FFNN 을 사용한 P2P 디바이스 디스커버리

O
차크라*, 권기현*, 김상춘*, 변형기*, 김남용*
*강원대학교 정보통신공학과
e-mail : kweon@kangwon.ac.kr

Device Discovery in P2P Environment using Feed Forward Neural Network

Chakra B. Balayar*, Ki-Hyeon Kwon*, Sang-Choon Kim*, Nam-Yong Kim*, Hyung-Gi Byun*
*Dept. of Information & Communication Engineering, Kangwon Natl. University

요 약

P2P(Peer to Peer) 기술은 1990 년대 후반기부터 산업계 및 학계에 주목을 받고 있는 기술 분야중의 하나로 이 기술의 장점은 인터넷 환경에 산재하여 있는 컴퓨팅 파워, 공간, 네트워크 대역을 인터넷 기반으로 효과적으로 활용하여 협력작업을 가능하게 한다는데 있다. 최근에는 모바일 환경 응용을 위한 P2P 디바이스 탐색 분야에 관심사가 증대되고 있으며, P2P 시스템은 중앙통제 장치가 결여 되어 있기 때문에 중앙통제 장치 개입을 최소로 하면서 P2P 를 운영하기 위한 효율적인 기법 및 체계가 요구되고 있다. 본 논문에서는 기존의 접근방법을 검토하여 FFNN(feed forward neural network)을 이용한 디바이스 탐색 기법을 제시한다. 제시한 FFNN 은 BP(back propagation) 알고리즘을 통해 훈련하고 디바이스를 탐색한다. 제시한 시스템의 성능을 보이기 위해 일정한 계산량을 가지는 작업을 에이전트를 활용, 탐색된 디바이스간에 분배하여 처리한다. 본 논문에서는 제한된 자원을 가지는 디바이스 간에 P2P 를 사용하는 기법에 대해 제시하였다.

1. INTRODUCTION

Peer-to-Peer, commonly known as P2P, has been gripping the WWW rapidly making itself choice of many end users for sharing resources, disseminating information and distributing task.[1,6] One main reason is due to economic slowdown where users are looking for getting maximum benefit from the hardware they have. Some prominent advantages of P2P systems are greater bandwidth, more computing power (storage, memory, CPU cycles) available, and more people connected and more data generated. In its short period, P2P systems have overtaken client-server model due to its unique character i.e. every networked device acts as both client and server [1]. However device discovery still remains the biggest hurdle within P2P systems. The overnight rise of Napster [2], a music file sharing application, first brought P2P in limelight which led a lot of research interests on P2P related applications like Gnutella [3] and Kazaa [4] and JXTA [5] etc.

Furthermore, millions of users connecting to the Internet have started to work in group, in collaboration knowingly or unknowingly possessing the possibility of becoming supercomputers if integrated properly. It is believed that less than half of present computer processing power is used in real [8,9]. The question arises, “How can we utilize wasting computing power in P2P environment?” What can be better way to discover those devices which are sitting idle or possess sufficient free CPU cycles?

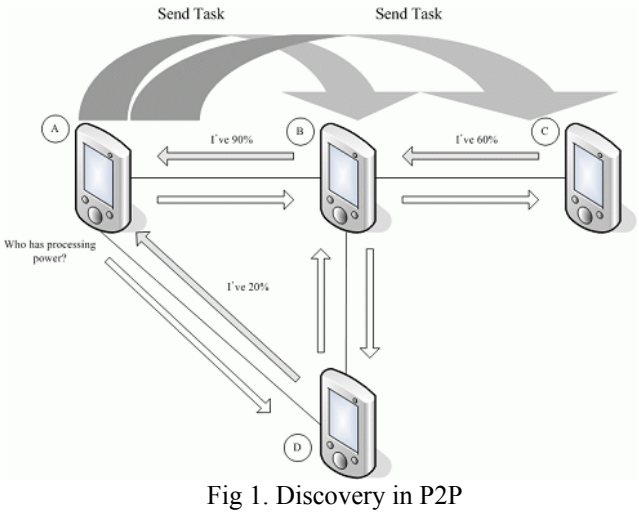


Fig 1. Discovery in P2P

Recently, researchers have started to apply Neural Network (NN), a human decision making like phenomena, in various computing fields to give intelligent solution. In this paper, we implemented feed forward neural network [10] for device discovery trained with BP algorithm [11]. FFNN uses various factors as input values such as free CPU, the number of neighbors any device has and the number of hops etc. Then, we used agent technology to solve any large computation, diving and distributing the computation task among those found devices in parallel way.

The paper is structured as follows. Section II discusses the related works. Section III describes about neural network. Section IV introduces our proposed architecture for device discovery where as section V describes the implementation and preliminary results in section V. Finally, we have summarized the conclusions in section VI.

2. RELATED WORKS

2.1. Napster and Gnutella

In Napster, a node sends request query to the centralized server which begins search with those nodes which are registered in it. After successful search, the file exchange occurs direct between nodes without any control from the server. The problem is in the central information storage which means single point failure, non-scalable and the requirement of central administration.

Gnutella algorithm propagates the request to all its neighbors until TTL (time to live) value. As search completes, the result is forwarded to their neighbors and finally to the requested node. But the flooding produces high overload to large number of peers and it doesn't scale too. There is also problem to define proper value for TTL.

2.2. Random Walks

In Random Walks, the query sender node sends k-query messages to those k-nodes which are selected in random basis. The queries are known as walkers and terminate either on success or failure. Though it decreases the number of messages or search results significantly [12,13], the result varies according to network topology and random choices.

2.3. Intelligent BFS

It's modified version of BFS (Breadth First Search). Nodes keeps the update information about their neighbors in order to rank them which helps the node to forward the query to the selected neighbors that have returned the most results for similar queries. Though it scales well in accuracy and knowledge sharing and it also induces no overhead during nodes arrival/departure, but it produces large number of messages and shows no easy adaptation of resource deletions or peer departures.

2.4. Adaptive Probabilistic Search (APS)

In APS [14], each node keeps track of the success rates of earlier queries for particular resource which reflects the relative probability of this node's neighbor to be chosen as the next hop in a future request for that particular resource. The searching is based on random walks forwarding the query to one of its neighbor with probability. It updates the indexing using feedback from the walkers. APS has feature of learning. It induces zero overhead over the network at join or leave/update and also high degree of robustness.

2.5. Local Indices and Routing Indices

In Local Indices [15], each node indexes the files stored at all nodes inside a certain peripheral, i.e. within a certain radius. And, it can answer on behalf of those nodes performing the search in BFS-like manner. But, in Routing Indices [16], documents are supposed to fall into a number of thematic categories and each node has information about approximate number of documents from each category that can be retrieved through each outgoing link. The forward process is similar to DFS (Depth First Search) and index maintenance requires flooding messages initiated from nodes that arrive or update their collections.

2.6. Distributed Resource Location Protocol (DRLP)

The nodes that have no information about the requested query forwards the query to all of its neighbors with certain probability in DRLP[17]. In case of found resource, the query takes the reverse path to the requester and registers the resource location and in the subsequent search, it contacts directly to the specific node.

3. ARCHITECTURE

3.1. Device Architecture

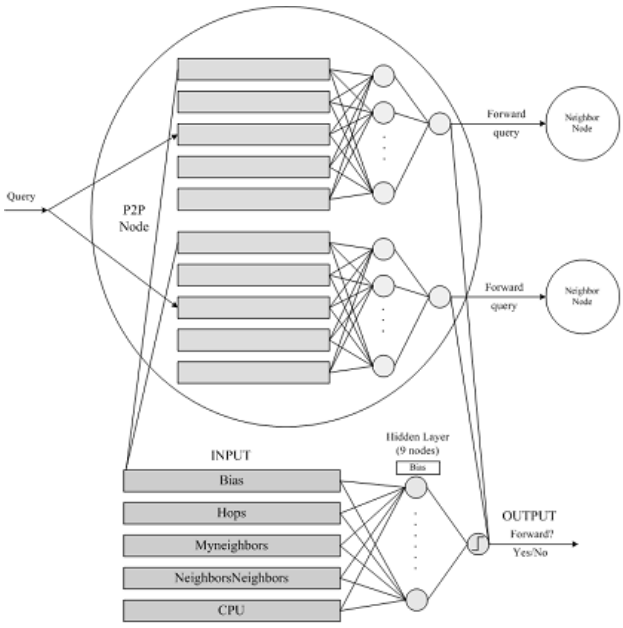


Fig 2. Query Processing of FFNN Architecture

As in figure 2, our device discovery architecture is based on feed forward neural network (FFNN). Once a peer gets any query, it decides where to forward or not running FFNN for all individual peers. In figure 2, the peer is connected with two other peers (devices). It runs FFNN for both peers and decides whether to forward or not to any particular peer. It can forward both of them or one or none according to the result of FFNN.

3.2. Feed Forward Neural Network

Our proposed feed forward neural network architecture is as in figure 3. It has an input layer, a hidden layer and an output layer. The input layer is connected to hidden layer and hidden layer to output layer as in the figure 3.

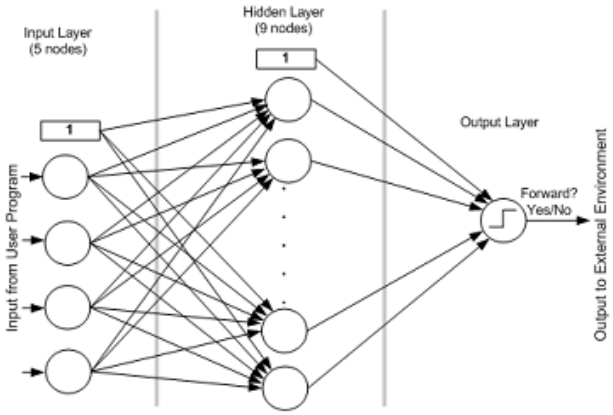


Fig 3. Feed Forward Neural Network Structure

The Input Layer : The input layer is the conduit through which the external environment sends a pattern to the neural network. It should represent the condition for which we're training the network for. As in figure 2, we defined input values as Bias, Hops, Myneighbors, NeighborsNeighbors and CPU where Bias is always 1, Myneighbors is the number of neighbors any node has, NeighborsNeighbors is the number of nodes any node's particular neighbor has. CPU is the available CPU percentile of any particular node.

The Hidden Layer : The hardest job in neural network is to define the number of hidden layers. Since neural networks with two hidden layers can represent functions with any kind of shape and with one hidden layer is enough for many practical problems, we used one hidden layer. The number of hidden layer's neuron is very important part of deciding the overall performance of any neural network since this layer influences the output most. Using rule-of-thumb, we decided 9-neurons for the hidden layer.

The Output Layer : Since our system needs to decide whether to forward or not forward, we fixed output as a single value which gets either 1 (forward) or 0 (not forward).

3.3. Algorithm

The BP algorithm is one of the most important and widely used training methodologies for neural network. Learning takes place based upon mean squared error and gradient descent. And, BP makes it easy to find the networks error weight gradient for a given pattern. It is sometimes known as generalized delta rule.

The steps of algorithm are as follows.

STEP 1 Initialize weights

1.1 Each weight in the network initialized to some small random value

STEP 2 For next pattern

2.1 Perform a forward propagation step

$$u_i = f(S_i)$$

$$\text{where } S_i = \sum^{j} w_{ij} u_j \text{ and, } f(x) = \frac{1}{1+e^{-x}}$$

First, net weighted sum Si is calculated and activation ui for each neurons using sigmoidal activation function.

2.2 Perform backward propagation

$$f'(x) = f(x)(1-f(x)) \text{ or } f'(S_i) = u_i(1-u_i)$$

If u_i is an output unit,

$$\delta_i = u_i(1-u_i)(C_i - u_i)$$

if u_i is a hidden unit,

$$\delta_i = u_i(1-u_i)\sum^{h} \delta_h w_{hi}$$

Error is calculated starting from outputs and propagated back to the hidden layer and input layer as above. Di is the weighted sum of the errors.

2.3 Update weights

$$w^*_{i,j} = w_{i,j} + \rho \delta_i u_j$$

Weight update is done online immediately after the forward propagation as above. Momentum term was added to reduce the training time.

$$w^*_{i,j} = w_{i,j} + \alpha \Delta w_{i,j} + \rho \delta_i u_j$$

where $\Delta w_{i,j}$ is the previous weight change.

And, α is the momentum term.

Weight update is done online immediately after the forward propagation as above.

STEP 3 Stop when total error is acceptable

3.1 Compute total error

3.2 If acceptable STOP, otherwise GO back STEP 2

Algorithm stops when the value of the error function has been sufficiently small.

4. IMPLEMNETATION ARCHITECTURE

4.1. Implementation Environment

We have successfully implemented our system on Intel PXA250 embedded kits (CPU 400MHz, RAM 64MB, Flash 32MB). There were 20-kits and they were arranged in an ad-hoc network using wireless LAN. and, we used Java as developing language (JDK1.3.1, JRE1.3.1) and Linux as operating system (Kernel 2.4.18).

4.2. Prototype Implementation

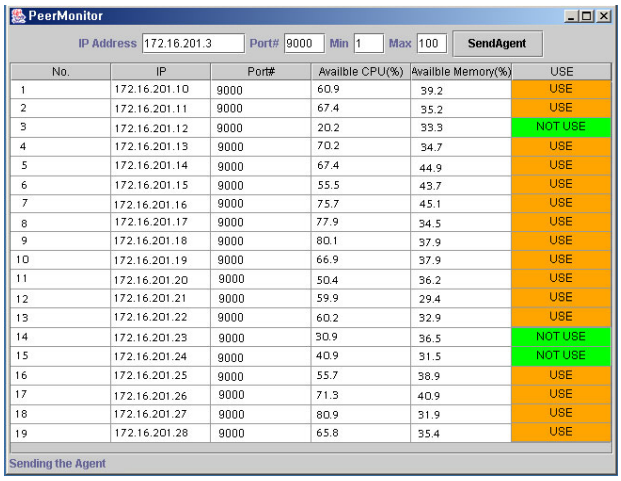
The prototype implementation consists of the following components: (1) a PeerMonitor (PM), (2) an agent and load balancing Mechanism, and (3) a Feed Forward Neural Network.

4.3 Training Feed Forward Neural Network

We formed four subnets from twenty embedded devices in our laboratory, placing five in each subnet and run FFNN training with BP to decide whether its any particular neighbor has sufficient resource or not.

Let's suppose the peer 1 has a task. It needs to decide whether some of its neighbor can help him or not, for example Peer 1, Peer 2, Peer 3 and Peer 4. First, Peer 1 secures the values for the parameter Hops, Myneighbors, NeighborsNeighbors and CPU in a set [Hops,, Myneighbors, NeighborsNeighbors, CPU] for each neighbor peers. Then, it feed these values to feed forward neural network and trains with backpropagation. In the case of Peer 1, since the query starts from Peer 1, so Hops value is zero, Myneighrbors value is 4 since it has four neighbors: Peer2, Peer 3, Peer 4, and Peer 5. And, the value of NeighborsNeighbors and CPU are different for each of its Myneighbors. Let's take the case of Peer 2, for whom Peer 1 gets NeighborsNeighbors value is 0 since Peer 2 has no neighbors and CPU value is the available CPU of Peer 2. Then, these values forms the set[0,.4,0,1] which is fed into the neural network for training.

Since Peer 1 has four neighbors, it first decides who are capable of computing the task. Then, it sends the task to them using agent (ex. XAgent). Similarly, Peer 3 and Peer 4 can again divide the task since they have neighbors. In this way, large computation is divided into smaller ones and computed by various available peers in collaborative and distributed manner where the unused CPU of each capable device (which FFNN decides) is used. All the peers have parameters Hops, Myneighbors, NeighborsNeighbors, CPU and their values for each neighbor.



No.	IP	Port#	Availble CPU(%)	Availble Memory(%)	USE
1	172.16.201.10	9000	60.9	39.2	USE
2	172.16.201.11	9000	67.4	35.2	USE
3	172.16.201.12	9000	20.2	33.3	NOT USE
4	172.16.201.13	9000	70.2	34.7	USE
5	172.16.201.14	9000	67.4	44.9	USE
6	172.16.201.15	9000	55.5	43.7	USE
7	172.16.201.16	9000	75.7	45.1	USE
8	172.16.201.17	9000	77.9	34.5	USE
9	172.16.201.18	9000	80.1	37.9	USE
10	172.16.201.19	9000	66.9	37.9	USE
11	172.16.201.20	9000	50.4	36.2	USE
12	172.16.201.21	9000	59.9	29.4	USE
13	172.16.201.22	9000	60.2	32.9	USE
14	172.16.201.23	9000	30.9	36.5	NOT USE
15	172.16.201.24	9000	40.9	31.5	NOT USE
16	172.16.201.25	9000	55.7	38.9	USE
17	172.16.201.26	9000	71.3	40.9	USE
18	172.16.201.27	9000	80.9	31.9	USE
19	172.16.201.28	9000	65.8	35.4	USE

Fig 4. PeerMonitor and Peer Status

The following was the execution snapshot in our laboratory.

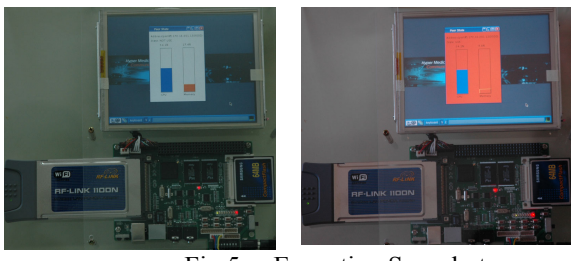


Fig 5. Execution Snapshot

5. CONCLUSION

This work showed new direction to solve this credential problem of discovery using intelligent mechanism, neural network. Feed Forward Neural Network (FFNN) trained with backprogation(BP) was used to discover the efficient devices from ocean of connected devices. Then, we distributed a large computation task using agent technology among those capable devices. This work also showed how to utilize the unused resources for handling any task. We implemented rudimentary neural network based architecture and achieved the feat to share or to lease the computing power/resources. Hence, P2P computing seems to have the potential to offer a better and intelligent solution in combination with neural network in device discovery. Although other techniques may prove accurate at the same task, the neural network seems to be suitable and sufficiently accurate choice.

Since the work is in preliminary stage, it needs to invest more research time to cement its legitimacy. Future works will cover performance evaluation and advantages over other technologies

REFERENCES

[1] Clay Shirky, "What is P2P and What isn't?", O'Reilly Network., 2000
[2] Napster, Inc., http://www.napster.com
[3] Gnutella Web Site, http://www.gnutella.com
[4] Kazaa Web Site, http://www.kazaa.com.
[5] Project JXTA web site, http://www.jxta.org
[6] A. Acharya, G. Edijlali, and J. Saltz, "The Utility of Exploiting Idle Workstations for Parallel Computation", SIGMETRICS 1997.
[7] M. Litzkow, M. Livny, M. W. Mutka, "Condour – A Hunter of Idle Workstations", In Proceedings of the 8th International Conference of Distributed Computing Systems, pages pp. 104-111, 1998.
[8] J. Hertz, A. Krogh and R. G. Palmer, "Introduction to the Theory of Neural Computer", Addison-Wesley, 1991.
[9] M. H. Hasson, "Fundamental of Artificial Neural Networks", MIT Press, 1995.
[10] J. Heaton, "Introduction to Neural Networks with Java", SAMS Publishing, 2003.
[11] D. McAuley, "Back Propagation Network: Learning by Example",http://www2.psy.uq.edu.au/~brainwav/Manual/BackProp.html 1997.
[12] S. Kurkovsky, Bhagyavati, "Modeling Computational Grid of Mobile Devices as a Multi-Agent System", International Conference on Artificial Intelligent, 2003.
[13] B. J. Kim, C. N. Yoon, S. K. Han, and H. Jeong, "Path Finding Strategies in Scale-Free Networks", Physical Review, 65, 2002.
[14] D. Tsoumakos and N. Roussopoulos, "Adaptive Probabilistic Search for Peer-to-Peer Networks", In 3rd, IEEE Intl. Conference on P2P Computing 2003.
[15] B. Yang, H. Garcia-Molina, "Improving Search in Peer-to-Peer Networks", in ICDCS, 2002.
[16] A. Crespo, H. Garcia-Mollina, "Routing Indices for Peer-to-Peer Systems", ICDCS, July 2002.
[17] T. Lin, H. Wang, "Search Performance Analysis in Peer-to-Peer Networks", In Proceedings of the 3rd International Conference on P2P Computing, 2003.