맵 리듀스

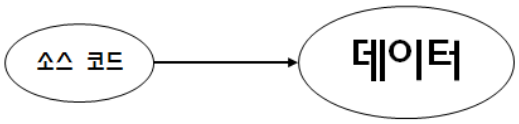
맵리듀스는 큰 규모의 클러스터에 분산되어 있는 대용량 데이터 셋을 처리하기 위한 프로그래밍 기법이다.
하둡의 맵리듀스는 구글의 맵리듀스에서 유래했다.

맵리듀스 아키텍처

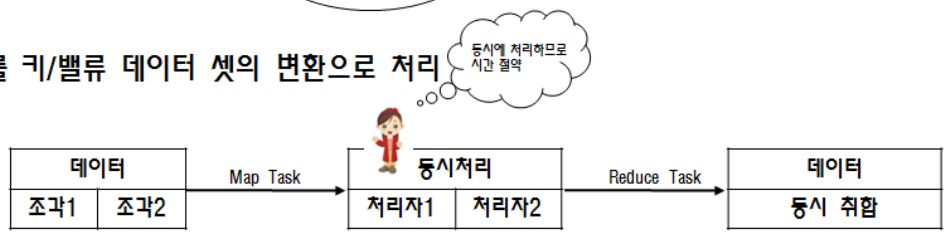
맵리듀스는 큰 규모의 클러스터에 저장되어 있는 데이터를 신뢰성을 보장하며, 병렬로 처리할 수 있게 해준다.
기본적인 패러다임은 Map과 Reduce의 2단계로 나뉘며, 주로 키와 값 쌍에 대한 데이터를 다룬다.

MapReduce 프레임워크의 특징

- 데이터가 있는 서버로 코드를 전송



- 데이터를 키/밸류 데이터 셋의 변환으로 처리



- Share Nothing 아키텍처
  - Task 간의 서로 의존이 없으므로 병렬 처리에 적합하다.
- 오프 라인 배치에 적합
  - 오프라인 배치에 적합
  - RealTime 배치에는 부적합

MapReduce 적합/부적합 분야

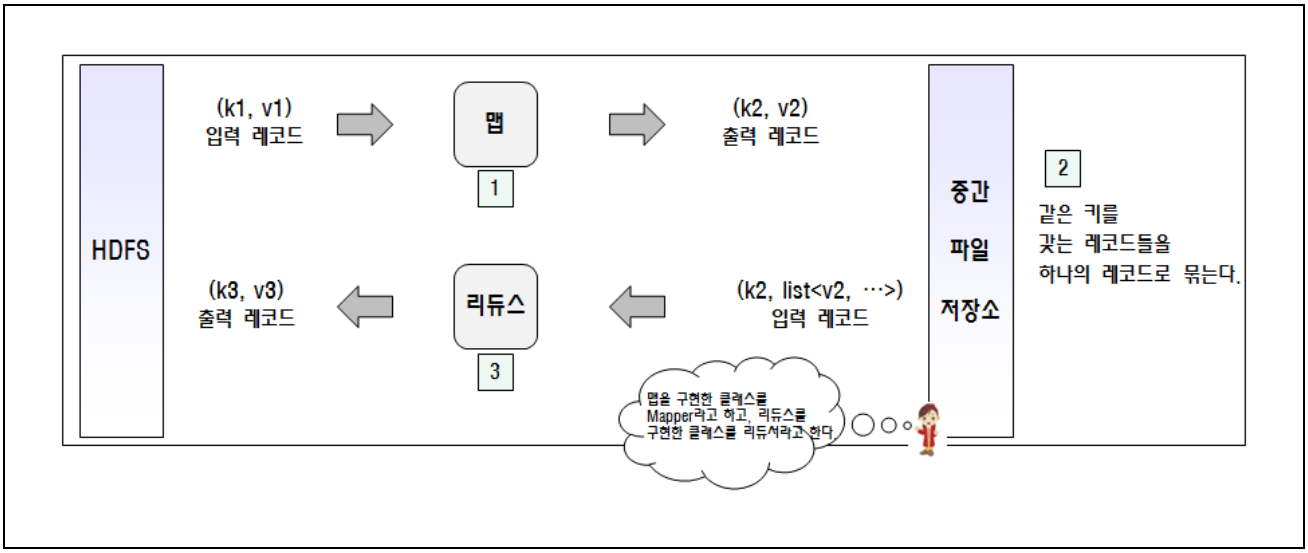
MapReduce를 사용하기에 적합한 분야와 비적합한 분야는 다음과 같다.

적합한 분야	설명
병렬도가 굉장히 높은 단순 작업	큰 용량의 이미지 파일들에 대한 포맷 변환 작업. 텍스트 파일에서 특정 단어 찾기
로그 분석	검색어 로그 아파치 웹 서버의 로그
머신 러닝	Clustering Classification

비적합한 분야	설명
리얼 타임 스트림 처리	대체 수단 Storm(트위트에서 오픈 소스화) 아파치 S4(야후에서 오픈 소스화) MongoDB 기반의 솔루션 들
반복 실행이 많이 필요한 작업들	-

Map과 Reduce의 개요

맵리듀스를 활용하려면 맵과 리듀스 두 단계로 작성 및 디자인을 해야 한다.
입력 데이터/출력 데이터 모두가 Key와 Value으로 구성된다.

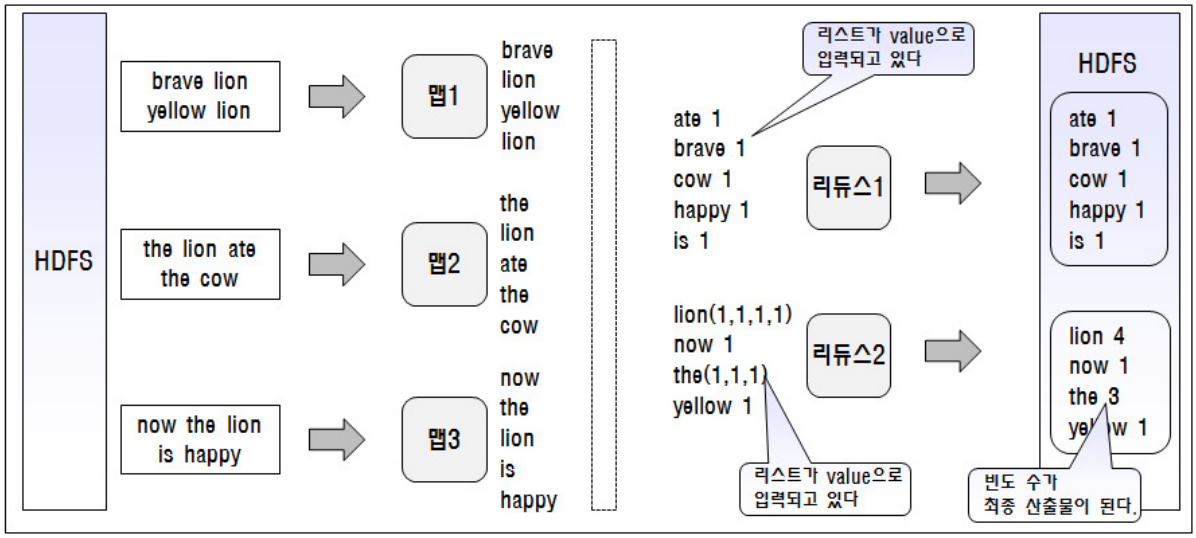


업무 Flow :
입력 레코드가 Map으로 들어간다. Map에서 출력된 레코드(키/밸류)들을 정렬한다. 이때 같은 밸류는 list 형태로 만들어서 Reduce단에 하나의 입력으로 들어간다.

개발자가 할 일 :
단순히 맵과 리듀스를 구현하기만 하면 된다.

WordCount 프로그램에서 Map/Reduce 동작

3개의 입력 텍스트가 존재하고 3개의 Map Task와 2개의 Reduce Task가 존재한다고 가정하자.



Reducer 영역에서 각 단어를 key로 하고, 그 단어의 빈도수 리스트를 value으로 만들어서 리듀스의 입력으로 지정한다.
Reducer의 출력은 결국 해당 단어가 key로 되고, value는 그 단어의 총 빈도수가 되고 총 빈도수가 바로 이 작업의 최종 결과물이 된다.

Map 클래스의 기본 골격

Mapper 클래스를 상속 받는 사용자 정의 클래스를 작성한다.
map() 메소드를 오버라이딩한다.
map() 메소드는 입력 레코드를 key/value를 받아서 다른 형태의 key/value 형태로 출력을 해주는 가장 핵심적인 메소드이다.

```
import org.apache.hadoop.mapreduce.Mapper;

//K1 : 입력 키의 타입, V1 : 입력 키의 밸류
//K2 : 출력 키의 타입, V2 : 출력 키의 밸류
public class MyMapper extends Mapper<K1, V1, K2, V2> {
    K2 k2 = new K2();

    V2 v2 = new V2();

    //context 객체는 변환된 키(k2), 밸류(v2)를 출력하는 용도로 사용된다.
    public void map(K1 key, V1 value, Context context){
        ... 중략
        //입력 받은 key, value를 K2 타입의 키와 V2 타입의 밸류를 프레임워크로 출력
        context.write( k2, v2 );
    }
}
```

Reduce 클래스의 기본 골격

Reducer 클래스를 상속 받는 사용자 정의 클래스를 작성한다.
reduce() 메소드를 오버라이딩한다.

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

//K2 : 리듀스의 입력 키의 타입, V2 : 리듀스의 입력 키의 밸류
//K3 : 리듀스의 출력 키의 타입, V3 : 리듀스의 출력 키의 밸류
public class MyReducer extends Reducer<K2, V2, K3, V3> {
    K3 k3 = new K3();

    V3 v3 = new V3();

    public void reduce(K2 key, Iterable<V2> values, Context context){
        ...
        //value를 루프를 돌면서 처리한 후 최종적으로
        //k3 타입의 키와, v3 타입의 밸류를 프레임워크로 출력한다.
        context.write( k3, v3 );
    }
}
```

main 메소드의 기본 골격

main 메소드를 구현한다.
해당 맵과 리듀스 클래스를 지정하고, 입력 파일 및 출력 파일들의 위치 등을 지정해주는 역할을 한다.

```
public class WordCount {
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = new Job(conf, "잡_이름");

        //job 인스턴스를 가지고 다음과 각종 초기화 작업을 수행한다.

        //맵, 리듀스 클래스 지정

        //입력 파일 위치, 출력 디렉토리 지정, 둘다 HDFS 상에 위치해야 한다.

        //입출력의 포맷을 지정한다.

        //최종 출력 키와 밸류의 타입 지정한다.

        //맵의 출력 키와 밸류의 타입 지정한다.

        //최종적으로 실행
        job.waitForCompletion(true);
    }
}
```

MyMapper 클래스의 구현

Mapper 클래스를 구현하는 일반적인 절차는 다음과 같다.

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class MyMapper extends Mapper<LongWritable, Text, Text, LongWritable> {
    //LongWritable : 자바의 Long 타입 정도로 이해하면 된다.
    private final static LongWritable one = new LongWritable(1);
    private Text word = new Text(); //Text : 자바의 String 정도로 이해하면 된다.

    //map 메소드는 주어진 텍스트를 단어 단위로 나누어서, 각 단어를 키로, 밸류를 1로 만들어 프레임워크로 내보낸다.
    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {

        StringTokenizer itr = new StringTokenizer(value.toString());

        while (itr.hasMoreTokens()) {
            word.set(itr.nextToken());
            //word는 텍스트로 변환된 단어이다
            //이를 밸류 1로 내보내고 있다.
            context.write(word, one); //one 객체는 항상 1이다.
        }
    }
}
```

MyReducer 클래스의 구현

Reducer 클래스를 구현하는 일반적인 절차는 다음과 같다.

```
import java.io.IOException;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class MyReducer extends Reducer<Text, LongWritable, Text, LongWritable> {
    private LongWritable result = new LongWritable();

    //reduce 메소드의 키로는 단어가 들어오고, 밸류 자리에는 그 단어가 처리 단계에서
    //나타난 횟수 만큼 1이라는 값으로 구성된 리스트가 들어온다.
    //여기서 Iterable<LongWritable>는 값을 담고 있는 list 목록들이다.
    public void reduce(Text key, Iterable<LongWritable> values, Context context)
            throws IOException, InterruptedException {
        int sum = 0; //여기서 sum은 단어의 빈도 수를 의미한다.
        for (LongWritable val : values) {
            sum += val.get(); //get() : 값을 구해주는 메소드
        }
        result.set(sum);
        //해당 단어를 키로, 총 빈도수를 밸류로 프레임워크에 출력한다.
        context.write(key, result);
    }
}
```

main 메소드의 구현

첫 번째 인자는 입력 파일의 위치이고, 두 번째 인자는 출력 파일이 저장될 디렉토리의 위치를 지정해야 한다.

```
public class WordCount {
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = new Job(conf, "WordCount");

        job.setJarByClass(WordCount.class); //jar 파일 이름의 지정
        job.setMapperClass(WordCountMapper.class); //맵 객체 지정
        job.setReducerClass(WordCountReducer.class); //리듀수 객체 지정
        job.setInputFormatClass(TextInputFormat.class); //입력 포맷을 지정한다.
        job.setOutputFormatClass(TextOutputFormat.class); //출력 포맷을 지정한다.
        job.setOutputKeyClass(Text.class); //최종 출력의 키 타입
        job.setOutputValueClass(IntWritable.class); //최종 출력의 밸류의 타입

        // 파일 시스템 제어 객체 생성
        FileSystem hdfs = FileSystem.get(conf);
        Path path = new Path(args[1]); // 경로 체크
        if (hdfs.exists(path)) {
            hdfs.delete(path, true);
        }
        FileInputFormat.addInputPath(job, new Path(args[0])); //입력 파일의 위치
        FileOutputFormat.setOutputPath(job, new Path(args[1])); //출력 파일의 위치

        //true 인자를 입력하게 되면, 진행 상황이 stdout으로 출력이 된다.
        job.waitForCompletion(true);
    }
}
```