SCALLOP: A Scalable and Load-Balanced Peer-to-Peer Lookup Protocol

Jerry C.~Y. Chou, Member, IEEE, Tai-Yi Huang, Member, IEEE, Kuang-Li Huang and Tsung-Yen Chen
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan 300, R.O.C
Abstract

Many large-scaled servers are implemented as a peer-to-peer (P2P) distributed system to avoid centralized management and increase scalability. Traditional P2P lookup protocols often utilize binomial lookup trees to shorten lookup paths. Such a lookup protocol introduces routing bottlenecks when certain nodes become hot spots. In this paper, we present a scalable and load-balanced P2P lookup protocol called SCALLOP. This protocol utilizes balanced lookup trees to evenly distribute routing traffic among nodes and, therefore, reduce or eliminate the occurrence of routing bottlenecks. SCALLOP achieves scalability by storing in each node $O(\log N)$ routing information in an $N$-node distributed system. Furthermore, a self-organized mechanism is proposed to efficiently recover a system when nodes join, leave, and fail dynamically. We conduct a series of experiments to compare SCALLOP and Chord, the most-referenced and representative binomial lookup protocol. The experimental results show that, with balanced lookup trees, SCALLOP delivers more balanced routing loads and avoids routing bottlenecks. In addition, its customizable feature reduces lookup paths and the total number of routed requests at the cost of larger lookup tables.

Index Terms

peer-to-peer distributed systems, lookup protocols, routing bottlenecks, load-balanced protocols

I. INTRODUCTION

Many large-scaled servers are often implemented in the manner of a peer-to-peer (P2P) distributed system due to the low cost of workstations and the availability of high-speed networks. Examples include web servers and storage servers [26], [12], [1], [27], [6], [7], [18], [3]. A P2P system is one without any centralized control or hierarchical organization such that each node is equivalent in functionality. A P2P lookup protocol resolves a lookup request in a P2P system to locate the target node storing the requested item. Without centralized information, a lookup request may be routed several times before it reaches its target node. Consequently, achieving rapid lookup response requires a lookup protocol that shortens lookup paths and evenly distributes routing traffic to avoid routing bottlenecks.

Traditional P2P lookup protocols often utilize a binomial lookup tree to route a lookup request. Such a lookup protocol delivers bounded lookup paths at $O(\log N)$, where $N$ is the number of nodes in a system. However, it ignores the issue of routing bottlenecks by assuming that lookup requests are evenly targeted at every node and there is no hot spot. A node becomes a hot spot
when a large number of requests targeting at this node arrive in the system simultaneously. Due to its unbalanced feature, a binomial lookup protocol introduces routing bottlenecks when certain nodes become hot spots. A routing bottleneck prolongs the response time of a lookup request and hinders scalability.

In this paper, we present SCALLOP, a SCALable and LOad-balanced P2P lookup protocol. Based on the technique of distributed hash table (DHT), SCALLOP distributes routing information among nodes and requires no centralized management. SCALLOP constructs a balanced lookup tree for each node to route lookup requests targeting at this node. The balanced lookup tree of a node evenly distributes routing traffic to avoid routing bottlenecks when this node becomes a hot spot. In addition, the degree $d$ of a balanced lookup tree is customizable to trade between memory space and lookup performance. The routing information required by a node is called its lookup table. SCALLOP is scalable because each node has $O(d \log_d N)$ entries in its lookup table, and a lookup request reaches its target node within $O(\log_d N)$ hops. The larger $d$ is, the more entries a lookup table has, and the smaller hops a lookup request takes to reach its target node. To further reduce lookup path, we present a bidirectional model of SCALLOP to allow a lookup request to be forwarded in both directions. Finally, to be suitable for a large-scaled distributed system where nodes may join, leave, and fail dynamically, we provide a self-organized mechanism to update lookup tables efficiently.

We conducted a series of experiments to compare the performance of SCALLOP and Chord [24], the most-referenced and representative binomial lookup protocol. We first measure its effectiveness in evenly distributing routing traffic and avoiding bottlenecks. The experiments simulate both a single hot spot and a popular 80/20 scenario where 80% of requests are served by 20% of nodes. In both cases, SCALLOP delivers more balanced routing loads and reduces or eliminates the occurrence of routing bottlenecks. We next demonstrate the performance of SCALLOP in dealing with one-node and multi-node status changes. We measure the time it takes to update all lookup tables and the increased percentage of average lookup path. Compared with Chord, SCALLOP takes less time to update and increases average path length by a smaller percentage. Finally, we show the customizable performance of SCALLOP in reducing lookup paths and total routed requests.

The rest of the paper is structured as follows. Section 2 describes the basics of SCALLOP. Section 3 presents the bidirectional SCALLOP model. Section 4 describes the self-organized
mechanism for a dynamic system where nodes change status dynamically. The experimental results are presented in Section 5. Section 6 describes related work. Finally, Section 7 concludes this paper.

II. THE SCALLOP LOOKUP PROTOCOL

We begin with the description of the mechanism of assigning data items to nodes. Next, we describe the construction of a balanced lookup tree of a node. Each lookup table is then built using the distance relationship in each balanced lookup tree. Finally, we use an example to illustrate how SCALLOP resolves a lookup request.

A. Balanced Data Distribution

Our protocol uses consistent hashing [10] to assign data items to nodes. Each data item is stored in one node with no replication. Each data item and each node is first given an $m$-bit identifier. The identifier of a node, called its virtual identifier or VID, is obtained by hashing the IP address of the node. The identifier of a data item, called its key, is obtained by hashing the unique information of the data item such as its URL address. The consistent hashing requires that $m$ be large enough to make the probability of two nodes hashing to the same identifier negligible. Thus, we assume that each VID is unique in the system. For the sake of simplicity, we will use the term “key” to refer to both a data item and its identifier.

Our protocol arranges all the nodes in an identifier circle according to their VID modulo $2^m$. In this identifier circle each node has two neighboring nodes: its successor (the first node clockwise from it) and its predecessor (the first node counterclockwise from it). Our protocol stores the key $k$ in a node with the VID $n$ where the absolute distance $|k - n|$ is minimal. In the case of a tie, we store the key in the node clockwise from $k$. Figure 1 shows an example of 3 nodes storing 4 keys. The VIDs of the three nodes are 1, 3, and 7, and the keys are 0, 4, 5, and 6. The key 4 is stored in node 3, and the keys 5 and 6 are stored in node 7.

Two important (and proven) properties of consistent hashing make it a data-balanced and scalable hash function. First, consistent hashing tends to evenly distribute all keys among nodes such that each node stores a similar number of keys. Second, it requires relatively little movements of keys when a node joins or leaves. We will discuss how SCALLOP moves keys in the case
Fig. 1. The data distribution in a system with 3 nodes and 4 keys

Fig. 2. The lookup tree of node $k$

of node joining or node leaving in Section IV. Readers who are interested in the proof of these two properties are referred to [10].

B. Balanced Lookup Trees

In addition to the VID, our protocol assigns each node another identifier called SID (SCALLOP identifier). The node with the smallest VID has the SID 0, and each of the rest nodes is assigned a SID equal to its relative position clockwise from the node of SID 0 in the identifier circle. For example, in the 3-node identifier circle shown in Figure 1, the node of VID 1 has the SID 0, and the node of VID 7 has the SID 2. To simplify the presentation, we will use the term “node $k$” to refer to the node of SID $k$ for the rest of the paper unless otherwise specified.

To make our protocol scalable, we use the DHT technique to scatter the routing information
among all nodes. Each node stores routing information about only a few other nodes. Consequently, a lookup request may be forwarded several times before it reaches its target node. We call the collection of all routing information regarding the same target node the lookup tree of the node. The root of the lookup tree of a node is the node itself. SCALLOP constructs a unique and balanced lookup tree for each node. Every key stored in the same node uses the lookup tree of the node. Each lookup tree is balanced to evenly distribute routing traffic. In addition, our protocol allows a customized degree $d$ of a lookup tree.

Figure 2 shows the generic lookup tree of node $k$ in an $N$-node system. The node identifier used in this figure is SID. The degree of the lookup tree is $d$; each node has at most $d$ children nodes, and the number of nodes in the $i$-th level is at most $d^i$, $i \geq 0$. The root of the lookup tree is node $k$ itself. The rest $N - 1$ nodes are laid out counterclockwise from node $k$ in a breadth-first and evenly distributed manner. Figure 3(a) and (b) show the lookup trees of node 0 and node 1, respectively, in a 16-node system with $d = 4$. When node 8 receives a lookup request whose target node is node 0, it forwards the lookup request to node 12, which in turn forwards to node 0. On the other hand, if its target node is node 1, node 8 forwards the request to node 0. Because of balanced trees, the sibling nodes of a lookup tree route a similar number of lookup requests. The even distribution of routing traffic reduces or eliminates the occurrence of routing bottlenecks. The length of a lookup path is the number of hops for a request to reach its target node, and is therefore bounded by the depth of the lookup tree. SCALLOP bounds the lookup path by $O(\log_d N)$. The larger $d$ is, the lower the lookup path is bounded.

C. Construction of Lookup Table

The lookup table of a node stores the routing information to resolve lookup requests. A lookup table is a list of $(target, forwarder)$ pairs, where $target$ is the node storing the requested key, and $forwarder$ is the next node in the lookup path to the $target$ node. For instance, by the lookup tree shown in Figure 3(a), node 5 has a $(0, 13)$ pair in its lookup table as node 13 is the next node in the lookup path from node 5 to node 0. Similarly, node 0 has a $(1, 1)$ pair in its lookup table according to the lookup tree shown in Figure 3(b). We can obtain the lookup table for each node by examining all lookup trees. Table I lists the lookup table of node 0 as an example. Column 2 of Table I shows the range of targets that use the same forwarder given
in Column 3 of the same table. For example, the entry[4] shows that, given a lookup request, if the target node of the request is larger than or equal to 4 and smaller than or equal to 8, the request will be forwarded to node 4.

Instead of examining every lookup tree, we utilize distance relationship between nodes to provide a generic lookup table. Given $N$ and $d$ in a P2P system, we use this generic table to construct lookup tables for each node without examining any lookup tree. Furthermore, the generic lookup table bounds the memory requirement for lookup tables.

We replace each node in the generic lookup tree shown in Figure 2 with a pair of $(D2T, D2F)$, where $D2T$ is the clockwise distance between the node and node $k$ (i.e., the target node), and $D2F$ is the clockwise distance between the node and its parent node (i.e., the forwarder node). We call the transformed tree the distance tree. Figure 4 shows the distance tree for the generic lookup tree in Figure 2. We observe that any pair of (D2T, D2F) is independent of $k$, the root of the generic lookup tree. Consequently, there is only one unique distance tree in an $N$-node system although there are $N$ different lookup trees. Furthermore, each node has a unique D2T
<table>
<thead>
<tr>
<th>entry[i]</th>
<th>entry[i].range</th>
<th>entry[i].forwarder</th>
</tr>
</thead>
<tbody>
<tr>
<td>entry[0]</td>
<td>[0,0]</td>
<td>0</td>
</tr>
<tr>
<td>entry[1]</td>
<td>[1,1]</td>
<td>1</td>
</tr>
<tr>
<td>entry[2]</td>
<td>[2,2]</td>
<td>2</td>
</tr>
<tr>
<td>entry[3]</td>
<td>[3,3]</td>
<td>3</td>
</tr>
<tr>
<td>entry[4]</td>
<td>[4,8]</td>
<td>4</td>
</tr>
<tr>
<td>entry[5]</td>
<td>[9,12]</td>
<td>8</td>
</tr>
<tr>
<td>entry[6]</td>
<td>[13,15]</td>
<td>12</td>
</tr>
</tbody>
</table>

**TABLE I**

The lookup table of node 0 in a 16-node system with d = 4

![Distance Tree Diagram](image)

Fig. 4. The unique distance tree

value in the distance tree. Figure 3(c) shows the distance tree in a 16-node system as an example.

We use D2T\[i, j, l\] and D2F\[i, j, l\], \(l \geq 1\), to denote the D2T and the D2F values of the node located at the \(l\)-th level, the \(j\)-th subtree of the level, and the \(i\)-th node of the subtree, respectively. A subtree is a set of nodes that have the same parent in the same level. Take Figure 3(c) as an example. D2T\[1, 3, 2\] is equal to 7 and D2F\[1, 3, 2\] is equal to 4. We can obtain the expression of D2T\[i, j, l\] and D2F\[i, j, l\] by the following two theorems.

**Theorem 1:** The expression of D2T\[i, j, l\], \(l \geq 1\), in the distance tree is

\[
\text{D2T}[i, j, l] = \sum_{x=0}^{l-1} (d^x) + (j - 1) + (i - 1)d^{l-1}. \quad (1)
\]

**Proof:** The D2T value of the leftmost node in \(l\)-th level can be represented by D2T\[1, 1, l\].
Because there are totally $d^l$ nodes in $l$-th level, we obtain

$$D2T[1, 1, l] = \sum_{x=0}^{l-1} (d^x).$$

Because SCALLOP lays out the nodes in a breadth-first and evenly distributed manner, we know $D2T[1, j, l] = D2T[1, j - 1, l] + 1$. Therefore,

$$D2T[1, j, l] = \sum_{x=0}^{l-1} (d^x) + (j - 1).$$

Finally, because there are $d^{l-1}$ subtrees at $l$-th level, we have $D2T[i, j, l] = D2T[i - 1, j, l] + d^{l-1}$. Consequently, we prove that

$$D2T[i, j, l] = D2T[1, j, l] + (i - 1)d^{l-1} = \sum_{x=0}^{l-1} (d^x) + (j - 1) + (i - 1)d^{l-1}.$$

\[\Box\]

**Theorem 2:** The expression of $D2F[i, j, l]$, $l \geq 1$, in the distance tree is

$$D2F[i, j, l] = id^{l-1}. \quad (2)$$

**Proof:** Let node $m$ in the generic lookup tree be replaced by $(D2T[i, j, l], D2F[i, j, l])$ in the distance tree. Let node $m'$ be the parent of node $m$ and be replaced by $(D2T[i', j', l'], D2F[i', j', l'])$ in the distance tree. Because $D2F[i, j, l] = m' - m = (k - m) - (k - m')$, we know that

$$D2F[i, j, l] = D2T[i, j, l] - D2T[i', j', l'].$$

It is obvious that

$$l = l' + 1.$$

In addition, because each node in the level of node $m'$ will have a subtree at the level of node $m$, we have

$$j = D2T[i', j', l'] - D2T[1, 1, l'] + 1 = j' + (i' - 1)d'^{l'-1}.$$

Based on the observations described above, we obtain the definition of $D2F[i, j, l]$. 

July 13, 2004
\[
D2F[i, j, l] = D2T[i, j, l] - D2T[i', j', l']
\[
= \left[ \sum_{x=0}^{l-1} d^x + (j - 1) + (i - 1)d^{l-1} \right] - \left[ \sum_{x=0}^{i'-1} d^x + (j' - 1) + (i' - 1)d^{l'-1} \right]
\]
\[
= id^{l'} = id^{l-1}.
\]

**Theorem 3:** The mappings between D2T and D2F in the distance tree can be represented by a generic lookup table with \(O(d \log_d N)\) or \(O(\log N)\) entries.

**Proof:** We summarize the mappings between D2T and D2F in Table II. Each row of Table II represents the mappings in a specific level in the distance tree. According to Theorem 2, the value of \(D2F[i, j, l]\) is independent of \(j\). In other words, at the same level, all nodes with the same \(i\) have the same D2F value. Therefore, we divide the possible D2T values at the same level into \(i\) different ranges, each range sharing the same D2F value. Column 2 of Table II shows the range of D2T values that map to the same D2F given by Column 3 of the same table. Each range of D2T values is between \(D2T[i, 1, l]\) and \(D2T[i, d^{l-1}, l]\). Because there are at most \(d\) such entries in each level, the number of entries in Table II is bounded by

\[
O(dl) = O(d \log_d N) = O(\log N).
\]

To construct the lookup table of node \(k\), we add \(k\) to each D2T range and D2F value given in Table II module \(N\). For example, by adding 0 to Table II, we obtain the lookup table shown in Table I. We can construct the lookup tables for the rest \(N - 1\) nodes in a similar way. By Theorem 3, a lookup table has \(O(d \log_d N)\) or \(O(\log N)\) entries. In other words, SCALLOP is scalable such that each node only needs to know the routing information about \(O(\log N)\) other nodes.

**D. Lookup Resolution**

Figure 5 shows an 8-node system with \(d = 2\). Each lookup table shows target ranges and forwarders in both SIDs and VIDs. The ranges and forwarders in SIDs are obtained by adding
<table>
<thead>
<tr>
<th>level</th>
<th>D2T range</th>
<th>D2F value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[0, 0]</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>[i, i]</td>
<td>i</td>
</tr>
<tr>
<td>2</td>
<td>[id + 1, id + d]</td>
<td>id</td>
</tr>
<tr>
<td>3</td>
<td>[id^2 + 1, id^2 + d^2]</td>
<td>id^2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

| l     | \[\sum_{x=0}^{l-1} (d^x) + (i-1)d^{l-1}, \sum_{x=0}^{l-1} (d^x) + id^{l-1} - 1] | id^{l-1} |

**TABLE II**

The generic lookup table, \( i = 1, 2, \ldots, d \)

the SIDs of a node to the generic lookup table shown in Table II. The ranges and forwarders in VIDs are obtained by mapping SIDs to VIDs, which will be described later in the discussion of joining a node.

Let node 4 receive a lookup request whose key is 13. This data item is stored in node 7 (VID 14) according to the data distribution protocol described in Section II-A. Node 4 determines that key 13 is in the VID range of \([9, 1]\) and forwards the request to node 6 (VID 10). Similarly, node 6 forwards the request to node 7 that in turn returns the data item. Because a data item is stored in a node with minimal distance, we may need to backward a request to its predecessor.

Let node 6 receive a lookup request whose key is 11. Although the data item is stored in node 6, the request will still be forwarded to node 7 (VID 14). After an unsuccessful search, node 7 learns that the data item is stored in node 6 and sends a backward request to node 6. Upon receiving a backward request, node 6 conducts a search and returns the data item.

**III. BIDIRECTIONAL MODEL**

The basic SCALLOP model forwards a lookup request clockwise until it reaches its target node. To further reduce lookup path, we present the bidirectional SCALLOP model that allows a lookup request to be forwarded either clockwise or counterclockwise. In the bidirectional model, each node divides the identifier circle into two symmetrical regions, each of which starts with the
Fig. 5. An 8-node system with $d = 2$ node itself and contains $N/2$ nodes. The region in clockwise direction is called the positive region and the other region is called the negative region. A node maintains a lookup table for each region.

Let $k$ denote the SID of a node. Similar to the basic model, the lookup table of the positive region is obtained by $k \oplus i$ the generic lookup table shown in Table II, where $i \oplus j = (i + j) \mod N$. By replacing the clockwise distance with the counterclockwise distance, we resolve a lookup request in the negative region as in the positive region. Consequently, we obtain the lookup table of the negative region by $k \ominus i$ the generic lookup table, where $i \ominus j = (i - j + N) \mod N$. Figure 6 shows the lookup tables for node 0 in an 8-node bidirectional system with $d = 2$ as an example. Node 0 uses the positive lookup table to forward a request if its key falls in the positive region. Otherwise, node 0 uses the negative lookup table to resolve the lookup request.

The bidirectional model reduces the lookup path by limiting a key search in $N/2$ nodes. The improvement is achieved at the cost of doubling the memory requirement for lookup tables. In addition, because $k \oplus i = m$ implies $m \ominus i = k$, the bidirectional model exhibits routing symmetry. By routing symmetry, we mean that every edge in a lookup tree is bidirectional. In other words, for any two nodes $n$ and $m$, if $n$ is a forwarder in $m$'s lookup tables, $m$ is also a forwarder in $n$'s lookup tables. The routing symmetry allows the root node of a lookup tree to propagate information downstream, which is not possible in the basic model. Consequently,
it significantly reduces the number of messages to update each node’s lookup tables when a system changes dynamically, as described in the next section.

IV. SELF-ORGANIZED MECHANISM

Nodes join, leave, and fail dynamically in a P2P system. To maintain SCALLOP integrity, we need to update lookup tables each time a node joins or leaves the system. In the following, we first describe the simple procedure to recover the system at the event of one-node change. We next describe the advanced procedure to deal with multi-node changes that take place concurrently.

A. Simple Stability Procedure

We describe the stability procedure to join a node. A similar but reverse procedure is taken when a node is leaving the system. The simple stability procedure updates each lookup table concurrently by requiring that a new lookup table of a node depend only on the old lookup tables of the node itself and its adjacent nodes (i.e., its predecessor and its successor). For this reason, each node maintains a link to its adjacent nodes.

When a node wants to join an N-node system, it first obtains its VID by hashing its IP address. Let \( r \) denote the joining node and \( r.\text{VID} \) denote its VID. The joining node next issues a \texttt{FindPredecessor}(\( r.\text{VID} \)) request into the system which in turn returns the predecessor node \( s \) of \( r \). Depending on the SID and VID of \( s \), the joining node \( r \) determines its location in

Fig. 6. Lookup tables for node 0 in the bidirectional model

<table>
<thead>
<tr>
<th>Entry</th>
<th>Target range (SID)</th>
<th>Target range (VID)</th>
<th>Forwarder (SID)</th>
<th>Forwarder (VID)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[0,0]</td>
<td>(14,1]</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>[1,1]</td>
<td>(1,3]</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>[2,4]</td>
<td>(3,7]</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Entry</th>
<th>Target range (SID)</th>
<th>Target range (VID)</th>
<th>Forwarder (SID)</th>
<th>Forwarder (VID)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[0,0]</td>
<td>(14,1)</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>[7,7]</td>
<td>(10,14)</td>
<td>7</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>[4,6]</td>
<td>(7,10]</td>
<td>6</td>
<td>10</td>
</tr>
</tbody>
</table>
the identifier circle and sets its SID, denoted by \( r.\text{SID} \). Both \( r \) and \( s \) update their predecessor and successor links accordingly. Finally, the adjacent nodes of \( r \) migrate a data item to \( r \) if its key is closer to \( r.\text{VID} \).

Each node needs to be aware of \( r.\text{SID} \) and \( r.\text{VID} \) to update its lookup tables. In the basic SCALLOP model, a node propagates this information either through forwarders or adjacent nodes. The former approach reaches every node in \( O(\log N) \) hops but generates many redundant messages. On the other hand, the latter approach takes \( O(N) \) hops to deliver a message to every node. In contrast, the bidirectional model takes advantage of routing symmetry to propagate information in one lookup tree. According to the generic lookup tree shown in Figure 2, \( r \) must be a leaf node in the lookup tree of \( s \). In other words, the lookup tree of \( s \) remains intact at the join of \( r \). The information of \( r.\text{SID} \) and \( r.\text{VID} \) can be propagated downstream the lookup tree of \( s \). All nodes are reached in \( O(\log N) \) hops and no redundant message is generated.

Upon receiving the node-join message, a node takes three steps to update its lookup tables. It first updates its SID based on its relative position to \( r.\text{SID} \). It next applies \( \oplus \) and \( \ominus \) operations on its new SID and the generic lookup table shown in Table II module \( N + 1 \) to obtain the new lookup tables. Because the lookup tables of adjacent nodes are offset by 1, we complete the SID-to-VID mapping in the new lookup tables using the information in the old lookup tables of a node itself and its adjacent nodes. Therefore, to be capable for each node of updating its lookup table concurrently, SCALLOP requires that each node keep its old lookup tables until its adjacent nodes complete the SID-to-VID mapping.

Nodes that fail without any notice cause incomplete lookup trees and broken lookup paths. To detect such failed nodes and recover SCALLOP promptly, each node periodically sends out a \textit{hello} message to its adjacent nodes and waits for a reply. If a node fails to reply within a timeout period, the node is treated as a failed node. Either the predecessor or the successor of a failed node is responsible for initiating a node-leave stability procedure to reconstruct the lookup tables.

\textbf{B. Advanced Stability Procedure}

When multi-node changes are possible, a more complicated process is developed to reconstruct the lookup tables of a node as its adjacent nodes may change simultaneously. The advanced stability procedure requires no more than \( k \) contiguous nodes in the identifier circle should fail
in a recovery period. Let a node fail with probability $1/p$. Accordingly, the probability of $k$ contiguous failed nodes is $p^{-k}$. An appropriate value of $k$ is chosen to approach $p^{-k}$ to 0.

The advanced procedure follows similar steps to join or leave nodes. Each joining or leaving node propagates a message to every other node and, upon receiving such a message, each node rebuilds its lookup table using its new SID and the generic lookup table. The SID-to-VID mapping cannot be completed by the old lookup tables of its adjacent nodes due to the scenario of multi-node changes. Instead, each node issues a $\text{SID2VID}(t)$ request to query the VID of the node with SID $t$. Unlike $\text{FindPredecessor}$ that is resolved by VID information of each lookup table, $\text{SID2VID}$ is resolved by SID information of each lookup table. After the $\text{SID2VID}(t)$ request reaches the node with SID $t$, it returns its VID to the querying node which in turn updates its lookup table. Similar to $\text{FindPredecessor}$, $\text{SID2VID}$ is a lookup request whose path is bounded by $O(\log N)$.

Three different selecting policies are proposed for one node to complete SID-to-VID mapping. The first policy is to treat all lookup table entries as a circular list and use $\text{SID2VID}$ to update each entry in a rotational manner. To save bandwidth, only a fixed number of entries are updated in a period. Eventually, all nodes will complete SID-to-VID mapping and the system becomes stabilized. The second policy is to assign each entry a probability and update them according to a probability distribution. Because entries with closer SIDs may influence the path length more seriously, higher probabilities are assigned to them. The last policy is to assign an equal probability to each entry and randomly choose a few entries to update. Section V evaluates the performance of each policy in stabilizing a dynamic system. The experimental results show that it takes a similar number of messages for each of three policies to achieve stability.

To detect a failed node in the environment of multi-node changes, the advanced procedure requires that each node maintains a predecessor queue of $k$ entries, each of which is a pointer to a predecessor. In other words, each node is pointed by its next $k$ successors. Because no more than $k$ contiguous nodes should fail simultaneously, a failed node is always detected by a successor. The successor propagates a node-leave message to reconstruct lookup tables.

Finally, we need to prove that a lookup request can be resolved successfully and efficiently in a dynamic environment of nodes joining, leaving, and failing concurrently. Such a stability proof requires careful analysis and lengthy presentations. Fortunately, SCALLOP and Chord [25] carry the following similarities:
The lookup path is bounded by $O(\log N)$,

- data is distributed by consistent hashing on an identifier circle,
- each node maintains a processor queue of $k = O(\log N)$ entries, and
- each node has a link to its immediate successor.

The technical report [25] proves that, with high probability, Chord resolves a lookup request in $O(\log N)$ hops in a dynamic environment. With the similarities required for a stability proof, we prove that SCALLOP exhibits the same stability.

V. EXPERIMENTAL RESULTS

We conducted a series of experiments to demonstrate the effectiveness of SCALLOP in a large-scaled P2P system. We first measure how SCALLOP balances routing traffic around hot spots and avoids routing bottlenecks. A couple of cases are discussed: a single hot spot phenomenon where a large percentage of lookup requests demand for data items stored in one node and the 80/20 scenario where 80% of requests are served by 20% of nodes. We next show the performance of the self-organized mechanism in dealing with one-node and multi-node changes. Here we measure the time to update all lookup tables and the increased percentage of average lookup path. Finally, we show the customizable performance of SCALLOP using different degree $d$ of a balanced lookup tree.

A binomial lookup tree is often used in a P2P lookup protocol. Such a protocol includes, but not limited to, Chord [24], Tapestry [29], Pastry [22], AChord [8], and Hyperchord [14]. Among them, Chord [24] is the most-referenced and representative binomial-tree lookup protocol. For this reason, we compare SCALLOP and Chord in each of the following experiments to demonstrate the performance impact of balanced lookup trees.

A. Single Hot Spot Phenomenon

Our experiments model a P2P system of $10^4$ nodes receiving $10^6$ lookup requests concurrently. Figure 7(a) compares the performance of SCALLOP ($d = 4$) and Chord when all requests are evenly distributed among all nodes. In other words, Figure 7(a) gives the performance when there is no hot spot phenomenon and no routing bottleneck. We measure the performance of each node in terms of the number of lookup requests routed by the node. To demonstrate the difference, we only show the load of the nodes that matter. In this case, they are nodes 9900 to
Fig. 7. Performance comparison when node 0 becomes a hot spot

9999. Figure 7(a) shows that Chord, a binomial-tree protocol, delivers fluctuated loads among nodes. In comparison, SCALLOP delivers more balanced loads because of balanced lookup trees.

To simulate a single hot spot phenomenon, we further let a certain percentage of requests demand for data items stored in a particular node, node 0 in this case, and the rest of the requests demand for data items evenly scattered among all other nodes. Obviously, node 0 becomes a hot spot. Figure 7(b), (c), and (d) show the performance when 10%, 20%, and 30% of lookup requests demand for data items stored in node 0, respectively. In each scenario, Chord routes a large number of lookup requests through node 9999 which becomes a routing bottleneck. SCALLOP eliminates such a bottleneck by significantly reducing its routed requests. Table III lists the number of routed requests by node 9999 in each experiment. During the scenario of 30% concentration, SCALLOP routes 75,449 requests through this node, a 49% reduction from 148,621 requests by Chord. We expect an even larger reduction by SCALLOP with a larger d.
Table III: The number of lookup requests routed by node 9999

<table>
<thead>
<tr>
<th>Concentration</th>
<th>0%</th>
<th>10%</th>
<th>20%</th>
<th>30%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chord</td>
<td>523</td>
<td>52,479</td>
<td>103,937</td>
<td>148,621</td>
</tr>
<tr>
<td>SCALLOP</td>
<td>565</td>
<td>25,597</td>
<td>50,753</td>
<td>75,449</td>
</tr>
<tr>
<td>Reduced (%)</td>
<td>-8%</td>
<td>51%</td>
<td>51%</td>
<td>49%</td>
</tr>
</tbody>
</table>

B. 80/20 Scenario

Figure 8 compares the performance of SCALLOP and Chord under the popular 80/20 scenario where 80% of requests are served by 20% of nodes. Similar to the setup in Section V-A, we model a P2P system of $10^4$ nodes receiving $10^6$ lookup requests concurrently. However, 80% of requests demand for data items stored in 20% of nodes while the rest 20% of requests evenly target at the rest 80% of nodes. Again, we measure the number of lookup requests routed by each node. Figure 8(a) records the performance of Chord. Figure 8(b) and (c) record the performance of SCALLOP with $d = 2$ and $d = 4$. The number of routed requests by Chord ranges between 0 and 6,000 and varies drastically between different nodes. In contrast, SCALLOP ($d = 2$) bounds the range of loads between 700 and 2200 and SCALLOP ($d = 4$) further reduces the range within 400 and 1500.

Figure 8(d) summarizes this experiment by listing the number of routing bottlenecks. Among all $10^4$ nodes, Chord requires 111 nodes to route more than 3,000 lookup requests. In addition, it requires 14 nodes to route more than 4,000, and 5 nodes to route more than 5,000 lookup requests. On the other hand, no node is routing more than 2,500 requests with SCALLOP. With a larger $d$, SCALLOP reduces the peak load per node to less than 1,500 requests. If a node becomes a routing bottleneck when receiving more than 3,000 requests in a period of time, 111 nodes are bottlenecks by Chord while none by SCALLOP. The experimental results demonstrate that, by constructing balanced lookup trees, SCALLOP delivers more balanced loads among nodes. In addition, SCALLOP successfully reduces or eliminates the number of routing bottlenecks in a popular 80/20 scenario.
C. One-Node Change

This experiment demonstrates the efficiency of the simple stability procedure in stabilizing a system when a node joins, leaves, or fails dynamically. We evaluate its performance in terms of time to update all lookup tables. The simple stability procedure takes two steps to complete a recovery process. First, it must propagate a message to every node. Upon receiving this message, each node updates its lookup table by the old lookup tables of itself and its adjacent nodes. Because lookup tables are updated concurrently, a recovery process is bounded by the time of propagating a message to every node.

Both SCALLOP and Chord are evaluated by the number of hops to propagate an update message to every node. SCALLOP operates in the bidirectional model such that a message is propagated downstream a balanced lookup tree. Figure 9 shows the experimental results. The number of nodes in a system ranges from 10,000 to 1,000,000. SCALLOP takes exactly \( \log_d N \) hops to reach every node, where \( N \) denotes the number of nodes in the system. On the other
hand, Chord takes a lookup path, bounded by $\log_2 N$, and a linear search for a node to deliver a message to any other node. Consequently, SCALLOP takes smaller hops and less time to update all lookup tables and stabilize the system. Particularly, with $d = 4$, SCALLOP reaches every node at the cost of less than half of what’s required by Chord.

D. Multi-Node Changes

We now consider a system where multi-node changes are possible at the same time. Our experiment models a P2P system of 1,000 nodes receiving 1,000 lookup requests per time unit. The experiment lasts for 100 time units. During time 0 to 10 and 50 to 100, no node changes. A certain percentage of nodes join, leave, or fail between time 10 to 50. We conducted two sets of experiments. The first set simulates 20% of nodes changing status for every 10 time units. The second set increases the percentage of changing nodes to 30%. We evaluate the performance of the advanced stability procedure in terms of table fault rate. By table fault rate, we mean the ratio of the number of incorrect table entries to the number of entries in all lookup tables. All three updating policies described in Section IV-B deliver similar performance in our measurements. For simplicity, we only show the performance of the random policy that assigns an equal probability to each entry and randomly chooses 5 entries to update its SID-to-VID mapping per time unit.

Figure 10 shows the experimental results where $d = 2$. The table fault rate is 0% from time 0 to 10 as no node changes. At time 10, the first group of changes arrives and the table fault rate rises up to 60%. The last group of changes arrives at time 50. The advanced stability procedure cannot reduce fault rate much during time 10 to 50 as the SID-to-VID mapping changes constantly; the
VID returned by a SID2VID call becomes invalid instantly after one more node changes. The table fault rate drops rapidly after time 60 when each node has a stable SID. All lookup tables are updated by time 80 to have a 0% fault rate. We observed similar performance on different percentage of changing nodes.

All lookup requests reach its target throughout our experiments whether lookup tables are updated. However, a lookup request may take extra hops to reach its target when incorrect entries exist in lookup tables. To demonstrate how the advanced stability procedure impacts lookup performance, we measure the average lookup path in each time instant and compare with the one with 0% fault rate. Figure 11 shows the increased path length of Chord and SCALLOP with 20% and 30% multi-node changes. When 20% nodes change status concurrently, the average path length is increased by nearly 50% with Chord and 26% with SCALLOP. With 30% changes, Chord increases average path length by more than 70% while SCALLOP resolves a lookup request with less than 40% path increase. On the other hand, SCALLOP takes 20 more time units to reduce average path length after each node has a stable SID. Such behaviors result from additional table entries required by SCALLOP for balanced lookup trees. With Figure 10 and 11, we show that the advanced procedure efficiently updates lookup tables and stabilizes a system after multi-node changes.
E. Customized Performance

SCALLOP allows a customizable degree $d$ of a balanced lookup tree to trade between memory space required by lookup tables and lookup paths. To observe the performance impact of different $d$, we generate a number of lookup requests into a P2P system and measure the longest lookup path. Figure 12 plots the longest path length as a function of the number of nodes. We vary the number of nodes from 1 to $10^5$ in multiplier of 10, and conduct the same experiment using Chord and SCALLOP with $d = 2, 4, \text{and } 6$. The larger $d$ is, the shorter the longest lookup path is. Particularly, SCALLOP delivers shorter longest lookup paths than Chord when $d$ is 4 or larger.

It is also important to note that, by using larger $d$, we can reduce the total number of routed requests in a P2P system and, therefore, improve system performance. Figure 13 plots the total number of routed requests as a function of the number of nodes in the P2P system. Again, we compare the performance of Chord and SCALLOP with $d = 2, 4, \text{and } 6$. The larger $d$ is, the smaller hops a request reaches its target node, and the smaller number of requests SCALLOP routes. Finally, because SCALLOP with larger $d$ spends less overhead in routing lookup requests, resources can be saved for better performance.

VI. RELATED WORK

The research of P2P distributed systems has received a lot of attentions recently due to the low cost of workstations and the availability of high-speed networks. With different requirement
on memory resource, routing topology, and computational complexity, a number of P2P lookup protocols have been developed to resolve a lookup request in a P2P distributed system [19], [30], [22], [20], [24], [21], [4], [23], [15], [13]. These lookup protocols can be classified into three categories: centralized, decentralized and unstructured, and decentralized and structured.

A centralized lookup protocol, such as Napster [18], has one centralized server that stores location information of all data in the distributed system. All lookup requests are routed to and resolved in this server. The main concern of a centralized protocol is its scalability and stability. The centralized server becomes the performance bottleneck as the number of nodes increases. In addition, once the server fails or the location database corrupts without a backup, the system becomes malfunction. As a result, such a protocol is not suitable for a large-scaled distributed system.

Examples of a decentralized and unstructured lookup protocol include Gnutella [7], Morpheus [16], KaZaa [11], Freenet [3], and the work of Pandurangan et al. [19]. Such a protocol requires no centralized server to scale with the number of nodes and remain relatively stable when a part of nodes fail. On the other hand, an unstructured protocol places no accurate binding between routing topology and data location; a lookup request is forwarded using approximate information. For this reason, it is difficult to bound the lookup path and analyze the worst-case performance of a lookup protocol. In comparison, a decentralized and structured protocol such as SCALLOP delivers bounded lookup path and worst-case performance.

Different from unstructured protocols, a structured protocol place accurate binding between routing topology and data location. As a result, the performance of such a protocol is determined by its routing topology and a lookup path is bounded by the diameter of the routing topology. However, to successfully resolve a lookup request, it is required to maintain a strongly-connected routing topology. This requirement becomes an expensive operation when nodes dynamically join or leave the system. Several routing topologies such as binomial trees [30], [22], [24], d-dimension space [21], butterfly [4], [23], [15], [13] and de Bruijn graph [9], [17], [5] have been carefully studied.

Tapestry [29], [30] and Pastry [22] are two similar structured protocols that adopt a binomial lookup tree. Both protocols are developed from the work of Plaxton et al. [20]. Each node is assigned an identifier of a string of b-based numbers. In addition, each node maintains a lookup table consisting of \( O(\log_b N) \) levels of entries. The lookup path is bounded by \( O(\log_b N) \) or
Due to randomized approximation, a lookup request may not be resolved successfully. In contrast, SCALLOP follows the stability proof [25] to guarantee that a lookup request be resolved successfully even in a dynamic system. Because a binomial tree is an unbalanced tree, both Tapestry and Pastry deliver fluctuated routing loads and introduce routing bottlenecks. In comparison, SCALLOP adopts a balanced lookup tree, delivers more balanced loads, and reduces or eliminates routing bottlenecks.

Chord [24] is the most-referenced binomial lookup protocol. Several extensions such as AChord [8] and Hyperchord [14] have already been made. A lookup request is guaranteed to reach its target within $O(\log_2 N)$ hops, and each node is aware of $O(\log_2 N)$ other nodes. On the other hand, SCALLOP allows a customizable degree of $d$ to trade between memory requirement $O(d \log_d N)$ for a lookup table and bounded lookup path at $O(\log_d N)$. Similar to Tapestry [30] and Pastry [22], Chord introduces routing bottlenecks when hot spots occur. Both Chord and SCALLOP handle dynamic changes effectively at similar complexities. In terms of efficiency, our experimental results show that SCALLOP takes less time to update all lookup tables and introduces a smaller increased percentage on the average lookup path.

CAN [21] is another structured protocol that adopts $d$-dimension routing topology. It assigns nodes and data items into a $d$-dimension space. Each node is responsible for data items stored in a particular region and is only aware of its neighbors on the $d$-dimension space. As a result, the size of a lookup table is bounded by $d$, the number of its neighboring nodes. However, since a lookup request can only be forwarded by a node to its adjacent regions, the arrangement of regions directly affects the lookup path. Although the average lookup path is bounded by $O(dN^{1/d})$ as proved in [21], the worst-case lookup path is a linear search through all available regions. In addition, due to the greedy search algorithm used by CAN, a lookup request is not guaranteed to reach its target node in all cases.

Among all structured lookup protocols, de Bruijn graph and butterfly offer minimum bounded lookup path at $O(\log N/(\log \log N))$, shown in [28]. However, none of the work extended from de Bruijn graph [9], [17], [5] provides an efficient self-organized mechanism to make it practical for a large-scaled system whose state changes constantly. The work [28] also shows that butterfly topology reveals a congestion problem where the worst-case traffic for certain paths cannot be bounded. Consequently, butterfly-based protocols [4], [23], [15], [13] are not suitable for a distributed system with hot spots.
In summary, to the best of our knowledge, SCALLOP [2] is the only decentralized and structured lookup protocol that addresses on the issue of routing bottlenecks. By adopting a balanced lookup tree, SCALLOP evenly distributes routing loads and reduces or eliminates routing bottlenecks. An efficient self-organized mechanism is incorporated to guarantee a lookup request is resolved successfully even in a dynamic environment. Finally, it provides a bounded lookup path and allows a tradeoff between resource requirement and routing efficiency.

VII. CONCLUSIONS

Many large-scaled servers are implemented as a peer-to-peer distributed systems. A scalable and load-balanced lookup protocol is essential for such servers to deliver rapid response time. Traditional lookup protocols make use of binomial lookup trees to route requests and shorten lookup paths. Although it scales with the number of nodes, the unbalanced feature of a binomial lookup tree introduces routing bottlenecks when certain nodes become hot spots. In this paper, we presented SCALLOP, a scalable and load-balanced lookup protocol based on balanced lookup trees. SCALLOP requires that each node be aware of $O(\log N)$ other nodes in the system. In addition, a balanced lookup tree evenly distributes routing loads and, therefore, avoids routing bottlenecks. A bidirectional SCALLOP model is introduced to further reduce lookup paths and efficiently update all lookup tables when nodes join, leave, or fail dynamically. Finally, SCALLOP provides a customizable degree of a balanced lookup tree to trade between memory and lookup performance.

We conducted a series of experiments to compare SCALLOP and Chord, the most-referenced and representative binomial lookup protocol. The experimental results show that, with balanced lookup trees, SCALLOP delivers more balanced routing loads and reduces routing bottlenecks in cases of a single hot spot and the popular 80/20 scenario. In addition, we simulated a system whose state changes constantly. In terms of efficiency at stabilizing a dynamic system, SCALLOP takes less time to update all lookup tables and place a smaller increased percentage on the average lookup path.

ACKNOWLEDGMENTS

This research was supported in part by National Science Council, R.O.C., under Grant NSC 91-2213-E-007-034 and by Ministry of Education, R.O.C., under Grant MOE 89-E-FA04-1-4.
REFERENCES


