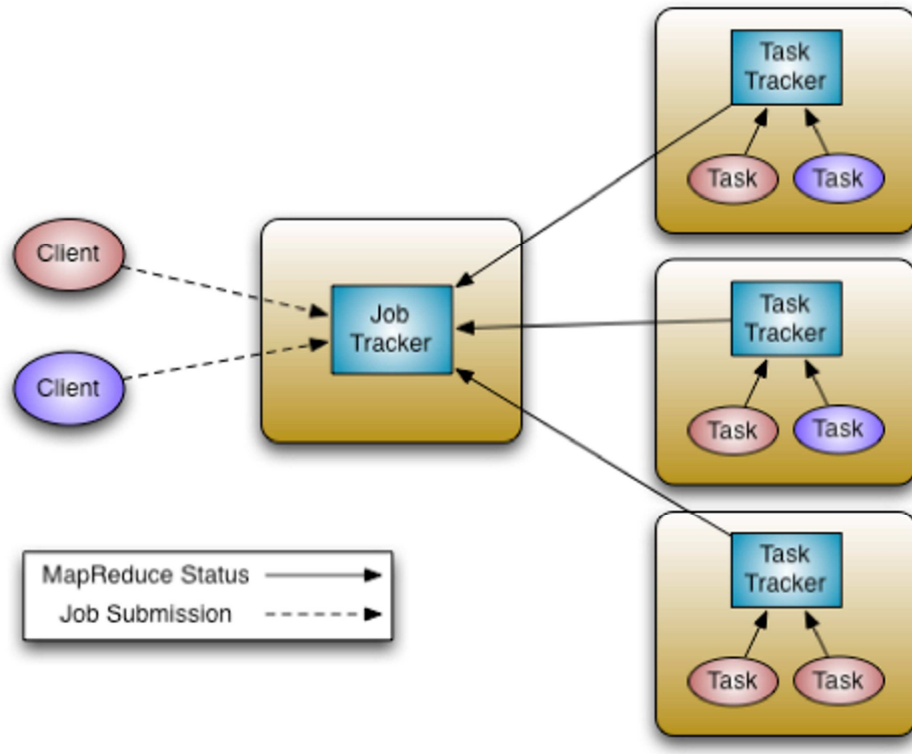


05. MapReduce

2020년 7월 14일 화요일 오후 9:30

1. 맵리듀스 아키텍처



1) Client

- a. 사용자가 실행한 맵리듀스 프로그램, 하둡에서 제공하는 맵리듀스 API

2) Job Tracker

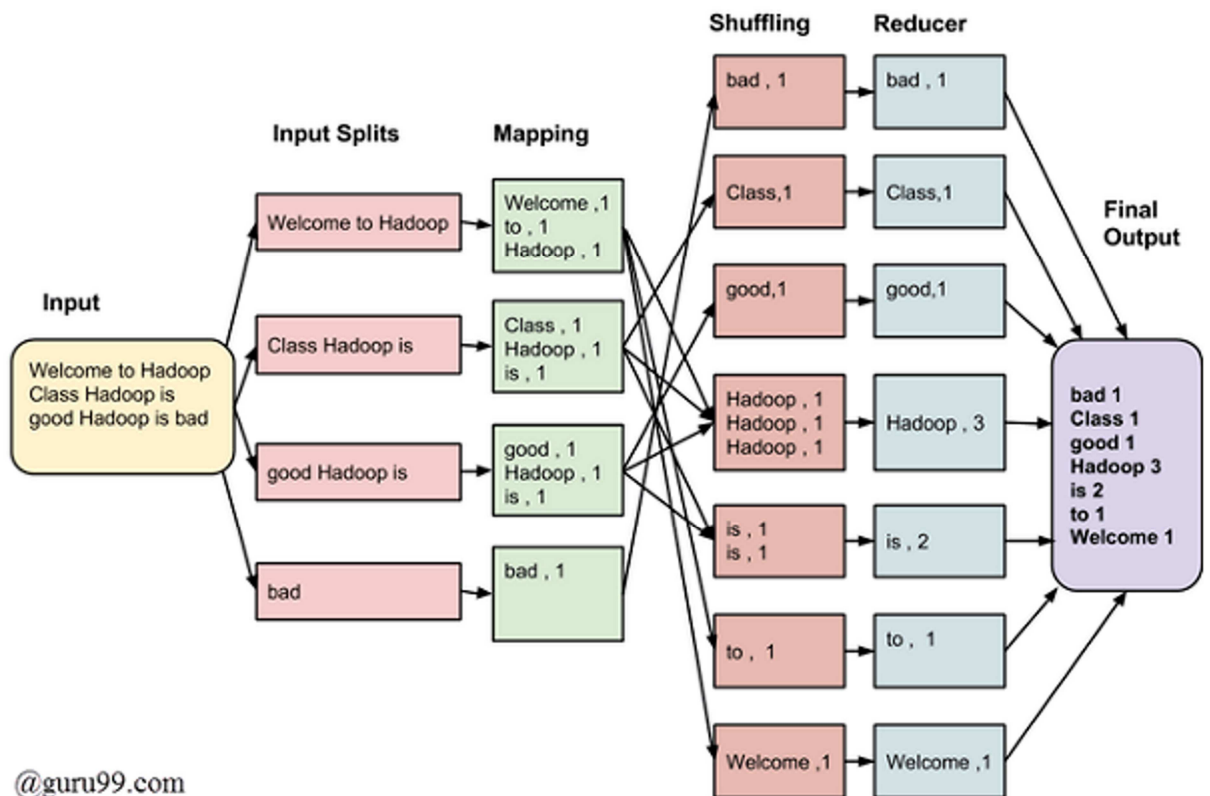
- a. 클라이언트가 하둡으로 요청하는 하나의 작업 단위
- b. Job Tracker는 하둡 클러스터에 등록된 전체 Job의 스케줄을 관리, 모니터
- c. 전체 하둡 클러스터에 하나의 Job Tracker가 실행
- d. 일반적으로 네임노드 서버에서 실행되나, 반드시 그럴 필요는 없음
- e. 사용자가 새로운 Job을 요청하면 Job Tracker는 Job을 처리하기 위해 몇 개의 맵과 리듀스를 실행할 지 계산
- f. 계산된 맵과 리듀스를 어떤 Task Tracker에게 실행할지 결정, 해당 Task Tracker에 Job을 할당

- g. Job Tracker와 Task Tracker는 Heartbeat 메서드로 네트워크 통신을 하며 Task Tracker의 상태와 작업실행 정보를 주고 받음
- h. Task Tracker에 장애가 발생 시 Job Tracker는 다른 대기중인 Task Tracker를 찾아 태스크를 재실행

3) Task Tracker

- a. Task Tracker는 사용자가 설정한 맵리듀스 프로그램을 실행(하둡의 데이터노드에서 실행되는 데몬)
- b. Task Tracker는 Job Tracker의 작업을 요청받고, Job Tracker가 맵과 리듀스 개수만큼 map task와 reduce task를 생성
- c. Map task와 reduce task가 생성되면 새로운 JVM을 구동해 맵과 리듀스 태스크를 실행
- d. 이 때 태스크를 실행하기 위한 JVM은 재사용할 수 있게 설정할 수 있음
- e. 서버가 부족해서 하나의 데이터노드를 구성했다라도 여러 개의 JVM을 실행해 데이터를 동시에 분석함

2. 데이터 처리



1) InputSplit

- HDFS에서 입력데이터를 읽어들이
- 파일이 하나라도 HDFS에서 설정한 블록 크기 단위로 split을 생성하고 split 당 하나의 mapTask를 생성

- ex) HDFS 블록 크기가 128MB 이며, 읽어들일 파일이 300MB면 split은 3개 생성됨 (mapTask도 3개 생성됨)
- MapTask는 레코드 단위로 데이터를 읽어 사용자가 설정한 Mapper의 map() 함수를 수행
- map() 함수를 거치면 결과는 (key, value) 형태로 저장
- 이때 생성되는 결과는 MapTask가 실행된 서버 내부에 저장되며, job이 완료되면 삭제

2) Shuffle and Sorting

- MapTask의 결과를 ReduceTask로 전달하기 위해 정렬하는 단계
- ReduceTask의 개수만큼 파티션을 만들고 파티셔너는 MapTask의 결과를 각 partition으로 보냄
- MapTask 결과의 key에서 hash값을 구하고 hash에 따라 어떤 partition으로 이동될지가 결정
- shuffle 단계가 완료되면 각 파티션에는 동일한 key를 갖는 값들이 모이게 되고, 각 key들로 sorting됨

3) Reduce

- key별로 정렬된 결과를 마지막 형태로 가공하여 hdfs에 저장
- 이때 ReduceTask는 정렬된 결과를 레코드 단위로 읽어 사용자가 정의한 reduce 함수에 전달
- reduce함수로 처리된 결과가 마지막으로 hdfs에 part-xxxxx라는 파일로 저장
- xxxxx는 00000 부터 1씩 증가하며, 파티션 번호를 나타냄
- 만약 블록단위가 128MB 인 경우 그 이하 크기인 파일을 하둡으로 돌린다면 part-00000 하나만 생성

3. 데이터 타입

- 맵리듀스는 입출력을 모두 key, value 형태로 사용하며, 이런 데이터들은 네트워크를 통해 전송됨
- 따라서 입출력 데이터들은 이런 맵 형태 및 serialize 가능한 형태로 만들어져야 하며, 이를 위해 WritableComparable interface를 구현해야 함
- WritableComparable interface는 이름처럼 Writable과 Comparable interface를 상속받고 있음
- 따라서 custom 데이터 타입을 사용하기 위해서 WritableComparable를 상속받아 구현한다면, 직렬화와 역직렬화(write(), readFields()) 함수와 compareTo() 함수를 오버라이드 해야 함

1) 하둡에서 기본적으로 제공하는 데이터 타입

- BooleanWritable
- ByteWritable
- DoubleWritable
- FloatWritable
- IntWritable
- LongWritable
- TextWrapper (UTF8 형태의 문자열)

- NullWritable (데이터 값이 필요 없을때)

2) InputFormat

- InputSplit을 통하여 MapTask에 데이터가 입력될때 데이터의 타입을 정의
- job에 따로 정의하지 않을 경우 기본값은 TextInputFormat

- TextInputFormat: \n 기준으로 읽음 key = 라인번호 (LongWritable) | value = 라인내용 (Text)
- KeyValueTextInputFormat: 키값을 라인번호가 아닌 다른 값을 지정 할 때 사용
- NLineInputFormat: MapTask가 입력받을 텍스트 파일의 라인수를 제한 할 때 사용
- DelegatingInputFormat: 서로 다른 입력 포맷이 필요할때 각 경로에 작업을 위임 할 때 사용
- CombineFileInputFormat: 여러개의 파일을 묶어서 split 시킬때 사용
- SequenceFileInputFormat: SequenceFile을 입력데이터로 쓸때 사용
- SequenceFileAsBinaryInputFormat: SequenceFile의 키값을 임의의 바이너리 객체로 변환해서 사용
- SequenceFileAsTextInputFormat: SequenceFile의 키값을 Text 객체로 변환해서 사용

3) OutputFormat

- ReduceTask에서 reduce가 끝나고 출력할 데이터 포맷을 나타냄
- job에서 setter를 이용하여 값을 설정하나, 미설정시 TextOutputFormat 을 기본값을 사용
- 모든 데이터 타입은 최상위 클래스인 OutputFormat과 이를 상속받은 FileOutputFormat 을 상속받아 구현되어 있음

- TextOutputFormat: 텍스트 파일에 레코드를 출력 key, value는 탭으로 구분함
- SequenceFileOutputFormat: SequenceFile을 출력 할 때 사용
- SequenceFileByBinaryOutputFormat: SequenceFile에 key, value를 binary 값으로 씀
- FilterOutputFormat OutputFormat의 클래스를 편하게 사용할 수 있는 메서드 제공
- LazyOutputFormat: FileOutputFormat를 상속받은 다른 타입들을 출력내용이 없더라도 출력 파일(part-xxxxx)를 생성하나, 레코드가 파티션으로 보내질때만 출력 파일을 생성함
- NullOutputFormat: 출력파일이 없을때 사용

4) Sequence file

- 데이터를 키와 값 형태로 저장하는 바이너리 파일
- Writer, Reader, Sorter를 제공하며, Writer를 이용하여 무압축, 레코드 압축, 블록 압축을 할수 있음
- 다양한 압축 코덱을 이용할 수 있으며, 압축이 되더라도 split이 가능 (** Text의 파일의 경우 압축 코덱에 따라 split이 되지 않을수 도 있음)