Optimizing Lookup Efficiency of Chord Peer-to-peer System Via Cooperative Reinforcement Learning

Chaolun Wang   Yaoguang Zhong   Bolong Zhang
Florida State University

4/27/2018

Abstract

As a kind of widely used distributed systems, peer-to-peer systems possess distinct features such as ability to tolerate errors and capability in growth of a large number of nodes[20]. Unlike a centralized distributed network, in which there is a centralized control agent that can coordinate all the network nodes to accomplish all the file lookup, in a peer-to-peer network, there is not centralized control. Locating a node can be only accomplished by the node itself and via the knowledge of the node. Though some simple routing approaches may be able to arrive in a simple routing functionality in a network. However, these approaches are not efficient in term of its leverage of local information and utilization of knowledge of the surrounding neighbors. As a result, the performance of the lookup process is low. Some existing approaches takes advantages of well-design of a node and store helpful information in the node for lookup processes. This design makes the lookup more efficient since data stored in the node can guide the control toward the target node. However, the knowledge leveraged is not optimized. In this paper, we propose a novel and self-evolving Cooperative Q Learning (CQL) algorithm to optimize the Chord ring peer-to-peer system. We also arrive with our own design of the peer-to-peer nodes. In addition, we ingrate RPC (Remote Procedure Call) into our design. The efficiency of CQL has been studied by using simulated underlying network where the latency has been pre-set as certain distribution. We compare the proposed approach with the existing advanced algorithms, including the greedy node selection and the server selection algorithms. The experimental results demonstrate how CQL can self-evolving and outperform the two approaches via gradually learning, especially when the distribution of the latency on the Chord ring is not uniform. This method is also scalable and can get a satisfactory performance as the number of the nodes in the Chord ring increase.

1 Introduction

Peer-to-peer system is one of the most widely used distributed system. Compared with traditional client-server system, peer-to-peer system is decentralized and has a lot of good attribute such as error tolerance and the ability to grow into arbitrarily large size[20]. This property is important as the development of big data requires large scale distributed system, and the peer-to-peer system has been widely used in file sharing systems[16], web based computing[17] and even electric currency[15]. In a peer-to-peer network, there is not centralized control. Locating a node can be only based on the knowledge of the nodes. Some simple routing approaches may be able to accomplish the routing functionality. However, these algorithms and approaches are not efficient in term of number of hops required for a lookup and time consumed for the lookup. Moreover, the design may not be scalable or it may be scaled in high cost. It turns out that knowledge of the node plays a key role in searching nodes along the peers in a network. The well-design of the internal structure of node can drastically accelerate the lookup process. Advanced design of the node will increase the challenge when involving RPC (Remote Procedure Call[18]) in node lookup. The design for the RPC also needs to adapt the design of the node and the employed algorithms.

Although the original lookup algorithm in Chord ring system is efficient[9], it haven't took the latency of the underlying network into consideration. One of the improved approach is the server selection method[11, 13]. The general idea of this method is to estimate the statistical patterns of the latency of the whole Chord ring by the local information available. The latency between the adjacent nodes were estimated by the delay of RPC, and then recorded in finger table. The mean latency of the connection between nodes were estimated by calculating the mean latency of the nodes in finger table. The total number of nodes N also needs to be estimated in order to give an approximation to the total number of internet hops required for the lookups. In order to give an estimation to N, each of the node needs to keep a succinct list, which record the key and address of M successors[13]. The total number of nodes can be roughly estimated by M/p where p is the fraction of Chord ring covered by the succinct list[13]. Each of the estimation above needs to be accurate to get a good performance.

In this research we developed a reinforcement learning algorithm, the Cooperative Q Learning(CQL), to optimize the lookup efficiency of Chord ring peer-to-peer system. Different from the classic reinforcement learning approach which has a central agent and a clear boundary between agent and environment, CQL was developed for the decentralized system and for each node all of the other nodes were considered as environment. The learning process is a cooperative behavior in CQL and the experience is shared among all of the lookups. During the lookup process, each node will propagate the local information to other nodes, and eventually to the entire Chord ring to help other nodes to make a better decision. Although the experience of individual lookup is limited, it can compensate the experience of each other via parameter sharing. This cooperative nature makes the algorithm converge fast and no exploration is required. The information is propagated via the return value of Remote Procedure Call(RPC) so that no extra communication channel is needed. The efficiency of CQL has been studied by using simulated underlying network where the latency has been pre-set as certain distribution. We compared our approach to the two other algorithm available nowadays: the greedy approach and the server selection algorithm. CQL can self-evolving and finally get a better result compared to other method, especially when the distribution of the latency on the Chord ring is not uniform. This method is also scalable and can get a satisfactory performance as the number of the nodes in the Chord ring increasing.

In the reminder of the paper, we will first introduce the basic concept of Chord, a popular peer-to-peer protocol and algorithm to accelerate file lookup, and then present our methodology. Then we will elaborate the involvement of RPC for the proposed approach and the integration of the new algorithm in lookup. Lastly, we will demonstrate the experimental results and shows how the proposed approach outperforms the existing two algorithms and the power of the proposed approach in lookup processes.

2 Chord Protocol

The fundamental problem for a peer-to-peer system is the efficiency location of node that stores a desired data. Chord ring algorithm is a distributed lookup protocol can address this problem. Chord ring provides the support for one thing: given a key, it maps the key onto a node. In this part, we will introduce the chord ring algorithm.

The Chord protocol defines the rules to find the locations of keys, how new nodes join the system and how the node leaves the system. Also tell us how to recovery from the failure. The Chord protocol assign keys to node with the consistent hashing [7] [10]. By using this method, the hash function can balance the node with high probability. In order to improve the scalability of the consistent hashing, the chord try to avoid the situation that each node knows every information about other nodes. The Chord ring need a small amount of the information. In an N-node network, each node maintains information about only $O(logN)$other nodes, and a lookup requires $O(logN)$ messages.

2.0.1 Consisting Hashing

The consisting hashing function, which based on the SHA-1 [8], hash the node's identifier by its IP address and hash the key identifier by hash the key. Consisting hashing assigns the keys to nodes as follows. Identifiers are ordered on an identifier circle module 2^m. Key k is assigned to the first node whose identifier is equal to or follows k in the identifier space. This node is called the *successor* node of key k, denoted by *successor(k).* If identifiers are represented as a circle of numbers from 0 to 2^m1, then *successor(k)* is the first node clockwise from k. In the remainder of this paper, we will also refer to the identifier circle as the *Chord ring*

The Figure 1, shows an chord ring with $m = 6$ The chord ring has 10 nodes and stores five keys. The successor of identifier 10 is node 14, so key 10 would be located at node 14. Similarly, keys 24 and 30 would be located at node 32, key 38 at node 38, and key 54 at node 56.

Consistent hashing is designed to let nodes enter and leave the network with minimal disruption. The details we can find at paper [9]

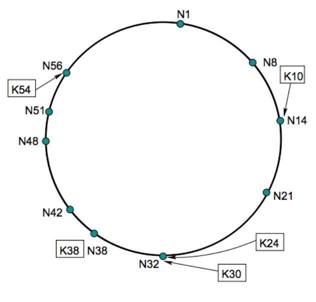


Figure 1: An identifier circle consisting of 10 nodes and five keys

2.0.2 Key Location

In this section, we will discuss how how to lookup keys in the Chord Ring. Firstly, we will introduce a simple version of the Look up algorithm and then we will extend it into a scalable key look up algorithm.

Simple key lookup: For the simple version lookup, the pseudo-code can be found in the Figure 2 (a). We can see that each node only know Each node need only know how to contact its current successor node on the identifier circle. Queries for a given identifier could be passed around the circle via these successor pointers until they encounter a pair of nodes that straddle the desired identifier; the second in the pair is the node the query maps to.

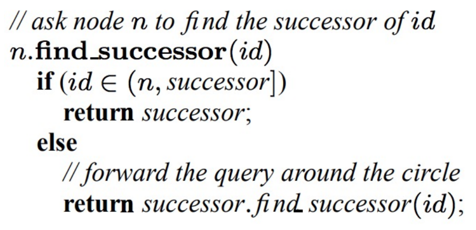
Figure 3(b) shows an example in which node 8 performs a lookup for key 54. Node 8 invokes find successor for key 54 which eventually returns the successor of that key, node 56. The query visits every node on the circle between nodes 8 and 56. The result returns along the reverse of the path followed by the query.

Lookup with finger tables:

The above lookup process is a little bit slow in the real application. To accelerate the Lookup process, in the paper [9], the author introduced the $fingertable$ to speedup the look up process. The main idea is that for each node we maintain a $fingertable$: The i th entry in the table at node n contains the identity of the first node s that succeeds n by at least 2^{i1} on the identifier circle, i.e., $s = successor(n + 2^{i1})$, where $1 \leq i \leq m$ (and all arithmetic is modulo 2 m). We call node s the ith finger of node n, and denote it by $n.finger[i]$.

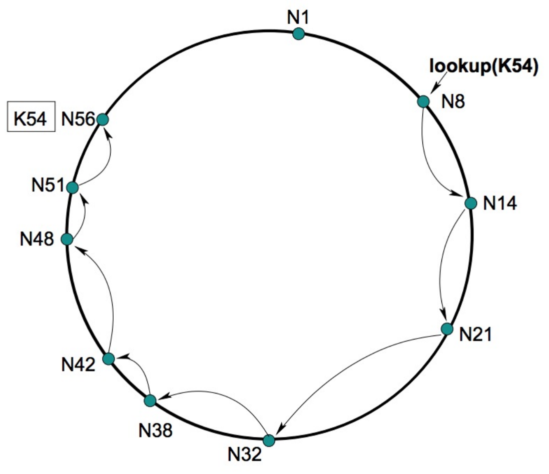
The Pseudocode is shown in the Figure 3, If id falls between n and its successor, find successor is finished and node n returns its successor. Otherwise, n searches its finger table for the node $n^{'}$ whose ID most immediately precedes id, and then invokes find successor at $n^{'}$. The Figure 4 shows the finger table for the node 8 and the path a query by using the finger table.

For the chord ring part, there are method how to maintain the Finger table when the node joins and leaves, we can find those techniques in the paper [9]



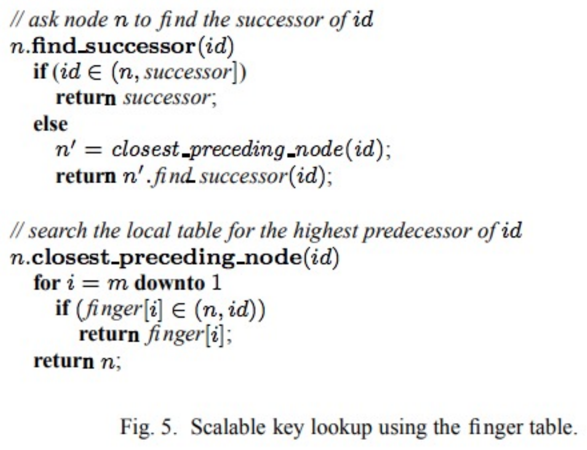
```
// ask node n to find the successor of id
n.find_successor(id)
  if (id ∈ (n, successor])
    return successor;
  else
    // forward the query around the circle
    return successor.find_successor(id);
```

(a)



(b)

Figure 2: (a) pseudo-code to find successor of an indentifier *id* (b) the path taken by a query from node 8 for key 54, using the pseudo-code in (a)



```
// ask node n to find the successor of id
n.find_successor(id)
  if (id ∈ (n, successor])
    return successor;
  else
    n' = closest_preceding_node(id);
    return n'.find_successor(id);

// search the local table for the highest predecessor of id
n.closest_preceding_node(id)
  for i = m downto 1
    if (finger[i] ∈ (n, id))
      return finger[i];
  return n;
```

Fig. 5. Scalable key lookup using the finger table.

Figure 3: Scalable key lookup using the finger table

3 Methodologly

3.1 Cooperative Q Learning.

Q learning is a reinforcement learning algorithm which has been widely used in optimization problem[5, 6, 3]. There are previous research using Q learning approach tuning the parameter of complicate system which gives considerable improvement[12]. One of the main advantage is that the Q learning algorithm do not need an explicit modeling of the environment[6, 2], so that when applying the Q learning algorithm, the environment can safely be considered as ”black box”. But the classic Q learning cannot be used on Chord ring lookups for the following reasons: firstly, each lookup has its unique start node and target. This property makes the number of stages to be n^2 given that there are n nodes on the Chord ring, if each of the stage satisfy the requirement of finite Marcov Decision Process(FMDP)[1, 14]. Secondly, a centralized agent is not allowed in peer-to-peer system, both of the learning process and the information gained from the learning needs to be distributed on the entire system. Also this decentralization cannot introduce an intensive communication between the nodes.

Taking all of the aspects into consideration, we developed a reinforcement learning algorithm which works explicitly for the optimization of Chord ring lookups. We named our algorithm Cooperative Q Learning(CQL) because it will allow all of the lookups sharing the learning process. The main idea

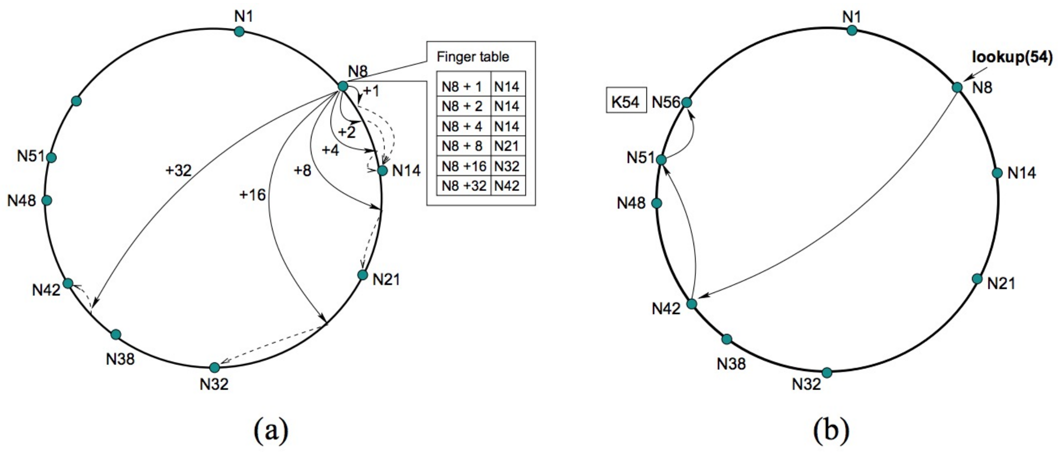


Figure 4: (a) Finger table entries for node 8 (b) The path a query for key 54 starting at node 8

of CQL is to consider each of the lookup as a subsampling of the entire Chord ring. We got inspired by the Deep Residual Net(DRN) with stochastic depth[4]. In this approach, a DRN with hundreds of layers can be trained by stochastically dropping most of the layers. The intuition is to treat the whole network as the accumulation of the subsamples during each training. We design the CQL with similar idea: although each lookup will travel only a fraction of Chord ring, but different lookups will eventually cover the entire path available. The sharing of parameter between each lookups makes them a subsampling of the whole system.

Under this setting, we can safely drop the number of stage from n^2 to n given that each of the stage satisfy the requirement of FMDP. The stage will be considered as each of the nodes and the actions for each stage will be the fingers in finger table, which can be chosen as the next internet hop. For the stage s, the $Q(s, a_i)$ value for each action a_i in a was defined as the maximum reward gained to travel the entire Chord ring from s, taking the action a_i as the first hop. Considering the circular pattern of Chord ring, the $Q(s, a)$ can be recursively estimated by:

$$Q(s, a_i) \approx r + (1 - p)max(Q(s', a'))$$

where r is the local reward gained by taking the action a_i and p is the fraction of Chord ring traveled by taking the action a_i. a_i is the finger in node s which corresponding to the node s'. The local reward was formulated as:

$$r = -\frac{latency(s, a_i)}{p}$$

This formula was chosen because the factor we want to optimize is the latency of lookup, for this purpose a good internet hop should have small latency and travel as far as possible on the Chord ring. If CQL was used to the optimization of other factors of peer-to-peer system, other local reward function can be chosen correspondingly. The fraction traveled by taking action a_i can be calculated as:

$$p = \frac{dis(s, a_i)}{ringSize}$$

where $dis(a, b)$ is the function calculate the distance from key a to key b on the Chord ring. Note that the traversal on Chord ring can only be done clockwise, so the function $dis()$ has the following property:

$$dis(a, b) = ringSize - dis(b, a)$$

3.2 Distributed Q table and Q value back propagation.

To make the algorithm decentralize, the storage of the information gained needs to be separated among

the nodes. Therefore each nodes needs to store the most important information to itself in the aspect of decision making process. Taking this requirement into consideration, we add a row into the finger table of node s which stores the value $Q(s, a_i)$ for the corresponding finger a_i.
For the forward path of CQL, the finger a_i which corresponding to $max(Q(s, \hat{a}))$ was chosen as the next internet hop. Where the $\hat{a}$ is a subset of fingers in a which do not overshoot. Which means each element j in $\hat{a}$ satisfy:

$$dis(a_j, target) < dis(s, target)$$

After the decision making step, the node s will send RPC request and pass the target key to the node s' which corresponding to finger a_i.
The backward path requires the node s' to find $q' = max(Q(s', a'))$ and return it to node s as the return value of RPC. After receiving q', the node s can then update Q value by:

$$Q(s, a_i) = r + (1 - p)q'$$

If the internet latency is not stable and has large fluctuation overtime, the learning process above may not be stable. One approach to overcome this shortcoming is to record the moving average of latency estimated by RPC. Another approach is to add a learning rate to the updation function to stabilize the learning process:

$$Q(s, a_i) = Q(s, a_i) + \gamma(r + (1 - p)q' - Q(s, a_i))$$

Where the value γ was technically chosen from 0.1 to 1. Since our simulation was done using a fixed latency which do not fluctuate overtime, we choose the learning rate to be 1 for simplicity.

3.3 About Exploration.

In classic QL algorithm, the exploration is required as a way to let agent explore new stages. This is the major way for the learning algorithm to gain new knowledge about the system. But the exploration requires extra effort: it requires the agent to try some seemingly bad actions give existing clues, which will have a large chance to be an actual bad action. Usually a high exploration rate will make the learning process converge faster but decreasing the average performance. Generally, this shortcoming of exploration is handed by dynamically drop the exploration rate or develop an action selection engine. Both approaches require problem specific knowledge and intensive parameter tuning.
One of the most significant difference of CQL compared with classic QL algorithm is that no exploration is needed. In CQL algorithm, since all of the lookups share a common set of parameters, the limited experience of one lookup can be compensated by other lookups. This property can make the exploration of individual lookup process unnecessary, each lookup only need to take the best decision given the existing clues.

4 RPC on Node Lookup

Though we could simulate the environments for evaluating performance of the aforementioned algorithms, for more practical use, we extended peer-to-peer lookup on nodes from the single program to a more advanced version of the program, a network-based application. The a significantly distinct feature of a single program and a network-based application is that, in peer-to-peer network, there is not centralized control. Locating the target node will be accomplished by iteratively requesting service from the next node. Calculating the next node is based on the data of the node and the algorithm employed. The options for the algorithms can be one of the three algorithms including Cooperative Q Learning(CQL), greedy node selection algorithm, and server selection. The difference between a single program and the network-based program will post challenge to involvement of RPC in the node lookup. This entails radical change on the code we have done for the single program.

4.1 Data Passed between Nodes

RPC node lookup is accomplished by passing string data between nodes. The data passed needs to have some sort of format so that operations on the data are not prone to mistakes when extracting properties. To make the data small in size and easy to be understood, we design a simple but functionally-sufficient data structure for the data request. The data is in the form of:

```
{prop1=val1}{prop2=val2}{prop3=val3}
```

The string data above contains three properties: prop1, prop2, and prop3. val1, val2, and val3 are the values of the properties respectively. The value of the property can be easily to be extracted as we can use a string function to extract the value between delimiters "=" and "}". There is not maximum for the properties encoded in a string data as long as the size of the data package passed between servers is allowed by the UDP protocol. Below is an example of a data string used in our program.

```
{type=lookup}{targetKey=121}{flag=1}
{nodeIp=127.0.0.193}
```

When a node sends a request string as the example above to another node, the node intends to have the recipient node help it to find the key 121 and use the algorithm represented by flag 1, i.e. greedy algorithm. nodeIp in the string data is the IP of the node that will receive the result of the lookup.

As the string data above is sent to another node, the recipient node will undergo the following processes. Upon receipt of the lookup request, the server function in the recipient node will look up its local data and check whether the node is corresponding to the target key. This will be accomplished by calculation. If yes, the recipient node will send a response request back to the initiating node for an indication that the node corresponding to the key has been found. As the result of this lookup, the IP of the node corresponding to the key will be contained in the response request.

4.2 The RPC Lookup Process

As in peer-to-peer network, there is not centralized server that will help to cooperate the communication between nodes. That means finding a node can only rely on the nodes themselves and their own data. A node can calculate the next node by looking up its local data, i.e. the finger table [9]. However, calculation of the next node also needs to take account of the algorithms employed. The algorithm used for a lookup will be represented by a property in the string data, i.e. flag.

Interaction between nodes will be accomplished via passing string data. A example of string data is "{key=16}{ip=127.0.0.8}". The string data used in our program is more complicated since it contains more properties.

4.3 Design of RPC Node Lookup

With respect to the design of the node lookup via RPC, finding node will be accomplished by repeated lookup and request-response mechanism.

4.3.1 Repeated lookup

The lookup process is essentially a repeated lookup. For example, if node A wants node B to help it to find the key 23, node A will send a RPC lookup request to node B. All the necessary information including the key are included in the request data. In node B side, there is a server function, which is waiting for receiving requests. Upon receipt of a lookup request, node B will check whether node B is the node corresponding to the key. The algorithm for checking the relation between the node and the key is specified by a flag, which is contained in the request string. If node B is not the node corresponding to the key, node B will search its local data, e.g. finger table, to find the next node. When using greedy approach, the program will first try to find the node corresponding to the longest preceding finger. If this finger is found, node B has the node corresponding to the finger to help it to find the key by passing a lookup request to the node. Otherwise, node B will have its successor to help it to find the key. Similarly, this will be accomplished by sending the lookup request to its successor. The lookup process will continue until the node corresponding to the key is found. Once the node corresponding to the key is found, the node will send the result back the initiating node, in this case, node A.

4.3.2 In Request-response fashion

The program was implemented in request-response fashion. That means data passed between node servers will be accomplished by sending request string and receiving response string. Every node in the network is both a client and a server. When trying to have other nodes to find the key for it, a node will perform like a client and request lookup service from other nodes. When processing the requested service, a node will play its role as a server. It will invoke the related functions to complete the requested service. Once the process is completed, the node will take actions accordingly.

4.3.3 The state of a node

All the properties related to the node will be encapsulated in the node. Manipulation of the properties will be accomplished by the node itself. Other node will not able to change the properties of the node. For example, the finger table of a node can only be changed if the node aggressively to update the finger table periodically. This design will disallow any "underground" modifications on the node. We do not expect every subtle change on the node since this will make programming debugging more difficult.

4.4 Implementation

The program uses sockets to pass data between node servers. We take advantage of Local Network Interface [21] for local network test. The Local Network Interface allows a machine to use a range of IPs from 127.0.0.1 to 127.255.255.254. The port the program uses is 12345. The protocol used to send data between nodes is UDP. We do not want the program establish every connection since connections between nodes will consume a tremendous amount of resource, which is unnecessary. UDP protocol is a good match for request-respond mechanism.

When implementing RPC in the program, we encountered a concurrency issue. That is, when the program tried to invoke methods from some server objects, the program hanged. This problem was caused by calling methods from uninitialized server objects since the servers were initialized in different processes. There is not way to guarantee the order of instructions between different processes. In other words, the processes run concurrently. To solve this problem, we have the main process delay five seconds after initializing server objects. Likewise, we have the program sleep two seconds before a node sends a request to another.

4.5 Pseudo-code for finding a node via RPC requests

Below is the pseudo-code for the main function and request function in our program.

```
// Node A haves Node B to help it find the node
// corresponding to key 23
main()
{
   initialize all nodes.

   string request = "{type=lookup}{key=23}{flag=1}
   ..."
   nodeA.rpc_request(nodeB.ip, request);
}

// The rpc_request method
void rpc_request(string ipNodeLocal,
string ipNodeNext, string request)
{
    sendMessage(ipNodeLocal, ipNodeNext, request);
}
```

The pseudo-code above shows a program, in which node A wants node B to help it to find key 23. This task can be accomplished by these steps. In the main function, the program will first initialize all nodes. It then formulates the lookup request string. The request string will contain necessary information for the lookup including the type of lookup, the key to be searched for, the flag indicating what algorithm will be used in the lookup, and the node initiating the lookup process. After that, node A will send a lookup request to node B.

Every node has a function that will receive requests from other nodes. It will then process the requests and take actions based on the result of calculation. Below is such a function in every node.

```
// Server function: waiting for request
node.listeningForRequest()
{
     // Session 1: Handle lookup request
     if ( getType(dataReceived) == "lookup" )
     {
          key = getKey(dataReceived);

         if( {is the node preceding the key} )
         {
             // found the node preceding the key
             request = "{type=respond}{key=23}
             {flag=1}{targetIp=127.0.0.19}..."
             ndoe.rpc_request(nodeInitiate.ip
             , request);
         }else{
             if( {is there any finger preceding
             key for the node} )
                 nextHopIp = {get the closest
                 finger preceding the key}
             else
                 nextHopIp = node.successorIp;

             request = "{type=lookup}{key=23}
             {flag=1}..."
             ndoe.rpc_request(nextHopIp, request);
         }
     }

     // Session 2: Handle respond request
     if ( getType(dataReceived) == "respond" )
     {
          targetIp = getTargetIp(dataReceived);
          print("Found: The node searched is: "
          + targetIp );
      }
}
```

In the node B side, there is a server function, which is running to wait for requests. Upon receipt of a lookup request, the node will check its local data and see whether it is the node corresponding to the key. If it is not the node corresponding to the key, the node will find the next node. Calculation of the next node will depend on the algorithm employed in this lookup. For example, if the lookup request is set as greedy approach, the algorithm will search its finger table and try to find the IP of the next node. It then sends a lookup request to the next node and have the next node to help node A to find the node corresponding to the key. Receiving the lookup request, the next node will process the request in the same procedure as node B. The lookup process will not stop until the node corresponding to the key is found. Once the node corresponding to the key is found, the node will send a response request back the node A, the node initiates the lookup process, and indicates to node A that the node node A is looking for is the node. Once the response request is received by node A, node A will extract the IP from the response request. Once these have been done, the lookup process is completed.

5 Simulation Result

In this paper, we propose a novel and self-evolving Cooperative Q Learning (CQL) algorithm, and seamlessly integrate the algorithm with the nodes. We also incorporate RPC to the proposed approach to extend the single program to a network-based application. In this section the efficiency of CQL has been studied by using simulated underlying network where the latency has been pre-set as certain distribution. We compared our approach to the two other algorithm available nowadays: the greedy approach and the server selection algorithm.

5.1 Performance of CQL Compared With Other Approach

We compared the performance of CQL with other lookup algorithm used in Chord ring nowadays. The total number of nodes is 400. The underlying latency of network between nodes were set as an uniform distribution between 0ms to 20ms. The lookups were generated which start at a random node on Chord ring, looking for an random target. The moving average of the latency was recorded for each lookup. The experiment has been repeated 5 times and the average performance has been chosen as the final criteria.

The greedy approach is the original lookup algorithm of Chord ring, for each internet hop, the longest finger which do not overshoot was chosen. The server selection method is the improved version with took the underlying latency into account, by estimate the network statistic and using it to make decision. Note that in this study, we do not keep a succinct list to estimate the total number nodes in Chord ring, for simplicity, N was pre-set as a fix value and available for each node. So the server selection method can use it directly. This may potentially increase the prediction accuracy of server selection method. The Q value of CQL was randomly initialized in order to justify the learning process. In practice, this Q value can be pre-calculated by using other algorithms to enhance the initial performance. The result was shown in Figure 5.

We can see that CQL algorithm can self-evolving and converged when there are 5000 to 6000 lookups performed on the entire Chord ring. After the CQL algorithm converged, it have the average lookup latency around 33ms, which is much better than the average lookup latency for greedy approach(40ms). The increasing of performance compared with the server selection method is not significant because the uniformly distributed latency. In this case, the local statistic can generally represent the trend of the global information accurately and the total number of nodes N is the true value. In order to analyze the source of the performance gained from the CQL, we calculated the statistical attributes for 100 randomly selected lookup samples using different lookup algorithms. The statistical attributes studied including: average lookup latency, average number of internet hops and the average number of hops per lookup. The result has been shown in Table 1. We can see that the greedy algorithm has less internet hops on average for each lookup compared to server selection method and CQL. By taking the underlying latency into consideration, each internet hop cannot always be the longest finger which corresponding to the hop travels the longest distance on the Chord ring. However, the average latency required for each of the internet hop is largely reduced in server selection and CQL method. The small average hop latency is much more significant in CQL compared with the case in server selection method. Also we can see that the learning process of CQL will not significantly decrease the average number of internet hop per lookup. On the contrary, the decreasing of the average hop latency is very significant after the training process. Based on this result, we can make the conclusion that the major source of the performance gained during the learning process is from the decreasing of average hop latency.

We also tested the situation when the distribution of latency is not uniform. In this simulated situation, there are 5% of nodes with a bad connection to their adjacent nodes, which has additional 200ms latency penalty for the internet hop. The simulation result was shown in Figure 6. In this situation the server selection method can no longer make accurate prediction, since the local statistical attribute available for each node can no longer reflect the global property of the Chord ring accurately. On the other hand, the CQL can propagate the local information via RPC. Each lookup involves multiple Q value propagation and updation and will gradually spread the local information about the latency to a global scope. This property of CQL makes the Q value contains intuition about the global situation. In this situation, we can see that the CQL algorithm can have a significant better performance compared with both greedy approach and server selection method. The improvement is as large as 30% when CQL converges to a stable stage.

5.2 Scalability of CQL

Since the Chord ring peer-to-peer system will potentially have a large amount of users, it is very important that the CQL algorithm can be used when there is a large number of nodes on line, which calls a high requirement of the scalability. We tested whether the CQL method is scalable as the number of node increase. The latency between the nodes were ran-

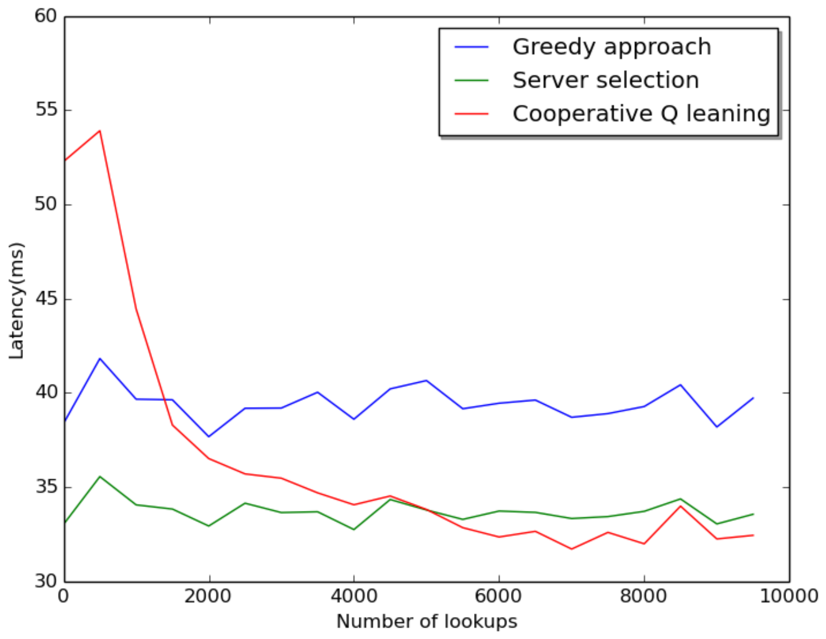
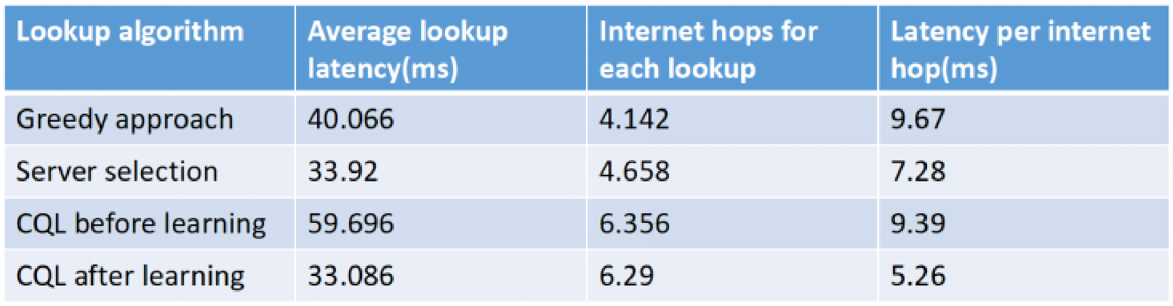


Figure 5: The comparison of Cooperative Q learning, greedy approach and server selection method under uniformly distributed network latency between 0ms to 20ms.

domly set as 2ms or 20ms in this experiment(to get a smooth curve). The number of node(N) has been chosen as 100, 200, 400 and 800. The result has been plotted in Figure 7. We can see that when the algorithm converges, the average lookup latency between each of the groups do not have significant difference. The amount of lookups required for the entire Chord ring is proportional to N. It require 10 to 15 lookups per node for CQL to converge. The number of lookups is supposed to be proportional to the number of nodes, cause if the number of clients using the Chord ring system increasing and the total amount of lookups will also increase correspondingly.This makes the CQL algorithm highly scalable as the number of nodes in Chord ring growth. Each of the user will be responsible for initialize a fixed number of lookups for CQL to converge, no matter how large the total number of users on line.

5.3 Cooperative Nature of CQL

One of the major difference between CQL and classic QL algorithm is the sharing of parameter between each of the lookups, without considering the lookup start node and target. This parameter sharing can make the information gained by learning shared among different lookups. In order to test the cooperative nature of CQL, we separate the lookups into training group, which start at nodes with odd index, and test group, which start at nodes with even index. Only the training group have the backward path of the CQL algorithm, which involve Q value back propagation and updation. The test group only used the Q value to make decision. The comparison of training group and test group was plotted in Figure 8. We can see that the efficiency of test group can also be im-



Lookup algorithm	Average lookup latency(ms)	Internet hops for each lookup	Latency per internet hop(ms)
Greedy approach	40.066	4.142	9.67
Server selection	33.92	4.658	7.28
CQL before learning	59.696	6.356	9.39
CQL after learning	33.086	6.29	5.26

Table 1: The comparison of the lookup attributes for Cooperative Q learning, greedy approach and server selection method under uniformly distributed network latency.The statistical attributes including average lookup latency, average number of internet hop and the average number of hops per lookup has been compared.

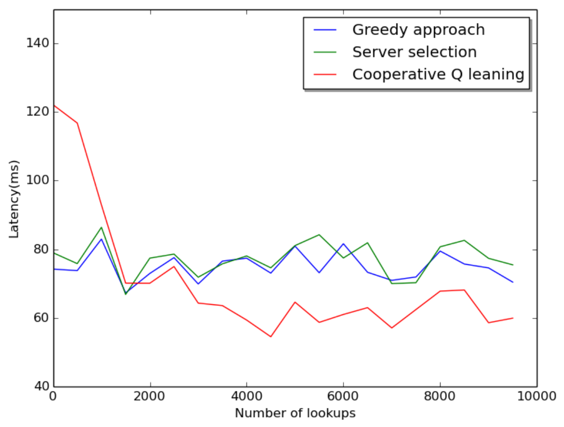


Figure 6: The comparison of Cooperative Q learning, greedy approach and server selection method with 5% bad connected node, which has additional 200ms latency penalty for the internet hop to its adjacent nodes.

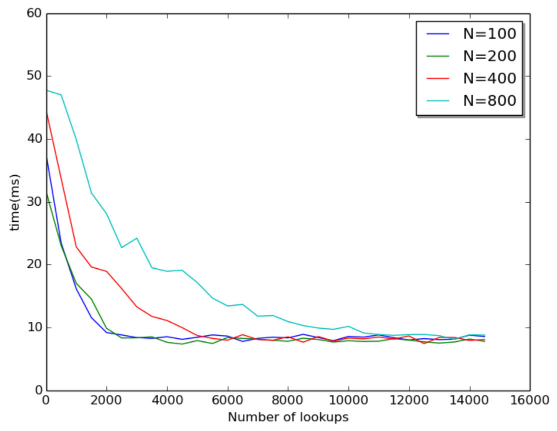


Figure 7: The comparison of the performance of Cooperative Q learning with the number of nodes equal to: 100, 200, 400 and 800.

proved corresponding to the training group. Which means the learning of training group can be generalized to the test group. This result is as expected because the information gained by learning is shared by different lookups.

6 Conclusion

As one of the most efficient peer to peer system, Chord ring can perform lookups with only log(N) internet hops. However, one of the shortcoming of the Chord ring is that it doesn't take the latency of underlying network into consideration when deciding lookup path. In this research, we developed

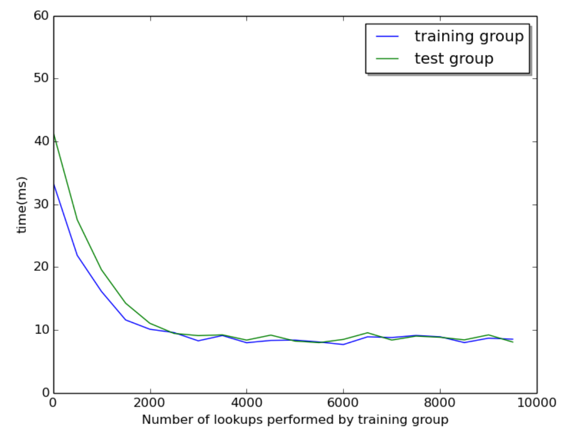


Figure 8: The comparison of the lookup efficiency between training group(initialized at the nodes with even index) and test group(initialized at the nodes with odd index). Only the training group can update Q value. The test group is only using the Q value to make decision.

a reinforcement learning approach, Cooperative Q Learning(CQL), which can optimize the lookup efficiency of Chord ring system. CQL can self-evolve and get a better result compared with other lookup algorithm used in Chord ring system. By sharing parameters between lookups, CQL can converge in a fast speed and handling different lookups in an efficient way. This method has a lot of advantages: it is scalable and do not need extra communication between nodes; compared with other q learning method, it do not need exploration and hyper-parameter tuning. Also the computation required is not intensive($O(log(N)^2)$ for each lookup) and distributed among $log(N)$ nodes. For each node it only require $O(log(N))$ extra space to store the Q value. The prototype of Chord ring peer to peer system has been implemented and tested using a simulated underlying network environment.

One of a potential disadvantage of CQL is the requirement of certain number of lookups(10 to 15 per node) before the convergence. If the the system involve intensively addition and deletion of the node but infrequent lookups, the CQL method may take a long time to converge. y take aOne of the approach to overcome this shortcoming is to make clever initialization of Q value. Greedy algorithm and server selection approach can be used for a better initialization which can overcome the relatively high startup latency. Another potentially approach is to develop a hybridized algorithm between CQL and traditional method. Also the CQL can be easily applied on other optimization problems in the Chord ring peer to peer systems, such as the optimization of data consistency, without elaborated modification. We saw a strong potential in CQL on developing peer to peer system which can self optimize.

References

[1] Levin, Esther, Roberto Pieraccini, and Wieland Eckert. "Using Markov decision process for learning dialogue strategies." Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on. Vol. 1. IEEE, 1998.

[2] Erev, Ido, and Greg Barron. "On adaptation, maximization, and reinforcement learning among cognitive strategies." Psychological review 112.4 (2005): 912.

[3] Bertsekas, Dimitri P., et al. Dynamic programming and optimal control. Vol. 1. No. 2. Belmont, MA: Athena scientific, 1995.

[4] Huang, Gao, et al. "Deep networks with stochastic depth." European Conference on Computer Vision. Springer, Cham, 2016.

[5] Bradtke, Steven J. "Reinforcement learning applied to linear quadratic regulation." Advances in neural information processing systems. 1993.

[6] Erev, Ido, and Alvin E. Roth. "Predicting how people play games: Reinforcement learning in experimental games with unique, mixed strategy equilibria." American economic review (1998): 848-881.

[7] Lewin, Daniel Mark. Consistent hashing and random trees: Algorithms for caching in distributed networks. Diss. Massachusetts Institute of Technology, 1998.

[8] http://cpansearch.perl.org/src/GAAS/Digest-SHA1-2.13/fip180-1.html

[9] Stoica, Ion, et al. "Chord: a scalable peer-to-peer lookup protocol for internet applications." IEEE/ACM Transactions on Networking (TON) 11.1 (2003): 17-32.

[10] Karger, David, et al. "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web." Proceedings of the twenty-ninth annual ACM symposium on Theory of computing. ACM, 1997.

[11] Dabek, Frank Frank Edward. A cooperative file system. Diss. Massachusetts Institute of Technology, 2001.

[12] Claus, Caroline, and Craig Boutilier. "The dynamics of reinforcement learning in cooperative multiagent systems." AAAI/IAAI 1998 (1998): 746-752.

[13] Binzenhfer, Andreas, Dirk Staehle, and Robert Henjes. "Estimating the size of a Chord ring." University of Wrzburg, Tech. Rep. 2005.

[14] Kaelbling, Leslie Pack, Michael L. Littman, and Andrew W. Moore. "Reinforcement learning: A survey." Journal of artificial intelligence research 4 (1996): 237-285.

[15] Nakamoto, Satoshi. "Bitcoin: A peer-to-peer electronic cash system." (2008).

[16] Saroiu, Stefan, P. Krishna Gummadi, and Steven D. Gribble. "Measurement study of peer-to-peer file sharing systems." Multimedia Computing and Networking 2002. Vol. 4673. International Society for Optics and Photonics, 2001.

[17] Van Engelen, Robert A., and Kyle A. Gallivan. "The gSOAP toolkit for web services and peer-to-peer computing networks." Cluster Computing and the Grid, 2002. 2nd IEEE/ACM International Symposium on. IEEE, 2002.

[18] Nelson, Bruce Jay. "Remote procedure call." (1981).

[19] Kis, Zoltan Lajos, and Robert Szabo. "Chord-zip: a chord-ring merger algorithm." IEEE communications letters 12.8 (2008).

[20] Pouwelse, Johan A., et al. "TRIBLER: a socialbased peertopeer system." Concurrency and computation: Practice and experience 20.2 (2008): 127-138.

[21] R. Morla and N. Davies, "Evaluating a location-based application: a hybrid test and simulation environment," in IEEE Pervasive Computing, vol. 3, no. 3, pp. 48-56, July-Sept. 2004.