**GFS 논문정리**

**들어가기 전의 용어 설명**

GFS : Google File system

파일시스템 : 파일시스템이란 파일의 이름을 정하고 저장 및 검색을 위해 논리적으로 어디에 위치 시켜야 하는지에 대한 방법을 구성하는 시스템이다. 저장 매체의 공간과 보관해야할 파일의 수 가 점점 증가함으로써 별도의 파일 관리 시스템이 필요하게 되었다.

파일시스템의 구조는 해당 파일에 대한 메타데이터영역과 데이터 영역이 존재한다. 메타 데이터는 파일 데이터의 데이터가 저장된 영역이다 즉 파일이름 , 파일위치, 파일크기 ,파일형 등의 정보들이 담겨 있음. 데이터 영역은 실제 데이터가 기록된 영역임 실제 데이터들이 담겨 있음

클러스터 : 여러대의 컴퓨터들이 연결되어 하나의 시스템처럼 작동하는 컴퓨터들의 집합

Fault tolerance(무정지 기능) : 시스템을 구성하는 hardware에 일부에서 결함(fault)나 고장(failure)이 발생하여도 이를 수용하며 정상적으로 작동하거나 혹은 부분적으로도 작동 할 수 있게 하는 성질? 이라 할 수 있음 , 고장이 발생하면 그 즉시 백업 hardware가 작동하거나 역할을 대체해 손실 없는 서비스를 제공하는 것이 목적임.

Scalability : 한국말로 확장성, 확장 가능하다는 의미이며 분산 시스템에서 가장 중요한 목적중 하나라고 할 수 있음 , scalability는 세가지 측면이 존재한다고 함 size : 하드웨어 규모에 대한 scalability , Geography : 사용하는 각 서버가 지리적으로 멀리 떨어져 있을 때에도 동일한 퀄리티의 성능을 제공하는 점 , administration : 하나의 분산 시스템을 여러 관리자가 관리 할 수 있게 하는 점

Data storage : 하나의 스토리지 서버에 데이터를 저장하는 것을 의미

Clustered storage : 성능 , 용량 또는 안정성을 높이기 위해 함께 작동하는 둘 이상의 스토리치 서버를 묶어 사용하는 것을 의미

Meta data : 파일 데이터의 데이터가 저장된 영역 , 파일 데이터의 데이터란 파일이름, 파일위치, 파일크기 ,파일 유형등 파일의 대한 정보.

POSIX :  **p**ortable **o**perating **s**ystem **i**nterface 의 약자로 서로 다른 UNIX OS 의 공통 API를 정리하여 이식성이 높은 유닉스 계열 응용 프로그램을 개발하기 위한 목적으로 IEEE가 책정한 애플리케이션 인터페이스 규격임 -> 쉽게 말해 운영체제 종류마다 버전마다 컴파일러가 차이가 있을 수 있는데 컴파일러의 차이가 있더라도 프로그램이 호환이 되도록 표준 api interface?라고 할 수 있음

**Abstract**

구글 파일 시스템은 간단히 말하면 구글에서 만든 확장이 가능한 대규모 분산형 파일 시스템이라 할 수 있음. 구글 파일 시스템은 저렴한 일반적인 컴퓨터를 사용함에도 fault tolerance를 제공하고 합산된 높은 성능을 다수의 clients에게 제공한다.

GFS는 구글 어플리케이션과 기술 환경의 관찰을 통해 설계되었음

이 GFS paper에서는 GFS의 분산 프로그램을 지원하기위한 파일 시스템의 인터페이스 확장기법과 microbenchmark와 real word에 대한 측정에 대해 소개되어 있음

**Introduction**

GFS는 이전의 파일 시스템과 비슷한 목적을 가지고 있지만 이전 시스템들과 새로운 관점들이 추가 됬다.

1. 어느 한 노드의 고장은 예외적인 상황이 아니라 일반적인 상황으로 여긴다. 이 일반적인 상황을 해결하기 위해 constant monitoring , error detection, fault tolerance , automatic recovery 같은 기술이 이 시스템에 내장되어야 했음 ( 고장의 종류는 application bug, os bug, 사람의 잘못, 네트워크, 메모리 접근 실패 등이 존재 함)
2. 파일의 크기가 이전보다 커졌으므로 파일을 나누는 block과 I/O작업의 크기 재검토다 필요해짐
3. 파일이 바뀔 때 기존 파일에 overwrite하지 않고 추가확장됨 한번 작성된 파일은 단순히 읽기만 가능하고 대부분 순차적으로 읽음, 추가확장기능은 아토믹하게 확장시키므로 별도의 동기화를 필요하지 않고도 다수의 클라이언트가 동시에 하나의 파일에 추가할수 있도록 함.
4. Application 과 파일 시스템 API를 같이 설계하여 전체 시스템의 효율 향상

**Design Overview**

1. Assumptions

* 시스템은 고장날 확률이 있는 저렴한 hardware을 통해 구축
* 시스템은 적은 수의 대용량 파일을 저장
* Workload는 주로 두 가지 유형의 읽기임, 대용량 스트리밍 읽기와 작은 랜덤한 읽기로 구성. (대용량 스트리밍 읽기는 각각의 분산된 수백 KB 보통 1MB이상을 읽고 동일한 클라이언트에 연속된 동작은 그 파일의 연장영역에서 읽어옴; 소량의 랜덤 읽기는 임의의 좌표?에서 수 킬로바이트 단위를 읽게됨)
* 파일에 데이터를 추가하는 workload는 file에 append를 하는 순차적 쓰기 가 있음 일반적으로 작업의 크기는 읽기작업과 비슷하며 한번 쓰여지면 다시 수정될 일은 거의 없다.
* 여러 클라이언트가 동시에 한 파일에 append를 할 때 이 것을 효율적으로 구현해야 함 , 큐 또는 다방향 병함(many-way merging)을 사용함. 결국은 한번에 하나씩 append 해야 동기화가 유지 되므로 이때 생기는 overhead는 필수적일수 밖에 없음.
* 빠른 반응속도(low latency)보다는 높은 부하를 견뎌낼수 있는 대역(bandwith)가 더 중요함. 특정 높은 반응속도가 필요한 application이 아닌 이상 엄격한 응답 시간 요구를 가지는 application은 거의 없음.

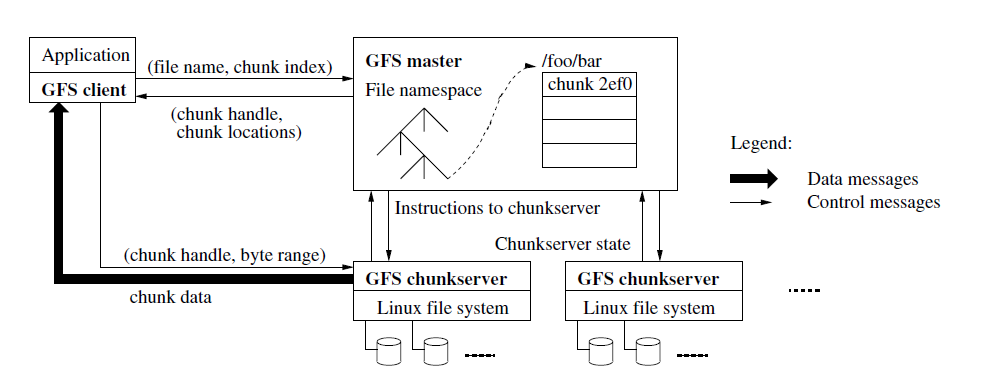
1. Interface

GFS는 POSIX와 같은 표준 API를 가지고 있진 않지만 어느정도 친숙한 파일 시스템 인터페이스를 제공함 스냅샷 기능과 레코드 추가 기능을 지원

스냅샷 : 복사본을 간단히 만들 수 있는 기능

레코드 추가 : 복수의 클라이언트가 추가하려는 각 데이터의 내용을 보장하면서 동시에 모든 클라이언트가 하나의 파일에 그 데이터들을 추가할 수 있도록 해줌.

1. Architecture

* GFS 파일 시스템의 클러스터는 하나의 마스터와 여러 개의 청크서버로 구성됨 **Master**: GFS 전체를 관리하고 통제하는 중앙 서버의 역할

**Chunk Server**: 물리적인 서버, 실제 입출력을 처리

**Client**: 파일 입출력을 요청하는 클라이언트 어플리케이션

파일 시스템에 저장되는 파일들은 고정된 크기의 조각(chunk)로 나누어지고 이 각각의 조각은 마스터 서버로 부터 고유의 64bit의 값을 할당 받는데 이것을 chunk handle이라 하고 이것을 통해 각 조각이 구분되고 이 값은 절대로 변경될 수 없도록 한다. 또한 각 청크는 여러 청크서버에 복사되어 저장된다. 마스터 서버는 모든 파일 시스템의 메타데이터(네임스페이스,접근조절 정보, 파일 조각 매핑, 조각의 현재 위치)를 갖고 있고 시스템 단위의 활동(조각 임대 관리, 연결이 유실된 조각, 가비지 컬렉션, 청크 서버들간의 청크 이동 )들을 관리한다.

마스터 서버는 주기적으로 각각의 청크서버와 맥박신호(heartbeat message)를 교환함으로써 명령을 전달하고 각 청크서버의 상태를 수집한다.

클라이언트는 마스터와 통신하여 메타데이터를 통해 관련 작업을 수행하지만 실제 데이터 관련 통신은 모두 청크서버로 직접 전달된다.

1. Single Master

마스터 서버는 메타데이터만 클라이언트에게 제공하고 파일 제공은 하지 않음. 따라서 마스터가 모든 파일의 읽기와 쓰기 작업에 관여 하는 정도를 최소화하여 병목 현상이 생기지 않도록 함.

가장 단순한 읽기 작업 과정을 살펴보면 , 클라이언트는 지정된 조각 사이즈에 따라 어플리케이션이 배정한 파일의 이름을 분석하고 파일내의 조각 인덱스 오프셋을 계산한다 그리고 나서 마스터서버에게 파일이름과 청크 인덱스를 포함한 요청을 전달하면 마스터는 해당 요청의 반환 값으로 해당 조각핸들과 복제의 위치를 반환하게되고 이 정보를 파일이름과 조각인덱스를 key 값으로 자신의 캐시에 저장한다.

이 후 클라이언트는 반환 값을 통해 복제 중 하나, 대부분 가장 가까운 청크서버에 파일 요청을 보내 파일 데이터에 접근 하게 된다. 받아온 값은 클라이언트의 캐시에 저장해놓기 때문에 동일하고 빈번한 파일 조각에 대한 읽기 작업은 마스터와의 통신이 필요 없게 됨

1. Chunk Size

청크사이즈는 기본적으로 64MB(이 논문 발표 당시)로 설정하고 지연 공간 할당(lazy space allocation)을 사용해서 큰 사이즈의 청크를 사용함으로써 생길 수 있는 internal fragmentation을 최소화 한다.

1. Meta Data

마스터 서버에는 세가지의 메타데이터를 저장한다. 파일과 청크의 네임스페이스, 파일이 청크로 매핑되는 것, 각 청크 복사본의 위치.

1. In-memory data structure

메타데이터가 마스터의 로컬 메모리에 저장되므로 이에 대한 마스터의 반응이 빠르고 전체를 점검하기에 효율적 구조를 가지게 됨. 이 때 시스템에 있는 파일들이 많아지면 마스터 메모리의 capacity 측면이 문제가 될 수 있지만 한 64MB청크마다 64bytes 의 chunk handle를 사용하기 때문에 비용적으로 큰 문제가 되지 않음

1. Chunk Locations

마스터는 어떤 청크의 복제본이 어느 청크서버에 있는지에 대해 영구적 기록 을 가지고 있지 않음. 처음 시작할 때 초기 위치 값을 설정한 뒤로는 heartbeat이라는 신호를 통해 청크서버들을 모니터링함.

1. Operation Log

작업 기록들은 중요한 metadata의 변화를 기록한다. 이 log는 중요하기 때문에 주로 다른 원격 장치에 복사 저장한다. 마스터는 이 log를 이용해 유실되거나 한 파일 시스템을 복구할 수 있다. 또한 이 log를 작성하는 중에 error가 생겼다면 복구코드가 자동으로 불완전하게 종료된 기록들을 무시하여 데이터의 무결성을 유지함.

1. Guarantees by GFS

파일의 네임스페이스 변경은 atomic한 작업이며 마스터에만 의해 관리된다. 그리고 마스터의 operation log가 이런 atomic한 작업의 순서를 기록한다. 그리고 어떠한 파일에 write나 append가 발생하여 파일에 변화가 생겨도 read를 할 때에는 어떤 시점이여도 그 file과 그 file 의 복제 file들은 같은 내용 즉 consistent를 보장해야한다. 이 것을 파일에 변화가 있을 때 파일의 상태를 consistent, defined , inconsistent 상태 등으로 정의해주고 파일 버전 번호를 사용하여 만약 클라이언트의 요청에 의해 업데이트되지 않은 stale(이전 버전) 청크가 제공될 상황이여도 제공되지 않고 garbage collected 가 된다. 이후 마스터가 모든 청크서버와 통신하며 checksumming으로 데이터 결함을 찾아내고 데이터는 믿을만한 복제본으로 복구됨.

**System Interaction**

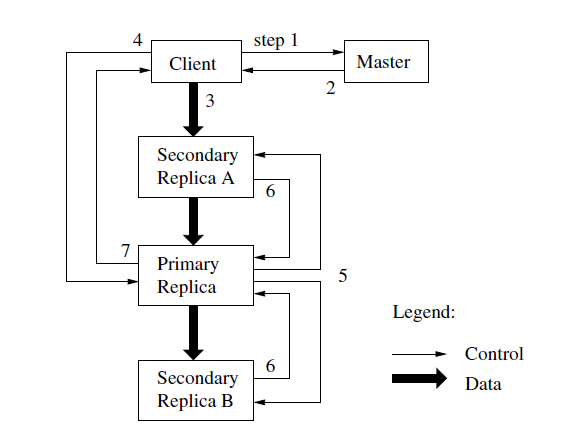
GFS 는 마스터서버의 개입을 최소화하는 목적을 가지고 시스템을 설계함

1. Leases and Mutation Order

Write나 append operation을 통해 파일의 내용이나 metadata가 변경되는 것을 통틀어 mutation이라 함. 각각의 mutation은 같은 종류의 파일의 복사본 전체에서 수행되어야 consistent를 유지 할 수 있다. 그래서 GFS에서는 LEASES라는 기법을 사용함.

leases라는 기법은 마스터서버가 처음에 복사본들중 한 파일을 primary라고 지정하고 primary는 나머지 복사본들의 mutation순서를 정하고 모든 복사본들은 이 순서를 따른다. Master는 primary만 정하기만하고 primary가 mutation 순서를 고르므로 master의 오버헤드를 줄여줄 수 있다.

다음은 leases 기법의 작동 순서사진이다.

step1. 클라이언트가 마스터에게 어떤 청크서버가 lease를 가지고 있는지, 그리고lease가 아닌 다른 복사본들은 어디에 있는지에 대한 요청을 하고 만약 lease가 없으면 master 서버는 복사본들중 하나를 primary로 지정한다.

step2. 마스터는 primary와 다른 복사본들의 location을 클라이언트에게 리턴해준다.

Step3. 클라이언트는 모든 복사본에 변경될 data를 전송한다.

Step4. 모든 복제본들에게 data가 전송되면 쓰기 요청을 primary에게 전달

Step5. 프라이머리는 지정한 순서대로 모든 복사본들에게 쓰기 요청 전달

Step6. 각 복사본들이 mutation이 끝나면 primary에게 완료 메시지를 전달

Step7. 모든 복사본들이 mutation이 끝났다는 메시지를 클라이언트에게 전달

1. Data flow

위 그림에서 볼 수 있듯이 Data는 client에게 가장 가까운 서버부터 linear 하게 이동이 된다. 그 이유로는 각 머신의 네트워크 대역폭을 최대한 활용하기 위해서 라고 한다. 또한 파이프라이닝을 함으로써 latency를 최대한으로 줄인다.

1. Atomic record appends

이전의 write는 클라이언트가 data가 쓰여질 offset(영역)을 정하여 여러 클라이언트들이 동일한 파일에 동시에 쓰여지는 것이 허용되지 않았다(serializable 하지 않았다고 함). 하지만 GFS의 record append에서 클라이언트는 data만 정하고 GFS가 그 data를 어느 위치에 추가할지 정해서 기록하고 그 data의 offset을 클라이언트에게 알려준다.

1. Snapshot

스냅샷은 파일이나 디렉토리의 복사본을 만드는 operation이다. 단순히 window에서 파일 복사와 동일한 기능임. 마스터가 스냅샷 요청을 받게 되면 제일 먼저 그 파일의 청크에 대한 lease 작업을 철회 시킨다. 이로 인해 snapshot 전에 해당 파일의 청크에 추가적인 write 작업이 이루어질 수 없도록 한다. Lease가 철회된 후 마스터 서버는 메타데이터를 복제하여 메모리 마스터의 local disk에 저장하고 original chunk를 가르키도록 한다.

**Master operation**

마스터 서버는 모든 네임스페이스 작업, 시스템 전체의 청크 복제본들을 관리(배치를 결정, 새로운 청크를 생성, 다양한 시스템 차원의 작업을 복제하고 조직)한다.

1. Namespace Management and Locking

GFS는 이전의 파일 시스템과 다르게 per-directory 자료구조(디렉토리 안에 존재하는 모든 파일을 나열하는)를 갖지 않음. 대신 GFS는 경로 이름에 대해 look up table 매핑으로 네임스페이스를 논리적으로 나타냄. 또한 파일의 namespace에 lock 기능을 사용해 작업들이 겹치지 않고 delay를 줄이도록 한다.

1. Replica Placement

GFS 클러스터의 청크서버는 굉장히 많이 분산되어 있다. 일반적으로 수백개의 서버가 여러 곳에 분산되어 있다. 파일의 청크 분산 배치 정책은 data reliability와 availability 그리고 network bandwidth utilization을 극대화 하기위한 목적을 가지고 있음. 이 목표를 얻기 위해 파일의 chunk 파일들은 node 기준이 아니라 node의 rack 기준으로 분산 저장된다.

1. Creation, Re-replication, Rebalancing

청크 본제본들은 세가지 이유로 인해 생성됨 : chunk creation, re replication 그리고 rebalancing. 마스터서버는 청크를 생성할 때 복제본을 어디다 둘지 다음과 같은 조건을 두고 고려한다. 1. 디스,크 사용량이 적은 청크서버에 새 복제본을 저장. 2. 각 청크마다 최근 복제본 생성 수 제한. 3. rack들에 걸쳐서 복제본을 뿌림, 사용 가능한 복제본의 수가 특정 수 이하로 줄어들면 마스터는 re-replicate함. 또한 마스터는 주기적으로 복제본을 rebalancing 한다. 복제본이 어떻게 퍼져있는지를 보고 복제본을 더 나은 disk로 옴긴다

1. Garbage Collection

파일이 삭제된후 GFS는 그 physical storage를 즉시 수거하진 않음. 천천히 수고함. 이렇게 하는 것이 시스템을 훨씬 단순하고 신뢰할 수 있게 한다는 것을 발견했음.

1. 어플리케이션이 파일을 지우면 마스터는 삭제되었다는 log를 남기지만 파일은 삭제된 시간이 포함된 hidden name으로 renamed된다. 이후 마스터는 정기적으로 파일 시스템의 namespace를 스캔 하는대 이때 3일 이상 지난 hidden name을 가진 file이 있으면 local disk에서 지운다. 또한 master는 namespace가 없는 chunk 파일 aka orphaned chunks를 찾아 meta data를 삭제하고 heartbeat 신호를 통해 chunk sever에게 meta data가 없는 청크를 알려주고 그러면 청크서버가 지운다.
2. Stale Replica Dectection

만약 청크서버에서 청크에 대해 mutation이 실패하고 청크서버가 다운 된다면 그 청크서버에 있는 청크 복제본이 stale상태가 되는 상황이 발생할 수 있는데 이상황을 해결하기 위해서 master에서는 최신의 복제품을 구별하기 위해 version number을 생성하고 갖고 있어서 그 청크 본제본이 최신버전인지 아님 stale인지 구별할 수 있게 된다. 이 버전 번호는 lease가 실행 될 때마다 증가시키는 구조이며 마스터는 garbage collection 떄 stale 상태인 복사본을 제거한다.

**Fault Tolerance and Diagnosis**

Fault 에 관해 다루는 것은 시스템을 설계하는 데에 가장 큰 어려움 중 하나임. 앞에서도 말했듯이 fault는 예외적인 문제보다 일반적인 문제로 컴퓨터와 컴퓨터의 디스크를 완전히 신뢰할 수 없다는 문제에서 생성된다. 따라서 이러한 일반적인 문제를 진단하고 해결하는 방법이 필요함

1. High Availability
2. Fast recovery

마스터와 청크 서버는 어떻게 종료되든간에 그들의 상태를 그대로 복원 가능하도록 설계되었음. 일반적으로 정상적인 종료와 비정상인 종료를 구별하지 않음 단지 프로세스를 중지 시키는 행위로 서버를 종료할 수 있음.

1. Chunk replication

앞에서 이야기한 대로 청크는 랙 레벨에서 여러 번 복제되어 분산 저장됨

1. Master Replication

안정성을 위해 마스터의 상태가 복제됨. Master sever의 operation log와 checkpoint가 다수의 머신에 복제되어 저장됨. 이로 인해 master server가 다운 됬을 때 이 것을 사용해 shadow master가 master server의 기능을 어느정도 해줄 수 있음.

1. Data Integrity

각 청크 서버는 저장된 데이터의 변질을 탐색하기 위해 checksum을 사용한다. 각 청크는 32bit의 체크섬을 가지고 다른 메타데이터처럼 청크의 checksum data도 master server의 in memory에 저장되어 관리됨. 읽기 작업이 오면 청크서버는 checksum을 확인하고 여기서 오류가 발생하면 청크서버는 클라이언트에게 error를 리턴하고 마스터에게 알린다. 그러면 클라이언트는 다른 복제본에 다시 요청을 할 것이고 마스터는 다른 복제본을 복제한다. 그 뒤 master는 error 가 발생하게된 청크를 지우라고 청크서버에게 지시함으로써 data integrity를 지킨다.

**Conclusion**

GFS의 가장 중요한 점은 위에서도 말했듯이 이전의 파일 시스템과 달리 component failure 을 예외로 보지않고 일반적인 현상으로 인지하고 이것을 해결 하기위한 다양한 기술(데이터 복제 , 자동 복구 기능 등)들을 내장시켜 fault tolerance를 제공하고 동시에 비용적인 문제도 cover하여 대용량 파일들에 대한 적합한 파일 시스템을 구성했다.