Apache Hadoop

> https://ecycle.tistory.com/6
>
> https://www.youtube.com/watch?v=Jx9rjPTWYPQ&list=PL9mhQYIlKEheGLT1V_PEby_I9pOXr1o3r&index=5

1. 하둡이란?

우리는 데이터 홍수의 시대, 하둡은 비정형 데이터를 포함한 빅데이터를 다루기 위한 가장 적절한 플랫폼이다.

구글 파일 시스템 -> 하둡 분산 파일 시스템

구글 맵리듀스 -> 하둡 맵리듀스

DBMS, RDBMS도 CPU, Memory, Disk로 구성된 시스템이다. Disk I/O가 급증하면 CPU자원에 병목현사잉 일어나 성능이 급격하게 떨어질 수 있다. 이를 해결하기 위해서 분산 저장 처리 기술을 가진 HBase가 나왔다.

Linux에서 top 명령에서 체크할 수 있다.

process 처리에서 Disk I/O가 700ms초, CPU 처리가 300ms라고 하면 이 프로세스의 wait 타임은 (700/1000*100)=70%이다.

CPU코어수를 n개라고 했을때 wait이 (1/n)*100보다 크면 병목현상이라고 판단한다.

- 하둡은 JAVA로 개발되어있다. (꼭 자바만 지원은 아님, 단지 플랫폼이 자바로 구현되어있다는것)
- 비정형데이터, 반정형데이터는 SQL 질의처리로 데이터 분석이 어렵다. 그래서 맵리듀스로 직접 프로그래밍하는 경우도 생긴다.

2. 하둡 생태계

- HBase : 분산 데이터베이스(NoSQL류 : 카산드라, MongoDB)
- Pig : script language(데이터셋을 다루기 위한 스크립트 언어)
- Zookeeper : 하둡 생태계 플랫폼의 Coordination
- Mahout : 머신러닝 알고리즘을 분산 데이터로 처리할 수 있도록 구현되어있는 것
- Hive : SQL로 하둡의 데이터를 다루는 엔진 (비슷한걸로 Spark, Pig도 있음)
- Sqoop : 여전히 관계형 데이터베이스가 많이 활용된다. 관계형 데이터베이스와 하둡이 데이터를 주고받을때 쉽게 해주는 프레임워크 (일반적으로 많이 사용됨)

이렇게 다양한 언어와 방법으로 하둡에 있는 데이터를 다룬다. 어쨌든 하둡 데이터를 공유하는데에 있어서 기본적인 스키마 정보, 메타 정보를 관리해야하는데 이것을 관리하는 것을 HCatalog이다.

<진화 하둡 생태계>

- 하둡 : 데이터 처리, 분산 자원 관리, 파일 저장, MapReduce
- Hive, HCatalog : 메타데이터 저장
- HBase : 분산 데이터베이스
- Zookeeper : Coordination
- Flume : log, event 관리
- Storm : Stream
- Sqoop : RDBMS

3. 분산처리 구조

3-1. Master-Slave 서버 구조

- slave를 여러개 두면서 확장해나간다.
- master가 부하가 걸리지 않게 하는 것이 중요하다. (master 안정성이 진짜 중요!!!)
- master-client : 데이터 정보를 주고받는 역할이지 데이터를 직접 마스터가 처리해서 보내는것은 아니다. (데이터 위치 정보, 즉, slave정보를 알려줌)
- slave-client : 데이터를 처리하고 주고받는 일을 함.

3-2. Scale Up보다는 Scale Out

Scale Up은 하드웨어, 메모리 등의 성능을 높이는 것이다. 하지만 이것은 이제 더이상 한계가 있다고 본다. Scale Out은 시스템을 여러대를 두어 확장하여 처리 효율을 높인다.

3-3. 분산과 병렬의 차이

병렬처리는 CPU를 여러대 두어 동시에 처리한다에 초점이 맞춰져 있으면 분산처리는 데이터를 여러대에 분산하여 저장하여 처리한다에 초점이 맞춰져 있다.

4. 하둡 특성

- 리눅스 기반 범용 서버들을 하나의 클러스터로 사용
- 마스터-슬레이브 구조
- 파일을 블록단위로 저장
- 블록 데이터의 복제본 유지로 인한 안정성

4-1. 하둡 분산파일시스템

- Name Node : 마스터 서버 역할
- Data Node+Task Tracker : 슬레이브 서버 역할
- Job Tracker : 분산파일시스템에 있는 데이터 Job을 처리(어플리케이션 요청)하기 위한 마스터 서버 역할

- 하둡 블록이란?

- 하나의 파일을 여러개의 블록으로 저장
- 보통 64MB, 128MB로 나누어 저장 (버전 업그레이드 되면서 128MB로 저장)
- 블록의 크기는 바꿀 수 있다. default값이 128MB라는것.
- 블록이 큰 이유는 탐색 비용을 최소화 할 수 있다.
- 블록이 크면 하드디스크에서 블록의 시작점을 탐색하는데 걸리는 시간을 줄일 수 있다. 네트워크를 통해 데이터를 전송하는데 많은 시간 할당이 가능하다.

<장점>

- 디스크 사이즈보다 더 큰 파일을 저장할 수 있다.
- 메타정보를 저장할 때 사이즈가 고정되어 있어서 편리하다.

<블록의 지역성>

데이터노드에 있는 TaskTracker은 자신이 가지고 있는 로컬 데이터를 처리한다. 만약 해당 데이터노드에 있는 TaskTracker가 매우 busy한 상태면 같은 Rack에 있는 데이터노드에게 데이터를 읽어와 처리할 수 있도록 하는 locality 보장을 해준다.

- 하둡 전체적으로 보았을 때 분산되어 그 데이터 노드에서 각기 처리하기때문에(Locality) 굉장히 빠르게 처리될 수 있다.
-

- 하둡에 데이터 저장 단계

1. 클라이언트가 네임노드에 저장 요청
2. 네임노드는 파일을 128MB 블록으로 자른다.
3. 블록마다 3개씩 복사하여 데이터노드에 저장한다.

- HDFS 장애 대응

1. 데이터노드는 주기적으로 네임노드에게 Heartbeat을 보낸다.
2. 데이터노드는 주기적으로 네임노드에게 블록리포트를 보낸다.
3. 데이터노드가 장애가 생기면 네임노드는 해당 데이터노드가 가진 데이터들이 어느 데이터노드들에 복제되어있는지 알고있다.
4. 복제된 다른 데이터노드가 다른 데이터노드에 블록하나 더 복제할 수 있도록 명령한다. (항상 3copy 유지)

-> 만약 네트워크 장애의 문제였다면 ?

네트워크가 복구가 되면 블록은 4개가 되어버린다. 이것도 네임노드가 알아서 관리.

- 하둡 데이터 유실이 나는가?

같은 데이터를 보유하고 있는 3대의 데이터노드가 장애가 날경우에 데이터 유실이 난다. (거의 희박!!!!!)

1. 파일 시스템의 클라이언트가 쓰기 동작을 하면 네임노드의 에디트 로그에 기록이 된다.
2. 네임노드는 파일시스템의 메타데이터를 인메모리로 관리하는데 에디트 로그를 먼저 변경한 후 메모리상의 메타데이터도 변경한다.
3. 클라이언트의 읽기 요청에는 인메모리 데이터만 사용된다.

- 하둡 버전 1.x / 2.x / 3.x

하둡 1.x에서 2.x로 업그레이드 되었을때 HDFS입장에서 가장 크게 변한것은 마스터 서버 이중화이다. 마스터 서버가 장애가 났을 경우 전체 플랫폼 장애라고 볼 수 있다. 그렇기 때문에 2.x버전부터는 마스터 서버 이중화로 마스터 서버의 안정성을 더우 높였다.

하둡 2.x에서 3.x로 갈때는 HDFS 입장에서는 3copy가 아닌 2copy로 줄였다는 것이다.

<2.x의 가장 중요한 것은 네임노드의 안정성!!!>

세컨더리 네임노드가 추가가 되었다. 에디터로그는 네임노드에서 매우 중요한 부분이다. 에디터로그에 어느정도 데이터가 쌓이면 fsimage로 저장한다. fsimage는 메타데이터의 체크포인트 스냅샷이다. 이과정은 네임노드가 세컨더리 네임노드에 보내면 세컨더리 네임노드가 수행한다.

-> 세컨더리 네임노드가 장애가 발생하면 에디터로그만 무한히 커질뿐 큰문제는 발생하지 않는다. 하지만 네임노드를 재시작해야하는 경우 fsimage로 우선적으로 메모리에 반영하고 에디터로그를 확인하는데 이때 에디터로그가 너무커서 out of exception이 발생하는 장애가 발생할 수 있다.

스탠바이 네임노드는 주 네임노드가 장애가 발생할시 대신 역할을 해주는 노드이다. 스탠바이 네임노드와 주 네임노드는 에디터로그를 공유 스토리지(저널노드)에서 공유하고있다.

엑티브 네임노드(주 네임노드)가 장애가 발생한것을 주키퍼라는 것이 알고있는데 만약 엑티브 네임노드가 주키퍼랑 통신이 안된다면 주키퍼는 장애로 알고서는 스탠바이 네임노드에게 엑티브 네임노드 역할을 명령한다. 그러면 엑티브 네임노드와 스탠바이 네임노드가 에디터로그를 동시에 쉐어하면서 변경을 일으키는 크랙이 발생하는데 이것을 SplitBrain이라고 한다. 그래서 공유 스토리지에 에디터로그를 공유하는 방법보다는 QJM(Quorum Journal Manager)라고 엑티브 네임노드 내부에 구현되어서 관리하는 방법을 선택한다.

HDFS Federation이란?

하나의 네임노드가 관리하는 메타정보들이 너무 쌓여서 네임노드도 여러대로 두어 관리해야하는데 이걸 HDFS Federation이라고 한다. 2.x이상 버전에서는 데이터노드를 묶어서 네임노드가 관리하는 block pools 기법을 사용한다.

- 데이터노드 스펙

요즘에는 데이터 노드에서 메모리 영역에서 연산을 많이 하는 편이기 때문에 메모리 영역을 크게 두는 편이다. ETL 처리(Extract, Transform, Load / 추출, 변환, 로드)와

분석처리 등을 Spark엔진에서 많이 한다. Spark엔진은 메모리 베이스에서 처리를 한다.

데이터노드 하나에 128GB, 256GB정도의 메모리를 사용한다. CPU코어는 안좋은 것을 쓰는 경우가 많다. (비싸지 않는것을 쓰는편) 코어는 일반적으로 16개 사용한다면 Hyper Threading기술로 거의 2배의 가까운 성능을 낸다. 디스크를 너무 큰 디스크를 사용하기보다는 작은 디스크를 6개~12개정도 사용한다. (2TB정도의 크기 여러대)

** Hyper Threading이란 코어에 둘 이상의 스레드를 실행할 수 있는 것이다.

- 블록 캐싱

데이터 노드에 저장된 데이터 중 자주 읽는 블록은 블록 캐시라는 데이터 노드의 메모리에 명시적으로 캐싱할 수 있다.

파일 단위로 캐싱할 수도 있어서 조인에 사용되는 데이터들을 등록하여 읽기 성능을 높일 수 있다.

- Safe mode : Missing Block

HDFS의 세이프모드는 하둡의 missing block이 어느정도 많아지면 발생하는 모드이다. missing block은 클라이언트가 요청하는 자료가 없을 때 (3copy가 기본인데... 한개라도 없을때!)를 말한다. 세이프모드가 되면 데이터의 변경이나 읽기 쓰기가 모두 중단된다. 클러스터 재구동했을때도 세이프모드이다.

- 휴지통 기능

하둡은 데이터 삭제시 trash 디렉토리로 옮겨진다. trash 디렉토리는 기한을 설정한 후 일정 기간이 지나면 데이터를 영구적으로 삭제한다.

- 커맨드

- report : 네임노드의 관리 상태를 운영자가 확인
- balancers : 하둡을 운영하다보면 스케일아웃하였을때 서로 다른 스펙의 데이터노드로 클러스터가 구성될 수 있다. 이 경우 디스크 크기가 다른 수있고 전체 데이터의 밸런싱이 되지 않는 문제가 발생할 수 있다. 이 경우 네임노드에서 데이터 적재량이 적은 노드를 우선적으로 선정하여 데이터 블록을 저장한다. 운영중인 클러스터에서 밸런서를 조정하는 것은 매우 조심스럽다. job들에 최소 영향을 주면서 조절해야한다. 데이터노드간의 대역폭을 조정하던지 쓰레드 운영 조정을 하던지 ..

4-2. 하둡 맵리듀스

HDFS에 분산 저장되어 있는 데이터를 병렬로 처리하여 취합하는 역할

- 기본 알고리즘

Map : (key1, value1) -> (key2, value2)

Reduce : (key2, list of value2) -> (key3, value3)

- 장/단점

- 단순하고 사용이 편리
- 데이터 처리 유연성
- 작은 데이터 저장/처리하기에는 비효율적(노드들간의 통신 오버헤드)
- SQL질의보다는 복잡하긴함

- 맵리듀스1.x 구동절차

1. 클라이언트가 Job을 Jop Tracker에 제출
2. Job Tracker가 HDFS에 저장하여 Job을 공유한다.
3. Task Tracker가 참조하여 역할을 분담하여 Job 수행
4. Task Tracker가 Map task를 수행하면 SequenceFile(key2, value2)을 binary파일로 저장함
5. 같은 key값끼리 같은 데이터 노드로 전송되어 Reduce task가 수행된다.

- RecordReader : 처리 데이터를 하나씩 읽어서 key-value로 형태로 변환하여 mapper에 전달
- Combiner : 데이터 전송량을 줄이기 위해서 같은 mapper 쪽 서버에서 작은 reducer를 해준다. (물리적으로 같은 키를 가진 데이터를 동일한 reducer로 보내기 위함)
- Partitioner : key값을 reducer 개수만큼 해싱하여 같은 key값끼리는 동일한 reducer로 보낸다.
- Shuffling : 다른 맵퍼에서 날라온 같은 key 데이터끼리 취합하여 reducer에게 전달
- reducer : 같은 key 데이터끼리 원하는 작업을 수행하여 최종 결과 output

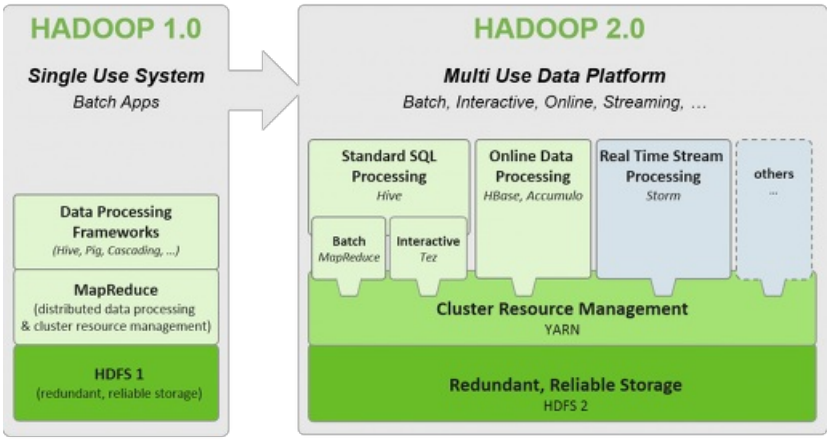
- 맵리듀스 장애대응

Task Tracker는 Job Tracker에게 Heartbet을 보내며 task의 실행 상태를 보고한다. 클라이언트는 Job Tracker를 폴링하여 최신 정보를 갱신한다.

Task Tracker가 장애가 나면 Job Tracker가 다른 Task Tracker에게 명령 수행

- 하둡 버전 1.x / 2.x

하둡 2.x가 되면서 YARN이라는 것이 생겼다. YARN은 리소스 매니저 역할을 한다. YARN이 생기면서 MapReduce말고도 다른 데이터 처리(ex. MPI)를 할 수 있다.



Interact with all data in multiple ways simultaneously

- 맵리듀스2.x 구동절차

1. 클라이언트가 리소스 매니저한테 수행 요청
2. 클라이언트는 input data를 HDFS에 저장한다.
3. 클라이언트는 리소스매니저한테 job을 제출한다.
4. 리소스매니저는 노드매니저에게 job수행 명령을 내린다.
5. 노드매니저는 어플리케이션매니저에게 job을 할당한다.
6. 어플리케이션매니저는 HDFS에서 input파일을 가져와 task 수행 노드매니저에게 수행 명령을 내린다. map task와 reduce task를 생성하고 수행방법을 결정한다. (어플리케이션 매니저는 리소스 매니저에게 컨테이너 요청을 한다)
7. task는 데이터 locality를 고려하여 할당된다. 필요한 리소스가 있다면 통신을 통해 가져와서 수행한다.
8. map task와 reduce task가 수행되면 결과를 HDFS에 저장한다.

- 맵리듀스 최적화

1. 맵태스크 메모리 버퍼 크기 크게 하기

   맵 작업이 끝난 뒤에 리듀스로 병합하는 과정에서 중간단계의 임시 결과 파일 개수를 줄이는 것으로 최적화를 할 수 있다. 리듀서의 병합시간, 네트워크 전송 시간이 단축된다.

   맵 태스크는 메모리 버퍼를 생성하고 맵의 출력 데이터를 버퍼에 기록한다. 버퍼의 크기는 io.sort.mb속성에 정의되며 기본값은 100MB이다. 하지만 이것을 조정할 수 있다.

   메모리 버퍼의 크기를 키우면 맵 디스크에 저장되는 파일(reduce task로 보내질 파일) 수를 줄일 수 있다.

2. 콤바이너 클래스 사용

   맵 출력 데이터를 네트워크 통해서 리듀스에 전달되기 전에 미니 reducer를 실행한다.

5. 하둡 활용

- 장점

  대용량 데이터 처리에 매우 적합하다.

- 단점
  - 작은 데이터 처리 단위로 관리한다면 네임노드에 부하가 올 수 있다.
  - 맵리듀스는 프로그래밍 레벨의 개발이 필요하다. (SQL은 쿼리 엔진이 매우 단순)
  - SQL처럼 사용하기 위해서 Hive 구현
  - 최근 Spark 엔진은 하둡에 저장되어있는 데이터 처리를 메모리단에서 하여 성능을 더욱 높임. 대신 메모리단이기 때문에 매우 큰 데이터는 Hive가 더 좋음