Computer Science Journal of Moldova, vol.**16**, no.2(47), 2008

Using of P2P Networks for Acceleration of RTE Tasks Solving*

Adrian Iftene

Abstract

In the last years the computational Grids have become an important research area in large-scale scientific and engineering research. Our approach is based on Peer-to-peer (P2P) networks, which are recognized as one of most used architectures in order to achieve scalability in key components of Grid systems.

The main scope in using of a computational Grid was to improve the computational speed of systems that solve complex problems from Natural Language processing field. We will see how can be implemented a computational Grid using the P2P model, and how can be used SMB protocol for file transfer. After that we will see how we can use this computational Grid, in order to improve the computational speed of a system used in RTE competition [1], a new complex challenge from Natural Language processing field.

Keywords: Computational Grid, P2P Network, SMB protocol, RTE competition

1 Introduction

To achieve their envisioned global-scale deployment, Grid systems [2, 3] need to be scalable. Peer-to-peer (P2P) techniques are widely viewed as one of the prominent ways to reach the desired scalability. Resource discovery is one of the most important functionalities of a Grid system and, at the same time, one of the most difficult to scale. Indeed, the

*A preliminary version of this paper was presented at 9^{th} SACCS International Conference, Iasi, Romania, 16-17 October 2007

duty of a resource discovery system (such as the Globus MDS [4]) is to provide system-wide up-to-date information, a task which has inherently limited scalability. To add to the challenge, Grid resource discovery systems need to manage not only static resources, but also resources whose characteristics change dynamically over time, making the design critical.

The popularity of distributed file systems has continued to grow significantly in the last years, due to the success of peer-to-peer services like file sharing applications, VoIP applications, Instant messaging and TV on the Internet. Peer-to-peer systems offer a decentralized, self-sustained, scalable, fault tolerant and symmetric network of machines providing an effective balancing of storage and bandwidth resources. The peer-to-peer system described in this paper is **completely distributed** (*requires no form of centralized control, coordination or configuration*), **scalable** (*its nodes control only a certain region which is a part of the whole system and does not depend on the total number of nodes in the system*), and **fault tolerant** (*it can avoid some failures*).

Solving complex problems has become a usual fact in the Natural Language Processing domain, where it is normal to use large information databases like lexicons, semantic relations, dictionaries.

We will see how we can configure a peer-to-peer system in order to find the global solution for the Third Recognising Textual Entailment competition (RTE3) task. The system architecture is based on the peer-to-peer model, using the SMB protocol for file transfer. The system functionality was optimized by using a caching mechanism and a quota for synchronizing the termination of all processes.

After the peer-to-peer network is configured, one computer becomes the initiator and builds the list of available neighbors. Subsequently, the initiator has the following additional roles: split the initial problem into sub-problems, send the sub-problems to the list of neighbors for solving, receive the partial output files and build the final solution.

2 Computational GRID

Grid computing is a phrase in distributed computing which can have

several meanings [5]:

- A local computer cluster which is like a "grid" because it is composed of multiple nodes.
- This computer can offer online computation or storage as a metered commercial service, known as utility computing, computing on demand, or cloud computing.
- It can permit the creation of a "virtual supercomputer" by using spare computing resources within an organization.
- Also, it can create a "virtual supercomputer" by using a network of geographically dispersed computers. Volunteer computing, which generally focuses on scientific, mathematical, and academic problems, is the most common application of this technology.

These varying definitions cover the spectrum of "distributed computing", and sometimes the two terms are used as synonyms.

Functionally, one can also speak of several types of grids:

- *Computational grids* (including CPU Scavenging grids) which are focuses primarily on computationally-intensive operations.
- *Data grids* or the controlled sharing and management of large amounts of distributed data.
- *Equipment grids* which have a primary piece of equipment e.g. a telescope, and where the surrounding Grid is used to control the equipment remotely and to analyze the data produced.

Usually, a *computational Grid* consists of a set of resources, such as computers, networks, on-line instruments, data servers or sensors that are tied together by a set of common services which allow the users of the resources to view the collection as a seamless computing/information environment. The standard Grid services include [6]:

- security services which support user authentication, authorization and privacy

- information services, which allow users to see what resources (machines, software, other services) are available for use,
- job submission services, which allow a user to submit a job to any compute resource that the user is authorized to use,
- co-scheduling services, which allow multiple resources to be scheduled concurrently,
- user support services, which provide users access to "trouble ticket" systems that span the resources of an entire grid.

Our P2P system can be considered like a computational Grid, focused on complex linguistic operations. All standard services were implemented except the security services which are supported by our operating system.

3 P2P Networks

The term Peer-to-Peer (P2P) refers to a class of systems (hardware and software) which employ distributed resources to perform a function in a decentralized manner. Each node of such a system has the same responsibility.

Basic goals are decentralization, immediate connectivity, reduced cost of ownership and anonymity. P2P systems are defined in [7] as "*applications that take advantages of resources (storage, cycles, content, human presence) available at the edges of the Internet*". The P2P architecture can help to reduce storage system costs and allow cost sharing by using the existing infrastructure and resources of several sites. Considering these factors, the P2P systems could be very useful in designing the future generation of distributed file systems.

3.1 Design Issues in P2P Networks

Peer-to-Peer systems have basic properties that separate them from conventional distributed systems. We describe in this section different design issues and their effect in system performance [8].

Symmetry: Symmetry comes among the roles of the participant nodes. We assume that all nodes have the same characteristics.

Decentralization: P2P systems are decentralized by nature, and they have mechanisms for distributed storage, processing, information sharing. In this way scalability, resilience to faults and availability are increased.

Operations with Volunteer Participants: An important design issue is that the participants can neither be expected nor enforced. They haven't a manager or a centralized authority and can be inserted or removed from the system at any time.

Fast Resource Location: One of the most important P2P design is the method used for resource location. Because resources are distributed in diverse peers, an efficient mechanism for object location becomes the deciding factor in the performance of such system.

Load Balancing: Load balancing is very important in one robust P2P system. The system must have optimal distribution of resources based on capability and availability of node resources.

Anonymity: With scope to prevent censorship we need to have anonymity in our system. Our network must to assure anonymity for producer and for consumer of information.

Scalability: The number of peers from network should be any value (hundreds, thousands or millions), and that not affect the network behavior. Unfortunately actual systems are not scalable over few hundreds or thousands of nodes.

Persistence of Information: Methods from our system should be able to provide persistent data to consumers, the data stored in the system is safe, protected against destruction, and highly available in a transparent manner.

Security: Security from attacks and system failure are design goals for every system. The system should be able to assure data security, and this can be achieved with encryption, different coding schemes, etc.

4 Peer to Peer GRID Concepts

P2P systems are divided into two categories in [6]:

- *File sharing utilities* how we already see in FreeNet, which can be characterized as providing a global namespace and file caching and a directory service for sharing files in a wide-area distributed environment. In the most interesting cases, the resources and all services are completely distributed. Each user client program, which can access local files, is also a data server for files local to that host.
- *CPU cycle sharing* of unused user resources usually managed by a central system, which distributes work in small pieces to contributing clients. Examples of this include Seti-athome, Entropia [11] and Parabon [12].

Both of these cases are interesting distributed system architectures. From the perspective of Grid computing there are several compelling features to these systems. First, the deployment model of P2P systems is purely user-space based. They require no system administrator.

Security, when it exists in these systems is based on users deciding how much of their resource they wish to make public to the collective or to the central resource manager. Also P2P systems are designed to be very dynamic, with peers coming and going from the collective constantly as users machines go on and off the network. P2P systems are also doing a good job of bypassing firewalls. This stands in contrast to large-scale scientific Grid systems, which manage large expensive resources and must be constantly maintained and managed by the system administration staff.

5 The System

5.1 CAN

The system P2P architecture is based on CAN model [15]. In order to implement a P2P architecture that respects all these issues, we use

the Content Addressable Network (CAN) presented in [9]. The basic operations performed on a CAN are the insertion, lookup and deletion of nodes. Our implementation is based on [10] and it provides a completely distributed system, scalable and fault-tolerant.

CAN Construction

Our design is over a virtual d-dimensional Cartesian coordinate space. This space is completely logical and we haven't any relation between it and any physical system. In this d-dimensional coordinate space, two nodes are neighbors if their coordinates are the same along d-1 dimensions and are different over one dimension. In this way, if all zones from this space are approximate the same every node has 2d neighbors. To allow the CAN to grow, a new node that joins to this space must to receive its own portion of the coordinate space.

CAN Insertion

The process has three steps:

- The new node must find a node that is already in CAN (source node).
- After that, it knows CAN dimensions and it generates a d-point in this space (this point is in a node zone - destination node). We route from source node to destination node following the straight-line path through the Cartesian space. The destination node then splits its zone in half and assigns one half to the new node.
- In the last step, the neighbors of the split zone must be notified about new node from CAN.

CAN Deletion

When nodes leave from our CAN, we need to ensure that the remaining neighbors take their zones. If the zone of a neighbor can be merged with the node's zone to produce a valid single zone, then this is done.

If not, then the zone is split in zones accordingly with neighborhoods structure. In some cases it is possible like this zone to remain non-allocate, but with first occasion it is used.

5.2 Transfer Protocol

The transfer protocol we used in our approach is based on CIFS (Common Internet File System) [13], the Microsoft version of SMB (Server Message Block) [14]. The main scope of using this transfer protocol is to manage the download and upload of files between nodes from our P2P network. Using our protocol, advantages come from possibility to retry in case of failures and in possibility to use bandwidth partially in order to by-pass network overloaded.

SMBs have a specific format that is very similar for both requests and responses. After connecting at the network level, the client is ready to request services from the server. However, the client and server must first identify which protocol variant they each understand.

The client negotiates the protocol which will be further used in its communication with the server. Once a protocol has been established, the client can proceed to login to the server, if required. One of the most important aspects of the response is the UID of the logged on user. This UID must be submitted with all subsequent SMBs on that connection to the server. The client sends a SMB specifying the network name of the share that they wish to connect to, and if all is proper, the server responds with a TID that the client will use in all future SMBs relating to that share.

Having connected to a tree, the client can now open a file with an open SMB, followed by reading it with read SMBs, writing it with write SMBs, and closing it with close SMBs.

Download and Upload files

Our CIFS implementation respects the following design issues:

- Possibility to split and transfer files in blocks;

- Possibility to retry and resume when connection is lost, in case of power failures;
- Upload/Download files to/from one computer which need user/ password login;
- Possibility to use the bandwidth partially, in order not to overload the network.

Download a file

In order to download a file, the following steps are performed in our implementation:

1. **Login**: First the dialect of CIFS to use is negotiating and the Protocol to use is obtained. Secondly, the connection was set up to the server using credentials and set the UID assigned by the server to the client.
2. **TreeConnect** - connect to the specified folder and set TID identifier
3. **Open File** - open file for read and set FID
4. **ReadFromFile** - read from remote file starting with position offset, length bytes and put the result in buffer

Upload a file

In order to upload a file, the first two steps described above are performed, followed by:

3'. **Open File** - create file or open file for write, append and set FID (flags should be FILE_OPEN_IF)

4'. **WriteToFile** - write to remote file starting with position offset, length bytes from buffer

5.3 System Architecture

The system presented below consists of more core modules (CMs) and databases of linguistic resources. In order to solve the task from RTE competition we must connect to a computer from this computational Grid in order to initiate the problems solving. The system main components and transfer protocols were implemented using Microsoft platform and the C# language.

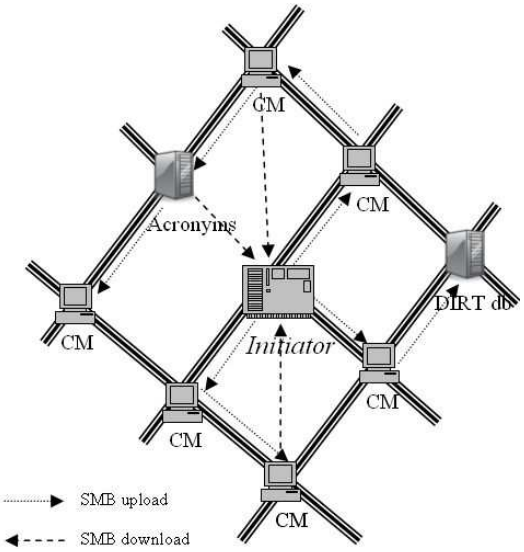

Figure 1. *P2P Network*

Between CMs, the upload and download operations are done using a special component based on the SMB protocol.

Any computer from this network can initiate the solving of the RTE task and it becomes the *Initiator*. First of all it checks its list with neighbors in order to know the number of computers which can be involved in the problem solving (the future CMs). After that it updates all these CMs with the last version of the TE module (this module identifies if we have textual entailment between text and hypothesis).

In parallel, all pairs are sent to the LingPipe and Minipar modules [16, 17], which send back the pairs on which the central module can run the TE module. All necessary files are automatically uploaded from the initiator to the other computers and eventually, in the end the partial results from the other computers are automatically downloaded to the initiator.

After that, the *Initiator* automatically splits the initial problem (consisting of 800 pairs) in sub-problems (a range between the number of the first and last pair) according to the number of its neighbors and using a dynamic quota. At first, this quota has an initial value, but in time it is decreased and eventually becomes the default minimal value. The main goal of using this quota is to send to any neighbor a number of problems according to its computational power and how busy it is in the running moment. In order to accomplish this, each computer is given an initial number of problems for solving and when it finishes, it receives automatically another quota according to the remaining number of problems.

We run in a network with different system configurations of computers (dual core - running two processes and normal - running one process) and with different degrees of business in the execution moment. For that we run Disk Defragmenter on some computers (those marked with *) to see how the initiator distributes the sub-problems to its neighbors.

Table 1. Number of Problems Solved by Computers

quota	**100**	**100**	**100**	**20**	**20**	**20**
Dual1_1	165	163	176	171	178	209
Dual1_2	197	136*	179	145*	159*	221
Dual2_1	175	175	155*	185	138*	108*
Dual2_2	152	183	163	180	129*	130*
Normal	111	143	127	119	196	132

The table above demonstrates the correctness of our supposition regarding the quota mechanism. In all cases presented, the processor solves fewer problems. After splitting the initial problem in sub-

problems, the "Client Module" from the initiator creates threads for each sub-problem. It also creates for every neighbor a sub-client in order to communicate with server components from neighbor and solve the sub-problem (see Figure 2).

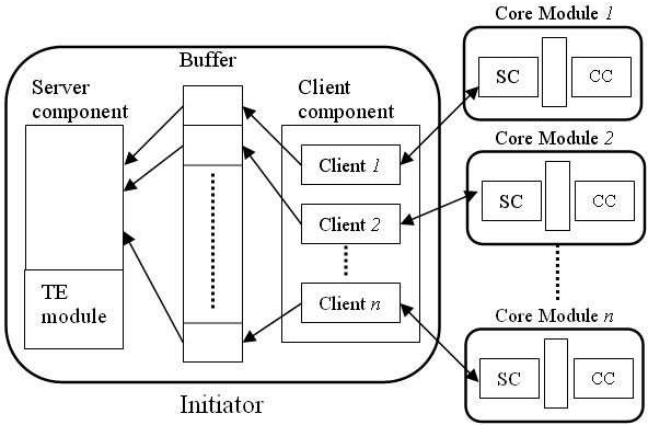

Figure 2. *Client Server architecture*

This sub-client is responsible for sending sub-problem, for receiving the results and saving them to a file with partial results, and informing the initiator server when the solving of sub-problem is finished. In order to inform the server about ending of solving process the client module uses the buffer zone in order to inform the server about that. The client uses the buffer like a "producer" which puts here information about ending of processes, and the server uses the buffer like a "consumer" [18]. The server takes the information and sees how the sub-problem is solved on the corresponding computer and starts the solving of a new sub-problem when the previous one is finished.

5.4 Results

The idea to use a computational Grid came after the first run of the first system build for RTE3 [19]. This first run had no optimization and

took around 26 hours on one computer. In order to observe the system changes faster, we improved this system and during the competition period we used a system which took around 8 hours for a full run.

This initial system was installed on four computers and after manual splitting of problems in sub-problems a single run took only two hours. After all system finished its runs we manually built the final solution from the four partial solutions.

Now, in the current solution the *Initiator* is responsable for splitting of problems in sub-problems, and for collecting of partial results in order to build the final result. Also, a common caching mechanism is used for the large databases of Dirt and WordNet in order to increase the computational speed. For the building of the caching resource, the system was run on one computer. After that, the computational Grid was configured on a P2P network with 5 computers (3 normal and 2 with dual core processor) and a run of the system now takes 6.7 seconds (see Table 2).

Table 2. Details on the Duration of Runs

No	Run details	Duration
1	One computer without caching mechanism	5:28:45
2	One computer with caching mechanism, with empty cache at start	2:03:13
3	One computer with full cache at start	0:00:41
4	5 computers with 7 processes	0:00:06.7

Another big problem was synchronizing the termination of processes. When we manually split the initial problem in sub-problems, if another process (like Windows update or a scanning computer after viruses) starts on some computer, then this computer works slowly. In this case, we have up to 30 minutes delay due to the fact that we have to wait for it to finish solving its sub-problems in order to build the final solution.

Now, using the quota mechanism for synchronizing the ending of all processes, the difference between the time when the first computer from the network finishes its work, and the last computer is around 0.26

seconds. Also, the powerful computers from the network have time to finish more problems while the slow ones solve less.

6 Conclusions and Future Work

In this paper we have shown how to build the system for RTE competition. This system is based on a P2P arhitecture in order to observe the system changes faster. This approach seems as most appropriate given the complexity of the task and the fact that the impact of any change brought to the system must be quantified quickly (the competition has a duration of 3 days). To our knowledge, there were no similar efforts to optimize RTE systems from the computational speed point of view (presently, there are systems whose run takes days to complete).

One important further development of the system will be a web-service allowing users to run in a GRID environment NLP applications. For that we will use for the middleware part of the GRID, the Globus Toolkit [20], a software "work-in-progress" which is being developed by the Globus Alliance [21]. The toolkit provides a set of software tools to implement the basic services and capabilities required to construct a computational GRID, such as security, resource location, resource management, and communications. Our scope is to offer basic linguistic GRID services, and after that, based on these services we will build complex services. Like basic services we already implement in Java GRID services for lemmatization, POS, tokenizing, name entity recognition, WordNet (in English and in Romanian). As complex services we will be focused on special services necessary in Question Answering and in Textual Entailment.

Acknowledgement

The author thanks the members of the NLP group in Iasi for their help and support at different stages of the system development.

The work on this project is partially financed by the CEEX GRAI (Grid Computing and Artificial Intelligence) project number 94 and by

Siemens VDO Iasi.

References

[1] RTE competition: http://www.pascal-network.org/Challenges/RTE3.

[2] G. C. Fox, D. Gannon, *Computational Grids*, IEEE Comput Sci Eng. Vol 3, No. 4, pp. 74–77, 2001.

[3] D. Gannon, and A. Grimshaw, *Object-Based Approaches*, The Grid: Blueprint for a New Computing Infrastructure, Ian Foster and Carl Kesselman (Eds.), pp. 205–236, Morgan–Kaufman, 1998.

[4] Globus MDS. http://www.globus.org/toolkit/mds.

[5] Grid Computing: http://en.wikipedia.org/wiki/Grid_computing.

[6] D. Gannon, R. Bramley, G. Fox, S. Smallen, A. Rossi, R. Ananthakrishnan, F. Bertrand, K. Chiu, M. Farrellee, M. Govindaraju, S. Krishnan, L. Ramakrishnan, Y. Simmhan, A. Slominski, Y. Ma, C. Olariu, N. Rey–Cenvaz, *Programming the Grid: Distributed Software Components, P2P and Grid Web Services for Scientific Applications*, In Cluster Computing journal, Volume 5, Number 3, Pp. 325–336. 2002

[7] S. Androutsellis-Theotokis and D. Spinellis, *A survey of peer-to-peer files sharing technologies*, White paper, Electronic Trading Research Unit (ELTRUN), Athens, University of Economics and Business, 2002.

[8] H. Ragib, A. Zahid, *A survey of peer–to–peer storage techniques for distributed file system*, National Center for Supercomputing Applications Department of Computer Science, April 2005.

[9] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, *A scalable content addressable network*, TR-00-010, 2000.

[10] A. Iftene, C. Croitoru, *Graph Coloring using Peer–to–Peer Networks*, In Proceedings, of 5th International Conference RoEduNet IEEE. Pp. 181–185. Sibiu, Romnia. 1-3 June, 2006.

[11] Entropia Distributed Computing, see http://www.entropia.com.

[12] Parabon Computation, see http://www.parabon.com.

[13] CIFS or Public SMB Information on Common Internet File System: http://support.microsoft.com/kb/199072.

[14] Server Message Block: http://en.wikipedia.org/wiki/Server_Message_Block.

[15] A. Iftene, A., Balahur-Dobrescu, and D. Matei, D. *A Distributed Architecture System for Recognizing Textual Entailment.* In proceedings of 9th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing. Timisoara, Romnia. September 26–29, 2007.

[16] LingPipe: http://www.alias-i.com/lingpipe/.

[17] D. Lin, *Dependency-based Evaluation of MINIPAR*, In Workshop on the Evaluation of Parsing Systems, G ranada, Spain, May, 1998.

[18] G. R. Andrews. *Foundations of parallel and distributed programming.* Addison-Wesley Longman Publishing Co., Inc., 1999.

[19] A. Iftene, A. Balahur-Dobrescu. *Hypothesis Transformation and Semantic Variability Rules Used in Recognizing Textual Entailment.* In Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing, Pp. 125-130, Prague, June 2007.

[20] Globus Toolkit: http://www.globus.org/toolkit/

[21] Globus Alliance: http://www.globus.org/alliance/

Adrian Iftene

"Al. I. Cuza" University,
Faculty of Computer Science,
General Berthelot Street, No. 16,
Code 700483, Iasi, Romania
E–mail: *adiftene@info.uaic.ro*

Received December 2, 2007