PVFS를 위한 I/O Tracer 설계 및 구현

조혜영°, 차광호, 김성호, 이상동
한국과학기술정보연구원 슈퍼컴퓨팅센터
e-mail:{chohy°, khocha, sungho, sdlee}@kisti.re.kr

Design and Implementation of I/O Tracer for PVFS

Hyeyoung Cho°, Kwangho Cha, Sungho Kim, SangDong Lee
Supercomputing Center
Korea Institute of Science and Technology Information

요 약

사용자 프로그램의 I/O 패턴을 분석하거나 파일 시스템의 워크로드를 보다 정확하게 분석하기 위해서 실제 가동중인 파일 시스템의 동적 I/O 로그를 확보하기 위한 연구들이 많이 진행되어 왔다. 그러나 대량의 I/O 트렌젝션(transcation)이 처리되는 파일 시스템에서 동적 I/O 로그를 확보하는 일은 시스템의 부하와 막대한 데이터량 때문에 한계가 많다. 특히 다수의 이용자가 사용하는 대용량 분산/병렬 파일 시스템에서의 I/O Tracing은 로컬 파일 시스템에서 I/O Tracing에 비해 더욱 복잡하고 오버헤드가 크다. 본 논문에서는 기존의 파일 시스템 로깅 방법들을 알아보고, 클러스터 시스템에서 널리 이용되고 있는 분산 파일 시스템인 PVFS(Parallel Virtual File System)에서 동적 I/O 연산들의 로그를 생성할 수 있는 로깅 시스템을 제안하고 설계하였다.

1. 서론

다양한 파일 시스템의 성능을 분석하고 모니터링하기 위해여 파일 시스템에 로그를 남기고, 파일 시스템의 동작을 추적(tracing)하려는 연구들이 많이 진행되어왔다[1,2,3]. 실제 파일 시스템의 동작을 로깅함으로써 사용자 프로그램의 I/O 패턴을 분석하고 사용자 프로그램을 보다 최적화 할 수 있으며, 파일 시스템을 이용하는 사용자의 이용 패턴에 따라 파일 시스템을 최적화하여 파일 시스템의 성능을 최대화 할 수 있다. 또한 파일 시스템을 추적(trace)하는 기술은 실제 파일 시스템을 설계하고 디버깅할 때 필요한 기술이며, 이렇게 로깅된 정보를 기반으로 파일 시스템의 워크로드를 분석하고 미래의 요구사항도 예측함으로써 새로운 시스템을 설계할 때 중요 자료로 이용된다.

그러나 대량의 I/O 트렌젝션(transcation)이 처리되는 파일 시스템에서 동적 I/O 로그를 확보하는 일은 시스템의 부하와 막대한 데이터량 때문에 한계가 많다[4]. 특히 고속 네트워크와 시스템 기술의 발전과 함께 병렬 컴퓨터 및 클러스터 시스템의 사용이 증가하면서 관심이 높아지고 있는 분산 및 병렬 파일 시스템에서의 I/O 연산들의 로그를 생성하는 일은 로컬 파일 시스템(local file system)에서의 로깅보다 더욱 복잡하고 오버헤드도 크다.

본 논문에서는 기존의 파일 시스템들의 로깅 방법들을 알아보고, 클러스터 시스템에서 널리 이용되고 있는 파일 시스템인 PVFS(Parallel Virtual File System)에서 동적 I/O 연산들의 로그를 생성할 수 있는 로깅 시스템을 제안하고 설계하였다.

본 논문의 구성은 다음과 같다. 2장에서는 소규모 클러스터에서 널리 사용되고 있는 오픈 소스 파일 시스템인 PVFS(Parallel Virtual File System)에 대해서 살펴보고, 3장에서는 기존의 I/O 로깅 기법들을 비교 설명한다. 4장에서는 PVFS에서 로깅 시스템을 제안, 설계하고 5장에서는 구현 환경, 생성된 로그 및 성능 측정 결과를 기술한다. 마지막으로 5장에서는 결론에 대하여 기술한다.

2. PVFS(Parallel Virtual File System)

PVFS(Parallel Virtual File System)은 Clemson 대학과 Argonne National Laboratory에서 개발되어 소형 클러스터 시스템에 많이 이용되는 분산 파일 시스템이다. 동시에 대량의 I/O 접근(access)가 일어나는 병렬 어플리케이션을 위해 데이터를 쪼개어서 다수의 서버에 저장하는 방식으로 고성능을 제공한다[5,6].

그림 1에 PVFS의 전체 구조를 나타내었다. PVFS는 기능적으로 메타 서버, I/O 서버, 계산 노드로 구분할 수 있다. 메타 서버는 파일 권한이라든지 실제 파일 데이터가 존재하는 위치 등 메타데이터를 관리한다. I/O 서버는 실제 파일 데이터를 읽고, 쓰는 것을 담당한다. PVFS는 성능 향상을 위해 PVFS를 사용하는 어플리케이션이 메터데이터 정보를 얻은 후 직접 I/O 서버에 접속하여 파일 데이터를 얻는 구조를 가진다. 계산 노드는 어플리케이션이 PVFS Client를 통해 PVFS를 접근할 수 있도록 인터페이스를 제공한다[7,8,9].

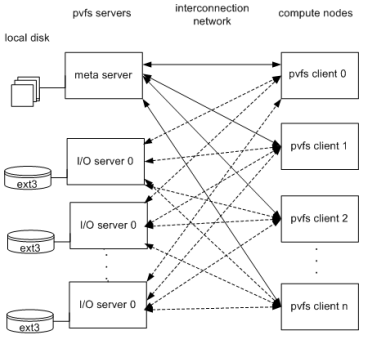

그림 1. PVFS의 구조

3. I/O Logging 기법

응용 소프트웨어의 I/O 패턴을 분석하거나 파일 시스템을 최적화하여 성능을 향상시키기 위해서 I/O 연산(operation)에 대해 로그를 생성하려는 연구들이 다양하게 진행되었다. 기존의 연구들을 살펴보면 system call을 hooking하는 방법을 이용한 연구들이 많이 있었으며, system call을 hooking하는 경우 overhead를 줄이기 위한 연구들이 진행되었다[1,2,3].

MPI(Message Passing Interface) 프로그램에 대해서 로그를 남기고 프로그램을 분석하기 위한 방법으로는 MPE(Multi-Processing Environment)에서 제공하는 로깅 도구를 이용하는 방법이 있다[10,11]. MPE 라이브러리에는 순차적으로 편의성이 추가된 ALOG, CLOG, SLOG, SLOG-2 등 다양한 로그 파일 포맷을 제공한다. 그러나 이러한 MPE에서 제공하는 로깅 기법은 사용자가 컴파일 할 때 옵션을 주어 컴파일하는 방법으로 사용자별로 하나의 응용프로그램을 대상으로 로그를 남기는 것은 가능하지만, 파일 시스템 전체에 대한 I/O 로그를 생성할 수 없어, 파일 시스템 전체의 I/O 워크로드를 분석하기 위한 방법으로는 한계가 있다[10].

현재 PVFS는 I/O 패턴 분석과 같은 동적인 로그 생성을 지원하지는 않고 있다. 다만 디버깅을 목적으로 하는 디버거를 부수적으로 제공하고 있다. 그러나 이러한 방법은 실시간에 파일에 기술하는 형태이기 때문에 5장에서 제시한 바와 같이 오버헤드가 크다.

4. 시스템 설계

PVFS에서의 I/O 연산이 처리되는 과정의 요소들을 그림 2에 나타내었다. PVFS I/O 연산 절차도에서 보듯이 I/O request는 다양한 단계를 거치는데, 파일 I/O의 workload에 대한 로그를 어떤 단계에서 남기느냐에 따라 장단점이 있다. VFS에서 로그를 남기는 경우, 명령어(command)로 요청된 요구(request)와 단일 시스템상의 싱글 프로그램은 로그는 생성할 수 있지만, MPI 응용 프로그램은 로그를 남길 수 없다. 또한 VFS는 커널 영역이기 때문에 커널 영역에서 사용자 영역으로 정보를 전달해주는 overhead를 최소로 하여 workload에 영향을 주지 않도록 하는 방법이 필요하다. 이와 다르게 MPICH2상에서 라이브러리에 로그를 남길 경우, 상위 프로그램에서 명령어로 요청된 요구나 C 프로그램에 대해서 로그를 남길 수 없다는 단점이 있다. 본 논문에서는 명령어로 요청된 request, C 프로그램, MPI 프로그램의 모든 I/O 연산을 수용하기 위해서 PVFS client에서 로그를 남기도록 설계하였다.

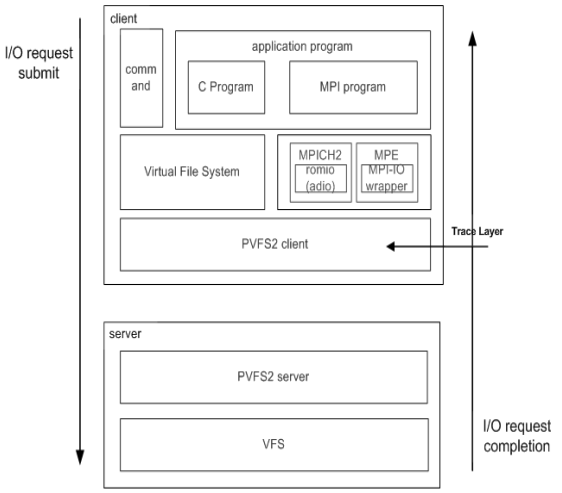

그림 2. PVFS I/O 연산 절차도

그림 3에 전체 구조도를 나타내었다. 메모리에 있는 I/O 요구(request)에 대한 내용을 실제 디스크로 저장하기까지 오버헤드를 분석해보면 로그 정보를 원하는 자료구조로 포맷팅(formatting) 하는 시간과 디스크에 저장하는 시간으로 나눌 수 있다. 그 중에서 특히 workload 로그로 인한 추가 disk I/O 오버헤드를 줄이기 위해 그림 3과 같이 공유 메모리(shared memory)를 이용하여 Logging 클라이언트에 trace data를 전달한다. 이에 따라 PVFS 클라이언트는 I/O request에 대해 이후 처리를 계속 수행할 수 있게 됨으로써, 시스템 성능 분석에 가장 중요한 요소인 지연 시간을 최소화하도록 하였다. Logging 클라이언트는 공유 메모리를 모니터링 하면서 주기적으로 데이터를 Collection 서버로 송신한다. Collection 서버의 Logging Server는 이 데이터를 수신받아 disk에 저장한다.

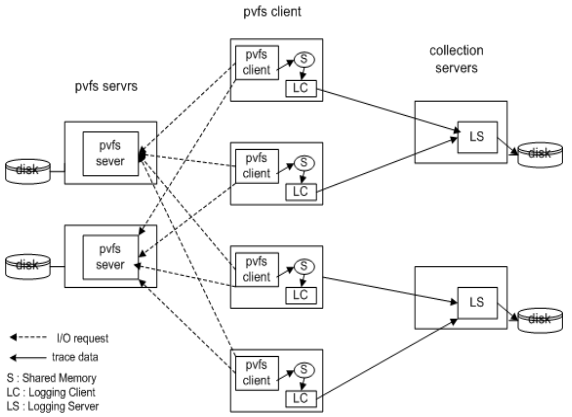

그림 3. 전체 구성도

본 시스템은 I/O request를 원하는 자료구조로 포맷팅하고 디스크에 쓰는 시간을 Logging 서버에 이관하고, PVFS 클라이언트에서는 trace data를 공유 메모리에 저장하고 바로 다음 request를 처리하여, 지연 시간을 최소화하였다. 이러한 시스템 구조를 통해 대량의 I/O 트랜잭션(transaction)의 workload를 효율적으로 로깅할 수 있으며, 또한 로깅 절차에서 disk 저장에 따른 추가 overhead를 구조적으로 최소화할 수 있어 대용량의 분산 파일 시스템의 I/O 워크로드 분석이 가능하도록 하였다.

5. 구현 및 실험 결과

구현은 구축이 용이한 소규모 클러스터 시스템에서 개발되었다. 표 1은 구현에 사용된 시스템의 하드웨어 및 소프트웨어 구성을 보여준다.

표 1. 시스템 사양

OS	Linux 2.6.18
PVFS	PVFS 2.6.1
CPU	Intel Xeon 2.00GHz
Memory	2GB
HDD	40GB SCSI
Network	Gigabit Ethernet

표 2는 시스템에서 수집된 I/O 오퍼레이션 로그를 기록한 것이다. 표 2와 같이 request 기간, handle 번호, 오퍼레이션 타입, 오퍼레이션 수행 데이터 길이, 오프셋, 사용자 ID, 그룹 ID 등이 기록된다.

그림 4과 5에 본 시스템과 PVFS에서 제공되는 로깅 기법의 오버헤드를 비교한 결과를 보였다. 오버헤드를 측정하기 위해 병렬 파일 시스템의 성능을 측정하는데 널리 이용되고 있는 LLNL(Lawrence Livermore National Laboratory)의 IOR 벤치마크를 이용하였다[12].

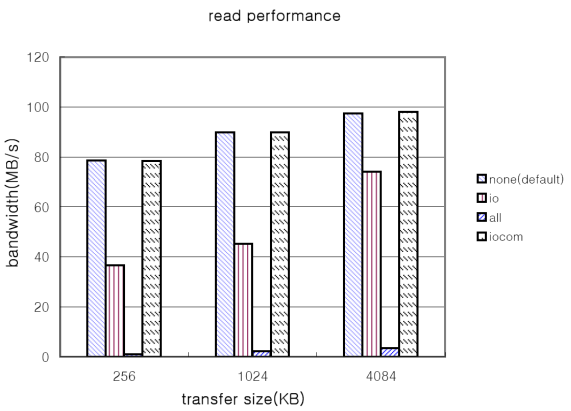

그림 4. read 성능

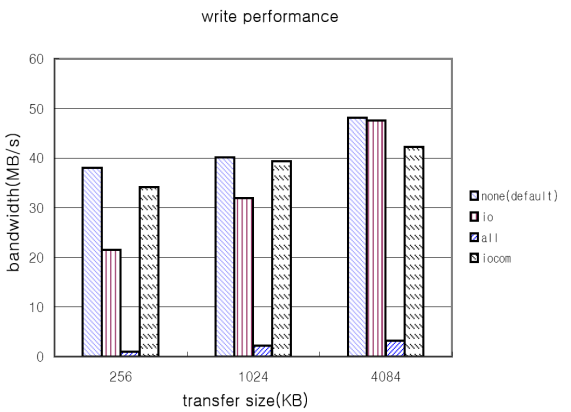

그림 5. write 성능

표 2 테스트 결과 생성된 로그 예제

OPLOG: time[1213067665]:[450805], handle [1048574], operation [1], offset [81920] len [4096] uid [500] gid [500]
OPLOG: time[1213067665]:[451302], handle [1048574], operation [1], offset [77824] len [4096] uid [500] gid [500]
OPLOG: time[1213067665]:[451806], handle [1048574], operation [1], offset [36864] len [4096] uid [500] gid [500]
OPLOG: time[1213067665]:[452508], handle [1048574], operation [1], offset [40960] len [4096] uid [500] gid [500]
OPLOG: time[1213067665]:[476619], handle [1048569], operation [2], offset [0] len [262144] uid [500] gid [500]
OPLOG: time[1213067665]:[484275], handle [1048569], operation [2], offset [262144] len [262144] uid [500] gid [500]
OPLOG: time[1213067665]:[484893], handle [1048574], operation [1], offset [221184] len [4096] uid [500] gid [500]
OPLOG: time[1213067665]:[488735], handle [1048574], operation [1], offset [552960] len [4096] uid [500] gid [500]
OPLOG: time[1213067665]:[490234], handle [1048568], operation [2], offset [0] len [262144] uid [0] gid [0]

Read 연산의 경우, PVFS에서 로그 없이 수행했을 때와 io 옵션 사용했을 때를 비교해 보면 transfer size에 따라 46~76% 성능 차이를 보였다. all 옵션인 경우는 성능이 너무 낮아서 작은 transfer size에서는 성능 수치를 측정하기가 힘든 정도로 오버헤드가 컸다. 이에 반에 본 논문에서 설계, 구현된 시스템의 경우 기본 PVFS와 99%내외로 비슷한 성능을 보였다. Write 연산의 경우도 transfer size가 4MB일 때를 제외하고 비슷한 결과를 보여주었다. 이와 같이 오버헤드가 작은 것은 오버헤드가 작은 것은 공유메모리를 이용한 빠른 서비스 복귀로 지연 시간을 단축하고, disk 쓰기에서 오는 오버헤드를 Collection 서버로 구조적으로 분산시켰기 때문이다. 따라서 본 시스템은 I/O 트랜잭션이 많은 대규모 분산 파일 시스템에서 I/O 추척(trace)하기에 기존의 PVFS의 추적에 비해 용이하다고 할 수 있다.

6. 결론

본 논문에서는 기존 파일 시스템의 로깅 방법들을 알아보고, 클러스터 시스템에서 널리 이용되고 있는 분산 파일 시스템인 PVFS(Parallel Virtual File System)에서 동적 I/O 연산들의 로깅을 남길 수 있는 로깅 시스템을 제안하고 설계하였다. 기존의 I/O Tracer들이 로컬 파일 시스템의 로그만을 기록하는데 집중하는데 비해, 본 시스템은 구조적으로 분산 파일 시스템의 로그를 처리할 수 있도록 설계하였다. 공유메모리를 사용하여 원하는 자료 구조로 포맷하는 시간과 디스크에 저장하는 시간을 최소화하여 대량의 I/O 트랜잭션(transaction)을 효율적으로 로깅할 수 있도록 설계하였다. 앞으로 본 설계를 바탕으로 실제 가동중인 분산 파일 시스템에서 동적 I/O 정보를 수집하여 파일 시스템의 동적 I/O 연산에 대한 워크로드 분석도 진행 할 계획이다.

참고문헌

[1] John K. Ousterhout, Hervg Da Costa, David Harrison, John A. Kunze, Mike Kupfer, and James G. Thompson, “A Trace-Driven Analysis of the UNIX 4.2 BSD File System,” 1985.

[2] Drew Roselli, Jacob R. Lorch, and Thomas E. Anderson, "A comparison of file system workloads," Proc. of USENIX Annual Technical Conference, pp. 41~54, 2000.

[3] Akshat Aranya, Charles P. Wright, and Erez Zadok, “Tracefs: A File System to Trace Them All,” FAST 2004.

[4] L. Mummert and M. Satyanarayana, "Long Term Distributed File Reference Tracing: Implementation and Experience," Software – Practice and Experience 26(6), p. 705-736, 1994.

[5] John M. May, “Parallel I/O for High Performance Computing,” Morgan Kaufmann, 2000.

[6] W.B. Ligon III, and R.B.Ross, "Implementation and performance of a parallel file system for high per-formance distributed applications," Proc. of 5th IEEE International Symposium on High Performance Distributed Computing, pp 471 ~ 480, 1996.

[7] Ibrahim F. Haddad inSysAdmin, “PVFS: A Parallel Virtual File System for Linux Cluster,” Linux Journal, 2000.

[8] Parallel Virtual File System version 2, http://www.pvfs.org/pvfs2

[9] Rob Latham, Neill Miller, Robert Ross, and Phil Carns, "A Next-Generation Parallel File System for Linux Clusters," Linux World, pp 56~59, Jan. 2004.

[10] Anthony Chan, William Gropp, and Ewing Lusk, “User’s Guide for MPE: Extensions for MPI Programs”, from MPICH2 web site, http://www.mcs.anl.gov/research/projects/mpich2

[11] MPICH2 web site, http://www.mcs.anl.gov /research/projects/mpich2

[12] Interleaved or Random (IOR) benchmarks. http://www.llnl.gov/icc/lc/siop/downloads/download.html