|  |
| --- |
| FICS |
| Hadoop |
| MapReduce |

|  |
| --- |
| SeokRae Kim  2017-7-31 |

목 차

[**I.** **MapReduce Application 개발** 1](#_Toc491201846)

[**1.** **항공 데이터 분석** 1](#_Toc491201847)

[**2.** **공통 클래스(AirlinePerformanceParser) 구현** 5](#_Toc491201848)

[**3.** **항공 출발 지연 데이터 분석** 5](#_Toc491201849)

[**4.** **항공 도착 지연 데이터 분석** 9](#_Toc491201850)

[**5.** **사용자 정의 옵션 사용** 12](#_Toc491201851)

[**6.** **카운터 사용** 15](#_Toc491201852)

[**7.** **다수의 파일 출력** 17](#_Toc491201853)

[**II.** **정렬** 20](#_Toc491201854)

[**1.** **보조 정렬** 20](#_Toc491201855)

[**2.** **부분 정렬** 24](#_Toc491201856)

[**3.** **전체 정렬** 29](#_Toc491201857)

[**III.** **조인** 32](#_Toc491201858)

[**1.** **조인 데이터 준비** 32](#_Toc491201859)

[**2.** **맵 사이드 조인** 33](#_Toc491201860)

[**3.** **리듀스 사이드 조인** 35](#_Toc491201861)

1. **MapReduce Application 개발**
   1. **항공 데이터 분석**

* 데이터 다운로드를 위한 셸스크립트 작성
* CSV 파일 HDFS에 업로드
* 공통 클래스(AirlinePerformanceParser) 작성
* 항공 출발 지연 데이터 분석을 위한 매퍼(DepartureDelayCountMapper) 작성
* 매퍼에서 출력한 데이터를 처리하는 리듀서(DelayCountReducer) 작성
* 매퍼와 리듀서를 실행하는 드라이버(DepartureDelayCount) 작성
* 드라이버 실행

|  |
| --- |
| **#데이터 내려 받기** |
| **http://stat-computing.org/dataexpo/2009/**    **#1987년 ~ 2008년까지 총 12GB의 데이터로 하둡을 테스트하기로 한다.** |

|  |
| --- |
| **#경로 만들기** |
| **[root@wikibooks01 hadoop]# mkdir dataexpo** |

|  |
| --- |
| **#다운로드 받기 위한 셸 스크립트 작성** |
| **[root@wikibooks01 dataexpo]# vi downloads.sh**  **#!/bin/sh**  **for((i = 1987; i <= 2008; i++)) ; do**  **wget http://stat-computing.org/dataexpo/2009/$i.csv.bz2**  **bzip2 -d $i.csv.bz2**  **sed -e '1d' $i.csv > $i\_temp.csv**  **mv $i\_temp.csv $i.csv**  **done** |

|  |
| --- |
| **#셸스크립트 실행 (다운 받는데 시간이 오래 걸림)** |
| **[root@wikibooks01 dataexpo]# chmod 755 downloads.sh**  **[root@wikibooks01 dataexpo]# ./downloads.sh** |

|  |
| --- |
| **#다운로드 확인 및 용량 확인** |
| **[root@wikibooks01 dataexpo]# ls –al**  **[root@wikibooks01 dataexpo]# du -h** |

|  |
| --- |
| **#CSV 파일을 HDFS에 업로드 및 확인 – 오래 걸림** |
| **[root@wikibooks01 dataexpo]# hadoop dfs –mkdir airdata**  **[root@wikibooks01 dataexpo]# hadoop dfs –put /home/hadoop/dataexpo/\*.csv airdata**  **[root@wikibooks01 dataexpo]# hadoop dfs –ls** |

지금까지 항공 데이터 분석을 위한 데이터를 모두 다운로드 받았다.

이 데이터를 통해 앞으로 “항공 출발 지연 데이터 분석”과 “항공 도착 지연 데이터 분석”을 실행한다.

이제 데이터 분석을 위해 필요한 프로그램 개발을 설명한다.

* 1. **공통 클래스(AirlinePerformanceParser) 구현**

|  |
| --- |
| **#프로그램 구조** |
| **chapter05**  - 매퍼(Mapper)  - 리듀서(Reducer)  - 드라이버(Driver) **common**  - Parser |

|  |
| --- |
| **#항공 데이터를 파싱 할 클래스(AirlinePerformanceParser) 구현** |
| <https://github.com/SeokRae/hadoop_beginner/tree/SeokRae-branch/src/main/java/wikibooks/hadoop/common> |

* 1. **항공 출발 지연 데이터 분석**
     1. **매퍼(DepartureDelayCountMapper) 구현**

|  |
| --- |
| **#DepartureDelayCountMapper 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter05/DepartureDelayCountMapper.java> |

* + 1. **리듀서(DelayCountReducer) 구현**

|  |
| --- |
| **#DelatCountReducer 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter05/DelayCountReducer.java> |

* + 1. **드라이버(DepartureDelayCount) 클래스 구현**

|  |
| --- |
| **#DepartureDelayCount 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter05/DepartureDelayCount.java> |

* + 1. **드라이버 클래스 실행**

|  |
| --- |
| **#맵리듀스 잡 실행 (10분정도 소모)** |
| 실행 전 Maven build로 jar파일 생성을 잊지 말 것.  **[root@wikibooks01 hadoop]# hadoop jar hadoop-beginner-airportdata-0.0.1-SNAPSHOT.jar wikibooks.hadoop.chapter05.DepartureDelayCount airdata dep\_delay\_count**    매퍼에서 데이터를 받아 리듀서에서 최종적으로 생성한 데이터를 보여주는 핵심 로그  **17/07/31 21:31:22 INFO mapred.JobClient: Map input records=123534969**  **17/07/31 21:31:22 INFO mapred.JobClient: Map output records=50018329**  **17/07/31 21:31:22 INFO mapred.JobClient: Reduce input records=50018329**  **17/07/31 21:31:22 INFO mapred.JobClient: Reduce input groups=255** |

|  |
| --- |
| **#HDFS에 저장된 출력 데이터확인** |
| **[root@wikibooks01 hadoop]# hadoop fs –ls**  **[root@wikibooks01 hadoop]# hadoop fs -cat dep-delay-count/part-r-00000 | head -10**  **[root@wikibooks01 hadoop]# hadoop fs -cat dep-delay-count/part-r-00000 | tail -10** |

|  |
| --- |
| **#엑셀로 시각화하기 위해 CSV 파일로 만들기** |
| **[root@wikibooks01 hadoop]# hadoop fs -get dep\_delay\_count/part-r-00000 dep\_delay\_count.csv**  dep\_delay\_count.csv 파일이 만들어졌는지 확인한다. |

|  |
| --- |
| **#엑셀 피벗차트를 이용해서 데이터 확인** |
| **CentOS에 있는 LibreOffice Calc를 이용해서 데이터를 읽은 후,**  **엑셀프로그램으로 피벗차트를 만들어 보았다.** |

* 1. **항공 도착 지연 데이터 분석**
     1. **매퍼(ArrivalDelayCountMapper) 구현**

|  |
| --- |
| **# ArrivalDelayCountMapper 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter05/ArrivalDelayCount.java> |

* + 1. **리듀서(DelayCountReducer) 구현**

|  |
| --- |
| **# DelayCountReducer 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter05/DelayCountReducer.java> |

* + 1. **드라이버 클래스 구현**

|  |
| --- |
| **# ArrivalDelayCount 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter05/ArrivalDelayCount.java> |

* + 1. **드라이버 클래스 실행**

|  |
| --- |
| **#맵 리듀스 실행 (10분정도 소요)** |
| **[root@wikibooks01 hadoop]# hadoop jar hadoop-beginner-airportdata-0.0.1-SNAPSHOT.jar wikibooks.hadoop.chapter05.ArrivalDelayCount airdata arr\_delay\_count**    **17/08/01 21:29:33 INFO mapred.JobClient: Map input records=123534969**  **17/08/01 21:29:33 INFO mapred.JobClient: Map output records=57839171**  **17/08/01 21:29:33 INFO mapred.JobClient: Reduce input records=57839171**  **17/08/01 21:29:33 INFO mapred.JobClient: Reduce output records=255** |

|  |
| --- |
| **#head 와 tail로 데이터 확인** |
| **[root@wikibooks01 hadoop]# hadoop fs -cat arr\_delay\_count/part-r-00000 | head -10**  **[root@wikibooks01 hadoop]# hadoop fs -cat arr\_delay\_count/part-r-00000 | tail -10** |

* + 1. **항공 도착 지연 데이터 차트 만들기**

|  |
| --- |
| **# CSV 파일 만들기** |
| **[root@wikibooks01 hadoop]# hadoop fs -get arr\_delay\_count/part-r-00000 arr \_delay\_count.csv** |

|  |
| --- |
| **#엑셀 피벗차트를 이용해서 데이터 확인** |
| **CentOS에 있는 LibreOffice Calc를 이용해서 데이터를 읽은 후,**  **엑셀프로그램으로 피벗차트를 만들어 보았다.** |

* 1. **사용자 정의 옵션 사용**
* 사용자가 정의한 파라미터를 통해 **도착** 혹은 **출발** 지연 데이터를 분석
  + 1. **매퍼(DelayCountMapper) 구현**

|  |
| --- |
| **# DelayCountMapper 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter05/DelayCountMapper.java> |

* + 1. **리듀서 구현**

|  |
| --- |
| **#** **DelayCountReducer 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter05/DelayCountReducer.java> |

* + 1. **드라이버 클래스 구현**

|  |
| --- |
| **# DelayCount 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter05/DelayCount.java> |

* + 1. **드라이버 클래스 실행**
       1. **workType = departure**

|  |
| --- |
| **#departure 파라미터를 이용해 출발 데이터를 분석** |
| **[root@wikibooks01 hadoop]# hadoop jar hadoop-beginner-airportdata-0.0.1-SNAPSHOT.jar wikibooks.hadoop.chapter05.DelayCount -D workType=departure airdata departure\_delay\_count** |

|  |
| --- |
| **# 결과물 확인** |
| **3. 항공 출발 지연 데이터 분석과 같은 결과 인지를 확인**  **17/08/02 15:54:50 INFO mapred.JobClient: Map input records=123534969**  **17/08/02 15:54:50 INFO mapred.JobClient: Map output records=50018329**  **17/08/02 15:54:50 INFO mapred.JobClient: Reduce input records=50018329**  **17/08/02 15:54:50 INFO mapred.JobClient: Reduce output records=255** |

|  |
| --- |
| **#HDFS에 저장된 데이터 확인** |
| **[root@wikibooks01 hadoop]# hadoop fs -cat departure\_delay\_count/part-r-00000 | head -10**  **[root@wikibooks01 hadoop]# hadoop fs -cat departure\_delay\_count/part-r-00000 | tail -10** |

* + - 1. **workType = arrival**

|  |
| --- |
| **#arrival 파라미터를 이용해 도착 지연 데이터 분석** |
| **[root@wikibooks01 hadoop]# hadoop jar hadoop-beginner-airportdata-0.0.1-SNAPSHOT.jar wikibooks.hadoop.chapter05.DelayCount -D workType=arrival airdata arrival\_delay\_count**    **4. 항공 도착 지연 데이터 분석의 결과와 동일한지 확인**  **17/08/02 16:17:59 INFO mapred.JobClient: Map input records=123534969**  **17/08/02 16:17:59 INFO mapred.JobClient: Map output records=57839171**  **17/08/02 16:17:59 INFO mapred.JobClient: Reduce input records=57839171**  **17/08/02 16:17:59 INFO mapred.JobClient: Reduce output records=255** |

* 1. **카운터 사용**
* **Job 실행 결과 메시지와 웹 인터페이스에서 간편하게 데이터를 확인하기 위한 작업**
  + 1. **사용자 정의 카운터 구현**

|  |
| --- |
| **# DelayCounters 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter05/DelayCounters.java> |

* + 1. **매퍼 구현**

|  |
| --- |
| **# DelayCountMapperWithCounter 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter05/DelayCountMapperWithCounter.java> |

* + 1. **드라이버 클래스 구현**

|  |
| --- |
| **# DelayCountWithCounter 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter05/DelayCountWithCounter.java> |

* + 1. **드라이버 클래스 실행**

|  |
| --- |
| **# 맵리듀스 실행하기** |
| **[root@wikibooks01 hadoop]# hadoop jar hadoop-beginner-airportdata-0.0.1-SNAPSHOT.jar wikibooks.hadoop.chapter05.DelayCountWithCounter -D workType=departure airdata departure\_delay\_count\_counter**    **도착 지연을 확인할 수 없는 경우 2,302, 136건**  **스케줄 대로 도착한 경우 26,416,798건**  **스케줄 보다 일찍 도착한 경우 44,797,706건** |

|  |
| --- |
| **#맵리듀스 잡 관리자용 웹 화면** |
| **DelayCounters 확인** |

* 1. **다수의 파일 출력**
     1. **MultipleOutputs 이해**

MultipleOutputs는 여러 개의 출력 데이터를 쉽게 생성하도록 돕는 기능을 제공합니다. MultipleOutputs는 여러 개의 OutputCollectors를 만들고 각 OutputCollectors에 대한 출력 경로, 출력 포맷, 키와 값 유형을 설정합니다.

* + 1. **매퍼 구현**

|  |
| --- |
| **#** **DelayCountMapperWithMultipleOutputs 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter05/DelayCountMapperWithMultipleOutputs.java> |

* + 1. **리듀서 구현**

|  |
| --- |
| **# DelayCountReducerWithMultipleOutputs 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter05/DelayCountReducerWithMultipleOutputs.java> |

* + 1. **드라이버 클래스 구현**

|  |
| --- |
| **# DelayCountWithMultipleOutputs 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter05/DelayCountWithMultipleOutputs.java> |

* + 1. **드라이버 클래스 실행**

|  |
| --- |
| **#전체 카운터 정보를 출력** |
| **[root@wikibooks01 hadoop]# hadoop jar hadoop-beginner-airportdata-0.0.1-SNAPSHOT.jar wikibooks.hadoop.chapter05.DelayCountWithMultipleOutputs airdata delay\_count\_mos** |

|  |
| --- |
| **#출력 데이터 확인** |
| **[root@wikibooks01 hadoop]# hadoop fs -ls delay\_count\_mos**  **arrival-r-00000, departure-r-00000 파일 확인**    **\*\* part-r-00000 파일의 크기가 0인 이유는 reduce 메서드에서 context 객체에 write를 하지 않았기 때문이다.** |

|  |
| --- |
| **#출발 지연 데이터 확인** |
| **[root@wikibooks01 hadoop]# hadoop fs –cat delay\_count\_mos/departure-r-00000 | head -10**  **[root@wikibooks01 hadoop]# hadoop fs –cat delay\_count\_mos/departure-r-00000 | tail-10** |

|  |
| --- |
| **#도착 지연 데이터 확인** |
| **[root@wikibooks01 hadoop]# hadoop fs –cat delay\_count\_mos/arrival-r-00000 | head -10**  **[root@wikibooks01 hadoop]# hadoop fs –cat delay\_count\_mos/arrival-r-00000 | tail-10** |

1. **정렬**

* 항공 데이터 분석 프로그램에 보조 정렬을 적용
* 부분 정렬과 전체 정렬을 이용해 단순하게 입력 데이터만 정렬하는 맵리듀스 프로그램 개발
  1. **보조 정렬**
* 키의 값들을 그룹핑하고, 그룹핑된 레코드에 순서를 부여하는 방식

|  |
| --- |
| **#보조 정렬의 알고리즘** |
| 1. 기존 키의 값들을 조합한 복합 키(Composite Key)를 정의, 이때 키의 값 중에서 어떤 키를 그룹핑 키로 사용할지 결정 2. 복합키의 레코드를 정렬하기 위한 비교기(Comparator)를 정의 3. 그룹핑 키를 파티셔닝할 파티셔너(Partitioner)를 정의 4. 그룹핑 키를 정렬하기 위한 비교기(Comparator)를 정의 |

* + 1. **복합키 구현**

|  |
| --- |
| **# DataKey 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter06/DateKey.java> |

* + 1. **복합키 비교기 구현**

|  |
| --- |
| **# DataKeyComparator 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter06/DateKeyComparator.java> |

* + 1. **그룹키 파티셔너 구현**

|  |
| --- |
| **# GroupKeyPartitioner 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter06/GroupKeyPartitioner.java> |

* + 1. **그룹키 비교기 구현**

|  |
| --- |
| **# GroupKeyComparator 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter06/GroupKeyComparator.java> |

* + 1. **매퍼 구현**

|  |
| --- |
| **# DelayCountMapperWithDateKey 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter06/DelayCountMapperWithDateKey.java> |

* + 1. **리듀서 구현**

|  |
| --- |
| **# DelayCountReducerWithDateKey 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter06/DelayCountReducerWithDateKey.java> |

* + 1. **드라이버 구현**

|  |
| --- |
| **# DelayCountWithDateKey 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter06/DelayCountWithDateKey.java> |

* + 1. **드라이버 실행**

|  |
| --- |
| **# 드라이버 클래스 실행** |
| **[root@wikibooks01 hadoop]# hadoop jar hadoop-beginner-sort-0.0.1-SNAPSHOT.jar wikibooks.hadoop.chapter06.DelayCountWithDateKey airdata delay\_count\_sort**    **17/08/08 20:00:49 INFO mapred.JobClient: Reduce input groups=44**  **리듀서의 입력 데이터 그룹이 510개였는데, 44개만 출력되어 있음을 확인**  **왜냐하면 보조 정렬에서는 연도만 그룹 처리를 했기 때문에 1987 ~ 2008년까지 총 22개월로 출발 지연과 도착 지연에 대한 두 개의 데이터 셋을 만들어서 44개 되었다.** |

|  |
| --- |
| **# 출력 데이터 조회** |
| **[root@wikibooks01 hadoop]# hadoop fs –ls delay\_count\_sort**    **HDFS에 저장된 출력데이터 조회 시  출발 지연(departure-r-00000)과 도착 지연(arrival-r-00000)에 대한  처리 데이터가 모두 생성 되었음을 확인** |

|  |
| --- |
| **# 도착 지연 데이터 head, tail 데이터 조회** |
| **[root@wikibooks01 hadoop]# hadoop fs –cat delay\_count\_sort/arrival-r-00000 | head -10**  **[root@wikibooks01 hadoop]# hadoop fs –cat delay\_count\_sort/arrival-r-00000 | tail -10** |

|  |
| --- |
| **# 출발 지연 데이터 head, tail 데이터 조회** |
| **[root@wikibooks01 hadoop]# hadoop fs –cat delay\_count\_sort/departure-r-00000 | head -10**  **[root@wikibooks01 hadoop]# hadoop fs –cat delay\_count\_sort/ departure-r-00000 | tail -10** |

* 1. **부분 정렬**
* 매퍼의 출력 데이터를 맵파일(MapFile)로 변경해 데이터를 검색하는 방법
* 데이터 검색 시 주로 사용 됨

|  |
| --- |
| **#항공 지연 통계 데이터를 항공 운항 거리 순서대로 정렬하는 부분정렬 프로그램** |
| 1. 입력 데이터를 시퀀스 파일로 생성 2. 시퀀스 파일을 맵파일로 변경 3. 맵파일에서 데이터를 검색 |

* org.apache.hadoop.mapred 패키지를 이용해 맵리듀스 프로그램을 개발해야 한다.
  + 1. **시퀀스파일 생성**

|  |
| --- |
| **# SequenceFileCreator 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter06/SequenceFileCreator.java> |

|  |
| --- |
| **#2008년 통계 데이터로 맵리듀스 잡을 실행** |
| **[root@wikibooks01 hadoop]# hadoop jar hadoop-beginner-sort-0.0.1-SNAPSHOT.jar wikibooks.hadoop.chapter06.SequenceFileCreator airdata/2008.csv 2008\_sequencefile**    **매퍼는 입력 데이터를 연산하지 않기 때문에 리듀서는 필요하지 않다.** |

|  |
| --- |
| **# 출력 폴더에 시퀀스파일이 만들어져 있는지 확인** |
| **[root@wikibooks01 hadoop]# hadoop fs -ls 2008\_sequencefile** |

|  |
| --- |
| **#text 옵션을 이용해 시퀀스파일의 10개 라인을 확인** |
| **[root@wikibooks01 hadoop]# hadoop fs -text 2008\_sequencefile/part-00000 | head -10**     |  | | --- | | **810 2008,1,3,4,2003,1955,2211,2225,WN,335,N712SW,128,150,116,-14,8,IAD,TPA,810,4,8,0,,0,NA,NA,NA,NA,NA**  **810 2008,1,3,4,754,735,1002,1000,WN,3231,N772SW,128,145,113,2,19,IAD,TPA,810,5,10,0,,0,NA,NA,NA,NA,NA**  **515 2008,1,3,4,628,620,804,750,WN,448,N428WN,96,90,76,14,8,IND,BWI,515,3,17,0,,0,NA,NA,NA,NA,NA**  **515 2008,1,3,4,926,930,1054,1100,WN,1746,N612SW,88,90,78,-6,-4,IND,BWI,515,3,7,0,,0,NA,NA,NA,NA,NA**  **515 2008,1,3,4,1829,1755,1959,1925,WN,3920,N464WN,90,90,77,34,34,IND,BWI,515,3,10,0,,0,2,0,0,0,32**  **688 2008,1,3,4,1940,1915,2121,2110,WN,378,N726SW,101,115,87,11,25,IND,JAX,688,4,10,0,,0,NA,NA,NA,NA,NA**  **1591 2008,1,3,4,1937,1830,2037,1940,WN,509,N763SW,240,250,230,57,67,IND,LAS,1591,3,7,0,,0,10,0,0,0,47**  **1591 2008,1,3,4,1039,1040,1132,1150,WN,535,N428WN,233,250,219,-18,-1,IND,LAS,1591,7,7,0,,0,NA,NA,NA,NA,NA**  **451 2008,1,3,4,617,615,652,650,WN,11,N689SW,95,95,70,2,2,IND,MCI,451,6,19,0,,0,NA,NA,NA,NA,NA**   1. **2008,1,3,4,1620,1620,1639,1655,WN,810,N648SW,79,95,70,-16,0,IND,MCI,451,3,6,0,,0,NA,NA,NA,NA,NA** |   **데이터의 키 순서가 810 -> 515 -> 688 -> 1591로 표시되었는지 확인**  **-> 맵리듀스는 입력 데이터의 키를 기준으로 정렬되기 때문에 줄 번호를 기준으로 출력** |

* + 1. **맵파일 생성**
* **맵파일은 키 값을 검색할 수 있게 색인과 함께 정렬된 시퀀스 파일이다.**

|  |
| --- |
| **# MapFileCreator 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter06/MapFileCreator.java> |

|  |
| --- |
| **#시퀀스파일의 압축 포맷과 입출력 데이터 경로를 설정하고 잡을 실행** |
| **[root@wikibooks01 hadoop]# hadoop jar hadoop-beginner-sort-0.0.1-SNAPSHOT.jar wikibooks.hadoop.chapter06.MapFileCreator 2008\_sequencefile 2008\_mapfile** |

|  |
| --- |
| **#2008\_map\_file 폴더 조회** |
| **part-00000 디렉터리가 생성되어 있음을 확인**  **[root@wikibooks01 hadoop]# hadoop fs -ls 2008\_mapfile** |

|  |
| --- |
| **#part-00000 폴더 조회** |
| **맵파일의 규격에 맞게 data와 index 파일이 생성되어 있음을 확인**  **[root@wikibooks01 hadoop]# hadoop fs -ls 2008\_mapfile/part-00000** |

|  |
| --- |
| **#data 파일에 출력한 키와 값이 저장되어 있는지 확인** |
| **[root@wikibooks01 hadoop]# hadoop fs -text 2008\_mapfile/part-00000/data | head -10**     |  | | --- | | **11 2008,8,10,7,1315,1220,1415,1320,OH,5572,N819CA,60,60,14,55,55,JFK,LGA,11,8,38,0,,0,55,0,0,0,0**  **11 2008,5,15,4,2037,1800,2125,1900,OH,4988,N806CA,48,60,31,145,157,JFK,LGA,11,10,7,0,,0,145,0,0,0,0**  **17 2008,3,8,6,NA,1105,NA,1128,AA,1368,,NA,23,NA,NA,NA,EWR,LGA,17,NA,NA,1,B,0,NA,NA,NA,NA,NA**  **21 2008,5,9,5,48,100,117,130,AA,588,N061AA,29,30,11,-13,-12,MIA,FLL,21,6,12,0,,0,NA,NA,NA,NA,NA**  **21 2008,2,8,5,NA,1910,NA,1931,AA,1668,,NA,21,NA,NA,NA,FLL,MIA,21,NA,NA,1,A,0,NA,NA,NA,NA,NA**  **24 2008,3,12,3,955,931,1021,948,9E,2009,91619E,26,17,10,33,24,IAH,HOU,24,7,9,0,,0,0,0,9,0,24**  **24 2008,1,2,3,1245,1025,1340,1125,OH,5610,N806CA,55,60,11,135,140,IAD,DCA,24,5,39,0,,0,135,0,0,0,0**  **24 2008,11,27,4,943,940,1014,956,9E,5816,91469E,31,16,9,18,3,IAH,HOU,24,5,17,0,,0,0,0,18,0,0**  **28 2008,2,22,5,2046,2050,NA,2156,OO,3698,N298SW,NA,66,NA,NA,-4,SLC,OGD,28,NA,12,0,,1,NA,NA,NA,NA,NA**  **30 2008,12,31,3,NA,1800,NA,1820,OO,5613,N579SW,NA,20,NA,NA,NA,SFO,SJC,30,NA,NA,1,B,0,NA,NA,NA,NA,NA** |   **운항거리가 11 -> 17 -> 21 -> 24 -> 28 -> 30의 순서대로 정렬되어 있음을 확인** |

* + 1. **검색 프로그램 구현**
* 맵파일에서 원하는 키에 해당하는 값을 검색하는 방법
* 검색하고자 하는 키가 속하는 파티션 번호를 조회한 후, 파티션 번호로 맵파일에 접근해 데이터를 검색

|  |
| --- |
| **#SearchValueList 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter06/SearchValueList.java> |

|  |
| --- |
| **#운항 거리가 100마일인 데이터 목록을 조회** |
| **[root@wikibooks01 hadoop]# hadoop jar hadoop-beginner-sort-0.0.1-SNAPSHOT.jar wikibooks.hadoop.chapter06.SearchValueList 2008\_mapfile 100**    **파일을 찾을 수 없다는 에러가 발생하는 것이 정상**  **이유는 MapFileOutputFormat의 메서드는 파라미터로 전달받은 폴더에 있는 모든 폴더에서 맵파일이 존재하는지 체크를 한다.**  **2008\_mapfile 폴더에는 맵리듀스 잡에서 생성한 로그 파일 폴더가 있는데, MapFileOutputFormat이 로그 폴더를 체크해서 맵파일이 없다는 오류를 발생시킨 것이다.**  **검색 프로그램이 제대로 동작하기 위해서는 로그 폴더를 모두 삭제 한다.** |

|  |
| --- |
| **#로그 폴더 삭제** |
| **[root@wikibooks01 hadoop]# hadoop fs -rmr 2008\_mapfile/\_\*** |

|  |
| --- |
| **#검색 폴더 실행** |
| **[root@wikibooks01 hadoop]# hadoop jar hadoop-beginner-sort-0.0.1-SNAPSHOT.jar wikibooks.hadoop.chapter06.SearchValueList 2008\_mapfile 100 | head -10**    **운항 거리가 100마일인 데이터가 정상적으로 출력**  **특정 키에 해당하는 데이터만 검색해서 사용하는 경우 부분 정렬을 활용하는 예제 결과 확인** |

* 1. **전체 정렬**

|  |
| --- |
| **#전체 정렬 프로세스** |
| 1. 입력 데이터를 샘플링해서 데이터의 분포도를 조사 2. 데이터의 분포도에 맞게 파티션 정보를 미리 생성 3. 미리 생성한 파티션 정보에 맞게 출력 데이터를 생성 4. 각 출력 데이터를 병합 |

|  |
| --- |
| **#** **Name node is in safe mode – Issue** |
| **Hadoop이 정상적인 종료를 하지 않았을 때 에러 발생**  **비정상적인 종료 시 hadoop은 safe 모드로 이동하는데, 종료 시 restart할 때 문제가 없도록 해야 한다.**  **# hadoop dfsadmin –safemode leave**  **명령어를 통해 세이프 모드를 종료한다.** |

|  |
| --- |
| **#SequenceFileTotalSort 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter06/SequenceFileTotalSort.java> |

|  |
| --- |
| **#전체 정렬 실행** |
| **[root@wikibooks01 hadoop]# hadoop jar hadoop-beginner-sort-0.0.1-SNAPSHOT.jar wikibooks.hadoop.chapter06.SequenceFileTotalSort 2008\_sequencefile 2008\_totalsort**    #부분 정렬의 1단계에서 11개의 시퀀스 파일을 생성했던 로그  **17/08/14 19:01:29 INFO mapred.FileInputFormat: Total input paths to process : 11**  #RandomSampler에 설정한 대로 1,000개의 샘플을 사용하고 있다는 InputSampler 로그 출력  **17/08/14 19:01:40 INFO lib.InputSampler: Using 1000 samples** |

|  |
| --- |
| **#2008\_totalsort 디렉터리 조회** |
| **[root@wikibooks01 hadoop]# hadoop fs -ls 2008\_totalsort**  **Part-00000파일 생성 확인** |

|  |
| --- |
| **#시퀀스 파일을 조회한 결과를 텍스트로 저장해 정렬됐는지 확인** |
| **[root@wikibooks01 hadoop]# hadoop fs -text /user/root/2008\_totalsort/part-00000 | head -10**    **키 순서대로 정렬돼 있는 것을 확인, 이 파일들을 합치면 전체 정렬한 효과를 볼 수 있다.**  **정렬된 결과에서 데이터를 검색할 필요가 있다면 부분 정렬**  **단순히 정렬된 전체 데이터만 필요하다면 전체 정렬** |

1. **조인**

* 서로 다른 두 개의 테이블에서 키가 같은 데이터만 조회하는 것
  1. **조인 데이터 준비**
* **미국 항공 운항 통계 데이터와 항공사 코드 데이터에 대한 조인 연산을 수행하는 맵리듀스 프로그램 작성**
  + 1. **항공사 코드 데이터 다운로드**

|  |
| --- |
| **#항공사 코드 다운** |
| **[root@wikibooks01 ~]# mkdir /home/hadoop/meta**  **[root@wikibooks01 ~]# cd /home/hadoop/meta/**  **[root@wikibooks01 meta]# wget http://stat-computing.org/dataexpo/2009/carriers.csv** |

|  |
| --- |
| **#큰 따옴표(“)와 첫 번째 줄의 칼럼 정보 제거** |
| **[root@wikibooks01 meta]# find . -name carriers.csv -exec perl -p -i -e 's/"//g' {} \;**  **[root@wikibooks01 meta]# sed -e '1d' carriers.csv > carriers\_new.csv**  **[root@wikibooks01 meta]# mv carriers\_new.csv carriers.csv** |

|  |
| --- |
| **#HDFS에 carriers.csv 파일 업로드** |
| **[root@wikibooks01 hadoop]# hadoop fs -mkdir meta**  **[root@wikibooks01 hadoop]# hadoop fs -put /home/hadoop/meta/carriers.csv meta**  **[root@wikibooks01 hadoop]# hadoop fs -ls meta** |

* 1. **맵 사이드 조인**
* 매퍼가 단독으로 조인을 수행하는 방법
* 조인을 걸 데이터를 분산 캐시에 저장한 후, 매퍼가 실행될 때 분산 캐시에 저장된 데이터를 조회하는 방식으로 조인을 처리
  + 1. **분산 캐시**

|  |
| --- |
| **#분산 캐시의 동작 방식** |
| 1. 잡을 실행하면 사용자가 캐시로 설정한 파일을 HDFS로 복사 2. 태스크트래커는 태스크를 실행하기 전에 HDFS에 저장된 캐시 파일을 로컬 파일 시스템으로 복사, 이때 로컬 파일 시스템은 태스크가 실행될 노드의 로컬 파일 시스템이다. 3. 태스크는 로컬 파일 시스템에 복사된 파일을 조회한다. |

* + 1. **공통 클래스**

|  |
| --- |
| **#AirlinePerformanceParser 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/common/AirlinePerformanceParser.java> |

|  |
| --- |
| **#CarrierCodeParser 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/common/CarrierCodeParser.java> |

* + 1. **복합키 구현**

|  |
| --- |
| **#TaggedKey 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter07/TaggedKey.java> |

* + 1. **항공기 코드 데이터 매퍼 구현**

|  |
| --- |
| **#CarrierCodeMapper 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter07/CarrierCodeMapper.java> |

* + 1. **매퍼 구현**

|  |
| --- |
| **#MapperWithMapSideJoin 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter07/MapperWithMapSideJoin.java> |

* + 1. **드라이버 구현**

|  |
| --- |
| **#MapSideJoin 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter07/MapSideJoin.java> |

* + 1. **드라이버 클래스 실행**

|  |
| --- |
| **#항공사 코드 데이터의 경로, 맵리듀스 잡에서 사용할 입출력 경로로 클래스 실행** |
| [root@wikibooks01 hadoop]# hadoop jar hadoop-beginner-join-0.0.1-SNAPSHOT.jar wikibooks.hadoop.chapter07.MapSideJoin meta/carriers.csv airdata map\_join     * 현재 예제는 본 PC의 램이 16GB에서 8GB로 바꾼 상태이고 * 하둡 메모리 설정을 4GB에서 1GB로 바꿔 속도가 많이 느려진 상태이다. |

|  |
| --- |
| **#HDFS에 저장된 잡의 출력 데이터 조회** |
| [root@wikibooks01 hadoop]# hadoop fs -ls map\_join       * 총 187개의 맵 태스크의 출력 데이터가 생성되어 있다.   리듀서가 없는 맵 리듀스 잡이기 때문에 매퍼의 출력 데이터가 모두 생성되어 있는 것이다.  만약, 리듀서를 정의 했다면 part-m-xxxxx 파일은 생성되지 않고,  리듀서의 출력 데이터인 part-r-xxxxx 파일들이 생성되어있을 것이다. |

|  |
| --- |
| **#part-m-00000 조회** |
| [root@wikibooks01 hadoop]# hadoop fs -cat map\_join/part-m-00000 | head -5    맵 사이드 조인은 크기가 작은 데이터와 조인을 거는 경우에 적당하다.  왜냐하면 데이터 크기가 너무 클 경우 초기에 HashMap을 만드는 데 시간이 너무 오래 걸리고, 생성된 HashMap 객체가 시스템 리소스를 많이 잡아먹게 되어 결국에는 맵 리듀스 잡의 성능을 떨어뜨리기 때문이다. |

* 1. **리듀스 사이드 조인**
* 리듀서가 단독으로 조인을 처리하는 것, 하지만 매퍼도 함께 구현해야 한다.
* 매퍼는 조인할 데이터를 입력받아 입력 데이터의 태그, 조인키, 레코드를 출력 한다.
* 이때 태그는 RDBMS에서 조인을 걸 때 사용하는 테이블의 별칭(Alias)로 이해
* 매퍼는 출력 데이터에서 태그를 분류해서 조인을 수행
* 리듀서는 매퍼의 출력 데이터에서 태그를 분류해서 조인을 수행

|  |
| --- |
| **#리듀스 사이드 조인의 수행순서** |
| 1. 조인키(항공사 코드)와 태그를 조합한 복합키를 정의 2. 복합키의 레코드를 정렬하기 위한 비교기를 정의 3. 그룹핑 키(항공사 코드)를 파티셔닝하기 위한 파티셔너를 정의 4. 그룹핑 키를 정렬하기 위한 비교기를 정의 5. 항공사 코드를 태그하는 매퍼를 정의 6. 운항 통계 데이터를 태그하는 매퍼를 정의 7. 조인을 수행하는 리듀서를 정의 8. 드라이버 클래스를 정의 |

* + 1. **복합키 구현**
* 리듀스 사이드 조인의 복합키는 조인키와 태그의 조합으로 구성
* 조인키는 항공사 코드를 사용하며, 태그는 정수형 변수를 선언

|  |
| --- |
| **#TaggedKey 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter07/TaggedKey.java> |

* + 1. **복합키 비교기 구현**
* TaggedKey로 생성된 레코드의 정렬을 보장하려면 WritableComparator를 구현한 비교기를 구현해야 한다.

|  |
| --- |
| **#TaggedKeyComparator 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter07/TaggedKeyComparator.java> |

* + 1. **그룹키 파티셔너 구현**
* 각 매퍼는 다음과 같이 두 종류의 데이터를 출력한다.
* 각 데이터는 동일한 복합키를 처리하는 리듀서로 전달돼야 조인이 정상적으로 처리될 수 있다.

|  |
| --- |
| 1. 출력키: 복합키(TaggedKey), 출력값: 항공사 이름 2. 출력키: 복합키(TaggedKey), 출력값: 운항 통계 레코드 |

|  |
| --- |
| **#TaggedGroupKeyPartitioner 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter07/TaggedGroupKeyPartitioner.java> |

* + 1. **그룹키 비교기 구현**
* TaggedGroupKeyPartitioner를 적용하면 리듀서는 동일한 항공사 코드의 데이터를 내려 받을 수 있다.
* 하지만 이 데이터들은 순서가 보장되지 않으므로 그룹키 비교기를 구현해 데이터를 정렬해야 한다.
* 정렬 순서는 항공사 코드를 비교해 결정한다.

|  |
| --- |
| **#TaggedGroupKeyComparator 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter07/TaggedGroupKeyComparator.java> |

* + 1. **항공기 코드 데이터 매퍼 구현**
* 항공사 코드 데이터를 태깅하는 매퍼를 구현
* 매퍼는 항공사 코드 데이터를 입력 데이터로 전달받는다.
* 출력키는 “항공사 코드, 태그(0)”, 출력값은 “해당 항공사 코드의 항공사 이름”을 출력

|  |
| --- |
| **#CarrierCodeMapper 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter07/CarrierCodeMapper.java> |

* + 1. **항공기 운항 통계 데이터 매퍼 구현**
* 항공 운항 통계 데이터를 태깅하는 매퍼 클래스
* 항공 운항 통계 데이터를 입력 데이터로 전달
* 출력키는 TaggedKey, 출력값은 입력 데이터의 값을 그대로 출력

|  |
| --- |
| **#MapperWithReduceSideJoin 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter07/MapperWithReduceSideJoin.java> |

* + 1. **조인 리듀서 구현**
* 리듀스에는 입력키로 TaggedKey, 입력값으로 Text 타입의 항공사 이름과 운항 통계 레코드로 구성하여 처리 한다.
* 각 데이터 목록의 첫 번째 레코드는 반드시 태그 값이 0인, 항공사 이름이 전달되도록 한다.

|  |
| --- |
| **#ReducerWithReduceSideJoin 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter07/ReducerWithReduceSideJoin.java> |

* + 1. **드라이버 클래스 구현**
* 맵 리듀스 잡을 실행하는 드라이버 클래스를 구현
* 드라이버 클래스는 사용자가 입력한 조인할 두 개의 데이터를 MultipleInputs의 static 메서드인 addInputPath를 호출해 입력 데이터에 추가
* 이때 입력 데이터 타입과 매퍼도 함께 설정

|  |
| --- |
| **#ReduceSideJoin 작성** |
| <https://github.com/SeokRae/hadoop_beginner/blob/SeokRae-branch/src/main/java/wikibooks/hadoop/chapter07/ReduceSideJoin.java> |

* + 1. **드라이버 클래스 실행**

|  |
| --- |
| **#잡 실행** |
| * 첫 번째 파라미터로 항공사 코드 데이터 경로 * 두 번째 파라미터로 항공 운항 통계 데이터 경로 * 세 번째 파라미터로 조인 결과를 저장할 경로   [root@wikibooks01 hadoop]# hadoop jar hadoop-beginner-join-0.0.1-SNAPSHOT.jar wikibooks.hadoop.chapter07.ReduceSideJoin meta/carriers.csv airdata reduce\_join\_ouput     * 메모리 할당 문제로 램 바꾸고나서 다시 해볼 것 |