Collaborative Load Balancing Scheme for Improving Search Performance in Unstructured P2P Networks

***Sabu M. Thampi and Chandra Sekaran K*[1]**

L.B.S College of Engineering (Kannur University), Kasaragod, Kerala – 671542
[1]National Institute of Technology Karnataka, Surathkal, Karnataka – 575025
E-mail : smtlbs@yahoo.co.in, kch@nitk.ac.in

ABSTRACT

Peer-to-peer structures are becoming more and more popular and an exhilarating new class of ground-breaking, internet-based data management systems. Query load balancing is an important problem for the efficient operation of unstructured P2P networks. The key issue is to identify overloaded peers and reassign their loads to others. This paper proposes a novel mobile agent based two-way load balancing technique for dynamic unstructured P2P networks. In this scheme, target peers are selected based on the result of reinforcement learning. Simulation results indicate that our technique manages the load on peers effectively and increases the search performance significantly.

1. INTRODUCTION

In the past few years, peer-to-peer networks have become a popular method for extensive content sharing. Unlike conventional client-server applications, P2P systems distribute the load of data storage, computation, communications and management among thousands of peers. Peers can join and depart the network at any time, at their will. Since the distribution of data can be random, the data stored in a P2P network is spread across a large number of nodes. One of the most popular applications of P2P networks is file sharing.

The existing P2P systems are broadly classified into two types: unstructured P2P networks and structured networks. For unstructured networks, the data objects do not have global unique ids and queries are submitted as keywords. The peers in structured networks maintain unique identification tag for each object. Nowadays, most of the peer-to-peer applications function on unstructured P2P networks. This architecture demands a very efficient search technique for the retrieval of data [1]. A search for an object in a P2P network is successful if it discovers at least one replica of the object. Peers connect in an ad-hoc fashion, the location of the documents is not controlled by the system and no guarantees for the success or the complexity of a search are offered [2].

Search methods for unstructured networks can be grouped as either blind or informed. In a blind search, nodes do not store any information regarding object locations. In informed approaches, nodes locally store metadata that helps in the search for the queried objects. Existing blind methods ravage a lot of bandwidth to achieve utmost performance. Every search requires contacting several nodes within some distance called Time-To-Live (TTL), creating enormous overhead to all nodes involved. Informed methods use their indices to achieve similar quality results, and to shrink overhead. The limitation of most informed methods is the maintenance cost of the indices following peers join/leave the network or update the objects in the shared folder.

Load balancing is a main problem to be solved for the efficient operation of P2P networks. Most of the search methods cause serious load balance problems. One of the major issues is to identify overloaded nodes and reassign the incoming loads to others. This paper proposes a Distributed Search Technique (DST) to provide collaborative load balancing among peers. The algorithm is formulated with the aim of achieving good response time, high hit ratio, low network traffic and adaptive behavior. The main contributions of the proposed search algorithm are: Q-learning based search, two-way load balancing, priority for specialized nodes, power peer concept, and the application of query history details.

Like random walk – a blind search method [3], the proposed method does not select walkers randomly. Power peers and ordinary peers together join the search process. In order to achieve the objectives, the proposed search scheme maintains a few tables. The tables are updated according to search results. Queries are distributed to both ordinary peers and power peers simultaneously so that the crowding effect on a few peers can be reduced.

In a P2P network, a few nodes are high degree nodes and majority of the nodes have less number of neighbors. Due to this, high degree nodes are queried frequently, thus the performance is reduced due to heavy query load. Simultaneous searching among ordinary peers and power peers alone cannot balance the load effectively. Hence in the proposed scheme, query is routed to a power peer based on load data collected by mobile agents from various power peers.

The remainder of this paper is organized as follows. Section 2 reviews the related work. An overview of the proposed search technique is given in section 3. The Q-table update operations are discussed in section 4. Section 5 discusses the simulation methodology. Section 6 concludes the paper.

2. RELATED WORK

Flooding based search is extensively used in unstructured P2P networks like Gnutella. Flooding schemes generate a large amount of network traffic. To overcome this problem, a random walk [3], [4] technique is often used. Whereas this approach manages to reduce messages significantly, it shows low performance because of its random character and inability to adjust to different query loads. The proposed technique makes use of Q-table data for selecting walkers.

Adaptive Probabilistic Search (APS) [5] forwards a single file look up query probabilistically based on the query history and the guesses of query sources. APS can be viewed as an ad-hoc application of reinforcement learning [6]. APS assigns equal status for all the nodes in the network while searching, without considering the nodes' degree, number of available objects and storage. Our method does not follow probabilistic forwarding; as an alternative, it uses Q-learning for selecting peers. In Gnutella UDP Extension for Scalable Searches (GUESS) [7], each ultrapeer is linked to other ultrapeers and to set of leaf-nodes. During a search operation, different ultrapeers are iteratively contacted followed by searching in their leaf-nodes. However, the order in which ultrapeers are chosen is not specified [2]. In Gnutella 2.0 [8], while a super-peer receives a query from a leaf node, it forwards it to appropriate leaves and to its neighboring super-peers.

In Intelligent-BFS [9], nodes maintain tables to store query-neighborID tuples for recently responded requests from their neighbors. The accuracy of the algorithm depends on the assumption that nodes specialize in certain documents [2]. Reinforcement learning based search [6] explores new paths by forwarding queries to randomly chosen neighbors. It selects the best path from the returned results.

Several solutions have been proposed to address the load balancing issues in structured P2P networks using the concept of virtual server [12, 13]. None of the strategies is suitable for unstructured P2P networks. The unstructured networks lack the distributed hash tables and unique identifiers. Due to these, load balancing is even harder in unstructured networks. Our scheme utilizes a distributed load balancing scheme for improving search performance in unstructured networks.

3. OVERVIEW OF DST

Reinforcement learning (RL) is a powerful framework in which an agent learns most favorable actions through a trial and error exploration of the environment and by receiving rewards for its actions. The agent's goal is to maximize the total reward it receives [10]. Q-learning is a new form of reinforcement learning algorithm that does not need a model of its environment.

In the context of P2P search, Q-learning is used to select suitable peers for searching. More than one walker is required to carry out a search operation. For this reason, rather than selecting the highest Q-value, depending on number of walkers, more peers are selected in line with their Q-values.

3.1 Data Structures and Major Features

The various data structures and important features of distributed search algorithm are discussed.

Query Q-table: Every time user enters a query, the peer's shared folder is searched and if the object is not found, the system checks whether an entry for the query keyword exists in the Query Q-table (Table I). Incase the query keyword is present in the table, K walkers are chosen from the query Q-table in the descending order of Q-values. For a successful search through a neighbor, corresponding Q-value is modified according to number of hops and results; otherwise, penalty is awarded. For all neighbors who have responded with successful results, associated entries are added to the table. The Q-table contains the list of most recent past queries and Q-values. The table grows as the entries for successful queries with new keywords are added.

Table 1: Querry Q-Table for Six Neighbors of a Node

	Q-values of neighbors					
Query Keyword	N1	N2	N3	N4	N5	N6
Peer	45	110	77	65	78	34
Sky	62	81	117	45	56	87
Moon	56	62	87	67	43	115

Table 2: Neighbor Q-Table for six Neighbors of a Node

N1	N2	N3	N4	N5	N6
175	85	78	134	50	89

Neighbor Q-table (NQ): A peer maintains a Neighbor Q-table (Table II**)** which contains Q-values of neighbors, NQ(n_1, n2…..n_n). Occasionally the query keyword may be a new one; hence, appropriate data may not be available in the Query Q-table. In this case, walkers are selected from both Neighbor Q-table and power peer Q-table in the descending order of Q-values. The Neighbor Q-table provides an overall picture with reference to the performance of neighbors in the past.

Power peers and Mobile Agents: Power peers are similar to ultrapeers but they declare themselves as power peers whenever some criteria are met. Existing systems select ultrapeers by their computing capabilities such as bandwidth, CPU power, and memory spaces [9]. In this paper, parameters such as number of neighbors (degree of a node), number of shared objects, and available storage are used to select a power peer. The presence of large number of objects can provide improved success rate. High degree peers have large number of neighbors. On the other hand, several peers query power peers for results. Even though a peer is powerful for housing large number of objects, it should have minimum storage available for hosting new objects in the future. The minimum level may be the user choice, say 30% of the total storage. A peer achieves power peer status when the number of objects, number of neighbors and available storage reach some threshold.

The moment a node becomes a power peer, it broadcasts the news to all the nodes within N hops away by dispatching mobile agents. Clones of mobile agents are created to visit several sites. The broadcast message is also propagated through neighbors and power peers listed in the power peer table. A node maintains a list for power peers in its power peer Q-table, PQ(p_1, p2…p_p). The format of the table is same as neighbor Q-table. An entry for power peer is added to the table each time a node receives broadcast message or the requested object is found in another power peer, which is not listed in the power peer table. Each node, irrespective of its class it belongs as a power node or an ordinary node, maintains a variable to store the number of hits occurred in the node. During search if a hit occurs in a power peer and the entry of that peer is not listed in the power peer Q-table, it is added to the table with initial Q-value 100.

After a hit, the node that holds the object (object node) transmits the reply message along the reverse path. Some of the parameters in the reply message include query source-id, message-id, address of object node, and its status (power peer or ordinary peer). In case, the node status is 'power peer', and the entry for that node is not there in the power peer table, new entry for the power peer is added by the nodes along the reply path. Query source-id is required for TTL enhancement operation.

Walker selection: If the query to be processed is a new one (i.e. entry for that query is not listed in the Query Q-table), walkers are selected from neighbor Q-table and power peer Q-table. Assume K-walkers are used for searching. Using Q-values in appropriate tables, after setting TTL values, neighbors and power peers are selected as walkers. Merge the values in NQ(n_1, n2…n_n) and PQ(p_1, p2…p_p) into one table in descending order. Select first K nodes from the list for routing query messages. Thus both ordinary peers and power peers are chosen for routing queries.

Message identification: A query source generates K messages for walkers. The query source forwards the query to K nodes based on the walker selection policy. The nodes on the path forward it to only one. Since messages are forwarded or processed by both ordinary peers and power peers, it is necessary to identify the preceding source of query. As mentioned earlier a power peer merely forwards a query message to another power peer. The messages from the walkers selected from neighbor Q-table are forwarded to their neighbors. The neighbor may be a power peer or an ordinary peer.

To identify the previous query resource, each message carries an identifier. While *query source* dispatches the message to its neighbor, the status of message is '0' even if the neighbor is a power peer This gives equal priority to all the nodes in the neighbor list. The situation changes if the peer in the next hop is a power peer. The power peer subsequently replaces the identifier value by '1' and from there onwards, the message is simply forwarded to power peers. The query message dispatched from a power peer listed in the power peer Q-table carries an identifier value equal to '1' and the message is further forwarded through power peers. Therefore, value of the identifier does not change until query is dropped. Each message generated from a query source carries a unique-id to identify itself from other query messages. A peer stores recently processed or forwarded message-ids. The node to which the message is routed and the address of previous node, which forwarded the message to the node, is kept in a table called *message history table*. This is useful when the result is sent back on the reverse path. The table follows a First-in First-out strategy for removing entries.

Duplicate Messages: A duplicate message is forwarded to another neighbor or power peer based on Q-value of the node and class of message. The target peer is selected by excluding the nodes, which forwarded the message earlier. If the Q-value is greater than or equal to 100, a node with the next highest Q-value is chosen as the target node; subsequently message is forwarded to the selected node. If no nodes are at hand, duplicate messages are discarded. For example, in Table II, assume node N has forwarded a query message first time to N1, which has the highest-value in the neighbor Q-table. Next time N receives the same message, it is forwarded to N4 because its Q-value is greater than 100. Other incoming duplicate messages are discarded, as no other neighbor with required Q-value exist. Ordinary peers and power peers follow the same policy for forwarding duplicate messages.

Specialized peers: Occasionally search process returns multiple results from different peers for a query. At the same time, a single peer may also produce more than one matching result. This means that the peer holds several similar objects of same subject area. The chance of getting more results for such queries is high. These nodes are called 'specialized nodes' and they are given importance while updating Q-values.

Load balancing: Power peers may be overcrowded by incoming query messages. The proposed algorithm performs load balancing on power peers in two ways. The distribution of query processing load among neighbors and power peers tends to reduce load among power peers. This method of load balancing alone is not adequate to save high degree power peers from crowding because queries are also directly passed through power peers frequently. So, an effective load balancing scheme using mobile agents is presented.

The scheme utilizes load information available on power peers. When a power peer is overcrowded, the system can direct the query traffic to least loaded power peers. A mobile agent periodically collect load data from power peers selected from power peer list. The scheme works as follows: The average of Q-values (AvgQ) of peers listed in the Q-table of a power peer is computed. Peers with Q-values greater than or equal to AvgQ are selected for collecting load information. Every peer is provided with a mobile agent platform.

Clones of mobile agents are dispatched from a power peer to collect load data from selected power peers. The agent collects cpu-load and free memory on each node and computes load metric as, *load= w1*cpu_load + w2 * free_mem,* where *cpu_load* is the work load on the power peer measured in the length of the job queue, *free_mem* is the percentage of free memory space, and w1, w2 are the weights of the parameters, w1 + w2 =1. The load data is reported back to the parent node and it replaces the previous data.

Search termination: All the neighbors and power peers who received the query message follows k-walk procedure for searching. As mentioned earlier a power peer forwards a query message to another power peer only. This will continue till TTL expires. The result is sent back to the requester node on the reverse path. The nodes in the path update their Q-tables accordingly.

4. Q-TABLE UPDATE

This section explains how the Q-learning process is employed in ordinary peers, and power peers to compute rewards and update Q-values in different Q-tables [11]. The initial Q-value for a node is set as 100.When a required object is found, all peers on the reverse path update the Q-values. The reward computation and Q-value update process are discussed.

Query Q-table update: For each hit, reward is computed and Q-values in Query Q-tables of each node coming between requester node and node, in which the object is found, are updated on the reverse path. Each result carries a reinforcement signal containing the number of hops (hp) visited by the peer and the number of results (nr) returned for the query. The reinforcement signal is translated into a reward (r_{nq}) function.

$$\rho_i = [a_i * 1/hp + (1- a_i) * nr] * 100 \qquad ...(1)$$

$$r_{nq} = \text{sign}(\rho_i) \qquad ...(2)$$

Since less number of hop count results good response time, the value for a_i is set at a_i=0.2. Specialized nodes may generate a number of matching results; for this reason, weight (1- a_i) is associated to number of results (nr) returned. The Query Q-table is updated for a particular query word using the Q-function

$$Q_{i,t+1} \leftarrow Q_{i,t} + \alpha (r_{nq} - Q_{i,t}) \qquad ...(3)$$

α is the learning rate. The Q-values of neighbors (walkers) that positively responded are updated. All other walkers receive a negative reinforcement (r_{nq} =0). The reward of those nodes are zero, the Q-value is updated as $Q_{i,t+1} \leftarrow Q_{i,t}(1- \alpha)$. The neighbors who have not participated in the search process keep the Q-values as such, i.e. $Q_{i,t+1} \leftarrow Q_{i,t.}$

Neighbor Q-table update: The Neighbor Q-table is updated for each query search operation. A hit is considered as reward. The Q-value of the node (walker) is modified as $Q_{i,t+1} \leftarrow (Q_{i,t.} + 10)$. In case the object is not found, the present Q-value is decremented by five, $Q_{i,t+1} \leftarrow (Q_{i,t.} - 5)$. Thus if a hit occurs, Q-values of all the successful walkers are incremented by 10, otherwise decremented. Q-values of remaining neighbors who have not participated in searching remain unchanged. Update process in a neighbor Q-table also results addition of new entry into Query Q-table. Therefore, the keyword and appropriate Q-values of neighbors are added to the query Q-table as per the update process.

Power Peer Q-table update: If past successful search data for the query keyword is not available in the query Q-table, walkers are also selected from power peer list of a node. Q-values are updated if a hit occurs through power peer (walker). The hop count (hp) is used as a parameter for Q-value update. The steps in calculating the reward, r_{pq} is explained below:

T = TTL.

$T_{max} = T + round\ (T/2)$ ...(4)

where, T_{max} is the maximum TTL allowed for a power peer, i.e. if the object is not found within the TTL limit, after checking process, search is extended to TTL/2 hops.

$r_{pq} = [T_{max} / hp] * 100$ ...(5)

For a hit, Q-value is updated as $Q_{i,t+1} \leftarrow Q_{i,t} + \alpha\,(r_{pq} - Q_{i,t})$. Q-values for the remaining walkers who have not produced a hit, update their Q-value as $Q_{i,t+1} \leftarrow Q_{i,t}(1-\alpha)$. No power peers not participated in the search alters their Q-values for the query.

5. SIMULATION METHODOLOGY

We describe the simulation environment and performance evaluation of distributed search technique.

5.1 Simulation Setup

The performance of the proposed technique is evaluated using a simulator developed in Java and IBM's Aglet Workbench. Aglets project is a Java based implementation that was originally developed by IBM Japan. An aglet can be dispatched to any remote host that supports the Java Virtual Machine. This requires from the remote host to pre-install Tahiti, a tiny aglet server program implemented in Java and provided by the Aglet Framework.

We simulated the search algorithms using random graphs that have 8000 nodes. There are 100 objects replicated to various nodes. The objects are replicated based on autonomous replication [11]. The query sources are chosen randomly. We assume that 80% of the nodes are up during simulation. The Q-values of neighbors and power peers in the corresponding tables are initialized with the value 100. Table III lists the various simulation parameters and their default values.

5.2 Performance Evaluation

We performed extensive simulations to assess the efficacy of proposed Distributed Search Technique (DST). The performance of the algorithm is compared with that of random walk and Adaptive Probabilistic Search (APS). The numbers of walkers vary from 1 to 15.

The search process follows two way searching: in case the keyword is not found in the query Q-table, walkers are deployed from neighbor Q-table and power peer Q-table. The selected walkers might have higher Q-values. This increases the chance of finding the object near the query source. There is no association between object updates and Q-values; Q-values are updated based on search results. The query is propagated to neighboring nodes and power peers simultaneously, which increases the possibility of finding rare objects from ordinary peers.

Table 3: Simulation Parameters

Parameters	Default Values
Topology	Random
Network type	Unstructured
No. of nodes	8000
TTL	06
No. of objects	100
Object Replication	Autonomous replication using Q-learning
Initial Q-value	100
Load balancing	Mobile agent based
Peers	Ordinary peers, power peers
Power peer selection	Node degree ≥7; available storage ≥ 30% of total storage allocated to the shared folder; No. of objects in a node ≥ 30

The simulation results are plotted as graphs and shown in figures. 1, 2,3,4,5, 6, 7 and 8. The success rates of three algorithms are presented in Fig. 1. DST has high success rate even for small K values and it outperforms both random walk and APS. In random walks, about 70% of the walkers fail and waste TTL messages each [5]. It is observed from Fig. 2 that average number of messages created by DST for a search operation is less than APS and somewhat

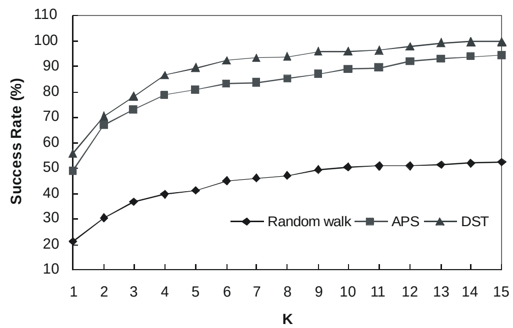

Fig. 1: Success rate vs. number of deployed walkers

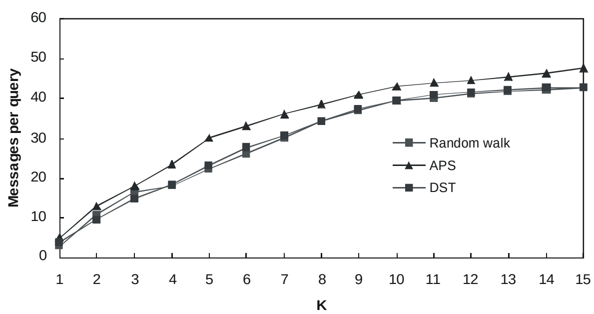

Fig. 2: Message per query vs. no. of deployed walkers

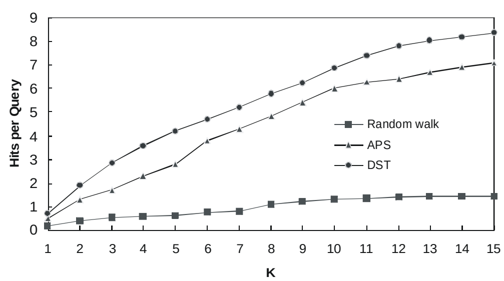

Fig. 3: Hits per query vs. number of deployed walkers

greater than random walk. This is because of the two-way search scheme followed, and use of past query data. Therefore, the majority of search actions create hits before they arrive at the TTL limit. Besides, the mobile agent based load balancing scheme assists the search process to keep away from heavily loaded power peers to diminish network traffic.

The number of objects discovered per query for different number of walkers is presented in Fig. 3. The distributed search algorithm generates more precise results than random walk and APS. DST achieves this much of performance by effectively utilizing Q-values of better performing nodes including that of specialized peers.

Fig. 4 compares the average number of hops visited for a search operation by the three search schemes. Performance of DST is superior to both APS and random walk. This is attained by exploiting the Q-tables data, load balancing and two-way searching. Power peers host several objects as compared to ordinary peers.

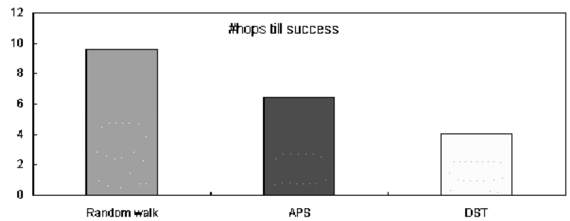

Fig. 4: Search delay Comparison

Fig. 5 shows the link between query hits and hop distance for three search schemes. The distributed search algorithm finds out large number of objects for short hop distances. This reduces the number of messages for search operation. In case of random walk, this cannot be achieved because no knowledge about objects in other nodes is available while walkers are deployed. In case of DST, neighbors and power peers together participate in a search operation. The participation of both categories of nodes is essential for a successful search in case the query keyword is a new one.

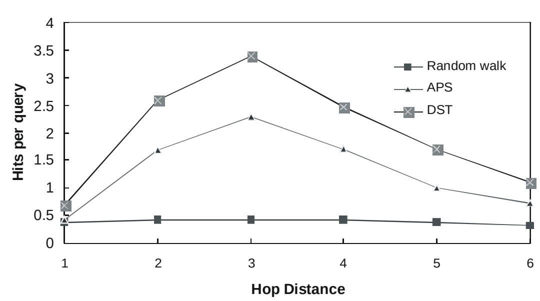

Fig. 5: Hits per query vs. hop distance from requesters

APS discards duplicate message while processing a query. However, DST forwards the message to possible nodes as per the node selection policy for duplicate messages. Query is effectively routed through neighbors and power peers. This causes reduction in number of duplicate messages during searching. This is evident from Fig. 6.

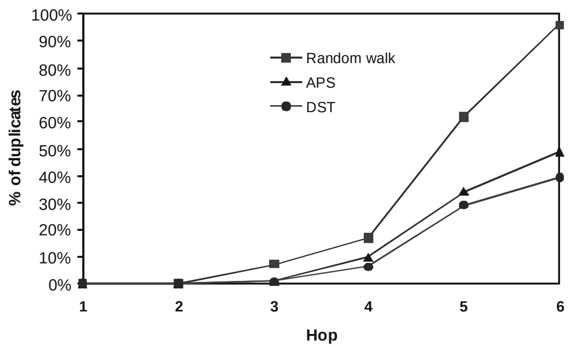

Fig. 6: Percentage of duplicate messages generated per hop

The simulation is also conducted with the presence and absence of mobile agent based load balancing for evaluating the effect of load balancing scheme on 50 most loaded power peers. As shown in Fig 7, mobile agent based load balancing scheme effectively distribute the load among power peers.

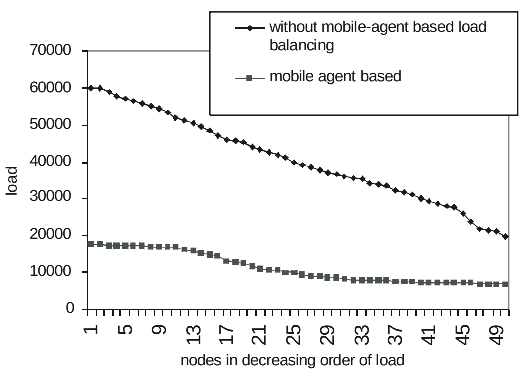

Fig. 7: Load balancing using mobile agents

The dynamic nature of P2P network is studied by removing a certain number of peers randomly. Simulation is conducted for different number of walkers (K) with varying peer availability. Fig. 8 shows success rate with K values varying from 1 to 15 and node availability from 10% to 80%. The scheme makes fine results even if 70% of the nodes are down.

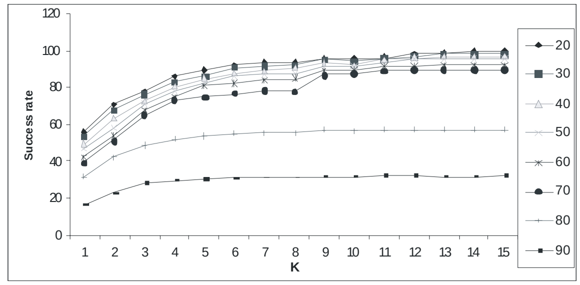

Fig. 8: Success rate for varying peer availability

6. CONCLUSIONS

In this work, we introduced a two-way load balancing scheme for unstructured P2P networks. Simulation results are also presented. The search scheme uses power peers, specialized peers, and mobile agents. Basic idea is to distribute the search processing load on ordinary peers and power peers for achieving better load balancing. The load on power peers has been reduced further after applying the mobile agent based load balancing technique. Simulation results show that DST achieves, improved load balancing and query success rate.

REFERENCES

1. Prakash. A, "A Survey of Advanced Search in P2P Networks," Available: www.medianet.kent.edu/surveys/IAD06Sp2psearch-alok/index.html.
2. Tsoumakos.D and Roussopoulos.N, "Analysis and Comparison of P2P Search Methods," in 2006 Proc.1st Int. Conf. Scalable Information Systems (INFOSCALE 2006), Article No. 25.
3. Lv. C, Cao.P, Cohen.E, Li.K, and Shenker. S. "Search and replication in unstructured peer-to-peer networks," in 2002 Proc.16th Int. Conf. Supercomputing, pp. 84-95.
4. Gkantsidis. C, Mihail. M, and Saberi.A, "Random walks in peer-to-peer networks," in 2004 Proc.23rd Annual Joint Conference of the IEEE Computer and Communications Societies, Volume 1, Page. 130.
5. Tsoumakos. D, and Roussopoulos. N, "Adaptive Probabilistic Search for Peer-to-Peer Networks," in 2003 Proc.3rd Int. Conf. P2P Computing, pp. 102 – 109.
6. Li. X, and Wu. J, "Improve Searching by Reinforcement Learning in Unstructured P2Ps," in 2004 Proc.26th Conf.Distributed Computing Systems ,pp. 75.

7. Daswani. S and Fisk. A, *Guess protocol specification*, Available: http://groups.yahoo.com/group/the gdf/files/ Proposals/ GUESS/ guess 01.txt.
8. Stokes. M, Gnutella2 Specifications Part One, Available: http://www.gnutella2.com/gnutella2.
9. Kalogeraki. V, Gunopulos. D and, Zeinalipour-Yazti. D, "A local search mechanism for peer-to-peer networks," in 2002 Proc.11th Conf. Information and knowledge management, pp.300-307.
10. Galstyan. A, Czajkowski. K, and Lerman. K, "Resource Allocation in the Grid Using Reinforcement Learning," in 2004 Proc.3rd Conf. AAMAS'04 – Vol. 3, pp.1314-1315.
11. Sabu M. Thampi and Chandra Sekaran. K, "Autonomous Data Replication Using Q-Learning for Unstructured P2P Networks," in 2007 Proc.6th Conf. IEEE Int. Sym. NCA 2007, pp. 311-317.
12. Rao. A, Lakshinaryanan, K, Surana, Karp, and Stoica," Load Balancing in Structured P2P Systems," In proc. of IPTPS 2004, pp. 68-79.
13. Godfrey B, Lakshiminarayanan K, Surana, Karp, and Stoica, "Load Balncing in Dynamic Structured P2P Systems," in proc. INFOCOM 2004, pp. 46-50.