J Supercomput (2010) 51: 149–166
DOI 10.1007/s11227-009-0282-9

Sharable file searching in unstructured Peer-to-peer systems

Wenyu Qu · Wanlei Zhou · Masaru Kitsuregawa

Published online: 26 April 2009

Abstract The existing sharable file searching methods have at least one of the following disadvantages: (1) they are applicable only to certain topology patterns, (2) suffer single point failure, or (3) incur prohibitive maintenance cost. These drawbacks prevent their effective application in unstructured Peer-to-peer (P2P) systems (where the system topologies are changed time to time due to peers' frequently entering and leaving the systems), despite the considerable success of sharing file search in conventional peer-to-peer systems. Motivated by this, we develop several fully dynamic algorithms for searching sharing files in unstructured peer to peer systems. Our solutions can handle any topology pattern with small search time and computational overhead. We also present an in-depth analysis that provides valuable insight into the characteristics of alternative effective search strategies and leads to precision guarantees. Extensive experiments validate our theoretical findings and demonstrate the efficiency of our techniques in practice.

Keywords Peer-to-peer (P2P) · Unstructured · Remote destination · Search

W. Qu (✉)
School of Information Science and Technology, Dalian Maritime University, 1 Linghai Road, Dalian, 116026, China
e-mail: quwenyo@tkl.iis.u-tokyo.ac.jp

W. Zhou
School of Engineering and Information, Deakin University, 221 Burwood Hwy, Burwood, VIC 3125, Australia

M. Kitsuregawa
Institute of Industrial Science, The University of Tokyo, 4-6-1 Komaba, Meguro-ku, Tokyo, 153-8505, Japan

1 Introduction

Peer-to-peer (P2P) systems have become a popular medium to share a huge amount of data in that they distribute the main cost of sharing data across the peers in the system [5]; thus, enabling applications to scale without the need for powerful and expensive servers [9]. With the explosive growth of the World Wide Web (WWW), the amount of files sharable via networks has been dramatically increased and is still increasing at a high speed [6]. The main concern of the P2P system user-base has been moved from what kind of files is sharable on the Internet to how to find desirable files on the Internet. Efficient search is therefore of extremely importance to the popularization of P2P systems [3, 10].

Currently, all search on the Internet and large networks is carried out by dedicated and centralized search engines such as Yahoo and Google, and most Internet search engines maintain a very large centralized database which is updated by crawling the Internet and indexing web sites [7]. Developing such systems tends to be extremely expensive in terms of hardware, bandwidth requirements, and also the specialized algorithms and software that are necessary. When a query is received, the database replies with a list of web sites that are deemed to be in some way related to the original query [8]. This kind of indexing works is unsuitable for large dynamic P2P systems in spite of the fact that it works well on web sites that contain static information, as the search engine may not visit the web site regularly enough to be able to index new content. Some other works focus on data placement to improve search efficiency [6–8], which we will not take into concern in this paper.

We are interested in solutions that incur small computational overhead for processing each incoming query, as opposed to methods (e.g., the random walk reviewed in the next section) that have low amortized computational cost but poor worst-cast performance on response time. As elaborated in Sect. 2, the existing searching algorithms have at least one of the following disadvantages: (1) they rely on certain assumptions on client queries (e.g., the query has been performed before and its horizon does not change much), (2) they require considerable redundancy (e.g., replicated super peers for single point failure), or (3) they incur prohibitive maintenance cost (e.g., they must periodically scan the entire system by flooding). These problems prevent their effective deployment on P2P systems.

In this paper, we develop searching algorithms that are fully dynamic (supporting any sequence of peers' joining and leaving), efficient (processing each query with very low computational overhead and response latency), and harmonious (producing answers for client queries with consideration on alleviating traffic congestions). Specifically, for client request, our methods significantly improve the well-known flooding and random walk approaches in the presence of intensive dynamics and heavy workloads. For system performance, we propose the first search algorithm that not only makes inference on the known traffic information, but also balances the traffic load on each link in the system (previous solutions discuss only the known traffic information). In addition, we present an in-depth analysis that provides valuable insight into the characteristics of alternative solutions. Extensive experiments validate our theoretical findings and confirm the efficiency of the proposed techniques in practice.

The rest of this paper is organized as follows. Section 2 reviews previous work that is directly related to ours. Section 3 and Sect. 4 present algorithms for searching in conventional systems, and analyze their effectiveness on client queries. Section 5 focuses on searching on a hierarchical system structure. Section 6 contains the experimental results, and Sect. 7 summarizes our work and concludes this paper.

2 Related work

2.1 Breadth-first-search

Breadth-first search (BFS) is a search algorithm that begins at the root node and explores all the neighboring nodes. Then for each of those nearest nodes, it explores their unexplored neighbor nodes, and so on, until it finds the goal.

BFS is an uninformed search method that aims to expand and examine all nodes of a graph systematically in search of a solution. In other words, it exhaustively searches the entire graph without considering the goal until it finds it. It does not use a heuristic.

From the standpoint of the algorithm, all child nodes obtained by expanding a node are added to a FIFO queue. In typical implementations, nodes that have not yet been examined for their neighbors are placed in some container (such as a queue or linked list) called “open” and then once examined are placed in the container “closed.” The algorithm can be mainly described as follows: First, put the ending node (the root node) in the queue. Then pull a node from the beginning of the queue and examine it. If the searched element is found in this node, quit the search and return a result. Otherwise, push all the (so-far-unexamined) successors (the direct child nodes) of this node into the end of the queue, if there are any. If the queue is empty, every node on the graph has been examined—quit the search and return “not found.” This process will continue until all nodes are checked. Since all nodes discovered so far have to be saved, the space complexity of breadth-first search is $O(|V| + |E|)$ where $|V|$ is the number of nodes and $|E|$ the number of edges in the graph. Note: another way of saying this is that it is O(BM) where B is the maximum branching factor and M is the maximum path length of the tree. This immense demand for space is the reason why breadth-first search is impractical for larger problems.

If there is a solution breadth-first search will find it regardless of the kind of graph. However, if the graph is infinite and there is no solution breadth-first search will diverge. In the worst case, breadth-first search has to consider all paths to all possible nodes the time complexity of breadth-first search is $O(|V| + |E|)$ where $|V|$ is the number of nodes and $|E|$ the number of edges in the graph. The best case of this search is $O(1)$. It occurs when the node is found at first time.

For unit-step cost, breadth-first search is optimal. In general, breadth-first search is not optimal since it always returns the result with the fewest edges between the start node and the goal node. If the graph is a weighted graph and, therefore, has costs associated with each step, a goal next to the start does not have to be the cheapest goal available. This problem is solved by improving breadth-first search to uniform-cost search which considers the path costs. Nevertheless, if the graph is not weighted and, therefore, all step costs are equal, breadth-first search will find the nearest and the best solution.

2.2 Depth-first-search

Depth-first search (DFS) is an algorithm for traversing or searching a tree, tree structure, or graph. Intuitively, one starts at the root (selecting some node as the root in the graph case) and explores as far as possible along each branch before backtracking.

Formally, DFS is an uninformed search that progresses by expanding the first child node of the search tree that appears, and thus going deeper and deeper until a goal node is found, or until it hits a node that has no children. Then the search backtracks, returning to the most recent node it had not finished exploring. In a nonrecursive implementation, all freshly expanded nodes are added to a LIFO stack for exploration.

Space complexity of DFS is much lower than BFS (breadth-first search). It also lends itself much better to heuristic methods of choosing a likely-looking branch. Time complexity of both algorithms are proportional to the number of vertices plus the number of edges in the graphs they traverse $(O(|V|+|E|))$.

When searching large graphs that cannot be fully contained in memory, DFS suffers from nontermination when the length of a path in the search tree is infinite. The simple solution of "remember which nodes I have already seen" does not always work because there can be insufficient memory. This can be solved by maintaining an increasing limit on the depth of the tree, which is called iterative deepening depth-first search.

3 Directed depth-first-search

In this section, we present the DDFS algorithm, an extension of Sequence Search that not only makes reference on known traffic information but also balances the traffic load in the system. Section 3.1 discusses the algorithmic details of DDFS and Sect. 3.2 analyzes its characteristics.

3.1 Algorithm

Let $P=\{v_i, i=1,\ldots,n\}$ be a set of nodes in multidimensional space. A sharable file query retrieves the optimal route (e.g., the shortest or the fastest) in a segment $q_{ij}=[v_i,v_j]$. In particular, the result contains a set of $\langle v_k, O_{v_k}\rangle$ tuples, where v_k (for results) is a point of P, and O_{v_k} is the order of v_k in the route. Thus, $v_i \in P$ maintains an array vm_i with size M_i, whose value is determined by the amount of available memory. At any time, only a subset of valid files in vm_i belongs to the current sharable file set. Each element $vm_i[j]$ $(1 \le j \le M_i)$ is associated with a tag $vm_i[j].valid$ that equals TRUE if $vm_i[j]$ is valid, and FALSE, otherwise. Initially, every $vm_i.valid$ equals FALSE, indicating an empty sharable file set. Specifically, for each share-command (let t be the file being shared), a random integer x is generated in the range $[1, n.s_i]$, where $n.s_i$ is the total number of share processed on v_i so far. If $x \le M_i$, t is placed at the xth position $vm_i[x]$ of vm_i, and set $vm_i[x].valid$ to TRUE. Otherwise, $(x > M)$, no further action is taken and t is ignored. To handle an unshare-command $\{unS, id\}$, on the other hand, it will be checked whether the tuple with the requested id belongs to the sharable file set, namely, whether there exists a number

x $(1 \leq x \leq M_i)$ such that the id of $vm_i[x]$ equals id, and $vm_i[x].valid = \text{TRUE}$. If s is found, unsharing is completed by simply modifying $vm_i[x].valid$ to FALSE, without affecting the other elements in vm_i. In order to efficiently retrieve files with particular ids, an B-tree index [2, 12] $I(vs_i)$ on the file ids is created for each $v_i \in P$ $(i = 1, \ldots, n)$. The size of $I(vs_i)$ is $1/d$ of the occupied space where d is the number of attributes of a tuple, since $I(vs_i)$ contains only the ids of the tuples.

3.2 Analysis

In a mobile agent-based P2P system, there will be a number of mobile agents running in the system which will certainly consume a certain amount of computational resource on each peer in the system. The purpose of using the LNSS is to help traveling SeA select neighboring peers to move to. The main idea of the LNSS is that when a SeA makes its selection, it should consider about not only the traffic information collected by InAs and OuAs, but also the load balance of the system.

Let p_{ij} be the probability that a SeA on peer v_i selects the peer v_j from its neighboring set $NB(i)$ to move. Suppose that originally, each peer in the P2P system knows nothing about the traffic situation of the system. Therefore, for a SeA on peer v_i, every peer v_j in $NB(i)$ has an equal probability $p_{ij} = 1/|NB(i)|$ to be selected as the target to move to. Obviously, this probability distribution is unbiased to every peer in $NB(i)$ and can mostly balance the traffic load in the current situation. This uniform probability distribution of agents' neighboring peer selection will be updated with time going. After a certain period, the probability distribution of neighboring peers to be selected by a mobile agent as its target will be affected by collected traffic information. The new probability distribution should satisfy two constraints:

1. It makes inference on all the known traffic information.
2. It is unbiased. That is, the probability should mostly balance the traffic cost on each link.

In the following, we will mathematically model and solve the probability distribution that both makes inference on the known information and approximates to the unbiased (uniform) distribution.

Objective 1: making inference on known information

The effect of the known traffic information on an agent's migrating decision making is expressed by the function $f_{ji}(x)$ which is defined as the traffic cost of an agent moving from v_j to v_i where $i \in NB(j)$. Thus, the maximum traffic cost function, $f_{\max}^{(j)}(x)$, of an agent moving out from v_j can be defined as

$$f_{\max}^{(j)}(x) \equiv \max_{i \in NB(j)} \{f_{ji}(x)\}, \tag{1}$$

which is also decided by collected traffic information. Therefore, the traffic cost balance on each link can be mathematically modeled by minimizing the maximum cost to neighboring peers which is a min-max problem as follows:

$$\min_{x \in R^n} f_{\max}^{(j)}(x). \tag{2}$$

Without lose of generality, we assume that functions $f_{ji}(i \in NB(j))$ are differentiable. Obviously, the maximum value function $f_{\max}^{(j)}(x)$ is an nondifferentiable function.

Let us look at the following Lagrange function:

$$\ell_j(x, p_j) = \sum_{i \in NB(j)} p_{ji} f_{ji}(x) \quad \forall x \in R^n,\ p_j \in \Delta_j, \tag{3}$$

where $p_j = (p_{j1}, p_{j2}, \ldots, p_{j,|NB(j)|})^T$ is the vector of Lagrange multiplier, Δ_j is a simplex set defined as follows:

$$\Delta_j \equiv \left\{ p_j \in R^{|NB(j)|} \,\middle|\, \sum_{i \in NB(j)} p_{ji} = 1, p_{ji} \geq 0 \right\}. \tag{4}$$

It is easy to see that no matter which value the multiplier vector p_j is chosen, the value of the Lagrange function $\ell_j(x, p_j)$ is less than or equal to the maximum value function $f_{\max}^{(j)}(x)$, i.e.,

$$\ell_j(x, p_j) \leq f_{\max}^{(j)}(x). \tag{5}$$

From the definition of Lagrange function $\ell_j(x, p_j)$, we have the following lemma.

Lemma 1 *The maximum value function $f_{\max}^{(j)}(x)$, defined in* (1), *can be expressed as follows*:

$$f_{\max}^{(j)}(x) = \sup_{p_j \in \Delta_j} \ell_j(x, p_j) = \max_{p_j \in \Delta_j} \ell_j(x, p_j). \tag{6}$$

Proof $\forall x \in R^n$ and $p_j \in \Delta_j$, it is easy to see that

$$\sum_{i \in NB(j)} p_{ji} f_{ji}(x) \leq f_{\max}^{(j)}(x). \tag{7}$$

Therefore, we have

$$\sup_{p_j \in \Delta_j} \ell_j(x, p_j) \leq f_{\max}^{(j)}(x). \tag{8}$$

Let $I_{\max}^{(j)}(x)$ be the indicator set of element functions $f_{ji}(x)(i \in NB(j))$ that is equal to the maximum value function $f_{\max}^{(j)}$ at point x, i.e.,

$$I_{\max}^{(j)}(x) := \{k | f_{jk}(x) = f_{\max}^{(j)}(x)\}. \tag{9}$$

If $k \in I_{\max}^{(j)}(x)$, then $\forall x \in R^n$ and $p_j \in \Delta_j$, we have

$$\sup_{p_j \in \Delta_j} \ell_j(x, p_j) \geq \sum_{i \in NB(j)} \bar{p}_{ji} f_{ji}(x) = f_{\max}^{(j)}(x)b, \tag{10}$$

where

$$\bar{p}_{ji} = \begin{cases} 1, & i = k; \\ 0, & i \neq k. \end{cases} \tag{11}$$

From (8) and (10), the first equality in (6) holds. Consider that Δ_j is a tight set and $\ell_j(x, p_j)$ is a continuous function on p_j, the second equality in (6) also holds. □

From Lemma 1, it can be seen that since the Lagrange function $\ell_j(x, p_j)$ is a linear function on variable p_j, (6) has multisolutions. Therefore, function $f_{\max}^{(j)}(x)$ defined in (1) is a nondifferentiable function.

From Lemma 1, it can also be seen that since the multiplier vector p_j is limited inside the simplex Δ_j, the Lagrange function $\ell_j(x, p_j)$ can be interpreted as a convex combination of all element functions $f_{ji}(x)$ $(i \in NB(j))$ and multipliers p_{ji} are the combination coefficients. Therefore, (2) can be solved by solving an equivalent problem of finding a set of value of p_{ji}, $(i \in NB(j))$ such that the Lagrange function $\ell_j(x, p_j)$ approximates to the maximum value function, i.e., to find the optimal combination $\hat{p}_j$ from all combinations that satisfies (4) such that (5) becomes the following equality:

$$\ell_j(x, \hat{p}_j) = \sum_{i \in NB(j)} \hat{p}_{ji} f_{ji}(x) = f_{\max}^{(j)}(x). \tag{12}$$

On the other hand, if the Lagrange multipliers p_{ji} $(i \in NB(j))$, also called the combination coefficients, is endued with a probability sense, i.e., describing them as the corresponding probabilities such that the element function $f_{ji}(x)$ becomes the maximum value function $f_{\max}^{(j)}(x)$, then from the concept of probability, (2) can be transferred into a maximized problem of finding the optimal probability distribution that satisfies:

$$\begin{cases} \max\limits_{p_j \in R^{|NB(j)|}} & \ell_j(x, p_j) \\ \text{s.t.} & \sum\limits_{i \in NB(j)} p_{ji} = 1; \\ & p_{ji} \geq 0, \quad i \in NB(j). \end{cases} \tag{13}$$

Objective 2: unbiased selection

Suppose that there are a set of possible peers in $NB(i)$ whose probabilities to be selected are $p_{i1}, p_{i2}, \ldots, p_{in}$. These probabilities are known but that is all we know concerning which peer to be selected. Can we find a measure of how much "choice" is involved in the selection of the peer or of how uncertain it is of the outcome? It is proved in [11] that the "entropy" function $H = -k \sum_{j=1}^{n} p_{ij} \ln p_{ij}$ is the only measure that satisfies the following properties:

1. H is continuous on p_{ij}.
2. If all p_{ij} are equal, i.e., $p_{ij} = 1/n$, then H is a monotonically increasing function of n.

3. If a choice is broken down into two successive choices, the original H is the weighted sum of the individual values of H.

Here, k is a positive constant decided by measurement units. Usually, k is set to be 1. "Entropy" is a measurement of the degree of uncertainty, and the greater the entropy's value, the less known information.

In many probabilistic executions, the probability distribution p_{ij} cannot foreseen; thus, the entropy cannot be calculated. In [4], Jaynes claimed "in making inference on the basis of partial information we must use that probability distribution which has maximum entropy subject to whatever is known. This is the only unbiased assignment we can make; to use any other would amount to arbitrary assumption of information which by hypothesis we do not have." This is the famous "maximum entropy theory." The maximum entropy theory can be mathematically expressed as follows:

$$\begin{cases} \max\limits_{p_i \in R^{|NB(i)|}} & H(p_{ij}) = -\sum\limits_{j \in NB(i)} p_{ij} \ln p_{ij} \\ \text{s.t.} & \sum\limits_{j \in NB(i)} p_{ij} = 1; \\ & p_{ij} \geq 0, \quad j \in NB(i). \end{cases} \tag{14}$$

where $p_i = \{p_{ij}, j \in NB(i)\}^T$.

In [13], Templeman et al. applied the maximum entropy theory to solve optimization problems in which the objective function is unanimously approximated by a smooth one. By solving the resulting problem, an approximate solution of the original problem can be obtained. The purpose of deploying maximum entropy theory in agents' search process in our model is to find a probability distribution that both satisfies the known routing information and mostly approximate to the unbiased (uniform) distribution.

Multiobjective problem

According to the analysis above, there are two objective functions to be maximized:

1. To maximize the Lagrange function through selecting the optimal multiplier vector;
2. To maximize the entropy function by finding an unbiased probability distribution.

Therefore, the problem to be solved is a multiobjective problem as follows:

$$\max_{p_j \in R^{|NB(j)|}} \left\{\ell_j(x, p_j), H(p_j)\right\} \tag{15}$$

which can be transformed into a single-objective problem by the weighting coefficient method as follows:

$$\max_{p_j} L_\theta^{(j)}(x, p_j) = \sum_{i \in NB(j)} \ell_j(x, p_j) + \frac{1}{\theta} H(p_j), \tag{16}$$

where $\theta \geq 0$ is a weighting coefficient. Obviously, when θ is small, the second item of the objective function $L_\theta^{(j)}(x, p_j)$ is dominative. Then the gained probability distribution mainly reflects the requirement of unbiased distribution. With the increase of θ's value, the effect of the first item increases; thus, maximizing the Lagrange function is the dominative objective.

Unfortunate, it is hard to get the solution of function $L_\theta^{(j)}(x, p_j)$ as it is multiple-objective and nondifferentiable.

By defining a function $F_\theta^{(j)}(x)$ as

$$F_\theta^{(j)}(x) = \sup_{p_j \in \Delta_j} L_\theta^{(j)}(x, p_j), \tag{17}$$

we have the following theorem.

Theorem 1 *The function $F_\theta^{(j)}(x)$ defined by (17) is differentiable and uniformly approximate to function $f_{\max}^{(j)}(x)$ on the whole space R^n.*

Since function $F_\theta^{(j)}(x)$ uniformly converges to the objective function $f_{\max}^{(j)}(x)$, the optimal solution, $\hat{p}_j(x)$, of (16) can be easily derived from applying the $K - T$ condition as follows:

$$\hat{p}_{ji}(x) = \frac{\exp\{\theta f_{ji}(x)\}}{\sum_{l \in NB(j)} \exp\{\theta f_{jl}(x)\}}, \quad i \in NB(j). \tag{18}$$

where p_{ji} is the probability that an agent on peer v_i migrates to peer v_j, $\theta \geq 0$ is a weight coefficient defined according to the effect of the known traffic state of the system. This probability is the only unbiased probability distribution for a mobile agent on peer v_i to select a neighboring peer from $NB(i)$ and move to which makes inference on all known traffic information.

Substitute the analytical solution $\hat{p}_j(x)$ of the multiplier p_j in the objective function of (17) and we have

$$\begin{aligned} \hat{F}_\theta^{(j)}(x) &= L_\theta^{(j)}\big(x, \hat{p}_j(x)\big) \\ &= \frac{1}{\theta} \ln\left\{ \sum_{i \in NB(j)} \exp[\theta f_{ji}(x)] \right\}, \end{aligned} \tag{19}$$

which is the maximum traffic cost for an agent migrating from peer v_j to a neighboring peer.

4 Controlled flooding

In this section, we develop an alternative searching approach CF, which is motivated by the flooding reviewed in Sect. 2, but improves its updating performance considerably.

This algorithm using mobile agents to update available data information and traffic information inside a peer group. Each peer periodically generates an inner agent (InA) and dispatches the InA to other peers in the peer group (or groups when the peer belongs to multiple peer groups) where it resides. The peer is known as the "home peer (Hp)" for the InA and all future replicas of the InA. The InA has the newest data information of its Hp and collects the newest traffic information during its journey, including the bandwidth of a link, the transmission delay of a link, etc. Without losses of generality, we assume that there are n items each of which is denoted by a random variable x_i $(i = 1, 2, \ldots, n)$, $x = (x_1, x_2, \ldots, x_n)^T$ is a column vector with n entries, and $f_{ji}(x)$ is the traffic cost function from peer v_j to v_i.

The working mechanism of the InA is as follows: Before moving out from the Hp, the InA broadcasts replicas to each group mate that has a direct link to its Hp. Once an InA or one of its replica arrives at a peer, the InA updates the local resource information by the data information of its Hp, collects local traffic information $x = (x_1, x_2, \ldots, x_n)^T$, marks the peer "updated," and the link from which it enters the peer as "upstream" link. The InA randomly selects one of the links (if any) other than the upstream link and moves to the peer directly connected to the link. Before moving to the selected peer, the InA spawns one InA for each of the links except the upstream link and the link taken by itself, and injects the newly spawned InAs into those peers directly connected to these links. If two InAs from the same Hp arrive at one single peer, then the second InA terminates itself. This process continues until the InA arrives at a peer being "updated" or finds the peer it updated has no other link except the upstream link. Then the InA moves along the upstream links back to the Hp, updates the traffic information on the peers along its path, reports the updating results to the Hp, and terminates itself. If an InA reaches a peer updated by other InA from the same Hp before it updates any peer, it terminates itself immediately. The self-terminating mechanism keeps the information up to date, and prevents peers that form cycles from having to deal repeatedly with InAs from the same Hp. In this way, the InAs spawn themselves and flood the peer group as fast as possible, and the flooding is controlled by checking the status of the peers. When an Hp receives the report of an InA, it updates its routing table and stores the updating information in its resource database. To determine whether all group mates have been updated, the Hp checks the status of each group mate in its database to see if there is any group mate that has not been updated by an InA. If no, the complete updating process is finished. To do this, the Hp need not check all the group mates. A counter can be maintained to count the number of peers in the database that has not been updated by an InA. The Hp only needs to update the counter each time when it receives new reports. Thus, each peer in the peer group maintains current resource information and routing information of the peer group/groups.

5 Hybrid search methods on hierarchical system structure

Based on the techniques developed in the previous sections, we proceed to discuss the following.

Section 5.1 presents a hierarchical system structure that solves the problem by reducing the size of the system. Then Sect. 5.2 proposes a hybrid search approach with considerably less response time and computation overhead.

5.1 Hierarchical system structure

P2P systems are overlay networks built on top of the Internet where nodes in the overlay are connected by virtual or logical links in the underlying network and maintain information abut a set of other nodes in the P2P layer. They are dynamic in nature in that nodes/peers in a P2P system join or leave frequently. These peers form a virtual overlay network on top of the Internet. Each link in a P2P overlay corresponds to a sequence of physical links in the underlying network.

Most existing search techniques are forwarding-based. Starting with the requesting peers, a query is forwarded (or routed) peer to peer until the peer which has the desired data (or a pointer to the desired data) is reached. To forward query messages, each peer must keep information about some other peers called neighbors. The information of these neighbors constitutes the routing table of a peer. With the current size of P2P systems and the rate at which it is growing, the information updating of a P2P system becomes more and more difficult. There is a need for designing new algorithms for search in P2P systems.

In this paper, we focus on the search method in wide distributed large scale P2P systems where hierarchical organization is adopted. Let $G = \{V, E\}$ be a graph corresponding to a fixed system, where $V = \{v_1, v_2, \ldots\}$ is the set of vertices (hosts) and E is the set of edges. We assume that the topology of a system is a connected graph in order to ensure that communication is able to be made between any two hosts. The system in our model is organized into peer groups. Each peer group consists of a certain number of peers with each peer inside the peer group equally behaving as a representation peer connected to other peers in the P2P system. The logical system infrastructure is defined by the collection of peer groups which are connected by an arbitrary topology. Inside this infrastructure, all the peer groups are at the same hierarchical level and "flat" routing is performed among them. In this way, the computational complexity of the routing problem is much reduced, while the complexity of the design and management of the routing protocol is much increased.

A peer group may consist of multiple peers or only one single peer. Peers in the same peer group are "group mates" to each other. For a specified peer v_i, $nb(i)$ is the set of group mates in the peer group of v_i and $|nb(i)|$ is the number of group mates in $nb(i)$. The size of a peer group is decided by the connectivity of the group topology. Meanwhile, one single peer may belong to several peer groups so that it may have different group mates comparing with any of its group mates. Note that a peer in a peer group is also a node in the P2P system. If there is a direct link between two peers that belong to different peer groups, we call these two peers are "neighboring peers." Let $NB(i)$ denote the set of neighboring peers of v_i and $|NB(i)|$ is the number of peers in $NB(i)$.

5.2 Hierarchical search approach

The desired features of search algorithms in P2P systems include high-quality query results, minimal routing state maintained per peer, high routing efficiency, load balance, resilience to peer failures, and support of complex queries.

For efficiently addressing these features, we propose a model that applies hierarchical search approach. To collect and disseminate routing information in different territories, there are three kinds of agents employed in our model, namely inner agents, outer agents, and search agents. For each kind of agents, there is a special search algorithm that takes advantage of the specialty of their activity territory.

This algorithm uses mobile agents to update data and traffic information among peer groups. Each peer periodically generates an outer agent (OuA) and sends the OuA to neighboring peers outside its peer group/groups. The peer is known as the "source peer (Sp)" for the OuA and all future replicas of the OuA. The OuA has the newest data information of its Sp and collects the newest traffic information between the Sp and the neighboring peers, including the bandwidth of a link, the transmission delay of a link, etc. Similar to that in a controlled flooding algorithm, there are n items relatively each of which is a random variable x_i $(i = 1, 2, \dots, n)$, $x = (x_1, x_2, \dots, x_n)^T$ is a column vector with n entries, and $f_{ji}(x)$ is the traffic cost function from peer v_j to v_i. Before moving out from the Sp, the OuA broadcasts replicas to each neighboring peer that has a direct link to its Sp. The OuA or any of its replicas collects local traffic information $x = (x_1, x_2, \dots, x_n)^T$ of the link it passes, updates the local data information by the data information of its Sp, and returns back to the Sp with the data information of the neighboring peer. The Sp updates local data information and traffic information by the information collected by the OuAs. Thus, a peer in one peer group has the data information of its group mates, its neighboring peers, and group mates of its neighboring peers.

6 Experimental studies

The goal of our experimental studies was to gain a better understanding of the working mechanisms and performance of our algorithm. We used the open simulator OMNet++ [14] as our simulation platform which contains several models for unstructured peer-to-peer protocols. Peers are organized in an overlay network. Each peer has a set of group mates and a set of neighboring peers. Links are directed and traffic can flow in both directions on the links. We consider a peer-to-peer network made of 10,000 peers, which corresponds to a middle-size P2P network. The number of links of each peer is between 3 to 20. The links between neighboring peers are chosen randomly, and links between group mates adapt dynamically, which indicates the join/leave of a peer. The link bandwidth is 6 Mbits/sec and the propagation delay is from 2 to 5 millisecond. The size of data packet is 512 bytes and the size of an agent is 48 bytes. Traffic is defined in terms of open sessions. A session is defined between two nodes and it remains active until a certain amount of data are transferred at a given rate. Each session is characterized completely by session size, the generation rate of sessions (GRS), and the sending rate of the agents (SRA). In the

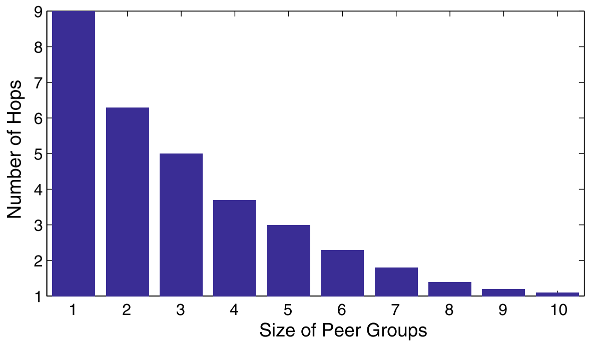


Fig. 1 The relationship between the number of hops and the size of peer groups

simulation, session size is set to be 2 Mbits. Originally, 1,500 peers are randomly chosen to construct peer groups. These chosen peers cannot be group mates to each other. If an unchosen peer could be reached in 3 hops by a chosen peer, it will join the chosen peer's group. Each peer has 3 tables to record its own data, group mates data, and neighboring peers data, respectively. One hundred keywords and 100,000 distinct files are created and each of files are assigned exactly one keyword following uniform distribution. Each peer is assigned uniformly at random a storage to hold 10 to 1000 files. After peer groups have been established, inner agents and outer agents are sent out to collect and disseminate data information. We implemented our algorithm (OA) together with OSPF [1] and algorithm in [15] (shorted by "Ant") to evaluate and compare the efficiency of these three algorithms.

In each search round of our simulation, 1,000 randomly selected peers issue requests. To generate a request/search agent, a peer first selects one keyword and selects a file by that keyword that it does not hold. The peer then issues a request for that file. Each request is assigned a hop limitation (HL). When a peer receives a request, it decreases the HL value and propagates the request further. The number of hops that a request need to reach the first peer that has the requested file is a measure of the response time. It is helpful for choosing suitable HL values to achieve a good success rate without overloading the network. We compute the average number of hops over all successful requests issued during each round. Figure 1 shows the relationship between the number of hops and the size of peer groups. From the figure, we can see that with the size of peer groups increase, the needed number of hops decreases. We can also see that when the size of peer group reaches a certain scope, the needed number of hops decreases slow when the size of peer groups increases.

Two parameters are used in our comparison, throughput and search time. The throughput is defined as the number of requests/search agents forwarded per second, which shows the delivering ability of the algorithms. The search time is defined as the time interval from the creation of a request/search agent to its arrival at the destination. It indicates the quality of paths selecting.

Originally, there is a normal load of GRS = 2.7 second and SRA = 0.3 second. From 500 seconds to 1,000 seconds, all peers issued requests of a selected file with SRA = 0.05 second. Figure 2 and Fig. 3 show the comparison results of the average throughput and the average search time between OSPF, Ant, and OA.

It can be seen that both OA and Ant are able to cope with the transient overload. OSPF shows the poorest performance. It can also be seen that the average search

Fig. 2 The average throughput when all peers issued requests of a selected file from 500 second to 1000 second

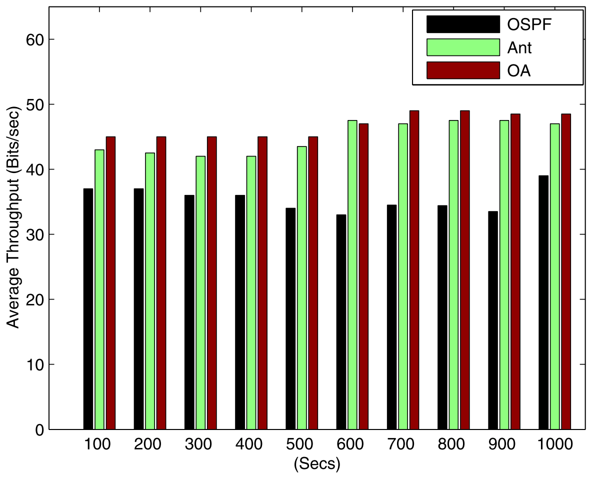
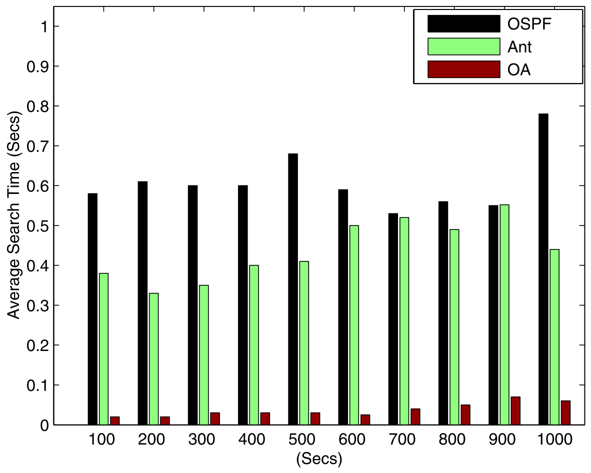


Fig. 3 The average search time when all peers issued requests of a selected file from 500 second to 1000 second



time for OA is less than 0.1 second as compared to 0.5 second for Ant. Again, OSPF shows the poorest performance.

We also compared the success rate, i.e., the ratio of number of real results to the number of total results, among these three algorithms which states the proportion of agents arrived at destinations correctly. Table 1 shows the results of the success rate. From the simulation results, we can see that OA achieves a similar performance to Ant, which is much better than OSPF.

To check the fault-tolerant ability of OA, we use the Japanese Internet Backbone (NTTNET) as the simulation framework (see Fig. 4). It is a 57 node, 162 bidirectional links network.

Figure 5 and Fig. 6 show the results of the comparison between OA and Ant in which node 21 crashed at 300 second, node 40 crashed at 500 second, and both of them were repaired at 800 second. Here, GRS = 4.7 second and SRA = 0.05. The

Table 1 The comparison of success rate

Parameters		Success rate (%)		
GRS(Sec)	SRA(Sec)	OSPF	Ant	OA
4.5	0.5	83.21	96.85	99.99
2.5	0.5	82.46	97.31	99.99
1.5	0.5	80.13	97.24	99.99
2.5	0.05	83.94	95.94	99.68

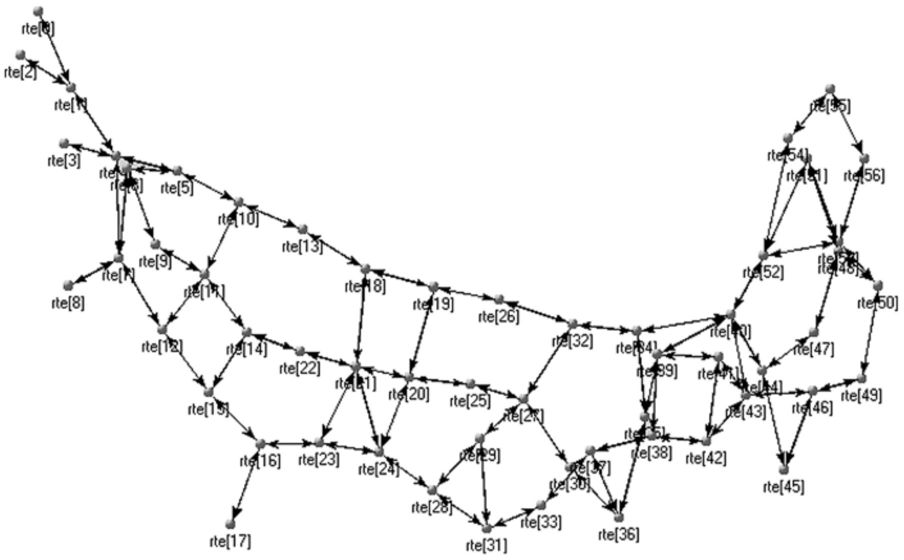


Fig. 4 The Japanese Backbone (NTTNET)

purpose of this experiment was to analyze the fault tolerant behavior of OA. The GRS and SRA is selected to ensure that no agents are terminated because of the congestion. Based on our experimental results, OA is able to deliver 97% of agents as compared to 89% by Ant. From the figure, we can see that OA has a superior throughput and lesser search time. But once node 40 crashes, the search time of OA increases because of higher load at node 43. From Fig. 4, it is obvious that the only path to the upper part of the network is via node 43 once node 40 crashed. Since OA is able to deliver more agents, the queue length at node 43 increased and this led to relatively longer search time as compared to Ant. On the contrary, although node 21 is critical, but in case of its crash still multiple paths exist to the middle and upper part of the topology.

OA does not need any global information such as the structure of the topology and cost of links among peers. It cannot only balance the local traffic flow, but also enhance fault tolerance. Compared to Ant, this performance enhancement is achieved with a little more traffic burden incurred by the inner agents inside peer groups, but this overhead is only a small segment to the total traffic in the network.

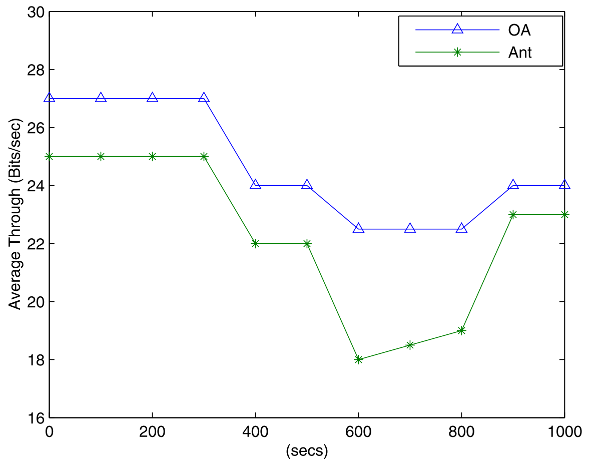


Fig. 5 The average throughput node 21 is down at 300 and node 40 is down at 500 and both repaired at 800

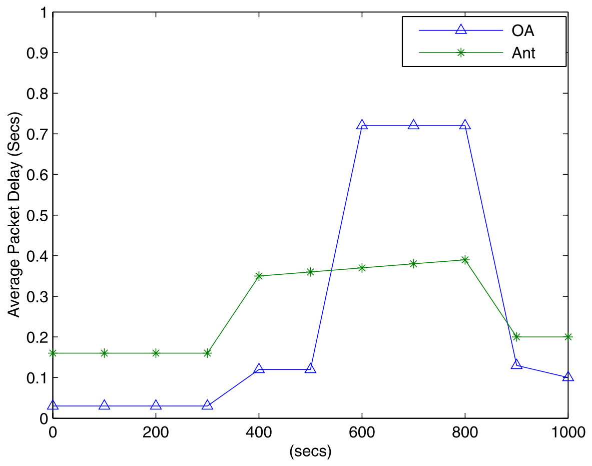


Fig. 6 The average search time when node 21 is down at 300 and node 40 is down at 500 and both repaired at 800

7 Conclusion

In this paper, we proposed a mobile agent-based execution model for use to search in peer-to-peer networks in which the traffic congestion is considered. Our model divided a P2P system into peer groups. Peers in a peer group act equally so that the total system performance will not be affected when a peer fails or leaves. Thus, the fault-tolerant ability is greatly enhanced. There are three kinds of agents employed in our models: inner agents, outer agents, and search agents. Inner agents respond to information updating inside a peer group by using the controlled flooding algorithm, outer agents respond to information updating among peer groups by the one-step flooding algorithm, and search agents respond to locate queried data in the system by using of the directed k-walker random walk algorithm. The traffic load is greatly decreased compared to the flooding algorithms and the possibility that a peer be neglected is greatly decreased compared to the random walk algorithm. A traffic cost function is defined for each link based on known traffic information of the network and a probability distribution is given by the local-optimal neighboring search strategy for search agents selecting a neighboring node and move to. We proved that this probability distribution is the only unbiased distribution that only makes inference on the known traffic information but also balances the traffic load. Extensive experiments verified our analytical results and showed that our model results in significant performance improvements compared to existing models.

Acknowledgements This work was co-supported by National Natural Science Foundation of China (90718030 and 90818002), ARC Supported Discovery Project (DP0773264), and Scientific Research Foundation for the Returned Overseas Chinese Scholars.

References

1. Dijkstra E (1959) A note on two problems in connection with graphs. Numer Math 269–271
2. Graefe G, Larson P-A (2001) B-tree indexes and CPU cashes. In: ICDE2001 conference proceeding, 2001, pp 349–358
3. Jagadish HV, Ooi BC, Vu QH, Zhang R, Zhou A (2006) Vbi-tree: a Peer-to-peer framework for supporting multi-dimensional indexing schemes. In: ICDE 2006 conference proceedings, 2006, p 34
4. Joynes ET (1957) Information theory and statistical mechanics. Phys Rev 108:171–190
5. Koloniari G, Pitoura E (2005) Peer-to-peer management of XML data: issues and research challenges. ACM SIGMOD Record 34(2):6–17
6. Li K, Shen H (2005) Coordinated en-route multimedia object caching in transcoding proxies for tree networks. ACM Trans Multimed Comput Commun Appl (TOMCAPP) 5(3):289–314
7. Li K, Shen H, Chin F, Zhang W (2007) Multimedia object placement for transparent data replication. IEEE Trans Parallel Distrib Syst (TPDS) 18(2):212–224
8. Li K, Shen H, Chin F, Zheng S (2005) Optimal methods for coordinated En-route Web caching for tree networks. ACM Trans Internet Technol (TOIT) 5(3):480–507
9. Ng WS, Ooi BC, Tan K (2002) Bestpeer: a self-configurable Peer-to-peer system. In: ICDE 2002 conference proceedings, 2002, p 272
10. Risson J, Moors T (2007) Survey of research towards robust Peer-to-peer networks: search methods. Comput Netw 5(7):3485–3521
11. Shannon CE (1948) A mathematical theory of communication. Bell Syst Tech J 27(3):379–428
12. Tao Y, Lian X, Papadias D, Hadjieleffheriou M (2007) Random sampling for continuous streams with arbitrary updates. IEEE Trans Knowl Data Eng 19(1):96–110
13. Templeman AB, Li X (1987) A maximum entropy approach to constrained non-linear programming. Eng Optim 12(3):191–205
14. Varga A (2006) OMNeT++: discrete event simulation system: user manual. http://www.omnetpp.org/. July 2006
15. Yang K, Wu C, Ho J (2006) AntSearch: an Ant search algorithm in unstructured Peer-to-peer networks. IEICE Trans Commun E89-B(9):2300–2308

Wenyu Qu received her Bachelor and Master degrees both from Dalian University of Technology, China, in 1994 and 1997, and her Doctoral Degree from Japan Advanced Institute of Science and Technology in 2006. She is currently Professor at the College of Computer Science and Technology, Dalian Maritime University, China. Wenyu Qu's research interests include mobile agent-based technology, distributed computing, computer networks, and grid computing. Wenyu Qu has published more than 40 technical papers in international journals and conferences. She is on the committee board for a couple of international conferences.

Wanlei Zhou received his Ph.D. degree from The Australian National University, Canberra, Australia, in October 1991. He also received the D.Sc. degree from Deakin University, Victoria, Australia, in 2002. He is currently the Chair Professor of Information Technology and the Head of School of Information Technology, Deakin University, Melbourne, Australia. Before joining Deakin University, Professor Zhou worked in both industry and academia, including University of Electronic Science and Technology of China; HP R&D Labs at Massachusetts, USA; Monash University in Melbourne, Australia; and National University of Singapore. His research interests include distributed and parallel systems, network security, mobile computing, bioinformatics and e-learning. Professor Zhou has received a number of National Competitive Research Grants from various government agencies including the Australian Research Council. Professor Zhou has published more than 200 papers in refereed international journals and refereed international conference proceedings, edited 5 books and authored 1 book. Professor Zhou has been involved in organizing many international conferences as General Chair, Steering Chair, PC Chair, Session Chair, Publication Chair, and PC Member, and have presented a number of keynote speeches in international conferences.

Masaru Kitsuregawa received a Ph.D. degree from the University of Tokyo in 1983. He is currently Professor and Director of the Center for Information Fusion at the Institute of Industrial Science of the University of Tokyo. His current research interests include database engineering, web mining, parallel computer architecture, parallel database processing/data mining, storage system architecture, digital earth, and transaction processing. He had been a VLDB Trustee and served as the General Chair of ICDE 2005 at Tokyo. He is currently an Asian Coordinator of the IEEE Technical Committee on Data Engineering, and a Steering Committee Member of PAKDD and WAIM. In Japan, he chaired the data engineering technical group of IEICE, and served as ACM SIGMOD Japan Chapter Chair. He is currently Adviser to the Storage Networking Industry Association Japan and Director of the Database Society of Japan. He is a member of ACM and the IEEE Computer Society, and a fellow of IEICE and the Information Processing Society of Japan.