A Quorum-Based Protocol for Searching Objects in Peer-to-peer Networks

Ken Miura, Taro Tagawa, and Hirotsugu Kakugawa, Member, IEEE

Abstract—Peer-to-peer (P2P) system is an overlay network of peer computers without centralized servers, and many applications have been developed for such networks such as file sharing systems. Because a set of peers dynamically changes, design and verification of efficient protocols is a challenging task.

In this paper, we consider an object searching problem under a resource model such that there are some replicas in a system and the lower bound of the ratio \( p = n'/n \) is known in advance, where \( n' \) is a lower bound of the number of peers that hold original or replica for any object type, and \( n \) is the total number of peers. In addition, we consider object searching with probabilistic success, i.e., for each object search, object must be found with at least probability \( 0 < \sigma < 1 \). To solve such a problem efficiently, we propose a new communication structure, named probabilistic weak quorum systems (PWQS), which is an extension of coterie. Then, we propose a fault-tolerant protocol for searching objects in P2P system.

In our method, each peer does not maintain global information such as the set of all peers and a logical topology with global consistency. In our protocol, each peer communicates only a small part of a peer set, and thus, our protocol is adaptive for huge scale P2P network.

Index Terms—D.4.7.b - Distributed systems, E.1.b - Distributed data structures, G.4.g - Reliability and robustness, G.2.1.a - Combinatorial algorithms, J.8.d - Distributed file systems.

I. INTRODUCTION

P2P (peer-to-peer) network is an overlay network of peer computers without centralized servers [1]. Applications of P2P systems can be classified into three categories: (1) file sharing, (2) high performance distributed computing, and (3) collaborative works with users. In this paper, we focus on file sharing P2P systems.

File exchange is an interesting application of P2P systems. A user who joins a P2P file exchange system can exchange his/her files with other peers in the system. Because there is no centralized server that maintains a list of files and their location in network (i.e., hostname that holds a file), a user program executed at a peer must find the location of a target file by communicating with other peers that are taking part in the P2P system at the moment.

Different from a system with centralized servers, in P2P systems, member peers dynamically changes, because each peer that may hold target object can join and leave the system at any time. Difficulty of designing P2P systems comes from such dynamic nature. Managing information of a set of peers and shared files up-to-date with small overhead is an important problem in design of P2P systems.

We abstract manual, data sheet, and music file in P2P network, etc. as object. Below, we use terminology ‘object’ instead of ‘file’.

A. Our contribution

In this paper, we propose a problem to find a target object probabilistically. This is motivated from an observation that, a replica of a popular document or music file is likely to exist on some peers in a P2P system. If we were to find a target object deterministically, each peer may gather global information of object location or member peers, which may be less efficient. By the probabilistic requirement, objects may be efficiently searched without global information.

To solve such a problem efficiently, we propose probabilistic weak quorum systems (PWQS) as a communication structure. Then, we propose a fault-tolerant protocol for probabilistically searching objects in P2P system.

Then, we propose a quorum-based protocol for finding objects in P2P networks. Importance of our method is that each peer does not need to maintain global information such as the set of all peers and a logical topology with global consistency. In our protocol, each peer communicates with only a small part of the peer set. Therefore, our protocol is adaptive for huge scale P2P networks.

In the design of P2P systems, fault tolerance and scalability are important factors. Different from other protocols, our protocol uses neither logical topology nor virtual space. By such a construction, our scheme is adaptive for (1) dynamic change of the peer set, and (2) faults.

In this paper, we assume that there are some replicas of object. Especially, we propose a resource model such that \( p = n'/n \) is a constant, where \( n \) is the number of peers in a system, and \( n' \) is a lower bound of the number of peers that hold replica or original for any object type. For any real \( 0 < \sigma < 1 \), our protocol guarantees that each object is found at least with probability \( \sigma \). The number of messages for finding an object or replica is \( O(\sqrt{1/p}\log(1/(1-\sigma))) \).

B. Related works

Many P2P systems have been proposed so far, for example, CAN [2], Chord [3], Gnutella [4], FreeNet [5], etc. CAN (Contents Addressable Network) uses a distributed hash table for managing index of shared objects. A set of peers in a P2P system maintains a virtual hash space. Each peer is responsible

K. Miura is with Oki Data Corporation, 4-11-22 Shibaura, Minato-ku, Tokyo 108-8551 Japan. E-Mail: miura828@oki.com, Tel: +81-3-5445-6156, Fax: +81-3-5445-6175.

T. Tagawa and H. Kakugawa are with the Department of Information Engineering, Hiroshima University, 1-4-1 Kagamiyama, Higashi Hiroshima, Hiroshima, 739-8527 Japan. E-Mail: {onet,kakugawa}@aial.hiroshima-u.ac.jp, Tel: +81-824-24-7681, Fax: +81-824-22-7195
for a zone (subspace) of the hash space, and maintains indices of objects whose hash value is in the zone. Thus, peers must maintain a set of zones to cover the whole hash space. When an object is searched, a query message is sent to a zone that contains the hash value of the searching object. A query message is sent by repeatedly forwarding it from a zone to a neighbor zone in the direction of the destination. Chord [3] also uses a distributed hash for indexing shared objects. Gnutella [4] uses no logical structure. When a peer searches an object, it broadcasts a query message to neighbor peers it knows, and the query is forwarded by flooding. A query message contains a Time-to-Live (TTL) value, and the value is decremented when the message is forwarded. Forwarding a query message stops when the TTL value becomes zero. As a result, a query is flooded in the system within a certain distance from the requesting peer. FreeNet [5] uses keys (hash values) to identify shared objects. Peers in the system cache a set of shared objects whose key values are similar each other. When a peer looks for an object, it computes corresponding key values and checks if the object is cached locally. Otherwise, a query message is forwarded to a neighbor peer that holds key values similar to the target key value. GUESS [6], [7] is an extension of Gnutella without flooding. Each peer maintains a set of neighbor peers and sends queries to its neighbors directly. Unfortunately, it gives no theoretical foundation of its performance and fault tolerance.

C. Quorums

As an efficient and fault-tolerant communication structure in distributed systems, coterie is proposed by Garcia-Molina and Barbara [8]. Coterie is used in many distributed protocols such as distributed mutual exclusion [9], for example. Coterie is a set of quorums, and a quorum is a set of nodes such that any two quorums have at least one common node. (Formal definition will be given by Definition 7.) In the context of distributed computing, a node corresponds to a peer. For mutual exclusion in distributed systems, a peer asks permission to enter its critical section from each peer in a quorum, and releases permission on exit from its critical section. Because there is a common peer (node) in any two quorums, no two peers can enter their critical sections at a time, as long as each peer in quorums gives permission at most one peer at a time.

Because of its flexibility and fault-tolerant nature, its variants have been proposed in many literatures. Kakugawa et al. proposed $k$-coterie for the distributed $k$ mutual exclusion problem in [10], [11]. Kakugawa and Yamashita proposed local coterie for distributed resource allocation problem in [12]. Joung proposed surficial quorum system for the group mutual exclusion problem in [13]. Manabe et al. proposed $k$-arbiter for the distributed $k$-out of- $M$ resources allocation problem in [14]. Malkhi et al. proposed probabilistic quorum system, which is a variant of coterie with probabilistic factor in [15], [16]. Applications of probabilistic quorum systems are found in [17]–[19], for example.

Generally speaking, coterie and its variants above are designed for static networks. For dynamic networks, dynamic paths quorums system (DPQS) and non-uniform probabilistic quorum system (NPQS) are proposed in [20] and [21], respectively. A DPQS maintains a Voronoi diagram in a two dimensional virtual space $[0,1] 	imes [0,1] \subset R^2$, and each Voronoi point corresponds to a peer. A query message moves from a Voronoi cell to a neighbor cell in a greedy way so that peers in the path form a quorum. This approach is similar to CAN. A NPQS constructs a virtual de Bruijn network to estimate the number of peers that join a P2P system, and based on the estimation, a probabilistic quorum system are constructed.

D. Organization of this paper

In section II, we describe the system model, and propose a concept of probabilistic weak quorum system. In section III, we propose a protocol for searching objects in P2P systems. In section IV, we show correctness proof and performance analysis of the proposed protocol. In section V, we show asymptotic probability analysis of the quorum system proposed. In section VI, we give concluding remarks. In Table I, symbols used in this paper are summarized.

II. PRELIMINARIES

A. System model and the problem statement

A distributed system is a set of processes (peers) $U = \{p_1, p_2, \ldots, p_n\}$. Below, we use the term process instead of ‘peer’. A process $p_i$ can communicate with any other process $p_j$ as long as $p_i$ knows $p_j$’s process name1 (i.e., network address). We assume that each process $p_i$ initially knows some process names only. Formally, by $N_0$, we denote a set of process names that $p_i$ initially knows for each $p_i \in U$. We assume that a graph implicitly formed by $N_0, N_2, \ldots, N_m$ is weakly connected, because there is no way to know process names in different connected components2.

Each process joins and leaves the P2P network arbitrarily at any time, and we assume that a process in the system can

\begin{table}
\centering
\caption{Symbols.}
\begin{tabular}{|c|c|}
\hline
$U$ & $= \{p_1, p_2, \ldots, p_n\}$, a set of processes. \\
$n$ & $= |U|$, the total number of processes. \\
$N_0$ & A set of processes that $p_i$ knows initially. \\
$n'$ & The number of processes that hold object replicas. Constant. \\
$\rho$ & $= n'/n$. Constant. \\
$\sigma$ & Probability that an object search succeeds. \\
$\beta$ & Constant factor for intersection probability of PQWS. \\
$k$ & Order of PQWS. \\
$q$ & $= \beta \sqrt{1/\rho}$, the size of quorums. \\
$R_i$ & A set of processes from which $p_i$ constructs quorums. \\
r & $= |R_i|$. \\
$T_N$ & Time-to-live value of process names, used in the name thread. \\
$T_I$ & Time-to-live value of object indices, used in the index thread. \\
\hline
\end{tabular}
\end{table}

\begin{table}
\centering
\caption{Symbols.}
\begin{tabular}{|c|c|}
\hline
$U$ & $= \{p_1, p_2, \ldots, p_n\}$, a set of processes. \\
$n$ & $= |U|$, the total number of processes. \\
$N_0$ & A set of processes that $p_i$ knows initially. \\
$n'$ & The number of processes that hold object replicas. Constant. \\
$\rho$ & $= n'/n$. Constant. \\
$\sigma$ & Probability that an object search succeeds. \\
$\beta$ & Constant factor for intersection probability of PQWS. \\
$k$ & Order of PQWS. \\
$q$ & $= \beta \sqrt{1/\rho}$, the size of quorums. \\
$R_i$ & A set of processes from which $p_i$ constructs quorums. \\
r & $= |R_i|$. \\
$T_N$ & Time-to-live value of process names, used in the name thread. \\
$T_I$ & Time-to-live value of object indices, used in the index thread. \\
\hline
\end{tabular}
\end{table}

1This assumption is the same as the Internet. A process must know the IP address of a destination to send a message.

2It is not necessary that a graph is strongly connected. For instance, consider a graph such that $p_i$ knows $p_j$ but $p_j$ does not know $p_i$. Eventually $p_j$ contacts with $p_i$, then $p_j$ knows $p_i$. Thus an edge of opposite direction is added, and $p_j$ may send a message to $p_i$. 
communicate only with processes that are taking part in the system.

We assume synchronized execution model for processes. Round is a synchronized execution unit that consists of several statements. Each process starts execution of a round when all processes finish the previous round. A message sent by a process arrives its destination in the next round, and received in that round. Each process has a message queue of enough capacity for receiving messages, and we assume that a message buffer never overflows.

There are many objects shared in P2P network. For each object, a keyword is determined from its title, author or file name etc. (How to define a keyword and to match keywords are out of the scope of this paper.) There may be some replicas for each object, and we assume that original and replicas are never updated. Let \( n' \) be a lower bound of the number of original and replicas for any object. Let \( 0 < \rho < 1 \) be the ratio of \( n' \) to \( n \), i.e., \( \rho = n'/n \), where \( n \) is the total number of processes. We call \( \rho \) as replica factor.

We propose an object model such that the replica factor \( \rho \) is constant, and investigate the problem of searching object efficiently for such model. Because we assume that the original and replicas are never updated, we make no distinction between the original and replicas. Intuitively, our object model is that ‘(at least) 0.1% of the peers hold target object,’ when \( \rho = 10^{-3} \) regardless of the total number of peers \( n \). This is the resource model for finding popular objects which are likely to be replicated by certain percentage of peers. The constant value of \( \rho \) is the main issue of our model, and estimation of \( n \) is not necessary in our resource model.

The probabilistic object search problem is formally stated as follows.

**Definition 1:** The probabilistic object search problem with replicas is a problem to find a process that holds an object of the given keyword with given success probability \( 0 < \sigma < 1 \), provided that replica factor \( \rho \) is given as a constant.

**B. Probabilistic weak quorum systems**

To solve the object search problem, we propose probabilistic weak quorum systems (PWQS) as a communication structure between processes.

**Definition 2:** (Set system [22]) A set system \( Q \) of \( U \) is a family of sets of \( U \), i.e., \( Q \subseteq 2^U \). Each element \( Q \) in \( Q \) is called a quorum.

A quorum to be accessed is selected probabilistically. Probability distribution of quorums is defined by access strategy defined as follows.

**Definition 3:** (Access strategy [22]) Let \( Q \) be a set system of \( U \). Then, an access strategy of \( p_i \in U \) is a probability distribution \( w_i : Q \rightarrow [0, 1] \) such that \( \sum_{Q \in \mathcal{Q}} w_i(Q) = 1 \).

**Definition 4:** (Access strategy vector) Let \( Q \) be a set system of \( U \) and \( w_i \) be an access strategy for each \( p_i \in U \). Then, access strategy vector of \( U \) is a vector \( \overrightarrow{w} = (w_1, w_2, \ldots, w_n) \).

Now we describe the definition of PWQS.

**Definition 5:** (Probabilistic weak quorum system) For any integer \( k \geq 1 \), a tuple \( (k, Q, \overrightarrow{w}, \varepsilon) \) is a probabilistic weak quorum system of order \( k \) under \( U \) if and only if the following conditions are satisfied.

- \( Q \) is a set system of \( U \),
- \( \overrightarrow{w} = (w_1, w_2, \ldots, w_n) \) is an access strategy vector, and
- For any \( k + 1 \) processes \( p_x, p_{y_1}, p_{y_2}, \ldots, p_{y_k} \in U \),
  \[
  \sum_{j=1}^{k} w_x(Q_{y_j}) \geq 1 - \varepsilon,
  \]
  where the sum is taken over every combination of \( Q_{y_1}, Q_{y_2}, \ldots, Q_{y_k} \in Q \) such that \( Q_x \cap (Q_{y_1} \cup \cdots \cup Q_{y_k}) \neq \emptyset \).

Intuitively, any quorum is likely to have nonempty intersection with at least one quorum among any \( k \) quorums, and such an event occurs with probability at least \( 1 - \varepsilon \).

When \( k = 1 \), the definition above is identical to probabilistic quorum system which is proposed in [15], [16]. In this sense, PWQS is a generalization of probabilistic quorum system.

We give intuitive explanation how processes can use a PWQS \( Q \) of order \( k \) for searching an object.

1) Each process \( p_i \), that owns an object randomly selects a quorum \( Q_{y_i} \in Q \) according to its access strategy, and sends a keyword of the object for each process in \( Q_{y_i} \).
2) When a process \( p_x \) wishes to search an object, it randomly selects a quorum \( Q_x \in Q \) according to its access strategy, then, it sends a query message for each process in \( Q_x \).
3) A process in \( Q_x \cap Q_{y_i} \) for some \( p_i \) informs \( p_x \) that \( p_y \) owns the target object.
4) If the number of processes that hold the target object is \( k \), there is such a \( p_y \), with probability at least \( 1 - \varepsilon \) by definition of PWQS.

Below, we show an example of PWQS.

**Example 1:** (PWQS construction 1) For any integer \( k \geq 1 \) and real \( \beta > 0 \), let
- \( U = \{p_1, p_2, \ldots, p_n\} \),
- \( Q = \{Q \subseteq U : |Q| = [\beta \sqrt{n/k}]\} \), and
- \( \overrightarrow{w} = (w_{A,1}, w_{A,2}, \ldots, w_{A,n}) \), where \( w_{A,i}(Q) = 1/|Q| \) for each \( 1 \leq i \leq n \).

We define \( \pi_A \equiv (k, Q, \overrightarrow{w}, e^{-\beta^2}) \), where \( e \) is the base of the natural logarithm function.

**Lemma 1:** \( \pi_A \) is a probabilistic weak quorum system of order \( k \) under \( U \).

**Proof:** By definition, each process selects a quorum uniformly at random from a quorum set in \( Q \). Let \( Q, Q' \in Q \) be quorums selected randomly, and \( q \) be the quorum size. Because \( q = |Q| = |Q'| = [\beta \sqrt{n/k}] \), we have the following inequality.

\[
\Pr[Q \cap Q' = \emptyset] = \binom{n-q}{q} \frac{n!}{q! (n-q)!} \leq \left( \frac{n-q}{n} \right)^q \leq e^{-q^2/n} \leq e^{-\beta^2/k}
\]

Note that in the above inequality, we used the relation \( (1 + t/n)^n \leq e^t \) for all \( n, t \in \mathbb{R} \) such that \( n \geq 1 \) and \( |t| \leq n \).
The probability that a quorum \( Q \) has nonempty intersection with at least one quorum among \( k \) quorums \( Q_1, Q_2, \ldots, Q_k \) is evaluated as follows.

\[
\Pr[Q \cap (Q_1 \cup \cdots \cup Q_k) \neq \emptyset] = 1 - (\Pr[Q \cap Q' = \emptyset])^k \\
\geq 1 - e^{-\beta^2}
\]

(1)

Thus, \( Q \) satisfies the condition of PWQS.

The PWQS \( \pi_A \) defined above is uniform in a sense that each process selects a subset \( Q \) of \( U \) uniformly at random. Thus, each process must have knowledge of \( U \) to construct such a PWQS.

### C. Probabilistic weak quorum system for P2P

In P2P networks, it may be difficult to construct a PWQS \( \pi_A \), because each process need to know \( U \). Next, we propose a PWQS \( \pi_B \) suitable for P2P environment, which requires only a partial knowledge of \( U \).

**Example 2: (PWQS construction 2)** For any integer \( k \geq 1 \) and real \( \beta > 0 \), let

- \( U = \{p_1, p_2, \ldots, p_n\} \),
- \( Q = \{Q \subseteq U : |Q| = \lfloor \beta \sqrt{n/k} \rfloor \} \), and
- \( r \) be any constant such that \( \lfloor \beta \sqrt{n/k} \rfloor \leq r \leq n \).

We implicitly define \( w_{B,i} \) and \( w_{B,i} \) as follows.

- Initially, each \( p_i \) selects a set \( R_i \subseteq U \) such that \( |R_i| = r \) uniformly at random, and then,
- Each \( p_i \) selects a quorum \( Q \in Q \) such that \( Q \subseteq R_i \) uniformly at random, i.e., \( w_B(Q) = \frac{1}{r} \) if and only if \( Q \subseteq R_i \).

We define \( \pi_B \equiv \{k, \pi_B, e^{-\beta^2}\} \).

**Lemma 2:** \( \pi_B \) is a PWQS of order \( k \).

**Proof:** (Outline) Because each process \( p_i \) selects \( R_i \) and then \( Q \subseteq R_i \), both uniformly at random, \( p_i \) selects each \( Q \subseteq Q \) with uniform probability as a result. Thus, intersection probability of quorums in PWQS \( \pi_B \) is the same as those of \( \pi_A \).

Because replica factor \( \rho \) is a constant such that \( \rho = k/n \), size of quorums \( |Q| = \lfloor \beta \sqrt{1/\rho} \rfloor \) is a constant. In addition, if each process can select \( R_i \) uniformly at random from its local information, each process can form \( \pi_B \) without global knowledge of a system.

In Table II, size of quorums in \( \pi_A \) and \( \pi_B \) are shown. The quorum sizes are computed according to Example 1 and (1) so that the probability of quorum intersection, i.e., \( \Pr[Q \cap (Q_1 \cup \cdots \cup Q_k) \neq \emptyset] \), is achieved. For example, when \( n/k = 1/\rho = 10^3 \), quorum of size 68 is enough to achieve intersection probability 0.99. Figure 1 shows a relation between quorum size \( q \) and intersection probability.

### D. Weak quorum system and coterie

For applications in which an object should be found deterministically, we propose weak quorum system defined as follows. Any quorum of a weak quorum system must have non-empty intersection with at least one quorum among any distinct \( k \) quorums.

**Definition 6: (Weak quorum system)** A tuple \( (k, Q) \) is a weak quorum system of order \( k \) under \( U \) if and only if \( Q \) satisfies the following conditions.

- \( Q \) is a set system of \( U \), and
- For any \( k + 1 \) quorums \( Q_0, Q_1, Q_2, \ldots, Q_k \) in \( Q \), \( Q_0 \cap \bigcup_{i=1}^{k} Q_i \neq \emptyset \).

Note that the object search scheme outlined above can be used for weak quorum systems without modification. Probabilistic weak quorum systems and weak quorum systems are extension of coterie defined as follows. In the definitions of coterie, any two quorums must have nonempty intersection.

**Definition 7:** (Coterie [8]) \( Q \) is a coterie under \( U \) if and only if the following conditions are satisfied.

- \( Q \) is a set system of \( U \), and
- (Intersection Property) For any quorums \( Q, Q' \in Q \), \( Q \cap Q' \neq \emptyset \) holds.
- (Minimality) \( \forall Q, Q' \in Q : Q \cap Q' \neq \emptyset \).

For the purpose of distributed mutual exclusion, any two processes must not be in their critical sections at any time. Thus, coterie can be used for this purpose since any two quorums strictly have non-empty intersection. For purpose in which strictness is not necessary, a condition for quorum intersections can be relaxed, and probabilistic quorum system [15], [16] is such an example. PWQS is a further extension of probabilistic quorum systems — when \( k = 1 \), PWQS of degree \( k \) is the same as probabilistic quorum systems.

**Table II**

<table>
<thead>
<tr>
<th>Size of quorums in ( \pi_A ) and ( \pi_B ).</th>
<th>Quorum size</th>
<th>( n/k = 10^2 )</th>
<th>( 10^3 )</th>
<th>( 10^4 )</th>
<th>( 10^5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.900000</td>
<td>16</td>
<td>43</td>
<td>152</td>
<td>480</td>
<td></td>
</tr>
<tr>
<td>0.990000</td>
<td>22</td>
<td>68</td>
<td>215</td>
<td>679</td>
<td></td>
</tr>
<tr>
<td>0.999000</td>
<td>27</td>
<td>84</td>
<td>263</td>
<td>832</td>
<td></td>
</tr>
<tr>
<td>0.999900</td>
<td>31</td>
<td>96</td>
<td>304</td>
<td>960</td>
<td></td>
</tr>
<tr>
<td>0.999990</td>
<td>34</td>
<td>108</td>
<td>340</td>
<td>1073</td>
<td></td>
</tr>
<tr>
<td>0.999999</td>
<td>38</td>
<td>120</td>
<td>372</td>
<td>1176</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 1. Quorums size \( q \) and intersection probability, when \( n = 10^5 \) and \( n/k = 10^3 \).
III. A PROTOCOL FOR SEARCHING OBJECTS

A. Outline

Now we propose a protocol for searching objects in P2P networks. Parameters of the protocol are ρ (replica factor) and σ (success probability of searching) as described in Section II. To search objects, we use PQWS πR of order k such that ρ = k/n. Each process pi collects a process set Ri by the protocol described in subsection III-B. Thus, the size of each

\[ p_i \text{ collects } R_i \text{ by the protocol described in subsection III-B.} \]

We assume that \( |R_i| \leq n \).

The proposed protocol executes the following three threads in parallel at each process pi.

1) The name thread for dynamically collecting a set of process names into a variable Ri.
2) The index thread for a database of objects.
3) The query thread for asking object location.

The name and index threads are executed all the time when pi is participating in the system. The query thread is invoked at a process only when the process searches an object.

The outline of the protocol is described as follows.

- **When a process pi joins the P2P system:** It starts the name and index threads. These two threads must be invoked before the following steps are executed.
- **Collecting a set of process names:** The name thread collects a set of process names in a variable Ri. See Subsection III-B for details.
- **If pi possesses an object:** The index thread at pi periodically sends a keyword to every process in a quorum Q ⊆ Ri selected uniformly at random, when \( |R_i| \geq r \). Such a keyword is maintained by the index thread at each process in Q.
- **When pi searches an object:** The query thread is invoked at pi, and it sends a query message with a keyword to every process in a quorum Q ⊆ Ri selected uniformly at random. The index thread at some process that knows the keyword will reply the process name, say pj, that possesses the corresponding object. Then, pi asks pj to send a copy of the object. When pi obtains a copy, the query thread terminates.

In case that pi receives no reply for certain time period, pi may repeat the query until it succeeds.

Because processes may be aborted by user or by some reasons, a process may not leave the system properly. In such a case, a process set Ri becomes obsolete. The index thread automatically deletes such obsolete information from the system as follows.

- Each process name in Ri is paired with TTL (time-to-live) value, and each process exchanges Ri together with TTL values. Each process pi periodically decrements the TTL value by one for each entry in Ri. If the TTL value becomes zero such an item is removed from Ri.
- Even if pi successfully collects Ri so as to \( |R_i| \geq r \), entries in Ri decrease as the time goes by. To refresh Ri, each pi periodically sends its Ri to a process pj ∈ Ri selected randomly according to the protocol in Subsection III-B.
- The index information of objects, maintained by the index thread, is handled in the same way.

In many other P2P systems, a process must execute some procedure when it leaves a system to maintain logical topology or virtual space properly. Unfortunately, such a procedure may not be executed by, for example, network disconnection and abort of P2P software, and logical topology or virtual space may be broken. On the other hand, our protocol does not have a procedure on exit from a system, because it is correct without such a procedure, as discussed in subsection IV-B.

B. The name thread

We propose a protocol to obtain Ri for each process. The proposed protocol is based on the Name-Dropper algorithm proposed by Harchol-Balter et al. [23]. It collects the set all process names U in a network, provided that each process pi initially knows only a subset of process names \( N_i^0 \subseteq U \).

In the Name-Dropper algorithm, execution of processes is synchronized, and each process simply repeats the following three steps as a single round:

1) Select a process from a process name set it collected so far, (2) Send a process name set to the selected process, and (3) Receives process names and merges. It is known that, by the Name-Dropper algorithm, each process pi obtains \( R_i = U \) after \( O(\log^2 n) \) rounds with probability at least \( 1 - 1/n^{O(1)} \) [23].

For our purpose, it is enough to collect \( r \) process names.

Outline of the name thread is described below.

**Local variables of each process pi:**

- Input: \( N_i^0 \subseteq U \) — a set of process names that \( p_i \) knows initially.
- Output: \( R_i \subseteq U \) — a set of process names that \( p_i \) collects.

**Action of each process pi:**

1) Use a working variable \( N_i \), whose value is initially \( N_i^0 \).
2) If \( |N_i| < r \), select a process \( p \neq p_i \) uniformly at random from \( N_i \), and send \( N_i \) to \( p \).
3) Receive each message \( N \) at \( p_i \) from \( p_k \), if any. If \( |N_k| < r \), \( p_i \) updates \( N_i \) as \( N_i \cup N \cup \{p_k\} \). Otherwise, if \( |N_k| \geq r \), \( p_i \) adds \( p_k \) into \( N_i \), and then it sends \( N_i \) to \( p_k \).
4) Receive each message \( N \) sent in step 2, if any, and compute \( N_i := N_i \cup N \).
5) If \( |N_i| \geq r \), select \( R_i \subseteq N_i \) such that \( |R_i| = r \) uniformly at random.
6) Repeat steps (2)–(5) above forever.

In our protocol, a process \( p_i \) does not send process names in step 2 if it holds enough number of process names (i.e., \( |N_i| \geq r \)). This implies that a process that sends process names in step 2 does not have enough number of process names. When a process \( p_i \) receives process names from, say \( p_k \), in step 3, \( p_i \) sends \( N_i \) to \( p_k \) in step 3, and such messages are received in step 4. Although the Name-Dropper algorithm does not have this action, it is required in our protocol for a situation such that \( |N_i| < r \) for some \( p_i \) and \( |N_i| \geq r \) for each \( p_j \neq p_i \). In this situation, \( p_i \) never receives process names without such an action.
constant $T_X : integer$
// Time-to-live value of process names.
custom constant $T_Y : integer$
// Time-to-live value of object indices.
custom constant $r : integer$
// Size of $R_i$ for quorum construction.
shared variable $N_i \subseteq U \times \{1..T_X\}$
// A set of process names with TTL values.
shared variable $R_i \subseteq U$
// A set of process names.

Fig. 2. Variables.

C. Formal description

Formal description of the proposed algorithm is shown in Figure 2 (variables), Figure 3 (the name thread), Figure 4 (the index thread), and Figure 5 (the query thread). In the description of the algorithm, we use the following message types:

- **names**: Used by the name thread to propagate a set of process names.
- **index**: Used by the index thread to inform object keywords to members in a quorum.
- **query**: Used by the query thread to search objects.
- **reply**: Used by the index thread, as a reply to a query message, to tell process name that holds an object.

Although quorums may be constructed from $N_i$ instead of $R_i$ in practice, our description strictly implements PWQS $\pi_B$.

D. Evaluation of the name thread by simulation

We show simulation results of the name thread for collecting process names in a P2P network. A set $N^0_i \subseteq U$ of process names that $p_i$ knows initially is given by generating a spanning tree of $U$ randomly, and direction of each edge is assigned randomly. Then, $j \in N^0_i$ iff $(i,j)$ is an edge of the spanning tree. Possibly $p_i$ obtains process names more than $r$, because other processes may inform $p_i$ a set of process names $N_j$ when $|N_j| \geq r$.

We start a simulation with random $N^0_i$ values, and a simulation is stopped just when every thread obtains more than $r$ process names in $N_i$. We observe the followings when the simulation terminates.

- The size of $N_i$, and
- The number of rounds of the simulation.

The simulation\(^3\) is done for the cases of $r = 25, 50, 75$ and $100$, and the number of trials is $10000$. For comparison, we also simulate the Name-Dropper. Figure 6 (a) shows the expected size of $N_i$ when the simulation terminates, and Figure 6 (b) shows the expected number of rounds to terminate the simulation.

From the results of our simulation, (1) The expected size of $N_i$ is approximately $2r$, and (2) the number of rounds to obtain process names is proportional to logarithm of the number of processes in a P2P system. On the other hand, the Name-Dropper collects too many process names for our purpose, and it requires more rounds to finish.

\(^3\)The simulation is executed on a PC (Pentium IV CPU, 1.78 GHz clock, 1G byte memory) with Fedora Core Linux Release 1 (kernel 2.4.22). The simulation program is written in C, and compiled by gcc 3.3.2.

Consider a process, say $p_i$, such that $N^0_i = \{p_j\}$ and $p_i \not\in N_k$ for any $p_k$. If $N_i$ is a large set, the probability that $p_i$ receives a name set from $p_j$. While $p_i$ is waiting for such an event, other processes enlarges their process sets. This is the reason the Name Dropper collects large set. On the other hand, by our name protocol, a process does not collect process names actively when its name set is large enough, and the name set does not grow so fast.

Although the protocol of the name thread is a minor change of the Name-Dropper, it does not collect too many process names, and average performance seems to be independent of the total number of processes in a P2P system. Thus, our protocol scales well for huge P2P systems.
IV. PROOFS

A. Correctness and performance

Lemma 3: The proposed algorithm (the name thread) collects a $r$-set of processes in $O(\log r \log n)$ rounds with probability at least $1 - 1/n^{O(1)}$.

Proof: The proof is based on the proof for the Name-Dropper algorithm [23]. In [23], it is shown that, for any constant $c$, the distance between every pair of processes in a graph formed by $N_i$s decreases by a constant factor in $O(c \log n)$ rounds with probability at least $1 - 1/n^{O(c)}$. A stage is defined as such a consecutive of $O(c \log n)$ rounds.

In case of our protocol, it is enough that every pair of processes whose initial distance is $r$ becomes distance zero. Thus, $\log r$ stages are enough, and we obtain $O(c \log r \log n)$ rounds in total. Selection of appropriate $c$ yields the lemma.

Because the name thread at each process sends single message in each round, message complexity per process is $O(\log r \log n)$ with high probability (the size of a message can be large since it contains $R_i$).

Lemma 4: The name thread computes PWQS $\pi_B$, if $N_i^0$ is given independently and uniformly at random for each $p_i$.

Proof: Because distribution of $N_i^0$ is uniform and the name thread selects destination of $N_i$ uniformly at random from $N_i$, distribution of resulting $N_i$ is also uniform.

When $p_i$ constructs a quorum set, $p_i$ selects $r$ processes from $R_i$ uniformly at random, and then quorums are selected from the resulting process set. Each $p_i$ obtains a set of quorums $Q$ such that $w_i(Q) > 0$, which is essentially the same as $\pi_B$.

Theorem 1: For any given success probability $0 < \sigma < 1$ and replica factor $0 < \rho < 1$, the proposed protocol solves the probabilistic object search problem, provided a set of processes in a system is static and no faults occur.

Proof: By the definition of the proposed protocol, a PWQS adopted by the protocol is $\pi_B$ with quorum size $\beta \sqrt{1/\rho}$, where $\beta = \sqrt{\log(1/(1-\sigma))}$, and the order of $\pi_B$ is $k = r\beta$.

The probability that any quorum has non-empty intersection with at least one quorum among any $k$ quorums, selected randomly and independently, is evaluated as follows.

$$1 - e^{-\beta^2} = 1 - e^{-\log(1/(1-\sigma))} = 1 - (1 - \sigma) = \sigma$$

From the quorum size, it is clear that the message complexity for searching an object is

$$O(q) = O\left(\frac{1}{\sqrt{\rho \log(1/(1-\sigma))}}\right),$$

which is independent from $n = |U|$. Time complexity for searching an object is also independent from $n$.

B. Fault-tolerance

Next, we discuss fault-tolerance of the proposed protocol. Fault class we consider is finite number of crash faults (a faulty process stops and does nothing from the point on) and transient faults (local memory of a faulty process is corrupted,
The proposed protocol is self-stabilizing.

V. Probability Analysis of PWQS $\pi_B$

In this section, we show some analysis of intersection probability of PWQS $\pi_B \equiv \langle k, Q, \pi_B, e^{-\beta^2} \rangle$.

A. Asymptotical optimality of intersection probability

Next theorem shows that the lower bound of (1) is asymptotically tight, i.e., constructions of PWQS $\pi_A$ and $\pi_B$ are asymptotically optimal in the sense of quorum intersection probability.

Theorem 3: Let $k = k(n) \equiv \rho m$, where $\rho$ is replica factor. Then, \( \lim_{n \to \infty} \text{Pr}[X_1 = 1] = 1 - e^{-q^2} \). Especially, when $\beta \sqrt{1}/\rho$ is an integer, \( \lim_{n \to \infty} \text{Pr}[X_1 = 1] = 1 - e^{-\beta^2} \).
Consider the following random process.

Thus, we have the following lower and upper bounds.

\[ Pr[X_1 = 1] \geq 1 - \left( 1 - \frac{q}{n} \right)^{qk} \]
\[ Pr[X_1 = 1] \leq 1 - \left( 1 - \frac{q}{n - q + 1} \right)^{qk} \]

Because \((1 + t/n)^n \leq e^t (n \geq 1 \text{ and } |t| \leq n)\),
\[ 1 - (1 - \frac{q}{n})^{qk} = 1 - (1 + \frac{q}{n})^{qk} \leq 1 - e^{-q^2\rho}. \]

Thus, we have \( Pr[X_1 = 1] \geq 1 - e^{-q^2\rho} \).

Because \((1 - t/(n+1))^n \geq e^t (n \geq 1 \text{ and } 0 \leq t < n)\),
\[ 1 - \left( 1 - \frac{q}{n - q + 1} \right)^{qk} \leq 1 - \left( e^{-q}(1 - \frac{q}{n - q + 1}) \right)^{qk} \]
\[ \rightarrow 1 - e^{-q^2\rho}. \]

Therefore, \( \lim_{n \to \infty} Pr[X_1 = 1] = 1 - e^{-q^2\rho} \) holds.

Because \( q = \lceil \beta \sqrt{1/\rho} \rceil \), we have \(-q^2\rho = -\beta^2\) when \( \beta \sqrt{1/\rho} \) is an integer.

**Proof:**

By Lemma 1, we have

\[
Pr[X_1 = 1] = 1 - \left( \frac{n-q}{q} \right)^k
= 1 - \left\{ \frac{(n-q)! (n-q)!}{n! (n-2q)!} \right\}^k
= 1 - \prod_{j=0}^{q-1} \frac{n-q-j}{n-j}^k
= 1 - \left\{ \prod_{j=0}^{q-1} (1 - \frac{q}{n-j}) \right\}^k.
\]

Thus, we have the following lower and upper bounds.

\[
Pr[X_1 = 1] \geq 1 - \left( 1 - \frac{q}{n} \right)^{qk}
Pr[X_1 = 1] \leq 1 - \left( 1 - \frac{q}{n - q + 1} \right)^{qk}
\]

Because \((1 + t/n)^n \leq e^t (n \geq 1 \text{ and } |t| \leq n)\),
\[ 1 - (1 - \frac{q}{n})^{qk} = 1 - (1 + \frac{q}{n})^{qk} \leq 1 - e^{-q^2\rho}. \]

Thus, we have \( Pr[X_1 = 1] \geq 1 - e^{-q^2\rho} \).

Because \((1 - t/(n+1))^n \geq e^t (n \geq 1 \text{ and } 0 \leq t < n)\),
\[ 1 - \left( 1 - \frac{q}{n - q + 1} \right)^{qk} \leq 1 - \left( e^{-q}(1 - \frac{q}{n - q + 1}) \right)^{qk} \]
\[ \rightarrow 1 - e^{-q^2\rho}. \]

Therefore, \( \lim_{n \to \infty} Pr[X_1 = 1] = 1 - e^{-q^2\rho} \) holds.

Because \( q = \lceil \beta \sqrt{1/\rho} \rceil \), we have \(-q^2\rho = -\beta^2\) when \( \beta \sqrt{1/\rho} \) is an integer.

**Proof:**

By Lemma 1, we have

\[
Pr[X_1 = 1] = 1 - \left( \frac{n-q}{q} \right)^k
= 1 - \left\{ \frac{(n-q)! (n-q)!}{n! (n-2q)!} \right\}^k
= 1 - \prod_{j=0}^{q-1} \frac{n-q-j}{n-j}^k
= 1 - \left\{ \prod_{j=0}^{q-1} (1 - \frac{q}{n-j}) \right\}^k.
\]

Thus, we have the following lower and upper bounds.

\[
Pr[X_1 = 1] \geq 1 - \left( 1 - \frac{q}{n} \right)^{qk}
Pr[X_1 = 1] \leq 1 - \left( 1 - \frac{q}{n - q + 1} \right)^{qk}
\]

Because \((1 + t/n)^n \leq e^t (n \geq 1 \text{ and } |t| \leq n)\),
\[ 1 - (1 - \frac{q}{n})^{qk} = 1 - (1 + \frac{q}{n})^{qk} \leq 1 - e^{-q^2\rho}. \]

Thus, we have \( Pr[X_1 = 1] \geq 1 - e^{-q^2\rho} \).

Because \((1 - t/(n+1))^n \geq e^t (n \geq 1 \text{ and } 0 \leq t < n)\),
\[ 1 - \left( 1 - \frac{q}{n - q + 1} \right)^{qk} \leq 1 - \left( e^{-q}(1 - \frac{q}{n - q + 1}) \right)^{qk} \]
\[ \rightarrow 1 - e^{-q^2\rho}. \]

Therefore, \( \lim_{n \to \infty} Pr[X_1 = 1] = 1 - e^{-q^2\rho} \) holds.

Because \( q = \lceil \beta \sqrt{1/\rho} \rceil \), we have \(-q^2\rho = -\beta^2\) when \( \beta \sqrt{1/\rho} \) is an integer.
Fig. 7. Relation of $r$ and $\Pr[X_2 = 1|X_1 = 0]$, when $q = 16$, $n = 10000$, and $k = 100$.

**Proof:** By Lemma 5, we have

$$
\Pr[X_2 = 1|X_1 = 0] = \sum_{i=1}^{q} \left( \binom{q}{i} \frac{q-i}{2q} \right) \left( 1 - \frac{(n-q-i)! (n-2q)!}{(n-i)! (n-2q-i)!} \right)^k
$$

$$
= \sum_{i=1}^{q} \left( \binom{q}{i} \frac{q-i}{2q} \right) \left( 1 - \prod_{j=0}^{i-1} \frac{n-q-i-j}{n-q-j} \right)^k
$$

$$
= \sum_{i=1}^{q} \left( \binom{q}{i} \frac{q-i}{2q} \right) \left( 1 - \prod_{j=0}^{i-1} \left( 1 + \frac{-i}{n-q-j} \right) \right)^k
$$

Thus, we have the following lower and upper bounds.

$$
\Pr[X_2 = 1|X_1 = 0] \geq \sum_{i=1}^{q} \left( \binom{q}{i} \frac{q-i}{2q} \right) \left( 1 - \left( 1 + \frac{-i}{n-q} \right)^{g_k} \right)
$$

$$
\Pr[X_2 = 1|X_1 = 0] \leq \sum_{i=1}^{q} \left( \binom{q}{i} \frac{q-i}{2q} \right) \left( 1 - \left( 1 + \frac{-i}{n-2q+1} \right)^{g_k} \right)
$$

Because

$$
1 - (1 + \frac{-i}{n-q})^{g_k} = 1 - \left( 1 + \frac{-i}{n-q} \right)^{q_{n-pn}}
$$

$$
= 1 - \left\{ (1 + \frac{-i}{n-q})^{n-q} \left( 1 + \frac{-i}{n-q} \right)^q \right\}^{q_p}
$$

$$
\geq 1 - \left\{ e^{-i} \left( 1 + \frac{-i}{n-q} \right)^q \right\}^{q_p}
$$

$$
\to 1 - (e^{-i})^{q_p} \ (n \rightarrow \infty)
$$

$$
= 1 - e^{-i^{q_p}}
$$

for lower bound and

$$
1 - (1 + \frac{-i}{n-2q+1})^{g_k}
$$

$$
= 1 - \left( 1 + \frac{-i}{n-2q+1} \right)^{n_{q_p}}
$$

$$
= 1 - \left\{ (1 + \frac{-i}{n-2q+1})^{n-2q} \right\}^{q_p}
$$

for upper bound, the proof is done.

Next, when $r = 2q$, we show a positive constant lower bound of $\lim_{n \rightarrow \infty} \Pr[X_2 = 1|X_1 = 0] \geq (1 - e^{-q^2/2})/2$. Especially, when $\beta \sqrt{1/\rho}$ is an integer, $\lim_{n \rightarrow \infty} \Pr[X_2 = 1|X_1 = 0] \geq (1 - e^{-3^2/2})/2$.

**Proof:** By using a relation [28] $\sum_{i=0}^{q} \left( \binom{q}{i} \right) = \sum_{i=0}^{q} \left( \binom{2q}{i} \right)$ and the fact that $1 - e^{-x} > 1 - e^{-y}$ for any $x > y > 0$, we have the following inequalities. When $q$ is even,

$$
\lim_{n \rightarrow \infty} \Pr[X_2 = 1|X_1 = 0] = \sum_{i=1}^{q} \left( \binom{q}{i} \frac{q-i}{2q} \right) (1 - e^{-i^{q_p}})
$$

$$
\geq \sum_{i=q/2}^{q} \left( \binom{q}{i} \frac{q-i}{2q} \right) (1 - e^{-i^{q_p}})
$$

$$
\geq \sum_{i=q/2}^{q} \left( \binom{q}{i} \frac{q-i}{2q} \right) (1 - e^{-q/2 \cdot q_p})
$$

$$
= (1 - e^{-q^2/2}) \sum_{i=q/2}^{q} \left( \binom{q}{i} \frac{q-i}{2q} \right)
$$

$$
\geq (1 - e^{-q^2/2})/2,
$$

and when $q$ is odd,

$$
\lim_{n \rightarrow \infty} \Pr[X_2 = 1|X_1 = 0] = \sum_{i=1}^{q} \left( \binom{q}{i} \frac{q-i}{2q} \right) (1 - e^{-i^{q_p}})
$$

$$
\geq \sum_{i=1}^{q} \left( \binom{q}{i} \frac{q-i}{2q} \right) (1 - e^{-q^2/2})/2
$$

$$
\geq (1 - e^{-q^2/2})/2.
$$
\[
\begin{align*}
&\geq \sum_{i=(q+1)/2}^{q} \binom{q}{i} (q^{-i}) (1 - e^{-iq\rho}) \\
&\geq \sum_{i=(q+1)/2}^{q} \binom{q}{q-i} (1 - e^{-(q+1)/2}q\rho) \\
&> \sum_{i=(q+1)/2}^{q} \binom{q}{i} (1 - e^{-iq\rho}) \\
&= (1 - e^{-q^2\rho/2}) \sum_{i=(q+1)/2}^{q} \binom{q}{i} \\
&= (1 - e^{-q^2\rho/2})/2.
\end{align*}
\]

When \( \beta \sqrt{1/\rho} \) is an integer, \( q^2 = \beta^2/\rho \) and thus we have
\[
\lim_{n\to\infty} P[X_2 = 1|X_1 = 0] = (1 - e^{-\beta^2/2})/2.
\]

For example, when \( \beta \sqrt{1/\rho} \) is an integer, the lower bound is \( \approx 0.196734 \) when \( \beta = 1 \), \( \approx 0.432332 \) when \( \beta = 2 \), and \( \approx 0.494445 \) when \( \beta = 3 \). In Figure 8, the following values are shown for the case of \( \rho = 0.01 \).

- \( \lim_{n\to\infty} P[X_1 = 1] \)
- \( \lim_{n\to\infty} P[X_2 = 1|X_1 = 0] \), and
- its lower bound \( (1 - e^{-q^2\rho/2})/2 \).

These values are computed by expressions shown in Theorem 3, Lemma 6, and Theorem 4, respectively. Because of the rough analysis, \( P[X_2 = 1|X_1 = 0] \) computed is about two times larger than the lower bound obtained by Theorem 4.

Thus, small value for \( r = Cq \), where \( C \geq 2 \), suffices so that retry succeeds with reasonable probability. As a result, it is enough for the name thread to collect \( r = Cq \) process names for some \( C \geq 2 \), which proves that execution of our protocol is efficient.

### VI. CONCLUSION

In this paper, we proposed a quorum-based protocol for searching objects in P2P systems. Our protocol guarantees that a target object is found with any probability, which is given in advance. We proposed a probabilistic weak quorum system to theoretically guarantee such success probability. We gave a lower bound of the success probability for retry of object search when \( r = |R_1| \) is equal to twice of quorum size. Thus, it is enough for each process to collect constant number of process names (determined by success probability and replica factor) for construction of quorums.

In Table III, a comparison of quorum-based approaches for P2P systems is shown. The number of rounds is the number of execution steps of a protocol by assuming that messages can be transmitted in parallel as possible. The number of messages shown is the value for each process. Note that other methods DPQS [20] and NPQS [21] in the table assume that there is no replica of object in the system, which is a different resource model from ours.

Different from flooding-based protocols such as Gnutella, our protocol theoretically guarantees the success probability of object search. In case of flooding-based protocols, a target object must be within a certain distance, and increase of search range causes an explosion of query messages. Thus, our protocol is obviously efficient in message complexity.

Because our protocol does not maintain logical topology or space formed by peers, it does not need to consider special procedure for recovery on faults or changes of member peers. Especially, our protocol is self-stabilizing and tolerates finite number of any transient faults. It also allows that any number of members can enter and leave the system at any timing. From the analysis, our protocol is scalable in both message complexity and time complexity.

The description of the index and query threads shown is strictly based on the basic idea. When our protocol is implemented as a network application software, there can be some practical modifications to make it efficient. Modification and evaluation of the proposed protocol under practical setting is left for future task.

Our PWQS proposed in this paper uses quorums of the uniform size which is determined in advance by \( \rho \). PWQS with variable quorum size seems to be more realistic for searching objects with replica factor smaller than \( \rho \) expected in advance. Formulation of such quorums and probabilistic analysis is an interesting task.

### ACKNOWLEDGMENT

This work is partially supported by the Ministry of Education, Culture, Sports, and Technology, Grant-in-Aid 15700017.

### REFERENCES


TABLE III

Comparison of quorum-based protocols.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Quorum construction</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(or Joining to the system)</td>
<td>#rounds</td>
<td>#messages (per process)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$O(\log r \log n)$ w.h.p.</td>
<td>$O(\log n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td><strong>Object search</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>#rounds</td>
<td>#messages (per request)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$O(1)$</td>
<td>$O(\sqrt{\pi \log n})$</td>
<td>$O(\sqrt{\pi \log n})$</td>
</tr>
<tr>
<td>Deterministic / Nondeterministic?</td>
<td>probabilistic</td>
<td>deterministic</td>
<td>probabilistic</td>
</tr>
</tbody>
</table>


Ken Miura received the B.E. and M.E. degrees from Hiroshima University in 2002 and 2004 respectively. He is currently with Oki Data Corporation.

Taro Tagawa received the B.E. degree in 2004 from Hiroshima University. He is currently a master student of Hiroshima University. His research interests include distributed algorithms.

Hirotugu Kakugawa received the B.E. degree in engineering in 1990 from Yamaguchi University, and the M.E. and D.E. degrees in information engineering in 1992, 1995 respectively from Hiroshima University. He is currently an assistant professor of Hiroshima University. He is a member of the IEEE Computer Society and the Information Processing Society of Japan.
Consider an execution starting at time $t$ be a process such that the while-loop of the name thread (Figure 3) is executed in a time unit. Below, we use a term ‘time’ instead of ‘round’. For the index thread (Figure 4), we assume a round corresponds only by $p$ from corruption of local variables. We have the following two lemmas to show recovery of faults. We assume that (any number of) transient faults occur at time $t$. We denote the value of $N_i$ at time $t$.

We define that a configuration is legitimate if and only if the following conditions hold.

- No garbage (corrupted) messages are not in transit and not buffered.
- For any $p_i \in U(t)$ and for any $p_j$ such that $p_j \not\in U(t')$ for all $t' (0 \leq t - T_N \leq t' \leq t)$, $p_j$ is not in a set $\{p : (p, *) \in N_i(t)\}$, and
- For any $p_i \in U(t)$ and for any $p_j$ such that $p_j \not\in U(t')$ for all $t' (0 \leq t - T_I \leq t' \leq t)$, $p_j$ is not in a set $\{p : (p, *) \in X_i(t)\}$.

Specification predicate $S$ of the system is true if and only if a configuration is legitimate.

We show below how our protocol recovers from transient faults. We assume that (any number of) transient faults occur before time $t_0$ and no faults occur thereafter.

**Lemma 7:** At time $t_0 + 1$, there is no corrupted message in transit or buffered in the system.

**Proof:** Because every corrupted message is received no later than $t_0 + 1$, there is no corrupted message in transit at time $t_0 + 1$.

Although local variables may be contaminated by corrupted messages, we can handle such a case as corruption of local variables. We have the following two lemmas to show recovery from corruption of local variables.

**Lemma 8:** For any $p_i \in U(t)$ and for any $p_j$ such that $p_j \not\in U(t')$ for all $t' (0 \leq t - T_N \leq t' \leq t)$, $p_j$ is not in a set $\{p : (p, *) \in N_i(t)\}$.

**Proof:** Because the value of local variable $N_i$ is affected only by names messages, we consider the name thread only. Consider an execution starting at time $t_0 \geq 0$ with arbitrary configuration in which messages and local variables may be just corrupted, and we assume that no transient faults occur after time $t_0$.

Suppose contrary that the lemma is false. Let $p_i$ (resp. $p_j$) be a process such that $p_i \in U(t)$ (resp. $p_j \not\in U(t)$) for each $t_0 \leq t \leq t_0 + T_N$. Although $p_i$ may include an entry $(p_j, t)$ in $N_i$ for some $0 \leq t \leq T_N$ at time before $t_0$ by faults, such an event never occur at time $t$ such that $t \geq t_0$ because no faults occur any more.

For each time $t$ and process $p_j$, let $M(t, p_j) = \max\{0, \ell_j : (p_j, \ell_j) \in N_i, p_i \in U(t)\}$, which is the maximum TTL value of items for $p_j$ in all $N_i$s at time $t$. Because the maximum TTL values for $p_j$ is decreasing regardless transmissions of $N_i$ by names messages, we have $M(t, p_j) > M(t + 1, p_j)$ if $M(t, p_j) > 0$. When $M(t, p_j) = 0$, we have $M(t, p_j) = M(t + 1, p_j)$. Thus, $M(t_0 + T_N, p_j) = 0$ holds, and no process holds an entry for $p_j$ in $N_i$ at time $t_0 + T_N$.

**Lemma 9:** For any $p_i \in U(t)$ and for any $p_j$ such that $p_j \not\in U(t')$ for all $t' (0 \leq t - T_I \leq t' \leq t)$, $p_j$ is not in the set $\{p : (p, *) \in X_i(t)\}$.

**Proof:** Proof of this lemma is omitted since it is similar to the proof of Lemma 8.

**Theorem 5:** The proposed protocol is self-stabilizing with respect to $S$, and convergence (recovery) time is $O(\max(T_N, T_I))$.

**Proof:** By Lemma 7, garbage messages are received at time $t_0 + 1$. By receiving corrupted messages, local variables may be contaminated. By Lemmas 8 and 9, any contaminated and corrupted items in local variables are eliminated by time $t + \max(T_N, T_I) + 1$. It is clear that once the system converges, it remains so forever.

Once the system converges, by Theorem 1, a process finds an object with success probability $\sigma$, because the index thread at each process sends index information of an object it possesses at every $T_I$ time steps.

Note on the values of $T_N$ and $T_I$: The value of $T_N$ must be larger than the time to collect process names by the name thread. The value of $T_I$ should be determined by a trade-off of network traffic by index messages and validity of object index information in $X_i$. If $T_I$ is small, many index messages are transmitted, and if it is large, index information of an object at a process that is no longer in the system is not eliminated from $X_i$ for a long time.

**APPENDIX II**

**IMPLEMENTATION ISSUE**

Since the protocol and assumption described in this section are theoretical and unrealistic, we cannot directly implement the proposed protocol. We describe below some idea for modification of the protocol to implement in practice. Although there can be many heuristics based on the real world to improve its performance, such discussion is out of the scope of this paper.

On $N_i^0$: We assumed that $N_i^0$, an initial set of processes that $p_i$ knows, is given such that $N_i^0$ is taken uniformly at random from $U$. To make this possible for each process, each process must know $U$ correctly.

In reality, we may use host catchers that maintain a large set of processes. Let $M_i^0$ be a set of host catchers that process $p_i$ knows. When $p_i$ joins a P2P system, it contacts with some host catchers randomly selected from $M_i^0$, and asks each host catcher a random set of processes. Let $M_i^1$ be a set of processes obtained. By repeating this procedure $t$ times for some $t \geq 1$, we obtain a set of processes $M_i^t$. Then we select $r$ processes randomly from $M_i^t$, and use it as $N_i^0$ for the name thread. It is clear such a scheme does not offer uniform randomness (randomness and $t$ is a trade-off), we can use it as an approximation.

On $\rho$: We discuss estimation of $\rho = n'/n$ (replication factor). In real life, replication factors are different by objects. When a P2P system starts, the value of $\rho$ must be determined in advance. It is clear from context, object with replication...
factor larger than $\rho$ is found with probability larger than $\sigma$. The problem is the case when $\rho$ is smaller than estimated. A simple way to overcome this is retry of query with larger quorum size. A larger quorum is more likely to intersect with a quorum selected by a process that holds rare objects. Thus, popular objects are searched with small overhead, and rare objects are searched with reasonable overhead.

**On object update:** In real situation, objects can be updated. We can modify the protocol to find the latest version as follows. A timestamp of revision is attached to each object, and the index thread (Figure 4) is modified so that it also manages a timestamp of each object. When a process searches an object, it selects the latest one among replies.

**On turnover rate:** In a real system, processes may join and leave frequently. Our protocol can continue object search if small fraction of processes join and leave frequently because such a situation results in small loss of quorum members. That is, although the probability that an object is found becomes small, the protocol is not broken down. On the other hand, large fraction of processes join and leave frequently, most of the process name set $N_i$ is no longer in the system, and the success probability of object search becomes quite small. A practical way to keep success probability is to enlarge quorum size that depends on turnover rate.

**On size of $R_i$:** In the description above, the size of $R_i$ for each process $p_i$ is exactly $r(\geq q)$. In practice, the larger is the better for the size of $R_i$. We will discuss the size of $R_i$ formally in subsection V-B.

**On underlay protocol:** Because each process in our protocol must communicate many processes, UDP should be used rather than TCP because the number of simultaneous TCP connections is limited and TCP has large overhead.