The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

Google

2016104124 박해연

- Motivation

  파일 시스템이 다루어야 할 데이터는 점점 커지고 데이터를 처리하는 데에 있어서 더 많은 요구사항들이 생겨났다. 잦은 component failure 에 대해서 적절하게 대응하지 못했고, 파일은 이전의 파일시스템이 감당하기 어려울 만큼 커졌다. 데이터는 over write, random write 보다는 append new data 를 하는 경우가 많다는 관찰을 적절하게 활용할 수 있는 파일시스템이 필요했다. 기존의 파일 시스템은 이 역할들을 수행하는데 어려움이 있었고 그 대안으로 나온 것이 분산 파일시스템인 Google file system 이다.

  이렇게 해서 나오게 된 구글 파일 시스템은 몇가지 가정을 염두에 두고 설계되었다. component failure 는 빈번하기에 지속적인 모니터링이나 복구 매커니즘이 필요하다는 것, 파일의 크기는 100 메가 이상인 경우가 대부분이라는 것, 같은 파일에 여러 클라이언트가 동시에 접근할 경우에 대한 처리가 분명해야 하고, 대역폭(bandwidth)을 나누어서 작업을 수행하는 것보다 latency 가 있더라도 대역폭을 최대로 사용하는 것이 중요하다는 점이 고려사항들이다.

- Architecture
  - Interface: POSIX 는 아니지만 친숙한 인터페이스 API 를 가지고 있다. gfs 의 인터페이스는 일반 파일시스템이 제공하는 기본적인 연산들(read, write, create 등)을 모두 제공하고 거기에 snapshot 과 record append 연산도 추가적으로 가지고 있다.
  - 전체 구조: 싱글 마스터와 여러 chunk server 로 이루어져 있으며 client 와 상호작용한다. 파일은 고정된 크기의 chunk 에 나누어져 저장이 되고 chunk server 는 이 chunk 를 리눅스 파일처럼 가지고 있다.

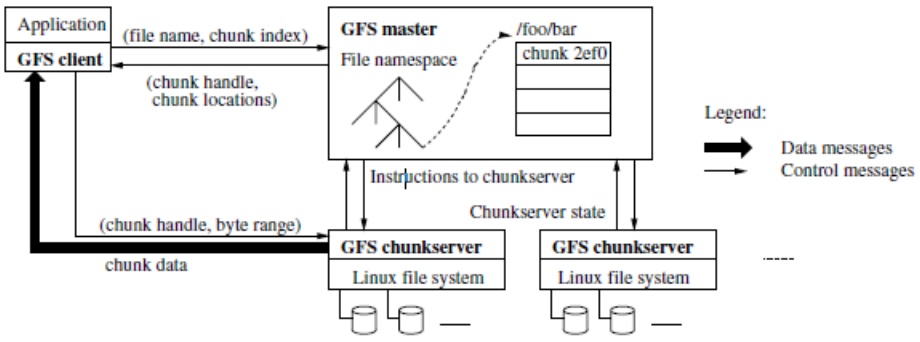
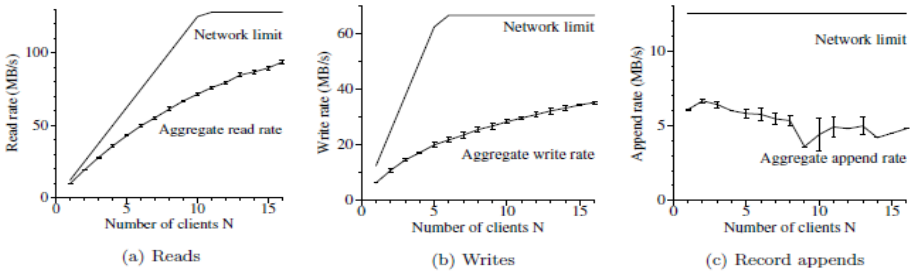


Figure 1: GFS Architecture

- Master: 마스터는 파일 시스템에 대한 메타데이터를 가지고 있다. Chunk lease management 나 garbage collection(뒤에서 설명)과 같은 구글 파일 시스템에 전반적인 활동을 관리한다. Chunk server 에게 명령을 보낸다거나 chunk server 의 상태를 모니터링하기 위한 용도로 heartbeat message 를 주기적으로 보낸다.
- Chunk server: chunk 를 리눅스 파일처럼 로컬디스크에 저장하고 있다. Reliability 를 위해 chunk server 에 문제가 생겼을 때를 대비해서 chunk 에 대한 복사본을 여러 chunk server 에 둔다. 이 복사본의 개수는 조정할 수 있으며 default=3 이다.
- Client: client 는 마스터로부터 필요한 파일을 저장하는 청크에 관한 메타데이터를 얻고, client 는 이 메타데이터를 가지고 chunk server 에 가서 필요한 연산을 수행한다. 이렇게 master 와 chunk server 양측에서 일을 분담하게 되어서 네트워크 병목현상을 막고 전체적인 시스템 performance 를 높이게 되었다.
- Read operation 시 시스템의 동작: 우선 client 는 master 에게 필요한 파일이 있는 chunk 정보를 요청한다. 그러면 master 는 해당되는 chunk 의 고유 번호인 chunk handle 과 chunk 의 위치를 알려준다. Client 는 이 정보를 caching 해둔다. 이후 파일을 다시 연다거나 캐시에 이 정보가 남아있을 때까지는 client 가 master 에게 연락하지 않아도 된다. 또한 보통 한번에 여러 chunk 정보를 master 에게 요청하고 받기 때문에 오버헤드를 줄일 수 있다. Client 는 이 chunk 를 가진 복제본 중 하나에 read 요청을 한다. (가장 가까운 chunk server 로 이 요청을 보낸다.) 그리고 원하는 chunk data 를 얻게 된다.
- Large chunk size: gfs 에서 데이터를 저장하는 단위인 chunk 는 64mb 로 꽤 크다. 이렇게 chunk size 를 크게 설계한 이유가 있다. 한번에 많이 읽어 들일 수 있기 때문에 마스터와 클라이언트간의 communication 을 덜해도 되고, 마스터가 보유해야하는 metadata size 도 작게 되고, client 가 chunk server 와 통신하는 데에 있어서 네트워크 오버헤드도 줄일 수 있다. 물론 단점도 있다. 작은 파일의 경우 적은 수의 chunk(하나의 chunk 이기도)에 저장되는데 만약 client 1000 명이 동시에 이에 접근하려고 한다면 1000/3 의 요청을 chunk server 가 처리해야 하는 상황이 생긴다.(hotspot 이 된다)
- Metadata: 메타 데이터에는 3 가지 종류가 있다. 파일과 chunk 의 namespace, 파일과 chunk 의 매핑, 각 chunk 복사본의 위치이다. 이 3 가지는 모두 master memory 에 저장되어 있다. Chunk 복사본의 위치를 제외하고는 모두 log 로 master disk 에 저장되어 있다.
- Consistency model: namespace 에 대한 mutation 은 atomic 하게 이루어진다. Mutation 을 거친 파일의 상태는 mutation 의 종류가 무엇인지, mutation 이 성공했는지, 다른 mutation 과 동시에 요청 되었는지에 따라서 나뉜다. 파일의 상태가 consistent 하다는 것은 모든 클라이언트가 동일 파일에 대해서는 같은 내용을 본다는 것이다. Defined 하다는 것은 consistent 한 상태에 추가적으로 모든 mutation 이 다 반영된 상태임을 나타낸다. Undefined 하지만 consistent 하다는 것은 즉 동시에 mutation 을 하게 된 상황에서 모든 클라이언트는 같은 내용을 보지만 이것은 하나이상의 mutation 이 반영되지 않은 상태이다. Mutation 이 아예

실패했다면 inconsistent 한 상태가 되어 클라이언트들이 보는 데이터가 불일치하게 되는 상황이 발생한다.

- System interactions
  - Leases and mutation order: 모든 mutation 은 모든 복제본에도 반영이 되어야 한다. Lease 는 복사본간에 일관된 mutation order 를 유지하기 위해 사용되는데 master 는 replica 중 하나에 이 chunk lease 를 주고 이 복사본은 primary replica 가 된다. Primary replica 는 청크의 모든 mutation 에 순서를 부여한다. 모든 복사 본은 이와 동일하게 mutation 순서를 유지한다.
  - Decoupling control flow and data flow: control flow 와 data flow 를 나눔으로써 network 를 보다 효율적으로 활용했다. Data 는 파이프라인 방식으로 linearly 하게 push 된다. 이때 네트워크 대역폭을 나누지 않고 한번에 최대로 보내도록 한다. Client 와 chunk server 는 데이터를 가장 가까운 chunk server 로 전송한다.
  - Snapshot: 스냅샷이란 파일이나 디렉토리를 순식간에 복사하는 GFS 의 기능이다. 마스터는 스냅샷을 할 파일의 chunk 에서 lease 권한을 회수하고 해당 파일의 메타데이터를 복제하여 메모리에 올린다. 이후 클라이언트가 그 chunk 를 수정하려고 할 때 master 는 두개의 메타 데이터가 하나의 chunk 를 가리키고 있음을 인지하고 새로운 replica 을 만든다.
- analysis of experimental results



위 표는 2003 년 구글에서 16 개의 chunk server, client, 하나의 master(1.4Ghz cpu, 2G ram, 5400rpm disk)에서 Reads, writes, record appends 를 할 때 network limit 과 aggregate rate 를 비교해본 결과이다. Read 의 경우 물리적인 한계로 인해 실제 측정치의 75%~80%효율을 얻을 수 있음이 드러났다. Write 의 경우 chunk 복제를 디폴트로 3 개로 하여 클라이언트가 많아질수록 네트워크에 부하가 심해져 예측 값과 잘 맞지 않는 것을 알 수 있다. Data append 의 경우 마지막 데이터를 추가하는 chunk 의 네트워크 대역폭에 의해 값이 결정되어 일정하지 않은 모습을 볼 수 있다.

- opinion or improvement point

GFS 설계에는 여러 trade off 관계에서 선택한 특성들이 많았다(ex. High Bandwidth<->low latency, efficient chunk memory usage<->reliability). 시스템에서 중요한 고려사항이 무엇인지에 따라 고려된 사항들이겠지만 improve point 는 늘 이런 trade off 관계에 있는 특성들 양쪽을 최대한 고려할 수 있는 방법을 더욱더 모색하는 것이라고 생각한다.