Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844
Vol. V (2010), No. 4, pp. 418-431

Probabilistic Proximity-aware Resource Location in Peer-to-Peer Networks Using Resource Replication

M. Analoui, M. Sharifi, M.H. Rezvani

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

Abstract: Nowadays, content distribution has received remarkable attention in distributed computing researches and its applications typically allow personal computers, called peers, to cooperate with each other in order to accomplish distributed operations such as query search and acquiring digital contents. In a very large network, 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 in terms of the probability of finding objects, overall miss rate of the system, fraction of involved peers in the search process, and the amount of system load.
Keywords: Distributed systems, Peer-to-Peer network, Content Distribution, Resource Location, Performance Evaluation.

1 Introduction

1.1 Motivation

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]. The P2P systems are classified as unstructured and structured. In the structured systems such as CAN [2] the overlay topology is tightly controlled and files are placed at exact locations. These systems provide a distributed routing table, so that queries can be routed to the corresponding peer who manages the desired content. Unlike the structured systems, in the unstructured systems such as Gnutella [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. In general, the shared resource can be any kind of data which make sense, even records stored in a relational database. 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, and so on.

Most of the current P2P systems such as Gnutella and KazaA 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. Fig. 1 illustrates a possible topology of the unstructured P2P networks. Each super-peer is a powerful master node that acts as a local central indexer for files which are shared by its local peers, whereas acts as an ordinary peer for other super-peers. In graph representation, each pair of peers is connected by an edge representing a TCP connection between them. The number of neighbors of a super-peer is called its out-degree. If two nodes are not connected by an edge, they could communicate through an indirect path which passes across some other nodes. The length of a path through which two nodes communicate with each other is known as hop-count. Upon delivery of a query request message to a super-peer, it looks for matches over its local database. If any matches are found, it will send a single response message back to the node which has requested the query. If no match is found, the super-peer may forward the query request to its neighbor super-peers.

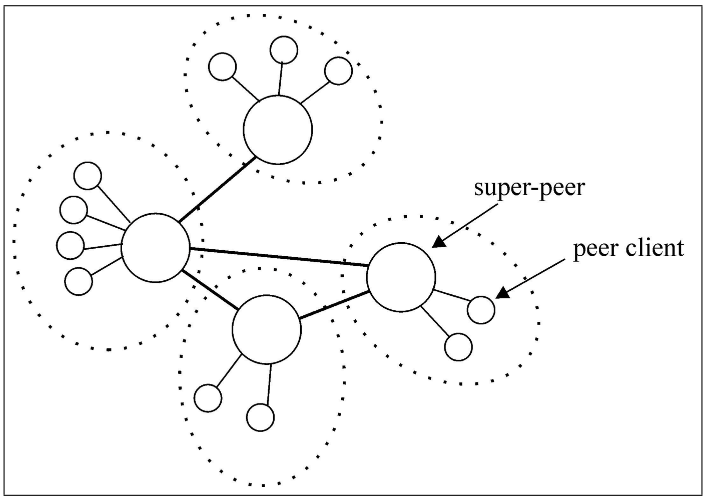

Figure 1: A typical super-peer network

In general, the performance of the P2P systems is strictly evaluated by metrics such as response time, number of hops, aggregated load, throughput, overall miss rate, fraction of participating nodes in the search operation, and so on. To meet these requirements, previous researches resorted to heuristics to locate the resources by incorporating proximity concerns.

1.2 Challenges

There already exists significant body of researches toward proximity-aware P2P systems. The proximity-aware resource location method in the P2P unstructured systems has been investigated in [7,8]. The proposed method uses flooding mechanism to forward a query request to all neighbors of a peer. It uses the hop-count as the proximity metric. In order to reduce the number of broadcast messages in the network, the header of each query message contains a time-to-live (TTL) field whose value is decremented at each hop. Finally, when the TTL reaches zero, the query message is dropped from the network. After locating the resource, a direct connection is established between the originating peer and the destined peers and the file is downloaded. The flooding approach employed in [7] is probabilistic in the sense that each peer replicates the query to its neighbors with a fixed probability.

An analytical study on the impact of proximity-aware methodology for the content distribution is presented in [9]. They have evaluated the performance of video streaming P2P network via key scalability metric, namely network load. Similar to the previous works, they have used the pair-wise latency of the peers as the proximity criterion.

Regard to the aforementioned works, 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 [7, 8]. These works assume a given topology setting such as mesh or tree for the P2P system. It has been shown in [10, 11] 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.

1.3 Contributions

In this paper, we are concerned with the design of a resource location service by using 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 who 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. Section 2 presents our proposed proximity-aware resource location mechanism along with its specification such as resource replication. Section 3 provides an analytical study of the probabilistic search method along with numerical results. Section 4 presents the experimental validation of the proposed mechanism. Finally, we discuss the related works in Section 5, and conclude in Section 6.

2 Proximity-aware Distributed Caching

Each pair of nodes is associated with a latency distance representing the average 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 [12]. Due to large number of nodes in a P2P system, we have adopted the latter approach to measure the latency between each pair of nodes. The virtual coordinate service which has been used in [12] is provided by VIVALDI [13], a distributed protocol developed at MIT. Due to space limitations, we do not explain the details of the virtual coordinate service here. Interested readers can refer to [12, 13] for it. As sated above, some works use the hop-count as the criterion for the distance estimation rather than using VIVALDI estimation method.

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. In a distributed online bookstore application, developers could use ISBNs as GUIDs [8]. 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 has a local neighborhood defined as the set of super-peers who have connected to it. Table 1 and Table 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 is searching a resource, namely . The string path $s_1, ..., s_m$ is the sequence of super-peers who 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 being searched by source has been found at super-peer . In this procedure, the max_latency is the latency distance between the super-peer who manages and the farthest super-peer in the reverse path.

Figure 2 illustrates a typical unstructured P2P computing system. A given resource is managed by nodes $S_3, S_5, S_7, S_8, S_9, S_{13}$, and S_{18}. The resource is saved as a file on disk memory corresponding to clients who are clustered by aforementioned super-peers. Inspecting the DC of super-peers S_{11}, S_{12}, and S_{16} for example, reveals that the resource is located in the super-peer S18. The LITs corresponding to the nodes $S_1, S_2, S_4, S_6, S_{10}, S_{11}, S_{12}, S_{14}, S_{15}, S_{16}$, and S_{17} are empty; indicating that they are not themselves managers of the resource . Also, the DCs corresponding to the nodes $S_1, S_{10}, S_{11}, S_{12}, S_{14}, S_{15}$, and S_{17} are empty; indicating that they do not know the address of the owner of the resource res.

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 of 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. To this end, each intermediate node r performs the following actions with a probability proportional to the latency distance between itself and the peer who 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 into client *c* who has enough available space.
3) updating the LIT by adding the entry *(res,c)* to it.

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

```
QuerySearch(source, res, (s_1, ..., s_m), TTL)
begin
  If res ∈ LIT then
  begin
    max_latency= $\max\{(lat(r,s_1), lat(r,s_2), \ldots, lat(r,s_m)\}$
    send QueryFound(source, res, max_latency, (s_1, ..., s_{m-1}), r) to s_m
  end
  else if (res, loc) ∈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= $\max\{(lat(r,v_1), lat(r,v_2), \ldots, lat(r,v_m)\}$
        Send QuerySearch(source, res, (s_1, ..., s_m, r), TTL-1) with probability $p$ to $v_i$.
        /* The probability $p$ is proportional to $\frac{lat(r,v_i)}{max\_latency}$ */
      end
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 resource res from it
      find a local client, c, with enough available memory
      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 resource res from it.
end
```

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 in its LIT or 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.

Figure 3 illustrates how a QS message would be propagated in the network. In the figure, the maximum number of nodes to be traversed by a QS message is defined to be equal to 3 hops (apart from the source node). 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 node receiving a query 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. 3, the QS message originating from source S_1 is probabilistically sent to super-peers S_2, S_3, and S_4 due to 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 nodes 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 probability of forwarding is randomly selected to be zero.

Figure 4 illustrates an example of returning QF messages in a reversed path from the location where the resource *res* is found, to the node who has originated the query request. 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. The probability

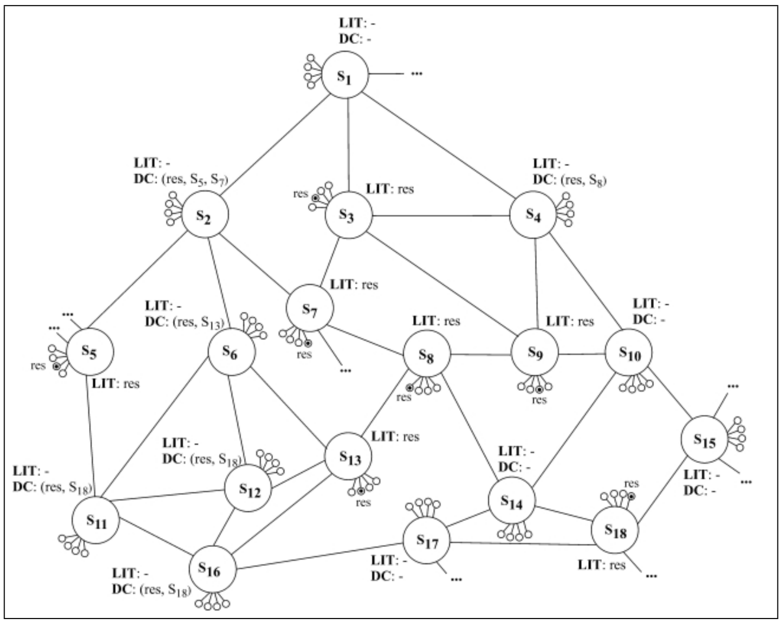

Figure 2: The topology of a typical unstructured P2P system

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 node $_1$ caches it with the most probability. The probability of caching the resource *res* by S2 is larger than that of S_6 and smaller than that of S1.

3 The Analytic Study of the Proposed Mechanism

Considering that the corresponding super-peer of a query is located at the root of a tree of height d, each super-peer r has k_r neighbors, where k_r follows a power-law distribution. In such a distribution, the majority of nodes have relatively few local connections to other nodes, but a significant small number of nodes have large wide-ranging sets of connections. The power-law distribution gives small-world networks a high degree of fault tolerance, because random failures are most likely to eliminate nodes from the poorly connected majority [14]. Hence, each query originating from a client must visit at most d-1 super-peers. Note that the levels of the tree are numbered 1,...,d from the root down. Let q_i be the probability that a super-peer at level i has a local index for the resource *res*. Let R(i) be the number of super-peers at level i who receive a QuerySearch message from upstream super-peers, and S(i) be the number of super-peers at level i who may forward the QuerySearch message to downstream super-peers one level down. So, it must be that R(1)=0 and q_1=0.

The reason for $q_1 = 0$ lies in the fact that the resource *res* only may be found in the LIT of the super-peers who are located in the levels 2,...,d of the broadcast tree.

Let us consider j super-peers at level i-1 may not find the resource in their LIT and forward query request message one level down. Each of j super-peers, say r , located at level i-1 can select at most k_r super-peers among its neighbors to send a query request message. Let us suppose that each super-peer forwards these query messages independently to the children who are located at level i-1 with probability p_i. Let $m_1, ..., m_j$ be the number of level i children to receive the query request message from super-peers 1,...,j located at level i-1, respectively. Let us assume n super-peers receive these query requests at level i. So, we can define the tuple S as follows

$$S(j,n,k_1,...,k_j) = \{\overrightarrow{m} = (m_1,...,m_j) | \sum_{s=1}^{j} m_s = n \;\&\; 0 \leqslant m_s \leqslant k_s\} \tag{1}$$

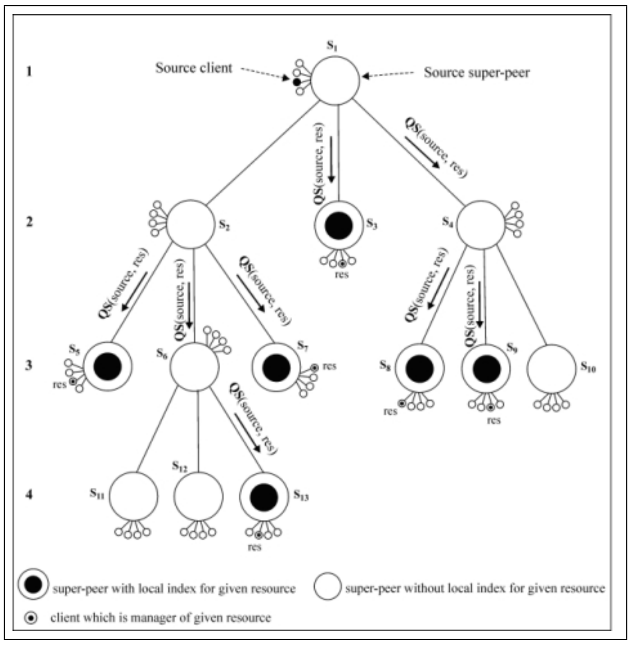
1
2
3
4
Source client
Source super-peer
S_1
S_2
S_3
S_4
S_5
S_6
S_7
S_8
S_9
S_{10}
S_{11}
S_{12}
S_{13}
QS(source, res)
res
super-peer with local index for given resource
super-peer without local index for given resource
client which is manager of given resource

Figure 3: Forwarding a QS message using maximum hop-count equal to 3

Where, $k_1, ..., k_j$ are out-degrees of nodes 1,...,j which are located at level i-1 , respectively. Now we are in a position to define conditional probability Pr[R(i)=n|S(i-1)=j] . This is the probability of receiving the *QuerySearch* messages by *n* super-peers at level *i* given that *j* super-peers at level i-1 may forward the messages one level down. With respect to the above premises we have:

$$Pr[R(1) = n|S(i-1) = j] = \sum_{\forall \overrightarrow{m} \in S(j,n,k_1,...k_s)} \prod_{s=1}^{j} \binom{k_s}{m_s} \times p_i^{m_s} \cdot (1-p_i)^{k_s m_s} \tag{2}$$

For example let us consider that,j-3, n=5, $k_1 = 2$, $k_2 = 3$ and $k_3 = 2$. Three nodes at level i-1 forward the QuerySearch message to five super-peers at level *i*. Thus, S(3,4,2,3,2)={(0,3,2), (2,1,2),(2,3,0),(2,2,1), (1,2,2),(1,3,1)}. This example shows that Eq. (2) does not depend on the order of appearance of values $m_1, ..., m_j$ in $\overline{m}$.

We can derive the probability Pr[R(i)=n] that *n* super-peers receive a QuerySearch message at level *i* as follows

$$Pr[R(i) = n] = \sum_{j=0}^{n_{i-1}} Pr[R(i) = n|S(i-1) = j] \cdot Pr[S(i-1) = j] \tag{3}$$

Where, n=0,...,$\sum_{r=1}^{j} k_r$, and n_{i-1} is the total number of nodes located at level i-1. Eq. (3) can be computed recursively by following conditions:

Pr[R(1)=1]=1, Pr[R(1=0)]=0, Pr[R(i)=0|S(i-1)=0]=1, and

Pr[R(i)=n|S(i-1)=0]=0 for n>0.

We are now in a position to compute the average number of super-peers, , involved in query search (apart from the source) as follows

$$\overline{N} = \sum_{i=2}^{d} \sum_{n=0}^{n_i} n \cdot Pr[R(i) = n] \tag{4}$$

Where, n_i is the total number of nodes located at level *i*. Note that S(i-1) is not necessarily equal to R(i-1). The probability Pr[S(i-1)=j] in Eq. (3) can be computed by conditioning on *r* super-peers that receive *QuerySearch* messages at level i-1. So, we have:

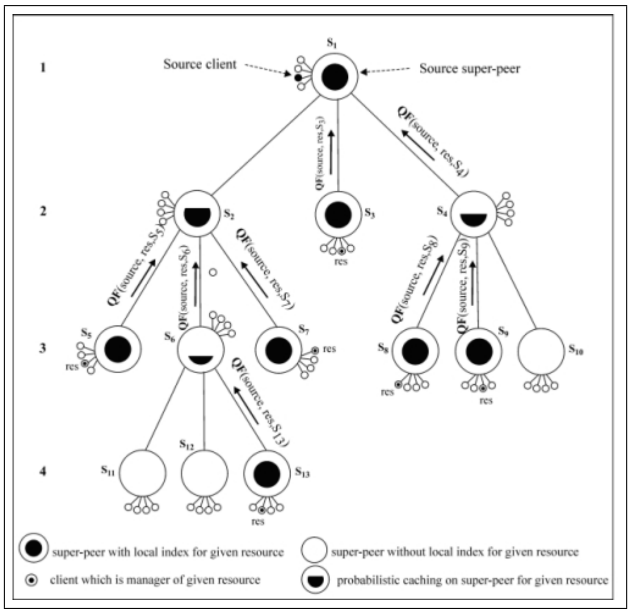

Figure 4: Forwarding a QS message using maximum hop-count equal to 3

$$Pr[S(i-1)=j]=\sum_{r=j}^{n_{i-1}} Pr[S(i-1)=j|R(i-1)=r].Pr[R(i-1)=r] \tag{5}$$

The above equation can be simplified as follows

$$Pr[S(i-1)=j]=\sum_{r=j}^{n_{i-1}} \binom{r}{j} q_{i-1}^{r-j}(1-q_{i-1})^j.Pr[R(i-1)=r] \tag{6}$$

The probability P_f that a local index for a resource is found can be computed as

$$P_f = 1-\prod_{i=2}^{d} Pr[no\ local\ index\ is\ found\ at\ level\ i] \tag{7}$$

Formally, the above equation can be expressed as follows

$$P_f = 1-\prod_{i=2}^{d}\sum_{n=0}^{n_i}(1-q_i)^n.Pr[R(i)=n] \tag{8}$$

Now, the probability that an entry for the resource is not found, namely P_{nf}, can be computed as follows

$$P_{nf}(s)=\sum_{n=0}^{n_i}(1-q_s)^n.Pr[R(s)=n] \tag{9}$$

For computing the average number of hops required to find a local index for a resource, namely $\overline{H}$, we first define $P_f^{min}(i)$ with the assumption that the first level wherein a resource is found at level i:

$$P_f^{min}(i)=\frac{\prod_{s=2}^{i-1}P_{nf}(s)][1-P_{nf}(i)]}{P_f} \tag{10}$$

Now, the average number of hops can be computed as follows

$$\overline{H}=\sum_{i=2}^{d}(i-1)P_f^{min}(i) \tag{11}$$

As mentioned earlier, Eq. (11) is approximately equivalent to mean latency distance. Before proceeding, let us define F as the ratio between the average number of super-peers, namely $\overline{N}$, involved in the query operation and the total number of super-peers except the originating super-peer:

$$F = \frac{\overline{N}}{\sum_{i=2}^{d} n_i} \tag{12}$$

Another important factor in our analysis is the load of query requests imposed by other nodes on each super-peer node. Assuming that in a deterministic query search case, the overall load on each node is $1_{Byte/sec}$, then in a probabilistic case, the overall load will be $1_{Byte/sec}$. The reason is that in the probabilistic search case, each node sees a fraction F of the requests generated by each peer. We anticipate that the idea of caching the local indices, proposed by us, causes the value of F to tend to very low values, thus resulting in a reduction in the processing load of each peer.

Now, we provide the numerical results of the above analytical model. We assume d=5. For each node j, the out-degree, namely k_j, comes from a power-law distribution with $\alpha = 1.5$. Figure 5 shows the variation of P_f and F versus p. This figure assumes a fixed message broadcasting probability, i.e., $p_i = p$ for i=2,...,d. Also the probability of caching in the DCs at each level is assumed to be $q_2 = 0.5$, $q_3 = 0.25$, $q_4 = 0.1$, and $q_5 = 0.01$. As the broadcasting probability p increases, the probability that a directory entry for the resource is found increases and exceeds 0.97 for a value of equal to 0.7. At that point, only 12% of the super-peers would participate in the search (i.e., F= 0.12).

Figure 6 shows the variation of F versus p_f. The assumptions used in this figure are the same as those of Fig. 5. It can be seen from Fig. 5 that by adjusting the broadcasting probability, one can find the probability that the resource is found. Given this, one can tune the fraction of participating nodes from Fig. 6.

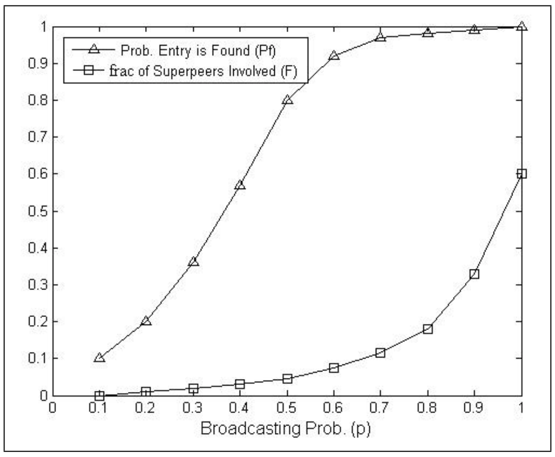

Figure 5: Probability of finding entry and fraction of participating super-peers vs. different broadcasting probabilities

4 Experimental Validation

We have performed a large number of experiments to validate the effectiveness of our proximity-aware distributed caching scheme. We 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 query requests, and system load. All of the aforementioned metrics, except system load, are already defined in the previous sections. The *load* metric is defined as the amount of work 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 the all 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 [15] for the efficient design of the P2P network. So, as the first step, we had to

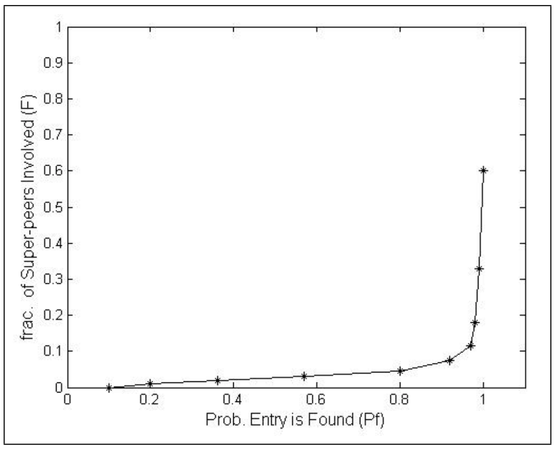

Figure 6: Fraction of super-peers involved vs. probability of finding entry

generate an instance topology based on a power-law distribution. We used the PLOD algorithm presented in [16] 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 [15], 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 [15]. 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 in such a way that is defined in [15]. 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 [15] are listed in Table 3.

Table 3: Gnutella bandwidth costs for atomic actions [15]

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 results which are returned to a super-peer r, we have used the query model developed in [17] which is applicable to super-peer file-sharing systems. 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 returned results, i.e. $E[N_r|I]$. Since the cost of the query is a linear function of $[N_r|I]$ 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 [15].

In the third step, we must calculate the system load using the actions. For a given query originating from node s and terminating in 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 [15] (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 node r for the given network instance I as follows

$$E[M_r|I] = \sum_{s \in Network} E[C_{sr}|I].E[F_s] \tag{13}$$

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

$$E[M_Q|I] = \frac{\sum_{n\in Q} E[M_n|I]}{|Q|} \tag{14}$$

Also, the aggregate load is defined as follows

$$E[\overline{M}|I] = \sum_{n\in Network} E[M_n|I] \tag{15}$$

We ran the simulation over several topology instances and averaged E[M|I] over these trials to calculate E[E[MlI]]=E[M] . We came up with 95% confidence intervals for E[MlI]. 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 have 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 $\delta = 0.1$ [12]. Then, to find the pair-wise latency estimation, namely *est(s,r)* , we ran the VIVALDI 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 nodes 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 the all super-peers. In a distributed system with highly variant reference patterns, it is better to use frequency-based replacement policies. The frequency-based policy takes into account the frequency information which indicates the popularity of an object. The Least-Frequency-Used (LFU) is a typical frequency-based policy which has been proved to be an efficient policy [18]. In LFU, the decision to replace an object from the cache is made by 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

$$Cost_{Object} = Frequency_{Object} \times Recency_{Object} \tag{16}$$

Where, $Frequency_{Object}$ 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 it. 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 of the new object is higher than that of the candidate object. Otherwise, no replacement occurs.

Figure 7 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 various broadcasting probabilities. It can be seen from the figure that P_f attains high values for much smaller values of *p*. The experimental result in Fig. 7 shows a trend similar to what the analytical study provides in Fig. 5. 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. Its cause lies in the fact that in the case of using the resource replication method, some intermediate nodes replicate the queries in their local disks (cache the queries into their LIT); leading to a decrease in the LITs miss ratio, thus resulting an increase in the probability of finding the queries. Such nodes do not need to propagate the *QuerySearch* message to other super-peers anymore.

Fig. 8 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. 8, 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 the values of p greater than 0.8. At this point, the use of resource replication method yields a miss ratio of 0.72; giving 20% improvement over the 0.9 miss ratio when no resource replication is used.

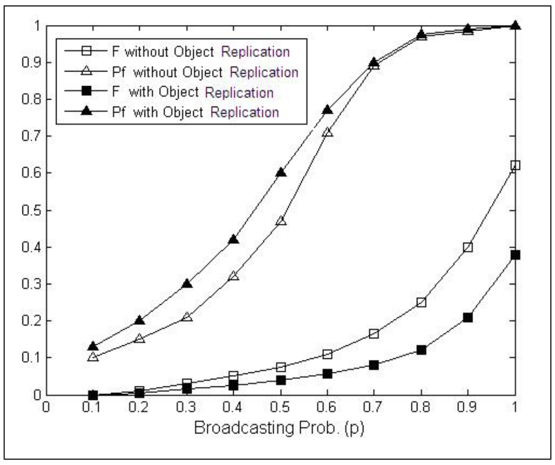

Figure 7: The effect of resource replication on the fraction of participating peers and the probability of finding objects for various broadcasting probabilities

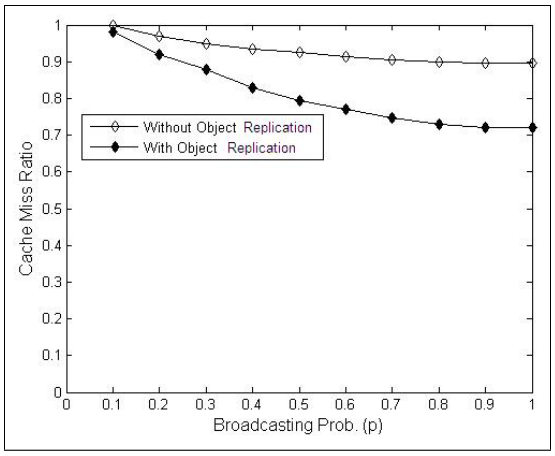

Figure 8: The effect of resource replication on overall cache miss ratio for various broadcasting probabilities

Figure 9 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 Fig. 9 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 decreasing the value of H. It is clear from Fig. 9 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, whereas the hop ratio decreases from 0.08 to 0.055.

Figure 10 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

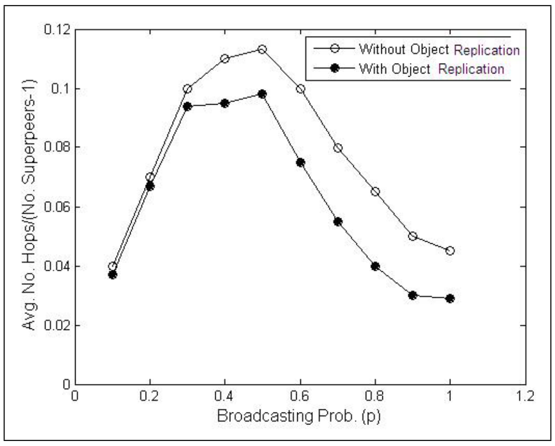

Figure 9: The effect of resource replication on hop ratio for various broadcasting probabilities

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 [15]. The upper curve in Fig. 10 shows the required bandwidth in the absence of resource replication. In this case, as the broadcasting probability *p* increases, the required bandwidth of super-peers increases and reaches to 7.7 108 bps for a value of 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. 10 shows an improvement in the required bandwidth in the presence of 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.

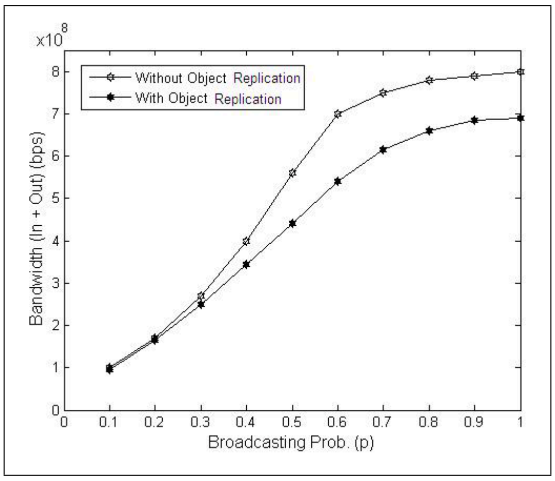

Figure 10: The effect of resource replication on total required bandwidth for various broadcasting probabilities

5 Related Works

Unstructured architectures are divided into three sub-classes [1]: 1) hybrid decentralized, 2) purely decentralized, and 3) partially centralized. In a hybrid decentralized P2P system, all peers connect to a central directory server that maintains a table for their IP address, connection bandwidth and other information, and another table that keeps the list of files that each peer holds. Upon receiving a request from a peer, the server searches for any matches in its index table and returns a list of peers who hold the matching file. Then a direct connection is established between the originating peer and the peers who hold the requested files to download them. Although the implementation of hybrid decentralized systems is easy, they suffer from vulnerabilities and attacks as well as scalability problems. Napster [19] is an example of such systems. Gnutella [3] is a well-known example of purely decentralized system.

A significant research toward proximity-aware resource location services in typical Gnutella-based unstructured P2P system has been done in [7, 8]. The proximity metric in this works is the 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. The analytical study on the impact of the proximity-aware methodology for making the P2P streaming more scalable has been presented in [9]. They have proposed a geometric model which maps the network topology from the space of discrete graph to the continuous geometric domain, meanwhile capturing the power-law property of the Internet. They found that, although random peer selection methods can maximally save the server resources, they introduce the maximum load to the network.

In systems where availability is not guaranteed, such as Gnutella [3], resource location techniques can afford to have loose guarantees [20]. Current search techniques in "loosely controlled" P2P systems are rather inefficient because they impose a heavy load on system as well as high response times. The main motivation for many researches in the area of P2P systems was early "loosely controlled" systems such as Gnutella, Freenet [14], Napster [19], and Morpheus [21]. Other resource location techniques for "loosely guaranteed" systems are mentioned in [20]. In the other proposed techniques which mentioned in [20] 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 [7]. CAN [2] is an example of systems with "strong guarantee" that employs search techniques. These systems can locate an object by its global identifier within a limited number of hops. It is concluded from the literature that selecting a resource locating methodology depends on the type of system which is planned. A complete literature survey relevant to search techniques is collected in [20] which interested readers can refer to it for more detail.

The resource locating techniques for partially centralized P2P networks are addressed in [15, 20]. They also evaluate some proposed query search broadcasting policies using Gnutella system and compare their performance with each other. The first query search policy evaluated by Yang et al. on Gnutella is called iterative deepening in which, a query is sent iteratively to more nodes until the query is answered. The second proposed technique, directed Breath First Search (DBFS) technique, forwards a limited set of nodes selected to maximize the probability that the query is answered. Their experimental analysis shows that if nodes are allowed to answer queries on behalf of other nodes, then the number of nodes that process a query will be reduced without decreasing the number of results. This very nice conclusion has formed our original motivation for designing distributed P2P caching protocol based on super-peers. The last technique which is investigated by them is called local indices technique in which, nodes maintain simple indices over other client's data. Queries are then processed by a smaller set of nodes. This is also quite similar to the concept of the super-peer nodes which we have adopted in our proposal.

The performance of hybrid P2P systems such as Napster is investigated by Yang and Garcia-Mollina in [17]. Morpheus [21] is another hybrid P2P system whose architecture is similar to Gnutella. Upon joining a new peer to the system, P2P network contacts a centralized server which then directs it to a super-peer. The authors of [17] study the behavior and performance of hybrid P2P systems and develop a probabilistic model to capture the query characteristic of such systems.

6 Conclusions

In this paper we have targeted the scalable proximity-aware location service for P2P systems. The proposed protocol provides a scalable distributed caching mechanism to find the peers who manage a given resource. The proposed mechanism enhances the mechanisms which have been proposed in previous researches by replicating objects based on latency distance metric, resulting in less aggregate load of the system. The simulation results showed that the use of probabilistic resource discovery service in P2P 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 number of contributing peers in the search process. Using the proposed mechanism yields at most 20% improvement in miss ratio in comparison with the case in which no resource replication is used. Also, in reasonable practical points of broadcasting probability, it yields about 30% reduction in hop ratio as well as 14% reduction in required bandwidth of the system.

Bibliography

[1] S. Androutsellis-Theotokis, D. Spinellis, *A Survey of Peer-to-Peer Content Distribution Technologies*, ACM Computing Surveys, vol. 36, no. 4, pp. 335-371, 2004

[2] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, *A Scalable Content Addressable Network*, Proc. ACM Sigcomm, August 2001.

[3] M. Ripeanu, I. Foster, A. Iamnitchi, *Mapping the Gnutella Network: Properties of Large-Scale Peer-to-Peer Systems and Implications for System Design*, IEEE Internet Computing, 6(1), February 2002.

[4] http://www.kazaa.com.

[5] Q. Lv, P. Cao, E. Cohen, K. Li, and S. Shenker, *Search and Replication in Unstructured Peer-to-Peer Networks*, the 16th ACM International Conference on Supercomputing (ICS'02). New York, NY., 2002.

[6] A. Crespo, H. Garcia-Molina, *Routing Indices for Peer-to-Peer Systems*, Proc. of Int. Conf. on Distributed Computing Systems, Vienna, Austria, 2002.

[7] D.A. Menascé, L. Kanchanapalli, *Probabilistic Scalable P2P Resource Location Services*, ACM Sigmetrics Performance Evaluation Rev., Volume 30, No. 2, pp. 48-58, 2002.

[8] D. Menascé, *Scalable P2P Search*, IEEE Internet Computing, Volume 7, No. 2, March/April 2003.

[9] L. Dai, Y. Cao, Y. Cui and Y. Xue, *On Scalability of Proximity-Aware Peer-to-Peer Streaming*, in Computer Communications, Elsevier, vol. 32, no 1, pp. 144-153, 2009.

[10] Y. Zhu, B. Li., *Overlay Networks with Linear Capacity Constraints*, IEEE Transactions on Parallel and Distributed Systems, 19 (2), pp. 159-173, February 2008.

[11] Y. Zhu, B. Li, K. Q. Pu., *Dynamic Multicast in Overlay Networks with Linear Capacity Constraints*, IEEE Transactions on Parallel and Distributed Systems, Vol. 20, No. 7, pp. 925-939, 2009.

[12] G.P. Jesi, A. Montresor, O. Babaoglu, *Proximity-Aware Superpeer Overlay Topologies*, IEEE Transactions on Network and Service Management, September 2007.

[13] F. Dabek, R. Cox, F. Kaashoek, and R. Morris., *VIVALDI: A Decentralized Network Coordinate System*, The SIGCOMM '04, Portland, Oregon, August 2004.

[14] I. Clarke, S. G. Miller, T. W. Hong, O. Sandberg, and B. Wiley, *Protecting Free Expression Online with Freenet*, IEEE Internet Computing , Volume 5, No. 1, pp. 40-49, 2002.

[15] B. Yang, H. Garcia-Molina, *Designing a Super-Peer Network*, Proc. Int'l Conf. Data Eng. (ICDE), pp. 49-63, Mar. 2003.

[16] C. Palmer, J. Steffan, *Generating network topologies that obey power laws*, The GLOBECOM 2000, November 2000.

[17] B. Yang, H. Garcia-Molina, *Comparing Hybrid Peer-to-Peer Systems*, Proc. 27th Int. Conf. on Very Large Data Bases, Rome, 2001.

[18] J.W. Song, K.S. Park, S.B. Yang, *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.

[19] http://www.napster.com.

[20] B. Yang, H. Garcia-Molina, *Improving Search in Peer-to-Peer Networks*, The 22nd International Conference on Distributed Computing Systems (ICDCS'02), Vienna, Austria, 2002.

[21] http://www.morpheus-os.com.