Optimizations Based on Hints in a Parallel File System

María S. Pérez, Alberto Sánchez, Víctor Robles, José M. Peña, and Fernando Pérez

DATSI. FI. Universidad Politécnica de Madrid. Spain
{mperez,ascampos,vrobles,jmpena,fperez}@fi.upm.es

Abstract. Existing parallel file systems provide applications a little control for optimizing I/O accesses. Most of these systems use optimization techniques transparent to the applications, limiting the performance achieved by these solutions. Furthermore, there is a big gap between the interface provided by parallel file systems and the needs of applications. In fact, most of the parallel file systems do not use intuitive I/O hints or other optimizations approaches. In this sense, applications programmers cannot take advantage of optimization techniques suitable for the application domain. This paper describes I/O optimizations techniques used in MAPFS, a multiagent I/O architecture. These techniques are configured by means of a double interface for specifying access patterns or hints that increase the performance of I/O operations. An example of this interface is shown.

Keywords: Parallel I/O, optimizations, caching, prefetching, hints.

1 Introduction

I/O systems have traditionally been the bottleneck of computing systems because of the difference between I/O devices access and computing time, avoiding data-intensive applications achieve enhanced improvements. The usage of parallel I/O architectures and, in this particular case, parallel file systems has involved a widely used approach that mitigates in some sense this problem, known as *I/O crisis*.

Parallel file systems achieve good performance in some scientific applications, by means of *ad hoc* solutions, tailored to the particular problem by an I/O expert. This is not the usual scenario, since applications are developed by software teams, which do not usually know the underlying I/O architecture deeply. On the other hand, data-intensive applications have very different data access patterns. Therefore, it is not possible to improve these applications performance with a non-flexible architecture, which applies the same techniques to applications, without taking into account their requirements.

I/O models arise within MAPFS (MultiAgent Parallel File System) [10] with the aim of fixing the following problems:

- Applications have different data access patterns, depending on their domain.
- Additional information for increasing data access performance is also based on the particular application.
- Most of the studied access patterns correspond to scientific applications. In fact, a great amount of studies has been performed on scientific I/O workloads, as much in parallel as sequential frameworks.

M. Bubak et al. (Eds.): ICCS 2004, LNCS 3038, pp. 347–354, 2004.

Although MAPFS is a general purpose file system, it is very flexible and it can be configured for adapting to different I/O access patterns. In MAPFS, data items from secondary storage are "cached" in memory for faster access time. This approach allows MAPFS to increase the performance of the I/O operations. This operation is made in MAPFS by means of an independent subsystem, constituted by a multi-agent system. This task can be made in a more efficient way using hints on future access patterns.

The outline of this paper is as follows. Section 2 describes the I/O optimization techniques implemented by MAPFS. Section 3 presents MAPFS configuration for using these optimization techniques. Section 4 shows the evaluation of a scientific application using MAPFS optimization techniques. Section 5 describes the related work. Finally, Section 6 summarizes our conclusions and suggests further future work.

2 I/O Optimizations Techniques

Although I/O systems have traditionally tackled the I/O phase in the same way, independent of the applications domain and their access pattern, some studies [8], [5] have demonstrated that a higher control of the user applications over the I/O system can increase their performance. MAPFS provides this control as part of its functionality. Furthermore, MAPFS increases its interface in such way that user applications can make use of this control. Next we are going to analyze these MAPFS capacities.

2.1 I/O Caching and Prefetching

If the behaviour of an algorithm is known, I/O requirements could be achieved in advance before the algorithm actually needs the data. An optimal usage of computational resources, like I/O operations or disk caches is also a key factor for high-performance algorithms. With the usage of data access knowledge both accurate prefetching and resource management could be achieved.

For using these features, it is necessary to have a cache structure, where the I/O data items are stored in memory. The effectiveness of the cache is determined largely by the replacement policy. A typical cache replacement policy is *Least Recently Used (LRU)*. This policy assumes that data items used recently are likely to be used again. Other alternatives widely used are the FIFO policy, which replaces the oldest item and MRU policy, which replaces the most recently used item. Historical information is often used for both file caching and prefetching. In the case of the prefetching, the most used approach is sequential readahead. All these policies are used in conventional systems. Nevertheless, these policies are not suitable for all the different access patterns of applications. A more general scheme consists in using hints about applications with the aim of increasing of the caching and prefetching phases.

2.2 Usage of Hints

As we have mentioned previously, MAPFS uses hints in order to access in an efficient way to files. Hints are structures known and built by the file system, which are used for improving the read and write routines performance. In MAPFS, hints can be determined in two ways: (i) they can be given by the user, that is, the user application provides

the necessary specifications to the file system for increasing the performance of the I/O routines, and (ii) they can be built by the MAPFS multiagent subsystem. If this option is selected, the multi-agent system must study the access patterns of the applications in order to build hints that improve both the caching and prefetching stages.

The semantic of the applications that use the I/O system must configure the hints of the underlying I/O system. Therefore, this I/O system must be flexible enough to allow the application to modify the hints. In a generic way, hints can be based on different attributes values related to I/O configuration parameters.

3 MAPFS Configuration

As we have mentioned previously, most of the data accesses can be settled by means of *attributes*. For example, in Data Mining applications, the queries are based on values of certain database attributes. In other kind of applications, metadata, which stores additional information about data, can be used to access them in a more efficient fashion. This metadata also can be structured by means of attributes. Therefore, hints in MAPFS must be able to store simple regular expressions, composed of boolean operators applied to these attributes. These expressions allow stored data to be classified.

MAPFS provides a double interface for specifying access patterns or hints. The first one (*user control operations*) is a high level interface, adapted to applications, which may be used in terms of attributes. Applications can manage the performance of the system only by means of this API. The second one (*hints setting operations*) is nearer to the file system and may be used for subsystems that belong to the parallel file system or expert I/O programmers. In [11], a complete syntax of the MAPFS configurations operations is shown. This paper is going to describe a sample of scientific application using the MAPFS interface.

3.1 Matrix Multiplication as Sample Application

Operations about matrices are significative examples of scientific applications. From the point of view of hints usage, the addition of matrices offers the following alternatives. When the first element of the row *i* of the first matrix is accessed, the I/O operation may be optimized if complete rows *i* of both matrices were prefetched. The problem appears when a row does not fit in the cache structure. In this case, this rule is not so trivial. This example will be analyzed in Section 3.2. In the case of the multiplication of matrices, rules for achieving optimal prefetching are not obvious either.

Let be a multiplication operation between matrices: $A[M,N]*B[N,P] = C[M,P]$. in such way that the matrices are stored by rows in their respective files and they have a considerable size, so that a row or column does not fit completely in the cache structure.

On the other hand, let be a traditional multiplication algorithm:

```
for (i=0; i<M; i++)
  for (j=0; j<P; j++)
    for (k=0; k<N; k++)
       C[i][j]+=A[i][k]*B[k][j];
```

Next, possible prefetching strategies are analyzed.

3.2 Prefetching Rules on the Matrix A

Regarding to the matrix A, instead of reading in advance the row corresponding to the requested element, other strategies are more suitable because of the following facts:

- Whenever the current row has been used, that is, the index *i* advances one position after using the row *P* times, a hole has been left in the prefetching. In this case, it would be advisable to request in advance elements of the row *i+1*.
- It is possible that the cache has not enough space to store the complete row in the cache. Then, it would be better reading in advance only one window of K values, which would correspond to the K following elements of the request, but in a circular fashion (N module), where K can be determined by different factors, such as the cache size or the system load.

According to this, the following rules can be defined:

- **RULE 1:**
  Row prefetching with circular window K (N module), a P-times cycle by row, sequential advance by rows (and without matrix repetition cycle). This rule is equivalent to:

  $read(A[i,j]) \Longrightarrow$

  If the current cycle is not the last one (number of accesses $A[i,j] < P$) :

  • Prefetching from $A[i, j+1]$ until $A[i, (j+K)\%N]$

  If the current cycle is the last one (=P) $\Longrightarrow$ advance to the following row:

  • Prefetching of the K following elements both elements of the present row,

  if they are left, and first elements of the "$i+1$" row, if it is necessary and $i < M$

3.3 Prefetching Rules on the Matrix B

Prefetching on the matrix B is more complex, because it is necessary to access matrix data by columns, which decreases the performance, since small accesses are made. In this case, it is possible to raise different strategies.

- **Horizontal Prefetching**
  When the element $B[i,j]$ is accessed, it is advisable to prefetch a small data window of size K' of that same row, corresponding to later elements than required element $B[i,j]$, in a circular fashion (module P). These values will not be used inmediately, but they are used when the index "j" of the loop is increased, that is, when an element of C is calculated. Thus, K´ must not be a large value because, otherwise, this window is bigger than the cache size.
- **RULE 2**
  On the matrix B, it can be made a row prefetching with circular window of size K'(module P), a cycle of M repetitions by row and without advance by rows (nor matrix repetition cycle), because after the M cycles, the algorithm will have finished.

- **Vertical or Column Prefetching**
  It seems reasonable that the read request of $B[i, j]$ causes a prefetching in that column ($B[i+1, j], B[i+2, j], \ldots$). Although the vertical prefetching is theoretically feasible, it can involve small reads, achieving a poor performance. On the other hand, if the prefetching is transitive (see Section 3.4), when the rule that activates the prefetching of $B[i+1, j]$ is triggered, the previous rule (RULE 2) will be activated by transitivity, requesting the horizontal K'-size window. This will improve the I/O performance. Considering that the column can be bigger than the cache and it is not possible to prefetch it completely, the following two rules can be used:
  1. Row prefetching with K'-size window according to RULE 2.
  2. The new rule corresponding to vertical prefetching (RULE 3).
- **RULE 3**
  On the matrix B, it would be made a column prefetching with K"-size window, without repetition cycle by column, with sequential advance by columns and a cycle of M repetitions of matrix. In the case that the complete column does not fit in the cache, the positive effect of RULE 2 disappears, because prefetched data is discarded. Therefore, K' would have to be equal to 0.

3.4 Transitive or Induced Prefetching

A prefetching rule provides prefetching of probably referenced data in next read operations. This prefetching rule is associated to a concrete data block. In the case of the matrix multiplication, it is associated to an element or a set of elements. If transitives rules are allowed, the access to a concrete data block, which is associated to a certain prefetching rule, involves another prefetching rule, associated to the data block that has been prefetched by the first rule. Nevertheless, it is necessary to control this transitivity, because in certain cases, it is possible that the complete file may be read.

3.5 Induced Prefetching between Files

This kind of prefetching implies prefetching data of a file, induced by accesses in another file.

- **RULE 4**
  In the case of the multiplication of matrices, it is possible to do an induced P-cycle prefetching, so that the first access to $A[i, j]$ causes the prefetching of $B[j, 0], B[j, 1]$ and so on.

3.6 Combination of Prefetching Rules

Joining all the previous rules, an access to $A[i, j]$ causes the following accesses:

1. Using RULE 1, a prefetching of the K next elements in the row "i" is made in a circular fashion. In the last cycle, we advance to the row "i+1".
2. Using RULE 4, a prefetching of $B[j, x]$ is made (if it is the x-th time that the element is accessed)
3. By transivity, the following rules can be applied:
   a) RULE 2, that is, the K' elements of the row i of B are accessed.
   b) RULE 3, that is, the K" elements of the column j of B are accessed.

4 Hints Evaluation

As is described in the previous section, in the case of the matrix multiplication, it is possible to apply different rules and even a combination of such rules. These rules use different configuration parameters, which are the basic expressions or component of such syntactic rules. In the case of the prefetching, these parameters or attributes are:

- Type: It may be per row (horizontal) or per column (vertical).
- Window type: It may be circular or not.
- Window size: Number of elements to read in every cycle of prefetching.
- Cycle per row or per column: It specifies if there exists cycle per row or per column or not. This last case is indicated with a value 0.
- Cycle per matrix: It specifies if there exists cycle per matrix or not. This last case is indicated with a value 0.

The user control structure provided by the application follows this boolean expression:

```
Type=Row AND Window Type=Circular AND Window Size=K
         AND Cycle Per Row=P AND Cycle Per Matrix=0
```

This user control structure is translated to hints by the multiagent subsystem. Hints are used as data tags for processing the elements in an efficient manner, prefetching data provided by the corresponding rule. In this case, hints have been used in order to increase the performance of the prefetching phase. However, the hints can be used together with other techniques or algorithms.

In order to evaluate the performance of the MAPFS hints, we have implemented the matrix multiplication in MAPFS and in the native interface of the *Parallel Virtual File System* PVFS [1]. This section shows the comparison between the time of the matrix multiplication in MAPFS and in PVFS.

Figure 1 represents this comparison for the multiplication of a single row of two 100 MB-size matrices. As this figure represents, we can see that MAPFS solution without hints is lightly slower than PVFS solution. However, MAPFS solution with hints is faster than PVFS solution. Therefore, the usage of hints increases the performance in a flexible way.

The more rows are processed, the more differences are in the execution times. Figure 2 shows the evaluation of the multiplication of 100 MB-size matrices. As can be seen, in this case the differences are more significative.

5 Related Work

A suitable data placement is extremely important in the increase of the performance of I/O operations. Most of the file systems provide transparency to user applications, hiding details about the data distribution or data layout. Some file system such as nCUBE [4] or Vesta [3] provide programmer a higher control over the layout.

Panda [2] hides physic details of the I/O systems to the applications, defining transparent schemas known as *Panda schemas*.

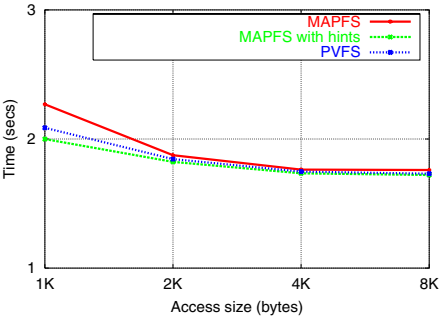

Fig. 1. Comparison of the matrix multiplication in PVFS and MAPFS (a single row)

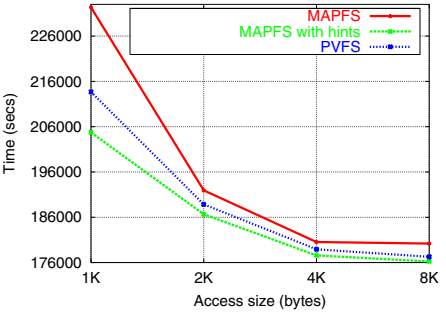

Fig. 2. Comparison of the matrix multiplication in PVFS and MAPFS (100 MB-size file)

PPFS defines a set of access pattern specified when a parallel file is opened [6].

MPI-IO uses *MPI datatypes* to describe data layout both in memory and in file, in such way that it is possible to specify non-contiguous access [12]. Data layout in memory is specified in every I/O call in MPI-IO. Data layout in file is defined by a file view. The function `MPI_File_set_view()` is used for setting this view.

On the other hand, hints are widely used for increasing the system performance, in general topics in computing systems. In the context of the file systems, hints are usually used as historical information for optimizing caching and prefetching techniques. For example, hints can be used in the prefetching phase for deciding how many blocks are read in advance. OSF/1 system prefetchs up to 64 data blocks when large sequential request are detected [9]. Other works detect more complex access patterns for non-sequential prefetching. There exists a great number of works which infer future accesses based on past accesses. In [7], historical information is used in order to predict the I/O workload and, thus, increase the prefetching stage.

6 Conclusions and Future Work

This paper presents the usage of hints in MAPFS, showing a sample application, a multiplication of matrices. We use the MAPFS configuration interface in order to provide hints to the I/O system. As the experimentations show, the usage of hints is a flexible way of increasing the performance of the I/O phase. MAPFS provides a double interface, a high level interface, adapted to applications and a second one, used for subsystems belong to the parallel file system or expert I/O programmers.

Nevertheless, the hints configuration is not an obvious task. As future work, we are applying autonomic computing in order to allow the I/O system to be configured itself.

References

1. P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur. PVFS: A parallel file system for linux clusters. In *Proceedings of the 4th Annual Linux Showcase and Conference*, pages 317–327, October 2000.
2. Yong E. Cho. *Efficient resource utilization for parallel I/O in cluster environments*. PhD thesis, University of Illinois at Urbana-Champaign, 1999.
3. P. Corbett, D. Feitelson, J. Prost, and Johnson S. Parallel access to files in the Vesta file system. In *Proc. of the 15th. Int. Symp. on Operating Systems Principles*, pages 472–481. ACM, 1993.
4. E. DeBenedictis and J. M. del Rosario. nCUBE parallel I/O software. In *Eleventh Annual IEEE International Phoenix Conference on Computers and Communications (IPCCC)*, pages 117–124, Apr 1992.
5. J.M. del Rosario and A.N. Choudhary. High-performance I/O for massively parallel computers: Problems and prospects. *IEEE Computer*, 27(3):59–68, 1994.
6. C. Elford, C. Kuszmaul, J. Huber, and T. Madhyastha. Scenarios for the Portable Parallel File System, 1993.
7. David Kotz and Carla Schlatter Ellis. Practical prefetching techniques for parallel file systems. In *Proceedings of the First International Conference on Parallel and Distributed Information Systems*, pages 182–189. IEEE Computer Society Press, 1991.
8. Barbara K. Pasquale and George C. Polyzos. A static analysis of I/O characteristics of scientific applications in a production workload. In *Proceedings of Supercomputing '93*, pages 388–397, 1993.
9. R. Hugo Patterson, Garth A. Gibson, Eka Ginting, Daniel Stodolsky, and Jim Zelenka. Informed prefetching and caching. In Hai Jin, Toni Cortes, and Rajkumar Buyya, editors, *High Performance Mass Storage and Parallel I/O: Technologies and Applications*, pages 224–244. IEEE Computer Society Press and Wiley, New York, NY, 2001.
10. María S. Pérez, Jesús Carretero, Félix García, José M. Peña, and Víctor Robles. A flexible multiagent parallel file system for clusters. *International Workshop on Parallel I/O Management Techniques (PIOMT'2003) (Lecture Notes in Computer Science)*, June 2003.
11. María S. Pérez, Ramón A. Pons, Félix García, Jesús Carretero, and Víctor Robles. A proposal for I/O access profiles in parallel data mining algorithms. In *3rd ACIS International Conference on SNPD*, June 2002.
12. Rajeev Thakur, William Gropp, and Ewing Lusk. Optimizing noncontiguous accesses in MPI-IO. *Parallel Computing*, 28(1):83–105, 2002.