ISSN 2383-6318(Print) / ISSN 2383-6326(Online)
KIISE Transactions on Computing Practices, Vol. 22, No. 2, pp. 95-100, 2016. 2
http://dx.doi.org/10.5626/KTCP.2016.22.2.95

SSD 환경 아래에서 GlusterFS 성능 최적화

(Performance Optimization in GlusterFS on SSDs)

김 덕 상 † 엄 현 상 †† 염 헌 영 †††
(Deoksang Kim) (Hyeonsang Eom) (Heonyoung Yeom)

요 약 빅데이터, 클라우드 컴퓨팅 시대가 오면서 데이터 사용량이 점점 증가하고 있고 이러한 빅데이터를 신속히 처리하기 위한 시스템들이 개발되고 있다. 그 중 데이터를 저장하기 위한 시스템으로 분산 파일 시스템이 널리 사용되고 있다, 이러한 분산 파일 시스템 중에는 글러스터 파일 시스템(GlusterFS)이 있다. 또한 기술의 발달로 고성능 장비인 Nand flash SSD (Solid State Drive)의 가격이 하락함에 따라서 데이터센터로 도입이 증가되는 추세이다. 따라서 GlusterFS에서도 SSD를 도입하려고 하지만, GlusterFS는 하드디스크를 기반으로 설계되었기 때문에 SSD를 이용했을 시 구조적인 문제로 성능 저하가 발생하게 된다. 이러한 구조적인 문제점들에는 I/O-cache, Read-ahead, Write-behind Translator들이 있다. 랜덤 I/O에 장점이 있는 SSD에 맞지 않는 기능들을 제거함으로써, 4KB 랜덤 읽기의 경우 255%까지의 성능 향상 결과와, 64KB 랜덤 읽기의 경우 50%까지의 성능 향상 결과를 얻었다.

키워드: GlusterFS, SSD, 작은 랜덤 I/O, 성능 최적화

Abstract In the current era of big data and cloud computing, the amount of data utilized is increasing, and various systems to process this big data rapidly are being developed. A distributed file system is often used to store the data, and glusterFS is one of popular distributed file systems. As computer technology has advanced, NAND flash SSDs (Solid State Drives), which are high performance storage devices, have become cheaper. For this reason, datacenter operators attempt to use SSDs in their systems. They also try to install glusterFS on SSDs. However, since the glusterFS is designed to use HDDs (Hard Disk Drives), when SSDs are used instead of HDDs, the performance is degraded due to structural problems. The problems include the use of I/O-cache, Read-ahead, and Write-behind Translators. By removing these features that do not fit SSDs which are advantageous for random I/O, we have achieved performance improvements, by up to 255% in the case of 4KB random reads, and by up to 50% in the case of 64KB random reads.

Keywords: glusterFS, SSD, small random I/O, performance optimization

· 이 논문은 삼성전자 메모리 사업부의 지원을 받아 수행된 과제임
· 이 논문은 2014년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No. 2013R1A1A2064629)
· 이 논문은 2015년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임(No. R0190-15-2012, 빅데이터 처리 고도화 핵심기술개발 사업 총괄 및 고성능컴퓨팅 기술을 활용한 성능 가속화 기술 개발)
· 본 연구는 한국과학기술정보연구원에서 수행하는 사용자 환경개선을 위한 초고성능 플랫폼 개발(K-15-L01-C01) 사업의 위탁연구로 수행되었습니다.
· 이 논문은 2015 한국컴퓨터종합학술대회에서 'SSD 환경 아래에서 GlusterFS 성능 최적화'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 서울대학교 컴퓨터공학부
dskim@dcslab.snu.ac.kr
†† 정 회 원 : 서울대학교 컴퓨터공학부 교수(Seoul National Univ.)
hseom@cse.snu.ac.kr
(Corresponding author임)
††† 종신회원 : 서울대학교 컴퓨터공학부 교수
yeom@snu.ac.kr

논문접수 : 2015년 9월 9일
(Received 9 September 2015)
논문수정 : 2015년 11월 3일
(Revised 3 November 2015)
심사완료 : 2015년 11월 13일
(Accepted 13 November 2015)

정보과학회 컴퓨팅의 실제 논문지 제22권 제2호(2016. 2)

1. 서 론

최근 빅데이터(Big data), 클라우드(Cloud) 컴퓨팅의 시대가 오면서 기업 및 데이터 센터에서 사용 및 전송되는 데이터 양이 점점 더 증가하고 있고, 이러한 데이터를 수집하고 분석하는 일이 중요해지고 있다. 빅데이터를 처리하기 위해서는 기본적으로 많은 양의 데이터를 효율적으로 저장하고 관리하는 방법이 필요한데, 이러한 필요에 따라, 분산 파일시스템의 중요성이 높아지고 있다. 대표적으로 널리 사용되는 분산 파일 시스템 중에는 Ceph[1], 글러스터 파일 시스템(GlusterFS)[2,3] 등이 있고 수평적 확장(Scale-out)이 가능한 환경을 제공하고 있다. 또한 Nand flash SSD (Solid State Drive)의 가격이 점점 하락함에 따라 데이터 센터에서 SSD를 도입하려는 시도가 증가하는 추세이다. 이런 상황으로 인해, 분산 파일 시스템에도 SSD를 도입하려는 시도가 있어 왔다. 분산 파일 시스템에서 SSD를 사용하는 방법은 그림 1처럼 여러 가지가 있다. 서버의 하드디스크 전체를 SSD로 교체하는 방법, 서버 쪽에서 SSD를 캐시(Cache)로 이용하는 방법, 클라이언트 쪽에서 SSD를 캐시로 이용하는 방법, 그리고 서버 쪽에서 Hot data와 Cold data를 구분해서 Hot data는 SSD에 Cold data는 하드디스크에 저장하는 방법 등이 있다. 위의 방법들 중에 서버 전체의 하드디스크를 SSD로 교체하는 방법이 비용은 많이 들겠지만 분산 파일 시스템의 성능을 최대한 높일 수 있다. 하지만 분산 파일 시스템은 하드디스크를 기반으로 설계되었기 때문에, 단순히 하드디스크 대신에 SSD를 사용할 경우 구조적인 문제로 인해 고성능 장비인 SSD 성능을 다 발휘하지 못하게 되는 상황이 발생할 수 있다. 이 논문에서는 분산 파일 시스템의 하나인 GlusterFS에서 하드디스크를 SSD로 교체했을 때 발생할 수 있는 구조적인 문제점에 대해서 이야기하고 있다. GlusterFS의 경우 I/O-cache, Read-ahead, Write-behind와 같은 크고 순차적인 I/O (Large sequential I/O)의 성능을 위한 Translator들, 내부적으로 처리되는 큰 블록 사이즈 그리고 클라이언트에서 동기적(Synchronization)으로 I/O 처리를 담당하고 있는 단일 폴링 스레드(Polling thread) 등의 기능들이 SSD의 성능을 다 발휘하지 못하게 한다. 이러한 문제점을 해결하기 위해서 논문에서는 순차적인 I/O를 위한 Translator들을 제거하였다. fio 벤치마크[4]를 이용한 실험 결과, 4KB 랜덤 읽기의 경우 255%까지의 성능 향상을 보였고, 64KB랜덤 읽기의 경우 50%까지의 성능 향상을 보였다. 순차적인 읽기의 경우에도 4KB, 64KB, 및 128KB 모두 기존 GlusterFS와 비슷하거나 다소 좋은 성능을 보였다.

2. 배 경

GlusterFS는 분산 파일 시스템의 하나로, 데이터를 분산 저장하기 위해서 메타 데이터 서버를 이용하지 않고 DHT(Distributed Hash Table)를 이용하는 것이 특징이다. 따라서 메타 데이터 서버에 메타 데이터 요청이 집중되는 문제로 성능 병목 현상이 발생하지 않는다는 장점이 있다. 그리고 GlusterFS의 Volume을 구성하는 기본적인 방법에는 Replicate, Distribute, 및 Stripe가 있다. Replicate는 파일 시스템의 신뢰성(Reliability)을 높이기 위해서 똑같은 데이터를 여러 Brick에 복제해서 저장하는 방법이고, Distribute 방법은 파일 시스템의 성능을 위해서 파일 단위로 분산되어 있는 각각의 Brick에 번갈아 가며 저장하는 방법이고, 그리고 Stripe는 파

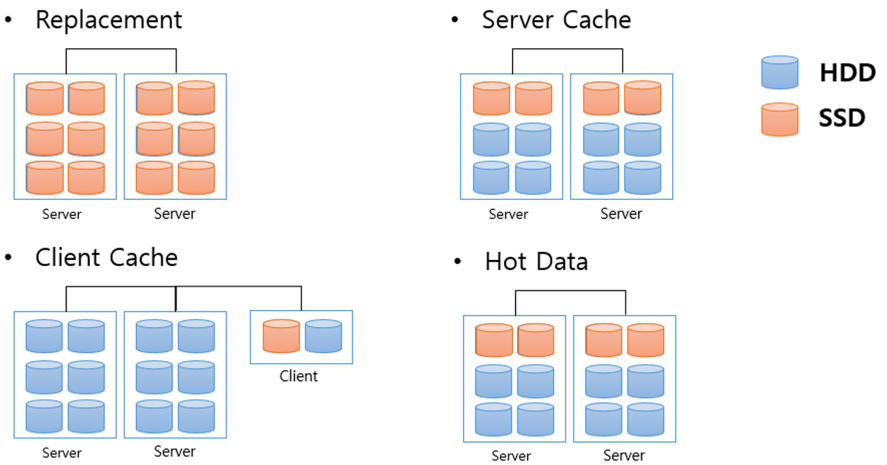

그림 1 분산 파일 시스템에서 SSD 활용 방법
Fig. 1 Methods of using SSDs in the Distributed File System

일 시스템의 성능을 위해서 파일을 일정한 블록 사이즈로 Brick에 번갈아 가며 저장하는 방법이다. 이러한 세 가지 방법 이외에도 Distributed Replicate, Distributed Stripe등 여러 개의 방법들을 동시에 적용해서 사용할 수 있다. 여기서의 Brick은 GlusterFS에서 저장소의 기본 단위로서, 로컬 파일 시스템 내의 파일을 저장할 수 있는 폴더이다. 이외에도, GlusterFS는 Translator라는 것이 존재를 하는데, Translator는 I/O-cache, Read-ahead 등의 Translator들이 있으며, 이들은 각각 특정한 기능을 담당하고 있다. 그리고 Translator는 다음 Translator에게 전달하는 방식의 과정을 가지고 있으며, I/O는 이런 전반적인 Translator 처리 과정을 지나가게 된다. GlusterFS는 일반적으로 클라이언트 쪽에서 FUSE (Filesystem in Userspace)를 이용해서 마운트해서 사용한다.

3. SSD환경에서 GlusterFS 문제점

3.1 내부 블록 사이즈 및 순차적인 I/O를 위한 구조

GlusterFS는 순차적인 I/O의 성능을 향상시키기 위해서 I/O-cache, Read-ahead, Write-behind의 기능을 이용하고 있다. I/O-cache는 I/O를 할 때 캐시의 페이지 크기만큼 블록을 읽어 오며, GlusterFS의 기본 페이지 사이즈는 내부적으로 128KB으로 설정되어 있다. 따라서 4KB, 64KB 등의 작은 블록 사이즈로 I/O를 하여도 GlusterFS는 128KB씩 데이터를 읽어오게 된다. 이는 순차적인 I/O일 때는 큰 문제가 없지만 랜덤 I/O의 경우 불필요한 데이터를 읽어오게 되는 것이기 때문에 성능 저하의 원인이 된다. 이는 요청하는 데이터의 크기가 작으면 작을수록 상대적으로 필요 없는 데이터를 더 많이 가져오기 때문에 점점 더 큰 성능 하락이 일어나게 된다. Read-ahead는 I/O시 여러 페이지들을 미리 읽어오는 기능으로 I/O-cache와 마찬가지로 랜덤 I/O시에는 연속적으로 접근을 하지 않기 때문에 불필요한 데이터를 읽어오게 된다. Write-behind는 128KB 단위로 배칭해서 쓰기를 처리하는 기능이다. 이러한 기능들은 크고 순차적인 I/O의 성능을 높일 수 있는 기능들이다. 그리고 시스템이 하드디스크로 이루어졌을 때 기대되는 성능을 보이며 동작한다. 왜냐하면, 하드디스크는 물리적인 장치로 디스크에 I/O를 할 때 마다 탐색 시간(Seek Time)이 있기 때문에 랜덤 I/O의 성능보다 순차적인 I/O가 월등한 성능을 보여주기 때문이다. 따라서 한번 데이터를 읽을 때, 순차적으로 데이터를 많이 읽어오는 것이 시스템의 성능 향상에 도움이 될 수 있다. 하지만 SSD를 이용하였을 경우는 이러한 기능들이 오히려 성능 하락에 주요한 요인이 된다. 이는 SSD의 경우, 랜덤 I/O의 성능과 순차적인 I/O의 성능이 하드디스크에 비해 월등히 차이가 적기 때문이다. 따라서 작은 랜덤 읽기 I/O (Small random I/O)의 요청이 왔을 때 기존의 방법대로 공간 지역성(Spatial Locality)을 고려하여 큰 사이즈의 블록을 읽게 되면 랜덤 I/O시 불필요한 많은 블록들이 클라이언트 서버간 전송되기 때문에 클라이언트 응용 프로그램의 관점에서 현저한 성능 저하가 발생한다.

3.2 클라이언트의 단일 폴링 스레드

GlusterFS의 클라이언트의 I/O요청은 epoll이라는 폴링을 하고 있는 스레드에 의해서 처리가 된다. 이 스레드는 I/O에 대한 정보를 가져온 다음, RPC (Remote Procedure Call)를 통해 동기적으로 서버에 I/O 작업을 한다. 따라서 단일 클라이언트가 낼 수 있는 최대 성능은 이 스레드에 의해서 결정된다. 이러한 특성 상 단일 클라이언트가 초 당 서버에 요청할 수 있는 Request양은 SSD의 성능을 보이기에 매우 부족하다.

그림 2는 하나의 서버 내에 SSD 두 개로 이루어진 GlusterFS 서버가 있고, 두 개의 클라이언트들이 동시적으로 서버에 I/O요청을 보내고 있는 상황의 대략적인 구조도이다. 서버 내에서 SSD로 요청을 보내는 glusterfsd 프로세스는 마운트된 SSD마다 있으며 내부적으로 16개의 스레드로 이루어져 있다. 반면에 앞에서 설명하였듯이 클라이언트에서 FUSE를 통해서 GlusterFS를 마운트하면 생기는 epoll 스레드의 개수는 한 개이다. 따라서 단일 클라이언트에서는 GlusterFS에 많은 양의 요청이 들어와도 epoll 스레드에 의해 동기적으로 서버에 하나씩 요청이 전달된다. 이로 인해 하드디스크와 달리 여러 개의 채널이 있는 SSD의 특성을 제대로 활용하지 못하게 된다[5-7]. 즉, 여러 개의 클라이언트가 I/O 요청을 보내야만 서버의 SSD의 여러 채널들에 I/O 요청들이 전송되고 그 때서야 각각의 SSD들은 제 성능을 다 발휘할 수 있게 된다.

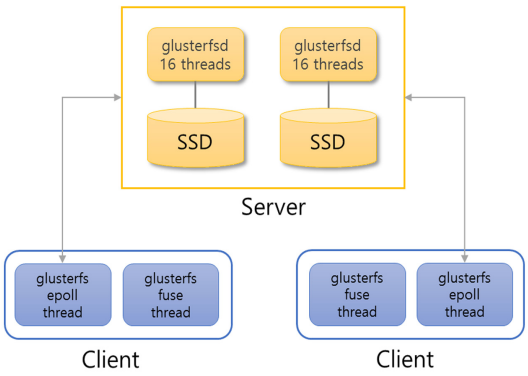

그림 2 대략적인 GlusterFS 구조도

Fig. 2 Approximated Architecture of the GlusterFS

4. SSD환경에서 GlusterFS 최적화

Brick을 위한 디바이스로 하드디스크를 고려하여 설계된 GlusterFS에서 하드디스크를 SSD로 교체했을 때, GlusterFS의 성능을 향상시키기 위해서 그림 3과 같이 크고 순차적인 I/O의 성능을 위한 기능인 I/O-cache, Read-ahead, 및 Write-behind 기능들을 제거하였다. I/O-cache Translator를 제거함에 따라 캐시의 페이지 사이즈 단위로 I/O를 하지 않아도 되기 때문에 요청 받는 크기에 맞게 I/O를 할 수 있게 되었고 이에 따라 4KB, 64KB등의 작은 랜덤 I/O에서 성능 향상을 이룰 수 있었다. Read-ahead Translator제거로 인해서 랜덤 I/O시 불필요한 데이터를 가져옴으로써 성능 저하를 발생시켰던 추가적인 I/O 또한 발생하지 않게 되었다. Write-behind Translator는 하드디스크의 탐색 시간으로 인해 랜덤 쓰기가 느려서 배칭을 통해서 성능을 향상시키기 위해서 도입된 기능이기 때문에 제거하였다. 이러한 Translator를 제거함으로써 그림 3과 같이 I/O를 처리하기 위한 Translator과정이 단순해졌다. 이러한 문제 이외에 단일 epoll 스레드로 인해 SSD 성능을 다 발휘할 수 없는 문제는, 멀티 스레드 epoll을 통해서 성능을 높일 수 있다. 하지만, 이 논문에서는 이에 대해서 자세하게 다루지 않겠다. 이는 GlusterFS의 차기 버전인 3.7 버전부터는 작은 파일 I/O를 위해서 멀티 스레드 epoll이 추가되었기 때문이다. 하지만 논문을 작성한 시점에는 GlusterFS 3.7 버전이 발표되지 않아 3.6버전에서 진행을 하였다. 추후 GlusterFS 3.7버전에서 추가적인 실험을 통해서, 멀티 스레드 epoll로 인해 GlusterFS의 성능개선이 얼마나 발생했는지, 그리고 SSD 환경에 맞게 더 성능개선을 할 부분이 있는지를 알아보려고 한다.

5. 실험 결과

5.1 실험 환경

실험은 2개의 서버와 2개의 클라이언트에서 진행이 되었다. 각각의 서버 및 클라이언트는 Intel® Core™ i7-4790 @ 3.60GHz 4코어 CPU 1개, 8GB 메인 메모리, Samsung 850 PRO SSD 256GB 1개, 그리고 10GbE NIC의 사양으로 이루어졌다. 그리고 운영체제로는 CentOS 6.7이 설치되어있었으며, GlusterFS의 경우 3.6버전을 이용 하였다. GlusterFS의 설정을 위해서 서버의 SSD에 각각 xfs 파일 시스템을 만들었고 각 서버에 1개씩 마운트를 하였다. 이 파일 시스템 내에 각각 1개의 폴더를 만들어 Brick으로 이용을 하였다. 실험은 서버당 1개의 Brick을 이용해서 총 2개의 Brick으로 GlusterFS를 Distributed mode로 설정을 해서 실험을 진행하였다. 실험은 1 개의 클라이언트만 이용해 GlusterFS 서버에서 I/O를 수행하는 테스트 실험 및 2개 클라이언트가 동시에 GlusterFS서버에 I/O를 수행하는 테스트를 진행하였다. 실험을 위한 벤치마크로는 fio를 이용했으며 buffered I/O 및 16 numjobs로 설정을 해서 16개의 스레드들이 각각 8 GB 파일을 4KB, 64KB, 및 128KB의 블록 사이즈로 600초 동안 랜덤 읽기 및 순차적 읽기를 하는 실험을 진행하였다.

5.2 실험 결과

GlusterFS를 2개의 Brick을 이용한 Distributed mode로 구성을 해서 기본 실험을 해보았고, SSD 환경에서 오히려 성능에 악영향을 주었던 I/O-cache, Read-ahead, Write-behind Translator들의 기능을 제거하고 실험을 진행하였다. 그림 4에 나와있듯이 4KB의 랜덤 읽기의 경우 기본 GlusterFS에서 실험한 경우 33,664KB/s의 성능을 보여줬다. 그리고 성능에 악영향을 주었던 Translator들을 제거를 하고 실험을 했을 때 119,539KB/s의 성능으로 기존 GlusterFS 대비 255%의 성능 향상을 보여주었다. 64KB의 블록 사이즈 변화에도 GlusterFS의 설정을 수정한 것이 기존 성능보다 더 좋은 성능을 보여주었다. 64KB 랜덤 읽기의 경우 기존 504,973KB/s에서 761,079KB/s로 50% 성능 향상을 보여줬다. 128KB 랜덤 읽기는 914,323KB/s에서 921,774KB/s로 유사한 성능을 보여 줬

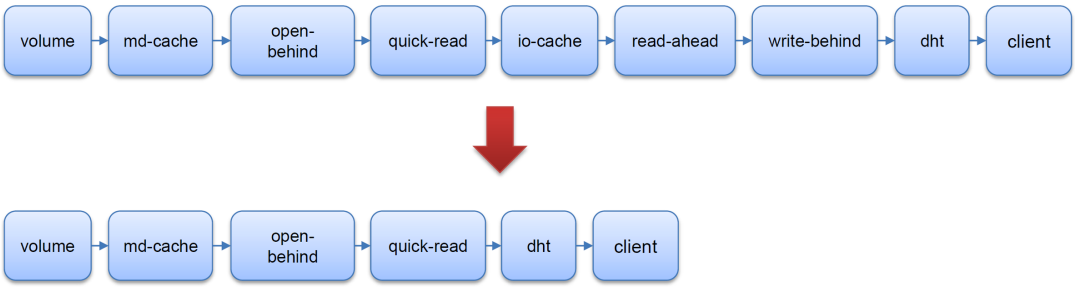

그림 3 GlusterFS Translator 처리 과정 변화
Fig. 3 Change of the GlusterFS Translator Processing Flow

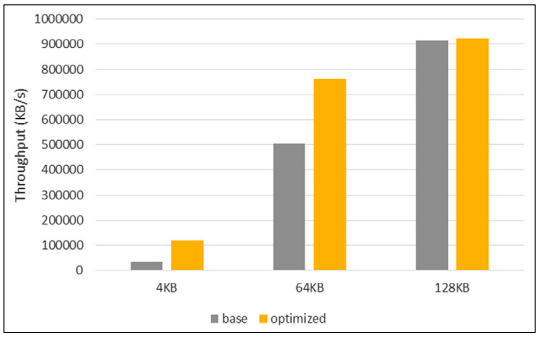

그림 4 블록 사이즈 별 랜덤 읽기 실험 결과
Fig. 4 Random I/O Performance according to block size

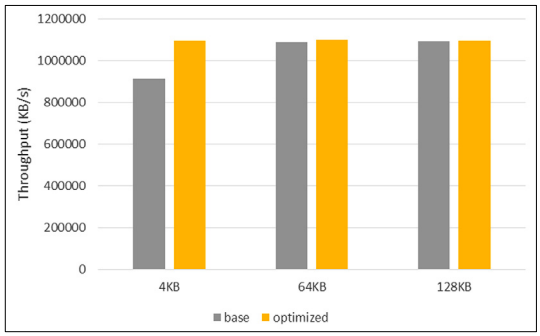

그림 5 블록 사이즈 별 순차적 읽기 실험 결과
Fig. 5 Sequential I/O performance according to block size

다. 순차적인 읽기의 경우 그림 5에서 보다시피 4KB 블록 사이즈로 읽을 시 914,627KB/s에서 1,096,098KB/s로 19% 향상이 있었다. 64KB 및 128KB의 경우 Translator 제거 전과 제거 후 모두 10GbE의 최대 성능을 발휘해서 차이점을 비교할 수 없었으나 랜덤 읽기 및 4KB 순차적 읽기의 실험 결과로 보았을 때 비슷하거나 더 좋은 성능을 보일 것이라고 예상된다. 이러한 성능 향상도, 특히 랜덤 읽기 4KB 및 64KB의 경우, SSD가 낼 수 있는 최대 성능에는 미치지 못한다. 이는 클라이언트에서 I/O를 처리하는 epoll 스레드가 하나이기 때문에 발생하는 문제이다. 실제로 실험을 할 때 CPU 사용률을 보면 epoll 스레드가 100%의 CPU 사용률을 이용하고 있음을 알 수 있다. 이는 이 epoll 스레드를 멀티 스레드로 구성을 하면 여러 코어에서 동시에 epoll 스레드가 동작할 수 있기 때문에 CPU 에 의해서 병목되었던 성능이 더 향상될 수 있을 것이다. 멀티 epoll 스레드에 대한 기능은 GlusterFS의 3.7 버전에서 추가되었기 때문에, 3.7버전에서는 4KB 및 64KB의 경우 더 큰 I/O 성능 향상이 일어날 것이다. 논문을 작성할 시점에는 3.7 버전이 발표되지 않아

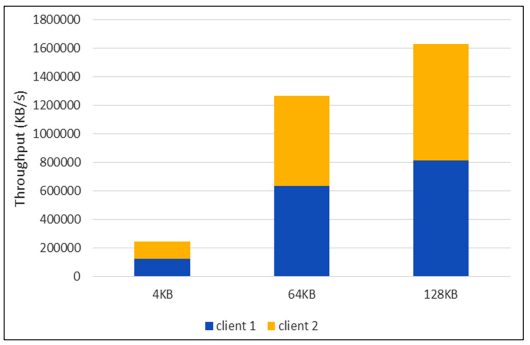

그림 6 동시에 2개 클라이언트 I/O시 실험 결과
Fig. 6 Random I/O performance when 2 clients conduct I/O at the same time

멀티 스레드 epoll에 대해서 시뮬레이션 해보기 위해 두 개의 클라이언트가 동시에 I/O를 하는 실험을 진행하였다. 그림 6에서 보다시피 4KB의 경우 243,979KB/s, 64KB의 경우 1,266,392KB/s, 128KB의 경우 1,631,996KB/s의 성능을 보여주었다. 이는 멀티 스레드 epoll로 인해서 클라이언트가 동시에 여러 개의 I/O를 서버에 할 수 있게 되면, GlusterFS의 단일 클라이언트의 I/O 성능이 향상될 것임을 암시한다.

6. 결론 및 향후 연구 계획

GlusterFS는 순차적이고 용량이 큰 파일의 성능을 위해 내부적으로 디자인 되어 있기 때문에 SSD 환경의 작고 랜덤한 I/O에 대해서 좋은 성능을 보이지 못하고 있었다. 이러한 문제점들로는 순차적인 I/O를 위해 도입된 I/O-cache, Read-ahead, Write-behind 등의 Translator들, 그리고 클라이언트가 RPC 요청을 하는 단일 epoll 스레드의 사용이 있다. 이러한 문제들을 해결하기 위해서 크고 순차적인 I/O를 위한 Translator들을 제거하였고, 이에 따라 기본 블록 I/O 사이즈가 I/O 캐시의 페이지 사이즈인 128KB였던 기존 GlusterFS를 요청된 블록 사이즈에 맞게 클라이언트와 서버 간에 I/O를 수행할 수 있게 되었다. 이러한 변경으로 인해서, 랜덤 읽기 4KB의 경우 기존 환경에 비해서 255%의 성능 향상, 64KB의 경우 50%의 성능 향상이 있었다. 하지만 이는 아직 더 개선할 부분이 있고, 이는 멀티 스레드 epoll을 통해서 해결 할 수 있을 것이다. GlusterFS에서는 3.7 버전부터는 멀티 스레드 epoll기능을 추가하였으니, 추후 SSD의 작고 랜덤한 I/O에 대해서 성능을 측정을 해보고, 추가로 성능을 개선할 수 있는 사항이 있는지 확인해 보려고 한다.

References

[1] S. Weil, S. Brandt, E. Miller, D. Long, and C.

Maltzahn, "Ceph: A Scalable, High-Performance Distributed File System," *Proc. of the 7th Symposium on Operating Systems Design and Implementation*, 2006.
[2] Red Hat. (2015, Sep. 9). GlusterFS [Online]. Available: http://www.gluster.org (downloaded 2015, Sep. 9)
[3] A. Davies and A. Orsaria, "Scale out with glusterfs," *Linux Journal*, Vol. 2013, No. 235, Nov. 2013.
[4] J. Axboe. (2015, Sep. 9). Flexible IO Tester [Online]. Available: http://git.kernel.dk/?p=fio.git;a=summary (downloaded 2015, Sep. 9)
[5] C. Park, P. Talawar, D. Won, M. Jung, J. Im, S. Kim, and Y. Choi, "A High Performance Controller for NAND Flash-based Solid State Disk (NSSD)," *Proc. of the 21st IEEE Non-Volatile Semiconductor Memory Workshop*, Monterey, 2006.
[6] C. Dirik and B. Jacob, "The performance of PC solid-state disks (SSDs) as a function of bandwidth, concurrency, device architecture, and system organization," *Proc. of the 36th Annual International Symposium on Computer Architecture*, 2009.
[7] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron, "Migrating server storage to SSDs: analysis of trade-offs," *Proc. of the 4th ACM European Conference on Computer Systems*, 2009.

김 덕 상
2014년 홍익대학교 컴퓨터공학과(학사) 2015년~현재 서울대학교 컴퓨터공학과 석사과정. 관심분야는 분산시스템, 클라우드 시스템 등

엄 현 상
1992년 서울대학교 계산통계학과(학사) 1996년 University of Maryland 컴퓨터과학과(석사). 1997년 Sun Microsystems, Inc. Data Engineering Group 인턴. 2003년 University of Maryland 컴퓨터과학과(박사). 2003년~2005년 삼성전자 정보통신총괄 책임연구원. 2005년~현재 서울대학교 컴퓨터공학부 교수. 관심분야는 운영체제, 스토리지 시스템, 분산/클라우드 시스템, 에너지 효율적 시스템, 내결함성 시스템, 성능공학, 보안, 정보 다이나믹스

염 헌 영
1984년 서울대학교 계산통계학과(학사) 1986년 Texas A&M 컴퓨터공학과(석사). 1992년 Texas A&M 컴퓨터공학과(박사). 1986년~1990년 Texas Transportation institute 시스템 분석가. 1992년~1993년 삼성 데이터 시스템. 1993년~현재 서울대학교 교수. 관심분야는 분산시스템, 스토리지 등