A Case for Content Distribution in Peer-to-peer Networks

Morteza Analoui, Mohsen Sharifi , and Mohammad Hossein Rezvani
Department of Computer Engineering, Iran University of Science and Technology (IUST)
16846-13114, Hengam Street, Resalat Square, Narmak, Tehran, Iran
{analoui, msharifi, rezvani}@iust.ac.ir

Abstract. In large scale peer-to-peer networks, it is impossible to perform a query request by visiting all peers. There are some works that try to find the location of resources probabilistically (i.e. non-deterministically). They all have used inefficient protocols for finding the probable location of peers who manage the resources. This paper presents a more efficient protocol that is proximity-aware in the sense that it is able to cache and replicate the popular queries proportional to distance latency. The protocol dictates that the farther the resources are located from the origin of a query, the more should be the probability of their replication in the caches of intermediate peers. We have validated the proposed distributed caching scheme by running it on a simulated peer-to-peer network using the well-known Gnutella system parameters. The simulation results show that the proximity-aware distributed caching can improve the efficiency of peer-to-peer resource location services.

Keywords: Distributed systems, Peer-to-Peer network, Content Distribution, Performance Evaluation.

1. Introduction

A peer-to-peer (P2P) system is a distributed system consisting of interconnected peers who are able to self-organize into network topologies with the purpose of sharing resources such as CPU or bandwidth, capable of adapting to dynamic conditions of network, without requiring the support of a global centralized server [1]. Unlike the structured systems, in the unstructured systems such as Gnutella [2, 3] and KazaA [4] searching mechanisms are employed to discover the location of the resources. Each peer owns a set of resources to be shared with other peers. The most significant searching mechanisms include brute force methods (e.g. flooding the network with propagating queries in a breath-first or depth-first manner until the desired content is discovered) [1], probabilistic searches [5], routing indices [6], randomized gossiping, semantic spaces [7] and, so on.

Most of the current P2P systems such as Gnutella, KazaA, and Pastry [8] fall within the category of P2P *"content distribution"* systems. A typical P2P content distribution system creates a distributed storage medium and allows doing services such as searching and retrieving query messages which are known as *"resource location"* services. The area of *"content distribution systems"* has a large overlap with the issue of *"resource location services"* in the literature.

In general, there are two strands of work concerning the proximity-aware methodology. First, there are works on content distribution via constructing the P2P topology [9]. Second, there are works on resource location services [10, 11, 12]. These works assume a given topology setting such as mesh or tree for the P2P system. It has been shown by [13, 14] that finding an optimal-bandwidth topology for the P2P network is an NP-complete problem. So, we shall not try to solve the NP problem of topology construction here. Instead, we will try to optimize the proximity-aware resource locating problem within the given topology setting in the P2P system.

In this paper, we are concerned with the design of a resource location service via scalable proximity-aware distributed caching mechanism. We define the resource location service as “given a resource name, find with a proximity probability, the location of peers that manage the resource.” We use round-trip time (RTT) latency distance as the criterion for the probabilistic caching of each query. Each peer, upon receiving a query, at first searches its local cache. If the query is found, the peer returns it to the original requesting peer along with the reverse path which is traversed by the query. In this order, the so called query is cached in the memory of each intermediate node using replication method based on the proposed proximity-aware distributed caching mechanism. The probability of the resource replication and updating of the caches in each intermediate node is proportional to the latency distance between that node and the location where the resource is found. To the best of our knowledge, there has been no investigation on designing the proximity-aware probabilistic caching in the P2P systems. The rest of the paper is organized as follows. We discuss the related researches in Section 2. Section 3 presents our proposed proximity-aware resource location mechanism. Section 4 presents the performance evaluation of the proposed mechanism. Finally, we conclude in Section 5.

2. Related work

A significant research toward proximity-aware resource location services in typical Gnutella-based unstructured P2P system has been done in [10, 11, 15, 16]. Some query search broadcasting policies using Gnutella system has been proposed in [15, 16] and their performance has also been compared with each other. The proximity metric in [10, 11] is time-to-live (TTL) of the query messages. Forwarding the queries is done with a fixed probability. When a query message is reached to a peer, its TTL is decremented. The forwarding of the query messages will be stopped if its TTL is reached to zero.

Another work which incorporates the temporal proximity information in resource location services is [12]. They have used the so called information to minimize the load movement cost in the system by allowing higher capacity peers carry more loads. Also, a graph-based analytical study on the impact of the proximity-aware methodology for making the P2P streaming more scalable have been presented in [9].

The main motivation for many researches in the area of P2P systems was early "loosely controlled" systems such as Gnutella, Freenet [17], Napster [18], and Morpheus [19]. The search technique proposed in [20] is similar to local indices technique which is proposed in [15] with different routing policy for query message. In the other proposed techniques which mentioned in [15] each node maintains "hints" as to which nodes contain data that answer certain queries, and route messages via local decisions based on these hints. This idea itself is similar to the philosophy of hints which is used by Menasce et al. in [10]. Pastry [8], CAN [21], and Chord [22] are examples of systems with "strong guarantee" that employ search techniques. These systems can locate an object by its global identifier within a limited number of hops.

3. Proximity-Aware Distributed Caching

Each pair of nodes (s,r) is associated with a latency distance $lat(s,r)$ representing the average round-trip-time (RTT) experienced by communication between them. The latency distance corresponding to a specific pair of nodes may be measured either directly through *ping* messages, or estimated approximately through a virtual coordinate service. Due to space limitations, we do not explain the details of the virtual coordinate service here. Interested readers can refer to [23] for it.

Every super-peer in our system has a Local Index Table (LIT) that points to locally managed resources (such as files, Web pages, processes, and devices). Each resource has a location-independent globally unique identifier (GUID) that can be provided by developers of the P2P network using different means. For example, Freenet GUID keys are calculated using SHA-1 secure hashes. In a distributed online bookstore application, developers could use ISBNs as GUIDs [11]. Each super-peer has a directory cache (DC) that points to the presumed location of resources managed by other super-peers. An entry in the DC is a pair *(id, loc)* in which *id* is the GUID of a resource and *loc* is the network address of a super-peer who might store the resource locally. Each peer s has a local neighborhood $N(s)$ defined as the set of super-peers who have connected to it.

Tables 1 and 2 provide a high-level description of the proposed proximity-aware distributed caching mechanism. The *QuerySearch* (QS) procedure describes the operations in which a source s is looking for a resource, namely res. The string path $s_1,\ldots,s_m$ is the sequence of super-peers that have received this message so far. This sequence is used as a reverse path to the source. The header of each query message contains a TTL field which is used to control the depth of the broadcast tree. For example, Gnutella has been implemented with a TTL parameter equal to 7. The *QueryFound* (QF) procedure indicates that the resource res being searched by the source s has been found at super-peer v. In this procedure, the *max_latency* is the latency distance between the super-peer who manages res and the farthest super-peer in the reverse path.

Each super-peer, upon receiving the QS message, at first searches within its LIT. If it finds the resource in the

LIT, it will return a QF message. The QF message is forwarded to the source following the reverse path which has been used by the QS message. It updates the DCs corresponding to each of the intermediate nodes as well. The contribution of our work emerges at this point where the QF message updates the LIT in each of the intermediate nodes using replication of resources based on the proposed proximity-aware distributed caching mechanism.

The probability of resource replication and updating the LIT corresponding to each intermediate node is proportional to the latency distance between that node and the location where the resource has been found.

Table 1- *QuerySearch* message received by super-peer r .

```
QuerySearch( source , res , (s_1, …, s_m), TTL)
begin
  If $res \in$ LIT then
  begin
    max_latency= $\max\{(lat(r,s_1),...,lat(r,s_m)\}$
    send QueryFound( source , res , max_latency, (s_1, …, s_{m-1}), r ) to s_m
  end
  else if ( res , loc) $\in$ DC then
      /* send request to presumed location */
      Send QuerySearch( source , res , (s_1, …, s_m, r ), TTL-1) to loc
      else if (TTL > 0) then
        for $v_i = v_1$ to $v_m$ do      /* $v_i \in N(r)$ */
        begin
          max_latency= $\max\{(lat(r,v_1),...,lat(r,v_m)\}$
          Send QuerySearch( source , res ,(s_1, …, s_m, r), TTL-1) with probability $p$ to $v_i$
        /* probability $p$ is proportional to $\frac{lat(r,v_i)}{\max\_latency}$ */
        end for
      end if
  end if
end
```

Table 2- *QueryFound message received by super-peer* r .

```
QueryFound( source , res , max_latency, (s_1, …, s_m), v )
begin
  if $r \neq source$ then
  begin
    add ( res , v ) to DC
    with probability proportional to $\frac{lat(r,v)}{\max\_latency}$ do
    begin
      Connect to super-peer v to get res from it
      /* finds a local client with enough available memory */
      find local client c
      add ( res , c ) to LIT
    end
    send QueryFound( source , res , max_latency, (s_1, …, s_{m-1}), v) to s_m
  end
  else   /* end of query search process */
    connect to super-peer v to get res from it.
end
```

To this end, each intermediate node r performs the following actions with a probability that is proportional to the latency distance between itself and the node which has been found as the manager of the resource: 1) establishing a TCP connection with the super-peer who manages the resource, 2) downloading the resource object and saving it in the client c who has enough available space, and 3) updating the LIT via adding the entry (res, c).

If the super-peer does not find the resource in its LIT but finds it in the DC, it will send a QS message to the super-peer who is pointed to by that DC. If this super-peer no longer has the resource, the search process will be continued from that point forward. If a super-peer does not find the resource neither in its LIT nor DC, it will forward the request to each super-peer in its neighborhood with a certain probability p which is called the *"broadcasting probability."* This probability could vary with the length of the path that the request traverses.

Fig. 1 illustrates how a QS message would be propagated in the network. In the figure, the maximum number of the nodes to be traversed by a QS message is defined to be equal to 3 hops (apart from the source node itself). Similar to Gnutella, our system uses a Breath-First-Search (BFS) mechanism in which the depth of the broadcast tree is limited by the TTL criterion. The difference is that in Gnutella every query recipient node forwards the message to all of its neighbors, while in our proposal, the propagation is performed probabilistically and is done if the query is not found neither in the LIT nor in the DC of a node. In Fig. 1, the QS message originating from source S_1 is probabilistically sent to super-peers S_2, S_3, and S_4 in response to the search for the resource res. The super-peer S_3 finds the resource in its LIT, but S_2 and S_4 do not find such an entry, hence probabilistically forward the message to the peers who have been registered in their DCs. Note that the super-peer S_4 does not forward the message to S_{10} because, for example, in this case the forwarding probability is randomly selected to be zero.

Figure 2 illustrates an example of returning QF messages in a reversed path from the location where the resource res is found to the node which the query has been originated from. The QF message is routed to the source (node S_1) following the reverse path which is used by the QS message. The QF message updates the corresponding DC of each intermediate node based on the proposed proximity-aware distributed caching mechanism.

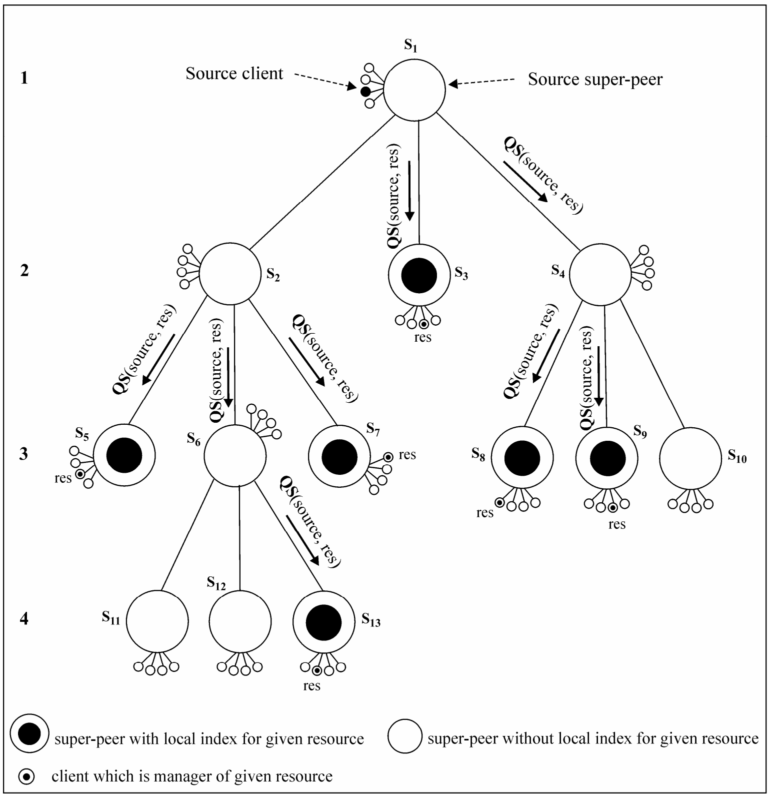

Fig. 1- Forwarding a QS message using maximum hop-count equal to 3.

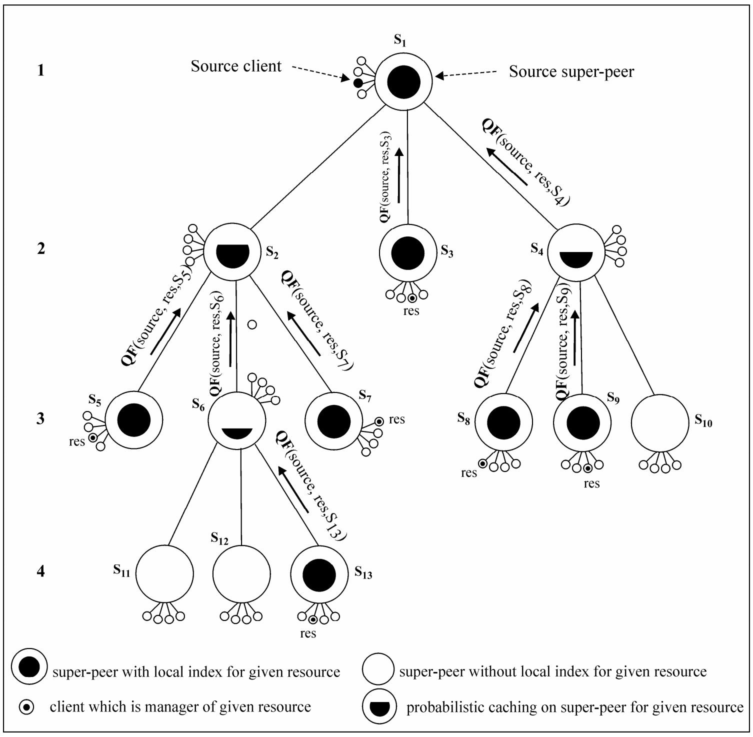

Fig. 2- Forwarding the QF message through the reversed path.

The probability of replication and caching the resource object in the LIT of each intermediate node is proportional to the latency distance between that node and the location where the resource is found. The closer is the intermediate node to the discovered resource; the less will be the probability of caching the resource in the node's LIT. This probability is shown by a graphical representation with partially boldfaced circles. In the sequence of nodes which consists of S_1, S_2, S_6, and S_{13}, the node S_6 caches the address of the resource *res* with the least probability; whereas the node S_1 caches it with the most probability. The probability of caching the resource *res* by S_2 is larger than that of S_6 and is smaller than that of S_1.

4. Experimental Analysis

We have performed a large number of experiments to validate the effectiveness of our proximity-aware distributed caching scheme. We have evaluated the performance of the system with a file-sharing application based on several metrics. These metrics include fraction of involving super-peers in the query search, probability of finding an entry in DCs, overall cache miss ratio, average number of hops to perform the query requests, and the system load. Among these metrics, the *load* metric is defined as the amount of work that an entity must do per unit of time. It is measured in terms of two resource types: incoming bandwidth, and outgoing bandwidth. Since the availability of the incoming and the outgoing bandwidths is often asymmetric, we have treated them as separate resources. Also, due to heterogeneity of the system, it is useful to study the aggregate load, i.e., the sum of the loads concerning to all the nodes in the system. All of the results are averaged over 10 runs of experiments and have been come up with 95% confidence intervals. We followed the general routine devised in [24] for the efficient design of the P2P network. So, as the first step, we had to generate an instance topology based on a power-law distribution. We used the PLOD algorithm presented in [25] to generate the power-law topology for the network. The second step was calculating the expected cost of actions. Among three "*macro*" actions, i.e., *query*, *join*, and *update*, which exist in a cost model [24], we have restricted our attention to the query operations. Each of these actions is composed of smaller "*atomic*" actions for which the costs are given in [24]. In terms of bandwidth, the cost of an action is the number of bytes being transferred. We used the specified size of the messages for Gnutella protocol as is defined in [24]. For example, the query messages in Gnutella include a 22-byte Gnutella header, a 2 byte field for flags, and a null-terminated query string. The total size of a query message, including Ethernet and TCP/IP headers, is therefore 82 plus the query string length. Some values, such as the size of a metadata record are not specified by the protocol, rather are functions of the type of the data which is being shared. The values which are used from [24] are listed in Table 3.

Table 3- Gnutella bandwidth costs for atomic actions.

Atomic Action	Bandwidth Cost (Bytes)
Send Query	82 + query length
Recv. Query	82 + query length
Send Response	80+28×#addresses+76×#results
Recv Response	80+28×#addresses+76×#results

To determine the number of the results which are returned to a super-peer r, we have used the query model developed in [26] which is applicable to super-peer file-sharing systems as well. The number of files in the super-peer's index depends on the particular generated instance topology I. We have used the so called query model to determine the expected number of the returned results, i.e. $E[N_r \mid I]$. Since the cost of the query is a linear function of $(N_r \mid I)$ and also since the load is a linear function of the cost of the queries, we can use these expected values to calculate the expected load of the system [24].

In the third step, we must calculate the system load using the actions. For a given query originating from the node s and terminating in the node r we can calculate the expected cost, namely C_{sr}. Then, we need to know the rate at which the query action occurs. The default value for the query rate is 9.26×10^{-3} which is taken from the general statistics provided by [24] (see Table 4). The query requests in our experiments have been generated by a workload generator. The parameters of the workload generator can be set up to produce uniform or non-uniform distributions. Considering the cost and the rate of each query action, we can now calculate the expected load which is incurred by the node r for the given network instance I as follows

$$E[M_r \mid I] = \sum_{s \in Network} E[C_{sr} \mid I].E[F_s] \tag{1}$$

Where, F_s is the number of the queries submitted by the node s in the time unit, and $E[F_s]$ is simply the query rate per user.

Let us define Q as the set of all super-peer nodes. Then, the expected load of all such nodes, namely M_Q is defined as follows

$$E[M_Q \mid I] = \frac{\sum_{n\in Q} E[M_n \mid I]}{|Q|} \tag{2}$$

Also, the aggregate load is defined as follows

$$E[\overline{M} \mid I] = \sum_{n \in network} E[M_n \mid I] \tag{3}$$

We ran the simulation over several topology instances and averaged $E[M \mid I]$ over these trials to calculate $E[E[M \mid I]] = E[M]$. We came up with 95% confidence intervals for $E[M \mid I]$. The settings used in our experiments are listed in Table 4. In our experiments, the network size was fixed at 10000 nodes. As mentioned before, the generated network has a power-law topology with the average out-degree of 3.1 and TTL=7. These parameters reflect Gnutella topology specifications which has been used by many researchers so far. For each pair of the super-peers (s, r), the latency distance $lat(s, r)$ was generated using a normal distribution with an average $\mu = 250_{ms}$ and a variance $\delta = 0.1$ [23]. Then, to find the pair-wise latency estimation, namely $est(s, r)$, we ran the virtual coordinate service method over the generated topology.

Table 4- Experimental settings

Name	Default	Description
Graph Type	Power-law	The type of network which may be strongly connected or power-law
Graph Size	10000	The number of peers in the network
Cluster Size	10	The number of peers per cluster
Avg. out-degree	3.1	The average out-degree of a super-peer
TTL	7	The time-to-live of a query message
Query Rate	9.26×10^{-3}	The expected number of queries per user per second

In order to be able to compare the results with the previous works, we chose a cache size per super-peer equal to 1% of the total number of the resources managed by all super-peers. The Least-Frequency-Used (LFU) is a typical frequency-based caching policy which has been proved to be an efficient policy in the area of distributed systems [27]. In LFU, the decision to replace an object from the cache is proportional to the frequency of the references to that object. All objects in the cache maintain the reference count and the object with the smallest reference count will be replaced. The criterion for replacing an object from the cache is computed as follows

$$Cost_{Object} = frequency_{Object} \times recency_{Object} \tag{4}$$

Where, $frequency_{Objeect}$ and recency_{Object} denote the *"access frequency"* and the *"elapsed time from recent access"*, respectively. If the cache has enough room, LFU will store the new object in itself. Otherwise, LFU selects a candidate object which has the lowest $Cost_{Object}$ value among all cached objects. Then, LFU will replace the candidate object by the new object if the $Cost_{Object}$ of the new object is higher than that of the candidate object. Otherwise, no replacement occurs.

Figure 3 shows the experimental results concerning the effect of the resource replication on the fraction of participating super-peers, namely F, and the probability of finding objects, namely P_f, versus different broadcasting probabilities. It can be seen from the figure that P_f attains high values for much smaller values of p. By adjusting the broadcasting probability, one can tune the probability of finding the resource. In the case of using resource replication, P_f achieves larger values in comparison with the case in which the resource replication is not used. In contrast to P_f, the metric F achieves smaller values in the case of using the resource replication in comparison with the case in which the resource replication is not used. The reason lies in the fact that in the case of using the resource replication method, some intermediate nodes replicate the queries in their local disks (i.e., they cache the queries into their LIT); leading to a decrease in the LITs miss ratio, thus an increase in the probability of finding the queries. Such nodes do not need to propagate the *QuerySearch* message to other super-peers anymore.

Figure 4 shows the effect of using the resource replication on the cache miss ratio as a function of the broadcasting probability p. In the both cases of Fig. 4, the cache miss ratio decreases by an increase in p. Its cause lies in the fact that when p increases, more super-peers participate in the search; hence it is more likely that the resource is found by more than one super-peer. So, more DCs of the intermediate nodes in the reverse path to the source will be aware of the resource; leading to a decrease in the DCs miss ratio. The use of the resource replication decreases the miss ratio compared to the case in which the resource replication is not used. However, the amount of the reduction in the miss ratio is not remarkable in both cases for values of p greater than 0.8. At this point, the use of resource replication yields a miss ratio of 0.72; giving approximately 20% improvement over the 0.9 miss ratio when no resource replication is used

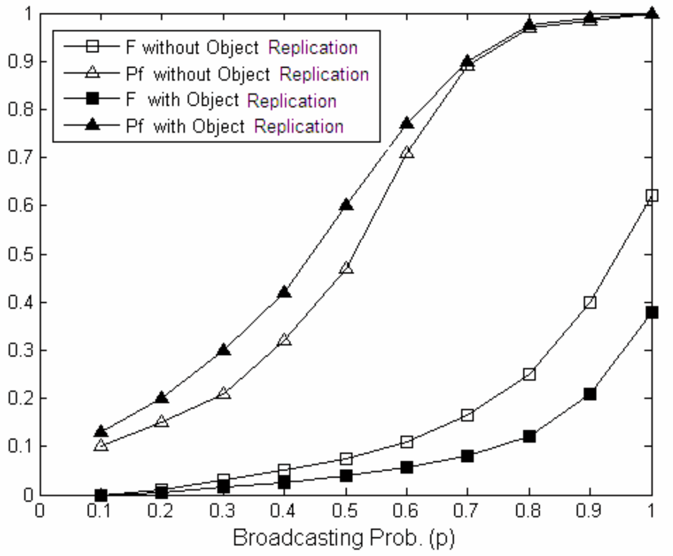

Fig. 3- The effect of resource replication on the fraction of participating peers and the probability of finding objects for various broadcasting probabilities.

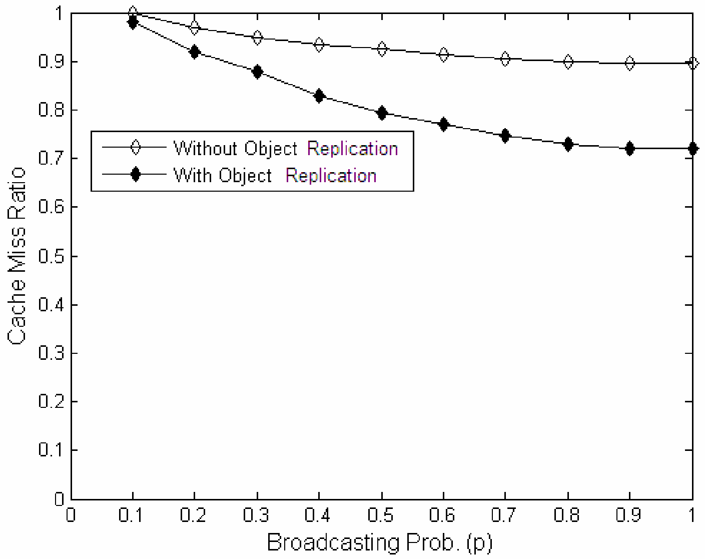

Fig. 4- The effect of resource replication on overall cache miss ratio for various broadcasting probabilities.

Figure 5 shows the average number of the required hops to find the resource, namely H, which is normalized by the total number of super-peers (except the original source). The figure shows the effect of the resource replication method in various broadcasting probabilities. It can be seen in both curves of the Fig. 5 that the average number of hops initially increases until reaches to a maximum point and then begins to decrease. A higher broadcasting probability means that the super-peers who are located further away from the original source are contacted and the resource tends to be found further away from the original source. As p continues to increase, the increased values of hit ratio concerning to intermediate DCs allow the resource to be found in locations where are closer to the original source; hence causes a decrease in the value of H . It is clear from the Fig. 5 that the use of resource replication reduces the number of hops needed to find the resource. For example, in a reasonable practical point of broadcasting probability, such as 0.7, it yields a 31% improvement, where the hop ratio decreases from 0.08 to 0.055

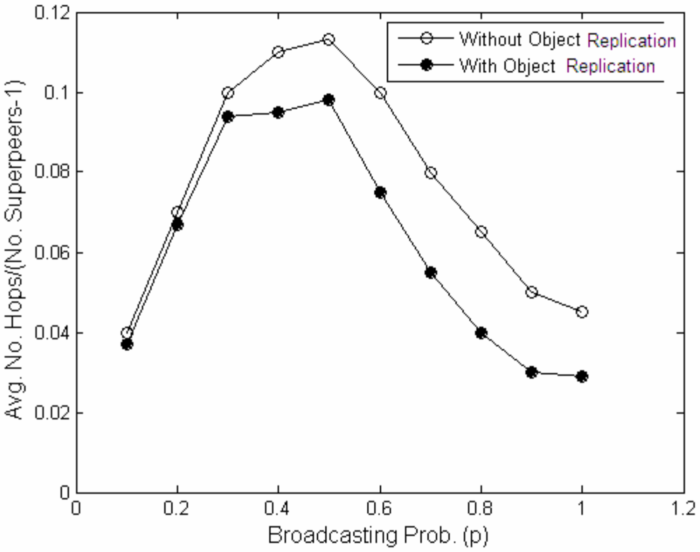

Fig. 5- The effect of resource replication on hop ratio for various broadcasting probabilities.

Figure 6 shows the effect of resource replication on the total required bandwidth of the system, i.e. the required incoming and outgoing bandwidth of super-peers for various broadcasting probabilities. By increasing the broadcasting probability, some additional costs are imposed to the system. The most important costs include the cost of sending queries to each super-peer, a startup cost for each super-peer as they process the query, and the overhead of additional packet headers for individual query responses. Some of these factors are mentioned in the literature by prior researchers. Interested readers can find useful hints in [24]. The upper curve in Fig. 6 shows the required bandwidth in the absence of the resource replication. In this case, as the broadcasting probability p increases, the required bandwidth of the super-peers increases and reaches to 7.7×10^{8} *bps* for a value of p equal to 0.8. From this point forward, the growing of bandwidth occurs more slightly until reaches to 7.9×10^{8}*bps* at the value of p equal to 1. The lower curve in Fig. 6 shows an improvement in the required bandwidth in the presence of the resource replication. In this case, the required bandwidth decreases to 6.6×10^{8}*bps* for a value of p equal to 0.8, resulting in a 14% improvement in comparison with the same point in the upper curve.

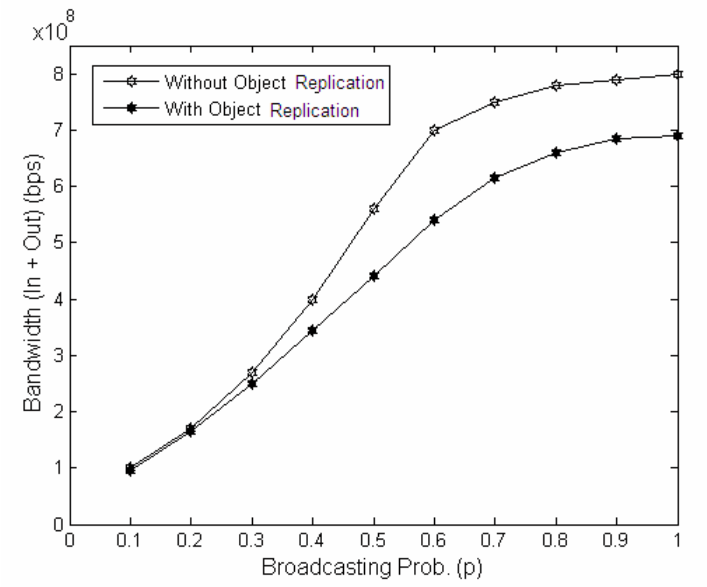

Fig. 6- The effect of resource replication on total required bandwidth for various broadcasting probabilities.

5. Conclusions

In this paper we have targeted the proximity-aware location service for peer-to-peer systems. The proposed protocol provides a scalable distributed caching mechanism to find the peers who manage a given resource and achieves an enhancement by replicating the objects based on the latency distance factor, resulting in less aggregate load over the system. The simulation results showed that using the probabilistic resource discovery service in peer-to-peer systems combined with latency-aware probabilistic resource replication, improves the overall performance of the system in terms of aggregated load, throughput, response time, number of hops, and the number of the peers who are involved in the search process.

References

1. Androutsellis-Theotokis, S., Spinellis, D.: A Survey of Peer-to-Peer Content Distribution Technologies. ACM Computing Surveys, vol. 36, no. 4, pp. 335-371 (2004)
2. Ripeanu, M., Foster, I., Iamnitchi, A.: Mapping the Gnutella Network: Properties of Large-Scale Peer-to-Peer Systems and Implications for System Design. IEEE Internet Computing, 6(1) (2002)
3. Hughes, D., Coulson, G., Walkerdine, J.: Free Riding on Gnutella Revisited: The Bell Tolls?. IEEE Distributed Systems Online, Volume 6, Issue 6 (2005)
4. The Kazaa web site (2007), http://www.kazaa.com.
5. Lv, Q., Cao, P., Cohen, E., Li, K., and Shenker, S.: Search and Replication in Unstructured Peer-to-Peer Networks. the 16th ACM International Conference on Supercomputing (ICS'02). New York, NY.(2002)
6. Crespo, A., Garcia-Molina, H. : Routing Indices for Peer-to-Peer Systems. Proc. of Int. Conf. on Distributed Computing Systems, Vienna, Austria (2002)
7. Cuenca-Acuna, F.M., Nguyen, T.D.: Text-Based Content Search and Retrieval in Ad Hoc P2P Communities. International Workshop on Peer-to-Peer Computing, Springer-Verlag (2002)
8. Rowstron, A., and Druschel, P. : Pastry: Scalable, Distributed, Object Location and Routing for Large-Scale Peer-to-Peer Systems. IFIP/ACM International Conference on Distributed System Platforms (Middleware), Heidelberg, Germany, pp. 329–350, (2001)

9. Dai, L., Cao, Y., Cui, Y., and Xue, Y. : On Scalability of Proximity-Aware Peer-to-Peer Streaming, in Computer Communications. Elsevier, vol. 32, no 1, pp. 144-153 (2009)
10. Menascé, D.A., Kanchanapalli, L.: Probabilistic Scalable P2P Resource Location Services. ACM Sigmetrics Performance Evaluation Rev., Volume 30, No. 2, pp. 48–58 (2002)
11. Menascé, D.: Scalable P2P Search. IEEE Internet Computing, Volume 7, No. 2 (2003)
12. Zhu, Y., Hu, Y.: Efficient, Proximity-Aware Load Balancing for DHT-Based P2P Systems. IEEE Transactions on Parallel and Distributed Systems, Vol. 16, No. 1, pp. 349-361 (2005)
13. Zhu, Y., Li., B.: Overlay Networks with Linear Capacity Constraints. IEEE Transactions on Parallel and Distributed Systems, 19 (2), pp. 159-173 (2008)
14. Zhu, Y., Li, B., Pu., K. Q.: Dynamic Multicast in Overlay Networks with Linear Capacity Constraints. IEEE Transactions on Parallel and Distributed Systems, 20 (7), pp. 925-939 (2009)
15. Yang, B., Garcia-Molina, H. : Improving Search in Peer-to-Peer Networks. The 22nd International Conference on Distributed Computing Systems (ICDCS'02), Vienna, Austria (2002)
16. Cooper, B.F., Garcia-Molina, H.: SIL: A Model for Analyzing Scalable Peer-to-Peer Search Networks. Computer Networks, Volume 50, No. 13, pp. 2380-2400 (2006)
17. Clarke, I., Miller, S. G., Hong, T. W., Sandberg, O., and Wiley, B.: Protecting Free Expression Online with Freenet. IEEE Internet Computing , Volume 5, No. 1, pp. 40-49 (2002)
18. Napster website. http://www.napster.com.
19. Morpheus website. http://www.morpheus-os.com.
20. Adamic, L., Lukose, R., Puniyani, A., and Huberman, B.: Search in Power-Law Networks. Available at http://www.parc.xerox.com/istl/groups/iea/papers/plsearch/ (2001)
21. Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Shenker, S.: A Scalable Content Addressable Network. Proc. ACM Sigcomm (2001)
22. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., and Balakrishnan, H.: Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. Proc. ACM Sigcomm, (2001)
23. Jesi, G.P., Montresor, A., Babaoglu, O.: Proximity-Aware Superpeer Overlay Topologies. IEEE Transactions on Network and Service Management (2007)
24. Yang, B., Garcia-Molina, H.: Designing a Super-Peer Network. Proc. Int'l Conf. Data Eng. (ICDE), pp. 49-63 (2003)
25. Palmer, C., Steffan, J.: Generating network topologies that obey power laws. The GLOBECOM (2000)
26. Yang, B., Garcia-Molina, H.: Comparing Hybrid Peer-to-Peer Systems. Proc. 27th Int. Conf. on Very Large Data Bases, Rome (2001)
27. Song, J.W., Park, K.S., Yang, S.B.: An Effective Cooperative Cache Replacement Policy for Mobile P2P Environments. In proceeding of IEEE International Conference on Hybrid Information Technology (ICHIT'06), Korea, Vol. 2, pp. 24-30 (2006)