Tracing Performance of MPI-I/O with PVFS2: A Case Study of Optimization

Yuichi TSUJITA [a], Julian KUNKEL [b], Stephan KREMPEL [b], Thomas LUDWIG [c]
[a] *Kinki University*
1 Umenobe, Takaya, Higashi-Hiroshima, Hiroshima 739-2116, Japan
[b] *Ruprecht-Karls-Universität, Heidelberg*
Im Neuenheimer Feld 348, 69120 Heidelberg, Germany
[c] *Universität Hamburg & German Climate Computing Centre (DKRZ)*
Bundesstraße 45a, 20146 Hamburg, Germany

Abstract. Parallel computing manages huge amounts of data due to a dramatic increase in computing scale. The parallel file system PVFS version 2 (PVFS2) realizes a scalable file system for such huge data on a cluster system. Although several MPI tracing tools can check the behavior of MPI functions, tracing PVFS server activities has not been available. Hence, we have missed chances to optimize MPI applications regarding PVFS server activities although effective usage of limited resources is important even in PVFS servers. An off-line performance analysis tool named PIOviz traces both MPI-I/O calls and associated PVFS server activities to assist optimization for MPI applications. Besides, tracing statistical values of PVFS servers such as CPU usage and PVFS internal statistics assists optimization of MPI applications. In this paper, we demonstrate two performance evaluation tests of the HPIO benchmark, and carry out off-line analysis by using PIOviz. The evaluation shows effectiveness of PIOviz in detecting bottlenecks of MPI-I/O.

Keywords. cluster computing, tracing tool, MPI-I/O, PVFS, performance optimization

Introduction

The Parallel Virtual File System (PVFS) [1] was developed to realize parallel file systems on a cluster environment by deploying metadata and data servers on cluster nodes. Each data server has a data storage space to store chunks of striped data, while metadata servers hold information of a cluster-wide consistent name space and file distributions associated with the striped data. The file system is accessible from POSIX I/O and MPI-I/O APIs.

In parallel computing, there is a variety of I/O patterns in both contiguous and non-contiguous data accesses provided by the MPI-I/O. One of the implementations is the ROMIO in MPICH2 [2]. In independent operations with a non-contiguous access pattern, each client process might carry out a large number of small data accesses. In contrast, collective operations adopt the two-phase I/O protocol [3] to improve performance. Data exchanges and synchronizations among client MPI processes are carried out to make large contiguous data accesses on file domains.

Performance optimization is carried out by measuring throughput for example. However, it is difficult to examine reasons for inefficient operation. Variety of data access patterns such as non-contiguous accesses also makes optimization rather difficult. Furthermore, utilization of a parallel file system such as PVFS brings complexity in optimization. However, there is not sufficient research work to reveal effects of PVFS activities under complex I/O patterns. PIOviz [4] is developed to assist such application optimization. In this paper, several measurements for PVFS server statistics were carried out by using the HPIO benchmark [5] with PIOviz.

The rest of this paper is organized as follows. Section 1 provides brief overview of related work. Section 2 describes functionality of PIOviz, followed by short explanations of MPI-I/O access patterns in the HPIO benchmark in section 3. Section 4 discusses performance evaluation of MPI-I/O operations using the HPIO benchmark with the help of PIOviz. Conclusions and future work are discussed in Section 5.

1. Related Work

Jumpshot [6] and Vampir [7] are well-known tracing tools for MPI applications. Jumpshot works in cooperation with the MPICH2 library. It supports analysis of MPI functions for data communications and parallel I/Os. Vampir visualizes MPI calls and records performance data according to the Open Trace Format [8]. Recently it supports MPI-I/O tracing [7]. Another framework SCALASCA [9] supports runtime summarization of measurements during execution and event trace collection for postmortem trace analysis. TAU [10] provides robust, flexible, and portable tools for tracing and visualization of applications. It can also generate trace information which can be displayed with the Vampir. The Paraver toolset [11] provides fruitful analysis tools such as hardware counter and system activity monitor in addition to profiler for applications. However, these tools do not support tracing activities within parallel file systems such as PVFS.

Our project PIOviz is an off-line trace-based environment for MPI operations. It extends existing tracing implementation by incorporating PVFS instrumentation to support tracing PVFS activities in conjunction with MPI-I/O calls. As data format of the trace file is based on SLOG2 format [6], a user can analyze trace information with Jumpshot.

2. PIOviz

PIOviz traces MPI calls and typical PVFS server activities such as network communications (effected by the BMI layer) and disk accesses (Trove layer) in conjunction with MPI-I/O calls. In addition, it also collects statistics of CPU usage and PVFS internal statistics [12]. As PIOviz uses standard PMPI APIs, an application user can utilize it without modification in source codes. Once a user executes a program, trace files are generated by PIOviz in both client and server sides. Correspondence of trace information between client and server sides is analyzed, and then a combined trace file is generated from the trace files through several merging stages. A user can analyze all the MPI and MPI-I/O calls in conjunction with associated PVFS activities with Jumpshot.

3. HPIO Benchmark

The MPI-I/O benchmark HPIO reveals performance statistics about non-contiguous data access patterns in collective and independent operations. For the non-contiguous data

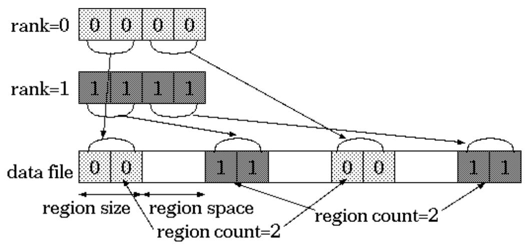


Figure 1. Example of a derived data type in the HPIO benchmark

accesses, derived data types are created with an ensemble of region size, region count, and region space, where region stands for a data area. Figure 1 illustrates an example of a data pattern for two processes. Here we assume that data is stored in a memory contiguously and in a data file non-contiguously. Gaps between data regions are specified by region space of this benchmark. According to a file view created by a derived data type, each client process accesses the data file.

4. Performance evaluation

We used a cluster of 9 PC nodes for our performance measurement. All nodes, DELL PowerEdge 830 PCs, were interconnected via a Gigabit Ethernet switch, 3Com SuperStack 3 3824. Table 1 summarizes specifications of the cluster system. Since a 64 bit operating system was used, all the libraries in this test were built as 64 bit applications. A PVFS file system was prepared on five nodes. A head node was configured as a metadata server and four computation nodes as data servers. Data storage on each data server was prepared on an ext3 file system configured on a RAID-1 volume. The remainder of the nodes (four nodes) was used for client MPI processes. PIOviz incorporated MPICH2 version 1.0.5p4 and PVFS version 2.6.2, and it was used to build the HPIO.

In the following experiments, we used a contiguous access pattern in a memory (client MPI application side) and a non-contiguous access pattern on a PVFS file system. We arranged short and long data region tests with 32 Bytes and 1024 Bytes for region size, respectively. In both cases, region count was 32768 and region space was

Table 1. Specifications of the test cluster system

CPU	Intel Pentium-D 3.2 GHz, 2×2 MBytes L2 cache	
Chipset	Intel E7230	
Memory	1.5 GBytes DDR2 667 SDRAM	
Disk system (system)	Western Digital WD1600JS-75N (available in each node)	
Disk system (storage space)	RAID-1 (data nodes)	RAID controller: 3ware Escalade 9550SX-4LP
		Disk: 2 × Seagate Barracuda 7200.9 160 GBytes
Network I/F	Broadcom NetXtreme BCM5721 (on-board)	
Linux kernel	2.6.9-42.ELsmp (CentOS 4.4 for x86_64)	
C compiler	gcc version 3.4.6	
Ethernet driver	Broadcom tg3 version 3.52	

128 Bytes. In the short data region test, total data size was 4 MBytes (32 Bytes $\times$ 32768 $\times$ 4 processes) and total file size including data gaps was about 20 MBytes ((32 Bytes + 128 Bytes) $\times$ 32768 $\times$ 4 processes - 128 Bytes). On the other hand, total data size in the long data region test was 128 MBytes (1024 Bytes $\times$ 32768 $\times$ 4 processes) and total file size including data gaps was about 144 MBytes ((1024 Bytes + 128 Bytes) $\times$ 32768 $\times$ 4 processes - 128 Bytes).

In this chapter, we show (1) an evaluation between independent and collective operations and (2) an evaluation in terms of sizes for collective buffer (hereafter CB) in the two phase I/O. In both cases, mean values of traced statistics obtained by PIOviz were calculated through 10 iterations. For reference, mean values of throughput (including `MPI_File_sync()`) calculated by the HPIO are included. Note that we set the HPIO to compute the mean throughput values by excluding the highest and lowest value.

4.1. Independent and collective I/O

`MPI_File_write()` and `MPI_File_write_all()` were used in the independent and collective cases, respectively. Here `MPI_File_write()` accessed only data regions, while `MPI_File_write_all()` accessed data regions and data gaps because of the two-phase I/O protocol. Table 2 shows mean statistical values of four data servers. Note that BMI load is higher than Trove load if the network is a bottleneck due to many network transfer, and vice versa if there are many disk I/Os. From a user perspective, utilization of available server resources is important, thus a high load is preferable. However, evaluation of the load values together is essential to assess efficient utilization. In the default configuration, a PVFS server issues an I/O operation for a maximum of 256 KBytes. In our case, the write-behind of the Linux kernel can combine multiple small requests into one. It is desirable to issue as much large I/O operations as possible to the kernel to allow efficient write-back. Consequently, it is also important to check Trove size as close to the maximum size.

We can see that BMI load in collective operations is higher than that in independent operations. This was due to the two-phase I/O protocol. On the other hand, Trove load and Trove access in independent operations are higher than those in collective operations due to large number of small data accesses in independent operations.

Table 2. Statistics of data servers obtained by PIOviz in collective (C) and independent (I) I/O operations of the HPIO benchmark

Region size (Bytes)	I/O mode	BMI load (ops/s)	Trove load (ops/s)	Trove access (ops)	Trove size (B/op)	CPU usage (%)	Throughput (MB/s)
32	C	0.425	10	800 (read, write)	262137.6 (read,write)	2.03	15.842
	I	0	49.9	30719 (write)	1365.4 (write)	11.55	5.352
1024	C	0.875	10.3	5760 (read,write)	262143.1 (read,write)	5.02	71.169
	I	0.3	59.4	81919 (write)	16384 (write)	15.135	92.954

In the case of 32 Bytes in a region size, collective operation outperformed independent one. By focusing on CPU usage, the value in the independent case is higher than that in the collective one. This was caused by many small Trove accesses in the independent case. So, the independent case was not efficient in this configuration from the viewpoint of throughput and CPU usage.

On the other hand, independent operation outperformed collective operation with region size of 1024 Bytes. In the collective case, we found inefficient operations of the two-phase I/O in screenshots of trace files generated by PIOviz. Later, this will be discussed in 4.2. Although higher throughput is preferable, users need to pay attention to CPU usage in the independent mode. The high Trove load value is caused by many small I/O operations. As a result, CPU usage is higher than that in the collective mode. If many users share a PVFS file system, performance of independent case might be degraded due to short of CPU resources. As there is a trade-off between throughput and other factors such as CPU usage due to limited computing resources, PIOviz can help to select preferable operation pattern.

4.2. Effects of collective buffer size in the two-phase I/O

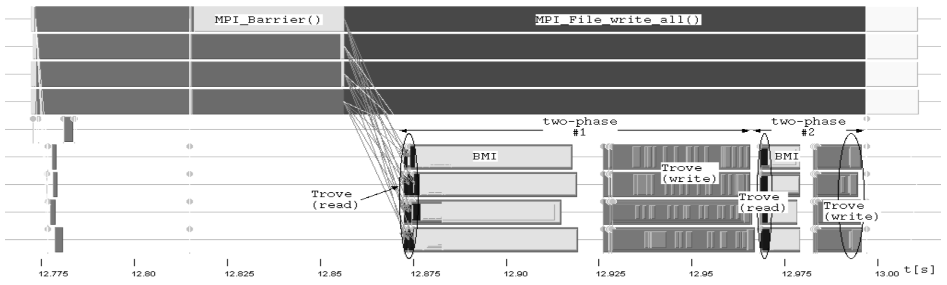
We also measured statistical values of the PVFS servers in `MPI_File_write_all()` with changing CB size. Note that default CB size of the ROMIO is 4 MBytes. Table 3 shows measured values from trace files in the case of 32 Bytes in region size. In this test, 16 MBytes was desirable for CB size regarding small CPU usage and higher throughput values. Here, a CB with this size could store the whole data ($\sim$ 5 MBytes per process). Trove load, Trove access, and Trove size are almost constant with more than 4 MBytes for CB size. This means that a CB with 4 MBytes was sufficient for PVFS servers to achieve its peak performance. However, throughput in the case of 4 MBytes was smaller than that of 16 MBytes due to larger overhead caused by the two-phase I/O.

Measured throughput values sometimes showed high variance in 10 iterations runs. Four screenshots in Figure 2 show what was going on in client MPI applications and PVFS servers. In every screenshot, the upper 4 time-lines show activities of client MPI applications, while the first line in the lower 5 lines shows activities of a PVFS metadata server and the rest of them shows those of PVFS data servers.

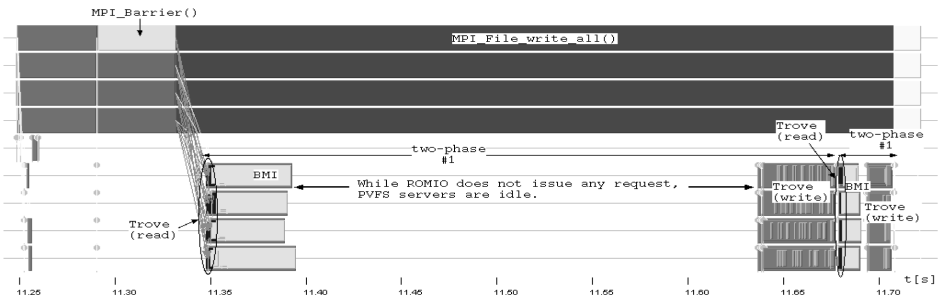
Fig. 2 (a) shows an effective iteration in the case of 4 MBytes in CB size. We can see consecutive operation by Trove read, BMI, and Trove write on every PVFS data server twice for the two-phase I/O protocol. When the MPI I/O function was called, data on a file domain was read by Trove read. The data was transferred to a CB of each client

Table 3. Statistics of data servers with different collective buffer (CB) size in collective operations with region size of 32 Bytes

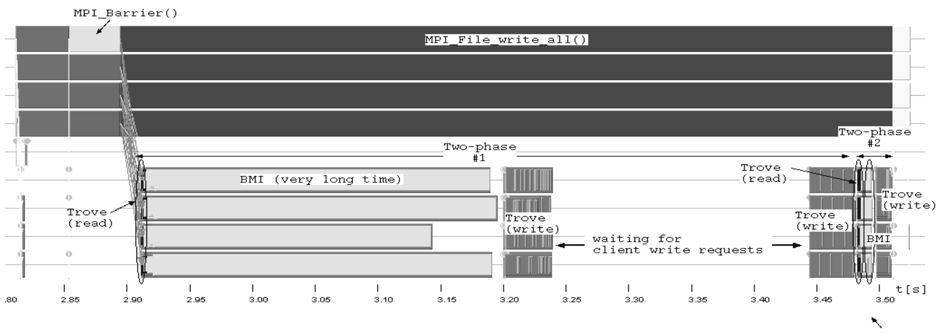
CB size (MBytes)	BMI load (ops/s)	Trove load (ops/s)	Trove access (ops)	Trove size (B/op)	CPU usage (%)	Throughput (MB/s)
0.5	0.2	10.275	1600 (read,write)	131068.8 (read,write)	2.605	14.962
4	0.425	10	800 (read,write)	262137.6 (read,write)	2.0325	15.842
16	0.5	10.5	800 (read,write)	262137.6 (read,write)	2.0675	23.571



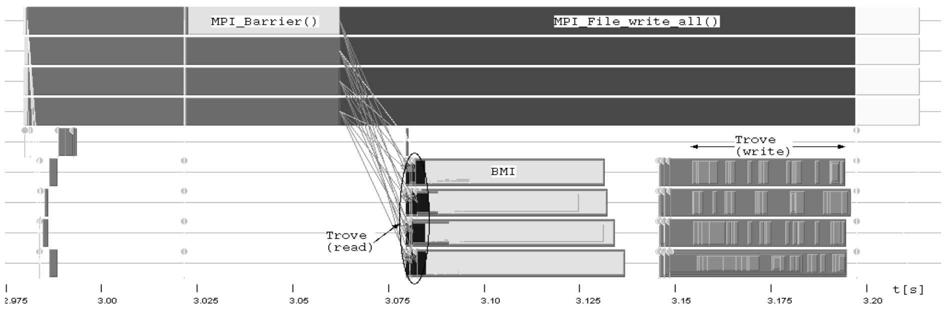
(a) efficient case (CB size: 4 MBytes)



(b) inefficient in client (CB size: 4 MBytes)



(c) inefficient in network transfer (CB size: 4 MBytes)



(d) efficient case (CB size:16 MBytes)

Figure 2. Screenshots of one iteration in the HPIO benchmark

Table 4. Statistics of data servers with different CB size in collective operations with region size of 1024 Bytes

CB size (MBytes)	BMI load (ops/s)	Trove load (ops/s)	Trove access (ops)	Trove size (B/op)	CPU usage (%)	Throughput (MB/s)
0.5	0.4	10.475	11520 (read,write)	131071.6 (read,write)	5.3475	46.955
4	0.875	10.325	5760 (read,write)	262143.1 (read,write)	5.02	71.169
16	2.025	10.7	5760 (read,write)	262143.1 (read,write)	5.355	96.138
64	2.6	10.5	5760 (read,write)	262143.1 (read,write)	5.38	114.634

process by using BMI. Later on, data to be written was exchanged among client processes and overwritten in each CB non-contiguously. Finally data in each CB was transferred to a PVFS data server by using BMI. Then it was written back to an assigned file domain by using Trove write.

Figs. 2 (b), and (c) show typical inefficient cases. In the case (b), PVFS servers were idle for a long time ($\sim$ 240 ms) because ROMIO did not issue any request. Note that the waiting time was about 7 ms in Fig. 2 (a). It is considered that network transfer among client applications took a long time. The case (c) shows long network transfer by BMI between client applications and PVFS servers. Here, there could be inefficient network transfer or inefficient MPI communications inside ROMIO. From both examples, it is considered that MPI operations inside ROMIO had some inefficient parts. However we could not examine the network transfer and MPI calls inside ROMIO in detail because the PIOviz could not trace them this time.

In the case of 16 MBytes in CB size, the number of the consecutive operation in the two-phase I/O was one due to enough CB size as shown in Fig. 2 (d). Thus, the screen-shots from trace files of PIOviz are useful for off-line analysis of bottleneck detection.

Table 4 summarizes measured statistics with region size of 1024 Bytes. A 64 MBytes CB provided the best throughput. Here, BMI load values raised up with an increase in CB size. This means that network throughput between client processes and PVFS servers raised up with an increase in CB size. However, Trove load, Trove access, and Trove size are constant with more than 4 MBytes for CB size. A 4 MBytes CB is considered to be enough to achieve peak performance of PVFS servers. By increasing CB size to 64 MBytes, each client process could store whole data ($\sim$ 36 MBytes per process) in a CB. As a result, throughput was improved.

5. Summary

MPI-I/O and a PVFS file system provide scalable I/O operations on a cluster system. In order to optimize resource usage of MPI applications, it is helpful to trace relevant performance data. This is composed of not only application related information but also other statistics like e.g. CPU and network usages. PIOviz traces PVFS server activities in conjunction with MPI-I/O calls. Besides, it traces performance statistics and PVFS internal statistics. We demonstrated how PIOviz was used for application optimization. In our experiments of the HPIO benchmark, collective operations were better than independent ones with short data regions regarding both throughput and CPU usage. Independent operations outperformed collective ones with longer data regions. However, we should pay attention to CPU usage because CPU utilization in the independent operations was higher than that in the collective ones.

We also demonstrated tracing of collective operations in terms of collective buffer size in the two-phase I/O protocol. Obviously we could obtain higher throughput by increasing the buffer size. Other statistics such as CPU usage, Trove load, and the number of Trove accesses informed by PIOviz were useful in application optimization. PIOviz is expected to be helpful for tuning MPI applications with paying much attention to effective resource usage of PVFS servers. In our experiments, we showed examples which have inefficient aspects in client applications or network communications. However, we could not find reasons for the inefficient operation because PIOviz could not trace statistics of network transfer and MPI communications inside ROMIO in client applications at this moment. As a future work, implementation of functions to trace performance statistics and MPI calls used inside ROMIO is considered.

Acknowledgments

This research was partially supported by MEXT, Grant-in-Aid for Young Scientists (B), 21700063. The authors would like to thank Olga Mordvinova for her useful advice to improve this paper.

References

[1] PVFS2. `http://www.pvfs.org/pvfs2/`.

[2] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A high-performance, portable implementation of the MPI Message-Passing Interface standard," *Parallel Computing*, vol. 22, no. 6, pp. 789–828, 1996.

[3] R. Thakur, W. Gropp, and E. Lusk, "Optimizing noncontiguous accesses in MPI-IO," *Parallel Computing*, vol. 28, no. 1, pp. 83–105, 2002.

[4] T. Ludwig, S. Krempel, M. Kuhn, J. M. Kunkel, and C. Lohse, "Analysis of the MPI-IO optimization levels with the PIOViz jumpshot enhancement," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, vol. 4757 of *LNCS*, pp. 213–222, Springer, 2007.

[5] A. Ching, A. Choudhary, W. keng Liao, L. Ward, and N. Pundit, "Evaluating I/O characteristics and methods for storing structured scientific data," in *Proceedings 20th IEEE International Parallel and Distributed Processing Symposium*, p. 49, IEEE Computer Society, April 2006.

[6] A. Chan, W. Gropp, and E. Lusk, "An efficient format for nearly constant-time access to arbitrary time intervals in large trace files," *Scientific Programming*, vol. 16, no. 2-3, pp. 155–165, 2008.

[7] Vampir. `http://www.vampir.eu/`.

[8] A. Knüpfer, R. Brendel, H. Brunst, H. Mix, and W. E. Nagel, "Introducing the open trace format (OTF)," in *Computational Science - ICCS 2006, Part II*, vol. 3992 of *LNCS*, pp. 526–533, Springer, 2006.

[9] Z. Szebenyi, B. J. N. Wylie, and F. Wolf, "SCALASCA parallel performance analyses of SPEC MPI2007 applications," in *Performance Evaluation: Metrics, Models and Benchmarks*, vol. 5119 of *LNCS*, pp. 99–123, Springer, 2008.

[10] S. S. Shende and A. D. Malony, "The tau parallel performance system," *The International Journal of High Performance Computing Applications*, vol. 20, pp. 287–311, Summer 2006.

[11] J. Labarta, J. Giménez, E. Martínez, P. González, H. Servat, G. Llort, and X. Aguilar, "Scalability of visualization and tracing tools," in *Proceedings of the International Conference ParCo 2005*, vol. 33 of *NIC Series*, pp. 869–876, John von Neumann Institute for Computing, Jülich, 2006.

[12] J. M. Kunkel and T. Ludwig, "Bottleneck detection in parallel file systems with trace-based performance monitoring," in *Euro-Par 2008 - Parallel Processing*, vol. 5168 of *LNCS*, pp. 212–221, Springer, 2008.