http://dx.doi.org/10.7840/kics.2016.41.11.1515

분산 파일시스템의 소거 코딩 구현 및 성능 비교

김 재 열◆, 김 영 철*, 김 동 오*, 김 홍 연*, 김 영 균*, 서 대 화°

Implementation and Performance Measuring of Erasure Coding of Distributed File System

Cheiyol Kim◆, Youngchul Kim*, Dongoh Kim*, Hongyeon Kim*,
Youngkyun Kim*, Daewha Seo°

요 약

최근의 빅데이터, 머신러닝, 클라우드 컴퓨팅 분야의 성장에 따라 대용량의 비정형 데이터를 저장할 수 있는 스토리지의 중요성은 날로 커지고 있다. 이에 따라 MAHA-FS, GlusterFS, Ceph 등의 개방형 하드웨어 기반의 분산 파일시스템 기술이 많은 주목을 받고 있다. 이러한 저비용 분산 파일시스템들은 데이터의 내결함성을 보장하기 위하여 초기에 복제 방식을 사용하였으나, 스토리지의 용량이 커질수록 복제 방식이 가지는 스토리지 공간의 저효율성이 점차 부각되면서 이를 보완하려는 방향으로 연구가 진행되고 있다. 본 논문은 복제방식을 대체하여 스토리지 공간 효율성을 향상시킬 수 있는 소거코딩 기법을 MAHA-FS 분산 파일시스템에 적용하여 스토리지의 효율성을 높이고, 소거코딩 지원에 따라 발생하는 데이터 일관성 문제를 해결하는 효율적인 방식으로 VDelta 기법을 제안하고 적용하였다. 본 논문은 MAHA-FS와 GlusterFS의 소거코딩의 구조적 차이점을 기술하고 두 파일시스템의 성능을 비교하여 MAHA-FS의 소거코딩 성능이 GlusterFS에 비해 우수함을 확인하였다.

Key Words : Erasure Coding, Data Consistency, Distributed Storage, Fault Tolerance, MAHA-FS

ABSTRACT

With the growth of big data, machine learning, and cloud computing, the importance of storage that can store large amounts of unstructured data is growing recently. So the commodity hardware based distributed file systems such as MAHA-FS, GlusterFS, and Ceph file system have received a lot of attention because of their scale-out and low-cost property. For the data fault tolerance, most of these file systems uses replication in the beginning. But as storage size is growing to tens or hundreds of petabytes, the low space efficiency of the replication has been considered as a problem. This paper applied erasure coding data fault tolerance policy to MAHA-FS for high space efficiency and introduces VDelta technique to solve data consistency problem. In this paper, we compares the performance of two file systems, MAHA-FS and GlusterFS. They have different IO processing architecture, the former is server centric and the latter is client centric architecture. We found the erasure coding performance of MAHA-FS is better than GlusterFS.

※ 이 논문은 2016년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원을 받아수행된 연구임(No. R0126-15-1082, ICBMS(IoT, 클라우드, 빅데이터, 모바일, 정보보호) 핵심 기술 개발 사업 총괄 및 엑사스케일급 클라우드 스토리지 기술 개발)
◆ First Author : Electronics and Telecommunications Research Institute, gauri@etri.re.kr, 정회원
° Corresponding Author : Dept. of Electronics Eng., Kyungpook National University, dwseo@ee.knu.ac.kr, 종신회원
* Electronics and Telecommunications Research Institute, {kimyc, dokim, kimhy, kimyoung}@etri.re.kr
논문번호 : KICS2016-10-299, Received October 10, 2016; Revised October 24, 2016; Accepted October 31, 2016

Ⅰ. 서 론

클라우드 컴퓨팅, 빅데이터, 머신러닝, IoT 분야의 성장과 더불어 디지털 데이터의 증가는 매년 폭발적으로 늘어나고 있다. 이에 따라 대용량의 데이터를 저장할 수 있는 스토리지 기술이 ICT 분야에서 필수적인 요소가 되었다. 요구되는 스토리지의 용량이 커짐에 따라 단일 스토리지(scale-up)만으로는 감당하기 어려워져, 여러 대의 스토리지 노드(scale-out)에 분산 저장하는 분산 파일시스템이 각광을 받게 되었다. 분산 파일시스템은 고가의 NAS나 SAN과 달리 표준 하드웨어(commodity hardware) 서버를 사용하여 소프트웨어적으로 동작하기 때문에 저비용으로 대용량의 스토리지 시스템을 구축할 수 있는 장점을 가지고 있다.

GlusterFS[2]는 범용 파일 인터페이스를 제공하는 분산 파일시스템으로 2005년에 오픈소스로 개발을 시작하여 2011년에 RedHat사에 인수되었다. Ceph[3]은 초기에 오브젝트 파일스템으로 개발되었다가, 블록과 파일 인터페이스를 추가로 제공하고 있는 분산 파일시스템으로 GlusterFS와 마찬가지로 오픈소스 기반이다. Ceph 또한 GlusterFS와 마찬가지로 2014년에 RedHat사에 인수되었다. RehHat사의 잇단 분산 파일시스템 기술회사의 인수는 소프트웨어 기반의 분산 스토리지 시스템의 가치를 보여주는 것으로 해석될 수 있을 것이다.

국내에서 개발된 분산 파일시스템으로는 대표적으로 GloryFS[5], MAHA-FS[6], OwFS[7]가 있다. GloryFS와 MAHA-FS는 클라우드용 파일시스템으로 표준 파일 인터페이스를 제공한다. OwFS는 네이버에서 자사의 서비스를 위해 개발한 분산 파일시스템으로 전용의 인터페이스만을 제공한다. GloryFS와 MAHA-FS는 국내의 대표적인 이동통신사에서 클라우드 스토리지 서비스에 사용되고 있으며 수백 페타바이트 규모로 구축되어 있다.

많은 분산 스토리지 시스템들이 데이터의 내결함성을 지원하는 방법으로 초기에 복제(replication) 방식을 주로 사용하였으나 최근에는 스토리지 공간 효율성을 높일 수 있는 소거코딩(erasure coding) 방식을 도입하고 있다. 소거코딩 방식은 디지털통신에서 사용되던 네트워크 코딩 기법[8]을 스토리지에 적용한 것으로, RAID-5, 6의 패리티 기반 데이터 복원 방식을 소프트웨어적으로 구현한 것이다. 소거코딩 기법은 데이터를 인코딩 후 패리티를 생성하고 데이터와 패리티를 여러 서버에 분산 저장함으로써 서버 고장과 디스크 고장으로부터 데이터를 보호하는 방식이다.

본 논문은 국내에서 개발된 MAHA-FS에 스토리지 공간 효율성을 향상시키기 위하여 소거코딩을 지원할 때의 설계상의 특징과 소거코딩에 따른 데이터 일관성 문제를 해결하기 위한 버전과 로그 기반의 VDelta 기법을 소개한다. 2장은 분산 파일시스템에서의 소거코딩 관련연구 동향을 살펴보고, 3장에서는 MAHA-FS의 소거코딩 설계 이슈에 대해서 논의한다. 4장에서는 MAHA-FS와 GlusterFS의 소거코딩 성능을 비교하고 성능의 결과에 대해서 분석한다. 5장에서는 본 논문의 결론을 서술한다.

Ⅱ. 관련 연구

표준 하드웨어 기반의 스케일 아웃 특성을 지원하는 분산 파일시스템은 국외에서 개발된 파일시스템들[1-3]이 높은 인지도와 많은 사용자층을 확보하고 있는 반면에 국내의 연구와 개발은 미약한 것이 사실이다. 그럼에도 불구하고 스토리지 시스템은 클라우드 시스템에서 가장 많은 비용을 차지하고, 확장성에 큰 영향을 주며, 데이터 보안의 핵심 요소로서 기술적 영향이 큰 부분으로, 클라우드 시스템을 도입하는 기관들은 무작정 해외 기술을 도입하는 것에 우려를 갖고 있는 것 또한 사실이다. 이러한 국내의 기술 요구에 따라 GloryFS와 MAHA-FS가 한국전자통신연구원에서 개발되었다. GloryFS와 MAHA-FS는 단일 MDS(Meta Data Server) 구조를 가지고 있으며 데이터 보호 방식으로 복제 방식을 지원하고 있다. GloryFS는 데이터베이스 기반의 메타데이터 엔진을 사용하는 반면 MAHA-FS는 메타데이터 연산 성능 향상을 위하여 자체적으로 개발한 메타데이터 엔진을 사용하고 있다.

국내에서 개발된 네이버의 OwFS도 단일 MDS 구조로 GloryFS나 MAHA-FS와 동일하며 GloryFS와 같이 데이터베이스를 이용하여 메타데이터를 유지한다. OwFS는 GloryFS와 MAHA-FS와 달리 표준 파일 인터페이스를 제공하지 않고 전용의 인터페이스만을 제공하기 때문에 기존의 응용 프로그램이 OwFS를 사용하기 위해서는 응용프로그램을 수정해야 하는 단점이 있다.

국내외 대부분의 분산 파일시스템들[1-7]이 데이터 보호 방법으로 복제 방식을 초기에 지원하였으나 스토리지의 용량이 페타바이트 이상으로 확장되자 스토리지 공간 효율성이 중요해지면서 최근에 소거코딩을 지원하기 시작했다. GlusterFS는 2014년부터 소거코딩을 지원하기 시작했다[9]. GlusterFS는 소거코딩의 적용분야로 의료데이터 등을 저장하는 아카이브에 초

점을 맞추고 있다. 이는 복제 방식에 비해 소거코딩 방식이 성능이 떨어지기 때문이다. GlusterFS는 소거코딩시 필요한 인코딩/디코딩 및 데이터 서버로의 데이터의 분배/수집 작업을 클라이언트에서 수행하는 구조적 특징을 가지고 있다. Ceph도 2014년부터 소거코딩을 지원하기 시작했다. Ceph은 GlusterFS와 달리 서버 노드에서 인코딩/디코딩 및 분배/수집을 수행한다. HDFS[4]도 저장공간 효율화를 위해 소거코딩을 도입하였다[10]. 네이버도 OwFS에 소거코딩을 적용한 Papyrus[11]를 아카이브용 스토리지에 사용하고 있다.

이와 같이 국내외의 대부분의 분산 파일시스템들이 스토리지 용량이 페타바이트 이상으로 확장되면서 스토리지 공간 효율성이 중요해 짐에 따라 최근에 소거코딩을 지원하기 시작했다. 하지만 아직은 소거코딩이 복제 방식에 비해서 충분한 성능을 보장하지 못하여 아카이브나 백업 용도로 주로 사용되고 있다.

소거코딩 방식은 데이터를 인코딩/디코딩하는 단위인 스트라이프(stripe) 단위로 쪼갠다. 스트라이프는 K개의 데이터 블록과 M개의 패리티 블록으로 구성된다. 소거코딩은 N(=K+M)개의 블록을 서로 다른 서버 노드에 저장하여 서버 노드의 장애시에도 데이터의 유실을 방지한다.

그림 1에서 파일의 데이터는 스트라이프 단위로 쪼개져, 하나의 스트라이프에 속한 데이터 블록을 인코딩하여 패리티를 생성하고(그림1에서 하나의 스트라이프는 데이터 블록 2개(K)와 패리티 블록 2개(M)로 구성), 각각의 블록을 서로 다른 데이터 서버(DS1~4)에 저장한다. 그림 1에서 알 수 있듯이 하나의 스트라이프를 구성하는 4개의 블록은 서로 연관성을 가지게 된다. 4개의 블록 중 잘못 기록된 블록이 있다면 이 블록은 스트라이프에서 제외되어야 한다. 이를 소거코딩시에 발생하는 분산 데이터의 데이터 일관성 문제라고 한다. 데이터의 일관성 문제는 정상적인 상황에서는 발생하지 않지만, 시스템의 구성 요소 중 장애가 발생하면 일어날 수 있다.

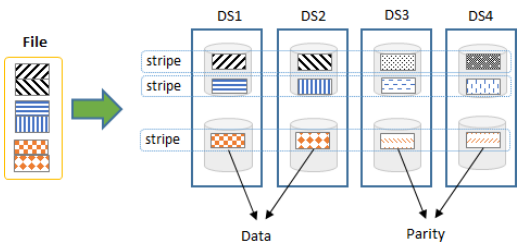


그림 1. 소거코딩시의 데이터 분산
Fig. 1. Data distribution of erasure coding

소거코딩시의 데이터 일관성이 깨지는 문제를 불완전-쓰기(Partial Write)문제라고 한다. 불완전 쓰기 문제를 해결하는 전통적인 방법은 2단계 커밋(Two-phase commit)방법[12]이다. 2단계 커밋은 순차성(Serializability)과 복구기능(Recoverability)으로 구성된다. 순차성 즉 동시성 제어에 관한 연구는 [13, 14] 등이 있다. 순차성 보장을 포함하는 2단계 커밋을 비교적 쉽게 해결하는 방법으로는 데이터를 읽기 전용으로 사용하는 것이다[15, 16]. 오브젝트 파일시스템인 GFS[1]와 Ceph는 이러한 읽기 전용 파일시스템으로 생성(create)만 지원할 뿐 생성된 파일에 대한 갱신을 지원하지 않음으로써 2단계 커밋의 요구사항을 쉽게 만족시킬 수 있다. 하지만 이러한 방식은 데이터 갱신을 지원하지 못하는 단점을 가지고 있어, 범용의 목적으로 사용할 수는 없다.

[17, 18]은 동시성 제어를 관리하는 중앙 제어 서버가 없는 상황에서 다수의 클라이언트에서 소거코딩 기반의 스토리지에 접근할 때의 순차성 보장과 데이터 일관성을 보장하는 기법들을 소개한다. 하지만 이러한 방식은 클라이언트의 모든 갱신 기록을 스토리지 서버 노드마다 로그로 모두 저장하고 있어야 하는 단점을 가지고 있다.

본 논문은 스토리지의 공간 효율성을 높이기 위하여 국내에서 개발된 MAHA-FS에 소거코딩을 적용하였다. 그리고 소거코딩 적용시에 직면하게 되는 데이터 일관성 문제를 해결하는 방법으로 데이터 청크의 버전과 델타로그를 이용하는 VDelta 기법을 제안한다.

Ⅲ. MAHA-FS의 소거코딩 특징

이번 장에서는 MAHA-FS의 소거코딩 지원시의 특징을 기술한다. MAHA-FS는 인코딩/디코딩을 스토리지 서버 노드에서 수행하며 인코딩/디코딩의 성능을 높이기 위하여 하드웨어 가속 기능을 지원한다. 또한 인코딩/디코딩의 단위인 스트라이프의 크기를 효율적인 업데이트가 가능하도록 지원하며, VDelta 기법을 통하여 데이터 일관성이 깨지지 않도록 지원한다. 각각의 항목에 대한 구체적인 설명은 아래에 기술한다.

3.1 데이터 서버 중심 소거코딩 구조

소거코딩을 통해 데이터를 저장하는 경우 데이터는 인코딩을 거쳐 패리티를 생성하고 이를 여러 데이터 서버에 분산해 저장한다. 데이터의 인코딩/디코딩 및 분배/수집을 클라이언트에서 담당하는 클라이언트 중심 구조와 데이터 서버에서 담당하는 데이터 서버 중

심구조가 있다. 그림 2는 데이터 서버 중심 구조에서 데이터를 저장할 때의 동작 모습을 보여준다. 그림 2에서 클라이언트는 파일의 구성 정보를 MDS(메타데이터 서버)에 요청해서 받은 후, MasterDS에 저장할 데이터를 전송한다. MasterDS는 인코딩/디코딩 및 분배/수집을 담당하는 서버로 클라이언트로부터 받은 데이터를 인코딩한 후 데이터와 패리티 조각을 정해진 데이터 서버로 전송해 분산 저장한다.

본 논문의 MAHA-FS에서는 데이터 서버 중심 소거코딩 구조를 채택하였다. 이에 반해 GlusterFS는 클라이언트에서 이러한 작업을 수행하는 클라이언트 중심 소거코딩 구조를 가지고 있다.

표 1은 두 구조 사이의 차이점을 보여준다. 첫 번째는 데이터 서버 중심 구조는 소거코딩의 유무와 상관없이 클라이언트의 부하는 큰 차이가 없지만, 클라이언트 중심 구조에서는 소거코딩에 따라 발생하는 인코딩/디코딩과 분배/수집의 부하가 모두 클라이언트에서 발생하게 된다. 이는 스토리지의 부하가 클라이언트로 전이되는 것으로 클라이언트에서 동작하는 응용프로그램이 파일시스템의 입출력 부하에 영향을 받게 되는 것을 의미한다. 이러한 상황은 클라이언트의 응용프로그램이 CPU나 메모리, 네트워크 자원을 많이 필요로 하지 않는다면 큰 문제가 되지 않지만, 반대의 경우에는 심각하게 응용프로그램의 성능을 저하시킬 수 있는 원인이 될 수 있기 때문에 바람직한 구조라고 볼 수 없다.

두 번째는 소거코딩시의 전체 네트워크 지연시간으로, 데이터 서버 중심 구조의 경우 클라이언트에서 MasterDS로의 추가적인 지연시간이 발생할 것이다.

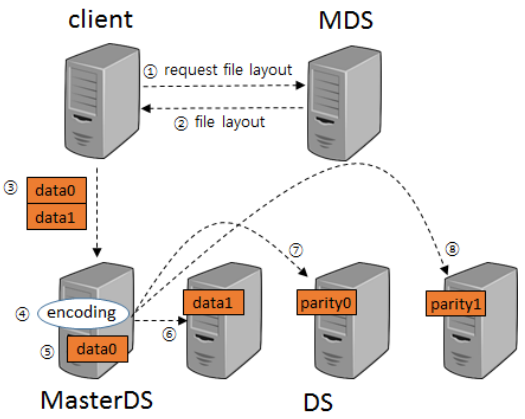


그림 2. 데이터 서버 중심 구조에서의 데이터 저장 시 동작 모습
Fig. 2. Storing data in data server centric architecture

표 1. 데이터 서버 중심 구조와 클라이언트 중심 구조의 특징 비교
Table 1. Features of data server centric architecture and client centric architecture

	Data server centric	Client centric
Overhead of Client	no	yes (encoding/ decoding, scattering/ gathering)
Network latency	Latency of client to MasterDS + latency of MasterDS to each DS	Latency of client to each DS
concurrency control	simple	complex

이는 지연시간이 전체 입출력에서 주요한 성능의 요인이 된다면 큰 문제가 될 것으로 보이지만, 그렇지 않은 경우에는 성능에 큰 영향을 끼치지 않을 것으로 판단된다. 추가적인 네트워크 지연시간의 성능 영향 유무를 4장에서 클라이언트 중심 구조를 가지는 GlusterFS와의 비교시험으로 확인할 것이다.

세 번째는 다중 클라이언트의 동시 접근시의 순차성 보장의 복잡성 유무이다. MAHA-FS의 경우에는 특정 데이터에 대한 입출력 요청이 지정된 인코딩/디코딩을 담당하는 MasterDS로 먼저 전달되기 때문에 MasterDS에서 순차성을 보장 할 수 있어 비교적 구조와 구현이 간단하다. 하지만 클라이언트 중심 구조에서는 각 클라이언트에서 개별적으로 데이터 서버에 접근하기 때문에 클라이언트의 데이터 접근 시 순차성을 보장하기 위해서는 순차성 보장을 위한 특정 서버나 서비스가 필요하다. 이는 입출력의 성능을 떨어뜨릴 수도 있으며 전체 시스템의 구조가 복잡하게 될 가능성이 높다.

MAHA-FS에서는 두 구조의 장, 단점을 비교한 결과 소거코딩 시에도 클라이언트에 추가적인 부하를 주지 않고, 클라이언트 동시 접근시의 데이터 접근 순차성 보장에 유리한 데이터 서버 중심 구조를 채택하였다.

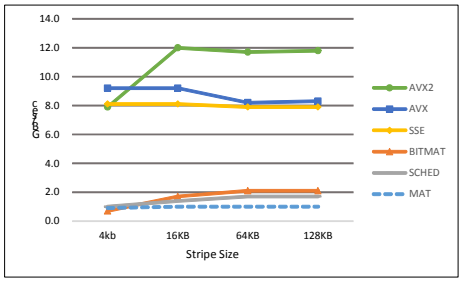
3.2 SIMD 기반 하드웨어 가속 인코딩/디코딩 지원

소거코딩에서 적용되는 인코딩/디코딩은 CPU에 구애받지 않고 사용할 수 있는 소프트웨어 방식과 특정 CPU의 SIMD(Single-Instruction Multiple-Data)명령어를 이용하는 하드웨어 가속 방식이 있다. 소프트

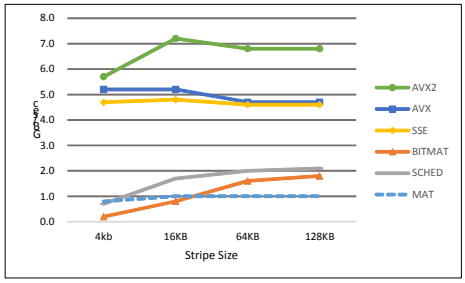
웨어 인코딩/디코딩 도구로는 가장 많이 알려진 것이 Jerasure[19] 라이브러리이며, 하드웨어 가속 방식으로 가장 많이 사용되는 것은 Intel의 ISA-L[20] 라이브러리이다. ISA-L은 인코딩/디코딩 가속 명령어로 Intel CPU에서 지원하는 SSE, AVX, AVX2 명령어를 이용하여 구현된 라이브러리이다. 소프트웨어 기반의 Jearsure와 하드웨어 가속 방식인 ISA-L을 이용한 인코딩/디코딩의 성능을 시험을 통해 비교해 보았다.

성능 측정에는 Intel Xeon E5-2623 v3 3.0Ghz (4-core x 2-socket) CPU와 32GB의 메모리를 가진 서버가 사용되었다. 디스크와 네트워크 등의 기타 하드웨어 사양은 성능에 영향을 끼치지 않기 때문에 명시하지 않는다. 실험에 사용된 인코딩 알고리즘은 Reed-Solomon 알고리즘이다.

그림 3은 인코딩/디코딩 성능 측정결과로 SSE, AVX, AVX2는 ISA 라이브러리 결과이며 SCHED, BITMAT, MAT은 Jerasure 라이브러리에서 지원하는 소프트웨어 인코딩 방법들이다. 그림 3의 결과는 하드웨어 가속 방식을 사용하는 것이 월등히 높은 성능을 보여준다. 하드웨어 가속 방식은 4KB의 작은 스트라이프 크기에서도 인코딩과 디코딩시 8GB/s와 4.5GB/s 이상의 성능을 보여준다. 이러한 성능 결과



(a)



(b)

그림 3. Jerasure와 ISA-L의 인코딩/디코딩 성능 (a) 인코딩 성능 (b) 디코딩 성능
Fig. 3. Performance of en/decoding of Jerasure library and ISA-L (a) encoding performance (b) decoding performance

는 3장 3절의 스트라이프 크기 결정에 영향을 주었다.

MAHA-FS는 하드웨어 가속 기능을 통한 인코딩/디코딩의 우수한 성능 결과에 따라서 인텔 CPU 기반 ISA-L을 이용한 하드웨어 가속 인코딩/디코딩 기능을 지원한다.

3.3 RMW(Read-Modify-Write)를 최소화하는 4KB 스트라이프 크기 지원

그림 3에서와 같이 스트라이프 크기가 클수록 인코딩/디코딩 성능이 좋으며, 읽기에서도 더 좋은 성능을 기대할 수 있다. 왜냐하면 스트라이프의 크기가 커지면 스트라이프에 포함된 단일 데이터 블록의 크기도 커져 읽기 처리 시 읽어야 하는 데이터 서버의 숫자가 줄어들 수 있으며, 이는 읽기 성능이 향상되는 효과를 보인다. 하지만 스트라이프의 일부 데이터가 갱신되는 경우에는 새로운 패리티 생성을 위해 스트라이프에 포함된 모든 데이터를 읽어서 인코딩하고 이를 다시 저장해야 하는데, 이를 RMW(Read-Modify-Write)라고 부른다. GFS와 Ceph등의 오브젝트 파일 시스템은 데이터를 최초에 한 번 생성할 뿐 갱신은 지원하지 않기 때문에 스트라이프의 크기가 크면 유리한 반면, 파일 인터페이스를 지원하는 MAHA-FS에서는 갱신이 발생하기 때문에 오브젝트 파일시스템처럼 스트라이프의 크기를 키우면 갱신시의 성능이 급격히 떨어질 수 있다.

스트라이프의 크기를 줄이더라도 인코딩/디코딩의 성능이 충분하다면 갱신 성능을 위해서 스트라이프의 크기를 줄이는 것을 고려해 볼 수 있다. 그림 3에서 스트라이프의 크기가 4KB인 경우의 성능을 살펴보면 ISA-L인 경우 최소 인코딩시 8GB/s, 디코딩시 4.5GB/s의 성능을 보여준다. 이 성능은 40Gbps 이하의 네트워크에서는 충분한 성능으로 판단된다. 따라서 MAHA-FS에서는 4KB의 스트라이프 크기를 지원하여 리눅스 커널의 페이지 캐시 블록의 크기인 4KB 쓰기에 대해서 RMW가 발생하지 않도록 지원한다.

3.4 VDelta 기법을 이용한 MAHA-FS의 데이터 일관성 지원

2장에서 언급한 불완전 쓰기를 방지하는 방법으로 MAHA-FS에서는 VDelta 기법을 제안한다. VDelta 기법은 데이터 서버에 데이터를 저장할 때 사용하는 청크 파일에 버전(Version)과 이전 버전의 로그(Delta)를 유지하고 이를 이용해 불완전 쓰기와 데이터 복구를 지원한다. 3장의 1절에서 언급한 바와 같이 MAHA-FS의 소거코딩은 데이터 서버 중 하나인

MasterDS을 중심으로 입출력을 수행한다. 스트라이프를 구성하는 청크 파일들의 집합을 청크세트(chunk set)라고 부르며, MasterDS는 청크세트 단위로 할당된다. 즉 하나의 파일이 N개의 청크세트로 구성되어 있다면 해당 파일은 N개의 MasterDS를 가지고 있다. 데이터 서버의 숫자가 충분하지 않다면 MasterDS는 서로 중복될 것이다. 아래에서는 MAHA-FS에서 VDelta 기법을 사용하여 정상적인 경우와 비정상적인 경우 데이터의 일관성을 유지하는 방식을 기술한다.

* 정상 상황

파일에 대한 쓰기 요청이 처음 발생하면 이를 받은 MasterDS는 해당 청크세트에 포함된 데이터 서버에 각 청크의 버전을 확인한다. 해당 청크에 아직 데이터가 없다면 각 데이터 서버는 청크의 버전으로 '0'을 반환해 줄 것이다. 모든 청크의 버전이 '0'인 경우 MasterDS는 해당 청크세트의 버전을 '0'으로 지정하고 최초의 쓰기를 각 데이터 서버에 전달한다. 이 때 MasterDS는 현재의 청크세트의 버전정보인 '0'을 함께 보낸다. 쓰기 요청을 전달받은 데이터 서버들은 전달된 버전이 자신이 가지고 있는 청크의 버전인 '0'과 같으면 쓰기를 수행하고 버전을 '1'로 업데이트 한 후 MasterDS에 쓰기요청의 완료를 반환한다. 쓰기 요청이 올 때 마다 데이터 서버는 이전 버전과의 차이(Delta)를 로그로 기록한다. 청크에 저장된 로그는 불완전-쓰기가 발생할 때 쓰기요청 이전의 상태로 돌아가기 위해서 사용된다.

그림 4는 5번째 쓰기 요청이 MasterDS에서 데이터 서버(DS)로 전달될 때의 동작을 보여준다. 요청을 처리하면 데이터 서버의 청크는 '4'의 청크버전과 '3'의 로그버전에서 '5'의 청크버전과 '4'의 로그버전으로 변경될 것이다. MasterDS는 청크세트의 버전 정보를 입출력 요청이 존재하는 한 유지하며 특정 시간 이상 입출력 요청이 없을 때 삭제한다.

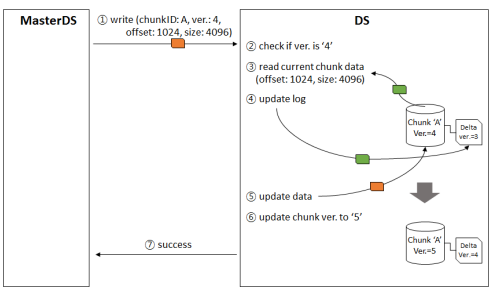


그림 4. VDelta의 정상 상황에서의 동작
Fig. 4. Normal operation of VDelta

* 불완전 쓰기가 발생한 상황

불완전 쓰기가 발생한 경우에는 MasterDS는 이를 확인하고 이미 업데이트가 발생한 데이터 서버에 로그를 이용하여 이전 버전으로 rollback할 것을 요청한다.

그림 5는 DS2의 일시적인 장애(네트워크 단선 등)로 인해 불완전 쓰기가 발생했을 때 Delta 로그를 통해 rollback을 수행하여 데이터의 일관성을 유지한 후에, 재시도 하여 쓰기 요청을 완료하는 경우를 보여주고 있다. 그림 5와 달리 DS2에서 영구적인 장애가 발생한 경우에는 MasterDS는 정해진 시간동안 재시도를 하고 최종적으로 실패를 클라이언트에 보고한다. 장애가 발생한 DS의 정보는 Heartbeat을 통해 MDS에 보고되고 DS의 해당 청크는 청크세트에서 제외된다.

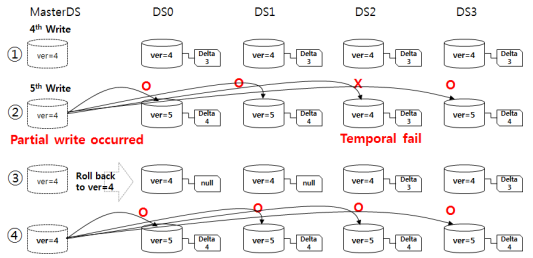


그림 5. DS의 일시적인 장애로 발생한 불완전-쓰기 처리 방법
Fig. 5. Recover from partial-write occurred from DS' temporal fail

* MasterDS의 장애 상황

DS에 장애가 발생한 경우와 달리 MasterDS에 장애가 발생한 상황은 훨씬 복잡해진다. MasterDS의 장애 상황은 2가지 경우로 나눌 수 있다. 첫 번째 경우는 MasterDS에 영구적인 장애가 발생한 상황이며 두 번째 경우는 일시적으로 장애가 발생한 상황으로 하나의 청크세트에 대해 2개의 MasterDS가 동작하는, 일명 Split-brain 상황이 발생할 수 있다.

그림 6은 MasterDS에 영구적으로 장애가 발생한 경우의 MAHA-FS의 동작을 보여준다. ②에서 DS0과 DS1의 청크를 갱신하고 DS2와 DS3의 청크는 갱신되지 않은 상태에서 MasterDS에 장애가 발생한 경우이다. MasterDS의 장애를 감지한 MDS는 새로운 MasterDS를 할당한다. 장애 MasterDS로부터 쓰기 요청을 처리하지 못한 클라이언트는 MDS를 통해 새로운 MasterDS의 정보를 받아 새로운 MasterDS에 쓰기 처리를 요청한다. ③에서 요청을 받은 새로운 MasterDS는 각 청크의 버전을 DS들에 요청하고 '4'와 '5'로 이루어진 청크버전을 수신한다. 새로운

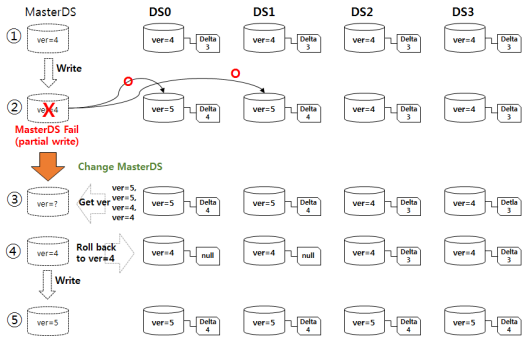


그림 6. MasterDS의 영구적인 장애시의 VDelta의 동작
Fig. 6. VDelta operation when the permanent MasterDS' fail occurred

MasterDS는 청크의 버전이 서로 다르고 버전 정보가 '1'밖에 차이가 나지 않으므로 불완전-쓰기로 판단하고 각 청크의 버전을 '4'번으로 회귀하는 rollback을 요청해 ④에서와 같이 청크의 버전을 모두 '4'로 통일하고 MasterDS도 '4'번의 청크세트 버전을 유지하고 새로운 입출력 요청을 요청한다. ⑤번은 정상적으로 쓰기 요청이 처리된 후의 상황으로 모두 '5'의 청크 버전과 '4'의 Delta 로그 버전을 가지고 있는 것을 볼 수 있다.

MasterDS 장애의 두 번째 경우는 MaterDS의 Split-brain이 발생한 상황이다. 이는 MDS가 새로운 MasterDS를 할당하고 장애가 발생한 MasterDS가 정상상황으로 돌아온 경우에 발생한다. 이 때 과거 정보를 가지고 있는 클라이언트가 과거의 MasterDS를 통해서 입출력을 시도할 수 있다. 이러한 MasterDS의 Split-brain을 방지하기 위해 DS는 청크마다 MasterDS의 정보를 유지한다. DS는 청크의 MasterDS가 변경되면 입출력 요청을 던진 MasterDS에 해당 청크세트의 MasterDS 정보를 확인해 달라고 요청하고, 이를 수신한 MasterDS가 자신이 유효한 MasterDS인지 MDS에 문의한다. 자신이 유효한 MasterDS라면 DS에 자신으로 MasterDS를 변경할 것을 요청한 후 입출력을 수행한다. 과거의 MasterDS는 새로운 MasterDS가 입출력 요청을 수행하기 전까지는 정상적으로 동작되다가 새로운 MasterDS의 MasterDS 변경요청이 DS에 전달된 후에는 DS로부터 입출력 요청이 거부되어 새로운 MasterDS로 제어권이 완전히 넘어가게 된다. 이러한 방식으로 VDelta 기법은 MasterDS의 split-brain 상황을 해소한다.

데이터 서버의 장애로 인해 소거코딩시에 발생할 수 있는 데이터 일관성 문제를 해결하기 위하여 MahaFS에서는 청크별 버전과 직전 버전의 로그를 저장하는 VDelta 기법을 사용한다. VDelta 기법은 DS의 장애로 인한 불완전-쓰기 및 MasterDS의 장애로 발생하는 Split-brain 문제들을 MDS와의 빈번한 통신 없이도 방지할 수 있어 데이터 일관성 보장에 따르는 성능저하를 최소화 시킬 수 있다.

Ⅳ. 시험 및 결과분석

3장에서 기술한 특징을 갖는 소거코딩 기능을 MAHA-FS에 구현하였다. 4장에서는 MAHA-FS의 소거코딩 성능을 GlusterFS의 소거코딩 성능과 시험하여 비교하였다. GlusterFS는 현재 국내외적으로 가장 많이 사용되는 오픈소스 기반의 분산 파일시스템이다. MAHA-FS가 데이터 서버 중심의 소거코딩 구조인데 반해 GlusterFS는 클라이언트 중심의 구조로 서로간의 차이점을 성능과 함께 비교해 볼 수 있다.

시험에 사용된 하드웨어와 소프트웨어 환경은 표 2와 같다. 시험은 클라이언트 5대와 데이터 서버 6대로 구성된 환경에서 이루어졌으며 클라이언트와 데이터 서버는 동일한 사양으로 구성되어 있다. 운영체제는 별도의 디스크에 설치되어 데이터 저장에 사용되는 디스크와의 간섭을 배제하였다. 데이터 서버는 하나의 디스크를 4개의 파티션으로 나눈 후 각각의 파티션을 데이터 서버의 저장소로 사용하였다. 시험에 사용된 소거코딩은 모두 4+2(K+M) 설정을 사용하였으며, MAHA-FS는 스트라이프의 크기로 4KB를 적용하였으며, GlusterFS는 별도의 설정사항이 없어 기본 설정이 적용되었다.

시험 도구로는 Iozone[21] 벤치마크 도구를 사용하였으며, 단일 클라이언트 성능과 다중 클라이언트 성능을 측정하였다. 단일 클라이언트 시험은 쓰레드의 개수를 1, 5, 10로 증가하면서 측정하였으며, 다중 클라이언트 시험은 클라이언트의 개수를 1~5개로 증가

표 2. 시험에 사용된 하드웨어와 소프트웨어 환경
Table 2. Hardware and software specifications of testbed

	Client	Data server
Number of machines	5	6
CPU	Intel Xeon E5-2623 v3 3.0Ghz(4-core x 2-socket)	
Memory	32GB	
Network	10Gbps ethernet	
Disk	HDD(Sata3, 7200rpm)	
OS	CentOS 7.0	
Erasure coding configuration	4+2(4:data, 2:parity)	

하면서 측정하였다. 시험은 128KB의 레코드 크기를 가지는 순차 쓰기/읽기와 4KB의 레코드 크기를 가지는 랜덤 쓰기/읽기로 구성되었다.

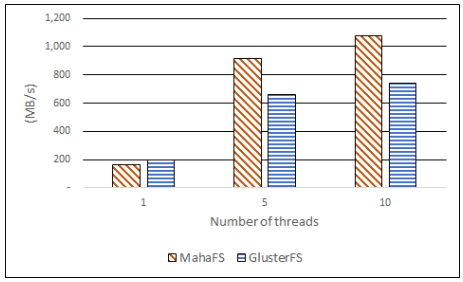
MAHA-FS와 GlusterFS 모두 FUSE[22]기반의 파일시스템으로 읽기 시에는 클라이언트의 캐시와 데이터 서버의 캐시의 유무에 따라 큰 성능차이를 보인다. 클라이언트에 데이터가 캐싱된 경우에는 읽기 요청은 클라이언트의 커널 페이지 캐시에서 처리되어 파일시스템의 성능을 제대로 확인할 수 없다. 또한 데이터 서버에서는 데이터가 캐시 되어 있지 않으면 파일시스템의 성능이 데이터 서버 디스크의 성능에 제한되는 문제가 발생할 수 있다. 충분한 성능을 가진 디스크가 데이터 서버에 장착되어 있다면 이는 문제가 되지 않겠지만, 본 시험 환경과 같이 충분하지 않는 경우에 파일시스템의 성능을 제대로 측정할 수 없게 된다. 이러한 이유로 본 시험의 읽기 성능은 모두 클라이언트의 캐시는 비워지고, 데이터 서버에는 읽을 데이터가 모두 캐싱되어 있는 상황에서 측정되었다.

4.1 단일 클라이언트 성능

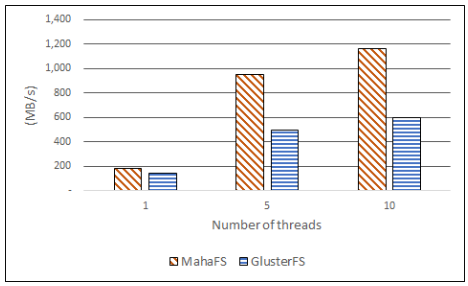
그림 7은 단일 클라이언트에서 측정한 MAHA-FS와 GlusterFS의 순차성능과 랜덤성능이다.

순차쓰기 성능은 쓰레드가 1개일 때는 GlusterFS가 근소하게 우수하였으나, 쓰레드가 5개, 10개로 증가할수록 MAHA-FS가 훨씬 우수한 성능을 보여주었다. 성능의 차이는 MAHA-FS가 약 40%정도 우수한 성능을 보여주었다. 순차 읽기 성능은 모든 경우에 MAHA-FS가 우수하였으며 순차 쓰기와 마찬가지로 쓰레드의 개수가 늘어날수록 성능의 차이도 더 커짐을 알 수 있었다. 순차 읽기의 경우에는 다중 쓰레드의 경우 거의 2배 가까운 성능의 차이를 MAHA-FS에서 보여주었다.

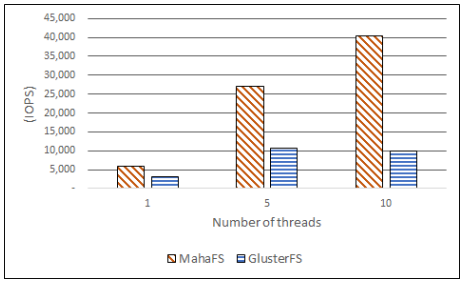
랜덤성능의 경우에는 MAHA-FS가 절대적으로 우수한 성능을 보여주었다. MAHA-FS가 10개 쓰레드를 기준으로 랜덤쓰기의 경우는 4배, 랜덤읽기의 경우는 6배의 우수한 성능을 보여주었다. 이러한 차이는 클라이언트와 데이터 서버간의 통신 구조에 따른 차이로 보인다. MAHA-FS의 경우는 MasterDS에서 데이터 서버로의 통신 시 가능한 최대한의 병렬성을 가지도록 구현되었다. 병렬성이 부족하게 구현되면 비교적 연속적으로 큰 데이터를 전달하는 순차성능에서는 큰 차이를 보이지 않지만, 랜덤입출력 시험처럼 작은 크기의 데이터를 전송하는 경우 그림 7과 같이 큰 차이를 보일 것으로 판단된다.



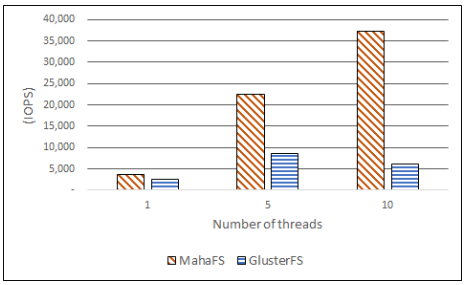
(a)



(b)



(c)

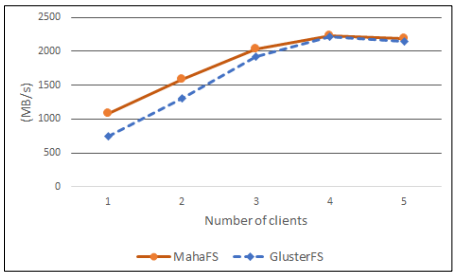


(d)

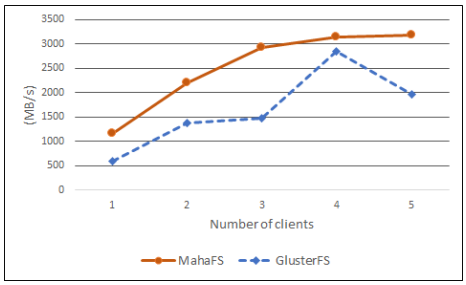
그림 7. 단일 클라이언트 성능 (a) 순차쓰기 (b) 순차읽기 (c) 랜덤쓰기 (d) 랜덤읽기
Fig. 7. Single client IO performance (a) sequential write (b) sequential read (c) random write (d) random read

4.2 다중 클라이언트 성능

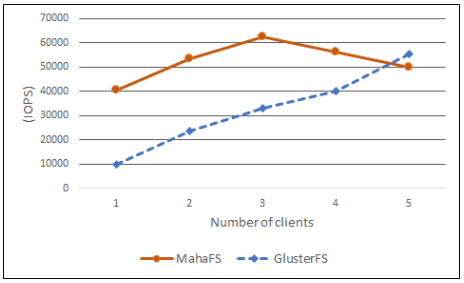
그림 8은 다중 클라이언트에서 측정한 MAHA-FS와 GlusterFS의 순차성능과 랜덤성능이다. 시험은 10개의 쓰레드를 수행하는 클라이언트를 1~5개로 늘려가며 측정하였다. 다중 클라이언트 측정은 Iozone의



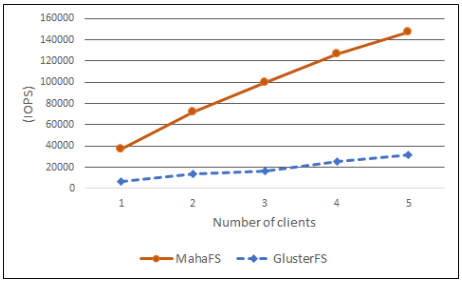
(a)



(b)



(c)



(d)

그림 8. 다중 클라이언트 성능 (a) 순차쓰기 (b) 순차읽기 (c) 랜덤쓰기 (d) 랜덤읽기
Fig. 8. Multiple clients IO performance (a) sequential write (b) sequential read (c) random write (d) random read

클러스터 모드를 이용하였다.

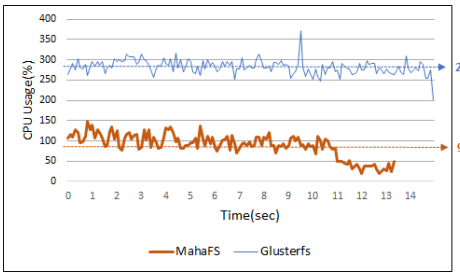
다중 클라이언트의 순차성능은 단일 클라이언트의 순차성능에 비해 더욱 복잡한 양상을 보여준다. 순차쓰기 성능은 MAHA-FS가 1개의 클라이언트일 때 45%정도 우수하다가 점차 성능의 차이가 줄어들다가 4개의 클라이언트부터는 거의 동일한 성능을 보여준다. 이러한 성능양상은 랜덤쓰기에서도 살펴볼 수 있다. 랜덤쓰기에서는 성능의 차이가 순차쓰기에서처럼 점진적으로 줄어들지는 않지만 클라이언트의 개수가 4개가 되는 지점부터 급격히 줄어 5개인 경우에는 오히려 역전이 되는 현상을 볼 수 있다. 쓰기성능의 이러한 양상을 MAHA-FS와 GlusterFS의 구조적 차이점으로 설명할 수 있다. MAHA-FS의 경우 인코딩/디코딩, 분배/수집의 부하를 모두 데이터 서버에서 담당하는데 비해 GlusterFS의 경우 이를 클라이언트에서 담당한다. 따라서 GlusterFS의 경우 클라이언트가 추가될수록 파일시스템에서 사용할 수 있는 자원이 늘어나는 것과 같은 효과를 볼 수 있어 클라이언트의 증가에 따라 파일시스템의 성능이 올라갈 수 있다. 쓰기 입출력의 경우 분배와 더불어 인코딩이 더해짐으로 인해 읽기에 비해 이러한 효과가 더 확실하게 나타나는 것으로 판단된다. 이와는 달리 읽기의 경우는 인코딩이 필요 없어 파일시스템에 더해지는 부하가 쓰기에 비해 적기 때문에 그림 8과 같이 MAHA-FS가 GlusterFS보다 항상 우수한 성능을 보여주는 것으로 생각된다. 순차읽기의 경우는 MAHA-FS가 최소 10%에서 최대 약 2배의 우수한 성능을 보여주고, 랜덤읽기에서는 최소 4.5배에서 최대 6배의 성능 차이를 보여주고 있다.

다음 3절에서는 앞서 언급한 것과 같이 쓰기와 읽기 처리 시에 GlusterFS가 MAHA-FS에 비해 얼마나 더 많은 CPU를 사용하는지를 단일 클라이언트 상황에서의 순차쓰기와 순차읽기의 경우에 대해서 측정한 결과로 보여준다.

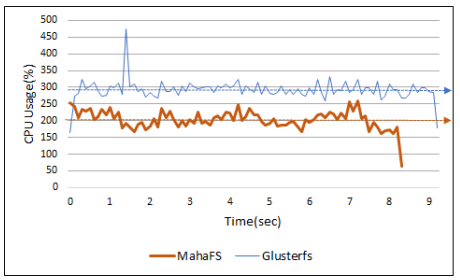
4.3 MAHA-FS와 GlusterFS의 클라이언트 부하 비교

그림 9는 클라이언트에서 10개의 쓰레드를 이용해 순차 쓰기/읽기를 수행했을 때의 CPU 사용률을 측정한 그래프이다. CPU의 사용률은 Iozone이 수행되는 동안 측정되어 MAHA-FS와 GlusterFS의 측정시간이 서로 다르다. CPU의 사용률에는 파일 클라이언트의 사용률과 함께 Iozone의 사용률도 포함되어 있다.

순차쓰기의 경우 GlusterFS가 평균 282%의 CPU 사용률을 보인데 비해, MAHA-FS의 경우 평균 90%의 사용률을 보여주었다. 순차쓰기 경우는 GlusterFS가 MAHA-FS에 비해 약 3배의 CPU를 더 사용함을 확인할 수 있다. 순차읽기의 경우는 GlusterFS의 경우 평균 292%, MAHA-FS의 경우 평균 203%로 약 1.5배의 차이를 보여준다. 쓰기의 경우 인코딩 부하가 추



(a)



(b)

그림 9. 순차 입출력시의 클라이언트의 부하 (a) 순차쓰기 (b) 순차읽기
Fig. 9. CPU usage of client (a) sequential write (b) sequential read

가됨으로써 읽기에 비해 더 큰 차이를 보이는 것으로 판단된다.

그림 9의 결과를 바탕으로 GlusterFS의 경우 파일시스템의 부하를 클라이언트가 많은 부분 담당함으로써 클라이언트가 추가될수록 입출력 성능을 향상시키는 효과를 볼 수 있음을 확인할 수 있었다. 하지만 이런 구조는 클라이언트의 부하가 크지 않은 경우에는 큰 문제가 되지 않지만 클라이언트에서 많은 부하가 발생하는 서비스나 응용프로그램이 수행되는 경우에는 파일시스템의 성능 저하도 발생할 수 있을뿐더러, 파일시스템의 부하가 응용프로그램의 성능을 저하시키는 간섭을 일으킬 가능성이 높아 효과적인 구조라고 할 수는 없을 것이다.

V. 결 론

본 논문은 국내에서 개발된 표준 하드웨어 기반의 분산 파일시스템인 MAHA-FS에 스토리지 공간 효율성 향상을 위해 구현된 소거코딩 구조의 특징을 소개하고, 소거코딩 지원 시에 발생하는 데이터 일관성 문제를 해결할 수 있는 버전과 로그 기반의 VDelta 기법을 소개하였다.

MAHA-FS의 소거코딩은 데이터 서버 중심 구조로 설계되어 클라이언트와 파일시스템 부하의 간섭을 배제하여, 클라이언트의 응용프로그램을 제약 없이 운영할 수 있게 한다. 또한 소거코딩시에 발생하는 데이터 일관성 문제를 청크 파일의 버전과 직전 버전의 로그인 Delta 로그를 기록하는 VDelta 기법을 제안하고 적용하여 MDS의 개입을 최소화함으로써 큰 성능저하 없이 해결할 수 있었다.

MAHA-FS의 소거코딩 성능은 국내외적으로 가장 많이 쓰이는 분산 파일시스템인 GlusterFS의 소거코딩 성능과 비교하여 시험하였다. 단일 클라이언트에서는 순차쓰기 성능은 약 40%, 순차읽기 성능은 약 2배, 랜덤쓰기 성능은 2~4배, 랜덤읽기 성능은 2.5~6배의 우수한 성능을 보여주었다. 다중 클라이언트의 순차쓰기 성능은 클라이언트 수가 작은 경우는 MAHA-FS가 우수한 성능을 보이다가 클라이언트가 많아질수록 거의 대등한 성능을 보였으며, 순차읽기 성능은 최대 2배 정도로 MAHA-FS의 성능이 우수하였다. 다중 클라이언트의 랜덤쓰기 성능은 클라이언트의 개수가 4개까지는 MAHA-FS의 성능이 우수하다가 5개가 되는 지점에서는 성능이 역전되는 현상이 발생했다. 이는 4장에서 기술한 바와 같이 GlusterFS가 클라이언트 중심 구조를 가지고 있어 파일시스템의 부하를 클라이언트가 분담하면서 생기는 현상이다. 랜덤읽기 성능은 MAHA-FS가 4~6배까지 GlusterFS에 비해 우수한 성능을 보여주었다. 이러한 성능차이는 분산 입출력시의 병렬성의 정도에 따른 것으로 보인다.

4장의 시험은 데이터 서버측에 데이터가 캐싱된 상태에서 읽기 성능을 측정한 결과로 데이터 서버에 캐싱이 되지 않은 상태에서 측정하면 랜덤읽기 성능은 수십 IOPS 이하로 떨어진다. 이를 해결하기 위해서는 물리적으로 빠른 접근이 보장되는 SSD[23]와 같은 고속 디스크를 사용하거나, 별도의 캐시 구조[24]를 추가하여 해결해야 할 것으로 보인다.

MAHA-FS의 소거코딩 성능은 GlusterFS 보다 우수한 소거코딩 성능을 보이지만, 복제 방식의 성능에 비하면 부족한 것이 사실이다. 이는 소거코딩 방식이 데이터를 작은 크기로 쪼개어 다수의 데이터 서버에 대해서 쓰기/읽기를 수행하기 때문이다. 이러한 성능 차이를 해결하기 위해서는 파일의 원본 데이터를 유지하는 캐시를 클라이언트와 데이터 서버 사이에 두어 성능을 보상하는 것이 가장 일반적으로 사용되는 방법이다.

최근의 컴퓨팅 및 통신 환경은 대용량의 저비용 스토리지가 필수적인 상황으로, ICT 분야에서 기술적,

상업적 우위를 점하기 위해서는 중추적인 컴퓨팅 인프라에 해당되는 분산 파일시스템의 기술적인 자립이 반드시 필요하다. 이런 이유로 순수 국내 기술로 완성된 MAHA-FS의 기술적 성숙도를 높이는 한편 분산 파일시스템의 기술 확산이 필요하다고 생각된다.

References

[1] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," in *Proc. ACM SOSP*, pp. 29-43, 2003.

[2] GlusterFS. http://www.gluster.com/ (accessed Sept. 2016).

[3] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. Long, and C. Maltzahn, "Ceph: A scalable, high-performance distributed file system," in *Proc. 7th Symp. Operating Syst. Design and Implementation,* USENIX Association, 2006.

[4] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," *IEEE MSST*, pp. 1-10, 2010.

[5] H. Y. Kim, G. S. Jin, M. H. Cha, S. M. Lee, S. M. Lee, Y. C. Kim, and Y. K. Kim, "GLORY-FS: A distributed file system for large-scale internet service," *KICS Inf. and Commun. Mag.*, vol. 30, no. 4, pp. 16-22, Mar. 2013.

[6] Y. C. Kim, D. O. Kim, H. Y. Kim, Y. K. Kim, and W. Choi, "MAHA-FS: A distributed file system for high performance metadata processing and random IO," *KIPS Trans. Software and Data Eng.*, vol. 2, no. 2, pp. 91-96, Feb. 2013.

[7] J. S. Kim and T. W. Kim, "OwFS: A distributed file system for large-scale internet services," *J. Korean Data & Inf. Sci. Soc.*, vol. 27, no. 5, pp. 77-85, May 2009.

[8] G. J. Lee, Y. C. Shin, J. H. Koo, and S. H. Choi, "Practical implementation and performance evaluation of random linear network coding," *J. KICS*, vol. 40, no. 9, pp. 1786-1792, Sept. 2015.

[9] D. Lambright, *Erasure Codes and Storage Tiers on Gluster*, SA summit, Sept. 23, 2014.

[10] A. Ajisaka, *HDFS 2015: Past, Present, and Future*(2015), Retrieved Sep., 30, 2016, from http://events.linuxfoundation.org/sites/events/files/slides/HDFS2015_Past_present_future.pdf

[11] T. Y. Kim, *Lessons learned from deploying SSD in NAVER services*(2014), Retrieved Sep., 30, 2016, http://dcslab.hanyang.ac.kr/nvramos/nvramos14/presentation/s1.pdf

[12] J. N. Gray, "Notes on data base operating systems," *Springer-Verlag*, vol. 60, pp. 393-481, Berlin, 1978.

[13] C. Gray and D. Cheriton, "Leases: An efficient fault-tolerant mechanism for distributed file cache consistency," *ACM SIGOPS Operating Systems Rev.*, vol. 23, no. 5, pp. 202-210, Dec. 1989.

[14] H.-T. Kung and John T. Robinson, "On optimistic methods for concurrency control," *ACM TODS*, vol. 6, no. 2, pp. 213-226, Jun. 1981.

[15] David P. Reed and L. Svobodova, "SWALLOW: A distributed data storage system for a local network," in *Proc. 1FIP Working Group 6.4 Int. Workshop on Local Netw.*, pp. 355-373, Aug. 1980.

[16] David P. Reed, "Implementing atomic actions on decentralized data," *ACM Trans. Computer Systems (TOCS)*, vol. 1, no. 1, pp. 3-23, Feb. 1983.

[17] G. R. Garth, J. J. Wylie, R. G. Ganger, and M. K. Reiter, "Efficient consistency for erasure-coded data via versioning servers," *Carnegie-Mellon Univ. Pittsburgh PA School of Comput. Sci.*, no. CMU-CS-03-127, Mar. 2003.

[18] S. Frolund, A. Merchant, Y. Saito, S. Spence, and A. Veitch, "A decentralized algorithm for erasure-coded virtual disks," *IEEE Int. Conf. Dependable Syst. and Netw.*, pp. 125-134, Jun. 2004.

[19] J. S. Plank, S. Simmerman, and C. D. Schuman, *Jerasure: A library in C/C++ facilitating erasure coding for storage applications-Version 1.2*, Technical Report CS-08-627, University of Tennessee, 2008.

[20] Intel Storage Acceleration Library,

https://software.intel.com/en-us/storage/ISA-L (accessed Sept. 2016).
[21] Iozone Filesystem Benchmark, http://www.iozone.org/docs/IOzone_msword_98.pdf(accessed Sept. 2016).
[22] "Filesystem in Userspace," https://sourceforge.net/projects/fuse/(accessed May 2016).
[23] D. H. Kim and S. Y. Hwang, "An efficient wear-leveling algorithm for NAND flash SSD with multi-channel and multi-way architecture," *J. KICS*, vol. 39B, no. 7, pp. 425-432, Jul. 2014.
[24] S. M. Han, H. S. Park, and T. W. Kwon. "Shelf-life time based cache replacement policy suitable for web environment," *J. KICS*, vol. 40, no. 6, pp. 1091-1101, Jun. 2015.

김 재 열 (Cheiyol Kim)

1999년 2월 : 경북대학교 전자공학과 졸업
2001년 2월 : 경북대학교 전자공학과 석사
2012년 2월 : 경북대학교 전자공학과 박사수료
2001년 2월~현재 : 한국전자통신연구원 책임연구원
<관심분야> 분산 파일시스템, 클라우드 컴퓨팅, 파일시스템 캐시

김 영 철 (Youngchul Kim)

1995년 2월 : 강원대학교 전자계산학과 졸업
1999년 2월 : 강원대학교 전자계산학과 석사
2000년 2월~현재 : 한국전자통신연구원 책임연구원
<관심분야> 분산 파일시스템, 클라우드 스토리지, 데이터베이스 시스템

김 동 오 (Dongoh Kim)

2000년 2월 : 건국대학교 컴퓨터공학과 졸업
2002년 2월 : 건국대학교 컴퓨터·정보통신공학과 석사
2006년 2월 : 건국대학교 컴퓨터·정보통신공학과 박사
2006년~2009년 : 건국대학교 강의교수
2009년~현재 : 한국전자통신연구원 선임연구원
<관심분야> 고성능 분산 파일시스템, 데이터베이스, 공간 데이터 관리

김 홍 연 (Hongyeon Kim)

1992년 2월 : 인하대학교 통계학과 졸업
1994년 2월 : 인하대학교 전자계산학과 석사
1999년 2월 : 인하대학교 전자계산학과 박사
1999년~현재 : 한국전자통신연구원 스토리지시스템연구실장 / 책임연구원
<관심분야> 분산 파일시스템, 클라우드 컴퓨팅, 빅데이터

김 영 균 (Youngkyun Kim)

1991년 2월 : 전남대학교 전산통계학과 졸업
1993년 2월 : 전남대학교 전산통계학과 석사
1995년 2월 : 전남대학교 전산통계학과 박사
1996년~현재 : 한국전자통신연구원 고성능컴퓨팅연구부장 / 책임연구원
2014년~현재 : 한국정보과학회 컴퓨터시스템연구회/데이터베이스 소사이어티 운영위원
<관심분야> 데이터베이스 시스템, 파일시스템, 클라우드 컴퓨팅

서 대 화 (Daewha Seo)

1981년 2월 : 경북대학교 전자공학과 졸업
1983년 2월 : 한국과학기술원 전산학과 석사
1993년 2월 : 한국과학기술원 전산학과 박사
1983년~1995년 : 한국전자통신연구원
1995년~현재 : 경북대학교 IT대학 전자공학부 교수
2004년~현재 : 경북대학교 임베디드 소프트웨어 연구센터 센터장
<관심분야> 임베디드 SW, 병렬처리, 분산운영체제