클라우드 스토리지 시스템 기술 고찰

KT 종합기술원 ▌김 미 점

1. 서 론

기업 IT 인프라는 물리적인 하드웨어 및 소프트웨어 라이센스/유지 비용으로부터 IT 인프라의 모니터링, 관리 유지보수에 대한 비용 등을 포함하여 높은 비용을 유발한다. 최근에 주목 받기 시작한 클라우드 컴퓨팅의 출현은 이러한 IT 인프라에 수반되는 여러 비용들을 획기적으로 감소시켜 주는데 기여를 하는 것과 동시에 응용의 빠른 적용 및 IT 서비스의 유연성과 확장성을 제공함으로써 IT 업계에 중요한 이슈로 각광받고 있다.

클라우드 컴퓨팅은 인터넷 망을 통해 여러 가지 IT 리소스들을 분배하여 서비스하는 개념으로, 예를 들어, 대규모의 인터넷 서비스 출시를 위해 더 이상 고가의 H/W를 구축하거나 그것을 운용하고 관리하기 위해 많은 IT 인력을 고용할 필요성을 제거한다. 그로 인해, 출시된 서비스가 초기의 기대치에 부응하지 못해 많은 여분의 리소스들을 낭비할 필요도 없으며, 반대로 기대하지 못했던 사용자 폭증으로 IT 리소스가 수요를 감당하지 못해 잠재 고객이나 매출을 놓치는 우를 범하는 리스크를 피할 수 있다. 특히, 대규모의 IT 리소스가 필요한 응용은 클라우드 컴퓨팅을 활용하여 상대적 저비용으로 단시간에 결과를 도출할 수 있다. 1시간 동안 1000대의 서버를 쓰는 비용과 1대의 서버를 1000시간 쓰는 비용이 큰 차이가 나지 않기 때문이다[1].

일반적으로 많이 인용되는 NIST(National Institute of Standards) 분류[2]로 클라우드 컴퓨팅 서비스를 구분해 보면, 첫째, 하드웨어 인프라를 서비스로 제공하는 IaaS(Infrastructure As A Service), 둘째, 응용 개발 및 실행 플랫폼을 서비스로 제공하는 PaaS(Platform As A Service), 그리고 마지막으로 응용(Applications)을 서비스로 제공하는 SaaS(Software As A Service)로 나눈다.

IaaS는 서비스 형태로 제공되는 컴퓨팅 자원(서버, 스토리지, 네트워크)을 의미하는데 보통 운영체계 (Operating System), 가상화, 클러스터링과 같은 소프트웨어들이 포함된다. IaaS로 가장 잘 알려진 상용 서비스로는 아마존(Amazon)의 웹 서비스(AWS)[3]이며 AWS는 컴퓨팅 서버를 제공하는 Elastic Compute Cloud(EC2) 서비스와 스토리지를 제공하는 Simple Storage Service (S3)가 포함된다.

본 논문에서는 IaaS로 제공될 수 있는 클라우드 스토리지 서비스를 제공하고자 할 때 활용 가능한 두 가지 스토리지 시스템을 소개하고자 한다. 이 두 시스템은 공통적으로 실제 상용 클라우드 스토리지 서비스를 제공하고 있어 이의 구조 및 시스템 기술들을 살펴보는 것은 의미 있는 고찰이 될 것으로 생각된다. 기존의 스토리지 서비스와 차별화되는 클라우드 스토리지 시스템의 가장 큰 특징은 일반 사양의 하드웨어를 활용하여 탄력적으로 스토리지 요구사항에 대처할 수 있는 것이다. 즉 적시적인 확장성과 유연성이다. 첫 번째로 소개할 클라우드 스토리지 시스템은 스마트 폰을 비롯한 다양한 단말간 동기화에 촛점을 맞춘 고급 스토리지 서비스를 위한 시스템으로 동기 서버 및 스토리지 서버 등을 포함한다. 두 번째로는 아마존의 S3와 같은 단순 백업용의 스토리지 서비스로 활용할 수 있는 시스템으로 오픈 스택(OpenStack) 프로젝트[4]의 오브젝트 스토리지 시스템인 스위프트(Swift)[5]에 대해 소개하고자 한다. 추가하여, 이 두 스토리지 시스템의 한계 및 개선 사항에 대한 의미 있는 고찰을 통해 이상적인 클라우드 스토리지 시스템에 대한 그림을 그려보고자 한다.

본 논문의 구성은 다음과 같다. 우선 2장에서는 분산 파일 시스템과 스토리지 시스템 등의 관련 기술 동향을 살펴보고 제 3장에서는 동기화 지원 스토리지 시

† 바쁘신 일정에도 본 논문을 리뷰해 주시고 조언 주신 KT 종합기술원 OSS 클라우드 플랫폼 프로젝트의 황진경 박사님과 안재석 박사님께 진심으로 감사드립니다.

스템의 구조 및 구성, 기능 등을 소개한다. 4장에서는 백업용 스토리지 시스템인 오픈 스택 프로젝트의 스위프트에 대한 구조 및 내부 시스템 구성 등을 살펴보고 5장에서는 앞 장들에서 소개한 두 스토리지 시스템의 비교 분석 후 개선 사항을 제시하고자 한다. 6장에서는 결론 및 앞으로의 연구 방향에 대해 논의한다.

2. 관련 시스템 및 기술 동향

데이터를 저장하는 파일 시스템은 오랜 동안 아주 다양한 형태로 발전되어 왔다. 가장 일반적으로 사용되고 있는 파일 시스템은 디스크 파일 시스템으로 블록 지향적이라, 파일들은 일련의 블록들로 구성된다. 디스크 파일 시스템에 저장된 파일들은 완전한 랜덤 접근이 가능한 특징을 가진다. 많은 양의 데이터 저장과 확장성을 지원하기 위한, 디스크 파일 시스템의 발전된 형태가 공유 디스크 파일 시스템으로 공유 스토리지 파일 시스템이나 SAN(Storage Area Network) 파일 시스템 또는 클러스터 파일 시스템으로 불리어지고 있다. 공유 디스크 파일 시스템은 모든 노드들이 파일 시스템이 위치한 블록 스토리지로 바로 접근이 가능한 SAN에서 주로 사용되고 있다. 또한 공유 디스크 파일 시스템은 하드웨어 RAID(Redundant Array of Independent Disks)[6]의 스토리지와 함께 높은 가용성이 지원되는 클러스터내에서 사용되며 일반적으로 64나 128 노드 이상으로 확장되기는 어렵다.

분산 파일 시스템은 디스크 파일 시스템의 확장성 한계를 극복하기 위한 시스템으로 네트워크 파일 시스템이라고도 한다. 분산 파일 시스템은 클라이언트가 서버상에 저장된 데이터를 마치 자신에게 저장되어 있는 것처럼 접근하고 처리할 수 있는 클라이언트/서버 기반의 파일 시스템[7]으로 일반적으로 접근 제어 리스트(Access Control Lists)를 유지하고 있다. 분산 파일 시스템 중 좀 더 발전된 형태인 오브젝트 스토리지 시스템은 다음과 같은 잇점으로 인해 클라우드 스토리지 서비스를 위한 표준으로서 자리매김하고 있다. 우선 메타 데이터와 오브젝트를 함께 관리함[8]으로 인해 특별히 이미지나 비디오, 오디오 파일들과 같은 비정형 구조의 컨텐츠 저장과 분산을 위해서는 아주 적합한 시스템이다. 즉, 파일 시스템의 종속적을 제거하기 위해 물리적인 저장 공간 관리 기능을 스토리지 디바이스 자체에서 직접 수행하게 함으로써 성능의 향상, 확장성, 그리고 플랫폼 독립적 데이터의 안전한 공유 등을 제공하는 특징을 가진다[9]. 최근의 클라우드 스토리지 서비스를 위한 대용량 분산 파일 시스템 및 오브젝트 스토리지 시스템이 몇 년 사이 많은 주목을 받고 있는데 아래에서는 그 중 대표적인 두 시스템인 구글 파일 시스템[10]과 아마존의 S3[11]를 소개하도록 한다.

구글 파일 시스템(GFS: Google File System)은 수십~수백 메가 바이트 이상의 대용량 데이터 저장과 고가용성, 확장성에 목표를 둔 병렬, 분산 처리에 친화적인 분산 파일 시스템이다. 구글 파일 시스템은 64MB 단위의 데이터 조각인 청크(chunk)로 대용량 파일을 관리하며 3중 이상의 백업과 청크 서버의 빠른 등록과 사용 등으로 고가용성과 확장성을 확보한다. 또한, 클라이언트가 증가하면 청크의 사본을 증가시킴으로서 Disk I/O나 네트워크의 대역폭을 확장시킨다. 메타 데이터는 마스터 서버에서 관리하며 실제 데이터는 청크 서버에서 관리하여 데이터 채널과 제어 채널을 분리하여 효율성을 향상시켰다. 단일 마스트 서버를 사용함으로 병목 현상이 있을 수 있으나 제어 채널 분리 및 클라이언트에서의 메타 데이터 캐쉬 등을 사용하여 완화할 수 있다.

아마존의 S3 오브젝트 스토리지는 5기가바이트까지의 오브젝트를 2킬로바이트 크기까지의 메타 데이타와 함께 저장할 수 있다. 오브젝트들의 집합은 버켓을 구성하며 각 버켓은 아마존 웹 서비스 어카운트에 의해 소유된다. 버켓은 버켓의 속성을 설명하는 정보인 메타 데이타와 클라이언트가 접근할 수 있는 권한 등을 나타내는 접근 정책을 가진다. 오브젝트들은 소속된 버켓 내에서 사용자가 할당한 유일한 키를 가지고 구분된다. 오브젝트는 속성을 설명하는 메타 데이타와 실제 데이터를 가진다. 버켓이나 오브젝트들은 Representational State Transfer(REST) 형태의 HTTP 인터페이스나 Simple Object Access Protocol(SOAP) 인터페이스를 사용하여 생성되고 조회될 수 있다.

3. 동기화 지원 스토리지 시스템

기존의 웹 하드 등의 스토리지 서비스와 비교하여 동기화 지원 고급 스토리지 서비스가 가지는 가장 큰 차별화 특징은 각 디바이스간의 자동 동기 기능의 지원이다. 즉 동기 폴더를 지정하면 그 폴더 내 모든 파일들은 자신의 사무실 PC, 집 PC, 그리고 스마트 폰과 같은 모바일 단말에서도 자동 동기화되는 것이다.

아래 그림 1은 동기화 지원 고급 스토리지 서비스를 위한 전체 플랫폼 구조로, 앞서 언급한 것처럼 파일 동기화에 초점을 맞추고 있다. 크게 프론트 엔드 시스템, 동기화 서브시스템, 스토리지 서브시스템, 그

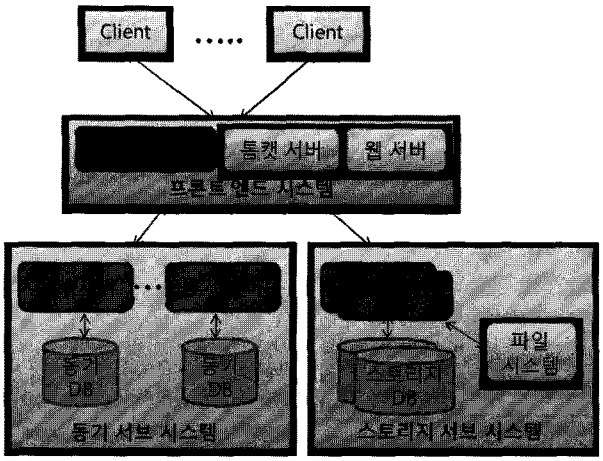

그림 1 동기화 지원 스토리지 시스템 구조

리고 지원 서비스와 같이 4가지 구성 요소로 나누어진다.

3.1 프론트 엔드 시스템

클라이언트에서 직접적으로 통신하는 관문에 있는 시스템들로 동기화 지원 스토리지 서비스를 인터넷에 노출시킨다. 즉, 클라이언트 프로그램이나 웹 프라우저들과 같은 클라이언트들이 스토리지 응용에 접속하는 시작 포인트이다. 프론트 엔드 시스템은 프락시 서버(Sync Proxies)와 자바 기반 웹 서버를 구성할 수 있는 톰캣(Tomcat) 서버 및 아파치 서버들로 구성될 수 있다. 프락시 서버들은 모바일이나 PC의 클라이언트에서 접속하는 시스템으로 클라이언트 연결을 위한 인증, 클라이언트 연결들의 통합(aggregation), 파일 전송, 그리고 클라이언트 메시지들을 적당한 동기 서버(sync server)로 전달하는 역할을 한다.

인증은 디렉토리 서비스와 결합하여 처리되는데 클라이언트는 프락시 서버와 연결이 성립된 후 사용자 키(credential)를 포함하는 인증 요구 메시지를 보낸다. 프락시 서버는 디렉토리 서비스에서 사용자를 검색 후 DB에 저장된 키와 제공된 키를 비교한다. 인증이 완료된 후 클라이언트는 질의나 수정 요구 메시지를 동기 프락시 서버에게 보낼 수 있고, 이러한 요구를 받은 후 프락시 서버는 디렉토리 서비스의 정보를 사용하여 사용자의 메타 데이터를 관리하는 특정 동기 서버에게 수신된 요구 메시지를 라우팅한다. 결과적으로 클라이언트는 자신들의 데이터가 어디에 보관되어 있는지 상관하지 않고 단순히 프락시 서버에게 요구사항을 전달하면 프락시 서버는 해당 동기 서버에게 그 요구사항을 라우팅한다. 마지막으로 프락시 서버는 클라이언트로부터의 파일 전송 요구를 처리하는데 이를 위해, 클라이언트 명령을 스토리지에서 사용가능한 REST 인터페이스로 매핑한다. 프락시 서버는 로드 밸런서를 통해 접근 가능하고 어떠한 상태 정보도 공유하지 않기 때문에 추가적인 로드를 지원하기 위해 수평적으로 쉽게 확장 가능하다.

톰캣 서버들은 스토리지 응용을 위해 웹 인터페이스를 구현한다. 이러한 시스템들을 통해 사용자들은 그들의 파일에 접근할 수 있고 자신의 계정을 관리한다. 톰캣 서버들은 동기 서브시스템과 상호작용을 통해 사용자 인증과 사용자 메타 데이터를 관리한다. 마찬가지로 톰캣 서버는 사용자 파일을 저장하고 조회하기 위해 스토리지 서브시스템과 상호 작용한다. 프락시 서버와 마찬가지로 로드 밸런서를 이용하여 수평적으로 확장 가능하다.

3.2 동기 서브시스템

동기 서브시스템은 사용자 메타 데이터의 관리와 접근을 제공하는데 동기 서버들과 동기 DB들로 구성된다. 동기 서버들은 동기화 지원을 위한 응용을 구현하고 동기 DB들은 원시 메타 데이터를 저장한다.

동기 서버는 타 백 엔드 시스템에서 사용자의 메타 데이터를 접근하기 위한 인터페이스를 제공한다. 즉 메타 데이타를 수정하고 조회하는 기본 인터페이스를 지원하고 다음과 같은 세 가지 추가적인 기능들을 수행한다. 첫째 사용자들이 자신의 데이터 외 다른 데이터에 접근하지 못하도록 제어하는 액세스 제어, 둘째 다중 클라이언트에서의 오브젝트 수정를 관리하기 위한 중재, 그리고 클라이언트가 관심 있어 하는 오브젝트가 수정되는 경우, 실시간 수정 정보를 제공하는 변경 통지 기능이다. 특정 오브젝트의 수정은 특정 동기 서버에 의해 처리되기 때문에 관심 있는 오브젝트의 변경 통지를 등록하거나 변경시 통지가 가능하다. 동기 서버들은 액티브-스탠드바이 설정으로 배치가 가능하며 액티브 서버가 실패시 자동으로 스탠드바이 시스템에게 제어를 넘겨줌으로써 고가용성을 제공한다. 각 쌍의 동기 서버들은 할당된 사용자 메타 데이터에 책임을 진다. 동기 서버들은 실제 메타 데이터를 저장하지 않고 단지 메타 데이타 접근을 제공하며 동기 DB들이 실제 메타 데이타를 저장한다. 각 사용자 파티션은 액티브-스탠드바이로 설정된 한 쌍의 DB를 가져서 중복 데이터를 유지한다.

디렉토리 서비스는 동기 서버와 DB 쌍으로 구성되며 사용자 메타 데이타가 아닌 사용자 정보를 가지며 일반적인 동기 서버들과 같은 프로토콜과 질의 인터페이스를 통해 접근 가능하다. 프론트 엔드 서버들은 다음의 두 가지 경우 디렉토리 서비스를 사용하는데, 첫

째는 사용자 인증을 위한 키 값을 찾을 때, 그리고 두 번째는 사용자를 사용자 메타 데이터를 관리하는 특정 동기 서버에 매핑하고 그에 따라 요구를 라우팅할 때 사용한다.

3.3 스토리지 서브시스템

스토리지 서브시스템은 타 구성 요소들이 파일 데이터를 저장하고 조회할 수 있는 단순 인터페이스를 제공한다. 스토리지 접근은 REST, 즉 HTTP 기반 인터페이스를 통해 제공되고 스토리지의 파일들은 Uniform Resource Locator(URL)로 구분된다. 스토리지 서브시스템은 파티션들로 나뉘는데 각 파티션은 특정 수의 스토리지 서버들과 스토리지 DB 쌍으로 구성된다. 스토리지는 파일 중복 체크를 위해 파일 해쉬(hash)에 의해 파티션된다. 파일을 특정 해쉬에 할당하는 일은 해쉬 서버에서 담당한다.

스토리지 서버는 스토리지 서브시스템에 의해 제공되는 REST 인터페이스를 구현하는데, 표준 HTTP 요구를 통한 파일 생성, 삽입, 조회 등의 기본적인 동작들을 지원한다. 스토리지 서버 인터페이스는 하부 파일 시스템에게 추상화를 제공함으로써 클라이언트에게 영향을 미치지 않고 물리 스토리지 메카니즘을 변경 가능하게 한다. 스토리지 서버는 다양한 하부 스토리지 구현을 지원하는데 로컬 파일 시스템이나 MogileFS [12] 및 아마존의 S3가 포함된다. 이들은 클라이언트 측의 수정 없이 단독으로 사용되거나 또는 조합하여 사용될 수 있다. 추가하여 스토리지 서버들은 다양한 부가 서비스들을 제공하는데 랜덤하게 생성된 키로 고급 암호 표준(AES: Advanced Encryption Standard)를 사용한 파일들의 암호화, 큰 파일들을 지원하지 못하는 파일 시스템을 위해 파일을 고정 크기로 나누는 청킹(chunking) 등을 지원하며, 또한 하나의 파일 시스템에서 다른 파일 시스템(예를 들어 MogileFS에서 S3로)으로 복제본을 만들 수 있다. 그리고 중복된 파일 제거(deduplication), 즉 이미 저장되어 있는 파일을 중복하여 업로딩하는 것을 방지하는 기능도 제공한다. 이를 위해 해싱(hashing)을 사용하는데 클라이언트는 파일을 업로드하기 전에 먼저 해싱하여 스토리지 서브 시스템에 이미 그 파일이 존재하는지를 확인한다. 이런 절차를 통해(중복되는 파일이) 비록 다른 사용자가 올린 것이라도 같은 내용의 파일을 두 번 중복하여 업로드하지 않는다. 스토리지 서버는 스토리지 DB에 파일 메타 데이터를 저장하고 각 스토리지 서버는 그 파티션과 연관된 하나의 DB와 상호 통신한다. 스토리지 서버들은 DB 밖에서 상태정보를 공유하지 않기 때문에 추가적인 스토리지 서버들은 파티션에 쉽게 추가될 수 있으며 파티션내의 스토리지 서버들에 대한 접근은 로드 밸런서에 의해 관리된다.

해쉬 서버들은 파일 생성 요구를 수용함으로써 파티션 정책을 구현하고 특정 파티션으로 클라이언트의 요구를 분배한다. 해쉬 서버는 스토리지 서버와 동일한 HTTP 기반의 파일 생성 프로토콜을 구현하기 때문에 클라이언트는 해쉬 서버와 통신하는지 알 필요가 없다. 파일은 파일 생성 요구 메시지에 포함된 파일 해쉬에 기반하여 특정 파티션으로 할당된다. 일관된 해쉬 알고리즘이 타겟 파티션을 결정하는데 사용되며 이로 인해 시스템으로부터 특정 파티션이 추가되거나 삭제되었을 때의 파일 분배시 생길 수 있는 지연을 최소화한다. 웹 기반의 업로드에서와 같이 파일 해쉬가 명시되지 않을 경우는 랜덤하게 선택된 파티션으로 파일이 할당된다. 타 서버들과 마찬가지로 해쉬 서버들 간 공유되는 상태 정보가 없기 때문에 추가적인 서버가 필요할 때 쉽게 배치할 수 있으며 역시 로드 밸런서를 통해 클라이언트들은 해쉬 서버를 접근하게 된다.

스토리지 DB는 스토리지에 파일들을 관리하기 위해 스토리지 서버에 의해 사용되는 메타 데이터를 저장한다. 동기 DB에 저장된 정보와는 달리 스토리지 메타 데이터는 스토리지 서브시스템 내부적으로 사용되며 사용자에게는 노출되지 않는다. 스토리지 DB 역시 쌍들로 구성되며 각 쌍이 하나의 파티션에 할당된다. 각 쌍은 액티브-스탠드바이 설정으로 두 개의 마스터 중복 데이터를 저장한다.

그림 2는 기본 로컬 파일 시스템으로 MogileFS를 이용해 구성한 스토리지 서브시스템의 예를 보여준다.

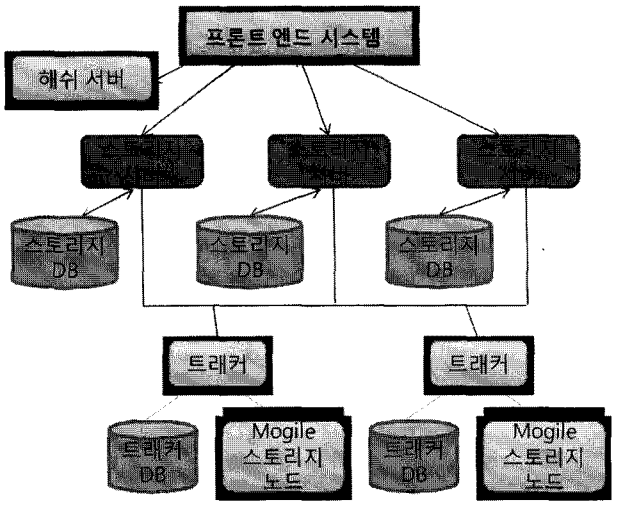

그림 2 MogileFS을 이용한 스토리지 서브 시스템의 예

MogileFS는 오픈 소스 기반의 분산 파일 시스템으로 수평한 네임 스페이스(namespace), 자동 파일 복제, shared-nothing 등의 특징을 가지며 RAID가 필요 없는 응용 레벨의 시스템이다. 분리된 디스크에 걸쳐 파일의 중복 복사(replication)를 저가의 낮은 오버헤드로 수행하는 데에 최적화되어 있다. MogileFS 인스턴스는 트랙커, 트랙커 DB, 스토리지 노드의 세 가지 노드 형태를 사용한다. 트랙커는 파일 시스템 네임 스페이스을 참조하여 클라이언트가 파일을 생성하고 위치시키는 것을 허용하며 파일 복제 과정을 관리한다. 트랙커 DB는 트랙커에 의해 관리되는 파일 메타데이타를 저장하며 스토리지 노드는 단순 HTTP GET과 PUT 요구를 통해 파일 데이타를 저장한다. 트랙커는 로드 밸런서와 함께 구성되며 연관 DB는 동기 DB 및 스토리지 DB와 마찬가지로 높은 가용성 지원을 위해 쌍으로 구성된다. 각 MogileFS 인스턴스는 다수의 스토리지 노드를 포함하며 각 스토리지 노드는 많은 수의 고용량의 디스크로 구성된다. MogileFS 인스턴스는 특정 스토리지 파티션에 종속되지 않기 때문에, 인스턴스가 관리하는 스토리지는 스토리지 서브시스템 내 전 파티션에 걸쳐서 활용될 수 있다.

아마존의 S3는 사용자 파일을 장기간 특정 장소에서 보관하고자 할 때 사용될 수 있으며 파일들은 클라이언트에 의해 업로드됨과 동시에 MogileFS에서 S3로 복제된다. 특정 기간이 지난 후 MogileFS의 로컬 복제본은, 새로운 파일들을 위한 공간을 늘리기 위해 삭제될 수 있으며, 그 이후에 그 파일이 필요시에는 S3에서 접근 가능하다. 스토리지 서버 인터페이스는 실제 데이터의 위치를 숨기기 때문에 클라이언트는 이러한 내부 전환을 알 필요가 없다.

3.4 지원 서비스

지원 서비스는 플랫폼 백 엔드에서 다양한 인프라 관련 태스크들을 수행하다. 대부분이 백그라운드에서 동작하는 프로세스들로 사용자 요구사항들을 직접적으로 처리하는 작업들은 아니다. 예를 들면 클라이언트가 이미지 파일을 다듬거나 사이즈 조정 같은 변환을 요구할 수 있도록 하는 웹 서비스 인터페이스를 제공한다.

4. 스위프트(Swift)

랙스페이스(Rackspace)와 NASA는 지난 7월에 오픈스택 (OpenStack) 프로젝트에 착수했는데, 이 프로젝트는 클라우드 서비스가 한 기술에 종속되지 않고 데이터와 애플리케이션을 한 클라우드에서 다른 클라우드로 더 쉽게 이전할 수 있도록 하는 것을 목표로 한 오픈 소스(open source), 오픈 표준(open standard) 기반의 개방 커뮤니티이다. 오픈스택은 출범 당시 델과 인텔, AMD, 라이트스케일, 시트릭스 등 35개 업체가 지지를 선언할 정도로 많은 기업들의 활발한 참여로 이루어지고 있다. 랙스페이스는 자사의 클라우드 스토리지 서비스인 클라우드 파일(Cloud Files)[13]의 소스를 오픈스택에 개방했는데 이 오픈 소스가 스위프트(Swift)이다.

스위프트는 오브젝트 스토리지로 데이터가 전 시스템에 걸쳐 골고루 분산되어 있으며, REST 기반 API를 지원하며 중앙 DB가 없다. 기본 개발 언어는 Pythone으로 특정 하드웨어에 무관하게 일반사양의 하드웨어를 지원하며 RAID 등의 구조를 필요로 하지 않는다. 파일시스템이 아니라 계층 구조를 지원하지 않으며 다수의 웹 서버들로 구성되며 사용자 계정에 대응하는 어카운트(account), 디렉토리에 대응하는 컨테이너(container)와 개별 파일에 해당하는 오브젝트(object)라는 논리적인 구조를 갖는다. 어카운트, 컨테이너 및 오브젝트는 각 각 다른 개수의 복사본(replica)을 가질 수 있으며 수 페타 바이트(peta byte)까지 확장성이 제공된다. 구글 파일시스템이나 아마존의 S3가 큰 오브젝트에 최적의 성능을 보이는 반면 스위프트는 큰 오브젝트 뿐 아니라 수 없이 많은 적절한 크기의 파일들 처리에도 아주 뛰어난 특징을 가진다.

최초 공식 릴리즈인 오스틴 릴리즈(Austin release)가 2010년 10월 21일 발표되었으며 차기 버전인 베어 릴리즈(Bexar release)가 2011년 2월 목표로 개발중에 있다. 오스틴 릴리즈의 추가 사항으로는 컨테이너별 오브젝트에 대한 접근 제어(access control)를 지정할 수 있으며 누구나 접근 가능한 공개 컨테이너(public container)를 지정할 수 있도록 한 것이다. 또한 어카운트나 컨테이너에 대해 사용자 정의 메타 데이터를 구현하여 메타 데이타를 API를 통해 접근 가능하게 했으며 로그 파일(log file)들을 묶어서 분기별, 월별, 주별, 일별로 시간당 통계 화일을 어카운트 단위로 제공한다. 통계에는 어카운트 개수, 바이트 사용량, 컨테이너 개수, 오브젝트 개수 등의 스토리지 사용량과 사용 대역폭(incoming bandwdith, outgoing bandwidth) 등을 포함한다.

4.1 논리적 구성

파일 시스템과 같은 다중 계층적 구조는 지원하지 않으나 어카운트, 컨테이너, 오브젝트의 3단계 계층을

갖는다. 사용자는 어카운트들을 가지며, 하나의 어카운트내에 여러 컨테이너(디렉토리)가 포함되며 하나의 컨테이너내에 여러 오브젝트(파일)들이 포함된다.

그림 3은 스위프트의 논리적인 구성도를 보여주는데, 논리적으로는 어카운트 내에 컨테이너들이 존재하며, 컨테이너 내에 오브젝트들이 존재한다. 그러나 물리적으로 이 세 논리적 엔티티들은 각 각 독립적으로 존재하는 해당 서버들에 의해 조작 된다. 즉, 복수개의 어카운트 서버, 컨테이너 서버, 그리고 오브젝트 서버들이 개별적으로 존재한다. 어카운트 서버는 자신이 관리하는 어카운트의 컨테이너 리스트들을 관리하며, 컨테이너 서버는 자신이 관리하는 컨테이너의 오브젝트 리스트를 관리한다. 어카운트 서버와 컨테이너 서버는 SQLite DB로 리스트를 저장한다. 오브젝트 서버는 실제 오브젝트의 저장 및 조회 등의 운영을 지원한다. 오브젝트들은 로컬 파일 시스템에 실제 바이너리 파일들로 저장되며 메타 데이터는 파일의 확장 속성(xattrs)에 저장된다. 따라서, 확장 속성을 제공하는 파일 시스템이면 어떤 것이던 로컬 파일 시스템으로 활용 가능하다.

스위프트의 실제 동작은 다음과 같다. 클라이언트는 어카운트 생성시 인증을 위해 인증 시스템(Auth system)에 접속한다. 인증 시스템은 스위프트 밖에서 인증을 수행하는 서버로 인증된 클라이언트에게 토큰과 프락시 서버(Proxy Server)의 주소(URL)를 전달한다. 클라이언트는 전달 받은 토큰과 주소를 사용해 프락시 서버에 요구사항을 전달한다. 프락시 서버는 스위프트의 모든 요소들을 묶어주는 역할을 하며 클라이언트들의 요구사항에 대해 링(Ring)을 참조하여 어카운트, 컨테이너, 오브젝트의 물리적인 위치를 파악하여 해당 서버에 전달하는 역할을 하며 모든 응답 또한 프락시 서버를 통해 클라이언트에게 전달된다. 공개 Application Program Interface(API)도 프락시 서버를 통해 외부에 노출되게 된다. 링은 논리적 엔티티, 즉 어카운트, 컨테이너 및 오브젝트의 물리적인 위치를 매핑하는 역할을 하며 링 빌더(ring bulider)라는 도구에 의해 외부에서 수동적으로 만들어진다. 업데이터 서버(Updater Server)는 즉각적인 수정(update)이 실패한 동작에 대한 처리를 하며, 오디터 서버(Auditor Server)는 오브젝트, 컨테이너, 어카운트의 무결성 검사를 주로 하며 만약 결점이 발견되면 다른 사본(replica)으로 부터 복제하여 대체하는 역할까지 하고 있다.

복제 서버(Replicator)는 어카운트와 컨테이너 DB의 복제를 담당하는 DB 복제 서버와 오브젝트 파일의 복제를 담당하는 오브젝트 복제 서버가 별도로 존재한다. 복제 과정은 비동적기적이고 peer-to-peer 방식으로 처리되며 푸시 모델(push model)을 사용한다. 복제본의 배치는 링에 의해 처리되며 복제 서버가 원격 드라이버의 실패를 감지할 경우는 링의 인터페이스를 통해 대체할 노드를 선택하게 된다. 삭제된 파일이나 레코드는 툼스톤(tomstone)으로 마크되고 툼스톤으로 마크된 엔티티들은 일정 시간(consistent window) 후에 제거되는 지연 삭제 방식을 사용한다.

4.2 물리적 구성

스위프트 스토리지의 물리적인 구성 역시 계층적인 구조를 따르며 클러스터(Cluster) - 존(Zone) - 디바이스(Device) - 파티션(Partition) 으로 그림 4와 같이 구성된다.

클러스트는 전제 스위프트 인스턴스를 포괄하며 존은 각 오브젝트의 복제본(replica)를 물리적으로 다른 지역에 분산하기 위해 도입된 개념으로 하나의 존은 하나의 드라이버, 서버, 캐비넷, 스위치, 또는 데이터 센터까지 유연하게 해당될 수 있다. 하나의 오브젝트의 복제본은 동일 존 내에 위치할 수 없고 다른 존에 위치하게 하여 오브젝트의 신뢰성과 가용성을 높인다. 하나의 클러스터내에 최소 5개의 존을 권고하고 있다.

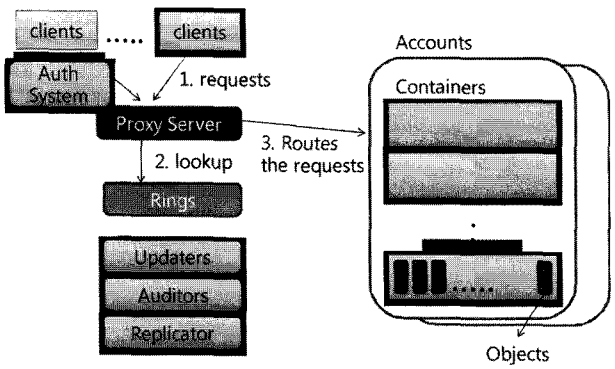

그림 3 스위프트의 논리적 구성도

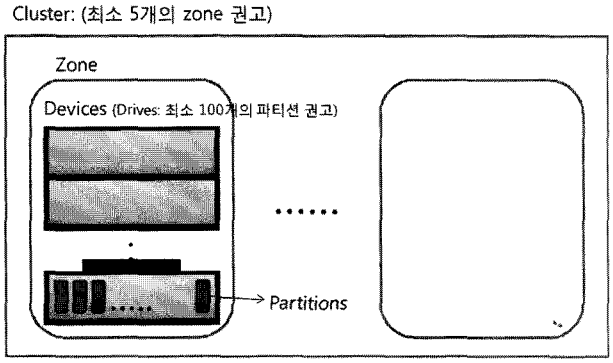

그림 4 스위프트의 물리적인 구성도

존은 다수의 디바이스를 포함하며 하나의 디바이스는 다수의 파티션으로 구성된다. 하나의 디바이스는 최소 100개의 파티션을 권고하고 있다. 파티션은 가장 낮은 단계의 스토리지 단위이며 실제 오브젝트들이 저장되는 곳이다.

디바이스에 가중치(weight) 값을 설정해 클러스터 내 각 디바이스에 파티션이 골고루 분산되도록 하기 위해 사용된다. 가중치 값은 클러스터내에 다른 크기의 디바이스들이 존재할 때 유용하게 사용할 수 있다.

4.3 Ring 구조 및 동작

링은 실제 스토리지에 저장되어 있는 각 엔티티 이름과 클러스터내 물리적 파티션 위치를 매핑하는 역할을 하며 어카운트, 컨테이너 그리고 오브젝트에 대해 별도의 링이 존재한다. 링은 이러한 매핑을 존, 디바이스 그리고 파티션 개념을 이용하여 유지하며 링 내의 각 파티션은 클러스터내 디폴트로 세 개 복제되어 존재한다. 링은 또한 조회 실패 시나리오에서 핸드오프(handoff) 할 디바이스를 결정하는 책임도 가진다. 링은 프락시 서버에 의해 주로 사용되고 그 외 복제 서버와 같은 백그라운드 프로세서에 의해 사용된다.

링은 링 빌더에 의해 외부에서 수동적으로 만들어지고 관리되는데 링 빌더는 파티션을 디바이스에 할당하는 일을 하며 빌더 파일(builder file)을 유지하여 링을 관리한다. 링 빌더는 여러 개의 빌더 파일(builder file)을 유지하는데 이 파일에는 링 정보와 추 후 링을 새로 만들기 위해 필요한 추가적인 데이터들을 유지한다. 링 빌더는 디바이스의 가중치(weight) 값에 따라 이상적인 파티션 개수를 결정하며, 각 파티션의 복제본을 디바이스에 할당하고 다시 밸런스를 맞추는 작업들을 반복한다.

링의 데이터 구조는 크게 두 가지 리스트를 포함하는데 첫째가 디바이스들의 리스트이고 두 번째가 파티션 할당 리스트이다. 클러스터내의 모든 디바이스들의 정보는 디바이스 리스트에서 관리되며 각 디바이스마다 디바이스 ID, 소속된 존, 가중치, IP 주소 및 메타 데이터 등의 정보를 관리한다. 파티션 할당 리스트는 그림 5에서 보는 것과 마찬가지로 이중 배열로 구성되는데 디바이스 ID 배열의 리스트이다. 디바이스 ID 배열은 파티션의 개수만큼의 길이를 가지며 이 배열을 복제본의 개수만큼의 리스트로 관리하게 된다. 디바이스 ID 배열은 각 파티션이 어느 디바이스에 할당되는지의 관계를 저장하고 있다. 그림 5에서 각 각의 컬럼이 한 파티션이 저장되는 3개의 디바이스 ID를 보여주고 있다.

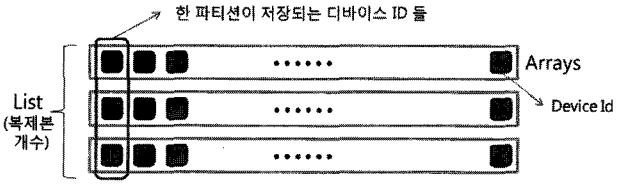

그림 5 스위프트의 파티션 할당 리스트

링 빌더가 링을 생성시 한 파티션이 저장되는 디바이스는 다른 존에 있는 디바이스로 구성되도록 하며 각 디바이스의 가중치를 고려하여 파티션을 디바이스에 할당한다. 프락시 서버에서 각 엔티티를 저장하는 요구사항을 받으면, 각 엔티티들의 패스 (path)를 MD5 해쉬 함수의 입력으로 하여 그 결과값의 일부를 이용하여 저장될 파티션을 결정하게 된다. 이렇게 결정된 파티션이 어느 디바이스에 할당되었는지를 알기 위해 링을 참조하게 되는 것이다.

5. 개선 방향 및 발전 방향

앞 장들에서 동기화 지원 스토리지 시스템과 오브젝트 스토리지 시스템인 스위프트의 구조 및 기능들에 대해 살펴보았다. 이 장에서는 앞의 두 시스템의 장단점을 비교해 보고 어떤 개선 사항이 있는지 등의 발전 방향을 논의해 보고자 한다. 표 1은 두 시스템을 비교한 것이다.

이미 언급한 것처럼 동기화 지원 스토리지 시스템이 더 다양한 고급 기능들을 제공한다. 대표적으로 동기화 지원 스토리지 시스템은 파일 해쉬를 기반으로 파일을 배치함으로써 파일 중복 방지(deduplication)를 지원한다. 그러나 스위프트는 단순한 오브젝트 스토리지 서비스에 아주 적합한 구조를 가지고 있다. 주요 장점으로는 스토리지 노드들간의 로드 리밸런싱을 자동으로 지원하는 기능과 복제본을 지역적으로 분리된 존에 저장함으로 가용성을 높이는 점 등이다. 살펴본 바와 같이, 이 두 시스템은 목표 서비스 모델과 고객군

표 1 동기화 지원 스토리지 시스템과 스위프트의 비교

	동기화 지원 스토리지 시스템	오브젝트 스토리지 스위프트
특징 (차별화 포인트)	개인 PC 및 다양한 모바일 디바이스와의 자동 동기	큰 파일뿐 아니라 공간히 많은 작은 파일들까지 지원하는 오브젝트 스토리지
하부 지원 파일 시스템	다양한 파일 시스템 지원: 로컬 파일 시스템, MogileFS, Amazon S3	XFS로 테스트하였으나 Extended Att를 지원하면 하부 로컬 파일 시스템으로 사용 가능
가용성 (Availability)	지역적 분산 개념은 없으나 응용 주요 메타 데이터의 마중화 및 파일 등록 지원	지역적으로 분리된 존에 복제본을 둠으로 높은 가용성
노드 자동 rebalance	미지원, 단순 해시함수로 partition 결정함	지원, 각 노드의 가중치에 따라 load balance 함
파일 Chunking	큰 파일을 지원하지 않는 파일 시스템을 위해 지원	5G 이상의 파일 지원을 위한 기능 개선 기능임 (Blueprint에 추가됨)
파일 암호화 (encryption)	지원 (AES 사용)	미지원
중복 파일 업로드 방지 (Deduplication)	지원 (기존에 존재하는 파일 업로드 방지, 타 user가 올린 파일 포함)	미지원

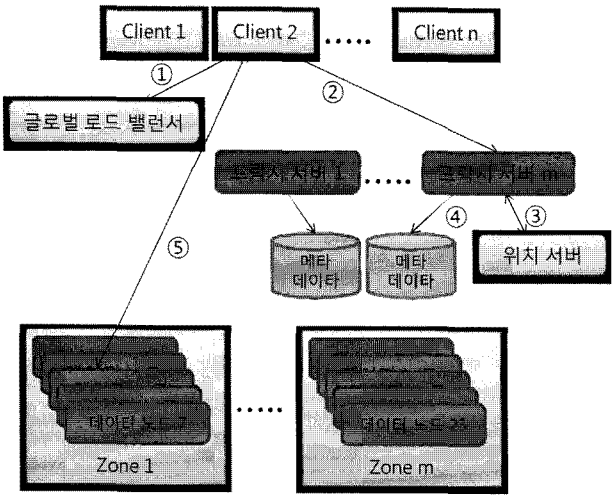

그림 6 스토리지 시스템 개선안

이 다르다. 동기화 지원 스토리지 시스템은 동기화나 파일 암호화가 지원되는 고급 스토리지 서비스를 목표로 할 수 있으며 스위프트는 아마존의 S3와 같은 단순한 백업 스토리지 서비스를 목표로 할 수 있다.

이 두 스토리지 시스템에서 개선될 부분은 서로 목표하는 고객군과 서비스가 다르기 때문에 서로가 없는 기능을 보완하는 것이 아니다. 다음에서는 두 시스템을 개선하기 위한 몇 가지 방안에 대해 살펴보도록 한다. 우선 스토리지에 파일을 업로드하거나 다운로드할 때 프락시 서버를 거쳐서 처리를 하기 때문에 처리 속도 및 프락시 서버의 부하가 증가하게 된다. 클라이언트에서 직접 데이터 노드에 접속하여 제어 흐름과 데이터 흐름을 분리할 필요가 있다. 그러나 이 경우에는 데이터 노드들 역시 API 인터페이스 등을 이용해 클라이언트와의 직접 통신을 지원하여야 한다. 그리고 다중 프락시 서버를 구성하고 메타 데이타를 이중화하여 모든 프락시 서버에서 독립적으로 메타 데이타를 참조하는 모델이 이상적이다. 글로벌 로드 밸런서를 두어 클라이언트를 프락시 서버에 할당해 주고 또한 인증까지 겸할 수 있게 하면 효율적인 구성이 될 것이다. 마지막으로 스위프트에서처럼 데이터 노드들을 물리적인 존으로 나누어 구성하되 위치 서버를 두어 클라이언트에 가장 가까운 데이터 노드를 선택할 수 있도록 하면 데이터 업로드 및 다운로드시 트래픽이나 지연을 줄여 전체 성능에 도움이 될 것으로 사료된다. 위에서 제시한 개선안들을 그림 6에서 전체적으로 도식해 보았다.

6. 결론 및 앞으로의 연구 방향

클라우드 컴퓨팅은 현재의 IT 트랜드에서 가장 주목 받는 영역으로 인터넷을 통해 시간에 구애 없이 온 디맨드(on demand)로 쓴 만큼 과금하는 서비스 유형을 탄생시켰다. 본 논문에서는 인프라스트럭처를 서비스로 제공하는 IaaS 클라우드 서비스 중 스토리지 서비스를 제공하기 위한 두 가지 시스템, 즉 동기화 지원 스토리지 시스템과 오브젝트 스토리지인 스위프트의 구성 및 기능 등을 살펴보았다. 이 두 시스템은 각 각 타겟 고객이 다르고 목표 서비스가 다르기 때문에 단순 비교가 어려우나 이 두 시스템을 기반으로 전체적인 스토리지 시스템에서의 개선 사항을 도출해 보았다. 주요 개선 사항으로는 메타 데이타의 이중화로 다중 프락시 서버에서의 독립적인 접근 보장, 위치 서버 도입으로 실제 데이터 노드의 물리적 위치를 고려한 파일 할당 등이다.

향후에는 도출된 개선 사항을 직접 시스템에 적용할 수 있는지를 검증해 볼 계획이다. 예를 들어 스위프트에서 멀티 프락시 서버로 구성 후 링을 이중화하여 각 프락시 서버에서 독립적 접근이 가능한지에 대한 feasibility 테스트 등 을 수행해 볼 계획이다. 또한 위치 서버를 구체화하여 Content Delivery Network(CDN) 등의 응용 적용에 대한 가능성을 타진해 보고자 한다.

참고문헌

[1] Armbrust, M., Fox, A., Griffith, R., D. Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M., "A View of Cloud Computing", Communication of the ACM, Vol. 53, No. 4, pp. 50-58, 2010

[2] NIST 클라우드 서비스 분류, The NIST Definition of Cloud Computing, version 15 (2009), http://csrc.nist.gov/groups/SNS/cloud-computing/

[3] 아마존의 웹 서비스, http://aws.amazon.com

[4] OpenStack 프로젝트 사이트, http://openstack.org, wiki http://wiki.openstack.org, 개발자 커뮤니티 https://launchpad.net/openstack

[5] Swift 다큐먼테이션 사이트, http://swift.openstack.org/

[6] RAID의 Wiki, http://en.wikipedia.org/wiki/RAID

[7] 민영수, 진기성, 김홍연, 김영균, "클라우드 컴퓨팅을 위한 분산 파일 시스템 기술 동향", ETRI 전자통신동향분석 제24권 제4호, 2009년

[8] Factor, M., Meth, K., Naor, D., Rodeh, O., "Object Storage: The Future Building Block for Storage Systems", In Proc. of the 2nd International IEEE Symposium on Mass Storage Systems and Technologies, 2005

[9] 김영철, 박근태, 이상민, 김홍연, 김영균, "클러스터 파일 시스템 기술 동향", ETRI 전자통신동향분석 제22권 제6호, 2007년
[10] Ghemawat, S., Gobioff, H., Leung, S., "The Google File System," In Proc. of ACM Symp. on Operating Systems Principles, pp.20-40, 2003.
[11] 아마존의 S3 Wiki, http://en.wikipedia.org/wiki/Amazon_S3, 아마존의 S3 사이트, http://aws.amazon.com/s3/
[12] MogileFS 프로젝트 홈 페이지 및 Wiki, http://code.google.com/p/mogilefs
[13] Rackspace의 Cloud Files 사이트, http://www.rackspacecloud.com/cloud_hosting_products/files

약 력

김 미 점

1993 부산대학교 전자계산학과 졸업
1995 부산대학교 전자계산학과(이학석사)
2006 텍사스 주립대학교 컴퓨터공학과(이학박사)
1995~현재 KT 종합기술원
관심분야 : Cloud Storage System, SaaS(PaaS) Platform, USN/M2M 미들웨어(Platform)
E-mail : mjkim@kt.com