

Concurrency and Parallelism

Concurrency, Parallelism

기본 개념

- Concurrency : 싱글 코어에서 소프트웨어(논리적인) 기법으로 여러 작업(Task, Multi-Thread)을 시분할(Time slicing)하여 교차(Context switching)하면서 실행. (case. Mutex, Deadlock)
- Parallelism : 멀티 코어에서 물리적인 여러 작업(Task, Multi-Thread)을 각 코어들이 동시에 실행. (case. CUDA, OpenMP, MPI)
 - Data Parallelism : 전체 데이터를 쪼개어 병렬 처리. Java에서 제공하는 방식. (Java 7의 Fork/Join Framework, Java 8의 Stream API의 Collection.parallelStream())
 - Task Parallelism : 서로 다른 작업들을 병렬 처리.

Reactive Streams

<https://www.reactivemanifesto.org/>

Back pressure

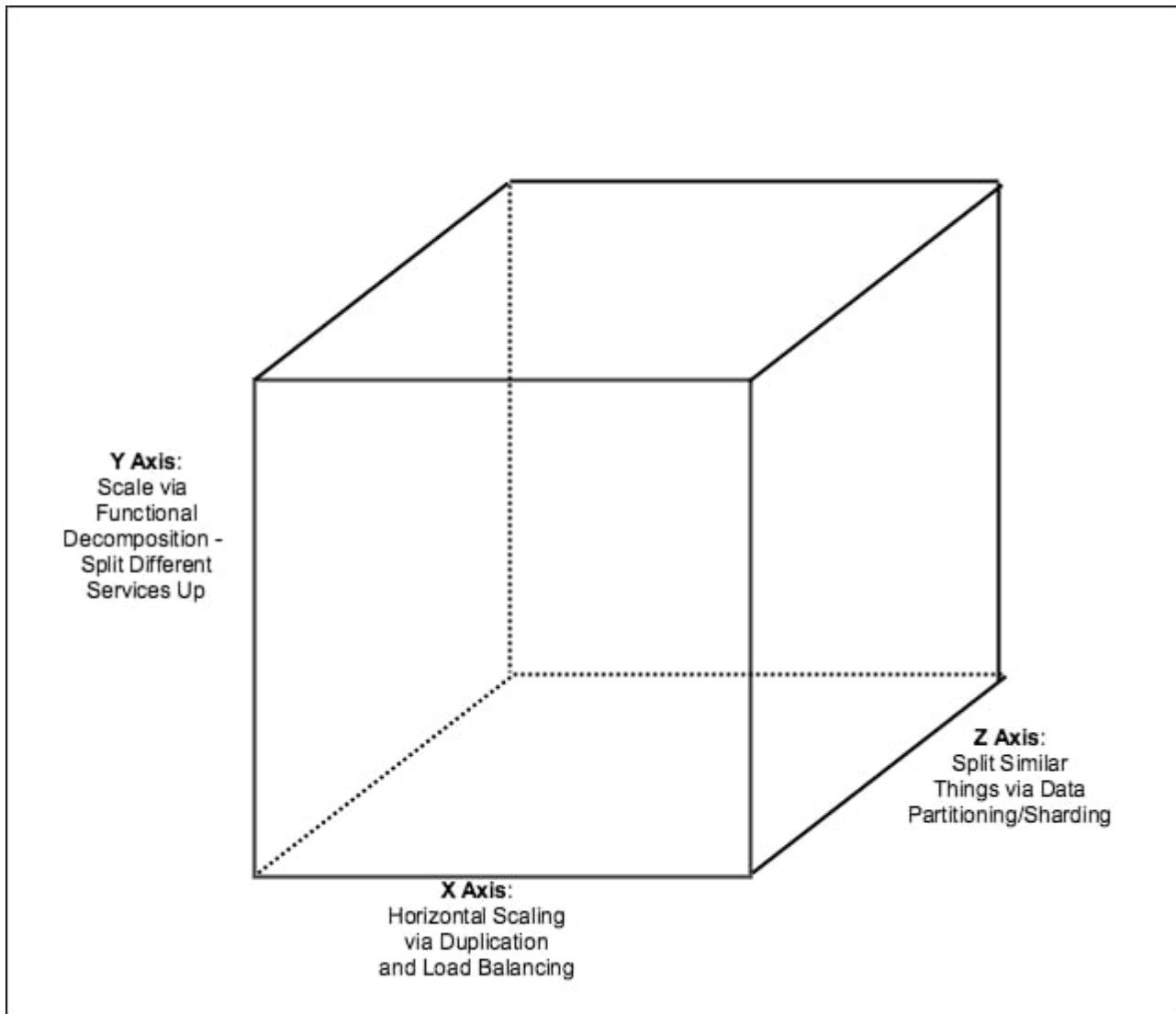
Publisher가 생산하는 데이터 처리 속도보다 **Subscriber**가 소비하는 데이터 처리 속도가 늦을 때, 처리를 늦추거나 중단하는 기술.

- push mode : "Publisher → Subscriber". Subscriber의 데이터 처리 속도가 빠른 경우. (정상)
- pull mode : "Subscriber ← Publisher". Subscriber의 데이터 처리 속도가 느린 경우. 데이터를 처리할 수 있을 때 **당겨(pull)**온다.

Scale Cube

<https://microservices.io/articles/scalecube.html>

Mastering Akka



- X axis scaling : 복제. 서버 앞에 Load Balancer를 설정하여 Traffic 분산, 높은 가용성 제공.
Monolithic system에서 많이 사용하는 스케일링 방법.
 - 단점 : 기능 수정시 전체 애플리케이션을 새로 배포해야 한다. 그리고 자체적인 스케일링 프로파일을 가질 수 없음. (처리량에 따라 가변적일 수 없음.) 그렇기에 자원(CPU, RAM)은 다다익선이어야 함.
- U axis scaling : 분할. 기능별로 애플리케이션 배포.
 - 단점 : 잘못된 MSA 선택은 복잡도 증가.
- Z axis scaling : 복제 + 분할. 같은 서비스 컴포넌트를 모든 서버에 복제하지만, 각 서버는 **일부 데이터**만을 다룬다. 일반적으로 **Sharding**으로 알려짐. 대표적인 예시가 In-memory caching (like Redis)

활용

Java

- Runnable : return 값이 없음
- Callable<T> : return 값이 있음. Future와 함께 사용.

Executor

<https://docs.oracle.com/javase/8/docs/api/index.html?java/util/concurrent/package-summary.html>

- void execute(Runnable command)

ExecutorService

<https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/ExecutorService.html>

- `<T> Future<T> submit(Callable<T> task)`
- `Future<?> submit(Runnable task)`
- `<T> Future<T> submit(Runnable task, T result)`

ScheduledExecutorService

<https://docs.oracle.com/javase/8/docs/api/index.html?java/util/concurrent/package-summary.html>

ForkJoinPool

<https://docs.oracle.com/javase/8/docs/api/index.html?java/util/concurrent/package-summary.html>

Fork/Join 작업을 수행하는 Thread Pool. workstealing 알고리즘 사용하여 유휴 Task를 훔쳐와서 작업.

구현해야 하는 Task는 두 가지. 두 클래스 모두 Future 의 자식.

- `RecursiveTask<T>` : T compute() 구현 필요. T를 return.
- `RecursiveAction` : return 값 없음. 즉, Join 작업 필요 없음.

Future

<https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/Future.html>

작업 완료 여부를 확인, 대기, 조회

CompletableFuture

<https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html>

Future의 조합(상호 연동)이 필요할 때 사용

Scala

Futures and Promises

<https://docs.scala-lang.org/overviews/core/futures.html>

용어

- **Failover**: 장애 극복, 시스템 대체 작동. 장애 발생시 예비 시스템으로 전환되는 기능
- **Redundancy**: 장애에 대비하기 위한 복제된 예비 장치나 기능
- **Data Stream**: 원소가 열거된 것. Producer가 스트림에 원소를 제공하고, Consumer가 스트림에서 원소를 읽는 동안에만 존재하는 **일시적인** 개념. 처리해야할 데이터가 얼마나 많을지 알 수 없고, Producer와 Consumer의 각기 다른 속도를 잘 처리할 수 있을지 어려운 문제를 갖고 있다.¹⁾

¹⁾ <https://livebook.manning.com/book/akka-in-action/chapter-13>

From:

<http://wiki.ledyx.xyz/> - **Ledyx WIKI**

Permanent link:

http://wiki.ledyx.xyz/doku.php?id=concurrency_and_parallelism&rev=1618207048

Last update: **2021/04/12 05:57**

