8

OwFS: 대규모 인터넷 서비스를 위한 분산 파일 시스템

성균관대학교 ▌김진수*
NHN(주) ▌김태웅

1. 서 론

현대인들에게 있어서 다양한 인터넷 서비스는 소수의 전문가 혹은 신세대들만의 전유물에서 벗어나 일상적인 문화의 일부분으로 자리잡아 가고 있다. 무료 이메일 계정을 통해 메일을 주고받고, 카페나 소셜 네트워킹 사이트 등을 통해 친목을 도모하며, 자신이 촬영한 사진이나 손수 제작한 동영상을 공유 사이트에 업로드 하는 등의 활동이 보편화되고 있다. 이 밖에도 게임, 블로그, 위성사진, 지도, 뉴스, 도서, 쇼핑몰 등의 다양한 서비스들이 인터넷을 통해 이루어지고 있으며, 서비스 규모 또한 날로 대형화되고 있는 추세이다.

이에 따라 인터넷 상의 정보량 또한 폭발적으로 증가하고 있다. 특히, 사용자가 적극적으로 사용자 제작 컨텐츠(UGC: User Generated Contents)의 생성, 유통, 공유, 재사용에 참여하는 웹 2.0 시대로 접어들면서 그 현상은 더욱 가속화되고 있다. 미국의 시장조사업체 IDC는 개인에 의해 만들어지는 UGC의 증가, 아날로그 정보의 디지털 전환, 이메일의 증가 등으로 인해 앞으로 수년간 전 세계의 디지털 정보가 연평균 57%씩 증가할 것으로 예상하였다[1].

인터넷 서비스 업체들은 보다 많은 사용자를 확보하기 위하여 경쟁적으로 대규모의 저장 공간을 제공하고 있다. 당시로서는 파격적이었던, 1GB 이상의 메일 저장 공간을 최초로 제공해서 화제가 되었던 구글 메일(Gmail)은 현재 사용자당 7.3GB를 제공하고 있으며, 야후 메일(Yahoo!Mail)은 무한대의 저장 공간 제공을 선언하고 나섰다. 사진 공유 사이트인 Picasa는 사용자 계정 당 1GB의 저장 공간을 지원하고, 유사한 서비스를 제공하는 Flickr는 총 사용량의 제한 없이 매달 100MB 미만의 사진을 무제한 업로드 할 수 있도록 허용하고 있다. 동영상 공유 사이트인 YouTube에는 재생시간 10분 이내, 크기 1GB 미만의 동영상을 무한대로 업로드 할 수 있다. 대부분 이러한 서비스들은 사용자들에게 무료로 제공되기 때문에 인터넷 서비스 업체의 입장에서는 최소의 비용으로 필요한 저장 공간을 확보하는 것이 해당 서비스의 경쟁력, 더 나아가서는 회사의 생존과 직결되는 매우 중요한 요소가 되고 있다.

일례로, 인터넷 메일 서비스의 후발 주자였던 Gmail이 서비스 개시 몇 년 만에 수천만명 이상의 사용자를 확보할 수 있었던 이면에는 적은 비용으로 성능이 우수하고, 신뢰성 있는 저장 장치를 대용량으로 구축할 수 있었던 구글의 기술력이 있었기 때문이다. 구글은 수십만대 이상의 저가 하드웨어를 클러스터링하여 대규모 저장 용량을 제공하는 구글 파일 시스템(GFS)[2]을 자체적으로 개발한 후, 자사의 웹 검색 서비스를 비롯한 다양한 서비스에 적용하고 있으며, 이를 통해 단위 저장 공간당 요구되는 비용을 경쟁업체 대비 획기적으로 절감하고 있는 것으로 알려지고 있다. 아마존(Amazon)도 Dynamo[3]라 불리는 자체적인 저장 시스템을 개발하여 쇼핑 카트 서비스를 포함한 아마존 전자상거래 플랫폼의 핵심 서비스에 적용하고 있다.

GFS나 Dynamo와 같이 특정 서비스에 특화된 저장 장치의 개발 및 활용은 대용량 저장 공간 확보에 필요한 비용을 줄이고 효율성을 높일 수 있는 새로운 방법으로 주목받고 있다. 이것은 폭발적으로 증가하는 데이터를 고가의 범용 저장 장치에 저장할 경우 서비스 비용이 심각하게 증가하기 때문이다. 구글이나 아마존은 목표 서비스들이 공통적으로 요구하는 최소한의 기능만을 구현하였으며, 저가의 하드웨어를 사용하는데 따르는 데이터 가용성의 저하는 복제본을 이용한 소프트웨어 기술로 보완하는 접근 방법을 취하였다.

이와 같은 해외 선두 업체의 움직임에 영향을 받아 국내의 대규모 인터넷 서비스 업체들도 고가의 외산

* 종신회원

저장 장치에 의존하던 기존의 방법에서 탈피, 저장 장치의 경쟁력을 높이려는 노력을 본격화하기 시작하였다. 본 고에서는 이러한 노력의 하나로 탄생한 분산 파일 시스템 OwFS(Owner-based File System)를 소개하고자 한다. 지난 2006년부터 김진수 교수 연구팀과 NHN이 개발해 온 OwFS는 고성능, 신뢰성, 확장성, 대용량 처리, 관리의 용이성을 갖춘 분산 파일 시스템이다[4]. 순수 독자적인 기술로 개발된 OwFS는 2년여에 걸친 개발과 집중적인 테스팅 과정을 거친 후 2007년 하반기부터 NHN의 실제 서비스에 적용되었으며, 이후 점진적으로 적용 범위를 확대해 나가고 있다.

2. OwFS 요구 사항

2.1 설계 목표 및 특징

OwFS의 주요 설계 목표 및 특징을 요약하면 다음과 같다.

• 네이버 커뮤니티 서비스를 위한 분산 파일 시스템

초기 OwFS는 메일, 카페, 블로그 등 네이버 커뮤니티 서비스에의 적용을 목표로 설계되었다. 이들 서비스들의 특징은 다루는 정보가 크게 데이터 부분(메일 원문, 카페 및 블로그의 이미지, 첨부파일 등)과 데이터에 대한 메타정보 부분(메일 헤더 정보, 카페 및 블로그의 게시 관련 정보 등)으로 구성되어 있다는 점이다. OwFS는 기존의 저장 장치를 완전히 대체하는 용도로 개발되기 보다는 커뮤니티 서비스가 다루는 정보 중 데이터 부분에 해당하는 정보만을 저장하는 보완적인 저장 장치로 개발되었다. 그 이유는 데이터 부분이 저장 공간 용량의 대부분을 차지하는데 반해 그 접근 특성이 단순하여 특화된 저장 장치 개발이 용이하였기 때문이다. OwFS가 대상으로 하는 파일 시스템의 접근 특성에 대해서는 다음 절에서 자세히 살펴보기로 한다.

• 고성능, 저비용, 확장성 지원

OwFS는 수백 TB에서 수십 PB에 이르는 대용량 저장 공간 지원을 목표로 설계되었으며, 저장 장치 구축에 소요되는 비용을 낮추기 위하여 GFS와 같이 다수의 저가 서버를 활용하는 접근 방법을 택하였다. 모든 서버가 파일 접근 요청을 처리할 수 있기 때문에 용량의 증가는 곧 성능의 증가로 이어지며, 몇몇 파일 서버만이 요청을 처리하는 기존 방식에 비해 파일 접근 성능을 보다 향상시킬 수 있다.

• 데이터 가용성 유지

기존의 고가 저장 장치와는 달리 OwFS는 저가 하드웨어를 활용하기 때문에 개개 구성요소의 장애 발생 확률이 상대적으로 높으며, 서버와 디스크의 수가 증가할수록 이는 더욱 문제가 된다. 특히, 일반 사용자를 대상으로 하는 커뮤니티 서비스의 특성상 데이터의 유실은 사용자 감소 및 대외 이미지 저하로 직결되므로 서버나 네트워크 장애 발생시에도 데이터가 유실되지 않도록 높은 데이터 가용성을 지원해야 한다. OwFS는 GFS나 Dynamo와 같이 복제본을 활용하여 데이터 가용성을 확보하는 방안을 채택하였다.

• 관리 편의성 제공

OwFS가 실제 서비스에 적용될 경우 별도의 인력이 OwFS 운용을 담당하게 되므로, 저장 장치의 관리 편이성은 OwFS 개발 초기부터 매우 강조되었다. OwFS는 서비스를 중단하지 않고 저장 장치의 추가, 제거, 업그레이드가 가능하도록 설계되었으며, 관리자의 개입 없이 자동적으로 저장 장치간 부하를 분산하는 기능도 제공된다. 또한 OwFS를 구성하는 각 서버의 상태를 모니터링하고, 장애 발생을 통지하는 등의 전용 관리 도구들도 개발되었다.

• 고유한 파일 시스템 API 제공

대부분의 코드가 내부에서 직접 개발되는 웹 응용의 특성상 표준 POSIX 파일 시스템 인터페이스의 제공은 불필요한 것으로 판단되었다. 대신 OwFS는 POSIX와 유사한 OwFS 고유의 파일 시스템 API와 OwFS 클라이언트 라이브러리를 제공한다. OwFS를 사용하기 위해서 기존의 응용들이 수정되어야 하는 부담은 있지만, POSIX 파일 시스템 인터페이스를 제공할 필요가 없게 됨으로써 커널 수정이 불필요하게 되었으며, OwFS 개발 시간도 단축할 수 있게 되었다.

2.2 파일 시스템 접근 특성

전술한 바와 같이 OwFS는 메일 원문, 카페나 블로그의 첨부 파일, 이미지, 음악 및 동영상 파일 등과 같은 불변(immutable) 파일들의 저장을 목표로 한다. 불변 파일이란 한 번 파일이 생성되면 더 이상 파일의 내용이 변경되지 않는 파일을 의미한다. 이들 불변 파일들은 인터넷 서비스 업체들이 저장해야 하는 데이터의 대부분을 차지하며, 일반적인 파일과는 다른 접근 특성을 가지고 있기 때문에 이와 같은 특성을 고려하여 OwFS를 최적화하는 것이 필요하다. OwFS가 대상으로 하는 불변 파일들의 접근 특성을 요약하면 다음과 같다.

첫째, 동영상을 제외한 대부분의 불변 파일들은 크기가 상대적으로 매우 작다. 이메일 서비스의 경우 95%의 메시지들이 55KB 미만이었으며, 98.5%의 메시지들도 100KB 미만에 불과하였다. 따라서 OwFS는 매우

많은 수의 작은 크기 파일들을 효율적으로 다룰 수 있어야 한다. 둘째, 불변 파일들은 생성, 읽기, 삭제와 같은 세 종류의 주된 연산에 의해 접근된다. 파일이 생성될 때에는 순차적으로 쓰여지며, 한 번 쓰여지고 나면 그 내용이 변경되지 않는다. 또한 파일이 생성될 때 마다 시스템에서 고유한 파일 이름을 부여하므로, 동일 파일에 대한 동시 쓰기 요구도 발생하지 않는다. 읽기 요청은 해당 파일이 완전히 생성된 이후에만 허용되며, 쓰기 요청과는 달리 동일 파일에 대해 동시 읽기 요구가 있을 수 있다. 이러한 특징들은 OwFS에서 데이터 일관성 유지를 단순화하는데 이용된다.

3. 관련 연구

3.1 NAS(Network Attached Storage)

NAS 방식은 저장 공간을 제공하는 NAS 서버가 LAN을 통해 NFS 혹은 SMB/CIFS와 같은 업계 표준의 네트워크 파일 시스템 프로토콜을 이용하여 클라이언트와 연결되는 구조를 가지고 있다[5]. NAS 방식은 설치가 쉽고, 이기종 시스템 지원이 용이하다는 장점이 있으나 확장성에 제약이 있고 관리 비용이 크다는 단점이 있다. 특히, 저장 용량을 확장할 경우 부하 분배를 위해서는 수동으로 데이터를 이동시키는 작업이 필요하며, 파일 시스템을 접근하는 경로가 정적으로 결정되어 있기 때문에 파일 이동이 클라이언트에 투명하게 이루어지기 어렵다.

3.2 Coda

Coda는 미국 CMU 대학에서 개발된 분산 파일 시스템이다[6]. 서버나 네트워크에 장애가 발생하여도 서비스의 중단 없이 클라이언트가 원하는 파일을 접근할 수 있도록 파일을 볼륨(volume) 단위로 구성하고, 볼륨을 두 개 이상의 서버에 중복하여 저장하는 방식을 사용한다. 한 볼륨이 어떤 서버에 중복되어 저장되어 있는지의 정보는 모든 서버에 분산되어 관리된다. 볼륨이 여러 서버에 중복해서 저장되고, 일부 서버나 네트워크 연결에 장애가 발생한 사이 다른 서버의 파일이 갱신될 수 있기 때문에 Coda에서는 여러 서버에 있는 복제본 간의 일관성 유지가 가장 큰 부담 중의 하나이다. Coda에서는 각 파일이나 볼륨 별로 버전 정보를 관리함으로써 일관성을 유지하는데, 이와 같은 복잡한 일관성 유지 방법은 불변 파일들을 위해서는 불필요하다. 또한 Coda는 볼륨의 복사본을 분산시키는 자동화된 정책을 가지고 있지 않으며, 기존의 서버 중 하나가 새로운 서버로 대체되더라도 자동으로 이전의 서버가 가지고 있던 내용을 복구하는 등의 기능이 지원되지 않는다.

3.3 Lustre

Lustre는 오브젝트 기반의 분산 파일 시스템으로 확장성이 뛰어나기 때문에 고성능 컴퓨팅 환경에서 많이 사용되고 있다[7]. 대규모의 I/O를 효과적으로 처리하기 위해 Lustre에서는 파일의 메타데이터와 데이터를 독립시켜 관리한다. 파일에 대한 I/O를 수행할 경우, 클라이언트는 중앙의 메타데이타 서버(MDS)로부터 파일에 대한 정보만을 얻은 다음 분산되어 있는 데이터 서버들에게 직접 I/O를 요청한다 그러나 Lustre는 MDS 내의 로컬 파일 시스템을 이용하여 메타데이타를 관리하기 때문에, 파일의 개수가 많을 경우 MDS에 많은 부하가 걸리게 된다. 또한 장애 발생에 대응하기 위해서는 시스템 구축 비용이 증가하고, 자동으로 부하를 분산하는 기능도 제공되지 않는다.

3.4 GFS(Google File System)

GFS는 구글이 검색엔진을 비롯한 자사의 다양한 서비스를 위해 개발한 분산 파일 시스템이다[4]. GFS는 하나의 마스터 서버와 다수의 청크 서버(chunk server)

표 1 주요 저장 시스템의 특징 비교

	NAS	Coda	Lustre	GFS	OwFS
파일 서비스 구현	Kernel-level	User-level	Kernel-level	User-level	User-level
POSIX API 지원	O	O	O	X	X
마스터 서버의 유무	X	X	O	O	O
마스터 서버 장애 대응	—	—	HA failover	HA failover	HA failover
데이터 서버 장애 대응	—	Replication	HA failover	Replication	Replication
복제 단위	—	Volume	—	Chunk	Owner
자동 장애 검출 및 복구	△	△	△	O	O
자동 데이터 이동	X	X	X	O	O
주요 용도	범용	유닉스 환경	고성능 컴퓨팅 환경	큰 파일 저장	다수의 불변파일 저장

로 구성되어 있다. 하나의 파일은 고정된 크기의 청크 단위로 나뉘어서 청크 서버에 저장되고, 장애에 대응하기 위하여 각각의 청크는 기본적으로 3개의 서로 다른 청크 서버에 복제된다. 마스터 서버는 파일 시스템의 이름공간(namespace)과 청크의 위치 정보 등을 관리하는데, Lustre와는 달리 이들 정보는 메모리에 저장된다. 그러나 파일의 개수가 많아질 경우 모든 파일에 대한 메타데이터를 마스터 서버의 메모리에 유지시키기에는 무리가 있다.

표 1은 이들 시스템의 주요 특징을 OwFS와 비교한 것이다.

4. OwFS 구조

4.1 전체적인 구조 및 동작

OwFS의 전체적인 구조는 그림 1에 보이는 바와 같다. GFS와 유사하게 OwFS는 하나의 메타데이터 서버(MDS: MetaData Server)와 다수의 데이터 서버(DS: Data Server)들로 구성된다. MDS는 파일의 저장공간과 관련된 메타데이터를 관리하고, DS 서버의 상태를 감시하여 장애 상황에 대응한다. DS는 파일의 저장을 담당하고 있어, 파일에 대한 연산은 실제로 DS에서 수행된다. 메타데이터 서버와 데이터 서버는 모두 리눅스 상에서 사용자 수준의 프로세스로 구현되었으며, 클라이언트 프로그램은 OwFS 클라이언트 라이브러리와 링크되어 사용된다. 현재 OwFS 클라이언트 라이브러리는 C와 Java 버전으로 제공되며, 리눅스와 윈도우 환경을 모두 지원한다. OwFS로부터 파일 다운로드를 지원하기 위한 Apache 모듈도 지원된다.

OwFS가 GFS와 구별되는 가장 큰 차이점 중의 하나는 파일 저장 공간을 owner라는 단위로 묶어서 관리한다는 점이다. 서로 관련있는 파일들은 동일한 owner에 속하게 되며, 어떤 파일을 어떤 owner에 저장할 것인지의 여부는 전적으로 OwFS를 사용하는 응용에 의해 결정된다. 예를 들어, 이메일 서비스의 경우 owner는 각각의 사용자 계정에 대응될 수 있으며, 해당 사용자가 가지고 있는 메일 메시지는 각 owner에 속한 파일들로 사상될 수 있다. 카페나 블로그 서비스의 경우에는 owner가 각각 카페나 블로그의 고유 식별자에 대응되도록 하는 것도 가능하다.

OwFS에서 각각의 owner는 복제본을 유지하는 단위가 된다. OwFS는 각 owner에 대해 기본적으로 3개의 복제본을 유지하며, 이 때 동일한 owner에 속하는 파일들은 모두 동일한 3대의 DS에 저장된다. OwFS에서 owner라는 개념을 사용하는 주된 이유는 OwFS가 다른 파일 시스템들과는 달리 매우 많은 수의 파일들을 다룰 수 있도록 설계되었기 때문이다. 만일 개개의 파일마다 해당 파일의 메타데이터를 MDS에 저장한다면 MDS의 부담이 너무 커지게 되고, MDS가 성능 향상의 장애 요인이 될 수 있다. 반면 OwFS는 응용의 필요에 따라 적절한 수의 owner를 생성하고, 해당 owner에 대한 복제본 위치 정보만을 MDS에 유지함으로써 파일 수의 증가에 따른 MDS의 부담을 경감시킨다.

OwFS에서 파일을 읽는 과정을 간단히 설명하면 다음과 같다. 먼저, 응용 프로그램이 읽고자 하는 파일의 owner 명을 MDS에게 전달하면, MDS는 해당 owner의 아이디와 3개의 복제본 위치를 알려준다. 그러면, 클라이언트 라이브러리는 복제본을 저장하고 있는 3개의 DS 중 임의의 DS를 접근하여 파일을 읽게 된다. 파일을 쓰는 경우도 유사하게 진행되지만, 읽기의 경우

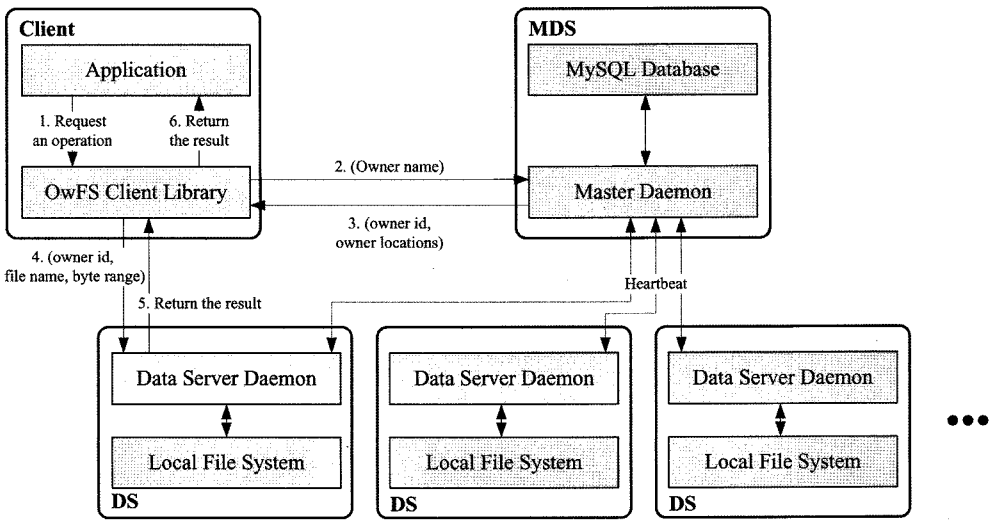

그림 1 OwFS의 구조

와는 달리 쓰기의 경우에는 세 개의 DS 모두에게 쓰기 요청을 전달한다.

4.2 MDS

MDS는 모든 owner들의 최신 복제본 위치와 DS들의 상태 정보를 관리한다. OwFS에서는 owner에 대해 열기(open), 닫기(close), 생성(create), 삭제(delete), 삭제 취소(undelete), 이름 변경(rename), 리스트 조회(read_owner_list) 등의 연산을 제공하는데, 이들 연산들은 모두 MDS에 의해 처리된다. 또한, 장애 복구나 부하 분산을 위해 복제본을 이동하는 작업에도 MDS가 관여하게 된다. 각각의 DS는 주기적인 heartbeat 메시지를 이용하여 자신의 정상 동작 여부를 MDS에게 보고하며, MDS는 이를 통해 DS의 장애 유무와 CPU, 네트워크, 로컬 디스크의 사용량 등을 모니터링한다.

MDS가 관리하는 각종 정보들은 MySQL에 의해 데이터베이스 테이블로 관리된다. 이것은 DBMS를 사용하는 것이 Lustre와 같이 로컬 파일 시스템에 정보를 저장하는 것보다 측정 결과 우수한 성능을 보였기 때문이다. 또한 DBMS를 사용하게 되면 백업 및 복제가 용이하고, 메타데이타에 대한 각종 조회 및 변경 작업을 SQL 질의문을 통해 손쉽게 행할 수 있다는 이점이 있다. MDS는 2개의 백업 서버에 의해 3중으로 보호되며, 공유 스토리지와 MySQL에서 제공하는 비동기 복제를 이용하여 장애에 대비하고 있다.

4.3 DS

DS는 파일을 저장하고, 파일에 대한 각종 연산을 처리하는 서버이다. OwFS는 파일에 대해 읽기(read), 쓰기(write), 위치 변경(lseek), 삭제(delete), 삭제 취소(undelete), 이름 변경(rename), 존재 여부 검사(exists), 파일 정보 검사(stat), 파일 리스트 조회(readdir) 등의 연산을 제공한다. 파일들은 XFS와 같은 리눅스 파일 시스템을 이용하여 저장된다. Owner별로 별도의 디렉토리가 생성되며, owner에 속하는 파일들은 해당 owner 디렉토리 하부에 저장된다. 또한 DS는 자신이 관리하는 owner의 상태 정보를 메모리에 캐싱하여 연산 처리 속도를 높인다.

DS는 그 수가 많기 때문에 장애 발생 확률이 상대적으로 높다. DS에 장애가 발생할 경우 복제본 수의 감소로 가용성이 저하되고, 복제본 간 일관성이 유지되지 않을 수 있으므로 신속하고 정확한 대처가 필요하다. DS의 장애 발생시 대처 방안에 대해서는 5절에서 자세히 살펴보기로 한다.

4.4 FlexRPC

클라이언트와 MDS, DS는 모두 FlexRPC[8]라 불리는 전용의 RPC(Remote Procedure Call) 계층을 이용하여 통신한다. OwFS 개발 초기에는 Coda에서 사용하는 RPC2 계층이나 리눅스에서 기본적으로 제공되는 RPC 계층을 멀티스레드 환경으로 확장하여 사용하려는 시도를 하였으나, 모두 OwFS에서 요구되는 기능들을 제공하기에는 한계가 있어 새로운 RPC 계층을 개발하였다.

FlexRPC 계층의 특징은 다양한 호출 형태를 지원한다는 점이다. 가장 기본적인 호출 형태는 기존의 RPC에서와 같이 하나의 클라이언트가 하나의 서버와 통신하는 단일 호출(single call)이다. OwFS에서는 동일한 파일을 서로 다른 DS에 복제하기 위하여 하나의 클라이언트가 다수의 서버에게 같은 내용의 매개변수를 이용하여 호출할 필요가 있는데 이를 멀티캐스팅(multicasting) 호출이라 부른다. 멀티캐스팅 호출은 클라이언트가 모든 서버에게 병렬적으로 매개변수를 전송하는 병렬 멀티캐스팅(parallel multicasting) 호출과 매개변수를 여러 서버를 거쳐 파이프라인 형태로 전송하는 직렬 멀티캐스팅(serial multicasting) 호출[9]로 다시 세분화 될 수 있는데 FlexRPC에서는 이러한 두 가지 종류의 멀티캐스팅 호출 형태를 모두 지원한다. 병렬 멀티캐스팅의 경우 빠른 응답시간을 가질 수 있는 반면 네트워크 대역폭을 많이 사용하게 되고, 직렬 멀티캐스팅은 네트워크 사용량을 분산시킬 수 있지만 상대적으로 긴 응답시간을 가진다.

FlexRPC 계층은 TCP와 UDP 프로토콜을 모두 지원하며 필요에 따라 일부는 TCP 프로토콜을, 다른 일부는 UDP 프로토콜을 사용하는 것이 가능하다. 특히, UDP 프로토콜을 사용할 경우에는 서버 측에 응답 캐시(response cache)를 두어 at-most-once 시맨틱을 보장하며, TCP 프로토콜을 사용할 경우에는 핸들 캐시(handle cache)를 두어 매번 연결을 새로 설정하는 오버헤드를 감소시킨다. 이 밖에도 FlexRPC 계층은 클라이언트와 서버에서 모두 멀티스레드 환경을 지원하며, 기존의 SunRPC와 비교하여 IDL(Interface Description Language) 수준에서 호환되고 제공되는 인터페이스도 거의 유사하다.

5. OwFS 데이터 서버 장애 대응 방법

5.1 장애 유형

OwFS에서는 장애 유형을 일시적 장애와 영구적 장애의 두 가지로 분류하고, 각각의 장애 유형에 대해

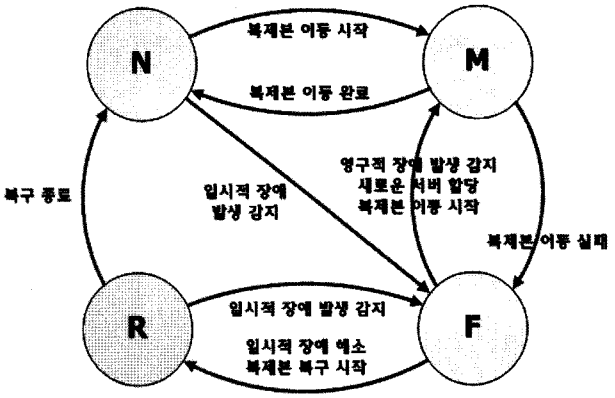

그림 2 DS의 복제본 상태 변화

서로 다른 대응 방법을 사용한다. 일시적 장애란 네트워크 장비의 일시적인 오동작, 관리자에 의한 시스템 재시작과 같이 일시적으로 DS가 파일 연산 요청에 응답하지 못하는 경우를 의미한다. 일시적 장애가 해소된 후에는 장애 발생 직전의 데이터를 이용할 수 있으므로, OwFS는 복제본 복구(replica recovery)를 통하여 일시적 장애가 있던 동안 업데이트된 파일의 일관성을 유지시킨다. 반면 영구적 장애는 하드디스크의 고장, 파일 시스템 오류, 커널 패닉, 전원 공급장치 고장 등과 같이 해당 DS가 정상적으로 서비스를 지속할 수 없는 심각한 상황을 의미하며, 이 경우에는 장애 발생 직전의 데이터를 이용할 수 없는 경우가 대부분이므로 OwFS는 복제본 이동(replica migration)을 통해 다른 DS에 새로운 복제본을 생성한다. 일시적인 장애와 영구적인 장애는 장애 지속 시간이나 관리자의 판단에 의해 구분된다.

5.2 복제본 상태 변화

그림 2는 일시적/영구적 장애 검출과 해소에 따른 복제본의 상태 변화를 요약한 그림이다. 각각의 복제본은 어느 순간 ***N***(Normal), ***M***(Migrating), ***R***(Recovering), ***F***(Fail) 중 하나의 상태에 있게 된다. ***N*** 상태는 정상적인 복제본을 의미하고, ***F*** 상태는 복제본을 저장하고 있는 서버가 장애에 빠져 해당 복제본을 접근할 수 없음을 뜻한다. 만일 발생한 장애가 일시적인 장애였다면, 장애가 해소된 후 해당 복제본은 ***R*** 상태로 전이하고, 복제본 복구 과정을 시작한다. 복제본 복구 과정이 완료되면 다시 정상 상태인 ***N*** 상태로 전이한다. 반면 발생한 장애가 영구적인 장애인 경우, 해당 복제본의 상태는 ***M*** 상태로 바뀌고 MDS는 복제본 이동 과정을 시작한다. 복제본 이동이 정상적으로 완료되면 해당 복제본의 상태는 다시 ***N*** 상태로 돌아간다. 복제본 복구나 이동 중 관련된 DS가 다시 장애에 빠질 수 있으며, 이 경우 해당 복제본은 다시 ***F*** 상태로 빠지게 된다.

각 owner의 상태는 해당 owner의 복제본 상태를 이용하여 표시한다. 예를 들어 어떤 owner의 상태가 O_1 = [***N***, ***N***, ***N***]와 같이 표시되었다면, 이것은 owner O_1의 모든 복제본 상태가 정상임을 표시한다. 반면, 어떤 owner의 상태가 O_2 = [***N***, ***N***, ***F***] 라면 이는 owner O_2의 한 복제본이 장애로 인해 가용하지 않음을 의미한다. OwFS의 초기 버전에서는 모든 owner의 복제본 상태 정보 또한 MDS가 DB 형태로 저장하도록 하였다. 그러나 이와 같은 방법은 어떤 DS가 장애에 빠질 때마다 해당 DS가 저장하고 있는 모든 복제본의 상태 정보를 ***N*** → ***F***로 변경하고, 복제본 복구를 시작하게 되면 다시 복제본의 상태를 ***F*** → ***R***로 변경하는 등의 빈번한 복제본 상태 업데이트를 필요로 한다. 보통 하나의 DS가 저장하고 있는 owner의 수는 수십만개 이상이 되기 때문에 수많은 owner들에 대한 복제본 상태 업데이트는 상당한 오버헤드를 수반하며, 결과적으로 장애가 발생할 때마다 MDS의 성능을 급격히 저하시키는 요인이 되었다.

이와 같은 문제점은 MDS에서 개개 복제본의 상태 정보를 제거하고, DS의 상태 정보만을 관리함으로써 해결되었다. 개선된 방법에서는 서버의 상태를 통해 각 복제본의 상태를 유추해 내는 방법을 사용한다. 예를 들어, DS의 상태가 ***N***이나 ***F***라면, 이것은 해당 DS에 저장되어 있는 모든 복제본이 ***N***이나 ***F***상태와 같다는 것을 의미한다. DS의 상태가 ***R***이나 ***M***인 경우에는 약간의 주의가 필요한데, 그것은 DS에 저장되어 있는 복제본 중 일부의 복제본은 이미 복구가 완료되거나 (***R*** 상태인 경우) 복제본 이동에 참여하지 않아 (***M*** 상태인 경우) ***N*** 상태로 남아있을 수 있기 때문이다. 그러나 이 경우에도 해당 DS에 문의하면 특정 복제본의 정확한 상태를 알아낼 수 있기 때문에 동작에는 문제가 없게 된다.

5.3 복제본 복구

일시적인 장애에 빠졌다가 돌아온 경우 DS는 복제본 복구 과정을 수행한다. 복제본 복구의 목표는 해당 DS가 일시적인 장애에 있는 동안 발생한 변경을 반영하여 다른 복제본과의 데이터 일관성을 유지하는 것이다.

일시적인 장애로부터의 복구를 위해 OwFS는 로깅 방법을 사용한다. 어떤 복제본이 ***F*** 상태에 빠진 경우, 다른 정상 상태의 복제본들은 MO-LOG(Missed Operations Log)라 불리는 로그를 작성하여 그동안 일어

난 업데이트를 기록한다. 남아 있는 정상 상태의 복제본 중 하나가 또 다른 장애에 빠질 수 있기 때문에, MO-LOG는 모든 ***N*** 상태의 복제본에 의해 기록되어야 한다. 일시적인 장애가 해소되면 DS는 MDS에게 자신이 복제본 복구 과정을 개시했음을 알리고, ***F*** → ***R***로의 상태 변경을 요청한다. 그리고는 자신이 저장하고 있는 모든 owner에 대해 정상 상태의 복제본을 소유하고 있는 서버를 접근하여 MO-LOG를 수신하고, 이를 재연하는 과정을 반복한다.

한편 MO-LOG를 수신하여 재연하는 동안 해당 owner에 대해 새로운 업데이트 요청이 수신될 수 있다. MO-LOG가 완전히 재연되기 전까지 새로 도착하는 업데이트 요청은 바로 처리될 수 없기 때문에 OwFS는 이들을 DO-LOG(Delayed Operations Log)라 불리는 별도의 로그에 임시로 저장한다. DO-LOG는 MO-LOG의 재연이 완료된 후 이어서 처리되며, 모든 owner에 대해 이 과정이 완료되면 해당 DS는 자신의 상태를 ***N*** 상태로 변경한 후 복제본 복구 과정을 종료한다.

5.4 복제본 이동

어떤 DS가 영구적인 장애에 빠진 것으로 판단되면, 해당 DS에 속한 복제본은 더 이상 가용하지 않으므로 일정한 복제본 수를 유지하기 위하여 MDS는 복제본 이동 과정을 개시한다. 복제본 이동은 각 대상 owner에 대해 새로운 복제본을 저장할 DS의 위치를 결정하고, ***N*** 상태의 복제본으로부터 파일을 복사함으로써 이루어진다. 남아있는 복제본의 수가 적은 owner일수록 우선적으로 복제본 이동이 행해지며, 시스템의 성능을 유지하기 위해 하나의 DS는 동시에 두 개 이상의 서로 다른 복제본 이동에 참여하지 않도록 제한한다. 필요에 따라 관리자는 특정 복제본을 임의의 위치로 이동시킬 수 있으며, 특정 DS에 저장 공간이 부족해 질 경우 자동으로 부하를 분산하는 기능도 유사한 방법을 사용하여 구현된다.

5.5 복제본 상태 동기화

OwFS에서는 복제본의 상태에 따라 클라이언트의 동작이 결정된다. 예를 들어, 읽기의 경우 ***N*** 상태의 정상적인 복제본에게만 읽기 요청을 보내며, 쓰기의 경우에는 ***F*** 상태가 아닌 모든 복제본에게 쓰기 요청을 보낸다. 따라서 클라이언트는 복제본의 상태를 정확히 아는 것이 중요하게 된다. 그러나 클라이언트와 무관하게 DS는 언제나 장애에 빠질 수 있기 때문에 현재 클라이언트가 알고 있는 복제본의 상태가 정확한 상태라고 볼 수는 없다. 마찬가지로, MDS가 heart-beat 통신을 통하여 DS의 장애 여부를 파악한다고 해도 그 정보가 최신 상태라고 볼 수는 없다. OwFS에서는 상태 매칭(state matching)이라 불리는 기법을 사용하여 복제본 상태 동기화를 행한다.

상태 매칭은 클라이언트가 특정 owner에 대한 연산을 DS에 요청할 경우, 자신이 알고 있는 owner의 현재 상태를 RPC를 통해 함께 실어 보내는 방법이다. 만일 연산을 요청받은 DS에서도 해당 owner에 대해 동일한 상태를 알고 있다면 연산은 성공한다. 그러나 상태 매칭에 실패했을 경우 DS는 MDS로부터 해당 owner에 대한 최신 정보를 갱신하고, 그래도 상태 매칭이 이루어지지 않으면 클라이언트에게 상태 매칭 실패를 알린다. 이를 받은 클라이언트는 자신도 해당 owner의 최신 정보를 MDS로부터 갱신한 다음 동일한 연산을 재시도 한다.

그림 3은 DS의 복구를 전후하여 상태 매칭을 통해 복제본 상태를 동기화하는 예를 보인 것이다. 어떤 owner ***O***의 복제본이 DS1, DS2, DS3 세 대의 서버에 저장되어 있을 때, 이 중 DS3가 현재 ***F*** 상태에 있고, MDS와 클라이언트, DS1, DS2 모두 이 사실을 알고 있다고 하자(Phase 1). DS3에 일시적인 장애가 해소되면 DS3는 MDS에게 복제본 복구 과정을 시작했음을 알리고, 이 과정에서 DS3와 MDS의 상태는 [***N***, ***N***, ***R***]로 변경된다. 그러나 클라이언트와 DS1 및 DS2는 아직 이 사실을 모르고 있기 때문에 클라이언트는 업데이트

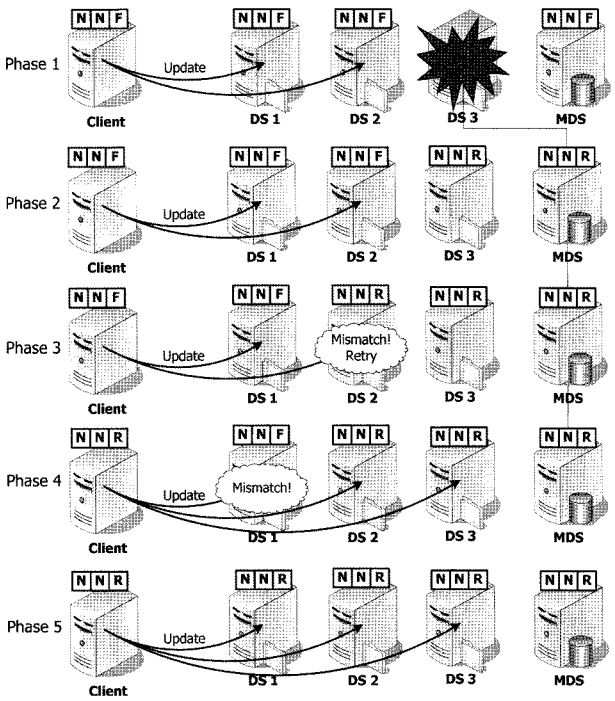

그림 3 상태 매칭의 예

요청을 DS1과 DS2에게만 전송하게 되고, DS1과 DS2는 아직 DS3가 ***F***상태에 있다고 판단, MO-LOG를 계속 기록하게 된다(Phase 2). 한편 DS3는 복제본 복구를 위하여 DS2에게 MO-LOG를 요청한다. 이 과정에서 DS2는 DS3가 ***R***상태로 전이하였음을 알고 자신도 ***O***의 상태를 ***R***로 변경한다. 이 사실을 모르는 클라이언트는 여전히 ***O***의 상태를 [***N***, ***N***, ***F***]로 알고 업데이트 요청을 전송하게 되나, DS2에 의해 상태 매칭을 실패하게 된다(Phase 3). 상태 매칭을 실패한 클라이언트는 MDS로부터 ***O***의 상태를 새로 갱신하여 요청을 재시도하게 되지만, 이번에는 아직까지 DS3의 상태가 변경된 줄 모르는 DS1에 의해 상태 매칭이 실패하게 된다(Phase 4). 상태 매칭을 실패한 DS1가 MDS로부터 DS3의 상태를 갱신하고 나면 관련된 모든 서버가 ***O***에 대해 모두 동일한 [***N***, ***N***, ***R***] 상태를 갖게 되고, 이후의 연산은 상태 매칭의 실패없이 정상적으로 수행된다. 상태 매칭 기법은 장애가 발생하지 않는 대부분의 정상적인 경우 상태 정보를 RPC에 함께 실어 보내는 것 외의 다른 오버헤드를 수반하지 않는다.

5.6 복제본 배치 정책의 개선

OwFS 초기 버전은 하나의 owner가 생성될 때, 해당 owner의 복제본을 디스크 사용률과 워크로드가 낮은 3개의 DS 서버에 할당하였다. OwFS에서는 DS가 3대 이상 고장나면 일부 owner에 대해서는 데이터 가용성을 보장할 수 없다. 따라서 수백대 규모로 OwFS를 구성하는 환경에서 네트워크 스위치 또는 전원에 장애가 발생하면, 3개의 복제본을 모두 접근할 수 없는 owner 수가 많아질 수 있다. 이러한 문제점을 보완하기 위해 네트워크 스위치나 전원 장애 발생시 접근할 수 없는 DS 서버들을 그룹별로 나누어 DS 그룹당 1개의 owner 복제본을 가지도록 복제본 배치 정책을 변경하였다. DS 서버 장애가 동시에 DS 그룹 3개 이상에 걸치지 않는다면, DS 서버 장애 수와 관계없이 적어도 1개의 정상 복제본이 존재하므로 데이터 가용성이 향상된다.

DS 그룹별 복제본 배치 정책은 장애에 대한 데이터 가용성을 높이지만, DS 서버의 영구적 장애로 인한 복제본 이동의 성능은 저하시킨다. 그 이유는 특정 DS 그룹 내의 한 DS가 고장나면 해당 DS가 가지고 있던 복제본들을 다시 생성해야 하는데, 새로운 복제본을 저장해야 할 DS 서버 후보군의 수가 DS 그룹 제약조건에 의해 줄어들기 때문이다. 따라서 동시에 수행할 수 있는 복제본 이동 작업 개수가 줄어들어 복제본 이동에 걸리는 시간이 늘어난다. 복제본 이동의 성능을 저하시키는 또 하나의 요소는 네트워크 스위치간 전송 지연시간이다. 복제본을 재생성하기 위해서는 다른 DS 그룹내의 서버에서 복제본을 읽어와야 하는데, 데이터 전송시간은 동일 네트워크 스위치 내의 DS 서버간 통신에 비해 약간 증가하게 된다.

6. NHN 서비스 적용 현황

OwFS는 2006년 개발이 시작되었으며, 2007년 하반기부터 네이버와 한게임 서비스에 적용되어 현재까지 안정적으로 운용되고 있다. 네이버 서비스에서는 사용자들에 의해 업로드/다운로드되는 파일 서비스에 주로 사용되고 있으며, 한게임 서비스에서도 게임 이용자들에 의해 발생되는 게임 데이터를 저장하는 용도로 사용되고 있다.

OwFS를 기존 서비스들에 적용하는 경우 새로운 API를 익혀야 하는 점에 대한 개발자들의 부담과 코드 수정으로 인한 새로운 버그 발생의 우려가 가장 큰 문제점으로 대두되었다. 그러나 이와 같은 점에도 불구하고 파일 시스템의 변경으로 인한 코드 수정은 대부분 2주 이내로 완료 가능하였다. 설계 초기부터 OwFS를 가정하고 개발하는 신규 개발 서비스의 경우에는 개발자로부터의 피드백이 좋았으며, OwFS 적용으로 인한 추가적인 개발 기간의 증가는 없었다.

OwFS는 기존 저장 시스템으로 사용되던 NAS 방식보다 총소유비용(TCO) 면에서 우수하기 때문에 저장장치 구축 비용을 절감하는 효과가 있을 것으로 기대된다. 이에 따라 NHN 서비스에 OwFS를 적용하려는 움직임은 확대되고 있고, 신규로 개발되는 네이버/한게임 서비스 중 대규모의 저장 공간을 필요로 하는 경우는 대부분 OwFS 기반으로 구축될 것으로 예상된다.

7. 결 론

지금까지 대규모 인터넷 서비스를 위해 개발된 분산 파일 시스템 OwFS의 설계 목표 및 전체적인 구조, 장애 대응 방법, NHN 서비스 적용 현황 등에 대해 간략하게 살펴보았다. OwFS의 공동 개발은 산학협동의 모범적인 성공 사례로써, OwFS 분산 파일 시스템은 현재 NHN 내부 표준 파일 시스템의 하나로 자리매김 하고 있다. OwFS는 지속적으로 새로운 기능이 추가되고 있으며, 그 적용 범위 또한 확대되어 갈 전망이다.

인터넷 보급의 증가로 다양한 종류의 신규 인터넷 서비스들이 하루가 다르게 쏟아져 나오고 있다. 날로 심화되는 글로벌 경쟁에서 살아남기 위해서는 새로운

인터넷 서비스와 비즈니스 모델의 발굴도 중요하지만, 이에 못지않게 수천만명 이상의 사용자들에게 안정적인 서비스를 지속할 수 있는 대규모 서버 및 스토리지 시스템 기술 또한 매우 중요하다. 현재 우리나라의 컴퓨터 시스템 기술은 매우 취약한 상황이며, 관련 인력도 부족한 실정이다. 국내 인터넷 서비스 업체들이 구글이나 야후 등과 같은 글로벌 업체들과 경쟁하기 위해서는 이제부터라도 시스템 관련 기술과 인력을 확보하기 위해 노력을 경주해야 할 것으로 보인다.

감사의 글

OwFS 개발에 참여한 KAIST 컴퓨터구조연구실의 정욱, 이대우, 박은지, 이영재, 김상훈 연구원과 NHN(주)가상화플랫폼개발팀의 전성원 수석, 주윤철 수석, 양명아 대리에게 감사드립니다.

참고문헌

[1] International Data Corporation (IDC), The Diverse and Exploding Digital Universe, 2007 & 2008.

[2] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP), pp. 29-43, 2003.

[3] G. DeCandia et al., "Dynamo: Amazon's Highly Available Key-value Store," Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP), pp. 205-220, 2007.

[4] 김태웅, "사용자를 위한 보이지 않는 노력, 대용량 분산 파일 시스템", NHN Story, http://story.nhncorp.com/story.nhn?story_id=86

[5] W. Singer, "NAS and iSCSI Technology Overview," SNIA Technical Tutorials, Fall 2007.

[6] J. H. Howard et al., "Scale and Performance in a Distributed File System," ACM Transactions on Computer Systems (TOCS), 6(1):51-81, 1988.

[7] Cluster File Systems Inc. Lustre: A Scalable, High Performance File System, http://www.clusterfs.com.

[8] Sang-Hoon Kim, Youngjae Lee, and Jin-Soo Kim, "FlexRPC: A Flexible Remote Procedure Call Facility for Modern Cluster File Systems," Proceedings of the 9th IEEE International Conference on Cluster Computing, September 1997.

[9] Sang-Hoon Kim, Jin-Soo Kim, and Seungryoul Maeng, "Modeling and Evaluation of Serial Multicast Remote Procedure Calls (RPCs)" IEEE Communications Letters, 13(4):283-285, 2009.

김 진 수

1991 서울대학교 컴퓨터공학과 공학사
1993 서울대학교 컴퓨터공학과 공학석사
1999 서울대학교 컴퓨터공학과 공학박사
1998~1999 IBM T. J. Watson Research Center, Academic Visitor
1999~2002 한국전자통신연구원 선임연구원
2002~2008 KAIST 전산학과 부교수
2007~2008 삼성전자 고문
2008~현재 성균관대학교 정보통신공학부 부교수
관심분야 : 운영체제, 스토리지 시스템, 내장형 시스템, 분산 시스템
E-mail : jinsookim@skku.edu

김 태 웅

1993 서울대학교 컴퓨터공학과 공학사
1995 서울대학교 컴퓨터공학과 공학석사
2000 서울대학교 전기,컴퓨터공학부 공학박사
2000~2000 서울대학교 컴퓨터신기술공동연구소 특별연구원
2000~2006 데이터코러스(주) 기술이사
2006~현재 NHN(주) 가상화플랫폼개발팀장
관심분야 : 분산파일시스템, 운영체제, 가상머신, 스토리지 가상화
E-mail : taewoong.kim@nhn.com