NFS behavior with multimedia clusters. Problems and Solutions

TERESA OLIVARES teresa@info-ab.uclm.es
FRANCISCO TORRES ftorres@info-ab.uclm.es
LUIS OROZCO lorozco@info-ab.uclm.es
FRANCISCO J. QUILES paco@info-ab.uclm.es
ANTONIO GARRIDO antonio@info-ab.uclm.es
Departamento de Informática, Universidad de Castilla-La Mancha, 02071 Albacete, España.

Abstract

We present a new NFS client, implemented after a complete evaluation of the Input/Output in Mirynet clusters working with NFSv3 and NFSv4. In modern day clusters, I/O is quickly emerging as the main bottleneck limiting performance. Our clusters support multimedia traffic and are used as an ideal platform for parallel processing. One of the main elements is the file system used to share the information. Therefore, we have evaluated the behavior of NFS, carrying out an exhaustive and detailed study, where major factors have been varied. The results of this analysis allow us to outline a real proposal of improvement in the NFS client. This improvement consists in endowing the client with an efficient prefetching technique through files to advance the data before they are requested by the application. This advancement of data also takes place while the client processing its information. This way avoids a server overloading which is one of the weak points in NFS, when workink with concurrent access to the information. With this new client we achieve a real improvement in the reading times thereby transforming NFS into a very competitive option for the current multimedia era.

Keywords: NFS, file systems, clustering, prefetching, multithread, multimedia information, MPEG-2

1. Introduction

Nowadays, the use of clusters in research centres is extremely widespread, as is their integration in business.

Scientific applications that imply modeling, simulations or analysis have seen satisfied their highest necessities of computation. Besides, multiple clusters of servers also exist which serve as the starter engine and operation for multimedia applications. Indeed, the evolution on these applications in recent years has been really spectacular. An ample range of applications oriented to the entertainment (TV, plays, etc.) has appeared, to give new services to the user through Internet (e-commerce, e-bank, e-museum, e-anything), video on demand, and so.

For this type of application, which involves a high computational cost, the option to parallel them has also become the natural form of processing. The parallel processing is very mature and consolidated and can now be operated commercially.

Our research work starts off studying the main problems of clusters of computers with multimedia applications. We have contrasted the two parts, architecture and applications in a “real” form, without simulations. We have started from a real platform, formed by a high speed cluster, and a flexible parallel multimedia application. This application injects the multimedia load into the cluster and demands particular requirements.

Numerous and interesting works on input/output of clusters exist. All have tried to make a novel contribution, for improving some weak point in the I/O. In our preliminary research, we have made an exhaustive study of all those works [16] to make a new hybrid method of data advancement for client-server architectures with NFS (Network File System), the most widely used file systems at the moment. This is not an algorithmic and theoretical exposition. It has been implemented in the Linux kernel. The final experiments confirm the improvements brought about by the developed implementation.

2. A Network File System Overview

NFS is a component of the ONC+, a distributed computation solution of SunSoft. It is a company solution that provides security, high performance and transparent access to the information in heterogeneous networks worldwide.

From its first appearance, NFS has continued to develop with the aim of satisfying the requirements of distributed files sharing. NFS qualifies computers of different architectures running different operating systems, to share file systems through the network. NFS can be implemented in different operating systems because it defines an abstract model of file system, instead of an architecture specification [2].

NFS was designed to give the users substantial benefits and transparent access, by serving file systems in global networks. Some of the more important principles of design are the transparent access, portability, fast recovery of failures, independence of the network protocol, performance, and security. NFS is constructed on the base of XDR (External Data Representation) protocol and ONC RPC (Open Network Computing Remote Procedure Call), MOUNT protocol and NLM (Network Lock Manager) protocol. NFS consists of a set of remote procedures that qualify the client to manipulate remote files and directories in the server as if they were local. Using the routines of the NFS protocol, the clients send requests about file operations to the servers, and the server respond trying to conduct the operation sending the results or the error values.

The first version, NFSv1, was never published. The second version, NFSv2, was implemented in a great variety of operating systems and hardware platforms. In 1993, a new revision of the protocol, NFSv3, was designed and tested for the first time. This version offers many improvements over version 2. The last version of NFS developed in the University of Michigan is NFSv4, already appears in the Linux distributions. This last version is a distributed file system that inherits characteristics of versions 2 and 3 of NFS protocol. Unlike previous versions, version 4 supports the traditional access to the files integrating *locking file* and the *mount protocol*. In addition, it supports a strong security (and its negotiation), *compound operations*, client cache and

internationalization. Of course, special attention has been paid to NFSv4 operating in Internet [1].

One of the most characteristics is the use of the cache by NFS. In addition to caching, prefetching is the following logical stage to increase the performance of the I/O systems. Various interesting works have appeared on prefetching in the last few years, oriented to multimedia, wireless or embedded systems, for example [3, 4, 5]. An entire series of works exists in which are designed, implemented, simulated and tested different useful algorithms from prefetching in different situations, with multiprocessor or clusters, for sequential or parallel access, for prediction of data block or complete files. The goal of all these is to improve the performance of the system anticipating user requests. Therefore, prefetching could be defined as a general technique that improves the I/O reading data from the cache before they are requested, overlapping I/O with computation.

As a final comment, multiple studies and comparisons of NFS with other file systems, like xFS, and AFS have been made with some interesting conclusions. One of clearest and most realistic [7] affirms that NFS is used for practical reasons (NFS is more mature and available, allowing suitable comparison and a good frame of reference) but has limitations with respect to the scalability.

2.1. NFSv3 versus NFSv4

NFS version 3 arises out of the need to improve different aspects of connectivity between systems of different characteristics. In 1993, a group of salesmen including IBM, Digitalis, SunSoft and others, considered the necessity to make necessary changes in version 2, the result of which was NFS v3, which offers substantial improvements on the previous version. Among these were: the improvement of reading and writing operations on the client side, server load reduction due to the increase of scalability and performance, improved support for systems that use ACLs, and support for very large files.

NFS version 4 is the latest version of NFS, while inheriting the characteristics of its predecessors, incorporates a fundamental difference: NFS version 4 supports access through Internet, providing agreed security systems with these necessities. The aim of NFS is to allow corporative companies to use Internet as part of their network to maintain distributed files in different sites. Protocol NFS is open source, thanks to the cooperation of different organisms, for example, Sun Microsystems, and IETF. NFS version 4 is fast, trustworthy, and transparent, allowing its implantation in Internet, and improving the performance, allows interoperability between diverse platforms and provides a protocol that can be extended for the growth in demand in a near future.

2.2. NFS in Linux

When somebody accesses a remote file, the kernel sends a RPC to the nfsd program of the remote node, in which are included the file descriptor, the name of the file, and the identifiers of user and group. With these identifiers access permissions to the remote host can be checked. The client code is integrated in the VFS layer. On the other hand, the server code runs totally in the user space and several execution threads for the same nfsd process can be sent.

The server receives and processes all the requests that arrive. In that processing it will also be the function to identify the client has making that request. Some important procedures are used: Lookup and Read [6].

In Figure 1 the client-server communication appears in a summarized form, when the client wants to read any three files f1, f2 and f3. In order to read a file, it is first necessary to create a filehandle for that file, and later to send the necessary reads until completing the amount of bytes that are to be read.

.

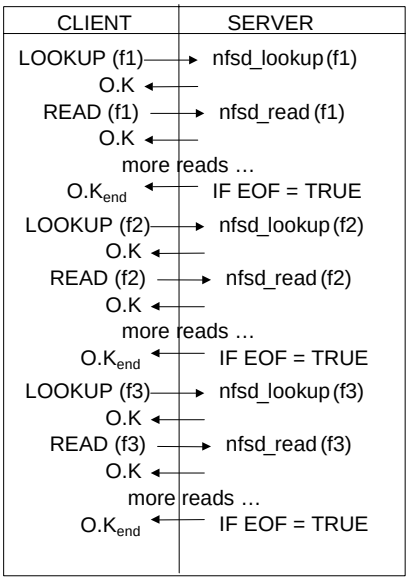

Figure 1. Client-server communication

2.3. Current Situation of NFS

At the present time there are many interesting projects working with NFS extending its specification of this one. Although it is nearly twenty years old, NFS continues to be the object of continuous updates and modifications by the international community, due to the necessity of new connections between new technologies and devices, with bandwidth, security and reliability constraints. These continuous changes make NFS, considered like a standard and at the moment the most used with respect to other network file systems.

The necessity to improve NFS for Linux is evident; this is reflected in the numerous works, projects and dissertations that focus on the network file system. In the University of Michigan, where the NFS project originates, several projects are currently being developed to improve NFS and to implant it in the industry. These projects can be seen in [19]. At the moment, projects related to NFSv4 are for example: NFS Version 4 Open Source Reference Implementation, NFS Client Performance, NFSv4 for ASCI, NFSv4 RDMA, Cluster Coherent NFSv4 and GridNFS. In the works of NFSv4 for ASCI, we can find *pNFS* [15] where the problem of limited bandwidth for the NFS server is discussed with the proposed solution to provide the parallelization of file services, by enhancing NFSv4 in a minor version. Some interesting related work is [11, 12].

3. Initial Positions

Initially we have performed two large groups of experiments: one with NFSv3 in a Myrinet cluster of 8 processors and 2 switches (NFSv3 testing) and another with NFSv4 in a Myrinet 2000 cluster of 16 processors and 4 switches (NFSv4 testing).

In the cluster software configuration, one processor is the NFS server, and the rest are the clients. The clients run a parallel MPEG-2 video encoder developed in HKUST (Hong-Kong University of Science and Technology) [10]. For this application, the clients have a real need for speed in the transmission, which will be satisfied with the use of these gigabits-per-second networks.

In our experiments, only the server has the video sequence that will be requested by the clients. The server will not participate in the code tasks. We will be interested, mainly, in measuring the latencies in the reading requests from the clients to the server. We will analyze the most interesting NFS parameters, the possible implications of the network itself and the different load patterns that could be presented with the multimedia traffic. Our aim here is to detect and expose the possible problems that this file system can present.

Our work begins with the parallelization of the multimedia application. First, we analyze this traffic type and its repercussion in multicomputing systems [8, 9]. Later, we use the clusters as ideal platforms of investigation of parallel computing. Then, we begin to develop and evaluate technical multiple load division among the different processors of a cluster [10]. Here appear the first problems for reading times among processors carrying out he same tasks. At this point we begin to study the importance of sharing the multimedia information using file systems allowing the data transfer through the network. We first used NFS in the ATM cluster of the video laboratory of the HKUST. Then used NFSv3 in the Myrinet cluster of our university (NFSv3 testing), and NFSv4 in the Myrinet 2000 cluster of the Albacete Research Institute of Informatics (NFSv4 testing).

4. The first Results

4.1. The NFSv3 testing

We have used a Myrinet cluster with two switches and eight processors to carry out all the experiments. In our tests, we have used the MPEG-2 *Composed* sequence, representative of different types of movement. This is a mixture of different sequences consisting of 400 pictures to 25 fps (16 s. of video sequence) in PAL CCIR 601 format (720×576) and 4:2:0. Each one of the processors, after a previous process of synchronization and initial parameter computation, executes a loop, of equal size to the number of pictures to encode, in the following steps: reading data, encoding the piece of data and writing the encoded data in the server. After this loop, it presents the reading and writing times for the total sequence.

In these experiments, each test was repeated ten times and the average of the ten executions was taken. During the tests, there was no any other operation being carried out on the machines and we used cold cache, that is to say, we cleaned the cache

between executions, both in the client and in the server, to obtain reliable data and avoid dependences with the previous executions. In Figure 2 we show schematically the experimental study carried out.

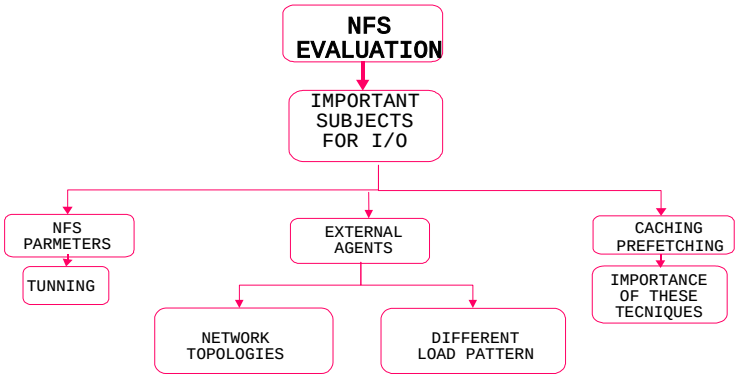

Figure 2.Experiment development

The first phase was a complete tuning and testing of NFS to check the most important performance problems. After this, we began to discover the improvement necessities in the NFS client. From then on, our objective was to plan a serious proposal of improvement in NFS. First, however, it was necessary to have a complete understanding of NFS and to see how NFS was affected by other changes in the cluster, such as the network topology and the different load pattern used with the multimedia application. The principal problems detected were due to some NFS factors like caching, and to the overflow in the server [13, 14, and 16].

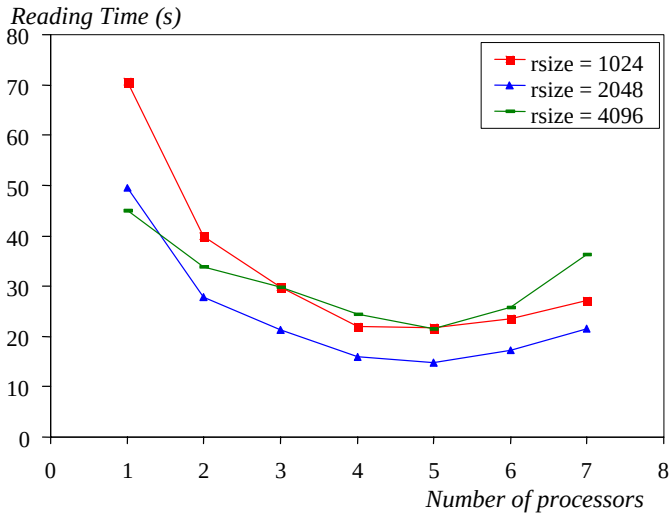

Figure 3. Reading Times for different number of processors

In the tests with variations in the NFS parameters we observed the following (see Figure 3):

- Important differences between the sequential case (1 processor) and the parallel cases. In the sequential case the best times are obtained with rsize = 4096 byte

- For parallel cases values obtained with 4096 bytes are the worst ones of all.
- For the parallel cases, from 5 processors, the reading times do not diminish with the increase in the number of processors, but quite the opposite. Therefore, problems of scalability appear. In addition, this happens for all the values of rsize, being more pronounced for 4096 bytes.

Something similar happens for the writing times taken for different values of wsize and a different number of clients. But, as it takes much less time in writing than in reading, the problem is not so serious.

We present, on the other hand, the times obtained by the processor (since the total average can hide what happens in fact), focusing on the reading time and rsize=4096 bytes. If we take the obtained reading times with 2 processors we find some interesting results. With 2 processors, both read the same amount of information and, nevertheless, very different reading times are obtained (25.14 seconds as opposed to 42.4 seconds). The question is why these differences exist? In order to answer this question we have graphically represented the reading times for each one of the 400 pictures, by each processor, including the moments of request and the times between requests [14]. We have seen that the processor with the greater reading time is the one that goes always ahead, that is to say, the one that asks the NFS server for the files first, and therefore the one that is going to make the accesses to disk.

In summary, the operation of NFS server is as follows: when receiving a file request, it verifies if it has the data stored in his cache, if so it sends data, if not it accesses to the disk, overturns the data in cache and sends them. Therefore, the processor asking the server for data, of a file requested previously is going to obtain them almost immediately, because the server has those data in cache.

With the other parallel cases something similar happens, there are very important differences between the obtained reading times. The amount of bytes read directly does not imply the time spent in reading. For the case of 7 processors the differences are still greater. The smaller reading time they are the 15.65 second and greater 48.3713 seconds. Peculiarly, both processors that offer these times read the same amount of information, but again, the one that request first is disadvantaged.

Also we have made a study of the effects of the timeo parameter. This parameter, defined as an integer, has a default value 0.7 seconds,(too great time for the present high speed networks). We have also measured the times that the requests take, that is to say, whatever time the server takes to respond to a request. We have seen that 95% of the requests are taken care of in less than 0.001 second, and that 99% in less than 0.01 seconds, why this parameter is unsuitable for more recent communications requirements. The smallest value we could use he would be 1 (0.1 seconds) which would not be useful. The case of 7 processors presents the greatest number of retransmits, and hence elimination would also imply a considerable reduction of time.

We have also carried the same tests for different network topologies, since we want to be sure that the problems are not related to the network used. But the behaviour is the same, indicating that the network is not the cause of the problem.

In addition, we tried other access patterns to the shared data. For that purpose we defined new representative forms of read/write of other multimedia applications, thus emulating in our platform, and with our parallel encoder, the I/O behavior of other applications.

- Pattern 1: overlapped reading of data segments - data writing
- Pattern 2: reading of complete files - data writing
- Pattern 3: reading of complete files - non-writing
- Pattern 4: overlapped reading of data segments - non writing
- Pattern 5: non overlapped reading of data segments - data writing
- Pattern 6: non overlapped reading of data segments - non-data writing

When representing the reading times with these patterns we emphasized the following (see Figure 4):

- The worse results are obtained with pattern 2 (all the processors read the complete files) where the scalability does not exist and the results are impossible.

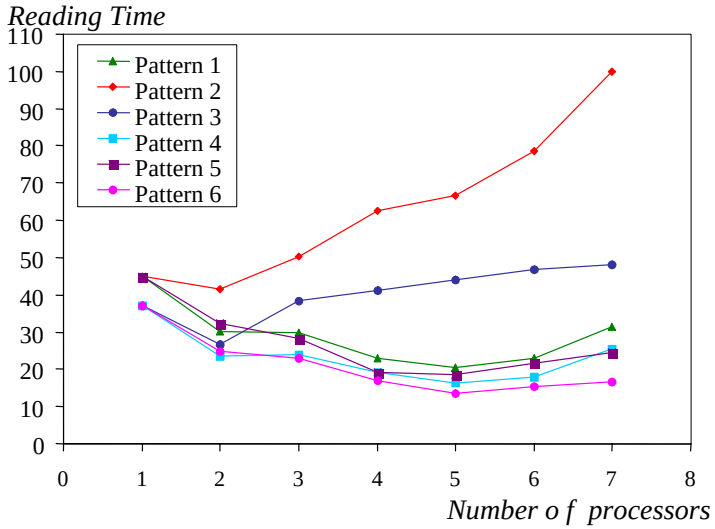

Figure 4. Reading Times for different I/O load patterns

- Next worse is pattern 3 (like the previous one, but without writing operations), a little better than pattern 2, but with the same tendency.
- With patterns 1 and 4 very similar times are obtained, thus demonstrating that the writing operations slightly affect the process.
- If we compare patterns 1 and 5 (reading of pieces without overlapping), we see that few differences exist; therefore, the scalability problems are not due to the overlapping reading. The same happens with patterns 4 and 6.

Therefore, a clear scalability problem exists in the NFS server, we think which can be improved if the reading operation is optimized (main cause of the problem). We have seen that the network does not cause any problem, and that if we try different access to the information, the most problematic case is that in which all the clients want access to read all the complete files and to write the results in the server.

4.2. The NFSv4 testing

NFS is the most used file system world-wide. Without a doubt, it is a key element for multimedia cluster work properly, as has already been mentioned. An example is that NFS4 already comes integrated in the Linux distributions, like the Fedora Core 2, which we have used in the experiments. The developers of this operating system believe that it is a file system of future. Our aim is to discover if, with this new version of NFS, the serious scalability problem detected in version 3, i.e saturation of the server, disappears.

When the server files are exported, different configuration options can be set, we will try to find the best combination to obtain the best reading times. With respect to the client, at the moment of mount process, different parameters can also be used. Of these, which we also try start with different possibilities. We begin with the tuning of parameters. An *initial configuration* in the server: rw, fsid=0, Insecure, no_subtree_check, sync, Wdelay, anonuid=65534, anongid=65534. In the clients: rw, hard, intr, rsize=4096, wsize=4096, proto=tcp, port=2049, noauto, dump=0, fsckorder=0.

For these tests we have used a new Myrinet 2000 cluster with 16 machines, where 15 of them are going to work as clients and one as a server. We can see the benefits of this platform in [17]. In Figure 5, we can see the first results of reading times for this initial configuration. In this Figure, observed that from six processors the reading time increases. This situation is very familiar for us, and it seems that the scalability problem detected in the server is repeated in this new version.

We made some parameters modifications to compare with the initial configuration:

1. In the server, we changed in the server the parameter sync by async
2. When the parameter sync is used, by defect wdelay is used, in the following modification that parameter is replaced by no_wdelay.
3. In the clients, we changed in the clients the way to establish the connection, replacing proto = TCP by proto = UDP.
4. We changed the size of package of communication between the clients and the server; by defect in NFSv4 the size of package is 4096 bytes. First, we changed it to 8192 bytes, and later to 16384 bytes and 32768 bytes
5. Finally, we modified the number of threads in the server from 8 to 16

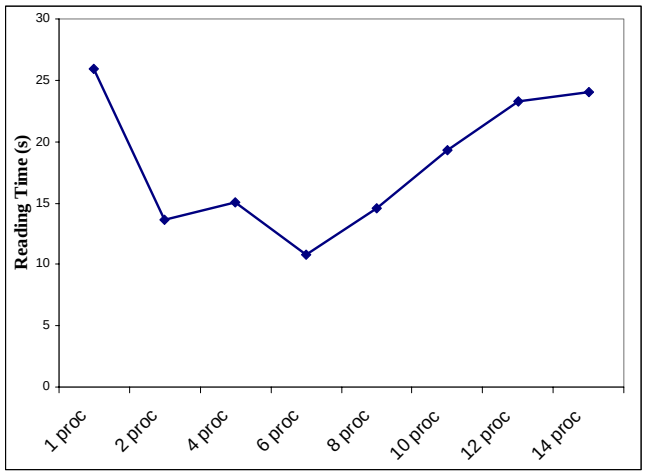

Figure 5.Reading Times of NFSv4 for Initial Configuration

The result of all these tests can be seen in Figure 6, from which we can conclude that the scalability problem continues in this new version of NFS.

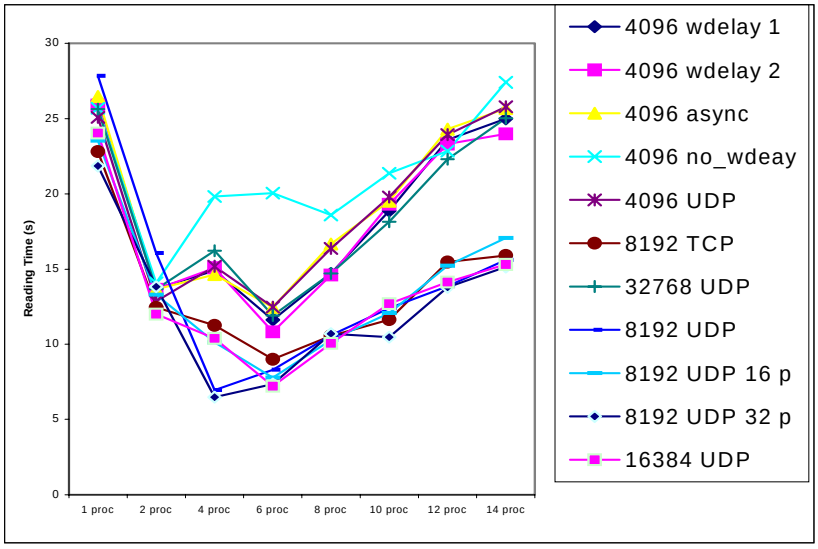

Figure 6. Different tests with NFSv4

5. The New multithread predictive NFS client

5.1. Possible improvements in NFS

At first, we considered improving the NFS server, modifying the form in which the reading requests are processed. But, a more detailed knowledge of the same, as well as the reading of some works on prefetching, led to us to think that it would be more convenient make the modifications in the client..

Next, we see the procedures that are called in the client before a reading request from an application. When the VFS receives a reading request from an application, first it decides if it is a request of a file located in the local disk or if it is a request of a file located in a remote machine. In the case of NFS, it is necessary to go request the information to a remote machine and this makes the following calls:

1. The call to the reading function: nfs_file_read. The VFS will call this function repeatedly, until completing the amount of information that the application is asking for. In each call the file is specified, the amount of data to read and the position from which it is going to begin to read.
2. Within this function the call to the generic reading function is made: do_generic_file_read, which is in charge of all the reading process. This function evaluates the cache of the client to assess if the data requested are already there and to avoid a request to the server.
3. Finally nfs_file_read verify if file has been arrived to end (we will initially use this mark to invoke our function of prefetching).

Our idea then is to create a thread process, just after detecting eof (end of file). This thread process creates an *access graph* in the first accesses to the files and later, will look for the following file in that graph to which it has been read at the time. This following file will be read in the same way as another conventional file. Hence, the generic function of reading will be called to complete the process. This graph is generated the first time that the files are accessed. Later, when one to access to them, we will use this graph to determine the following file.

The idea of prefetching through files in the kernel, can be seen in the Figure 7, in which one already assumes that the graph is created. Here appears the decision about the existence of the data in the cache.

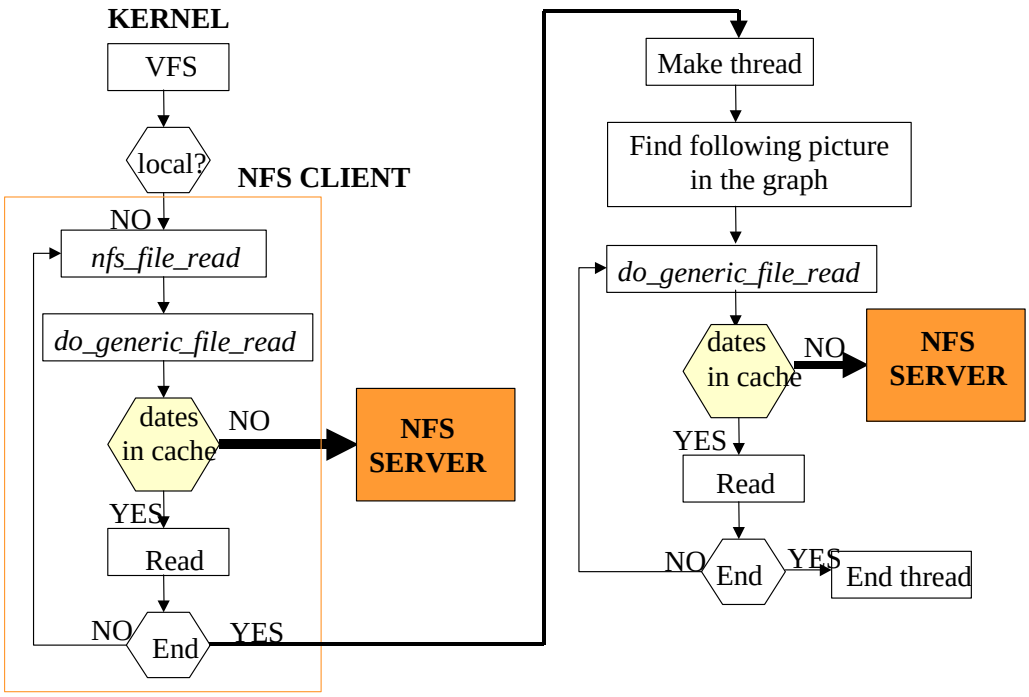

Figure 7. New prefetching in the kernel

When a request is made to read some information, this is translated into reading a certain number of memory pages. The first thing it does is to calculate the corresponding page and to look for that page. If the page is not null, then the function generic_file_readahead is called, which will read the following pages. But, if not, a new page must be created, and the read_page function is called, which will make requests to the server. Also, after this request it tries to read the following page calling to the generic_file_readahead function. When the read pages agree with the amount that is specified, the do_generic_file_read function is finished.

We should also mention that for our work the collaboration with the group of Toni Cortes group of the Polytechnical University of Catalonia, http://www.ac.upc.es/homes/toni, on prefetching has been of vital importance.

5.2. The Predictive NFS Client Design

From all the existing bibliography, it is evident that a great amount of research work on prefetching techniques with different intentions exists. There are two main situations, the world of relating to operating systems or the applications, and their interfaces with the operating system. These last works are those that interest us most for the similarity of the idea, although our proposal is much more aggressive and realistic.

We took from [18] the idea of the *Last Successor* algorithm, which is based on the following: if file f2 has been read after f1, then it will always happen like that. On this basic idea other possibilities that incorporate probabilities, number of accesses and other possibilities are developed. But the fundamental idea is the repeatability in the accesses, that is to say, as the kernel is not able to guess, based on which it has happened before, goes to predict what is happening next. And this is our new idea for NFS, the capacity to predict the following file to read.

Whith different load patterns, we have been able to form an idea of the different situations that could arise. There were several fundamental questions to answer, like: When should we use prefetching? How much information? How should be carry it out? In order to answer them it was necessary to study usual situations with multimedia applications.

In order to respond to first, When should we use prefetching?, we suppose that there is a part of data reading and a part of processing of those data. We want to send a process that sends ahead the following data, in parallel with the processing stage, to take advantage of the server's idlest stages. Therefore, the stage of data reading it will be necessary to send the prefetching process.

Besides, the advancement of files must be done during a processing stage in which we are sure that we are going to allow time to bring the data to the client cache. In the case of MPEG-2 encode, the reading of a picture implies reading three files consecutively (one of luminance and two of crominancia). Once the reading of those three files is finished, prefetching of the three following ones would be made. But, it is necessary to carry out this process in a general way. So, we can conclude, in the first place, that prefetching will be made *when there is time*.

In order to respond to the second question, How much information?, and to know exactly how much information to advance, we cannot be guided by one application in particular, so we will establish the following: *information will go ahead while time is available.*

Now we are going put these ideas in practice and respond to the third question, How should be carry it out?. We have already commented that we are going to use an access graph, since this has been used in the bibliography successfully, and we want to really apply it and to optimize it with new ideas. The access has two main functions. In the first place we will use the access graph as a storage device, in each node, the files that have been asked the server to read, as well as useful information. Secondly, we will use the graph as a structure of historical knowledge, where one goes to look for the file that is going to be needed, based on an order of previous file accesses.

In order to operate all the previous functionality one must have stored in each node some additional information, in addition to the name of the file at issue. The structure of a node will be something like this: a field *previous* with the name of the previous file, a field *current* with the name of the file that has just been accessed, a field *tbetween* with time passed between the reading of the previous file and the present one (if this time is not superior to a certain threshold, the file will not go ahead), a field *offsets* that will keep offsets from beginning and aim of the present file, to specify the piece of data read in the first access, and a field *others* with another fundamental information for the location of the files, as they are the corresponding inodes and certain information of the file and its location.

5.3. Different test with the multithread predictive client

We will expound the results obtained for the reading times with load pattern two, since the most general case is that the client reads the complete file. The incorporation of this new prefetching technique in NFS client is going to allow the anticipated reading of data, and therefore, it tries to eliminate the cache faults in the client.

The prefetching technique is thought to advance the piece of data that is needed by each one the clients which collaborate in the parallel processing of a certain multimedia application. The concept to advance blocks has been rejected totally and now it goes to advance pieces of data. In addition, the system must learn what the reading pattern of a certain application is, that is to say, if file to file is read, or if a consecutive number of these are read before the processing of the data begins.

The greatest restriction arises with load pattern 1, where each client reads only one slice of the picture. In these cases it may be that certain clients do not arrive at the end of file. Thus, this mark could only be used for the pattern of load 2 where the complete files are read. But as we want a general implementation, for any pattern, we must look for another point in the code where to shoot prefetching. This was solved knowing that immediately after data reading function, the client use the nfs_file_flush function. Therefore, it will be in this function where the thread process is created.

The execution of this technique is not going to suppose any overload for the client, since it will be a different process (the thread process) that deals with the creation

and search in the graph. We will have more processes in cluster, more processes sending requests to the server, but they never coincide in the time.

In the tests, we use two different kernels, one without prefetching, the traditional, kernel as it comes with the Linux distribution, and our new kernel with prefetching, with the NFS client modified. For each one of the experiments 10 executions have been made to assure the data reliability. Between each one of the executions we must clean the client cache and the server cache, since if anything remained in these caches the results would not be independent and the effect of this technique could not be seen with clarity. Therefore it is very important to ensure that, between executions, the caches are totally clean.

In the kernel with prefetching, the first execution implies the creation the access graph. In the following executions (when they access the files), as the graph is already constructed, the prefetching demonstrates its utility. Therefore, in the first execution with this kernel, the results must be similar to the ones obtained with the kernel without prefetching, and with the following executions, a reduction in the times should be noticed.

In the following table (Table 1) the average reading times for 400 pictures and 7 processors appear. We began analyzing these results to verify the effectiveness of the method and to avoid confusion with the change of other parameters. We are now going to present the most problematic case, i.e. with a greater number of machines, in both clients (without prefetching and with prefetching), and later, all the other cases with a different number of processors. In order to see if this technique was effective we also compared it with other two values:

- *Application with prefetching*: the result of prefetching implementation in the application itself, the parallel MPEG-2 video encoder
- *Clients with local readings*: the result of copying all the video sequence in the hard disk of the client, so that NFS requests to the server do not exist

These two last tests are carried out with the kernel without prefetching. All the times are expressed in seconds.

Table 1. Reading Times with seven processors

Cases Comparison	**Reading Times**
Kernel without prefetching	46,6369
Kernel with prefetching	**30,151**
Application with prefetching	25,5374
Client with local reading	30,269

As we see, the time reduction when prefetching is used is almost 40% with respect to kernel without prefetching. In addition, we obtain a result very similar to that obtained when the client takes a local reading of the video pictures, which are indeed what we wanted to obtain. The local reading of data is a case that cannot be raised in fact because we would be speaking of original video sequences with many gigabits, so that is far better to have them in a central data server and to access them through a file system like NFS. But it suits our purposes well as an example of times at which we want to arrive. Let us see what happens now in the remaining cases. In Table 2, we can

see that there is a considerable reduction in all the cases. Let us also see graphically the results (see Figure 8).

Tabla 2. Reading Times for different number of processors

# PROC	WITHOUTPREFETCHING	WITH PREFETCHING
1	38,1026	30,7603
3	35,3641	22,1744
5	37,383	28,0724
7	46,6389	30,151

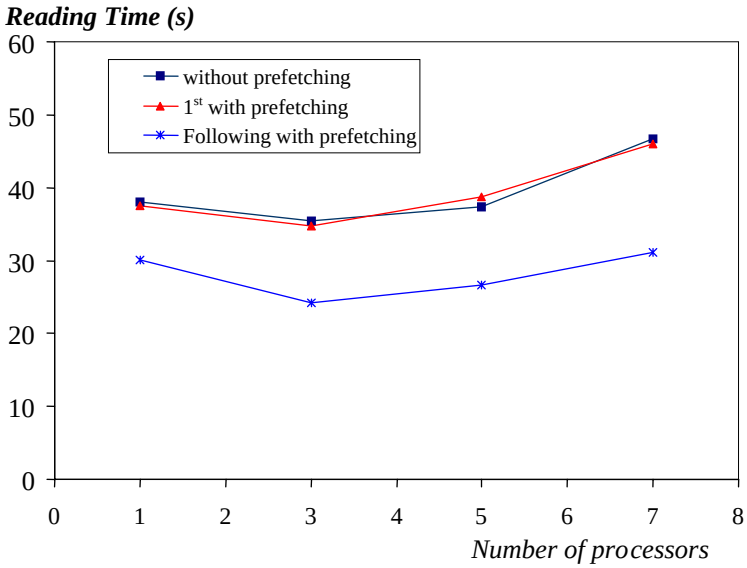

Figure 8. Reading Times for different number of processors

Our aim is to represent in Figure 8 three different times. First, the ones obtained without prefetching that illustrate the problem:The increase in times with the increase in the number of processors. Later, we see the times obtained with prefetching. We distinguish the first execution from the rest. In this first execution, the thread process constructs the access graph, which will remain for all the others. The purpose of this is to show that during this execution, there will be no problems of overload in the client or negative effect on its normal operation. For that reason, the times obtained are almost identical to those obtained without prefetching. Finally, we represent the times obtained using the prefetching technique, which we obtain by making the average of the executions following the first, where is apparent an important reduction in the reading times.

5. Conclusion and Future work

With the increasing complexity of networks and applications, best techniques to share the information are clearly needed. We have identified a serious scalability problem in the most widely used files system in these days. This problem has been showed with real results for the two versions of NFS (NFS v3 and NFS v4).

We present the implementation of a prefetching technique through files as an useful solution contributing to reduce the effects of server overload. We have described how this technique works and the good results obtained.

However, there is still some work to do. We asked ourselves if the reading times could be reduced still more. After numerous code revisions we reached the following conclusion: we are working in an immersed module in the kernel and this will mark the operation of all the system. The operating system continues to manage everything and, in particular, the memory pages. It is not our aim to change this, but we have confirmed the existence of randomly page faults in all studied load patterns, which damage the benefit of our new prefetching technique. So, we establish that this is the cause of not having obtained more reductions in the reading times with our prefetching technique.

Now, after implement the first version of a predictive NFS client, we start a new work phase. Encouraged by the good results obtained we present the following work to do. One challenge is the maintenance of the access graph. It is possible to extend the maintenance of the access graph which we used to discover the file that is due to be read next. Regarding the optimization of the prefetching detection phase, we are working to improve the *automatic* detection of prefething to be applied based on the way to read the information for different multimedia applications. Finally, a new patche with the predictive client would be made. We are preparing the new code so that it comprises the *predictive NFS client*, which can be compiled in the kernel.

References

[1] RFC 3530, *Network File System version 4 Protocol,* April 2003.

[2] RFC 1813, *Network File System version 3 Protocol Specification*, June 1995

[3] H. Sbeyti, S. Niar, L. Eackhout, *Adaptative Prefetching for Multimedia Applications in Embedded Systems*, Proceedings of the Design, Automation and Test in Europe Conference and Exhibition , DATE'04, 2004

[4] T. Mitra, C. Yang y T. Chiweh, *Application-Specific File Prefetching for Multimedia Programs*, Proceedings de la Conferencia Internacional IEEE: Multimedia and Expo, ICME'2000

[5] F. H. P. Fitzek and M. Reislein, *A Prefetching Protocol for Continuous Media Streaming in Wireless Environments*, Technical Report TKN-00-05, Telecommunications Network Group, Technical University Berlin, 2005

[6] B. Callaghan, *NFS Illustrated***,** Addison Wesley, 2000

[7] W. von Hagen, *Linux File systems*, Sams Publishers, 2001

[8] T. Olivares, P. Cuenca, F. Quiles, A. Garrido, J. Sanchez and J.Duato, *Interconnection Network Behavior on a Multicomputer in the Parallelization of the MPEG Coding Algorithm. Worm-hole vs Packet-Switching Routing*, Proceedings de la cuarta Conferencia Internacional: High Performace Computing (HiPC'97), IEEE Computer Society Press, ISBN 0-8186-8067-9, Vol. 1, pp. 48-53, Diciembre 1997, Bangalore (India).

[9] T. Olivares, P. Cuenca, F. Quiles and A. Garrido, *Parallelization of the MPEG coding Algorithm over a Multicomputer. A Proposal to Evaluate its Interconection Network.* Proceedings del IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM´97), IEEE Computer Society Press, Vol. 1, pp. 113-116, 1997, CANADA.

[10] T.Olivares, F.Quiles, P.Cuenca, L. Orozco y I. Ahmad, *Study of Data Distribution Techniques for the Implementation of an MPEG-2 Video Encoder*, Proceedings de la Conferencia Internacional IASTED: Parallel and Distributed Computing and Systems (PDCS'99), ISBN: 0-88986-275-3, Vol. 1, pp. 537-542, 1999, Cambridge, Massachussets (USA)

[11] Dean Hildebrand and Peter Honeyman, *Scaling NFSv4 with Parallel File Systems*, Cluster Computing and Grid (CCGrid05), Cardiff, United Kingdom, May 2005.

[12] Dean Hildebrand and Peter Honeyman, *Exporting Storage Systems in a Scalable Manner with pNFS*. 22nd IEEE - 13th NASA Goddard (MSST2005) Conference on Mass Storage Systems and Technologies, Monterey, California, April 2005

[13] T. Olivares, F.J. Quiles, A. Garrido, P.J. García, L. Orozco-Barbosa, *Performance Study of NFS over Myrinet-based Clusters for Parallel Multimedia Applications*, Proceedings de la Canadian Conference On Electrical And Computer Engineering, (IEEE Computer Society Press), Mayo 2001, Toronto (Canadá)

[14] T. Olivares, F.J. Quiles, A. Garrido, P.J. García, L. Orozco-Barbosa, *The Need of Multicast Predictive NFS Servers for High-Speed Networks used as Parallel Multimedia Platforms*, Proceedings del ICPP'01 Worshop on Scheduling and Resource Management for Cluster Computing, ISBN: 0-7695-1260-7, pp.391-396, IEEE Computer Society Press, Septiembre 2001, Valencia (España)

[15] Garth Gibson, and Peter Corbett *pNFS Problem Statement: Abstract*, Panasas Inc. & CMU and Network Appliance, Inc. July 2004. http://www.pdl.cmu.edu/pNFS/

[16] T. Olivares, *Clusters Optimization like Multimedia Platforms using Predictive Multithread Clients*, Doctoral Dissertation, February 2003

[17] Myricom, Pioneering Higher Performance Computing, http://www.myri.com

[18] T.M. Kroeger y D. D.E. Long, *Design and Implementation of a predictive File Prefetching Algortihm*, Proccedings of the USENIX, June 2001, Boston.

[19] Center for Information Technology Integration, University of Michigan, http://www.citi.umich.edu/projects/