On Improving the Performance Dependability of Unstructured P2P Systems via Replication

Anirban Mondal Yi Lifu Masaru Kitsuregawa

Institute of Industrial Science
University of Tokyo, Japan
{anirban,yilifu,kitsure}@tkl.iis.u-tokyo.ac.jp

Abstract. The ever-increasing popularity of peer-to-peer (P2P) systems provides a strong motivation for designing a *dependable* P2P system. Dependability in P2P systems can be viewed from two different perspectives, namely system reliability (the availability of the individual peers) and system performance (data availability). This paper looks at dependability from the viewpoint of system performance and aims at enhancing the dependability of unstructured P2P systems via dynamic replication, while taking into account the disproportionately large number of 'free riders' that characterize P2P systems. Notably, the sheer size of P2P networks and the inherent heterogeneity and dynamism of the environment pose significant challenges to the improvement of dependability in P2P systems. The main contributions of our proposal are two-fold. First, we propose a dynamic data placement strategy involving data replication, the objective being to reduce the loads of the overloaded peers. Second, we present a dynamic query redirection technique which aims at reducing response times. Our performance evaluation demonstrates that our proposed technique is indeed effective in improving user response times significantly, thereby increasing the dependability of P2P systems.

1 Introduction

The ever-increasing popularity of peer-to-peer (P2P) systems provides a strong motivation for designing a *dependable* P2P system. Dependability in P2P systems can be viewed from two different perspectives, namely system reliability (the availability of the individual peers) and system performance (data availability). Incidentally, the peers are typically distributively owned, thereby implying that we do *not* have much control over the availability of the individual peers and hence, this paper specifically addresses performance issues concerning data availability. We define a *performance-dependable* P2P system as one that the users can rely on for obtaining data files of their interest in real-time. In other words, the data should largely remain available as well as easily accessible to users. Hence, we shall use the term "dependability" throughout this paper to imply "performance-dependability". Moreover, we shall use the terms 'peers' and 'nodes' interchangeably throughout this paper.

Given the unprecedented growth of data in existing P2P systems such as Gnutella[5] and Kazaa[7], efficient data management has become a necessity to provide real-time response to user requests. Incidentally, it is now well-known that most peers in a P2P system do *not* offer any data i.e., a majority of the peers typically download data from a small percentage of peers that offer data[1]. As a result of such skews in the initial data distribution among the peers, a disproportionately high number of queries need to be answered by a few 'hot' peers, thereby leading to severe load imbalance throughout the system. The job queues of the 'hot' peers keep increasing, thereby resulting in significantly increased waiting times and consequently high response times for queries directed to them. This decreases the *dependability* of the system. The sheer size of P2P networks and the inherent heterogeneity and dynamism of the environment pose significant challenges to the improvement of dependability in P2P systems. This paper focusses on improving the dependability of unstructured P2P systems via dynamic data replication. The main contributions of our proposal are two-fold.

1. We propose a dynamic data placement strategy involving data replication, the objective being to reduce the loads of the overloaded peers.
2. We present a dynamic query redirection technique which aims at reducing response times.

Our performance evaluation demonstrates that our proposed technique is indeed effective in reducing user response times significantly, thereby increasing the dependability of P2P systems. The remainder of this paper is organized as follows. Section 2 discusses related work, while Section 3 presents an overview of our proposed system. Section 4 presents the proposed replication and query redirection strategy, while Section 5 reports our performance evaluation. Finally, we conclude in Section 6 with directions for future work.

2 Related Work

Existing P2P systems such as Pastry [11] and Chord [12] emphasize specifically on query routing, while the work in [3] proposes routing indices, the primary objective being to forward a query *only* to those peers which are likely to contain the answers to the query. Unlike broadcast approaches, routing indices attempt to avoid flooding the network with queries. Replication has also been studied in P2P systems primarily for improving search operations. The proposal in [6] investigates optimal replication of content in P2P systems and develops an adaptive, fully distributed algorithm which dynamically replicates content in a near-optimal manner. Notably, replication strategies for P2P systems have also been presented in [2, 8], but since the objective of replication in these works is to facilitate search, these works do *not* specifically address issues concerning dependability.

Dependability via load-balancing in structured P2P systems (using distributed hash tables or 'DHTs') has been addressed in [4, 10]. Moreover, the work in [13] discusses dependability via inter-cluster and intra-cluster load-balancing in a P2P system which is divided into clusters based on semantic categories. Note that our work differs from these works in that we address dependability issues in unstructured P2P systems which neither impose a logical structure on the P2P system (as in [13]) nor assume DHT abstraction (as in [4, 10]).

Incidentally, our previous work [9] concerning load-balancing in spatial GRIDs has some similarity with this work in that they both aim at reduction of response times in wide-area environments. However, there are several major differences. First, in contrast to this work, the proposal in [9] imposes a structure on the system by dividing the entire system into sets of clusters. Second, replicating one tuple of a spatial database in a spatial GRID entails data movement at most in the Kilobyte range, while P2P data movements are usually in the Megabyte range (e.g., for music files) or even in the Gigabyte range (e.g., for video files), the implication being that the communication cost of data movement can be expected to be significantly higher in case of P2P systems. Third, for spatial GRIDs, prevention of data scattering is a major concern, which is *not* really a concern in case of P2P systems. Fourth, in case of spatial GRIDs, individual nodes are usually dedicated and they may be expected to be available most of the time, while for P2P systems, nodes may join or leave arbitrarily at any point of time. Fifth, the work in [9] aims at load-balancing, while this work investigates dynamic replication issues in detail without any explicit load-balancing aims.

3 System Overview

In our proposed system, each peer is assigned a globally unique identifier *PID* and for search, we adopt a broadcast-based approach[5]. For detecting hotspots, every peer maintains its own access statistics i.e., the number of accesses made to each of its data files. Moreover, for each data file D_i, each peer keeps track of all the peers which have downloaded D_i from itself. Additionally, every peer provides a certain amount $Space_i$ of its disk space to the P2P system for storing the replicas of other peers' 'hot' data files. In other words, $Space_i$ is the available disk space at each peer that can be used for replication purposes, whenever the need arises. To optimize the usage of $Space_i$ at each peer, we adopt the commonly used LRU (Least Recently Used) scheme. To address dynamically changing popularities of files in P2P systems, each peer checks the number of accesses N_k (for recent time intervals) for each data file replicated at itself and deletes files, for which N_k falls below a pre-specified threshold. Additionally, in consonance with most existing works concerning replication in P2P systems, we sacrifice replica consistency for improving response times.

We define *distance* between two peers as the communication time τ between them and two peers are regarded as neighbours if they are directly connected to each other. Messages concerning load status and available disk space are periodically exchanged between neighbouring peers. Additionally, we define the load L_{P_i} of a peer P_i as the number of queries waiting in P_i's job queue. Given that the loads of two peers P_i and P_j are L_{P_i} and L_{P_j} respectively and assuming without loss of generality that $L_{P_i} > L_{P_j}$, the normalized load difference Δ between P_i and P_j is computed as follows:

$$\Delta = ((L_{P_i} \times CPU_{P_i}) - (L_{P_j} \times CPU_{P_j}))/(CPU_{P_i} + CPU_{P_j}) \quad (1)$$

where CPU_{P_i} and CPU_{P_j} are the processing capacities of P_i and P_j respectively. Moreover, we assume that peers know transfer rates between themselves and other peers and every peer has knowledge concerning the availability information of its neighbouring peers. In practice, after the system has been in operation for a significant period of time, the peers will have exchanged several messages between themselves and over a period of time, such information can be obtained by the peers. Given that very 'hot' files may be aggressively replicated across hundreds of peers in a very transitive manner and some peers may quickly become out of reach of the primary copy owner, each peer keeps track *only* of the replications that it has performed i.e., whenever a peer replicates any of its data files at other peers, it notes the PIDs of those peers. For example, when a 'hot' data file D_i is replicated by peer P_i to another peer P_j, P_i will note that D_i has been replicated at P_j. However, if subsequently P_j replicates D_i at another peer P_k, P_j (and *not* P_i) would note this replication information.

4 Proposed Replication and Query Redirection Strategy for P2P systems

This section presents our proposed strategy for replication and query redirection in P2P systems.

Initiation of replication

Each peer P_i periodically checks the loads of its neighbouring peers and if it finds that its load exceeds the average loads of its neighbouring peers by 10%, it decides that it is overloaded and initiates replication by selecting the 'hot' data files. For hotspot detection purposes, P_i maintains its own access statistics comprising a list, each entry of which is of the form ($dataID$,f), where $dataID$ represents the identifier of a specific data file and f indicates the number of times the data file had been queried. Notably, in order to deal with the inherent dynamism of P2P environments where the popularity of data files typically change from time to time, we take *only* recent access statistics information into consideration for detecting hotspots. P_i sorts all its data files in descending order of the access frequencies of the files. For identifying hotspots, P_i traverses this sorted list of data files and selects as 'hot' files the top N files whose access frequency exceeds a pre-defined threshold T_{freq}. The number of replicas to be created for each 'hot' data file D_i is decided by the number of accesses to D_i. In particular, for every N_d accesses to D_i, a new replica is created for D_i. Notably, the values of

T_{freq} and N_d are pre-specified by the system at design time. Now the destination peer(s) where D_i should be replicated must be determined efficiently. Hence, we shall now discuss how the destination peer(s) for D_i are selected.

Proposed Replication Strategy

The 'hot' peer P_{Hot} considers the following selection criteria for selecting a destination peer P_{Dest} for replication of D_i.

- P_{Dest} should have a high probability of being online (available).
- P_{Dest} should have adequate available disk space for replication. If P_{Dest} does not have sufficient disk space, D_i's replica at P_{Dest} may be subsequently deleted by the LRU scheme used at P_{Dest} in favour of hotter data items.
- Load difference between P_{Hot} and P_{Dest} should be significant enough to call for replication at P_{Dest}.
- Transfer time T_{Rep} between P_{Hot} and P_{Dest} should be minimized. T_{Rep} can be computed as $F_i \div T_i$, where F_i is the size of the file to be replicated and T_i is the transfer rate of the network connection between P_{Hot} and P_{Dest}. Since files in P2P systems are typically in the range of Megabytes (for music files) and Gigabytes (for video files), T_{Rep} can be expected to be a significant cost. Interestingly, D_i is a 'hot' data file, the implication being that D_i is likely to exist in the disk of at least some of the peers which had earlier queried for D_i and downloaded D_i from P_{Hot}. Hence, we propose that P_{Dest} should be chosen from the peers which have already downloaded D_i. This has the advantage of making T_{Rep} effectively equal to 0.

Based on the above criteria for selecting P_{Dest}, we shall now present our replication strategy. For each 'hot' data file D_i, the 'hot' peer P_{Hot} sends a message to each peer which has downloaded D_i during recent time intervals, enquiring whether a copy of D_i is *still* stored in them. (Some of the peers which have downloaded D_i may have subsequently deleted D_i.) The peers in which a copy of D_i exists reply to P_{Hot} with their respective load status information as well as the amount of available disk space that they have for replication purposes. Among these peers, only those with high availability and sufficient available disk space for replication of D_i are candidates for being the destination peer. Now, among these candidate peers, P_{Hot} first puts the peer with the lowest load into a set which we designate as *Candidate*. Additionally, peers whose normalized load difference with the least loaded peer is less than δ are also put into *Candidate*. Note that δ is a small integer, the significance of δ being that two peers are considered to be having approximately the same load if the load difference between them is less than δ. Then the peer in *Candidate* whose available disk space for replication is maximum is selected as the destination peer. Figure 1 depicts the algorithm for selecting the destination peer.

Proposed technique for Query Redirection

When a peer P_{Issue} issues a query Q for data item D_i to a 'hot' peer P_{Hot}, P_{Hot} needs to make a decision concerning the redirection of Q to a peer containing

Algorithm *Select_DestPeer*
P_{Hot}: The 'hot' peer which needs to select a destination peer for its 'hot' data file D_i
$SetP_{Download}$: Set of peers which have downloaded D_i from P_{Hot}

P_{Hot} sends a message to each peer in set $SetP_{Download}$ enquiring whether they still have a copy of D_i and if so, their load and available disk space information.

Upon receiving the replies, P_{Hot} deletes those peers from $SetP_{Download}$ that do *not* have a copy of D_i.
P_{Hot} deletes the peers with low availability from $SetP_{Download}$.
P_{Hot} deletes the peers, whose available disk space is not adequate for D_i, from $SetP_{Download}$.
if ($P_{Download}$ is an empty set) {
  end
} else {
  P_{Hot} selects the peer P_{min} with the least load from $SetP_{Download}$ and puts it into a set $Candidate$.

  Peers of $SetP_{Download}$ whose normalized load difference with P_{min} falls below δ are put into $Candidate$.
  Among the members of $Candidate$, the peer with maximum available disk space for replication is selected as the destination peer.
}
end

Fig. 1. Algorithm for selecting the destination peer

D_i's replica, if any such replica exists. The peer $P_{Redirect}$ to which Q should be redirected must be selected such that Q's response time is minimized. In our proposed strategy, P_{Hot} checks the list L_{Rep} comprising the PIDs of the peers where it had replicated D_i and selects $P_{Redirect}$ based on the following criteria:

- $P_{Redirect}$ should have a high probability of being online (available).
- Load difference between P_{Hot} and $P_{Redirect}$ should be significant.
- Transfer time between $P_{Redirect}$ and P_{Issue} should be low.

In consonance with the above criteria, our query redirection technique works as follows. The 'hot' peer P_{Hot} first selects a set of peers which contain a replica of the data file D_i associated with the query Q and whose load difference with itself exceeds T_{Diff}. T_{Diff} is a parameter which is application-dependent and also depends on how one considers the system to be imbalanced. A small value of T_{Diff} would encourage replications (albeit at the cost of disk space), while a large value of T_{Diff} would possibly result in lesser number of replications. Note that the normalized load difference is compared with T_{Diff} to take the heterogeneity in processing capabilities of different peers into consideration. Among these selected peers, the peer with the *maximum* transfer rate with the query issuing peer P_{Issue} is selected for query redirection. Notably, this is in consonance with our objective of reducing response times. Figure 2 depicts the query redirection algorithm.

Algorithm Query_Redirect
L_{Rep}: List comprising the PIDs of peers which contain a replica of the 'hot' data file D_i associated with the query Q.
P_{Issue}: The peer which originally issued the query.
P_{Hot}: The 'hot' peer which needs to redirect Q.

for each peer P_j in L_{Rep} {
  P_{Hot} checks the normalized load difference L_D between itself and P_j.
  if ($L_D \geq T_{Diff}$) {
    P_{Hot} puts P_j into a set $Set_{Redirect}$.
  }
}

P_{Hot} selects the peer, whose transfer rate with P_{Issue} is maximum from $Set_{Redirect}$.
end

Fig. 2. Query redirection algorithm

5 Performance Study

We conducted extensive simulation experiments to evaluate the performance of our proposed replication strategy. Our simulation environment comprised a machine running the Solaris 8 operating system. The machine has 4 CPUs, each of which has a processing power of 900 MHz. Main memory size of the machine is 16 Gigabytes, while the total disk space is 2 Terabytes. We used a maximum of 4 neighbouring peers corresponding to each peer. The interarrival time for queries arriving at each peer was fixed at 1 millisecond. Table 1 summarizes the parameters used for our performance study. In Table 1, z is a parameter whose value equals **(1 - zipf factor)**. This implies that when z=0.1, the skew is high and when z=0.9, the skew is low. Note that in all our experiments, in order to model free-riders, we directed queries *only* to 1% of the total number of peers in the system and these peers become the 'hot' peers (data providers), the rest of the peers being free-riders. For all our experiments, the system was allowed to run for sometime for collection of access statistics information and we started recording the results only after the system had reached a stable state. Hence, all our experimental results indicate the performance of our proposed strategy during the stable state. Additionally, in all our experiments, the 'hot' peers always remained available (*online*), while the availability of other (non-hot) peers was randomly selected in the range of 10% to 90%. Our main performance metric is query response time. For the sake of convenience, we shall henceforth refer to our proposed dynamic replication scheme as **DRep** (Dependability via Replication) and the policy of *not* performing replications as **NoRep** (no replication).

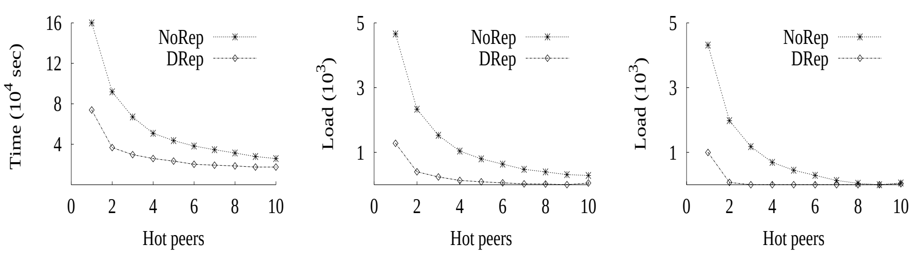
Performance of DRep

Figure 3 indicates the results for the default values of the parameters i.e., the case in which the total number of peers was 1000, queries were directed to only 10 of

Parameter	Default value	Variations
No. of peers	1000	5000, 10000
No. of peers to which queries are directed	10	50,100
No. of queries	20000	100000, 200000
z	0.1	0.5, 0.9
Number of replicas	4	
Interarrival time between queries	1ms	
Transfer rate between peers	0.5 Mb/s to 1 Mb/s	
Latency	10 ms to 20 ms	
Size of a file	1 MB to 10 MB	

Table 1. Parameters used in Performance Study

these peers, the number of replicas initially being 4 and z=0.1. Figure 3a depicts the *average response times* at each of the 10 'hot' peers. Observe that there is significant reduction of average response times at each of the 'hot' peers, the reduction in average response time being maximum for the hottest peer. Further investigation of the experimental log files revealed that DRep was able to reduce the average response time of the hottest peer by upto 50%. Such reduction in average response time is possible owing to load reduction at the 'hot' peers as shown in Figures 3b and 3c, which present two snapshots (taken at different points in time) of the load distribution at the 'hot' peers.



(a) Average Response times at the hot nodes

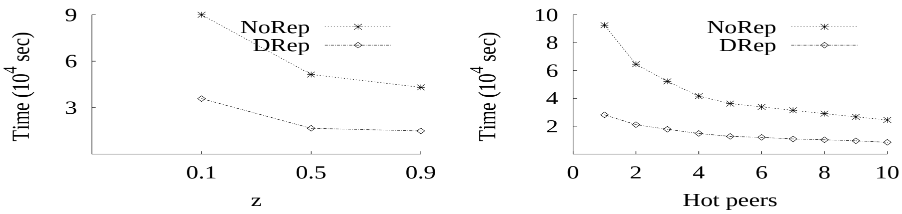
(b) Snapshot of Load distribution

(c) Snapshot of Load distribution

Fig. 3. Performance of DRep

Variations in Workload Skew

Now let us examine the effect of variations in workload skews among the 10 'hot' peers on the average query response times. For this purpose, we varied z to 0.5 and 0.9. Figure 4a displays the average response time of all the queries in the system for different values of z. The results show that DRep significantly outperforms NoRep for variations in workload skews. However, the gain in terms



(a) Average Response times at the hot nodes

(b) Average Response times at the hot nodes for z=0.5

Fig. 4. Effect of varying the Workload Skew

of average response time is higher in case of highly skewed workload (i.e., z=0.1) and the gain in average response time keeps decreasing as the workload skew decreases. This occurs because as the workload skew decreases, the load becomes more evenly distributed among the 'hot' nodes, the implication being that the load at the hottest peer also decreases, thereby reducing the waiting times of queries at the hottest peer. Figure 4b depicts the average response times at the hot peers when z was fixed at 0.5, the explanations for these results being essentially the same as the explanations for Figure 3.

Variation in the number of peers

Now we shall investigate the scalability of DRep with respect to the total number of peers in the system. For this experiment, as the total number of peers in the system is increased, the number of queries in the system is increased in a proportional manner. This is in consonance with real-life situations because as the number of peers in the system increases, more peers are likely to issue queries, thereby increasing the number of queries circulating in the system. The number of queries for systems comprising 1000, 5000 and 10000 peers was 20000, 100000 and 200000 respectively. Moreover, the number of 'hot' peers for systems consisting of 1000, 5000 and 10000 peers was fixed at 10, 50 and 100 respectively i.e., in each case, the number of 'hot' peers was 1% of the total number of peers in the system. The number of replicas was initially 4. Figure 5 shows the average response time of *all* the queries when the total number of peers was varied. The results in Figure 5 demonstrate the scalability of DRep and indicate that DRep provides more performance gain over NoRep as the number of peers increases primarily because increased number of peers implies more options for performing replication and more possibilities for query redirection. The implication is that the load imposed on the 'hot' peers by queries on the 'hot' data files can be distributed among a larger number of peers by replicating the 'hot' data files at those peers.

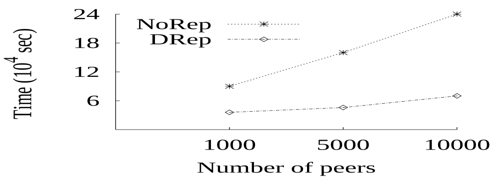


Fig. 5. Effect of varying the number of peers

6 Conclusion

The sheer scale, dynamism and heterogeneity of P2P environments coupled with the presence of disproportionately large number of 'free-riders' pose significant challenges to dependability (in terms of data availability) of P2P systems. In this regard, we have proposed a novel strategy for enhancing the dependability of P2P systems via dynamic replication. Our performance evaluation demonstrate that our proposed technique is indeed able to enhance the dependability of P2P systems by reducing response times significantly. In the near future, we plan to extend this work by considering issues concerning replication of very large data items such as video files.

References

1. Adar and Huberman. http://news.bbc.co.uk/1/hi/sci/tech/948448.stm.
2. E. Cohen and S. Shenker. Replication strategies in unstructured peer-to-peer networks. *Proc. ACM SIGCOMM*, 2002.
3. A. Crespo and H. G. Molina. Routing indices for Peer-to-Peer systems. *Proc. ICDCS*, 2002.
4. F. Dabek, M.F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. *Proc. SOSP*, 2001.
5. Gnutella. http://www.gnutella.com/.
6. J. Kangasharju, K. W. Ross, and D. A. Turner. Optimal content replication in P2P communities. *Manuscript*, 2002.
7. Kazaa. http://www.kazaa.com/.
8. Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker. Search and replication in unstructured peer-to-peer networks. *Proc. ACM ICS*, 2002.
9. A. Mondal, K. Goda, and M. Kitsuregawa. Effective load-balancing via migration and replication in spatial GRIDs. *Proc. DEXA*, 2003.
10. A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica. Load balancing in structured P2P systems. *In Proc.IPTPS*, 2003.
11. A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. *Proc. IFIP/ACM*, 2001.
12. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *Proc. ACM SIGCOMM*, 2001.
13. P. Triantafillou, C. Xiruhaki, M. Koubarakis, and N. Ntarmo. Towards high performance peer-to-peer content and resource sharing systems. *Proc. CIDR*, 2003.