서버리스 컴퓨팅 시스템을 위한 효율적인 마이크로 평션 관리기 구현에 관한 연구

장수민, 온진호, 김영호, 김재열, 차규일
한국전자통신연구원
e-mail : {jsm,onjinho,kyh05,gauri ,gicha}@etri.re.kr

A Study on the Implementation of Efficient Micro Function Manager for Serverless Computing Systems

Su-Min Jang, Jin-Ho On, Young-Ho Kim, Chei-Yol Kim, Gyu-Il Cha
Electronics and Telecommunications Research Institute

요 약

최근에 이슈가 되고 있는 서버리스 컴퓨팅 시스템은 실행 요청이 있을 때만 동적으로 머신 자원의 할당으로 해당되는 마이크로 평션을 실행되고 그 요청이 증가할 경우 그에 비례하는 자원을 할당하여 동시에 처리하는 방법으로 자원 사용률과 서비스 확장성이 매우 좋은 장점을 갖는다. 그러나 서버리스 컴퓨팅 시스템 관련 개발자는 여전히 마이크로 평션에 대한 관리 측면에 대한 다양한 대응을 스스로 해결해야 하는 여러가지 문제점을 가지고 있다. 그래서 본 논문은 서버리스 컴퓨팅 시스템의 효율적인 마이크로 평션 관리와 마이크로 평션의 실행환경설정에서 발생되는 불필요한 자원 낭비를 최소화하는 방안을 제안한다.

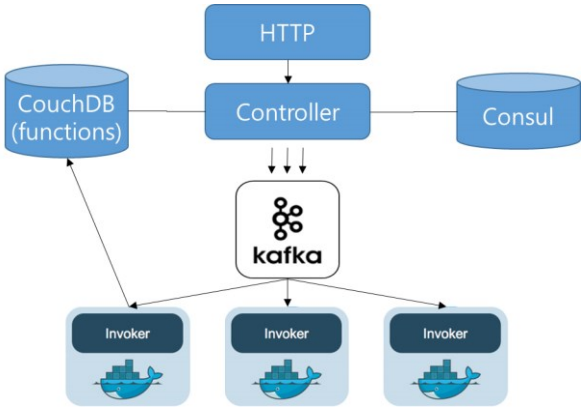
1. 서론

서버리스 컴퓨팅[1][2]은 클라우드 응용 프로그램 배포를 위한 새로운 패러다임을 제공하고 있다. 서버리스 컴퓨팅과 유사한 의미를 갖는 마이크로 서비스 아키텍처[3-5]도 기존 아키텍처와 달리 이벤트 트리거 되고 클라우드 제공자에서 완전히 동적으로 머신 자원의 할당을 관리하며 컨테이너에서 실행되는 것이 특징을 갖는다. 이러한 서버리스 컴퓨팅의 대표적인 사례가 AWS Lambda [6] 및 Google Cloud[7]와 같은 프레임 워크 등이 이에 해당된다. 서버리스 컴퓨팅에서 실행되는 마이크로 평션은 다양한 언어로 작성되고 웹 서비스로 접근이 가능하다. 그 작성된 마이크로 평션은 호출하면 이전 실행 컨테이너 또는 밀리 초 내에 새로운 컨테이너나 재사용 된 컨테이너를 통해 이를 실행되는 방식을 기본 구조로 하고 있다. 서버리스 컴퓨팅의 대표적인 장점은 아래와 같다.

- 서버 및 운영 환경에 대부분의 제반 사항을 클라우드 공급자가 제공하고 개발자는 비즈니스 로직에 집중하고 서버 관련 개발에 대한 부담이 적다.
- 함수 실행 단위로 사용된 시간이나 자원에 대한 세분화된 지불 모델은 개발 시간과 운영 비용을 효율적으로 관리할 수 있다.
- 실행 요청이 있을 때만 필요한 해당되는 마이크로 함수가 실행되고 그 요청이 증가할 경우 그에 비례하는 자원을 할당하여 동시에 처리하는 방법으로 자원 사용률과 서비스 확장성의 극대화가 용이하다.

이러한 서버리스 컴퓨팅의 장점으로 인해 최근 그 적용이 확산되고, 일반적인 클라우드 애플리케이션을 서버리스 플랫폼 상에서 개발 요구가 증가하고 있다[8][9]. 아래 (그림 1)은 Apache OpenWhisk 의 실행 아키텍처를 보여준다. Apache OpenWhisk[10]의 경우에는 서버리스 플랫폼이 차세대 클라우드 플랫폼으로 주목받으면서 IBM 이 주도적으로 개발하면서 퍼블릭/프라이빗 클라우드를 모두 제공하고 오픈소스 서버리스 플랫폼같은 서버리스 플랫폼으로 최근에 각광을 받고 있다. OpenWhisk 에서 마이크로 평션의 저장 및 관리하는 주체는 CouchDB 인데 다양한 버전 마이크로 평션 유지나 마이크로 평션 사전 최적화, 마이크로 평션에 대한 일시적 정지와 같은 다양한 서비스 운영에 대응하는 부분은 미미하다. 그래서 본 논문은 서비리스 컴퓨팅에 마이크로 평션의 효율적인 관리와 마이크로 평션의 실행 환경에서 발생하는 불필요한 자원 낭비를 최소화하는 방안을 제안한다.

이 논문은 2019 년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임 (No.2017-0-00053,심혈관질환을 위한 인공지능 주치의 기술 개발)



(그림 1) OpenWhisk 실행 구성도

2. 마이크로 펑션 관리

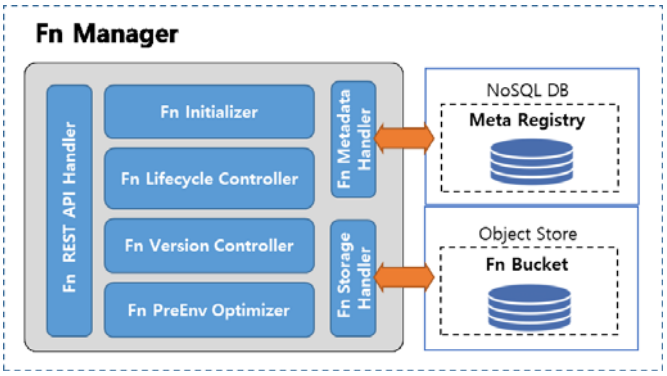
본 장에서는 마이크로 펑션 관리는 마이크로 펑션 개발자가 마이크로 함수 생성/수정/삭제/유지/관리 측면에서 편리성을 제공하고 마이크로 펑션을 실행하는 과정에서 특정 함수에 대한 정지/해제 등과 같은 다양한 기능을 어떻게 제공되는지 서술한다. 특히, 제안하는 마이크로 펑션 관리기의 특징인 마이크로 펑션의 실행 환경에서 낭비되는 자원을 최소화하는 방법을 제시한다.

2.1 마이크로 펑션 관리기 구성도

서버리스 마이크로 펑션 서비스 플랫폼에서 마이크로 펑션 관리기(Fn Manager)는 다음과 같은 모듈들로 구성된다.

- Fn Initializer
- Fn REST API Handler
- Fn Version Controller
- Fn Lifecycle Controller
- Fn Metadata Handler
- Fn Storage Handler
- Fn PreEnv Optimizer

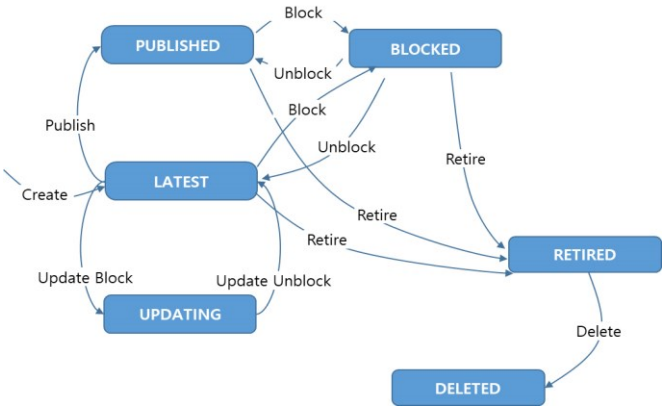
Fn Initializer 는 메타 데이터 저장소를 초기화 및 로딩하거나 REST API 인터페이스를 통한 마이크로 펑션 실행 요청에 서비스를 제공하는 관련 프로세스들을 초기화하고 실행하는데 그 역할을 하게 된다. Fn REST API handler 는 API 게이트웨이는 등록된 API 에 대해 통합된 단일 엔드포인트(endpoint)를 제공하여 클라이언트 혹은 API 단위 간의 호출을 단순화시키며 API 단위 추가 시 동적 라우팅 및 부하 분산 기능을 제공한다. Fn Version Controller 는 마이크로 펑션을 생성하고 수정하는 과정에서 버전들을 유지하거나 개발 과정에서 개발, 베타, 프로덕션 등 다양하게 변형된 마이크로 펑션으로 관리하는 역할을 한다. 즉 마이크로 펑션에 대한 다양한 버전을 관리하는 부분이다. Fn Lifecycle Controller 는 마이크로 펑션이 생성되어서 폐기될때까지 상태를 설정 및 관리하는 부분이다. 이외에 Meta Registry 에 마이크로 펑션의 정보를 저장하기 위해 사용되는 Fn Metadata Handler 와 마이크로 펑션 실행 코드와 환경에 관련된 파일들을 저장하기 위해 사용되는 Fn Storage Handler 모듈이 있다. Fn PreEnv Optimizer 는 2.5 에서 자세히 설명한다. (그림 2)은 Function Manager 구성도를 보여준다.



(그림 2) Function Manager 구성도

2.2 마이크로 펑션 라이프사이클 관리

Function Manager 는 Fn Lifecycle Controller 을 이용하여 마이크로 펑션을 현재 어떤 상태인지 식별할 수 있는 5 단계의 Lifecycle 상태를 관리한다. (그림 3)는 마이크로 펑션 라이프사이클의 상태 흐름을 보여준다.



(그림 3) Function Lifecycle 상태도

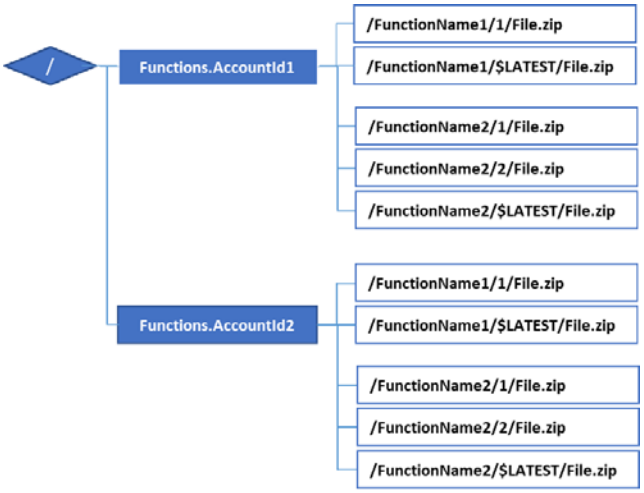
Lifecycle 각 단계의 역할은 다음과 같다.

- LATEST: 마이크로 펑션이 새롭게 생성되거나 기존에 등록된 마이크로 펑션이 마지막 수정된 상태가 LATEST 이다.
- PUBLISHED: 마이크로 펑션이 생성되고 수정되면서 다양한 버전을 등록되는 상태를 의미한다.
- BLOCKED: 마이크로 펑션에 대한 액세스가 일시적으로 차단하기 위한 상태이다.
- UPDATING: 마이크로 펑션이 수정(Updating)되는 동안에 Update Block 상태로 실행이 차단되는 상태이다. BLOCKED 와 차이점은 수정이 끝나면 Unblock 이 되면 LATEST 상태가 되고 삭제 상태로 바로 변경되지않는다.

RETIRED: 마이크로 펑션이 실행이 취소되고 최종 삭제되기 위한 상태이다.

2.3 사용자별 마이크로 펑션 파일 관리

마이크로 펑션에 대한 다양한 버전을 관리하는 마이크로 펑션 파일 저장 구조는 아래 (그림 4)와 같다. 특징은 사용자(Accouut)별로 각각의 시작 디렉토리(Bucket)를 만들고 그 아래에 해당하는 마이크로 펑션들을 저장한 구조이다. 이렇게 생성된 디렉토리안에는 "/마이크로 펑션명/버전/파일에 관련된 압축 파일명" 과 같은 개념적인 구조로 파일이 저장되는 구조이다.



(그림 4) Function 파일 저장 구조

2.4 효율적인 마이크로 펑션 실행 환경 변수

마이크로 펑션은 실행 코드와 마이크로 펑션이 실행되는 환경 설정(Configure)정보를 갖는다. 아래와 같은 실행 정보들을 마이크로 펑션을 생성할 때 지정한다.

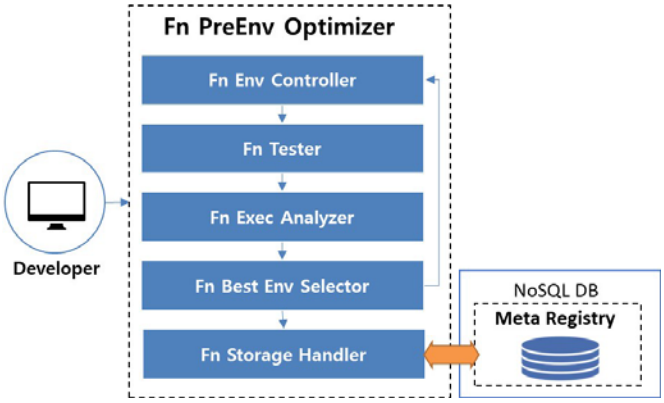
- BaseImage: 자신이 실행될 컨테이너 기본 이미지
- MemorySize: 64MB 에서 MAX_MEMORY(한 개의 Node 가 갖는 최대 Memory Size)까지 64MB 단위로 요청할 수 있다.
- GPUNum: 0 ~ MAX_GPU_NUM(한 개의 Node 가 갖는 최대 GPU 수)까지 요청할 수 있다.
- GPUMemorySize: 256MB 단위로 요청할 수 있다.
- GPUPercentage: GPU 사용율에 대한 백분율

2.5 마이크로 펑션 사전 최적화

일반적인 마이크로 펑션을 생성하는 절차에서 실행 환경 설정을 마이크로 펑션 생성자가 임의로 선택하여 등록한다. 그러나 설정한 자원 정보가 마이크로 펑션을 실행하기에 최적화된 자원 정보를 설정되지 못하는 경우가 매우 많다. 그래서 본 논문에서 제안하는 마이크로 펑션 사전 최적화기(Fn Env PreOptimizer)은 사용자가 마이크로 펑션을 등록할 때 지정한 자원 변수에 대하여 사전 성능테스트를 통하여 불필요한 자원 낭비가 최소화하도록 하는 역할을 한다. Fn Env PreOptimizer 의 실행 절차는 아래와 같다.

1. 마이크로 펑션 환경 조절기(Fn Env Controller)를 통하여 마이크로 펑션을 실행하기 전에 메모리 CPU, 메모리와 같은 실행 환경에 관련된 변수를 변경한다.
2. 마이크로 펑션 테스트기(Fn Tester)를 통해 마이크로 펑션을 실행한다.
3. 그 실행 결과는 마이크로 펑션 실행 분석기(Fn Exec Analyzer)를 통하여 분석한다.
4. 최적화 환경 변수 선택기(Fn Best Env Selector)를 통하여 환경 변수 변경에 따른 마이크로 펑션 실행 결과를 분석하는 과정을 중단할지를 판단하여 최종적인 최적화 환경 변수를 설정한다. 만약 최적화가 되지 않았다면 1 번으로 다시 시작한다.
5. 마이크로 펑션 저장기(Fn Storage Handler)를 통하여 마이크로 펑션 관련 파일을 저장한다.

(그림 5)는 개발자로부터 마이크로 펑션을 등록부터 최적화과정을 통하여 마이크로 펑션의 메타데이터 저장까지 처리는 모듈들의 구성을 보여주는 Fn PreEnv Optimizer 구성도이다.



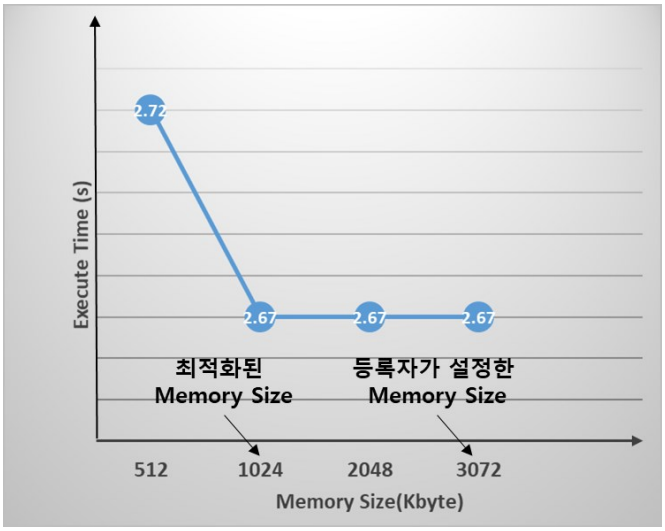
(그림 5) Fn PreEnv Optimizer 구성도

(그림 6)는 Fn PreEnv Optimizer 의 최적화 과정을 예시적으로 보여주기 위하여 특정 서버에서 성능 평가한 실험 결과를 보여준다. 성능 실험의 방향은 마이크로 펑션을 실행하는 환경 변수인 메모리 크기만을 변경하면서 마이크로 펑션 실행 시간을 측정한 결과를 보여주는 실험이다. 성능 실험을 위한 실행 환경은 아래 <표 1>과 같다.

<표 1> 성능실험을 위한 실행 환경

Processor	Intel(R) Xeon(R) CPU D-1557 @ 1.50GHz
Disk	256GB Samsung SSD 850
Memory	32GiB System Memory
Function	TensorFlow MNIST

각 단계를 보면 메모리가 3072Kbytes 일때 2.67s 가 소요되고 메모리가 2048kbytes, 1024kbytes 일 때도 모두 마이크로 평션 실행 시간이 2.67s 로 같은 값을 보여준다. 이 결과가 보여주는 것은 메모리를 3072Kbytes 로 설정하여 마이크로 평션을 실행하는 것은 자원의 낭비를 의미한다. 더 적은 메모리 1024kbytes 설정에서 실행해도 같은 결과를 보여주기 때문이다. 이처럼 마이크로 평션 등록자가 설정한 3072Kbytes 가 아닌 1024kbytes 를 선정함으로써 불필요한 자원 낭비를 제거하는 효과가 있다.



(그림 6) 메모리 크기 변경에 따른 마이크로 함수 실행 시간에 대한 실험 결과

3. 결론

본 논문은 효율적인 마이크로 평션 관리기를 제공하여 마이크로 평션 개발자가 마이크로 함수 생성/수정/삭제/유지/관리 측면에서 편리성 및 다양한 버전을 유지하도록 5 단계로 구분되는 라이프사이크 관리 기능도 제공하고 있다. 특히, 제안하는 마이크로 평션 사전 최적화기을 통하여 마이크로 평션의 실행 환경에서 낭비되는 자원을 효과적으로 최소화하였다.

참고문헌

[1] Paul Castro (2017.03.05) “ Serverless Programming” 2017 IEEE 37th international Conference on Distributed Computing Systems.

[2]김재욱 외, “오픈소스 서버리스 클라우드 컴퓨팅 프레임워크 비교 분석”, 한국정보과학회 학술발표 논문집, pp. 1448-1450, 2017.

[3] N. Alshuqayran, N. Ali and R. Evans, "A Systematic Mapping Study in Microservice Architecture," Proc.of the IEEE International Conference on Service-Oriented computing and Applications (SOCA), pp.44-51, 2016.

[4] S. Hassan, N. Ali and R. Bahsoon, "Microservice Ambients: An Architectural Meta-modelling Approach for Microservice Granularity," Proc. of the IEEE International Conference on Software Architecture (ICSA), pp. 1-10, 2017.

[5] F. Rademacher, S. Sachweh and A. Zundorf, "Differences Between Model-driven Development of Service-oriented and Microservice Architecture," Proc. of the IEEE International Conference on Software Architecture Workshops (ICSAW), pp. 38-45, 2017.

[6] “Aws lambda.” https://aws.amazon.com/lambda.

[7] “Google cloud function.” https://cloud.google.com/functions/docs/

[8] BALDINI, Ioana, et al. Serverless computing: Current trends and open problems. In: Research Advances in Cloud Computing. Springer, Singapore, pp. 1-20, 2017.

[9] Scott Hendrickson, et al. “Serverless Computation with OpenLambda”, HotCloud'16 Proceedings of the 8th USENIX Conference on Hot Topics in Cloud Computing, pp 33-39, 2016.

[10] Dr. Andreas Nauerz “ Open Lab Session 3400/9002 Event-Driven and Serverless Computing with IBM Bluemix OpenWhisk” InterConnect 2017