JKIICE Journal of the Korea Institute of Information and Communication Engineering

한국정보통신학회논문지 Vol. 24, No. 11: 1539~1545, Nov. 2020

다양한 요구사항 수용 및 데이터 조회 성능 향상을 위한 CQRS 패턴 기반의 사물인터넷 플랫폼 설계

전철호[1] · 전현식[2] · 박현주[3*]

Design of A IoT Platform Based on CQRS Pattern to Accommodate Various Requirements and Improve Data Query Performance

Cheol-Ho Jeon[1] · Hyeon-Sig Jeon[2] · Hyun-Ju Park[3*]

[1]Ph.D student, Department of Mobile Convergence and Engineering, Hanbat National University, Daejeon, 34158 Korea
[2]Postdoc. Researcher, Department of Mobile Convergence and Engineering, Hanbat National University, Daejeon, 34158 Korea
[3*]Professor, Department of Information and Communication Engineering, Hanbat National University, Daejeon, 34158 Korea

요 약

현대 사회의 과학기술 발전으로 유비쿼터스 시대가 도래함에 따라 사회적으로 사물인터넷 환경에서 발생하는 데이터에 관한 관심이 증가하였다. 하지만 기존 사물인터넷 플랫폼은 대용량 데이터에 대한 통계 처리와 같이 많은 처리량을 요구하는 조회 요청 처리에 어려움이 있다. 이에 따라 본 논문에서는 조회 요청에 대한 요구사항을 유연하게 수용할 수 있고 조회 성능을 향상 시킬 수 있는 사물인터넷 플랫폼을 제안한다. 본 논문에서 제안하는 플랫폼은 별도의 읽기 데이터베이스를 도입하여 평균 응답시간 측면에서 약 1200배의 성능 향상을 보였으며, 객체 모델을 명령 사이드와 조회 사이드로 분리함으로써 객체의 복잡도를 낮춰 플랫폼에 대한 다양한 요구사항을 빠르게 수용할 수 있도록 하였다.

ABSTRACT

With the advent of the ubiquitous era due to the development of science and technology in the modern society, interest in data generated in the IoT environment has increased socially. However, the existing IoT platform has difficulties in processing inquiry requests that require large amounts of throughput, such as statistical processing of large amounts of data. Accordingly, in this paper, we propose an IoT platform that can flexibly accommodate requirements for inquiry requests and improve inquiry performance. The platform proposed in this paper showed a performance improvement of about 1200 times in terms of average response time by introducing a separate read database. By separating the object model into a command side and a query side, the complexity of the object is reduced to meet the various demands on the platform. It was made to allow quick acceptance of the matter.

키워드 : 사물인터넷 플랫폼, 명령 및 쿼리 책임 분리 패턴, 조회 사이드, 명령 사이드, 읽기 데이터베이스
Keywords : IoT Platform, CQRS Pattern, Query Side, Command Side, Read Database

Received 25 July 2020, **Revised** 27 July 2020, **Accepted** 16 August 2020

*** Corresponding Author** Hyun-ju Pack(E-mail:phj@hanbat.ac.kr, Tel:+82-42-821-1220)
Professor, Department of Information and Communication Engineering, Hanbat National University, Daejeon, 34158 Korea

Open Access http://doi.org/10.6109/jkiice.2020.24.11.1539 print ISSN: 2234-4772 online ISSN: 2288-4165

Ⅰ. 서 론

우리는 데이터가 다양한 목적에 따라 지속적으로 수집되는 데이터 중심 시대를 맞이했다. 가용 데이터를 기반으로 적시에 의사 결정을 내릴 수 있는 능력은 비즈니스 성공, 임상 치료, 사이버 및 국가 안보 및 재난 관리에 매우 중요하다. 또한 대규모 시뮬레이션, 천문 관측소, 높은 처리량 실험 또는 고해상도 센서에서 생성된 데이터는 과학자가 지식을 추출할 수 있는 적절한 도구가 있는 경우 새로운 발견을 유도하는 데 도움이 된다[1].

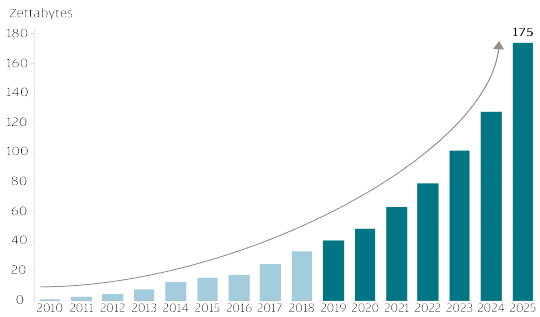

Fig. 1 Annual Size of the Global Datasphere

IT 및 통신, 컨슈머 테크놀로지 부문 시장조사 기관 IDC는 전 세계에서 생성, 캡처 또는 복사한 모든 데이터의 총량을 글로벌 데이터스피어(Global Datasphere)라고 정의하고 글로벌 데이터스피어의 총 규모가 2018년 33 ZB(Zettabytes)에서 2025년에는 175 ZB로 연평균 61% 증가할 것으로 전망했다. IDC는 데이터 세계를 세 영역으로 나눈다. 첫 번째는 코어 영역으로, 전통적인 데이터센터와 클라우드 인프라이다. 두 번째는 에지 영역으로, 통신탑이나 지사 환경을 말하며, 세 번째는 엔드포인트로 PC나 스마트폰, 사물인터넷 디바이스 등이다. 이에 대하여 IDC는 2025년 예상치인 175 ZB 중 90 ZB가 사물인터넷 디바이스에서 발생할 것으로 전망했다[2]. [그림 1]은 IDC에서 추정한 글로벌 데이터스피어의 연간 증가량을 보여준다.

이러한 배경들을 견주어 볼 때 IoT 플랫폼 기술은 빠른 폭으로 증가하는 데이터를 수용하고 엔드포인트의 다양한 요구사항을 수용할 수 있어야 한다. 이에 본 논문에서는 다양한 요구사항 수용 및 데이터 조회 성능 향상을 위한 CQRS 패턴 기반의 사물인터넷 플랫폼 설계에 관한 연구를 진행한다.

Ⅱ. 관련 연구

2.1. CQRS

CQRS는 Bertrand Meyer의 저서인 Object Oriented Software Construction에서 처음 언급된 Command and Query Separation(CQS) 원칙에서 시작되었다. CQS 원칙은 객체의 메서드를 조회(Query)와 명령(Command)으로 분리해야 한다는 것이다[3]. 조회는 시스템의 관찰 가능한 상태를 변경하지 않으면서 상태를 반환하는 메서드며, 명령은 시스템의 상태를 변경하지만 값을 반환하지 않는 메서드다.

이 원칙의 핵심은 만약 상태를 변경하는 메서드와 그렇지 않은 메서드를 분리할 수 있다면 소프트웨어를 개발하는 측면에서 매우 편리하다는 것이다. 조회 메서드의 경우 시스템의 상태를 변경하지 않는다. 이러한 특징으로 인해 어디서나 조회 메서드를 도입하고 순서를 변경할 수 있다. 구현 과정에서 시스템을 변경하는 명령 메서드의 사용에만 유의하면 된다[4].

CQRS(Command and Query Responsibility Segregation)는 모델을 각각 명령 및 조회가 의미하는 업데이트 및 조회를 위해 별도의 모델로 분할하는 것이다. 명령과 조회를 분리하는 이유는 복잡한 도메인에서 명령과 조회에 대해 동일한 모델을 사용하면 더 복잡한 모델로 발전될 수 있기 때문이다[5][6]. [그림 2]는 CQRS가 적용된 시스템 구조를 보여준다.

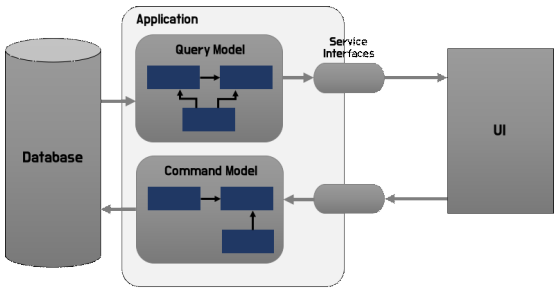

Fig. 2 System Architecture with CQRS

간단한 과정이지만 CQRS는 전통적인 아키텍처의 많은 문제를 해결할 수 있다. CQRS를 통해 각 객체들은 명령 사이드(Command Side)와 조회 사이드(Query Side)로 분리된다. 이러한 분리는 명령 사이드와 조회 사이드가 각각 다른 요구사항을 가진다는 개념을 강제한다[7].

2.2. 기존연구

Vehicle-to-everything (V2X) 환경에서 차량에 설치된 다양한 장치가 수집하는 데이터를 효율적으로 처리하기 하기 위해 CQRS을 적용한 연구가 있다[8]. 해당 연구는 CQRS를 통해 많은 양의 차량 데이터를 효과적으로 처리하였으며, 처리해야하는 데이터의 양에 따라 데이터베이스를 효과적으로 확장할 수 있음을 증명하였다.

특정 사물인터넷 응용에 CQRS를 적용한 연구 외에도 다양한 사물인터넷 응용의 공통적인 기능을 처리하는 사물인터넷 플랫폼에 대해 CQRS를 적용한 연구도 진행되었지만[9], 플랫폼에 마이크로서비스 아키텍처를 적용함에 따라 발생하는 각 마이크로서비스 간의 데이터 일관성 유지 문제를 해결하기 위해 CQRS를 활용했다는 한계점이 있다.

Ⅲ. 본 론

3.1. 선택적 프로토콜 기반의 IoT 플랫폼

본 절에서는 CQRS 패턴 적용 대상인 선택적 프로토콜 기반의 사물인터넷 플랫폼에 대해 기술한다[10]. 플랫폼은 HTTP, CoAP, MQTT 프로토콜을 제공하며, 이러한 프로토콜은 Data Collection & Control Server(CnC)를 통해 통신이 가능하다. 플랫폼에서는 다양한 프로토콜을 사용하는 IoT 디바이스, 데이터 수집 및 IoT 디바이스를 제어하기 위한 CnC, 데이터 전달의 신뢰성을 위한 캐시 용도의 임베디드 데이터베이스, 데이터가 최종적으로 저장되는 메인 데이터베이스, 사용자에게 다양한 인터페이스 제공과 모니터링을 위한 Web Application Server(WAS)로 구성된다. 선택적 프로토콜 기반의 IoT 플랫폼을 활용한 데이터 수집 솔루션의 전체 구성도는 [그림 3]과 같다.

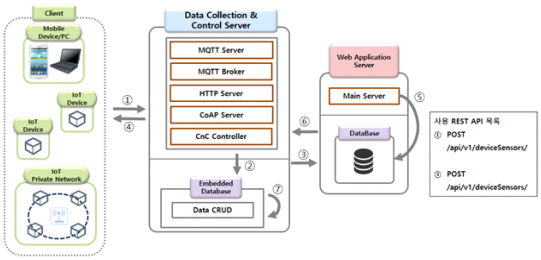

Fig. 3 Platform Architecture

WAS는 사용자를 위한 인터페이스와 모니터링 기능을 HTTP API 형태로 제공하며, 다양한 요구사항 수용하고 유연한 확장성을 제공하기 위해 마이크로서비스 아키텍처(Microservices)로 구현되었다[11]. 마이크로서비스 아키텍처 특성상 하나의 애플리케이션은 여러 마이크로서비스로 구성되며 각 서비스는 하위 도메인을 기준으로 분리되었다[12][13].

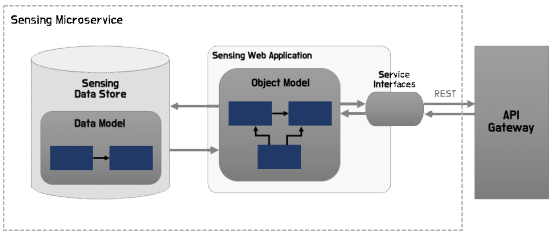

Fig. 4 Architecture of Sensing Microservice

센싱 마이크로서비스는 WAS를 이루고 있는 여러 마이크로서비스 중 하나로, IoT 디바이스에 부착된 센서에서 측정되는 센싱 데이터를 영속적으로 관리하고 통계 처리와 같은 다양한 형태의 조회 기능을 플랫폼 사용자에게 제공한다. [그림 4]는 센싱 마이크로서비스의 구성도이다.

하나의 IoT 디바이스에 세 개의 센서가 부착되어있고 1초에 한 번씩 센싱 값을 측정한다면, 하나의 디바이스에서 하루에 약 26 만 건, 한 달에 약 780 만 건, 한 해에 약 9300 만 건의 센싱 데이터가 발생할 수 있다. 사물인터넷 플랫폼 특성상 다수의 IoT 디바이스가 연결될 수 있기 때문에 센싱 서비스는 천문학적인 수의 센싱 데이터를 관리한다. 더불어 센싱 데이터에 대한 다양한 통계 요청을 처리해야하므로 많은 컴퓨팅 자원을 요구한다. 하여 본 연구에서는 복잡한 요구사항을 유연하게 수용하고 대용량 데이터 처리 성능 향상을 위해 CQRS 패턴을 센싱 서비스에 적용하는 연구를 진행한다.

3.2. CQRS가 적용된 IoT 플랫폼 설계

플랫폼의 여러 마이크로서비스 중 센싱 서비스에 CQRS 패턴을 적용한다. CQRS를 적용함으로써 객체 모델과 데이터 모델은 조회 사이드와 명령 사이드로 분리된다. 분리된 객체 모델의 경우 각각 명령 애플리케이션, 조회 애플리케이션으로 분배되고 분리된 데이터 모델의 경우 각각 명령 스토어, 조회 스토어(읽기 데이터

베이스)로 분배된다.

조회 애플리케이션은 조회를 위한 기능만 제공할 뿐 조회 스토어의 데이터를 업데이트하지 않으므로, 명령 스토어에 업데이트된 정보를 조회 스토어에 반영하는 별도의 동기화 컴포넌트가 필요하다. [그림 5]는 CQRS가 적용된 센싱 서비스의 아키텍처를 나타낸다.

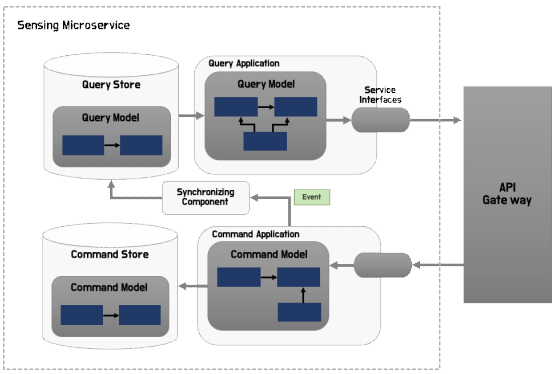

Fig. 5 Architecture of Sensing Microservice with CQRS

3.2.1. 객체 모델 설계

센싱 서비스의 웹 애플리케이션은 객체 지향 프로그래밍(Object-Oriented Programming, OOP) 패러다임을 바탕으로 설계되었다. 센싱 애플리케이션의 객체 모델(Object Model)은 클라이언트의 요청을 처리하는 서비스 모델, 클라이언트의 요청 및 응답 데이터 전달을 위한 DTO 모델, 트랜잭션 처리를 위한 도메인 모델로 구분될 수 있다. 센싱 서비스의 객체 모델을 클래스 다이어그램으로 나타내면 [그림 6]과 같다.

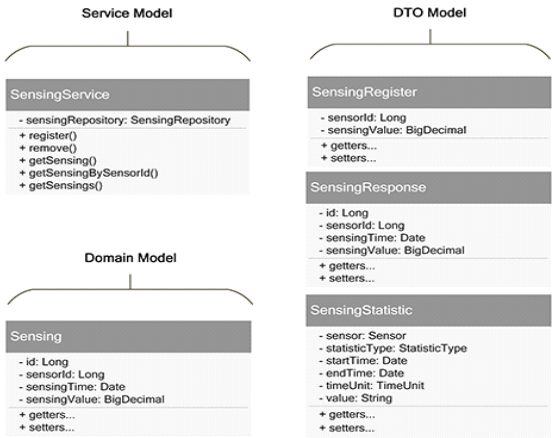

Fig. 6 Object Model of Sensing Microserivce

조회 사이드는 데이터 스토어의 정보를 업데이트하지 않으며 오직 클라이언트의 조회 요청에 필요한 기능만 제공하기 때문에 조회 사이드의 서비스 모델은 데이터를 반환하는 메서드만 포함한다. 기존 서비스 모델인 SensingService 클래스는 클라이언트의 조회 요청에 대한 응답을 위해 DTO를 반환하는 getSensing(), getSensingBySensorId(), getSensings() 메서드를 포함한다. 모두 조회를 위한 메서드이므로 조회 사이드의 서비스 모델인 SensingQueryService 클래스에 포함된다.

명령 사이드는 클라이언트의 조회 요청에 필요한 기능을 제공하지 않으며 데이터 스토어의 정보를 업데이트하기 위한 메서드만 포함한다. 기존 서비스 모델인 SensingService 클래스는 데이터 스토어의 정보를 업데이트하는 register(), remove() 메서드를 포함한다. 모두 업데이트를 위한 메서드이므로 명령 사이드의 서비스 모델인 SensingCommandService 클래스에 포함된다. [그림 7]은 CQRS가 적용된 서비스 모델을 보여준다.

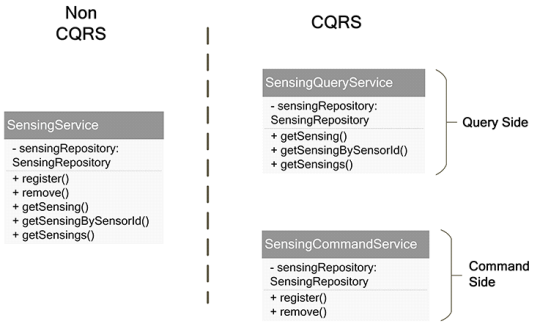

Fig. 7 Service Model with CQRS

도메인 모델의 경우 트랜잭션을 나타내고 처리하므로 조회 사이드에서는 제외되며 거의 대부분의 도메인 모델은 명령 사이드로 옮겨진다. 기존 센서 웹 애플리케이션에서 도메인 모델을 관리하는 도메인 계층은 그래도 명령 사이드로 옮겨지고 조회 사이드는 도메인 모델을 관리하지 않으므로 데이터 스토어에서 데이터를 읽어와 응답을 위한 DTO로 변환하는 읽기 계층만이 존재하게 된다.

3.2.2. 데이터 모델 설계

명령 사이드의 데이터 모델은 기존 센싱 데이터 스토어의 데이터 모델과 동일하다. 데이터 스토어는 관계형 데이터베이스인 PostgreSQL을 사용한다. 조회 사이드

데이터 스토어인 읽기 데이터베이스는 조회 성능을 위해 NoSQL 데이터베이스로 분류되는 MongoDB를 사용한다.

```
statistic collection
{
 sensor_id: 5,
 aggregation: [
  {time: 2019-01-01T12:00:00, sum: 1000, count: 100,
max: 15, min: 5},
  {time: 2019-01-01T13:00:00, sum: 2300, count: 220,
max: 22, min: 2},
  ...
 ]
},
...

sensing collection
{
 sensing_id: 1,
 sensor_id: 5,
 sensing_time: 2019-01-01T12:12:33,
 sensing_value: 12.3
},
...
```

Fig. 8 Data Model of Query Side

조회 사이드는 단일 센싱 데이터 조회를 위해 sensing 컬렉션을 관리한다. sensing 컬렉션은 명령 사이드의 sensing 테이블과 동일한 스키마를 가진다. 즉, sensing 컬렉션은 sensing 테이블의 컬럼(column)과 같은 필드(field)를 가진다. statistic 컬렉션은 센싱 데이터에 대해 합계, 건수, 최대, 최소와 같은 통계 값을 관리한다. 통계 값은 센서 별로 관리되며 한 시간 단위로 기본적인 통계 값을 저장하여 관리한다. 평균 값은 합계와 건수 데이터를 통해 도출할 수 있으므로 별도로 저장하지 않는다. statistic 컬렉션에 저장된 데이터를 통해 사용자의 다양한 조회 요청을 처리할 수 있다. [그림 8]은 조회 데이터 모델을 보여준다.

3.2.3. 동기화 컴포넌트

CQRS에서 조회 애플리케이션은 읽기 데이터베이스에 대해 조회 요청만 하므로, 읽기 데이터베이스의 데이터를 최신으로 반영하는 컴포넌트가 별도로 필요하다. 센싱 서비스의 동기화 컴포넌트는 명령 사이드에서의 데이터 업데이트에 대한 이벤트를 메시지 큐로 전달받고 일정한 시간을 주기로 조회 사이드의 읽기 데이터베이스에 반영한다.

명령 사이드에 발생하는 이벤트 메시지를 저장하기 위한 메시지 큐는 RabbitMQ를 사용한다. 동기화 애플리케이션은 일정 기간 동안 수집되는 이벤트를 임시로 저장하기 위해 별도의 이벤트 스토어를 가지며 이때 MongoDB를 사용한다. [그림 9]는 동기화 컴포넌트의 구성도이다.

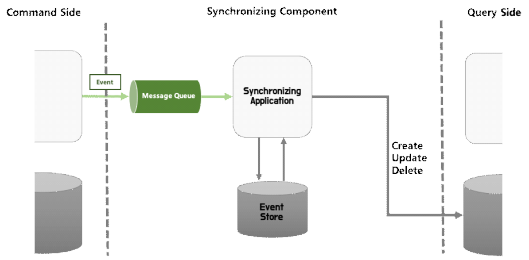

Fig. 9 Architecture of Synchronizing Application

클라이언트가 센싱 데이터에 대한 등록 요청을 하면 명령 애플리케이션은 유효성 검사를 진행하고 센싱 데이터를 명령 데이터 스토어에 저장한다. 이때 명령 애플리케이션은 센싱 데이터가 잘 저장됐음을 의미하는 SensedEvent를 메시지 큐로 발행한다. 동기화 애플리케이션은 SensedEvent 메시지를 메시지 큐로부터 전달받아 이벤트 스토어에 저장한다. 동기화 애플리케이션은 수정된 정보를 조회 사이드의 저장소에 반영하기 위해 일정 시간 간격으로 이벤트 스토어에 저장된 이벤트를 조회 사이드의 데이터 모델 형식으로 변환하여 조회 저장소에 투영하고 반영된 이벤트는 이벤트 스토어에서 삭제된다.

동기화 애플리케이션은 SensedEvent 클래스에 저장된 내용을 활용하여 읽기 데이터베이스에 업데이트된 정보를 반영한다. SensedEvent 클래스는 조회 요청에 필요한 정보를 포함하기 위해 센싱 도메인 모델의 식별자, 센싱 데이터를 측정한 센서의 식별자, 센싱 데이터의 측정 시간, 센서에서 측정된 값 속성을 가진다.

통계 조회 요청을 위해 사용되는 statistic 컬렉션은 SensedEvent에 저장된 속성과 상이한 필드를 가지므로 동기화 애플리케이션은 한 시간 간격으로 별도의 변환 작업을 수행하고 읽기 데이터베이스에 Create 명령을 실행한다. 한 시간 동안 발생한 센싱 값에 대해 센서 별

로 합계, 건수, 최대값, 최소값 처리를 별도로 진행한 뒤 읽기 데이터베이스 적용한다. 이때 변환 작업이 완료된 데이터의 스키마는 statistic 컬렉션의 스키마와 동일하다.

Ⅳ. 검증 및 평가

4.1. 테스트 환경 구축 및 시나리오

성능 평가 시나리오는 Apache 재단에서 관리하는 성능 테스트 도구인 JMeter를 통해 각 시스템에 대하여 데이터 조회를 위한 HTTP 요청을 하며, CQRS가 적용된 플랫폼의 경우 조회 사이드 시스템에 요청한다. 이때 요청 수를 점진적으로 늘림으로써 각 시스템의 성능을 분석한다.

각 시스템의 구축환경은 [표 1]과 같다. CQRS 패턴이 적용되지 않은 플랫폼의 백엔드 시스템(Strereotypical Application)과 본 논문에서 제안하는 CQRS 패턴이 적용된 플랫폼의 백엔드 시스템의 조회 시스템(Query Application)은 비교 분석을 위해 하드웨어 적으로 동일한 환경으로 구성하였다.

Table. 1 System Building Environment

Component	Element	Information
Command Application	CPU	4vCPUs
	Memory	8 GB
	OS	Ubuntu 18.04 Server LTS
	Framework	Spring Boot v2.1.9.RELEASE
Synchronizing Application & Query Application & Stereotypical Application	CPU	Intel Core i7 6700HQ 2.6 GHz
	Memory	LPDDR3 16 GB 2133 MHz
	OS	macOS Catalina v10.15
	Framework	Spring Boot v2.1.9.RELEASE
Command Store & Stereotypical Store	CPU	8vCPUs
	Memory	32 GB
	OS	Ubuntu 18.04 Server LTS
	DBMS	PostgreSQL v12.1
Query Store	CPU	Intel Core i7 6700HQ 2.6 GHz
	Memory	LPDDR3 16 GB 2133 MHz
	OS	macOS Catalina v10.15
	DBMS	mongoDB v4.2.1

4.2. 조회 성능 평가

이 절에서는 대량의 센싱 데이터를 처리하는 통계 조회 요청을 함으로써 두 시스템의 성능을 비교 분석한다. 본 평가에서는 동시 요청 수를 의미하는 Thread를 100, 200, 300으로 늘려가며 테스트를 진행하였다. 이 때 통계 조회 요청을 위해 통계 처리되는 센싱 데이터는 약 2400 만개이다. [표 2]는 통계 조회 요청에 대한 성능 평가를 요약한다.

Table. 2 Summary Table of Performance Evaluation for Statistical Query Requests

	Non CQRS	CQRS		
Threads	100	100	200	300
Average Response (ms)	29359	25	949	2309
Errors(%)	0.4	0	0	0
Throughput(/sec)	3.2	98.6	154.1	145.9

두 시스템의 평가 결과를 통해 제안하는 시스템은 에러율과 처리량 측면에서 확연한 성능 향상을 보였으며, 평균 응답시간 측면에서는 약 1200배의 성능 향상을 보였다. 동기화 컴포넌트를 통해 명령 사이드에서 변경된 이력을 주기적으로 조회 사이드에 반영함으로써 조회 요청 시점의 필요한 데이터 처리량을 대폭 감소시킬 수 있었으며, 이에 따라 제안하는 플랫폼이 짧은 시간 내에 응답할 수 있었을 것으로 예상한다.

Ⅴ. 결론 및 향후 연구

본 연구에서 제안하는 시스템은 대용량 데이터에 대한 통계 질의와 같이 많은 처리량을 요구하는 요청에 대한 응답시간을 최소화시키고 시스템에 대한 책임을 명령과 조회로 분리함으로써 다양한 요구사항을 수용할 수 있는 CQRS 패턴을 적용하였다. 사용자의 조회 요청에 적합한 형태로 데이터를 미리 전처리하여 저장하고 조회 성능에 특화된 DBMS를 저장소로 사용할 수 있는 CQRS 패턴의 조회 사이드를 통해 많은 양의 데이터를 처리해야 하는 조회 요청에 대한 응답 시간을 대폭 감소시킬 수 있었다. 또한 각각 명령과 조회의 책임을 담당하는 명령 사이드와 조회 사이드로 시스템을 분리함으

로써 도메인의 복잡도를 단순화하였다. 그리고 모의실험을 활용한 성능검증을 통해 제안하는 시스템의 우수성을 증명하였다. 연구에서 도출한 연구 결과를 통해 빅데이터를 관리해야 하는 사물인터넷의 발전을 가속화시킬 수 있을 것이라 예상한다.

향후 연구 과제로는 동기화 로직의 구현 복잡도를 단순화하기 위해 이벤트 소싱(Event Sourcing) 패턴을 적용해야 한다. 이벤트 소싱을 적용하는 경우 데이터 스토어에 저장되는 대상이 도메인의 현재 상태가 아닌 이벤트가 된다. 각 마이크로서비스 간의 동기화를 위해 이벤트를 사용할 수 있는데 이벤트 소싱이 적용되는 경우 동기화를 위해 별도의 이벤트를 생성할 필요가 없으므로 구현 복잡도를 단순화시킬 수 있다[7][14]. 이때 명령 사이드와 조회 사이드의 동기화를 위한 이벤트 또한 같은 이점을 취할 수 있다. 더불어 시스템에 요구되는 조회 요청이 다양해짐에 따라 조회 모델이 많아지는 문제를 해결하기 위해, 하나의 조회 모델을 통해 다양한 조회 요청을 처리할 수 있는 조회 모델 고도화에 관한 연구를 진행한다.

REFERENCES

[1] O. Kumar and A. Goyal, “Visualization: a novel approach for big data analytics,” *2016 Second International Conference on Computational Intelligence & Communication Technology (CICT). IEEE*, pp. 121-124, 2016.

[2] D. Reinsel, G. John, and R. John, “The digitization of the world from edge to core,” *IDC White Paper*, 2018.

[3] B. Meyer, *Object-oriented software construction*, vol. 2. New York: Prentice hall, 1988.

[4] M. Fowler. “Command Query Separation” Martin Fowler-ThoughtWorks (2005) [Internet]. Available:https://martinfowler.com/bliki/CommandQuerySeparation.html

[5] M. Fowler. “Cqrs.” Martin Fowler's Blog (2011) [Internet]. Available: https://martinfowler.com/bliki/CQRS.html

[6] G. Young, “Cqrs documents by greg young,” Young 56, 2010.

[7] D. Betts, J. Domingues, G. Melnik, F. Simonazzi, and M. Subramanian, “Exploring CQRS and Event Sourcing: A journey into high scalability, availability, and maintainability with Windows Azure,” Microsoft patterns & practices, 2013.

[8] S. K. Han and J. I. Choi, “Data Processing System Using CQRS Pattern and NoSQL in V2X Environment,” *International Journal of Smart Device and Appliance*, pp. 1-8, 2020.

[9] T. Rogojanu, M. Ghita, V. Stanciu, R. Ciobanu., R. Marin, and C. Dobre. “Netiot: A versatile iot platform integrating sensors and applications,” *Global Internet of Things Summit. IEEE*, pp. 1-6, 2018.

[10] D. H. Kim, H. S. Oh, H. S. Jeon, and H. J. Park, “The IoT Data Collection Platform based on Optional Protocols,” *Korea Institute Of Communication Sciences*, pp. 1047-1048, 2017.

[11] G. Márquez, M. M. Villegas, and H. Astudillo, “A pattern language for scalable microservices-based systems,” *Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings*, 2018.

[12] L. Baresi and M. Garriga, “Microservices: The Evolution and Extinction of Web Services?,” *Microservices. Springer*, Cham, pp. 3-28, 2020.

[13] A. Henry and Y. Ridene, “Migrating to Microservices,” *Microservices. Springer*, Cham, pp. 45-72, 2020.

[14] S. O. Diakov, T. E. Zubrei, and A. S. Samoidiuk, “Application of event sourcing and CQRS patterns in distributed systems,” 2019.

전철호(Cheol-Ho Jeon)

한밭대학교 정보통신공학과 학사 (2018)
한밭대학교 모바일융합공학과 석사 (2020)
한밭대학교 모바일융합공학과 박사과정 (현재)
※관심분야 : 데이터베이스, 웹 프로그래밍, 사물인터넷

전현식(Hyeon-Sig Jeon)

한밭대학교 전파공학과 석사 (2005)
한밭대학교 전파공학과 박사 (2011)
※관심분야 : 데이터베이스, 공간 데이터베이스, 위치 추정 알고리즘

박현주(Hyun-Ju Park)

서울시립대학교 전산통계학과 공학사 (1990)
서울대학교 전산과학과 공학석사 (1992)
서울대학교 전산과학과 공학박사 (1997)
대전 산업대학교 정보통신공학과 전임강사 (1998~2000)
국립 한밭대학교 정보통신공학과 교수 (현재)
※관심분야 : 프로그래밍언어, 운영체제, 데이터베이스