터벅터벅터벅…
우리 또 왔어☆

알파팀

아니

김예찬
김준서
박시언
서희나
이지윤
조웅빈

0 Intro

콘솔 창이 팡팡파라바라팡팡팡

좋은 컴퓨터 *(Good Computer)*

짱 비싸고 짱 좋은 컴퓨터를 사야지

그리고 짱 센 데이터들을 모두 돌려 버릴거야

난 있잖아 ~
통계가 세상에서 제일 싫어♫

THANK YOU!

알파카는 축제를 즐기러
유유히 떠났다…☆
- The End -

난 있잖아 ~
통계가 세상에서 제일 싫어♫

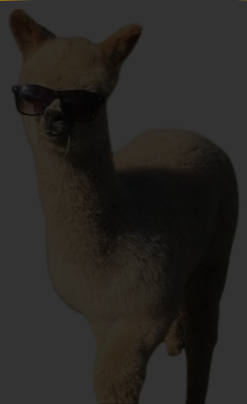
대용량 데이터 처리

THANK YOU!

① 대용량 데이터는 무엇인지

② 어떻게 **빠르고 효율적**으로 처리할 수 있는지

③ 이를 기반으로 학습하는 **모델**은 무엇이 있는지



알파카는 축제를 즐기러
유유히 떠났다…☆
- The End -

INDEX

1

대용량 데이터

1 대용량 데이터

대용량 데이터

기존의 관리 시스템/데이터 처리 도구 등으로 처리하기 어려운 규모의 데이터의 집합

→ **다양성 · 복잡성**으로 인해 '빅'데이터로 정의됨

테라바이트(TB)이상, **실시간으로 수집**되는 데이터도 존재 (소셜 미디어, 주가, 센서 등등)

대용량 데이터

전통적인 데이터 과학

작은 용량의 데이터를 가지고
파이썬, R과 같은 언어로 기계 학습 모델 개발

데이터의 양이 방대 → 단일 컴퓨터 또는 서버에서 처리하기에는 한계 존재

실시간으로 생성되는 데이터 – 신속한 반응 및 데이터 처리가 가능하지 않음

2

컴퓨터의 구조

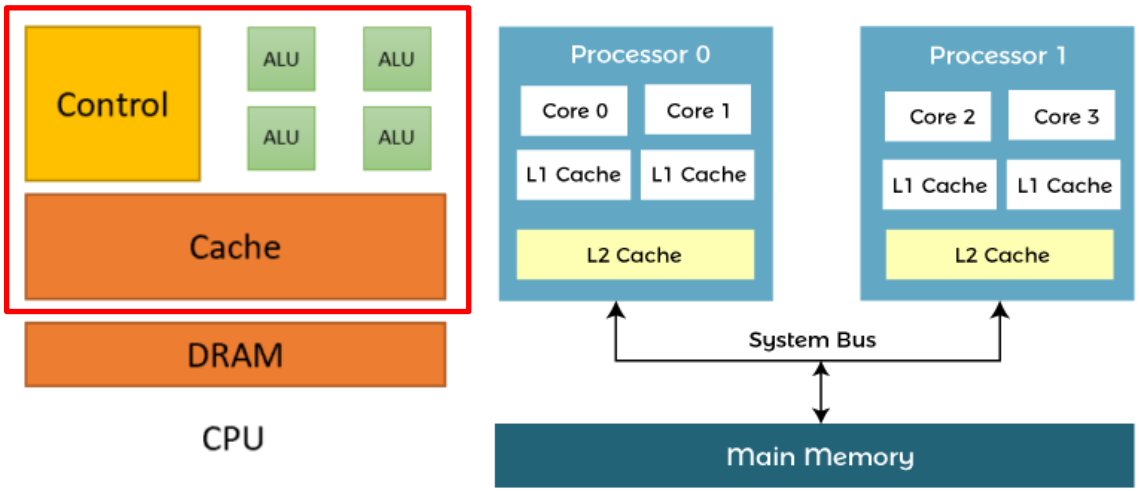
CPU (Central Processing Unit)

CPU *(Central Processing Unit)*

컴퓨터의 모든 프로그램 및 작업을 **실행**하고 **제어**하는 중앙 처리 장치

명령어를 해독하고 실행, 메모리에서 데이터를 읽고, 쓰는 작업을 수행

- 다른 하드웨어를 제어하는 역할 역시 진행



2 컴퓨터의 구조

CPU (Central Processing Unit)

CPU의 특징

① 순차적인 일처리 : 스케줄링에 따라서 우선 순위 부여, 다음 행동 결정

② 작은 규모 : 개별 코어가 각자의 일 담당 → **코어의 개별 성능 중시**

③ ALU(산술 논리 연산 장치) 구조 복잡 → **복잡한 일을 담당**

- 하나의 명령어로 처리하는 기능 多
- 각종 제어 처리 위한 부분 多
- 스케줄링 : 복잡한 분석 필요 → CPU 처리

** But, 코어 하나의 성능이 뛰어나다고 해서 전체 CPU의 성능이 항상 더 높지는 않음*

다른 하드웨어를 **제어**하며, 명령어가 입력되는 순서대로 처리하는 **직렬 처리방식**에 특화

적은 양의 데이터를 매우 빠르게 처리 → 컴퓨터 시스템의 성능과 속도를 좌우

2 컴퓨터의 구조

CPU의 연산

연산 단계

① 인출단계
(Fetch)

CPU : 메모리에서 다음 실행할 명령어를 인출
메모리: 명령어 주소를 전달

② 해독 단계
(Decode)

명령어를 해독하여 명령어가
어떤 동작을 수행해야 하는지 결정

③ 실행 단계
(Execute)

해독된 명령어를 기반으로 작업을 수행
추가적인 데이터를 메모리에서 읽어와서 사용

2 컴퓨터의 구조

CPU의 연산

연산 방법

반복문

① 하나씩 데이터를 가져와서 처리

② 데이터의 양이 많아질수록 시간이 오래 걸림

③ 여러 번의 메모리 접근이 필요

→ CPU 캐시와 메모리 간의 데이터 전송에 따른 오버헤드 발생 가능

벡터연산

벡터 연산 유닛이라는 하드웨어를 사용

→ 동시에 여러 데이터를 처리

→ 데이터를 병렬적으로 처리할 수 있음

대용량 데이터일 경우

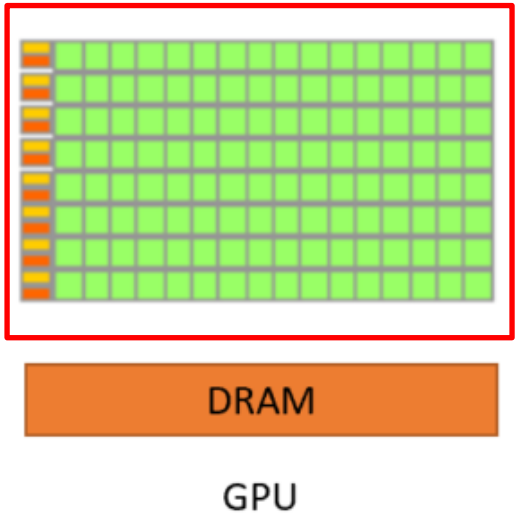
벡터 연산이 여러 데이터를 한번에 처리 → 훨씬 빠른 속도로 처리 가능

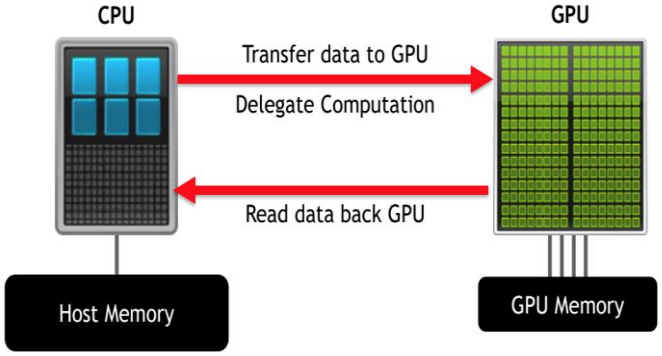
GPU (Graphics Processing Unit)

GPU *(Graphics Processing Unit)*

초기에는 주로 그래픽 처리를 위해 개발, 이후에는 일반적인 연산에도 활용

반복적이고 비슷한, 대량의 연산을 수행하며 이를 병렬적으로(Parallel) 나누어 작업





2 컴퓨터의 구조

GPU (Graphics Processing Unit)

GPU의 특징

① 그래픽 작업 : 픽셀 하나하나에 대해 연산 수행

→ 단순한 ALU 多 : **특화된 연산** 빠르게 수행

→ 비교적 떨어지는 CPU가 GPU로 데이터를 보내 재빠르게 처리

② CPU가 없으면 **단일적으로 작업을 진행하지 못함**

③ **매우 많은 수의 코어** 有

→ 압도적인 코어 숫자로 동시에 처리, 작업의 양을 폭발적으로 증폭

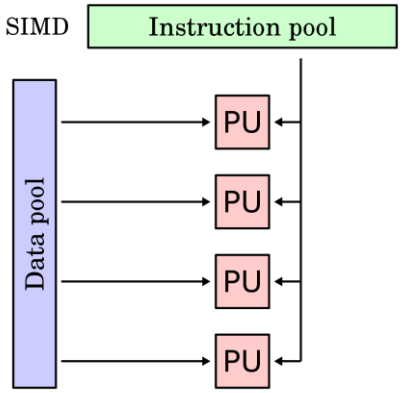
쉽고 단순한 작업을 **병렬**적으로 **대량 처리**하는 것에 특화

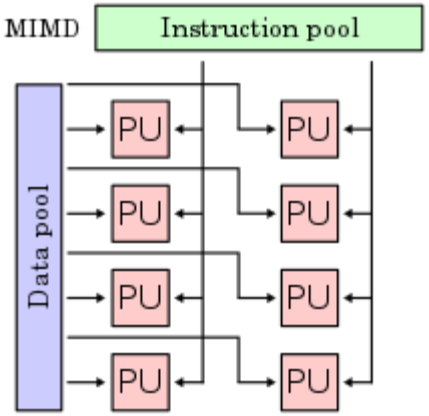
CPU 연산과정에서 GPU에 할당해야 할 경우 →GPU가 **할당** 받아 연산을 처리

GPU의 연산

SIMD

한 번에 여러 개의 데이터를 처리할 수 있는 방식으로,
같은 명령어를 다수의 데이터에 대해 수행
→ 병렬성을 높여서 연산 속도를 빠르게 할 수 있음





MIMD

여러 개의 연산을 동시에 처리할 수 있는 방식으로,
여러 개의 명령어와 데이터를 동시에 처리 가능
→ SIMD 방식에 비해 조금 더 유연한 병렬성을 제공

2 컴퓨터의 구조

Memory: RAM

Memory *(RAM-Random Access Memory)*

컴퓨터에서 데이터를 일시적으로 저장하는데 사용되는 주기억장치

CPU가 작업을 처리할 때, 필요한 데이터를 RAM에 저장

특징

① CPU와 하드디스크 사이에서 데이터를 주고받는 데 중요한 역할

② 컴퓨터가 실행 중인 프로그램과 작업 중인 데이터를 저장하는 데 사용

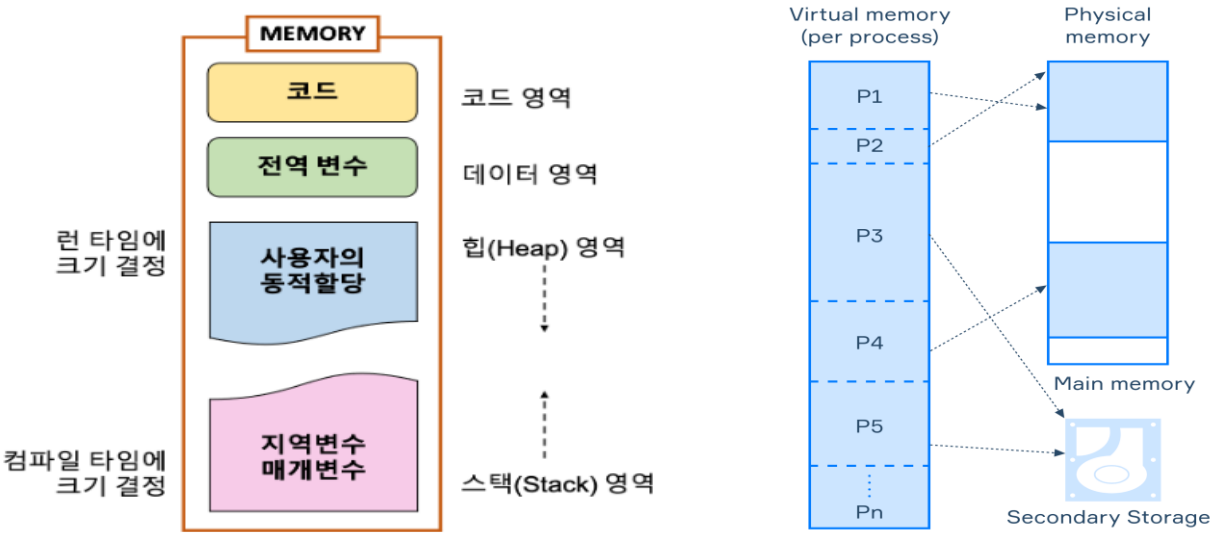
③ RAM의 용량 : 컴퓨터의 성능에 큰 영향을 줌

(저장 가능 데이터 ↑ → 실행 · 처리 능력 ↑)

④ 휘발성 메모리 : 전원이 꺼지면 데이터가 손실→ 재시작 시 다시 데이터를 불러와야 함

2 컴퓨터의 구조

Physical Memory 및 Virtual Memory



Physical Memory: 컴퓨터의 실제 메모리 하드웨어 ← CPU가 직접 접근 가능

Virtual Memory: 물리 메모리와 하드디스크 등의 보조 저장 장치를 조합하여

사용자가 인식하는 메모리 공간을 확장하는 기술 → 사용 가능한 메모리 용량을 늘리는 효과

DISK

DISK

데이터를 저장하는 주요한 저장장치 중 하나

데이터를 비휘발성으로 저장하므로 컴퓨터가 종료되어도 데이터가 유지

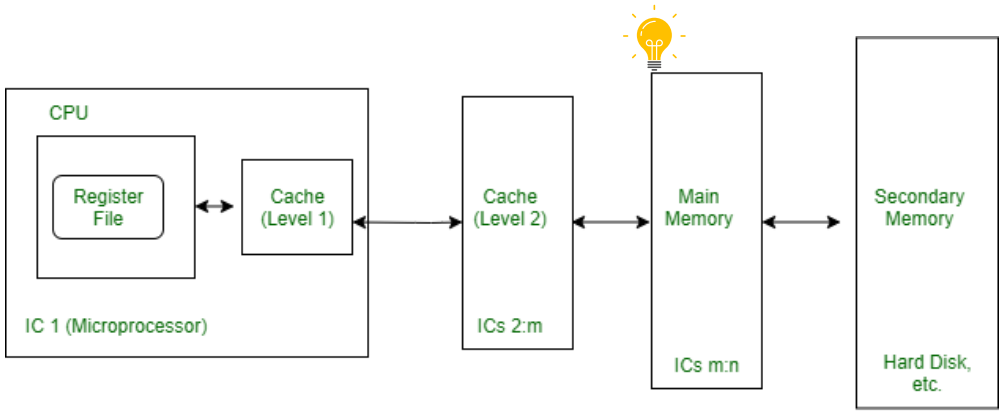
특징

① 주요 역할 :데이터의 영속적인 보존과 검색

② 랜덤 액세스(Random Access): 읽고 쓰는 데 필요한 랜덤한 위치로 빠르게 이동가능

2 데이터를 많이 사용할 수 없는 이유

데이터 이동 과정



Block Diagram Memory Hierarchy

Disk – Memory – CPU의 이동 과정

① Disk에 저장되어 있는 코드와 데이터를 Memory 위에 load

② 변수들을 스택과 힙에 저장, 이를 통해서 CPU는 연산 시작

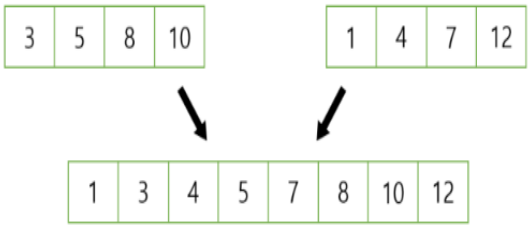
→ Random Access : CPU는 변수의 주소를 통해서 Memory 값들에 접근

③ 연산 결과를 Memory에 저장, 프로그램 종료

→ 저장 명령이 입력 시에 Memory에 있는 값들을 Disk에 저장

2 데이터를 많이 사용할 수 없는 이유

Sort는 어디에서 처리되는 것일까?




```python
with open('data.txt', 'r') as f: ## 데이터를 읽어온다.
    data = f.readlines()

data = [int(x.strip()) for x in data]  # 파일에서 읽어온 데이터를 정수형으로 변환

data.sort()  # 오름차순으로 정렬

with open('sorted.txt', 'w') as f: ## 데이터를 저장한다.
    for d in data:
        f.write(str(d) + '\n')
```

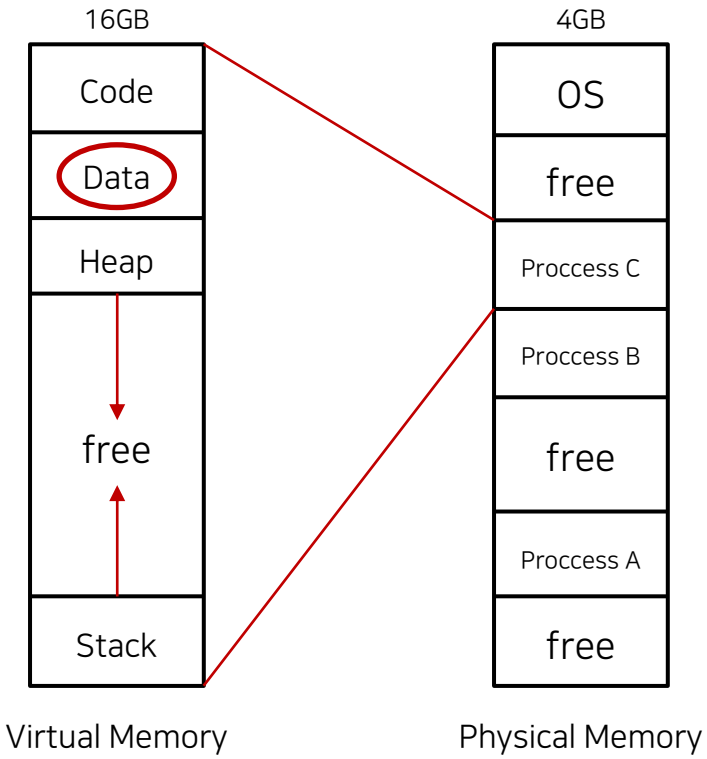
Data가 저장되어 있는 Disk에서 실행되는 것이 아님

데이터들을 Memory 단계까지 load → 이를 Sort하고 다시 저장

많은 데이터라고 하면, 이를 한번에 Memory에서 처리 불가 → **분산컴퓨팅 사용**

2 데이터를 많이 사용할 수 없는 이유

동시에 어느 정도의 데이터를 사용할 수 있는가?



Virtual Memory

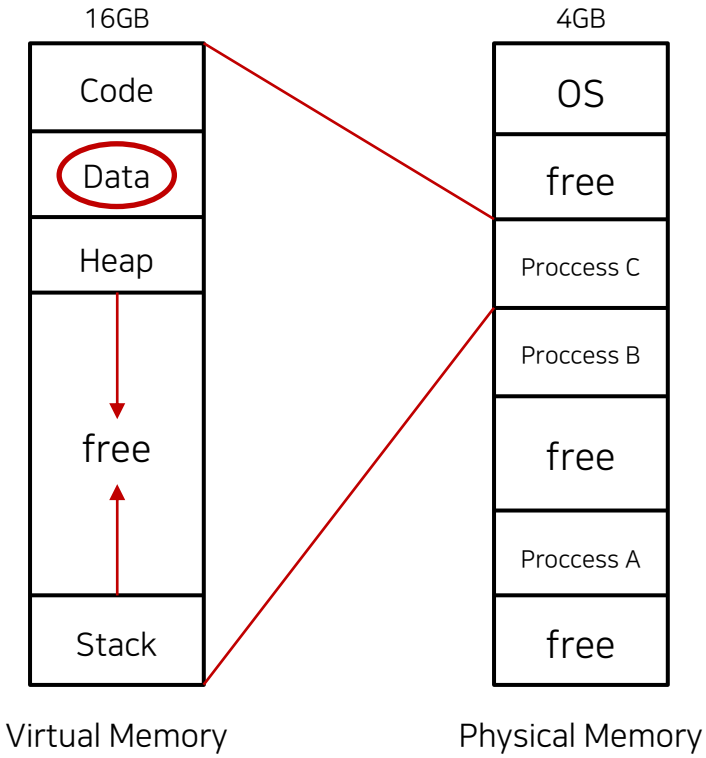
① Process에 하나씩 생성
② 실제 Physical Memory보다 커질 수 있음
③ 필요한 데이터 disk에서 가져옴
→ Physical Memory에 load
④ 필요하지 않은 데이터는 다시 disk에 저장
→ Memory보다 큰 사이즈의 공간 사용 가능

But!

동시에 처리하는 양의 데이터는
Physical Memory 크기와 이론상 동일 → 4GB

2 데이터를 많이 사용할 수 없는 이유

동시에 어느 정도의 데이터를 사용할 수 있는가?



하지만 4GB를 모두 사용 불가
한 작업에서 4GB데이터를
모두 사용하고 싶다고 해서 모두 사용할 수 없음

Page 기법

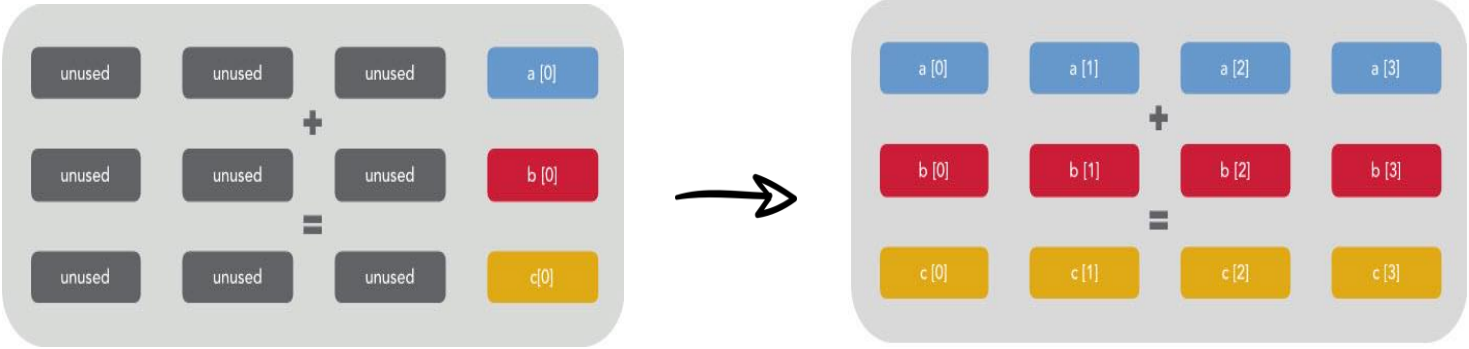
Data 말고도 다양한 정보를 저장하고 처리
다양한 작업 또한 공유하여 처리

3

벡터화

3 벡터화

벡터화 (Vectorization)



- 수학적인 의미: 행렬을 세로 벡터로 바꾸는 선형변환의 하나
  - 반복문을 사용하여 요소별로 연산을 수행하는 대신

  전체 배열 또는 행렬에 대해 **동시에 연산을 수행**하는 방법
- 배열이나 행렬과 같은 다차원 데이터 구조를 사용 → 연산을 효율적으로 수행

3 벡터화

Python에서의 벡터화

Methodology	Average single run time	Marginal performance improvement
Crude looping	645 ms	
Looping with iterrows()	166 ms	3.9x
Looping with apply()	90.6 ms	1.8x
Vectorization with Pandas series	1.62 ms	55.9x
Vectorization with NumPy arrays	0.37 ms	4.4x

A Beginner's Guide to Optimizing Pandas Code for Speed

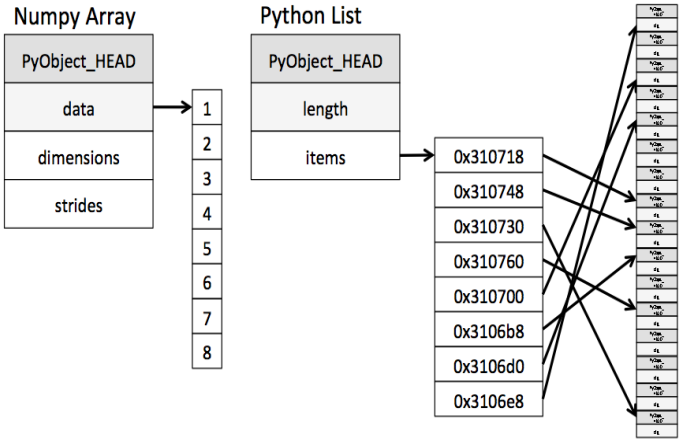
반복해야 하는 경우 벡터 연산을 이용하는 경우 계산 속도가 빨라짐을 확인가능

3 벡터화

Python에서의 벡터화

NumPy

- 다차원의 배열 자료구조 클래스인 ndarray 클래스를 지원 → 쉽게 연산 처리
  - np.vectorize : 넘파이 배열을 벡터화된 연산으로 처리



파이썬 리스트	NumPy
동적으로 크기 조정	초기화 단계에서 크기 지정
유연한 자료형 지정	모든 원소가 같은 자료형
명시적인 루프를 사용하여 연산	브로드캐스팅 선형대수 연산

3 벡터화

Python에서의 벡터화

NumPy

- 다차원의 배열 자료구조 클래스인 ndarray 클래스를 지원 → 쉽게 연산 처리
  - np.vectorize : 넘파이 배열을 벡터화된 연산으로 처리

Tensor / Torch

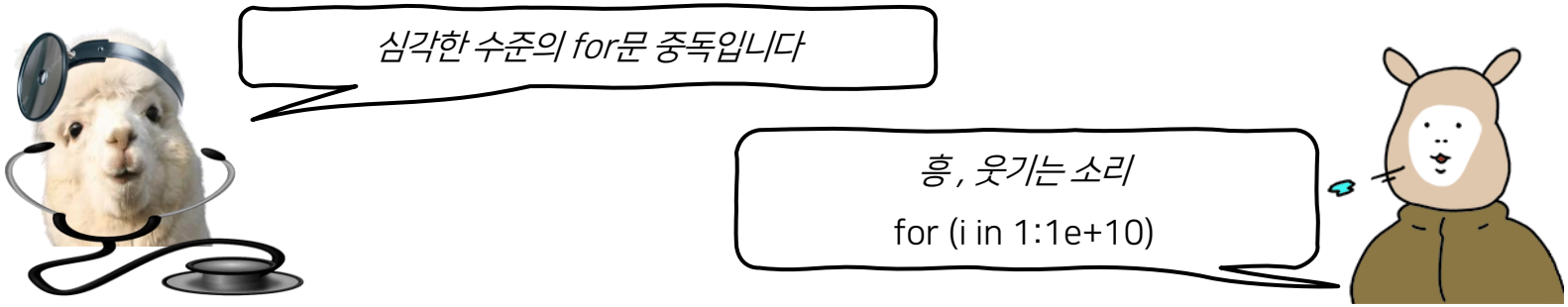
- 다차원 배열로, 벡터나 행렬뿐만 아니라, 3차원, 4차원과 같은 고차원 배열도 표현 가능
  - 딥러닝 라이브러리인 PyTorch, TensorFlow 등에서 주로 사용
  - 수치 계산에 최적화된 **GPU활용 연산 지원** → 대규모 연산에 특화

3 벡터화

R에서의 벡터화

baseR

- R 함수 대부분은 벡터화되었음
- apply() : 대규모의 데이터를 처리할 때 루프를 대신하여 반복되는 작업을 함수화하여 처리
- 적용하는 데이터의 종류에 따라 lapply(), sapply(), tapply()들을 사용

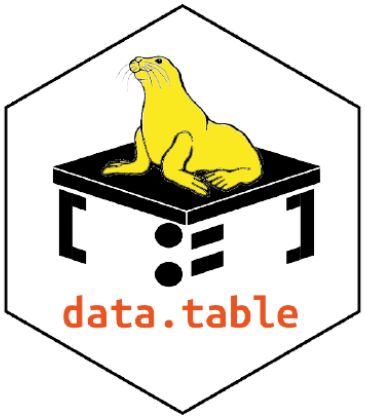


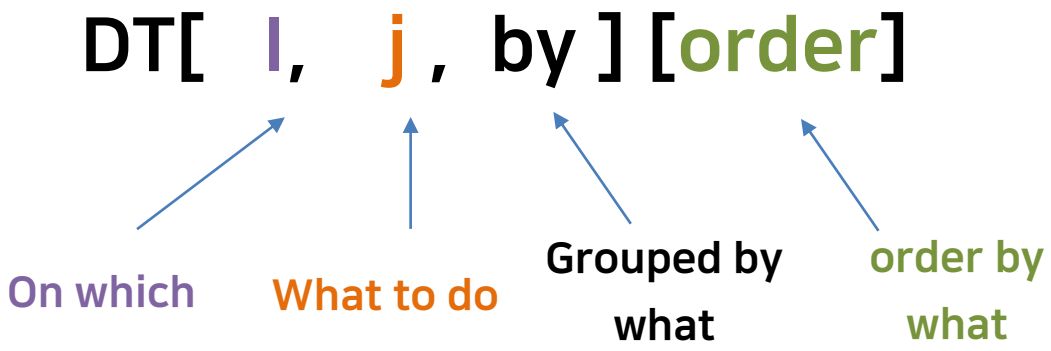
dplyr

- tibble : 개별 vector에서 새로운 tibble을 만들 수 있음
- 길이가 1인 input을 자동으로 재활용하고 방금 생성한 변수를 참조할 수 있게 함

3 벡터화

R에서의 벡터화





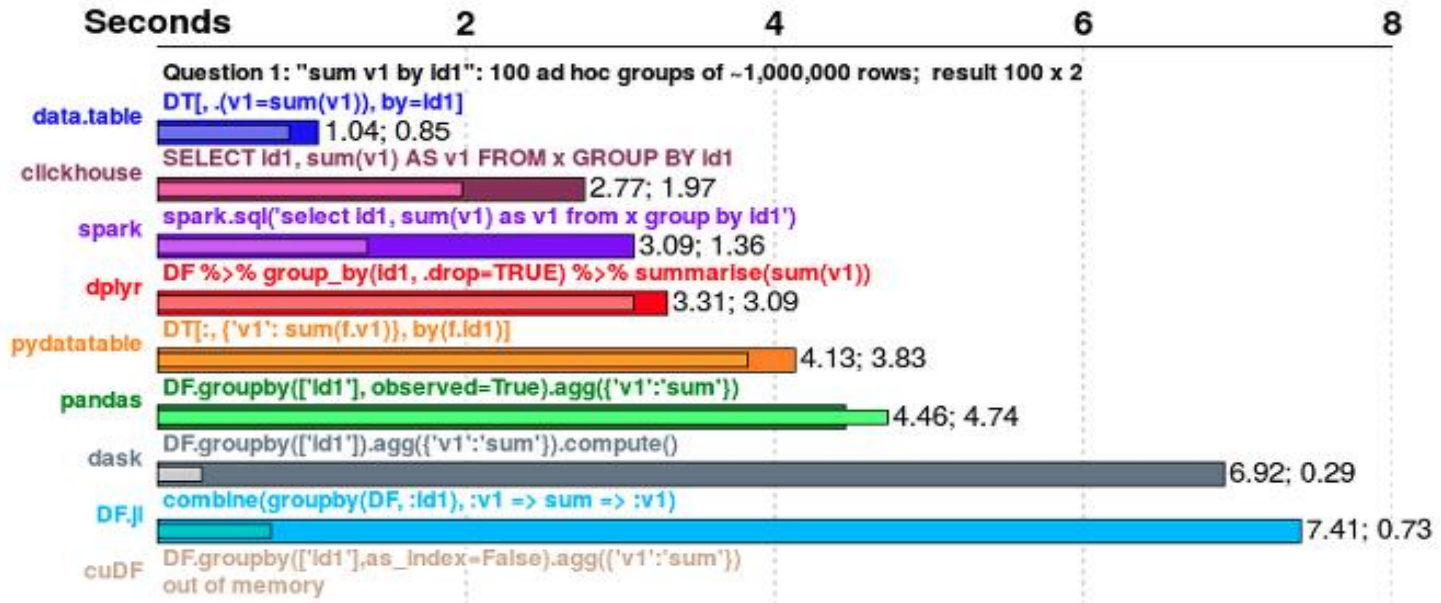
data.table

- 열 지향 연산 → 벡터화된 연산 활용
- fread() : 자료형의 변환 없이 모든 것을 문자로 읽어들임
- Shallow copy : 물리적으로 시스템 메모리에 새로 복사 X ,컬럼명만 복사

csv는 fread로 읽어줘야지

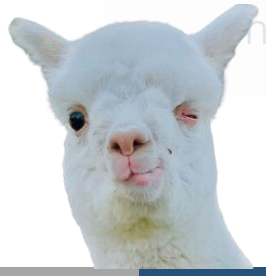
3 벡터화

R에서의 벡터화



대용량 데이터에 대한 처리 속도가 빠름

→ 대규모 데이터셋에 대한 처리 작업에서 유용하게 사용 가능



따봉벡터야 고마워~~

csv는 fread로 읽어줘야지

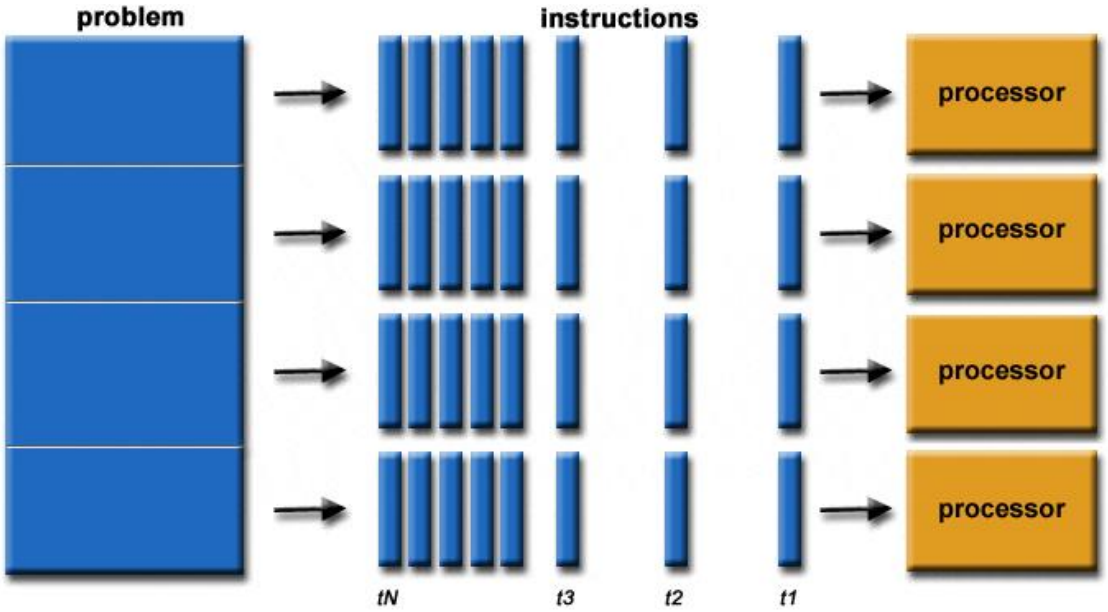
4

분산컴퓨팅

병렬컴퓨팅

병렬컴퓨팅 *(parallel computing)*

여러 개의 복잡한 연산을 여러 컴퓨터가 병렬적으로 동시에 처리하여
서로 독립된 결과를 얻는 처리 단계



4 분산컴퓨팅

병렬컴퓨팅

병렬컴퓨팅 *(parallel computing)*

여러 개의 복잡한 연산을 여러 컴퓨터가 **병렬적**으로 동시에 처리하여
서로 독립된 결과를 얻는 처리 단계

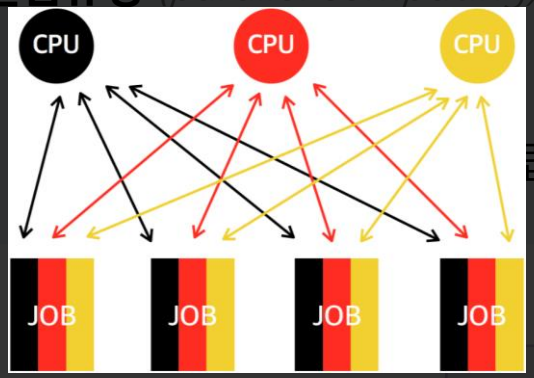
특징

- 한 대의 컴퓨터가 **동시에 여러 작업**을 처리
- **여러 개의 중앙처리장치(CPU)**를 이용하여 동시에 다수의 작업을 처리
- **규모 데이터 처리 및 복잡한 계산 작업** 큰 성능 향상 가능

4 분산컴퓨팅

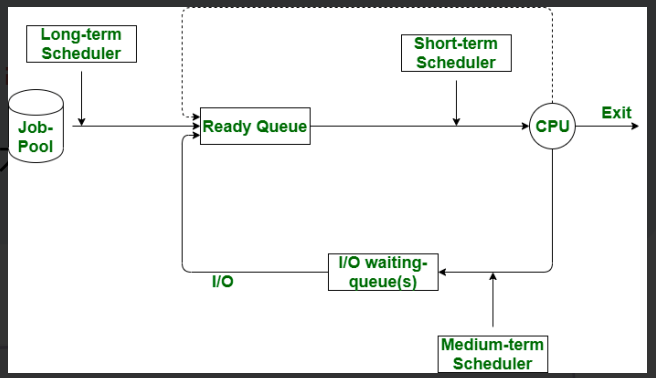
병렬컴퓨팅

병렬컴퓨팅 *(parallel computing)*



여러 컴퓨터가 병

립된 결과를 얻는



한 대의 컴퓨터가 동시에 여러 작업을 처리

Multiprocessing이란?

하나의 컴퓨터에서 동시에

여러 개의 프로세스를 실행하는 것

Scheduler란?

다음 실행 시간에 실행할 수 있는

여러 개의 중앙처리장치(CPU)를 이용하여

프로세스 중에 하나를 선택하는 것

동시에 다수의 작업을 처리

규모 데이터 처리 및 복잡한 계산 작업

스케쥴러와 함께 멀티 프로세싱을 사용하면

일정한 간격으로 실행되는 여러 개의 작업을 동시에 처리 가능!

병렬컴퓨팅 in *R & Python*

병렬계산 라이프 사이클

parallel 패키지 로드 ⇒ CPU 코어 개수 획득 ⇒ 획득된 개수만큼 클러스터 등록
⇒ 병렬 연산 수행 ⇒ 클러스터 중지

라이브러리

parallel

- 병렬 컴퓨팅 기능을 제공하는 r의 기본 패키지
  - makeCluster() : 클러스터 생성 함수
  - clusterApply() : 워커에게 작업 할당
- parLapply(), parSapply(), mclapply()
  :병렬 처리를 수행

foreach & doParallel

- 루프(Loop)와 lapply() 함수를 융합한 것
- .combine 속성 : 결과의 결합 형태 정의

4 분산컴퓨팅

병렬컴퓨팅 in *R & Python*

schedule

- 시간에 맞춰 작업을 실행할 수 있도록 지원
- 백그라운드에서 실행되는 작업 스케줄링을 쉽게 구현
- 단일 프로세서에 적합, 복잡한 연산 X

multiprocessing

cpu 코어가 각자의 역할을
스케줄링 작업을 통해서 병렬로 실행

joblib

- numpy / 파이썬 함수병렬 처리 기능을 제공
- delayed 사용
  : loop →delayed가 각 코어에 작업을 할당
  → 다시 하나로 합쳐줌

dask

- 멀티 코어로 병렬 처리를 가능하게 함
- Bag, Array, DataFrame 자료구조 제공
  → list, numpy, pandas와 유사

분산컴퓨팅

분산컴퓨팅 *(distributed computing)*

다수의 컴퓨터가 공통 문제를 해결하기 위해 **협업**하도록 만드는 방법

컴퓨터 네트워크를 복잡한 과제를 처리하기 위한 **하나의 대용량 컴퓨터**로 설정

분산컴퓨팅

분산컴퓨팅 *(distributed computing)*

다수의 컴퓨터가 공통 문제를 해결하기 위해 **협업**하도록 만드는 방법

컴퓨터 네트워크를 복잡한 과제를 처리하기 위한 **하나의 대용량 컴퓨터**로 설정

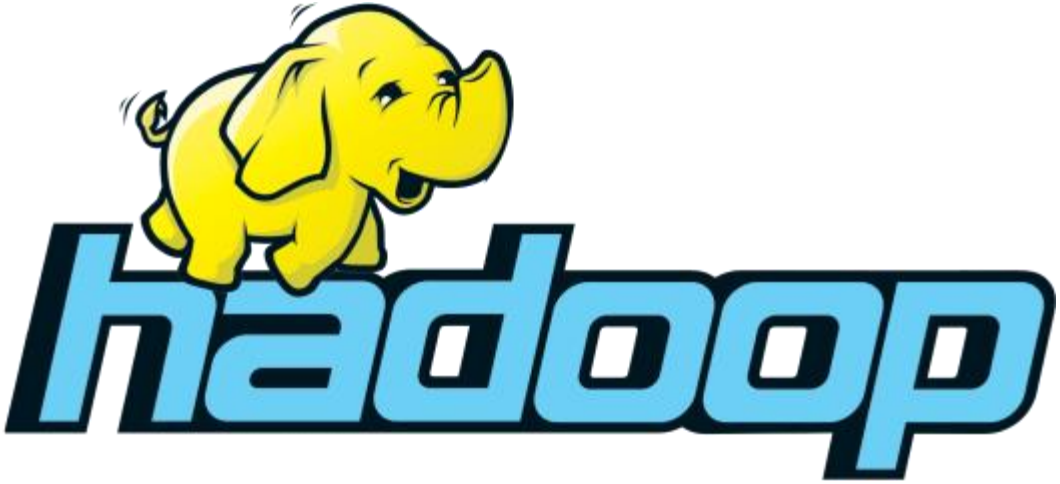
가용성 효율성 투명성 일관성 확장성

분산컴퓨팅의 이점

하둡 (Hadoop)

하둡 *(Hadoop)*

범용 컴퓨터 여러 대를 클러스터화하고, 큰 크기의 데이터를 클러스터에서 병렬로 동시에 처리하여 처리 속도를 높이는 것을 목적으로 하는 분산처리를 위한 오픈소스 프레임워크



하둡 (Hadoop)

하둡 *(Hadoop)*

범용 컴퓨터 여러 대를 **클러스터화**하고, 큰 크기의 데이터를 클러스터에서 **병렬로 동시에 처리**하여 처리 속도를 높이는 것을 목적으로 하는 **분산처리**를 위한 오픈소스 프레임워크

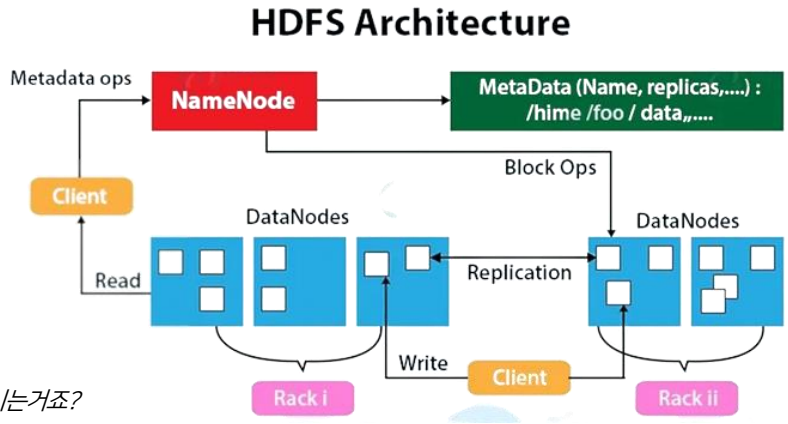
구성요소

Hadoop Common	하둡의 다른 모듈을 지원하기 위한 공통 컴포넌트 모듈
Hadoop HDFS	분산저장을 처리하기 위한 모듈
Hadoop YARN	병렬처리를 위한 클러스터 자원관리 및 스케줄링 담당
Hadoop Mapreduce	분산되어 저장된 데이터를 병렬처리하는 분산 처리 모듈
Hadoop Ozone	하둡을 위한 오브젝트 저장소

하둡 (Hadoop)

Hadoop HDFS

분산저장을 처리하기 위한 모듈



어딜 보시는거죠?

그건 제 복제본입니다만…

블록(block)으로 데이터를 분할하여 **여러 대의 서버**에 저장,

각 블록은 여러 개의 **복제본(replica)**을 가짐

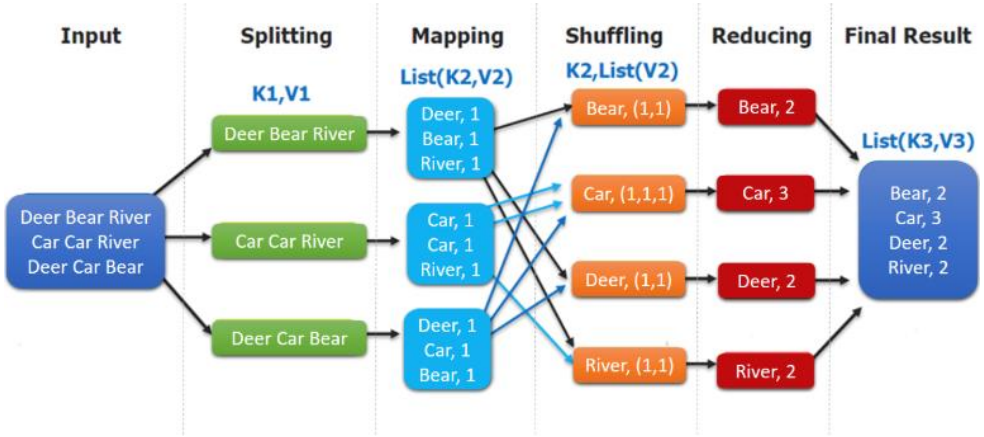
↓

복제본을 유지함으로써, 서버나 디스크의 고장 등으로 인한 데이터 유실을 방지 가능

하둡 (Hadoop)

Hadoop Mapreduce

분산되어 저장된 데이터를 병렬처리하는 분산 처리 모듈



대용량 데이터를 작은 블록으로 **분할**하고,

이 블록들을 여러 대의 컴퓨터에 **분산**하여 병렬 처리를 수행

하둡 (Hadoop)

장점

- 오픈소스로 라이선스 대한 비용 부담이 적음
- 시스템을 중단하지 않고, 장비의 추가가 용이
- 일부 장비에 장애가 발생하더라도 전체 시스템 사용성에 영향 적음
- 저렴한 구축 비용과 비용대비 빠른 데이터 처리
- 오프라인 배치 프로세싱에 최적화

단점

- HDFS에 저장된 데이터를 변경 불가
- 실시간 데이터 분석 같이 신속하게 처리해야 하는 작업에는 부적합
- 너무 많은 버전과 부실한 서포트
- 설정 어려움

아파치 스파크 (Aparche Spark)

아파치 스파크 *(Aparche Spark)*

인메모리 기반의 대용량 데이터 고속 처리 엔진으로 범용 분산 클러스터 컴퓨팅 프레임워크

(= 대용량의 데이터를 고속으로, 효율적으로 처리하는 빅데이터 분산처리 플랫폼)

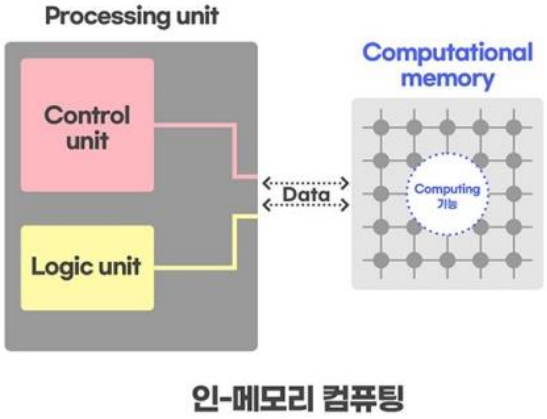
아파치 스파크 (Aparche Spark)

아파치 스파크 *(Aparche Spark)*

인메모리 기반의 대용량 데이터 고속 처리 엔진으로 범용 분산 클러스터 컴퓨팅 프레임워크

(= 대용량의 데이터를 고속으로, 효율적으로 처리하는 빅데이터 분산처리 플랫폼)

인메모리(In-Memory)란?



인-메모리 컴퓨팅

컴퓨터의 메모리(RAM)에 데이터를 저장하고 처리하는 기술

4 분산컴퓨팅

아파치 스파크 (Aparche Spark)

아파치 스파크 *(Aparche Spark)*

인메모리 기반의 대용량 데이터 고속 처리 엔진으로 범용 분산 클러스터 컴퓨팅 프레임워크

(= 대용량의 데이터를 고속으로, 효율적으로 처리하는 빅데이터 분산처리 플랫폼)

인메모리(In-Memory)란?

기존에는 디스크나 SSD 등의 저장장치를 사용하여 데이터를 처리하였지만,

인메모리를 사용하면 저장 장치를 거치지 않고

메모리에서 직접 데이터를 처리 가능

↓

But, RAM은 휘발성 메모리이기 때문에 전원이 꺼지면 모든 데이터가 소실됨

따라서 백업 시스템 구축 필요하다는 특징을 가짐

컴퓨터의 메모리(RAM)에 데이터를 저장하고 처리하는 기술

아파치 스파크 (Aparche Spark) in *Python*

Pyspark

Python 환경에서 Aparche Spark를 사용할 수 있는 인터페이스

Python API를 활용한 빅데이터 분산처리 플랫폼

Pyspark는 분석 목표에 따라 다양한 기능 및 라이브러리를 제공!

아파치 스파크 (Aparche Spark) in *Python*

PySparkSQL

대용량의 정형(sturctured) 데이터 처리를 위해
SQL 인터페이스를 지원하는 PySpark 라이브러리

Pandas API on Spark

Python 환경에서와 마찬가지로
Pandas API를 지원

MLlib

PySpark 기반 Wrapper로서
Spark의 머신러닝(ML) 라이브러리

GraphFrame

데이터프레임 기반 그래프 분석을 지원하는
Spark 패키지

Pyspark는 분석 목표에 따라 다양한 기능 및 라이브러리를 제공!

5

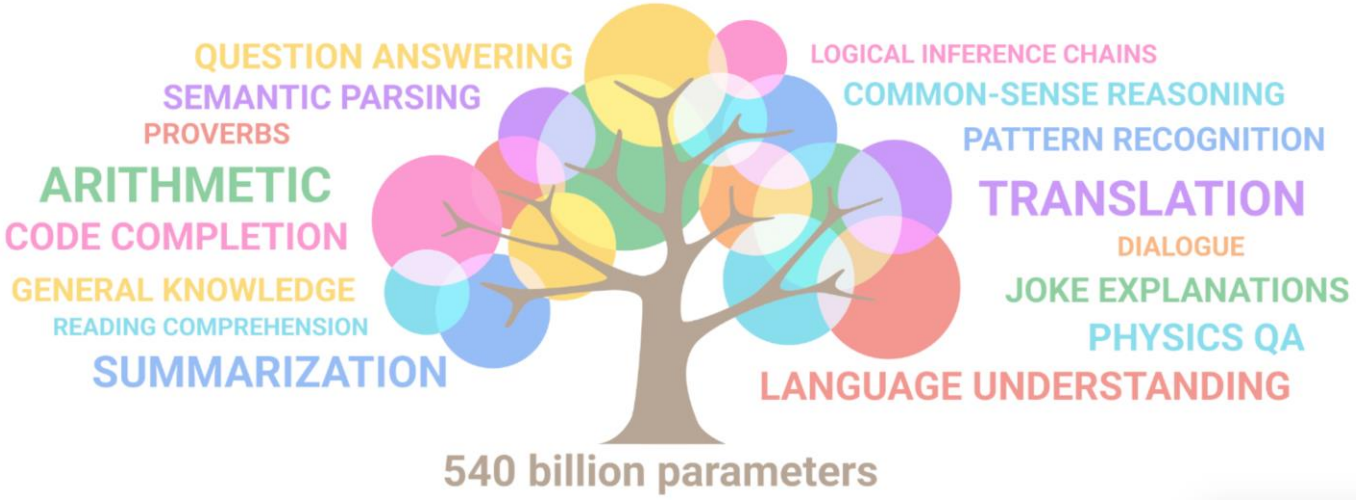
대용량 딥러닝 모델

대용량 딥러닝 모델

대용량 딥러닝 모델

매우 많은 수의 학습 데이터와 매개변수를 가진 딥러닝 모델을 의미

정확한 예측 결과를 얻을 수 있으며, 다양한 영역에서 높은 성능 보임



5 대용량 딥러닝 모델

대용량 딥러닝 모델 학습

1. GPU를 이용한 병렬 처리

그래픽카드의 메모리에 많은 양의 데이터를 적재 → 병렬 처리· 행렬화 연산 진행

2. 배치 학습(batch learning) 기법

데이터를 작은 배치(batch)로 나누어서 처리 → 메모리의 효율적 사용

3. 효율적인 데이터 로딩 및 전처리

필요한 부분만 미리 읽어들이거나, 데이터 압축, 메모리 매핑 사용
→ 전처리 과정에서 발생하는 메모리 사용량 최소화

4. 클라우드 서비스 /분산 처리 시스템 이용

필요한 만큼의 컴퓨팅 자원을 동적으로 할당하여 학습을 수행

5 대용량 딥러닝 모델

GPT

GPT *(Generative Pre-trained Transformer)*

Transformer encoder와 decoder를 사용하는 자연어 처리 모델

대규모의 텍스트 데이터를 학습하여 다양한 자연어 처리 태스크를 수행 가능

이를 '**Large Language Model**'이라 부름

대규모 텍스트 데이터를 기반으로 사전학습

→ 데이터를 배치(batch) 단위로 나누어 처리, 메모리 최소화 가능

분산 학습을 지원하기 때문에 여러 대의 컴퓨터에서 학습을 수행

5 대용량 딥러닝 모델

DALL-E

DALL-E 

OpenAI에서 개발한 이미지 생성 모델

영어로 텍스트를 입력하거나 이미지 파일을 삽입하면 인공지능이 그림을 생성

** Wall-E + Salvador Dali = DALL-E*

Original: Girl with a Pearl Earring by Johannes Vermeer

Outpainting: August Kamp × DALL·E

GPT와 유사한 방식으로 데이터를 배치(batch) 단위로 나누어 처리하며,

메모리 최소화 가능 이미지의 해상도를 낮춤

분산 학습을 통한 여러 대의 GPU를 사용하여 대용량 데이터를 처리

이미지 데이터를 디스크에서 읽어오는 작업도 최적화

5 대용량 딥러닝 모델

또다른 대용량 딥러닝 모델

다른 대용량 딥러닝 모델

StableLM

GPT-2와 유사한 구조를 지닌 언어모델
메타 학습을 활용하여 더욱 빠른 학습과 높은 성능 새로운 작업에 대해서 적은 양의 데이터만으로 빠른 적응

DeepFloyd_IF

이미지 스타일 변환을 위한 딥러닝 기반 알고리즘
입력 이미지와 스타일 이미지를 합성하여 새로운 이미지를 생성(실시간 가능)

segment-anything Model

미지에서 객체를 탐지하고 분할하는 세그멘테이션(segmentation) 기술을 사용하여, 픽셀 수준에서 객체를 인식

AlphaGo

딥러닝과 강화학습을 결합한 인공지능 바둑 프로그램으로, 파라미터와 데이터 수정을 통해 다른 테스크에서도 활용이 가능한 대용량 딥러닝 모델

난 있잖아 ~
통계가 세상에서 제일 좋아♫

다시 보니 선녀로구나

THANK YOU!

알파카는 축제를 즐기러
유유히 떠났다…….진짜☆
- 퍼펙트 수미상관-

안뇽☆