A Self-Organising Topology for Master-Slave Replication in Peer-to-Peer Environments

Jan Sacha and Jim Dowling

Distributed Systems Group, Trinity College Dublin
{jan.sacha, jim.dowling}@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 a specific problem for a master database that needs to decide on which peers to use as slave replicas. This paper describes a self-organising algorithm for generating a peer-to-peer topology that helps to solve the problem of slave replica placement through the clustering of peers with similar uptime and performance characteristics. In particular, our solution can be used to build a highly scalable knowledge base where the rate of querying is much larger than the rate of updates. We briefly discuss the design of such a knowledge base for a peer-to-peer service-oriented architecture.

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. In particular, for databases the P2P approach offers new possibilities, since it enables the utilisation of a large number of resources, such as the storage space or processing power of peers in the network.

However, the peer-to-peer 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 the specific problem in the database field of the data distribution of how to partition the data among the peers. A peer introducing new data, or creating a new replica, has to decide which of the other peers in the network is the most suitable to host the data.

Existing P2P systems, such as for example Distributed Hash Table (DHT) based approaches, usually assume that all peers are similar and have equal capabilities for maintaining data [11], which simplifies the design. For example it is often assumed that the distribution of resources among the peers is uniform [13]. However, this was shown not 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 [1,12,6,9]. Systems that do not address the heterogeneity of the system 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 [10,16,8].

In the Digital Business Ecosystem (DBE) Project we are building a large-scale distributed storage system, which we call the DBE Knowledge Base, where the system's topology and replica placement dynamically adapts to reflect the heterogeneities in the network and peer properties. The main assumptions of the storage system are that the data is persistent and highly replicated, and that the replicas are managed — the system keeps track of all replicas so that their owners are able to update or delete them. We restrict the replica placement to the most reliable, high performance peers only. We also make the assumption that the data is queried much more frequently than updated.

The main contribution of this paper is a self-organising neighbourhood selection algorithm, that clusters peers with similar reliability and performance characteristics and generates a network topology that helps to solve the problem of dynamic replica placement. We propose a replication strategy based on the master-slave paradigm and a heuristic for master election, which both exploit the properties of the emergent topology in order to improve the stability of the replicas and minimise the overhead for replica maintenance. We evaluate our approach 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 present our neighbour selection algorithm, and a replication strategy based on the resultant topologies. Section 5 contains a detailed description of the algorithm, the simulation and a discussion of the experimental results. Finally, in sections 6 and 7 we discuss the related work and our plans for future work.

2 Peer Reliability 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 nodes [11,13]. One potential problem when partitioning the data among peers is that the use of peers with lower bandwidth/stability/trust to store data would degrade the performance of the entire network [10].

To solve this problem, many systems only allow data to be stored on the fastest, highest bandwidth, and most reliable trusted peers, called *superpeers* [16,8]. It is non-trivial though, how to identify and select the superpeers from the set of peers in the system, mainly due to the lack of global knowledge of the system. Solutions based on flooding potentially require communication with all N nodes in the system. 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 relatively 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 to for example 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 believe that many of the above parameters are strongly correlated, that is stable peers are more likely to offer high bandwidth and low latency, have better reputation and probably provide more connections, storage space and CPU power then other peers.

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

In a closed system, where all peers trust each other, it's sufficient that every peer evaluates its own as reliability level. Neighbouring peers can exchange the reliability information without any verification procedure, since trust is assumed. In an open, untrusted environment, a separate mechanism is needed to assure that the reliability claimed by the peers corresponds to their actual status. In particular, the system should be protected against malicious peers providing fake reliability information, either about themselves or about other peers, peers changing the identity or creating multiple *virtual peers*. The system should be also robust against *cliques* of greedy nodes. These challenges, however, are beyond the scope of this paper and will not be discussed here.

A key principle of the system is that *persistent data is stored by the most reliable peers*. This strategy addresses a problem faced by many existing peer-to-peer 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 hosts.

3 Neighbour Selection Algorithm

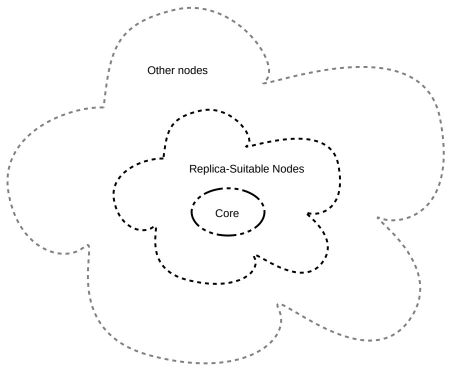


Fig. 1. Self-organising system topology based on peer reliability.

We propose an unstructured P2P architecture where the most reliable 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 the other peers, considered less reliable. Assuming the scale-free, heavy-tailed distribution of resources [12,6,9], a great majority of peers belongs to the latter category. The number of core peers is relatively small, but the amount of available resources and stability of core peers is significantly higher then of the outer peers. The main advantages of grouping reliable peers in a logical core are the following:

– Searching for reliable peers maintaining replicas, or suitable for maintaining replicas, is less expensive in terms of number of messages generated, since it only requires communication with a small subset of peers in the network.
– The overhead for replica synchronisation is reduced since the replicas are located close to each other, at least in terms of the numbers of hops.
– Routes between peers storing data are more stable and up-to-date.
– Trust evaluation between peers storing data is less expensive.

The structure of the system can be seen as a set of concentric zones, where each zone consists of peers with similar levels of reliability. The inner zone, called the core by us, contains the most reliable peers. The subsequent zones are delimited by gradually lower and lower status peers, down to the last outer zone that contains the least reliable peers. Figure 1 illustrates this concept.

In practice, the core peers act more as servers, while the outer peers act more as clients. The core nodes should be well-connected, have high bandwidth and processing power, and should be able to maintain a relatively high number of connections. Consequently, this enables fast data dissemination and more frequent replica updates to peers that host replicated data. 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 of the network since they would decrease system's performance.

In order to create the desired network topology and enable the emergence of a stable core in the network, we propose the following neighbour selection rule: *two peers may become neighbours if they have similar reliability status.* Additionally, reliable peers have more neighbours, since they have more resources to maintain available network connections.

The topology is intended to be used in a cooperative environment, such as the DBE Knowledge Base. The algorithm does not enforce fairness between peers, i.e., some peers may mainly consume resources while other peers may mainly provide them. However, peers are incentivised to contribute resources through improved performance with increased proximity to the core and to data replicas.

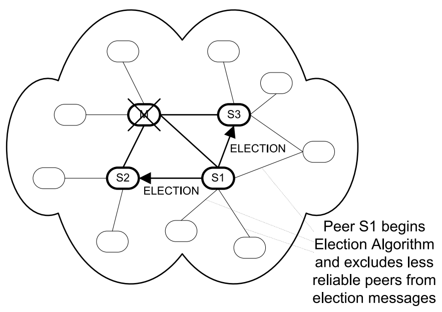
4 Replication Strategy

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

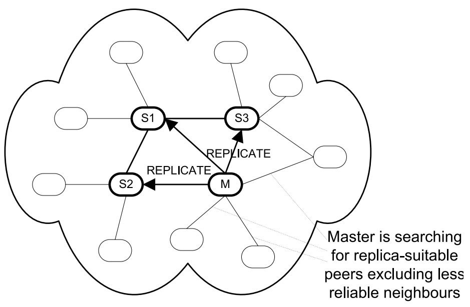
In our approach, we assume each peer can potentially create an independent database, and replicate it over the P2P network in order to improve its availability and persistence guarantees. A peer that creates the first copy of a database, which we call the *master replica*, becomes the database owner. 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's responsible for handling and synchronising updates. Slave replicas are created automatically, on demand.

We restrict the set of peers that are allowed to create slave replicas to those with reliability above replica-suitable *threshold.* Firstly, a peer accepting a slave replica may require from the peer initiating the placement a certain level of reliability, above some threshold, which we call the *replica creation threshold.* Secondly, the master replica may require that the slave replicas are created only by peers located in the replica-suitable core of the network, i.e., peers above the *replica acceptance threshold.* 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 by each peer individually.

There are two general approaches for the replica placement. In the *server-initiated* strategy, a peer that already hosts a replica, in particular the master, requests a new replica placement on one of its neighbours, for example when the


S3
ELECTION
S2
S1
ELECTION
M
Peer S1 begins Election Algorithm and excludes less reliable peers from election messages

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


S1
S3
REPLICATE
REPLICATE
S2
M
Master is searching for replica-suitable peers excluding less reliable neighbours

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

Fig. 2. Slave replica searching and master election algorithms exploiting the implicit information about peer reliability contained in the network topology.

number of user queries exceeds some critical level and a new replica is needed to handle the load. It's crucial in this approach that the peer initiating the replication must be able to find other peers suitable for hosting new replicas. This should be satisfied in the topology proposed in this paper, since peers storing replicas are located in the highly connected core, and share a similar, high reliability status (see Figure 2(b)).

In the *client-initiated* approach, the replication is initiated by a peer that doesn't maintain a replica yet. Such a peer, in most circumstances, forwards user queries to its neighbours. However, if a certain criteria is met, and the peer possesses enough resources, it may decide to store a replica. The simplest criteria for client-initiated replica placement is described by a threshold for the frequency of queries for a database, which is easy to implement since it requires that peers only maintain the statistics of queries. New replicas may be created with a probability *proportional* to the query frequencies, which has the advantage that popular data is stored by many peers and can be accessed easily. Another technique, called *square-root replication*, calculates replica placement probabilities from query search areas, the greater the search area the higher the probability, so that no replicas of the same data are created next to each other. This approach was shown to perform better then proportional or uniform replication, and has the advantage over proportional replication that rare data is less likely to be removed from the system [7].

Replicas are also removed on demand, adaptively. When a peer decides that the cost of a replica maintenance is higher then the profit from 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 with the LRU strategy (Least Recently Used) as in FreeNet [2].

4.1 Replica Synchronisation

Database replicas must be synchronised between the master and the slaves after update operations. We add the constraint that updates are only performed on the master, while queries can be handled by any slave. The system provides eventual consistency of the replicas [14].

An update operation, either a modification, an addition or a removal, must be performed on the master replica of a database. We can make this requirement, without reducing significantly the scalability of the system, since we assume that the database is designed for systems that are more frequently queried than updated, as in the the DBE Knowledge Base. 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. The design can be also improved by allowing the replicas to construct a hierarchy, a spanning tree for spreading the updates. In this case, a slave replica creates new replicas without notifying the master and takes the responsibility of updating them.

4.2 Master Election

In our P2P system, peers have relative positions in the topology, defined by their reliability 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 reliability, i.e., topologically further from the core, when sending election messages (see Figure 2(a)). This heuristic, however, does not guarantee that the most reliable peer will become 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 reliability, but still inside the core, and they can restrict further propagation of the message to only peers with higher stability. Given high enough connectivity between nodes in the core, within a certain probability the node with the highest reliability should win the election.

4.3 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

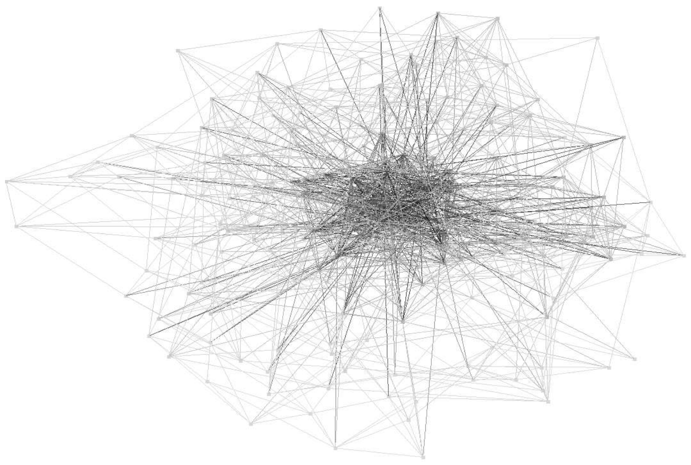


Fig. 3. Visualisation of a 200-node network in our RePast simulation using the Fruchmen-Reingold algorithm. Reliable peers are marked with dark colors, stable core of the network is visible in the center.

techniques have been proposed to improve search in unstructured peer-to-peer networks, such as random walk, iterative deepening or routing indices [15]. For the core-based architecture presented in this paper we are designing a probabilistic adaptive algorithm where routing is based on two main factors: heuristic values learned by the system (e.g., as in CRL/Sample [3] or FreeNet [2]), and using the neighbour reliability heuristic to effectively route queries towards the core of the network.

5 Evaluation

We evaluated our approach by modelling a P2P network in RePast, a multi-agent simulation toolkit for large-scale systems. Although the simulation is implemented in Java, we managed to create a network 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, the 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 the decentralisation and the 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

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 reliability;
    forall peers $p$ in the network do
        p.step();
    end
end
```

Fig. 4. Main loop of the simulation.

neighbours for the most powerful peers. Similarly, the reliability distribution follows the power-law. We model the network growth by adding new peers at each step 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 reliability. Bootstrap 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.

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

We improved the algorithm by allowing the peers to connect also to other non-similar peers, for example with the probability exponentially decreasing with the difference between nodes. It turned out, however, that a randomised strategy

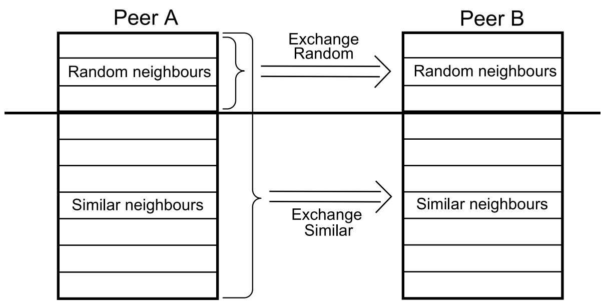


Fig. 5. Neighbourhood set exchange from Peer A to Peer B.

Algorithm 2: Agent step

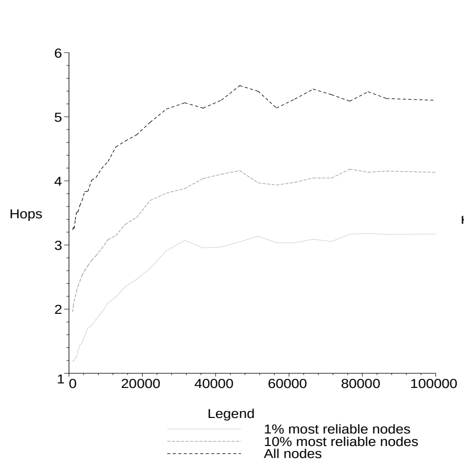
```
if number of neighbours = MAX_NEIGHBOURS then
    disconnect random neighbour;
end
if number of similar neighbours < MAX_SIMILAR then
    choose randomly neighbour $p$ from all known neighbours;
    get all neighbours $n_1..n_k$ from $p$;
    choose peer $n$ with the most similar reliability from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

Fig. 6. Algorithm performed by an agent at one step of the simulation.

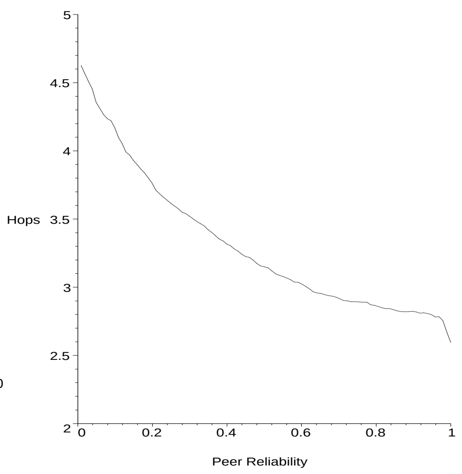
where a percentage of neighbours was always selected at random gave the best results (see Figure 5). Random connections serve several purposes. First of all, they allow the peers to discover potential neighbours with similar reliability, even in remote regions of the network, which in turn enabled the formation of a single cluster containing the most reliable peers. Figure 3 presents a visualisation of a sample network consisting of 200 peers, where we can see that the topology evolved to the desired form where the most reliable peers are clustered in the center of the network and constitute a stable core. 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 [4], 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.

Figure 6 shows the improved, randomised algorithm performed by the peers at each step of the simulation. A peer maintains two independent sets of neighbours, randomly-selected connections, and greedily-selected connections to peers with similar reliability status (see Figure 5). 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.

Figure 7 shows the experimental results obtained from the simulation of our neighbour selection algorithm. We can see that the average path length between peers varies with peer reliability, the average distance between the most reliable



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



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

Fig. 7. Results of the neighbourhood selection algorithm.

peers is lower than between less reliable peers. This confirms our observation that the most reliable peers are highly connected with each other and form a reliable core of the network.

Searching for reliable peers maintaining replicas, or suitable for maintaining replicas, is less expensive in terms of number of messages generated, since it doesn't require communicating with all peers in the network. Due to the information contained in the network structure, the selection of reliable peers suitable for replica placement can be optimised using a heuristic search and it does not require contacting all peers in the system.

6 Related Work

Most existing P2P systems that exploit the heterogeneity of the environment are based on structured P2P overlays. In OceanStore [5], it's 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, however, no specific algorithm for the election of such a tier is presented. Mizrak et. al. [8] proposes a super-peer based approach for the exploitation of the resource heterogeneity, however, unlike our architecture, it relies on a DHT structure.

In the field of unstructured P2P networks, there has been a lot of work devoted to searching (e.g. [15]) and to replication strategies [7], but there has been limited research on network topology generation and peer neighbourhood selection algorithms. Yang and Molina [16] investigate general principles of superpeer-based networks and give practical guidelines for the design of such networks, however, they don't give 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 [4]), however, he doesn't address the specific problem of replica placement in an open P2P system.

7 Conclusions and Future Work

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

The main advantage of the topology described in this paper is that the network structure contains information about the peer reliability, which allows the peers to discover other reliable peers without flooding the entire network with search messages. The topology allows the search space to be limited to a small subset of all peers in the system. This property helps us to solve the problem of dynamic slave replica placement, master replica election, and replica discovery in an open, decentralised environment. The topology should also reduce the cost for 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 the Collaborative Reinforcement Learning (CRL) [3], 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 platform.

References

1. A.-L. Barabási and E. Bonabeau. Scale-free networks. In *Scientific American*, volume 288, pages 60–69, May 2003.
2. I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Proceedings of the International Workshop on Designing Privacy Enhancing Technologies*, pages 46–66. Springer-Verlag, 2000.
3. J. Dowling, E. Curran, R. Cunningham, and V. Cahill. Using feedback in collaborative reinforcement learning to adaptively optimise manet routing. In *IEEE*

Transactions on Systems, Man and Cybernetics (Part A), Special Issue on Engineering Self-Organized Distributed Systems, volume 35, pages 362–372, May 2005.

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