3️⃣

증권 거래소

문제 이해 및 설계 범위 확정

비기능 요구사항

- 가용성
  - 최소 99.99%
  - 단 몇 초의 장애로도 평판이 손상될 수 있음
- 결함 내성
  - 프로덕션 장애의 파급을 줄이려면 결함 내성과 빠른 복구 메커니즘이 필요함
- 지연 시간
  - 왕복 지연 시간은 밀리초 수준이어야 함
    - 주문이 거래소에 들어오는 순간부터 주문의 체결 사실이 반환되는 시점까지
  - p99(99th 백분위수) 지연 시간이 중요함
    - p99 지연 시간이 계속 높으면 일부 사용자의 거래소 이용 경험이 아주 나빠짐
- 보안
  - 계정 관리 시스템을 갖추고 있어야 함
  - 법률 및 규정 준수를 위해 거래소는 새 계좌 개설 전에 사용자 신원 확일을 위한 KYC 확인을 수행함
  - 시장 데이터가 포함된 웹 페이지 등의 공개 자원의 경우에는 DDos 공격을 방지하는 장치를 구비해 두어야 함

개략적 규모 추정

- 100가지 주식
- 하루 10억 건의 주문
- 월요일부터 금요일, 오전 9시 30분 부터 오후 4시

- QPS = 10억 / (6.5시간 * 3600) = ~43,000
- 최대 QPS = 5 * QPS = 215,000
- 거래량은 장 시작 직후와 장 마감 직전이 제일 많음

개략적 설계안 제시 및 동의 구하기

증권 거래 101

- 브로커
  - 대부분의 개인 고객은 브로커 시스템을 통해 거래함
  - 개인 사용자가 증권을 거래하고 시장 데이터를 확인할 수 있도록 편리한 사용자 인터페이스를 제공함
- 기관 고객
  - 전문 증권 거래 소프트웨어를 사용하여 대량으로 거래함
  - 기관 고객마다 거래 시스템에 대한 요구사항이 다름
  - 주문 분할 같은 기능을 필요로 하기도 함
  - 낮은 응답 시간으로 거래하길 원함
  - 일반 사용자들처럼 웹페이지나 모바일 앱에서 시장 데이터를 확인하게 하면 곤란함
- 지정가 주문
  - 가격이 고정된 매수 또는 매도 주문
  - 시장가 주문과 달리 체글이 즉시 이루어지지 않을 수도 있음
  - 부분적으로만 체결될 수도 있음
- 시장가 주문
  - 가격을 지정하지 않는 주문
  - 시장가로 즉시 체결됨
  - 체결은 보장되나 비용 면에서는 손해를 볼 수 있음
- 시장 데이터 수준
  - 미국 주식 시장에는 L1, L2, L3 세 가지 가격 정보 등급이 있음

    - L1: 최고 매수 호가, 매도 호가 및 수량이 포함
    - L2: 더 많은 수준의 가격 정보(체결을 기다리는 물량의 호가)가 제공됨
    - L3: L2 + 각 주문 가격에 체결을 기다리는 물량 정보까지 보여줌
- 봉 차트
    - 특정 기간 동안의 주가
    - 하나의 봉 막대로 일정 시간 간격 동안 시장의 시작가, 종가, 최고가, 최저가를 표시할 수 있음
    - 일반적으로 지원되는 시간 간격은 1분, 5분, 1시간, 1일, 1주일, 1개월
- FIX
    - Financial Information Exchange Protocol
    - 금융 정보 교환 프로토콜
    - 증권 거래 정보 교환을 위한 기업 중립적 통신 프로토콜

개략적 설계안

전체적인 동작 원리

거래 흐름: 하나의 주문이 어떤 절차로 처리되는지

1. 고객이 브로커의 웹 또는 모바일 앱을 통해 주문함
2. 브로커가 주문을 거래소에 전송
3. 주문이 클라이언트 게이트웨이를 통해 거래소로 들어감
    a. 클라이언트 게이트웨이는 입력 유효성 검사, 속도 제한, 인증, 정규화 등과 같은 기본적인 게이트키핑 기능을 수행함
    b. 주문을 주문 관리자에게 전달
4. 주문 관리자가 위험 관리자가 설정한 규칙에 따라 위험성 점검을 수행
5.
6. 위험성 점검 과정을 통과한 주문에 대해, 주문 관리자는 지갑에 주문 처리 자금이 충분한지 확인함
7. 주문이 체결 엔진으로 전송됨

8. 체결 가능 주문이 발견되면 체결 엔진은 매수 측과 매도 측에 각각 하나씩 두개의 집행 기록을 생성함
9. 나중에 그 과정을 재생할 때 항상 결정론적으로 동일한 결과가 나오도록 보장하기 위해 시퀀서는 주문 및 집행 기록을 일정 순서로 정렬함
10. ~14 주문 집행 사실을 클라이언트에 전송함

시장 데이터 흐름(M): 하나의 주문이 체결 엔진부터 데이터 서비스를 거쳐 브로커로 전달되어 집행되기까지

1. 체결 엔진은 주문이 체결되면 집행 기록 스트림을 만들고, 이 스트림은 시장 데이터 게시 서비스로 전송됨
2. 시장 데이터 게시 서비스는 기록 및 주문 스트림에서 얻은 데이터를 시장 데이터로 사용하여 봉 차트와 호가 창을 구성하고, 시장 데이터를 데이터 서비스로 전송
3. 시장 데이터는 실시간 분석 전용 스토리지에 저장됨
    a. 브로커는 데이터 서비스를 통해 실시간 시장 데이터를 읽음
    b. 브로커는 이 시작 데이터를 고객에게 전달함

보고 흐름(R)

1. ~2 보고 서비스는 주문 및 실행 기록에서 보고에 필요한 모든 필드의 값을 모든 다음 그 값을 종합해 만든 레코드를 데이터베이스에 기록함

전체적인 동작 원리에 대한 상세

거래 흐름

- 체결 엔진
    - 교차 엔진
    - 각 주식 심벌에 대한 주문서 내지 호가 창을 유지 관리함
        - 주문서 또는 호가 창은 특정 주식에 대한 매수 및 매도 주문 목록임
    - 매수 주문과 매도 주문을 연결함
        - 주문 체결 결과로 두 개의 집행 기록이 만들어짐
        - 체결을 빠르고 신속하게 처리되어야 함

    - 집행 기록 스트림을 시작 데이터로 배포함
- 시퀀서
    - 체결 엔진을 결정론적으로 만드는 핵심 구성 요소
    - 체결 엔진에 주문을 전달하기 전에 순서 ID를 붙여서 보냄
    - 체결 엔진이 처리를 끝낸 모든 집행 기록 쌍에도 순서 ID를 붙임
    - 입력 시퀀서와 출력 시퀀서 두 가지가 있으며, 각각 고유한 순서를 유지함
    - 누락된 항목을 쉽게 발견할 수 있는 일련번호여야 함
    - 입력되는 주문과 출력하는 실행 명령에 순서 ID을 찍는 이유
        - 시의성 및 공정성
        - 빠른 복구 및 재생
        - 정확한 1회 실행 보증
    - 메시지 큐의 역할도 함
        - 체결 엔진에 메시지(수신된 주문)를 보내는 역할
        - 주문 관리자에게 메시지(집행 기록)를 보내는 역할
    - 주문과 집행 기록을 위한 이벤트 저장소의 역할도 함
        - 체결 엔진에 두 개 카프카 이벤트 스트림이 연결되어 있는 것과 비슷
        - 하나는 입력되는 주문용이고 다른 하나는 출력될 집행 기록용
- 주문 관리자
    - 한쪽에서는 주문을 받고 다른 한쪽에서는 집행 기록을 받음
    - 주문 상태를 관리
    - 클라이언트 게이트웨이를 통해 주문을 수신하고 다음을 실행
        - 종합적으로 위험 점검 담당 컴포넌트에 주문을 보내어 위험성을 검토
        - 사용자의 지갑에 충분한 자금이 있는지 확인
        - 주문을 시퀀서에 전달
    - 시퀀서를 통해 체결 엔진으로부터 집행 기록을 받음
    - 체결된 주문에 대한 집행 기록을 클라이언트 게이트웨이를 통해 브러커에 반환
    - 빠르고 효율적이며 정확해야 함
    - 주문의 현재 상태를 유지 관리함

    - 이벤트 소싱이 주문 관리자 설계에 적합함
- 클라이언트 게이트웨이
    - 거래소의 문지기
    - 클라이언트로부터 주문을 받아 주문 관리자에게 전송
    - 중요 경로상에 놓임
    - 지연 시간에 민감
    - 어떤 기능을 클라이언트 게이트웨이에 넣을지 말지는 타협적으로 생각해야 함
        - 일반적으로 적용 가능한 원칙은 복잡한 기능이라면 체결 엔진이나 위험 점검 컴포넌트에 맡겨야 한다는 것
    - 고객 유형별로 클라이언트 게이트웨이는 다양함

시장 데이터 흐름

- 시장 데이터 = 호가 창 + 봉 차트
- 시장 데이터 게시 서비스는 체결 엔진에서 집행 기록을 수신하고 집행 기록 스트림에서 시장 데이터를 만듦
- 시장 데이터는 데이터 서비스로 전송되어 해당 서비스의 구독자가 사용할 수 있게 됨

보고 흐름

- 보고 서비스는 거래의 중요 경로상에 있지는 않지만 여전히 시스템의 중요한 부분
- 거래 이력, 세금 보고, 규정 준수 여부 보고, 결산 등의 기능을 제공
- 정확성과 규정 준수가 핵심임
- 입력으로 들어오는 주문과 그 결과로 나가는 집행 기록 모두에서 정보를 모아 속성들을 구성하는 것이 일반덕인 관행
- 들어오는 새 주문 정보과 나가는 집행 기록 정보를 잘 병합하여 보고서를 만듦
    - 들어오는 새 주문에는 주문 세부 정보만
    - 나가는 집행 기록에는 주문 ID, 가격, 수량 및 집행 상태 정보만 있음

API 설계

POST	/v1/order	주문을 처리함
		인증 필요
			request
			{
			”symbol”: 주식을 나타내는 심벌,
			”side”: 매수(buy) 또는 매도(sell),
			”price”: 지정가 주문의 가격,
			”orderType”: 지정가(limit) 또는 시장가(market),
			”quantity”: 주문 수량
			}
			
			response
			{
			”id”: 주문 Id,
			”creationTime”: 주문이 시스템에 생성된 시간,
			”filledQuantity”: 집행이 완료된 수량,
			”remainingQuantity”: 아직 체결되지 않은 주문 수량,
			”status”: new/canceld/filled
			}

GET	/v1/execution	집행 정보를 질의함
		인증 필요
			param
			{
			”symbol”: 주식을 나타내는 심벌,
			”orderId”: 주문 Id,
			”startTime”: 질의 시작 시간,
			”endTime”: 질의 종료 시간
			}
			
			response
			{

			”executions”: 범위 내 모든 집행 기록의 배열, ”id”: 집행 기록 Id, ”orderId”: 주문 Id, ”symbol”: 주식을 나타내는 심벌, ”side”: 매수(buy) 또는 매도(sell), ”price”: 체결 가격, ”orderType”: 지정가(limit) 또는 시장가(market), ”quantity”: 주문 수량 }

GET	/v1/marketdata/orderBook/L2	주어진 주식 심벌, 주어진 깊이 값에 대한 L2 호가 창 질의 결과 반환	**param**
			{
			”symbol”: 주식을 나타내는 심벌,
			”depth”: 반환할 호가창의 호가 깊이,
			”startTime”: 질의 시작 시간,
			”endTime”: 질의 종료 시간
			}
			
			response
			{
			“bids”: 가격과 수량 정보를 담은 배열,
			“asks”: 가격과 수량 정보를 담은 배열
			}

GET	/v1/marketdata/candles	주어진 시간 범위, 해상도, 심벌에 대한 봉 차트 데이터 질의 결과를 반환	**param** { ”symbol”: 주식을 나타내는 심벌, ”resolution”: 봉 차트의 윈도 길이(초 단위), ”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 정의됨
  - 데이터가 자주 변경되지 않음
  - UI 표시를 위한 데이터
  - 아무 데이터베이스에나 저장 가능
  - 캐시를 적용하기 좋음
- 주문
  - 매수 또는 매도를 실행하라는 명령
- 집행 기록
  - 체결이 이루어진 결과
  - 충족이라고도 부름
  - 모든 주문이 집행되지는 않음
  - 체결 엔진은 하나의 주문 체결에 관여한 매수 행위와 매도 행위를 나타내는 두개의 집행 기록의 결과로 출력
- 세 정보 사이의 관계

    - Product 1 : 1 Order 1 : n Execution
- 주문과 집행 기록은 거래소가 취급하는 가장 중요한 데이터
- 중요 거래 결로는 주문과 집행 기록을 데이터베이스에 저장하지 않음
    - 성능을 높이기 위해 메모리에서 거래를 체결하고 하드디스크나 공유 메모리를 활용하여 주문과 집행 기록을 저장하고 공유함
    - 주문과 집행 기록은 빠른 복구를 위해 시퀀서에 저장
    - 데이터 보관은 장 마감 후에 실행
- 보고 서비스는 조정이나 세금 보고 등을 위해 데이터베이스에 주문 및 집행 기록을 저장함
- 집행 기록은 시장 데이터 프로세서로 전달되어 호가 창/주문서와 봉 차트 데이터 재구성에 쓰임

호가 창/주문서

- 특정 증권 또는 금융 상품에 대한 매수 및 매도 주문 목록
- 가격 수준별로 정리되어 있음
- 체결 엔진이 빠른 주문 체결을 위해 사용하는 핵심 자료 구조
- 호가 창의 자료 구조가 만족해야 하는 요구사항
    - 일정한 조회 시간
        - 특정 가격 수준의 주문량 조회, 특정 가격 범위 내의 주문량 조회, ...
    - 빠른 추가/취소/실행 속도
        - O(1) 시간 복잡도를 만족해야 함
        - 새 주문 넣기, 시존 주문 취소하기, 주문 체결하기, ...
    - 빠른 업데이트
        - 주문 교체, ...
    - 최고 매수 호가/최저 매도 호가 질의
    - 가격 수준 순회
- 주문이 집행되는 과정(애플 주식 2,700주에 대한 대량 시장가 매수 주문)
    1. 최저 매도 호가 큐의 모든 매도 주문과 체결된 후에

2. 호가 100.11 큐의 첫 번째 매도 주문과 체결되며 거래가 끝남
3. 대량 주문의 체결 결과로 매수/매도 호가 스프레드(둘 간의 가격 차이)가 넓어지고 주식 가격은 한 단계 상승

```
class PriceLevel {
    private Price limitPrice;
    private long totalVolume;
    private List<Order> orders;
}

class Book<Side> {
    private Side side;
    private Map<Price, PriceLevel> limitMap;
}

class OrderBook {
    private Book<Buy> buyBook;
    private Book<Sell> sellBook;
    private PriceLevel bestBid;
    private PriceLevel bestOffer;
    private Map<OrderId, Order> orderMap;
}
```

- 이 코드는 요구사항을 만족할 수 있을까? ⇒ 아니요
  - orders의 자료 구조를 이중 연결 리스트로 변경하여 모든 삭제 연산이 O(1)에 처리되도록 해야 함

- 시간 복잡도가 O(1)이 되는 이유
  - 새 주문을 넣는다는 것은 PriceLevel 리스트 마지막에 새 Order를 추가한다는 뜻
    - 이중 연결 리스트의 경우 이 연산의 시간 복잡도는 O(1)
  - 주문을 체결한다는 것은 PriceLevel 리스트의 맨 앞에 있는 Order를 삭제한다는 듯
    - 이중 연결 리스트의 경우 이 연산의 시간 복잡도는 O(1)
  - 주문을 취소한다는 것은 OrderBook에서 Order를 삭제한다는 뜻
    - OrderBook에 포함되어 있는 orderMap을 활용하면 O(1) 시간 내에 취소할 주문을 찾을 수 있음

        - 주문을 찾았더라도 orders가 단일 연결 리스트였더라면 전체 목록을 순회하여 이전 포인터를 찾아야 주문을 삭제 가능 = O(n)
        - 이중 연결 리스트의 경우 Order 안에 이전 주문을 가르키는 포인터가 있으므로 전체 목록을 순회할 필요가 없음

봉 차트

- 시장 데이터 프로세서가 시장 데이터를 만들 때 호가 창과 더불어 사용하는 핵심 구조
- 봉 차트를 모델링하기 위해서 Candelstick 클래스와 CandlestickChart 클래스를 사용함

```
class Candlestick {
    private long openPrice;
    private long closePrice;
    private long highPrice;
    private long lowPrice;
    private long volume;
    private long timestamp;
    private int interval;
}

class CandlestickChart {
    private LinkedList<Candlestick> sticks;
}
```

- 하나의 봉이 커퍼하는 시간 범위가 경과하면 다음 주기를 커버할 새 Candlestick 클래스 객체를 생성하여 CandlestickChart 객체 내부 연결리스트에 추가함
- 봉 차트에서 많은 종목의 가격 이력을 다양한 시간 간격을 사용해 추적하려면 많은 메모리가 필요함
- 최적화 방법
    - 미리 메모리를 할당해 둔 링 버퍼에 봉을 보관하면 세 객체 할당 횟수를 줄일 수 있음
    - 메모리에 두는 봉의 개수를 제한하고 나머지는 디스크에 보관
- 시장 데이터는 일반적으로 실시간 분석을 위해 메모리 상주 칼럼형 데이터베이스에 둠
- 시장이 마감된 후에는 데이터를 이력 유지 전용 데이터베이스에도 저장함

상세 설계

성능

지연 시간을 줄이는 방법

- 중요 경로에서 실행할 작업 수를 줄인다
  - 중요 경로에는 꼭 필요한 구성 요소만 둠
  - 로깅도 지연 시간을 줄이기 위해 중요 경로에서는 뺌
  - 중요 매매 경로
    - 게이트웨이 → 주문 관리자 → 시퀀서 → 체결 엔진
- 각 작업의 소요 시간을 줄인다
  - 네트워크 및 디스크 사용량 경감
    - 모든 거래소가 지연 시간을 극도로 낮추는 경쟁에 나서게 됨
    - 모든 것을 동일한 서버에 배치하여 네트워크를 통하는 구간을 없앰
    - 같은 서버 내 컴포넌트 간 통신은 이벤트 저장소인 mmap을 통함
  - 각 작업의 실행 시간 경감

모든 구성 요소를 단일 서버에 배치하여 낮은 지연 시간을 달성하는 설계안

애플리케이션 루프

- while 순한문을 통해 실행할 작업을 계속 폴링하는 것이 주된 실행 메커니즘
- 엄격한 지연 시간 요건을 만족하려면 목적 달성에 가장 중요한 작업만 순환문 안에서 처리해야 함
- 각 구성 요소의 실행 시간을 줄여 전체적인 실행 시간이 예측 가능하도록 보장하는 것이 목표
- CPU 효율성을 극대화하기 위해 애플리케이션 루프는 단일 스레드로 구성되며, 특정 CPU 코어에 고정시킴
- 애플리케이션 루프를 CPU에 고정할 때 장점
  - 문맥 전환(컨텍스트 스위치)가 없음

    - 상태를 업데이트하는 스레드가 하나뿐이라서 락을 사용할 필요도, 잠금 경합도 없음
- 애플리케이션 루프를 CPU에 고정할 때 단점
    - 코딩이 복잡해짐
    - 각 작업이 애플리케이션 루프 스레드를 너무 오래 점유하지 않도록 각 작업에 걸리는 시간을 신중하게 분석해야 함

mmap

- 파일을 프로세스의 메모리에 매핑하는 mmap(2)라는 이름의 POSIX 호환 UNIX 시스템 콜
- 프로세스 간 고성능 메모리 공유 메커니즘을 제공
- 매핑할 파일이 /dev/shm(메모리 기반 파일 시스템)에 있을 때 성능 이점이 더욱 커짐
    - 공유 메모리에 접근해도 디스트 I/O는 발생하지 않음
- 최신 거래소는 이를 활용하여 중요 경로에서 가능한 한 디스크 접근이 일어나지 않도록 함
- 서버에서 mmap(2)를 사용하여 중요 경로에 놓인 구성 요소가 서로 통신할 때 이용할 메시지 버스를 구현

이벤트 소싱

- 전통적 애플리케이션은 상태를 데이터베이스에 유지하는데 문제가 발생할 경우 원인을 추적하기 어려움
    - 데이터베이스에는 현재 상태만 유지하고 현재 상태를 초래한 이벤트의 기록은 없기 때문
- 이벤트 소싱 아키텍처는 현재 상태를 저장하는 대신 상태를 변경하는 이벤트의 변경 불가능한 로그를 유지함
    - 이 로그를 절대적 진실의 원천으로 삼는 것
- 주문 상태를 변경하는 모든 이벤트를 추적하기 때문에 모든 이벤트를 순서대로 재생하면 주문 상태를 복구할 수 있음

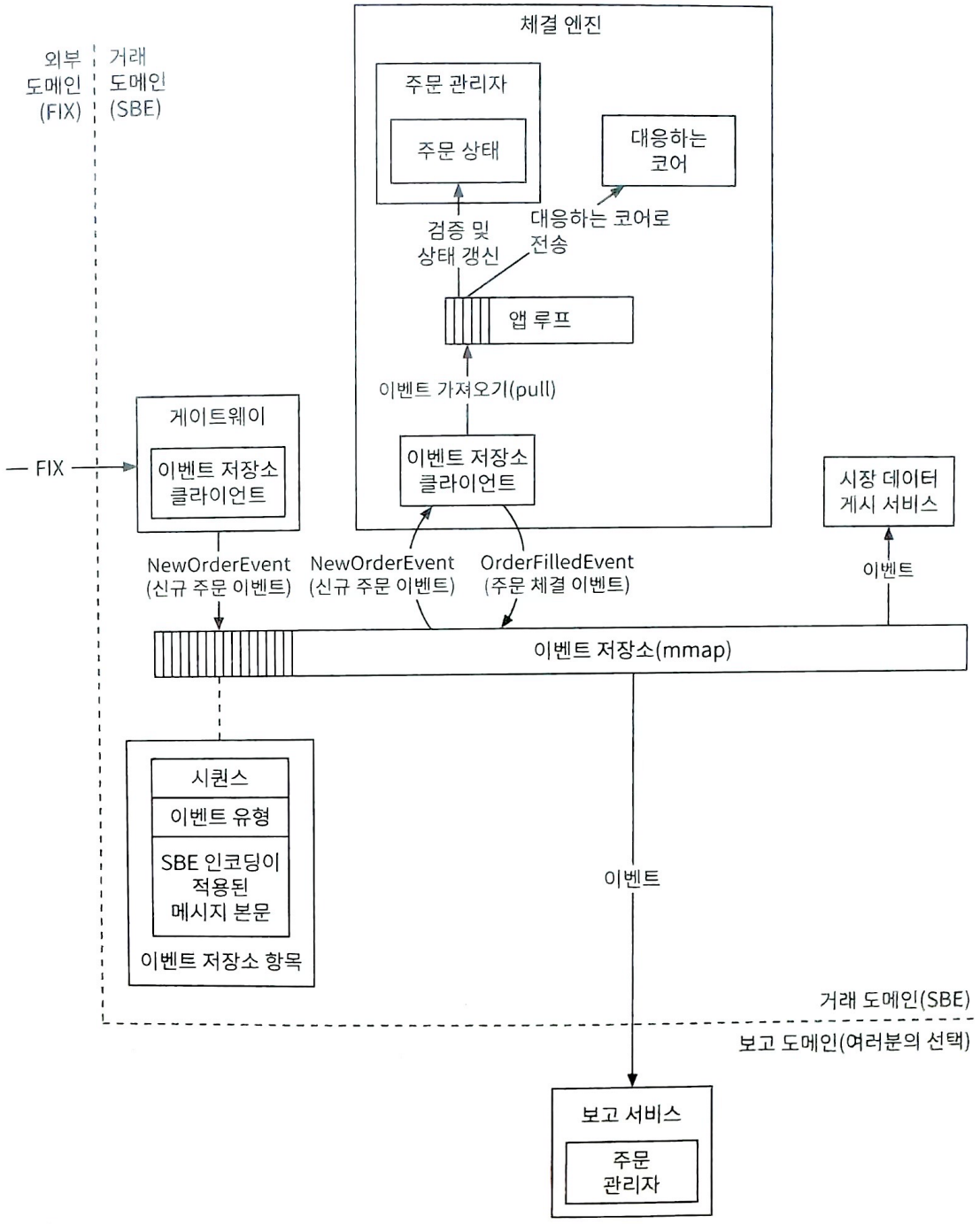

그림 13.18 이벤트 소싱 설계

1. 게이트웨이는 빠르고 간결한 인코딩을 위해 FIX를 SBE로 변환
    a. 각 주문을 이벤트 저장소 클라이언트를 사용하여 미리 정의된 형식의 NewOrderEvent 형태로 전송

2. 체결 엔진에 내장된 주문 관리자는 이벤트 저장소로부터 NewOrderEvent를 수신하면 유효성을 검사한 다음 내부 상태를 추가함
    a. 해당 주문 처리 담당은 CPU 코어로 전송
3. 주문이 체결되면 OrderFilledEvent가 생성되어 이벤트 저장소로 전송됨
4. 시장 데이터 프로세서 및 보고 서비스 같은 다른 구성요소는 이벤트 저장소를 구독하고, 이벤트를 받을 때마다 적절히 처리함

이벤트 소싱 아키텍처에서 더 효율적으로 동작할 수 있도록 조정한 부분

- 주문 관리자
    - 이벤트 소싱 아키텍처에서 주문 관리자는 컴포넌트에 내장되는 재사용 가능 라이브러리
    - 다른 컴포넌트가 주문 상태 업데이트나 질의를 위해 중앙화된 부문 관리자를 이용하도록 할 경우, 지연 시간은 길어질 수 있음
    - 각 컴포넌트가 주문 상태를 자체적으로 유지하기는 하겠으나 이벤트 소싱 아키텍처의 특성상 그 모두는 전부 동일하며 재현 가능함
- 시퀀서가 없음
    - 모든 메시지는 동일한 이벤트 저장소를 사용함
    - 이벤트 저장소에 보관되는 항목에는 sequence 필드가 있음
    - 각 이벤트 저장소에는 하나의 시퀀서만 있음
        - 여러 개 있으면 이벤트 저장소에 쓰는 권한을 두고 경쟁하게 되므로 좋지 않음
        - 락 경합에 낭비할 시간 없음
    - 시퀀서는 이벤트 저장소에 보내기 전에 이벤트를 순서대로 정렬하는 유일한 쓰기 연산 주체

고가용성

가용성을 높여야 할 때 살펴보아야 하는 사항

- 단일 장애 지점 식별
    - 체결 엔진에 발생하는 장애는 거래소 입장에서 재앙이기 때문에 주 인스턴스를 다중화해야 함

- 장애 감지 및 백업 인스턴스로의 장애 조치 결정이 빨라야 함
    - 클라이언트 게이트웨이와 같은 무상태 서비스는 서버를 추가하면 쉽게 수평적 확장 가능
    - 주문 관리자나 체결 엔진처럼 상태를 저장하는 컴포넌트는 사본 간에 상태 데이터를 복사할 수 있어야 함

주/부 체결 엔진 설계안

- 주 체결 엔진 = 주 인스턴스 / 부 체결 엔진 = 부 인스턴스
- 부 인스턴스는 동일한 이벤트를 수신하고 처리하지만 이벤트 저장소로 이벤트를 전송하지 않음
- 주 인스턴스가 다운되면 부 인스턴스는 즉시 주 인스턴스 지위를 승계한 후 이벤트를 전송
- 부 인스턴스가 다운된 경우, 일단 재시작을 하고 나서 이벤트 저장소 데이터를 사용해 모든 상태를 복구함
- 이벤트 소싱 아키텍처는 결정론적 특성 때문에 상태 복구가 쉽고 정확하기 때문에 거래소에 적합함

주 체결 엔진의 문제를 자동 감지할 메커니즘이 필요함

- 하드웨어와 프로세스를 모니터링하는 일반적인 방안
- 체결 엔진과 박동 메시지를 주고받는 방안

주/부 체결 엔진 설계안의 문제점

- 단일 서버에서만 동작한다는 것
- 고가용성을 달성하기 위해서는 이 개념을 여러 서버 또는 데이터 센터 전반으로 확장해야 함
- 주/부 체결 엔진이 아닌 주/부 서버의 클러스터를 구성해야 함
- 주 서버의 이벤트 저장소는 모든 부 서버로 복제해야 함
    - 이벤트 저장소를 여러 서버로 복제하는 데는 시간이 걸림
    - 안정적 UDP를 사용하면 모든 부 서버에 이벤트 메시지를 효과적으로 브로드캐스트할 수 있음

결함 내성

부 서버까지 전부 다운이 된다면?

- 대형 기술 기업이 직면한 문제임
  - 핵심 데이터를 여러 지역의 데이터센터에 복제하여 이 문제를 해결함

결함 내성 시스템을 만들려면 답변해야 하는 질문

- 주 서버가 다운되면 언제, 그리고 어떻게 부 서버로 자동 전환하는 결정을 내리나?
  - 장애가 생겼다는 것은 무슨 의미인가?
    - 상황
      - 시스템에서 잘못된 경보를 전송하면 불필요한 장애 극복 절차, 즉 부 시스템으로의 자동 전환이 발생할 수 있음
      - 코드의 버그로 인해 주 서버가 다운되었다면 부 서버로 자동 전환되더라도 같은 버그 때문에 부 서버까지 다운될 수 있고, 그 결과 모든 주/부 서버가 중단되면 시스템은 더 이상 사용할 수 없는 상태에 빠짐
    - 해결책
      - 새 시스템을 처음 출시할 때는 수동으로 장애 복구 조치를 수행
      - 충분한 시그널, 운영 경험을 축적하여 시스템에 자신이 생기면 그때 자동으로 장애를 감지하여 복구하는 프로세스를 도입
      - 카오스 엔지니어링
        - 드물게 발생하는 까다로운 사례를 수면으로 이끌어내고 운영 경험을 빠르게 축적하는 데 좋은 방법
  - 어떤 서버가 주 서버 역할을 인계 받을까?
    - 실전에서 검증된 리더 선출 알고리즘이 많음
      - 레프트, ...
- 부 서버 가운데 새로운 리더는 어떻게 선출하는가?
  1. 리더는 팔로어에게 박동 메시지를 보냄
  2. 일정 기간동안 박동 메시지를 받지 못한 팔로어는 새 리더를 선출하는 선거 타이머를 시작함

3. 가장 먼저 그 타이머가 타임아웃된 팔로어는 후보가 되고, 다른 너머지 팔로어에게 투표를 요청함
4. 그 팔로어가 과반수 이상의 표를 받으면 새로운 리더가 됨
5. 첫 번째 팔로어의 임기 값이 새 노드보다 짧으면 리더가 될 수 없음
6. 여러 명의 팔로어가 동시에 후보가 되는 경우는 '분할 투표'라고 함
    a. 기존 선거의 타임아웃을 선언하고 새로운 선거를 시작함

- 복구 시간 목표(RTO)는 얼마인가?
    - 애플리케이션이 다운되어도 사업에 심각한 피해가 없는 시간의 최댓값
    - 우선순위에 따라 서비스를 분류하고 최소 서비스 수준을 유지하기 위한 성능 저하 전략을 정의
- 어떤 기능을 복구해야 하는가(RPO)? 시스템이 성능 저하 상태로도 동작할 수 있는가?
    - 증권 거래 소는 데이터 손실을 용납할 수 없기 때문에 RPO가 0에 가까움

체결 알고리즘

- FIFO 체결 알고리즘 사용
- 특정 가격 수준에서 먼저 들어온 주문이 먼저 체결되고, 마지막 주문은 가장 나중에 체결됨
- 체결 알고리즘은 많으니 상황에 따라 잘 사용하길

결정론

- 기능적 결정론
    - 시퀀스나 이벤트 소싱 아키텍처를 도입함으로써 이벤트를 동일한 순서로 재생하면 항상 같은 결과를 얻을 수 있도록 보장
    - 이벤트가 발생하는 실제 시간은 대체로 중요하지 않음
    - 중요한 것은 순서
- 지연 시간 결정론
    - 각 거래의 처리 시간이 거의 같은 것
    - 사업에서 가장 중요한 부분

    - 측정하는 수학적인 방법 = 99번 백분위수 지연 시간(p99)이나 99.99번 백분위수 지연 시간을 재는 것
    - p99 지연 시간이 낮다는 것은 거래소가 거의 모든 거래에 안정적인 성능을 제공한다는 뜻
    - HdrHistofram을 활용할 수 있음
    - 지연 시간 변동 폭이 커지면 원인을 조사해야 함

거래소의 흥미로운 측면

시장 데이터 게시 서비스 최적화

- 많은 헤지 펀드가 거래소 실시간 API를 통해 데이터를 직접 기록하여 봉 차트를 비롯해 기술적 분석을 위한 많은 차트를 자체적으로 구축함
- 시장 데이터 게시 서비스는 체결 엔진의 체결 결과를 받아 이를 기반으로 호가 창과 봉 차트를 재구축한 다음 구독자에게 데이터를 게시함
- 호가 창 재구축 과정은 "체결 알고리즘"과 유사
- MDP는 다양한 수준의 서비스를 제공함
    - 개인 고객은 기본적으로 다섯 레벨의 L2 데이터만 볼 수 있으며 열 개 레벨을 보려면 추가 비용을 지불해야 함
    - MDP의 메모리는 무한대로 확장할 수 없기 때문에 봉 차트에는 상한선을 두어야 함
- 이 설계안은 링 버퍼를 활용함
    - 생산자는 계속 데이터를 넣고, 하나 이상의 소비자는 데이터를 꺼냄
    - 링 버퍼의 공간은 사전에 할당된 것으로 객체를 생성하거나 삭제하는 연산은 필요 없음
    - 락을 사용하지 않음

시장 데이터의 공정한 배포

- 거래소에서 다른 사람보다 지연 시간이 낮다는 것은 미래를 예측할 수 있다는 것과 같음
- 규제를 받는 거래소의 경우 모든 수신자가 동시에 시작 데이터를 받을 수 있도록 하는 것이 중요

    - 항상 첫 번째 구독자가 먼저 데이터를 수신하면 모든 고객들이 시작이 열리면 첫 번째 구독자가 되기 위해 달려들 것
- 완화할 수 있는 방법
    - 안정적 UDP를 사용하는 멀티캐스트는 한 번에 많은 참가자에게 업데이트를 브로드캐스트 하기 좋음
    - 구독자가 연결하는 순서로 데이터를 주는 대신 무작위 순서로 주는 방법
- 인터넷에서 데이터는 세 가지 유형의 프로토콜을 통해 전송됨
    - 유니캐스트: 하나의 출처에서 하나의 목적지만으로 보내는 방식
    - 브로드캐스트: 하나의 출처에서 전체 하위 네트워크로 보내는 방식
    - 멀티캐스트: 하나의 출처에서 다양한 하위 네트워크 상의 호스트들로 보내는 방식
- 거래소 설계에 보편적으로 이용되는 것은 멀티캐스트
    - 같은 멀티캐스트 그룹에 속한 수신자는 이론적으로는 동시에 데이터를 수신
    - UDP는 신뢰성이 낮은 프로토콜이며 그 데이터그램은 모든 수신자에게 도달하지 못할 수도 있음

코로케이션

- 많은 거래소가 헤지 펀드 또는 브로커의 서버를 거래소와 같은 데이터 센터에 둘 수 있도록 하는 코로케이션 서비스를 제공함
- 체결 엔진에 주문을 넣는 지연 시간은 기본적으로 전송 경로 길이에 비례함
- 코로케이션 서비스가 공정성을 훼손한다고 보지는 않음
- 유료 VIP 서비스로 봄

네트워크 보안

- 공개 서비스와 데이터를 비공개 서비스에서 분리하여 DDoS 공격이 가장 중요한 클라이언트에 영향을 미치지 않도록 함
    - 동일한 데이터를 제공해야 하는 경우에는 읽기 전용 사본을 여러 개 만들어 문제를 격리
- 자주 업데이트되지 않는 데이터는 캐싱
    - 캐싱이 잘 되어 있으면 대부분의 질의는 데이터베이스에 영향을 미치지 않음

- DDoS 공격에 대비해 URL을 강화
- 효과적인 허용/차단 리스트 메커니즘을 사용
- 처리율 제한 기능을 활용

마무리

끝~! 🥰