Autonomous Data Replication Using Q-Learning for Unstructured P2P Networks

Sabu M. Thampi
Department of CSE
L.B.S College of Engineering
Kasaragod-671542
Kerala, India
smtlbs@yahoo.co.in

K. Chandra Sekaran
Department of CSE
National Institute of Technology Karnataka
Surathkal-575025
Dakshina Kannada, Karnataka, India
kch@nitk.ac.in

Abstract

Resource discovery is an important problem in unstructured peer-to-peer networks as there is no centralized index where to search for information about resources. The solution for the problem is to use a search algorithm that locates the resources based on the local information about the network. Efficient data sharing in a peer-to-peer system is complicated by uneven node failure, unreliable network connectivity and limited bandwidth. A well-known technique for improving availability is replication. If multiple copies of data exist on independent nodes, then the chances of at least one copy being accessible are increased. Replication increases robustness. In this paper, we present a novel technique based on Q-learning for replicating objects to other nodes.

1. Introduction

The principle of a data-sharing P2P system is to accept queries from users, and locate and return data. Each node owns a collection of files to be shared with other nodes. The shared data usually consists of files, but is not restricted to files. Queries may take any form that is appropriate given the type of data shared. If the system is a file-sharing system, queries may be identifiers or keywords Nodes process queries and produce results independently, and the total result set for a query is the bag union of results from every node that processes the query [1].

We can view a P2P overlay network as an undirected graph, where the vertices correspond to nodes in the network, and the edges correspond to open connections maintained between nodes. Two nodes maintaining an open connection between themselves as neighbors. Messages may be transferred in either direction along the edges. For a message to travel from node A to node B, it must travel along a path in the graph. The length of this traveled path is known as the number of hops taken by the message.

When a user submits a query, the node becomes the source of the query. A source node S may send the query message to any of its neighbors. The routing policy in use decides to how many neighbors, and to which neighbors, the query is sent. When a node receives a query message, it will process the query over its local collection. If any results are found at that node, the node will send reply message back to the query source. In some systems, such as Gnutella [2], the address of the query source is unknown to the responding node. In this case, the replying node sends the response message along the reverse path by the query message. In other systems, the replying node may know the address of the query source, and will open a temporary connection with the source to transfer the response message [3]. When a node receives a query message, it must also decide whether to forward the message to other neighbors, or to drop it. Again, the routing policy determines whether to forward the query, and to whom the query is forwarded.

A basic requirement for every P2P system is fault-tolerance. Since the main objective is resource location and sharing, we require that this fundamental operation take place in a consistent manner. If a single peer possesses resources, required by a system service, a collapse of this peer breaks down the service, as well. In other words, the service together fails with its resource peers [3]. The system as a whole needs to be able to bear single peer and link failures to improve the reliability of its services. The problem of failing services caused by failing peers owning required resources is addressed by replication techniques.

Replication of a resource creates copies of the resource, called replicas, to be distributed across divergent hosts. Replication is employed for various reasons. It is a well-known means to enhance performance, increase availability, and tolerate faults. Performance is enhanced by placing a replica on a host convenient for a client. Availability is increased, because of the redundancy introduced by replicas. If a

Authorized licensed use limited to: NATIONAL INSTITUTE OF TECHNOLOGY SURATHKAL. Downloaded on February 23,2021 at 11:06:33 UTC from IEEE Xplore. Restrictions apply.

host of a replica fails, a client trying to access the resource can access another host with a replica. Single node failures, like crashes of nodes can be tolerated as faults within the system as a whole facilitated with the help of the redundancy introduced by replicas.

In P2P systems replication can be performed in a variety of manners: *owner replication, random replication, and path replication* [4, 5]. In case of owner replication, the peer, which received the service, keeps a copy so it can offer the service itself if requested by other peers in the future. In other words, the receiver peer also becomes a service provider. The number of replicas will increase in proportion to the number of requests for the service. Nevertheless, it is insufficient for fully improve performance.

In random replication, replicas are randomly distributed amongst other peers. If we use random forwarding k-walkers random walk, random replication is the most effective approach for achieving both smaller search delays and smaller deviations in searches. In random replication, once a search succeeds, we count the number of nodes on the path between the requester and provider, and then randomly pick P of the nodes to replicate the objects.

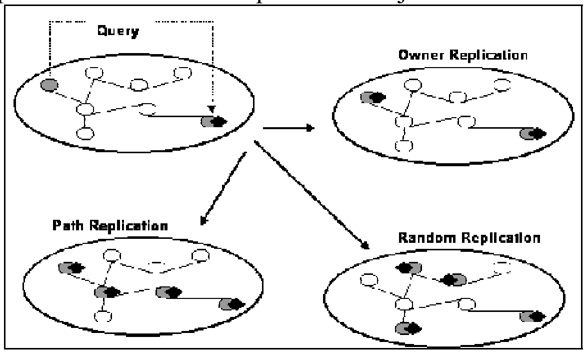

Figure 1. Examples of Replication

Path replication is another approach for distributing multiple replicas for each service. As the peers forward the query they record their address into the message. The service-providing peer receives the query, which contains information about the sequence of peers that forwarded the message *i.e.* when a search succeeds; the object is stored at all nodes along the path from the requester node to the provider node. The provider peer can then send a reply and replica of the service in the reverse direction of the forwarding route. Simulation results have shown that path replication achieves a similar performance to random replication [5], while implementation is less complex than the random replication.

Path replication and random replication reduces the overall message traffic by a factor of three to four. Hence, path replication and random replication techniques can improve the scalability of P2P systems significantly. Both path replication and random replication cause reduction in traffic due to reduction in the number of hops they take to find an object. They outperform the owner replication. However, the topological effects of replicating along the path do hurt performance of path replication [5].

The replication techniques discussed above couldn't replicate objects further than the entries in the search path. Moreover the existing techniques including erasure code based replication [8] do not consider the node behavior for hosting replicas. The decision to replicate the objects should be done autonomously. A novel mechanism is required to increase success rate further. In this report, we present a replication technique stands on Q-learning. An approach in which peers create replicas autonomously in a decentralized fashion using reinforcement-learning (RL) framework is proposed. Aim of this method is to ensure high data availability and to reduce hop counts in finding files. Parameters such as bandwidth, node's degree and storage cost are being used in this replication scheme. To the best of our knowledge, this is the first work that applies reinforcement learning to P2P replication.

The remainder of this paper is organized as follows. Section 2 reviews the related work. Section 3 introduces Q-learning based replication. Criteria for determining the sites for hosing new replica are described in section 4. Section 5 discusses the model of the system. P2P algorithm is explained in section 6. Results and discussion is in section 7. Section 8 concludes the paper.

2. Related Work

In [6] a decentralized model for dynamic creation of replicas in an unreliable peer-to-peer system is presented. [7] describes a lightweight, adaptive and system-neutral replication model for structured P2P networks. A replication scheme based on erasure code is discussed in [8]. This scheme does not cover node behaviour. RL is used in [9] to adapt P2P topologies for peer interests. [10] applies RL to P2P searching without topology adaptation and the approach systematically learns best path to desired files by exploring new paths and exploiting existing explored paths. [11] uses RL for improving the quality of resource allocation in large scale heterogeneous systems.

3. Q-Learning based Replication

The proposed replication method facilitates to place replicas in such a way that, in spite of constant changes to the connection, files are highly available. A node

Authorized licensed use limited to: NATIONAL INSTITUTE OF TECHNOLOGY SURATHKAL. Downloaded on February 23,2021 at 11:06:33 UTC from IEEE Xplore. Restrictions apply.

decides where to replicate an object taking into account the benefits of creating replicas of a particular file in certain sites.

Reinforcement learning [12, 13] comprises a family of incremental algorithms that construct control policy through real-world experimentation. An agent learns most favorable actions through a trial and error examination of the environment and by receiving rewards for its actions. The learning agent interacts with an environment over a series of time steps t= 0, 1, 2, 3… At any instant in the time the learner can monitor the state of the environment, denoted by $s \varepsilon S$ and apply an action, $a \varepsilon A$. Actions alter the state of environment, and also generate a scalar pay-off value (reward), denoted by $r \varepsilon R$. The next state and reward depend only on the previous state and action, but they may depend on these in a stochastic manner. The objective of the agent is to learn to utilize the expected value of reward received over time. It does this by learning a mapping from states to actions called a policy $\Pi : S \rightarrow A$, i.e. mapping from states $s \varepsilon S$ to actions $a \varepsilon A$. More precisely, the objective is to choose each action to maximize the projected return.

There are many different ways to incorporate reinforcement learning such as Q-learning [12, 13]. In case of Q-learning for each possible action, the agent keeps a Q-value that indicates the efficiency of that node in the past. Q learning is a form of reinforcement learning in which the agent learns to assign values to state-action pairs. In the simplest case, the Q-value for a state-action pair is the sum of all of these reinforcements, and the Q-value function is the function that maps from state-action pairs to values. However, Q-values depend on future reinforcements, as well as current ones. If an agent knows Q-values of every state-action pair, it can select an action for each state. However, the agent initially has no idea about the Q-values of every state-action pairs. The agent's goal, then, is to settle on an optimal Q-value function, one that that assigns the appropriate values for all state/action pairs.

Since in our replication scheme, several target nodes may be available to host replica, replication of an object to a single node alone is not sufficient. If more than, one node holding required parameter values exist, objects should be replicated to those nodes also. Hence, a modification in the Q-learning algorithm is needed. Since an object may be replicated to more than one node at a time, every node will produce rewards for the replication process. All these values should be considered for the efficient replication of objects. Q-learning assigns a ranking value to each resource that influences the decision to which resource a data object should be replicated. The ranking value is recalculated whenever a replication happens in the network.

4. Determining the Sites for a New Replica

The members for the Q-table are assigned after a simple operation: a message (Hello message) is send to nodes that come within a Time-To-Live (TTL), which is the number of hops the message should be propagated; the responded nodes become members of Q-table with initial Q-value equal to 100. Neighboring nodes forward the message to one of its neighbors; from there to next hop count. The message has a message-id. Nodes, which have already received a copy of the message, keep the message-id and address of the neighboring node to which the message was forwarded. Hence, when a node receives the same message another time it will not be forwarded to a node that has received the message previously.

As the first step of replication, the average value of Q-values listed in the Q-table is computed. Nodes with Q-values greater than or equal to the average value (AvgQ) are selected and a message is send to those nodes to verify whether a copy of the object exists in their shared folder. If the node is not up, a copy of the object is present, or the object's name appears in the *Replication List*, leave out those nodes. Nodes, which have Q-value greater than or equal to AvgQ are selected to host a replica.

For all the nodes in the network, minimum values are assigned for bandwidth and storage. The representation of bandwidth and available storage of each node in percentages stand on those minimum values. For example, the minimum values of bandwidth and storage are 28kbps and 100MB respectively. Let the bandwidth and available storage values returned by each node after replication are 64kbps and 120MB. Hence, the values in percentage for bandwidth and storage are being computed as 200 and 120 respectively.

When an object is downloaded into the shared folder of a node, replication starts autonomously. After creating a replica, a node receives a reinforcement signal (containing the bandwidth (b_w), available storage (s_{avbl}), calculates the metric ρ_i, ($\rho_i = a_i s_{avbl} + (1- ai) b_w$) and translates it into a reward r for node "i" that we have chosen as follows: r=sign (ρ_i). Using the second contribution in the metric (ai=0) would bias the selection towards the node with highest available bandwidth with no apprehension about free storage. This will lead to more replacement of files (LRU based) to house new files if adequate storage is not available. To avoid this from happening we use lower

IEEE COMPUTER SOCIETY

Authorized licensed use limited to: NATIONAL INSTITUTE OF TECHNOLOGY SURATHKAL. Downloaded on February 23,2021 at 11:06:33 UTC from IEEE Xplore. Restrictions apply.

bound for a_i at a_i=0.2. Finally, system updates Q-values using the following Q-function:

$Q_{i,t+1} \leftarrow Q_{i,t} + \alpha (r - Q_{i,t})$, where α is the learning rate.

The actual Q-values are retained for the nodes comprising a copy of the object by applying a reward equivalent to its present Q-value i.e. $Q_{i,t+1} \leftarrow Q_{i,t}$.

Nodes that are not up will receive a negative reward, r=0 and Q-values are updated according to,

$Q_{i,t+1} \leftarrow Q_{i,t}(1-\alpha)$.

5. The Algorithm

The shared folder of a node is periodically checked for new objects. The replication process is initiated the moment the presence of an object is noticed. The entire steps in replicating an object to different nodes are described as an algorithm below:

```
1. Select an object f for replication.
2. Compute the average of Q-values corresponding
   to node x (state), AvgQ.
3. For each entry Ti in the Q-table, select nodes
   with Q-values >= AvgQ.
4. Compute the hash value h of f.
5. For each selected node in step 3, check for the
   presence of the object using f's name or hash
   value.
6. if f exists in the searched nodes or node is not up
   or the object's name is in the REPLICATION
   LIST, leave out nodes from replication process
    else
      6.1 Select remaining nodes with Q-values >=
          AvgQ for replication.
      6.2 Insert the object's name in the
          REPLICATION LIST of selected nodes.
  // REPLICATION LIST of a node is a table that
  contains a list of object names reserved by other nodes
  during the object checking process. This evades other
  nodes to replicate the same object to the node selected
  by another node. //
7. For each chosen node:
   7.1 Replicate the object from source node to
       target node.
   7.2 Remove the object entry from the
       REPLICATION LIST of the destination node.
   7.3 Wait for reinforcement signal.
   7.4 Receive reinforcement signals - available free
       storage after storing the file, and bandwidth.
   7.5 Represent the available storage ($S_{avbl}$) as well
       as bandwidth ($b_w$) in percentages.
   7.6 Calculate weighted contribution (reward r) of
       free storage and bandwidth:
              $\rho_i = a_i s_{avbl} + (1 - ai) b_w$
              r = sign ($\rho_i$).
   7.7 update the Q-value corresponding to the node
       X according to Q-function
       $Q_{i,t+1} \leftarrow Q_{i,t} + \alpha (r - Q_{i,t})$ for each action,
       where $\alpha$ is the learning rate.
   7.8 Nodes, which are excluded in step 6, receive a
       reward r equivalent to its present Q-value.
       i.e. update Q-value by $Q_{i,t+1} \leftarrow Q_{i,t}$.
   7.9 Nodes that are not up (step 6), accept a
       reward, r=0,
       Update Q-values by $Q_{i,t+1} \leftarrow Q_{i,t}(1-\alpha)$.
```

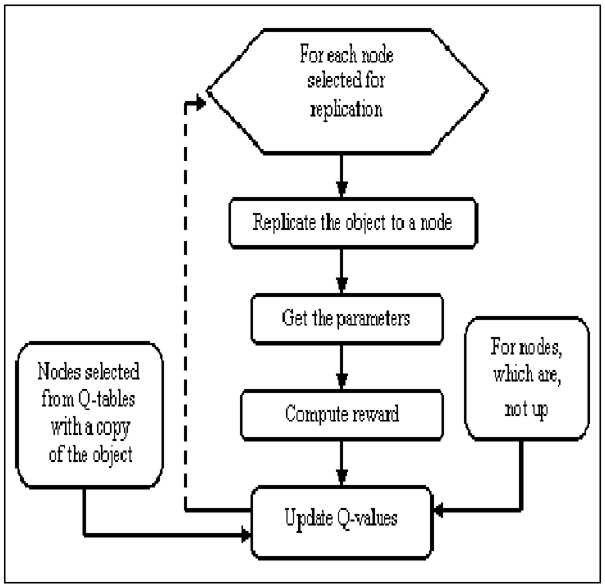
For each node selected for replication
Replicate the object to a node
Get the parameters
Compute reward
Update Q-values
Nodes selected from Q-tables with a copy of the object
For nodes, which are, not up

Figure 2: Set of actions taken by a node for replicating an object.

6. An Example

This section illustrates replication process with an example. A graphic representation of a simple P2P network shown in figure 3. The neighbors of node G are nodes D, H and K. The initial values of Q-table for node G after visiting two hop counts is shown in table 1. The selected nodes including neighbors (hop 1) are D, H, K, C, P, and L. From the second hop onwards, only one neighbor of each node is added to the Q-table.

Table 1: Q-Table

Nodes n-2 hop away from G	D	H	K	C	L	P
Initial Q-values	100	100	100	100	100	100

An object is available at node G for replication. Assume the object is not present in nodes listed in the Q-Table and all the nodes are up. The average of Q-values is found and destination nodes are selected from Q-Table. The object is replicated to the nodes. The parameters are collected and the rewards are computed. In order to provide a soft punishment to the nodes

IEEE COMPUTER SOCIETY

Authorized licensed use limited to: NATIONAL INSTITUTE OF TECHNOLOGY SURATHKAL. Downloaded on February 23,2021 at 11:06:33 UTC from IEEE Xplore. Restrictions apply.

which are not up, the value of α is set to 0.6. The Q-values are updated accordingly. Updated Q-values of node G is shown in table 2.

Table 2: An example to illustrate the status of Q-table of node G

Nodes x TTL away from node G		D	H	K	C	L	P
Parameters	Free storage, S_{avbl}	90	102	97	78	115	108
	Bandwidth, b_w	120	50	90	110	95	60
Reward (a=0.2), r		114	60.4	91.4	103.6	99	69.6
Initial Q-values, $Q_{i,t}$		100	100	100	100	100	100
Updated Q-values after replication, α =0 .6		108	76	95	102	99	82

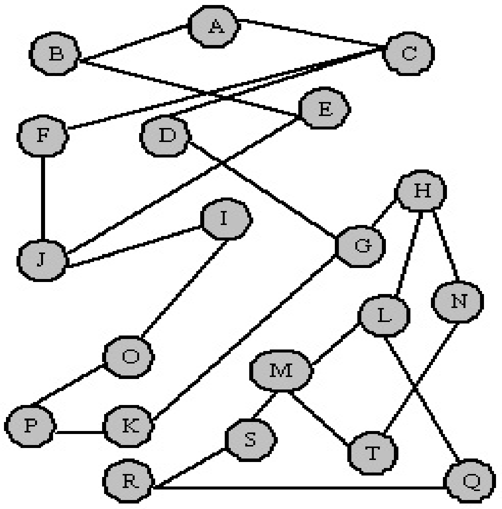

Figure 3: An Unstructured P2P Network

7. Simulation Results

We simulated a P2P network consisting of 100 nodes and 194 links. Peers in the logical network are connected randomly. Each peer has a handful of neighbors and the set of neighbor connections form a P2P network. To find a file, a node queries its neighbors. For simplicity we assume that the P2P network graph does not change during simulation. The simulation program is developed using Java language. Twenty different objects (files) are distributed randomly to twenty nodes in the network. *We assume that all nodes are up during simulation.* The experiment has been conducted and it is observed that availability is very high and it is close to 80%. Thus, the replication process consumes a great deal of network traffic and storage space. Even though the bandwidth, number of neighbors and available storage space are different, all nodes are treated equally in our approach.

The number of links to a node is called its degree. In the network, a few nodes have a large number of degrees while most other nodes have only a small number of degrees. Peers with a large number of degrees make many replicas as peers with a small number of degrees. In addition, replicas on large degree peers are used frequently as those on peers with small degrees [4]. Hence, the reward function *r,* is modified with a new attribute called the *degree* of the node for distributing replicas efficiently without consuming undue storage and bandwidth.

A modified reward function with a new parameter *degree of node* is used to compute the reward and it is computed as $r = (a_i\ s_{avbl} + (1\text{- } ai)\ b_w) \times (x/y)$, where x is the degree of the node into which replica to be placed and y is degree threshold, say eight. The value of x may be lower or higher than y. The addition of degree attribute contributes heavily in the Q-value of a node with lesser degree. The provider node collects the degree of a node when it receives response from other nodes against the Hello message. The Q-values of higher degree nodes are then initialized with a value greater than 100, say 120.

The performance of the Q-learning Replication and path replication are compared. The query is forwarded to different nodes by means of *k-walker algorithm* [5]. In random walks, the requesting node sends out k query messages to an equal number of randomly chosen neighbors. Each of these messages follows its own path, having intermediate nodes forward it to a randomly chosen neighbor at each step. These queries are also known as walkers. A walker terminates either with a success or with a failure. Failure is determined by a TTL-based method. Since the size of the network provided for simulation is small, the number of walkers is two and the TTL is limited to five. 300 search queries are submitted from different nodes in network. Based on the results, the number of hops visited by successful queries during search is noted. When an object is found, it will be copied into query node. In case of Q-learning replication, the query node further replicates the object to other nodes (figure 6). For path replication the object is replicated to nodes along the search path. The results of simulation are plotted on graphs 4, 5, 6, 7, and 8. From the graphs, we can conclude that Q-learning based replication outperforms path replication.

The simulation is extended for different TTL values and the result is shown in figure 9. For TTL values greater than equal to five, the result of search is relatively same for Q-replication.

Authorized licensed use limited to: NATIONAL INSTITUTE OF TECHNOLOGY SURATHKAL. Downloaded on February 23,2021 at 11:06:33 UTC from IEEE Xplore. Restrictions apply.

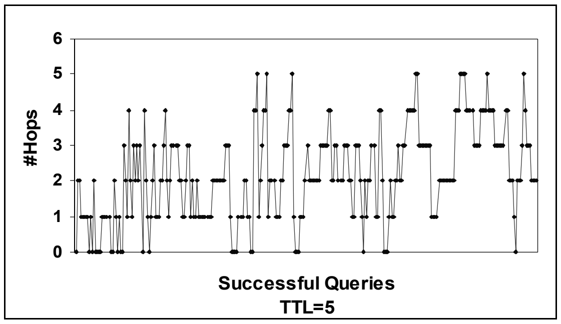

Figure 4: Number of hops for successful queries (Q- Replication).

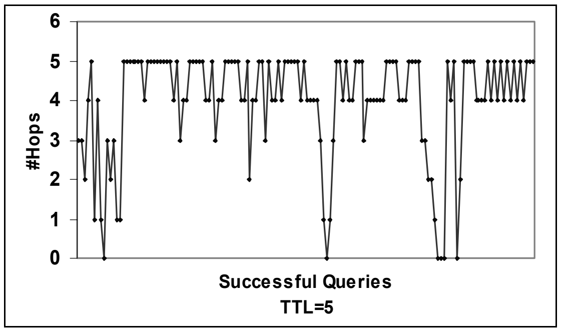

Figure 5: Number of hops for successful queries (path replication).

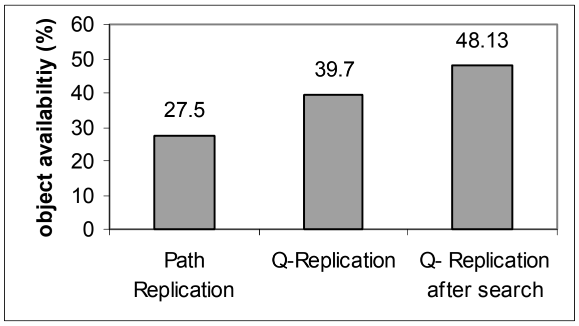

Figure 6: Objects' availability in percentages.

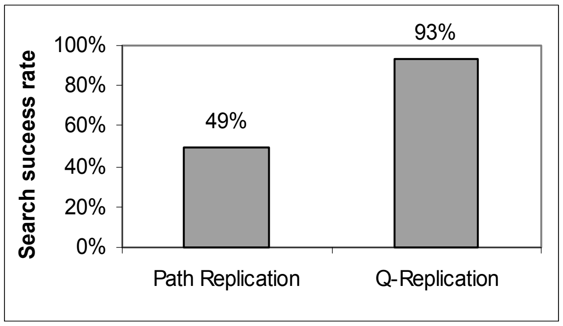

Figure 7: Success rate for 300 queries.

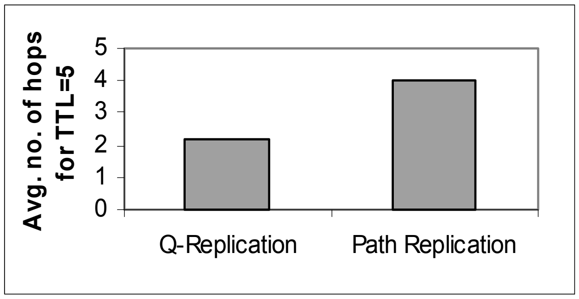

Figure 8: Average number of hops visited during search.

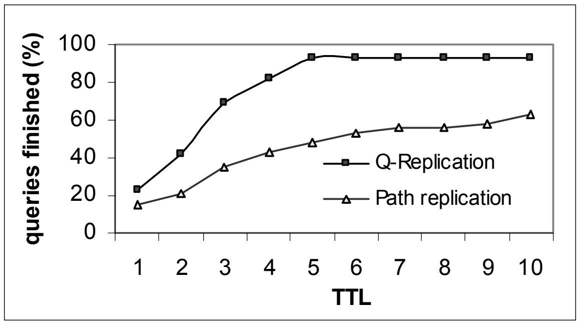

Figure 9: Queries finished for different TTLs under Q-replication and path replication.

8. Conclusion

We have proposed a decentralized model for creation of replicas in an unstructured peer-to-peer system. The aim of our model is to increase the data availability and thus to increase the search performance. The main advantage of the Q-learning based replication is reduction in number of hops visited by a search query. Accordingly, the network traffic is reduced further. The availability of objects is more than 48%, which is higher than the performance of path replication. The Q-replication does not rely on search path, unlike path replication and random replication. However, the proposed algorithm may consume network bandwidth when the node checks object's presence in the nodes listed in Q-table. This is not considered in the performance evaluation.

9. References

[1] Beverly Yang and Hector Garcia-Molina. Improving Search in Peer-to-Peer Networks. In *proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02).*

[2] Gnutella website: http://gnutella.wego.com.

[3] Timo Warns and Grenadierweg. Replication for Peer-to-Peer Systems to Improve Dependability.

Authorized licensed use limited to: NATIONAL INSTITUTE OF TECHNOLOGY SURATHKAL. Downloaded on February 23,2021 at 11:06:33 UTC from IEEE Xplore. Restrictions apply.

Diploma Thesis, Department of Computing Science, Software Engineering Group, Carl Von Ossietzky University, Oldenburg.

[4] Yoshihiro Gotou. Replication Methods for Enhancing Search Performance in Peer-to-peer Services. http://citeseer.ist.psu.edu/640281.html.

[5] Q.Lv, E.Cohen, K.Li, and S. Shenker. Search and Replication in Unstructured Peer-to-Peer Networks. In *proceedings of ICS 2002*.

[6] Kavitha Ranganathan, Adriana, and Ian Foster, Improving Data Availability through Dynamic Model-Driven Replication in Large Peer-to-Peer Communities. In *proceedings of the Workshop on Global and Peer-to-Peer Computing on Large Scale Distributed Systems,* Berlin, May 2002.

[7] Vijay Gopalakrishnan, Bujor Silagi, Bobby, and Pete Keleher. Adaptive Replication in Peer-to-Peer Systems. In p*roceedings of ICDCS'04.*

[8] Francisco Matias, Richard P. Martin, and Thu D. Nguyen. Autonomous Replication for High Availability in Unstructured P2P Systems. In *Proceedings of the 22nd IEEE International Symposium on Reliable Distributed Systems,* 2003.

[9] L.Gatani, G.L.Re A. Urso, and S. Gaglio. Reinforcement learning for P2P Searching. In *proceedings of the International Workshop on Computer Architecture for Machine Perception (CAMP'05)*, 2005.

[10] Xiuqi Li and Jie Wu. Improve Searching by Reinforcement Learning in Unstructured P2Ps. www.cse.fau/~jie/research/publications/Publicatio n_files/p2pdak06.pdf.

[11] Aram Galstyan, Karl Czajkowski, and Kristina Lerman. Resource Allocation in the Grid Using Reinforcement Learning. In proceedings of International Conference on Autonomous Agents and Multiagent Systems –Vol. 3.

[12] Watkins C, Dayan P. Technical Note: Q-Learning. Machine Learning, 8, pp. 279-292, 1992.

[13] Pierre Yves Glorennec. Reinforcement Learning: an Overview. ESIT 2000, 14-15 September 2000, Aachen, Germany.

[14] Sabu M. Thampi, K. Chandra Sekaran. An Agent Based Peer-To-Peer Network with Thesaurus Based Searching and Load Balancing. In *proceedings of CIMCA-IAWTIC'05, vol 1.*

IEEE COMPUTER SOCIETY

Authorized licensed use limited to: NATIONAL INSTITUTE OF TECHNOLOGY SURATHKAL. Downloaded on February 23,2021 at 11:06:33 UTC from IEEE Xplore. Restrictions apply.