaded nodes should be neighbors of heavily loaded nodes in order to reassign their load. They maximize utilization of load in nodes but they do not completely consider different node capacities. Moreover, It is not clear whether their algorithms are practical or not.

Roa et al. [9] propose three simple load balancing algorithms for DHT-based systems: *one-to-one, one-to-many* and *many-to-many*. They transfer load from heavy nodes to light nodes in every unit of virtual servers. In their load balancing approach, they use *directory* nodes to store load information of nodes and reassign virtual servers. *one-to-many* and *many-to-many* are extended by Godfrey et al. [10]to perform load balancing in dynamic P2P systems. Their results have shown that their approach is so effective, but they have two weak points. Firstly, their approach suffers from a single point of failure problem because of using *directory* nodes. Secondly, their approach does not notice to link latency between light and heavy nodes while transferring load.

Yingwu et al. [11] use a *k-ary* tree and virtual servers to perform load balancing in structured overlay networks. In their algorithm, the load information is collected by the *k-ary* tree and reassignments of virtual servers are scheduled by nodes of the *k-ary* tree. They use landmark binning [14] to manage virtual server assignments across nodes which are *close* to each other according to topology of underlying network. They determine *close* nodes by measuring from the landmark sites. Thus, landmark sites become hot spots while the P2P system size are increasing.

The topology-aware load balancing algorithm presented in this paper is similar to a distributed load balancing algorithm proposed by Zhenyu et al [8]. In both algorithms, load are transferred based on topology information, but we collect load information of *close* nodes by a restricted flooding algorithm with regard to topology of underlying network. Our approach improves the load balancing traffic and also provides rapid convergence on load balance. Also, our parametric algorithm increases the quality of load balancing among *close* nodes of overlay and also provides a different kind of tradeoff between the quality of load balancing and load transfer cost across all overlay nodes. Moreover, each node or group of nodes can perform the proposed load balancing algorithm based on its desired network distance to transfer extra load.

3 Overview of RAQNet

RAQNet [1] is a multi-dimensional topology-aware overlay network based on RAQ [2] data structure. In RAQNet overlay network, the search space is d-dimensional Cartesian coordinate space which is partitioned among n nodes of the overlay network by a partition tree. Each node has $O(\log n)$ links to other nodes. Each single point query is routed via $O(\log n)$ message passing. Each node is corresponded to a region and it is responsible for the queries targeting any point in its region. In RAQNet overlay, nodes are connected to each other if they have the same labels and also are *close* to each other with respect to the topology of the underlying network. A topological match between an overlay and its underlying network reduces routing delays and network link traffic. Every network node x which corresponds to a leaf in the partition tree is assigned a *Plane Equation* or PE to specify its region in the whole space. RAQNet seeks to exploit topology awareness from its underlying network in order to fill its routing table rows effectively. Topology aware neighbor selection selects the collection of *close* nodes among nodes with PE having the required prefix. Topology awareness relies on a proximity metric that indicates the "distance" between any given pair of nodes. The choice of a proximity metric depends on the desired quality of overlays (e.g., low delay, high bandwidth). Our proximity metric in RAQNet overlay is round trip delay, estimated by the *PIC* mechanism [4].

4 Topology Aware Load Balancing

In this section, we present the concept of virtual servers and then introduce our topology-aware load balancing algorithm.

4.1 Virtual Servers in RAQNet

The concept of virtual servers was first introduced in Chord [7] to improve load balancing of overlay nodes. A virtual server looks similar to a single node which is accountable for a region of the search space. Several virtual servers can be hosted by a physical node. Therefore, any physical node possesses noncontiguous regions of the search space. Each virtual server owns its routing tables and stores data items with IDs falling into its accountable region.

Any virtual server causes definite amount of load. For instance, serving queries which fall into accountable region of a virtual server generates load. Whenever a node becomes overloaded, it transfers portion of its load to some lightly loaded nodes to become light in which the basic unit of load transfer is virtual servers [9, 10]. Therefore, transferring virtual servers from heavy nodes to light nodes causes load balance. The transfer of a virtual server is implemented as a departure operation comes before a join operation, all overlays provide these operations.

When a node x leaves the overlay, its regions are taken by other nodes which have contiguous regions with virtual servers of node x [1]. If a virtual server v leaves the overlay, its responsible region is taken by another virtual server which has contiguous region with v. If there is no such virtual server, region of v is taken by a virtual server with PE closer to v. In the same way, When a new node x joins the overlay, it chooses $numVS$ (number of virtual servers) random point X in the search space and sends its join request.

One disadvantage of using virtual servers is that any overlay node maintains $numVS$ routing states for its virtual servers. Our experimental results show that our approach reaches good load balance when each node has $numVS = \log N$ virtual servers. In our belief, this overhead can be reasonable. One of the main benefits of using virtual servers is that no overlay modification is needed to perform load balancing algorithms.

4.2 Topology Aware Load Balancing Algorithm

The virtual server reassignments are done with regards to topology information of underlying network. As we described in RAQNet [1], RAQNet overlay nodes inherently maintains information of *close* nodes. Our load balancing algorithm use this information and the *P*IC mechanism [4] to predicate network distance(i.e., round-trip delay or network hops) between light and heavy nodes during virtual server reassignments, described in section 4.4. The *P*IC mechanism predicts the distance between two overlay nodes only by having their practical internet coordinates.

We have two assumptions in our load balancing approach: we attempt to optimize only one bottleneck resource and we suppose that the load on a virtual server is stable while carrying out our load balancing algorithm.

These are some definitions we use to explain our approach:

Utilization: u_i is the ratio of node i load to its capacity; $u_i = \frac{l_i}{c_i}$. The l_i shows load of node i at a definite time and each node i has a capacity c_i which may represent available storage, processor speed, or bandwidth.

Neighborhood Utilization: neighborhood utilization of node i is defined as $Neighutil_i = \frac{\sum_{i=1}^{l} Load_i}{\sum_{i=1}^{l} Capacity_i}$, where s is a set of *close* nodes which announce their load information to node i and l is the number of nodes in set S.

Load transfer cost: Load transfer cost is defined as $LTC = \sum_{i=1}^{n} Load_i * Dist_i$, where $Dist_i$ indicates the network distance to transfer load of node i. The amount of transferred load for node i is shown by $Load_i$.

4.3 Neighborhood Load Information Collection

We use a restricted flooding schema to collect neighborhood load information. The flooding schema with a few Time-to-Live (TTL) hops have been presented by S. Jiang et al. [12]. They have shown that is extremely effective and generates few excess messages. Regularly, each node

```
Procedure Reassign-VirtualServer
Node $i$ calculates it's neighborhood utilization
$T_i = (Neighutil_i + \varepsilon) * C_i$
// $T_i$ is target load of node $i$.
if ($L_i \leq T_i$)
    return; // Node $i$ is a light node.
end if
Candidate-VS = Node $i$ chooses one of its VS to leave.
/* VS is abbreviation of Virtual server*/
Receiving-Node = Find-LightNode($Candidate - VS$)
if (Receiving-Node!= null)
    Transfer Candidate-VS to Receiving-Node
end if
```

Figure 1. Reassigning virtual servers.

i sends a probing message including the origin address information(IP), its practical internet coordinate(for estimating network distance), the $DesiredVal$ value and a TTL value to some nodes that exist in its routing table entries, with network distance to node i less than the $DesiredVal$. Node j which receives a probing message replies to the origin node i with its address information, current load, capacity and its practical internet coordinate. Then, The TTL value is decreased by 1 and if the updated TTL value does not reach to 0, it resends the received probing message to its routing table entries with network distance to origin node i less than the $DesiredVal$. When origin node i receives the replied probing messages, it computes the round trip time to the responding nodes by using the practical internet coordinate of responding nodes and the *PIC* mechanism. After that, node i stores this information to its neighborhood load information set($NLIS_i$). The member count of this set can be represented as following: MemberCount($NLIS_i$) = $\sum_{j=1}^{TTL} numVS^j = \frac{numVS*(numVS^{TTL}-1)}{numVS-1} =$ $O(numVS^{TTL})$. In this formula, $numVS$ represents the number of virtual servers per overlay node. Based on our experimental evaluations in section 5, our approach reaches a fine load balance if $numVS$ is $O(\log n)$ and TTL is 2. So, The member count of $NLIS_i$ is $O(\log^2 n)$ in the worst-case. We consider the worst-case to be sending the probing messages to all nodes in the routing table entries and it happens only if we assign the biggest possible value to the $DesiredVal$.

4.4 Node Categorization and Virtual Server Reassignments

Whenever a neighborhood load information set is become ready, each node i knows the load and capacity of neighborhood nodes and then calculates its neighborhood utilization, $Neighutil_i$, and its target load, T_i. After that, if its current load, L_i, is bigger than its target load, T_i, it marks itself as a heavy node , then it chooses one of its virtual servers to leave node i and makes it light. Finding a proper virtual server takes $O(numVS)$ time.

We use procedure $Reassign - VirtualServer$ to reassign virtual servers. This process is described in figure 1. ε is a parameter for a tradeoff between the amount of load transferred and the quality of load balancing. ε ideally is 0. Calculating the best reassignments is equivalent to minimize maximum node utilization problem and is NP-complete [13]. So, it is impossible to reassign virtual servers across nodes perfectly but it can be solved by an approximate algorithm.

Our approach provides a different kind of tradeoff between the quality of load balancing and load transfer cost, based on parameter QLB. A heavy node chooses a light node with the smallest utilization and network distance less than the QLB. Otherwise it tries to find the closest light node with network distance more than the QLB, shown from line 9 to 16.

The worst-case running time (to be sending the probing message to all nodes in the routing table entries) of procedure *Find-LightNode* is $O(numVS^{TTL})$, where the average number of virtual servers is shown by $numVS$. Based on our experimental results, whenever $numVS$, TTL, QLB and $DesiredVal$ are equal to $O(\log N)$, 2, 130 and 400(based on GT-ITM[5]) , our algorithm achieves good load balance. Additionally, the worst-case running time of our proposed approach is $O(\log N + \log^2 N)$. So that it does well in term of scalability.

4.5 Synchronization between Light and Heavy Nodes

All overlay nodes collect load information and reassign virtual servers concurrently. Therefore, some virtual servers may be sent to a light node from different heavy nodes, which causes a light node to become overloaded. Hence, before sending virtual servers, a heavy node send a $synch$ message to a light node. If its load is not changed, it acknowledges the heavy node and does not acknowledge to others. After a distinct interval, if a heavy node does not receive any acknowledge message from a light node, it chooses another node to transfer its extra load.

5 Experimental Results

We present experimental results which evaluates our load balance approach in RAQNet overlay network. The results were achieved using a RAQNet overlay with 4096 nodes running on an Internet topology model. We assume that f is a fraction of the search space which belongs to a virtual server that is exponentially distributed. Also, μ and σ show the mean and the standard deviation of total load on RAQNet overlay. We use Gaussian distribution with mean

μf and the standard deviation $\sigma\sqrt{f}$[9] for the load on virtual servers. We also use $Gnutella-like$ capacity for capacity of nodes. Consequently, 20 percent, 45 percent, 30 percent, 4.9 percent, and 0.1 percent of node capacity is 1, 10, 100, 1000, 10000.

Our experiments run on a simulated network topology which was generated by the Georgia Tech transit-stub network topology model [5]. Ts4k-small includes 4 transit domains each with 4 transit nodes, 5 stub domains connected to each transit node, and 55 nodes in each stub domain on average.

5.1 The Effect of Load balancing Parameters

We assign $DesiredVal = 400$ and $QLB = 130$(based on GT-ITM[5]) while performing our topology-aware load balancing algorithm. Figure 2 shows that the TTL value affects on node utilization. It improves the quality of load balancing while TTL value changes from 1 to 2 because there exist more alternative light nodes in neighborhood load information set, as we said in section 4.3.

Increasing QLB , the algorithm gives priority to the quality of load balancing and it ignores the load transfer cost. When we assign the biggest possible amount to $DesiredVal$ and QLB, our approach performs topology unaware load balancing. By decreasing the value of $DesiredVal$, fewer nodes will report their load information to requesting nodes and the quality of load balancing will be decreased in overlay network. When TTL value is increased up to 4 or even more, the quality of load balancing will be decreased surprisingly. This is because it may ignore the nodes with small utilization and far network distance. However, the values of QLB and $desiredVal$ will affect the quality of load balancing. Thus, each overlay node can compromise between load transfer cost and its utilization. We separately compute the LTC (load transfer cost), defined in section4.2, with and without considering topology awareness. Then, we calculate $Benefit = \frac{LTC_{Withouttopology}-LTC_{topology}}{LTC_{Withouttopology}}$. $Benefit$ is 43% in GT-ITM topology model. Therefore, the network bandwidth is saved greatly in our approach.

5.2 Topology Aware Load Balancing

In this section, we show the effect of topology awareness on load balancing. In figure 3, cumulative distribution of transferred load is illustrated. It shows that 50% of load is transferred via the network distance with average link latency of 100 in GT-ITM topology. Also, more than 80% of load traverses the links with total average latency of 200. In contrast, regardless of topology awareness, the 50% of load is transferred having average of about 280 of link latency. Consequently, extra load of heavily loaded nodes is transferred among *close* nodes and it imposes less traffic to underlying network. Thus, Topology aware load balancing saves bandwidth considerably. Moreover, this algorithm converges quickly because it chooses nodes from *close* groups which are physically *close* together and therefore reduces the cost of transferring load. In figure 3 the load balancing algorithm, suggested by Godfrey et al [10], is indicated by "directory" line. It is obvious that their method is similar to topology unaware load balancing.

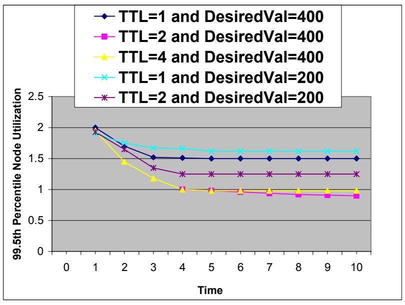

Figure 2. The effect of various load balancing parameters on node utilization.

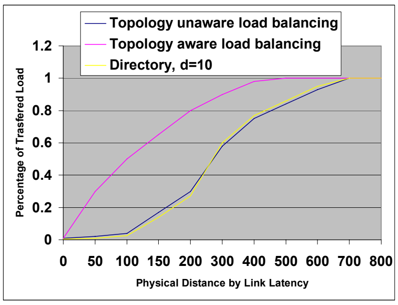

Figure 3. Cumulative distribution of transferred load in GT-ITM topology.

The scatter plots of load for the Gaussian distribution are shown in figure 4 and figure 5 and we use a $Gnutella-like$ capacity in our node capacity model. Our load balancing approach helps to rearrange a bad load distribution into an acceptable arrangement and eventually each overlay node will have the load proportional to its capacity.

6 Conclusion

This paper presents a simple distributed load balancing algorithm with topology-aware property for structured P2P overlay networks. In our approach, each node collects

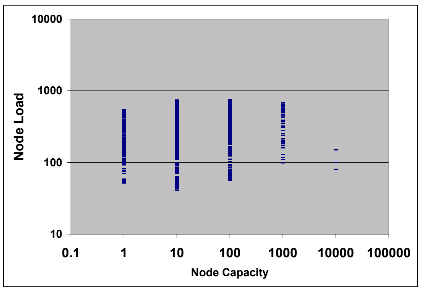

Figure 4. The scatter plots of load and capacities before load balancing.

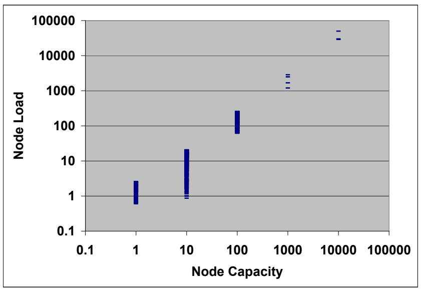

Figure 5. The result of load balancing algorithm.

neighborhood load information from *close* nodes and then it reassigns its own extra load according to topology of underlying network. Consequently, it provides rapid convergence on load balance and reduces the load transfer cost. Our parametric algorithm increases the quality of load balancing among *close* nodes and also provides a different kind of tradeoff between the quality of load balancing and load transfer cost. The experimental results show that this approach is effective and considerably saves network bandwidth.

We plan to enhance our load balancing approach to adapt in a dynamic system. Moreover, as a future improvement to our approach, imposing other constraints (e.g. utilization of nodes) during collecting the load information of nodes, may be considerably helpful.

Acknowledgments.

The authors would like to thank Yashar Heidari and Kambakhsh Jafari for their supports.

References

[1] Seyed Iman Mirrezaei, J. Shahparian, and M. Ghodsi. RAQNet: A Topology-aware Overlay Network. Autonomous Infrastructure, Management and Security Conference, to appear, AIMS'2007, *LNCS 4543 by Springer-Verlog*, pp. 13-24, 2007.

[2] H.Nazerzadeh, and M.Ghodsi, RAQ: A range queriable distributed data structure (extended version). In Proceeding of Sofsem 2005, 31st Annual Conference on Current Trends in Theory and Practice of Informatics, *LNCS 3381 by Springer-Verlog*, pp. 264-272, February 2005.

[3] A. Rowstron, and P. Druschel. Pastry: Scalable,distributed object location and routing for large-scale peer-to-peer systems. *In Proc. IFIP/ACM Middleware 2001*, Heidelberg, Germany, Nov. 2001

[4] M. Costa, M. Castro, A. Rowstron, and P. Key. PIC: Practical Internet Coordinates for Distance Estimation. *In 24th IEEE International Conference on Distributed Computing Systems (ICDCS' 04)*, Tokyo, Japan, March 2004.

[5] E. Zegura, K. Calvert, and S. Bhattacharjee. How to model an internetwork. *In INFOCOM96*, 1996

[6] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. *In Proc. of ACM SIGCOMM*, Aug. 2001.

[7] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. 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.

[8] Zhenyu Li, and Gaogang Xie. A Distributed Load Balancing Algorithm for Structured P2P Systems. *Proceedings of the 11th IEEE Symposium on Computers and Communications (ISCC'06)*, 2006.

[9] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load Balancing in Structured P2P Systems. Proc. *Second Intl Workshop Peer-to-Peer Systems (IPTPS)*, pp. 68-79, Feb. 2003.

[10] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load Balancing in Dynamic Structured P2P Systems. *Proc. IEEE INFOCOM*, Mar. 2004.

[11] Y. Zhu, and Y. Hu. Efficient. Proximity-Aware Load Balancing for DHT-Based P2P Systems. *In IEEE Transactions on parallel and distributed systems, Vol. 16, No.4*, April 2005.

[12] S. Jiang, L. Guo, and X. Zhang. Lighflood: an efficient flooding scheme for file search in unstructured peer-to-peer systems. *In Proceedings of ICPP 2003*.

[13] Horowitz, E., and Sahni, and S. K. Exact and approximate algorithms for scheduling nonidentical processors.*Journal of ACM, Vol. 23, No. 2*, April 1976.

[14] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-aware overlay construction and server selection. *In Proceedings of INFOCOM 2002*.

[15] F. Dabek and M. F. Kaashoek and D. Karger and R. Morris and I. Stoica. Wide-area Cooperative Storage with CFS.*Proc. ACM SOSP* 2001.

[16] P. Triantafillou and C. Xiruhaki and M. Koubarakis and N. Ntarmos, Towards High Performance Peer-to-Peer Content and Resource Sharing Systems. *Proc. of CIDR*, 2003.

[17] David Karger and Matthias Ruhl. New Algorithms for Load Balancing in Peer-to-Peer Systems. *Tech. Rep. MIT-LCS-TR-911, MIT LCS*, July 2003.