A Fault Tolerant, Peer-To-Peer Replication Network

Radu Potop, Otto Iovanici
Department of Science and Letters
"Petru Maior" University of Tg. Mures
Tg. Mures, Romania
radu.potop@wooptoo.com, ottoiovanici@gmail.com

Genge Bela, Haller Piroska
Department of Electrical Engineering
"Petru Maior" University of Tg. Mures
Tg. Mures, Romania
bgenge@engineering.upm.ro, phaller@engineering.upm.ro

Abstract — **We propose a fault tolerant, peer-to-peer replication network for synchronizing files across multiple hosts. The proposed topology is constructed by applying existing technologies and tools to ensure that files are kept synchronized even after subsequent modifications. One of its main advantages lies in the fact that there is no central authority to coordinate the process, hosts are connected in a peer-to-peer fashion, thus avoiding a single point of failure. Our proposal is intended for use in networks of personal computers where a small number of hosts have to be synchronized.**

Keywords: replication, peer-to-peer, synchronization.

I. Introduction

A peer-to-peer network is a distributed network composed of participants that make a portion of their resources (e.g. processing power, disk storage or network bandwidth) available to other network participants, without the need for central coordination [1]. Peers are both suppliers and consumers of resources, in contrast to the traditional client-server model where only servers supply and clients consume.

Peer-to-peer networks are typically used for connecting nodes via largely ad–hoc connections. Sharing content files containing audio, video, data or anything in digital format is very common, and real time data, such as telephony traffic, can also be transmitted using peer-to-peer technology [2].

A replication network basically manages replicas of the same set of files across multiple computers. These can be of various kinds, from simple file sharing to mirroring content and providing it to users based on their geographical location (e.g. Content Delivery Networks), replicating content with the intend of backing-up or, replicating content for immediate availability of data across multiple computers.

In this paper we propose such a fault tolerant, peer-to-peer replication network for synchronizing files across multiple hosts. The proposed topology is constructed by applying existing technologies and tools to ensure that files are kept synchronized even after subsequent modifications. One of its main advantages lies in the fact that there is no central authority to coordinate the process, hosts are connected in a peer-to-peer fashion, thus avoiding a single point of failure. Our proposal is intended for use in network of personal computers where a small number of hosts have to be synchronized.

The paper is structured as follows. In Section II we present the requirements for constructing a fault tolerant peer-to-peer replication network and we provide a presentation of our proposal. We continue with Section III where we provide several usage scenarios, illustrating the applicability of our proposal. In Section IV we evaluate the performance of the proposed replication network and we conclude with Section V.

II. Requirements and Deployment

A. Network Requirements

Implementing a fault-tolerant, peer-to-peer (Fig. 1) replication network is not a trivial task if we just look at the number of existing proposal found in the literature [3], [4]. Peer-to-peer replication involves many aspects that should be considered.

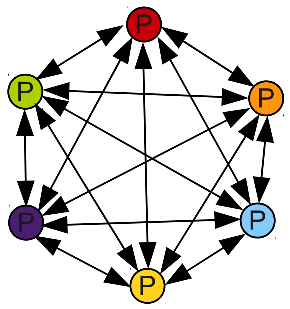

Fig. 1: Peer-to-peer Network

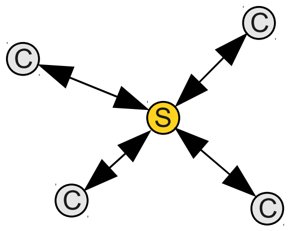

Fig. 2: Centralized Network

First of all, system administrators must decide if the architecture should involve a centralized (Fig. 2) approach. This way, the system defines several peers and one master node against which the synchronization is actually made. Such an architecture can have several disadvantages:

- There must be a central, node up and running that must handle all synchronization requests, which can be a single point of failure for the entire network;

- Each node must synchronize with the master, operation that involves serious bandwidth consumptions for larger files.

One of the main issues and challenges that administrators must deal with is the overall bandwidth consumption of the synchronization process. By using a large amount of bandwidth, the network can increase the delay of packages received by end users that can also lead to an unusable network during synchronization.

Another issue with which system administrators are confronted with is the security of the system and the overhead introduced by cryptographic operations. When setting up security requirements, administrators should not only look at the security requirements of data transmitted over the network, but they should also establish key management techniques, as these can be one of the main reasons of security failures.

One of the aspects that should also be taken into consideration is the use of existing, well-established technologies and tools for deploying such a network. With the large amount of solutions that can be adopted, system administrators should filter those solutions that do not involve using well-established tools. Such an approach can limit and possibly eliminate system failure due to programming errors.

To sum-up some of the major requirements that must be addressed by system administrators are the following:

- Adopt a decentralized architecture;
- Deploy only low bandwidth consumption algorithms;
- Ensure secure distribution of cryptographic keys and use only standardized security protocols for data transfers;
- Use only standardized and well-established software/tools to deploy the solution.

B. *Deployment*

1) Applications

In deploying the proposed network we have used several well-established software and tools such as: OpenSSH, Unison, Dcron, VirtualBox and Netem, all of these running on top of a Linux distribution. We are going to present each one of them briefly.

OpenSSH is a free/open-source Secure Shell (SSH) implementation [5] developed by the creators of the OpenBSD operating system. SSH was primarily designed to replace Telnet and other insecure remote shells that send sensitive data such as passwords in a plain text format. SSH ensures several security properties such as authentication, session key exchange, confidentiality and integrity and it is one of the most well-established security protocols because of its flexibility and reduced deployment issues. SSH is the network protocol that allows a secure data exchange between two peers in our proposal.

Unison is a file synchronization tool. It is used for synchronizing files between two directories, either on one computer, or between a computer and another storage device (e.g. another computer, or a removable disc) [6]. It runs on Unix-like operating systems (including Linux, Mac OS X, and Solaris), as well as on Windows. By using the *rsync* algorithm [7], Unison transfers only the fragments of a files that have changed, thus saving bandwidth. Unison can also detect *conflicts* where a file has been modified on two sources, and displays these to the user. Unison can deal with modifications to both versions of the directory structure, without the overhead of version control.

Dcron is a time-based job scheduler in Unix-like operating systems. The name Cron comes from the word chronograph (i.e. a time-piece). Cron enables users to schedule jobs (e.g. commands, shell scripts) to run automatically at a certain time or date. It is commonly used to automate system maintenance or administration, though its general purpose nature means that it can be used for other purposes, such as connecting to the Internet and downloading email. Cron is driven by *crontab*, a configuration file that specifies shell commands to run periodically on a given schedule.

In our proposal cron is used to periodically synchronize data across computers. Periodical synchronization makes on-demand synchronization faster, since a part of the data is already transferred.

VirtualBox is a virtualization software that provides a software-based emulation of the x86 architecture [8]. Using virtualization it runs a *guest* (or virtual) operating system as a process on the *host* (or physical) operating system. There are a lot of supported operating systems for both the guest and the host, ranging from Windows, to Linux distributions, and to Solaris.

In the case of our proposal, the virtualization software was used to run a number of guest Linux systems, that are interconnected by a network. These systems will exchange data between them in a peer-to-peer fashion, forming our replication network. Therefore, we could say that the replication network is sandboxed in a virtual environment.

Netem is a Linux kernel module. Netem provides network emulation functionality for testing protocols by emulating wide area networks [9]. The current version emulates variable delay, loss, duplication and re-ordering. Netem is controlled by the command line tool *tc* which is part of the iproute2 package of tools.

2) Scripts

A few Bash scripts accompany the applications and glue them together, making the replication network functional. Bash is the shell for the GNU operating system. It can be run on most Unix-like operating systems. It is the default shell on most systems built on top of the Linux kernel as well as on Mac OS X.

The main script reads a list of IP addressed from a file. These are the addresses of the computers that will be synchronized. For each IP address, Unison is ran and data from the local computer is synchronized with data from the remote computer, in a two-way fashion. Using this technique we will have a fully connected network that grants as soon as possible data availability on all the computers.

Secondary scripts handle tasks such as limiting the network throughtput and adding packet loss on each machine's startup, to simulate a wide area network. This is done using the tc tools.

Lastly, there are test scripts that generate files with fixed length and random content using Linux's built-in PRNG [10]. These files serve for the experiments done in chapter IV.

The sources of the scripts can be found at: http://bitbucket.org/wooptoo/p2prn/ as a mercurial repository.

III. Usage Scenarios

In this section we present several scenarios in which the proposed topology can be used to ensure file synchronization across multiple peers. The proposed network is designed to synchronize a folder on multiple computers belonging to the same user but not all at once. The files will become redundant, meaning that they will be present on every peer that is used.

A. *The first scenario*

In this first scenario we consider the situation where there is a user that owns a laptop computer (i.e. C1), a home desktop computer (i.e. C2) and a computer at his workplace (C3). Each computer will represent a peer in our system. The goal of this implementation is to provide the user a synchronized set of files (i.e. F1, F2, F3, F4) on any of the hosts he is working on (i.e. on C1, C2, C3). We consider that the user will work on the files from one computer at a time.

The initial setting is the following:

- C1 has F1, F2, F3;
- C2 has F4;
- C3 has no user files.

After the synchronization process is ended, F1, F2, F3 and F4 are available on each host C1, C2, C3.

In the first step, as shown in Fig. 3, the user is working on his home hosts (i.e. C1, C2) where he's using F4 on his desktop computer (i.e. C2). The laptop computer C1 is powered ON when the Cron job will execute the Bash script and the folder synchronization will start. When the job is finished, C1 and C2 will contain the same F1, F2, F3, F4 files. The user then powers down C2 and goes to work.

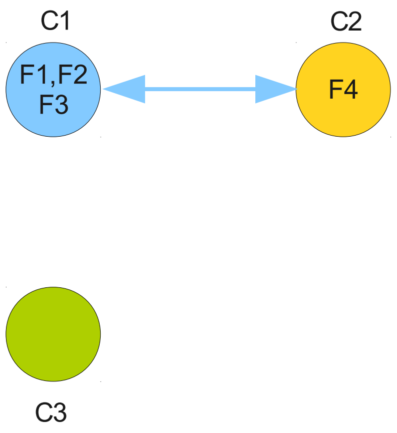

Fig. 3: Synchronization at step 1 – scenario 1

In the next step, illustrated in Fig. 4, the user arrives at the workplace where the synchronization between C1 and C3 will result in C3 having all 4 files (i.e. F1, F2, F3, F4). The user then creates a presentation (i.e. file F5) on C3. Through the synchronization process, F5 will be transferred on his laptop computer C1, that he is going to use at a client meeting.

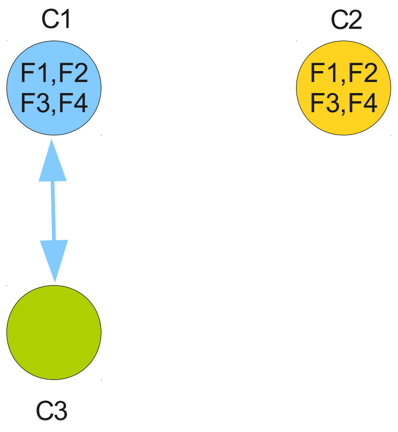

Fig. 4: Synchronization at step 2 – scenario 1

Once the user arrives home, after C1 and C2 are synchronized, he can edit file F5 in the comfort provided by the desktop computer C2, also shown in Fig. 5.

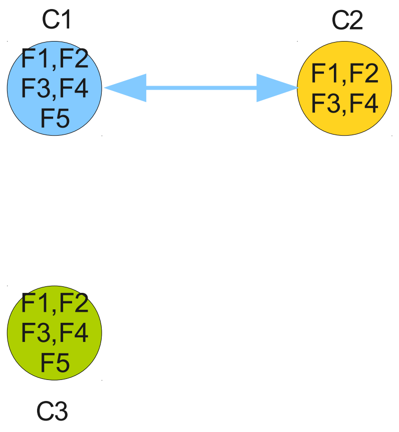

Fig. 5: Synchronization at step 3 – scenario 1

B. *The Second Scenario*

In the second scenario, we consider a situation where the user has four hosts to be synchronized. The hosts are: C1, C2, C3 and C4 and we consider the set of files F1, F2, F3, F4. The goal now is to synchronize files on all four machines over the network. Initially, files are distributed at hosts such that each host stores a different file than other hosts. At the end of the synchronization process, all four files are stored on every host.

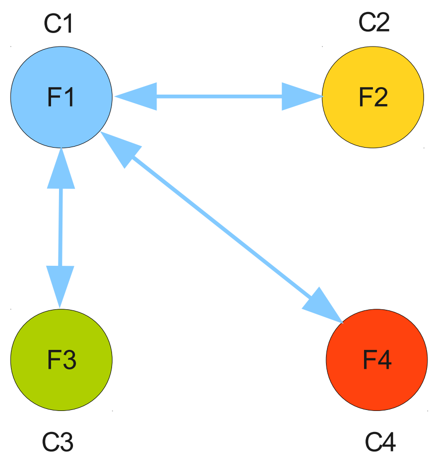

Fig. 6: Synchronization at step 1 – scenario 2

In the first step, illustrated in Fig. 6, host C1 is synchronized with hosts C2, C3 and C4 in this particular order. The result is the following:

- C1 – has F1, F2, F3, F4;
- C2 – has F1, F2;
- C3 – has F1, F2, F3;
- C4 – has F1, F2, F3;

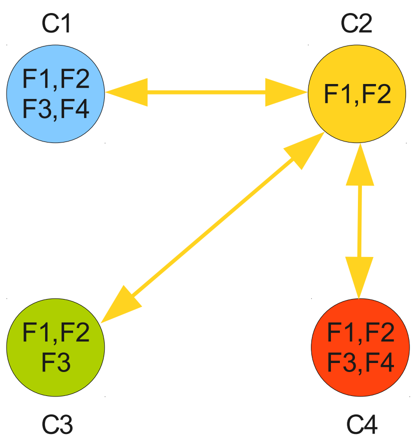

Fig. 7: Synchronization at step 2 – scenario 2

In the second step, illustrated in Fig. 7, host C2 has to synchronize with hosts C1, C3, C4 in this particular order. This will ensure that every host has the same file content once the synchronization process is finished.

IV. Experimental Results

A. *Preliminaries*

Before measuring the actual performance of our proposal, we have to limit the bandwidth of each peer to simulate a consumer-grade Internet connection. An ideal network setting would be similar to the one illustrated in Fig. 8. For such a setting the measured throughput varies around 30Mbits/s and there is almost no packet loss or network instability.

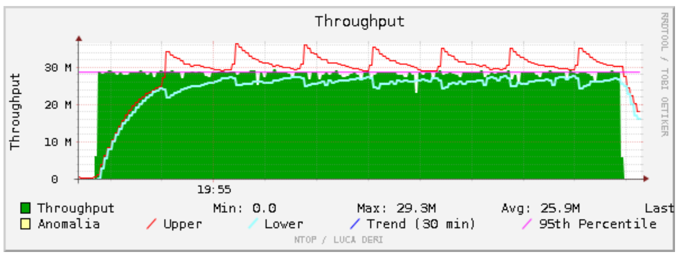

Fig. 8: Ideal network throughput

Using Netem we can simulate a more realistic network, where packets can be losed randomly and we can limit the overall throughput to 3Mbits/s, which is closer to consumer-grade connections. The result is shown in Fig. 9. For this case, we configured a delay of 10ms between packets, and a loss of 3%. The throughput has been altered significantly as can be seen from Fig. 9.

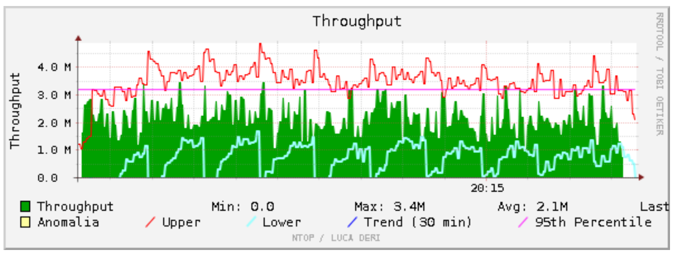

Fig. 9: A realistic network throughput

B. *First Scenario*

In the first scenario we used four hosts and we identified five different cases, described in the remaining of this sub-section. The results are shown in Fig. 10.

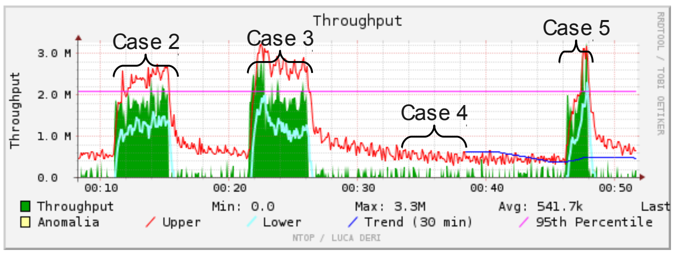

Fig. 10: First scenario – up to 4 hosts

1) *Case 1*

We use only two hosts where we send file F1 (10MiB) from host C1 to host C2. This takes 56 seconds over the network conditions mentioned above and illustrated in Fig. 7.

2) *Case 2*

To the two host mentioned in the previous case we add two additional hosts C3 and C4, and we also add file F2 (10MiB) to host C1. We start synchronizing from host C2. Host C2 already has file F1, so it only fetches file F2 from host C1. Then, files F1 and F2 are transferred to both host C3 and C4. This sums to a traffic of 50MiB that takes 4 minutes and 17 seconds to complete.

3) *Case 3*

In this case we add file F3 (10MiB) to host C1, file F4 (10MiB) to host C3, and we start synchronizing from host C3. File F3 is transferred from host C1 to host C3, and file F4 is transferred from host C3 to host C1. After this, both F3 and F4 files are transferred from host C3 to hosts C2 and C4, totaling 60MiB of traffic. This takes 4 minutes 32 seconds to complete.

4) *Case 4*

At this point every host (C1, C2, C3, C4) will have all the files (i.e. F1, F2, F3 and F4). In the fourth case we synchronize host C4 with the other three hosts. This will result in almost no traffic, since the files already exist on all hosts. This takes 25 seconds. Up until case 4 the total traffic was 160MiB, that required about 10 minutes to complete.

5) *Case 5*

In this last case for the first test, we append 5MiB to file F1 from host C1, and we also append 5MiB to file F4 from host C4. File F1 will be sent from host C1 to hosts C2, C3 and C4, and file F4 will be sent from host C4 to host C1. We can see that the fifth case took a lot less time and traffic to complete. This happens because the delta algorithm was used. Even though we appended (i.e. we modified) 5MiB to file F1, the synchronization process did not transfer the entire file across the network, it transferred only differences. This is because hosts C2, C3 and C4 already had the initial 10MiB of the file. The total transferred volume in case 5 is 20MiB, completed in 1 minute and 45 seconds.

From this first test we can already see that transferring large amounts of data between the hosts can be inefficient, not because of the system's architecture, but mostly because of the network (i.e. bandwidth) limitations. However, our system seems more efficient for sending small files or (small) modifications to existing files.

C. *Second Scenario*

In the second scenario we used six hosts, and 10 files (denoted by F1, F2, F3, …, F10). For this scenario, we identified 3 different cases that are detailed next.

1) *Case 1*

The files, from F1 to F10 are only on host C1 and need to be transferred to the other hosts. The total traffic required for synchronizing all 10 hosts is calculated as follows. Each file has a size of 10MiB, with each set of 10 files that must be transferred to five hosts. This results in 500MiB of traffic that is completed in 32 minutes and 47 seconds.

This case takes quite a lot of time, making the transfer inconvenient in some cases, such as an urgent need of files on remote hosts.

2) *Case 2*

In this case we append 1MiB to each file according to the following setting:

- C2 – files F1, F2;
- C3 – files F3, F4;
- C4 – files F5, F6;
- C5 – files F7, F8;
- C6 – files F9, F10.

The synchronization is started from host C1. For this case the synchronization is completed in 3 minutes and 12 seconds with 20MiB transferred.

3) *Case 3*

All hosts will be synchronized from host C5. Since each host already has all the files, no transfer will take place – Unison simply checks that all files are the same. This takes 1 minute and 33 seconds to complete.

If we compare case 3 from the second scenario (1m33s) and case 4 from the first scenario (25s), we can see that time is also wasted on establishing connections between hosts. Both cases do not make any transfer, they just check that files exist.

One of the advantages of our proposal is that it is able to propagate small changes of files in a reasonable amount of time. Since the system is intended for home users, *Case 2* from the second scenario is possibly the most realistic one, since users are unlikely to change a large amount of files over a short time period.

D. *Third Scenario*

In the third scenario we define only one case. We consider six hosts, with host C1 having 20 files, each of 500KiB, and the other hosts having no files. So 10MiB are transferred from host C1 to each other hosts, resulting a total of 50MiB of traffic, completing in 3 minutes and 59 seconds. The conclusion from this last scenario is that the number of files does not affect the time of the transfer. We can also conclude that for this scenario, the time needed to synchronize many smaller files is mostly equal to the time needed to synchronize a small number of larger files.

E. *Experiments Conclusions*

As a conclusion of the experiments we can say that the presented system is ideal for transferring a small volume of data at a time (about 50MiB) for the transfer to take place in a timely manner (under 4 minutes). These results can be confirmed from the first scenario – case 5 or the third scenario. These files could be mostly office documents that need to be consistent across multiple peers, or small batches of data that synchronize periodically.

Larger amounts of data can be transferred but with a significant increase in synchronization time. This happens mostly because of bandwidth limitations and packet loss (simulated with Netem), as can be seen from the second scenario – case 1. To overcome this issue we periodically synchronize files between peers by running the script from Cron. In these circumstances when an express synchronization is requested by the user it will take place as fast as possible.

By comparing case 3 from the second scenario and case 4 from the first scenario we can see that adding more peers to the network does increase synchronization time. Even though most of the synchronization time is spent on transferring files, some overhead exists.

V. Conclusions

We proposed a fault tolerant peer-to-peer replication network that enables an efficient file synchronization over multiple host. Our proposal does not depend on a host that can become a single point of failure, but instead uses a completely decentralized architecture where each host is synchronized against all other hosts in the network.

The network is deployed using only well-established, standard technologies such as SSH, Unison or Cron. This does not only ensure a rapid deployment, but provides a bug-free and a stable system from the very beginning.

Our proposal can provide an easy to use method for synchronizing a "home" directory. It is scalable to a reasonable number of hosts a user may have or want to work with, while most overhead is eliminated because the synchronization is made periodically due to the Cron daemon.

The security of the proposal is ensured by using the standard SSH security protocol that ensures a session key exchange, host authentication, data confidentiality, and integrity. In case a host is compromised, its public key can be simply removed from the list of trusted keys, thus ensuring that no synchronization requests are accepted from this particular host.

References

[1] Rüdiger Schollmeier, A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications, Proceedings of the First International Conference on Peer-to-Peer Computing, IEEE (2002).

[2] Peer-to-peer, http://en.wikipedia.org/wiki/Peer-to-peer

[3] Dan Teodosiu, Nikolaj Bjørner, Yuri Gurevich, Mark Manasse, Joe Porkka, Optimizing File Replication over Limited-Bandwidth Networks using Remote Differential Compression, Technical Report MSR-TR-2006-157, Microsoft Research, 2006.

[4] Tao Wu and David Starobinski, A Comparative Analysis of Server Selection in Content Replication Networks, IEEE/ACM Transactions on Networking, Vol. 16, Issue 6, pp. 1461 – 1474, 2008.

[5] RFC 4251, http://www.ietf.org/rfc/rfc4251.txt

[6] Unison manual, http://www.cis.upenn.edu/~bcpierce/unison/

[7] The rsync algorithm, http://samba.anu.edu.au/rsync/tech_report/node2.html

[8] The VirtualBox architecture, http://www.virtualbox.org/wiki/VirtualBox_architecture

[9] Netem – Emulating wide area network delays, http://www.linuxfoundation.org/collaborate/workgroups/networking/netem

[10] Pseudorandom number generator, http://en.wikipedia.org/wiki/Pseudorandom_number_generator