Journal of Korea Multimedia Society Vol. 19, No. 8, August 2016(pp. 1487-1497)
http://dx.doi.org/10.9717/kmms.2016.19.8.1487

토러스 연결망 기반의 대용량 멀티미디어용 분산 스토리지 시스템

김재열[†], 김동오[††], 김홍연[†††], 김영균[††††], 서대화[†††††]

Torus Network Based Distributed Storage System for Massive Multimedia Contents

Cheiyol Kim[†], Dongoh Kim[††], Hongyeon Kim[†††], Youngkyun Kim[††††], Daewha Seo[†††††]

ABSTRACT

Explosively growing service of digital multimedia data increases the need for highly scalable low-cost storage. This paper proposes the new storage architecture based on torus network which does not need network switch and erasure coding for efficient storage usage for high scalability and efficient disk utilization. The proposed model has to compensate for the disadvantage of long network latency and network processing overhead of torus network. The proposed storage model was compared to two most popular distributed file system, GlusterFS and Ceph distributed file systems through a prototype implementation. The performance of prototype system shows outstanding results than erasure coding policy of two file systems and mostly even better results than replication policy of them.

Key words: Distributed Storage, Torus Network, Multimedia Storage, Scalability

1. 서 론

인터넷과 모바일 통신의 발달로 동영상과 같은 대용량 멀티미디어 데이터가 폭발적으로 증가하고 있다. 이에 따라 증가하는 멀티미디어 데이터를 저장하기 위해 스토리지의 용량과 성능 향상[1]에 대한 요구 또한 마찬가지로 증가하고 있다. 엄청난 데이터의 증가속도로 인해 스트리밍 서비스 등의 멀티미디어 서비스 업체는 저비용으로 고확장성을 갖춘 스토리지를 필요로 한다. 이러한 요구사항을 만족시킬 수 있는 스토리지로 분산 스토리지 시스템이 있다. 분산 스토리지 시스템은 고가의 NAS(Network Attached Storage)나 SAN(Storage Area Nework) 스토리지가 아닌 개방형 하드웨어를 기반으로 고용

※ Corresponding Author : Daewha Seo, Address: (41566) 80, Daehak-ro, Buk-gu, Daegu, Korea, TEL : +82-53-940-8660, FAX : +82-53-950-5505, E-mail : dwseo@ee.knu.ac.kr
Receipt date : Jul. 18, 2016, Approval date : Jul. 29, 2016
[†] High Performance Computing Research Dept., Electronics and Telecommunications Research Institute (E-mail : gauri@etri.re.kr)
[††] High Performance Computing Research Dept., lectronics and Telecommunications Research Institute (E-mail : dokim@etri.re.kr)
[†††] High Performance Computing Research Dept., Electronics and Telecommunications Research Institute (E-mail : kimhy@etri.re.kr)
[††††] High Performance Computing Research Dept., Electronics and Telecommunications Research Institute (E-mail : kimyoung@etri.re.kr)
[†††††] Dept. of Electronics Eng., School of IT, Kyungpook National University
※ This work was supported by Institute for Information & communications Technology Promotion(IITP) grant funded by the Korea government(MSIP) (No.R0126-15-1082, Management of Developing ICBMS (IoT, Cloud, Bigdata, Mobile, Security) Core Technologies and Development of Exascale Cloud Storage Technology)

량의 스토리지 구축을 가능하게 한다. 분산 스토리지에서 데이터의 신뢰성(내결함성)을 높이는 방법으로 복제(replication) 방식이 주로 사용되었으나 낮은 디스크 활용률의 문제를 해결할 수 있는 소거코딩(Erasure Coding)[2] 방식이 최근 확산되고 있다. 하지만 소거코딩 방식은 복제 방식에 비해 성능이 떨어지는 단점이 있으며 이를 해결하기 위한 연구도 활발하게 진행되고 있다[11, 12].

본 논문은 스토리지의 확장성을 높이기 위해 스위치가 필요 없는 토러스[3] 네트워크를 스토리지 네트워크로 사용하면서 데이터 신뢰성 지원방법으로 디스크의 활용률을 높일 수 있는 소거코딩 방식을 지원하는 분산 스토리지 구조를 제안한다. 제안된 구조의 목표는 기존의 Fat-Tree 기반의 분산 파일시스템에 비해서 성능이 떨어지지 않으면서도 디스크의 활용률을 높이는 것이다.

대표적인 동영상 서비스 업체인 유튜브[4]의 경우 정확한 스토리지 용량은 밝히지 않지만 추측을 통해 적어도 1엑사바이트 이상일 것이라고 짐작된다. 1엑사바이트의 데이터를 저장하기 위해서는 4TB의 하드디스크를 100개씩 장착할 수 있는 서버를 사용하더라도 2500대 이상의 서버가 필요하다. 이와 같은 대량의 스토리지 서버를 스위치를 이용하는 Fat-Tree 네크워크로 연결하려면 엄청난 개수의 스위치가 필요하며 이는 곧 비용의 문제로 귀결되어 저비용 스토리지 구축의 걸림돌이 된다. 이러한 네트워크의 확장성 문제를 해결할 수 있는 방법으로 슈퍼컴퓨팅 시스템 네트워크[5]에서 주로 사용되는 토러스[3] 네트워크를 스토리지 네트워크로 사용하는 것을 고려해 볼 수 있다. 토러스 네트워크는 스위치를 사용하지 않고 각 서버 노드가 직접 연결되는 방식으로 Fat-Tree 네트워크와 다른 특성을 가지고 있으며 이를 잘 활용해야만 네트워크로 인한 시스템의 성능 저하 등의 단점을 방지할 수 있다.

대용량의 저장 공간을 제공하는 분산 파일시스템 중 가장 많이 알려진 것은 구글에서 개발한 구글 파일시스템(GFS)[6]이다. GFS[6]은 대용량의 파일이 읽기 전용으로 사용되는 경우에 최적화된 파일시스템이다. GFS[6]에서 데이터의 신뢰성을 높이는 방법으로 초기에는 복제 방식을 사용하다가 최근에는 소거코딩 방식을 적용하고 있다[7]. 복제 방식은 파일 데이터를 복제해 여러 데이터 서버들에 분산시켜 파일이 유실된 경우 다른 데이터 서버에 저장되어 있는 복제 파일로 복구하는 방식이다. 복제 방식은 대부분의 분산 파일시스템에서 기본적으로 제공하는 데이터 신뢰성 지원 방식으로, 글러스터 파일시스템(GlusterFS)[8], Ceph[9], 하둡 파일시스템(HDFS)[10] 등이 모두 사용하고 있다.

복제 방식은 구현과 복구 절차가 간단하다는 장점이 있는 반면 실제 디스크의 효율이 매우 떨어진다는 단점이 있다. 예를 들면 3중 복제를 사용하면 가용 스토리지 용량은 물리 스토리지 용량의 1/3로 줄어든다. 스토리지의 규모가 크지 않은 경우에는 디스크의 효율 저하가 큰 문제점이 되지 않지만, 엑사 규모의 스토리지 공간을 지원하는 경우에는 엄청난 비용의 증가를 야기한다. 이러한 디스크 효율 저하 문제를 해결할 수 있는 방법으로 소거코딩[2] 방식이 있다. 소거코딩 방식은 데이터를 그대로 복제하지 않고, 데이터를 인코딩하여 패리티를 생성하고 이를 저장했다가 데이터가 유실 되는 경우 남은 데이터와 패리티의 디코딩을 통해 원본 데이터를 복원하는 방식이다. 소거코딩은 복제 방식에 비해 디스크 효율이 높은 장점이 있지만, 데이터 저장 시 인코딩이 필요하고, 데이터 유실 시 디코딩을 수행하여 복원하기 때문에 입출력 성능과 데이터 복구 성능이 떨어지는 단점이 있다. 소거코딩 방식의 복구 성능을 높이기 위한 방법으로 MSR(Minimum-storage regenerating)[11]과 Pyramid Codes[12]등의 연구가 있다.

본 논문은 확장성이 높은 토러스 네트워크 기반으로 분산 스토리지 시스템을 구현할 때 발생하는 여러 문제점과 특성들을 살펴 볼 것이며 이를 가장 효율적으로 활용할 수 있는 구조와 모델을 제안한다. 제안된 구조는 프로토타입 구현을 통하여 기존의 분산 스토리지 시스템과 성능을 비교하여 제안된 구조의 적절성과 효과를 확인한다.

2. 토러스 네트워크

토러스[3] 네트워크는 네트워크 스위치 없이 네트워크 노드들을 연결하는 방법으로 Fat-Tree에 비해 높은 확장성을 가질 수 있다. 2D-토러스 네트워크는 메시 토폴로지와 비슷하지만, 가장 자리의 노드가 반대편 가장 자리의 노드와 연결되어 있는 점이 다

르다. 토러스 네트워크에서는 모든 노드가 상대적인 거리의 차이만 존재할 뿐 노드의 절대적인 위치간의 차이는 없다. 토러스 네트워크는 Fat-Tree 네트워크에 비해 네트워크 스위치가 필요 없다는 장점이 있는 반면, 노드와 노드간의 네트워크의 거리가 노드간의 위치 별로 서로 상이하며, 네트워크 홉이 늘어날수록 지연시간도 증가하는 단점이 있다. 토러스 네트워크는 모든 노드가 라우터의 역할을 해 별도의 네트워크 스위치가 필요 없다. 토러스 네트워크의 장점인 네트워크 스위치의 불필요성을 이용해 엑사 규모의 스토리지 네트워크로 사용하는 것을 고려할 수 있다. 반면에 토러스 네트워크에서는 노드간의 지연시간의 상이함과 지연시간의 증가가 발생할 수 있는데, 이를 고려한 설계가 있어야 토러스 네트워크를 분산 스토리지 시스템에 적용할 수 있을 것이다.

토러스 네트워크의 가장 큰 문제점은 노드간의 위치에 따른 상이한 지연시간과 대역폭이다. 인접한 노드간의 네트워크 홉 수는 '0'이며 두 노드 사이에 하나의 노드를 거쳐서 가는 경우는 네트워크 홉 수가 '1'이다. 토러스 네트워크에서는 각 노드가 라우터의 역할을 하게 되는데, 라우팅에 따른 지연 시간이 크지 않다면 지연 시간은 노드 간의 홉 수에 정비례하여 증가할 것이며, 라우팅 부하가 크다면 정비례를 넘어서는 증가 양상을 보일 것이다.

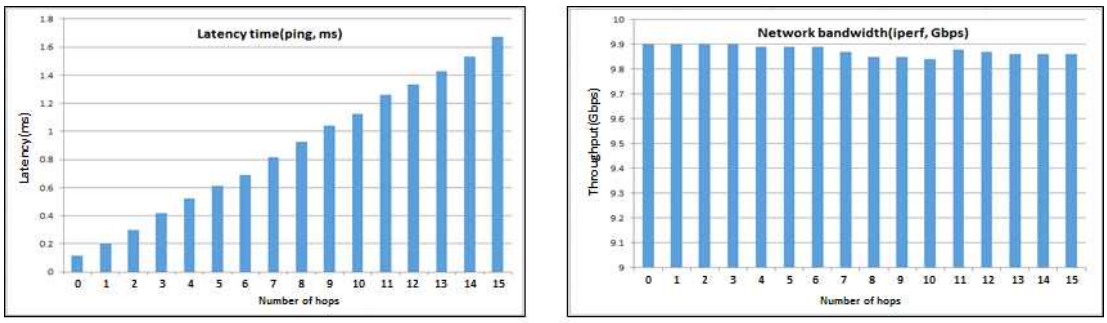
Fig. 1은 토러스 네트워크를 가정하여 16개 노드를 직접 연결한 후 네트워크 홉 수를 증가시키면서 지연시간과 대역폭을 측정한 그래프이다. 사용된 네트워크는 10Gbps의 이더넷이다.

Fig. 1의 (a)의 지연시간은 네트워크 홉수에 정비례하여 증가함을 알 수 있으며, (b)의 대역폭은 홉 수가 늘어남에도 불구하고 1% 이내의 성능 저하만이 발생함을 알 수 있다. 위의 실험 결과를 통해 네트워크 대역폭은 큰 문제가 되지 않지만 지연시간은 예상대로 홉 수에 따라 증가하기 때문에 데이터의 입출력 시에 이를 고려하여 근접한 서버 간에만 통신이 발생하도록 설계하는 것이 필요하다.

3. 소거코딩의 성능저하 원인

분산 스토리지 시스템은 로컬 파일시스템과 달리 다수의 스토리지 노드에 데이터를 분산하여 저장한다. 한대의 노드(혹은 디스크) 보다는 다수의 노드(혹은 디스크)에서 고장이 발생할 확률이 높기 때문에 스토리지 노드의 수가 많을수록 노드(혹은 디스크)의 고장 확률도 높아진다. 노드의 고장에 따른 데이터의 유실을 막기 위해서 분산 스토리지 시스템에서 데이터의 내결함성을 높이는 방법으로 크게 두 가지의 방법을 사용하고 있다.

가장 일반적으로 사용하는 복제(replication) 방식은 동일한 데이터를 다수의 노드나 디스크에 중복 저장하는 방법이다. 하나의 노드가 불능이 되더라도 동일한 데이터를 저장한 다른 노드가 있어 데이터의 유실을 방지할 수 있다. GFS[6]을 비롯한 다수의 분산 파일시스템이 복제 방식을 지원하며 일반적으로 두 개의 노드 고장까지 데이터를 보장할 수 있는 3중 복제를 지원한다. 복제 방식은 구현과 메타데이터 관리가 비교적 간단하다는 장점이 있는 반면, 동일한 데이터를 여러 개 복제함으로써 디스크의 활용률이 떨어지는 단점을 내포하고 있다. 3중 복제의 경우 물리적 디스크 용량이 100이라고 가정하면 실제 유효 저장 용량은 100/3 = 33.3이 되며, 디스크 활용



(a) Latency (b) Bandwidth

Fig. 1. Latency and bandwidth of torus network by the number of network hop.

률은 33.3%이다. 이러한 디스크 활용률 저하의 단점을 상쇄할 수 있는 방법으로 사용되는 것이 소거코딩 방식이다.

소거코딩 방식은 Raid-5에서 사용하는 패리티 기법과 비슷한 개념으로 데이터를 특정 크기 단위로 인코딩을 수행해 패리티를 생성하고 이를 서로 다른 노드나 디스크에 저장하는 방식이다. 소거코딩에서 일반적으로 사용하는 형식은 N=K+M로 표현되는데, K은 데이터 블록의 개수이며 M은 패리티 블록의 개수이다. 블록의 크기는 임의로 결정할 수 있다. 소거코딩에서는 N개의 블록 중 임의의 K개의 블록만 있으면 원본 데이터를 복원할 수 있다. 일반적으로 K의 크기가 M보다 크게 설정되며 디스크의 활용률은 K/N로 계산된다. 예를 들어 10+6의 경우는 10/16=62.5%의 디스크 활용률을 나타낸다. 10+6의 소거코딩은 16개의 디스크나 노드를 필요로 하며 이 중 6개까지의 디스크나 노드 실패를 허용할 수 있다. 이는 복제 방식에 비해 훨씬 높은 내결함성 보장 수준으로 복제 방식에서 동일한 수준을 보장하기 위해서는 7중 복제를 지원해야 가능한 수준이다. 소거코딩 방식이 복제 방식에 비해 높은 디스크 활용률을 지원하면서도 고 수준의 내결함성을 보장할 수 있지만, 이에 반해 여러 단점을 가지고 있다.

첫 번째 소거코딩의 단점은 데이터 저장 시 마다 인코딩을 수행해서 패리티 블록을 생성해야 하며 데이터 블록이 유실된 경우에 데이터를 복구하기 위해 디코딩을 수행해야 하는 점이다. 인코딩과 디코딩은 대부분 CPU 작업으로 CPU 성능이 충분하지 않은 경우 처리에 많은 시간이 걸릴 수 있으며 이는 입출력 성능을 저하시킬 수 있다.

두 번째 소거코딩의 단점은 단일 입출력 요청을 처리하기 위하여 여러 디스크나 노드에서 데이터 블록을 읽어 와야 하는 점이다. 10+6의 소거코딩이라면 스트라이프 크기의 데이터를 읽기 위해서는 10개의 디스크나 노드에서 데이터를 읽어야 한다. 스트라이프의 크기가 10KB라고 가정하면 1KB씩 10개의 디스크나 노드에서 데이터를 읽어야 한다. 일반적으로 네트워크 통신 시 큰 데이터를 한 번에 전송하는 것이 작은 데이터를 여러 번 전송하는 것 보다 효율적이기 때문에 복제 방식에 비해서 입출력 성능이 저하될 수 있다.

위에서 언급한 소거코딩의 성능상의 단점으로 인해 복제 방식에 비해 소거코딩 방식은 비교적 고성능을 필요로 하지 않는 아카이브 용도나 백업 스토리지 등 읽기 전용 스토리지로 많이 사용되고 있다.

3.1 인코딩/디코딩 부하

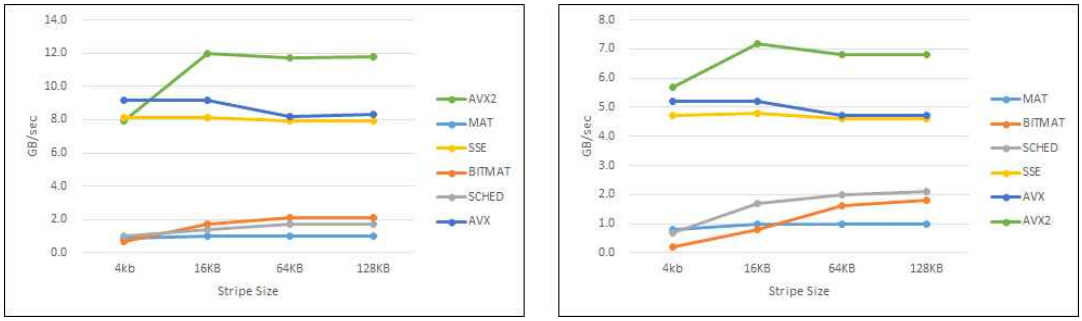
인코딩/디코딩은 CPU에 구애 받지 않고 사용할 수 있는 소프트웨어 방식과 특정 CPU의 명령어를 이용하는 하드웨어 가속 방식이 있다. 소프트웨어 인코딩/디코딩으로 가장 많이 알려진 것이 Jerasure [13] 라이브러리이며, 하드웨어 가속 방식으로는 Intel의 ISA-L(Intel Storage Acceleration Library) [14]이다. ISA-L은 인코딩/디코딩 SIMD(Single Instruction Multiple Data) 가속 명령어인 SSE, AVX, AVX2 명령을 이용하여 구현된 라이브러리이다.

테스트베드 시스템에서 인코딩/디코딩 성능을 측정하여 입출력 성능을 충분히 지원할 수 있는지 여부를 확인하였다. 테스트베드의 사양은 아래 Table 1과 같다.

Fig. 2는 Reed-Solomon 알고리즘으로 스트라이프의 크기를 변화시켜가며 인코딩/디코딩을 수행하였을 때의 라이브러리 별 인코딩/디코딩의 성능이다. SSE, AVX, AVX2는 인텔의 ISA 라이브러리를 이용한 결과이며, SCHED, BITMAT, MAT은 Jerasure[13] 라이브러리의 인코딩 방법들이다. 인코딩/디코딩 성능은 스트라이프 크기가 클수록 성

Table 1. Testbed specifications

Items	Specifications	
CPU	Intel Xeon E5-2623 v3 3.0Ghz, 4-core X 2-socket	SSE, AVX, AVX2 available
Memory	32GB	
Network	10Gbps Ethernet X 8	
DISK	Sata3 256GB HDD	
Operating system	CentOS 7.0	



(a) Encoding Performance (b) Decoding Performance

Fig. 2. Encoding/decoding performance of testbed system.

능이 향상됨을 볼 수 있다. SSE, AVX, AVX2와 같이 SIMD 명령어를 이용하는 ISA-L[14]이 BITMAT, SHED, MAT과 같이 소프트웨어적으로 처리하는 Jerasure 라이브러리보다 성능이 훨씬 우수함을 확인할 수 있다. ISA-L을 이용하는 경우에는 최소 8GB/s의 인코딩 성능과 4.5GB/s의 디코딩 성능을 기대할 수 있음을 실험결과로 확인하였다. 테스트베드의 네트워크 환경은 10Gbps의 이더넷으로, ISA-L의 최저 성능인 4.5GB/s(=36Gbps)는 10Gbps를 훨씬 넘어서 Infiniband의 30Gbps 성능도 넘어서는 수준이다. 따라서 Fig. 2의 실험결과를 통해 10Gbps의 네트워크 환경에서는 ISA의 인코딩/디코딩 성능이 충분히 빨라서 인코딩/디코딩에 따른 입출력 성능의 저하는 없을 것이라고 판단할 수 있다.

3.2 작은 데이터 병렬 전송 부하

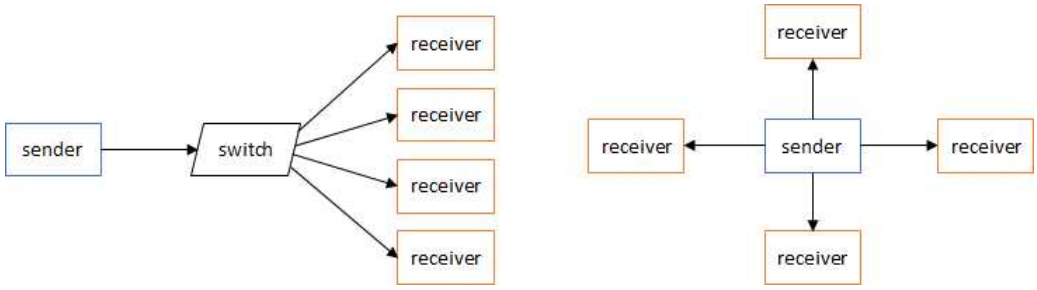
소거코딩 방식을 사용하면 복제 방식에 비해 하나의 데이터 노드에 저장하는 데이터의 단위가 작아지며 입출력 요청 시 데이터 전송 노드의 수도 늘어난다.

네트워크의 대역폭을 충분히 사용하기 위해서는 큰 데이터를 보내는 것이 작은 데이터를 여러 번 보내는 경우보다 유리한 것이 일반적이다. 따라서 복제 방식과 소거코딩방식을 직접적으로 비교했을 때 데이터 전송 효율 관점에서는 복제 방식이 유리하다고 볼 수 있다. 복제 방식의 "큰 데이터 단일 전송"이 소거코딩 방식의 "작은 데이터 다중 전송"에 비해 성능 측면에서 유리하다고 할 수 있다. 이러한 가정은 Fat-Tree 네트워크에서는 일반적으로 받아들일 수 있지만, 토러스 네트워크에서도 그대로 적용될 수 있을지는 확신할 수 없다. 왜냐하면 Fat-tree 네트워크는 하나의 서버 노드가 하나의 물리적 네트워크 포트를 가지는 반면에 토러스 네트워크에서는 하나의 서버 노드가 다수 개의 물리적 네트워크 포트(3D-토러스의 경우 6개)를 가지고 있기 때문이다. Fat-Tree에서는 다수의 프로세스나 쓰레드가 동시에 네트워크 전송을 하더라도 하나의 네트워크 포트에서 직렬화 되겠지만, 토러스 네트워크에서는 다수의 네트워크 전송이 다수의 네트워크 포트를 통해 전송될 수 있어 Fat-Tree에 비해 네트워크 전송의 병렬성을 높일 수 있는 가능성이 있다. 이러한 추측을 확인하기 위하여 "큰 데이터 단일 전송"과 "작은 데이터 병렬 전송"을 Fat-Tree 네트워크와 토러스 네트워크에서 동일한 내용으로 측정해 보았다.

Fig. 3은 "큰 데이터 단일 전송"과 "작은 데이터 병렬 전송" 시험 환경을 보여준다. Fig. 3의 (a)는 Fat-tree의 환경으로 송신 서버와 수신 서버는 서로 스위치를 사이에 두고 연결되어 있다. Fig. 3의 (b)는 토러스 네트워크 환경으로 송신 서버와 수신 서버는 스위치를 거치지 않고 직접 연결되어 있다.

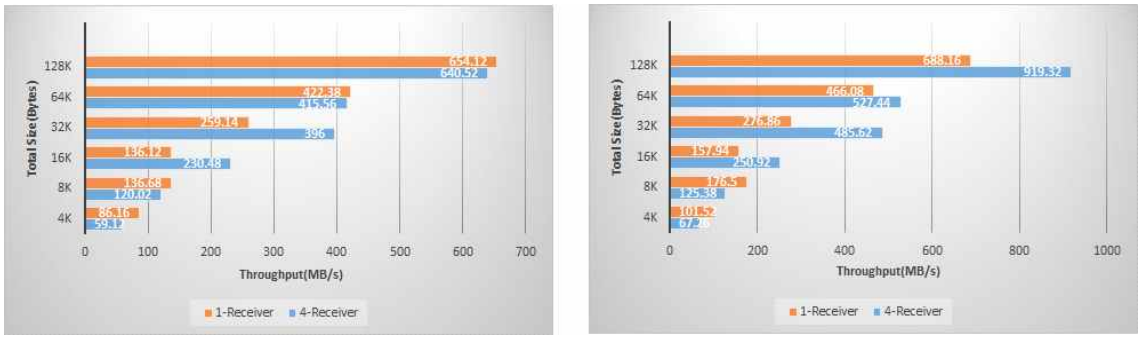
실험은 Fig. 3의 (a)와 (b)의 환경에서 송신 서버가 하나의 데이터를 단일 수신 서버에 보냈을 때의 성능과 해당 크기의 데이터를 4개로 쪼개어 4개의 수신 서버에 동시에 보냈을 때의 성능을 측정하였다.

Fig. 4는 Fig. 3의 실험 결과이다. Fig. 4의 (a)인 Fat-Tree의 경우는 "큰 데이터 단일 전송"이 전반적으로 "작은 데이터 병렬 전송"보다 우수 했으나 8KB와 4KB의 메시지에서는 성능이 역전되어 나타나는 예외적인 현상이 있었다. 예외현상은 메시지의



(a) Fat-Tree data transmission (b) Torus data transmission

Fig. 3. Transmission performance test environment.



(a) Fat-Tree transmission results (b) Torus transmission results

Fig. 4. Throughput of single transmission and four segments parallel transmission.

크기가 TCP/IP 프로토콜의 특성인 슬라이딩 윈도우와 수신 버퍼의 크기에 적절히 맞아 떨어져 하나의 수신 서버일 때 보다 우수한 성능이 나오는 것으로 추측된다. 하지만 전반적으로 단일 전송이 현격한 차이를 보이지는 않는 수준에서 우수한 성능을 보여주었다. Fig. 4의 (b)는 토러스 네트워크의 경우로 8KB 까지는 단일 전송이 우수한 성능을 보였지만 16KB 이상에서는 병렬 전송이 우수하였으며 또한 현격한 차이를 보여주었다. 이는 메시지가 아주 작지만 않으면 "작은 데이터 병렬 전송"이 토러스 네트워크에서는 더 효과적임을 보여준다.

4. 토러스 네트워크에 적합한 소거코딩 지원 분산 파일시스템 구조

소거코딩을 지원하는 분산 파일시스템은 인코딩/디코딩을 수행하는 주체에 따라 두 가지 구조로 나누어 볼 수 있다. GlusterFS[8]의 경우는 클라이언트에서 인코딩과 디코딩을 수행하여 데이터 서버에 병렬로 데이터를 전송하며, Ceph[9]의 경우는 데이터 서버측에서 인코딩과 디코딩을 수행한다[15]. Ceph[9]에서는 인코딩/디코딩을 담당하는 서버를 Primary OSD라고 한다. Primary OSD는 클라이언트의 쓰기 요청 시 데이터를 인코딩하여 스트라이프 저장으로 할당된 다른 OSD 서버에 데이터를 분산 저장하고, 읽기 요청 시에는 다른 OSD 서버에서 데이터를 읽어 모아서 클라이언트로 전달한다. GlusterFS[8]과 Ceph[9]은 모두 Fat-Tree 기반으로 설계되어 인코딩/디코딩을 담당하는 서버가 클라이언트가 되건 스토리지 쪽이 되건 문제가 되지 않는다. 하지만 본 논문에서 제안하는 분산 스토리지는 토러스 네트워크를 기반으로 하기 때문에 토러스 네트워크의 특성을 고려한 설계가 필요하다.

Fig. 5은 본 논문에서 제안하는 토러스 기반의 분산 스토리지와 클라이언트의 네트워크 연결 모델이다. Fig. 5에서와 같이 스토리지 서버들은 서로 토러스 네트워크로 연결되어 있고 맨 앞쪽의 프론트엔드-티어 스토리지 서버들만이 스위치를 통해 클라이

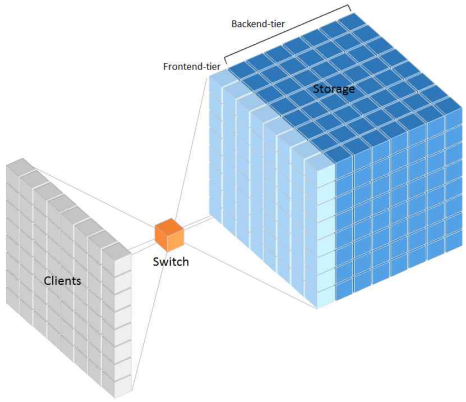


Fig. 5. Network architecture of torus based distributed storage.

언트와 Fat-Tree로 연결되어 있는 구조이다. 프론트엔드-티어의 스토리지를 제외한 백엔드-티어의 스토리지는 클라이언트와 네트워킹하기 위해서는 하나 이상의 스토리지 서버들을 거쳐야 한다. 이러한 구조에서 클라이언트와 서버 노드들간에 개별적인 네트워킹이 필요한 클라이언트측 인코딩/디코딩 구조는 매우 비효율적일 수밖에 없다. 이러한 이유로 토러스 기반의 분산 파일시스템에서는 Ceph[15]과 같이 스토리지 서버 측에서 인코딩/디코딩과 scattering/gathering을 담당한다. 본 논문에서는 인코딩/디코딩을 담당하는 서버를 프라이머리 데이터 서버라고 부른다.

Fig. 6은 프라이머리 데이터 서버 기반의 구조에서 클라이언트가 쓰기 요청을 했을 때 데이터의 흐름을 보여준다. 클라이언트의 데이터는 백엔드-티어의 프라이머리 데이터 서버로 전송된 다음 인코딩

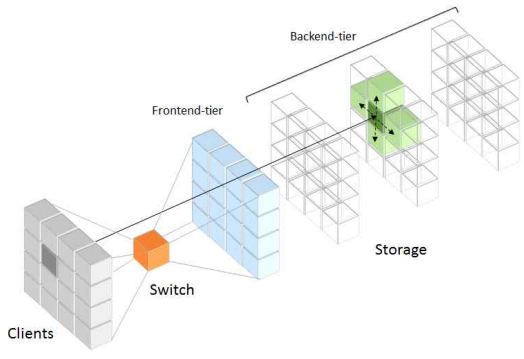


Fig. 6. Write operation of primary data server centric model.

을 거쳐 스트라이프를 저장할 데이터 서버로 전송된다. 이렇게 되면 클라이언트에서 직접 전송할 때에 비해 데이터가 거치게 되는 네트워크 홉의 수가 훨씬 줄어들어 토러스 네트워크의 단점인 지연시간 증가를 줄일 수 있게 된다. Fig. 6을 예로 들어 전체 5개의 데이터 서버에 데이터를 분산하는 경우에 클라이언트 중심 구조와 프라이머리 서버 중심 구조의 전체 네트워크 홉수의 합은 아래와 같다.

- 클라이언트 중심 구조 시 네트워크 홉 수: 2 hop * 5 = 10 hop
- 프라이머리 서버 중심 구조 시 네트워크 홉 수: 2 hop * 1 = 2 hop

클라이언트 중심 구조에서는 각각의 데이터 서버에 대해 2 홉이 발생하지만, 프라이머리 서버 중심 구조에서는 프라이머리 서버까지의 2 홉만 발생한다. 따라서 프라이머리 서버 중심 구조가 토러스 환경에 적합함을 확인 할 수 있다.

5. 성능 시험 및 결과 분석

앞서 4장까지는 분산 스토리지에서 소거코딩을 지원할 때의 성능 저하의 요인으로 보이는 요소들에 대한 분석과 토러스를 기반으로 소거코딩을 지원할 때의 효율적인 구조에 대해 살펴보았다. 이번 장에서는 앞서 분석한 요소들을 기반으로 분산 스토리지 시스템의 프로토타입을 구현하여 이의 성능을 측정하였다. 프로토타입 시스템의 성능은 이를 실제 구현했을 때 기대할 수 있는 시스템의 성능을 가늠하기 위한 목적이다.

5.1 시험 내용

다음은 구현된 프로토타입 시스템의 세부기능이다.

- 시스템은 클라이언트, 프라이머리 데이터 서버, 데이터 서버들로 구성
- 클라이언트는 FUSE[16]를 기반함
- 프라이머리 데이터 서버는 클라이언트의 쓰기 요청을 받아 write-back 방식으로 동작
- 클라이언트는 프라이머리 데이터 서버의 고장을 대비하여 쓰기 데이터를 버퍼링 함
- 프라이머리 데이터 서버는 쓰기 요청시 데이터의 인코딩을 수행
- 프라이머리 데이터 서버는 데이터 서버와 병렬

IO를 수행

- 프라이머리 데이터 서버는 데이터 서버의 위치를 고려하여 direct 전송과 파이프라인 전송을 병행
- 메타데이터 연산은 파일생성과 삭제 외에는 지원 않음

성능시험은 Table 1의 사양을 가지는 서버들이 사용되었다. 프로토타입 시스템의 성능은 GlusterFS[8]과 Ceph[9]의 성능과 비교하였다. 양쪽은 모두 1대의 클라이언트와 10대의 데이터 서버로 구성되어 있다. 프로토타입 시스템의 데이터 서버는 토러스 네트워크로 구성되어 있으며 GlusterFS[8]과 Ceph[9]은 Fat-Tree 네트워크로 연결되어 있는 것이 차이점이다.

Fig. 7은 프로토타입 시스템의 네트워크 연결구조를 보여준다. 클라이언트는 프라이머리 데이터 서버와 스위치를 통하여 직접 연결되어 있으며 데이터 서버들은 프라이머리 데이터 서버 주위에 배치되어 있다. 이는 최상의 입출력 성능을 얻기 위한 데이터 서버 배치이다.

성능 시험에 사용된 소거코딩과 복제 방식의 구성은 아래와 같다.

- 소거코딩: K+M(8+2), 스트라이프 크기(5KB), 스트라이프 유닛 크기(512Byte), 인코딩 알고리즘(Reed-Solomon)
- 복제 방식: 3중 복제

  복제 방식은 GlusterFS[8]과 Ceph[9]에서 소거코딩과 복제 방식의 성능을 모두 비교해 보기 위해 사용되었다.

시험에는 IOzone[17] 벤치마크 도구를 사용하였으며, 성능 시험은 쓰레드의 개수를 1개와 10개로 변경하면서 순차 읽기/쓰기, 랜덤 읽기/쓰기를 수행하였다.

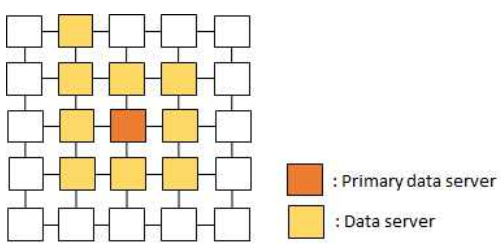


Fig. 7. Network configuration of prototype system.

5.2 시험 결과

5.2.1 순차 입출력 성능

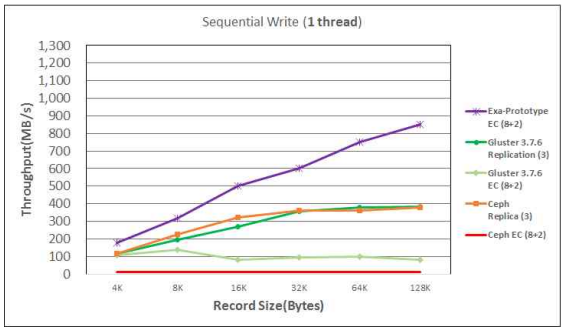
Fig. 8은 클라이언트에서 순차 입출력을 요청하였을 때의 성능이다. 프로토타입 시스템의 성능이 대부분 가장 우수하나 (b)의 쓰기 성능에서는 레코드 크기가 8KB 이하인 경우에 Ceph-replica와 GlusterFS[8]의 EC 성능이 더 우수한 결과를 보여주었다. 이는 Fig. 4의 (b)의 결과에서 보듯이 8KB 이하의 메시지 크기에서는 '큰 데이터 단일 전송'이 '작은 데이터 병렬 전송' 보다 우수한 성능을 보였기 때문으로 판단된다. Fig. 8의 (a)의 단일 쓰레드 읽기 성능은 Ceph[9]의 복제 방식이 가장 뛰어난 성능을 보여주었다. 하지만 Ceph[9]의 복제 방식의 경우는 (b)에서 볼 수 있듯이 10개의 쓰레드 읽기 성능에서도 전체 성능이 크게 향상되지 않고 약 700MB/s에 제한되는 것을 알 수 있다. 이는 아마도 클라이언트와 스토리지 서버와의 입출력 처리 구조에서 오는 제약으로 보인다.

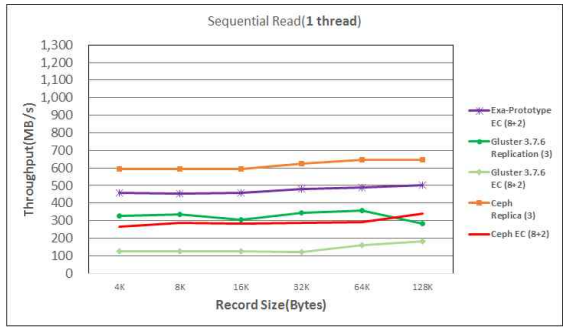
Fig. 8의 쓰기 성능에 있어서 프로토타입 시스템은 Ceph[9]이나 GlusterFS[8]과 달리 캐싱이나 미리 읽기 등의 부가적인 기능이 추가되지 않았기 때문에 레코드의 크기가 커질수록 쓰기 성능이 점진적으로 증가하는 경향을 보여준다. Fig. 8의 읽기 성능의 경우에는 리눅스의 VFS(Virtual File System) 계층에서 순차 읽기를 감지하여 미리 읽기를 수행하기 때문에 IOzone의 레코드의 크기에 상관없이 128KB의 레코드 크기의 읽기 성능이 모든 레코드 크기에서 나타나지만, GlusterFS-replica의 경우에만 64KB 이상에서 성능이 갑자기 증가하는 결과를 보이는 것이 특이하였다.

본 실험을 통해 토러스 기반의 소거코딩을 지원하는 프로토타입의 성능이 GlusterFS[8]나 Ceph[9]의 소거코딩보다는 압도적으로 우수하며, 복제 방식의 성능에 비해서도 대부분 우수함을 확인할 수 있었다. 본 실험에서 우리는 토러스 네트워크 기반의 분산 스토리지 구조에서 소거코딩 방식으로도 복제 방식을 능가하는 순차 입출력 성능을 기대할 수 있음을 확인할 수 있다.

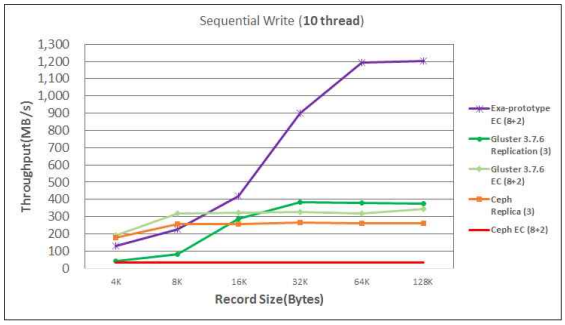
5.2.2 랜덤 입출력 성능

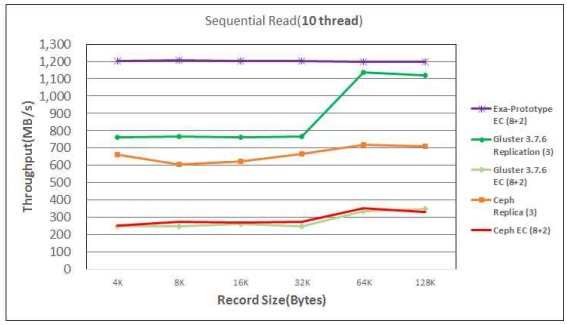
Fig. 9는 4KB 레코드의 랜덤 입출력 성능의 결과





(a) Single thread sequential IO





(b) 10 thread sequential IO

Fig. 8. Throughput of sequential write/read.

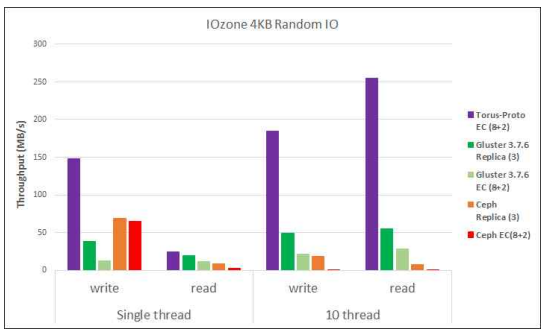


Fig. 9. Throughput of random write/read

이다. 랜덤 성능은 프로토타입 시스템의 성능이 모든 경우에 우수하였으며 단일 쓰레드 읽기를 제외하고는 압도적인 성능 차이를 보여주었다. 이는 아마도 프로토타입 시스템이 별도의 메타데이터 연산을 하지 않기 때문일 것으로 판단된다. GlusterFS[8]과 Ceph[9]의 경우는 파일에 쓰기나 읽기가 수행되는 경우에 이를 위한 메타데이터 연산이 필요하지만, 프로토타입 시스템은 이를 생략하였기 때문이다. 그럼에도 불구하고 본 실험에서 확인할 수 있는 것은 토러스 네트워크 기반의 소거코딩 지원 분산 스토리지 시스템에서도 랜덤 입출력에 대해 세밀한 구현이 된다면 충분히 높은 성능을 기대할 수 있음과 소거코딩을 사용할 때의 순수한 입출력 성능을 확인할 수 있었다는 점이다.

6. 결 론

본 논문은 대용량의 멀티미디어 파일을 저장하기 위해서 엑사스케일 급의 저장 용량을 지원할 수 있는 분산 스토리지로 토러스 네트워크 기반의 분산 스토리지 시스템을 제안하였다. 분산 스토리지는 데이터의 내결함성 지원을 위하여 복제나 소거코딩을 지원한다. 소거코딩 방식은 스토리지의 효율성을 높일 수 있는 반면에 입출력 성능이 떨어지는 단점이 있다. 본 논문에서는 네트워크 스위치가 필요 없는 토러스 네트워크 기반의 분산 스토리지 시스템에서 소거코딩 방식을 지원하면서도 복제 방식에 버금가는 성능을 낼 수 있는 구조를 제안하고 그 가능성을 프로토타입 시스템을 통해 확인하였다. 토러스 네트워크는 노드간 지연시간이 상이하면서 길어지는 단점이 있으나 본 논문은 이러한 단점을 상쇄할 수 있

는 구조로 소거코딩을 구현하면 복제 방식을 능가하는 좋은 입출력 성능을 기대할 수 있음을 확인할 수 있었다.

본 논문의 프로토타입 시스템은 메타데이터 연산 과정을 제외한 입출력 구조만을 시험하였기 때문에 실제 시스템과는 차이가 있다. 또한 실험에서 볼 수 있듯이 작은 레코드의 쓰기에서는 높은 성능을 기대하기 어렵기 때문에 이의 성능을 보완해 줄 수 있는 추가적인 기술이 필요함을 알 수 있었다.

본 논문은 토러스 네트워크 기반으로 효율적인 대용량의 멀티미디어 데이터를 저장할 수 있는 소거코딩 기반의 분산 스토리지 시스템의 구조를 제안하였으며. 프로토타입 시스템의 성능 시험을 통해 이러한 목표가 불가능하지 않음을 확인하였다.

REFERENCE

[1] Young Jin Nam, Soon Hwan Jeong, and Young Kyun Park. "A Hybrid Storage Architecture with a Content Caching Algorithm for Networked Digital Signage." Journal of Korea Multimedia Society, Vol. 15, No. 5, 2012.05. (pp. 651-663)

[2] J.S. Plank, and C. Huang, "Tutorial: Erasure Coding for Storage Applications," *Proceeding of Slides Presented at FAST-2013: 11th Usenix Conference on File and Storage Technologies*, Feb. 2013.

[3] Torus Interconnect, https://en.wikipedia.org/wiki/Torus_interconnect, (Accessed July, 17, 2016)

[4] P. Gill, M. Arlitt, Z. Li, and A. Mahanti, "Youtube traffic characterization: a view from the edge." Proceedings of the 7th ACM SIGCOMM conference on Internet measurement. ACM, 2007. (pp. 15-28)

[5] M. Blumrich, D. Chen, P. Coteus, A. Gara, M. Giampapa, and P. Heidelberger, et al, *Design and Analysis of the BlueGene/L Torus Interconnection Network*, Vol. 3, IBM Research Report RC23025 (W0312- 022), 2003.

[6] S. Ghemawat, H. Gobioff, and S.T. Leung, *The Google File System*, ACM SIGOPS Operating Systems Review, ACM 1-58113-757-5/03/0010, 2003.

[7] D. Ford, F. Labelle, F. I. Popovici, M. Stokely, V. A. Truong, L. Barroso et al., *Availability in Globally Distributed Storage Systems*, OSDI. 2010.

[8] GlusterFS, http://www.gluster.com/, (accessed July, 5, 2016).

[9] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. Long, and Carlos Maltzahn, "Ceph: A Scalable, High- performance Distributed File System," *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, USENIX Association, 2006. (pp. 307-320)

[10] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," *Proceeding of Mass Storage Systems and Technologies, 2010 IEEE 26th Symposium on IEEE*, 2010. (pp. 1-10)

[11] K.V. Rashmi, N.B. Shah, and P.V. Kumar, "Optimal Exact-regenerating Codes for Distributed Storage at the MSR and MBR Points via a Product-matrix Construction," , *IEEE Transactions on Information Theory*, Vol. 57, No. 8, 2011. (pp. 5227-5239)

[12] C. Huang, M. Chen, and J. Li, "Pyramid Codes: Flexible Schemes to Trade Space for Access Efficiency in Reliable Data Storage Systems," *ACM Transactions on Storage*, Vol. 9, No. 1, 2013. (pp. 3:1-3:28)

[13] 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.

[14] Intel ISA-L, https://software.intel.com/en-us/storage/ISA-L, (Accessed July, 20, 2016)

[15] Ceph Erasure Coding Introduction, https://software.intel.com/en-us/blogs/2015/04/06/ceph-erasure-coding-introduction, (Accessed July,

6, 2015).
[16] FUSE File System, https://github.com/libfuse/libfuse, (Accessed July, 10, 2015)
[17] IOzone Filesystem Benchmark, http://www.iozone.org/, (Accessed July, 15, 2015)

김 재 열

1999년 경북대학교 전자공학과(학사)
2001년 경북대학교 전자공학과(석사)
2013년 경북대학교 전자전기컴퓨터학부(박사수료)

2001년～현재 한국전자통신연구원 책임연구원
관심분야 : 분산 파일시스템, 디스크 캐시 기술, 가상화 기술

김 동 오

2000년 건국대학교 컴퓨터공학(학사)
2002년 건국대학교 컴퓨터·정보통신공학과(석사)
2006년 건국대학교 컴퓨터·정보통신공학과(박사)

2006년～2009년 건국대학교 강의교수
2009년～현재 한국전자통신연구원 선임연구원
관심분야 : 고성능 분산 파일 시스템, 데이터베이스, 공간 데이터 관리

김 홍 연

1992년 인하대학교 통계학과(학사)
1994년 인하대학교 전자계산학과(석사)
1999년 인하대학교 전자계산학과(박사)

1999년～현재 한국전자통신연구원 스토리지시스템연구실장/책임연구원
관심분야 : 분산 파일시스템, 클라우드 컴퓨팅, 빅데이터, 가상 데스크탑 인프라

김 영 균

1991년 전남대학교 전산통계학과(학사)
1993년 전남대학교 전산통계학과(석사)
1995년 전남대학교 전산통계학과(박사)

1996년～현재 한국전자통신연구원 고성능컴퓨팅연구부장/책임연구원
2014년～현재 한국정보과학회 컴퓨터시스템연구회/ 데이터베이스 소사이어티 운영위원
관심분야 : 데이터베이스 시스템, 파일시스템, 클라우드 컴퓨팅

서 대 화

1981년 경북대학교 전자공학과(학사)
1983년 한국과학기술원 전산학과(석사)
1993년 한국과학기술원 전산학과(박사)

1983년～1995년 한국전자통신연구원
1995년～현재 경북대학교 IT대학 전자공학부 교수
2004년～현재 경북대학교 임베디드 소프트웨어연구센터 센터장
관심분야 : 임베디드 SW, 병렬처리, 분산운영체제