경력기술서

정 순 구

chagchagchag.dev@gmail.com

웨이커 (2021.06 ~ 2022.06)

- 신규 서비스 API 개발

- 미국주식 실시간 데이터 처리 서버 개발

신규 서비스 API 개발

레거시 서비스(valuesight)의 개선작업으로 인한 신규서비스(waiker)의 API 개발업무를 진행했습니다. 테이블을 설계/유지보수, API 설계/구현하는 업무가 주된 업무였습니다.

대가들의 분석, 뉴스, 주요재무, AI종목분석, 종목정보 등 주식 탭에 존재하는 API들을 개발했습니다.

미국주식 실시간 데이터 처리 서버 개발

미국 주식 거래 체결 데이터를 TCP/IP 소켓 기반으로 받아서 현재가/일/시/분/초봉 테이블에 각각 BATCH UPSERT 하는 로직과 웹소켓으로 IOS/AOS/WEB 으로 메시지를 푸싱하는 기능을 개발했습니다. 당시 RabbitMQ 어드민 화면에서 확인해본 바로는 트래픽이 1MS 에 최소 10건, 최대 100건에 도달했습니다.

이러한 단건 트래픽 데이터를 데이터 저장/푸시 작업의 지연 현상 없이, 메모리 부족 현상 없이 정상적으로 IO처리가 이뤄지도록 서비스를 개발하고 확장해왔습니다.

미국 주식 외에도 채권,환율,지수,선물,장외거래 데이터(Pre/Aft 마켓데이터)를 취급했기에 여러가지 개장시간 별 경우의 수 조건들, 데이터별 필드 매핑이 다른 요소에 대한 조건들을 적용하기 위해 추상화와 다형성 개념을 통해 비즈니스 로직을 유연하게 구성했습니다.

주요 문제 해결 경험

웨이커에서 주식 데이터 서버를 개발한 경험이 가장 크고 커리어를 통틀어서 가장 난이도 높았던 작업이었습니다. 이 과정에서 메시지 큐 기반으로 작업을 분배하는 방식에 대한 관점의 변화도 생겼고, 작업의 영역을 개별 인스턴스의 영역으로 분리해서 각각의 인스턴스가 오프 힙 캐시 안에 작업을 쌓아두고 일정 간격으로 소비하도록 하는 방식으로 인프라의 구조를 전환했습니다.

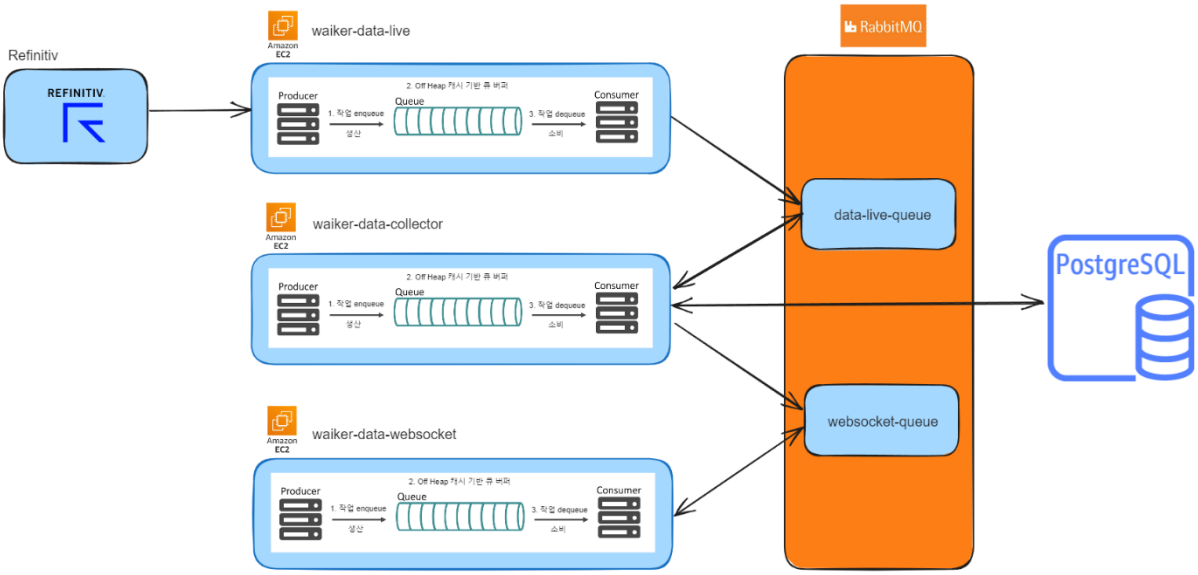
웨이커에서의 문제해결 경험은 분량이 많아서 가급적 가독성이 높은 아래 링크 를 참고해주셨으면 합니다.

웨이커에서의 주요 문제 해결 경험

- https://chagchagchag.github.io/intro/waiker-experience/4.0.WAIKER-MAJOR-EXPERIENCE/

메시지 큐 기반 인스턴스간 작업 분배방식 구축

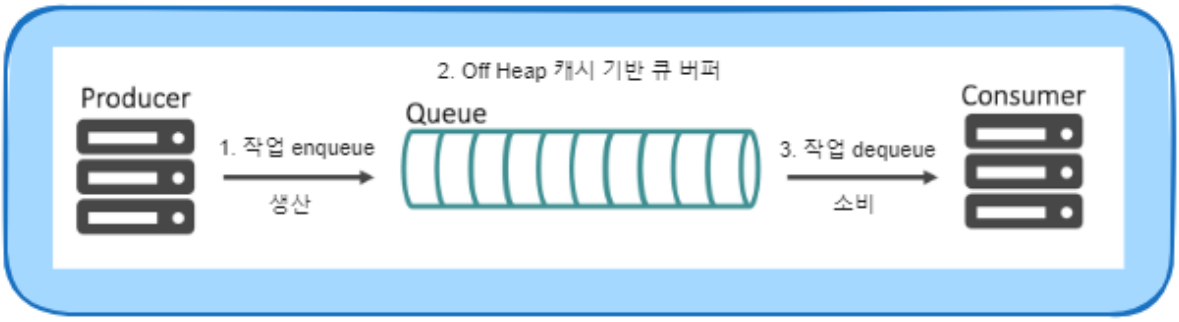
데이터 처리 서버가 하는 역할을 크게 waiker-data-live, waiker-data-collector, waiker-data-websocket 으로 크게 3가지 역할로 구분했고 메시지 큐를 통해서 각각의 인스턴스가 작업을 처리할 수 있도록 인프라를 설계했습니다.



MSA 패턴에서 흔히 이야기하는 SAGA, Outbox 를 혼합한 모습이였고 이렇게 역할을 분리 후에 각각의 인스턴스 간의 작업을 메시지 큐를 통해 분배하면서 시스템의 결합도를 낮추고 부하를 분산시킬 수 있었습니다.

실시간 거래 체결 데이터 웹소켓 Push, DB 저장 처리

위에서 봤던 전체 구성도에서 각각의 인스턴스 내의 생산자,소비자 구조를 살펴보면 아래와 같습니다. 개별 인스턴스 내에 Offheap 저장소를 이용해 작업 큐를 구성하고 생산자, 소비자 스레드를 별도로 두어 작업을 처리하도록 구성했습니다.



각각의 인스턴스는 생산자 용도의 ExecutorService 로 데이터를 Receive 할 때마다 별도의 처리를 거친 후에 오프 힙 작업 큐에 처리한 데이터를 enqueue 합니다. 그리고 소비자는 소비자 스레드 용도의 ExecutorService를 이용해 일정 주기마다 데이터를 일정 사이즈만큼 꺼내어서 작업을 배치처리합니다. 이 당시 Batch 작업의 적당한 스케쥴링 주기를 산출하기 위해 Offheap 캐시에 데이터를 INSERT 하는 속도, DB Insert 속도 등을 측정했고 조금의 여유 스케쥴링 기간을 두어서 스케쥴링 주기를 결정했습니다.

카프카의 프로듀서, 컨슈머의 내부동작을 보면 스케쥴링 주기, timeout, batchSize 등을 설정하기도 하고, Kafka Streams 의 내부에서는 Offheap 저장소를 사용하기도 하는데, 부하를 줄이기 위해 이런 개념들이 사용되었습니다.

스레드 풀 경량화 작업

Offheap 저장소에 작업큐를 만들어두고 생산자/소비자 구조로 작업을 처리하는 것 외에도 스레드 풀을 용도별로 가벼운 사이즈로 선언해서 사용하는 것 또한 중요했습니다.

이렇게 스레드 풀을 경량화 하는 것이 가능했던 것은 스케쥴링 기반으로 오프힙 저장소의 작업큐를 생산자/소비자가 각각 enqueue/dequeue 하는 방식으로 애플리케이션의 로직을 구성했기에 가능했습니다. 이렇게 각각의 스레드가 하는 일들이 명확히 구분되어 있어서 스레드 풀을 경량화 하는 데에 도움이 되었습니다.

waiker-data-collector 를 예로 들어보면 스레드 풀은 아래와 같이 구성했습니다.

- scheduler 스레드
  - ScheduledExecutorService
  - 전역적으로 대부분의 작업을 스케쥴링 하는 데에 공통으로 사용
  - corePoolSize : 2

- tickInsertExecutor : 일/시/분/초봉 데이터를 DB에 저장하는 역할

  - CompletableFuture 가 사용하는 Executor

  - corePoolSize : 1, maximumSize : 4

- latestInsertExecutor : 현재가격 데이터를 DB Upsert 하는 역할

  - CompletableFuture가 사용하는 Executor

  - corePoolSize : 1, maximumPoolSize : 2

- pricePushExecutor : 현재가격 데이터를 Websocket MQ에 푸시하는 역할

  - CompletableFuture 가 사용하는 Executor

  - corePoolSize : 2

- listenerExecutor : @RabbitListener 를 이용한 리슨 로직에서의 코드 처리를 비동기적으로 하기 위한 Executor

  - corePoolSize : 1, maximumPoolSize : 2

  - 일/시/분/초봉 가격이 계산된 데이터 들을 객체에 저장하고 가격 객체들을 작업 킷값에 맞도록 오프힙 캐시 기반 작업 큐에 저장

현업에서는 보통 오래 걸리는 작업에 대해 1차적으로는 스레드 풀을 단순히 크게 잡는 것으로 해결하는 경우가 일반적인데, 미국 주식의 경우는 스레드 풀을 크게 늘리는 것으로 해결이 되지 않아서 위와 같이 스케쥴링 기반의 생산자/소비자 작업 큐 구조와 스레드 풀 경량화 작업을 통해 IO 처리의 효율성을 높였습니다.

k8s와 같은 분산처리 환경을 사용하더라도 가끔은 부하를 처리하는 데에 있어서 Pod 인스턴스 내부에서도 스레드를 경량화해서 처리하는 것 역시 중요하다고 생각합니다. 이런 면에서 웨이커에서의 데이터처리 서버 개발시 스레드 풀 경량화를 했던 경험은 좋은 문제 해결 경험이었다고 생각합니다.

디케이테크인 (2019.11 ~ 2021.06)

뮤직 DNA 6.0 개편

- 월간/주간 개인 선호음악/추천음악 조회 백엔드 API 개발

피드, 마이로그 6.0 개편

- 개인화 영역 소식 내역 메시지 생성 기능 개발

멜론 댓글 서비스 운영/유지보수

- 슬로우 쿼리 유지보수/대응

- 카카오 클린플랫폼 연동 블랙리스트 기능 개발

멜론 HiFi/음원/스테이션/카카오뮤직/계정 유지보수

- 서비스 운영시 발생하는 수없이 많이 나타나는 각종 버그들, QA이슈 처리. 다크모드 지원 작업 등을 진행

주요 문제해결 경험

스테이션 배너 앱 랜딩에 멜론 키즈의 프로모션이 랜딩되지 않는 이슈가 있었습니다. 멜론모바일, 멜론웹, 멜론 공통 모듈까지 모두 검사해서 어떤 부분이 잘못되었는지 찾아가는 과정을 겪었고, 결론은 멜론 어드민 내의 상수 코드 값이 빠져있어서 생기는 이슈였다는 것을 파악했습니다.

이 상수 값은 내부 기획 업무시에 사내 업무 게시판에는 공유되었지만 엑셀에 추가를 하지 않은 것으로 인해 관련된 담당자분들(AOS, IOS)도 해당 내용을 반영하지 않았던 것으로 기억합니다.

멜론서비스를 운영할 때에는 코드의 기능 적으로 개선을 해야 하는 부분들도 있었지만 위의 경험처럼 제품결함의 이슈를 코드보다는 유입 flow 측면에서 어느 시점에 에러가 발생하는지 논리적으로 판단하고 이 기능과 연관된 모듈들이 무엇이 있는지 제품을 전반적으로 파악해서 판단해서 버그를 수정해야 했던 경험들도 자주 해왔습니다.

누리플렉스 (2018.10 ~ 2019.11)

SK E&S STEP 에너지 모니터링 솔루션 개발/운영업무

- 장비 개별정보 현황/이력(통계) 데이터 조회 API 설계/구현

- 데이터 시각화 (차트라이브러리, 그리드 라이브러리 연동 등)

SUSTERA PMS 에너지 모니터링 솔루션 개발/운영

- 장비 개별정보 현황/이력(통계) 데이터 조회 API 설계/구현

- 데이터 시각화 (차트라이브러리, 그리드 라이브러리 연동 등)

CJ/부산 신재생 에너지 혁신센터 EMS 운영/유지보수.

- MQ 데이터 트래픽 서버의 잦은 장애로 인한 SockJS 측에서의 예외처리

- 이슈 및 개별 장애 대응

주요 문제 해결 경험

- ActiveMQ 서버 개발 파트의 잦은 재기동이슈로 인한 SockJS 네트워크 유실문제

CJ/부산 신재생 에너지 혁신센터 시스템 운영 당시 개발팀에는 한전이나 LS로부터 전달받는 소켓 데이터를 Active MQ를 통해 웹소켓 데이터로 발송하거나 DB에 데이터를 INSERT하는 IO 작업을 담당하는 개발자 분이 계셨습니다. 한전/LS 로부터 전달받는 단건 데이터의 빈도에 비해 웹소켓/데이터저장 처리 속도가 물리적으로 느리기에 전기요금이 경부하 기간인 새벽시간 대에는 전기 충전 트래픽이 몰려 서버가 다운되는 현상이 있었습니다.

이런 이유로 MQ서버 개발자 분께서는 야간에 자주 서버를 재기동하셨습니다.

당시 저는 WAS 측의 운영을 담당하고 있었습니다. 서버 재기동이 영향을 주던 기능은 ESS 충방전현황 대시보드 내의 실시간 충전현황과 외기 온도를 보여주는 기능이었습니다. 서버를 재기동하기에 웹소켓 커넥션이 유실되는 것으로 인해 UI상으로는 관련된 기능이 다운된 것처럼 보이는 현상이 있었습니다.

이 문제에 대해 **‘소켓 접속이 끊어지더라도 주기적으로 서버에 재접속 요청을 하도록 구성하는 것’** 이라는 점에 포인트를 주어 해결해야겠다는 결론을 내렸습니다. 그리고 SockJS 공식 문서를 참고해 SockJS의 내부 코드 중 디폴트 설정을 파악해서 디폴트 설정을 수정하고 기본으로 제공되는 SockJS 내부 코드의 일부분을 커스터마이징 해 문제를 해결했습니다.

수정했던 디폴트 설정은 ‘네트워크 커넥션 타임아웃 기간’, ‘재접속 Retry 주기’ 등 이었고, 무한대로 접속 요청을 하는 것으로 인한 부하 역시 줄여야 하기에 ‘재접속 횟수’에 제한을 걸어서 재접속 요청을 하도록 SockJS 소스코드를 수정해 배포했습니다.

MQ 서버 처리로직에 안정성에 문제가 많았고 결함이 많았지만, 클라이언트 측(SockJS)의 코드를 유연하게 구성해서 제품의 문제가 발생하더라도 유연하게 장애에 대응했던 경험이라고 생각합니다.

주니코리아 (2015.05 ~ 2017.04)

3G/4G 무선신호를 WIFI 로 변환해주는 기업용 데이터 네트워크 라우터들의 모니터링, 관리/제어를 위한 솔루션 유지보수/개발 업무를 담당했습니다.

Hardware Replacement (TELSTRA)

- JAVA GUI, TCP/IP 통신 로직 개발

HEMS 유지보수/개발/운영

- QA 대응, 유지보수/개발/운영, 트러블 슈팅

주요 문제 해결 경험

Java GUI 애플리케이션에서 TCP/IP 통신 로직을 작성할 때 주로 외부 API를 호출할 때 GUI에 관련된 메인 스레드를 IO작업이 블로킹하지 않도록 로직을 구현하는 작업들이 많았습니다. 예를 들면 네트워크 설정파일(json, xml)의 SFTP Upload/Download 를 진행과 동시에 프로그래스바 UI에 진행률을 표시하는 기능을 구현하는 등의 작업 등을 해왔습니다.