

# MapReduce 프로그래밍

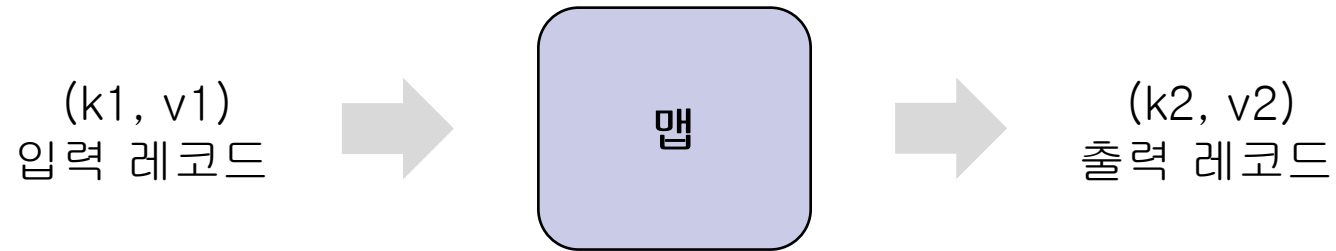
# 맵(Map)과 리듀스(Reduce)

## : Concept

### ▶ MapReduce 프로그램의 특성

- ▶ 맵과 리듀스의 두 단계로 구성
- ▶ 맵과 리듀스 모두 입력으로 주어지는 데이터나 출력으로 내보내는 데이터가 모두 키와 밸류로 구성

### ▶ 맵의 기본 동작

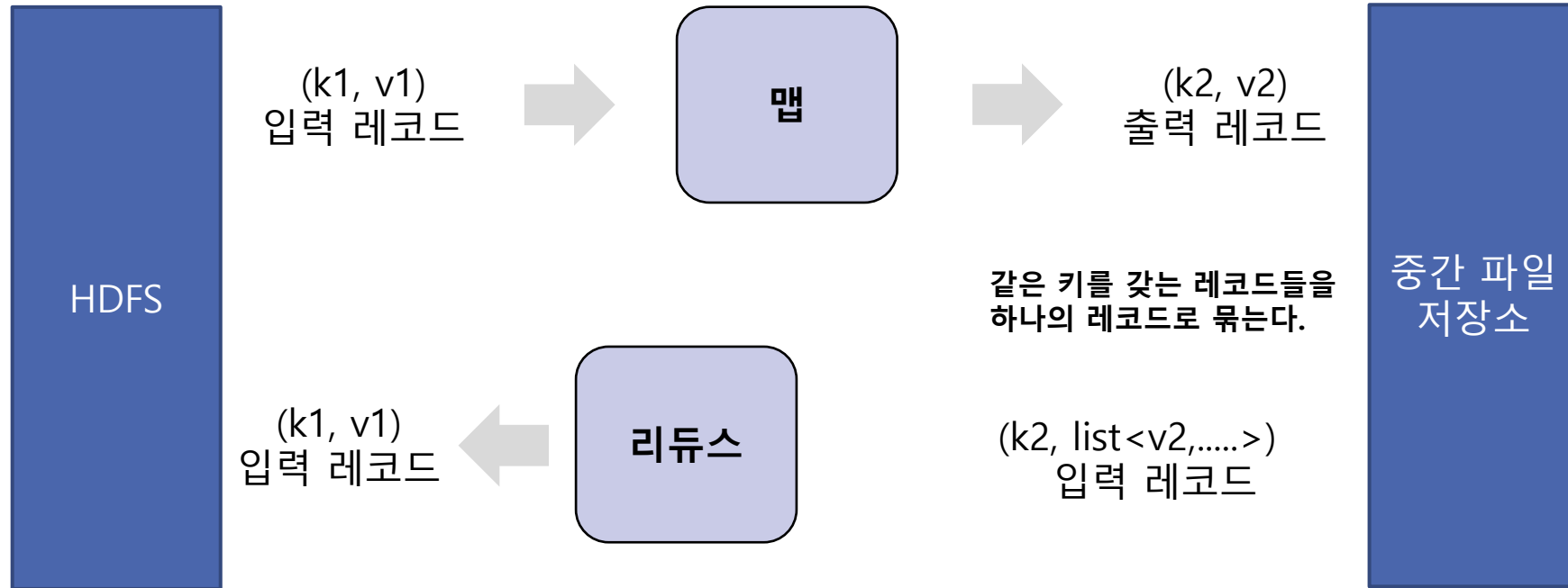


- ▶ 입력 데이터의 레코드를 하나씩 처리
- ▶ 입력 레코드는 하나의 키와 하나이 밸류로 구성
- ▶ 맵 단계에서는 주어진 입력 키와 밸류를 새로운 키와 밸류로 변환
- ▶ 모든 입력 레코드들이 맵을 통해 처리가 완료되면 리듀스 작업이 시작된다

# 맵(Map)과 리듀스(Reduce)

## : Concept

### ▶ 리듀스의 기본 동작



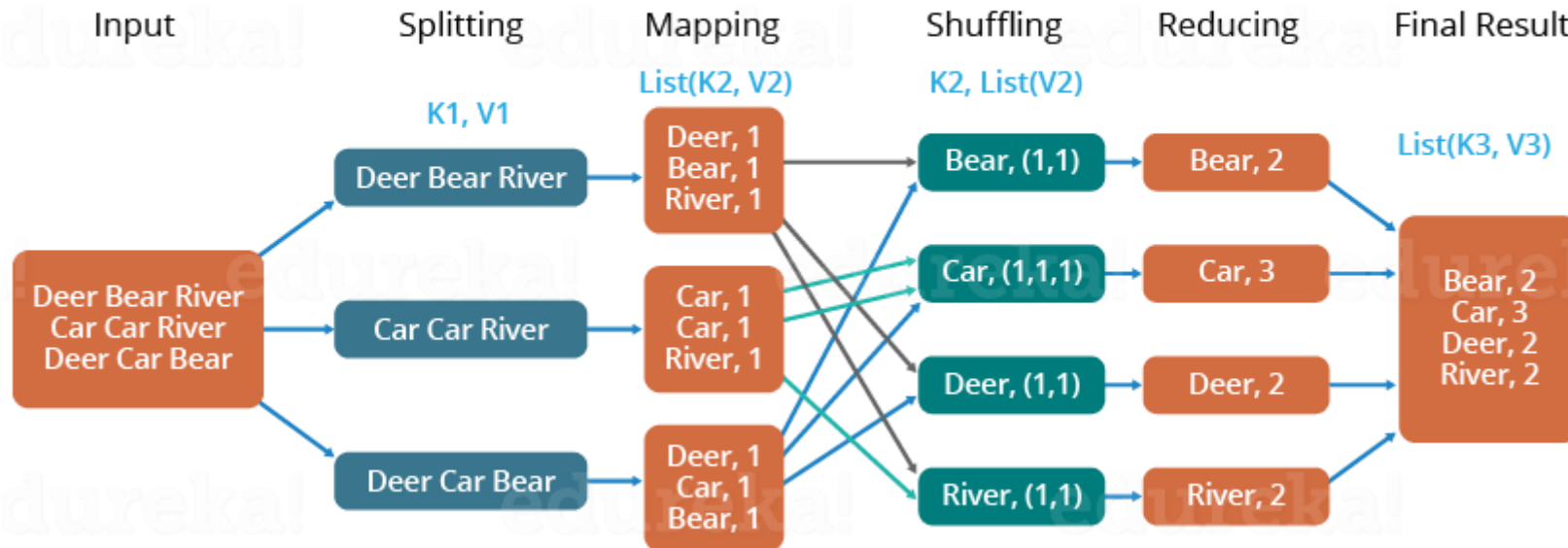
- ▶ 맵의 출력 레코드(키/밸류) 키를 기준으로 정렬한다
- ▶ 같은 키를 가진 레코드들을 묶는다
- ▶ 같은 키를 가진 맵의 출력 레코드들이 하나의 리듀스 입력 레코드로 만들어진다
- ▶ 리듀스는 또 다른 처리를 하고 새로운 키와 밸류를 출력한다

# 맵(Map)과 리듀스(Reduce)

## : MapReduce 프레임워크의 역할

- ▶ 입력 파일을 맵의 입력 레코드로 만들어주는 역할을 한다
- ▶ 맵에서 출력된 레코드들에서 같은 값을 갖는 키의 밸류들을 하나의 리스트로 묶어 리듀스로 넘겨주는 역할을 한다
- ▶ 맵과 리듀스는 여러 대의 머신에서 실행 가능하며 이를 프레임워크가 처리해 준다
- ▶ 맵을 클래스로 구현한 것을 매퍼(Mapper), 리듀스를 클래스로 구현한 것을 리듀서(Reducer)라 함

### The Overall MapReduce Word Count Process



# MapReduce 맛보기

## ▶ 대표적인 MapReduce 응용프로그램 WordCount 살펴보기

### ▶ HDFS에 텍스트파일을 저장

```
$ cd $HADOOP_HOME  
$ hdfs dfs -mkdir /example  
$ hdfs dfs -copyFromLocal README.txt /example  
$ hdfs dfs -ls /example
```

### ▶ WordCount 예제 실행

```
$ cd $HADOOP_HOME/share/hadoop/mapreduce  
$ hadoop jar hadoop-mapreduce-examples-2.9.2.jar wordcount /example/README.txt /output
```

### ▶ WordCount 결과의 확인

```
$ hdfs dfs -ls /output  
$ hdfs dfs -cat /output/part-r-0000 | more
```

### ▶ 결과를 로컬로 가져오기

```
$ hdfs dfs -get /output/part-r-0000 WordCount.txt
```

# 첫 번째 MapReduce 프로그램

: WordCount를 직접 만들어 보자

## ▶ Maven Project 생성

▶ Archetype : hadoop-archetype

## ▶ Project Info

▶ Group Id: com.bit

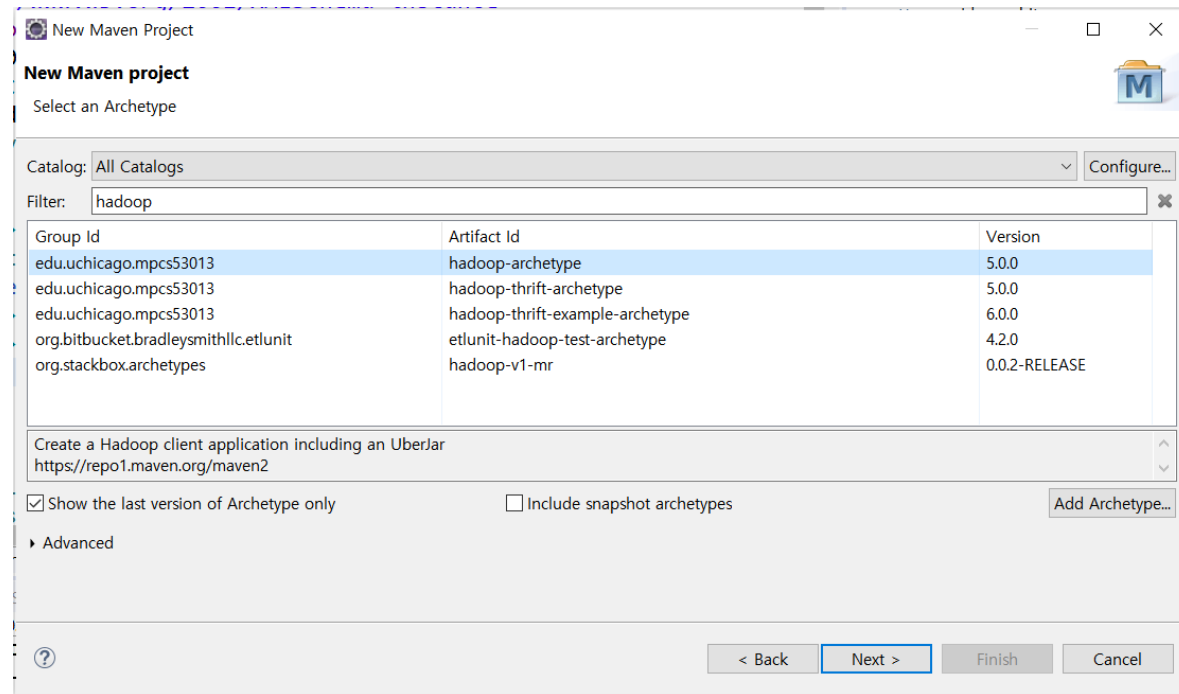
▶ Artifact Id: myhadoop

▶ version: 0.0.1

▶ class package: myhadoop

## ▶ Dependency 정보 변경

```
<dependency>
  <groupId>org.apache.hadoop</groupId>
  <artifactId>hadoop-client</artifactId>
  <!-- Client 버전은 서버의 Hadoop 버전과 일치 -->
  <version>2.9.2</version>
</dependency>
```



# 첫 번째 MapReduce 프로그램

: WordCount를 직접 만들어 보자

## ▶ Maven Project 생성

### ▶ compiler plugin 정보 변경

```
<plugin>
  <artifactId>maven-compiler-plugin</artifactId>
  <version>3.3</version>
  <!-- Java Version Config -->
  <configuration>
    <source>1.8</source>
    <target>1.8</target>
  </configuration>
</plugin>
```

### ▶ Maven > Update Project ...

# 첫 번째 MapReduce 프로그램

## : WordCount를 직접 만들어 보자

### ▶ 맵리듀스 프로그래밍의 요소

#### ▶ 데이터 타입

- ▶ 맵 리듀스는 네트워크 통신을 위한 최적화된 객체로 **WritableComparable** 인터페이스를 제공
- ▶ 맵 리듀스 프로그램에서 키/값으로 사용되는 모든 데이터 타입은 반드시 이 인터페이스가 구현되어 있어야 함

클래스 명	대상 데이터 타입
BooleanWritable	Boolean
ByteWritable	단일 byte
DoubleWritable	Double
FloatWritable	Float
IntWritable	Integer
LongWritable	Long
TextWrapper	UTF8 형식의 문자열
NullWritable	데이터 값이 필요 없을 때 사용



# 첫 번째 MapReduce 프로그램

## : WordCount를 직접 만들어 보자

### ▶ 맵리듀스 프로그래밍의 요소

#### ▶ InputFormat

- ▶ 입력 **Split**을 맵 메서드의 입력 파라미터로 사용할 수 있도록 해 주는 추상 클래스
- ▶ 입력 스플릿을 맵 메서드가 사용할 수 있게 **getSplits** 메서드를 제공

InputFormat	기능
TextInputFormat	텍스트 파일을 분석할 때 사용. 개행 문자를 기준으로 레코드를 분류. 키는 라인의 번호
KeyValueTextInputFormat	텍스트 파일을 입력 파일로 사용할 때, 임의의 키 값을 지정해 키/값의 목록으로 읽음
NLineInputFormat	맵 태스크가 입력받을 텍스트 파일의 라인 수를 제한하고 싶을 때 사용
DelegatingInputFormat	여러 개의 서로 다른 입력 포맷을 사용하는 경우 사용
CombineFileInputFormat	여러 개의 파일을 스플릿으로 묶어서 사용
SequenceFileInputFormat	<b>SequenceFile</b> 을 입력 데이터로 쓸 때 사용
SequenceFileAsBinaryInputFormat	<b>SequenceFile</b> 의 키와 값을 임의의 바이너리 객체로 변환하여 사용
SequenceFileAsTextInputFormat	<b>SequenceFile</b> 의 키와 값을 <b>Text</b> 객체로 변환하여 사용

# 첫 번째 MapReduce 프로그램

## : WordCount를 직접 만들어 보자

### ▶ 맵리듀스 프로그래밍의 요소

#### ▶ `OutputFormat`

- ▶ 맵리듀스 프로그램이 수행한 출력 데이터 포맷을 정의한 추상 클래스

OutputFormat	기능
<code>TextOutputFormat</code>	텍스트 파일에 레코드를 출력할 때 사용. 레코드 출력시 키와 값의 구분자는 탭을 사용
<code>SequenceFileOutputFormat</code>	<code>SequenceFile</code> 을 출력으로 쓸 때 사용
<code>SequenceFileAsBinaryOutputFormat</code>	바이너리 포맷의 키와 값을 <code>SequenceFile</code> 컨테이너에 저장
<code>FilterOutputFormat</code>	<code>OutputFormat</code> 클래스를 편리하게 사용할 수 있도록 한 래퍼 클래스
<code>LazyOutputFormat</code>	<code>FileOutputFormat</code> 상속 클래스는 출력 내용이 없어도 출력 파일을 생성. <code>LazyOutputFormat</code> 을 사용하면 첫 번째 레코드가 보내질 때만 출력 파일을 생성
<code>NullOutputFormat</code>	출력 데이터가 없을 때 사용

# 첫 번째 MapReduce 프로그램

## : WordCount를 직접 만들어 보자

### ▶ 맵리듀스 프로그래밍의 요소

#### ▶ Mapper의 구현

- ▶ 키는 입력 파일의 라인 번호, 값은 문장인 입력 파라미터

```
...
public class WordCountMapper extends Mapper<LongWritable, Text, Text, IntWritable> {

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    @Override
    protected void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {
        StringTokenizer itr = new StringTokenizer(value.toString());
        while(itr.hasMoreElements()) {
            word.set(itr.nextToken());
            context.write(word, one);
        }
    }
}
```

# 첫 번째 MapReduce 프로그램

## : WordCount를 직접 만들어 보자

### ▶ 맵리듀스 프로그래밍의 요소

#### ▶ Reducer의 구현

- ▶ 글자와 글자 수로 구성된 입력 파라미터를 받아 글자 수를 합산하여 출력

```
...
public class WordCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    private IntWritable result = new IntWritable();

    @Override
    protected void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val: values) {
            sum += val.get();
        }
        result.set(sum);
        context.write(key, result);
    }
}
```

# 첫 번째 MapReduce 프로그램

## : WordCount를 직접 만들어 보자

- ▶ 맵리듀스 프로그래밍의 요소
  - ▶ 드라이버 클래스 구현
    - ▶ Mapper와 Reducer를 실행하는 클래스

```
public class WordCount {  
    public static void main(String[] args) throws Exception {  
        Configuration conf = new Configuration();  
  
        if (args.length != 2) {  
            System.err.println("Usage: WordCount <input> <output>");  
            System.exit(2);  
        }  
  
        Job job = Job.getInstance(conf, "WordCount");  
  
        job.setJarByClass(WordCount.class);  
        job.setMapperClass(WordCountMapper.class);  
        job.setReducerClass(WordCountReducer.class);  
  
        //... cont
```

# 첫 번째 MapReduce 프로그램

## : WordCount를 직접 만들어 보자

- ▶ 맵리듀스 프로그래밍의 요소

- ▶ 드라이버 클래스 구현

- ▶ Mapper와 Reducer를 실행하는 클래스

```
//... cont
job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);

job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);

FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));

job.waitForCompletion(true);
}
```

- ▶ 프로젝트 빌드

- ▶ Run As > Maven Build ... > Goal: clean install

# 첫 번째 MapReduce 프로그램

## : WordCount를 직접 만들어 보자

### ▶ 맵리듀스 프로그래밍의 요소

#### ▶ 작성 클래스 테스트

```
$ hadoop jar myhadoop-0.0.1.jar myhadoop.WordCount /example/README.txt /output
```

#### ▶ 실행 결과의 확인

```
$ hdfs dfs -ls /output  
$ hdfs dfs -cat /output/part-r-0000 | more
```

#### ▶ 결과를 로컬로 가져오기

```
$ hdfs dfs -get /output/part-r-0000 WordCount.txt
```

실습: 미국 항공 데이터 맵 리듀스



# 미국 항공 데이터 MapReduce

## : 데이터의 준비

- ▶ 실습
- ▶ 다음 데이터를 받아 압축을 해제
  - ▶ <https://packages.revolutionanalytics.com/datasets/AirOnTime87to12>
  - ▶ HDFS의 `/user/Hadoop/input` 디렉터리에 저장
- ▶ 데이터의 구성과 구조를 파악
  - ▶ 참고:  
<https://packages.revolutionanalytics.com/datasets/AirOnTime87to12/AirOnTime87to12.dataset.description.txt>

# 미국 항공 데이터 MapReduce

## : 출발 지연 MapReduce 프로그래밍

### ▶ 목표:

- ▶ 출발 지연 시간(DepDelay) 컬럼이 0 초과인 레코드를 카운트하여 다음과 같은 포맷을 출력

```
1987,10 175568
1987,11 177218
1987,12 218858
...
2012,4 153453
2012,5 175546
2012,6 205523
2012,7 232461
2012,8 215957
2012,9 158063
```

# 미국 항공 데이터 MapReduce

## : 출발 지연 MapReduce 프로그래밍

- ▶ csv 분석 및 컬럼 데이터 반환을 위한 Parser 클래스 작성

- ▶ Class명 : AirlinePerformanceParser

- ▶ Field의 선언

```
private int year;  
private int month;  
  
private float arriveDelayTime = 0;  
private float departureDelayTime = 0;  
private float distance = 0;  
  
private String uniqueCarrier;
```

- ▶ 선언한 필드의 값을 받아올 수 있는 getter 메서드를 생성

# 미국 항공 데이터 MapReduce

## : 출발 지연 MapReduce 프로그래밍

- ▶ csv 분석 및 컬럼 데이터 반환을 위한 **Parser** 클래스 작성
  - ▶ 생성자 구현: csv의 1개 Line을 입력 받아 분할/저장하는 생성자 구현

```
public AirlinePerformanceParser(Text text) {  
    try {  
        String[] columns = text.toString().split(",");  
  
        // 운항 연도  
        year = Integer.parseInt(columns[0]);  
        // 운항 월  
        month = Integer.parseInt(columns[1]);  
        // 항공사 코드  
        uniqueCarrier = columns[5];  
  
        // ... cont
```

# 미국 항공 데이터 MapReduce

## : 출발 지연 MapReduce 프로그래밍

- ▶ csv 분석 및 컬럼 데이터 반환을 위한 Parser 클래스 작성
  - ▶ 생성자 구현: csv의 1개 Line을 입력 받아 분할/저장하는 생성자 구현

```
// ... cont

// 항공기 출발 지연 시간 설정
if (columns[16].length() != 0)
    departureDelayTime = Float.parseFloat(columns[16]);

// 항공기 도착 지연 시간 설정
if (columns[26].length() != 0)
    arriveDelayTime = Float.parseFloat(columns[26]);

// 운항 거리 설정
if (columns[37].length() != 0)
    distance = Float.parseFloat(columns[37]);
} catch (Exception e) {
    System.err.println("Error parsing a record: " + e.getMessage());
}
}
```

# 미국 항공 데이터 MapReduce

## : 출발 지연 MapReduce 프로그래밍

### ▶ Mapper 구현

```
public class DepartureDelayCountMapper
    extends Mapper<LongWritable, //입력 키의 타입, 입력 소스의 행
                  Text, //입력 데이터 행 내용
                  Text, //출력 키의 타입
                  IntWritable> { //출력 값의 타입

    // 맵 출력 값
    private final static IntWritable outputValue = new IntWritable(1);
    // 맵 출력 키
    private Text outputKey = new Text();

    // ... cont
```

# 미국 항공 데이터 MapReduce

## : 출발 지연 MapReduce 프로그래밍

### ▶ Mapper 구현

```
// ... cont
@Override
protected void map(LongWritable key, Text value, Context context)
    throws IOException, InterruptedException {
    // 첫 번째 행이 Header이면 skip
    if (key.get() == 0 && value.toString().contains("YEAR")) {
        return;
    }
    AirlinePerformanceParser parser = new AirlinePerformanceParser(value);
    // 출력 키 설정
    outputKey.set(parser.getYear() + "," + parser.getMonth());

    if (parser.getDepartureDelayTime() > 0)
        // 출력 데이터 생성
        context.write(outputKey, outputValue);
}
}
```

# 미국 항공 데이터 MapReduce

## : 출발 지연 MapReduce 프로그래밍

### ▶ Reducer 구현

```
public class DelayCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    private IntWritable result = new IntWritable();

    @Override
    protected void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException {
        int sum = 0;

        for (IntWritable value : values) {
            sum += value.get();
        }

        result.set(sum);
        context.write(key, result);
    }
}
```



# 미국 항공 데이터 MapReduce

## : 출발 지연 MapReduce 프로그래밍

### ▶ Driver 클래스 구현

- ▶ WordCount 예제를 참고하여 Mapper와 Reducer를 연결하여 DepartureDelayCount 맵 리듀스 프로그램을 작성해 봅시다.

- ▶ 일부 데이터를 대상으로 잘 작동하는지 테스트

# 작동 테스트

```
hadoop jar myhadoop-0.0.1.jar myhadoop.DepartureDelayCount input/airOT2010*.csv output
```

# 완료 후 확인

```
hdfs dfs -cat output/part-r-00000
```

- ▶ 전체 데이터를 대상으로 DepartureDelayCount를 실행해 봅시다.
  - ▶ 문제가 있으면 내용을 수정하고 분석을 완료해 봅시다.

### ▶ [추가 과제]

- ▶ ArrDelay(도착 지연) 컬럼의 값을 기반으로 도착 지연 MapReduce 프로그램을 작성해 봅시다.

사용자 정의 옵션

# 사용자 정의 옵션

- ▶ MapReduce 프로그램을 작성하게 되면 코드가 중복되는 경우가 많음
  - ▶ 비슷한 종류의 MapReduce 작업은 실행할 때 사용자가 정의한 파라미터에 따른 분기를 통해 공통된 코드를 정리할 수 있다.
- ▶ 사용자 정의 옵션을 돕기 위한 하둡의 클래스들
  - ▶ GenericOptionsParser : 하둡 콘솔 명령어에서 입력한 옵션을 분석

옵션	기능
-conf <파일명>	명시한 파일을 환경설정 리소스 정보에 추가
-D <옵션=값>	하둡 환경 옵션에 새 값을 추가
-fs <NameNode 호스트:포트>	네임노드를 새롭게 설정
-jtr <JobTracker 호스트:포트>	잡트래커를 새롭게 설정
-files <파일1, 파일2, ...>	로컬 파일을 HDFS 공유 파일시스템으로 복사
-libjars <jar1, jar2, ...>	로컬에 있는 jar 파일을 HDFS 공유 파일시스템으로 복사, 매퍼의 클래스 패스에 추가
-archives <arc1, arc2, ...>	로컬 아카이브 파일을 HDFS 공유 파일시스템으로 복사한 후 압축 해제

# 사용자 정의 옵션

- ▶ 사용자 정의 옵션을 돕기 위한 하둡의 클래스들
  - ▶ **Tool** : **GenericOptionsParser**의 콘솔 설정 옵션을 지원하기 위한 인터페이스
    - ▶ 내부에 **run** 메서드가 정의되어 있으므로 사용을 위해서는 **Override** 하여 구현하여야 한다.
  - ▶ **ToolRunner** : **Tool** 인터페이스를 구현한 클래스의 실행을 도와주는 헬퍼 클래스
    - ▶ **GenericOptionsParser**를 사용, 사용자가 콘솔 명령어에서 입력한 설정한 옵션을 분석
    - ▶ 분석 내용을 **Configuration** 객체에 설정
    - ▶ **Configuration** 객체를 **Tool** 인터페이스에 전달한 후
    - ▶ **Tool** 인터페이스의 **run** 메서드를 실행
- ▶ 사용자 정의 옵션 기능을 이용하여
  - ▶ 사용자 정의 옵션에 의한 파라미터(**workType**)에 의해 아래 두 작업을 분기하는 맵 리듀스 프로그램 작성해 봅니다.

# 사용자 정의 옵션

## ▶ 매퍼의 구현

- ▶ setup 메서드 : Mapper가 생성될 때 단 한번만 실행

```
...  
private String workType;  
  
@Override  
protected void setup(Context context)  
    throws IOException, InterruptedException {  
    workType = context.getConfiguration().get("workType");  
}  
...
```

# 사용자 정의 옵션

## ▶ 매퍼의 구현

- ▶ 전달 받은 파라미터(workType)에 따라 처리할 데이터를 분기

```
...
@Override
protected void map(LongWritable key, Text value, Context context)
    throws IOException, InterruptedException {
    ...
    if (workType.equals("departure")) {
        //workType 매개변수가 departure일 때의 분기 처리
    } else if (workType.equals("arrival")) {
        // workType 매개변수가 arrival일 때의 분기 처리도착 지연 시간을 가져와 매핑
    }
}
...
```

# 사용자 정의 옵션

- ▶ 드라이버 클래스 구현
  - ▶ 환경설정 정보를 제어할 수 있도록 `Configured` 클래스를 상속
  - ▶ 사용자 정의 옵션 조화를 위한 `Tool` 인터페이스를 구현
    - ▶ `Tool` 인터페이스 내에 선언된 `run` 메서드를 반드시 구현하여야 한다

```
...
public class DelayCount extends Configured implements Tool {
    public static void main(String[] args) throws Exception {
        //Tool 인터페이스 실행
        int res = ToolRunner.run(new Configuration(), new DelayCount(), args);
        System.out.println("MapReduce-Job Result:" + res);
    }
}
...
```

# 사용자 정의 옵션

- ▶ 드라이버 클래스 구현

- ▶ Tool 인터페이스의 run 메서드 구현

```
...  
@Override  
public int run(String[] args) throws Exception {  
    String[] otherArgs = new GenericOptionsParser(getConf(), args).getRemainingArgs();
```

- ▶ .getRemainingArgs()에서 반환하는 배열은 GenericOptionsParser에서 제공하는 파라미터를 제외한 나머지
    - ▶ -conf, -D, -fs 등 옵션을 설정한 파라미터는 모두 제외