A Gradient Topology for Master-Slave Replication in Peer-to-Peer Environments

Jan Sacha and Jim Dowling

Distributed Systems Group, Trinity College Dublin, Ireland
{jsacha,jdowling}@cs.tcd.ie

Abstract. Open peer-to-peer architectures offer many possibilities for replicating database content, but designers have to deal with problems such as peer churn rates and inherent uncertainty in decision making. The lack of global knowledge of peer characteristics poses the specific problem of reliable peer discovery for database replica placement. This paper describes a self-organising algorithm for generating a peer-to-peer gradient topology that helps to solve the problem of replica placement through the clustering of peers with similar uptime and performance characteristics. We evaluate the algorithm by simulation, and propose an approach for master-slave replication that exploits the properties of the presented topology.

1 Introduction

The peer-to-peer paradigm for building distributed systems has become extremely popular and it has received increased attention as more and more novel applications are invented and successfully deployed. The main advantage of the paradigm is that it allows the construction of systems with unprecedented size and robustness, mainly due to their inherent decentralisation and redundant structures. However, the P2P paradigm introduces challenges that are often not dealt with properly in many proposed P2P architectures. Massive scale and very high dynamism makes it impossible to capture and maintain a complete picture of the entire P2P network, consequently, a peer or any other entity is only able to maintain a partial or estimated view of the system. Inherent decentralisation, an open environment, lack of trust and unreliable communication introduce distributed decision making problems. In particular, there is a problem in the database field of how to select the most suitable peers for storing data.

Existing P2P systems, such as Distributed Hash Table (DHT) based approaches, usually assume that all peers are similar and have equal capabilities for maintaining data [1]. For example in Chord [2] it is assumed that the distribution of resources among the peers is uniform. However, this was shown not

The work described in this paper was partly supported by the Information Society Technology Programme of the Commission of the European Union under research contract IST-507953 (DBE).

to be the case in real-life systems, where the distribution of peer characteristics, such as the number of connections, the uptime, available bandwidth or the storage space, usually exhibit the so called scale-free or heavy-tail property [3–6]. Systems that do not address the heterogeneity of the environment and do not adapt their structures to the environment often suffer poor performance, especially in the face of high churn rates, i.e., high frequency of peers entering and leaving the system [7–9].

The main contribution of this paper is a self-organising neighbourhood selection algorithm, that clusters peers with similar performance characteristics and generates a gradient network topology that helps to solve the problem of dynamic replica placement. We briefly describe an approach for master-slave database replication, a heuristic for master election and a heuristic for replica discovery, which all exploit the properties of the gradient topology in order to improve the stability of the replicas and minimise the overhead for replica maintenance. We evaluate the neighbourhood selection algorithm through simulation and performance measurements.

The rest of the paper is organised as follows. In section 2 we discuss the general requirements for data distribution and replica placement. In section 3 and 4 we introduce the concept of gradient topology and we demonstrate a neighbour selection algorithm that generates the proposed topology. In section 5 an approach for master-slave replication based on the gradient topology is presented. Section 6 contains a detailed description of the simulation settings and the experimental results. Finally, in sections 7 and 8 we discuss the related work and our plans for future work.

2 Peer Utility Metrics

When addressing the persistent data requirements for a distributed system, we must decide on where to store the data. There are two extremes; one is to store all data in a centralised server, which introduces scalability problems, and the other one is to partition the data among a set of peers using some indexing scheme, for example a distributed hash table. Many existing P2P systems assume that all peers have identical capabilities and responsibilities, and that the data and load distribution is uniform among all peers [1, 2]. However, this approach has a drawback that the use of low bandwidth/stability/trust peers to store data can potentially degrade the performance of the entire network [7].

To solve this problem, many systems only allow data to be stored on the fastest, highest bandwidth, and most reliable, trusted peers, called *superpeers* [8, 9]. This approach, however, introduces another problem of superpeer election. It is not obvious how to identify and select the superpeers from the set of peers in the network, mainly due to the lack of a global knowledge about the system. Solutions based on flooding potentially require communication with all N peers in the network. Other solutions include hard-wiring them in the system or configuring them manually. However, this conflicts with the assumptions of self-management, decentralisation, and the lack of a central authority that controls

the structure of the system. An adaptive self-organising system is preferable, where the peers automatically and dynamically elect superpeers, accordingly to the demand, available resources and other runtime constraints. Alternatively, the system may resign from the two-state distinction between superpeers and ordinary peers and it may assign roles for peers relative to their capabilities.

The selection of peers for replica placement could potentially be based on criteria such as peer stability, available bandwidth and latency, storage space, processing performance, an open IP address and willingness to share resources. Peer availability, or uptime, is especially relevant, since every peer entering or leaving the system introduces extra overhead, due in part to required data transfers or routing structure reconfiguration. The system could also employ a peer reputation model and use it as a criteria for replica placement, for example only the most trusted peers might be allowed to host a replica.

We define a peer's *utility* as a function of the above parameters. The utility is a critical factor in the algorithm that generates the P2P gradient topology and is used in the replica placement strategy presented in this paper.

In a closed system, where all peers trust each other, it is sufficient that every peer evaluates its own utility level. Neighbouring peers can exchange the utility information without any verification procedure, since trust is assumed. In an open, untrusted environment, a separate mechanism is needed to assure that the utility claimed by the peers corresponds to their actual status. Managing trust in a decentralised system, however, is beyond the scope of this paper and will not be discussed further.

A key principle in our approach is that *persistent data is stored by the highest utility peers*. This strategy addresses a problem faced by many existing P2P systems, where some data items, especially less popular, are hardly accessible or even lost due to peers' instability or lack of resources such as storage space or bandwidth. In our design, the system tries to maximise data availability, security and the quality of service by placing data replicas on the most reliable, high performance hosts.

3 Gradient Topology

We propose a P2P topology, which we call a *gradient topology*, where the highest utility peers, maintaining persistent data, are highly connected with each other and form a logical *core* of the network, while the network around the core is composed of other peers considered less performant or less reliable (see Figure 1). Assuming the scale-free, heavy-tailed distribution of resources [4–6], a great majority of peers belongs to the latter category. The number of core peers is relatively small, but the amount of contributed resources and stability of core peers is significantly higher than the outer peers. The main advantages of grouping high utility peers in a logical core are the following:

– Searching for high utility peers maintaining replicas, or suitable for maintaining replicas, is less expensive in terms of number of messages generated,

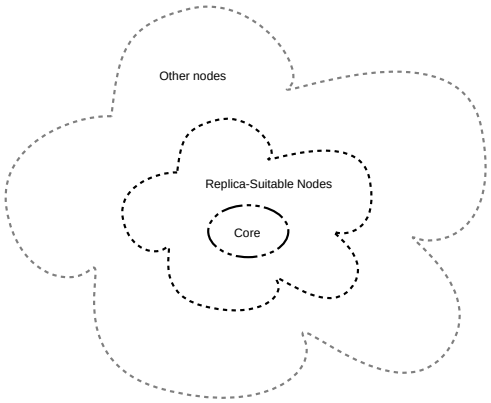


Fig. 1. Gradient topology based on peer utility.

since it only requires communication with a small subset of peers in the network.
- The overhead for replica synchronisation is reduced since peers storing replicas are located close to each other, at least in terms of number of hops, and are connected by stable, good quality routes.

In practice, the core peers act more as servers, while the outer peers act more as clients. The core peers should be well-connected, have high bandwidth and processing power, and should be able to maintain a relatively high number of connections. On the other hand, peers far from the core are not suitable for maintaining replicated data, due to poor reliability, resource constraints or the owner's unwillingness to contribute resources to the system. It is not desirable to place such peers in the core since they would decrease the system's performance.

4 Neighbour Selection Algorithm

In order to create the gradient P2P topology and enable the emergence of a stable core, we initially thought of the following neighbour selection rule: *two peers may become neighbours if they have similar utility status.* Additionally, high utility peers should have more neighbours, since they have more resources available to maintain network connections.

However, our early experiments showed that a greedy selection policy, where peers always select neighbours with the most similar characteristics to their own, leads to high network clustering and in consequence long distances between peers. In some cases the clusters got disconnected and the network became partitioned. Another problem was that the highest utility peers did not always connect to a single core, and sometimes there were multiple clusters of high utility peers separated from each other by a number of lower utility peers.

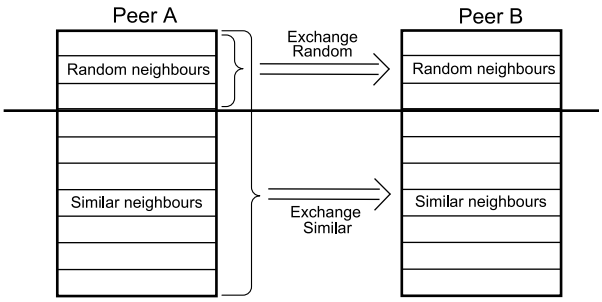


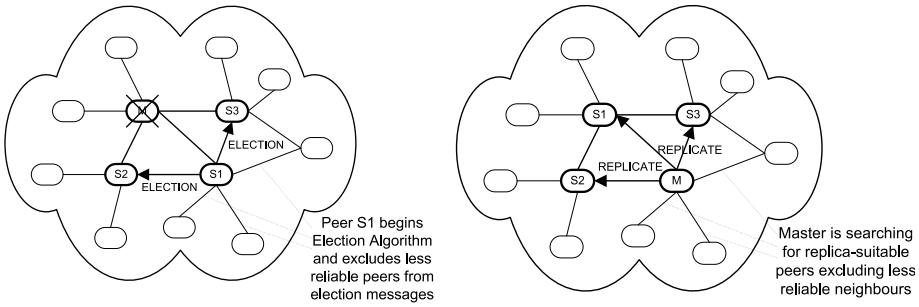
Fig. 2. Neighbourhood set exchange from Peer A to Peer B.

We improved the algorithm by allowing the peers to connect to other non-similar peers, with the connection probability exponentially decreasing with the difference in peer utility. However, it turned out that a randomised strategy, where a percentage of neighbours was always selected at random, gave the best results. Random connections serve several purposes. First of all, they allow the peers to discover potential neighbours with similar utility level, even in remote regions of the network, which in turn enabled the formation of a single cluster containing the highest utility peers. Random connections thus play a similar role to the exploration in traditional multi-agent systems. Second, random links prevented the graph from being disconnected. As shown in [10], even a small number of random connections, for example 20 per peer, is sufficient to make the probability of network partitioning negligibly small in practice. Finally, our randomised algorithm has the advantage that it is quite simple and it does not require tuning parameters specific to the deployment environment, such as the network size or average peer connectivity.

A peer maintains two independent sets of neighbours, randomly-selected connections, and greedily-selected connections to peers with similar utility status (see Figure 2). New connections are discovered by gossiping, i.e., by randomly contacting already existing neighbours and exchanging with them the lists of connections. It is important to note, that the connections inserted to the random sets are always selected from other peers' random sets, which guarantees that the sets remain random. The details of the algorithm implementation and the simulation settings are described in Section 6.

5 Replication Strategy

In this section, we demonstrate how the gradient network topology can be exploited by a master-slave database replication strategy. We present an approach, where the replica placement is based on peer utility, available resources and demand. Due to the information contained in the network structure, the selection of high utility peers suitable for replica placement does not require communication between all peers in the system.



(a) Peers S1, S2 and S3 compare their utility to elect a new master.

(b) Master compares the utility of peers S1, S2 and S3 to select the best peer for slave replica placement.

Fig. 3. Slave replica searching and master election algorithms exploiting the implicit information about peer utility contained in the network topology.

We assume that each peer can potentially create an independent database, and replicate it over some of the peers in the network in order to improve its availability and persistence guarantees. A peer that creates the first copy of a database, which we call the *master replica*, becomes the database owner. Subsequent replicas of the database hosted by other peers are called *slave replicas*. The users issue *queries* to the database that can be resolved by any replica. The owner, and potentially other authorised users, can also *update* or *delete* a database. There is only one master replica of a database and it is responsible for handling and synchronising updates. Slave replicas are created automatically, on demand.

We restrict the set of peers that are allowed to create database replicas to those with utility above a replica-suitable threshold. A *master replica threshold* defines a minimum peer utility to host a master replica, and a *slave replica threshold* defines a minimum utility level to host a slave replica. It is important to note, that the system does not require any form of consensus between peers on the threshold values, since the thresholds can be determined for each database individually.

5.1 Replica Placement

In our approach, a peer that already hosts a replica, in particular the master, requests a new replica placement on one of its neighbours when a certain condition is met, for example when the number of user queries exceeds some critical level and a new replica is needed to handle the load. It is crucial in this approach that a peer initiating the replication must be able to find other peers suitable for hosting new replicas, i.e., peers with utility above the slave replica threshold. This should be satisfied in the gradient topology, since all peers storing replicas

are clustered in the highly connected core, and share a similar, high utility status. Search messages do not need to be propagated to lower utility neighbours, since all high utility peers are located in the core of the network (see Figure 3(b)).

Replicas are removed on demand, adaptively. When a peer decides that the cost of a replica's maintenance is higher than the cost of handling queries, for example the frequency of queries drops below some threshold value, the replica can be deleted. Alternatively, replicas might be selected for removal using a Least Recently Used strategy as in Freenet [11].

5.2 Replica Synchronisation

Database replicas must be synchronised between the master and the slaves after update operations. We add the constraint that an update operation, either a modification, an addition or a removal, must be performed on the master replica, while queries can be handled by any slave. This requirement shouldn't significantly reduce the scalability of the system if the rate of queries is much higher than the rate of updates, which is commonly the case for example in name services or knowledge base systems. If an update is delivered to an ordinary replica, the replica forwards it to the master, and the master propagates the update to all replicas. This guarantees that concurrent updates from different peers are serialised and sent in the same order to all copies of the database, hence, there are no write-write conflicts.

The updates can be propagated either *instantaneously*, or in a *lazy* fashion, for example by *periodic gossiping*, with the latter technique being used to reduce bandwidth consumption and improve scalability at the cost of reduced data consistency. Lazy replica synchronisation can be initiated either by the master or by the slave, for instance after a peer crash or a restart. Thus the system provides eventual consistency of the replicas [12]. The core peers storing replicas should be well-connected, have high bandwidth and processing power, which should enable fast data dissemination and frequent replica synchronisation.

5.3 Master Election

In the P2P gradient topology, peers have relative positions defined by their utility metric. This allows us to develop an efficient election algorithm that doesn't require flooding, as peers can use a heuristic that excludes peers with lower utility, i.e., topologically further from the core, when sending election messages (see Figure 3(a)). This heuristic, however, does not guarantee that the highest utility peer will become a master unless all peers in the core are fully connected. Therefore, we modify our heuristic to provide a directed, gossiping election model. The election initiating peer also sends the election message to a certain number of neighbouring peers with lower utility, but still inside the core, and they can restrict further propagation of the message to only peers with higher utility. Given high enough connectivity between peers in the core, within a certain probability the peer with the highest utility should win the election.

Algorithm 1: Main loop of the simulation

```
for N steps of the simulation do
    increase the number of peers by 1%;
    probabilistically remove peers according to their utility;
    forall peers p in the network do
        p.step();
    end
end
```

Fig. 4. Main loop of the simulation.

5.4 Replica Discovery

A searching mechanism is needed for peers to discover nearby replicas of a database they request access to. Traditional unstructured systems, such as Gnutella, have used flooding algorithms for finding resources. This approach works well for highly replicated data, however, it doesn't scale in principle. A number of techniques have been proposed to improve search in unstructured P2P networks, such as random walk, iterative deepening or routing indices [13].

For the topology presented in this paper, we are designing a gradient routing algorithm that will be based on two main factors: the neighbour utility information (utility gradient), and heuristic values learned by the system (as for instance in Freenet [11]). By increasing the probability of forwarding queries to high utility neighbours, the algorithm can effectively route queries towards the core of the network.

6 Evaluation

We evaluated our approach by modelling a P2P network in RePast, a multi-agent simulation toolkit for large-scale systems. The simulation was implemented in Java. A network was created consisting of over 100,000 peers, which we believe is sufficiently large to model realistic large-scale applications. The experiments ran on a Pentium 4 machine with a 3GHz processor and 3GB main memory.

The simulation is based on a discrete time model and all events are executed in discrete time steps. The actions performed by peers are synchronous, however, our algorithm does not rely on the peer synchrony and hence the results obtained in the experiments can be generalised for asynchronous environments as well. Following the assumptions of decentralisation and a lack of a global view of the system, each peer maintains a limited number of neighbours and performs actions using local knowledge only. We assume also a scale-free distribution of resources with the maximum number of peer connections following the power-law (Pareto) distribution, starting from 10 connections and reaching about 50 neighbours for the most powerful peers. Similarly, the utility distribution follows the power-law. We model the network growth by adding new peers at each step

Algorithm 2: Agent step

```
if number of neighbours = MAX_NEIGHBOURS then
    disconnect random neighbour;
end
if number of similar neighbours < MAX_SIMILAR then
    choose randomly neighbour p from all known neighbours;
    get all neighbours n_1..n_k from p;
    choose peer n with the most similar utility from n_1..n_k;
    connect to n;
end
if number of random neighbours < MAX_RANDOM then
    choose randomly neighbour p from all known neighbours;
    get all random neighbours n_1..n_k of peer p;
    choose randomly peer n from n_1..n_k;
    connect to n;
end
```

Fig. 5. Algorithm performed by an agent at one step of the simulation.

of the simulation, where we start with a network containing only one peer, and at each time step the size is increased by 1 percent. Additionally, at each step a number of peers are removed, which models random failures or peer departures, with the probability of a peer departure being inversely proportional to its utility. Bootstrapping of the system is implemented with a centralised name repository containing the 50 most recent peer names, where peers are added and removed in FIFO order. Figure 4 shows the pseudo-code of the simulation.

Figure 5 shows the randomised algorithm performed by each peer at every step of the simulation. A peer maintains two independent sets of neighbours, randomly-selected connections, and greedily-selected connections to peers with similar utility status. The latter set is twice as large as the former. At every step, if the number of neighbours of a peer reaches the maximum, the peer drops a random connection, and attempts to establish a new one by gossiping with a neighbour.

Figure 6 presents a visualisation of a sample network consisting of 200 peers, generated by the neighbour selection algorithm, where we can see that the topology evolved to the desired form where the highest utility peers are clustered together and constitute a stable core.

Figure 7 shows the measurement results obtained from our simulation. We can see that the average path length between peers is relatively low (about 5-6 hops for 100,000 peers), and that it varies with peer utility. The average distance between the highest utility peers is lower than the average distance between lower utility peers. This confirms our observation that the highest utility peers are highly connected with each other and form a stable core of the network.

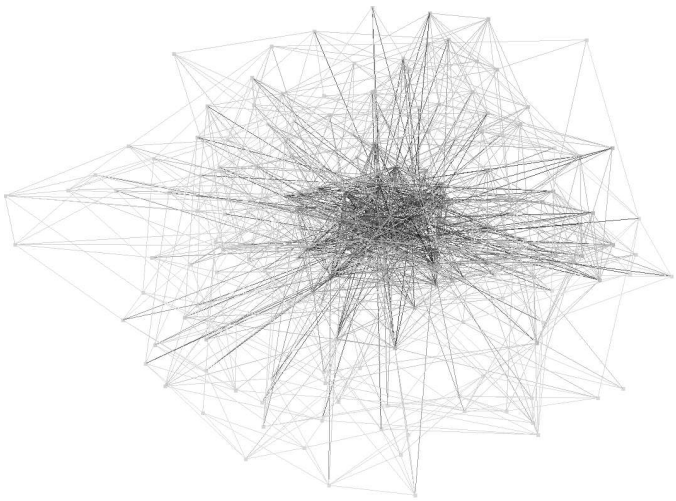
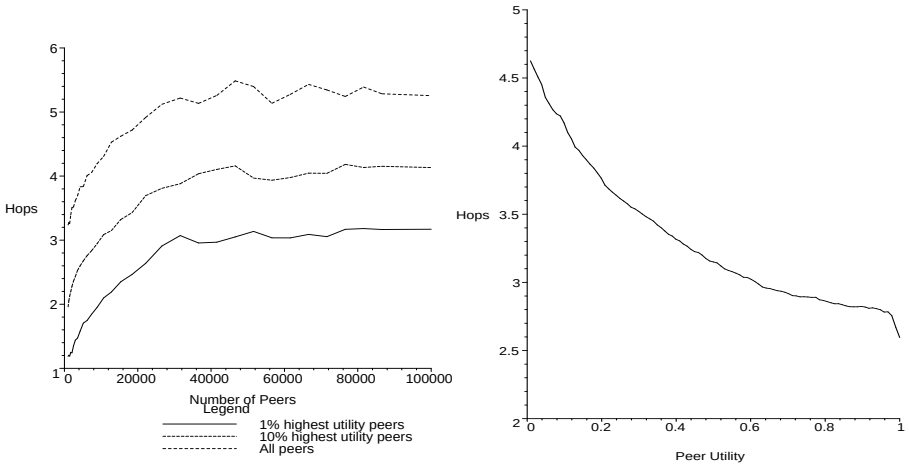


Fig. 6. Visualisation of a 200-node network in our RePast simulation using the Fruchmen-Reingold algorithm. High utility peers are marked with dark colors. Stable core of the network is visible in the center.

7 Related Work

Most existing P2P systems that exploit the heterogeneity of the environment are based on structured P2P overlays. In Chord [2] it has been noted that a single physical peer can host multiple *virtual peers,* and therefore, the heterogeneity of resources in a DHT network can be addressed by assigning more virtual peers to higher performance hosts. OceanStore [14] proposed to elect a primary tier "consisting of a small number of replicas located in high-bandwidth, high connectivity regions of the network" for the purpose of handling updates, but no specific algorithm for the election of such a tier is presented. Mizrak et. al. [9] proposes a super-peer based approach for the exploitation of resource heterogeneity, however, unlike our architecture, it relies on a DHT overlay.

In the field of unstructured P2P networks, there has been a lot of work devoted to searching (e.g. [13]) and to replication strategies [15], but there has been limited research on network topology generation and peer neighbourhood selection algorithms. Yang and Molina [8] investigate general principles of superpeer-based networks and give practical guidelines for the design of such networks, however, they don't describe any specific algorithms for super-peer election or network structure maintenance. The closest to our approach is the work of Jelasity, in particular his research on decentralised topology management (T-Man [10]), however, he doesn't address the problem of reliable peer discovery and dynamic replica placement in an open P2P system.



(a) Average distance between peers as a function of the network size.

(b) Average distance between peers as a function of peer utility, network size 100,000 peers.

Fig. 7. Results of the neighbourhood selection algorithm.

8 Conclusions and Future Work

This paper describes the general requirements for data replication in peer-to-peer environments. We have proposed a gradient network topology where the highest utility peers are highly connected with each other and form a logical core suitable for maintaining data replicas. A self-organising neighbourhood selection algorithm has been presented that generates the proposed gradient topology by clustering peers with similar uptime and performance characteristics. The algorithm has been evaluated through simulation, and measurement results have confirmed that the approach is scalable and robust.

The main advantage of the gradient topology is that the network structure contains information about the peer utility, which allows the peers to discover other high utility peers without flooding the entire network with search messages. The gradient topology allows the search space to be limited to a small subset of all peers in the system. This property allows us to address the problem of dynamic replica placement, master replica election, and replica discovery in an open, decentralised environment. The gradient topology should also reduce the cost of replica maintenance, since peers storing data replicas are located close to each other and are connected by stable, high-capacity routes.

Our project is still at an initial stage and requires a lot of further research. We are planning to develop a heuristic routing mechanism based on peer utility gradient, which will allow peers to discover database replicas in their proximity. We are also going to evaluate the proposed replica placement strategies and the

master election algorithm. We are building a prototype implementation based on the Berkeley DB middleware.

References

1. Rowstron, A.I.T., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Proceedings of the 18th International Conference on Distributed Systems Platforms, Springer (2001) 329–350
2. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: SIGCOMM Computer Communication Review. Volume 31., ACM Press (2001) 149–160
3. Barabsi, A.L., Bonabeau, E.: Scale-free networks. In: Scientific American. Volume 288. (2003) 60–69
4. Sen, S., Wong, J.: Analyzing peer-to-peer traffic across large networks. In: Transactions on Networking. Volume 12., ACM Press (2004) 219–232
5. Leibowitz, N., Ripeanu, M., Wierzbicki, A.: Deconstructing the kazaa network. In: Proceedings of the 3rd International Workshop on Internet Applications, IEEE Computer Society (2003) 112–120
6. Pouwelse, J., Garbacki, P., Epema, D., Sips, H.: The bittorrent p2p file-sharing system: Measurements and analysis. In: the 4th International Workshop on Peer-To-Peer Systems, Cornell, USA (2005)
7. Rhea, S., Geels, D., Roscoe, T., Kubiatowicz, J.: Handling churn in a dht. In: Proceedings of the USENIX 2004 Annual Technical Conference. (2004) 127–140
8. Yang, B., Garcia-Molina, H.: Designing a super-peer network. In: Proceedings of the 19th International Conference on Data Engineering, IEEE (2003) 49–60
9. Mizrak, A.T., Cheng, Y., Kumar, V., Savage, S.: Structured superpeers: Leveraging heterogeneity to provide constant-time lookup. In: Proceedings of the 3rd IEEE Workshop on Internet Applications. (2003) 104–111
10. Jelasity, M., Babaoglu, O.: T-man: Gossip-based overlay topology management. In: the 3rd International Workshop on Engineering Self-Organising Applications, Utrecht, The Netherlands (2005)
11. Clarke, I., Sandberg, O., Wiley, B., Hong, T.W.: Freenet: A distributed anonymous information storage and retrieval system. In: Proceedings of the 1st International Workshop on Designing Privacy Enhancing Technologies, Springer (2000) 46–66
12. Tanenbaum, A., van Steen, M.: Distributed Systems: Principles and Paradigms. Prentice Hall, New York (2002)
13. Yang, B., Garcia-Molina, H.: Improving search in peer-to-peer networks. In: Proceedings of the 22nd International Conference on Distributed Computing Systems, IEEE Computer Society (2002) 5–14
14. Kubiatowicz, J., Bindel, D., Chen, Y., Czerwinski, S., Eaton, P., Geels, D., Gummadi, R., Rhea, S., Weatherspoon, H., Weimer, W., Wells, C., , Zhao, B.: Oceanstore: An architecture for global-scale persistent storage. In: Proceedings of the 9th international Conference on Architectural Support for Programming Languages and Operating Systems. (2000) 190–201
15. Lv, Q., Cao, P., Cohen, E., Li, K., Shenker, S.: Search and replication in unstructured peer-to-peer networks. In: Proceedings of the 16th International Conference on Supercomputing, ACM Press (2002) 84–95