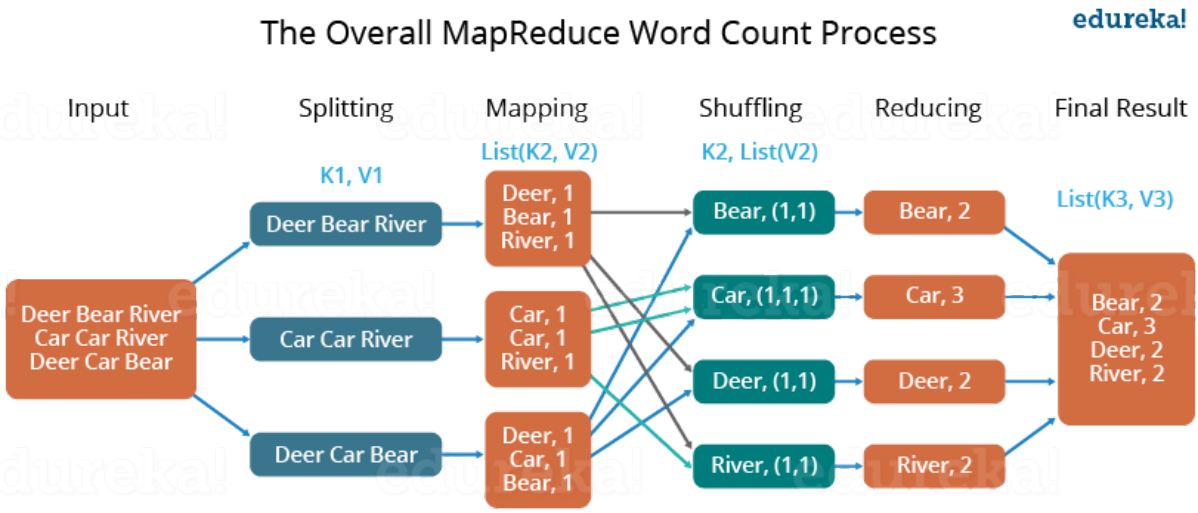
Chapter04. 맵리듀스 아키텍처

- 맵리듀스는 HDFS에 저장된 파일을 **분산 배치 분석**을 할 수 있게 도와주는 프레임워크.

4.1 맵리듀스의 개념

- 맵리듀스 프로그래밍 모델은 맵(Map)과 리듀스(Reduce)라는 두 가지 단계로 데이터를 처리
- 맵은 입력 파일을 한 줄씩 읽어서 데이터를 변형(transformation)하며, 리듀스는 맵의 처리 결과 데이터를 집계(aggregation).
- 맵의 데이터 변형 규칙은 개발자가 자유롭게 정의할 수 있으며, 한 줄에 하나의 데이터가 출력. 맵리듀스 프레임워크는 이러한 프로그래밍 모델을 구현할 수 있게 맵과 리듀스 인터페이스를 제공
- 예제) 입력 파일의 단어 개수를 계산하는 맵리듀스 프로그래밍 모델
    - 이러한 맵리듀스 프로그래밍 모델은 일반적으로 다음과 같은 함수로 표현됨
        - 맵 : (k1, v1) → list(k2, v2)
            - 맵의 입력값
                - k1 : 각 줄 번호, v2 : 각 줄 내용
            - 맵의 연산
                - 데이터를 입력받아 이를 가공하고 분류
            - 맵의 출력값
                - k2 : 단어, v2 : 단어의 개수

- 리듀스 : (k2, list(v2)) → (k3, list(v3))
  - 맵의 입력값
    - k2 : 새로운 키, list(v2) : 그루핑된 값의 목록
  - 맵의 연산
    - 값의 목록(list(v2))에 대한 집계 연산을 실행
  - 맵의 출력값
    - k3 : 새로운 키, list(v3) : 단어 개수의 합계

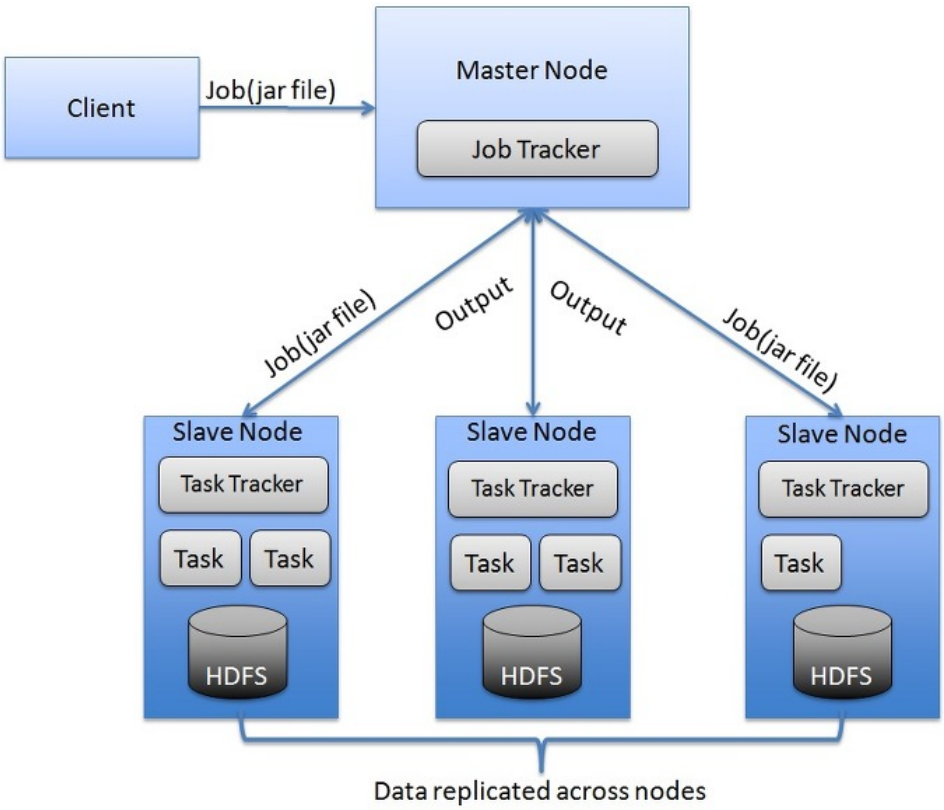


4.2 맵리듀스 아키텍처

- 맵리듀스 프레임워크는 개발자가 분석 로직을 구현하는 데 집중하게 해주고, 데이터에 대한 분산과 병렬 처리를 프레임워크가 담당.
- 맵리듀스 아키텍처를 알아야, 성능을 고려하여 개발할 수 있고, 정렬이나 병합 등 다양한 분석 코드를 구현할 수 있음. 또한 맵리듀스 환경설정 파일을 튜닝할 때도 도움을 받을 수 있음.

4.2.1 시스템 구성

- 맵리듀스 시스템은 클라이언트, 잡트래커, 태스크트래커로 구성됨



4.2.1.1 클라이언트

- 사용자가 실행한 맵리듀스 프로그램과 하둡에서 제공하는 맵리듀스 API를 의미. 사용자는 맵리듀스 API로 맵리듀스 프로그램을 개발하고, 개발한 프로그램을 하둡에서 실행할 수 있음

4.2.1.2 잡트래커

- 클라이언트가 하둡으로 실행을 요청하는 맵리듀스 프로그램은 잡(job)이라는 하나의 작업 단위로 관리됨. 잡트래커(Job Tracker)는 하둡 클러스터에 등록된 전체 잡의 스케줄링을 관리하고 모니터링. 전체 하둡 클러스터에서 하나의 잡트래커가 실행되며, 보통 하둡의 네임노드 서버에서 실행.
    - 하지만, 잡트래커를 반드시 네임노드 서버에서 실행할 필요는 없음. 페이스북은 잡트래커를 별개의 서버에서 동작하도록 구성
- 사용자가 새로운 잡을 요청하면 잡트래커는 잡을 처리하기 위해 몇 개의 맵과 리듀스를 실행할지 계산. 이렇게 계산된 맵과 리듀스를 어떤 태스크트래커에서 실행할지 결정하고, 해당 태스크트래커에 잡을 할당. 이때 태스크트래커는 잡트래커의 작업 수행 요청을 받아 맵리듀스 프로그램을 실행. 잡트래커와 태스크트래커는 하트비트라는 메서드로 네트워크 통신을 하면서 태스크트래커의 상태와 작업 실행 정보를 주고 받음. 만약 태스크트래커에 장애가 발생하면 잡트래커는 다른 대기 중인 태스크트래커를 찾아 태스크를 재실행

4.2.1.3 태스크트래커

- 태스크트래커(TaskTracker)는 사용자가 설정한 맵리듀스 프로그램을 실행하며, 하둡의 데이터노드에서 실행되는 데몬. 태스크트래커는 잡트래커의 작업을 요청받고, 잡트래커가 요청한 맵과 리듀스 개수만큼 맵 태스크(map task)와 리듀스 태스크(reduce task)를 생성.
- 맵 태스크와 리듀스 태스크가 생성되면 새로운 JVM을 구동해 맵 태스크와 리듀스 태스크를 실행. 이때 태스크를 실행하기 위한 JVM은 재사용할 수 있게 설정할 수 있음
- 서버가 부족해서 하나의 데이터노드를 구성했더라도 여러 개의 JVM을 실행해 데이터를 동시에 분석하므로 병렬 처리 작업에 문제가 없음.

4.2.2 데이터 플로우

- 웹 서버 접속 로그를 이용하여 URL별 접속 통계를 맵리듀스를 통해 계산

4.2.2.1 맵 단계

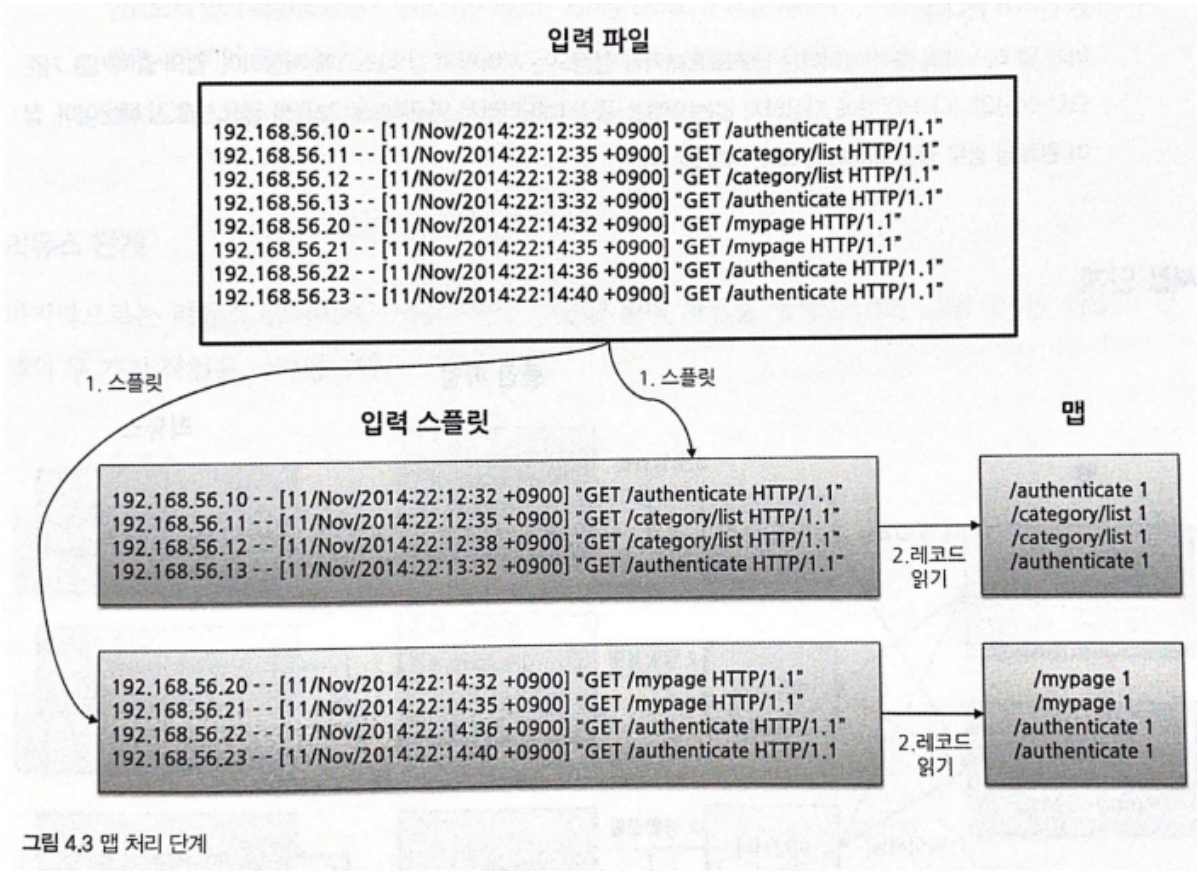


그림 4.3 맵 처리 단계

1. 스플릿

- 맵리듀스는 HDFS에 저장된 파일을 읽어서 배치 처리를 함. 이때 HDFS에 저장된 데이터는 대부분 큰 규모의 데이터임. 따라서 맵리듀스 프레임워크는 이러한 대용량 파일을 처리하기 위해 입력 데이터 파일을 입력 스플릿(InputSplit)이라는 고정된 크기의 조각으로 분리. 입력 스플릿별로 하나의 맵 태스크가 생성됨. 마지막으로 각 입력 스플릿은 맵 태스크의 입력 데이터로 전달
- 입력 스플릿은 기본적으로 HDFS 블록 크기를 기준으로 생성됨. 예를 들어, 입력 파일이 100MB이고, HDFS의 기본 블록 크기가 64MB라면 두 개의 입력 스플릿이 생성됨.

2. 레코드 읽기
    - 맵 태스크는 입력 스플릿의 데이터를 레코드 단위로, 즉 한 줄씩 읽어서 사용자가 정의한 맵 함수를 실행. 위 예제의 맵 함수는 하나의 레코드에 있는 URL과 각 URL의 개수를 출력. 웹 로그 파일에는 하나의 URL밖에 없기 때문에 URL별로 1만 출력되었음.
    - 맵 함수 : (줄 번호, 로그) → (URL, 건수)
    - 맵 태스크의 출력 데이터는 태스크트래커가 실행되는 서버의 로컬 디스크에 저장되며, 맵의 출력키를 기준으로 정렬됨. HDFS에 저장하지 않는 이유는 중간 데이터라서 영구적으로 보관할 필요가 없기 때문이며, 잡이 완료될 경우 중간데이터는 모두 삭제됨

4.2.2.2 셔플 단계

- 맵 태스크의 출력 데이터는 중간 데이터이며, 리듀스 테스크는 이 데이터를 내려받아 연산을 수행함. 맵리듀스 프레임워크는 이러한 작업이 진행될 수 있게 셔플(Shuffle)을 지원. 셔플은 맵 태스크의 출력 데이터가 리듀스 태스크에게 전달되는 일련의 과정

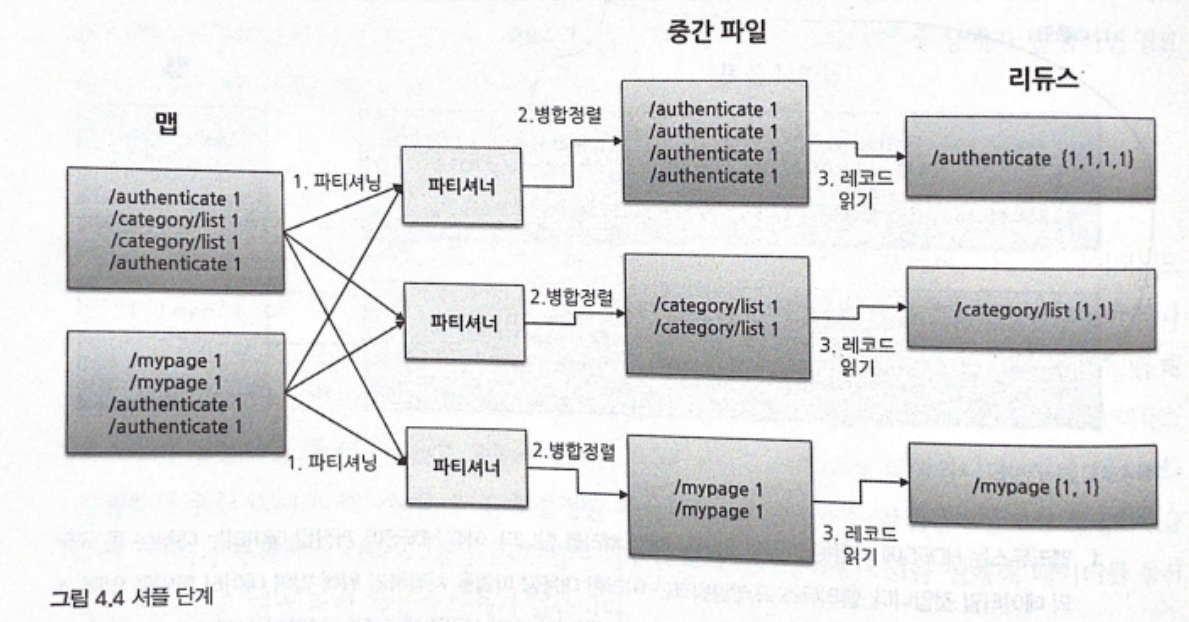


그림 4.4 셔플 단계

1. 파티셔닝
    - 파티셔너는 맵의 출력 레코드를 읽어서 출력키의 해시값을 구함. 각 해시값은 레코드가 속하는 파티션 번호로 사용됨. 즉, 출력키로 해시값을 구하고, 해시값을 파티션 번호로 사용해서 각 출력 레코드가 속할 파티션으로 보냄. 파티션은 실행될 리듀스 태스크 개수만큼 생성됨.
    - 예를 들어, 리듀스 태스크 개수가 2일 경우, 파티션은 두 개가 생성되고, 파티션 번호는 0과 1을 사용.
    - 맵리듀스가 제공하는 기본 파티셔너는 해시값으로 파티션 번호를 계산하지만, 사용자가 임의로 파티셔너를 개발해서 적용할 수 있음
2. 병합정렬
    - 파티셔닝된 맵의 출력 데이터는 네트워크를 통해 리듀스 태스크에 전달됨. 하지만 모든 맵의 출력이 동시에 완료되지 않기 때문에 리듀스 태스크는 자신이 처리할 데이터가 모일 때까지 대기.
    - 맵의 출력 데이터가 모두 모이면 리듀스 태스크는 데이터를 정렬하고, 하나의 입력 데이터로 병합.
3. 레코드 읽기
    - 리듀스 태스크는 병합된 데이터를 레코드 단위로 읽어들임

4.2.2.3 리듀스 단계

- 마지막으로 리듀스 단계는 사용자에게 전달할 출력 파일을 생성

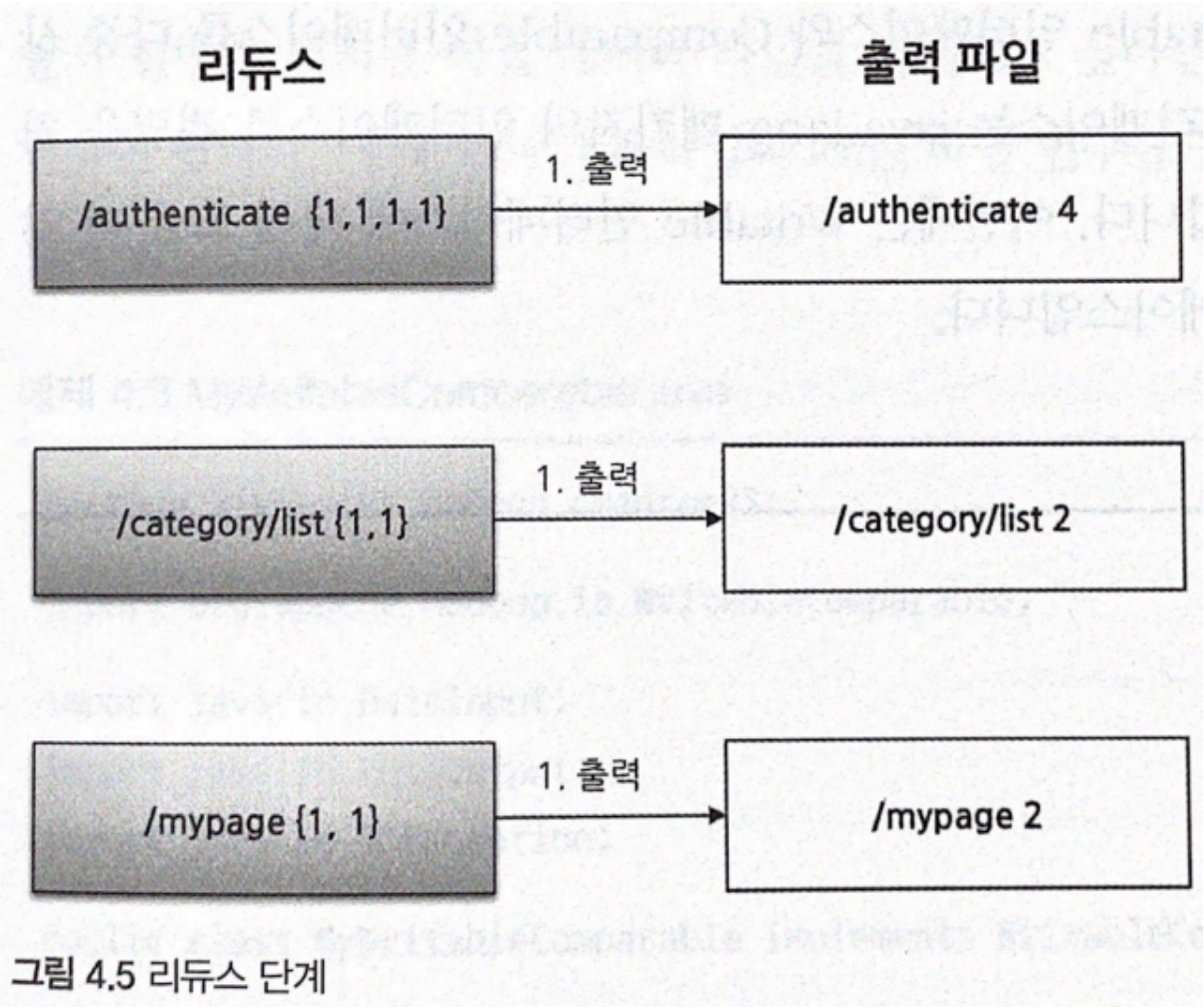


그림 4.5 리듀스 단계

1. 리듀스 태스크는 사용자가 정의한 리듀스 함수를 레코드 단위로 실행. 이때 리듀스 태스크가 읽어들이는 데이터는 입력키와 입력키에 해당하는 입력값의 목록으로 구성. 이 예제의 리듀스 함수는 입력키별로 입력값의 목록을 합산해서 출력. 이때 출력 데이터는 출력키와 출력키의 쌍으로 구성됨.
2. 리듀스 함수가 출력한 데이터는 HDFS에 저장됨. HDFS에는 리듀스 개수만큼 출력 파일이 생성되며, 파일명은 part-nnnnn으로 설정됨. 여기서 nnnnn은 파티션 번호를 의미하며, 00000부터 1씩 증가.