

A Parallel Out-of-Core Computing System

Using PVFS for Linux Clusters

Jianqi Tang Binxing Fang Mingzeng Hu Hongli Zhang
Department of Computer Science and Engineering
Harbin Institute of Technology, Harbin, 150001, China
tjq, bxfang, mzhu, zhl@pact518.hit.edu.cn

Abstract

Cluster systems become a new and popular approach to parallel computing. More and more scientists and engineers use clusters to solve problems with large sets of data for its high processing power, low price and good scalability. Since the traditional out-of-core programs are difficult to write and the virtual memory system does not perform well, we develop a parallel out-of-core computing system using PVFS named POCCS. POCCS provides convenient interface to write out-of-core codes and the global view of the out-of-core data. The software architecture, data storage model and system implementation are described in this paper. The experimental results show that POCCS extends the problem sizes that can be solved and the performance of POCCS is better than the virtual memory system while the data set is large.

Keywords out-of-core computing, parallel file system, out-of-core array, Linux cluster

1. Introduction

As the performance of workstations has been continuously improved, the

cluster of workstations is more and more used in the area of high performance computing. These clusters are often built using existing workstations which are connected with Ethernet or a similar networking technology. They are effective to carry out large-scale parallel calculations and provide parallel computing facilities at a lower cost than traditional massively parallel computers [1, 2].

Some scientists not only need powerful processing capacity, but also attempt to solve problems with very large data sets. For example, large scientific applications like Grand Challenge Applications require 100GBytes of data per run [3]. In these applications, all the data required by the program cannot fit in the main memory at the same time. Thus, data needs to be stored in files on disks and this computation is called out-of-core computation [4]. Accomplishing computation requires staging data in smaller granules that can fit in the main memory of the system. That is, the computation is divided into several phases, where in each phase part of the data is fetched into memory, processed, and stored back onto disks. A typical out-of-core code is organized around a few disk files and interspersed with explicit I/O statements,

which is difficult to write and modify.

On Linux clusters, programmers can easily handle the large data sets by the virtual memory system that is supported by the operating system. While this approach does yield a logically correct answer, the resulting performance is typically so poor that it is not considered a viable technique for solving out-of-core problems [5], and the virtual memory system only support handling large data sets in single process applications. Parallel file systems are one approach to providing the global viewpoint for programmers. PVFS is a leading parallel file system for Linux cluster systems [6]. It was designed to meet increasing I/O demands of parallel applications in cluster systems. Using the interface libraries provided by PVFS, programmers might implement out-of-core applications avoiding the use of virtual memory.

However, programmers still have to write a large number of I/O statements in the programs to solve out-of-core problems even by using PVFS. We are attempting to combine the advantages of virtual memory that makes programs relatively easy to write without explicit I/O operations and parallel file systems which provide global view of a large data set. To do this we have developed a runtime system POCCS (Parallel Out-of-Core Computing System) for efficiently and easily solving the out-of-core applications on Linux clusters.

The rest of this paper is organized as follows. An overview of PVFS is given in Section 2. Section 3 describes our POCCS system including software architecture, data storage model and interface design. The performance results are presented in Section 4. We draw our conclusions and discuss future work in Section 5.

2. Overview of PVFS

PVFS is a parallel file system for Linux clusters, called the Parallel Virtual File System. It is intended both as a high-performance parallel file system that anyone can download and use and as a tool for pursuing further research in parallel I/O and parallel file systems for Linux clusters [7, 8]. The PVFS project is conducted jointly between The Parallel Architecture Research Laboratory at Clemson University and The Mathematics and Computer Science Division at Argonne National Laboratory [9]. PVFS Version 2 (PVFS2) [10] has just been released. The PVFS overview in this section is about PVFS1, though some basic concepts may be applied to PVFS2 as well.

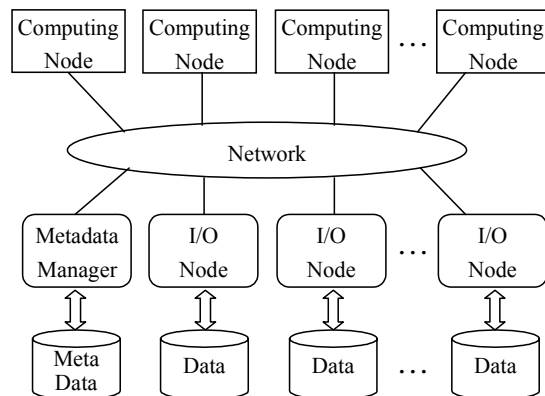


Figure 1. Typical PVFS Architecture

PVFS provides a global name space, striping of data across multiple I/O nodes, and multiple user interfaces. The system is implemented as a set of user-level daemons and an interface library that applications could use to interact with the system [11]. as shown in Figure 1, A number of nodes in a cluster system can be configured as I/O servers and one of them is also configured to be the metadata manager. It is possible for a node to host computations while serving as an I/O node.

PVFS files are striped across a set of I/O nodes in order to facilitate parallel access. The individual file stripes are stored by using the native file system on the I/O servers. A manager daemon runs on a metadata manager node. It handles metadata operations involving file permissions, truncation, file stripe characteristics, and so on. The metadata manager provides a clusterwide consistent name space to applications. Application processes communicate with the metadata manager when performing operations. The manager returns to the application the locations of the I/O nodes on which file data is stored. This information allows applications to communicate directly with I/O nodes when file data is accessed.

The PVFS developer team once designed a MDBI interface to help in the development of out-of-core algorithms operating on multi-dimensional datasets, which can read and write blocks by specifying their indices in an out-of-core array [12]. But MDBI is not included in the recently released PVFS version. Actually, MDBI simplifies the I/O statements writing and provides some buffering strategies, but it still can not decrease the explicit I/O operations in a program.

3. The Parallel Out-of-Core Computing System

POCCS is designed to achieve convenient and efficient access to the out-of-core data on Linux clusters. It is a runtime library system with its own interface for programmers. This section will describe the software architecture, data storage model and system implementation of POCCS.

3.1 Software Architecture

As shown in Figure 2, POCCS provides the programmer with a simple high-level interface, which is a level higher than the PVFS interfaces. PVFS gives a global view of the out-of-core data that is striped and stored on I/O nodes, while the POCCS interface makes the I/O operations transparent to programmers. The parallel node program is written with some message passing libraries such as MPI and PVM which provide a parallel executable platform for the parallel application.

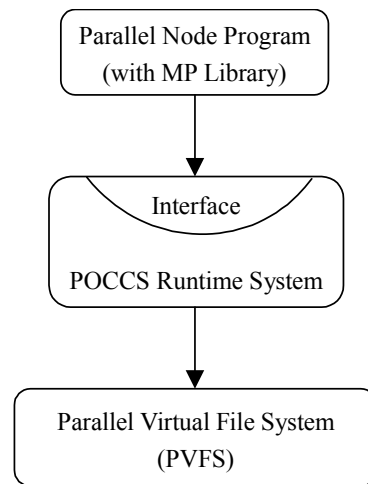


Figure 2. Software Level Architecture

3.2 Data Storage Model

Out-of-core data always exists as the data type of array. Such an array is called out-of-core array. We assume that in an out-of-core application each out-of-core array is stored in a PVFS file. A common Linux file can be configured to a PVFS file by placing the out-of-core file in the subtree of Linux file system that is related to PVFS subspace. Hence data in this file is distributed to the I/O nodes automatically by PVFS. Users may specify the temporary file size on each I/O node, the number of used I/O nodes and which I/O node is to be used. All I/O nodes are

used and the the temporary file size is set to 64Kbytes by default.

For one out-of-core array, during the program running only a portion of this array is fetched and stored in main memory, and is written back to the out-of-core array at proper time. This out-of-core-array can be accessed from any computing node benefiting from the consistent file name space of PVFS. Figure 3 shows the data storage model of POCCS.

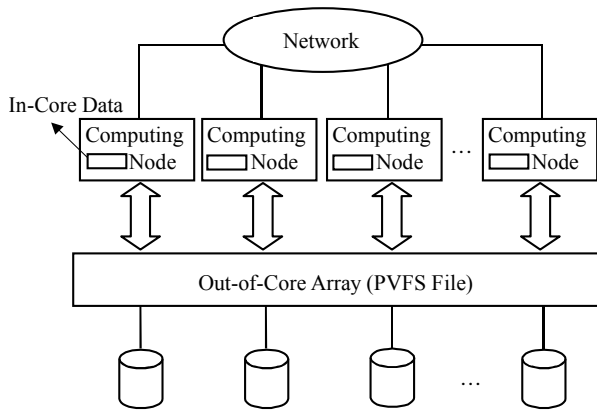


Figure 3. Data Storage Model of POCCS

3.3 System Implementation

POCCS is a library of calls designed for programmers who intend to solve an out-of-core application to write the code similar to the in-core code without worrying about the limited size of main memory. By using the interface, the programmer is allowed to declare a global out-of-core array corresponding to an existing file or a new file, and then use the elements of this array by specifying their indices like an in-core array.

Template class is used to implement the POCCS interface. An out-of-core array is defined as a template class so as to specify different element types for different arrays. It is declared by giving a set of parameters: the data type of the array

element, the number of dimensions, the size of each dimension, the PVFS file name, the flag indicating that it is an existing file or a new file to be created, and the buffer information including the number of buffers and the number of elements in each buffer.

The operator “[]” is overloaded and so the programmer may use such a template class array as an ordinary array without worrying about the limited memory size. The specified indices is sent to the POCCS library by the operator “[]”. Then a block of data that contains the specified element is read into the in-core buffer and the array element is returned. We use multiple in-core buffers corresponding to an out-of-core array. While there is not enough buffer space, the least recently used (LRU) algorithm is adopted to replace a buffer and data in this buffer is written back to the file if it has been modified. Each buffer has a flag indicating whether the data in it is valid.

The multi-dimensional array template class is implemented based on the one-dimensional one. A multi-dimensional array template class consists of several lower-dimensional array template classes as its member variables.

When the “shared” concept is introduced, the consistency of the shared data must be guaranteed. POCCS is designed as a user level library to provide users a convenient and flexible tool to write parallel out-of-core codes. Users are the one who know the data accessing sequence on each computing node along with the assigned tasks. To solve the consistency problem, POCCS provides calls of writeback() and invalidate() for users to adopt together with the barrier functions of the message passing library.

By using `writeback()`, all the buffer data write back to the PVFS file. `Invalidate()` write the buffer flags to makes all the buffers invalidate which implies that the buffer data is out of date. Users employ `writeback()` to ensure that the new values are written back to the disks, and use `invalidate()` to avoid using out-of-date data. The barrier functions of the message passing library guarantee the sequence of the above actions.

4. Experimental Results

Our experiment environment is a cluster system consisting of 4 nodes connected by a 10Mbps hub. The network is isolated from outside traffic. Each node has a Pentium III 450MHz processor, a 64M RAM and a 10GB disk. We used the Linux 2.4.7 kernel, GNU C/C++ 2.96, mpich 1.2.4 and pvfs-1.6.0. The system swap space is set to 128M. The 4 nodes are all configured as I/O nodes and all perform computation. Such mode is considered as the right choice if a fixed set of resources is given [12].

Our experiments were run to examine the performance and the range of problem sizes that could be efficiently solved respectively using POCCS and the virtual memory system. Table 1 shows the performance comparison of the vector inner product program for the one-dimension out-of-core array, and Table 2 shows the performance comparison of the Jacobi iteration program for the two-dimension out-of-core array. The data type of the array elements is integer which is 4 bytes in length. Both of the two programs need to declare two out-of-core arrays.

Table1. Performance of the vector inner product program

Number of Array Elements (N)	Total Array Space (2×4×N Bytes)	Runtime (seconds)	
		POCCS	VM
4 M	32M	0.9	0.2
8 M	64M	2.3	0.4
16M	128M	6.3	1.5
32M	256M	14.9	22.2
64M	512M	34.4	76.4
128M	1G	68.9	Killed
256M	2G	158.3	Killed

Table 2. Performance of the Jacobi iteration program

Number of Array Elements (N×N)	Total Array Space (2×4×N×N Bytes)	Runtime (seconds)	
		POCCS	VM
1K×1K	8M	0.7	0.1
2K×2K	32M	2.6	0.3
3K×3K	72M	6.1	3.1
4K×4K	128M	10.8	5.9
5K×5K	200M	18.9	22.3
6K×6K	288M	28.6	60.2
7K×7K	392M	39.4	100.1
8K×8K	512M	61.3	145.6
10K×10K	800M	112.4	Killed

When the amount of data is smaller, from 4M to 16M in vector inner product and from 1K×1K to 4K×4K in Jacobi iteration, the VM version outperforms the POCCS version. Actually, they are in-core problems in these cases because in VM method the array size on each compute node which is equal to (Total Array space / 4) is less than main memory size 64M. At this time, Linux system swapping is unneeded. Actually, applications with such

small amount of data are in-core problems. Because POCCS has extra overhead on template classes, its performance in in-core problems is not better than VM. Nevertheless, the performance of POCCS for relatively small problems such as 4M in Table 1 and 1K×1K in Table 2 is not tremendously bad, indicating that the POCCS system for applications with small data sets is feasible.

With the increasing array size, the performance of VM method decreases sharply for the program accesses the swap space frequently. When the array size on each node is larger than the sum of main memory and swap space at 128M elements in the vector inner product program and 10K×10K elements in the Jacobi iteration program, the VM version cannot solve the problem and the program was killed by the operating system. The swap size is 128M for that the swap space is usually set to be the double size of main memory. If the swap space is increased, the problem size at killed point will be larger. However, the performance will be terribly bad although the operating system does not kill the program at this time. The bottleneck is at main memory, but not the swap space.

The POCCS implementations show more consistent behavior, scaling well as the problem size grows for the entire tested range. The buffer strategy in POCCS is efficient because it is oriented to the user's applications, while swapping in VM is designed for common use which results in worse performance in given applications.

5. Conclusions

When the data sets involved in an application grow to exceed main memory size, it is troublesome to write the out-of-core codes because of the large

amount of I/O operations. While the virtual memory system can enable applications to run out of core, the results show that in some cases the operating system cannot solve the problems efficiently and it only support single node program. We develop a parallel out-of-core computing system (POCCS) for Linux clusters which is based on PVFS and provides programmers an convenient and efficient interface to light the burden of writing out-of-core programs. The experimental results show that when the data sets are really out-of-core the POCCS implementation outperforms the VM one. Next, we plan to expand the POCCS runtime library as well as the system interface and adopt more optimization strategies to improve the system performance.

References

- [1] J. Mache, J. Bower-Cooley, R. Broadhurst, J. Cranfill, and C. Kirkman IV. Parallel I/O performance of PC clusters. In 10th SIAM Conf. on Parallel Processing for Scientific Computing, Portsmouth, VA, USA, March 12-14 2001.
- [2] K. Castagnera, D. Cheng, R. Fatoohi, E. Hook, B. Kramer, C. Manning, J. Musch, C. Niggley, W. Saphir, D. Sheppard, M. Smith, I. Stockdale, S. Welch, R. Williams, and D. Yip, "Clustered workstations and their potential role as high speed compute processors," Tech. Rep. RNS-94-003, NAS Systems Division, NASA Ames Research Center, April 1994.
- [3] Applications Working Group of the Scalable I/O Initiative. Preliminary Survey of I/O Intensive Applications. Technical Report CCSF-38,

- Concurrent Supercomputing Consortium, Caltech, Pasadena, CA91125, January 1994. Scalable I/O Initiative Working Paper No. 1.
- [4] R. Bordawekar, A. Choudhary and J. Ramanujam. Compilation and Communication Strategies for Out-of-core programs on Distributed Memory Machines. In *Journal of Parallel and Distributed Computing*, 38(2): 277-288, Nov. 1996.
- [5] D. Womble, D. Greenberg, R. Riesen, and S. Wheat. Out of core, out of mind: Practical parallel I/O. In *Proceedings of the Conference on Scalable Parallel Libraries*, Washington DC, USA, 1993. IEEE Computer Society.
- [6] Jiesheng Wu, Pete Wyckoff, Dhableswar Panda and Rob Ross. Unifier: Unifying Cache Management and Communication Buffer Management for PVFS over InfiniBand. *Proceedings of the 4th IEEE/ACM International Symposium on Cluster Computing and the Grid*. IEEE Press, April 19-22, 2004, Chicago, USA.
- [7] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur, PVFS: A Parallel File System For Linux Clusters, *Proceedings of the 4th Annual Linux Showcase and Conference*, Atlanta, GA, October 2000, pages 317-327
- [8] Ligon, III, W.B., and Ross, R. B., "PVFS: Parallel Virtual File System," *Beowulf Cluster Computing with Linux*, Thomas Sterling, editor, pages 391-430, MIT Press, November, 2001
- [9] <http://www.parl.clemson.edu/pvfs>
- [10] <http://www.pvfs.org/pvfs2/>
- [11] W. B. Ligon III and R. B. Ross, An Overview of the Parallel Virtual File System, *Proceedings of the 1999 Extreme Linux Workshop*, June, 1999.
- [12] M. M. Cettei, W. B. Ligon III, and R. B. Ross, Support for Parallel Out of Core Applications on Beowulf Workstations, *Proceedings of the 1998 IEEE Aerospace Conference*, pages 355-365, March, 1998.