Design overview of the Multicasting Master-Slave Network Filesystem

Szymon Acedański
8th High School, Katowice, PL

January 18, 2004

Abstract

Network technology evolves quickly, but available bandwidths are still not sufficient for fast transfers of big files to a bunch of clients (for example populating operating systems' installation images). This paper describes Multicasting Master-Slave Network Filesystem (*mmsnfs*) – a new method of distributing entire filesystem among several client-machines simultaneously and efficiently. Moreover it provides mechanisms for modifying distibuted filesystem structure reliably and securing it against introducing inconsistencies. It is designed to be used in small networks of similarly-configured machines, for instance university labs or internet cafés.

1 Introduction

While physical networks bandwidths have increased dramaticaly, these are still not usable for setting up many machines via network simultaneously. Transferring large installation images to many clients using standard unicast protocols, such as FTP or NFS, takes long hours. Therefore the most commonly used techinque of setting up a group of machines is preparing several copies of distribution media (CDs, DVDs) and performing installations independently. Most modern operating systems support non-supervised (,,kick-start") installations to aid administrators in this task.

Another problem arises when a new application is to be installed on every node. If this application is relatively small, it can be easily installed by administrator by logging on every machine remotely and installing supplied packages. In practice such scenario is often realized using simple shell script. But if the application is big or needs some manual configuration, installation must be performed manually on every machine.

Using multicasting techniques I designed a network filesystem that is able to solve mentioned problems. After preparing one instance of completly configured filesystem (*mmsnfs* master), the task of populating it to all other machines (slaves) can be accomplished by simply preparing an empty disk partition, making it *mmsnfs*, and booting an *mmsnfs*-capable OS kernel. Such a mounted disk partition will behave like an NFS share, the system will boot instantly, without proper installation stage, and all transmitted files will be stored on the disk. A background daemon will download all other files, ending with individual complete on-disk *mmsnfs* filesystem. A reliable mechanism of updating all systems is also available.

In the rest of this paper, I discuss the goals of my work (Section 2), the principle of its operation (Section 3), current implementation state (Section 4) and some plans for my further research (Section 5).

2 Goals

The specific goals my filesystem is going to meet are:

1. Provide fast method of populating entire filesystem to numerous clients simultaneously over Ethernet network using multicasting techniques
2. Provide reliable method of updating files on every machine using *mmsnfs* on the local network
3. Ensure temporary fail-over for system files via network in case of damaged storage media
4. Optinally give client machines a very limited permissions to modify distributed files, even locally, to avoid unneeded configuration changes causing further global configuration changes leaving system inconsistent

The use of multicasting is motivated by rapidly increasing network bandwidth, and also by commonly used star-topology of networks. A quick observation reveals, that in such a networks the most common communication scheme is client-server one, with one dedicated machine being server and all the others being clients. Since client-to-client transactions are very rare, all the data passes the physical link between server machine and its neighbour switch/hub device, making this link eventually congested. Other physical links are usually either idle or moderately loaded. Heavy use of multicast packets for transferring data between server and clients does not introduce any additional overhead on the critical link, since unicast packets still would have passed it. But additional overhead is put on the other links. It won't hurt, because most of the data passing them sooner or later will pass the critical server link. The cost of filtering unwanted packets by clients is very low.

By using multicasting for transferring filesystems, this task can be accomplished efficiently even with 100BaseTX 100Mbit Ethernet, which is very widely used in small networks. Using gigabit Ethernet hardware will increase transfers considerably and such a link can be saturated only by fast enough harddisk. Nevertheless *mmsnfs* is not designed to be a scalable solution for large networks. It's intended to be a flexible solution for small ones.

3 Principle

3.1 Transport and caching

My filesystem uses UDP protocol. As it's not reliable, simple timeout-retransmit mechanism is provided for ensuring delivery. Every request sent to server is to be acknowledged. UDP datagrams are either broadcast on the local network [4, 5] or multicast [3] on an administratively-assigned multicast address. Using full multicasting, *mmsnfs* can also span multiple subnets interconnected with multicast routers. All datagrams sent by clients are called *requests* and they are unicast directly to server (if its IP is known, they're multicast otherwise). *Responses* are always multicast, but they have unique receiver ID attached, which is then used for matching requests and replies. Upon receiving a response packet by a client, two scenarios can take place:

1. Receiver ID of the packet matches client's ID, so this is a reply to previously originated request, so a started operation may continue its work.

2. Packet is not addressed directly to the client, but since it contains some information about filesystem state (for example, updated file information), received data can be cached on disk, so it won't be needed to request it separately introducing unwanted traffic.

In fact both scenarios can be handled similarly. Finally *mmsnfs* caches all the information on the disk, using mechanisms based on existing distributed filesystems like Coda [2] or Andrew [6]. After transferring entire filesystem, *mmsnfs* enters silent mode and listens only for updates sent by the server.

3.2 Master nodes and slave nodes

To accomplish goals No. 2 and 4 I divided all the files and directories in a filesystem in *master nodes* and *slave nodes*. Master ones can only be read by clients. This includes binaries, libraries and many configuration files (/bin, /sbin, /lib, most of /etc), generally – most of all the files. Master nodes are always kept in sync with server. Slave nodes are initially populated amongst clients, but they can be then modified locally on every machine independently. These changed files are stored on individual machines and are no more synced with server. These include machine-specific configuration files (for example storing MAC addresses of NICs etc.), most files in /var folder and perhaps /home (it's usually better to mount /home as an NFS share from central server).

In such an environment performing update of some software component on all the slaves can be done by updating the master copy on the server and all the changes are then populated automatically. It is possible only when all affected files are master files, otherwise some changes won't be populated leaving system possibly inconsistent. This scenario is much more convenient than doing the same update on every machine manually.

Also by disallowing changing master files, system is secured from unauthorized modification of system files.

3.3 On-demand transfer and failure resilience

While filesystem is not completely cached or replication is running in the background and access to some non-cached file is requested, the data is fetched from server, and computer continues normal operation. The same scenario takes place when the disk, where the cached data is stored, encounters read errors. But in this case, only master files and unchanged slave files can be fetched.

4 Implementation

Currently I'm implementing *mmsnfs* as a set of modules for the Linux kernel. My project consists of two parts: core VFS (Virtual File System) interface implementation with networking support, and a set of plugins for various caching schemes and operating modes (mostly for testing). Actually used technique is my own implementation of an on-disk filesystem based on simple *minix* filesystem. Currently the implementation is in pre-release stage. It needs further testing and fixing some issues. It's designed to work on multiprocessor systems, but I have no possibility to test it in such an environment. I'm going to publish source code on http://mmsnfs.sf.net to be freely available for everyone.

5 Further research

After solving most implementation problems and making the code stable enough, I'm going to perform some practical measurments of its performance, both disk and network. I would like to investigate the influence of network congestion and high packet loss rate on the operation of the timeout-retransmit mechanism. I'm going to measure real maintenance time and cost reduction while re-installing or upgrading software on *mmsnfs*-ized network.

Another approach would be modifying filesystem by adding full write support with robust and reliable populating mechanism and consistent locking semantics. This should result in a multiply backed up filesystem, with efficient recovery mechanisms.

6 Conclusions

Actual state-of-art in networking technology provides limited mechanisms of populating fileystens using multicasting. The alternative of using faster hardware links and widely-used unicast mechanisms implies higher cost of investment. Employing *mmsnfs* filesystem might give equally efficient method without introducing additinal expenses, as it is expected to work smoothly on top of common 100BaseTX Ethernet. Also maintenance costs should be reduced.

References

[1] *Assigned Numbers*. Internet Assigned Numbers Authority (http://www.iana.org).

[2] P. J. Braam *The Coda Distributed File System*. Linux Journal, June 1998.

[3] S. Deering *Host extensions for IP multicasting*. RFC 1112, 1989.

[4] J. Mogul *Broadcasting Internet datagrams*. RFC 919, 1984.

[5] J. Mogul *Broadcasting Internet datagrams in the presence of subnets*. RFC 922, 1984.

[6] M. Satyanarayanan *Integrating Security in a Large Distributed System*, Canergie Melon University, 1989.

[7] A. Silberschatz, P. B. Galvin *Operating system concepts*, Addison-Wesley, 1998, polish translation by WNT, 2002.