A Comparative Study of Replication Schemes for Structured P2P Networks

Moufida Rahmani, Mahfoud Benchaïba
University of Science and Technology Houari Boumediene
LSI, Computer-science Department
Algiers, Algeria
Emails: {morahmani, mbenchaiba@}usthb.dz

Abstract—**Structured Peer to Peer (P2P) networks provide efficient mechanisms for resource placement and lookup. However, these systems deal with irregular and frequent arrival/departure of nodes. Thus, these systems do not offer any guarantees about data availability. A well-known technique for improving resources availability and providing load balancing is replication. Many replication methods are proposed for structured P2P networks, with a specific main goal to achieve. This paper reviews and compares various existing replication techniques for structured P2P networks, which we classify based on their main objectives.**

Keywords–Structured Peer-to-Peer; Replication; DHT; Availability; Load balancing; Performance; Churn.

I. Introduction

Since 1999, P2P networks have been in continuous development. P2P networks are overlay distributed networks composed of a large number of autonomous nodes, i.e., virtual networks which may be totally unrelated to the physical network that connects the different nodes. These nodes, called peers, share a part of their own resources such as storage capacity, files and processing power. They play the role of both client and server. Communications between peers are direct, without passing intermediary entities.

Napster [1], a popular music exchange system, was the first to emerge as a P2P file sharing application. After that, several file-sharing software have succeeded, we quote: Gnutella [2], KaZaA [3], BitTorrent [4], Oceanstore [5], and PAST [6].

P2P networks can be classified as unstructured and structured, depending on the overlay structures. Unstructured P2P systems do not impose any structure on the overlay network and usually use flooding for searching objects. This method is expensive in terms of bandwidth consumption and is not efficient for locating unpopular files. Structured P2P systems, in the other hand, impose particular structures on the overlay networks (which are commonly referred to as Distributed Hash Tables (DHTs)). Any file can be located in a small number of overlay hops, which significantly reduces the search cost as compared to unstructured systems. Unfortunately, if connection/disconnection frequency is too high, data may be lost. To deal with these problems, the replication can be used as the efficient technique to improve resources availability and to provide load balancing enhance.

Many replication methods are proposed for structured P2P networks, with a specific main goal to achieve. Ktari et al. [7] have presented a comparative analysis of some replication algorithms for DHT architectures. Our paper's prime objectives are to:

- Highlight factors involved in replication, type of replication, and parameters affecting replication.
- Present the some existing replication techniques for structured P2P networks with a new classification. Each replication technique can be implemented for several objectives such as: improving availability, enhancing system performance, achieving load balancing. In this paper, we try to classify replication strategies existing in the literature based on their main objectives. Our classification lets to well study and compare them.

This paper is organized as follows: In Section II, structured P2P networks and examples of networks are presented. In Section III, we highlight some replication basic notions as well as factors and parameters involved in it. Section IV reviews, classifies and compares existing replication strategies. Finally, we conclude in Section V.

II. Backgrounds: structured P2P overlay networks

In structured P2P overlay networks, the topology is tightly controlled and the data is placed at specific location which makes queries more efficient [8]. Structured P2P systems use DHT as a substrate, in which the location information of object (value) is placed deterministically, at the peers with identifiers corresponding to the data objects unique key. In DHT, a peer's identifier *ID* is chosen by hashing its IP address and objects's *key* is chosen by hashing its name for example. Both peers and objects are identified in the same namespace. Each node is responsible for some of the *keys* in the system and each data object is stored on this node if the identifier of the object belongs to the range which node is responsible. The main operations used in DHT are: *put(key, value)* and *lookup(key)*.

- *put(key, value)*: This operation is used when the peer wants to publish an object in the system. Peer computes the *key* of the object and then sends a message put(key, value) to the peer responsible for this *key*.
- *lookup(key)*: This operation returns the value associated with the *key*, if any.

Several systems employing DHTs have been developed; among the most well-known Pastry [9], Tapestry [10], CAN [11], Kademlia [12] and Chord [13]. We present Chord as an example of this category. Additionally, a drawing is added to clarify the functioning of chord.

- ***Chord:*** Chord is based on a ring topology, a Chord peer has knowledge of its predecessor and its successor. A hash function (SHA-1) generates a regular identifier, an m-bit for each peer from its IP address. Then, each peer is placed in the ring so as to arrange the identifiers in ascending order. The successor (respectively predecessor) of a peer n is the peer whose identifier is immediately higher (respectively lower) to identifier of peer n. Thus, each peer n with identifier ID is responsible for the interval of *keys*] predecessor (n), n]. For a given peer, mere knowledge of its predecessor and its successor is not sufficient to ensure good performance of the ring, particularly in terms of number of hops per request. To overcome this problem, for a *key* space in the range [0, 2m [, each peer ID connects to other neighboring nodes, called fingers, with ID successor(ID+ 2^i-1) with $1 \leq i \leq m$. These fingers constitute its routing table. Thus, the number of fingers per node is *O*(log N). Thus, the maximum number of peers traveled to forward a query is expressed in terms *O*(log (N)), where N is the number of peers in the system.

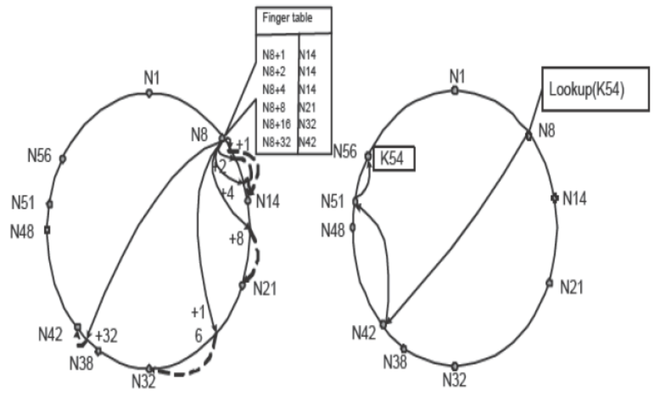

Figure 1. Chord ring with identifier circle consisting of ten peers and five data keys. It shows the path followed by a query originated from peer 8 for the lookup of key 54 [13].

When a node wants to find data object (value) for a key *lookup(key)*, it uses the following algorithm: it seeks among its fingers the peer whose identifier is the greatest and is lower than the key, and sends it the message. The node that receives the message, then in turn executes this same algorithm (see Figure 1) until the target node is found. Nodes are allowed to join and leave the system causing churn. A Chord regularly runs maintenance algorithms that detects failures and repairs routing tables, allowing requests for a *key* to be routed correctly to their owner despite node churn.

III. Replication

The main idea of replication data is to maintain several copies or replicas of the same data at various different sites. Data replication is recognized as an effective way to increase the availability and performance of distributed systems such as P2P. Replication is a solution that offers several advantages:

- **Increases availability:** Replication removes single points of failure (data is accessible from multiple nodes), thus increasing availability and fault tolerance.
- **Improves performance:** Replication improves performance of system in terms of response time. Data can be located closer to their access points. Therefore, the success rate will increase, the response time and the overhead will decrease. The response time can be constant for all the users if all the data are replicated uniformly over the network.
- **Achieves load balancing:** Replication can provide load balancing between the nodes, such as multiple nodes can serve the same object simultaneously. Therefore, it reduces load on the nodes that own the original data.

As replication has advantages, it also has significant costs such as the storage cost and consumption of bandwidth. To control the cost of replication, there are important factors that replication must take into consideration. These factors have a direct impact on the performance of the system. In order to avoid wastage of network and peer resources, the excessive use of replication is not recommended.

A. Factors of replication

When designing a replication strategy, the following important aspects should be taken into consideration:

1) ***What should be replicated****:* Most strategies of replication choose the files to replicate based on their popularity values. A general way of measuring the file popularity on a peer is by counting the number of requests. Other strategies prefer, for their part, to replicate all shared files or only replicate the rare objects.

After choosing the file to replicate, there are important point to decide: is it necessary to replicate the file or its index (since the index size is smaller that the file's)? Or is it preferable to adopt erasure code to replicate file? This point is related to the type of replication, which will be detailed in Section "III.B".

2) ***Where should replica be placed****:* The second important factor is selection of the best node to host a particular replica. Replicated copies should be placed in proximity to peers who are likely to request the resource in order to reduce delays of search and downloading. Moreover, some peers' characteristics, such as available storage space and availability, should be taken into consideration. Each replication strategy which aims to improve file availability must consider the peer availability: If a new replica is hosted by a peer that has low availability and may leave soon, we may need another replica in order to maintain the required availability for the file.

3) ***When should be replicate****:* Replication can occur periodically or at event. For example, suppose that the peer considers file's popularity. When it receives a request for a file, the peer increases file's popularity value and checks whether it exceeds some threshold. If so, it might decide to replicate the file. Some strategies ignore this factor.

*4) **How much is the number of replicas per file**:* Each replication strategy determines the number of replicas per file according to its objectives and the system parameters which it takes into account. For example, if the aim of the strategy is to maintain a threshold level of availability, it needs to consider the system's parameters that affect availability and performance such as online availability of the peers in the system.

*5) **Replica replacement strategy**:* It is essential to deploy replica replacement strategy because storage space is limited. A replacement strategy consists on removing some less efficient replicas to create space for new replicas. Least Recently Used (LRU) is the most widely used in P2P networks.

B. Type of replication

In the context of P2P networks, the replication is redundancy by creating copies of data, called replicas, which can be stored in peers other than the source peer (which holds original data). Replica can be a complete file, the index of the file or bloc of the file.

*1) **Traditional replication**:* is called also data replication. In this type, replica is a complete (entire) file.

*2) **Erasure code replication**:* An erasure code provides redundancy as replication for achieving high availability and reliability in storage and communication systems without imposing high bandwidth and storage overhead [14]. In erasure code, a file is divided into b (equal size) blocks and recoded into c blocks, where $c > b$ (of same size as before). The erasure-coded blocks are dependent each other. The key property of erasure code replication is that any b out of c blocks is enough to reassemble the original file. We call c/b the storage overhead S which is sometimes stated as the stretch factor.

Lin et al. [15] provided a comparison between traditional replication and erasure code replication. They came to an important result: in erasure code replication with a storage overhead of S, if a file is divided into b blocks, then each file block is replicated S times. Therefore we have $S*b$ number of blocks in the system. They also pointed out that when $b=1$, erasure code replication is equivalent to traditional replication.

*3) **Index replication**:* In some cases, it is best to replicate the index of a file that is the pointer to the peer that holds the file. This solution is recommended when the file size is very large to avoid the problem of the data file coherence when the original file is updated. The index replication consumes little storage space and bandwidth.

*4) **Message replication**:* The idea of the message replication, introduced by Hassan and Ramaswamy [16], is to replicate a message several times within the network to enable a large query coverage in the network.

When we design a replication strategy for unstructured P2P, we can use one of the four replication types. We can also use two types in the same strategy for example traditional and index replication as in [17]. In case of structured P2P network, we can find all types of replication except message replication. Additional, in few cases, routing tables or information about neighbors are also replicated.

C. The parameters that affect replication

The parameters that affect the efficiency of the replication, and must be taken into consideration are mainly: the file popularity, the peer availability, rare objects, storage space and data consistency.

*1) **The file popularity**:* Jacky et al. [18] present a study of popularity measurements of P2P file systems in Gnutella and Napster. They came to an important result: caching or replicating the most popular files on the system is strongly suggested in order to greatly improve system performance. Some strategies presented in this article are based on popularity in order to determine the candidate files for replication. The file popularity can be calculated by keeping track of the number of requests. This value is local and can change rapidly and therefore increase the replication cost. To avoid this, the system should calculate and predict the overall popularity values (for all the network). Manel and Mahfoud [19] define a way to calculate a global file popularity based on local estimation of the peer and estimations done by the other peers participating in the network. The simulation results show that their measurement is closer to the real one.

*2) **The peer availability**:* In P2P networks, peers are volatile: they join and leave the network unexpectedly. The main consequence is that the files (original copy or replica) that a peer stores might become unavailable. Accordingly, the peer availability affects the file availability. Thus, the replication strategy which aims to improve availability of resources must take into account this parameter to calculate the number of the file replicas. In [20], a study was made to understand the peer availability.

*3) **Rare objects**:* Studies have shown for Gnutella that 18% of all queries return no responses even when results are available [21]. That is due to search algorithms used in unstructured P2P, the most typical query method is flooding. This method is effective for locating highly replicated data and is not suited for locating rare data (those with few replicas). Two solutions are proposed in the literature to solve this problem: hybrid search and replication. Hybrid search combines two search methods, it uses flooding techniques for locating popular items and structured (DHT) search techniques for publishing and locating rare items. Replication is a well-known technique for improving resource availability. To the best of our knowledge, there are few works that propose a replication technique in order to improve search for rare objects ([22][23][24]). A key challenge for the hybrid search and replication is how to identify rare items.

In our opinion, this last point is not satisfactorily addressed in the case of replication, because some strategy as presented by Ma et al. [24] are based only on the sampling technologies and number of copies to determine if object is rare or not. Therefore, before proposing a replication technique aiming to improve search for rare objects, one must first define what is a rare object, how one can identify it and at the end one discuss the factors involved in replication. This parameter is only for unstructured P2P network because in the case of structured P2P network, any file can be located even for rare data if any.

*4) **Storage space**:* Each peer in a P2P network shares a part of its storage capacity which will be used to store its shared files and the replicas of files of others peers. Unfortunately, this storage space is limited and can store only a limited number of replicas. Therefore, replication strategy must consider this parameter when it chooses the best node to host a replica and it must deploy replica replacement strategy. However, most replication strategies do not consider storage capacity constraints thus the network and peer resources are abused.

5) ***Data consistency****:* Generally, we can not use data replication without evoking data consistency issue. It turns that in P2P systems (such as PAST, Gnutella) [25], the replicated resources often they are not subject to modification (static or read-only), this explains in part why replication techniques do not consider the consistency aspect. However, in case of resources update, it is recommended to trait the two aspects simultaneously. This can decrease the overhead of consistency maintenance and ensure that a file requester receive up-to-date files.

IV. Replication techniques for structured P2P networks

A. Replication in DHTs

The replication strategies, for the structured P2P networks that employ DHTs, use two algorithms: Data replication algorithm (or called Replica placement algorithm) and maintenance algorithm. The choice of these algorithms can have significant impact upon performance and reliability.

1) Data replication algorithm: In this kind of algorithm the peers decide what should be replicated, how many replicas should be created and where to replicate them in order to realize a well-determined objective such load balancing or improve availability of resources. There are three main basic replica placement strategies, which are the basis of the most replication strategies present in the section.

- Neighbor replication: Called also the simple replication method, each peer maintains a list of neighbors such as successor-lists and predecessor-lists in Chord [13] or leaf-sets in pastry [9]. In neighbor replication, the data objects are stored not only in root peer but also on its successor, or on its predecessor, or on its leaf-sets and or on the nodes belonging to the same group as it. The root is node that stores the object location information and it can be different to the owner which is the node that stores the master copy of the object. Chord employs successors-lists replication. Pastry and Kademlia DHTs employ leaf-sets replication.
- Path replication: Path replication replicates a data object along the search path that is traversed by the lookup message, from the requester to the provider node (root peer). Tapestry [10] employs path replication scheme.
- Multi Publication Key Replication : A key is mapped into *r* points in the coordinate space and accordingly replicated at *r* distinct nodes in the system. CAN implements this solution by using Multiple hash functions.

2) Maintenance protocols: For each data replication algorithm, there is a special maintenance protocol. The idea is that the maintenance protocols must maintain k copies of each data objects without violating the initial placement strategy. It means that the k copies of each data object have to be stored on the root-peer neighbors in the case of the neighbors replication scheme, on the root peers in the Multi Publication Key Replication scheme and on all the peers that exist in the search path in the case of path replication.

B. Classification of replication techniques for structured P2P networks

A replication technique can be performed for several objectives such as: improving availability, enhancing system's performance, achieving load balancing. To fulfil these objectives, a number of replication strategies have been proposed for structured P2P networks. Each strategy is proposed for a specific DHT, and employs different algorithms for placement and maintenance. In this section, various existing replication schemes are presented and classified into four categories according to replication objectives as described above. Each category defines a main replication objective to achieve (Figure 2). However, some techniques, which belong to one category, may possess secondary properties of another category.

Category1: *"Achieve load balancing"* Godfrey et al. [26] say that the load unbalancing problem in structured P2P network may result due to non-uniform distribution of objects in the identifiers space. Therefore, some nodes having $O(logN)$ times as many objects as the average node, where N is the number of peers in system. Additional, if a single node stores a popular file, then all requests for this file are directed to this node and this will make it and the path leading to it overloaded. Category 1 includes proposed strategies that solve load balancing issue by efficient data replication.

Category2: *"Increase availability of resources"* This category shows the replication strategies that seek to increase the availability of resources caused by irregular departure of peers.

Category3: *"Enhance churn tolerance"* When new peer joins the system, if its identifier is closer to an object's key than the identifier of its current root, the data object needs to be migrated on the new peer and the new peer will become the root for this object. The migration process can be also appeared in the case when the peer quits the network or when neighbors list is changed. If a high churn rate arrived, then maintenance algorithms must often be applied in order to adapt to the new structure by migrating data objects, which generates more traffic and consumes much bandwidth. In order to avoid this issue, the replication strategy must be more tolerate under higher churn rates.

Category4: *"Improve search performance"* It includes replication strategies that decrease the response time and the overhead, thus the search performance is improved.

1) Achieve load balancing:

a) Lightweight Adaptive system-neutral Replication protocol LAR [27]: LAR can efficiently deliver a good load balance and low query latencies even when demand is heavily skewed. LAR functions as follows:

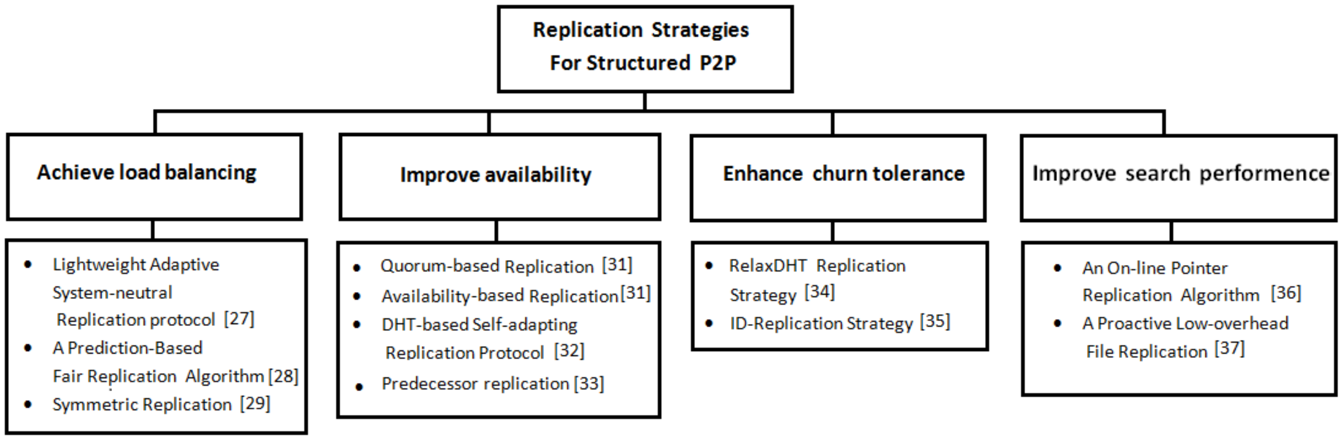
Replication Strategies For Structured P2P
Achieve load balancing
Improve availability
Enhance churn tolerance
Improve search performence
Lightweight Adaptive System-neutral Replication protocol [27]
A Prediction-Based Fair Replication Algorithm [28]
Symmetric Replication [29]
Quorum-based Replication [31]
Availability-based Replication [31]
DHT-based Self-adapting Replication Protocol [32]
Predecessor replication [33]
RelaxDHT Replication Strategy [34]
ID-Replication Strategy [35]
An On-line Pointer Replication Algorithm [36]
A Proactive Low-overhead File Replication [37]

Figure 2. Classification of replication techniques for structured P2P networks.

Each time the peer *Pi* receives a request for data item from the peer *Pj*. It checks if its current load *Li* exceeds a certain threshold. Load is redistributed according to a per-node capacity *Lmax*, high-load *Lhi* and low-load *Llo* thresholds. The node capacity is the number of queries that it can route or handle per second, there are two different cases:

- if ($Li > Lhi$), it indicates that load redistribution is necessary. In this case, *Pi* attempts to create new replicas on Pj, if *Li* is greater than *Lj* by some fixed value K. Pi then asks *Pj* to create replicas of the n most highly loaded items in *Pj*, such that the sum of the local loads due to these n items is greater than or equal to K.
- if ($Llo < Li < Lhi$) then load is redistributed as above on *Pj* only if $(Li - Lj \geq Llo)$.

If *Pj* has no space storage for hosting the new replica, it then uses LRU algorithm, to choose a victim replica to leave the space for the new replica.

The locations of the new replicas is then spread, by piggy-backing them, on subsequent messages that contain requests for the same items. The pointers to the replicas are cached in the peers along the path traversed by requests for these replicas. In the same way as for replicas, LRU policy is used: the least recently used pointer is deleted.

The presence of pointers of the new replicas can then respond more quickly to the subsequent requests and potentially reduce the number of hops needed for routing. Indeed, if a peer reaches a node with a pointer to a replica rather than the node with the resource originally, it follows the pointer.

When a node that has a replica leaves the network, or simply when LRU algorithm is applied to replace cache or replicas, LAR does not apply maintenance protocol. Therefore, the initial placement strategy is violated and it can find pointers to inexistent content.

When adapting LAR to Chord, the finger list is the default item of replication. LAR replicates the data item only if the load on the server due to the data item is more than that due to the finger list.

b) A Prediction-Based Fair Replication Algorithm [28]: It aims to maintain an excellent system performance when the query is highly skewed. Through the use of a simple prediction method, it can foresee traffic surge and replicate beforehand. The basic idea of the PFR algorithm is to create replicas for the node whose predicted load or current load reaches certain predefined threshold and adaptively adjusts the Replication Speed (RS) for each replication process. RS can be measured by the ratio of the number of nodes chosen to hold replicas to the number of all nodes that have encountered along the query path. The light-loaded nodes are always chosen to hold replicas. For example, if RS equals to 3N/4, this means that load should be redistributed to 3/4 amount of the total number of nodes along the query path, where N is number of node in the query path. PFR function as follows:

When query packet is routed through a node, it computes and piggybacks its predicted load on the query packet. At the same time, it checks the value of the predicted load *Preload* and current load to determine whether the load rebalancing is necessary or not. With respect to nodes' predicted load fraction, PFR defines 5 different replication levels, which specify the RS for each query. There are two types of nodes along a query path: query's destination node and the nodes just forwarding queries. For the first type of nodes, node replicates according to these 5 levels. However, for the second type of nodes, replication is necessary only when its *Preload* has reached the level one of threshold. The first level threshold indicates that a node is approaching its capacity and it is in an emergency state to shed load in order to prevent itself from overloading and consequent dropping queries. The node capacity is the number of queries that it can route or handle per second. If the replication is necessary based on Preload, the node creates replicas of the n heaviest-loaded items on each selected node, such that the sum of the local loads caused by these n items will be greater than or equal to the difference in loads between the two nodes. Else, the node checks whether the replication is necessary or not based on the node's current load fraction. If yes, the node replicates in the same way, but the corresponding replication level should be decreased by 1. It means that value of RS when the replication is based on current load is lower compared to the value of RS when the replication is based on Preload. Whenever stored replicas reach the nodes maximum storage size, the new replicas will replace the old replicas using the LRU algorithm.

PFR algorithm also uses the replica location dissemination method such as LAR. It helps replicas to be efficiently utilized in shedding load. But, it does not apply maintenance protocol. Therefore, we can find pointers to inexistent content.

When PFR is applied to Chord, the finger list is the default item of replication. PFR replicates the data item only if the load on the node caused by the actual data item is more than that caused by the finger list.

c) Symmetric Replication [29]: Symmetric replication can be used for load-balancing between the peers, end-to-end fault tolerance and to increase the security. The advantage of symmetric replication is that it can be applied to all structured P2P systems. It is closer to the methods that use several hashing functions for replication. The main idea behind symmetric replication is that each identifer in the system is associated with a set of f distinct identifers such that the following always holds: if the identifer i is associated with the set of identifers *r1, ..., rf*, then the identifer *rx*, for $1 \leq x \leq f$, is associated with the identifers *r1, ..., rf* as well. The identifier space is partitioned into *N/f* equivalence classes such that identifiers in an equivalence class are all associated with each other, where N is the size of the identifier space and f is replication degree. The replication degree is number of replicas made. The identifier i is associated to the f different identifiers given by the function H; $H(i,x)=i+(x - 1)\ N/f$.

In symmetric replication there is no root node, a data item with identifier i is stored on the f peers given by *Sp(H(x, i))*, for all x $(1 \leq x \leq f)$. *Sp* is pseudo-metric space as hash function (SHA-1) in Chord. Thus, the responsible peer of identifer i stores every data item with an identifer associated with i. This implies that to find a data item with identifier i, a request can be made for any of the identifiers associated with i. Each node storing a data can easily calculate the keys of the different replicas of this particular data item, it is sufficient that each node knows about the replication scheme. Therefore, it can achieve load-balancing between replicas by sending requests to a random replica.

Symmetric replication has put a set of replication algorithms, these algorithms are used for joins and leaves, inserting and looking up items and failures. It can enhance performance by sending multiple concurrent requests and picking the first response that arrives. Unlike the other replication methods that use multiple hash functions, successor-lists, or leaf-sets to select replica nodes, symmetric replication needs only *O(1)* messages for every join and leave operation to maintain any replication degree. The replica node is a node that stores a replica.

Every replica node cooperates to execute maintenance algorithm. The node storing the replica i checks if the replica $i+1$ is stored in the corresponding node. If not, it inserts a new replica in that node. The node storing the replica $i+1$ do the same with the replica $i+2$ an so on.

Discussion: The replication strategies presented in this category use different replica placement algorithm to achieve load balancing. The first technique, LAR, uses any type of replica placement algorithms described in Section IV.A.1. It resembles at owner replication, because only the requester node keeps the copy, the others nodes on the query path contain only the cache entries, pointers towards replicas. In the owner replication [30], if a search for an object is successful, this object is replicated on the requester node. The replicas can be efficiently utilized due to the use of the replica location dissemination method.

PFR is an improvement of LAR, because it uses the same principle. The both use the load value to determine whether load redistribution is necessary. After the replication, both use the dissemination method to spread the information about new replicas locations. In the case of LAR, RS value is always equal 1. Moreover, there are some difference between the two: First, in PFR not only the requester node keeps the copy, but also some node in the query path according to the RS value. Thus, PFR is a variety of replication path. Second, all the nodes in the query path can check the load value to determine whether load rebalancing is necessary or not. Unlike LAR, only the peer that receives a request checks. Third, LAR uses only the current load to check, but PFR uses two load values predict and current. Therefore, PFR adaptively adjusts the number of replicas created for each query process and can be scattered before flash crowd happens. Symmetric replication based on multi publication key replication is completely different compared to LAR and PFR, in different points:

First, symmetric replication replicates only data item, but in PFR and LRA, data item is not the default item of replication. Second, in symmetric replication, the replication degree is the same for all the replicas, but in PFR and LRA the replication degree is determined according to the load value. Third, only the symmetric replication applies maintenance algorithm. The maintenance protocol is applied in distributed manner and it needs cooperation of all replica nodes. Therefore, it is complex compared to the other techniques where the maintenance is provided by the root node. Fourth, LAR and PFR employ the replica location dissemination method in order to the replicas can be used. In symmetric replication, it is sufficient that each node knows about the replication scheme. Therefore, it can easily calculate the keys of the different replicas.

2) Increase availability:

a) Quorum-based replication and Availability-based replication [31]: Kim and Park have proposed efficient replication methods that can reduce network traffic enormously and achieve high data availability in DHT based on P2P storage system. In these methods, the replicas are loosely coupled to the consistent set such as the leaf-set and the successor-lists. Additionally, they are interleaved on the consistent set to reduce the compulsory copies which occur under churn, unlike the simple replication (neighbor replication) method that directly uses the consistent set. This set is tightly coupled to the current state of nodes and the traffic needed to support this replication can be high and bursty under churns.

Tow types of replication methods are proposed: *Quorum-based replication* and *Availability-based replication*.

The *Quorum-based replication* modifies the simple replication to prevent the compulsory copies which occur under churn. When a new node joins, it gets not only routing information such as DHT and consistent set, but also the replication set. The replication set indicates which node replicates the object among the consistent set. The size of the consistent set is bigger than the number of replicas, the replication does not occur frequently and the replication set can interleave the replicas on the consistent set. This behavior increases the chance to reduce the compulsory copies.

The replication only occurs when the number of replicas is fewer than the target quorum. The quorum is the fixed minimum number replicas necessary to archive an objective. In this case, the number of replicas must be more than the target quorum to achieve target data availability. When a node leaves and it is not a member of the replication set of peers, there is no need to replicate the data. Otherwise, if it is a member, the selection of node among non-members of the replication set of peers as a new replica (target node) is necessary. The target node became among the replication set for this peer. The *Quorum-based replication* considers that each node has the same availability. However, if a new replica is assigned by the node which has low availability, this node may leave soon and we need another new replica. Therefore, *Availability-based replication* takes into consideration the node availability and guarantees the high data availability by selecting the more available nodes as replicas. To do this, each node manages its availability and advertises it to all members of the consistent set by piggybacking it to the periodic ping message which has been already used to detect node failures on the consistent set. In this approach, the replication only occurs when the data availability is below the target availability. When a node needs a new replica, the most available node among non-members of the replication set is selected as a new replica.

If all members of a consistent set have averagely low availability, *Availability-based replication* needs more replicas than the quorum based replication. Sometimes this behavior takes more bandwidth, but when nodes leave, this subtle replication can reduce much more bandwidth than the *Quorum-based replication*.

When a node fails and its neighbor gets the lookup request, this neighbor may not have the replicas for the requested object. In this case, the neighbors must forward the request to the replicas of the failed node for the routing correctness. To do this, each node should have the replication sets of all members of its consistent set by piggybacking this information to the periodic ping message for its consistent set.

The both *Quorum-based replication* and *Availability-based replication* used the same maintenance algorithm : When a new node joins and a target node gets this join request, the object ranges of the target node is divided into two object range and the new node is responsible for one of them. In this case, the new node simply copies the replication set and adds the target node as a new replica because it already has the object for this range. When a node leaves or fails, its neighbor node is responsible for its object range. In this case, both replication sets of the failed node and its neighbor node are merged.

b) DHT-based Self-adapting Replication Protocol [32]: Knezevic et al. have presented a fully decentralized replication protocol suited for any DHT networks, that adjusts autonomously the number of replicas to deliver a configured data availability guarantee.

The main idea of this replication technique is to associate for a given object many keys. The node that publishes the object simply calculates different keys and then inserts the object in the corresponding nodes. To calculate these keys, it uses correlated hashing. The first replica key is generated using a random number generator. All other replica keys are correlated with the first one, i.e., they are derived from it by using the following rule: *replicaKey(1)=c*, *replicaKey(ron)= H(replicaKey(1) + ron)* if $ron \geq 2$, where *ron* is a replica ordinary number, *c* is a random byte array, *H* is a hash function with a low collision probability. *replicaKey* is observed as byte arrays, and + is an array concatenation function. To calculate the key of the ron^{th} replica, the peer requires access to the first replica key and the replica ordinary number (*ron*). All this information is wrapped in an instance of Entry class. Additional, all the nodes use same H.

Every peer calculates the number of replicas *R* from measured average peer online probability and the requested data availability. During joining phase, a peer can get an initial value for *R* from others peers, or can get assume an initial value of *R* for it.

To meet the requested data availability in a DHT, the number of replicas of stored data at each peer has to be adjusted. Therefore, every peer measures the current average peer online probability, knowing the requested data availability and it calculates the new value for the number of replicas *R*. By knowing the previous value R1, a peer removes the replicas with ordinary number *ron* greater than *R* from its local storage replicas. If $(R < R1)$ higher number of replicas are needed, a peer creates new replicas of the data in its local storage under the keys *replicaKey(j), j = R1 + 1, ..., R*.

c) Predecessor replication [33]: Predecessor replication is a simple and an efficient data replication approach. It ensures high data availability it can decrease the number of hops needed to locate the requested data. The node replicated each key whose it is responsible on its predecessor nodes in the same number of copies. According to the query routing mechanism used Chord, the lookup query can be routed to replica node (predecessor node) before reaching the root node. Therefore, the search path is minimized.

Two update strategies are used for the maintenance under churn: the basic update and the periodic update. In the basic update, when the node leaves the network, the data whose it is responsible will be migrate to its predecessor. The periodic update is periodically used to maintain the replication degree. Each node contacts all its replica nodes to ensure that they correctly maintain the appropriate replicas. Each replica node contacts the root nodes of all replicated keys which stores in order to keep the replicas up-to-date.

Discussion: Although the methods presented above are proposed to achieve high data availability in a DHT based P2P systems, there's a difference between them, we quote: The first and the third methods are based on neighbor replication and the second method is based on multi publication key replication.

In order that nodes can access to the replicas in case of the root node is failure, the information of the replication sets are spread in the first methods, by piggybacking this information to the periodic ping message. In case of the second method, it is not needed to know the information about the replica locations. When a peer wants to get a value, it is sufficient to return any available replica. The basic DHT lookup operation is applied until the data is retrieved. Like the second method, the third method does not need to know the information about the replica locations, the nodes can access to the replica before reaching the root node.

In *Quorum-based replication* and predecessor replication, the replication degree is fixed constant and the is same for each data. They do not take into account some characteristics such as the requested data availability and the node availability. Therefore, the storage space can be abused by the data which is not popular. Additionally, the bandwidth consumption is increased by the migrating data and replication process to maintain the replication degree, because the replicas can be stored by the nodes which have low availability and can leave the network soon.

Availability-based replication and DHT-based Self-adapting Replication Protocol adjusts the number of replicas according to the node availability and the requested data availability. Thus, if the number of replicas is not sufficient to ensure the requested data availability, new replicas will be created. Additional in the second method, if there are more replicas than needed, peers will remove some of them. Therefore, the second method generates less storage costs.

3) Enhance churn tolerance:

a) RelaxDHT replication strategy [34]: Legtchenko et al. have proposed RelaxDHT replication strategy that enhances churn tolerance by building an efficient Replica placement and maintenance mechanisms. The RelaxDHT strategy is based on neighbor replication, exactly as the leaf-sets replication applied in DHT pastry.

When the root peer receives *put message* for new data block. In the case of the leaf-sets replication with the replication degree equals to R, the root peer stores a copy of the data block for which it is the root. After, it sends the R-1 copies of this data block for its replica-sets. The replicas sets of a root peer are a subset of its leaf-sets.

In case of RelaxDHT replication strategy, the root peer does not necessarily store a copy of the data blocks for which it is the root. It maintains metadata describing the localization of replica sets, the goal of using localization metadata allows to be anywhere in the leaf-sets. Therefore, when a new peer joins a leaf-sets, the application maintenance protocol is not necessary. The replica sets are selected randomly among the R peers around the center of the leaf-sets. This choice will reduce the probability that a chosen peer quickly quits the leaf-sets due to the arrival of new peers.

The root peer sends *Store message* for its replica sets peers that contains in addition to the data block itself such as the Leaf-sets replication, the identity of the peers in the replica set and the identity of the root. A peer may be root for several data blocks and a part of the replica set of other data blocks. In the first case, the peer must store a list of data block identifiers with their associated replica-set-peer list for blocks for which it is the root. In the second case, peer stores a list of data blocks for which it is a part of the replica set, additional it stores the identifier of this data block, the associated replica set peer-list and the identity of the root peer.

Periodically, each peer executes two maintenance protocols. In the first protocol, it checks for each data block that it stores if the root peer for this data block has changed. If so, the peer sends massage for the future root peer. When the new root of this data blocks receives the message, it adds the data block identifier and the corresponding replica set in the list. In the second, it checks for each block for which it is the root, if all the replicas are placed in its leaf-sets around the center. For a data block, if one of its replicas has changed, then the root peer chooses randomly a new peer in the center of the leaf-sets and changes the replica set. After, it sends a *Store massage* with the replicas set for each peer in its replica set. When the peer receives this message and already stores a copy of the corresponding data block, it updates the corresponding replica set if necessary. In other case, if the peer does not store the associated data block because it is a new peer in the replica set, it fetches everything necessary to store (data block) from one of the peers mentioned in the received replicas set.

b) ID-Replication strategy [35]: ID-Replication can be used in any structured overlay network, however Shafaat et al. have adapted it to Chord for the sake of simplicity. ID-Replication strategy is less sensitive to churn compared to successor-list replication. In order to achieve this goal, ID-Replication uses sets of nodes, called groups, instead of individual nodes. Thus, each group like a node in Chord has unique identifier and is responsible for some of the keys.

Within each group, the nodes that compose it possess two identifiers. A global identifier that is the same identifier as the group and a local identifier that is unique for each node. Each group has successor list, predecessor and fingers as node in Chord.

In successor-list replication, with the replication degree equal to *R*, the root peer stores a copy of the data block for which it is the responsible, the *R-1* copies of this data objects are replicated in its successor-list. In ID-Replication, there are not the root peer, and all the nodes within group store a copy of data blocks which the group is the responsible for. Therefore, a request can be routed to a random node in the group thereby load balancing between the replica nodes. Moreover, in successor-list replication, a request is first routed to the root peer. ID-Replication can send out multiple concurrent requests and picking the first response that arrives.

The replication degree is not fixed, it is between two parameters *Rmin* and *Rmax*. Thus, the number of the nodes within group is specified by these parameters. To allow more copies for popular data objects than other data objects, *Rmin* and *Rmax* must have higher values.

Periodically, each node *p* checks if the size of its group is smaller then *Rmin* because a node is failed, then *p* searches for a standby node by gossiping or contraction a directory and tries to include it in this *p*'s group. If a standby node cannot be found, *p* triggers a merge *p*'s group members with others group such that the size of merged group must be less than *Rmax*. If the size of a group is more than *Rmin*, then standby nodes are (size of group - *Rmin*) nodes. If size of a group is larger than *Rmax*, p initiates the split operation by dividing the group into two groups.

Discussion: The two strategies mentioned above are proposed to be less sensitive under the churn compared to the leaf-sets replication and successor-list replication respectively. There are some differences between them, we quote: In the first, the root peer chooses randomly the replicas sets peers in the center of the leaf-sets then, it is necessary to maintain metadata describing the localization of its replicas set. Additional, each peer maintains information about replica set peer-list which it is part.

Unlike the first, in the second there is no root node and replicas set. Soon as a peer joins a group, it stores a copy of data blocks that the group is responsible for. Like the first, each peer can have information about the others replicas node. In order to realize this, the nodes in the group use gossiping between them.

The ID-Replication gives different replication degree, thus allowing popular data to have more copies, but in RelaxDHT replication strategy it is not mentioned.

The maintenance protocol in RelaxDHT strategy is more complicated then ID-Replication strategy, but the maintenance cost in the both is still moderate compared to the leaf-sets replication and successor-list replication. The maintenance cost is in term of generated overhead and bandwidth consummated by maintenance protocol.

4) Improve search performance:

a) Proactive Low-Overhead File Replication Scheme Plover [36]: Proactive Low-Overhead File Replication Scheme achieves high efficiency in file replication and supports low-cost and consistency maintenance, because it replicates files among physically close nodes based on node available capacities. Plover also includes an efficient file query redirection algorithm for load balancing between replica nodes.

In order to achieve efficient file replication, Plover uses clustering, the physically close nodes are grouped in clusters. Each cluster has a supernode, which is node with high capacity and fast connections. The others nodes are called regular nodes, which are nodes with lower capacity and slower connections.

Periodically, each lightly loaded node reports its information of available capacity. A node's capacity is presented by the number of bits it can transfer in responding file queries per second, and each heavily loaded node reports the information of its popular files to its super node. The lightly loaded node is the node whose actual load is no larger than its capacity, otherwise a it is heavily loaded node. The load caused by file access is determined by the file size and visit rate (popularity), which visit rate or popularity is measured by the number of visits during time unit (second). The super node collects all this information and arranges the file replication between them (among the clusters). The supernode notifies overloaded nodes to replicate popular files to lightly loaded nodes.

Plover addresses the problem of file consistency maintenance and tries to facilitate efficient file consistency maintenance with low-cost. When the node updates a file, it sends update message to its supernode.The super node forwards the message to the replica nodes following a predefined method instead to broadcast it.

When an overloaded node receives a file request, it should forward the request to one of the file's replica nodes. In order to load balancing between replica nodes, the overloaded node chooses the replica node according to Lottery scheduling method adopted by Plover.

b) An On-line Pointer Replication (OPR) algorithm [37]: It can efficiently reduce the query search latency. In order to realize this, OPR replicates the pointers of an object in multiple peers in the network. Therefore, the query for an object is forwarded to the nearest root among the others to fetch the location pointer, that reduces the query search latency as well as improves data availability. Latency incurred by a query is denoted by the number of hops it takes to route to the root.

OPR addresses two problems: Placement of Replicas and Extent of Replication. In Placement of Replicas, it decides how to place the replication pointers in the network to achieve the best performance. To find the best replica placement, OPR uses an heuristic approach called Greedy approach to select the roots. It bases on topologies hypercube to construct overlay topology of the network, because Greedy approach needs complete knowledge about the network layout. In a static network, hypercube can be easily embedded by connecting any two nodes that are one Hamming distance apart. In this article, Hamming distance of two nodes is the number of bits that are different in IDs of two nodes.

Using the greedy approach, the node with the largest distance to the nearest existing roots is selected as the next root and so until all roots will be selected. As the hamming distance represents a metric distance, OPR can easily identify the farthest node in the system.

In extent of Replication, it decides how to determine the replication degree for each object to achieve the best performance. OPR concludes that the optimal replication degree (number of pointers) is directly proportional to the query arrival rate and inversely proportional to the system churn rate.

Each root peer applies the maintenance protocol as follow: When a node wants to leave the network, it sends a message to its neighbors to inform them of its intention. Each neighbor receives this message should update its routing entries. The pointers kept on node leaving the network are pushed to the neighbor which has the ID numerically nearest to it.

Discussion: Plover strategy is different from all the strategies presented for structured P2P in different points:

Plover uses geographical clustering, such as making file replicas among physically close nodes based on nodes available capacities. By considering node available capacity and locality, plover achieves not only high efficiency in file replication but also facilitates efficient file consistency maintenance.

The consistency maintenance is an important issue. To the best of our knowledge, Plover is the only strategy which addressed this issue together with file replication relative to others strategies presented in this paper. It uses efficient file consistency maintenance with low-cost.

Additional, plover adopts lottery scheduling method to efficient achieve file query load balance between replica nodes, unlike symmetric and ID-Replication which use random method. In random load balancing, file query is routed randomly to a replica node. The random method is not efficient

TABLE I. COMPARISON BETWEEN DIFFERENT REPLICATION STRATEGIES IN STRUCTURED P2P NETWORKS.

Replication Technique and goal		Type of replica placement algorithm	Replica maintenance	Replication degree	Proximity	Replica nodes location	Feature
Achieve load balancing	Lightweight Adaptive system-neutral replication protocol [27]	owner replication	not mentioned	is one copy whenever the replication algorithm is applied	likely to find a replica preceding the root destination	piggybacking replica nodes location on messages	uses LRU as the replica replacement strategy
	A Prediction-Based Fair replication algorithm [28]	path replication	not mentioned	is adjusted according to the load value for each replication process	likely to find a replica preceding the root destination	piggybacking replicas nodes location on messages	uses LRU as the replica replacement strategy
	Symmetric replication [29]	multi publication key replication	more complex maintenance	theory is the same for each item	choose the numerically closet replica ID	each node can calculate the key of the different replicas	can achieve load balancing between replica by sending requests to a random replica. can be applied to all DHTs
Increase availability of resources	Quorum-based replication [31]	neighbor replication	provided by the root node	number of replicas can not exceed the neighbor list	request can be routed to the root node or to the replica node	piggybacking replicas node (replication sets) location on ping message	reduces the maintenance traffic under churn comparing the simple replication
	Availability-based replication [31]	neighbor replication	provided by the root node	is adjusted according to the node availability and the request data availability	request can be routed to the root node or to the replica node	piggybacking replicas node (replication sets) location on ping message	reduces the maintenance traffic under churn comparing the simple replication
	DHT-based self-adapting replication protocol [32]	multi publication key replication	not mentioned	is adjusted according to average peer online probability and the request data availability	choose the numerically closest replica ID	each node can calculate the key of the different replicas	generates less storage costs comparing to Availability based replication
	Predecessor replication [33]	neighbor replication	provided by the root node and by replica nodes	can not exceed the predecessor-lists size and is the same for each item	can find a replica node preceding the root node	root node maintains the localization of replica nodes	can reduce the number of hops needed to locate the requested data compared to neighbor, symmetric,
Enhance churn tolerance	RelaxDHT replication [34]	neighbor replication	provided by the root node and by replica nodes	number of replicas can not exceed the neighbor list size	can find a replica node preceding the root node	each node maintains metadata describing the localization of replicas sets	the root peer does not necessarily store a copy
	ID-Replication strategy [35]	neighbor replication	provided by replica nodes	is between two parameters Rmin and Rmax	request can be routed to random replica node in the group	it not necessary to maintain the information about the location of replica nodes	can allows more copies for popular data objects.
Improve search performance	Proactive Low-Overhead File replication scheme [36]	neighbor replication	not mentioned	is based on node available capacities	request is routed to the root node first	super node maintains metadata describing the localization of replica nodes for each replication file	adopts lottery scheduling method to achieve file query load balance uses efficient file consistency maintenance with low-cost
	An On-line Pointer replication algorithm [37]	multi publication key replication	provided by replica node	is directly proportional to the query arrival rate and inversely proportional to the system churn rate	choose the numerically closest replica ID	each node can calculate the keys of the different replicas	uses an heuristic called Greedy approach to select the roots

and it can choose the same replica node. Thus, the replica node can become overloaded.

Plover has not clearly stated how the peer applies the maintenance protocol. Unlike the others strategies which based on Multi Publication Key, OPR uses an heuristic approach called Greedy approach to select the roots. This method is simple to use and allows OPR to reduce the query search latency.

5) Summary: In this article, we explored all the techniques that have a significant scientific contribution. Each technique has a main replication objective to achieve and should take into consideration important factors and parameters. The latter have a direct impact on the system performance. Therefore, it is necessary to use the heuristics performing compromise between these factors and parameters proves to be necessary, in order to reduce the cost of replication without compromising its efficiency.

After this study, we try to draw some useful recommendations to take into consideration in replication strategies:

- Replication degree must not be the same for each data, because some of them are popular and others are not. Unpopular data must have less replicas than popular data. Thus, it decreases overhead of the maintenance protocol for unpopular data. Further, replication degree must be minimized as much as possible without compromising its efficiency. Therefore, it decreases the overhead of maintenance protocol and consistency maintenance.
- The search request must not be routed to the root node

first in order to not overcome it. This case can appear in the neighbor replication. Additional, it is necessary to increase the utilization of the replica nodes and to apply a mechanism of load balancing between replica nodes when it is possible.

- The consistency maintenance is an important issue, and it must be addressed together with the replication technique in order to reduce its overhead and to facilitate its execution. The replication technique must be designed to ease consistency maintenance like for example in neighbor replication, the root peer maintains the localization of replica nodes. Therefore, the root peer can easily forward the update message to the replica nodes if there is update. In path replication, it is very difficult to maintain the localization of all replica nodes. Then, not all data can be up-to-date.
- If P2P application requires mutual consistency, in our opinion the replication strategy which is based on multi publication key replication can facilitate mutual consistency. Each peer can calculate the key of different replica nodes. In this case, a replica peer can forward update message to other replica nodes. When a peer rejoins the network later, it contacts online replica nodes to recuperate updated data.
- Storage capacity constraint must not be ignored while the choice of suitable replica node and the replica placement strategy must applied when is necessary.

Table I summarizes the comparison of replication strategies for structured P2P networks presented above in function of some criteria: replica placement algorithm, replica maintenance, replication degree, proximity [7] (selecting a 'nearby' replica (in the ID space)), replica nodes location (existence of meta-information of the localization of replica nodes) and feature (other characteristics).

V. Conclusion and future work

Replication techniques are widely employed to improve the availability of data, enhancing performance of query latency and load balancing, in content distribution systems such as P2P. In this paper, a state of the art of the various replication techniques for structured P2P networks is presented. Thereafter, a new classification for these techniques is introduced, a detailed comparison is done. In our future work, we try to develop simultaneously data replication and data consistency maintenance methods and take into consideration recommendations that we presented in order to achieve high efficiency at a significantly lower cost.

References

[1] "Napster," [retreived: May, 2014]. [Online]. Available: http://www.napster.co.uk

[2] "Gnutella," [retreived: May, 2014]. [Online]. Available: http://www.gnutella.com

[3] "Kazaa," [retreived: May, 2014]. [Online]. Available: http://www.kazaa.com

[4] "Bittorrent," [retreived: May, 2014]. [Online]. Available: http://www.bittorrent.com

[5] J. Kubiatowicz et al., "Oceanstore: an architecture for global-scale persistent storage," SIGPLAN Not., vol. 35, no. 11, 2000, pp. 190–201.

[6] P. Druschel and A. I. T. Rowstron, "Past: A large-scale, persistent peer-to-peer storage utility," in HotOS, 2001, pp. 75–80.

[7] S. Ktari, M. Zoubert, A. Hecker, and H. Labiod, "Performance evaluation of replication strategies in dhts under churn," in Proceedings of the 6th international conference on Mobile and ubiquitous multimedia. ACM, 2007, pp. 90–97.

[8] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," IEEE Communications Surveys and Tutorials, vol. 7, 2005, pp. 72–93.

[9] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in Middleware '01: Proc. IFIP/ACM international conference on Distributed Systems Platforms. Springer Berlin / Heidelberg, 2001, pp. 329–350.

[10] B. Y. Zhao et al., "Tapestry: A resilient global-scale overlay for service deployment," IEEE Journal on Selected Areas in Communications, vol. 22, 2004, pp. 41–53.

[11] S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, and S. Shenker, "A scalable content-addressable network," in SIGCOMM, 2001, pp. 161–172.

[12] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," Peer-to-Peer Systems, 2002, pp. 53–65.

[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 '01: Proc. 2001 conference on Applications, Technologies, Architectures, and Protocols for Computer Communications. ACM, 2001, pp. 149–160.

[14] H. Weatherspoon and J. Kubiatowicz, "Erasure coding vs. replication: A quantitative comparison," in IPTPS, 2002, pp. 328–338.

[15] W. K. Lin, D. M. Chiu, and Y. B. Lee, "Erasure code replication revisited," in In PTP04: 4th International Conference on Peer-to-Peer Computing. IEEE, 2004, pp. 90–97.

[16] O. A.-H. Hassan and L. Ramaswamy, "Message replication in unstructured peer-to-peer network." in CollaborateCom. IEEE, 2007, pp. 337–344.

[17] S. Mohammadi, H. Pedram, S. Abdi, and A. Farrokhian, "An enhanced data replication method in p2p systems," Journal of computing, vol. 2, 2010, pp. 1–5.

[18] C. Jacky, L. Kevin, and N. L. Brian, "Availability and popularity measurements of peer-to-peer file systems," 2004, [retreived: May, 2014]. [Online]. Available: http://forensics.umass.edu/pubs/chu.labonte.p2pjournal.pdf

[19] S. Manel and B. Mahfoud, "Toward a global file popularity estimation in unstructured p2p networks," in ICSNC 2013, The Eighth International Conference on Systems and Networks Communications, 2013, pp. 77–81.

[20] R. Bhagwan, S. Savage, and G. M. Voelker, "Understanding availability." in IPTPS, ser. Lecture Notes in Computer Science, M. F. Kaashoek and I. Stoica, Eds., vol. 2735. Springer, 2003, pp. 256–267.

[21] B. T. Loo, R. Huebsch, I. Stoica, and J. M. Hellerstein, "The case for a hybrid p2p search infrastructure," in IPTPS, 2004, pp. 141–150.

[22] G. Gao, R. Li, K. Wen, and X. Gu, "Proactive replication for rare objects in unstructured peer-to-peer networks," Network and Computer Applications, 2012, pp. 85–96.

[23] K. Puttaswamy, A. Sala, and B. Y. Zhao, "Searching for rare objects using index replication," in INFOCOM 2008. 27th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 13-18 April 2008, Phoenix, AZ, USA. IEEE, 2008, pp. 1723–1731.

[24] W. Ma, Y. Zhang, and X. Meng, "Distribution aware collaborative spread replication for rare objects in unstructured peer-to-peer networks," Journal of Networks, vol. 8, no. 5, 2013, pp. 991–998.

[25] L. A. Sung, N. Ahmed, R. Blanco, H. Li, M. A. Soliman, and D. Hadaller, "A survey of data management in peer-to-peer systems," School of Computer Science, University of Waterloo, 2005.

[26] B. Godfrey, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load balancing in dynamic structured p2p systems," in INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies, vol. 4, 2004, pp. 2253–2262.

[27] V. Gopalakrishnan, B. Silaghi, B. Bhattacharjee, and P. Keleher, "Adaptive replication in peer-to-peer systems," 24th International Conference on Distributed Computing Systems 2004 Proceedings, 2004, pp. 360–369.

[28] X. Zhu, D. Zhang, W. Li, and K. Huang, "A prediction-based fair replication algorithm in structured p2p systems," in ATC, 2007, pp. 499–508.

[29] A. Ghodsi, L. O. Alima, and S. Haridi, "Symmetric replication for structured peer-to-peer systems," in DBISP2P, 2005, pp. 74–85.

[30] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, "Search and replication in unstructured peer-to-peer networks." in SIGMETRICS. ACM, 2002, pp. 258–259.

[31] K. Kim and D. Park, "Reducing replication overhead for data durability in dht based p2p system," IEICE Transactions, vol. 90, no. 9, 2007, pp. 1452–1455.

[32] P. Knezevic, A. Wombacher, and T. Risse, "Dht-based self-adapting replication protocol for achieving high data availability," in Advanced Internet Based Systems and Applications. Springer, 2009, pp. 201–210.

[33] F. Ben Guirat and I. Filali, "An efficient data replication approach for structured peer-to-peer systems," in Telecommunications (ICT), 2013 20th International Conference on. IEEE, 2013, pp. 1–5.

[34] S. Legtchenko, S. Monnet, P. Sens, and G. Muller, "Relaxdht: A churn-resilient replication strategy for peer-to-peer distributed hash-tables," TAAS, vol. 7, no. 2, 2012, p. 28.

[35] T. M. Shafaat, B. Ahmad, and S. Haridi, "Id-replication for structured peer-to-peer systems." Euro-Par'12 Proceedings of the 18th international conference on Parallel Processing, 2012, pp. 364–376.

[36] H. Shen and Y. Zhu, "Plover: A proactive low-overhead file replication scheme for structured p2p systems," in Proceedings of IEEE International Conference on Communications, ICC 2008, Beijing, China, 19-23 May 2008. IEEE, 2008, pp. 5619–5623.

[37] J. Zhou, L. N. Bhuyan, and A. Banerjee, "An effective pointer replication algorithm in p2p networks," in 22nd IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2008, Miami, Florida USA, April 14-18, 2008. IEEE, 2008, pp. 1–11.