3. Network Mgmt, DB & Object-Relational Mapping

Great Technology For Great Games

DK Moon
dkmoon@ifunfactory.com

Recap: Network Programming

- ✓ Protocol 이란? Protocol 의 핵심 부분?
- ✓ TCP vs. UDP?
- ✓ 여러 언어에서 존재하는 BSD Socket (TCP/UDP 별 통상적인 함수 호출 순서?)
- ✓ Polling vs. Multiplexing
- ✓ Network framework: C++ 의 boost, Python 의 Django/Flask

IP, Netmask, Gateway

✓ IP 주소
- 32비트 기준의 인터넷 주소 체계 (2^32 까지의 주소값 가능)
- offline 상에서 IP 소유권 관리 (online 상으로 소유자 파악할 수 있는 방법이 없음)

✓ Netmask
- 같은 네트워크인지 다른 네트워크인지 구분하기 위한 mask
- IP_A & net mask == IP_B & net mask 라면 같은 네트워크

✓ Gateway (Router)
- 한 네트워크에서 다른 네트워크로 이동하기 위한 관문 (gateway)
- 이 때문에 둘 이상의 서로 다른 네트워크에 물려 있게 됨

✓ Routing
- 각 목적지별로 갈 수 있는 (모든) 경로들을 알아내는 것

✓ (Packet) Forwarding
- Routing 으로 알아낸 결과 중 목적지별 경로 1개를 선택해서 실제 패킷을 보내는 행동

✓ DNS
- 사람이 읽을 수 있는 호스트 이름에서 IP 주소로 변환 (게임에서의 장단점?)

Network Mgmt Tools

✓ ifconfig / ipconfig
- IP, netmask 정보 확인

✓ netstat -an
- 현재 열려 있는 포트, 연결 상황 출력

✓ netstat -r
- 라우팅 테이블 출력

✓ iptables
- Linux 의 커널 수준의 소프트웨어 방화벽
- 사용 방법이 복잡해서 Ubuntu 에서는 인터페이스를 단순화한 ufw 이라는 프로그램 존재

✓ ping
- 대상 기계가 죽었는지 확인하는데 사용

✓ tcpdump / wireshark
- 패킷 캡춰 유틸리티

Database 기초 이론

04.

CAP Theorem

✓ 전제: 싱글 서버가 아니라 여러 서버로 구성된 시스템을 전제함 (예, master-slave)

✓ Consistency: A 서버에서 데이터를 읽으면, 가장 최근에 쓴 데이터가 읽여야 함.
그게 설령 B 서버에 데이터를 썼다고 하더라도...

✓ Availability: 요청을 보내면, (뭐가 됐든) 응답을 반드시 받을 수 있어야 함

✓ Partition Tolerance: 서버들간 연결이 안정적이지 않은 상황에서도 동작 가능해야 함

✓ Partition Tolerance 한 설계를 한다면 C 나 A 둘 중 하나를 선택해야 됨
(쉬운 버전: 셋 중 둘 밖에 취할 수 없음)

RDBMS

✓ 데이터를 테이블 (Relation) 형태로 저장하기 때문에 RDB 라고 불림

✓ Partition 에 대한 가정이 없었으므로 CA 에 해당하나,
최근 구현들은 Partition tolerance 를 넣으면서 CP 형태가 되기도 함
(Partition 이 일어났을 때 소수가 포함된 partition 에서는 availability 를 포기)

✓ 장점: strong consistency => 프로그래밍에 용이

✓ 단점: strong consistency 에 의한 부하에 취약
partition 에 대한 고려 미흡

ACID

✓ 전통적인 RDBMS 에서의 DB 속성

✓ Atomicity: 관련된 작업들은 all or nothing 방식으로 전체가 처리 되거나, 모두 안되거나

✓ Consistency: DB 의 상태가 일관된 상태를 계속해서 유지해야함
(DB 데이터 속성 제한이나, 테이블간 제약 등. CAP 의 C 와 다름)

✓ Isolation: 여러 작업이 동시에 실행되더라도, 이것이 순차적으로 실행된 것과 같은 결과를 내야함

✓ Durability: 작업이 완료되면 그 결과는 DB 가 설사 크래쉬하더라도 유효하게 적용된 상태여야 함

07.

No SQL

✓ 배경: Amazon, Google 등에서는 DB 가 한군데 있는 것이 아니라 여러 곳에 분산되어 존재
이런 환경에서는 Partition tolerance 에 대한 고려가 필요
또한 RDBMS 가 지나치게 강력한 C 요건을 갖는 점에 반해
C 와 A 를 적절하게 trade off 해서 다양한 설계가 가능함을 발견

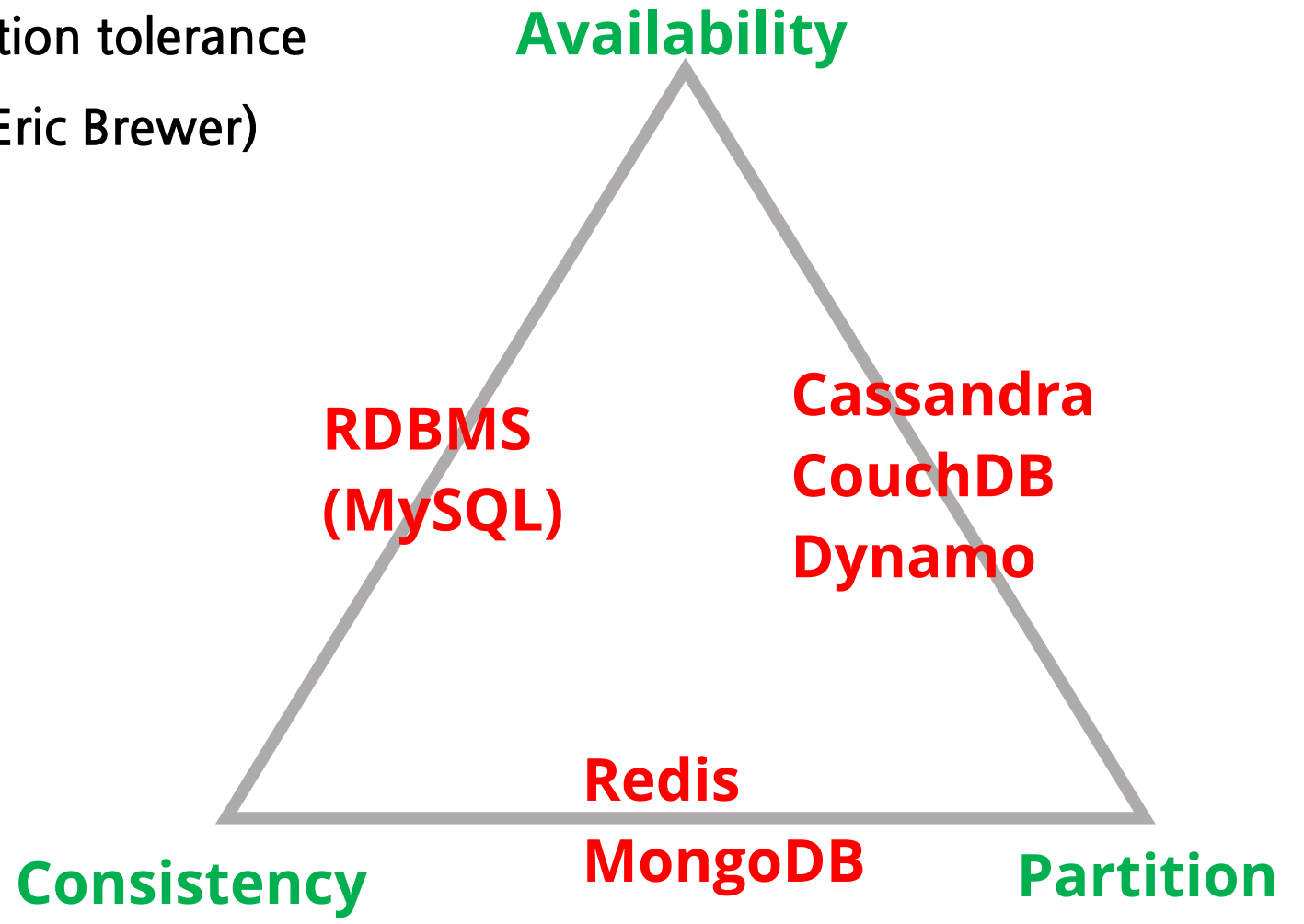
✓ Eventual consistency vs. Strong consistency
- CAP 의 C 에서는 A 서버에 데이터를 쓴 경우, B 에서 읽을 때도 같은 결과가 나와야 함
- Eventual consistency 는 언젠가 결과적으로 B 에서 그 결과가 나오긴 하겠지만,
A 에서 데이터를 쓰는 작업이 일어나자마자 B 에 그렇게 된다는 보장은 하지 않음

✓ 보통 No SQL 구조에서 Master-slave 구조가 있는 경우, A 보다는 C 를 위한 설계가 됨
Replication 을 peer-to-peer 같은 방식으로 하게 되면, C 보다는 A 를 위한 설계가 됨

Trade-off Example: CAP Theorem

Consistency, Availability, Partition tolerance

이 3개 중 2개만 가능 (by Prof. Eric Brewer)



NoSQL

✓ 장점: 용도에 따라 지나치게 강조된 C 때문에 성능상의 병목이 있는 경우 대안이 될 수 있음
(예, 구글 검색 결과는 웹 사이트가 갱신되는 바로 그 순간의 정보를 보여주지 못해도 됨,
로그성 데이터도 정밀도 대신 Throughput 이 중요할 때 사용 가능)

✓ 단점: RDBMS 보다 C 에 대한 보장이 약하기 때문에,
이에 대한 요구가 강한 환경에서는 프로그래밍이 어려움 (예, 게임 유저 DB)

✓ 많은 경우 실무에서의 SQL, NoSQL 배분
- 유저 데이터 => SQL
- 로그 데이터 => NoSQL

BASE

- ✓ SQL 의 ACID 에 대응하는 NoSQL 의 속성
- ✓ Basically Available, Soft-state, Eventual consistency

DB Programming Approach #1

✓ 가장 기본적인 방식: 언어별 라이브러리 (language binding) 를 통한 SQL query 문 전송
게임 서버는 DB 서버 입장에서 볼 때 클라이언트 임을 명심

✓ 단점
- ✓ 프로그래밍이 복잡해짐 => 디버깅이 너무 힘듦
- ✓ SQL query 문은 대개 문자열로 처리됨 => 게임 기획 변경시 일일히 찾아서 수정해야함
- ✓ 또한 SQL query 문장 자체가 엄청 복잡해짐

게임 서버

```
SELECT Name, HeadOfState FROM Country
WHERE Continent = '…';
```

DB Programming Approach #2

- ✓ Stored Procedure 를 활용하는 방법
  - ✓ SQL query 를 실행하는 프로시져를 DB 에 저장하는 방식

- ✓ 장점
  - ✓ 게임 서버 측면에서는 훨씬 단순하게 DB 를 활용 가능

- ✓ 단점
  - ✓ DB 서버를 여럿 돌리는 경우 DB 서버간 상호 참조하는 데이터를 처리할 수 없음
  - ✓ 게임 기획에 따라 procedure 를 다시 작성해야함. procedure 디버깅이 어려움

DB 서버

```
DELIMITER //
CREATE PROCEDURE country_hos
(IN con CHAR(20))
BEGIN
  SELECT Name, HeadOfState FROM Country
  WHERE Continent = con;
END //
DELIMITER ;
```

게임 서버

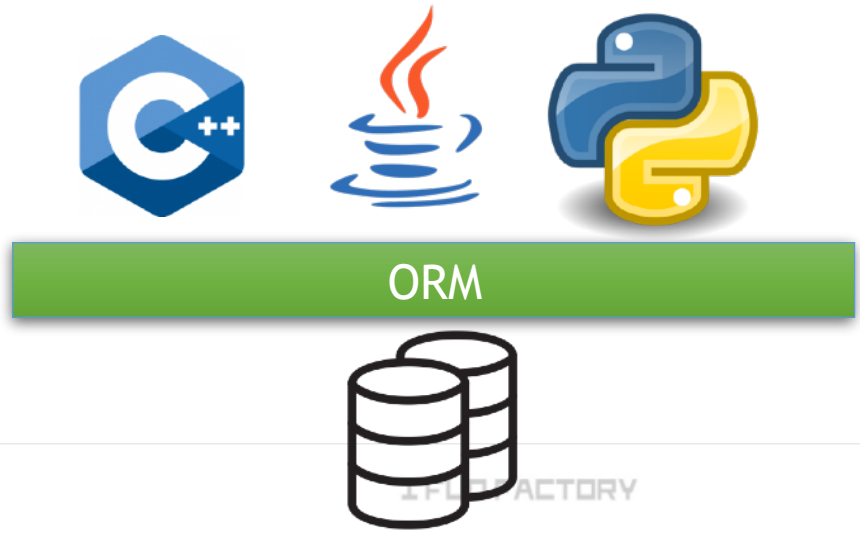
```
CALL country_hos('Europe');
```

DB Programming Approach #3

- ✓ Object Relational Mapping 을 활용하는 방법
  - ✓ 프로그래밍 언어의 Object Oriented 한 클래스를 생성
  - ✓ DB query 는 이 클래스가 처리하게 함
  - ✓ 결과적으로 DB 라는 물리 layer 위에, 프로그래밍 언어의 클래스라는 논리 layer 를 올려줌

- ✓ 장점
  - ✓ SQL 문이 더 이상 문자열이 아니라 프로그래밍 언어의 클래스 => 디버깅이 쉬움

- ✓ 단점
  - ✓ ORM 에 따른 오버헤드 (low-level 한 query 를 효율적으로 처리가히 어려울 수 있음)



Programming 실습: Django ORM

https://docs.djangoproject.com/en/1.10/intro/tutorial02/

Programming 실습:
iFun Engine ORM

https://www.ifunfactory.com/engine/documents/tutorial/ko/orm.html

THANKS!

Great Technology For Great Games, iFunFactory

DK Moon

dkmoon@ifunfactory.com

www.ifunfactory.com