논문 19-44-05-15 The Journal of Korean Institute of Communications and Information Sciences '19-05 Vol.44 No.05
https://doi.org/10.7840/kics.2019.44.5.895

자동 API 생성 및 데이터 처리 성능 향상을 위한 RESTful, Redis 기반의 소형 API 플랫폼 설계

서 보 민•, 장 병 순*, 오 형 석*, 박 현 주°

RESTful, Redis Based API Server Platform Design for Automatic API Generation and Data Processing Performance

Bomin Seo•, Byeong-soon Jang*, Hyeung-seok Oh*, Hyun-ju Park°

요 약

최근 사물인터넷(IoT)을 통해 다양한 분야에서 서비스를 제공하면서 IoT에 연동되는 디바이스 및 센서를 제공하는 IoT 관련 중, 소, 스타트업 기업들이 증가하고 있다. 그러나 IoT 관련 플랫폼을 구축하는 시간과 비용에 투자할 여력이 부족하여 초기 아이디어 실현 가능성에 대한 테스트를 진행함에 있어 어려움을 겪고 있다. 또한, 많은 기업과 단체에서 각자의 표준을 확립하여 사물인터넷 플랫폼을 구축하고 있는 현 상황에서, 신속히 플랫폼을 결정하지 못하면 개발에 차질이 생긴다. 이러한 점에 착안하여 본 논문에서는 데이터 생성 시 REST 기반의 API를 자동으로 생성하여 HTTP의 메소드 중 해당 자원에 대한 CRUD(Create, Read, Update, Delete) 요청을 신속히 처리할 수 있는 소형 API 플랫폼을 제안한다. 제안하는 소형 API 플랫폼은 JSON 형태의 데이터 처리가 가능하며 데이터의 형식, 크기 등에 대한 제한이 없으며 해당 자원에 대한 데이터의 속성 변경도 자유롭다. 또한, 적은 비용으로 플랫폼을 구동하기 위해 C++와 Redis를 사용하여 처리 성능을 향상시킴으로써 IoT에 연동되는 디바이스 및 센서가 처리하는 방대한 데이터를 신속히 처리할 수 있도록 하였다.

Key Words : IoT server, Redis, RESTful, C++, JSON

ABSTRACT

Recently, the Internet of Things has provided services in a variety of fields. As a result, the number of small start-up companies is increasing while providing IoT-related divers services. However, these companies have difficulty investing in the time and cost of building IoT-related platforms, making it difficult to test the feasibility of initial ideas. In addition, many companies nad organizations have established their own standards and are building Internet platforms for thigs. If the platform can not be determined quickly, there will be a problem in development. In this paper, we propose a platform that can process CRUD(Create, Read, Update,Delete) requests of corresponding resources among various methods of HTTP by automatically generating API based on REST when generating data. In addition, it is possible to process JSON type data, there is no restriction on the format and size of data, and the property change of data for the orresponding resource is also free. The proposed platform uses C++ and Redis to drive the platform at a low cost, which improves processing performance, enabling rapid processing of vast amounts of data processed by IoT-enabled devices and sensors.

◆ First Author : Hanbat National University Department of Mobile Convergence Enginerring, tjqh55@gmail.com, 학생회원
° Corresponding Author : Hanbat National University Department of Information and Communication Enginerring, phj@hanbat.ac.kr, 정회원
* Hanbat National University, Ubiquitous Media Technology
논문번호 : 201811-370-B-RN, Received November 27, 2018; Revised March 18, 2019; Accepted March 29, 2019

Ⅰ. 서 론

사물인터넷(Internet of Things)은 인터넷을 통해 연결된 사물들이 본연의 기능을 확대하고 가전제품에서부터 자동차, 웨어러블 기기, 헬스케어 등에 이르기까지 다양한 분야에서 사용된다. 사물인터넷 기술에 대한 기대와 관심이 고조되고 있는 가운데, 이런 시장 변화에 맞물려 국내의 창업 및 IoT 전문교육, 전문가 멘토링 등의 프로그램을 통해 IoT 관련 중, 소, 스타트업 기업들이 늘어나고 있고 IoT 관련 사업은 점차 현실적인 사업을 기반으로 진행될 것이다. 그러나 IoT에 연동되는 디바이스 및 센서를 제공하는 기업들은 신속히 플랫폼을 결정해야 하기 때문에 초기 아이디어 실현 가능성에 대한 테스트를 진행하는 데 어려움이 있다. 또한, IoT 플랫폼이란 인터넷에 연결된 모든 기기를 하나의 시스템 안에서 센싱, 통신 및 네트워크 인프라, 서비스 인터페이스의 주요 기술들을 총체적으로 요구하는 통합·관리 운용체계이다. IoT 플랫폼은 디바이스가 제공하는 데이터를 수집하기 위한 플랫폼을 직접 구축하는 시간과 비용도 상당하고 API 혹은 데이터 형식으로 인한 개발 기간의 지연이 발생한다. 이런 상황들은 초기 중, 소, 스타트업 기업들의 아이디어를 개발하는 과정에서 문제가 된다. 또한, 기존 RESTful에서는 URI에 대한 경로를 미리 정의해 사용함으로 URI를 통한 생성 요청이 불가하다. 이점에 착안하여 본 논문에서는 현재 개발 중인 IoT 플랫폼과는 별개로 IoT에 연동되는 디바이스 및 센서를 빠르게 테스트할 수 있는 소형 플랫폼을 제안한다. 제안하는 플랫폼은 IoT에 연동되는 디바이스 및 센서가 전달하는 데이터를 HTTP로 전달받 데이터 전송 시 사용한 URI를 이용해 HTTP의 메소드 중 CRUD(Create, Read, Update, Delete)에 해당하는 4가지 기능에 대한 API를 자동으로 생성한다. 또한, 데이터 형식의 자유로움을 보장하기 위해 어떠한 구조의 데이터든 JSON 형태로 전달하면 저장이 가능하고 성능을 향상하기 위해 In-memory Database인 Redis를 사용했고 C++로 구현함으로써 소형 디바이스에서도 동작이 가능하여 적은 비용으로 서비스를 운영할 수 있다.

Ⅱ. 관련연구

2.1 REST API

REST(Representational State Transfer)는 Roy Fielding의 2000년 논문[1]에서 분산 하이퍼미디어 시스템을 위한 소프트웨어 프로토콜 구조의 한 형식으로 제안되었다. REST는 HTTP로 데이터를 전달하고 클라이언트/서버 간의 구성 요소를 엄격히 분리하여 구현의 단순화와 확장성을 높인 아키텍처다. REST는 웹에 개방된 자원들을 쉽게 이용할 수 있는 웹 응용으로 정착되었으며 2005년 이후 공개되는 REST API의 수가 급속히 증가하고 있다. 이러한 REST 아키텍처 스타일에 따라 정의되고 이용되는 서비스나 응용 프로그램을 RESTful 웹 서비스라고 한다.

RESTful 웹 서비스는 자원을 등록하고 저장해두는 중간 매개체 없이 자원 제공자가 직접 자원 요청자에게 제공하는 ROA(Resource Oriented Architecture)로 SOA(Service Oriented Architecture)와 대비되는 개념이다[2].

REST의 기본요소는 리소스, 메소드, 메시지 3가지 요소로 구성된다. RESTful API 설계 시 리소스는 명사를 사용하여 직관적이고 단순하게 설계하며 자원의 집합이나 한 개의 자원을 나타낸다. 메소드는 HTTP에 여러 가지 메소드 중 CRUD에 해당하는 4가지 메소드만 사용한다. 마지막으로 메시지는 XML, JSON, HTML, 텍스트, 이미지 등으로 다양한 방식으로 표현된다.

2.2 Redis

Redis(Remote Dictionary Server)는 2009년 Salvatore Sanfilippo가 개발한 오픈소스 프로젝트로 메모리 기반의 Key/Value Store다. DBMS의 인기도 순위를 평가하는 DB-Engines에 따르면 Key/Value Store의 DBMS 중 1위를 기록했다[3]. Redis는 In-memory Database로 일반 디스크 저장소를 사용하는 Database에 비해 빠른 속도를 보장하고 다양한 자료형을 지원하여 데이터를 유연하게 관리할 수 있으므로 인스타그램, 네이버 LINE, StackOverflow, Blizzard 등 여러 소셜 서비스에 널리 사용되는 DBMS다. 현재는 VMWare에 인수되어 계속 업그레이드되고 있으며 C, C++, Java, Node.js, Python 등 대부분의 언어에서 사용 가능하다.

Redis는 In-memory Database의 특성상 시스템 전원이 꺼지면 모든 데이터가 손실되므로 메모리상 데이터를 디스크 파일로 저장하고, 서버 재시작 시 디스크 파일에서 데이터를 읽어와 메모리상에 다시 로드하는 방법으로 데이터를 보존한다[4].

2.3 JSON

JSON(JavaScript Object Notation)은 경량화된 데

이터 교환 방식이다. 모든 데이터를 태그로 저장해 사용하는 XML은 데이터 크기에 비해 효율이 낮고 DOM(Document Object Mode)을 통해 만들어진 객체 Parsing에 시간이 오래 걸리는 문제가 있어 이를 대신할 데이터 교환 방식으로 JSON이 대두되었다[5]. JSON은 완벽하게 언어로부터 독립적이다. 하지만 C, C++, Java, javaScript, Python 등 프로그래머들에게 친숙한 관습을 사용하는 텍스트 형식으로 이상적인 DATA 교환언어로 만들었다.

2.4 Crow

Crow는 2014년 3월에 시작된 프로젝트로 C++로 구현된 web framework이다[6]. Crow는 Python의 Flask web framework처럼 쉬운 표현으로 웹서버를 사용할 수 있게 구현되어 있으며, 성능은 Flask web framework와 비교할 수 없을 정도로 빠른 web framework이다. Crow는 BSD-3-Clause 라이센스로 해당 소프트웨어는 아무나 재작할 수 있고, 수정한 것을 제한 없이 배포 가능하며 상용 소프트웨어에서도 사용 가능하다.

Ⅲ. 본 문

본 논문에서는 IoT 분야에서 보편적으로 사용되고 있는 웹 프로토콜인 HTTP를 이용하여 데이터 요청을 처리하며, In-memory Database인 Redis와 C++용 web framework Crow 이용하여 데이터 처리 성능을 향상시킨 IoT용 소형 플랫폼(API Thin Platform, APITP)을 제안한다. 제안하는 Platform은 URI를 통해 자동으로 API를 생성하여 기본적인 CRUD 동작을 처리하며 JSON 형태의 데이터를 처리할 수 있다.

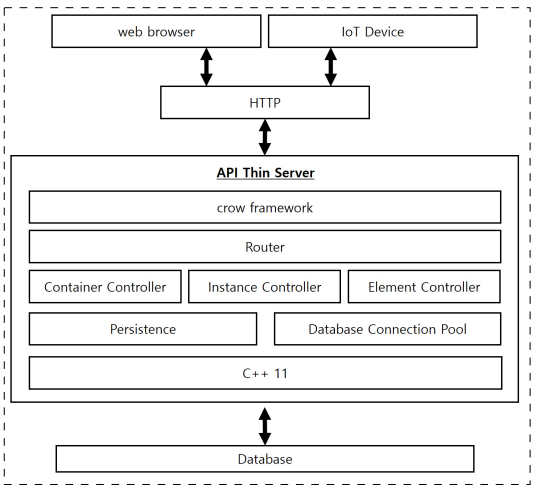

Fig. 1. APITP(API Thin Platform) Architecture

3.1 플랫폼 아키텍처

제안하는 플랫폼 아키텍처는 그림 1과 같은 구조를 가진다. 역할별로 APITP(API Thin Platform) 구동을 위한 Router, Controller, Persistence, Database Connection Pool, 데이터 저장을 위한 Database Server, IoT에 연동되는 디바이스 혹은 web browser로 접근하는 Client로 분류된다.

3.1.1 Database Server

IoT API Server에서 전달된 데이터를 저장하는 서버로 In-memory Database 중 가장 보편적으로 사용하는 Redis를 사용한다. In-memory Database 특성상 데이터가 들어오면 메모리에 저장되기 때문에 저장된 데이터를 일정 주기로 디스크 파일로 저장한다.

3.1.2 Database Connection Pool

플랫폼 초기 구동 시 처리속도 향상을 위해서 Database Server와 연결을 미리 해서 저장해 두는 모듈로 Persistence 모듈이 요청할 때 연결 객체를 전달한다. 또한, Persistence 모듈이 사용이 끝나면 반납받아 다시 Pool에 저장한다.

3.1.3 Persistence

Database Connection Pool로부터 Database Server와 연결 객체를 받아와 처리하는 동작에 맞는 쿼리를 수행하는 모듈로 JSON으로 넘어오는 데이터를 Redis에 저장 가능한 형태로 바꾸거나 Redis의 데이터를 JSON 형태로 변환해주는 모듈이다. Hiredis 라이브러리를 통해 구현했다.

3.1.4 Controller

IoT API Server의 동작을 총괄하는 모듈이다. Controller는 Container, Instance, Element 세 가지 모듈의 집합이다. Container는 플랫폼 사용자가 Instance, Element 데이터를 처리하기 위한 기본정보를 뜻한다. Instance는 플랫폼 사용자가 처리하길 원하는 디바이스나 센서 데이터를 뜻하며 Instance들의 각각의 데이터는 Element를 통해 저장된다. Controller는 Router로부터 Container, Instance, Element를 분리해서 그에 맞는 동작을 수행하도록 명령을 내리는 모듈이다.

3.1.5 Router

Router는 Crow web framwork를 이용해 구현했으며 URI를 분석하여 해당하는 Controller로 HTTP와 URI로 들어온 데이터를 JSON 형태로 변환하여 전달하는 모듈이다.

3.1.6 Client

IoT API Server로 데이터를 전달하거나 요청하는 개체로 HTTP를 사용하는 장치들을 지칭한다. IoT 디바이스뿐만 아니라 일반적인 PC나 스마트폰, 태블릿 PC도 Client로 사용하는 것이 가능하다.

3.2 Message Sequence

본 절에서는 instance, element 데이터를 처리하기 위한 기본정보를 담는 container에 대한 처리 흐름에 대한 절차와 실질적인 데이터를 담는 Instance/Element에 대한 처리 흐름에 대한 절차로 나누어 설명한다.

3.2.1 Container Message Sequence

Container Message Sequence의 CRUD 다이어그램은 그림 2와 같다. 먼저 플랫폼을 처음 실행 시 Database Connection Pool 모듈에서 플랫폼을 설치한 장치에 따라 connection 객체를 수용할 수 있는 범위만큼 지정해서 Database Server에게 connection 객체를 요청한다. 이 동작은 초기에만 진행하고 이후에는 Database Connection Pool 모듈이 Database Server 접근을 통제한다.

Client로부터 Container에 관련한 CRUD 요청이 플랫폼으로 들어올 경우 Router 모듈이 URI와 HTTP의 body 부분의 데이터를 JSON 형태로 만들어 각각에 맞는 Controller 모듈로 전달하게 된다. 요청을 전달받은 모듈은 URI와 JSON 형태의 데이터를 Persistence에게 전달하여 요청의 처리가 완료될 시점까지 대기한다. Persistence는 Database Connection Pool 모듈로부터 connection 객체를 받아 현재 들어온 요청이 제대로 된 요청인지 Database Server에서 검사한 후 해당 요청이 이상이 없을 때 데이터를 Database Server에 반영한다. 반영이 완료되면 Persistence는 connection 객체를 반환한다.

CRUD의 Create 경우 동일한 URI로 Database Server에 저장된 값이 없어야 진행되고 Read, Update, Delete의 경우는 반대로 Database Server에 저장된 값이 있어야 진행된다.

3.2.2 Instance/Element Message Sequence

Instance/Element Message Sequence의 CRUD 다이어그램은 그림 3과 같다. Container와 마찬가지로 먼저 플랫폼을 처음 실행 시 Database Connection Pool에서 connection 객체를 저장한 후 Database Connection Pool 모듈이 Database Server 접근을 통제한다.

Client로부터 CRUD 요청이 플랫폼으로 들어올 경우 Router 모듈이 URI와 HTTP의 body 부분의 데이터를 Instance/Element에 JSON 형태로 만들어 각각에 맞는 Controller 모듈로 전달하게 된다. Instance는 Create만 수행하고 Element는 Read, Update, Delete만 수행한다. 요청을 전달받은 Controller 모듈은 URI와 JSON 형태의 데이터를 Persistence 모듈에게 전달하여 해당 요청의 처리가 완료될 시점까지 대기한다. Persistence 모듈은 Database Connection Pool 모듈로부터 connection 객체를 받아와 현재 들어온 요청이 제대로 된 요청인지 Database Server에서 검사한다.

Instance의 Create 명령의 경우 동일한 URI로 Database Server에 저장된 값이 없어야 하고 있다면

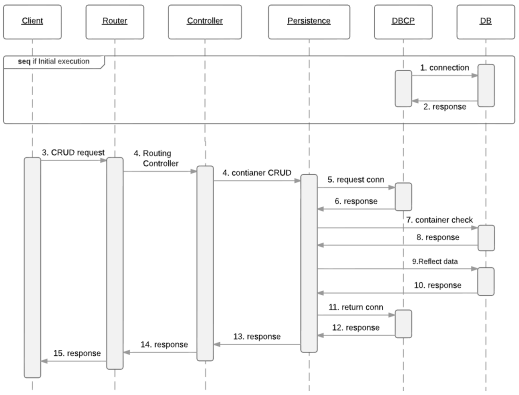

Fig. 2. Container CRUD Sequence Diagram

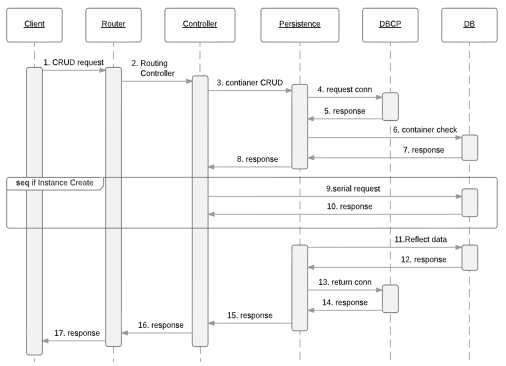

Fig. 3. Instance/Element CRUD Sequence Diagram

Database Server로부터 해당 Instance의 serial을 받아서 Database Server에 반영한다. Element의 경우 Database Server에 저장된 값이 있어야 반영된다. 해당 Element의 serial이나 Instance 중 하나라도 존재하지 않으면 진행되지 않는다.

진행이 완료되면 Persistence 모듈은 Database Connection Pool 모듈로 connection 객체를 반환하고, 결과 값을 Client에게 전달한다.

3.3 RESTful service를 위한 API 설계

관련 연구를 통해 RESTful Service를 제공하기 위해서는 URI가 자원의 집합이나 하나의 자원을 표현할 수 있어야 하며, 자원의 특성이나 특정 파라미터를 통해 구분 가능해야 한다.

자원의 집합은 플랫폼에서 Container라고 칭하는 부분에 처리되며 하위 계층 정보의 분류를 나타낼 때 사용한다. Conatiner의 내부는 그림 4와 같다. 플랫폼 사용 시 첫 번째 URI에 하위계층의 특성이 들어가도록 설계했지만, 플랫폼 사용자의 자유도를 위해 Container의 제약은 없다. 또한, Container도 데이터를 처리할 수 있도록 하였다.

단일 자원의 경우 플랫폼에서 Instance, Element로 처리되고 자원의 집합 중 단일자원에 대한 정보는 자원의 집합 URI 뒤에 나타낸다. 앞서 설명한 Container를 특성으로 사용할 경우에만 사용이 가능하다. 특정 파라미터 역시 Container처럼 자유도를 위해 이름에 특별한 제약은 없지만, Instance만으로는 데이터를 처리할 수 없다. 따라서 /Container/Instance 구조의 URI는 Create만 가능하며 플랫폼에서 자동으로 Element에 대한 ID를 생성하여 Instance에 대한 데이터를 저장한다.

Element의 경우 내부적으로 생성된 ID를 통해 Element에 접근하며 Read, Update, Delete 처리가 가능하다. Element의 경우 플랫폼 내부적으로 생성되어 숫자를 사용한 ID가 만들어진다. 만들어진 ID가 URI 뒤에 붙어 URI를 통해 데이터를 읽거나 수정하거나 삭제할 수 있다. 예를 들어 온습도 센서(dht22)를 이용해 온도 데이터 저장 시 'http://localhost:port/temperature/dht22' 같은 URI로 요청을 받는다. 플랫폼에서는 Container는 temperature, Instance는 dht22로 생성되고 Element는 ID가 1로 생성되었다 가정한다. 이때 Read, Update, Delete는 'http://localhost:port/temperature/dht22/1'과 같은 URI로 요청이 이루어진다.

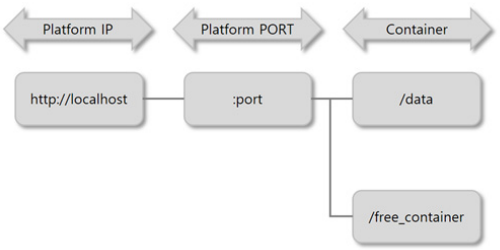

Fig. 4. Container Example

3.4 Redis 설계

REST API의 설계를 위해서 먼저 Redis의 자원에 대한 설계가 있어야 한다. 본 절에서는 Redis의 Key와 value의 저장방식에 대해 설명한다.

3.4.1 Redis Key 설계

Key는 플랫폼으로 전달되는 Container와 Instance 자리의 URI를 사용하여 Key를 생성한다. Container만으로 URI를 사용하며 URI가 플랫폼으로 전달되면 Container 자리의 값을 Key로 사용하는 Hashes를 생성한다. 한번 설정한 Hashes의 Key는 변경할 수 없고 Key를 변경하려면 해당 API를 삭제 후 다시 설정해야 한다.

Element의 Key를 설정하기 위한 자료구조로는 set을 이용한다. 해당 데이터는 Container와 Instance까지 사용한 URI가 플랫폼으로 POST 요청이 있을 경우 사용되는 데이터로 Key는 "Container:Instance:Serial"로 한다. Serial은 Element를 생성하기 위한 데이터로 "Container:Instance" 변환한 Key에 Serial을 추가하여 사용한다. Container:Instance:Serial"이라는 Key는 변경이 불가능하며 Redis Server가 재시작할 경우 데이터는 초기값으로 설정한다.

3.4.2 Redis Value 설계

Value를 저장하는 자료구조는 Hashes를 사용한다. Hashes로 자료구조를 선택한 이유는 JSON 데이터를 저장하기 위한 가장 적합한 자료구조이기 때문이다. JSON 형태의 특징 중 하나는 name과 value를 구분하여 사용하는 것이다. 플랫폼으로 요청이 들어오면 body 부분의 데이터를 추출하여 JSON 데이터로 변환하고 JSON 데이터 중 name에 해당하는 부분을 Hashes의 field로 하고 JSON 데이터의 value에 해당하는 부분을 field의 value로 저장한다. 또한, Hashes는 데이터 추가, 변경 시 새로운 field를 추가하거나 일부 field의 value를 변경하기에도 적합하다. 데이터를 추가할 경우는 JSON 데이터의 name 부분과 동일

Table 1. APITS 운영환경

Composition	Software
IoT Device (HTTP request), Client	Chrome v56.0.2924.87
	Postman v4.10.3
APITS (API Thin Server)	Crow web framework
	Boost v1.63.0
	hiredis v 0.13.3
	C++ 11
Database Server	Redis v3.2.8

한 field가 없을 경우 field가 추가되고, JSON 데이터의 name 부분과 같은 field가 존재할 경우 데이터를 입력하면 해당 field의 value를 변경한다. 즉, 데이터를 입력 시 JSON 데이터의 name 부분을 이용하여 데이터를 추가하거나 변경한다.

3.5 플랫폼 구현

3장에서 설계한 내용을 토대로 플랫폼을 구현했다. 전체 구성은 표1처럼 크게 3가지로 나뉜다. HTTP 요청을 하는 IoT Device, 데이터 형식 변환 및 Platform을 운영하는 APITS(API Thin Server), 데이터를 저장하는 Database Server로 구성했다. 플랫폼이 구동되는 PC의 CPU는 Intel(R) Core(TM) i5(CPU 1.6GHz), 메모리는 8GB이다. 또한, 소형 서버 기능을 수행하는 소형단말의 CPU는 4 × ARM Cortex-A53(1.2GHz) 이고 메모리는 1GB이다.

3.5.1 Client

Client는 REST API에 맞게 요청하는 상황을 위해 실제의 Client가 아닌 가상의 Client로 동작하는 환경을 두 가지로 구성했다. 먼저 Web browser 역할로는 Chrome, IoT Device는 Postman을 사용한다. Web browser는 플랫폼에 데이터를 요청하는 역할로 플랫폼에 해당 데이터를 요청하는 경우를 예로 들었다. Container의 경우 그림 5에서 "http://localhost:8005/management"와 같은 URI를 통해 요청하고, Instance와 Element는 "http://localhost:8005/management/users/1"와 같은 URI를 통해 요청하면 해당 데이터가 JSON 형태로 전달받는다. 만약 요청한 데이터가 없을 경우는 400 에러를 반환하고 플랫폼에서 제공하는 URI 형태에 맞지 않을 경우 404 에러를 반환한다. 데이터는 3.4절의 예시 데이터를 이용한다.

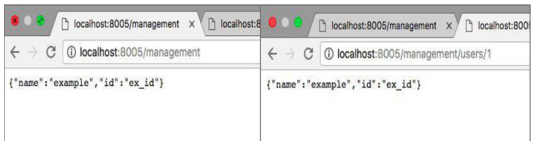

Fig. 5. Client(web browser)

IoT Device는 Postman을 이용해서 GET, POST, PUT, DELETE를 수행한다. Container의 URI는 그림 6에서 "http://host:port/management"로 Instance와 Element는 "http://host:port/management/users"와 같은 URI를 통해 사용한다. 각 URI의 host와 port는 Web browser와 마찬가지로 "localhost:8005"이다. 또한, users 데이터를 JSON 형태로 만들어서 전달한다.

요청이 성공적으로 완료되면 요청의 성공을 알려주고 잘못된 데이터 형식을 사용하면 400 에러를 반환하고 플랫폼에서 제공하는 URI 형태에 맞지 않을 경우 404 에러를 반환한다.

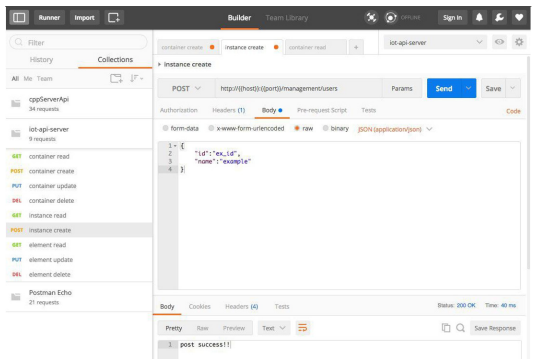

Fig. 6. Client(Postman)

3.5.2 IoT API Platform

IoT API Platform의 경우 C++ 언어를 사용하여 raspberry pi와 같은 소형 단말에서도 컴파일이나 구동에 무리 없게 구현했고, 현재 플랫폼은 macOS Sierra와 raspbian-jessie에서 컴파일 가능하다. 또한, RESTful service 규칙을 준수한 URL을 사용하며 8005번을 기본 포트로 사용한다. C++를 이용해 HTTP 요청을 처리하기 위해 "Crow"라는 web framework를 사용하여 Container, Instance, Element를 라우팅한다. "Crow"에 사용된 라이브러리로는 Boost(v1.63.0)이다. 데이터를 JSON 형태로 처리하기 위해 사용하는 라이브러리는 "Crow"에서 제공하는 JSON 라이브러리를 사용하여 데이터를 추출하거나 생성한다.

IoT API Platform에서 Persistence, Database Connection Pool은 Redis server에 접속하기 위해 C 기반으로 만들어진 hiredis 라이브러리를 C++로 변환

하여 사용한다.

3.5.3 Database Server

Database Server는 플랫폼으로 들어오는 데이터를 저장하는 서버로 Redis 3.2.8버전을 사용하여 플랫폼과 같은 장치에서 구동되며 Redis의 기본 포트인 6379를 사용한다. Database Server 구동은 daemon으로 실행하지 않으며 Database Server에 접속하는 Client(플랫폼)의 연결을 유지하기 위해서 timeout을 0으로 설정했다. Database Server의 경우 In-memory Database인 Redis의 특성상 Database Server가 재가동되면 데이터가 사라지므로 일정 주기(15분)에 1개 이상의 key 변경이 발생한 경우 데이터를 디스크 파일로 저장한다. 디스크의 저장되는 파일의 이름은 dump.rdb로 설정되어 있고 서버가 설치된 경로에 저장되며 초기 실행 시 dump.rdb 파일에 저장된 데이터를 메모리로 읽어온다.

플랫폼 초기 구동 시 Database connection pool 모듈에서 초기 연결 가능한 최대 Connection 객체를 요청하는데 최대 Connection 객체는 100개까지 가능하도록 설정했으나 플랫폼에서는 일부만으로 요청하는 데이터 처리가 가능하므로 50개의 Connection 객체만 요청한다. 그러나 플랫폼 사용자가 처리해야 하는 데이터의 크기가 클 경우는 Connection 객체의 요청을 늘려야 한다.

Ⅳ. 성능 테스트

4.1 테스트 환경 구축 및 시나리오 구상

플랫폼의 구현 환경은 mac의 경우 CPU는 Intel(R) Core(TM) i5(CPU 1.6GHz)이고, 메모리는 8GB이다. 또한, raspberry pi 3의 경우 CPU는 4 × ARM Cortex - A53(1.2GHz)이고 메모리는 1GB이다. 각각의 환경에서 컴파일된 실행 파일을 구동하며, Database Server는 Redis를 사용하며 동일한 환경에 각각 설치한다. Redis의 경우 설치 후 환경설정 과정을 마친 후 실행하여 플랫폼을 구축한다. 테스트 도구는 가장 널리 활용되고 있는 오픈소스 소프트웨어인 Apache JMeter를 활용한다. Apache JMeter의 구현 환경은 Intel core i5(CPU 2.5GHz), 메모리 8GB이다.

테스트 환경 구성은 그림 7과 같으며 플랫폼이 처리할 수 있는 CRUD 요청에 대한 성능 테스트를 진행하여 macbook air와 raspberry pi 3가 각각 처리할 수 있는 동시 접속자의 수를 확인하고 이때 서버의 응답 속도를 통해 응답 지연 시간에 대한 임계값을 산출한다.

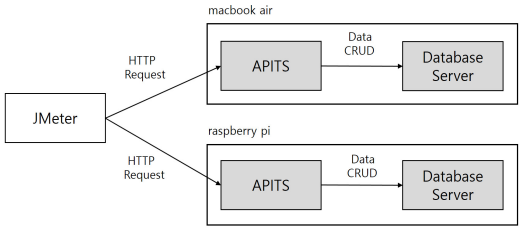

Fig. 7. APITS test environment configuration

4.2 성능 테스트 결과 및 요약

이 절에서는 4.1절을 토대로 진행한 성능 테스트에 대하여 요약한다. 성능 테스트는 GET, POST, PUT 요청에 대한 요약이며 DELETE 요청의 경우 처리 속도가 POST와 같아 추가 테스트를 진행하지 않았다.

GET 요청의 경우 Container와 Element 두 요청에 대한 테스트를 진행하였고 데이터는 그림 7에서 보이는 데이터를 사용했다. GET 요청에 대한 결과는 그림 8과 같다. 에러율은 1,000명의 동시 접속자가 사용할 동안 0%의 에러율을 보이며 안정적인 결과를 보인다. 처리 성능의 경우 동시접속자 수가 500명일 경우까지 증가하여 최대 처리량을 보여주며 그 이상의 동시 접속자 발생 시 처리량이 떨어지는 것을 확인할 수 있다. 응답 속도의 경우 동시 접속자 수가 증가함에 따라 증가하지만 1,000명의 동시 접속자가 사용하여도 평균 응답 속도가 1초가 넘지 않는 결과를 보인다.

server	macbook air			raspberry pi		
Thread (users)	100	500	1,000	100	500	1,000
Roof (N/A)	100	100	100	100	100	100
Average (ms)	58	299	468	88	401	799
Error (%)	0	0	0	0	0	0
Throughout (operations/sec)	1,151.1	1,229.3	1,103.9	713	798.6	752.3

Fig. 8. GET Request result

POST 요청의 경우 Instance 요청에 대한 테스트를 진행하였고 결과는 그림 9와 같다. GET 요청과 마찬가지로 동시 접속자가 1,000명까지 증가할 동안 에러율은 0%로 안정적인 결과를 보인다. 처리 성능의 경우 동시 접속자 수가 500명일 경우까지 증가하여 최대 처리량을 보여준다. 그 이상의 동시 접속자 발생 시 처리량은 점차 떨어진다. 응답 속도의 경우 동시 접속자 1,000명일 때 GET 요청보단 증가했지만 0.4초 차이로 미비하며 raspberry pi의 경우 동시 접속자 900명 이상이 되면 1초를 넘기지만 앞서 정한 최대 임계값인 3초보다 낮은 응답속도로 처리성능 또한, 안정적이다.

server	macbook air			raspberry pi		
Thread (users)	100	500	1,000	100	500	1,000
Roof (N/A)	100	100	100	100	100	100
Average (ms)	78	413	815	129	627	1,264
Error (%)	0	0	0	0	0	0
Throughout (operations/sec)	1,151.1	1,229.3	1,103.9	713	798.6	752.3

Fig. 9. POST Request result

server	macbook air			raspberry pi		
Thread (users)	100	500	1,000	100	500	1,000
Roof (N/A)	100	100	100	100	100	100
Average (ms)	41	206	608	100	520	1,055
Error (%)	0	0	0	0	0	0
Throughout (operations/sec)	1,992	2,149.5	1,298.4	844.4	909	874.8

Fig. 10. PUT Request result

PUT 요청의 경우 Element 요청에 대한 테스트를 진행하였고 데이터는 그림 10에서 보이는 데이터 중 일부를 수정하여 사용했다. GET 요청과 마찬가지로 동시 접속자가 1,000명까지 증가할 동안 에러율은 0%로 안정적인 결과를 보인다. 처리 성능의 경우 raspberry pi는 동시접속자 수가 500명일 경우까지 증가하지만 macbook air는 동시 접속자 700명까지 증가하여 최대 처리량을 보여준다. 그 이상의 동시 접속자 발생 시 처리량은 점차 떨어진다. 응답 속도의 경우 1,000명일 때 GET 요청보단 증가했지만 0.2초 차이로 POST 요청보다 미비하며, raspberry pi의 경우 동시 접속자 1,000명 이상이 되면 1초를 넘기지만 임계값인 3초보다 낮은 응답속도로 안정적인 결과를 보인다.

V. 결 론

본 논문에서는 초기 중, 소, 스타트업 기업들이 플랫폼을 결정하고 구축하는 기간과 비용을 절약하기 위한 방안으로 보편적으로 사용하는 웹 프로토콜인 HTTP를 이용하여 소형 단말에서 구동 가능한 플랫폼을 설계 및 구현하였다. 제안하는 플랫폼을 사용함으로써 IoT에 연동되는 디바이스 및 센서를 빠르게 테스트할 수 있으며, 적은 비용으로 서버를 구축하여 초기 테스트 비용을 절약할 수 있다. 또한, C++와 Redis를 사용함으로써 데이터 처리능력을 향상시켜 IoT에 연동되는 디바이스 및 센서가 처리하는 방대한 데이터를 신속히 처리할 수 있는 IoT용 시스템을 구축할 수 있다.

제안하는 플랫폼은 RESTful Service를 지원하며, 사용자의 요청에 맞게 데이터의 다양성을 수용 가능하다. 데이터 생성 시 REST API를 사용자의 임의대로 지정하면 자동으로 API 생성하여 플랫폼의 API 규칙만 파악하면 간단하게 해당 자원에 CRUD 요청을 처리할 수 있다.

향후 연구에서는 제안한 플랫폼에서 API 생성 규칙에 대한 자유도를 주기 위해 현재 Container, Instance, Element로 제공하고 있는 API 규칙을 플랫폼 사용자가 자유롭게 정할 수 있도록 할 필요가 있다. 예를 들어 Container가 Container를 여러 개 담을 수 있도록 진행한다. 또한, 플랫폼에 저장된 데이터를 다른 데이터베이스 서버로 자동으로 전달하여 플랫폼에 저장되는 데이터를 보다 안전하게 보존하기 위한 방법을 연구한다. 마지막으로 플랫폼의 보안을 위해 HTTPS를 적용한다.

References

[1] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, Information and Computer Science, Univ. of California, IRVINE, 2000.

[2] M. Sikong, "A study of RESTful web service definition for remote management and its appliance for integrated security management," *JKIIT*, vol. 12, no. 1, p. 11, 2014.

[3] *DB-ENGINES Ranking* [Internet], http://www.http://dbengines.com/en/ranking.

[4] J. M. Choi, D. W. Jeong, J. S. Yoon, and S. Lee, "Digital forensics investigation of redis database," *IPS Trans. Comp. and Commun. Sys.*, vol. 5, no. 5 pp. 117-126, May 2016.

[5] J. S. Oh and C. G. Song, "Transmission performance of improvements in mobile applications via XML and JSON data translation," in *Proc. KISSC*, vol. 39, pp. 129-131, Jun. 2012.

[6] Crow, https://github.com/ipkn/crow.

[7] OCEAN, http://www.iotocean.org.

서 보 민 (Bomin Seo)

2018년 2월 : 한밭대학교 정보통신공학과 (공학사)
2018년 3월~현재 : 한밭대학교 모바일융합공학과 석사과정
<관심분야> 데이터베이스, 웹 프로그래밍

장 병 순 (Byeong-soon Jang)

2018년 2월 : 한밭대학교 정보통신공학과 (공학사)
2018년 3월~현재 : 한밭대학교 모바일융합공학과 석사과정
<관심분야> 데이터베이스, 웹 프로그래밍

오 형 석 (Hyeuong-seok Oh)

2015년 2월 : 한국대학교 전파공학과 졸업
2015년 9월~2017년 8월 : 한밭대학교 전파공학과 석사 졸업
2017년 8월~현재 : (주)유미테크 사원
<관심분야> 사물인터넷, 시스템 프로그래밍, IoT 서버, 데이터베이스

박 현 주 (Hyun-ju Park)

1990년 2월 : 서울시립대학교 전산통계학화(공학사)
1992년 2월 : 서울대학교 전산과학과(공학석사)
1997년 2월 : 서울대학교 전산과학과(공학박사)
1998년 4월~2000년 3월 : 대전산업대학교 정보통신학과 전임강사
2000년 4월~현재 : 국립 한밭대학교 정보통신공학과 교수
<관심분야> 프로그래밍 언어, 운영체제, 데이터베이스