시야 처리

MM4220 게임서버 프로그래밍

정내훈

게임 서버 구현

- Standalone게임과 무엇이 다른가?
  - 동접
    - Stand-alone 게임은 동접이 1인 게임
    - MMO는 동접이 수천
  - 각각의 플레이어의 행동이 다른 플레이어에게 전달되어야 한다.
    - BroadCasting 필요

Broadcasting

- 문제
  - 클라이언트에서 게임 월드의 상황을 볼 수 있으려면?
    - 자신은 물론 다른 플레이어(Object)들의 상태도 보여주어야 함
  - 하나의 Object의 상태가 변경되면
    - 서버에 존재하는 모든 플레이어에게 변경내용을 전송해야 한다.

Broadcasting

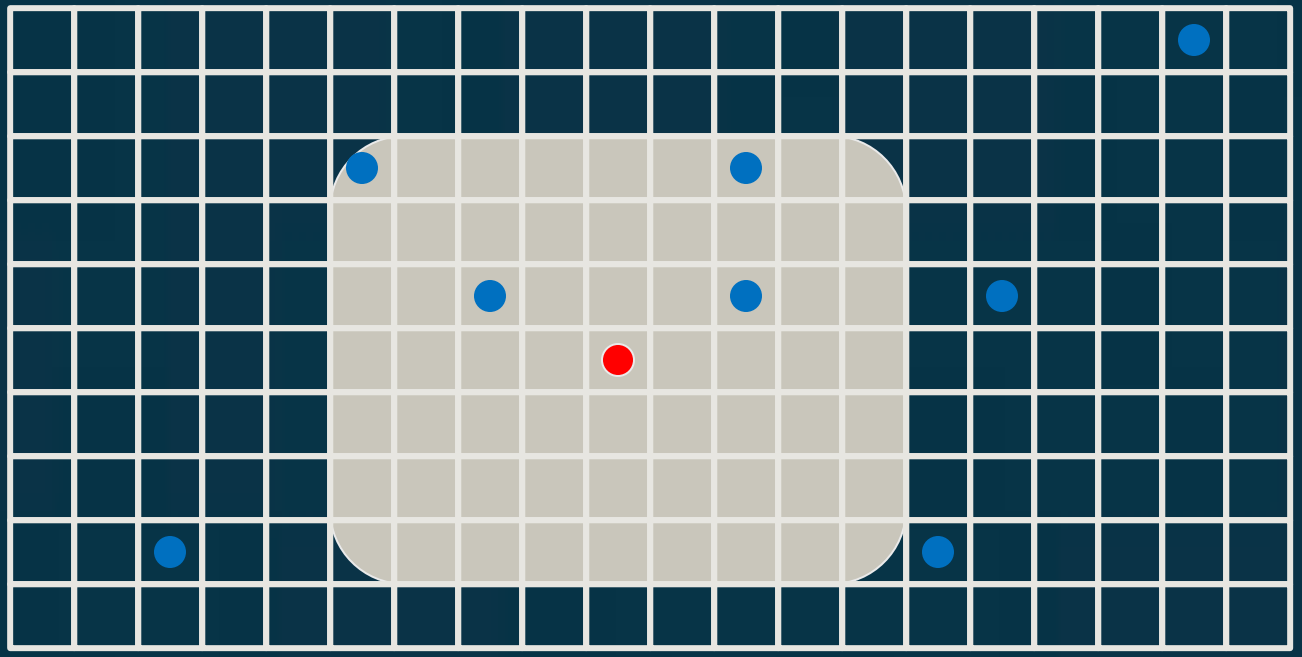
- 비용 문제
  - Object들의 상태변경을 모든 플레이어에게 보여주는 것
  - 동접 N 일때 N * N의 패킷 필요
    - 1초에 1번 이동
    - 동접 5000 => 25M packet / sec
    - 1000 CPU cycle per packet!!! => 25GHz필요
    - 패킷당 20byte => 4Gbps bandwith!!!!
  - Profiling 결과
    - BroadCasting(WSASend)이 BottleNeck

Broadcasting

- 해결책
  - Broadcasting을 최적화 한다?
    - WSASend를 우리가 최적화? Kernel최적화 필요.
    - RIO 또는 IO-uring 을 사용하여 최적화 하기도 함.
  - Broadcasting을 줄인다?
    - WSASend의 횟수를 감소 시킨다.
    - PLAN_A : 여러 개의 WSASend를 모아서 보낸다.
    - PLAN_B : 나의 움직임을 모든 플레이어에게 알려야 하는가? **NO!!!!!**
      - 나를 보고 있는 플레이어에게만 알리면 된다.

Broadcasting

- 근처의 Object? => 시야



시야처리

- 근처?
  - 시야로 필터링을 해야 함.

```
for (i=0; i<MAX_AVATAR;++i)
    if (RANGE >= Distance(avatar[i], me))
        SendState(i, me);
```

  - 효율적인 검색이 필요.
- 부하 감소
  - N * N => N * K (K는 시야내에 있는 플레이어 수)

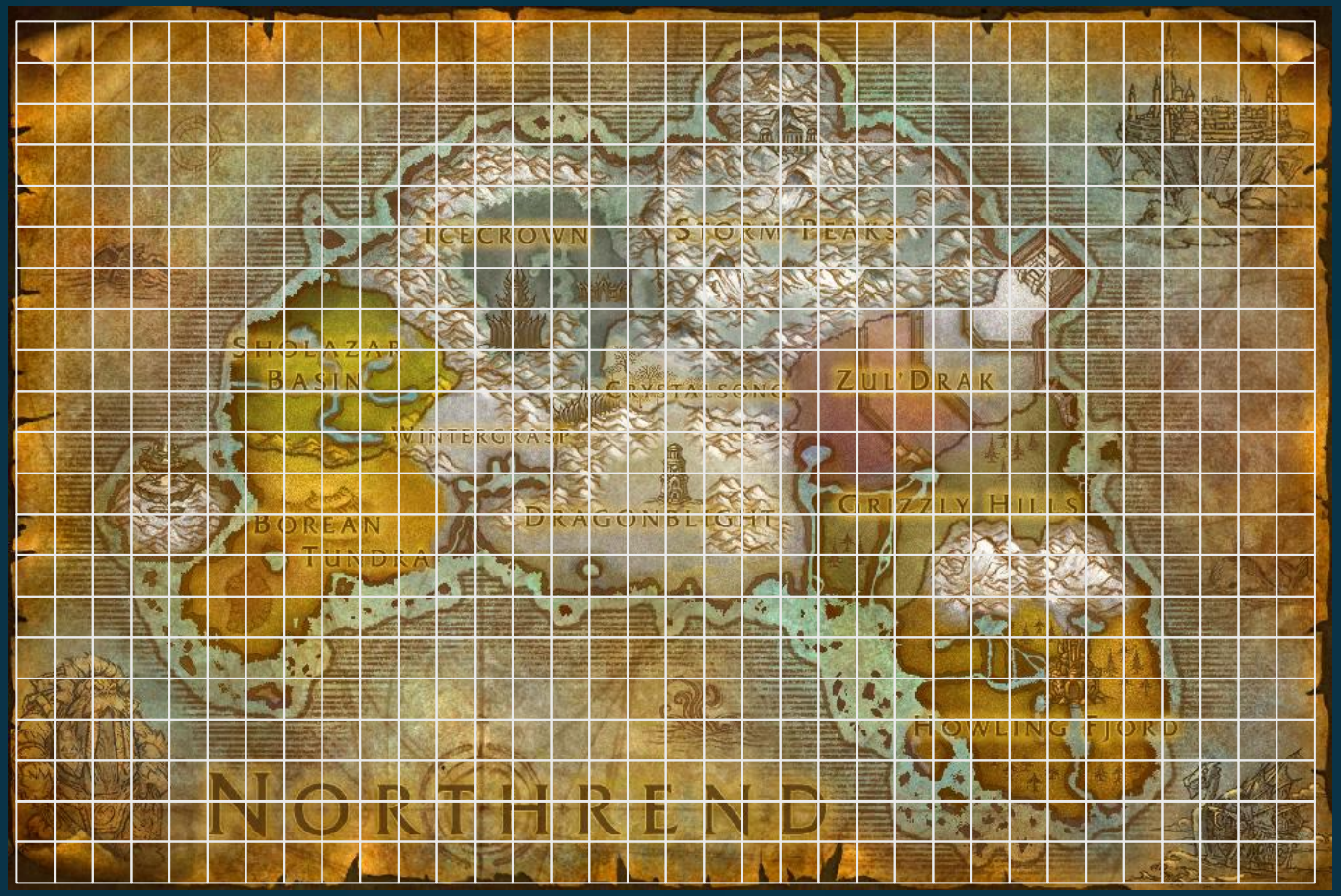
시야처리

- 효율적인 검색 => 최적화
  - Zone (용어는 게임마다 다름)
    - 효율적인 검색의 기본
    - 전체 월드를 큰 논리적인 단위로 쪼개기
      - 보통 차원이 다르거나, 바다가 가로막고 있거나...
      - Seemless하지 않아서 이동 시 로딩이 필요
    - 대륙, 월드, 인던
      - 서로 볼일이 절대 없음
  - Sector (용어는 게임마다 다름)
    - Zone도 너무 크다. 더 줄이자.
    - Cluster라고도 함.

Zone 구성

가상 분할

- Sector



노스랜드 (From World of Warcraft)

Sector 구성

- Sector는 서버 내부에서 검색효율성을 위해 도입한 개념
  - 검색 대상 object의 개수를 줄이기 위해 사용.
    - 자신과 인접 Sector만 검색
  - Sector의 크기는 적절해야 한다, 적당한 개수의 Sector검색으로 시야 범위 내의 모든 object를 찾을 수 있어야 한다.
    - 너무 크면 : 시야 범위 밖의 개체가 많이 검색됨, 검색 대상 증가
      - 병렬성이 떨어진다. (여러 스레드가 한 섹터를 공유)
    - 너무 작으면 : 많은 sector를 검색해야 한다.
      - 이동시 잦은 섹터 변경 오버헤드

Sector 구성

- 클라이언트와는 아무런 상관이 없는 개념이다.
- Sector마다 Sector에 존재하는 object의 목록을 관리하는 자료구조가 존재한다.
  - g_ObjectListSector[Row][Col]
- 모든 object의 이동/생성/소멸 시 자신이 속한 Sector의 object목록을 업데이트 하여야 한다.
  - 멀티쓰레드 자료구조 필요

시야 처리

- 모든 플레이어는 시야라는 개념을 갖고 있다.
  - 시야 밖의 object에 대한 정보는 클라이언트에 전송되지 않는다.
  - 서버 성능 향상 및 네트워크 전송량 감소에 많은 도움을 준다.

시야 처리

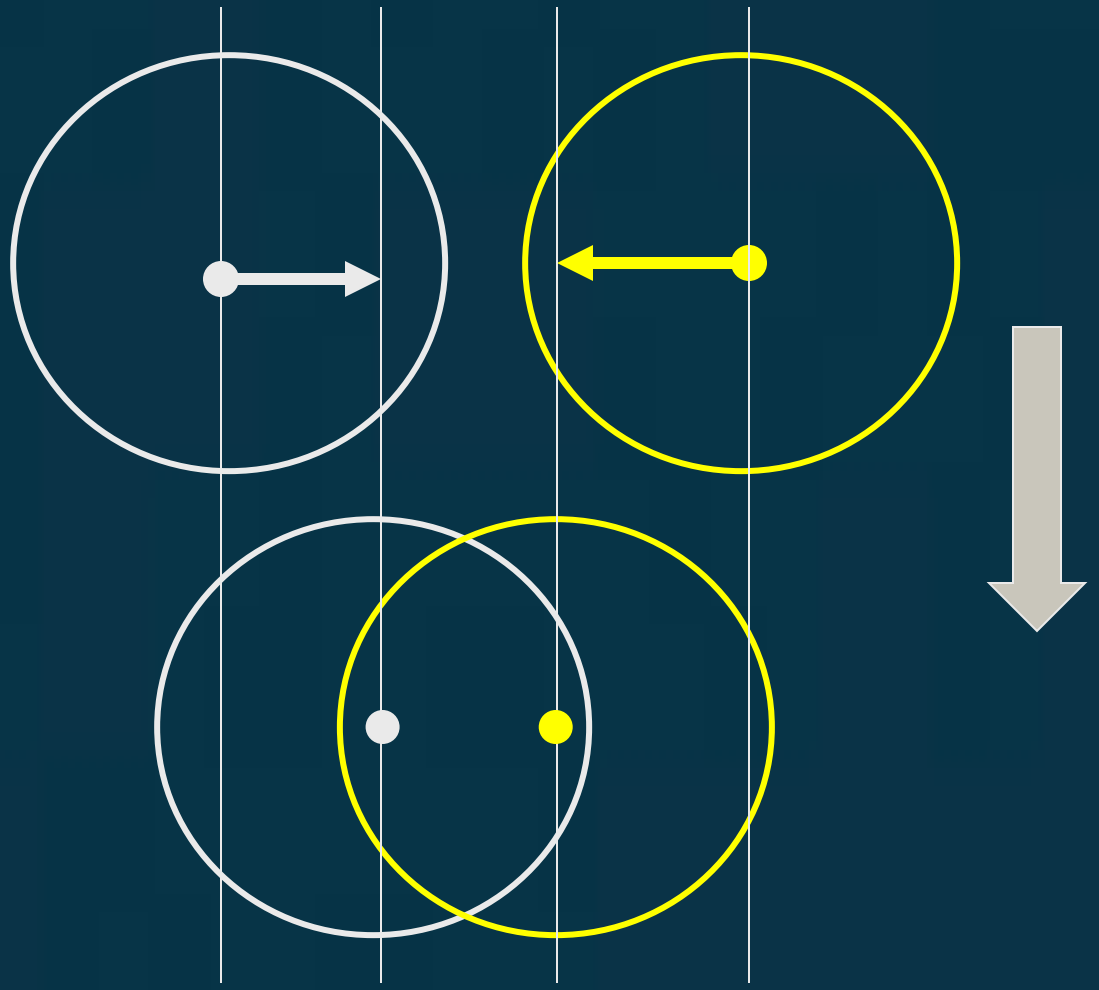
- 기본 알고리즘
  - 이동 전과 후의 시야내에 존재하는 객체 비교
  - [추가] 시야에 들어온다면 객체의 비주얼 정보를 클라이언트에 전송 (서로 전송)
  - [추가] 시야에서 사라진다면 클라이언트에 소멸 신호 전송 (서로 전송)
  - 계속 존재하는 플레이어에게는 나의 새 좌표 전송

시야 처리

- 기본 알고리즘의 문제
  - 멀티쓰레드에서 동작하므로 이동 자체에서 Data Race 효과 발생.
  - 이동 전후의 시야내 객체 리스트 비교 만으로는 올바른 동작이 불가능
  - 좀비 : 시야에서 사라졌는데 클라이언트에서 계속 존재
  - 유령 : 비주얼 정보를 받지 않은 객체가 이동, 보이지 않는 객체가 다른 객체들과 상호작용

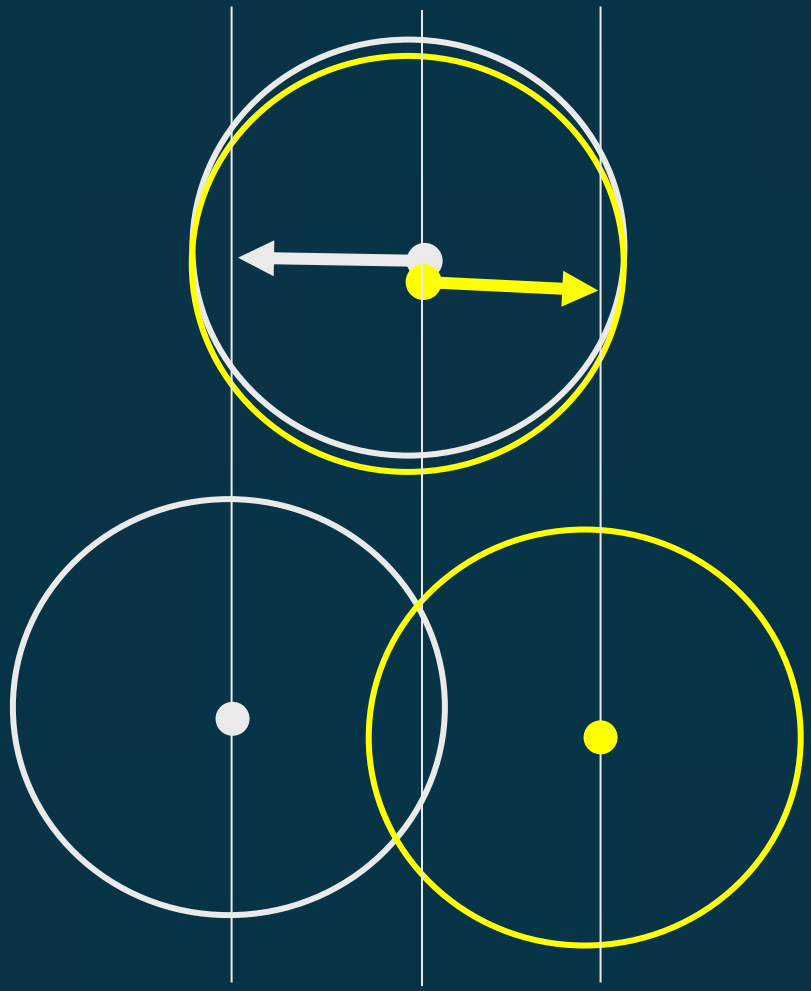
시야 처리

- 어긋남
  - 이동과 시야 리스트 업데이트는 Atomic하지 않을 경우



시야 처리

- 어긋남
  - 이동과 시야 리스트 업데이트는 Atomic하지 않을 경우



시야 처리

- 해결방법
  - 동시 이동 불허
    - Mutex 사용
    - 성능 하락 => Sector 내의 player들 끼리만 Locking으로 성능 개선
  - 보정
    - 클라이언트에 표시되는 객체들의 리스트를 서버에 저장
    - 이동 시 이 리스트를 기반으로 시야 처리
      - 유령 배제 : 리스트에 없으면 무조건 비주얼 정보 전송
      - 좀비 최소화 : 두 번째 이동에서는 무조건 삭제, 일정 시야 내에서는 좀비가 절대 존재하지 않음.

시야 처리

- 시야 리스트 (ViewList)
  - 클라이언트에서 보여주고 있는 객체의 리스트
  - 이동 시 업데이트 필요
    - 자신과 상대방 둘 다 업데이트
  - 시야에서 사라진 player에게 REMOVE_PLAYER 전송
  - 새로 시야에 들어온 player에게 PUT_PLAYER 전송
  - 계속 시야리스트에 있는 player에게 MOVE_PLAYER 전송

시야 처리

- 이동 시 시야처리 순서
  - sector 검색 후 Near List 생성
  - Near의 모든 객체에 대해
    - viewlist에 없으면
      - viewlist에 추가
      - 나<-put_pl(상대)
      - 상대 viewlist에 있으면
        - 상대 -> move_pl(나)
      - 상대 viewlist에 없으면
        - 상대 viewlist에 추가
        - 상대-> put_pl(나)
    - viewlist에 있으면
      - 상대 viewlist에 있으면
        - 상대->move_pl(나)
      - 상대 viewlist에 없으면
        - 상대 viewlist에 추가
        - 상대 -> put_pl(나)
  - ViewList에 있는 모든 객체에 대해
    - near에 없으면
      - viewlist에서 제거
      - 나<-remove_PL(상대)
      - 상대 viewlist에 있으면
        - 상대 viewlist에서 제거
        - 상대<-remove_PL(나)

시야 처리

- 최적화
  - ViewList의 Copy
    - 뷰리스트의 업데이트를 직접 하는 것이 아니라 복사하여 사용한 후 새 뷰리스트로 교체
    - lock이 걸려 있는 시간을 최소화 하여 lock으로 인한 성능저하 최소화
  - 주의
    - Dead Lock : 나의 뷰리스트와 상대의 뷰리스트를 동시에 Locking할 때 주의

11주차

- 시야처리 실습
  - 시야처리 이전 동접 약 450

숙제 (#5)

- 시야 처리를 사용한 성능 개선
  - 내용
    - 숙제 (#4)의 프로그램의 확장
      - IOCP 멀티 쓰레드 구현을 사용
      - 전체 지도는 400 x 400, 클라이언트는 16x16 표시, 시야는 자신을 중심으로 11x11, (반지름 5)
      - 이동 방향과 이동 속도를 확인할 수 있도록 클라이언트에서 맵을 표시
      - 실습시간에 작성한 시야처리 코드를 돌아가도록 수정
        - 실습시간에 만든 코드 첨부
  - 목적
    - 플레이어의 시야 구현으로 인한 성능 향상 측정, 스트레스 테스트 프로그램 사용
  - 제약
    - Windows에서 Visual Studio로 작성 할 것
  - 제출 : EClass
    - 소스코드 (서버, 클라이언트 둘다)
    - 컴파일/실행 가능 해야 함, 필요 없는 파일 제거
    - 시야처리를 적용했을 때와 안 했을 때의 성능 비교, 서버 컴퓨터 사양

다음시간

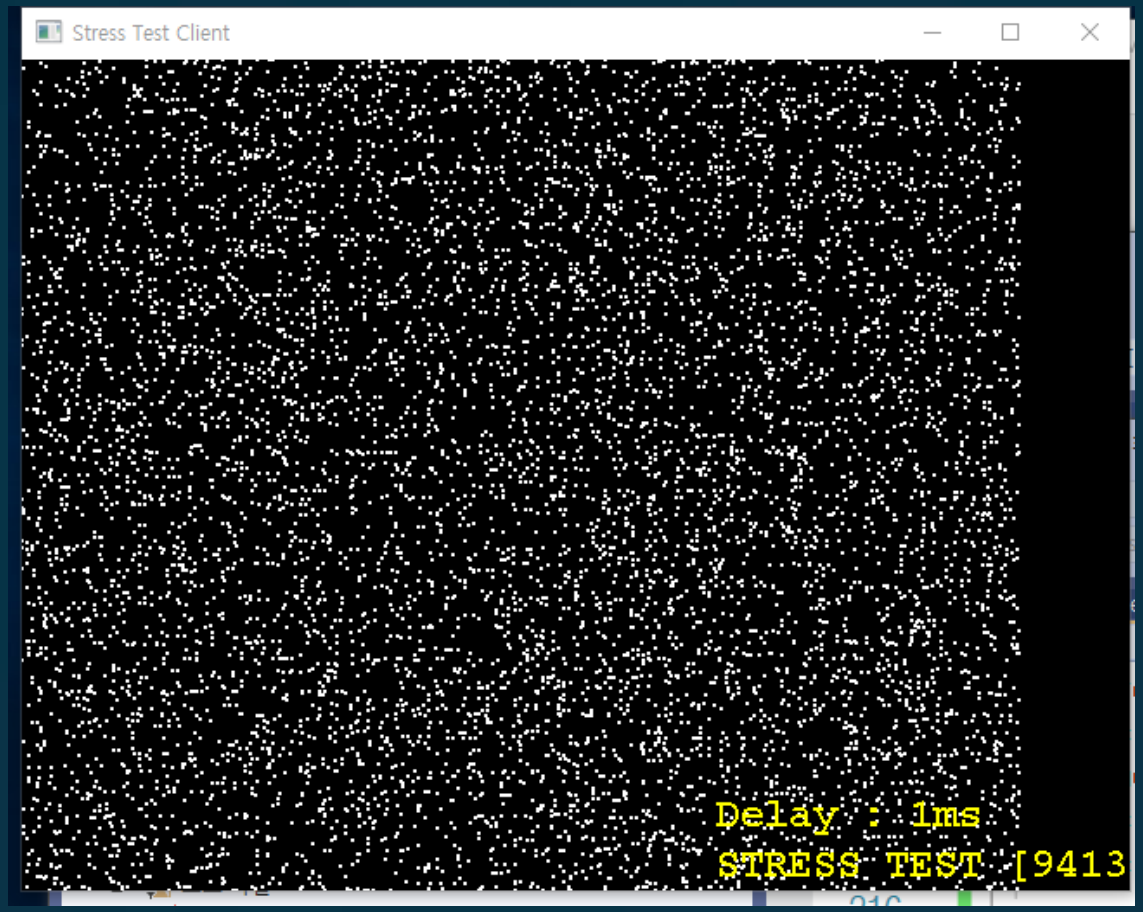
- 시야처리 실습

시야처리

- 시야처리 결과 :
  - 동접 274에서 6200이상으로 성능 향상

시야처리

- 시야처리 결과 : 9400동접에도 여유가 있음.



시야처리

- view_list 최적화
  - mutex를 통한 상호배제는 심한 성능 제한
  - 이동시 상대방의 view_list를 조작하지 않는다.
  - 대신, 상대방에게 접근/이동/이탈을 메시지로 보낸다.
  - 상대방은 메시지를 받아서 자신의 view_list를 업데이트
  - mutex가 필요 없어진다.
  - non blocking concurrent queue가 필요.

시야처리

- 섹터 분할
  - 현재 send_packet의 호출 회수는 최적화 되었음
  - 하지만 new_vl을 만들 때 모든 클라이언트를 검색하는 것은 없애지 못했음
  - 섹터 분할이 필요함.