증권 거래소

🔥 지연 시간, 처리량, 안정성 요구사항이 엄격한 주식 거래소를 설계해보자 !

▼ 1️⃣ 설계 범위 확정

기능 요구사항

- 새 주문(지정가 주문) 넣기
- 체결되지 않은 주문 취소하기
- 주문이 체결된 경우 실시간 알림
- 호가 창 정보 실시간 갱신
    - **호가 창**: 매수 및 매도 주문 목록이 표시되는 곳
- 최소 수만 명 사용자 동시 거래 가능
- 최소 100가지 주식 거래 가능
- 하루 수십억 건 주문 발생
- 위험성 점검 가능
    - 주식 하루 거래 최대 제한 규칙 점검
- 사용자 지갑 관리
    - 주문 전 충분한 자금이 있는지
    - 아직 체결되지 않은 주문이 있는 경우 해당 주문에 이용된 자금은 다른 주문에 쓰일 수 없음

비기능 요구사항

- **가용성**: 최소 99.99%
- **결함 내성**: 빠른 복구 메커니즘
- **지연 시간**: 왕복 지연 시간은 밀리초 수준이어야 함
- **보안**: 계정 관리, 신원 확인 수행, 공개 자원에 대한 DDoS 공격 방지 장치 구비

개략적 규모 추정

- 100가지 주식
- 하루 10억 건 주문
- 월 ~ 금 09:30 ~ 16:00 = 매일 6.5시간
- QPS: 10억 / 6.5시간 * 3600 = 약 43000
- 최대 QPS: 5배 (215,000)

▼ 2️⃣ 개략적 설계

증권 거래 101

- **브로커**
  - 개인 고객은 브로커 시스템을 통해 거래소와 거래한다.
  - 브로커 시스템은 개인 사용자가 증권을 거래하고 시장 데이터를 확인할 수 있도록 편리한 UI를 제공한다.
- **기관 고객**
  - 전문 증권 거래 소프트웨어를 사용하여 대량으로 거래한다.
  - 거래 빈도는 낮지만 거래량은 많다.
  - 대규모 주문이 시장에 미치는 영향을 최소화하기 위해 주문 분할 등이 필요하다.
  - 아주 낮은 응답 시간이 필요하다.
- **지정가 주문**
  - 가격이 고정된 매수 또는 매도 주문
  - 즉시 체결되지 않을수도 있음
  - 부분 체결될 수도 있음
- **시장가 주문**
  - 가격을 지정하지 않는 주문 (시장가로 즉시 체결)
  - 체결은 보장되나 비용 면에서 손해볼 수 있음
  - 급변하는 특정 시장 상황에서 유용
- **시장 데이터 수준**
  - L1 시장 데이터

        - 최고 매수 호가: 구매자가 주식에 지불할 의사가 있는 최고 가격
        - 매도 호가: 매도자가 주식을 팔고자하는 최저 가격
        - 수량
    - L2 시장 데이터
        - 깊이: 체결을 기다리는 물량의 호가를 어디까지 보여주는지 나타냄
    - L3 시장 데이터
        - 주문 가격에 체결을 기다리는 물량 정보까지 보여줌
- **봉 차트**
    - 특정 기간 동안의 주가
    - 일반적으로 지원되는 시간 간격: 1분, 5분, 1시간, 1일, 7일, 1개월
- **FIX** Financial Information Exchange Protocol
    - 금융 정보 교환 프로토콜

설계안

📈 거래 흐름

1. 고객이 브로커의 웹 또는 모바일 앱을 통해 **주문**
2. `브로커`가 주문을 거래소에 전송
3. 주문이 **클라이언트 게이트웨이**를 통해 거래소로 들어감
    - 클라이언트 게이트웨이는 입력 유효성 검사, 속도 제한 ,인증, 정규화 등과 같은 기본적 게이트키핑 기능 수행
    - 주문을 주문 관리자에게 전달
4. `주문 관리자`가 `위험 관리자`가 설정한 규칙에 따라 **위험성 점검** 수행
5. 위험성 점검 과정을 통과한 주문에 대해 `주문 관리자`는 지갑에 주문 처리 자금이 충분한지 확인
6. **주문**이 체결 엔진으로 전송
    - 체결 가능 주문이 발견되면 `체결 엔진`은 매수, 매도 측에 각각 하나씩 두 개 **집행 기록 생성**
    - 추후 과정을 재생할 때 항상 결정론적으로 동일한 결과가 나오도록 보장

7. 주문 집행 사실을 클라이언트에 전송

체결 엔진 (교차 엔진)

- 각 주식 심벌에 대한 주문서를 유지 관리
- 매수 주문과 매도 주문을 연결
- 집행 기록 스트림을 시장 데이터로 배포

시퀀서

- 체결 엔진을 결정론적으로 만드는 구성요소
- 체결 엔진에 주문을 전달하기 위해 순서 ID를 붙여 보냄 (입력 시퀀서)
- 체결 엔진이 처리를 끝낸 모든 집행 기록 쌍에 순서 ID를 붙임 (출력 시퀀서)

→ 시의성/공정성, 빠른 복구 및 재생, 정확한 1회 실행 보증

주문 관리자

- 한 쪽에서는 주문을 받고 다른 쪽에서는 집행 기록을 받아 주문 상태를 관리
- 시퀀서를 통해 체결 엔진으로부터 집행 기록을 받음
- 체결된 주문에 대한 집행 기록을 클라이언트 게이트웨이를 통해 브로커에 반환

클라이언트 게이트웨이

- 클라이언트로부터 주문을 받아 주문 관리자에게 보냄
- 주요 고려 사항: 지연 시간, 거래량, 보안 요구사항

🏬 시장 데이터 흐름

하나의 주문이 체결 엔진부터 데이터 서비스를 거쳐 브로커로 전달되어 집행되기까지의 과정

- **M1단계**
  - 체결 엔진은 주문이 체결되면 집행 기록 스트림을 만든다. (시장 데이터 `게시 서비스`로 전송)
- **M2단계**
  - 시장 데이터 게시 서비스는 집행 기록 및 주문 스트림에서 얻은 데이터를 시장 데이터로 사용하여 봉 차트와 호가 창을 구성한다. (시장 데이터 `데이터 서비스`로 전송)
- **M3단계**

- 시장 데이터는 실시간 분석 전용 스토리지에 저장된다.
- 브로커는 데이터 서비스를 통해 실시간 시장 데이터를 읽는다. (시장 데이터 `고객`에게 전송)

시장 데이터 게시 서비스

- 체결 엔진에서 집행 기록을 수신하고 집행 기록 스트림에서 호가 창, 봉 차트를 만들어냄
- 호가 창 + 봉 차트 = 시장 데이터
- 시장 데이터는 데이터 서비스로 전송되어 서비스의 구독자가 사용할 수 있게 함

👮 보고 흐름

- **R1~R2단계**
  - 보고 서비스는 주문 및 실행 기록에서 보고에 필요한 모든 필드에 값을 모은다.
  - 그 값을 종합해 만든 레코드를 데이터베이스에 기록한다.

보고 서비스

- 거래 이력, 세금 보고, 규정 준수 여부 보고, 결산 등의 기능 제공
- 정확성과 규정 준수가 핵심

API 설계

주문 (POST /v1/order)

Request

- `symbol`: 주식을 나타내는 심벌
- `side`: 매수/매도
- `price`: 지정가 주문의 가격
- `orderType`: 지정가/시장가
- `quantity`: 주문 수량

Response

- `id`: 주문 ID
- `creationTime`: 주문이 시스템에 생성된 시간

- `filledQuantity`: 집행이 완료된 수량
- `remainingQuantity`: 아직 체결되지 않은 주문 수량
- `status`: new/canceled/filled

집행 (GET /v1/execution)

Request

- `symbol`: 주식 심벌
- `orderId`: 주문 아이디
- `startTime`: 질의 시작 시간
- `endTime`: 질의 종료 시간

Response

- `executions`: 범위 내 모든 집행 기록의 배열
- `id`: 집행 기록 아이디
- `orderId`: 주문 아이디
- `symbol`: 주식 심벌
- `side`: 매수/매도
- `price`: 체결 가격
- `orderType`: 지정가/시장가
- `quantity`: 체결 수량

호가 주문서 (GET /v1/marketdata/orderBook/L2)

Request

- `symbol`: 주식 심벌
- `depth`: 반환할 호가 창의 호가 깊이
- `startTime`: 질의 시작 시간
- `endTime`: 질의 종료 시간

Response

- `bids`: 가격과 수량 정보를 담은 배열
- `asks`: 가격과 수량 정보를 담은 배열

가격 변동 이력(봉 차트) (GET /v1/marketdata/candles)

Request

- symbol: 주식 심벌
- resolution: 봉 차트의 윈도 길이
- startTime: 질의 시작 시간
- endTime: 질의 종료 시간

Response

- candles: 각 봉의 데이터를 담은 배열
- open: 해당 봉의 시가
- close: 해당 봉의 종가
- high: 해당 봉의 고가
- low: 해당 봉의 저가

데이터 모델

상품, 주문, 집행

- **상품**
  - 거래 대상 주식(심벌)이 가진 속성으로 정의
  - 자주 변경되지 않고 주로 UI 표시를 위한 데이터
    - 캐시 적용이 효과적
- **주문**
  - 매수 또는 매도를 실행하라는 명령
- **집행**
  - 체결이 이루어진 결과
- **주문, 집행은 거래소가 취급하는 가장 중요한 데이터**
  - 중요 거래 경로는 주문/집행 기록을 DB에 저장하지 않음
  - 성능 향상을 위해 메모리에서 거래를 체결하고 하드디스크나 공유 메모리에 저장하고 공유함

호가 창/주문서

- 일정한 조회 시간
- 빠른 추가/취소/실행 속도
- 빠른 업데이트
- 최고 매수 호가/최저 매도 호가 질의
- 가격 수준 순회

⇒ 효율적인 호가 창을 만들기 위해 orders의 자료 구조는 이중 연결 리스트로 변경하여 O(1)에 처리되도록 한다.

봉 차트

- 시장 데이터 프로세서가 시장 데이터를 만들 때 호가 창과 더불어 사용하는 핵심 자료구조
- 많은 종목의 가격 이력을 다양한 시간 간격을 사용해 추적하려면 많은 메모리가 필요
  - 미리 메모리를 할당해 둔 링 버퍼에 봉을 보관하여 새 객체 할당 수를 줄인다.
  - 메모리에 두는 봉의 개수를 제한하고 나머지는 디스크에 보관한다.

▼ 3️⃣ 상세 설계

성능

평균 지연 시간은 낮아야 하고 전반적인 지연 시간 분포는 안정적이어야 한다 !

지연 시간을 줄이는 방법

1. **중요 경로에서 실행할 작업 수 줄이기**

   게이트웨이 → 주문 관리자 → 시퀀서 → 체결 엔진

   - 꼭 필요한 구성 요소만 둘 것
   - 로깅도 뺄 것

2. **각 작업의 소요 시간 줄이기**

   - 네트워크 / 디스크 액세스 지연 시간을 모두 고려하면 총 end-to-end 지연 시간은 수십 밀리초!
   - 모든 것을 동일한 서버에 배치해서 네트워크 구간을 없앤다.

- 같은 서버 내 컴포넌트 간 통신은 이벤트 저장소 mmap을 통한다.

애플리케이션 루프

while문을 통해 실행할 작업을 계속 폴링하는 것

- 엄격한 지연 시간 요건 만족을 위해 가장 중요한 작업만 순환문 안에서 처리
- 각 구성 요소의 실행 시간을 줄여 전체 실행 시간 예측이 가능하도록 보장
- 단일 스레드로 구현, 특정 CPU 코어에 고정시킴
  - 👍🏻 문맥 전환이 없음
  - 👍🏻 락이 필요 없음
  - 👎 코딩이 더 복잡해짐

이벤트 소싱

현재 상태를 저장하는 대신 상태를 변경하는 모든 이벤트에 immutable 로그 유지

- 게이트웨이는 각 주문을 이벤트 저장소 클라이언트를 사용하여 전송
- 체결 엔진의 내장된 주문 관리자는 이벤트 저장소로부터 이벤트를 수신
- 체결 엔진은 유효성을 검사한 다음 내부 주문 상태에 추가
- 해당 주문은 처리 담당 CPU 코어로 전송
- 주문이 체결되면 이벤트가 생성되어 이벤트 저장소로 전송
- 다른 구성요소는 이벤트 저장소를 구독하고 이벤트를 받아 적절히 처리

더 효율적인 동작을 위해 ... 😎

- **주문 관리자**
  - 컴포넌트에 내장되는 재사용 가능 라이브러리
  - 중앙화된 주문 관리자 이용 시 지연 시간이 길어질 수 있으므로
- **시퀀서**
  - 이벤트 소싱 아키텍처를 따르면 모든 메시지는 동일한 이벤트 저장소를 사용하므로 이벤트 저장소에 있는 시퀀서가 값을 넣어줄 수 있다.
  - 시퀀서가 여러 개면 권한으로 경쟁하게 되므로 하나의 시퀀서만 사용
  - 시퀀서가 다운될 경우를 대비해 백업 시퀀서를 두어 가용성 향상

고가용성

SPOF 식별

- 무상태 서비스의 경우 수평적 확장
- 상태를 저장하는 컴포넌트는 사본 간 상태 데이터를 복사해야 함
    - 이벤트 저장소 데이터를 사용해 모든 상태 복구

장애 감지

- 하드웨어와 프로세스 모니터링 → 일반적 방법
- 체결 엔진과 박동 메시지를 주고받는 방안
    - 박동 메시지를 시간 내 받지 못하면 문제가 있는 것으로 판단

결함 내성

주/부 설계안에서 부 서버까지 다운된다면?

1. **주 서버가 다운되면 언제, 어떻게 부 서버로 자동 전환 결정을 내리는가?**
    - 시스템에서 경보 전송 시 부 시스템으로 자동 전환
    - 코드 버그의 경우 부 서버로 전환해도 같은 버그 발생 가능

    → 일단 수동 장애 복구 조치를 수행하고 자신감이 생겼을 때 자동 복구 프로세스를 도입하자.
2. **부 서버 중 새로운 리더는 어떻게 선출하는가?**
    - 래프트 메커니즘
3. **복구 시간 목표는 얼마인가?**
    - 2등급 RTO 달성 필요 → 자동 복구 필수
    - 성능 저하 전략
4. **어떤 기능을 복구해야 하는가?**
    - 손실 허용 범위가 0에 가까움
    - 래프트 메커니즘을 사용하면 데이터 사본이 많음

체결 알고리즘

FIFO 체결 알고리즘 사용

- 특정 가격 수준에서 먼저 들어온 주문이 먼저 체결
- 선물 거래에 흔히 사용

결정론

기능적 결정론

- 시퀀서나 이벤트 소싱 아키텍처를 도입해 이벤트를 동일한 순서로 실행 시 항상 같은 결과를 얻는 걸 보장하는 것
- 이벤트 시간보다 순서가 중요

지연 시간 결정론

- 각 거래의 처리 시간이 거의 같다
- 지연 시간 변동 폭이 커지면 원인 조사 필요
    - JVM Stop-the-world

시장 데이터 게시 서비스 최적화

체결 엔진의 체결 결과를 기반으로 호가 창과 봉 차트를 재구축한 후 구독자에게 게시

- 링 버퍼 활용
    - 앞과 끝이 연결된 고정 크기 큐
    - 생산자는 계속 데이터를 넣고 소비자는 데이터를 꺼냄
    - 락을 사용하지 않음
- 패딩
    - 링 버퍼의 순서 번호가 다른 것과 같은 캐시 라인에 오지 않도록 함

시장 데이터의 공정한 배포

다른 사람보다 지연 시간이 낮다 == 미래 예측이 가능하다

- 모든 수신자가 동시에 시장 데이터를 받는 것을 보장하는 것이 중요
    - 멀티캐스트: 한 번에 많은 참가자에게 업데이트를 브로드캐스트하기 좋은 솔루션
        - 유니캐스트: 하나의 출처에서 하나의 목적지로만 보내는 전송 프로토콜
        - 브로드캐스트: 하나의 출처에서 전체 하위 네트워크로 전송
        - 멀티캐스트: 하나의 출처에서 다양한 하위 네트워크상 호스트로 전송

코로케이션

체결 엔진에 주문을 넣는 지연 시간은 전송 경로 길이에 비례

네트워크 보안

1. 공개 서비스와 데이터를 비공개 서비스에서 분리하여 DDoS 공격에 영향받지 않도록 함
2. 동일한 데이터를 제공해야 하는 경우 읽기 전용 사본을 여러 개 만들어 문제 격리
3. 자주 업데이트되지 않는 데이터는 캐싱
4. 디도스 공격에 대비해 URL 강화
5. 효과적인 허용/차단 리스트 메커니즘 사용
6. 처리율 제한 기능 활용