논문번호 12-37C-08-05

한국통신학회논문지 '12-08 Vol.37C No.08
http://dx.doi.org/10.7840/kics.2012.37C.8.670

클라우드 스토리지 최적화를 위한 고속 캐싱 및 대용량 파일 전송 기법

김 태 훈◆, 김 정 한*, 엄 영 익°

A Scheme on High-Performance Caching and High-Capacity File Transmission for Cloud Storage Optimization

Tae-hun Kim◆, Junghan Kim*, Young Ik Eom°

요 약

최근 클라우드 컴퓨팅 환경의 보급과 함께 스토리지의 데이터양이 급증함에 따라 그에 따른 스토리지 저장 비용이 빠르게 증가하고 있다. 더불어, 사용자들의 다양한 서비스 및 데이터 요청으로 클라우드 스토리지의 부하 또한 급증하고 있다. 이러한 문제를 해결하기 위해 분산 파일 시스템을 통한 저비용 고성능 스토리지 환경을 제공하고자 하는 기존의 연구가 있었으나, 이에는 데이터 병렬처리, 임의위치 접근처리, 빈번한 작은 워크로드 접근처리 등의 취약점이 존재한다. 최근에는 캐싱 기술을 이용하여 이를 개선하려는 연구가 주목받고 있다. 본 논문에서는 분산 파일 시스템 환경에서 병렬 캐싱, 분산 캐싱과 공유 자원을 고려한 데이터 병렬 전송방법을 제공하는 CHPC(Cloud storage High-Performance Caching) 구조를 제안하며, 또한 이를 기존의 방법들과 비교 평가하여 스토리지 부하를 최적화하는 방법을 제시한다. 더불어, 제안 기법이 기존 클라우드 시스템에 비하여 스토리지 서버의 디스크 입출력 감소, 서버로 데이터의 요청이 집중되어 발생하는 병목현상 방지, 각 클라이언트의 중복되는 페이지 캐시 제거, 데이터 전송률 향상의 장점을 가짐을 보인다.

Key Words : 분산 파일 시스템, 고속 캐싱, 클라우드 스토리지, 블룸 필터, 병렬 전송

ABSTRACT

The recent dissemination of cloud computing makes the amount of data storage to be increased and the cost of storing the data grow rapidly. Accordingly, data and service requests from users also increases the load on the cloud storage. There have been many works that tries to provide low-cost and high-performance schemes on distributed file systems. However, most of them have some weaknesses on performing parallel and random data accesses as well as data accesses of frequent small workloads. Recently, improving the performance of distributed file system based on caching technology is getting much attention. In this paper, we propose a CHPC(Cloud storage High-Performance Caching) framework, providing parallel caching, distributed caching, and proxy caching in distributed file systems. This study compares the proposed framework with existing cloud systems in regard to the reduction of the server's disk I/O, prevention of the server-side bottleneck, deduplication of the page caches in each client, and improvement of overall IOPS. As a results, we show some optimization possibilities on the cloud storage systems based on some evaluations and comparisons with other conventional methods.

※ 본 연구는 지식경제부 및 한국산업기술평가관리원의 산업융합원천기술개발사업(정보통신)의 일환으로 수행하였음. [10041244, 스마트TV 2.0 소프트웨어 플랫폼]
◆ 주저자 : 성균관대학교 전자전기컴퓨터공학과, kth1224@ece.skku.ac.kr, 학생회원
* 성균관대학교 전자전기컴퓨터공학과, junghan@ece.skku.ac.kr, 정회원
° 성균관대학교 정보통신대학 컴퓨터공학과, yieom@skku.edu, 정회원
논문번호 : KICS2012-04-195, 접수일자 : 2012년 4월 14일, 최종논문접수일자 : 2012년 8월 6일

Ⅰ. 서 론

최근 클라우드 컴퓨팅 환경의 보급과 함께 스토리지에 저장되는 데이터의 양이 급증함에 따라서 이에 따른 스토리지 저장 비용이 빠르게 증가하고 있으며 이러한 증가 추세는 2020년까지 50배가 증가할 것으로 전망되고 있다. 또한 이에 따라 급증하는 서버의 데이터와 사용자들의 서비스 요청으로 클라우드 스토리지의 부하 또한 급증하고 있다[1].

이와 같은 문제를 해결하기 위해서 네트워크상의 사용자 요청과 데이터를 분산 처리하여 저비용/고성능을 확보하고자 하는 분산 파일 시스템을 이용한 기존의 연구들이 있다[2-5].

그러나 클라우드 컴퓨팅 환경과 같이 대규모의 분산 데이터 환경에서는 분산 파일 시스템만으로는 데이터 접근에 대한 병렬 처리 성능, 임의 위치에 대한 데이터 읽기 및 쓰기 성능, 작은 워크로드에 대한 빈번한 접근 시의 성능 등에서 취약점을 보인다[6].

이러한 기존 분산 파일 시스템의 문제를 개선하고자, 최근에는 HDFS(Hadoop Distributed File System)가 클라우드 스토리지 환경에서 널리 사용되고 있으며, 이는 맵리듀스(Map Reduce)를 통한 병렬 처리 및 워크로드 분산 처리와 Memcached를 통한 데이터 캐싱 기술로 데이터 처리의 낮은 지연시간과 높은 데이터 처리량을 보인다[2-4].

이와 더불어, 최근 클라우드 스토리지 환경에서는 데이터 접근의 지역성을 확보하여 성능 향상을 위한 캐싱 기술이 주목받고 있다. 이는 각 서버 노드의 가용 캐시 메모리를 공유하는 형태의 분산 캐싱과 동적 네트워크 환경에서 그리드 메모리(Grid Memory)를 이용한 병렬 캐싱과 같은 고속 캐싱 기술로 나뉜다. 또한, 클라우드 백업 파일 시스템에서 프락시 서버(Proxy Server)를 이용한 데이터 처리의 적중률을 높이는 BlueSky[7] 같은 캐싱 기술이 주목받고 있으며, 병렬로 데이터에 접근하여 데이터를 전송하는 GridFTP[8] 같은 데이터 전송 기술도 연구되고 있다[3].

본 논문에서는 분산 파일 시스템 환경에서 클라우드 스토리지를 위한 캐싱 기술 기반의 스토리지 최적화를 제공하는 CHPC(Cloud storage High-Performance Caching) 구조를 제안한다. CHPC는 원거리 통신망 환경에서의 클라이언트와 스토리지 서버 사이의 비용을 줄이는 고속 캐싱 기술과 스토리지에 빈번한 접근에 대한 병목 현상을 줄이기 위한 분산 캐싱 기술을 포함한다.

또한, 디스크 입출력의 연산을 최소화하기 위해 빈번하게 접근되는 핫 데이터를 구분하고 이와 더불어 누적된 캐싱 정보에 대한 중복 데이터 처리 기술을 제안한다. 마지막으로 클라우드 환경에서 공유되는 자원을 고려한 병렬 전송 방법에 대해 제안을 한다. 이를 기존의 NFS, SAN, DFS 등의 방법들과 비교 평가하여 본 CHPC의 스토리지 부하 최적화의 가능성을 보인다.

본 논문의 2장에서는 병렬 및 분산 캐싱으로 대표되는 고속 캐싱 기술과 분산 파일 시스템을 소개하며, 3장에서는 이와 관련한 연구들을 소개한다. 4장은 본 논문의 제안기법인 CHPC의 구조 및 읽기, 쓰기 정책, 공유자원을 고려한 데이터 병렬 전송 방법에 대해 제안하며 마지막으로 본 연구에 대한 평가와 결론으로 끝을 맺는다.

Ⅱ. 배경 지식

기존에는 스토리지에서 관리하는 데이터의 규모가 작았으며, 데이터 대부분은 데이터베이스 서버에 수용할 수 있었다. 그러나 스마트폰의 활성화와 더불어 소셜 네트워크 등의 발전으로 스토리지에서 관리하는 데이터가 급증하고 있다. 그 결과 기존의 관계형 데이터베이스보다는 클라우드 컴퓨팅 환경에 적합한 비관계형 데이터베이스가 현재 클라우드 컴퓨팅에서 효율성, 확장성, 유연성 등의 측면에서 더욱더 많이 사용되고 있다[13].

네트워크 파일 시스템 측면에서는 NFS(Network File System), AFS(Andrew File System), CIFS(Common Internet File System) 등이 존재 한다[14-17]. 이러한 네트워크 파일 시스템은 네트워크상에서 서로 다른 클라이언트의 파일 접근 시, 로컬 파일처럼 접근할 수 있다. 그러나 프로토콜의 오버헤드로 인해 데이터가 제타바이트 단위 이상으로 증가함에 따라서 유연한 처리에 어려움이 있다.

SAN(Storage Area Network)은 전통적인 네트워크 구조에서 대용량의 데이터 관리, 서버 간의 신속한 데이터 처리 및 원격 네트워크 접근 환경의 요구로 제안되었다. 이는 근거리통신망 환경에서 호스트 컴퓨터의 종류에 구애받지 않고 별도로 연결된 저장 장치 사이에서 대용량의 데이터 전송을 시킬 수 있는 고속 네트워크 스토리지 구조를 제공한다. 그러나 SAN은 클러스터 노드 간의 파일을 공유할 수 없으며, 빅데이터[18]를 저장하기에는 확장성이 낮은 문제점이 있다[19].

최근 클라우드 스토리지는 대표적으로 Amazon EC2, OpenStack , Microsoft Azure등이 있으며 각 분산된 스토리지를 로컬 파일처럼 접근할 수 있는 분산 파일 시스템을 기본적으로 포함하고 있다. 대표적인 분산 파일 시스템으로는 GFS(Google File System), HDFS, XtreemFS, Ceph, GlusterFS, MooseFS등이 있다[20-24].

최근 이러한 분산 파일 시스템은 기존의 분산 파일 시스템에서 제기되어온 데이터 접근에 대한 병렬 처리, 임의 위치에 대한 데이터 읽기 및 쓰기, 작은 워크로드에 대한 빈번한 접근 등의 문제에 개선된 성능을 보이나 최근 빠르게 급증하는 클라우드 데이터를 처리하기에는 여전히 어려움이 존재한다.

2.1. 분산 파일 시스템

분산 파일 시스템은 클라이언트 측의 네트워크를 통해 분산된 스토리지의 데이터에 접근하는 파일 시스템이다. 데이터는 네트워크상의 분산된 스토리지에 저장 및 관리된다. 그리고 분산된 데이터에 접근할 때에는 마치 클라이언트의 로컬 파일 시스템의 데이터인 것처럼 접근할 수 있는 투명성을 제공한다.

클라우드 컴퓨팅 환경에서는 여러 사용자가 동시에 같은 데이터에 접근하기 때문에, 접근 제어 및 데이터 일관성 유지를 위한 기술이 필요하다. 또한, 가용성을 위해서 장애가 발생하여도 서비스를 제공할 수 있어야 한다[9].

2.2. 병렬 캐싱

병렬 캐싱은 동적 네트워크 환경에서 그리드 메모리의 여유 메모리를 사용하여 클라이언트의 데이터 동시 접근을 가능하게 함으로써 처리량을 높인다. 병렬 캐싱의 구조는 동적 네트워크 환경에서 다수의 캐시 노드를 연결하여 노드 에이전트로 구성한다[12].

2.3. 분산 캐싱

분산 캐싱은 서버의 여분의 캐시메모리를 논리적으로 공유하여 캐시메모리의 활용률을 높임으로써 시스템상의 오버헤드를 낮추는 캐싱 방법이다.

이를 통하여 사용자 서비스 요청에 따라 균형 있는 캐시 메모리의 배분이 가능하며 네트워크상에 걸쳐 있는 여분의 캐시 메모리를 사용한다[10].

2.3.1. HDC(Hash Distributed Caching)

HDC는 블록의 디스크 주소를 해시 기반으로 나타내서 검색 및 데이터 매핑(Mapping)의 효율성을 제공한다. 이를 위해서 파일의 블록은 (I/O server, Inode index, File offset)으로 구성되며, 파일 블록은 I/O 서버의 디스크 주소와 파일의 오프셋을 이용하여 I/O 서버의 디스크에 매핑 된다[11].

2.3.2. Memcached

Memcached는 네트워크를 통해 Memcached 기능을 탑재한 서버들 간에 캐시메모리를 공유함으로써 실제 데이터를 처리하는 애플리케이션 서버가 Memcahced 기능을 탑재한 서버들 간의 여유 캐시 메모리를 사용할 수 있다. Memcached는 중요한 메모리 자원을 사용할 때 최소의 응답 반응 속도를 위해서 데이터를 최소로 조직하기 위해 데이터를 저장할 때 (키, 값)의 구조로 저장한다. 애플리케이션 서버는 키를 가지고 저장된 데이터를 획득하여 추가, 업데이트 및 삭제를 한다. Memcached는 서버의 중단, 캐시 메모리 고갈, 처리, 완료 등의 경우를 제외하고 데이터가 영구 보관된다[10].

2.4. 데이터 병렬 전송 관련 프로토콜

기존에는 분산 파일 시스템에서 파일 전송 시 TCP/IP, FTP, GridFTP등의 다양한 프로토콜을 사용한다. GirdFTP의 경우, FTP를 확장한 형태이며 데이터 전송 시 데이터를 병렬로 전송하며, 이는 로컬 노드에 접근하는 방법과 원격에서 데이터에 접근하는 두 가지의 접근방법으로 나뉜다. 그러나 이러한 전송방식은 데이터를 일정 시간 모아서 처리하는 배치 방식을 사용한다. 그 결과 네트워크의 대역폭과 높은 지연시간이 발생하게 되며, 데이터 전송 시 병목현상이 발생하게 되어 성능에 제한이 있다[8].

또한, 다른 연구로는 TCP와 UDP를 혼합한 형태의 데이터를 병렬 전송하는 기술인 STCP(Stream Control Transmission Protocol)이 있다[27-28]. 마지막으로 공유되는 자원을 고려한 데이터 전송의 연구가 있다. 로컬디스크에서는 NFS가 좋은 성능을 보이지만 일반적으로 자원을 공유하는 시스템에서는 CIFS가 더 좋은 성능을 보인다. 이러한 차이는 NFS의 경우 공유되는 자원에 대해 고려하지 않았지만 CIFS는 데이터 전송 시 공유되는 자원을 고려하였기 때문이다[29].

Ⅲ. 관련 연구

최근에는 클라우드 스토리지 환경에서는 데이터 접근의 지역성을 확보하여 성능을 향상하고자 하는 캐싱 기술이 주목받고 있다. 이는 각 서버 노드의 가용 캐시 메모리를 공유하는 형태의 분산 캐싱과 동적 네트워크 환경에서 그리드 메모리를 이용한 병렬 캐싱과 같은 고속 캐싱 기술로 나뉜다. 이를 통하여 HDFS에서는 효율적인 데이터 전송과 지연시간 및 처리량 향상 및 디스크의 입출력 최적화를 제공하고자 한다. 관련 연구로는 BlueSky와 원거리 통신망 환경에서의 분산 캐싱을 이용한 데이터 전송 향상 기법, RamCloud 등이 있다.

Bluesky는 프락시 서버의 캐시를 이용하여 데이터를 클라이언트에 효율적으로 전송하기 위한 연구로써 이전 관련 연구로는 Depsky, Nasuni 등이 있다[25-26]. Depsky는 클라우드의 가용성을 위해 데이터를 중복 저장하는 연구이며, Nasuni는 하드웨어적으로 가상 NAS를 구현한 연구이다. Bluesky는 프락시 서버로 큰 오브젝트를 병렬 업로딩이 가능하도록 지원하며, 클라이언트와 서버 간에 중계기 역할을 하며, 클라이언트의 요청을 캐시 메모리에 저장해두고 이를 원격 서버로 중계하여 제공한다. 이를 통해 클라이언트가 스토리지 서버에 접근하지 않고 데이터를 송수신하는 것이 가능하며, 전송 시간과 네트워크 병목현상을 줄일 수 있다.

원거리 통신망 환경의 데이터 전송으로는 HDFS(HaDoop File System) 기반의 원거리 통신망 환경에서의 데이터 전송에 관한 연구가 주목받고 있다. 대표적으로 GridFTP와 FDT(Fast Data Transfer)등이 있다[3,5]. HDFS에서는 “Static Global Resource”를 통해 Memcached 서버가 증가할 때마다 Memcached 서버에 저장된 변수에 병렬적으로 접근한다. 이는 캐싱 서버의 인스턴스가 증가할 경우 성능 또한 순차적으로 향상한다[2,4].

Ramcloud는 스토리지 서버의 최적화를 통하여 접근 지연시간의 감소와 처리율을 높이려는 연구이다. 최근 데이터의 급증으로 인하여 제타바이트 이상의 데이터가 생성되고 있다. 하지만 생성되는 데이터의 반 이상을 멀티미디어가 차지하며, 자주 접근되는 데이터는 작은 문서 파일들이 주류를 이룬다. 그 결과 생성되는 데이터의 비해 사용자의 의해 접근되는 데이터가 제한적이며 크기가 작다. 그러므로 자주 접근되는 소량의 데이터를 스토리지 서버의 디스크를 대신 하여 DRAM에 데이터를 저장함으로써 응답속도 및 지연시간을 줄인다[33].

Ⅳ. 제안 기법

본 논문에서는 클라우드 스토리지를 위한 분산 파일 시스템 환경에서 캐싱 기술을 이용하여 스토리지의 최적화를 위한 CHPC(Cloud storage High-Performance Caching) 구조를 제안한다. CHPC의 구조, 읽기/쓰기 정책, CHPC 시스템 내부적으로 필요한 블룸필터(Bloom Filter) 및 핫 데이터 등에 대해 서술한다[30-32]. 마지막으로 클라우드 환경에서 공유되는 자원을 고려한 데이터 병렬전송 기법에 대해 제안한다.

4.1. CHPC(Cloud storage High-Performance Caching) 시스템 구조

CHPC은 (M)개의 캐싱 기술을 포함하는 분산 스토리지 서버가 존재하며, 각 서버의 가용 캐시 용량은 (N)MB라고 가정한다. 이는 캐시를 논리적으로 공유할 경우 (M×N)MB의 캐시 공간만큼 저장할 수 있다.

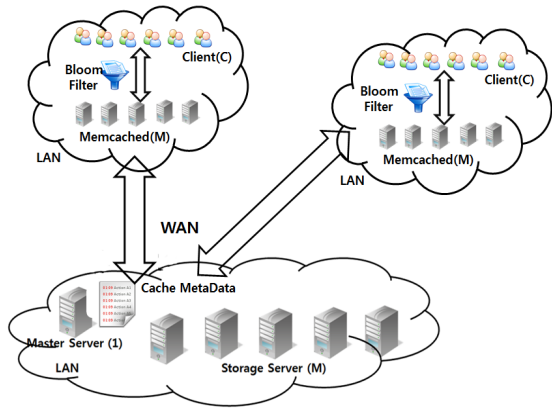

그림 1. CHPC(Cloud storage High-Performance Caching) 구조
Fig. 1. CHPC structure

CHPC의 시스템 구조는 그림 1과 같이 각각의 근거리통신망이 묶여있는 광역망 환경으로 이루어진다. 클라이언트는 분산 캐싱서버와 같은 근거리통신망으로 구성되며, 마스터 서버는 스토리지 서버와 같은 근거리통신망으로 구성된다. Memcached 분산 캐싱서버는 핫 데이터를 저장하며, 핫 데이터는 참조 지역성의 원리에 따라 자주 접근되는 데이터로 정의한다. 분산 캐싱 서버는 핫 데이터를 저장함으로써 디스크 입출력을 최소화한다.

클라이언트는 같은 네트워크 환경으로 묶여 있는 분산 캐싱서버에 블룸 필터를 통해 데이터의 존재의 여부를 빠르게 확인한다. 이때 분산 캐싱 서버에 데이터가 존재할 경우 전송받지만, 없을 경우 클라이언트는 마스터서버에 데이터의 요청을 하며 분산 캐싱 서버로 데이터를 갱신한 이후 클라이언트로 데이터를 전송한다. 이는 클라이언트의 데이터 요청을 마스터 서버와 분산 캐싱서버로 분산시킴으로써 데이터의 요청이 한곳으로 집중되는 것을 방지함과 동시에 이러한 병목현상을 완화한다.

4.2. Cold & Hot 페이지 캐시 관리

본 논문에서는 캐시 교체 정책 및 핫 데이터의 유지를 위하여 블룸 필터 계수를 정의한다. 이 계수는 페이지 캐시가 참조되는 횟수를 가지고 있으며, 페이지 캐시가 참조되면 1씩 증가 한다. 클라이언트가 분산 캐싱서버에 있는 페이지 캐시를 참조하면 계수를 1증가 시키고 콜드 데이터 리스트에 페이지 캐시를 할당한다. 계수가 4가 되면 핫 데이터 리스트로 이동하게 되며, 핫 데이터 리스트의 비어 있는 공간 중 끝의 가장 가까운 곳에 할당하며, 계수는 0으로 초기화 한다. 만약 핫 데이터 리스트가 다 차 있다면 핫 데이터 리스트 중 앞부터 참조하여 계수가 0인 것으로부터 교체한다. 페이지 캐시의 참조가 끝나면 계수가 감소한다. 이는 그림2와 같다.

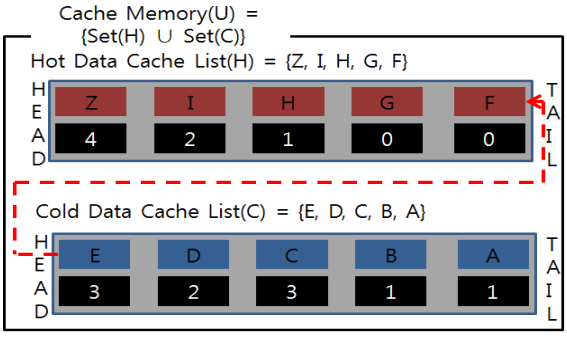

그림 2. 블룸 필터 계수
Fig. 2. Bloom Filter Count

아래 그림 3과 같이, 분산 캐싱서버는 클라이언트로부터 데이터 전송 요청이 들어오면 분산 캐싱서버에 데이터의 존재 여부를 블룸 필터를 통하여 빠르게 확인할 수 있다. 분산 캐싱서버는 데이터가 분산 캐싱서버에 존재하면 BFCB(Bloom Filter Check Bit)를 1로 설정하며, 없으면 0으로 설정하여 분산 캐싱서버에 데이터의 존재 여부를 알린다.

블룸 필터는 블룸필터 생성, 검색, 삭제의 세 가지 기능을 수행한다. 블룸필터 생성은 데이터를 3개의 해시 함수로 키 값을 생성하여 블룸 필터 배열에 인덱스를 추가한다. 블룸필터 검색은 데이터를 3개의 해시 함수로 키 값을 생성하고 키 값이 블룸 필터 배열의 인덱스에 존재 여부를 판단한다. 블룸 필터 삭제는 생성과 동일하게 해쉬 함수를 통해 배열의 인덱스에 해당하는 1을 0으로 교체하면서 블룸 필터 배열의 데이터 키 값을 삭제한다.

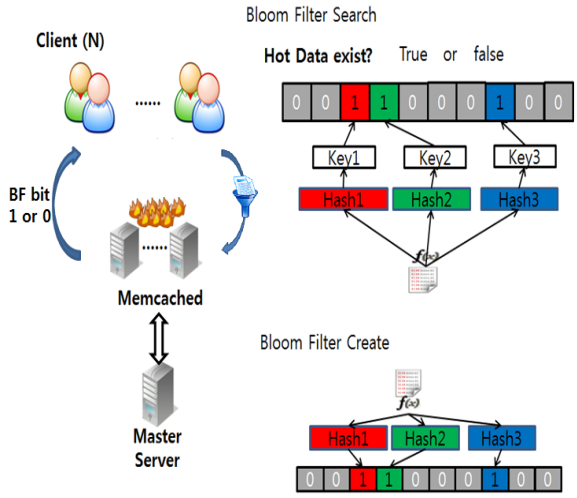

그림 3. 블룸 필터 구조
Fig. 3. Bloom Filter Structure

4.3. 페이지 캐시 중복 제거

기존의 구조는 클라이언트가 스토리지 서버로부터 원거리 통신망을 거쳐 데이터를 캐싱한다. 각 클라이언트는 서로 같은 데이터를 캐싱할 수 있으며, 이로 인해 중복된 페이지 캐시가 발생할 수 있다. 이러한 문제를 해결하기 위해서 CHPC에서는 분산 캐싱 서버를 통해 기존에 클라이언트가 가지고 있던 페이지 캐시를 분산캐싱 서버에 하나의 페이지 캐시만 저장하고, 관리하여 페이지 캐시의 중복을 감소시킨다. 이는 아래 그림4와 같이 각 클라이언트 간의 중복되는 페이지 캐시를 막고, 결과적으로 더 많은 가용 페이지 캐시를 확보할 수 있다.

4.4. 읽기 쓰기 정책

4.4.1. CHPC 읽기 정책

아래 그림 5와 같이, CHPC는 클라이언트가 같은 근거리 통신망 내의 분산 캐싱서버에 데이터를 요청한다. 분산 캐싱서버는 블룸 필터를 통해 데이터의 존재 유무를 확인하고 클라이언트에게 BFCB

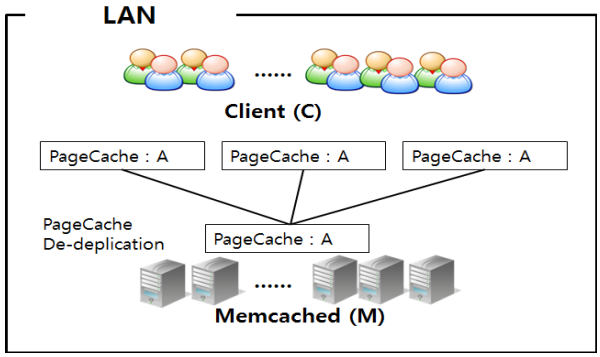

그림 4. 페이지캐시 중복 제거
Fig. 4. PageCache Dedeplication

를 통하여 데이터의 존재 유무를 알린다. 분산 캐싱 서버에 데이터가 존재할 경우 BFCB가 1로 설정된다. 그 후 클라이언트는 분산 캐싱서버로 Handler, Index를 전달하여 데이터를 요청한다. 그 결과, 근접한 분산 캐싱서버로 데이터의 요청 및 전송을 하므로 근거리 통신망에 비해 느린 원거리 통신망 환경을 거쳐서 스토리지 서버까지 가는 비용을 줄이고, 또한 외부로 나가는 트래픽을 최소로 할 수 있다.

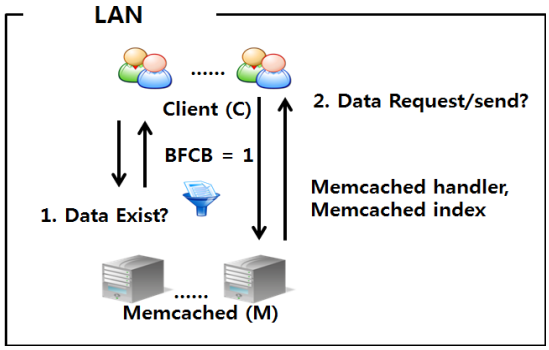

그림 5. 분산 캐싱 데이터 요청 및 전송
Fig. 5. Distributed Data Request & Send

그림 6과 같이 BFCB가 0일 경우 분산 캐싱 서버에 데이터가 존재하지 않으며 클라이언트는 마스터 서버로 데이터를 스토리지 서버로 Handler, Index를 통하여 데이터를 요청한다. 마스터 서버는 Handler와 Index를 통하여 데이터를 검색하여 분산 캐싱 서버에 페이지 캐시를 갱신한다. 분산 캐싱서버는 갱신된 페이지캐시를 클라이언트로 전송한다. 위와 같이 분산 캐싱 서버와 마스터 서버로 데이터의 요청을 분산함으로써 입력의 요청이 한곳으로 집중되는 것을 막고 병목현상을 방지한다.

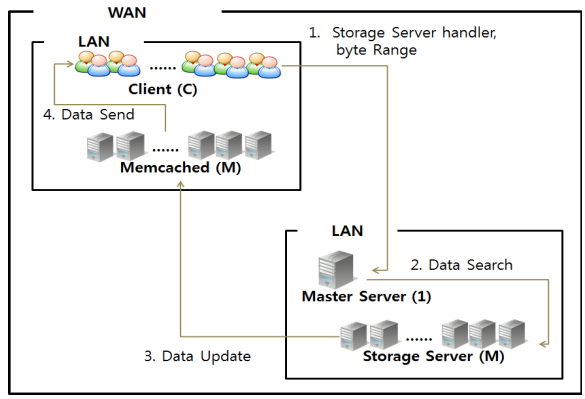

그림 6. 스토리지 서버 데이터 요청 및 전송
Fig. 6. Storage Server Data Request & Send

다음 그림 7은 CHPC를 순서도로 나타낸 것이다.

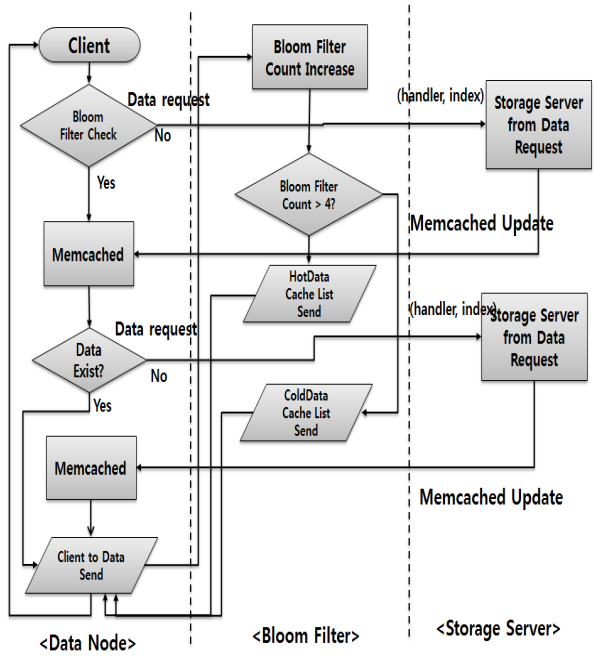

그림 7. 읽기 정책 순서도
Fig. 7. Read Policy Flow Chart

4.4.2. CHPC의 쓰기정책

분산 캐싱서버에 데이터를 쓸 경우 일관성 문제가 발생할 수 있다. 그러나 클라우드 스토리지는 데이터의 쓰기 연산보다는 읽기 연산이나 데이터의 추가 연산이 많다[14]. 그러므로 약간의 데이터 손실이나 일관성 문제를 허용하면서 성능상의 이점을 얻기 위하여 Write-Back Cache를 사용한다. 향후 일관성을 보장을 위한 동기화 방안에 대해 고려할 것이다.

표 1. 기술 적용 후 차이점
Table 1. Technology apply and difference

종류 / 차이	NFS(Network File System)	SAN(Storage Area Network)	DFS(Distributed File System)	CHPC(Cloud storage High-Performance Caching)
오버헤드 원인	통신 프로토콜	-	지연, 전송 시간 및 프로토콜 처리	페이지 캐시 관리
공유 자원	파일 공유	스토리지 공유	서버의 데이터	페이지 캐시
디스크 입출력 연산 횟수	높다	높다	높다	낮다
병목현상	발생 가능성 높음	발생 가능성 낮음	발생 가능성 높음	발생 가능성 낮음
통신 프로토콜	TCP/IP	SCSI, Fibre Channel(FCP), iSCSI	GridTFP, TCP/IP, FTP	GridTFP&SMB, TCP/IP, SCTP
기반 시스템	RPC(Remote Procedure Call)	DAS(Direct Attach Storage)	AFS(Andrew File System)	DFS & Distributed caching

4.4.3. 공유자원을 고려한 데이터 병렬 전송

CHPC는 데이터 전송 시 클라우드 스토리지에서 공유되는 캐시 메모리를 고려하여 데이터를 병렬로 전송한다. 기존의 데이터를 병렬 전송하는 방법인 GridFTP 공유되는 자원을 고려하지 않기 때문에 적합하지 않다. SMB(Server Message Block)는 공유자원을 고려하여 설계된 프로토콜로써 GridFTP를 혼합하여 공유되는 자원을 고려하여 병렬로 데이터를 전송한다.

전송할 데이터를 32개의 블록으로 나누고, 클라이언트와 서버간에 32개의 TCP 병렬 컨넥션을 맺으며, 각 컨넥션간 하나의 슬라이딩 윈도우를 통해 데이터를 전송한다.

V. 평 가

본 논문에서 제안한 CHPC의 요약과 비교 평가를 통하여 가능성을 검증하며 그 내용은 다음과 같다.

CHPC는 캐시 메모리의 공유를 통해 더 많은 가용 캐시 공간을 확보하여, 핫데이터를 유지함으로써 캐시 적중률이 상승한다. 그 결과 디스크의 I/O 연산을 줄일 수 있다. 또한, 클라이언트가 요청한 데이터가 분산 캐싱 서버에 존재하면 데이터를 같은 근거리 통신망 환경에서 수신한다. 이는 스토리지 서버까지 가는 네트워크 비용과 데이터 요청을 분산한다는 장점이 있다. CHPC의 경우 클라이언트마다 캐싱하고 있는 페이지 캐시를 분산 캐싱에 저장함으로써 각 클라이언트가 가지고 있는 캐싱된 데이터의 중복을 막을 수 있다.

CHPC는 분산 캐싱 서버를 효율적으로 사용하기 위하여 시간 지역성의 원리에 의해 같은 시간동안 접근이 자주 일어나는 핫 데이터를 저장함으로써 캐싱된 데이터를 효율적으로 활용한다. 또한, 핫데이터, 콜드데이터를 구분함으로써 핫 데이터를 분산 캐싱 서버에 오래 상주할 수 있게 하도록 설계하였다. 단, 분산 캐싱 서버에 페이지 캐시 관리를 위한 블룸 필터 관리비용이 추가로 든다.

위의 표 1은 NFS, SAN, DFS, CHPC를 비교 분석한 표이다. 각 시스템의 오버헤드의 원인으로는 NFS의 경우 통신 프로토콜이 무겁다는 오버헤드가 존재하며, SAN는 프로토콜의 오버헤드는 적지만 스토리지 확장 시 확장비용이 급증한다는 단점이 존재한다. DFS는 응답 지연시간, 전송시간 등의 오버헤드가 존재한다. CHPC의 경우, 분산 캐싱 서버의 페이지 캐시를 관리하기 위하여 블룸필터를 생성하고 관리하는 비용이 든다.

네트워크 측면에서 NFS의 경우, 대량의 클러스터 운영 시 병목현상이 발생할 수 있다. 또한, DFS는 데이터의 요청이 한곳으로 집중될 경우 병목현상이 발생할 수 있다. 반면 CHPC는 입력 요청을 스토리지 서버와 분산 캐싱서버로 분산시켜 병목현상을 방지한다. 전송 통신 프로토콜은 각각의 GridFTP, SCSI, FCP등의 프로토콜을 사용하며 CHPC의 경우 기존의 GridFTP와 SMB를 이용하여 CHPC 내의 공유되는 캐시 메모리를 고려한 병렬 데이터 전송을 제공한다.

표 2. 각 수식에 대한 값
Table 2. Values for each formula

Item	Description	Value
PN_C	Normal Cloud performance	-
PC_C	CHPC Cloud performance	-
NL_w	WAN Network Speed	150Mbit/s
NL_l	LAN Network Speed	1Gbit/s
DAt	Disk Access Time	8ms
MAt	Memory Access Time	6ns
CHr	Cache Hit Ratio	93%
CMr	Cache Miss Ratio	7%

다음은 수식을 통하여 CHPC 구조를 검증한다. 위의 표 2는 본 논문에서 제안하는 수식에 대한 각 변수 값이며, 이는 메모리 접근 시간, 디스크 접근 시간, 네트워크 속도, 캐시 적중 비율의 의미를 나타낸다.[34-35]. 위의 각 변수를 비율로 환산할 경우 NL_w과 NL_l는 6.7 : 1의 값을 가지며. DAt와 MAt는 1333.4 : 1의 비율을 가진다.

$$PN_c = 2NL_w + NL_l + DA_t \quad (1)$$

(1)번 수식은 그림 6과 같이 일반적인 분산 컴퓨팅 환경에서 클라이언트가 마스터 서버로 데이터를 요청한 시점부터 완료까지의 총비용을 의미한다. 클라이언트는 원거리 통신망을 거쳐서 마스터 서버로 데이터를 요청하며, 마스터 서버는 같은 근거리 통신망에 존재하는 스토리지 서버로 데이터를 요청한다. 이는 스토리지 서버에 존재하는 디스크에서 데이터를 찾으며, 원거리 통신망을 거쳐 클라이언트로 데이터를 전송하는 비용을 나타낸다.

$$PC_c = (PN_c + 2NL_l) \times CM_r + (2NL_l + MA_t) \times CH_r \quad (2)$$

(2)번 수식은 그림 5와 같이 CHPC에서 데이터를 요청할 때 산출되는 비용이다. 분산 캐싱 서버에 데이터를 요청하였지만 존재하지 않을 경우, 분산 캐싱서버의 데이터 요청비용과 PN_c의 비용을 합산한 비용을 나타낸다. 만약 데이터가 존재할 경우 캐시 메모리에 존재하는 데이터 접근 시간이 추가된다. 이 비용은 분산 캐싱 서버의 캐시 메모리 적중률과 비례하여 각 수식에 캐시 적중률을 통해 PC_c의 총 데이터 요청 비용을 나타낸다.

$$PN_c \geq PC_c \quad (3)$$

예를 들면 (1)번의 PN_c 수식을 표2의 값으로 계산할 경우 1347.8의 비용이 들며, (2)번의 PC_c 수식은 97.226의 비용이 나온다. 이는 대략 13배의 비용이 차이가 난다. 즉 CHPC를 이용하여 $PN_c \geq PC_c$, PC_c가 데이터를 더 적은 비용으로 송수신할 수 있으며, 성능 향상이 가능함을 보인다.

Ⅵ. 결 론

본 논문에서는 분산 파일 시스템 환경을 고려한 캐싱 기술을 이용하여, 클라우드 스토리지의 최적화를 제공하는 CHPC 기법을 제안하였다. CHPC는 기존의 네트워크 스토리지 및 분산 파일 시스템의 문제점을 다음과 같이 개선하였다.

첫째, 클라이언트는 분산 캐싱 기술을 통해 근거리통신망의 분산 캐싱서버에서 데이터를 수신하여, 원거리 통신망에 존재하는 스토리지 서버까지 송신하는 비용을 줄인다. 이는 외부로 나가는 트래픽의 감소와 서버의 데이터 요청을 분산 처리함으로써 병목현상 또한 방지할 수 있다. 둘째, 분산 캐싱 서버에 접근이 자주 일어나는 핫데이터를 저장함으로써 스토리지 서버의 디스크 입출력을 감소시킨다. 셋째, 클라이언트는 스토리지 서버로부터 데이터를 캐싱할 때 각 중복된 페이지 캐시를 가질 수 있다. 이를 분산 캐싱서버에 저장함으로써 페이지 캐시의 중복을 제거한다. 넷째, 각 클라우드 환경에서 공유되는 자원을 고려하여 병렬적으로 소켓연결을 이루고, 각 소켓마다 데이터를 전송함으로써 데이터 전송률을 향상시킨다.

이에 따라 본 논문에서 제안하는 CHPC는 고용량 및 실시간 데이터전송을 제공하는 클라우드 스토리지 인프라 구축에 활용할 수 있다. 향후에는 클라우드 스토리지 최적화 기술을 바탕으로 캐시 일관성을 보장하기 위한 동기화 기술 및 이와 관련된 최적화 기술에 관한 연구를 진행하고자 한다.

References

[1] J. Gantz and D. Reinsel, IDC's Digital Universe Study, *sponsored by EMC,* June 2011, http://idcdocserv.com/1142

[2] J. Lin, A. Bahety, S. Konda and S. Mahindrakar,

"Low-Latency, High-Throughput Access to Static G lobal Resources within the Hadoop Framework," *Te chnical Report HCIL*, 2009.

[3] L. Lin, and Z. Jia, "Research on Performance Opti mization for Grid Application Using Distributed Fil e System," *Frontier of Computer Science and Tech nology FCST*, 2010

[4] A. Amin, B. Bockelman, J. Letts, "High Throughpu t WAN Data Transfer with Hadoop-based Storage," *Journal of Physics: Conference Series Computing F abrics and Networking Technologies*, 2011.

[5] P. Arjan, K. Christiaan, S. Rogier and K. Paul, "Sur vey of Technologies for Wide Area Distributed Stor age," *surfnet.nl*, 2010

[6] M. Young-su, Jin. Ki-Sung, K. Hong-yun, "Di stributed technology trends for cloud computing" *El ectronics and Telecommunications Trends analysis*, 2009.

[7] P. Dongchul and H.C. David, "BlueSky: A Cloud-Backed File System for the Enterprise," *USENIX T HE ADVANCED COMPUTING SYSTEMS ASSOCI ATION FAST*, 2012.

[8] GridFTP, http://en.wikipedia.org/wiki/GridFTP

[9] Distributed File System, http://en.wikipedia.org/wiki /Distributed_filesystem

[10] B. Fitzpatrick, "Distributed caching with memcache d," *Linux Journal* volum 2004, Iuusue 124, 2004.

[11] I. Florin, M. Guido, O. Vlad, S. Gabor and T. Walt er, "Integrating collective I/O and cooperative cachi ng into the clusterfile parallel file system," *the 18th annual international conference on Supercomputin g*, 2004.

[12] C. Qingkui and L. Lichun, "Parallel Cache Model Based on GridMemor," *IEEExplore IFIP Internatio nal Conference on Network and Parallel Computin g*, 2007.

[13] Relational-database, http://www.readwriteweb.com/e nterprise/2009/02/is-the-relational-database-doomed. php

[14] I.Heizer, P. Leach, and D. Perry, "Common Internet File System Protocol (CIFS/1.0)," http://tools.ietf.or g/html/draft-heizer-cifs-v1-spec-00.

[15] D. Hitz, J. Lau, and M. Malcolm, "File System Des ign for an NFS File Server Appliance," t*he Winter USENIX Technical Conference*, 1994.

[16] R. Sandberg, D. Goldberg, S. Kleirnan, D. Walsh, and B. Lyon, "Design and Implementation of the S un Network Filesystem," *the Summer USENIX Tech nical Conference*, 1985.

[17] J. Howard, M. Kazar, S. Nichols, D. Nichols, M. Satyanarayanan, R. Sidebotham, and M. West. "Sca le and Performance in a Distributed File System," *ACM Transactionson Computer Systems (TOCS)*, 1 988.

[18] BigData, http://en.wikipedia.org/wiki/Big_data

[19] R. Steven, and e. Soltis, "The Global File system," *Mass Storage Systems and Technologies*, 1996.

[20] XtreemFS official site, http://www.xtreemfs.org/

[21] Ceph official site, http://ceph.newdream.net/

[22] GlusterFS official site, http://www.gluster.org/

[23] MooseFS official site, http://www.moosefs.org/

[24] S Ghemawat and H Gobioff, "The Google file syste m," *ACM SIGOPS Operating Systems*, 2003

[25] A. Bessani, M. Correia, B. Quaresma, F. Andre, an d P. Sousa. "DepSky: Dependable and Secure Stora ge in a Cloud-of-Clouds," *In EuroSys 2011*, 2011.

[26] Nasuni, http://www.nasuni.com

[27] SCTP, http://en.wikipedia.org/wiki/Stream_Control_ Transmission_Protocol

[28] SCTP Standards and Technology Analysis and Fore cast, http://ettrends.etri.re.kr/PDFData/18-3_011_02 0.pdf

[29] NFS & CINF, http://www.differencebetween.net/tec hnology/difference-between-nfs-and-cifs/

[30] P. Dongchul, and H.C David, "Hot Data Identificati on for Flash Memory Using Multiple Bloom Filter s," *USENIX THE ADVANCED COMPUTING SYST EMS ASSOCIATION FAST*, 2011.

[31] Bloom Filter, http://en.wikipedia.org/wiki/Bloom_filter

[32] Touch Count LRU, http://wiki.ex-em.com/index.php /Latch:_cache_buffers_chains

[33] RamCloud, https://ramcloud.stanford.edu/wiki/displa y/ramcloud/Home

[34] Values for each formula, http://www.computer-defini tion.com/access-time.php

[35] Values for each formula, http://www.content-net working.com/papers/web-caching-zipf.pdf

김 태 훈 (Tae-Hun Kim)

2012년 2월 한국산업기술대학교 컴퓨터공학과 학사
2012년~현재 성균관대학교 전자전기컴퓨터공학과 석사과정
<관심분야> 분산 파일 시스템, 운영체제, 가상화

김 정 한 (Junghan Kim)

2008년 2월 세종대학교 컴퓨터소프트웨어공학과 학사
2010년 2월 성균관대학교 전자전기컴퓨터공학과 석사
2010년~현재 성균관대학교 전자전기컴퓨터공학과 박사과정
<관심분야> 가상화, 운영체제, 분산 파일 시스템

엄 영 익 (Young Ik Eom)

1983년 2월 서울대학교 계산통계학과 학사
1985년 2월 서울대학교 전산과학과 석사
1991년 2월 서울대학교 전산과학과 박사
1993년~현재 성균관대학교 정보통신대학 교수
<관심분야> 분산 컴퓨팅, 시스템 소프트웨어, 운영체제, 가상화 기술, 미들웨어, 시스템 보안