Load sharing in peer-to-peer networks using dynamic replication

Rajasekhar, Sathish; Rong, Bin; Lai, Kwong; Khalil, Ibrahim; Tari, Zahir

Rajasekhar, S., Rong, B., Lai, K., Khalil, I., & Tari, Z. (2006). Load sharing in peer-to-peer networks using dynamic replication. Proceedings of the 20th International Conference on Advanced Information Networking and Applications, 1101–1106. https://doi.org/10.1109/AINA.2006.207

Published Version: https://doi.org/10.1109/AINA.2006.207

Load Sharing in Peer-to-Peer Networks using Dynamic Replication

S Rajasekhar, B Rong, K Y Lai, I Khalil and Z Tari
School of Computer Science and Information Technology
RMIT University, Melbourne 3000, Australia
(sathish,brong,kwonglai,ibrahimk,zahirt)@cs.rmit.edu.au

Abstract

The peer-to-peer (P2P) architecture provides support for the next generation of information sharing applications. A difficult challenge faced by these systems in the presence of non-uniform data distribution and dynamic network conditions is load sharing. This paper addresses the problem of load sharing in P2P networks across heterogeneous super-peers. We propose two load sharing techniques that use data replication to improve access performance. In the first technique, called Periodic Push-based Replication (PPR), super-peers periodically send replicas of the most frequently accessed files to remote super-peers. This effectively reduces the hop count to fetch these files. The second technique, called On-Demand Replication (ODR), performs replication based on access frequency. By performing replication on-demand, ODR provides adaptability to changes in access behavior. Extensive testing have been conducted to study the performance of the proposed techniques. The results obtained demonstrate significant performance improvements through replication.

1. Introduction

In recent years, P2P networks have become very popular due to its simplicity and its decentralised approach in supporting large scale applications such as file-sharing. A practical problem encountered by P2P systems is load imbalance. Peers with popular data files are accessed more frequently, putting these nodes under heavy load.

Achieving load sharing is of fundamental importance in P2P systems. By sharing the load, better utilisation and performance can be achieved. Some P2P systems such as Gnutella [1] flood information throughout the network creating heavy load on peer nodes. Peers have no knowledge about the identity and information of other peer devices in the network, hence, coordination amongst peers becomes a complex task with significant overheads. To overcome these drawbacks a super-peer (SP) based scalable and robust Quality of Service (QoS) architecture for WiFi P2P networks was proposed in [7]. A SP is a powerful device, which intelligently and collectively manages the operations of a peer community.

Load balancing has been extensively used in distributed architectures and is viewed as migrating units of work from heavily loaded to lightly loaded servers [9]. Load Sharing involves distribution of load to reduce idle on servers.

In this paper, we propose two novel techniques to share the load using replication techniques. In PPR, the hosting super-peer periodically send replicas of the most frequently accessed files to remote SPs based on the global access frequency. By replicating, the hop count to search for a file is reduced. A SP receiving a replica also informs its neighboring super-peers about the replica through a restricted gossiping algorithm [2, 5]. ODR, performs replication based on local access frequencies. A request for replication is initiated by a super-peer if the access frequency of a particular file reaches a predefined threshold. This technique allows super-peers to dynamically adapt to changes in access behavior, however, it is greedy as each super-peer tries to perform replication based on its own needs rather than replicating from a global perspective as done in PPR.

The objective of our work is to improve the performance. This is achieved in load sharing by replicating most frequently accessed files and restricted gossip of file location information to benefit nearby peers.

The main contribution of this paper are as follows:

1. Two algorithms are proposed:
   (a) Periodic Push-based Replication (PPR);
   (b) On Demand Replication (ODR)
2. Simulation results show that through replication and restricted gossiping, significant performance improvement can be achieved. The average hop count for peers to fetch data files is reduced by over 30% through PPR, while ODR reduces average hop count by over 10%.

The rest of the paper is organised as follows. Section 2 presents a summary of background material and related

Proceedings of the 20th International Conference on Advanced Information Networking and Applications (AINA'06)
1550-445X/06 $20.00 © 2006 IEEE

work. In Section 3, the PPR and ODR strategies are described. Analytical model of the proposed schemes in Section 4 are proposed. Simulation results are reported in Section 5. Finally, this paper concludes in Section 6.

2. Related Work

Load sharing minimises load on serving entity by replicating popular files based on their access probabilities and threshold values. This section discusses some of the replication strategies and traditional load balancing approaches that deal with minimising load on the serving entity and puts our work in context.

A number of replication approaches are discussed in [6]. The owner replication approach replicates data object to the peer that has successfully located the query. This causes huge burden on peer to carry more information. In this approach, data objects are replicated along the search path that is traversed as part of the search in path replication. This, however, causes congestion in the network. Random replication replicates data items randomly. This approach may or may not have any effect on the load. Data objects are replicated a pre-defined number of times to control the spread of replica. This method does not adapt to changes in the environment and variable resource availability. Finally, the replication process aims at uniformly exploiting the storage resources available at peers while also trying to achieve uniform distribution of the replicas of a data object, i.e., for each data object approximately the same number of replicas exist. While this controls the overhead of replication, replicas may be found in places where peers do not access the files. In another work [4], the uniform and proportional replication strategies are described. In the uniform strategy, replications are uniformly distributed throughout the network (similar to random), while the proportional strategy replicates popular files more frequently. A proportional strategy makes popular items easier to locate and less popular items harder to find therefore increases the search time for these data.

A balancing strategy for DHTs based on Chord is proposed in [3]. In order to provide load-balancing, multiple hash functions are used instead of only one, and multiple peers, each choose the keys generated from the hash functions. However, when a peer is loaded, it transfers the load to neighboring peer, resulting in more redirections.

Our approach takes the cost of searching a data item and successfully replicates the most frequently accessed data files based on the access probabilities. It also uses restrictive gossiping to notify nearby super peers of the data location.

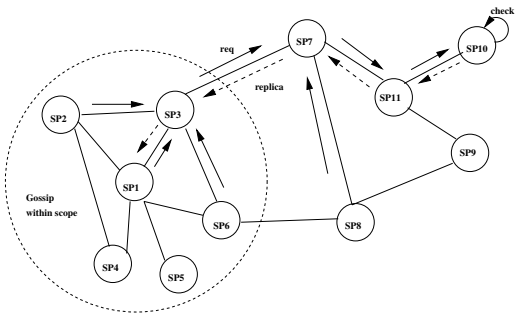

Figure 1. PPR

3. Replication Approaches

This section provides an overview of the proposed algorithms. SPs manage peer devices in a P2P system. Each peer has a unique id and register the files it wish to share with its SP. Each SP maintains access statistics of peers connected to it. This information is stored in file statistics module of the P2P architecture as described in [7]. Each SP coordinates activities of the peer devices. Any peer device joining/leaving the P2P system informs its SP. We propose the following two techniques:

3.1 Periodic Push-based Replication

In this algorithm, a periodic update is enforced on the SPs. Every SP maintains the access statistics of each file shared by its peers. If the access frequency of one of these files exceeds a predefined threshold value, the file is replicated to the SP which has requested the file most frequently. When a SP receives a replica, it uses the restricted gossip algorithm [2, 6] to propagate the file location to its neighboring SPs within its scope.

Figure 1 illustrates the operation of PPR. SPs $SP1$, $SP2$, $SP6$, and $SP8$ frequently requests file x. This file can be found at $SP10$. $SP10$ keeps track of the access statistics for file x. ie. the number of requests for file x from different SPs over time. If the access frequency of file x exceeds the threshold, then $SP10$ pushes the file to the SP which has requested x most frequently. The push operation is performed using the capacity-to-hop count routing algorithm as presented in [8]. For example, $SP10$ pushes file x to $SP1$. When $SP1$ receives the file, it restrictively gossips the location of file to other SPs within its scope (that is, $SP2$, $SP3$, $SP4$, $SP5$, and $SP6$) as shown in Figure 1. By doing so, these neighboring peers can take advantage of the replica, which is closer than the copy hosted at $SP10$.

Proceedings of the 20th International Conference on Advanced Information Networking and Applications (AINA'06)
1550-445X/06 $20.00 © 2006 IEEE

Algorithm 1 Periodic Push Based Algorithm

Start
for all peer devices attached to super-peer ($i = 1$ to N_p)
    Register peer ID
    Register sharable files with super-peer
At time = t (push interval)
    for $j = 1$ to N_{sp}
        Check global access frequency of each file
            if access frequency $>$ threshold
                PUSH replica to most requested SP (say SP_i)
                SP_i gossips within its scope
End

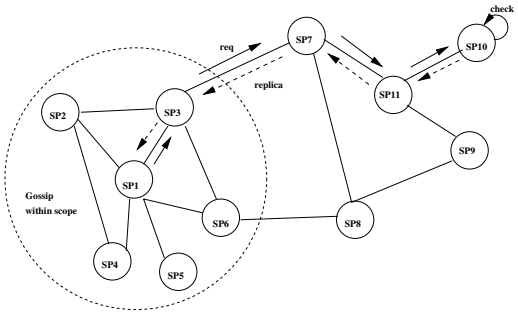

Figure 2. ODR

3.2 On Demand Replication

In this section, we propose On Demand Replication, that performs replication in a greedy manner. In ODR, SPs keep track of the access frequency of its connected peers (instead of the access frequency of the files these peers share as done in PPR). Whenever a SP (say SP_i) routes a request for one of its peers, it checks to see if the access frequency for the requested file has exceeded a predefined threshold. If the threshold is exceeded, SP_i sends a replication request to SP hosting the file (say SP_j). When SP_j receives the replication request, it sends SP_i a copy of the requested file. SP_i then will caches the file locally to answer further requests. Furthermore, SP_i uses the restricted gossip algorithm to propagate the file location to its neighboring SPs within its scope. The advantage of the ODR approach is that each SP request replicas based on access frequency of its peers. However, due to greedy nature of this strategy, ODR could lead to high replication overhead.

We illustrate in Figure 2 how ODR works. $SP1$ frequently requests for file x. This file x is found in $SP10$. $SP1$ checks its access frequency of file x. If the access frequency of file x by the peer community within $SP1$ exceeds the threshold, then a request is sent to $SP10$ to request for x. After receiving the request, $SP10$ sends a copy of the file back to $SP1$, which is then cached there for future requests. $SP1$ gossips location of file x to other SPs within its scope as shown in Figure 1.

Algorithm 2 On-Demand Algorithm

Start
for all peer devices attached to super-peer ($i = 1$ to N_p)
    Register peer ID
    Register sharable files with super-peer
if access frequency $>$ threshold
    for $j = 1$ to N_{sp}
        Send request to hosting SP to fetch replica
        Hosting SP push replica to requesting SP.
        Requesting SP gossips within its scope
End

4. Analytical Model

In this section, we develop analytical models to calculate the replication overhead and average access cost of PPR and ODR techniques. A P2P network consists of N_{sp} SPs, connected in a random graph topology. Each SP is connected to on average k other SPs. The distance between two SPs n_i and n_j is denoted as $d(n_i, n_j)$.

4.1 Access cost

Given a query for an object, average cost of the query is:

$$cost_{avg} = Pr(local) \times cost_{local} + Pr(remote) \times cost_{remote} \quad (1)$$

where $Pr(local)$ is the probability the query is answered by the local SP. $cost_{local}$ is the cost to fetch the object from the local SP. $Pr_(remote)$ is the probability the query is answered by a remote SP, and $cost_{remote}$ is the associated cost.

In our model, the traditional DHT (distributed hash table) technique is used to help normal peers locate the data objects. The cost of using DHT for a search is logarithmic [10], and can be calculated as:

$$cost_{search} = \frac{1}{2} \times log_2(N_{sp}) \times cost_{search\_msg} \quad (2)$$

where $cost_{search\_msg}$ is the cost to send a search request over one hop.

Assume the cost to transfer an object from one SP to an adjacent SP is $cost_{sp}$ and the cost to transfer an object from a SP to one of its child peer is $cost_p$. And denote the average distance between two SPs as d_{avg}.

For a query, if the local SP has the required object, then the search request only need to be sent over one hop. As a result:

$$cost_{local} = cost_p + cost_{search\_msg} \quad (3)$$

The cost of searching and accessing an object from a remote SP is higher, and can be calculated as:

$$cost_{remote} = cost_{search} + d_{avg} \times cost_{sp} + cost_p \quad (4)$$

$Pr(local)$ and $Pr(remote)$ can be calculated by looking at the query distribution. Assume the query distribution follows the Zipf distribution. The access probability of file i is equal to:

$$Pr(i) = \frac{1}{\sum_{x=1}^{D} \frac{1}{x}} \times \frac{1}{i} \quad (5)$$

Based on Eq. 5:

$$Pr(local) = \sum_{i=1}^{c} Pr(i) \quad (6)$$

where c is the number of objects available from the local SP.

If the query distribution is uniform, then $Pr(i) = \frac{1}{D}$ and $Pr(local) = \frac{c}{D}$.

Based on Eq. 6, $Pr(remote)$ can be calculated as:

$$Pr(remote) = 1 - Pr(local) \quad (7)$$

Finally, the average access cost per query can be calculated by substituting Eq.3, 4, 6 and 7 into Eq.1.

The access cost for a period t is therefore:

$$t \times N_p \times q \times cost_{avg} \quad (8)$$

where q is the query rate of each peer and $cost_{avg}$ is the average cost per query as defined in Eq. 1.

4.2 Replication overhead

4.2.1 PPR

In PPR approach, SPs pushes replicates to other SPs based on the predefined threshold denoted $thres$. The push frequency is denoted $freq$, therefore the time between pushes is $t_{push} = \frac{1}{frac}$.

The replication overhead for a SP n_i for each push period is:

$$thres \times d_{avg} \times cost_{sp} \quad (9)$$

Since there are N_{sp} SPs in the network, the total replication overhead per push period is therefore:

$$\text{overhead}_{push} = N_{sp} \times thres \times d_{avg} \times cost_{sp} \quad (10)$$

The replication overhead for the push-based method over a period t is therefore:

$$\text{overhead}_{push}(t) = \frac{t}{t_{push}} \times \text{overhead}_{push} \quad (11)$$

Table 1. Simulation parameters

parameter	Explanation	value
$time_{sim}$	Simulation duration	18000s
N_{SP}	Number of super-peers	57
N_P	Number of peers	570
U_P	Number of unique files	570
q	Query rate	0.5 queries/s
t_{push}	Gossip frequency (in PRR)	1 minutes
λ	Peer arrival rate	0.2/minute
μ	Peer departure rate	0.2/minute
d_{avg}	Average hops between super-peers	8.5
$CacheSize_{SP}$	Super-peer cache size	10 files
$Cost_{SP}$	Transfer cost between super-peers	1
$Cost_P$	Transfer cost between super-peers and peers	1

4.2.2 ODR

In the ODR scheme, replication occurs when the access frequency of a file exceeds the predefined threshold. The access probability of a file i is defined in Eq. 5. (If a uniform access distribution is used instead, $Pr(i) = \frac{1}{D}$ where D is the number of unique files in the system.

Given clients generate queries at a rate of q, in a time period t, the number of queries for file i equals:

$$numAccess(i) = N_p \times t \times q \times Pr(i) \quad (12)$$

Given a threshold $thres$, denote the probability that file i is access more than $thres$ times as $Pr(numAccess(i) > thres)$, the number of files to replicate in the period t is then:

$$numRep = \sum_{i=1}^{D} (Pr(numAccess(i) > thres)) \quad (13)$$

Based on Eq. 13, the replication overhead of the ODR method over a period t is equal to:

$$\text{overhead}_{onDemand} = numRep \times d_{avg} \times cost_{sp} \quad (14)$$

5. Simulation and Discussion

This section reports the performance evaluation of our proposed techniques using a discrete event simulator OMNETT++. The simulation model consists of 57 super-peers connected in a randomly formed P2P network. During simulation, peers join and leave the network following a Poisson process with an arrival rate of λ and departure rate of μ. Arrival and departures are uniformly distributed among the peers connected to different SPs. Peer queries are distributed among these data items based on the Zipf distribution with a skewness parameter $\alpha = 1$.

For the PPR scheme, replications occur periodically. We refer to the frequency of this happening as the gossip frequency, t_{push}. Table 5 summarises the parameters used for the simulation and their default values.

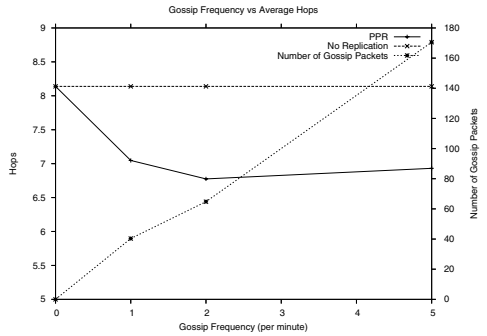

Figure 3. Gossip Frequency vs. Hops

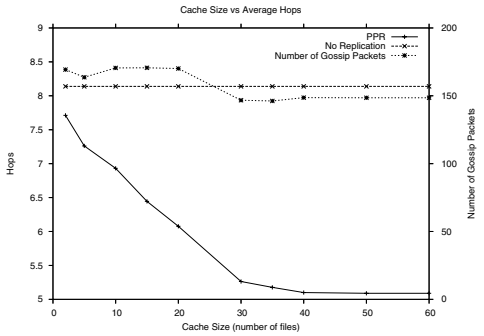

Figure 5. Cache size vs. Average hops

5.1 PPR Results

The results in Figure 3 show that an increase in gossip frequency does not necessarily lead to a drop in hop count. The reason for this is that the cache size of SPs are limited (default to 10 files in our simulation). Therefore, even when gossip frequency increases, SPs may not have the cache space to cater for replicas. It is found that the best performance for PPR is achieved when gossip frequency is around twice per minute. In this case, the average hop count is reduced from 8.2 to 6.8. This shows the benefit of replication where files are pushed and replicated at SPs where they are requested frequently. Further increase in gossip frequency beyond 1-2 times per minute only leads to marginal gain, while incurring a large replication overhead. The increase in gossip overhead is roughly linear to the increase in gossip frequency.

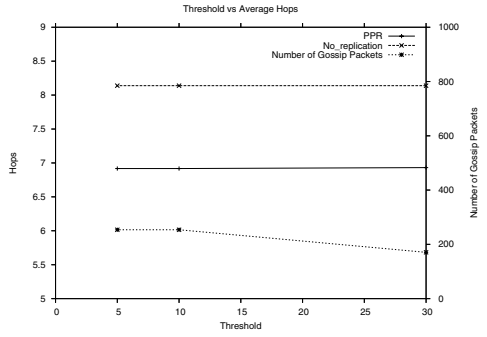

Figure 4. Threshold vs. Average Hops

We verify how the threshold setting will affect the performance in terms of the average hop counts for a peer to fetch a needed data file. We fix the cache size and replication frequency while varying the value of the threshold. Figure 4 presents the result for the PPR approach. The threshold value is the number of accesses of a particular data file before it is pushed to other SPs. It was found that the average hop count does not change significantly as the threshold value changes. This is due to queries being modeled following a Zipf distribution. This results in most queries targeting towards particular hot spots in the data set. This allows these hot files to reach the threshold quickly, while access of other files never reaches the threshold.

Varying the cache size of SPs in order to determine the optimal cache size is done in Figure 5. The replication frequency and replication threshold are fixed. It is found that the average hop count dropped by from over 7.7 to 5.1 when cache size is increase from 1 to 40. After this, further increase in cache size did not yield significant benefits. This is again due to the skewness of the access pattern. For each SP, after the most frequently accessed data have been cached, additional cache space provides diminishing gain. With a cache size of 30 files, PPR reduces average hop count by nearly 3 hops compared to where no replication is used. An interesting result is that even as cache size increases, the gossip overhead did not change significantly. This is because although larger caches can store more replicas, replication only occurs when access frequency exceeds the threshold. As a result, cache size has limited effect on replication overhead. It is found that a cache size of 30 provides good performance.

5.2 ODR Results

In Figure 6, the relationship between cache size and average hops for ODR is shown. The average hop count and gossip overhead remain steady as cache size changes. This is because in ODR, replication and gossiping only occurs when the access frequency of a file reaches the threshold. As most peers show common interest for hot spots, and the queries have been aggregated along the path to the hold of the data item. As long as hot data have been cached along the routing path, future queries can benefit from it.

Figure 7 plots the average hop count against replication threshold. It was found that the average hop count dropped significantly as the replication threshold dropped. However, reducing the replication threshold also result in fast increase

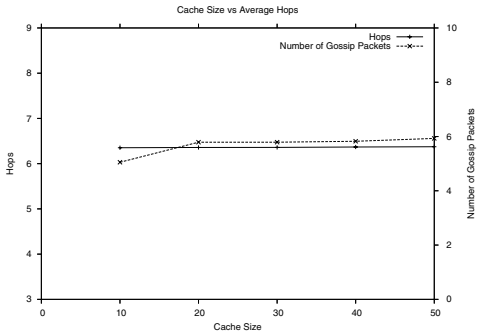

Figure 6. Cache Size vs. Average Hops

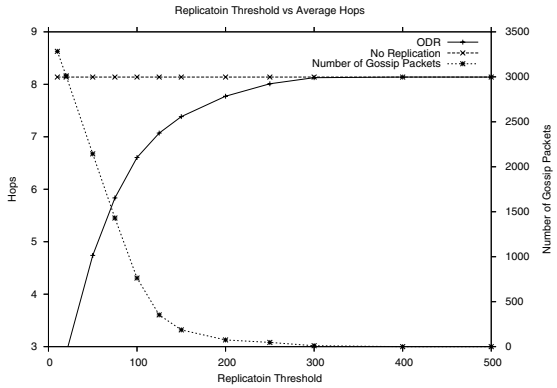

Figure 7. Replication Threshold vs. Hops

in gossip overhead. This result shows that there is a trade-off between the cost of replication and gossiping and the reduction in hop count. Since ODR is a greedy technique where each SP attempts to fetch replicas it most frequently accesses, huge savings (in terms of hop count) are achieved at the expense of flooding the network with gossip messages. The optimal threshold value depends on the size of the gossip messages and the cost to transmit over a hop. We can see that as the threshold value increases, the hop count of ODR approaches that of no replication. This is because as the threshold increases, fewer files are replicated.

It is found that ODR outperforms PPR when the threshold value is low. This is because in the ODR algorithm, replication is performed on demand, as a result ODR is more sensitive to changes in access probabilities. By capturing and adapting to these changes quickly through replication, ODR is able to reduce the average hop count to fetch a data file compared to PPR. However, ODR is somewhat greedy because each SP demand replicas based on its own needs. This result in higher variation between the performance received by different SPs. PPR algorithm focuses on balancing the global access cost. By periodically pushing replicas, the replication overhead of PPR can be controlled (by restricting the push frequency). An interesting result is that the two algorithms converges as the threshold value approaches 40, which corresponds to the standard deviation of the peer access pattern.

6. Conclusion

Resource sharing by mobile devices in P2P environment is vital to the success of next generation P2P networks that operate in wireless environments. In this paper, we have proposed two novel techniques to perform load sharing in mobile P2P networks through replication. In the first technique, Periodic Push-based Replication (PPR), hosting super-peers periodically send replicas of the most frequently accessed files to remote selected super-peers. The second proposed technique, On-Demand Replication (ODR), performs replication based on access frequency. The ODR approach allows super-peers to dynamically adapt to changes in access behavior. Extensive simulation has been conducted to evaluate the performance of the proposed techniques. It is found that through replication, the average number of hops to reach a data file can be reduced by over 30%. This presents a significant improvement, especially in mobile P2P networks where a reduction in hops result in lower transmission cost, reduced latency and improved QoS.

References

[1] K. Aberer. Efficient search in unbalanced, randomized peer-to-peer search trees. Technical report, EPFL.

[2] S. Banerjee, S. Lee, R. Braud, B. Bhattacharjee, and A. Srinivasan. Scalable resilient media streaming. In *IEEE INFOCOM 2004*.

[3] J. Byers, J. Considine, and M. Mitzenmacher. Simple load balancing for distributed hash tables. In *Proceedings of 2nd International Workshop on Peer-to-Peer Systems*, 2003.

[4] E. Cohen and S. Shenker. Replication strategies in unstructured peer-to-peer networks. In *Proc. ACM SIGCOMM*.

[5] A. D. et. al. Epidemic algorithms for replicated database maintenance. volume 22, pages 8–32, 1998.

[6] Q. L. et. al. Search and replication in unstructured peer-to-peer networks. In *Proc. of International conference on Supercomputing*, 2002.

[7] S. Rajasekhar, I. Khalil, and Z. Tari. A scalable and robust qos architecture for wi-fi p2p networks. In *Proc. LNCS ICDCIT'04*, volume 3347, pages 65–74, Bhubaneswar, India, Dec. 2004.

[8] S. Rajasekhar, B. Lloyd-Smith, and Z. Tari. Qos path routing based on capacity to link ratio in networks. In *Proc. of NPDPA*.

[9] M. Roussopoulos and M. Baker. Practical load balancing for content requests in p2p networks. Technical report, Stanford University.

[10] I. Stoica, R. Morris, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM*.