The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

Google, Inc

2014104141 장연우

1. Motivation

데이터 집약적인 애플리케이션의 등장과 엄청나게 많은 데이터가 쌓여가는 2000년대 초반. 기존의 파일시스템으로는 많은 데이터의 양을 처리하기에 한계를 느끼고 있었고, 또한 늘어나는 저장소에서의 고장은 예외적으로 일어나기보다는 이제 평범하게 일어나는 일이 되었다. 이에 하드웨어 중 일부가 고장 나도 데이터 복구가 가능해야 하며 클라이언트에게 정상적인 기능을 제공할 수 있는 확장 가능한 새로운 파일 시스템을 다시 고안해야했다. 그 설계의 목표는 빠른 데이터 프로세싱 뿐만 아니라 성능, 확장성, 신뢰성, 유효성 등 기존의 파일시스템이 요구를 목표를 충족해야 한다.

이에 따라 구글은 수천 대의 머신에서 수천 개의 디스크를 통해 동시에 수백 대의 클라이언트들이 접근해도 문제 없는 파일시스템을 설계하였고 구글 산업 전반에 배포되어 활용되게 되었다.

2. Background

구글 파일 시스템(이하 GFS)은 설계의 앞서 과거와 달리 현재 예상되는 애플리케이션의 작업 부하 및 기술적 환경을 예측하여 GFS에 주요한 관점에서 세밀한 가설을 둔다.

첫째, 시스템의 고장은 예외가 아니라 일상적이며 고장 날 수 있는 값싼 요소이다.

둘째, 시스템은 일반적으로 100Mb이상인 큰 수백만개 파일을 다룬다.

셋째, 파일은 한번 쓰여지면 거의 수정되지 않으며 데이터 추가는 큰 리소스가 필요.

넷째, latency보다 대역폭을 사용하는 것을 중요하게 여김. 즉 대역폭을 분할하지 않음.

3. Method or Solution Figure

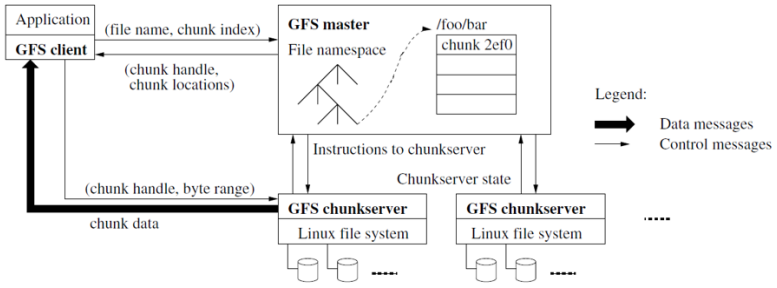

Figure 1 : GFS Architecture

Figure 1은 GFS의 전체적인 구조도이다. GFS는 하나의 마스터와 여러 개의 청크서버들로 구성되며 여러 클라이언트들에 접근된다.

마스터는 파일의 메타데이터와 청크서버들간의 청크이동과 같은 전반적인 활동 제어를 하며 청크서버에게 지시나 상태 파악을 위해 하트비트라는 메시지를 주기적으로 송신한다.

청크서버는 파일을 고정된 크기로 자른 하나의 조각을 뜻하는 청크들을 디스크에 담아 저장하며 청크들의 위치들을 저장하고 관리한다.

이에 단순히 클라이언트가 GFS에 파일 읽기 작업을 요청했을 때, Figure 1과 함께 설명해보면, 클라이언트는 파일의 이름과 파일 내의 조각 인덱스 오프셋을 계산한다. 이후 마스터에게 계산한 정보를 요청하면 마스터는 청크핸들과 청크위치를 반환하게 된다. 클라이언트는 이 정보를 키로 삼아 청크서버에 접근하여 데이터를 얻어 간다. 이러한 과정을 통해 마스터게만 트래픽이 부과되어 병목현상이 일어나는 것을 방지할 수 있으며 하나의 마스터로 시스템 전반을 관리할 수 있는 이점이 있다. 반면에 쓰기 작업을 요청했을 때에는 생성시점에 마스터의 의해 64비트의 번호를 부여받고 청크서버의 로컬디스크에 저장된다. 신뢰성을 위해 데이터의 복제본을 만들어 다른 청크서버에 각각 저장해둔다. 기본적으로는 3개가 디폴트 값이다.

청크사이즈를 일반적인 파일시스템보다 큰 64Mb를 사용함은 GFS의 큰 특징 중 하나이다. 이는 클라이언트가 데이터에 대해 마스터와 통신하는 횟수를 줄여주고, 고정된 TCP연결을 유지해야 할 필요가 없어 네트워크 부하를 줄일 수 있었다. 또한 전체적인 청크의 수를 줄여 마스터가 관리해야 할 메타데이터의 양이 크게 줄었다

마스터가 관리하는 메타데이터는 청크 간의 네임스페이스, 매핑정보, 레플리카의 위치정보, 로그정보가 저장된다. 네임스페이스와 매핑정보, 로그는 마스터의 로컬리스크에 영구저장 시키며 리모트 머신에 백업까지 한다. 이렇게 메타데이터을 백업해두면서 시스템에 치명적인 오류가 났을 때, 복구 할 수 있게 안전성을 높인다.

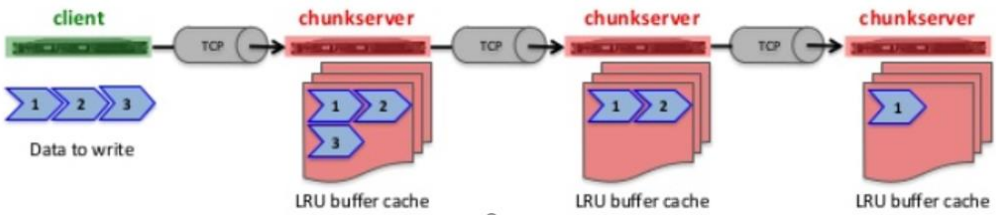

Figure2 : Data flow

목표는 네트워크 대역폭을 최대로 활용하고, 네트워크 병목같은 지연시간 방지, 데이터 처리하기 위한 대기시간 최소화. 각 시스템의 대역폭을 최대한 활용하기 위해 데이터는 다른 토폴로지(트리)에 분산되지 않고 청크서버에 체인을 따라 선형적으로 푸시된다. 따라서 각 시스템의 전체 아웃바운드 대역폭은 여러 수신자로 분할되지 않고 최대한 빨리 전송하는데 사용한다. 직접 전송할 수 없는 경우 데이터를 가까운 청크서버로 전달해 거쳐보낸다.

GFS은 snopshot이라는 데이터복제기능을 제공한다. 이는 매우 빨리 이루어져 사용자가 딜레이를 느끼기 어려울 정도이며 매카니즘은 다음과 같다. 복제할 데이터가 들어있는 청크의 메타데이터를 마스터는 작업을 디스크에 기록한다. 그 뒤 이를 레코드를 메모리 내 올린다. 이때 새로 생성된 스냅샷 파일은 원본 파일과 동일한 청크를 가르킨다. 스냅샷 후 클라이언트가 처음으로 청크C에 기록하려고 할 때 마스터는 그 청크를 가르키는 메타데이터가 2개임으로 이상현상을 감지하고 클라이언트 요청에 거부한뒤 새 청크 C'를 현재 복제본이 있는 각 청크 서버에 C'라는 새 청크를 만들 것을 요청하여 기능을 구현했다. 이는 원본과 동일한 청크서버에 청크를 할당하므로 네트워크가 필요하지 않고 로컬에서 아주 빠르게 생성하는 것을 보장한다.

4. Analysis of experimental results

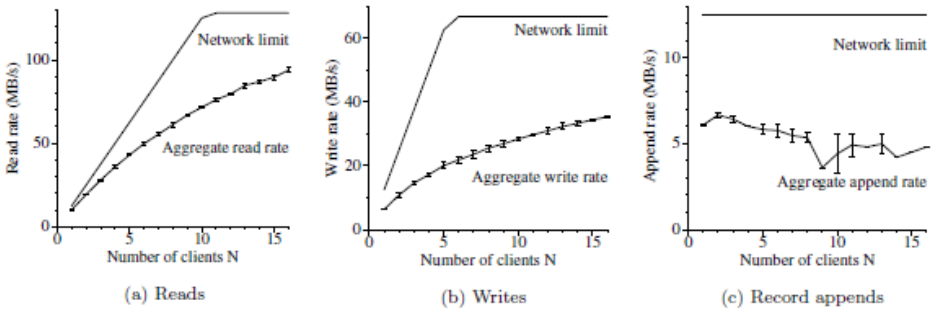

Figure 3: Aggregate Throughputs.

2003년 구글에서 16개의 청크서버, 클라이언트와 하나의 마스터(1.4Ghz cpu, 2G ram, 5400rpm disk)일 때 읽기, 쓰기, 추가 모드일 때 Network Limit(이론값)와 실제 rate(속도)를 비교해본 표다.

읽기모드 일 때, 이론값을 비교적 잘 따라 갔지만 물리적(기온, 손실률)로 인해 어느정도 일치하는 모습을 보였으며 쓰기모드 일 때, 클라이언트가 어느정도 추가 되면 이론 값과 멀어지는 것을 볼 수 있다. 이는 청크 복제가 3개로 인하여 클라이트 증가에 따라 네트워크에 부하가 심해져 보여지는 것으로 보인다. 추가모드일때는, 동일한 파일에 각 클라이언트가 추가를 할 때 마지막 데이터를 추가하는 청크의 네트워크 대역폭에 의해 속도가 정해지므로 속도가 일정하지 않은 모습을 보였다.

5. Opinion or Improvement point

네트워크가 발전할수록 상상할 수 없을 정도록 큰 데이터가 생성되는 오늘날 기존의 파일 시스템을 업그레이드를 하는 것으로는 데이터의 증가 속도에 따라가지 못할 것 같다. 몇 년 뒤에 GFS도 감당할 수 없는 양으로 늘어난다면, 늘어나는 데이터의 속성을 냉철히 판단하고 현실적으로 다룰 수 있는 architecture를 설계 할 수 있는 창의적인 생각을 해보게 된다. 지금의 GFS에 SSD로 교체한다면, SSD는 HDD보다 빠른 히트가 큰 장점인데 GFS의 데이터위치를 빠르게 찾아가는 구조 상 그다지 효율적이지 못할 것이 아쉬운 점이다. SSD 값이 점점 낮아지는 추세인데 이후 파일시스템은 SSD를 활용하여 만들면 차세대 파일시스템으로 구현해 본다면 좋을 것이다.