An Organization of System for Access Sharing to Information Resources

Radi Romansky[1], Iva Nikolova[2]

Abstract – **This paper examines and describes some organizational issues of the design of decentralized distributed information space that allows information resources as files to be shared in a synchronized and consistently manner amongst the distributed users. Our work focuses mainly on presenting some of the principles and technical approaches to the implementation of the networking and the system infrastructures, as well as on the possible approaches for optimization of the information access and effective scheduling of the distributed processes.**

Keywords – **peer-to-peer networks, distributed systems, distributed content sharing, file sharing, access sharing, collaboration**

I. INTRODUCTION

Organizing access to resources and deploying the computer and communication networks that underpin this access presents a number of challenges.

The sharing and coordinated use of resources is fundamental to an increasing range of computer applications. This sharing may involve not only file exchange, but also direct access to computers, software, data, and other resources, as is required by a range of collaborative problem-solving and resource-brokering strategies [1-3].

In the recent years, a lot of research interests focus on the content sharing networks and technologies [4-7] and their analysis and application. More and more popular become the computer applications that rely on the decentralization and equality of participants [8 -15].

The object of our current research interest is the design and organization of decentralized and dynamic, distributed information space, which to integrate the information resources that are shared by a group of users for support of research and educational process within the intranet network. The aim is to form a collaborative group of the PC's of the participants, in which the users are not only consumes of resources, but also are providers of such resources.

The main goal of this work is to offer an organization and implementation of collaborative application, in which the users can exchange information, such as educational materials. The problem actuality is verified by the work discussed in [11], [13], [14]. The computer application we consider here, represent a small file-sharing system. Its implementation is based on the architectural paradigm of the decentralized peer-to-peer network topologies [15].

The use of a collaborative environment based on peer-to-peer approach enables to take advantage of the following characteristics of the decentralized P2P systems: (1) *scalability* – constant complexity of the system regardless of number of nodes in the system; (2) *reliability* – peer nodes are connected directly without the need of a master server's arrangement, creating a flat, ad-hoc network topology, thus the malfunction on any given node will not effect the whole system. Only the data that has been stored on the peer that crashed gets lost. The rest of the system works as before.

Therefore, in the process of building of such distributed information space an important moment is the design of the policies of the system and very well understanding of all the scenarios that might occur. Following that principle, in the next sections, we will describe the main organizational features and technical aspects of implementation of the networking and the system infrastructures.

II. AN ORGANIZATION OF THE NETWORKING INFRASTRUCTURE

As it is well known the networking infrastructure sets up the base of all high-level operations.

The distributed information space is formed by the network integration of the PCs of the participants, called *member peer nodes* or only *peer nodes*. The system architecture, shown in Figure 1, is based on the decentralized model of P2P content-sharing networks [15]. It has a *Server-Client* structure where there is no centralized servers exists. All peer nodes are connected directly and are equal i.e. each peer node that is member of the collaborative group is empowered as both, a server of files and also as a client that can request files from other members, hence creating a flat network topology.

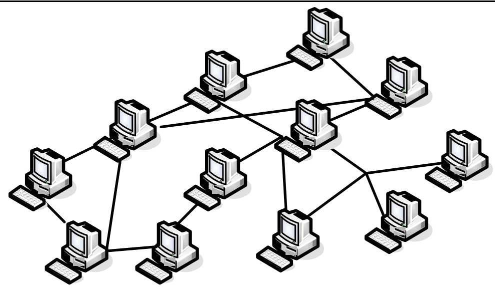


Fig.1. An illustration of the decentralized topology of the system

The system is designed to support two type operations – *join operations* and *file operations*. In order to join the network, a peer node must contact a node that is always

[1] Radi Romansky, Faculty of Computer Systems and Control, Computer Systems Department, Technical University of Sofia, Bulgaria E-mail: rrom@tu-sofia.bg

[2] Iva Nikolova, Faculty of Computer Systems and Control, Computer Systems Department, Technical University of Sofia, Bulgaria E-mail: inni@tu-sofia.bg

online, which give the joining peer the IP address of one the existing peer nodes that are already a members of the group. Each peer, however, will only have information about its neighbors, which are peers that have a direct edge to it in the network.

When the joining is made, a peer node can send its query (e.g searching for a particular file) to the other peer nodes in the network. File transfers made between member peer nodes are always done directly through a data connection that is made between peer sharing the file and the peer requesting for it.

The requirement for support of group working means that online resources may be shared and used by multiple concurrent readers and writers.

The collaboration amongst peer nodes can take two forms:

(a) each peer node can request local files, shared from remote peer node;

(b) each peer node can serve requests of remote peer nodes for local files it owns.

This form of collaboration between peer nodes requires concurrency control and usual concerns with liveness, safety and fairness [15].

To achieve a multiple concurrent queries for files to be processed simultaneously by a peer node, the network infrastructure is implemented as a multithread model. The simultaneously access of the multiple threads to the file directories is ensured by using critical section objects for each data structure.

Communications are built in the basis of numerous *messages passing.* A given message contains the request type and returned answer.

Two types of threads are empowered to take care of the system operations: *client* and *server* threads. The first ones are charged with client-side processing, forming the requests toward the server threads, and with the receiving the responses. They are created either by the main interface thread of the application or by other client threads. The second ones are created to correspond to the identity of the requesting threads and are responsible for the server-side processing. They accept all incoming connection requests and take the appropriate actions to their processing.

Join Process

As it was mentioned above, if a peer node wants to join the network, it needs to know the IP address of a peer node that is already a member of the group. The peer nodes will connect to each other and execute *Join Algorithm*. At its end, all the members of the collaborative group will know about the new member and the files that it has to share, and the new member will know about all the existing members and the files they have share.

We consider two steps join process: *Initial join* and *Group join*. The concept of the Initial join process is illustrated in Figure 2, where: *(1)* New peer node A connects to the Host Cache to get the list of available online member peer nodes already connected in the network; *(2)* The Host Cache sends back the list with member peer nodes to the peer node A; *(3)* Peer node A sends the connection request message to one of member peer nodes, for example peer node B; *(4)* Member peer node B replies with message granting peer node A to join the network. Next, new peer node A automatically receives a list with the IP addresses of all member peer nodes that are online. With these member nodes it begins the second step of the join process – *Group Join*. During that phase a connection with each of these nodes in the list is established. At its end, all the members will have information about the list of files that are available locally at the new peer node, as well as the new peer node will receives a list of files that are at the remote node.

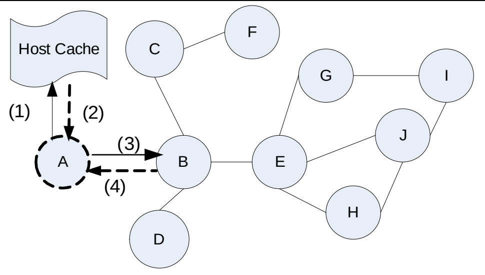


Fig.2. Join process.

From the point of view of the multithread model, there is a client and server parts of the join process that are implemented as client and server threads.

On receipt of the user's request to join in the group, the user interface server thread creates a *Initial Join Client Thread* (IJCT), maintained by *IJoinClientThreadProc()* function that sends a connection request *initial_join* to the listener server thread on the chosen remote peer node. As a result the server *AcceptThread* creates a new thread to serve that connection request. This server tread (IJST) is maintained by *IJoinServerThreadProc()* function.

The IJCT begins the join process by sending its computer name to the IJST. In return, the IJCT receives a list of the computer names of the existing member peer nodes. Next, the IJCT creates separate threads called *Group Join Client Thread* (GJCT) that establishes a connection with each of the peer nodes in the list. A *Group Join Server Thread* (GJST), maintained by G*JoinServerThreadProc()* function is created at each member peer node to correspond with the GJCTs on the non-member node. Then, by accessing a global data structures a GJCT receives a list of files that are available locally at that peer. This list is sent to the corresponding GJST. Mutually exclusive mechanisms and critical sections objects control the access to the global data structures, containing the identification of all member peer nodes and their files. These connections are broken with competition of join threads 'executions.

III. Design of the system infrastructure

To construct such a versatile working environment, in which the participated peer nodes can share and use information resources among themselves without relying on a dedicated server it is necessary all member nodes of the collaborative group to maintain information about all the members peer nodes and the files that they own. Additionally, in the context of the decentralized management, a natural tendency of such environment is to evolve over time (for example, with joining and leaving peer nodes along their resources). We are thus motivated to study how to improve the availability of shared files, so that users can access any file regardless of the current subset of online member peer nodes. Maintenance replicas of a particular file can achieve this. If replicas of files exist, they will serve the purpose of increasing concurrent reads. When member peer nodes need to download the latest copy of the file they can do from any location that holds a valid replica.

Figure 3 illustrates an example scenario of the collaboration amongst the member peer nodes: (1) Member peer node B and Peer node D collaborate on file; (2) Peer A access file on D.

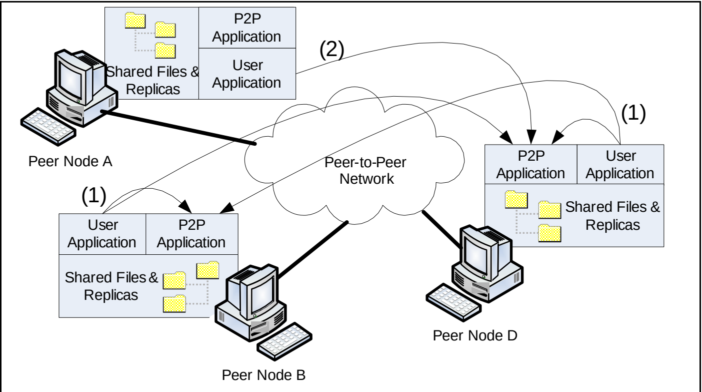


Fig.3. Example Scenario of collaboration

In reference to this, we consider the following system design features:

• All peer nodes must dedicate folders on their hard disks to store shared files or replicas. Before a peer node is ready to join the network, all the local files must be registered by creating replica lists for each file.

• Files present in the dedicated folder can be viewed or modified by all member peer nodes except in the case if a write operation is requested by a peer node for a file that is being written to by another peer node or that files are not valid replicas.

• When a peer node modifies a file, all previous replicas of that file lose their status of being a replica and become old copies of the file. Subsequent requests to use these old copies of the file will cause to download the latest copy of the file so that the users always work on a valid replica.

• When a peer nodes need to download the latest copy of a file they can do so from any locations that hold a valid replica. A concurrency can be improved by allowing a read operation on a file that is being written to. In this case the user should be warned that the copy being read might not be valid for too long. To optimize the number of file transfers that occur between peer nodes it considers a transfer to be only performing after comparing the local replica of file with the original file.

The system supports the following actions with shared files: adding a new files; removing files; reading a file that belongs to another member peer node; writing to a file that belongs to another peer node; reading a remote file by fetching it from one of its replica locations; choosing a location from where to fetch the replica of a file; getting "*Read*" permissions for a file that is currently being written to.

The following data structures are necessary to support these functional requirements. We will present here with a view of the object-oriented approach of the system implementation:

• Class *CPeerInfo* – each of the class objects contains the name of member peer node, node's IP address, a list of the files, shared by this node;

• Class *CFileInfo* – each of the class objects contains information about one of files, shared by a member peer node;

• *Class CreplicaInfo* – each of the class objects contains data about various locations, where copies of a particular file might exist.

The objects described are sorted in the linked lists:

• *Class CPeerInfoList* - contains the list of group members

• *Class CfileInfoList* – contains the list of all files, shared by a peer node

• *CReplicaInfoList* - contains the list of all peer nodes that have valid copies of a particular file. The first node in the list usually contains the identity of the owner of the file.

IV. System Functionality Tests

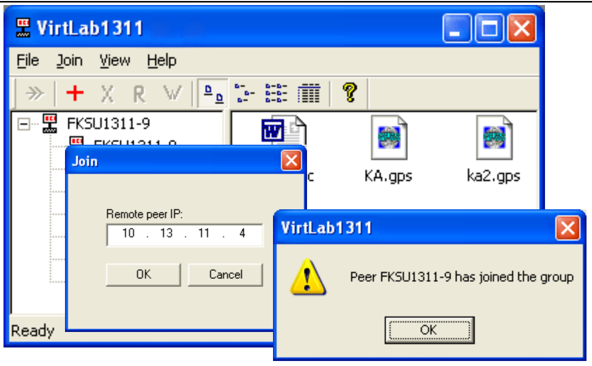
The P2P application has been developed on the top of the architecture described for file exchange. The actual file transfer happens over a direct TCP connection between the requester and the owner nodes. Socket classes are used to represent the connection between a client and server side.

The system is currently in a testing stage. It is completely developed in MS Visual C++. The graphical user interface has been programmed, using, document-view architecture, given by MFC, as a SDI Windows Explorer Style application.

System functionality was tested in the real conditions of the local computer nodes. The tests are carried out Windows XP Professional, using 12 computers connected via 100Mbps Ethernet LAN. The main purpose of the testing process was to check the correctness of our implementation of the algorithms and data structures in order to achieve the system functionality. In Figures below are shown screen shots of the tests we carried out with the application.

Bellow are presented some results of the experiments we are carried out to observe the system functionality:

• Testing the execution of the Join Algorithm (Fig. 4)

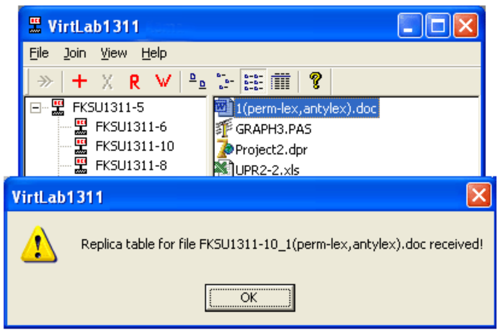


Remote peer IP: 10 . 13 . 11 . 4

Peer FKSU1311-9 has joined the group

Fig.4. New Peer Join - a node FKSU1311-9 joins to the node FKSU1311-4 and both form a group.

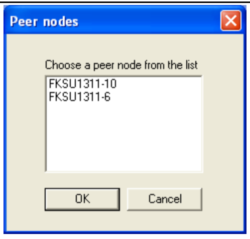
• Reading a file from remote peer (Fig.5) by fetching it from one of its replica locations



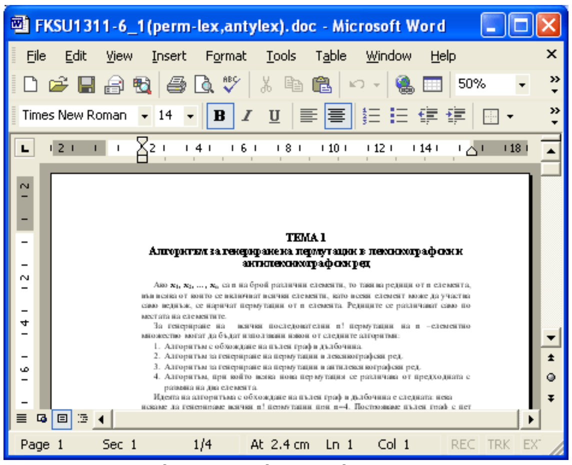
Replica table for file FKSU1311-10_1(perm-lex,antylex).doc received!

Fig.5. Receiving a replication table for a file, requested from node FKSU1311-5.

• Choosing a location from where to fetch the replica file (Fig.6).



(a) Choose a location


FKSU1311-6_1(perm-lex,antylex).doc - Microsoft Word
File Edit View Insert Format Tools Table Window Help
50%
Times New Roman
14
Page 1
Sec 1
1/4
At 2.4 cm
Ln 1
Col 1
REC TRK EX

ТЕМА 1
Алгоритми за генериране на пермутации в лексикографски и антилексикографски ред

Ако $x_1, x_2, \ldots, x_n$ са n на брой различни елементи, то такива редици от n елемента, във всяка от които се включват всички елементи, като всеки елемент може да участва само веднъж, се наричат пермутации от n елемента. Редиците се различават само по местата на елементите.

За генериране на всички последователни n! пермутации на n –елементно множество могат да бъдат използвани някои от следните алгоритми:

1. Алгоритъм с обхождане на пълен граф в дълбочина.
2. Алгоритъм за генериране на пермутации в лексикографски ред.
3. Алгоритъм за генериране на пермутации в антилексикографски ред.
4. Алгоритъм, при който всяка нова пермутация се различава от предходната с размяна на два елемента.

Идеята на алгоритъма с обхождане на пълен граф в дълбочина е следната: нека искаме да генерираме всички n! пермутации при n=4. Построяваме пълен граф с пет

(b) Read a file that is fetched from FKSU1311-6

Fig.6. Choice of location

V. Conclusions

This paper describes some aspects of the organization of decentralized file sharing system, which allows users to contribute and to share information resources amongst them. Our goal was to explore the feasibility of and the technical approaches to the implementation to some of the benefits of the architectural paradigm of the decentralized peer-to-peer network topologies. We have tried to build a collaborative group for a small number of peers as well as to outline the principal tasks of the design and development of the network and the system infrastructures, and the user interface that implements the system functionality. In reference to this, it is discussed some of the base algorithms and their implementation along with the required data structures in order to achieve the system functionalities.

There are several possible directions in expanding this application that might be a challenge, interesting to solve in the future: system performance characterization; providing of search mechanisms (discovering the desired information resources, giving its description); designing mechanisms that ensure robustness, security, file authenticity and access control. In the current implementation it considers that peer trust each other and there is no need for any authentication or encryption; User-level (instead of device-level) security and authentication.

References

[1] Foster, I., Kesselman, C. & Tuecke, S. (2001) “The Anatomy of the Grid: Enabling Scalable Virtual Organizations”, International Journal of High Performance Computing Applications, 15 (3), p 200-222, 2001.
[2] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman (2001), “Grid information services for distributed resource sharing”. In 10th IEEE International Symposium on High Performance Distributed Computing, 2001, San Francisco CA, IEEE Press.
[3] C. Shahabi, F. Banaei-Kashani (2002). “Decentralized Resopurce Management for a Distributed Continuous Media Server, IEEE Transactions on Parallel and Distributed Systems, Vol. 13, #6, June 2002.
[4] Open Source Community. The free network project – rewiring the internet. In http://freenet.sourceforge.net, 2001
[5] Fast Track Peer-to-peer technology company. In http://fasttrack.nu/, 2001
[6] KaZaA file sharing network. In http://www.kazaa.com, 2002
[7] Gnutella. In http://gnutella.wego.com 2001
[8] S. Ratnassamy, P. Francis, M. Handley, R. Karp, and S. Shenker (2001), “A scalable content-addressable network”. In Proceedings of the ACM SIGCOMM’01 Conference, 2001
[9] Nejl, W., Wolf W. et al “EDUTELLA: a P2P Networking infrastructure based on RDF”. In Proc. Of the 11th World Wide Web Conference (Hawaii, USA, May, 2002)
[10] Cuenca-Acuna, F. M., Peery, C., Martin, R. P., Nguyen, T. D. (2002), “PlanetP: using gossiping to build content addressable peer-to-peer information sharing communities”. Technical Report DCS-TR-487, Dept. of Computer Science, Ruthgers Univ. (2002)
[11] Vassileva J. (2002): Supporting Peer-to-Peer User Communities, in R. Meersman, Z. Tari et al. (Eds.) Proc. CoopIS, DOA, and ODBASE, LNCS 2519, Springer: Berlin, 230-247 (2002).
[12] Kan, G. (2001) “Peer-to-Peer: Harnessing the Power of disruptive technologies”, A. Oram (ed.), O’Reilly Press, USA, 2001.
[13] Bretze H., Vassileva J. (2003) “Motivating Cooperation in Peer to Peer Networks” Proceedings User Modeling UM03. Johnstown, PA, June 22-26, 2003, p. 218-227.
[14] Vassileva J. (2004) “Harnessing P2P Power in the Classroom”, In Proc. Of ITS’2004, Braazil, August 2004.
[15] Peter, B., Tim, W., Bart, D., & Piet, D., (2002), “A Comparison of Peer-to-Peer Architectures”, Broadband Communication Networks Group (IBCN), Department of Information Technology (ITEC), Ghent University, Belgium, 1-2, 2002.