

맵리듀스 소개

순천향대학교 컴퓨터공학과
이 상 정

맵리듀스 소개

학습 내용

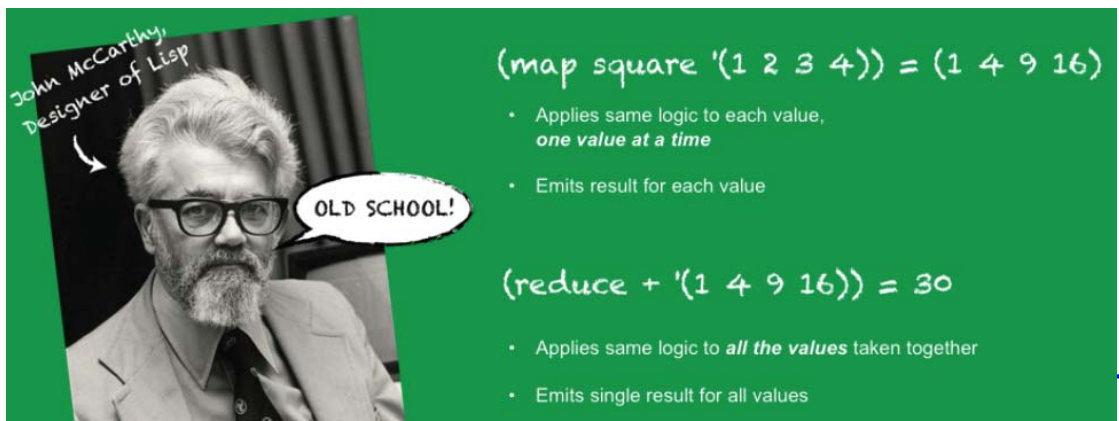
1. 맵리듀스 작업
2. 맵리듀스 데이터 흐름

1. 맵리듀스 작업

맵리듀스 소개

Lisp 맵-리듀스 (Map-Reduce)

- 맵리듀스(Mapreduce) 기본 개념은 Google 또는 하둡이 아닌 1970년대 Lisp 함수형 언어에서 도입
- Lisp 맵리듀스 모델 예
 - 맵 함수는 리스트의 각 입력 당 멱승 함수를 적용하여 출력 리스트 생성
 - 리듀스 함수는 맵의 출력 리스트를 모든 입력을 더하여 단일 출력 값 생성



`(map square '(1 2 3 4)) = (1 4 9 16)`

- Applies same logic to each value, *one value at a time*
- Emits result for each value

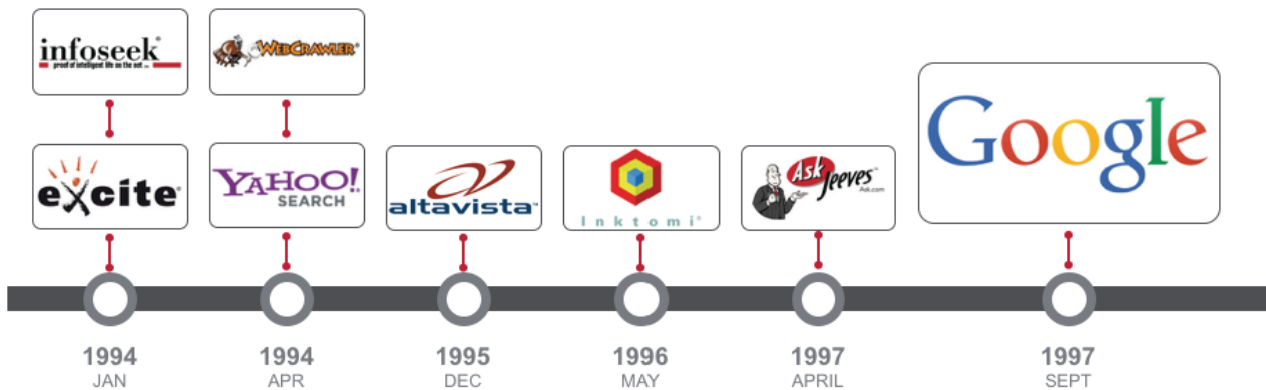
`(reduce + '(1 4 9 16)) = 30`

- Applies same logic to *all the values* taken together
- Emits single result for all values

웹 검색 엔진 (Web Search Engines)

□ 구글의 웹 검색 엔진에 맵리듀스 적용

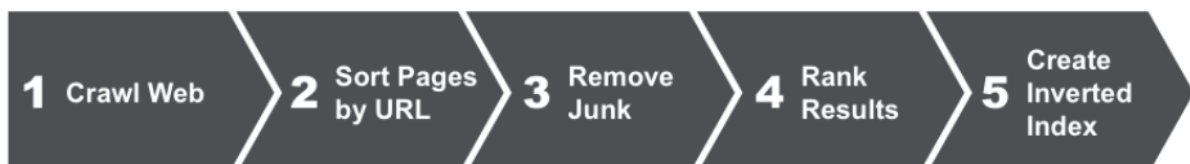
- 구글은 1997년에 19번째로 검색 엔진 개발



웹 검색 엔진 기법 개요

□ 웹 검색 단계

- 웹 페이지의 링크를 따라 웹 크롤링(crawling)
- URL 기준으로 페이지 정렬
- 잘못된 페이지(junk) 제거
- 주어진 단어의 빈도, 히트 수, 페이지의 갱신 여부에 따라 URL을 정렬하여 랭크(등수)를 부여
 - PageRank 논문(1998)
 - <http://infolab.stanford.edu/~backrub/google.html>
- 사용자에게 표시할 인덱스 생성
 - 각 단어에 대해, 단어를 포함하는 URL의 리스트를 생성



구글 논문 - 단어 카운트 알고리즘 (1)

□ MapReduce: Simplified Data Processing on Large Clusters

- OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004.
- <https://research.google.com/archive/mapreduce.html>

```
map(String key, String value):
    // key: document name
    // value: document contents
    for each word w in value:
        EmitIntermediate(w, "1");

reduce(String key, Iterator values):
    // key: a word
    // values: a list of counts
    int result = 0;
    for each v in values:
        result += ParseInt(v);
    Emit(AsString(result));
```

순천향대학교 컴퓨터공학과

7

구글 논문 - 단어 카운트 알고리즘 (2)

□ Map 메서드

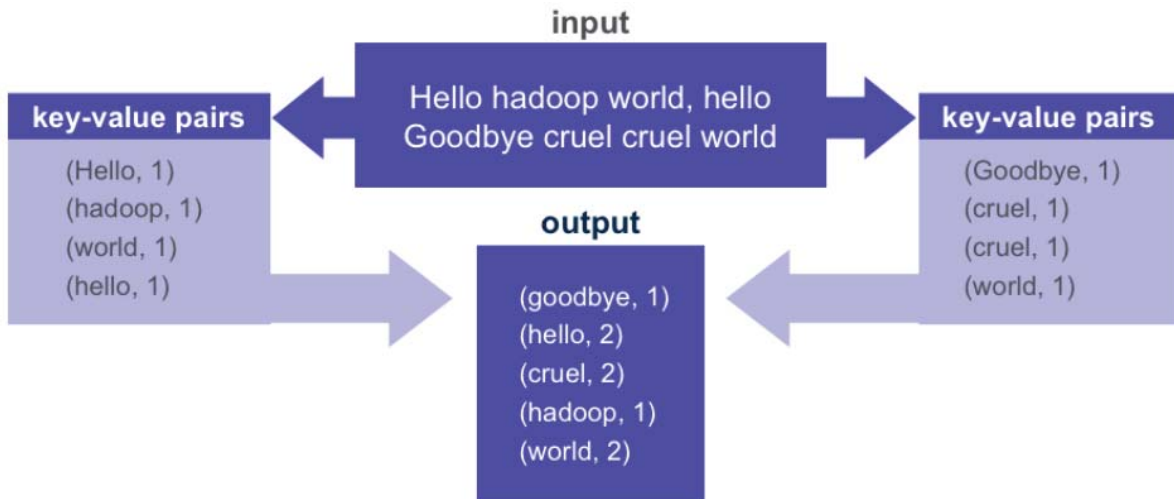
- 키와 값(key and value)을 입력으로 받음
 - 키는 문서의 이름, 값은 문서의 내용
- 문서의 각 단어에 대해 루프를 수행
- (단어, 1)의 2-튜플을 생성

□ Reduce 메서드

- 키와 값들의 리스트를 입력으로 받음
 - 키는 단어이고, 리스트는 단어의 카운트 리스트로 1 값들의 리스트
- 리스트의 값들에 대해 루프를 수행하여 더함
- 단어에 대한 최종 카운트 값을 생성

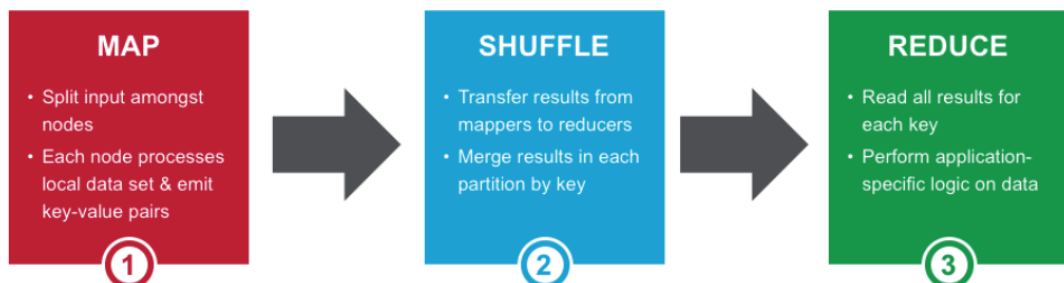
구글 논문 - 단어 카운트 예

- 두 개로 입력을 분할하여 맵 수행 예
 - 하둡은 소문자/대문자 구분 없음



맵리듀스 계산 모델

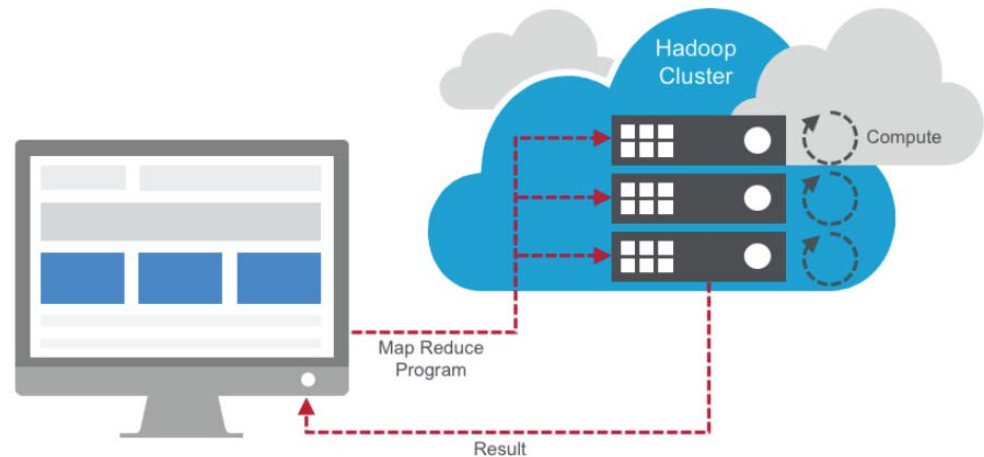
- 맵리듀스는 맵, 셔플, 리듀스 (map, shuffle, reduce)의 3 단계로 수행
 - **태스크 트래커(Task Tracker)** 노드들에 분산 배치된 입력 데이터에 대해 **맵 단계**에서 한 번에 한 레코드의 입력에 대해 맵 함수를 적용하여 키-값 들의 쌍을 출력
 - **셔플 단계**에서 각 노드의 **맵의 부분 결과를 통합**한 후 리듀스 단계로 전송
 - **리듀스 단계**에서는 각 키에 대해 노드 단위로 분할된 **모든 데이터를 리스트로 받아 리듀스 함수를 적용**하여 0개 이상의 키-값 쌍들을 출력



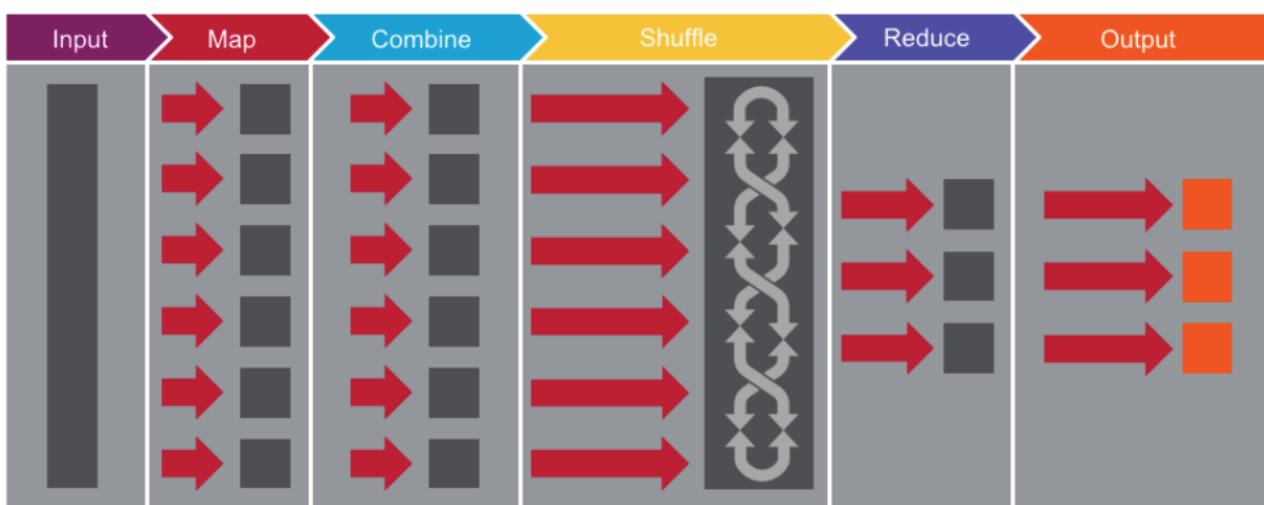
하둡 런타임 모델 (Hadoop Runtime Model)

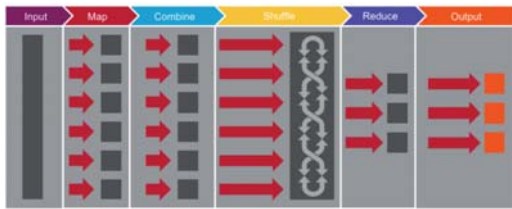
□ 맵리듀스 모델은 데이터가 위치한 노드로 계산 프로그램을 전송

- 맵리듀스 작업이 실행되면 맵과 리듀스 태스크들을 클러스터의 노드에 전송
- 노드의 로컬 디스크 데이터 상의 계산을 수행하여 네트워크 트래픽을 최소화



맵리듀스 계산 모델 (1)





맵리듀스 계산 모델 (2)

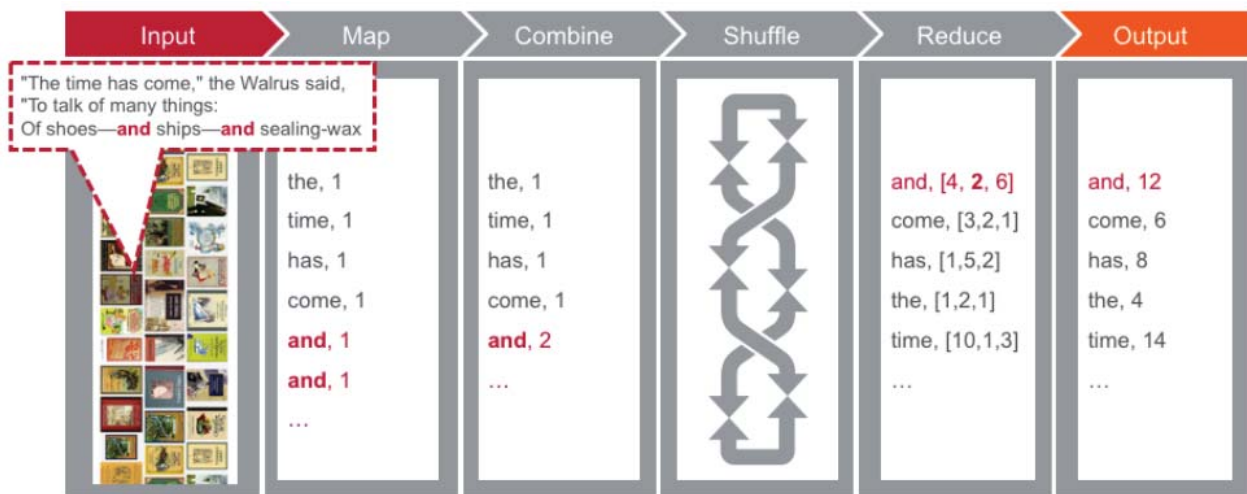
□ 맵리듀스 작업 단계

- 하둡 프레임워크가 입력 노드에 **데이터를 분할하여 저장**
 - 각 분할 데이터는 텍스트, 멀티미디어, 구조화된 데이터 등 임의의 타입의 많은 레코드로 구성
- **맵 태스크**가 각 분할 데이터를 처리
 - 특정 키의 모든 레코드들이 같은 리듀스 태스크에 전달되도록 맵의 출력들이 분할
- **병합(Combine) 단계**에서 같은 키의 레코드들을 결합하여 노드 간에 복사되는 레코드들의 수를 줄임
 - 리듀서와 동일한 메서드 사용
- **셔플 단계**에서 맵 태스크의 중간 결과를 리듀스 태스크로 전송
- **리듀스 태스크**를 수행하여 0개 이상의 키-값 쌍들을 생성
- 프레임워크가 리듀스 태스크의 결과를 취합하여 **출력**

□ 사용자가 맵과 리듀스 태스크의 코드만 작성하면 하둡 프레임워크가 대부분을 처리

맵리듀스 소개

맵리듀스 예 - 단어 카운트 (WordCount)



2. 맵리듀스 데이터 흐름

맵리듀스 소개

맵리듀스 작업 흐름 (Mapreduce Workflow)



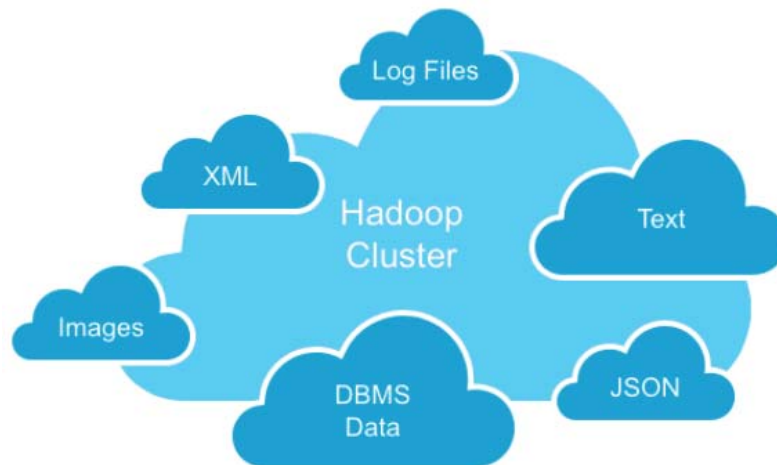
□ 맵리듀스 작업 흐름 단계

- 클러스터에 데이터를 적재
- 맵리듀스로 데이터를 분석 처리
- 분석 결과를 HDFS에 저장
- 클러스터의 결과를 읽어서 비즈니스 로직 분석

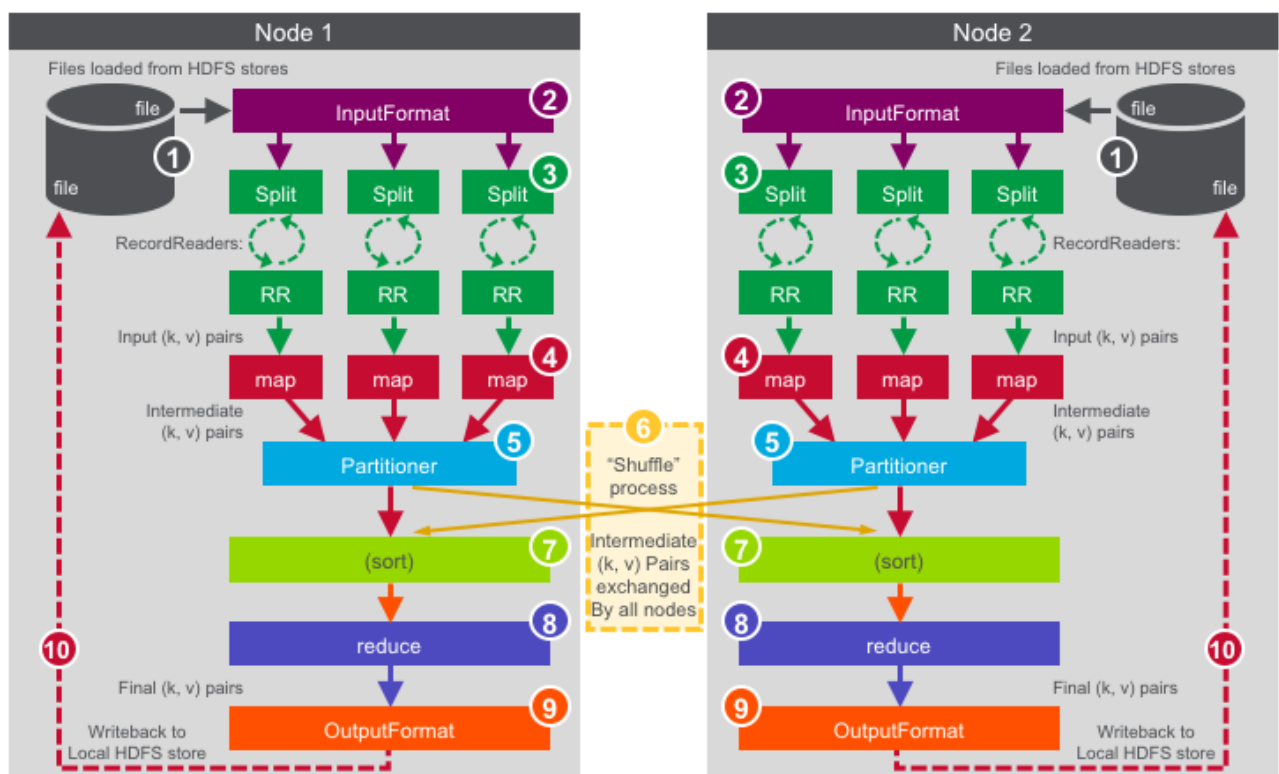
클러스터에 데이터 적재

□ 구조화 및 비구조화 데이터 소스에서 데이터를 하둡 클러스터에 적재하는 다양한 툴(에코 시스템) 사용

- 스쿱(Sqoop)은 SQL 데이터를 분산 파일 시스템에 적재
- 플럼(Flume)은 로그 데이터를 분산 파일 시스템에 적재



맵리듀스 데이터 실행 흐름 (1)

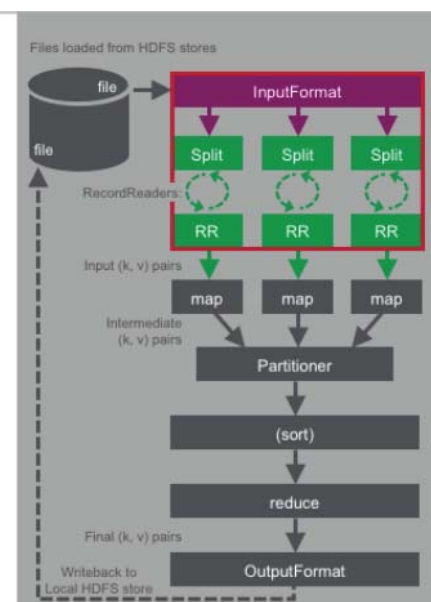
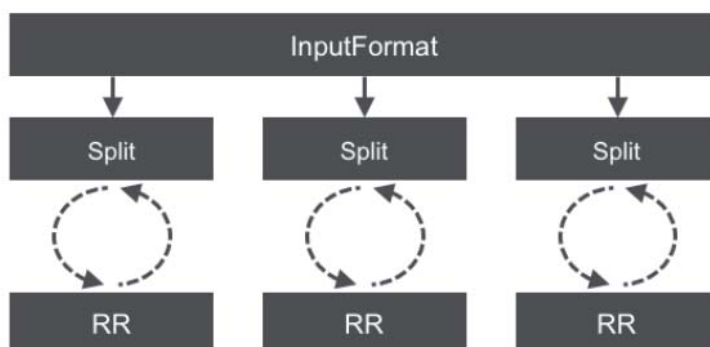


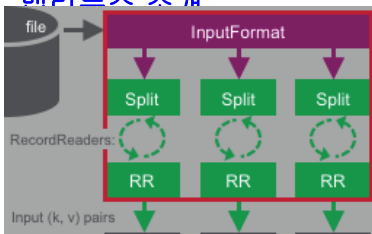
맵리듀스 데이터 실행 흐름 (2)

1. 하둡 파일 시스템으로부터 데이터를 적재
2. 작업(job)이 입력 데이터 형식을 정의 (InputFormat)
3. 모든 노드의 각 map() 메서드에서 실행될 데이터를 분리 (Split)
 - 레코드 리더(record reader)가 데이터를 map() 메서드의 입력이 되는 키-값 쌍(key-value pairs)으로 파싱 (RR)
4. map() 메서드가 파티셔너(partitioner)로 보내지는 키-값 쌍들을 생성 (map)
5. 여러 개의 리듀서(reducer)가 있는 경우 각 리듀스 태스크 당 하나의 파티션을 생성 (Partitioner)
6. 키 값 기준으로 하나의 파티션을 갖도록 셔플 (Shuffle)
7. 각 파티션에서 키 값 기준으로 키-값 쌍들을 정렬 (sort)
8. reduce() 메서드가 중간 키-값 쌍(intermediate k-value pairs)들을 입력 받아 최종 키-값 쌍들의 리스트로 줄임 (reduce)
9. 작업이 출력 데이터 형식을 정의 (OutputFormat)
10. 출력 데이터가 하둡 파일 시스템에 저장

InputFormat 클래스 (1)

- InputFormat 클래스는 작업의 입력 데이터를 검증하고, 맵 처리를 위해 파일들을 분리하고, RecordReader 객체의 인스턴스를 생성



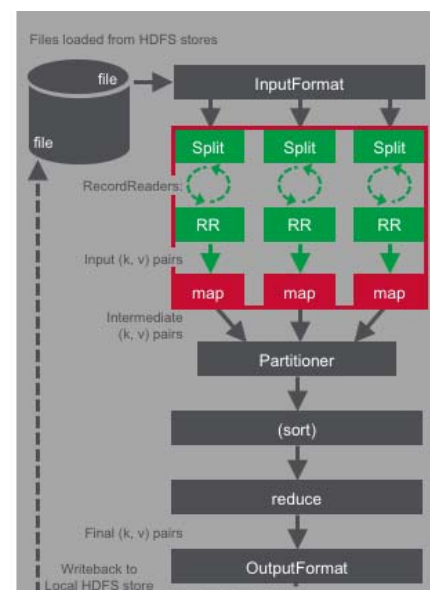
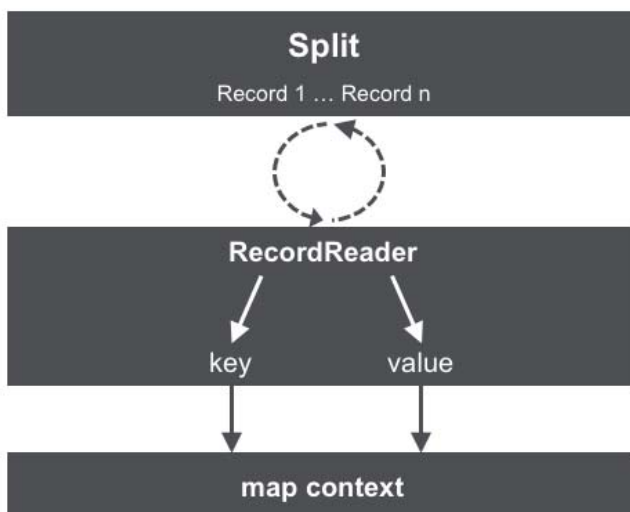


InputFormat 클래스 (2)

- 파일의 분할되는 크기는 디폴트로 블록의 크기
 - 하둡의 디폴트 블록 크기는 64MB
 - 분할된 데이터(InputSplit)는 레코드의 집합으로 맵 단계의 입력 키-값 쌍으로 전달
- 각 노드에 태스크가 할당되면 TaskTracker가 InputSplit을 RecordReader 생성자에게 전달
 - RecordReader는 레코드 단위로 읽어 들인 키-값 쌍들을 map() 메서드에게 전달
 - 디폴트로 RecordReader는 한 라인을 한 레코드로 간주
 - RecordReader와 InputFormat 클래스는 멀티-라인 레코드 등 다른 형식의 레코드 형식을 정의
 - InputSplit의 모든 레코드를 읽어 들였으면 RecordReader는 중지

Mapper 클래스 (1)

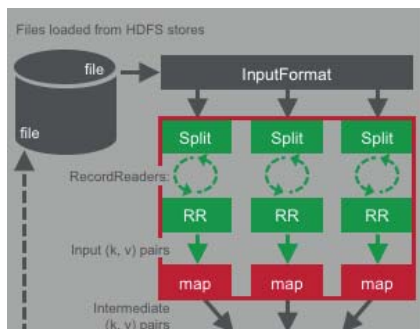
- 맵 단계는 Mapper 클래스의 map() 메서드로 구현



Mapper 클래스 (2)

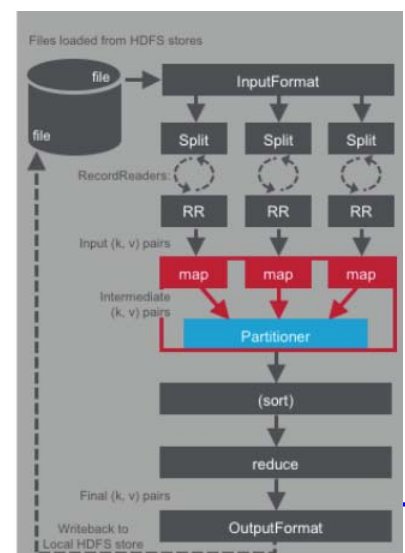
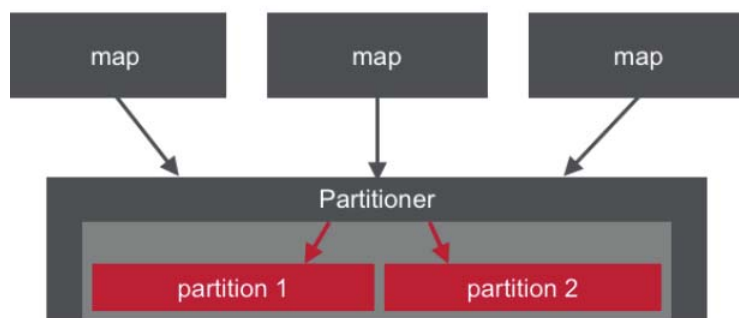
□ map() 메서드

- 3개의 인수: 키(key), 값(value), 컨텍스트(context)
 - 디폴트로 RecordReader는 입력 파일에서 레코드의 바이트 오프셋을 키로, 해당 바이트 오프셋의 라인을 값으로 정의
- map() 메서드는 입력의 값을 **토큰화**하여 처리
- 각 토큰에 대해 무엇을 할 것인지는 프로그램의 로직 내용에 좌우
- **맵 컨텍스트(map context)** 객체는 map() 메서드의 출력을 취합하여 다음 단계의 파티셔너로 전달



Partitioner 클래스

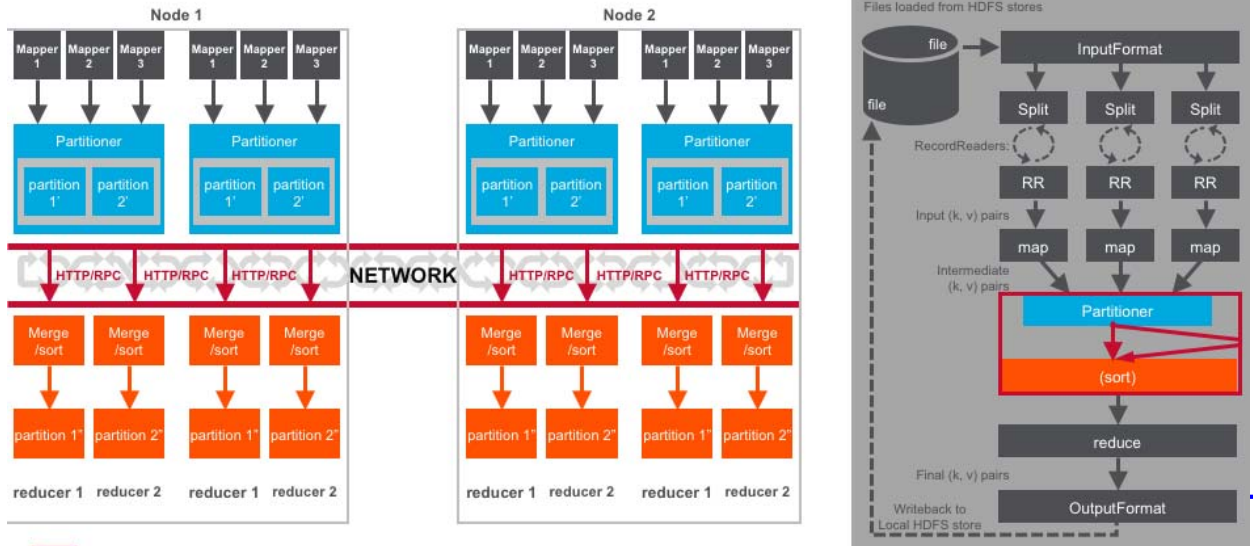
- **Partitioner** 클래스는 map() 메서드의 중간 키-값 쌍들을 입력 받아서, 레코드 키를 해싱(hasing)하고, 해시된 키에 기반하여 **파티션을 생성**
 - 같은 키의 레코드들은 같은 파티션으로 저장하여 같은 리듀서로 전송



셔플 단계 (Shuffle Phase)

- 셔플 단계에서는 파티션을 정렬하고 통합을 하여 새로운 파티션을 구성한 후 리듀서(reducer)에 전송

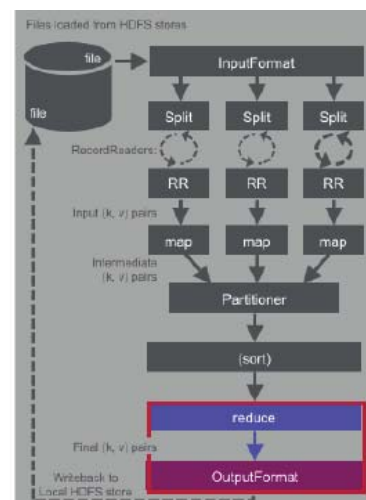
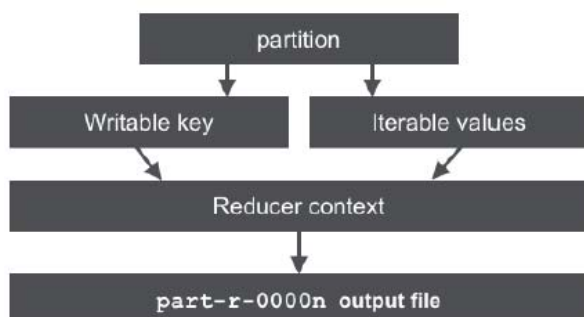
- HTTP나 RPC 등 네트워크 프로토콜을 사용하여 전송
- 맵리듀스 프로그램에서 가장 큰 네트워크 부하를 유발



Reducer 클래스

- 파티션의 각 키와 해당 값들의 리스트에 대해 `reduce()` 메서드가 호출

- 각 값들의 리스트에 대해 처리한 결과를 컨텍스트에 저장
- 컨텍스트의 `OutputComitter`가 실행되는 리듀서 당 하나의 출력 파일을 생성



맵리듀스 작업(MapReduce Job)의 결과

□ 맵리듀스 작업의 결과는 사용자가 지정한 디렉토리에 저장

- `_SUCCESS` 빈 파일은 작업의 성공을 표시
- `_logs/history*` 파일들에 작업의 이력들이 캡처
- `reduce()` 메서드의 출력은 각 리듀서에 대해 `part-r-00000`, `part-r-00001` ... 등의 파일들에 각각 저장
- 맵 단계만 수행되는 작업의 경우에는 `part-m-00000`, `part-m-00001` ... 등의 파일들에 맵의 출력이 저장

• `_SUCCESS`

• `_logs/history*`

• `part-r-00000`, `part-r-00001`, . . .

• `part-m-00000`, `part-m-00001`, . . .

27

참고 자료

□ MapR Academy, <http://learn.mapr.com/>

- Build Hadoop MapReduce Applications
 - <https://learn.mapr.com/series/hadoop-developer-series/dev-300-build-hadoop-mapreduce-applications>
 - Lesson 1: Introduction to MapReduce

□ 구글 논문

- MapReduce: Simplified Data Processing on Large Clusters
 - OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004.
 - <https://research.google.com/archive/mapreduce.html>