

# MapReduce: Simplified Data Processing on Large Clusters

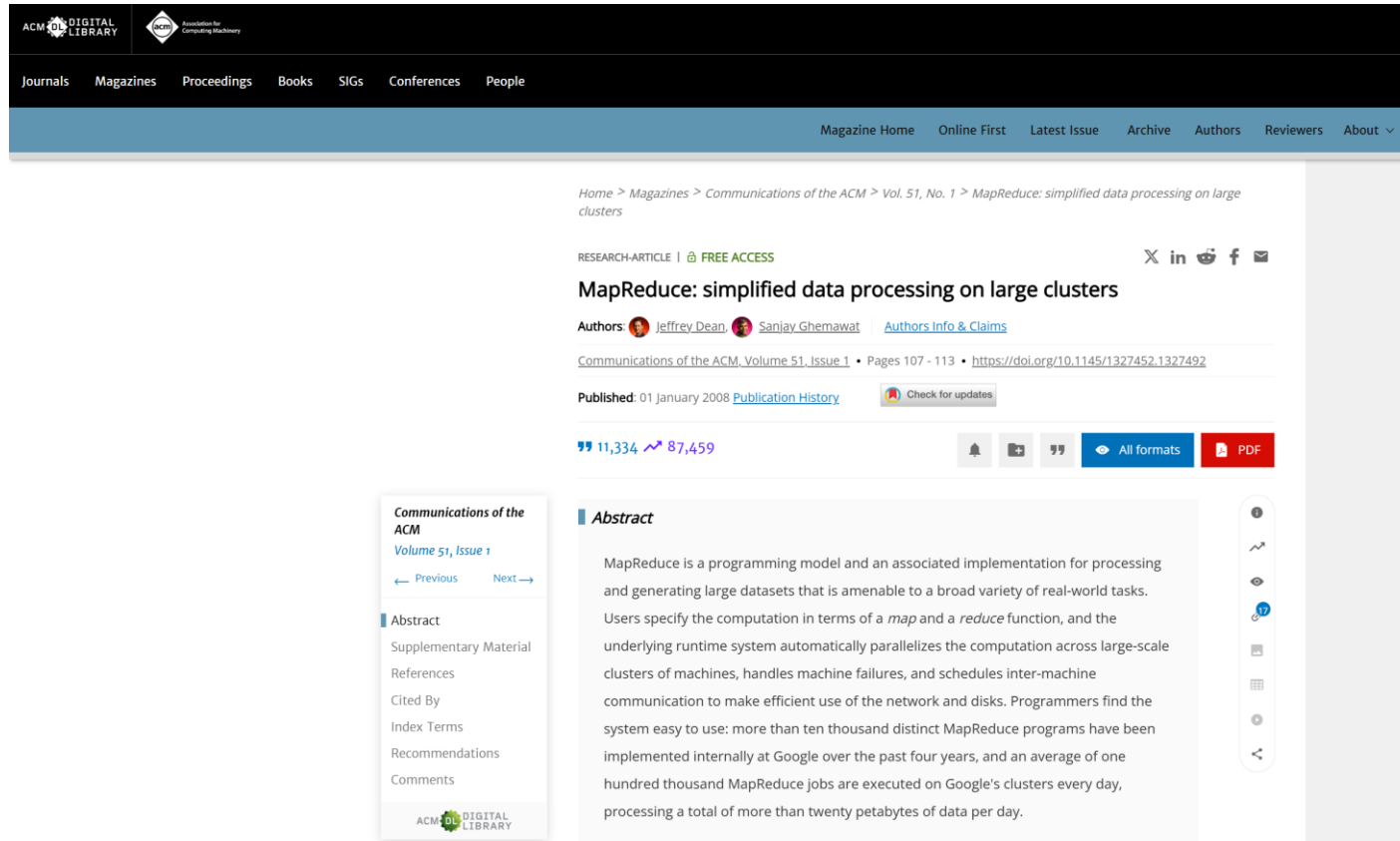
Jeffrey Dean<sup>†</sup> , Sanjay Ghemawat <sup>†</sup>

우도경

2025-07-14

# 0. paper

- MapReduce: Simplified Data Processing on Large Clusters
- CACM / 2008.01(Originally presented at OSDI 2004)
  - Affiliation: Google

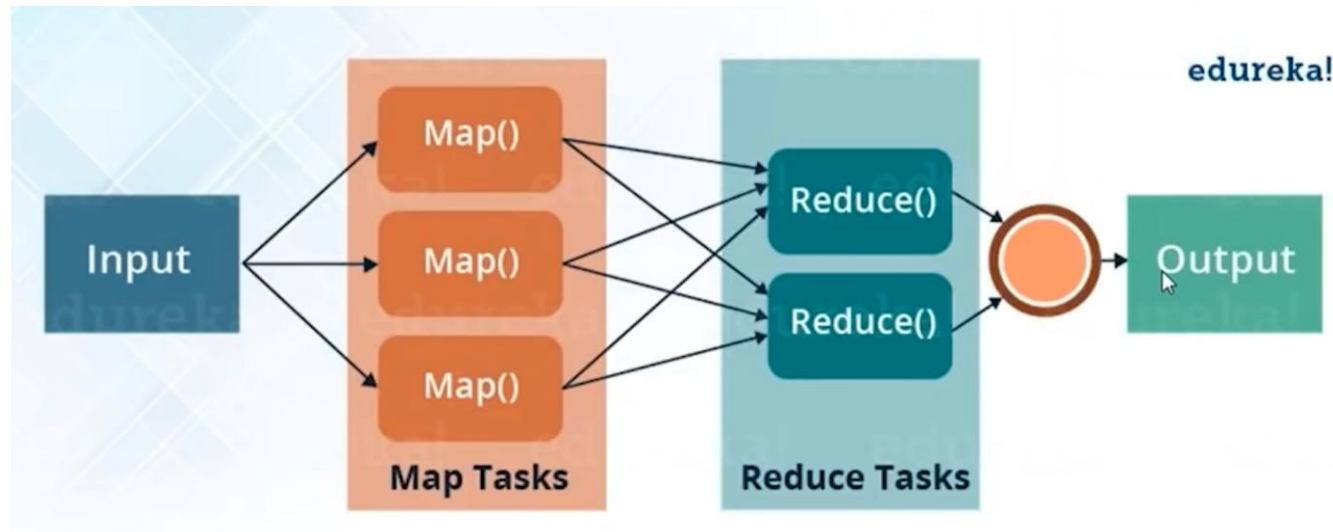


The screenshot shows the ACM Digital Library interface. At the top, there's a navigation bar with links for Journals, Magazines, Proceedings, Books, SIGs, Conferences, and People. Below this, a secondary bar contains links for Magazine Home, Online First, Latest Issue, Archive, Authors, Reviewers, and About. The main content area displays the paper title 'MapReduce: simplified data processing on large clusters' under the 'Communications of the ACM' journal. It lists the authors as Jeffrey Dean and Sanjay Ghemawat, and provides a link to the full text. The page also shows the publication date (01 January 2008) and a 'Check for updates' button. On the left side, there's a sidebar with a table of contents for the journal issue, including links to the Abstract, Supplementary Material, References, Cited By, Index Terms, Recommendations, and Comments. The abstract text is visible in the main content area, describing the MapReduce programming model and its implementation at Google.

# 1. Abstract

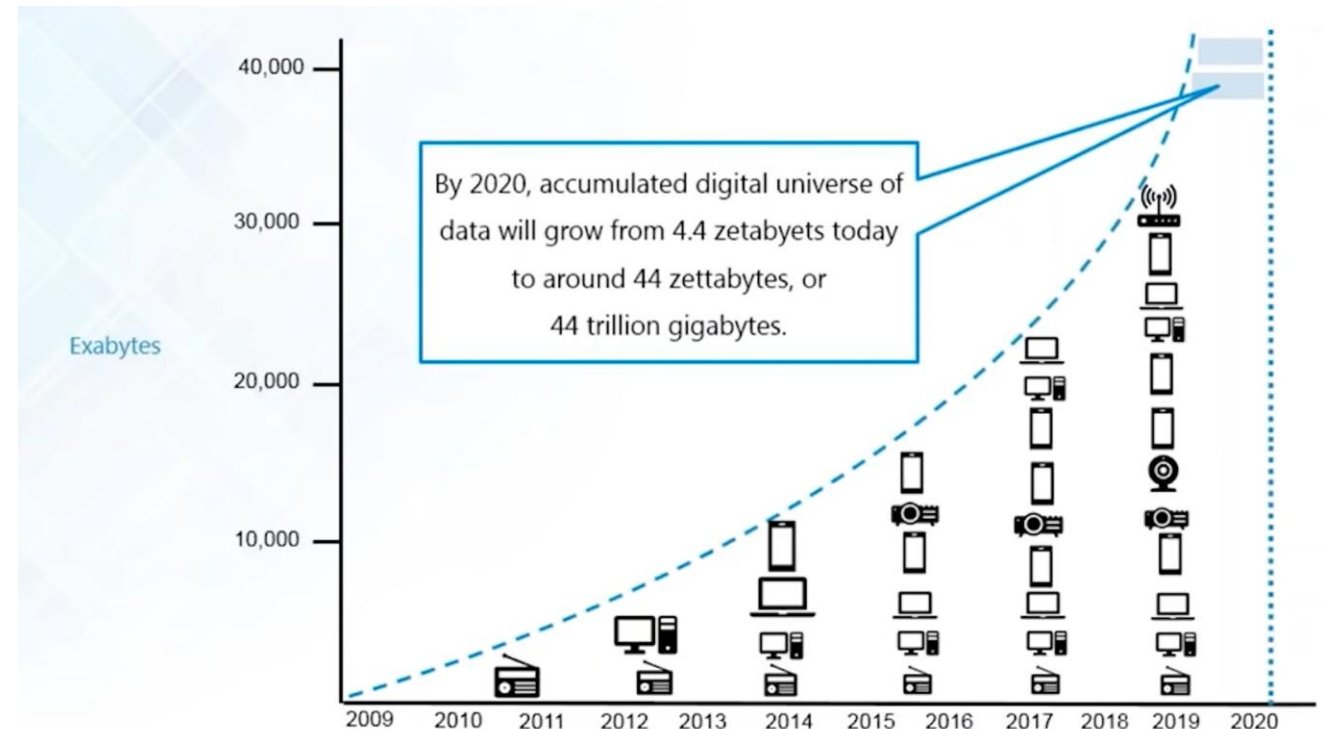
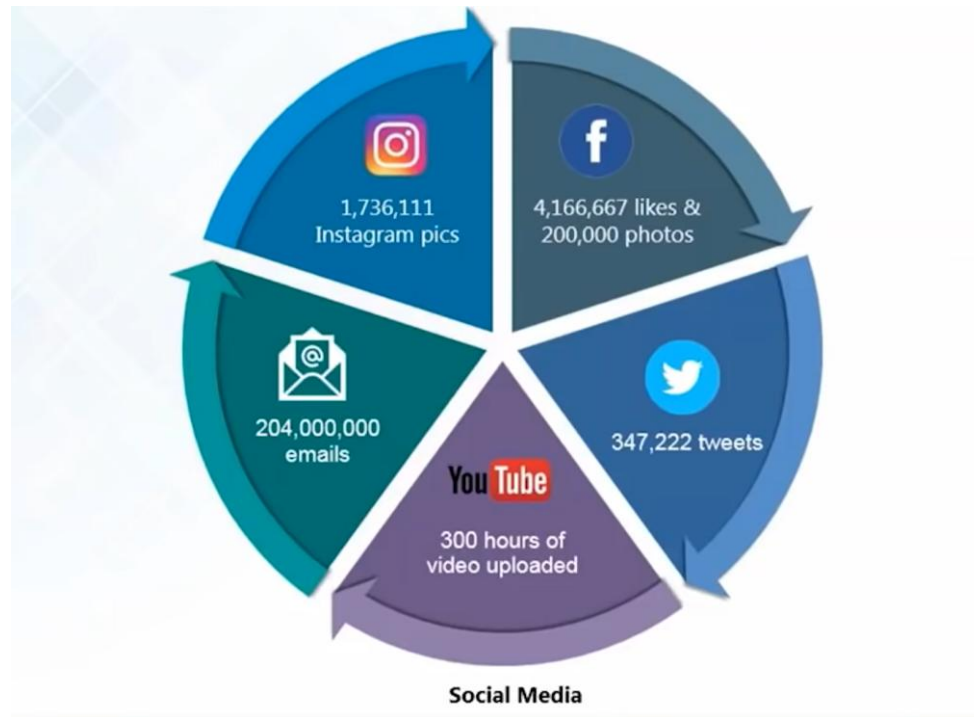
- MapReduce

- programming model and an associated implementation for processing and generating large datasets
- Users specify the computation in terms of a map and a reduce function
- the under-lying runtime system automatically parallelizes the computation across large-scale clusters of machines, handles machine failures, and schedules inter-machine communication



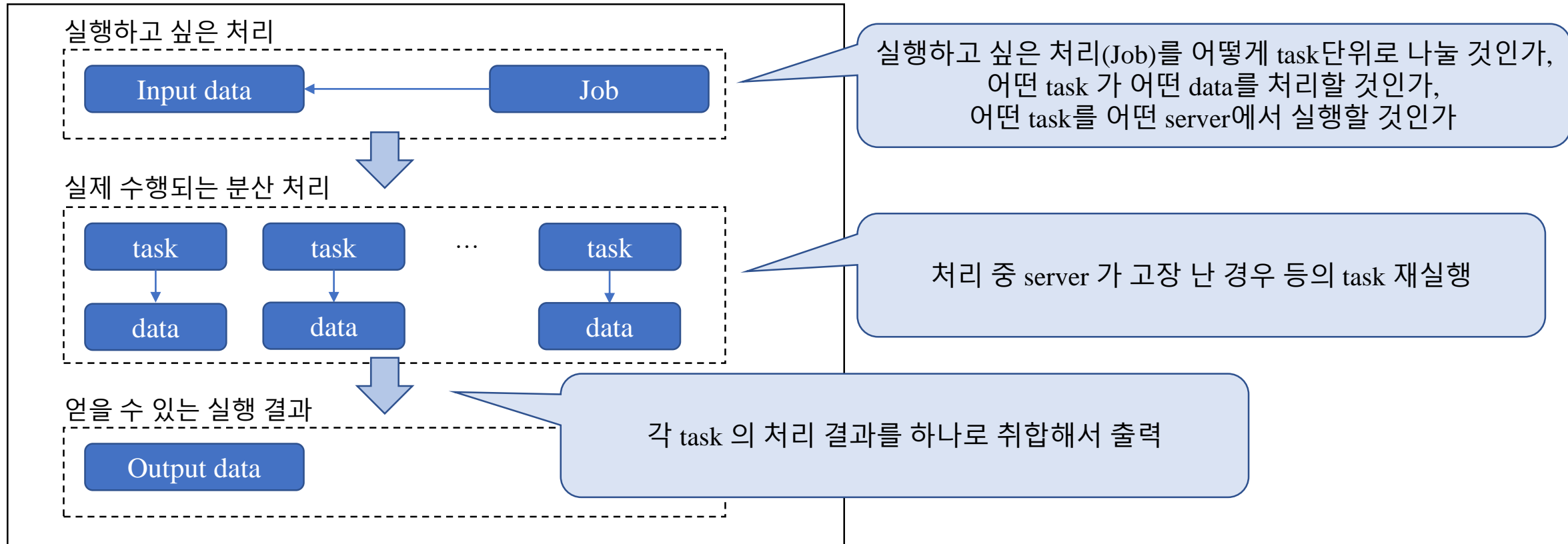
## 2. Introduction

- MapReduce 등장 이전
  - 구글은 크롤링된 문서, 웹 요청 로그 등과 같은 대량의 원시 데이터를 처리하여 크롤링된 페이지 수 요약, 특정 날짜에 가장 빈번한 쿼리 집합과 같은 다양한 종류의 파생 데이터를 계산하는 수백 가지의 특수 목적 계산 구현
  - 이러한 계산은 개념적으로 간단함
  - 그러나, 입력 데이터는 일반적으로 크고 계산은 합리적인 시간 내에 완료하기 위해 수백 또는 수천 대의 머신에 분산되어야 함



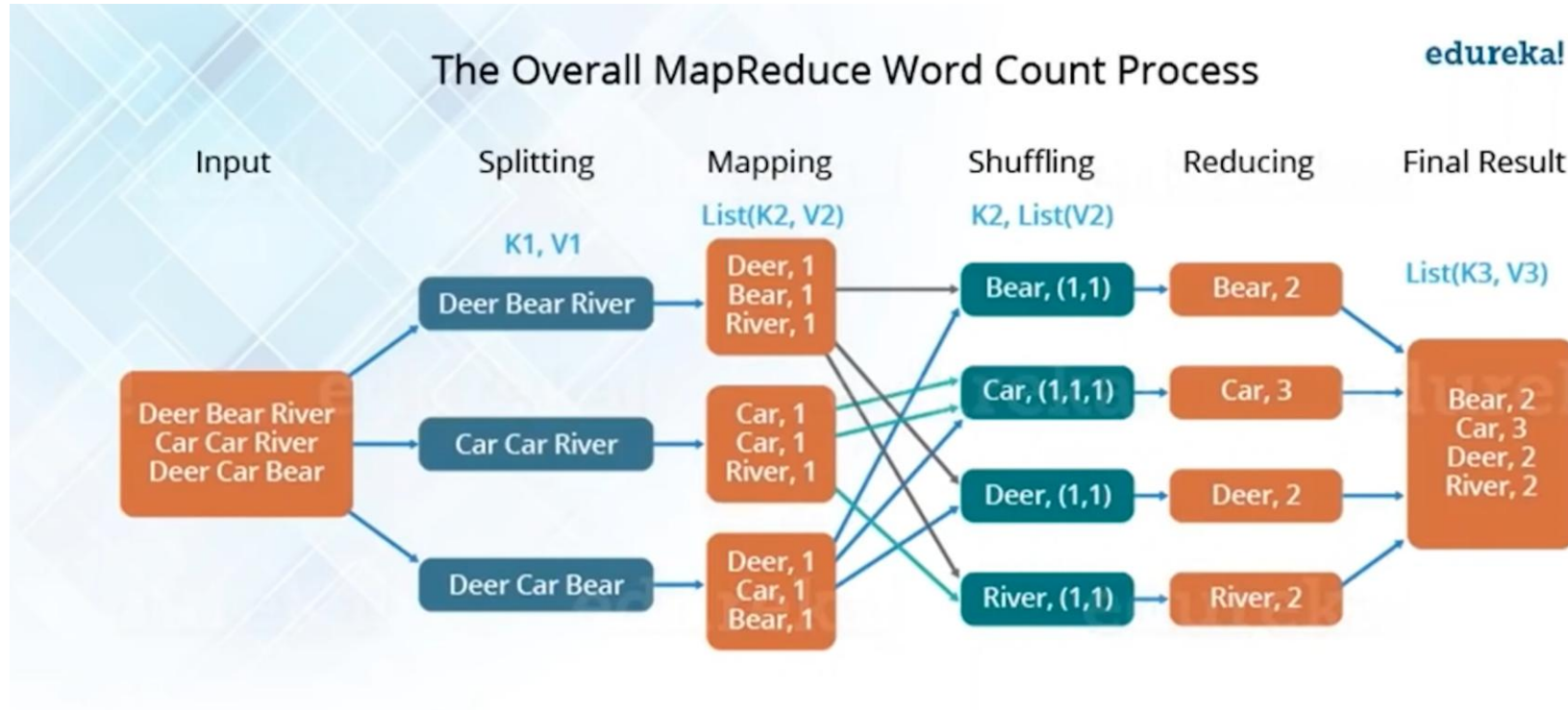
## 2. Introduction

- 병렬 분산 처리 시스템
  - 계산을 병렬화, 데이터 분산, 오류 처리하는 코드들을 직접 작성함.
  - 실제 연산 로직은 단순하지만, 그 외의 작업이 복잡성을 크게 증가시켰음.



## 2. Introduction

- MapReduce 처리 흐름
  - 대부분의 계산은 입력의 각 논리적 레코드에 map 연산을 적용한 다음, 파생된 데이터를 적절하게 결합하기 위해 동일한 키를 공유하는 모든 값에 reduce 연산을 적용하는 것과 관련됨
  - 사용자 지정 map 및 reduce 연산을 사용하는 함수형 모델을 사용하면 대규모 계산을 쉽게 병렬화할 수 있음
  - 병렬화, 내결함성, 데이터 분산 및 로드 밸런싱의 복잡한 세부 사항을 라이브러리에 숨기는 새로운 추상화 설계



### 3. Programming Model

- MapReduce 모델은 Lisp 및 기타 함수형 언어의 "map" 및 "reduce" 기본 요소에서 영감을 받음
  - **Map function** : takes an input pair and produces a set of intermediate key/value pairs
  - **Reduce function** : accepts an intermediate key I and a set of values for that key. It merges these values together to form a possibly smaller set of values

```
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));
```

- 예시: 단어 개수 세기 문서 모음에서 각 단어의 발생 횟수를 세는 문제의 경우:
  - Map 함수: (문서 이름, 문서 내용) 입력 쌍을 받아 문서 내용의 각 단어 w에 대해 (w, "1")을 출력
  - Reduce 함수: (단어, [1, 1, ..., 1]) 입력 쌍을 받아 주어진 단어에 대해 내보내진 모든 1을 합산하여 최종 단어 개수를 출력

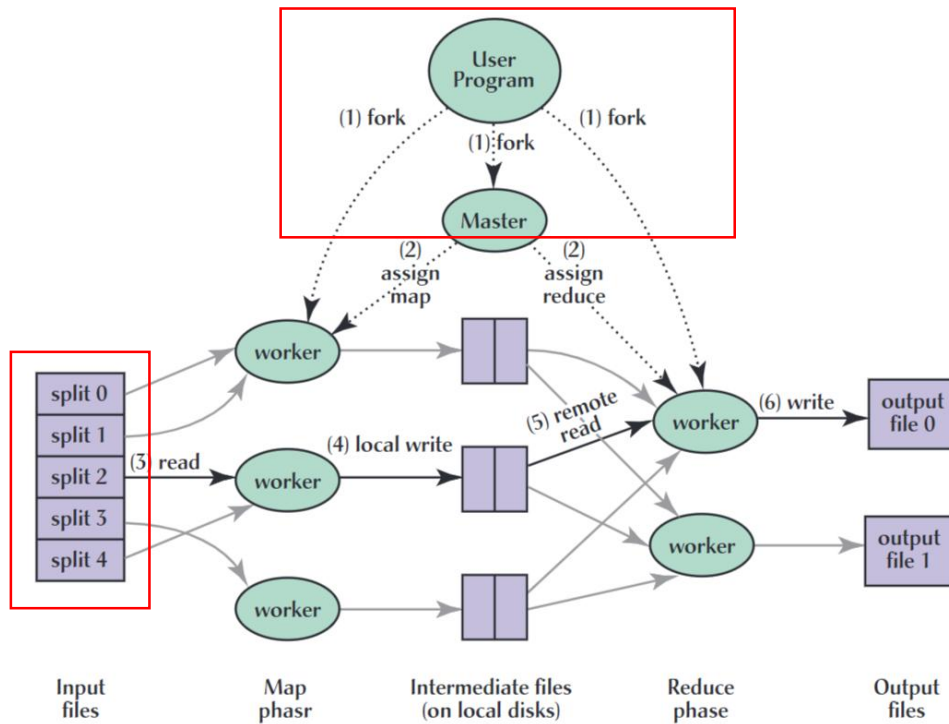
## 4. Implementation

- Google의 MapReduce 구현은 기가비트 이더넷으로 연결된 수천 대의 상용 PC로 구성된 대규모 클러스터를 대상으로 함
  - 듀얼 프로세서 x86 리눅스 머신 (4-8GB 메모리)
  - 개별 머신당 1Gbps 네트워크 대역폭
  - 수천 대의 머신으로 구성된 클러스터에서 기계 오류는 흔함
  - 저렴한 IDE 디스크를 사용하여 로컬에 스토리지 제공
  - GFS(Google File System)를 사용하여 분산 데이터 관리 및 복제를 통한 신뢰성 제공



## 4.1 Execution Overview

- (1) 입력 분할 및 프로그램 실행

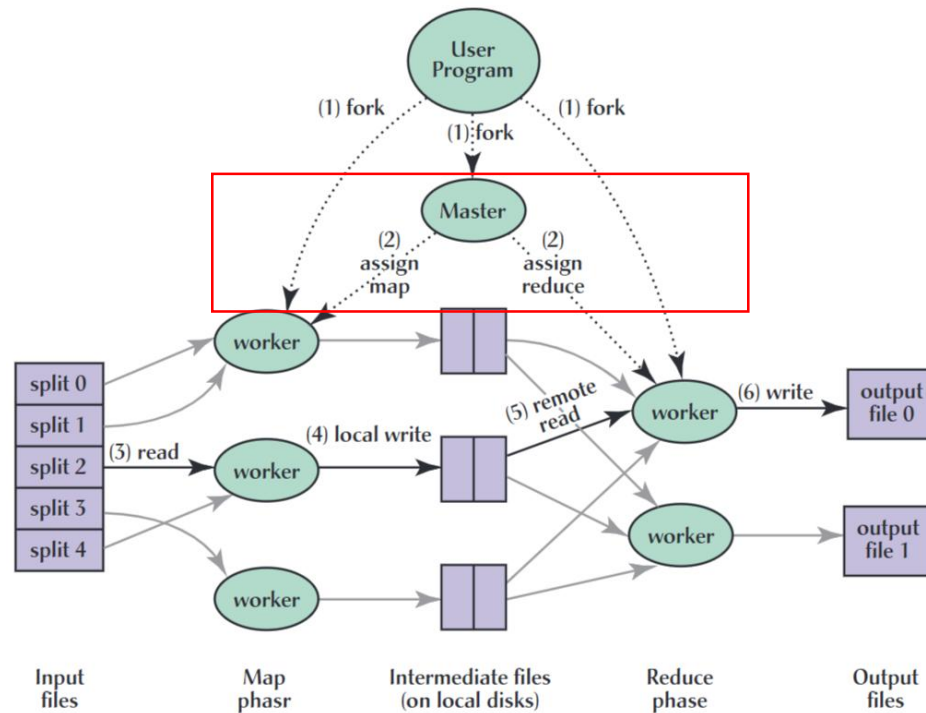


1. MapReduce 라이브러리는 먼저 입력 파일을 일반적으로 조각 당 16-64MB의 M개 조각으로 분할
2. 머신 클러스터에서 프로그램의 여러 복사본 시작

## 4.1 Execution Overview

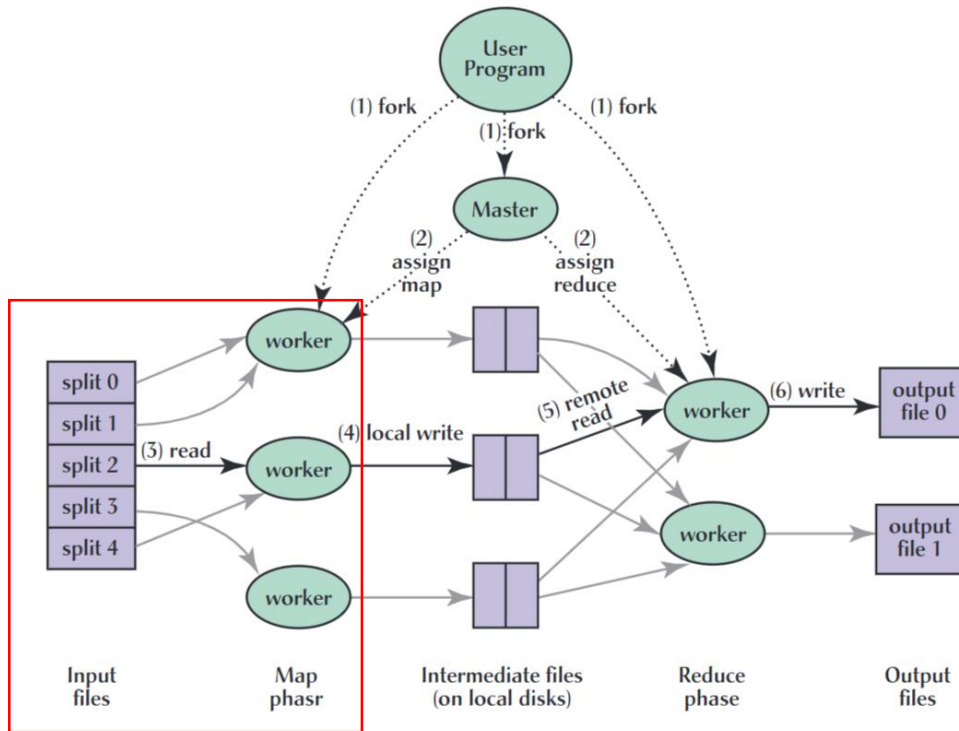
- (2) 마스터 및 작업자 역할 정의 및 태스크 할당

1. 프로그램의 복사본 중 하나인 Master는 작업을 할당하는 worker임
2. 할당할 M개의 map 작업과 R개의 reduce 작업 존재
3. Master는 idle workers 각각에게 map 작업 또는 reduce 작업을 할당



## 4.1 Execution Overview

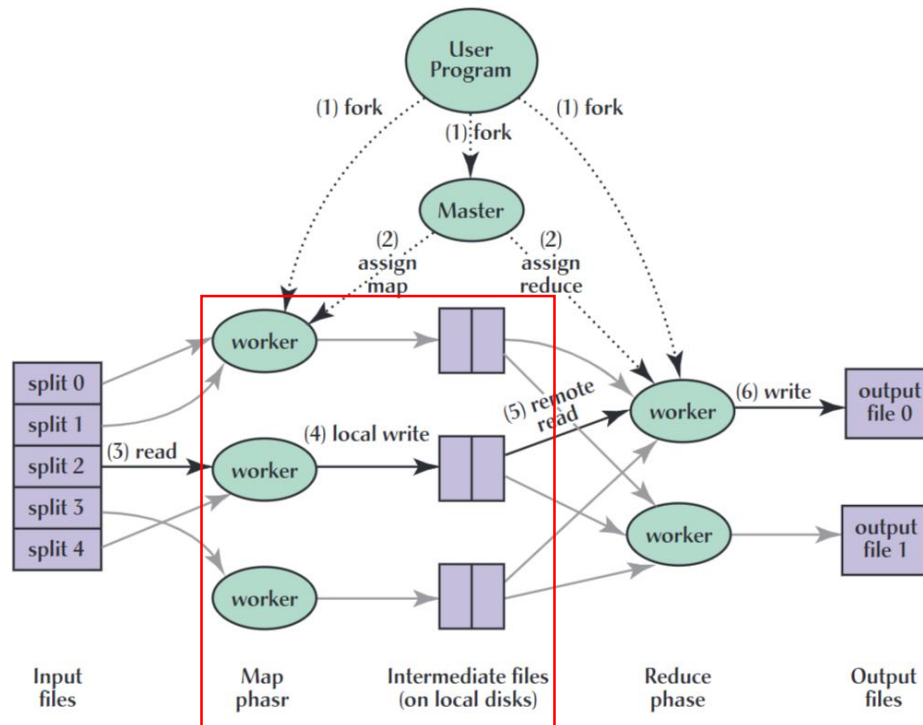
- (3) Map task 실행 및 중간 데이터 생성



1. Map 작업이 할당된 worker는 해당 입력 split의 내용을 읽음
2. 입력데이터에서 key/value 쌍을 구문 분석하고 각 쌍을 사용자 정의 map 함수에 전달
3. Map 함수에서 생성된 중간 key/value 쌍은 메모리에 buffering

## 4.1 Execution Overview

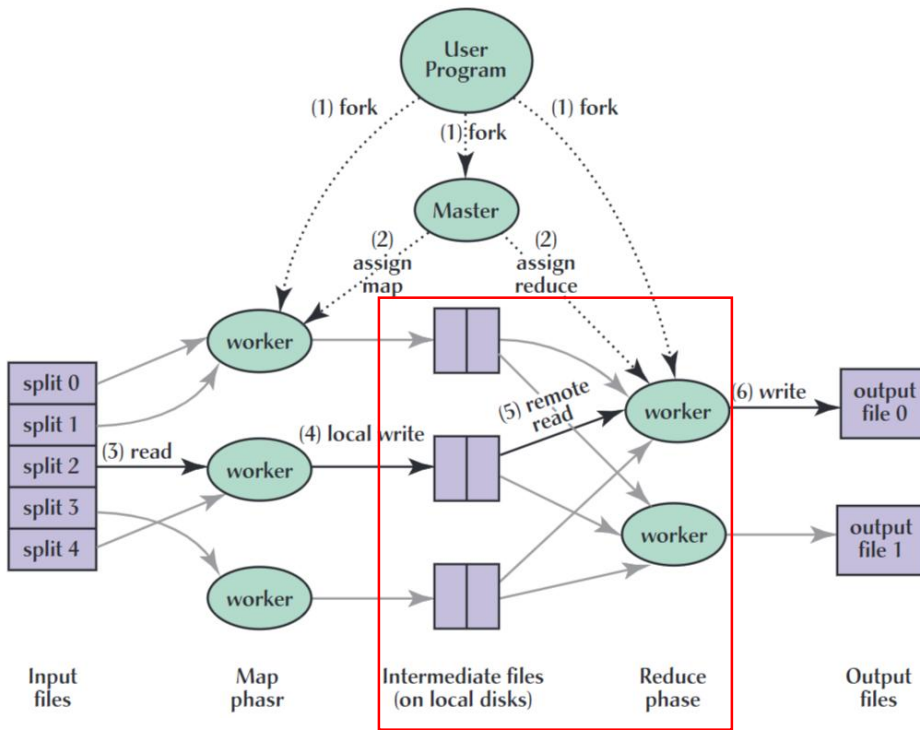
- (4) 중간 데이터 로컬 디스크 저장 및 위치 보고



- 주기적으로 buffering된 쌍은 partitioning function에 의해 R개의 영역으로 분할되어 로컬 디스크에 기록됨
- 로컬 디스크에 있는 이러한 buffering된 쌍의 위치는 Master에게 전달되며, Master는 이러한 위치를 reduce 작업자에게 전달함

## 4.1 Execution Overview

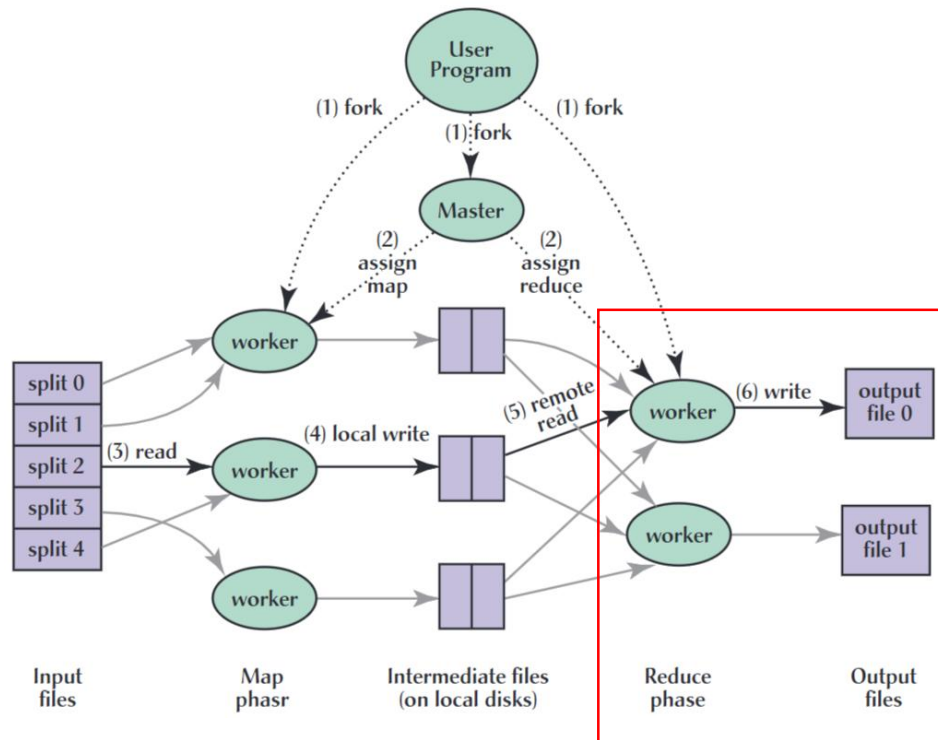
- (5) Reduce task를 위한 데이터 정렬



1. Reduce worker가 Master로부터 위치에 대한 알리를 받으면 remote procedure call을 사용하여 map worker의 로컬 디스크에서 buffering 된 데이터를 읽음
2. Reduce worker가 해당 partition에 대한 모든 중간 데이터를 읽으면 동일한 키 값들이 함께 grouping 되도록 중간 키별로 정렬

## 4.1 Execution Overview

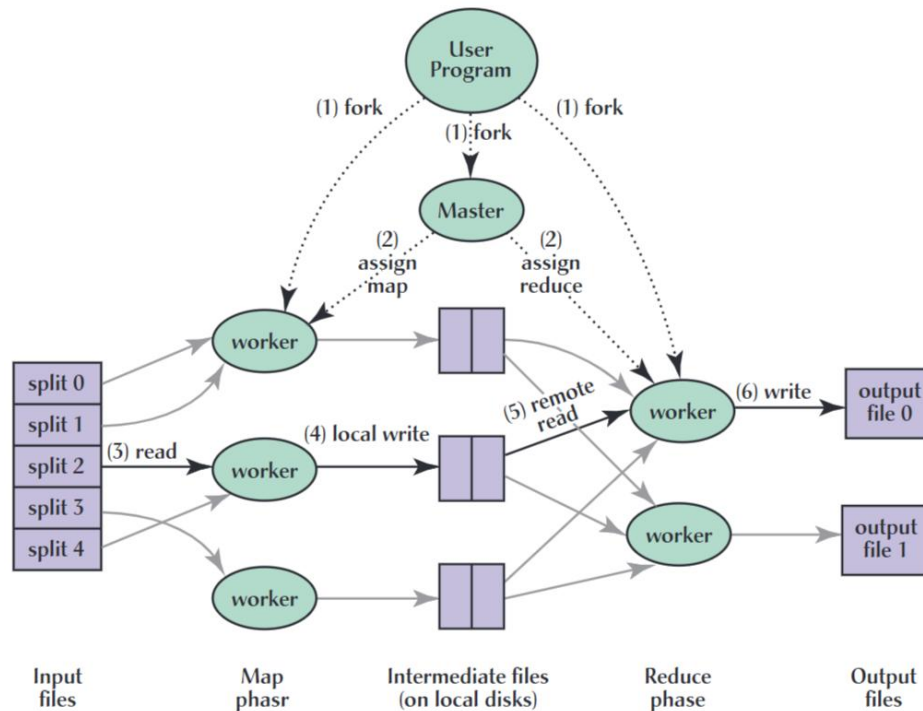
- (6) Reduce 함수 실행 및 최종 결과 파일 생성



1. Reduce worker는 정렬된 중간 데이터를 순회하고, 발견된 각 고유한 중간 키에 대해 키와 해당 중간 값 집합을 사용자 정의 reduce 함수에 전달
2. Reduce 함수의 출력은 이 reduce partition에 대한 최종 출력 파일에 추가됨

## 4.1 Execution Overview

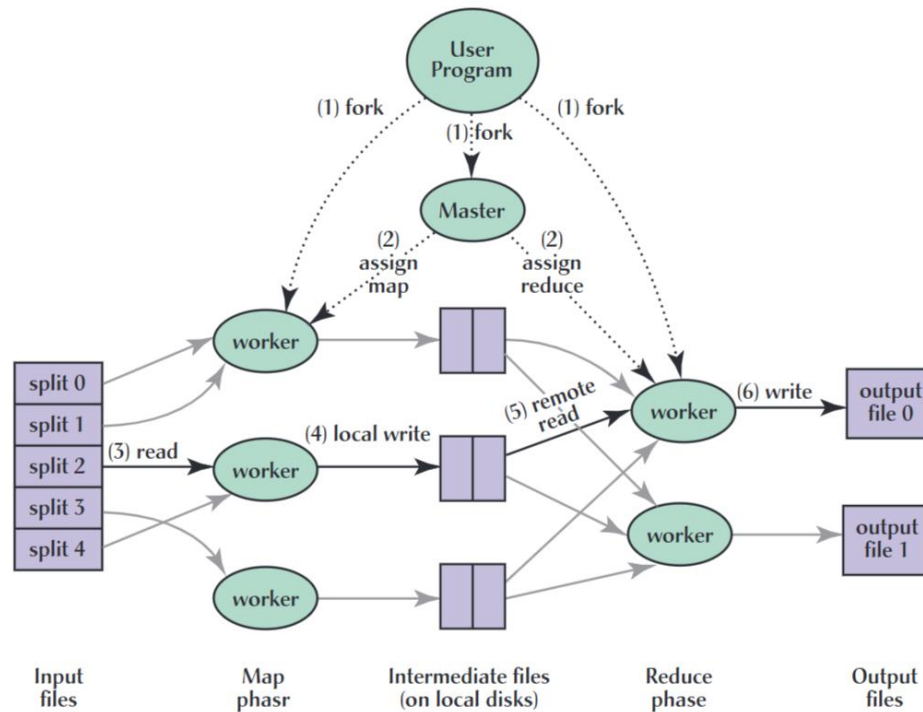
- (7) MapReduce 작업 완료 및 결과 반환



1. 모든 map task와 reduce task가 완료되면 master 는 사용자 프로그램을 깨움
2. 성공적으로 완료되면 MapReduce 의 실행결과는 R개의 output file에서 사용 가능
3. 사용자는 이 file을 다른 MapReduce 호출에 대한 입력으로 전달하거나 여러 파일로 분할된 입력을 처리할 수 있는 다른 분산 애플리케이션에서 사용

## 4.2 Master Data Structures

- Master 데이터 구조

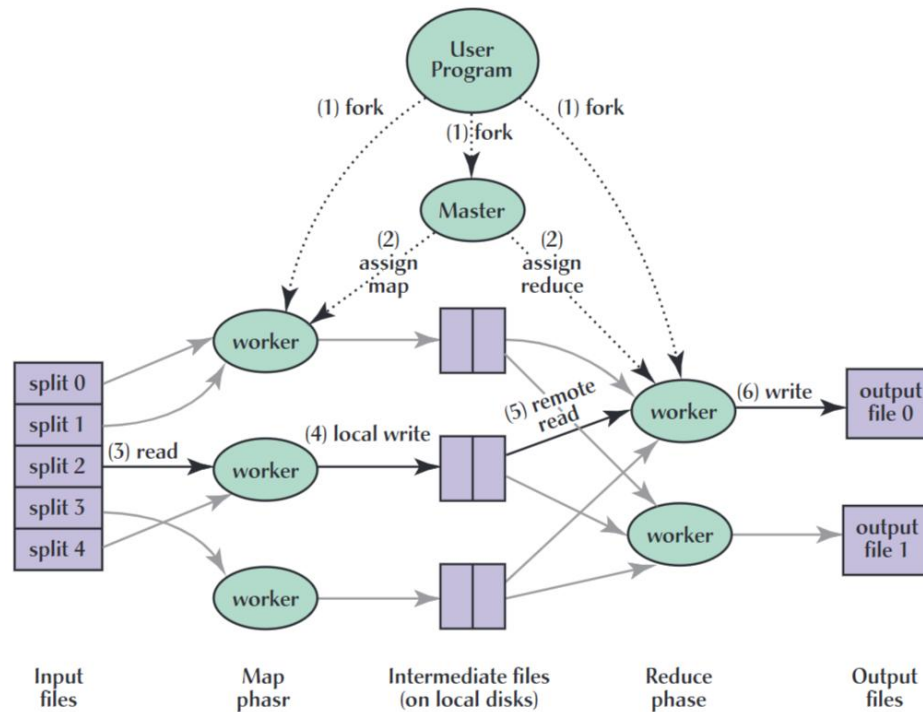


1. 각 map task와 reduce task에 대해 상태(idle, in-progress 또는 completed)와 worker machine 의 ID(nonidle tasks의 경우)
2. 완료된 각 map task에 대해 master는 map task에서 생성된 R개의 중간 파일 영역의 위치와 크기 저장



## 4.3 Fault Tolerance

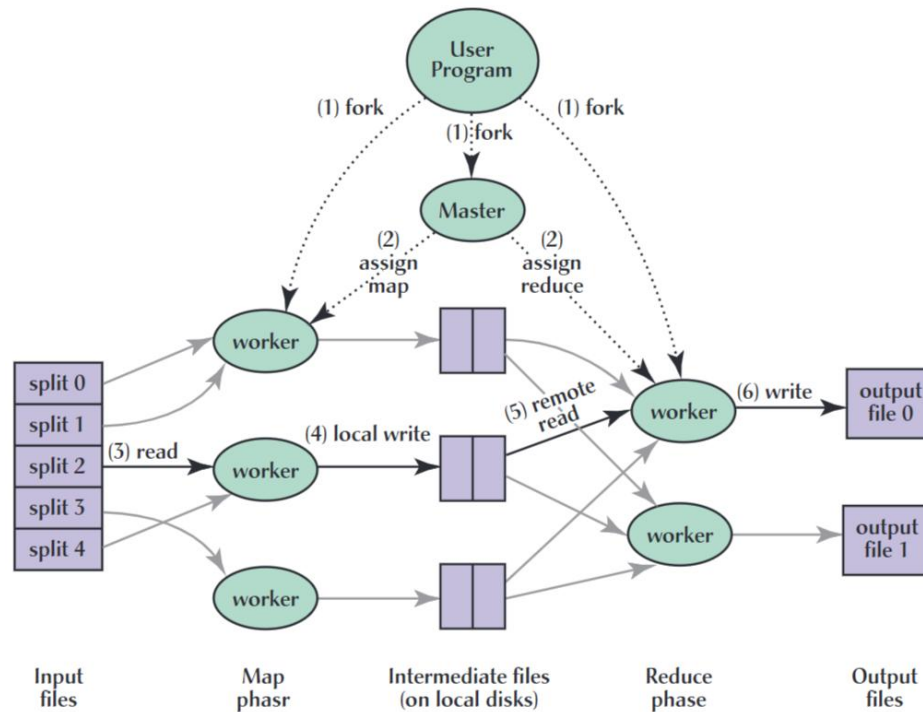
- Handling Worker Failures



1. Master는 모든 worker를 주기적으로 ping을 보냄
2. 특정 시간동안 worker로부터 응답이 없으면 master는 해당 worker를 실패한 것으로 표시
3. 실패한 worker에서 진행중인 모든 map task 또는 reduce task는 유휴상태로 재설정되고 다시 스케줄링 가능
4. MapReduce는 대규모 worker 오류에 탄력적

## 4.3 Fault Tolerance

- Semantics in the Presence of Failures



1. 사용자가 제공한 map 및 reduce 연산자가 입력 값의 결정적 함수인 경우, 분산 구현은 전체 프로그램의 오류가 없는 순차적 실행에 의해 생성되었을 결과와 동일한 출력 생성
2. Map 작업이 완료되면 작업자는 master에게 message를 보내고 이 message에는 R개의 임시 이름이 포함되어 있음
3. reduce 작업이 완료되면 reduce worker는 임시 출력 파일의 이름을 최종 출력 파일로 원자적으로 변경

## 4.4 Locality

- 네트워크 대역폭은 컴퓨팅 환경에서 비교적 부족한 리소스
- 입력데이터가 클러스터를 구성하는 시스템의 로컬 디스크에 저장된다는 점을 활용해 네트워크 대역폭 절약
- Master는 입력 파일의 위치 정보를 고려해 해당 입력 데이터의 복제본을 포함하는 시스템에서 map task를 예약하려고 시도

## 4.5 Task Granularity

- map 단계를 M개 조각으로, reduce 단계를 R개 조각으로 세분화
- 이상적으로, M과 R은 작업자 시스템 수보다 훨씬 커야 함
  - 각 작업자가 여러 개의 서로 다른 작업을 수행하면 동적 로드 밸런싱이 향상
  - 작업자가 실패할 때, 완료되지 않은 map 작업을 다른 모든 worker 시스템에 분산 가능

## 4.6 Backup Tasks

- 계산에서 마지막 몇개의 map 또는 reduce 작업 중 하나를 완료하는데 비정상적으로 오랜 시간이 걸림
- MapReduce 작업이 완료에 가까워지면 Master는 나머지 진행중인 작업의 백업 실행 예약

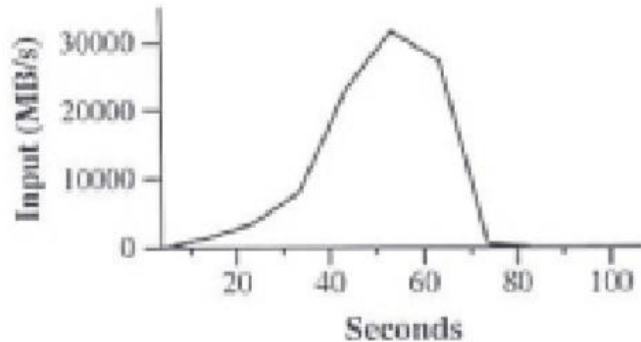
## 5. Performance

- 대규모 머신 클러스터에서 실행되는 두 가지 연산에 대한 MapReduce의 성능을 측정
  - 하나의 연산은 특정 패턴을 찾기 위해 약 1테라바이트의 데이터를 검색
  - 다른 연산은 약 1테라바이트의 데이터를 정렬
- 클러스터 구성
  - 약 1800대의 머신으로 구성된 클러스터에서 실행
  - 각 머신은 Hyper-Threading이 활성화된 2GHz 인텔 제온 프로세서 2개, 4GB 메모리, 160GB IDE 디스크 2개, 그리고 1Gbps 이더넷 링크를 갖추고 있음
  - 머신들은 약 100~200Gbps의 총 대역폭이 제공되는 2단계 트리 구조의 스위칭 네트워크로 구성

\* Hyper-Threading : 하나의 물리적인 CPU 코어에서 두 개의 스레드를 동시에 실행할 수 있도록 해주는 기술

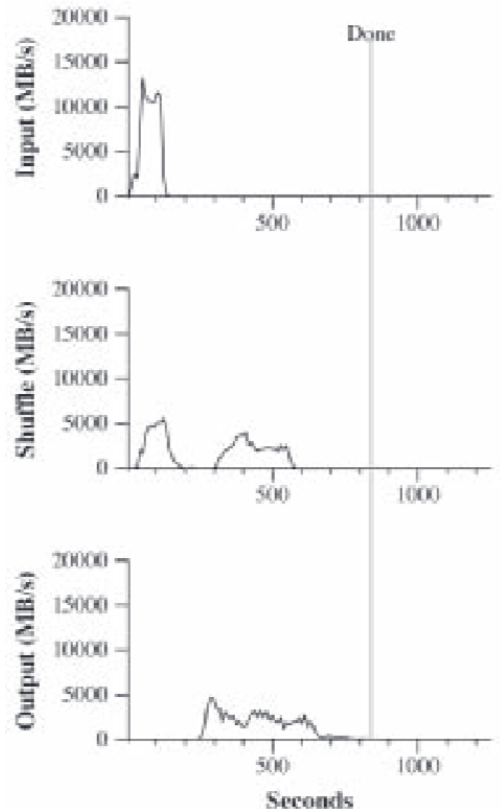
## 5. 1 Grep

- $10^{10}$ 개의 100byte의 레코드를 스캔하여 비교적 드문 세글자 패턴 검색 (패턴은 92,337 개의 레코드에서 발생)
- 입력은 약 64MB의 조각으로 분할되고, 전체 출력은 하나의 파일에 배치됨
- 시간에 따른 연산 진행상황 그래프
  - MapReduce 연산이 할당됨에 따라 속도가 점차 증가
  - 1,764 개의 작업자가 할당되었을 때, 30GB/s로 정점에 달함



## 5. 2 Sort

- $10^{10}$ 개의 100byte의 레코드(약 1 Tera byte)를 정렬
- 최종 정렬된 출력은 2-way 복제되어 2Tera byte가 기록됨
- 입력은 약 64MB 조각으로 분할되고, 정렬된 출력을 4000개의 파일로 분할함
- 시간에 따른 연산 진행상황 그래프

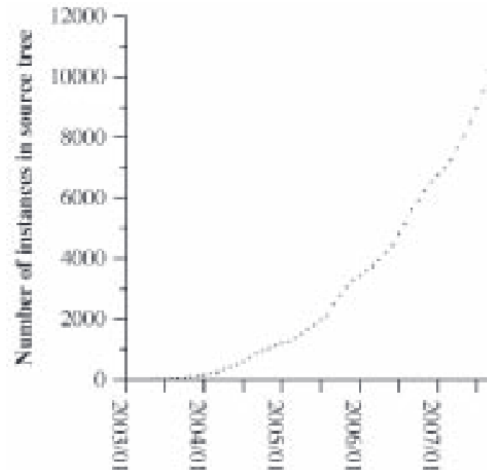


- Input : 약 13GB/s에서 정점에 달함. 입력 속도가 grep보다 낮음
- 정렬 map 작업이 로컬 디스크에 중간 출력을 쓰는데 시간과 I/O 대역폭 소모
- 속도 : Input > Shuffle > Output
- Locality로 인해 Input의 속도가 가장 빠름
- Output 단계에서는 안정성 및 가용성 위해 출력의 두 복제본을 만드므로 속도 가장 느림



## 6. Experience

- Usages
  - large-scale machine learning problems
  - clustering problems for the Google News and Froogle products
  - extracting data to produce reports of popular queries
  - extracting properties of Web pages for new experiments and products
  - processing of satellite imagery data
  - language model processing for statistical machine translation
  - large-scale graph computations



- 2003년 초 0개에서 2004년 9월 거의 900개, 2006년 3월 약 4000개로 시간이 지남에 따라 기본 소스 코드 관리 시스템에 체크인된 개별 MapReduce 프로그램의 수가 크게 증가

## 6.1 Large-Scale Indexing

- most significant uses of MapReduce : rewrite of the production indexing system
  - The indexing code is simpler, smaller, and easier to understand because the code that deals with fault tolerance, distribution, and parallelization is hidden within the MapReduce library.
  - The performance of the MapReduce library is good enough that we can keep conceptually unrelated computations separate instead of mixing them together to avoid extra passes over the data. This makes it easy to change the indexing process.

# Conclusions

- the model is easy to use, even for programmers without experience with parallel and distributed systems
- a large variety of problems are easily expressible as MapReduce computations.
- we have developed an implementation of MapReduce that scales to large clusters of machines comprising thousands of machines

Thank You