The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 17, No. 2, pp.45-50, Apr. 30, 2017. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2017.17.2.45
JIIBC 2017-2-7

비동기 기반 마이크로 서비스에 적용 가능한 이벤트 스트림 처리 프레임워크 제안

A Proposal of Event Stream Processing Frameworks applicable to Asynchronous-based Microservice

박상일*

Sang Il Park*

요 약 마이크로서비스 아키텍처(Microservice Architecture)는 실시간 실감 미디어 방송시스템과 같이 대규모 분산 시스템에 적합한 서비스 아키텍처의 하나이다. 스케일-아웃(Scale-Out)기법 과 같은 수평적 성능 확장이 쉽기 때문에 최근 넷플릭스나 트위터와 같은 서비스 플랫폼 업체들이 앞다투어 이와 같은 시스템을 도입하고 있다. 또한 마이크로서비스 아키텍처는 기존의 REST와 같은 웹 API에서 처리하기 어려운 영상처리나 실시간 데이터 분석 등을 비동기 기반의 프로세싱를 이용하여 처리 가능하게 하고 있다. 본 논문은 IoT 센서 데이터 분석이나 대용량 실감미디어를 실시간으로 편집하는 클라우드 기반 영상편집과 같은 다수의 이벤트들이 스트림으로 발생하며 플랫폼 내에서 비동기로 처리하는 상황에서 이벤트의 처리 순서가 보장되지 않음을 실험으로 증명하고 이에 알맞은 비동기 기반 마이크로서비스에 적용 가능한 이벤트 스트림 처리 프레임워크를 제안한다.

Abstract Micro-service Architecture is a service architecture optimized for large-scale distributed systems such as real-time realistic broadcasting systems, so that are fiercely adopted by Global leading service platform vendors such as Netflix and Twitter due to the merit of horizontal performance scalability enabling the scale-out technique. In addition, micro-service architecture makes it possible to execute image processing and real-time data analysis using an asynchronous-based processing that are difficult to handle in Web API such as REST. In this paper, an event stream processing framework applicable to asynchronous based micro services is proposed in the sense that the accountability of event processing order is not guaranteed in the events such as IoT sensor data analysis or cloud-based image editing because these are the situations where the real-time media editing generates multiple event streams and asynchronous processes in the platform.

Key Words : Microservice, AMQP, cloud computing, message queue, event stream

Ⅰ. 서 론

클라우드 컴퓨팅은‘ 인터넷 기술을 이용하여 가상화된 IT 자원을 서비스로 제공하는 것으로 사용자는 소프트웨어, 스토리지, 서버, 네트워크 등 IT 자원을 필요한 만큼 빌려서 이용하고, 서비스 부하가 커짐에 따라서 실시간 확장이 가능하며, 이용한 만큼 비용을 지불하는 컴퓨팅’으로 정의하고 있다[1].

*정회원, 서울과학기술대학교 전자IT미디어공학과(교신저자)
접수일자: 2017년 4월 7일, 수정완료: 2017년 4월 11일
게재확정일자: 2017년 4월 11일

Received: 7 April, 2017 / Revised: 11 April, 2017
Accepted: 11 April, 2017
*Corresponding Author: sangilparkmail@gmail.com
Electronic IT Media Engineering, Seoul National University of Science and Technology, Republic of Korea

이미 클라우드 컴퓨팅은 기존의 IT 인프라스트럭처(Infrastructure)구축을 위해 지출되는 비용을 줄여주고 고가용성과 확장성이 입증되어 다양한 서비스를 제공하는 플랫폼 사업자들에 의해 앞 다투어 도입되고 있다.

클라우드 컴퓨팅은 확장성을 위하여 가상화(Virtualization)되어 동작하는 여러 인스턴트 노드를 가지고 처리를 분산하여 성능을 높이는 스케일-아웃(Scale-Out)기법을 통해 대용량 고속처리가 가능하다는 특징을 가지고 있다[2]. 기존의 컴퓨팅이 단일 노드의 성능을 향상하여 처리 속도를 향상시키는 스케일-업(Scale-Up)기법을 통하여 성능을 확장시키는데 중점을 두어왔지만, 단일 프로세서의 집적 능력이 한계에 이르러 무어의 법칙이 깨지고 있는 현재로서는 스케일-아웃 기법을 이용하여 대용량 시스템을 구성하는 것이 활발하게 연구되고 있는 실정이다.

기존의 서버 아키텍처는 단일 서버 내에 모든 서비스가 집적되어 있는 모놀리딕(Monolithic) 구조를 가지고 있다. 기업 등에서 많이 채택하고 있는 스프링 프레임워크(Spring Framework)등이 모놀리딕 구조를 가진 프레임워크의 대표적인 예이다. 이와 같은 구조를 채택하면 서버 내에서 일어나는 제어의 흐름을 관리하기가 쉬워지기 때문에 OLAP(Online Analytical Processing), BI(Business Intelligence) 등의 트랜잭션 처리나 비즈니스 로직을 처리하기가 쉽다. 하지만 특정한 위치에서 일어나는 오류 하나가 서버의 중단을 야기할 수도 있고, 서버를 기동하는데 몇 시간 이상이 걸리기도 하는 등 가용성 측면에서 다양한 문제를 발생시키게 된다[3].

따라서 클라우드 컴퓨팅에 알맞은 새로운 아키텍처가 연구되게 되고 기존의 서비스 지향 아키텍처(Service Oriented Architecture)라고 불리던 아키텍처를 근간으로 마이크로서비스 아키텍처(Microservice Architecture)라는 새로운 아키텍처가 나타나게 된다. 2014년 M Fowler와 J Lewis가 제안한 마이크로 아키텍처는 기존의 모놀리딕 구조와는 다른 여러 가지 특징을 가지고 있다[4]. 그림 1은 모놀리딕 아키텍처와 마이크로 서비스 아키텍처의 구조를 나타낸 것이다. 기본적으로 마이크로 아키텍처는 작은 서비스들을 모아 어플리케이션을 만드는 아키텍처이다. 작은 서비스는 각각 하나의 프로세스면서 다른 서비스들과 통신할 수 있다. 이는 서비스 하나하나가 가볍고 오류의 책임을 작은 서비스가 지기 때문에 하나의 서비스가 중단되더라도 전체 서비스에 영향을 적게 미치게 된다. 이는 컨웨이의 법칙으로도 불리는 “소프트웨어의 구조는 그 소프트웨어를 만드는 조직의 구조와 일치한다.”는 주장에 따라 서비스를 책임지는 팀의 크기도 작아질 수 있다[5]. 또한 작은 서비스 단위로 빌드, 배포가 가능하기 때문에 빠르고 효율적인 서비스 수정 및 운영이 가능하다는 장점이 있다.

마이크로 서비스 아키텍처는 현재 넷플릭스와 트위터와 같은 서비스 사업자들에 의해 도입되기 시작했으며 기존의 웹 서비스에 적용되는데 그치지 않고 IoT서비스 플랫폼이나 빅데이터 분석, 대용량 비디오 처리등에도 도입되려는 움직임을 보이고 있다.

또한 마이크로 서비스는 비동기 기반의 처리를 근간으로 분산 처리의 효율성을 추구하고 있다. 비동기 처리는 싱글 스레드(Single Thread)기반의 이벤트 루프(Event Loop)를 이용하여 처리를 수행하며, 수행된 결과는 콜백(Call Back)함수를 통하여 응답하도록 하는 방법이다[6].

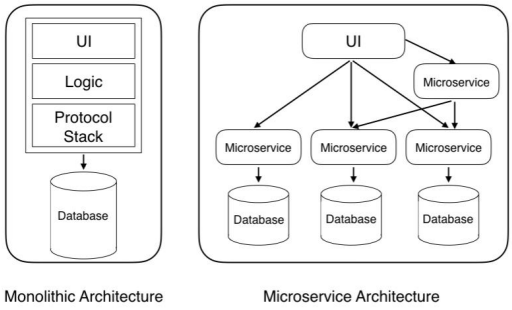

그림 1. 모놀리딕 아키텍처와 마이크로 서비스 아키텍처 구조
Fig. 1. Structures of monolithic architecture and microservice architecture.

비동기 기반의 처리를 이용하면 하나의 스레드로 여러개의 요청을 처리할 수 있기 때문에 요청이 빈번한 웹 서버 등에 사용하여 성능을 비약적으로 향상시킬 수 있으므로 C10K 문제라고 부르는 접속자가 많은 서버에 적용할 수 있다[7]. 비동기 기반의 웹서버는 이미 Nginx나 Node.JS등에 적용되어 LinkedIn이나 Paypal과 같은 다양한 서비스 사업자들이 선택하여 쓰고 있는 방식이다.

그러나 비동기 처리를 사용하게 되면 서비스의 처리 순서를 제어할 수 없게 되는 문제점이 생기게 된다. 이는 이벤트의 순서에 따라서 수행 결과가 나오는 동기 시스템과 달리 콜백 함수의 수행시간이나 네트워크의 지연시

간에 따라 결과의 수행이 이루어지는 비동기 시스템의 한계점이기도 하다.

따라서 본 논문은 2장에서 스테이트리스(Stateless) 서비스와 스테이트풀(Stateful) 서비스에서 비동기 처리가 야기할 수 있는 문제점들을 보여줄 것이다. 3장에서는 이벤트들에 순서가 있는 이벤트 스트림을 처리할 수 있는 비동기 마이크로서비스 처리 프레임워크를 제안하고 4장에서는 비동기 기반의 마이크로 서비스에서 실제로 처리시간에 따라 이벤트의 순서에 따라 결과값의 순서가 보장되지 않는 상황을 실험을 통하여 보여주고 제안한 시스템에서는 결과값의 순서가 보장됨을 증명할 것이다. 마지막으로 결론에서는 제안된 프레임워크가 IoT나 대용량 실감미디어 편집에 유용함을 보여줄 것이다.

II. 서비스의 분류

일반적인 클라우드에서 제공하는 서비스는 스테이트풀(Stateful)과 스테이스리스(Stateless)서비스로 분류될 수 있다[8]. 서비스를 분류하는 것은 서비스 아키텍쳐를 설계할 때에 중요한 제반사항이 될 수 있다.

1. 스테이트리스(Stateless) 서비스

스테이트리스 서비스는 사용자 인증이나 웹페이지를 제공하는 것과 같이 처리시간이 짧고 처리되는 상태가 즉각적으로 표현가능한 서비스를 뜻한다. 또한 이와같은 서비스의 처리 상태는 클라이언트(Client)측에서 관리된다. 스테이트리스 서비스는 Representational State Transfer Application Programming Interface(REST API) 게이트웨이(Gateway)에 직접 연결되어 바로 응답을 제공할 수 있다[8].

2. 스테이트풀(Stateful) 서비스

스테이트풀 서비스는 처리 구조가 복잡한 경우가 많다. 만약 사용자가 자신의 움직임을 실시간으로 반영하여 트랙킹 하는 지도 서비스나 푸쉬(Push)서비스를 서버에 요청하여야 하는 경우, 사용자가 발생시키는 이벤트가 실시간으로 서버에 삽입되며 서버는 실시간으로 응답을 돌려주어야 한다[8].

3. 서비스의 상태에 따른 비동기 기반 처리의 문제점

스테이스리스 서비스는 REST API에서 제어 흐름을 관리한다. 서비스의 처리 상태를 따로 알 필요가 없기 때문에 비동기 기반의 처리에 알맞다. 대부분의 웹 서비스는 요청이 일어나는 대로 즉시 처리되도록 설계하기 때문에 많은 요청을 처리 할 수 있는 비동기 기반의 웹서버를 적용하는 것으로도 많은 성능 향상이 일어난다.

그러나 클라우드 기반 대용량 미디어 편집이나 IoT센서 데이터 등을 가공, 처리하는 것과 같은 이벤트 스트림을 받아 처리해야 하는 작업은 각각의 요청에 대한 처리시간이 길고 명령의 순서가 제어되지 않으면 결과가 제대로 도출되지 않는다는 문제점이 있다. 이미 윤용익 등에 의하여 위의 사례와 유사한 실시간 분산 데이터베이스 시스템 내에서 비동기적으로 트랜젝션을 처리하는 방법 및 프로토콜에 관한 연구가 진행되었다[9]. 따라서 본 논문은 처리가 분산되어 있고, 각각의 처리가 마이크로서비스에서 병행성을 유지하면서 처리 시간과 관계없이 이벤트 스트림의 순서가 보존되는 새로운 프레임워크 구조와 프레임워크에서 주고받는 메시지의 형태를 제안하려고 한다.

III. 제안하려는 프레임워크

제안하려는 프레임워크는 현재 비동기 기반으로 구성되는 클라우드 상에서 이벤트 스트림 처리를 할 때 문제가 발생하는 상황을 개선하고자 제안하는 방법이다. 효율적인 자원 배분을 위하여 그림 2와 스테이트리스 서비스와 스테이트풀 서비스를 나누어 프레임워크에서 따로 처리하고, 모든 이벤트는 서비스 하모나이저(Service Harmonozer)혹은 브로커(Broker)라고 불리는중재 서비스가 서비스의 종류에 따라 나누어 서비스로 전달한다.

모든 이벤트 스트림은 그림 3과 같은 마이크로 시스템(Micro-system)이 처리한다. 클라이언트는 이벤트 스트림을 발생시킬 때 이벤트의 순서를 메시지 내부에 삽입하여 전달한다. 비동기 기반 마이크로서비스는 이벤트의 처리 순서와 상관없이 처리가 끝나는대로 콜백함수를 통하여 처리결과가 돌아오기 때문에 스트림과 같은 시계열(Time-sereis)데이터의 경우 처리결과를 단순히 큐(Queue)에 받아 반환하게 되면 결과값의 순서의 무결성

이 보장되지 않는다. 따라서 이벤트 스트림의 순서를 가지고 순서를 다시 재 정렬하는 과정이 필요하다.

그림 3의 태스크 리어셈블러(Task Re-assembler)는 큐에 쌓인 결과값을 정렬하여 돌려주는 역할을 한다. Redis와 같은 인-메모리(In-Memory)스토어를 이용하여 값을 저장하여 순서에 맞게 정렬하고 서비스 하모나이저에 전달하여 스트림 형식의 응답을 돌려주면 비동기 기반의 마이크로서비스에서도 이벤트 스트림 처리가 가능하다.

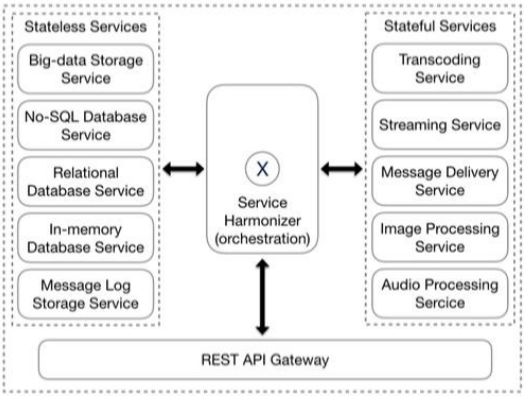

그림 2. 상태별 선택적 서비스를 제공하는 프레임워크 구성도
Fig. 2. A diagram of selective servce frameworks by status

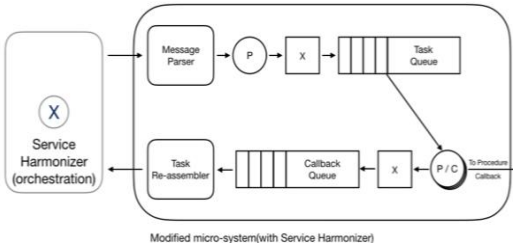

그림 3. 개선된 마이크로서비스 구조도
Fig. 3. A structure diagram of modified microservices

그림 4는 스테이트풀 서비스를 위하여 그림 3을 수정한 것으로 일반적으로 많이 사용되는 REST API를 사용할때에 발생하는 문제점을 해결하고자 하는 것이다. 기본적으로 HTTP는 서버에 메시지를 전달하고 응답을 돌려받는 폴링(Polling)방식을 이용한다. 이와 같은 경우 한 클라이언트와 서버는 세션이라고 불리는 메시지 전달 단위를 가지고 응답을 기다리게 되는데 만약 응답이 길어지는 경우 서버와 클라이언트가 가지고 있는 Time-To-Live 시간에 의해 일정시간이 지나게 되면 응답을 받지 못하게 된다. 이벤트 스트림 처리를 위한 클라우드 기반 대용량 미디어 편집이나 IoT센서 데이터등의 처리를 수행하기 위해서는 처리시간이 길어짐에 따라 응답을 받지 못하는 경우가 발생하며, 처리가 다 되었는지 상태를 확인하기 위해 새로운 세션을 다시 열어야 해서 자원이 낭비되는 문제가 발생한다. 그리고 프레임워크 내에 푸시 게이트웨이(PUSH gateway)를 구성하고 응답이 일정시간 이상 경과하게 되는 경우 기존의 HTTP 응답에 처리중이라는 상태응답을 전달하고, 처리가 끝나면 푸시 메시지로 서버쪽에서 직접 처리 결과를 클라이언트로 전달한다. 이때 푸시 메시지를 전달하는 프로토콜로 Advanced Message Queueing Protocol(AMQP)와 같은 메시지 전달의 신뢰성을 가지고 상태를 푸시할 수 있는 프로토콜을 사용하여 전체 시스템을 구성한다.

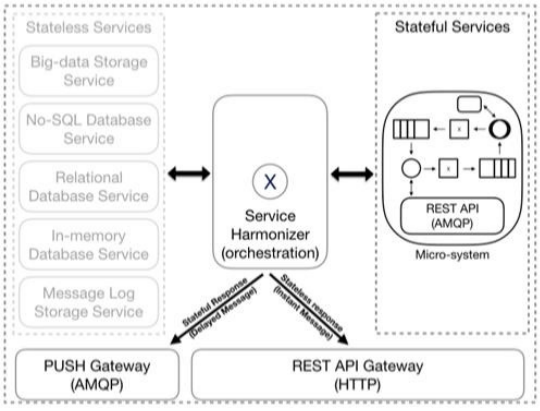

그림 4. 개선된 프레임워크 구조도
Fig. 4. A structure diagram of modified frameworks

Ⅳ. 실험 및 결과

일반적인 비동기 상황에서 이벤트들이 순서대로 들어오는 상황을 가정하기 위하여 간단한 실험을 해 보았다. 연속되는 문자열이 들어오는 상황을 만들기 위하여 세익스피어의 “The Sonnet 1”의 앞부분 텍스트의 제어문자와 공백을 모두 삭제한 2048byte를 가지고 테스트 시퀀스를 만들었다. 비동기 기반 웹서버 어플리케이션인 Node.Js v6.1.2로 서버를 구성하고 Python 2.7.10 버전으

로 클라이언트를 구성하였다. 클라이언트와 서버는각각 메모리에 테스트 시퀀스를 저장하고 있으며 클라이언트가 GET방식으로 연속하여 텍스트의 문자를 하나씩 서버로 전달하면 서버는 전달된 문자를 메모리에 저장하여 문자열을 만든다. 전송이 완료되면 서버에 저장된 문자열과 전달받은 문자열을 비교하여 잘못 전송되거나 순서가 잘못된 문자를 찾아서 전체 문자열에서 잘못된 문자의 비율을 계산한다. 일반적인 동기 처리 시스템에서는 순차적으로 처리하기 때문에 이와 같은 실험환경에서는 문자열의 순서에 큰 문제가 없으나 비동기 처리 시스템에서는 처리시간의 작은 차이에도 문자열의 순서가 크게 달라지게 된다. 네트워크의 전송지연과 비동기 처리 지연 상황을 시뮬레이션 하기 위하여 다음과 같은 공식을 통하여 표본의 크기를 결정하였다.

$$n \geq \left(\frac{z_{\alpha/2} s}{E} \right)^2 \quad (1)$$

식 1과 같이 $n = 20$인 예비표본을 취하여 $s = 0.01366$으로 추정하고 95% 신뢰수준에서 허용 오차를 ± 0.001366으로 결정하여 표본의 크기 $n = 389$을 구하였다. 서버의 콜백 함수에 지연상황을 가정하기 위한 함수를 추가하였으며, 평균 지연시간은 2, 5, 10ms로 가정하였다. 시뮬레이션을 위한 PC는 Intel I5 2.7Ghz CPU를 장착하고 8GB의 DDR3메모리를 장착하였다. OS는 macOS Sierra 10.12.4버전을 사용하였다. 테스트를 하는 모습은 다음 그림 5와 같다.

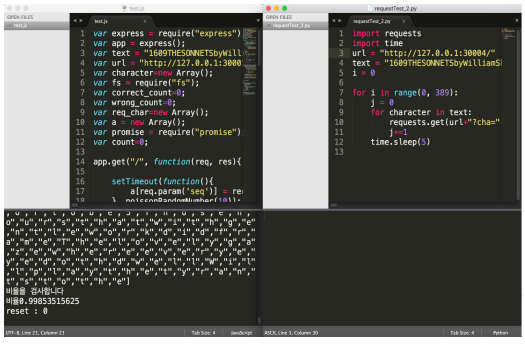

그림 5. 테스트 모습.
Fig. 5. A screenshot of test.

테스트를 통한 결과는 다음 그림 6과 같다.

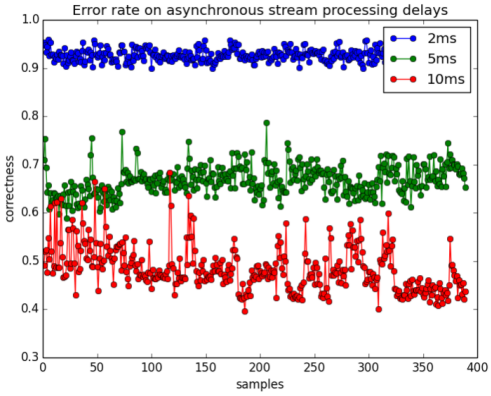

그림 6. 딜레이에 따른 비동기 스트림 처리의 에러율.
Fig. 6. Error rate on asynchronous stream processing delays

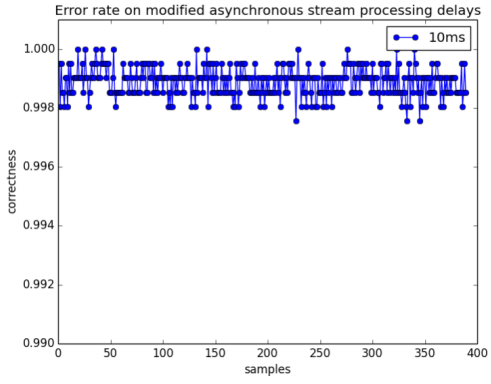

그림 7. 수정된 비동기 스트림 처리의 에러율.
Fig. 7. Error rate on modified asynchronous stream processing delays

위의 그래프와 같이 기존의 비동기 시스템에서는 평균 지연율의 비교적 작은 변화에도 전체 시스템의 순서가 매우 달라지게 된다. 이벤트의 순서를 보장하기 어렵기 때문에 스테이트풀 서비스나 스트림 처리는 거의 어렵게 된다. 그림 7은 클라이언트에서 이벤트의 순서를 전달하고, 추가적인 라이브러리 없이 키-밸류(Key-Value) 자료구조를 이용하여 소스코드를 수정한 것이다. 10ms의 지연에도 불구하고 거의 오류가 나타나지 않음을 볼 수 있다. 또한 Node.Js서버를 다룰 때 사용하는 자바스크립트의 특성에 따라 순서를 이용하여 이벤트 스트림 데이터를 처리하여도 처리 지연이 거의 발생하지 않는다.

V. 결 론

본 논문에서 제안하는 프레임워크는 이벤트 기반의 IoT 어플리케이션이나 클라우드 기반 비디오 편집 어플리케이션에 매우 적합한 구조를 가지고 있다. 특히 시스템의 규모가 커지고, 대용량 처리가 필요한 경우 서버 시스템을 구축하기 위한 노력과 비용을 크게 절약할 수 있다. 현재 제안된 프레임워크를 시험하기 위하여 AMQP를 구현하는 RabbitMQ 메시지 프레임워크와 Infrastructure as a Service(Iaas)기반의 클라우드 플랫폼을 활용한 테스트 벤치를 구축하고 있다. 현재 제안내용은 특허 출원중에 있다. 앞으로 IoT환경이나 대용량 실감미디어를 처리하는데에서 발생하는 이벤트 스트림을 비동기 기반으로 처리하는데 본 프레임워크가 많은 도움이 될 것으로 기대한다.

References

[1] ITU's Telecommunication Standardization Sector (ITU-T), "Cloud computing-Overview and Vocabulary(Y.3500)", 2014.7.7.

[2] ZHANG, Shuai, et al. "Cloud computing research and development trend.", In: Future Networks, 2010. ICFN'10. Second International Conference on. Ieee, 2010. p. 93-97.

[3] BALALAIE, Armin; HEYDARNOORI, Abbas; JAMSHIDI, Pooyan. "Microservices architecture enables DevOps: migration to a cloud-native architecture.", IEEE Software, 2016, 33.3: 42-52.

[4] FOWLER, Martin; LEWIS, James. "Microservices. ThoughtWorks.", http://martinfowler. com/articles/ microservices. html, 2014.

[5] Melvin E Conway. "How do committees invent",. Datamation, 14(4):28 - 31, 1968.

[6] TILKOV, Stefan; VINOSKI, Steve. "Node. js: Using JavaScript to build high-performance network programs." IEEE Internet Computing, 2010, 14.6: 80-83.

[7] LIU, Dong; DETERS, Ralph. "The reverse C10K problem for server-side mashups.", International Conference on Service-Oriented Computing. Springer Berlin Heidelberg, 2008. p. 166-177.

[8] Sunggeun Yoo et al. "A study of message-oriented service framework for serverless software architecture.", ISAAC 2016, AACL 08, pp. 214~215, 2016

[9] Yong-Ik Yoon. "A Study on the Asynchronous transaction Processing in Distributed Real-time Database Systems." Database Research, 11.4 (1995.12): 38-50.

저자 소개

박 상 일(정회원)

- 1977년 : 연세대학교 전자공학과 졸업(공학사)
- 1983년 : Kansas State University 전기전자공학과 졸업(석사)
- 1987년 : University of New Mexico 전기전자공학과 졸업(박사)
- 1987년 : University of Pittsburgh 전자공학과 조교수
- 1988년 : (미)Motorola Semiconductor DSP Design Manager
- 1995년 : 삼성전자 전무(반도체팀장, 비서실, 본사경영기획실장등)
- 2006년 : 스카이레이크 인큐베스트 투자팀 부사장
- 2009년 : 방송통신위원회 차세대 방송 PM
- 2012년 ~ 현재 : 서울과학기술대학교 전자IT미디어공학과 교수

<주관심분야 : 차세대 방송, 실감방송, 클라우드 플랫폼, IoT/IoL>

※ 본 연구는 미래창조과학부 및 정보통신기술진흥센터의 정보통신 · 방송 연구개발사업의 일환으로 수행하였음. [2016-0-00144, 시청자 이동형 자유시점 360VR 실감미디어 제공을 위한 시스템 설계 및 기반기술 연구]