**<8.19>**

**게임서버 (stateful, stateless)**

**서버 종류**

**Stateful : 상태가 있는서버. (MMORPG기반 동기화서버)**

게임의 모든것이 서버메모리 에 들어가있고, 이 메모리를 기준으로 모든것을 실시간으로 관리한다. (몬스터,좌표, 플레이어 등등..)

**MMO : Massive Multi Online (수천명) (ex, 리니지 등등)**

**MO : Multi Online (방을 만들어서 게임) (== 세션형 게임) (ex, 스타,디아블로, 배그 등등)**

**근래는 stateful과 stateless로 MMO와 MO를 구분짓기 힘들어졌다**

**P2P가 사용되지않는 이유**

근래 출시되는 대부분의 게임은 stateful형태이다. (오버워치, 클래시로얄…)

**P2P는 결국 로직을 검증할 수 없고, 따라서 핵을 막을 방법이 없다.**

**stateful 서버**

Stateful은 핵이 존재할수없으나, 헬퍼 같은 비인가 프로그램들은 존재한다.

**Stateful은 메모리에 저장된 데이터가 최신데이터이고, DB는 한박자늦은 백업의 목적이다.**

**로직변경 시 메모리에 있는 데이터가 즉시 반영되고, 메모리에 있는 이 데이터가 최신데이터이다.**

**게임서버가 갑자기 죽는등의 문제로 메모리가 DB로 백업하지못했다면, DB데이터로 롤백된다.**

(서버가 갑자기 죽었거나 등등)

**[ 서버한테 요청 -> 서버에서 데이터 반영(메모리) -> 비동기로 DB백업 -> 클라전송 ]**

**stateless**

**stateless는 상태가 없는 서버로, 보통 모바일 웹기반의 비동기서버가 이와 같은 형태이다.**

**Stateless는 대부분이 웹환경이며 메모리가 없고, 연결의 유무는 상관없지만 대부분의 경우 연결의 개념이 없다.**

**[ 서버한테 요청 -> 서버는 DB데이터를 가져와 로직처리 -> 그 결과를 DB에 반영 -> 클라 전송]**

**stateless에서 최신데이터는 DB이다.**

**동기화된 상태메모리가 없고 하나의 DB에 접근하기 때문에 똑같은 코드의 게임서버가 여러대가 붙어도 상관이없다.**

**단 DB에서 read/write가 계속되므로 성능은 떨어진다.**

이는 어플리케이션 서버형태와 같으며, 데이터가 없고 로직만이 존재하기에 스케일아웃이 가능한 형태이다.

**스케일 아웃과 스케일 업**



스테이트 풀에서의 스케일 아웃개념은 단지 서버군이나 채널이 늘어나는것으로, 아예 다른공간에 있기 때문에 상호 작용이 불가능하다.

(같은 DB를 사용한다면 채널군으로 분리, DB까지 다르다면 서버군으로 분리)

**서버 확장**

근래기준 동접이 5000명정도라면 꽤 잘만든 서버라 평할 수있다.

(ex 리니지 레볼루션이 동접 7000명으로 자랑. 1,2000짜리 서버가 허다하다.)

**가장간단하게 서버를 늘릴수 있는 방법은 하나의 전혀다른 서버를 만들어 플레이하는 것이다.**

**DB를 공유한다면 같은 서버군에서 채널개념으로 채널을 옮긴다면 만날수 있지만**

**DB또한 분리된다면 아예 만날 수 없는 서버군으로 분리가 된다.**

**서버군 통합 이슈**

**DB저장시 보통 auto increment(++시켜 저장하고 유니크값으로 지정)으르 이용해서 저장하는데, 이후 서버통합 시 ID가 중복될 수 있는 문제가 존재한다.**

(string을 가지고 검색하는 것은 속도가 매우떨어지므로 비효율적이므로 사용하지않는다)

**이미 중복된 id를 가진 경우라면 신규가입 차원에서 새로 꽂아넣는 방식을 채택한다.**

**위 사항을 처음부터 설계시 염두해 둔다면 (이런 설계가 없었다면 잘못만든 게임) 전서버를 대상으로 유니크하게 만들어야 한다.**

**방법 1. 유니크한 값을 매겨주는 단독서버를 둔다.**

가장 무난하고 안전한 방법이다. 이 단독서버는 직접만든 서버일수도있고, DB일수도 있다.

**방법 2. 서버번호에 공식을 만들어서 하는방법.**

IP로 규칙을 만드는것. 그냥 ++하는것이아니라 날짜+시간+카운팅 같은 방식을 사용

길이가 길어지고(8byte값으로 커버불가) 가독성이 떨어진다는 단점이 존재한다.

**방법3. 처음부터 지정하는 방법**

서버마다 값 ( 데이터 일련번호 ) 을 정해, 절대 서버안에서 다 못쓸만한 구역으로 나눈다.

**서버1 ( 1 ~ 1억 )**

**서버2 ( 1억1 ~ 2억 )**

**서버의 기능분산**

(기능에 따른 서버분산은 자유이나 책임역시 져야한다.)

**1. 로그인 서버**

게임서버 이전에 로그인서버가 있고 로그인서버를 거쳐서 게임서버로 들어가게된다.

**Stateful은 최초 로그인시 DB로부터 내 계정에 대한 데이터를 읽어오게 된다.**

**DB에서의 읽기쓰기는 매우 느리기 때문에 대부분은 분리한다.**

이 외에도 보안(토큰발급 등의 역할) 등이 있다.

**2. 채팅서버**

**오픈월드에서 보여지는 채팅(전체채팅, 지역채팅)은 문제가 없기 때문에 분리하는 경우가 많았다.**

(클랜채팅, 파티채팅, 귓속말 등은 컨텐츠가 포함되므로 게임서버 내에서 처리해야한다.)

**하지만 근래 채팅서버가 하는일이 점점 줄게되면서 게임서버로 통합하는 추세.**

채팅서버가 별로 하는일이 없게됨 --> 게임서버로 통합

**3. 경매장**

따로 떼어놓아도 상이한 기능이다.

**4. 길찾기 등(느린 알고리즘)**

따로떼어놓아도 상이한 기능중 대표적으로 길찾기가있다.

**서버 기능분산 이슈**

서버를 기능별로 분산시키면 로직은 매우 복잡해지므로 그럴만한 가치가있는지 충분히 고민하고 결정해야한다.

단순한 AI, 퀘스트 같은 것들은 오히려 서버를 분리함으로서 성능손해를 더많이보게된다.

(데이터를 주고받는데 더 오랜시간이 걸리게됨 : RTT)

**RTT(Round Trip Time)**

ㅇ 패킷망(인터넷) 상에서 상대측 호스트까지 패킷이 왕복하는데 걸리는 시간

- 이에 영향 주는 요소 : 망의 혼잡, 거리, 전송속도 등이 있음

**[ 요청 네트워크 패킷 -> 서버네트워크 계층 OS -> 어플리케이션 (처리 후) -> 서버쪽 OS 7계층 -> 요청 클라이언트 ]**

**클라 로직 (L7)에서 서버로직(L7)을 거치고 다시 내 로직으로 돌아오는 시간을 말한다.**

**(핑은 L3계층 까지를 뜻한다.)**

**RTT가 5,60m/s가 나오는 경우 서버네트워크가 느린게 아니라 로직이 느린 경우이다.**

**보통은 네트워크 레이턴시가 거의없다고 해도 1,2m/s는 존재하기 때문에 RTT까지 생각한다면 몇십m/s정도는 나오게됨.**

**레이턴시**

관찰되는 시스템에서의 어떠한 물리적 변화에 대한 원인과 결과 간의 지연 시간이다. 물리적 상호작용이 추진할 수 있는, 물리적으로 제한된 속도의 결과라고 할 수 있다.

**(렉은 게이밍에서의 레이턴시를 기술하는데 사용되는 용어)**

ex) 오버워치의 핑은 보통 5ms (5/1000초) = 500프레임

기본적인 모니터 60Hz ( 1초에 60번 주사한다 ) 이므로 5ms로 패킷이 갔다와도 모니터는 아직 주사조차 하지않은 상태

사실상 느끼는것은 불가능하다.

**일감을 단지 주는데도 불구하고 이런비용을 들여가며 서버를 분산하는것이 과연 효율적인가?...**

**동기화와 비동기화**

동기화 로직이라면 게임서버에서 분산서버한테 어떤 로직을 던졌을때 반환값이 올때까지 기다린다는 의미이기 때문에 별 필요성이 없어보이나, 기다리는것이 아무런의미가 없는것은 아니다.

**구조적으로 봤을때 일감을주고 나는 다른일을 하는것이 가장 좋지만 만약 CPU를 쓰지않는 것이기 때문에 그시간동안에 다른애가 돌수 있다.**

이렇게 설계가 된다면 효과를 볼수 있겠으나 설계하는 것 자체가 매우 복잡해진다.

**동기화로 한다면 구조가 복잡해지고, 비동기화를 한다면 다른일을 할 시간을 벌수 있기는 하지만 그만한 장점이 되는지 고민** ( 일감을 던지는데도 시간이 소요되기때문)

**분산은 한때 유행했지만 지금추세는 한대의 게임서버(단독)을 멀티스레드로 잘 돌리는 것을 선호한다. 단독게임서버이다.** (어차피 코어가 많으므로)

그러므로 서버를 뗀다는것은 보통 권장하지않음.

**서버의 지역분산**

**단, 지역으로 떼는것은 고민을 해볼만하다. 장점이 분명하기때문**

**지역분산이란, 똑같은 역할을하는 게임서버가 3개가있는데 맵이 분리가 되어있다는 것.**

**1,2,3,4맵 5,6,7,8맵**

**장점 : 한서버에 많은 인원을 수용해야 한다면 고민해볼만한 방법..**

(ex 5천명씩 5만명을 한서버군에 넣겠다)

**단점 : 서버대수가 물리적으로 많아져 관리가 매우힘들어지기 때문에 권장하지는 않는다.**

(서버가 많을 때 아이템복사가 가장 흔한 구조이기도하다.)

**인스턴스 던전은 완전히 분리된 형태로, 동기화가 필요없는 구조이기 때문에 걱정할 필요가없음**

**맵을 로드하는방식**

**Seamless**

**플레이어가 맵을 이동하기전에, 나타나야 할 지형을 예측하고 미리 맵을 준비해두는 방법**.

**월드가 끊어지지않고 하나로 구성되어 유저의 이동에 따라 자연스럽게 주변지역을 로딩**한다. 이동시에 로딩이 없으나 맵을 불러올시 로딩이 길어진다. ex) 검사, 마인크래프트, GTA, PC콘솔게임에 사용

**Zone**

Seamless와 반대되는 방식으로 맵을 이동할때마다 로딩이 길어질수 있으나 하드웨어적인 부담을 덜어주기때문에 퍼포먼스 최적화에 도움을 준다. ex) 마비노기, 와우, 아이온

**Room**

방이오픈되고 그속에서 주어진 미션또는 대결을 하는방식.

ex) 디아블로, 캐쥬얼게임

**Seamless맵에서의 게임서버 분산**

**분산을 하지않는 것을 권장하지만, 어쩔수없이 분산해야한다면일종의 경계를 두고 분산한다.**

**유저 입장에서는 경계를 두어도 잘 느끼지 못하나, 특정 컨텐츠가 불가능한 부분.**

(이러한 경계가 없다면 서버 분산이 없는 것)

이러한 경계에서 데이터를 주고받아야 할 상황이 생기면 어떻게 해야할지에 대한 많은 기술논의가 있었다. (두 서버만을 동기화하거나, 겹치는 부분서버만 따로 만들어 관리하는 등…)

하지만 이렇게까지 할만한 가치가 있는지는 의문.

여기서 데이터란, 조금 틀어져도 상관없는 좌표이동이 아니라 싸우는등의 인터렉션 문제.

**정리**

**게임서버에서 분산은 크게 기능분산과 물리(지역)분산**

**기능 분산해볼만한것 ( 처리속도가 매우느린것. ex복잡한 알고리즘, 경매장 등)**

**결론 : 서버분산은 왠만하면 하지말것**

**(책갈피)**

DB밑에 서버가 한대가됐든 두대가됐든 분산이 되어있다고 가정해보자

지금은 DB자체의 성능도 좋은데다가 CPU의 코어갯수도 많이 늘어났다.

**하나의 DB는 하나의 스레드를 사용하는데 만약 4코어짜리 CPU를 꽂았다고 하더라도 하나의 연결로 DB를 사용한다면 하나밖에(1/4) 쓰지못함.**

DB에 데이터는 순서가 정확하게 맞춰줘야하기 때문에 여러개의 연결을 사용하는것도 애매하다.

. --> (거의 단일로 간다.)

예전에는 DB부담이 커서 서버당 하나씩 물리구성을 서비스했는데 DB의 사양이 좋아짐으로써 채널이라는 개념이 등장한다.

**같은서버군이지만 다른채널이라면 만날수없는것으로, 하나의 서버안에서 여러개의 채널이 존재하며 같은DB를 공유한다.**

**하나의 커넥션으로 DB가 남아도니까 채널을 여러개 두어 한서버의 동접자를 늘리게된다.**

**그러므로 지금 추세는 서버를 분산하기보다는 채널을 두고있다. 훨씬 편하기 떄문**

비즈니스 모델측면에서 봐도 MMORPG의 한서버안에 무조건 많은 사람이 들어와야 된다고 요구하지않음. (랭커(VVIP)의 지속적인 과금유도와 진입유저들의 랭커진출의 길을 열어주기 위함)

프로그래머나 기획자의 입장으로서 한번쯤 생각해보는 대규모 1000vs1000은 사실상 돈벌이에 도움이 안됨.

중국쪽에서 나오는 RPG들은 동접자를 일부러 천명으로 고정시킨다. 굳이 몇년동안 고민해서 수만명을 수용하는 서버를 만들필요없이 간단하게 싱글스레드로 찍어낸후 서버군을 늘리는방식을 택함 심지어 천개까지 넘어가는 경우도있다고 한다. (ex 뮤오리진)

참고 : 분산서버의 구조

[https://www.slideshare.net/hyunjikbae1/ss-35\*206140](https://www.slideshare.net/hyunjikbae1/ss-35206140)

**<8.21>**

개발자입장이나, 유저입장에서도 서버군을 구분짓지않고 하나의 서버안에서 채널만 잔뜩 늘린다면 더 좋을 것이다. 하지만 서버군을 구분하지않고 채널만 늘리게되면 DB하나로 수많은 서버들을 감당해야하고, 이는 DB하나로 감당이 불가능하다.

DB가 좋아지긴했지만 이정도까지는 버틸수없음.

보통 MMORPG기준으로 하나의 서버에 단독DB를 한대 두게된다. (물리서버 한대)

(컴퓨터 한대에 MySQL, MsSQL, MariaDB 중 하나를 설치해놓고 그냥 끝 : 분산없는 단독DB)

단독인데도 불구하고 성능이 남아 채널을 늘려 운영하게 된 것.

하지만 10만명 이상이 넘어가는 수준을 하나의 DB로 감당하기에는 부담이 너무큼

**DB분산에 대한 방법**

(모든 저장해야할 데이터를 단하나의 DB로 해결한다고 가정)

**1. 용도별(컨텐츠)로 DB를 따로간다. 하나의 서버에 여러 개의 DB를 사용한다.**

**단점 : 더 이상의 확장이 불가능하고, 트랜잭션을 하고자할 때 성능이 떨어질 수 있음**

**지금은 쓰지않는 옛날방법임. 하드웨어 성능이 부족할때. 현재는 하나에 전부 포함시킨다…**

또한 트랜잭션 여부를 떠나서 오류발생확률도 2배가 된다.

((

관계형 데이터베이스( : RDBMS ) (ex SQL이 붙은것들)에는 트렌젝션이라는 기능이 있다.

**트랜젝션은 여러 개의 작업을 하나의 묶음으로 처리하는 것.**

작업이 마무리가 되기전에 문제가 생겼을 경우를 대비하기위함 (ex 은행권, 인터넷뱅킹 등이 해당됨)

커밋전에 작업에 문제가생겼을 경우 하던작업은 없던일이 되어버린다. 트렌젝션의 작업은 메모리에서만 작동되기때문. (가상작동 후 마무리되면 반영)

**(커밋(commit)**은 데이터베이스 트랜잭션의 내용 업데이트를 영구적으로 확정하는 것을 말한다. 일반적으로 트랜잭션 종료시 해당 업데이트를 확정한다는 의미에서 "커밋"이라고 사용한다. 다른 사람으로부터 자신의 업데이트를 볼 수 있도록 한다는 의미에서 "커밋"이라는 용어를 사용. 반대로 업데이틀 취소하는 것을 롤백이라한다.)

모든: RDBMS의 기본작동은 트랜잭션이 보장이된다.

게임에서도 아이템거래, 현거래 같은 경우 트랜잭션으로 묶어준다.

**그런데 물리적으로 DB가 분리가 되어있을 경우에는 두개의 DB를 대상으로 트랜젝션을 거는게 쉽지않다.**

**MsSQL은 기능자체가 제공이 되지만 이 마저도 성능이 많이 떨어진다.**

((((

여기서 게임서버에서 트랜잭션의 사용은 개발자마다 의견이 갈리는 부분이 있음.

**반대입장**

게임에서는 완벽한 트랜잭션이 불가능한경우가 어쩔수없이 나올때가 있음. 어차피 100%가 아니므로 성능만 떨어뜨리는 작업임.

트랜잭션을 하는 이유는 두가지 ( 1. 논리적인 오류, 2 . 서버가 죽은경우 ) 중 1번은 걱정할 필요가없음. stateful의 경우는 서버에서 로직이 돌고있기 때문에 DB에서 논리적인 오류가 날 일은 없다. (이미 서버에서 검증을 거치고 DB에 반영만 하는 것이기 때문) 그렇다면 남은 것은 2번으로 하드자체가 나가버리는 것. (ex블루스크린) 하지만 이런경우는 거의 가능성이 없다.

혹여라도 문제가 생길경우에는 그때가서 해결해야겠지만 보통의 서버라고한다면 3년에 한번씩 하드웨어를 교체하는 등의 조치가 있기 때문에 거의 가능성이없다고 본다

**찬성 입장**

그래도 안전하게 가려면 하는편이 낫다.

))))

))

**2. 복제를 하는 방법 (리플리케이션 : Replication)**

**하나의 DB가 Master가 되고 나머지가 Slave에게 실시간으로 복제를 실시한다.**

(MySQL에 있는 기능)

분산의 목적보다는 실시간 백업의 목적, 고가용성(시스템 품질 속성 중 시스템이 장애 없이 정상적으로 운영되는 능력)의 목적.

절대 이 DB에 접근이 불가능한 상황을 만들지 않겠다. 365/24/7 단한번도 연결이 끊기지 않게.

(성능을 높이자는 목적은 아님.)

혹시라도 Master가 죽게되어도 Slave에게 백업을 받는다.

이것을 분산의 용도로 쓰고자한다면 Master를 Write로, Slave를 read로 분산시킨다.

슬레이브는 받아들이는 입장이기 때문에 쓸수없으며, 몇대가 되든 상관없음 ( MySQL에서 설정만 하면된다)

**대부분에 현존하는 대부분의 웹사이트는 리플리케이션으로 돌아가고있다. ( ex 위키백과 )**

(페이스북이나 트위터, 구글 같은 곳은 예외지만 아시아정도로만 국한해도 충분히 버팀.)

( 같은 경우 접속자에 비해 글을 쓰는사람은 대게 1%미만이다.)

실시간이라고 하지만 실시간이라고 하기는 어렵다. 데이터를 반영하고 TCP로 쏘고…

반영하기까지 몇m/s의 오차는 나타나고, 성능이 부족하다면 몇초까지의 오차도 발생 할 수 있다.

웹 같은 경우는 몇초의 오차가 생긴다고해도 큰 문제가 없음.

(실제로 페이스북같은것도 국가마다 cache서버가 따로있기 때문에 오래걸림)

게임서버에서의 컨텐츠는 실시간인데 이런식의 오차는 허용되지않음.

* 이런구조는 게임에서 불가능하고 쓰이지 않음.

**3. DB샤딩**

똑같이 생긴DB가 여러명있지만 조건별로 저장소의 위치가 다른 것. (유저별로)



블로그 페이스북..등이 샤딩에 매우 적합한 구조이다.

어느유저가 어느DB에 들어갈것이냐만 결정하면된다.

기준은 어떠한 것도 될수있다.(지역, 유저, 캐릭터, 국가…등등)

배그나 옵치 같은 경우 서버선택이 없다. 크게 아시아, 유럽 정도로 나뉠뿐 DB를 선택하지도 않는다. 전세계 사람들이 같은위치에서 속할 서버만 고르게된다. 채널이랑 비슷한 맥락이라고 봐도된다.

(아시아 채널에서하다가 유럽으로 옮겨도 되듯이)

이 얘기는 글로벌을 묶어주는 하나의 DB시스템이 있다는 이야기이고, 그것이 샤딩이다.

샤딩은 확장성을 가진다 (계속 추가만 해주면 되므로)

단점 : 서로다른 DB에서 커넥팅이 있다면 동시에 진행이 되야된다. 하나의 행동으로 이어져야 하지못하기 때문에 위험하다.

때문에 택하는 방법들은 ( 옵치나 배그같은경우 )유저끼리 거래를 하거나 하는 일은 없다. 내아이템을 구매하고 장착하고 하는일은 로비에서 이루어진다.

또는 아예 거래자체가 되지않도록 컨텐츠적으로 막아버리고 경매장을 이용하도록 한다.

추가로.

DB성능이 떨어질 때 검색을해보면 리플리케이션..샤딩..등등 거창하게 나온다. 하지만 이것은 대부분 웹에있는 기술이다.

또한 책에있는 어떤 기술을 보고자한다면 그 기술은 대게 그 기술을 설명하기위한 코드인 경우가많다. 아무도 그런식으로 쓰지않음에도 불구하고. 하지만 책이라는 속성 때문에 아무도 안쓴다고 하더라도 온갖기능을 설명을 해야하는 필요성을 지니고는 있음.

같은맥락으로 DB의 경우도 마찬가지다..

웹쪽은 리플리케이션쓰는 것이 바람직하며 샤딩역시 페이스북과 같은 성질의 것을 만들고자한다면 괜찮다. **MMO는 절대 분산을 하지않음! 단독DB가는 것이 가장 일반적이고 무난하고 맞는방법.**

컨퍼런스의 목적(진실)

1/3 정도는 과장 1/3 이렇게 하고싶다 1/3 이건해봤다.

100%신용하기에는 매우어렵다.

지금은 한대를 제대로만들자는 추세고, 효율적인 방법이다.

거대한 서버를 만들고자 하거나 서버군을 없애야 한다면 (윗선또는 기획쪽에서) 이런 구조도 고민을 해볼만 하다. 대신 그것에 대한 안전장치를 반드시 기획쪽이랑 커버를 해야할 것

보통은 DB쪽에서 CPU를 100%사용하지못한다. stateful에서는.

**<8.27>**

**서버와 클라이언트의 동기화**

클라이언트는 오로지 뷰(view)만 보여줄뿐, 모든 로직은 서버가 한다.

이동하고자 클릭한 행동은 이동하겠다는 명령이 아니라, 서버에게 요청하는것

(모든 행동이나 결과는 오로지 서버가 시키는 대로 하는것이다.)

요청이 왔다면 서버는 이동가능한 지점인지 확인한 후에 이동하라고 클라이언트들에게 명령

**클라이언트의 데이터는 믿지않는다. 클라의 요청은 모두 서버의 검증과 판단을 거쳐야만 수행이 가능하다. 클라는 단지 Input의 역할.**

(따라서 동기화란 것은 애초에 없다. 모든것은 서버 중심이기 때문.)

Ex) 가고자하는 지점을 클릭했을 때 길을 찾아가는경우

(**서버가 무조건 길을 찾아야 하므로 서버CPU의 부하를 줄이는 것은 불가능**하다. 클라이언트에서 길을 찾게된다면 핵이 나올위험이 있기때문. 그렇다면 이상황에서 서버의 부하를 조금이라도 더 줄일 수 있는 방법은 **네트워크 트래픽을 줄이는 것.**

**1. 어떤길로 가는지까지 서버가 판단해서 클라이언트한테 전달.**

서버의 CPU부하증가, 네트워크 전송량 증가.

**2. 서버가 허용하면 각 클라이언트가 길을 찾아간다.**

클라이언트 요청 서버의 길찾기 후 검증완료 클라이언트에게 목적지 전달 클라이언트 각자 이동시작

**전제조건 : 막힌길이 유동적으로 바뀌는 경우가 있다면 이 방법은 사용할 수 없다.**

(ex 다른캐릭터로 인해 길이막히는 게임일 경우)

**다양한 문제점 때문에 서버가 길을 찾은 후 실시간으로 클라이언트에게 전달하는 방식이 무난한방법이다.**

만약, 서버와 클라사이의 레이턴시가 1초가 있다고 가정한다면

(실제로는 이런 일이 거의 없을 뿐더러 레이턴시가 1초나 된다면 연결이 끊어진다)

클라이언트에서 1초가 늦는 것은 단지 클라이언트가 늦는것일뿐… 이상황에서 클라이언트는 전송된 패킷을 빠르게 따라가면서 서버를 따라잡던가, 아니면 뛰어넘어서 현재전송되는 서버의 패킷을 받던가 하게된다. 이것은 클라이언트의 재량.

MMO의 경우 모든 로직이 서버를 따라가기 때문에 어찌보면 크게 고민할 일이 없다.

만약 서버가느리다? 그렇다면 그냥 모든유저가 느리게된다.

여기서 굳이 동기화이슈를 찾아보자면, 클라에서 얼마나 티안나게 서버의 명령을 빠르게 적용시킬 것이냐가 되겠다. 서버입장에서는 신경쓸게 없음. 서버는 그냥 서버의 상태를 클라이언트한테 전송만 하면 되기 때문.

**타겟팅과 논타겟팅**

**공격처리를 위해서 연산이 가장 많이 들어가는 부분은 “이 공격으로 인한 피해자를 찾아내기” 로서, 실질적으로 충돌을 처리하는 로직이다. 3D MMO라면 확인해야할 오브젝트도 많고 폴리곤 충돌 계산량이 많음.**

**타겟팅의 경우에는 정해진 상태에서 공격을 하기 때문에 충돌여부 판정의 계산이 간단하게 끝나버린다. 단순거리, 장애물 정도면 판단하면 끝나기때문.**

**논타겟팅의 경우 서버가 다수의 몬스터를 대상으로 충돌 판단을 해야하므로 서버의 부하가 크다.**

언급했듯이 서버와 클라이언트의 동기화가 엄격하지않다면 판정에 많은 불만이 생길 것이고, MMO에서의 원하는 타겟에 대한 공격이 매우 어려워질 것.

**타겟팅**

**플레이어가 마우스 등의 컨트롤을 사용해서 대상을 지정하여 이미 판단이 끝났기 때문에 클라든 서버든 각자 상황에 맞게끔 각자 상황에 맞게끔 보여주면 된다.**

클라이언트에서 대상을 지정하여 서버에 요청 가능한지 판단 및 검증 서버에서 허용해 주었다면 명령을 적용하여 결과를 클라이언트에 전송 서버로부터 받은 공격 결과에 따른 공격 액션 및 이펙트 표시

**오토타겟팅**

논타겟팅 같은 타겟팅 게임이다.

**직접 타겟을 선택해야했던 타겟팅 방식에서 플레이어 캐릭터의 시야 전방에 위치한 몬스터에게 자동으로 타겟팅이 되는 방식이다**. 유저의 조작은 논타겟팅 처럼 액션감을 느낄 수 있고, 실제로는 타겟팅 방식 이기 때문에 서버의 로직은 가벼워진다. 유저입장에서도 정확한 타겟팅을로 더 편리하고 시원한 플레이가가능헤진다.

(에임핵이 자체내장 되었다고 봐도 무방)

**논타겟팅의 경우**

좌표가 차이날경우 문제의 소지가 있음.

클라이언트와 서버가 완벽하게 동기화되어 돌도록 하는 것이 이상적이다.

하지만 이는 상당히 어려운 작업이고, 구현한다고 해도 전투의 액션성과 조작감의 재미를 위해 만드는것인데 이것이 게임의 재미나 완성도에 크게 기여를 하는 것이아니라 때문에 현재 논타겟게임이 거의 없음.

**키보드를 통한 캐릭터이동**

마우스의 클릭은 목적지가 정해져있는 반면에, 키보드는 목적지를 알 수가없음.

사용자가 키보드에서 손을 땔 때 가 목적지가 되고, 그 즉시 서버에게 전송한다고 하더라도 좌표가 틀어질 수밖에 없다.

네트워크 레이턴시(:반응 속도)가 없다고 해도 RTT개념으로 간다면 아무리 빠르게 만든다고 하더라도 2~3프레임(4-50m/s )정도는 발생할 수 밖에 없다.

**(RTT** (Round Trip Time, 왕복 시간) 패킷망(인터넷) 상에서 상대측 호스트까지 패킷이 왕복하는데 걸리는 시간)

그렇기 때문에 기획적으로 커버를하거나 이동단위를 넣는 것이 일반적이다. 단 한번 클릭했다고 하더라도 최소한의 이동단위에 따라서 이동하게된다. (손을뗀다고 즉시멈추지는 못함)

또는 키보드에서 손을 뗄떼 클라이언트에서 지정좌표를 잡고 서버에 요청한뒤 동일한 지점에 서는 것.

또는 키보드 조작임에도 특정 좌표를 서버에게 요청(전송)하는 것. (서버한테 계속해서 통보받아 가는 방식이므로 통신량이 증가하게 됨)

듀랑고나 리니지 같은 경우는 네트워크가 끊기는 경우 아예이동하지 않는다. 서버로부터 통보가 오지않을경우 아예 멈춰버리기 때문. (동기화가 안되면 아예 작동 X)

**마우스로 방향을잡고 키보드로 이동을 하는 경우**

복잡하게 이동하는 경우 매프레임을 보내서 동기화기시키지 않는이상 계속해서 서버와 통신한다고 해도 완벽하게 따라갈수 없음.

(이렇게 매프레임마다 동기화를 시키는 것은 MMO가 불가능함…FPS에서 이것을 완벽하게 잡고자 프레임마다 쏘고있음. Ex 배그)

그래서 채택한 방법은, **캐릭터를 조작하는 것이아니라 캐릭터앞에 더미를 하나 만들어두고, 더미가 이동하면 캐릭터가 따라 움직이게 된다. (일종의 목적지를 정하고 서버와 클라 모두 더미를 따라감.)**

그렇다면 키를뗀다고 하더라도 더미까지는 캐릭터가 움직이게 된다. 완벽하게 좌표를 일치시킬 수 없기 때문에 즉각적인 액션이 나오지않는 단점을 가지고있음.

더미의 범위(좌표)는 일정주기마다 게임스타일에 맞게 정해줘야한다. 계속해서 서버에게 전송하면 서버는 그 범위를 향해 가라고 지시.

더미의 방향과 로직에 대한 계산식을 그대로 똑같이 따라 받아서 이동해도 상관없음. 오히려 더 좋은방법

((

유니티는 유니티에있는 물리엔진을 써버리기 때문에 괜히 어설프게 흉내내다가 더 이상한 계산식이 나오기 때문에 그럴바에는 아예 안하는 것이 낫고, 아니라면 유니티 물리엔진을 쓰지말고 직접로직을 짜면된다.

사실 서버와 로직이 똑같아 져야하기 때문에 사실 유니티에 있는 길찾기도 쓰기어려움.

물리엔진에서 자기멋대로 처리해버리기 때문에 보통 MMO클라쪽에서는 물리엔진을 잘 쓰지않는다.

떨어져야 하는 경우인데도 1픽셀의 오차로 떨어지지않는 경우도 있음

**그렇기 때문에 물리엔진을 꺼버리고 서버가 명령하는대로 받아서 처리하게된다.**

))

((

오버워치 같은 경우 모든 로직을 서버에서 처리한다. 이상황에서 클라와 서버의 오차가 생긴다면 문제가 생김.

미묘한 차이까지 모두 따진다면 유저들의 반발이 심해진다. 때문에 어느정도의 허용치를 두고 허용해준다.

FPS게임에서의 이 비동기화문제는 언제나 논란이 되고있음. (클라-서버 구조든, P2P구조든) 완벽한 해결책이 없음

그래서 외국에서 해결책으로 쓰이는 방법은, (우리나라에서는 쓰이지않음)

로직차원에서 롤백을 시켜버린다. 만약 2프레임정도에 총을 쐈는데, 다른사람 화면에서는 안맞는 걸로 판정이 났다면 그 2프레임전으로 내부로직을 돌려서 충돌을 판단하게끔한다.

스타크래프트1는 락스텝동기화 ( 모든 프레임을 동기화 ) 를 사용한다.

일반적인 클라-서버 MMO구조에서는, 클라가 요청하고 서버가 수행하는동안에도 시간은 계속해서 흘러간다.

**하지만 락스텝 같은 경우는 (MO이고 모든 유저들이 클라 ) 본인 이외에 모든 클라에게 매 프레임마다 전송을 해야할 의무가있다.**

그리고 다른클라의 프레임이 도착하지않는다면 게임자체가 멈춰버린다. 즉, 모든프레임을 완벽하게 동기화시킨다.

수많은 유닛하나하나를 동기화 시키기는 것은 MMO급으로 거대하므로 유닛을 동기화시키지않고 매프레임을 동기화시키는 방법을 채택한다. 소규모 전략 시뮬레이션은 사실상 이렇게 할 수밖에 없음.

클라이언트가 3명이있다면 이 3명이 완전히 하나의 로직에서 도는것처럼 맞춘다.(Input의 동기화)

MMORPG에서는 쓸수없는방법.

))

**<8.30 수업 >**

모든 로직은 서버에서 돌고 클라이언트는 단지 뷰만 보여준다. 클라이언트가 느리던 빠르던 서버는 돌고있기 때문에 문제될것이 없음.

다만 키보드 입력을 받아서 처리할경우, 시각적인 어색함에 대한 해결책을 생각해봐야한다.

(타겟팅은 문제가 되지않음. 지금 말하고자 하는 이슈는 논타겟팅이다. )

(클라A가 클라B에게 칼을 휘둘렀을때, 서버에게 응답을 기다렸다가 이벤트를 주는것은 어색해보인다.)

방법 1.

클라A가 클라B에게 칼을 휘둘렀을때 일단 이펙트를 먼저 터트린후에 서버로 부터 검증을 받고데미지를 계산하여 적용한다. 하지만 서버에서 맞지않았다는 결론이 나오게된다면, 클라이언트에서는 이펙트만 나오게된다. (이 클라이언트만 혼자서 맞았다고 착각을 하기때문에 여전히 어색하다.)

방법 2. (ex 마비노기 영웅전)

클라 A가 클라B에게 칼을 휘둘렀을때 일단 맞았다는 모션만 보여준다. 이후 잠깐 정지하고, 서버로부터 검증이 올때까지 기다린다. 약간의 지연시간이 생겼지만 오히려 타격감이 생겼음.

**p2p게임**

네트워크 랜게임이 온라인화 된 구조라고 봐도 무방. (ex 스타 배틀넷 )

서버가 일체 관여하지않으며, 클라이언트끼리 모든것을 해결한다. 서버를 거치지않기 떄문에 더 빠른 반응속도를 가질수 있고, 서버의 부담자체도 없다. ( 따라서 대부분의 액션게임들은 p2p방식이었다)

**가장 큰 단점은 핵이 나온다는것. 각자가 본인것을 처리하기때문에, 값을 바꿔치기하면 그대로 적용이되기 때문이다.**

서버기술의 발전과 네트워크 환경의 개선으로 대부분 서버-클라형식으로 바뀌게된다.

기존에 p2p인 게임들도 서버를 하나두고 연결은한다. (ex 프리스타일 )

(클라에있던 코드엔진을 그대로 서버에 가져와서 검증만한다. 뭔가 이상하다고 판단된다면 블락시킨다.)

액션성이 강한게임일 경우 약간의 좌표만 틀어져도 오차가 생기기때문에 약간의 허용치를 두게되는데, 이 허용치를 찾아내서 핵을 또 발전시키고있다.

**<9.17>**

MO의 경우 로비서버를 가지고있는 경우가 많음.

던파 같은 경우는 걸어다니면서 채팅을하는 실시간 서버가 된다. MMO느낌이 나긴하지만, 단지 방에 대한 정보를 관리해주고 있을뿐 실제로 컨텐츠가 일어나고 있는 것은 아님.

로비에서 실제 게임으로 접속이 될 때 로비에서는 (준비완료 된) 유저에게 어디에 접속을 할건지 알려주게된다.

접속자중 한명이 호스트 역할을 하는 것이 가장 일반적인 경우이다. (방을 만든사람이 서버가된다)

**한명이 서버역할을 하는 HOST방식**

옛날 포트리스 시절에는 공유기를 사용하는사람은 방을 만들지 못했다.

공유기를 쓰는사람은 사설IP를 사용하고, 공인IP를 공유기가 알고있는 상태이다.

방을 만드는 사람이 접속해야될 IP를 다른유저들에게 알려주게되는데, 공유기를 사용하고 있다면 이 IP는 공유기의 공인IP가 된다. 그렇다면 유저들은 공유기에 접속을 시도하는 것이 되버린다.

때문에 **공유기를 사용한다면(사설IP유저) 아예 방을 만들지 못하도록 막아놓는 것이 일반적이고, 현재까지도 그런경우가 많다.**

**홀펀칭**

위와 같은 문제의 해결책으로 공유기를 뚫고 안쪽까지 어떻게든 들어가게 한다.

하지만 홀펀칭 역시 완벽하게 모든 공유기를 홀펀칭으로 성공시킨다는 것은 불가능에 가깝다.

( 프라우드넷 역시 전세계에 있는 공유기를 모두 테스트했다며 홀펀칭을 자랑했으나 이역시 80%언저리였음. )

**홀펀칭이 되지않았을 때 유저가 할 수 있는 차선책.**

때문에 공유기에 게임기능이 들어가는 경우도 생기게 되었는데, 해당 게임에 쓰는 포트를 열어주는 방식이다. (ex 5000포트를 내쪽으로 열었다고 한다면 공인IP에 5000으로 뭔가가 들어온다면 무조건 포워딩 (forwarding : 전송) 시키는 형식. 내가 마치 공유기 IP를 가지고 있는것처럼 )

**VPN**

보통 회사안에 인트라넷( 외부에서는 접근할 수 없는 회사 내부용 사설망 )같은경우, 해당 회사의 IP대역안에서만서버에 접근할 수 있다.

여기서 VPN을 설치하고 VPN서버로 로그인(접속)을 하게되면 VPN을 통해 해당 인트라넷에 접속이 가능해지게 된다.

외부에서는 VPN장비와 통신하는것이고, VPN장비는 마치 IP대역안에 있는것처럼 만들어준다.

((

Ex) 집에 IP타임 공유기가 있고 다른곳에 VPN기능을 셋팅을 했다고 가정한다면 어디있든간에 외부에서 집에있는 IP타임에 VPN로그인을 한뒤 IP를 사용하면 집IP를 사용하는것과 같아진다.

통신이 외부-> 집 -> 목적지 순으로 이루어지게된다. (경유지가 되는 것)

))

VPN은 비대칭키로 암호화가 되어있기 때문에 누군가가 복호화를 하거나 오픈이 불가능함.

중국의 경우 VPN을 통해서 외부로 나가게 된다.

**하마치**

하마치역시도 VPN이다. 바깥에 있는 다른유저들이 공유기에 막혀서 들어오질 못하게되는경우 앞서 언급했듯 포트를 열어주거나, **외부에 있는 VPN서비스를 사용해서 VPN으로 들어가고, 유저들한테는 VPN주소를 알려주게된다.**

이 유저들은 VPN으로 접속을 하는건데 그게 실질적으로는 나한테 오게된다. 왜냐하면 VPN으로 먼저연결을 했고 그 다음부터 IP에 해당 포트로 들어오는 모든 데이터는 모두 나한테 전달이 될것이기 떄문.

정리하자면 별도의 공인IP를 할당받는 느낌이고, 경유지를 거치기 때문에 성능은 떨어질수밖에 없다.

**P2P**

**super peer방식**

슈퍼피어는 p2p시스템의 모델이다..

P2P같은경우 서버(서버역할)로 연결을 하기는하지만 로직에대한 처리는 본인이 직접담당하는 것이 일반적이다.

(핵이 나올 수 있는 이유)

**Real P2P방식**

모든 접속자가 한명도 빠짐없이 연결을 하는 것

내 정보를 내가 판단하고 다른접속자들한테도 뿌리게된다.

응답속도는 빨라지나 관리가 힘들어진다. ( 모든 접속자들이 홀펀칭에 성공해야 되기 때문 )

비교하자면 게임에서 Real p2p방식보다는 super peer방식을 더 많이 사용한다.

이런식으로 진행이 되다보니, 관리만 하면 되는 로비는 사실상 할일이 없다.

또한 게임자체의 진행은 클라이언트 끼리 하게된다.

온라인 게임기준 클라이언트 프로그래머는 네트워크에 대한 지식이 0에 가깝다고 보면된다..

클라이언트에 들어가는 모듈, 네트워크파트 등등은 모두 서버프로그래머 담당.

함수만 호출하면 해결되게끔 만들어놔야하고, 그외적으로 문제가 발생하는것도 클라이언트 파트에 서버 프로그래머가 투입이 되는것이다.

P2P의 경우 서버가 들어있기 때문이다.

**릴레이 서버**

모든 유저들이 릴레이 서버로 접속을 하게된다.

**릴레이서버 자체는 어떤일도 처리하지않으며, 패킷이오면 다른 접속자들에게 최대한 빠르게 전달만 하는 역할.**

여기서 릴레이서버는 당연히 서버 프로그래머가 만들어서 세팅.

로직이 클라이언트에 들어갔기때문에 서버는 할일이 없어지기 때문에, 몇 만명 단위도 커버가 가능하게된다.

예전같은 경우는 다른방법이 없어서 릴레이 서버를 운영하는 경우였으나, 지금은 처음부터 홀펀칭 자체를 시도하지않고 100%릴레이로 가는 경우도 꽤나 많음.

홀펀칭은 경우의 수가 너무 많은데다가 우리나라 인터넷환경이 좋아서 응답지연을 느끼기도 힘들기 때문.

((

RTT가 느린것이지 핑이 느린것이아니라고 지난번에 언급! 네트워크 레이턴시가 있는것이 아니라 우리가만든 서버가 느린것.

제대로 만들었다면 모니터에 주사되기도 전에 이미 업데이트 될것이다.

게임서버에서는 프레임이 돌기 때문에 거기서 느려질수밖에 없다. 그런데 p2p는 오는족족 보내버리는 것이기 때문에 느릴이유가 없다.

ex) 스트리트 파이터 아레나. - 릴레이 서버로 구현

))

**(p2p) 정리**

결국 p2p로직은 똑같은데 방향(연결의개념) 만 바뀐 것.

(클라이언트안에 로직이 들어있음)

Ex프라우드넷 같은 경우

다양한 방법을 통해서 연결을 시도하고 다시 보내줄 테니 일단 P2P로 보내라.

내부적으로 홀펀칭, 릴레이서버 등 다양한 시도를 하고있음.

홀펀칭은 더이상은 사용되지않는기술...

p2p는 빠른응답속도, 액션성강화, 서버의 부담 (로직을 클라에 넣어서) 한 장점을 이유로 사용되었다.

콘솔게임은 대부분이 핵을 쓰는것이 불가능하고, 패키지로 팔고 서버를 운영하지 않았기에 p2p를 많이 사용했다.

하지만 PC-모바일의 경우 어쩔수없이 핵이 나오기때문에 서버-클라 구조로 가야한다.

대부분 p2p가 아닌 서버-클라 구조로 넘어가고 있는추세.

**최근 MO추세**

자동으로 매칭해버리기 때문에 매치메이킹서버, 배틀서버 등 전혀 다른 구조로 가게된다.

ex) 배틀그라운드, 오버워치, 하스스톤 등.

p2p자체가 많이 사용을 안함. 로비서버에서 설정을 마치고 배틀서버중에서 어디에서 플레이 할것인지 결정해준다. 이때 한서버에 몇개의 게임이 진행될 것인지는 개발자가 정할 것.

배그같은경우 한서버에 하나의 게임만 돌고있다. (언리얼에서 따라간다)

(게임을 하는곳이 배틀서버가 된다.)