The Transactions of the Korea Information Processing Society
TKIPS Vol.13, No.2, pp.60~75

eISSN: 3022-7011
https://doi.org/10.3745/TKIPS.2024.13.2.60

Survey on the Performance Enhancement in Serverless Computing: Current and Future Directions

Eunyoung Lee†

ABSTRACT

The demand of users, who want to focus on the core functionality of their applications without having to manage complex virtual environments in the cloud environment, has created a new computing model called serverless computing. Within the serverless paradigm, resource provisioning and server administration tasks are delegated to cloud services, facilitating application development exclusively focused on program logic. Serverless computing has upgraded the utilization of cloud computing by reducing the burden on cloud service users, and it is expected to become the basic model of cloud computing in the future. A serverless platform is responsible for managing the cloud virtual environment on behalf of users, and it is also responsible for executing serverless functions that compose applications in the cloud environment. Considering the characteristics of serverless computing in which users are billed in proportion to the resources used, the efficiency of the serverless platform is a very important factor for both users and service providers. This paper aims to identify various factors that affect the performance of serverless computing and analyze the latest research trends related to it. Drawing upon the analysis, the future directions for serverless computing that address key challenges and opportunities in serverless computing are proposed.

Keywords : Serverless Computing, Cloud Computing, Edge Computing, Function-as-a-service(FaaS), Resource Management

성능 향상을 위한 서버리스 컴퓨팅 동향과 발전 방향

이 은 영†

요 약

클라우드 환경에서 복잡한 가상 환경을 관리할 필요 없이 애플리케이션 본연의 작업에 집중하기를 원하는 사용자의 요구는 서버리스 컴퓨팅이라는 새로운 컴퓨팅 모델을 탄생시켰다. 서버리스 컴퓨팅 모델에서 사용자는 서버에서의 자원 할당이나 기타 서버 관리를 서비스 제공자에게 위임하고, 자신은 애플리케이션 코드 개발에만 집중하여 클라우드 서비스를 사용하는 것이 가능해졌다. 서버리스 컴퓨팅은 클라우드 서비스 사용자의 부담을 감소시킴으로써 클라우드 컴퓨팅의 활용도를 한 단계 업그레이드시켰으며, 향후 클라우드 컴퓨팅의 기반 모델로 자리 잡을 것으로 예상된다. 서버리스 플랫폼은 사용자를 대신하여 클라우드 가상 환경에 대한 관리를 담당하며, 애플리케이션을 구성하는 서버리스 함수를 클라우드 환경에서 실행시키는 역할을 담당한다. 사용하는 자원에 비례하여 사용자 과금이 이루어지는 서버리스 컴퓨팅의 특징을 고려 할 때 서버리스 플랫폼의 효율성은 사용자와 서비스 제공자 모두에게 매우 중요한 요소라고 볼 수 있다. 본 논문에서는 서버리스 컴퓨팅 성능에 영향을 미치는 다양한 요소를 판별하고, 관련된 최신 연구 동향을 분석하고자 한다. 그리고 분석 결과를 바탕으로 향후 서버리스 컴퓨팅의 발전 방향과 관련된 연구 방향을 논의한다.

키워드 : 서버리스 컴퓨팅, 클라우드 컴퓨팅, 에지 컴퓨팅, 함수 기반 서비스, 자원 관리

1. 서 론

클라우드 컴퓨팅은 등장과 함께 산업계, 학계, 그리고 정부 기관으로부터 지대한 관심을 받으며 짧은 시간에 매우 빠른 성장세를 보이고 있다. 이와 같은 빠른 성장세는 IT 인프라와 관련 소프트웨어 산업에도 동시에 많은 영향을 미치고 있다. 클라우드 컴퓨팅은 사회의 다른 기본 유틸러티처럼 IT 자원을 소유가 아닌 구독 방식으로 활용하는 컴퓨팅 모델이다[1, 2]. 클라우드 컴퓨팅 모델의 등장으로 개발자는 IT 인프라에 대한 걱정 없이 보다 쉽게 소프트웨어나 서비스를 개발하고 운영할 수 있는 방법이 열리게 되었다.

※ 이 논문은 2021년도 동덕여자대학교 연구년 제도 지원에 의하여 수행된 연구 결과물임.
† 정 회 원 : 동덕여자대학교 컴퓨터학과 교수
Manuscript Received : November 30, 2023
First Revision : December 26, 2023
Accepted : December 27, 2023
* Corresponding Author : Eunyoung Lee(elee@dongduk.ac.kr)

클라우드 컴퓨팅은 실물 하드웨어에 대한 가상화 기술을 바탕으로 사용자에게 가상화된 하드웨어 환경과 플랫폼, 소프트웨어 등의 서비스를 제공하는 구조를 가지고 있다. 클라우드 컴퓨팅 초기 단계에서 가상화 방식은 크게 2가지 방향으로 진행되었다. 첫 번째는 아마존 EC2로 대표되는 하드웨어 가상화였고[3], 나머지 하나는 구글 앱엔진(Google App Engine)을 필두로 하는 애플리케이션 특화된 플랫폼의 활용이었다.[4]. 2009년 당시 시장에서는 하드웨어 레벨의 가상 머신을 주축으로 하는 아마존 방식이 크게 인기를 얻었으며, 이에 따라 구글, 마이크로소프트 등의 클라우드 서비스도 아마존과 비슷한 방식의 인터페이스를 제공하게 되었다. 하드웨어 가상 머신에 중점을 둔 클라우드 서비스가 성공하게 된 이유에는 초기 클라우드 컴퓨팅 환경에서 사용자들이 이제까지 사용해 왔던 환경과 비슷한 컴퓨팅 환경을 클라우드에서도 그대로 유지하고 싶었던 욕구가 크게 작용했던 것으로 보인다[5, 6].

기존의 컴퓨팅 환경과 최대한 유사한 환경을 제공하는 방식은 클라우드 환경에 대한 사용자의 초기 장벽을 낮추는 역할을 했지만, 동시에 사용자는 가상 머신의 환경 설정에 대한 부담을 지게 되었다[7, 8]. 클라우드 컴퓨팅을 사용하기 위해서 사용자는 시스템 관리자가 되거나, 혹은 시스템 관리자의 도움을 받아 자신이 사용하는 가상 머신의 하드웨어 설정을 직접 관리할 필요가 있는데, 이는 많은 사용자들에게 적지 않은 부담으로 작용하였다. 결과적으로 클라우드 컴퓨팅 모델은 물리적 인프라를 관리하는 부담은 감소시켰지만 대신 가상 자원관리에 대한 어려움을 증가시켰다.

클라우드 환경에서 복잡한 가상 머신의 관리에서 벗어서 애플리케이션 본연의 작업에 집중하기를 원하는 사용자의 요구는 2010년대 중반에 서버리스 컴퓨팅의 탄생으로 이어졌다[9, 10]. 서버리스라는 이름은 서버에서의 자원 할당이나 서버 관리는 클라우드 서비스 제공자가 책임을 지고, 사용자는 자신의 애플리케이션 코드에만 집중하여 클라우드 환경을 사용할 수 있다는 것을 강조하기 위한 명칭이다.

서버리스 플랫폼은 사용자를 대신하여 클라우드 가상 환경에 대한 관리를 담당하며, 사용자 애플리케이션을 구성하는서버리스 함수를 클라우드 환경에서 실행시키는 역할을 담당한다[11]. 사용하는 자원에 비례하여 사용자 과금이 이루어지는 서버리스 컴퓨팅 아키텍처의 특징을 고려할 때 서버리스 플랫폼의 효율성은 사용자와 서비스 제공자 모두에게 매우 중요한 요소라고 볼 수 있다.

본 논문에서는 서버리스 컴퓨팅 시스템의 성능을 향상시키기 위해서 고려해야 할 요소들을 판별하고 각 요소별 최신 연구 동향을 분석한다. 그리고 분석 결과를 바탕으로 향후 서버리스 컴퓨팅 연구의 발전 방향을 살펴보고자 한다. 2장에서는 서버리스 컴퓨팅 모델의 개념을 설명하고 기존의 클라우드 시스템과의 비교를 통해서 서버리스 컴퓨팅의 특징을 제시한다. 3장에서는 서버리스 컴퓨팅의 성능 향상을위해 고려해야 할 분야별 기술요소 중에서 서버리스 워크플로우, 자원 관리, 마이그레이션 기법을 중심으로 요소별 연구 동향과 연구 주제들을 분석한다. 4장에서는 최근 관심을 끌고 있는 서버리스 에지 컴퓨팅과 상태유지 서버리스 컴퓨팅 분야를 중심으로 서버리스 컴퓨팅의 향후 발전 방향을 분석한다. 또한 성숙 단계로 접어들고 있는 서버리스 컴퓨팅을 보다 편리하고 안전하게 활용하기 위해서 필요한 애플리케이션 개발 환경과 보안 관련 이슈에 대해서도 논의한다. 마지막 5장에서는 결론과 향후 연구 방향에 대한 전망을 기술하고자 한다.

2. 서버리스 컴퓨팅의 개요와 특징

사용자가 모든 자원을 자신의 서버에서 직접 관리하고 이를 활용하여 원하는 연산과 서비스를 제공하는 방식은 클라우드 서비스가 등장하기 전까지 가장 많이 사용되었던 방식이다. 서버를 직접 소유하고 관리하는 방식은 높은 가용성을 제공하지만 동시에 피크 타임을 제외한 시간에 낭비되는 자원과 직접적인 자원 관리가 쉽지 않다는 점이 가장 큰 문제점으로 지적되었다.

2000년대 초반에 제안된 클라우드 컴퓨팅은 인프라 서비스(Iaas, Infrastructure as a Service), 플랫폼서비스(PaaS, Platform as a Service), 소프트웨어 서비스(SaaS, Software as a Service)로 대표되는 3가지 서비스 모델로부터 시작되었다. 사용자는 자신이 원하는 서비스를 클라우드 서비스 제공자로부터 구입하고 이에 대한 사용 비용을 지불하는 형태로 클라우드 서비스를 이용할 수 있다.

Fig. 1은 각 클라우드 서비스별로 사용자가 관리하는 자원과 서비스 제공자가 관리하는 자원이 어떻게 구분되고 있는지를 보여주고 있다[12]. 하나의 애플리케이션이 제대로 동작하기 위해서 관리되어야 하는 자원은 물리적 하드웨어부터 운영체제, 런타임 시스템, 데이터, 사용자 애플리케이션까지 다수의 레이어로 구분될 수 있는데, 그림에서 각 레이어별 관리 주체는 서로 다른 색상으로 표시되어 있다. 옅은 음영은 사용자가 직접 관리하는 레이어를, 짙은 음영은 서비스 제공자가 관리의 주체가 되는 레이어를 나타낸다.

Fig. 1의 가장 왼쪽 그림은 애플리케이션 개발자가 모든 자원을 구입하고 직접 관리하는 고전적인 서버 중심의 자원 관리 방식을, 나머지 그림들은 클라우드 서비스의 관리 방식을 보여주고 있는데, 클라우드 서비스 모델별로 사용자가 관리해야 하는 레이어의 개수에 차이가 있음을 알 수 있다. 그림을 통해서 클라우드 컴퓨팅의 3가지 서비스 모델은 가상화 등의 기법을 이용하여 사용자가 직접 관리했던 서비스 레이어들을 클라우드 서비스 제공자가 제공하는 모델임을 확인할 수 있다. 그리고 사용자가 하드웨어나 소프트웨어 여부에 구애받지 않고 사용량에 비례하여 비용을 지불하는 클라우드 컴퓨팅 과금 체계는 전기, 가스와 같은 유틸러티의 개념이 컴퓨팅 자원에 적용되었다고 볼 수 있다.

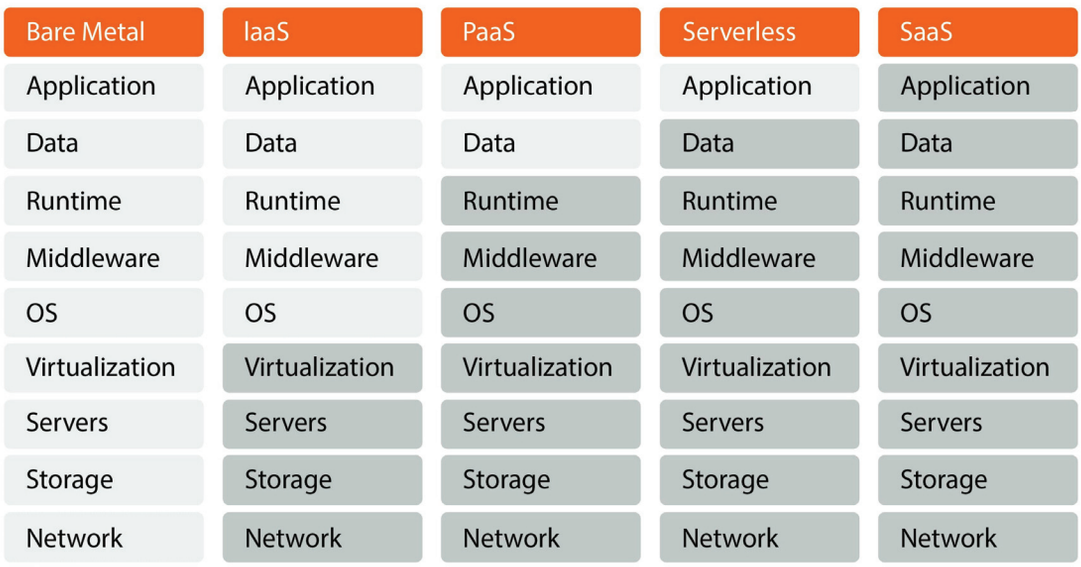

Fig. 1. Evolution of Cloud Service Models[12]

2.1 서버리스 컴퓨팅 아키텍처

서버리스 컴퓨팅은 서비스 이용자가 자신의 애플리케이션 로직을 무상태 함수(stateless function)의 형태로 정의하고 수행시킬 수 있는, 이벤트에 기반을 둔 컴퓨팅 모델이다[2, 10]. 서버리스 플랫폼은 무상태 함수들을 실행시킬 수 있는 일련의 이벤트 집합을 미리 정의해 두고 있다. 사용자는 해당 이벤트가 발생했을 때 실행되는 함수를 직접 정의하거나 서버리스 플랫폼이 제공하는 함수를 지정하는 방식으로 애플리케이션을 구성한다. 함수 호출의 시발점이 되는 이벤트는 사용자 인터페이스를 통한 HTTP 요청, 데이터베이스나 객체 스토리지에 대한 접근, IoT 디바이스에서 전달되는 알림 등이 해당된다.

Fig. 2는 서버리스 컴퓨팅 아키텍처와 기본적인 실행 흐름을 그림으로 설명하고 있다[13]. 이벤트가 발생하면 API 게이트웨이는 사용자가 미리 정의한 규칙에 의해서 관련 함수들을 실행시킨다. 서버리스 플랫폼 내부의 스케줄러는 함수 실행에 적합한 자원을 가진 워킹 노드(working node)를 찾고, 여기에 실행 명령을 전달한다. 워킹 노드 내부에는 각 함수별로 격리된 실행 환경이 구성되는데, 실행에 필요한 자원을 갖춘 환경을 구성하는 데는 클라우드 컴퓨팅에서 사용되는 컨테이너(container)가 기본 단위로 활용된다. 실행 환경이 준비되면 사용자가 요청한 함수가 실행되고, 함수 실행이 끝나면 결과는 사용자에게 전달된다. 함수 종료 후 실행 환경에 사용된 자원은 회수된다. 중간 과정에서 생성된 데이터와 상태 관리 정보 등은 필요한 경우 함수 종료 이후에도 사용자가 사용할 수 있도록 외부 스토리지 서비스에 저장된다.

함수 실행 과정에서 자원이 부족해지는 경우, 서버리스 플랫폼은 함수에 할당된 자원에 대한 스케일링을 수행한다. 스케일링 여부는 모니터링 시스템에 수집된 상태 데이터와 스케일링 결정 알고리즘에 따라 결정된다. 자원의 관리와 분배는 서버리스 컴퓨팅 분야에서 가장 활발한 연구가 진행되고 있는 분야로 많은 연구자들의 노력으로 현재까지 상당한 성과를 보이고 있으며, 앞으로도 꾸준한 연구가 진행될 것으로 예상되고 있다.

일반적으로 서버리스 컴퓨팅 서비스는 함수 기반 서비스와 백엔드 기반 서비스의 융합으로 정의된다[11, 14]. 함수 기반 서비스(FaaS, Function as a Service) 파트는 사용자가 원하는 애플리케이션의 기능을 개발하고, 애플리케이션 실행과 관리를 수행할 수 있는 부분을 지칭한다. 사용자는 함수 기반 서비스를 이용하면서 하부 인프라 구성이나 유지에 대해서 신경 쓸 필요 없이 애플리케이션 로직에만 집중하여 개발이 가능하다. 서버 리스 컴퓨팅의 나머지 파트는 백엔드 기반 서비스(BaaS, Backend as as Service) 파트로 통상적으로 클라우드 시스템에 위임하는 특정한 기능을 서비스 제공자가 온라인 서비스로 제공하는 부분이다. 대표적인 백엔드 기반 서비스는 인증이나 알림 서비스 등이다. 함수 기반 서비스는 전적으로 사용자가 정의한 함수를 실행시키는데 사용되고, 백엔드 기반 서비스는 서버리스 서비스 제공자가 사전에 정의된 기능을 온라인 형태로 제공한다.

사용자는 함수 기반 서비스와 백엔드 기반 서비스 중 어떤 것을 이용하든지 자원 관리에 대해 신경 쓸 필요가 없다. 다시 말해서, 서버리스 컴퓨팅은 기존의 서버형 클라우드 컴퓨팅이 가지는 가상환경 설정을 블랙박스화하여 사용자가 애플리케이션 특화된 서비스 개발과 구현에 좀 더 집중할 수 있도록 설계된 클라우드 컴퓨팅 모델이라고 말할 수 있다. Fig. 1를 보면 함수 기반 서비스(FaaS)를 나타내는 서버리스 컴퓨팅은 가상화 정도에서 플랫폼 서비스(PaaS)와 소프트웨어 서비스(SaaS) 사이에 위치하는 것을 알 수 있다.

최초의 상용 서버리스 서비스는 2014년 11월에 아마존이

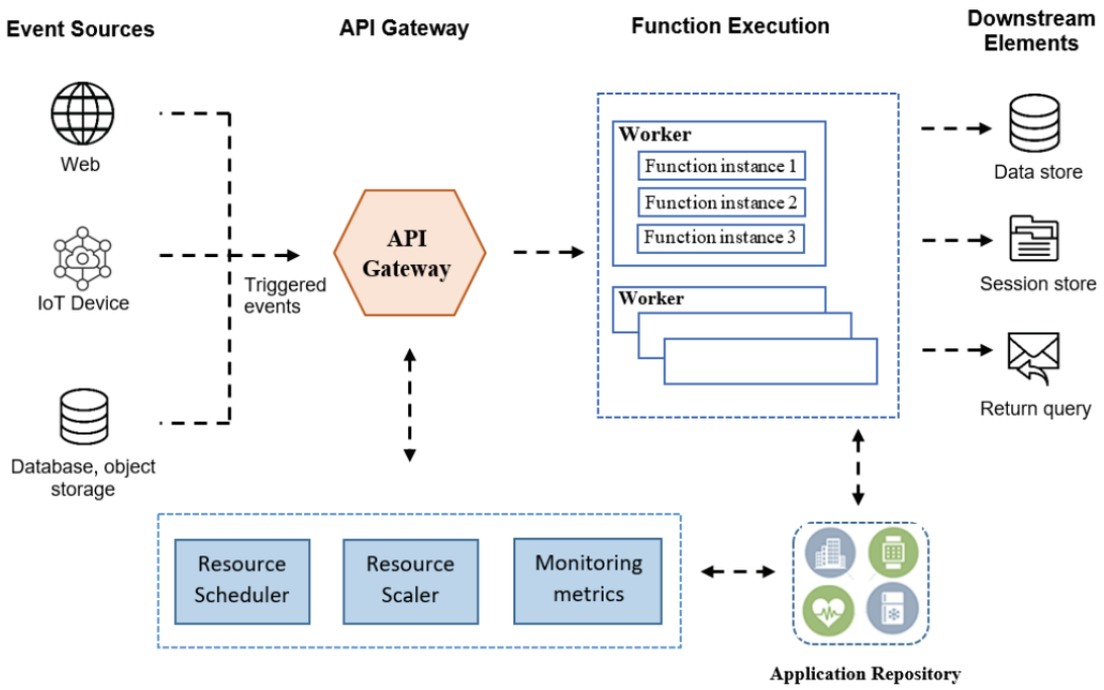

Fig. 2. Serverless Computing Architecture[13]

선보인 아마존 람다 서비스이다[15]. 대표적인 상용 서버리스 컴퓨팅 서비스로는 아마존의 람다 서비스(Lambda Service)[15], 구글의 클라우드 함수 (Cloud Functions)[16], 마이크로소프트의 애저 함수(Azure Functions)[17], 아파치 오픈위스크(Apache OpenWhisk)[18] 등이 있다.

2.2 서버리스 함수

서버리스 컴퓨팅 모델은 서버리스 서비스 제공자가 애플케이션 자원 관리의 모든 측면을 책임지는 프로그래밍 모델로, 사용자는 복잡한 컴퓨팅 인프라 관리에서 벗어나 자신이 개발하는 애플리케이션에만 집중할 수 있다는 장점을 가지고 있다. 일단 애플리케이션이 서버리스 플랫폼에 등록되면 서비스 제공자는 초기 자원 할당부터 스케줄링, 실행 모니터링과 자원 스케일링에 이르는 일련의 과정을 모두 책임지게 된다.

서버리스 플랫폼에서 실행되는 애플리케이션은 서버리스 함수로 구성된다. 서버리스 함수는 애플리케이션 로직을 포함하고 있으며, 하나의 애플리케이션은 1개 혹은 서로 연관성을 가지는 다수의 무상태 함수로 구성된다. 이런 의미에서 서버리스 함수는 서버리스 컴퓨팅의 근간을 이루는 단위가 된다. 많은 연구자들은 클라우드를 위한 범용 프로그래밍 모델에서 서버리스 함수가 추상화의 기본 단위가 될 것으로 예상하고 있다. 서버리스 플랫폼에서 사용자는 자신의 원하는 기능을 서버리스 함수의 형태로 작성하고, 해당 함수의 실행을 촉발시키는 이벤트를 선택하는 과정을 통하여 애플리케이션을 개발한다. 이 때 서버리스 함수는 서비스 플랫폼이 지원하는 하이 레벨 프로그래밍 언어(high-level programming language)를 이용하여 작성된다. 이외에 클라우드 컴퓨팅에서 필요한 작업들은 서버리스 플랫폼이 모두 처리하기 때문에 사용자는 실제 애플리케이션과 관련된 코드 외에 추가적인 환경 설정에 대해서는 신경 쓸 필요가 없다. 서버리스 플랫폼이 수행하는 작업에는 인스턴트 선택, 스케일링, 배포, 결함 포용, 모니터링, 로깅, 보안 패치 등이 포함되어 있다.

2.3 서버리스 컴퓨팅의 특징

초보 클라우드 사용자들은 클라우드 인프라 구조에 대한 고민 없이 애플리케이션을 쉽게 개발하고 배포할 수 있다는 점에서, 그리고 클라우드 전문가들은 애플리케이션 특화된 문제를 해결하는데 집중함으로써 개발 시간을 획기적으로 단축할 수 있다는 점에서 서버리스 컴퓨팅을 선호하고 있다. 서버리스 컴퓨팅을 지원하는 플랫폼이나 서비스는 크게 다음 2가지 측면에서 기존 클라우드 서비스와 차별점을 가진다. 우선 서버리스 플랫폼은 사용자가 명시적으로 자원을 할당할 필요가 없도록 자동 스케일링 기능을 제공해야 한다. 또한 사용자에게 할당된 자원이 아니라 실제 코드 수행에 사용된 자원에 대해서만 요금을 부과하는 과금 체계를 가지고 있어야 한다.

개발의 용이성 측면에서 기존 서버형 클라우드 컴퓨팅은 로우 레벨 프로그래밍 언어(low-level programming language)인 어셈블리어를 사용하는 프로그래밍에, 서버리스 클라우드 컴퓨팅은 하이 레벨 프로그래밍 언어(high-level programming language)를 활용한 프로그래밍에 비유할 수 있다[19]. 어셈블리어 프로그래머는 $c = a + b$와 같은 간단한 수식을 계산하기 위해서도 사용할 레지스터를 선택해서 레지스터에

원하는 값을 로드하고 덧셈을 수행한 후에 결과 값을 다시 저장하는 복잡한 과정을 거쳐야만 한다. 마찬가지로 서버형 클라우드 컴퓨팅에서는 아무리 간단한 연산이라도 우선 가용한 자원을 먼저 확보하고, 필요한 코드와 데이터를 해당 클라우드 환경에 업로드하는 과정을 거쳐야만 한다. 원하는 연산을 수행한 후에는 결과 값을 저장하는 작업뿐만 아니라 사용한 자원의 해지까지 모두 클라우드 사용자의 책임이 된다. 클라우드를 활용하는 방식이 서버형 컴퓨팅 모델에서 서버리스 컴퓨팅 모델로 변화하는 모습은 과거 어셈블리어 프로그래밍이 하이 레벨 프로그래밍 언어를 이용한 프로그래밍으로 발전했던 것과 같은 자연스러운 흐름이라 할 수 있다.

서버리스 컴퓨팅 모델은 기존의 클라우드 컴퓨팅 모델과 비교했을 때, 다음과 같은 장점 및 특징을 가지고 있다[11].

- 실행 환경에 대한 복잡한 내용을 감춰서 서비스 사용자가 가지는 플랫폼 인프라에 대한 관리 부담을 줄인다.
- 서비스 제공자는 사용자 요구에 부합하는 서비스 제공을 위해 오토 스케일링 기능을 제공한다.
- 사용한 자원의 사용량을 기준으로 하는 과금제도(pay-as-you-go model)를 제공한다.
- 서비스 제공자는 사용자 요청을 정해진 시간 내에 처리하기 위해서 최선의 노력을 한다.
- 서버리스 컴퓨팅의 기본 단위는 함수가 되며, 함수의 내용은 라이브러리와의 연관성, 실행 환경의 결정 등을 위해서 서비스 제공자에게 공개된다.

서버리스 컴퓨팅은 자원 할당 및 관리에 대해서 블랙박스 접근 방식을 제공하지만, 서버리스 플랫폼의 성능을 높이기 위해서 사용자는 자신이 개발한 애플리케이션의 워크플로우 의존 그래프(workflow dependency graphs)나 대략적인 자원 요구 사항을 서버리스 서비스 제공자에게 제공하기도 한다. 서버리스 서비스 제공자는 사용자가 제공한 정보를 서버리스 함수 스케줄링에 활용하여 플랫폼의 효율성을 높이기도 한다[19]. 서버리스 플랫폼의 성능 향상을 위한 스케줄링 기법에 대해서는 3.2절에서 자세히 다룬다. 서버리스 컴퓨팅 모델은 서로 다른 인프라 도메인을 포함하는 분산 자원을 활용하는 경우에 강점을 가지는 방식이다. 서버리스 컴퓨팅 서비스는 주로 퍼블릭 혹은 프라이빗 클라우드를 기반으로 서비스가 제공되고 있으며[15, 16, 18], 최근에는 에지 컴퓨팅이나 포그 컴퓨팅 아키텍처까지 그 범위가 확대되고 있다.

3. 성능향상 이슈

서버리스 컴퓨팅은 사용자들에게 많은 인기를 얻고 사용 영역이 확대되고 있으며, 동시에 다양한 분야에서 성능 향상을 위한 연구가 진행되고 있다. 서버리스 플랫폼 성능에 영향을 미치는 요인은 매우 복잡하고 다양한데, 그 중에서 본 논문에서는 서버리스 워크플로우와 자원 관리, 스케줄링, 마이그레이션 기법을 중심으로 성능 향상 이슈에 대해서 논의하고자 한다. 본 절에서는 제시된 기술 요소들을 보다 자세히 분석하고, 각 요소별 최신 연구 동향과 앞으로 추가적인 연구가 필요할 것으로 예상되는 연구 방향을 제시한다.

3.1 서버리스 워크플로우

서버리스 애플리케이션은 개별적으로 독립된 기능을 수행하는 다수의 함수를 다양한 방식으로 조합해서 완성된다. 독립적인 기능을 가지는 함수들을 원하는 워크플로우의 형태로 조합하는 과정을 서버리스 컴퓨팅에서는 오케스트레이션(orchestration)이라고 부른다. 단일 함수로 서버리스 애플리케이션을 구성하는 경우는 거의 없기 때문에 유사한 기능을 수행하는 서버리스 애플리케이션이라도 개발자의 취향과 의도에 따라 다양한 종류의 워크플로우가 존재하게 된다. 다시 말해서 개발자가 어떻게 오케스트레이션을 수행하느냐에 따라 서로 다른 워크플로우가 생성되는 것이다.

서버리스 서비스 제공자의 입장에서 보면 애플리케이션의 함수 호출 체인에 대한 정보가 많을수록 서버리스 플랫폼의 성능을 향상시키는 것이 용이해진다. 워크플로우에 대한 정보를 활용하여 서비스 플랫폼은 필요한 함수를 미리 준비하거나 최적화하여 애플리케이션 전체의 성능을 향상시키고 나아가 사용자의 비용 부담을 줄일 수 있다.

서비스 플랫폼은 개별 애플리케이션의 호출 체인에 대한 정보를 방향성이 있는 작업 그래프(directed task graph)의 형태로 제공받을 수 있다. 작업 그래프 내부에서 각 노드는 함수를, 간선은 함수 사이의 의존성이나 실행 순서를 표시하게 된다. 서비스 제공자는 작업 그래프를 분석하여 그래프 내부의 사이클이나 자체 반복(self-loop) 혹은 조건 분기 등의 특징을 분석해 내고 이를 바탕으로 병렬 실행 등 효율성을 높이기 위한 스케줄링 결정을 내린다. Lin[20]은 작업 그래프를 활용해서 시스템 전반의 성능을 향상시킬 수 있는 방법에 대해서 논의하였다. WiseFuse[21]도 작업 그래프를 실행 순서 최적화에 이용하는 시스템으로, 최적화 과정에서 사용자가 정의하는 지연 제한과 비용 제한을 동시에 고려하고 있다.

서비스 제공자는 크게 2가지 방식으로 서버리스 애플리케이션의 워크플로우 정보를 획득한다. 가장 일반적인 방식은 서비스 제공자가 미리 정의해 놓은 인터페이스를 활용하여 애플리케이션 개발자가 자신의 애플리케이션 워크플로우 정보를 제공하는 방식으로, 서비스 플랫폼은 사용자가 제공한 워크플로우를 분석하여 성능 향상을 위한 결정을 내리게 된다. 이 방식은 현재 대부분의 서버리스 플랫폼에서 사용되고 있는데, 아마존 스텝함수(Amazon Step Functions)[22], 애저 지속성 함수(Azure Durable Functions)[23], 구글 클라우드 컴포저(Google Cloud Composer)[24] 등이 사용자가 워크플로우 정보를 제공할 수 있는 대표적인 인터페이스들이다.

사용자가 자신의 워크플로우를 설명하기 위하여 사용하는 방식도 서버리스 플랫폼에 따라 다양하게 존재한다. 아마존

은 사용자가 상태 머신과 상태 전환함수를 기술하는 상태 언어(State Language)라는 이름의 언어를 제공한다[25]. 구글 클라우드 컴포저에서는 파이썬을 활용하여 사용자 워크플로우를 기술할 수 있다[24]. 마이크로소프트 Netherite[26]는 애저 지속성 함수를 활용하여 서버리스 워크플로우를 기술하고 애플리케이션을 실행시킬 수 있는 서버리스 아키텍처이다. Netherite는 가장 다양한 형태의 워크플로우 기술 언어를 지원하고 있는데, 사용자는 기존의 프로그래밍 언어(자바스크립트, 파이썬, C# 등)을 활용하여 워크플로우를 기술할 수 있다[27].

사용자가 워크플로우를 기술할 수 있는 방식이 서비스 제공자 별로 상이하다는 사실은 워크플로우 기능을 활용하는 사용자에게 또 다른 부담으로 작용할 수 있다. 따라서 서버리스 플랫폼에 구애받지 않고 워크플로우를 기술할 수 있는 플랫폼 독립적인 언어에 대한 요구가 증가하고 있으며, 이에 관련된 연구들도 꾸준하게 진행되고 있다. AFCL[28]은 서버리스 워크플로우를 기술할 수 있는 범용 언어로 기술된 워크플로우를 상이한 서버리스 플랫폼에 알맞게 변형하는 기능을 제공한다. Tiggerflow[29] 는 서버리스 시스템에서 이벤트에 기반을 둔 워크플로우를 구성하고 기술하는 방식을 제안하고 있다.

Burckhardt[27]와 Wen[30]은 서버리스 애플리케이션이 가지는 워크플로우를 분석하여 도메인별 애플리케이션의 특징을 도출하는 연구를 진행하였고, John[31]은 도메인의 특성을 반영한 워크플로우 구성을 지원해서 도메인 특화 애플리케이션 개발을 용이하게 하는 프레임워크를 제안하였다.

범용 언어나 도메인 특성을 반영된 워크플로우 패턴이 제공되더라도 개발자가 자신이 개발하는 서버리스 애플리케이션의 워크플로우를 직접 기술해야만 하는 상황은 서버리스 애플리케이션 개발자에게는 여전히 부담스러울 수밖에 없다. 그러므로 이미 개발된 서버리스 애플리케이션의 워크플로우를 분석해 주는 도구나 오케스트레이션 단계에서 개발자가 참고할 수 있도록 워크플로우의 속성을 분석할 수 있는 도구에 대한 연구가 보다 많이 필요한 상황이다. 이들은 향후 연구 분야로 충분한 가치가 있다고 판단된다.

3.2 자원관리와 스케줄링

자원 관리는 사용자가 요청한 서비스의 품질(QoS, Quality of Service)을 만족시킬 수 있도록 실행되는 애플리케이션에 적정한 양의 자원을 할당하고, 할당된 자원을 적절하게 관리하는 과정을 의미한다. 사용자는 서버리스 서비스 제공자에게 자신이 작성한 서버리스 함수나 서버리스 백엔드 함수에 대한 실행 요청(invocation requests)를 보내고, 서비스 제공자는 일정한 데드라인 안에 사용자 요청을 처리해야 한다. 따라서 요청 내용에 따라 어떤 연산 노드에서 언제 해당 함수를 실행시킬지 결정해야 하는데, 이 과정에서 서버리스 서비스 제공자는 서버리스 플랫폼의 전체적인 에너지 소비량, 자원 사용량 등을 모두 고려하여 연산에 참여할 노드를 결정하고 실행 순서를 결정해야 한다[13].

이와 같은 일련의 결정 과정을 스케줄링(scheduling)이라고 하며, 현재 서버리스 컴퓨팅 분야에서 가장 활발한 연구가 진행되고 있는 분야이다. 사용자 혹은 서버리스 플랫폼 관리자는 애플리케이션의 특성을 반영한 자원 관리와 스케줄링 기법을 선택함으로써 서버리스 시스템의 성능을 효과적으로 향상시킬 수 있다. 자원관리와 스케줄링 기법은 어떤 요소에 중점을 두고 자원을 배분하고 스케줄링을 진행하는지에 따라 다양한 기법들이 제안되고 있으며, 본 절에서는 스케줄링에 영향을 주는 요소별로 스케줄링 기법을 분류하여 분석하고자 한다.

1) 에너지 효율

에너지 사용량을 최소화하기 위해서 서버리스 플랫폼은 스케줄링에 사용되지 않는 컨테이너를 동면 모드(hibernate mode), 혹은 콜드 상태 모드(cold-state mode)에 두는 방법을 사용한다.

이 방법은 에너지 사용량을 줄일 수 있지만 컨테이너를 콜드 상태에서 활동 상태로 변경하는 과정에서 생기는 지연 시간 때문에 사용자가 지정한 데드라인을 맞추지 못할 위험성이 있다. 따라서 콜드 상태 모드를 사용하더라도 워밍업 지연을 최소화하는 방향으로 스케줄링이 진행되어야 한다. Ensure[32]는 지연 시간을 최소화하기 위해서 몇 개의 컨테이너는 사용하지 않더라도 활동 상태로 유지하는 기법을 제안하였다. Fifer[33]도 콜드 상태를 최대한 피하기 위해서 비슷한 방법을 채택하고 있다.

그렇지만 필요 이상의 컨테이너를 활동 상태로 유지하는 것은 결과적으로 에너지와 자원의 낭비를 가져오게 된다. 지연 시간 최소화를 위해서 활동 상태로 유지하는 컨테이너 개수를 충분히 유지하면서도 이로 인해서 추가적으로 사용되는 잉여 자원을 최소화하는 것이 필수적이다. Ensure[32]도 논문에서 자원의 총량을 최소화하기 위한 이론적 모델을 제시하였고, Roy[34]는 비교적 가격이 저렴한 이종 노드(heterogeneous nodes) 활용으로 워밍업에 드는 비용을 최소화하는 기법을 제안하였다. 에너지 효율과 비용을 고려한 스케줄링에 대한 연구는 앞으로 더욱 활발하게 진행되어야 할 것으로 예상된다.

2) 자원사용 패턴

서버리스 애플리케이션의 경우 애플리케이션 도메인에 따라 자원 사용량이 일정한 패턴을 보이는 경우가 있다. 서버리스 함수들이 경쟁을 벌이는 자원에는 CPU 뿐만 아니라 메모리, 디스크, 혹은 네트워크자원까지 포함된다. 예를 들어, 산술연산이 많이 필요한 애플리케이션의 경우는 CPU 사용량이 많은 반면 주어진 데이터에 대한 분석을 하는 애플리케이션의 경우는 메모리 사용량이 많은 경향이 있다. 이 경우 같은 자원

사용 패턴을 가진 함수들을 하나의 물리적 노드에 동시에 할당하게 되면 함수들 사이에서 자원에 대한 경쟁이 벌어지고 전체적인 성능이 낮아질 수밖에 없다. 따라서 주어진 물리적 노드의 자원을 균형 있게 사용하고 비슷한 자원사용 패턴을 보이는 함수들 사이의 경쟁을 최소화하는 방향으로 자원을 할당하는 스케줄링 방식은 서버리스 서비스의 성능을 향상시키는데 많은 도움이 된다.

FnSched[35]는 서버리스 서비스에서 자원과 성능 사이의 관계에 비교적 일찍 주목한 연구로 볼 수 있다. 해당 연구에서는 서버리스 함수들을 CPU 사용량에 따라 분류하고, 함수들 사이의 경쟁을 최소화할 수 있는 방향으로 함수들을 물리적 노드에 분배하는 방식을 제안하였다. Fifer[33]에서는 함수의 실행 시간을 예측하기 위하여 오프라인 프로파일링 을 활용하는 방식을 채택하였다. Akhtar[36]는 함수를 실행하기 전에 실행 시간을 예측하고 이를 통계적 기법을 사용하여 물리적 노드에 스케줄링하는 방식으로 자원의 배분하는 방식을 제안하였다. Roy[34]도 함수의 실행 여부를 동적으로 예측하고 함수를 실행할 노드를 미리 스케줄링해서 시스템의 효율을 높이는 방법을 제안하였다. Hoseinyfarahabady도 애플리케이션의 실행 시간을 예측하고 서비스 품질(QoS)을 만족시키기 위한 예측 모델을 제안하였다[37-39].

앞으로 더욱 다양한 도메인의 애플리케이션들이 서버리스 컴퓨팅 모델을 사용할 것으로 예상된다. 자원의 사용패턴은 애플리케이션의 특성을 반영할 수밖에 없기 때문에 향후 자원 사용 패턴을 활용한 스케줄링 기법에 대한 연구는 도메인별로 꾸준하게 지속될 것으로 예상된다.

3) 워크플로우

서버리스 애플리케이션은 서로 독립적인 무상태 함수들로 이루어지고, 실제 실행은 구성 함수들에 대한 일련의 호출로 이루어진다. 이와 같은 일련의 함수 호출은 워크플로우라 불리며, 서버리스 애플리케이션의 워크플로우와 오케스트레이션 작업의 완성도가 이를 수행하는 서버리스 플랫폼의 성능에 크게 영향을 미친다는 사실과 관련 연구 동향은 이미 3.1절에서 자세히 살펴보았다.

워크플로우 정보는 자원 관리와 스케줄링 성능 향상에도 역시 효과적으로 활용될 수 있다. 스케줄러는 함수 호출 순서에 대한 정보를 활용하여 보다 효율적인 스케줄링을 수행하고 결과적으로는 전체적인 서버리스 시스템의 성능을 향상시킬 수 있다. 다음에 실행될 함수를 미리 알거나 혹은 분기를 예측할 수 있는 경우 스케줄러는 함수 실행에 적당한 물리적 노드를 미리 찾아내고 노드 내부에서 필요한 자원을 미리 할당할 수 있다. 사전에 함수 실행에 필요한 자원을 미리 확보해 두면 콜드 상태 해지에 소요되는 지연 시간도 줄일 수 있다.

워크플로우 정보를 활용한 스케줄링 기법에 대한 연구도 역시 활발히 진행되고 있다. Xanadu[40]에서는 워밍업 지연 시간을 단축하기 위해서 워크플로우 정보를 구조를 활용하였다. Archipelago[41]는 사용될 함수풀(function pool)을 예측하기 위해서 DAG(directed acyclic graph) 구조를 사용한 시도를 보여주고 있다. Sequoia 프레임워크[42]에서는 서비스 품질(QoS)을 고려한 스케줄러 설계를 위하여 함수 호출체인 정보를 활용하였다. 서버리스 플랫폼에서 확률론과 DAG를 이용하여 실행 체인을 예측하고 스케줄링에 활용하는 기법은 향후 더욱 활발한 연구가 진행될 것으로 예상된다.

4) 데이터 흐름

이상적인 서버리스 컴퓨팅 환경에서 서버리스 애플리케이션은 무상태 함수로만 이루어지며, 외부 데이터 소스에 전혀 의존하지 않는다. 그렇지만 실제 상황에서 외부 데이터를 완전히 배제한 애플리케이션을 구성하는 것은 거의 불가능하다고 할 수 있다. 대표적으로 최근에 많은 관심을 받고 있는 기계 학습 애플리케이션은 외부에 저장된 데이터에 대한 의존도가 매우 높은 소프트웨어로 분류된다.

외부 데이터에 대한 의존도가 높고 외부 데이터가 중요한 역할을 차지하는 애플리케이션에 대해서는 데이터 흐름을 고려한 스케줄링인 진행되어야 하며, 이와 관련된 연구들도 활발히 진행되고 있다.

Freshen[43]에서는 애플리케이션 개발자 혹은 서비스 제공자가 런타임 재사용 기능들과 함께 필요한 데이터를 능동적으로 미리 가져오는 것을 허용하는 방식으로 서버리스 함수를 실행할 때 발생하는 데이터 오버헤드를 줄이는 방식을 제안하였다. Cloudburst[44]에서도 데이터지역성(data locality)에 우선순위를 두어 스케줄링을 수행하여 데이터 오버헤드를 줄이는 방식을 채택하고 있다. Rausch[45]는 에지 컴퓨팅 스케줄링에서 데이터 흐름과 도메인 애플리케이션의 특징을 모두 고려하는 시도를 보여주고 있다. Kaffes[46]도 데이터 지역성을 고려하여 함수 실행 시작 전에 대기 시간을 줄이는 기법을 아파치 오픈위스크를 활용하여 구현하고 성능을 실험하였다.

5) 패키징

최근의 서버리스 기법에서 패키징은 서버리스 플랫폼 성능을 좌우하는 중요한 요소로 등장하고 있다. 함수 실행 요청(function invocation request)가 발생하는 경우 서버리스 플랫폼은 해당 함수의 실행에 필요한 라이브러리와 관련 패키지들을 연산 노드(computation node)에 미리 인스톨한다. 연산 노드에 실행 환경을 설정하는 동안 일정 시간의 지연이 생기는 것이 불가피해진다. 오픈람다(OpenLambda)[47]는 패키징과 관련된 실행 지연이 발생하는 대표적인 시스템으로, Amumala[48]는 오픈람다에서 패키징으로 발생하는 실행 지연을 완화하기 위한 스케줄링 기법을 제안하였다.

대부분의 서버리스 플랫폼에서는 애플리케이션 개발 과정에서 개발자가 관련된 라이브러리와 패키지에 대한 정보를 미리 등록하도록 요청하고 있으며, 사용자가 제공한 정보를 활용하여 패키징과 관련된 지연 시간을 줄이려는 시도를 하고

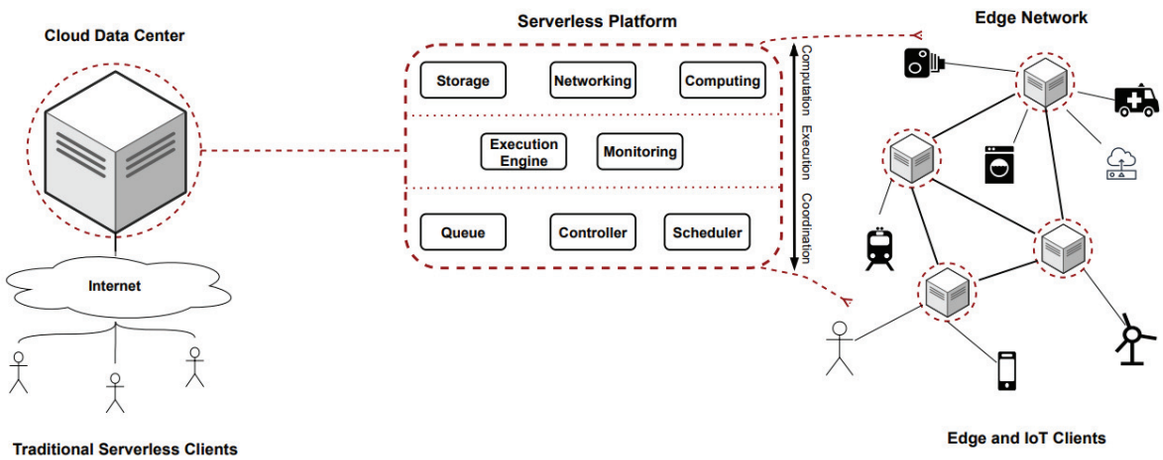

Fig. 3. Simplified Serverless Edge Computing Architecture[52]

있다. 아직까지 관련된 연구는 활발하지 않으나, 사용자에게 부여된 환경 설정이나 자원 관리의 부담을 경감시킨다는 서버리스 컴퓨팅의 취지에 비추어 패키징 지연과 관련된 연구도 향후 연구자들이 고민할 가치가 있는 주제라고 판단된다.

3.3 마이그레이션

서버리스 컴퓨팅의 대표적인 특징은 오토 스케일링(auto-scaling)에 대한 지원이다. 서버리스 컴퓨팅에서는 클라우드와 병렬 컴퓨팅의 장점을 최대한 살려 사용자의 서버리스 함수를 여러 개의 분산 노드에 복사하고, 각각의 복사본이 다른 요청 사항을 수행하는 것이 가능하다. 스케줄러는 해당 함수를 3.2절에서 살펴 본 다양한 기법에 따라 실제로 함수 실행을 담당할 물리적 노드에 할당한다. 이 과정에서 스케줄러의 판단에 따라 이미 실행되고 있는 함수가 다른 물리적 노드로 옮겨서 실행되기도 하는데, 이처럼 연산이 물리적 이동을 거쳐서 지속적으로 수행되는 과정을 마이그레이션(migration)이라고 한다.

함수 마이그레이션은 자동 스케일링을 구현하기 위한 필수적인 요소로, 마이그레이션으로 인한 성능 저하나 지연은 서버리스 플랫폼 성능에 영향을 줄 수 있다. 실제 애플리케이션에서 서버리스 함수들은 1개 혹은 다수의 데이터 소스에서 데이터를 가지고 와서 이를 바탕으로 연산을 수행한다. 따라서 실제 마이그레이션에서는 서버리스 함수 뿐만 아니라 관련된 데이터도 함께 연산 노드 사이를 이동해야 한다.

데이터의 이동을 포함한 마이그레이션은 무상태 함수만으로 이루어진 마이그레이션보다 더 많은 지연을 야기하기도 하는데, 이는 대부분 데이터 제공자 측의 낮은 트래픽 속도가 보틀넥(bottle-neck)으로 작용하기 때문이다[19]. 따라서 데이터 제공자의 네트워크 속도가 느릴 경우에는 함수를 최대한 연관된 데이터와 물리적으로 가까운 연산 노드로 이동시키는 것이 유리하다. 이와 같은 방식은 데이터를 중심으로 서버리스 함수를 패킹했다고 볼 수 있다.

Shredder[49]는 다중 사용자를 위한 클라우드 스토리지 시스템으로 서버리스 함수를 데이터 스토리지 노드 내부에서 수행하는 것을 허용한다. 그렇지만 Shredder의 방식은 데이터 저장과 연산에 사용되는 자원이 서로 공유되면서, 연산 자원의 제공과 활용이 지나치게 복잡해진다는 단점을 가지고 있다.

또 다른 방식은 데이터의 지역성을 높이기 위해서 서버리스 함수와 데이터를 함께 패킹해서 이동시키는 방법이다. Lambda-KV[50]에서는 연산 자원과 스토리지 자원을 하나의 엔터티(entity)로 패킹하여 함께 이동시키는 방식을 택하고 있다. 이와 같은 방식은 데이터의 지역성을 최대한 활용할 수 있다는 장점을 가지고 있으나, 센서 데이터처럼 데이터의 양이 많고 자주 변하는 경우에는 적당하지 않다. 그러나 데이터의 변화 속도가 상대적으로 느린 애플리케이션에서는 상당한 효과를 기대할 수 있다. 애플리케이션 도메인에 따라 2가지 방식의 장단점이 명확히 나누어지기 때문에, Kayak[51]은 상황에 따라 패킹 방식을 결정하는 하이브리드 방식을 제안하였다.

자동 스케일링과 마이그레이션 성능 향상을 위한 데이터 패킹 기법에 대한 연구는 상대적으로 시작 단계에 있기 때문에 앞으로 활발한 연구가 진행될 것으로 예상된다. 앞에서 언급한 데이터 패킹의 방향성 문제뿐만 아니라 다수의 서버리스 함수들간의 데이터 소스 공유 문제 등도 역시 추가적인 연구가 필요한 연구 주제이다.

4. 향후 발전 방향

서버리스 컴퓨팅은 컴퓨팅 자원 관리의 편리성을 바탕으로 비교적 짧은 시간에 빠른 속도로 발전하고 있으며, 기술적으로도 매우 빠르게 성숙하고 있다. 또한 성숙 단계로 접어들면서 서버리스 컴퓨팅을 이용하는 사용자도 증가하고 있다. 따라서 이제는 서버리스 컴퓨팅 서비스의 성능뿐만 아니라 사용자의 이용 편이성과 안전성, 적용 분야의 확장성까지 고려되어야 할 필요가 있다.

본 절에서는 최근 관심을 끌고 있는 서버리스 에지 컴퓨팅과 상태유지 서버리스 컴퓨팅 분야를 중심으로 서버리스 컴퓨팅의 향후 발전 방향을 분석한다. 그리고 사용자가 서버리스 컴퓨팅을 보다 편리하고 안전하게 활용하기 위해서 필요한 애플리케이션 개발 환경과 보안 관련 이슈에 대해서도 논의한다.

4.1 서버리스 에지 컴퓨팅

최근 IoT 애플리케이션 시장의 인기와 확장으로 IoT 관련 애플리케이션 개발과 연관된 요구 사항이 증가하고 있다. IoT에 특화된 요구 사항으로는 짧은 지연 시간과 실시간 실행, 이벤트 중심의 개발, 그리고 효율적인 전개 등을 들 수 있는데, 최근 들어 시장은 이와 같은 요구 사항을 적극적으로 반영하기 시작했다[53]. 클라우드 컴퓨팅은 이러한 IoT 시장의 요구 사항을 만족시킬 수 있는 효과적인 컴퓨팅 모델이지만 중앙 서버가 실제 IoT 데이터 소스에서 멀리 떨어져 있기 때문에 짧은 지연 시간 등의 요구 사항을 충족시키는 데는 어려움이 있다[53].

에지 컴퓨팅 혹은 포그 컴퓨팅은 앞에서 기술된 어려움을 보완하기 위하여 제안된 새로운 컴퓨팅 모델이다[54]. 에지 컴퓨팅에서 에지 노드 혹은 에지 서버는 데이터 소스에 상대적으로 가까운 위치에 위치하고 연산이 가능한 노드를 의미한다. 에지 컴퓨팅은 데이터 소스에서 얻어진 데이터를 멀리 떨어진 중앙 서버에서 처리하기보다 지리적으로 가까운 위치에 있는 에지 서버에서 처리되는 방식이다. 에지 서버에 사용되는 하드웨어로는 라즈베리 파이(Raspberry Pi), 비글보드(Beagle Board), 라떼판다(LattePanda), 엔비디아 젯슨 나노(NVIDIA Jetson Nano) 등이 있다.

에지 컴퓨팅은 에지 서버를 활용함으로써 연산이 데이터 소스와 훨씬 가까운 위치에서 수행되기 때문에 결과적으로 연산을 실시간으로 완료하는데 훨씬 유리한 구조를 가지고 있다. 그렇지만 에지 서버가 가지는 자원은 중앙 서버에 비해서 상대적으로 제한적이어서 가용 에너지나 가용 연산 능력 면에서 안정성이 떨어질 수 있다는 것도 사실이다[55]. 에지 컴퓨팅의 장점을 최대한 살리기 위해서는 에지 네트워크가 가지는 특성을 반영하여 분산되어 있는 에지 서버들을 세밀하게 구성하고 조정하는 과정이 필수적이다.

결과적으로 에지 컴퓨팅은 연산과 자원의 배분을 중앙 서버에서 모두 결정하는 클라우드 방식을 중앙 서버와 에지 서버의 컨소시엄 형태로 변경하여 IoT 애플리케이션 실행에 최적화된 컴퓨팅 모델이라고 할 수 있다. 서버리스 컴퓨팅의 개념이 대두된 2017년 이후 에지 컴퓨팅 영역에 서버리스 컴퓨팅 모델을 적용할 수 있는지에 대한 논의가 꾸준히 계속되고 있다[11, 19, 56]. 이와 같은 컴퓨팅 기법을 서버리스 에지 컴퓨팅(serverless edge computing)이라고 부른다. 산업계에서도 2017년에 아마존이 Lambda@Edge[57]를 발표한 이후 상용화 노력이 지속되고 있다.

Fig. 3는 서버리스 에지 컴퓨팅의 개념적인 모습을 보여 주고 있는데, 기존의 중앙 집중식 서버리스 컴퓨팅에서 사용되었던 서버의 위치와 역할이 에지 네트워크에서 어떻게 변화되었는지 살펴볼 수 있다[52]. 에지 컴퓨팅의 관점에서 보면 에지 서버는 자신에게 연결된 IoT 디바이스들에게 서버리스 기능을 제공하는 역할을 하게 된다. 서버리스 컴퓨팅의 관점으로 살펴보면 서버리스 에지 서버의 주된 역할은 에지에 위치한 IoT 디바이스에서 발생하는 이벤트를 효율적으로 처리할 수 있는 플랫폼을 제공하는 것 이다. IoT 관점으로 보면 시스템 전체적으로는 큰 변화는 없고, 단지 함수 기반 서비스(FaaS) 원칙을 준수한다는 점만이 달라졌다고 볼 수 있다. 클라우드 컴퓨팅의 관점에서 살펴보면 중앙 클라우드 서버를 중심으로 논의되었던 기존 서버리스 컴퓨팅을 분산 환경의 에지 서버들과 함께 통합된 형태로 제공하게 되었다고 볼 수 있다.

에지 컴퓨팅에서 에지 서버는 연산과 통신, 그리고 스토리지의 측면에서 클라우드 데이터 센터의 성격을 가진다. 그렇지만 에지 서버는 용량이 매우 작고, 숫자가 많으며, 완전히 분산된 형태로 존재한다는 면에서 클라우드 데이터 센터와는 차이가 있으며, 이로 인해서 에지 서버에 대한 제어와 관리는 기존의 중앙 집중식 서버보다 훨씬 복잡한 양상을 가진다[55].

서버리스 컴퓨팅은 자원 관리를 더욱 단순하고 간편하게 만들었으며, 이와 같은 장점을 대규모 IoT 센서 데이터에 적용하려는 노력은 앞으로도 지속될 것으로 예상된다. 이 분야는 비교적 새롭게 논의되는 분야로 아직도 해결해야 할 과제를 많이 가지고 있지만, 그렇기 때문에 앞으로 다양한 연구들이 더욱 활발하게 진행될 것으로 예상된다.

4.2 상태유지 서버리스 컴퓨팅

서버리스 컴퓨팅은 함수기반 서비스(FaaS)를 활용한 오토 스케일링에 초점을 맞추어 놀랄만한 성장과 성공을 이루었다. 오토 스케일링 성능을 높이기 위해서 서버리스 플랫폼이 적용한 가장 중요한 설계 원칙은 스토리지 서비스와 연산 서비스의 분리였다[58].

스토리지와 연산 서비스를 분리하면 연산 서비스 파트에서는 연산에 사용할 수 있는 자원을 보다 빨리 확보하고 데이터의 이동을 줄이면서 워크로드를 이동시킬 수 있다는 장점이 있다. 동시에 스토리지 서비스에서는 객체 혹은 키-값(key-value) 형태로 저장된 데이터를 보다 낮은 비용으로 다수의 사용자가 공유하는 서비스를 제공할 수 있다. 스토리지 서비스에서 데이터는 사용 기간에 따라 장기보관 스토리지와 단기보관 스토리지를 분리하여 관리된다. 이와 같은 분리로 효율적인 오토 스케일링의 구현이 가능해졌으며, 서버리스 서비스 이용자는 가격이 높은 연산 자원을 필요한 만큼만 소비하고, 연산 휴지 시간에 스토리지 서비스를 이용하는 방식으로 비용을 절감시킬 수 있다.

그러나 스토리지 서비스와 연산 서비스의 분리는 장점에도 불구하고 개발자에게 적지 않은 제약으로 작용할 수도 있다. 서버리스 플랫폼이 제공하는 오토 스케일링 스토리지 서비스로는 아마존 S3나 DynamoDB 등을 들 수 있는데, 이들 서비

스들은 데이터 접근 빈도와 관계없이 전체적으로 긴 지연시간을 보이는 것이 관찰되고 있다[59]. 특히 데이터에 대한 의존도와 활용 빈도가 높은 기계 학습이나 과학용 계산 애플리케이션의 경우 서버리스 컴퓨팅은 데이터 지연 시간이 길어져서 기대만큼의 성능을 보이지 못하고 있다[60]. 고전적인서버리스 컴퓨팅 모델의 또 다른 제약은 함수 사이에 네트워크를 통한 지점 연결 통신(point-to-point communication)이 허용되지 않는다는 점이다.

이와 같은 제약점으로 인해서 (1) 스토리지 서비스에 대한 개발자들의 요구 사항을 반영하고, (2) 기존의 서버리스 서비스가 가지는 오토 스케일링의 장점은 그대로 유지하면서, (3) 함수들 사이의 상태 공유나 통신을 보다 효율적이고 저렴한 비용으로 제공할 수 있는 서버리스 서비스에 대한 필요성이 증대되고 있다. 고전적인 서버리스 서비스와의 차별성을 강조하기 위해서 서로 독립적인 함수로 구성된 고전적인 서버리스 컴퓨팅을 무상태 서버리스 컴퓨팅(stateless serverless computing), 함수들 사이의 상태 공유에 중점을 둔 서버리스 컴퓨팅을 상태유지 서버리스 컴퓨팅(stateful serverless computing)으로 부르기도 한다.

무상태 서버리스 컴퓨팅은 기본적으로 영속적인 스토리지 서비스를 제공하지 않는다. 만약 서버리스 플랫폼에서 데이터베이스 시스템을 제공하고자 한다면 기본 서버리스 컴퓨팅 모델에서 지원하지 않는 영속적인 스토리지 서비스와 연결 지향적 프로토콜을 지원해야 한다. 또한 분산 데이터베이스는 성능을 높이기 위해서 공유 메모리를 사용하는 경우도 많은데, 서버리스 컴퓨팅에서 각각의 함수는 격리된 메모리에서 실행되고 메모리 자원을 공유하지 않는다[61]. 따라서 서버리스 컴퓨팅에서 사용 가능하도록 공유 메모리를 사용하지 않는 분산 데이터베이스에 대한 연구도 진행되고 있다[62, 63]. 공유 메모리를 사용하지 않는 경우 데이터베이스 서비스를 제공하는 분산 노드는 접근 가능성을 유지하기 위해서 계속 온라인 상태를 유지하게 되는데, 이것은 서버리스 컴퓨팅의 가장 큰 전제인 사용하지 않는 자원은 할당하지 않는다는 원칙에 위배된다는 단점이 있다.

상태유지 서버리스 컴퓨팅을 지원하는 경우의 또 다른 문제점은 멱등성(idempotence) 유지이다. 무상태 서버리스 컴퓨팅에서 사용자 서버리스 함수가 실행 도중에 실패하면 서버리스 플랫폼은 해당 서버리스 함수를 처음부터 다시 실행한다. 이 때 서버리스 함수가 실행할 때마다 같은 결과를 보이는 멱등성을 가지고 있다면 재실행에 기반을 둔 결함 감내(fault tolerance) 기법은 아무런 문제가 없다. 하지만 서버리스 함수의 상태를 유지할 필요가 있는 상태유지 서버리스 컴퓨팅에서 실행에 실패한 함수를 재실행했을 때의 결과는 이전 실행과 달라질 수 있다. Ding[64]은 Flux 시스템에서 상태유지 서버리스 애플리케이션의 멱등성을 자동으로 검증하는 기법을 제안하였다.

서버리스 시스템에서 공유되는 스토리지를 제공하는 방식은 일시적 스토리지(ephemeral storage)와 영속적 스토리지(persistent storage)를 제공하는 방식으로 나누어서 논의할 수 있다. 일시적 스토리지는 함수 사이에 함수의 상태를 전달하기 위해서 사용되는 스토리지를 의미하며 상대적으로 빠른 속도와 짧은 지연시간을 요건으로 한다. 애플리케이션의 성격에 따라 사용 빈도는 달라질 수 있지만, 기본적으로 애플리케이션이 실행되는 동안 해당 애플리케이션의 상태를 공유하는 스토리지를 운영하고 애플리케이션이 종료되는 시점에서 해당 스토리지를 해지하는 방식이다.

서버리스 애플리케이션에서 일시적 스토리지는 최적화된 네트워크 스택 위에 분산 인메모리 서비스를 활용하여 구현된다. RAMCloud[65]와 FaRM[66] 등이 인메모리 스토리지 서비스를 이용한 경우이다. 이와 같은 인메모리 스토리지 서비스는 스토리지 용량이 오토 스케일링되는 기능을 지원해야 하며, 메모리의 할당과 회수가 사용자에게 투명하게 진행되어야 한다. 특히 주의를 기울여야 하는 점은 애플리케이션이 예기치 않게 종료되는 경우에 이미 할당된 자원이 자동으로 회수되어야 하며, 이 과정에서 다른 애플리케이션이 회수되는 스토리지에 부적절하게 접근해서는 안 된다는 것이다. 또한 스토리지 해지 과정에서 다른 애플리케이션의 성능을 저하시키는 일이 발생해서도 안 된다. Pocket[67]의 경우는 일시적 스토리지의 추상화를 시도했으나 자동 스케일링 기능이 부족하고 애플리케이션이 사전에 스토리지 영역을 확보해야 한다는 단점을 보이고 있다.

서버리스에서 영속적인 스토리지 서비스를 제공하기 위해서 많이 사용되는 방식은 SSD를 활용한 분산 스토리지를 분산 인메모리 캐시와 함께 사용하는 방법이다[68, 69]. 서버리스를 활용한 영속적 스토리지 서비스의 경우도 일시적 스토리지 서비스와 마찬가지로 투명한 할당뿐만 아니라 서비스 사용자 영역 사이의 확실한 분리와 보호가 지원되어야 한다. 추가적으로 스토리지 회수에 있어서도 영속적 스토리지의 특성상 사용자가 명시적으로 자원 회수를 요청할 수 있어야 한다.

스토리지 서비스와 관련된 연구는 서버리스 컴퓨팅 영역이 다양한 분야로 확대되면서 필연적으로 등장한 사용자 요구를 해결하기 위한 분야이다. 다수의 애플리케이션이 실행 환경뿐만 아니라 스토리지 서비스를 공유하면서 애플리케이션 내부 정보에 대한 침해사고의 가능성 또한 증가되고 있는 것이 사실이다[70, 71]. 향후 사용자와 접근 권한이 상이한 데이터 사이의 장벽을 명확히 하고 데이터를 상호 보호하는 연구가 스토리지 연구에서 중요한 부분을 담당하게 될 것으로 예상된다.

4.3 소프트웨어 개발 환경

서버리스 컴퓨팅은 상대적으로 최근에 등장한 개념으로 개발과 관련된 환경이나 툴, 프로그래밍 기법, 추상화 모델 등이 아직 충분하지 않은 것이 사실이다. 따라서 서버리스 애플리케이션을 개발하려고 하는 개발자에게 이는 또 다른 개발 장벽으로 작용할 수 있다. 프로그래밍과 모델링 영역에서 충분한 연구와 합의가 진행되지 않은 현재의 상황은 다수의 통합

되지 않은 개발 방식이 공존하는 현상을 야기하였다. 이와 같은 현상은 단기적으로는 개발되는 코드의 질을 저하시킬 것으로 예상되며, 장기적으로는 개발자들 간의 협업을 어렵게 만들 것으로 예상된다.

초기 클라우드 시장에서 하드웨어 가상 머신에 중점을 둔 클라우드 서비스가 성공을 거둔 이유에는 사용자들이 이제까지 사용해 왔던 개발 환경을 클라우드에서도 최대한 유지하고 싶었던 욕구와 클라우드 이전에 개발되었던 시스템을 최소한의 수정으로 재사용하고 해야 하는 필요가 크게 작용했던 것으로 보인다. 이와 같은 현상은 클라우드 서비스가 정착된 이후에 새롭게 등장한 서버리스 컴퓨팅이나 엣지 컴퓨팅에서도 여전히 발생하고 있다.

개발 환경 개선에 대한 사용자들의 요청에 부응하기 위한 연구는 크게 2가지 방향으로 진행되고 있다. 첫 번째 방향은 개발자들이 익숙한 기존의 개발 환경과 유사한 개발 환경을 서버리스 컴퓨팅에서 제공하여 개발 장벽을 낮추고자하는 연구이고, 두 번째 방향은 기존의 소프트웨어를 서버리스 플랫폼에 실행 가능한 형태로 자동 변화하고자 하는 연구이다.

개발환경 개선과 관련된 분야에서는 기존의 프레임워크를 대체하여 사용자가 보다 편리하게 애플리케이션을 개발할 수 있도록 도와주는 미들웨어나 프레임워크에 관련된 연구들이 가장 활발하게 진행되고 있다. Perez[72]는 서버리스 플랫폼에 적합한 프로그래밍 모델과 서버리스 애플리케이션을 위한 미들웨어를 제안하였다. 제안된 프로그래밍 모델과 미들웨어는 파일 처리 애플리케이션에 초점을 맞추고 있다. Kappa[73]는 서버리스 컴퓨팅을 위해 개발된 프로그래밍 프레임워크이다. 저자들은 논문에서 서버리스 플랫폼에서 병렬 프로그래밍을 보다 쉽게 개발할 수 있는 방법을 제안하고 있는데, 제안된 프레임워크는 일반적인 파이썬 프로그램을 서버리스 플랫폼에서 실행 가능하도록 병렬 람다 함수를 사용하는 코드로 변환하는 기능을 제공하고 있다. Risco[74]는 과학 데이터 처리에 특화된 오픈소스 플랫폼을 제안하였다. 제안된 플랫폼은 에지에서 수집된 데이터 처리를 위해서 온프레미스(on-premises) 클라우드와 퍼블릭 클라우드를 함께 사용하고 있다.

기존 하이 레벨 프로그래밍 언어로 작성된 프로그램을 개발자가 다시 작성할 필요 없이 자동 변환을 통하여 서버리스 플랫폼에서 실행시킬 수 있다면 이미 개발된 소프트웨어를 다시 개발하는 부담을 줄일 수 있다. PyWren[75]은 기존의 파이썬 프로그램을 변경 없이 아마존 람다 플랫폼에서 실행시킬 수 있는 기능을 포함하고 있다.

서버리스 컴퓨팅 소프트웨어의 정형 분석을 위한 모델이나 운영 의미론(operational semantics)에 대한 연구도 꾸준히 진행되고 있다. 서버리스라는 새로운 패러다임에 정확한 이론적 의미를 부여하는 연구는 정적 분석이나 정형 분석 기법을 서버리스 컴퓨팅 분야에 활용하는 바탕이 될 것으로 예상된다. Gabbrielli[76]는 람다식(lambda calculus)을 활용하여 서버리스 컴퓨팅을 위한 정형 모델을 제안하였다. Winzinger[77]는 논문에서 서버리스 애플리케이션을 모델에 기반을 둔 분석 기법으로 설명하였다.

서버리스 컴퓨팅 방식은 사용자가 작성한 코드를 서버에서 실행시키는 방식을 사용하기 때문에 공격자가 제출한 악성 코드를 판별하고 분리해 내는 작업은 매우 중요하다. 악성코드의 판별과 분리가 제대로 이루어지지 않는 경우 서버리스 플랫폼 전체 혹은 실행 환경을 공유하는 다수의 사용자에 대한 공격이나 침해 사고가 발생할 수 있기 때문이다. 서버리스 함수들이 대부분 기존의 프로그래밍 언어로 작성되기 때문에 서버리스 함수를 분석해서 안전성을 판단하는 작업에는 기존의 소스 코드 분석 기법이 충분히 활용될 수 있다. 서버리스 컴퓨팅의 특성을 반영한 코드 분석 기법도 앞으로 많은 연구와 관심이 필요한 분야이다.

소프트웨어 개발 환경에서 코드 개발 못지않게 중요한 역할을 담당하는 영역은 디버깅과 테스팅, 그리고 벤치마크 기법이다. Lenarduzzi[78]는 서버리스 컴퓨팅의 특성을 고려한 디버깅과 테스팅에 관련된 이슈와 문제점 관련된 연구 동향을 분석하였다. Winzinger[79]는 서버리스 함수에 대한 통합 테스팅을 할 수 있는 기법을 제안하였다. Gan[80]은 논문에서 서버리스 플랫폼에서 유용한 다양한 벤치마크 기법에 대해서 논의하였다. ServerlessBench[81]는 오픈소스 벤치마크 스위트로 통신 효율성과 같은 유용한 메트릭을 측정할 수 있는 다양한 테스트 케이스들로 구성되어 있다.

시뮬레이션 기법은 각각의 서버리스 플랫폼의 주요 성능 지표를 예측하여 보다 나은 파라미터와 플랫폼을 선택하는데 도구로 활용될 수 있다. FaaSdom[82]은 서버리스 컴퓨팅 플랫폼에서 다양한 서비스 제공자 사이의 예상비용을 산출해 볼 수 있는 모델을 제안하고 있다. SimFaaS[83]은 대표적인 서버리스 컴퓨팅 시뮬레이터로서 다양한 서버리스 플랫폼에 해당하는 성능 시뮬레이션 기능을 제공한다.

4.4 시스템 및 정보보안

시스템과 데이터에 대한 보호는 컴퓨팅 서비스에서 반드시 제공되어야 할 요소이며, 클라우드 컴퓨팅과 서버리스 컴퓨팅은 관련 보안 이슈에서 상당히 많은 공통점을 보여주고 있다[84]. 서버리스 컴퓨팅에서 중요한 문제는 특정 함수를 실행시키는데 있어서 정확한 권한을 확인할 수 있어야 한다는 점이다. 이는 정보의 보호뿐만 아니라 자원 사용에 따른 과금과도 직결되는 문제로, 서비스 제공자들은 이를 해결하기 위해 독자적인 방법을 사용하고 있다. 가장 대표적인 방법은 함수요청 헤더에 인증(authentication) 및 권한(authorization)과 관련된 토큰을 포함시키는 방법으로, 아마존 서비스에서 사용되는 람다 권한 부여자(Lambda Authorizer)[85]나 JWT(JSON Web Token) 등이 대표적인 사례이다. Swedha[86]는 함수요청에 디지털 서명을 추가하는 방식을 사용하여 인증과 권한에 사용되는 에너지 사용량을 줄이는 기법을 제안하였다.

토큰을 활용하는 인증 방식은 토큰의 추출이나 리플레이 공

격 등을 막기 위하여 반드시 보안이 강화된 네트워크를 통하여 통신이 이루어져야 한다. 하지만 최근 부각되고 있는 에지 서버리스 컴퓨팅처럼 전력이 충분하지 않은 IoT 디바이스는 암호화가 필요한 보안 네트워크 사용이 쉽지 않다는 문제점이 지적되고 있으며, 이에 대한 추가적인 연구가 필요한 상황이다.

최근에 등장한 멜트다운 공격[70]과 스펙터 공격[71]은 추측 실행(pre-fetch execution) 과정에서 충분한 보안 검사가 이루어지지 않아서 보안 취약점이 발생하는 경우이다. 해당 논문에서 저자들은 이와 같은 보안 취약점을 악용한 침해 사고가 발생할 수 있다는 것을 증명하였다. 이와 같은 추측 실행 보안 취약점은 클라우드 컴퓨팅이나 서버리스 컴퓨팅처럼 다수의 애플리케이션이 실행 환경을 공유하는 경우 지속적으로 발생할 수 있으며, 다양한 침해 사고를 발생시킬 위험성이 있다.

다수의 애플리케이션들이 실행 환경을 공유하는 환경에서 애플리케이션들을 상호 보호하기 위해서는 런타임에 실행 중인 함수를 모니터링해서 악성 공격을 찾아내고 막는 기법이 제안되고 있다. SecLambda[87]는 컨테이너 런타임 환경을 개선하여 시스템의 안전성을 높였다. 제안된 시스템은 HTTP 프로토콜 형태의 함수 요청이나 입출력 연산을 사전 체크하여 해당 함수 호출이 시스템 내부에서 미리 정의된 보안 정책에 부합하는지 검사하는 방식으로 악성 공격으로부터 사용자를 보호한다. Valve[88]는 런타임 추적과 정보흐름 제어(information flow control) 기법을 활용한 방식을 제안하였으며, Trapeze[89]는 미리 정의된 정보흐름 정책에 따라 샌드 박스를 활용하는 방법을 제안하였다. 다수의 사용자가 공통의 실행 환경을 공유하는 환경에서 민감한 개인 정보를 다루는 경우에는 각별한 주의가 필요하며, 이는 요즘 활발한 사용이 진행되고 있는 에지 서버리스 컴퓨팅에서는 매우 중요한 이슈로 떠오르고 있다.

서비스 제공자와 연구자들의 노력으로 요즘에는 실제 데이터에 대한 직접적인 침해는 상당히 어려워진 것이 사실이다. 그렇지만 최근 공격자들은 공격을 통해서 직접적인 정보를 수집하기보다 문맥적인 데이터를 수집하여 희생자들의 정보를 획득하는 방식으로 공격의 양상을 변화시키고 있다. 공격자들은 희생자들이 어떤 함수를, 어떤 순서로, 언제, 어느 위치에서, 얼마나 자주 호출하는가에 대한 문맥 정보를 수집하고 이를 희생자에 대한 공격 방법을 결정하는데 활용한다. 다양한 채널로 수집되는 개인 정보 노출에 대한 위험성은 증가하고 있으나 아직 이에 대한 연구는 미흡한 상황이다. 따라서 향후 이 분야에 대한 보다 활발한 연구가 필요할 것으로 예상된다.

5. 결론 및 전망

클라우드 환경에서 복잡한 가상 환경 관리에서 벗어서 애플리케이션 본연의 작업에 집중하기를 원하는 사용자의 요구는 서버리스 컴퓨팅의 탄생을 가져왔다. 사용자는 서버리스 컴퓨팅 모델을 활용하여 서버에서의 자원 할당이나 복잡한 서버 관리는 서비스 제공자에게 위임하고, 사용자 자신은 애플리케이션 개발에만 집중하여 클라우드 환경을 이용할 수 있다. 서버리스 컴퓨팅은 클라우드 서비스 사용자의 부담을 감소시킴으로써 클라우드 컴퓨팅의 활용도를 한 단계 업그레이드시켰으며, 향후 클라우드 컴퓨팅의 기반 모델로 자리 잡을 것으로 예상된다.

본 논문에서는 서버리스 컴퓨팅 시스템의 성능을 향상시키기 위해서 고려해야 할 요소들을 판별하여 각 요소별 현재 연구 동향을 분석하고, 이를 바탕으로 향후 서버리스 컴퓨팅 연구의 발전 방향을 살펴보았다. 사용자와 활용 분야의 증가로 새로운 도메인들이 서버리스 컴퓨팅 모델에 계속적으로 추가되고 있으며, 에지 컴퓨팅과 스토리지 서비스가 그 대표적인 예이다. 해당 분야에 대한 연구는 이미 상당히 진행되고 있으나 추가적인 연구와 관심이 필요하다고 생각된다.

앞으로 서버리스 컴퓨팅 모델은 클라우드를 위한 기본컴퓨팅 모델로 자리를 잡을 것으로 예상되고 있다. 현재 서버리스 컴퓨팅의 인기와 해당 분야의 확장속도를 고려했을 때, 사용자뿐만 아니라 침해 사고 역시 함께 증가할 것으로 예상되며, 다수의 애플리케이션이 공통된 실행 환경을 공유하는 서버리스 컴퓨팅의 특성을 반영하는 이론적 모델의 수립과 이를 활용한 보안 관련 연구가 앞으로 더욱 활발하게 진행될 필요가 있다고 판단된다.

References

[1] M. Armbrust et al., “Above the clouds: A Berkeley view of cloud computing,” *Technical Report UCB/EECS-2009-28*, EECS Department, University of California at Berkeley, 2009.

[2] P. Castro, V. Ishakian, V. Muthusamy, and A. Slominski, “The rise of serverless computing,” *Communications of the ACM*, Vol.62, No.12, pp.44-54, 2019.

[3] “Amazon EC2.” https://aws.amazon.com/pm/ec2/. Online available; accessed at 2023/09/28.

[4] “Google App Engine.” https://cloud.google.com/appengine/. Online available; accessed at 2023/09/28.

[5] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa, “Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations,” in *Proceedings of the 44th annual IEEE/ACM International Symposium on Microarchitecture*, pp.248-259, 2011.

[6] H. Yang, A. Breslow, J. Mars, and L. Tang, “Bubble-flux: Precise online QoS management for increased utilization in warehouse scale computers,” *ACM SIGARCH Computer Architecture News*, Vol.41, pp.607-618, 2013.

[7] C. Delimitrou and C. Kozyrakis, “Quasar: Resource-efficient and QoS-aware cluster management,” *ACM SIGPLAN Notices,* Vol.49, pp.127-144, 2014.

[8] Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, Vol.1, pp.7-18, 2010.

[9] B. Varghese and R. Buyya, "Next generation cloud computing: New trends and research directions," *Future Generation Computer Systems*, Vol.79, pp.849-861, 2018.

[10] J. Wen, Z. Chen, X. Jin, and X. Liu, "Rise of the planet of serverless computing: A systematic review," *ACM Transactions on Software Engineering and Methodology*, 2023.

[11] H. Shafiei, A. Khonsari, and P. Mousavi, "Serverless computing: a survey of opportunities, challenges, and applications," *ACM Computing Surveys*, Vol.54, pp.1-32, 2022.

[12] "Serverless - the next step in cloud computing." https://quintagroup.com/blog/serverless-cloud-computing. Online available; accessed at 2023/12/23.

[13] A. Mampage, S. Karunasekera, and R. Buyya, "A holistic view on resource management in serverless computing environments: Taxonomy and future directions," *ACM Computing Surveys (CSUR)*, Vol.54, pp.1-36, 2022.

[14] Z. Li, L. Guo, J. Cheng, Q. Chen, B. He, and M. Guo, "The serverless computing survey: A technical primer for design architecture," *ACM Computing Surveys (CSUR)*, Vol.54, pp.1-34, 2022.

[15] "AWS Lambda." https://aws.amazon.com/lambda. Online available; accessed at 2023/09/28.

[16] "Google Cloud Functions." https://cloud.google.com/functions. Online available; accessed at 2023/09/28.

[17] "Azure Functions." https://azure.microsoft.com/products/functions. Online available; accessed at 2023/09/28.

[18] "Apache OpenWhisk." https://openwhisk.apache.org. Online available; accessed at 2023/09/28.

[19] E. Jonas et al., "Cloud programming simplified: A Berkeley view on serverless computing," *CoRR*, Vol.abs/1902.03383, 2019.

[20] C. Lin and H. Khazaei, "Modeling and optimization of performance and cost of serverless applications," *IEEE Transactions on Parallel and Distributed Systems*, Vol.32, No.3, pp.615-632, 2020.

[21] A. Mahgoub, E. B. Yi, K. Shankar, E. Minocha, S. Elnikety, S. Bagchi, and S. Chaterji, "WiseFuse: Workload characterization and DAG transformation for serverless workflows," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, Vol.6, No.2, pp.1-28, 2022.

[22] "AWS Step Functions." https://aws.amazon.com/step-functions/. Online available; accessed at 2023/09/28.

[23] "Azure Durable Functions." https://docs.microsoft.com/en-us/azure/azure-functions/durable/. Online available; accessed at 2023/09/28.

[24] "Google Cloud Composer." https://cloud.google.com/composer. Online available; accessed at 2023/09/28.

[25] "Amazon State Language." https://docs.aws.amazon.com/step-functions/latest/. Online available; accessed at 2023/09/28.

[26] S. Burckhardt et al., "Netherite: Efficient execution of serverless workflows," *Proceedings of the VLDB Endowment,* Vol.15, No.8, pp.1591-1604, 2022.

[27] S. Burckhardt, C. Gillum, D. Justo, K. Kallas, C. McMahon, and C. S. Meiklejohn, "Serverless workflows with durable functions and netherite," *arXiv preprint arXiv:2103.00033*, 2021.

[28] S. Ristov, S. Pedratscher, and T. Fahringer, "AFCL: An abstract function choreography language for serverless workflow specification," *Future Generation Computer Systems*, Vol.114, pp.368-382, 2021.

[29] P. G. López, A. Arjona, J. Sampé, A. Slominski, and L. Villard, "Triggerflow: trigger-based orchestration of serverless workflows," in *Proceedings of the 14th ACM international conference on distributed and event-based systems*, pp.3-14, 2020.

[30] J. Wen and Y. Liu, "An empirical study on serverless workflow service," *arXiv preprint arXiv:2101.03513*, 2021.

[31] A. John, K. Ausmees, K. Muenzen, C. Kuhn, and A. Tan, "Sweep: accelerating scientific research through scalable serverless workflows," in *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing Companion,* pp.43-50, 2019.

[32] A. Suresh, G. Somashekar, A. Varadarajan, V. R. Kakarla, H. Upadhyay, and A. Gandhi, "Ensure: Efficient scheduling and autonomous resource management in serverless environments," in *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS),* pp.1-10, 2020.

[33] J. R. Gunasekaran, P. Thinakaran, N. Chidambaram, M. T. Kandemir, and C. R. Das, "Fifer: Tackling underutilization in the serverless era," *arXiv preprint arXiv:2008.12819*, 2020.

[34] R. B. Roy, T. Patel, and D. Tiwari, "IceBreaker: Warming serverless functions better with heterogeneity," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pp.753-767, 2022.

[35] A. Suresh and A. Gandhi, "Fnsched: An efficient scheduler for serverless functions," in *Proceedings of the 5th international Workshop on Serverless Computing*, pp.19-24, 2019.

[36] N. Akhtar, A. Raza, V. Ishakian, and I. Matta, "Cose: Configuring serverless functions using statistical learning," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pp.129-138, 2020.

[37] M. HoseinyFarahabady, Y. C. Lee, A. Y. Zomaya, and Z. Tari, "A QoS-aware resource allocation controller for function as a service (FaaS) platform," in *Service-Oriented Computing: 15th International Conference, ICSOC 2017*, Malaga, Spain, November 13-16, pp.241-255, 2017.

[38] M. R. HoseinyFarahabady, A. Y. Zomaya, and Z. Tari, "A model predictive controller for managing QoS enforcements and microarchitecture-level interferences in a lambda platform," *IEEE Transactions on Parallel and Distributed Systems*, Vol.29, No.7, pp.1442-1455, 2017.

[39] Y. K. Kim, M. R. HoseinyFarahabady, Y. C. Lee, and A. Y. Zomaya, "Automated fine-grained CPU cap control in serverless computing platform," *IEEE Transactions on Parallel and Distributed Systems*, Vol.31, No.10, pp.2289-2301, 2020.

[40] N. Daw, U. Bellur, and P. Kulkarni, "Xanadu: Mitigating cascading cold starts in serverless function chain deployments," in *Proceedings of the 21st International Middleware Conference*, pp.356-370, 2020.

[41] A. Singhvi, K. Houck, A. Balasubramanian, M. D. Shaikh, S. Venkataraman, and A. Akella, "Archipelago: A scalable low-latency serverless platform," *arXiv preprint arXiv: 1911.09849*, 2019.

[42] A. Tariq, A. Pahl, S. Nimmagadda, E. Rozner, and S. Lanka, "Sequoia: Enabling quality-of-service in serverless computing," in *Proceedings of the 11th ACM Symposium on Cloud Computing*, pp.311-327, 2020.

[43] E. Hunhoff, S. Irshad, V. Thurimella, A. Tariq, and E. Rozner, "Proactive serverless function resource management," in *Proceedings of the 2020 Sixth International Workshop on Serverless Computing*, pp.61-66, 2020.

[44] V. Sreekanti, C. Wu, X. C. Lin, J. Schleier-Smith, J. M. Faleiro, J. E. Gonzalez, J. M. Hellerstein, and A. Tumanov, "Cloudburst: Stateful functions-as-a-service," *arXiv preprint arXiv:2001.04592*, 2020.

[45] T. Rausch, A. Rashed, and S. Dustdar, "Optimized container scheduling for data-intensive serverless edge computing," *Future Generation Computer Systems*, Vol.114, pp.259-271, 2021.

[46] K. Kaffes, N. J. Yadwadkar, and C. Kozyrakis, "Hermod: principled and practical scheduling for serverless functions," in *Proceedings of the 13th Symposium on Cloud Computing*, pp.289-305, 2022.

[47] S. Hendrickson, S. Sturdevant, T. Harter, V. Venkataramani, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Serverless computation with OpenLambda," in *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 16)*, 2016.

[48] G. Aumala, E. Boza, L. Ortiz-Avilés, G. Totoy, and C. Abad, "Beyond load balancing: Package-aware scheduling for serverless platforms," in *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, pp.282-291, 2019.

[49] T. Zhang, D. Xie, F. Li, and R. Stutsman, "Narrowing the gap between serverless and its state with storage functions," in *Proceedings of the ACM Symposium on Cloud Computing*, pp.1-12, 2019.

[50] R. Shashidhara, "Lambda-KV: Distributed key-value store with co-located serverless compute." unpublished.

[51] J. You, J. Wu, X. Jin, and M. Chowdhury, "Ship compute or ship data? why not both?," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pp.633-651, 2021.

[52] M. S. Aslanpour, S. S. Gill, and A. N. Toosi, "Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research," *Internet of Things*, Vol.12, 2020.

[53] R. Buyya et al., "A manifesto for future generation cloud computing: Research directions for the next decade," *ACM Computing Surveys (CSUR)*, Vol.51, No.5, pp.1-38, 2018.

[54] M. Ghobaei-Arani, A. Souri, and A. A. Rahmanian, "Resource management approaches in fog computing: a comprehensive review," *Journal of Grid Computing*, Vol.18, No.1, pp.1-42, 2020.

[55] R. Mahmud, K. Ramamohanarao, and R. Buyya, "Application management in fog computing environments: A taxonomy, review and future directions," *ACM Computing Surveys (CSUR)*, Vol.53, No.4, pp.1-43, 2020.

[56] I. Baldini, P. Cheng, S. J. Fink, N. Mitchell, V. Muthusamy, R. Rabbah, P. Suter, and O. Tardieu, "The serverless trilemma: Function composition for serverless computing," in *Proceedings of the 2017 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, pp.89-103, 2017.

[57] "Lambda@edge." https://aws.amazon.com/lambda/edge/. Online available; accessed at 2023/09/28.

[58] S. Han, N. Egi, A. Panda, S. Ratnasamy, G. Shi, and S. Shenker, "Network support for resource disaggregation in next-generation data centers," in *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*, pp.1-7, 2013.

[59] J. M. Hellerstein, J. Faleiro, J. E. Gonzalez, J. Schleier-Smith, V. Sreekanti, A. Tumanov, and C. Wu, "Serverless computing: One step forward, two steps back," *arXiv preprint arXiv:1812.03651*, 2018.

[60] D. Barcelona-Pons, P. Sutra, M. Sánchez-Artigas, G. París, and P. García-López, "Stateful serverless computing with crucial," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol.31, No.3, pp.1-38, 2022.

[61] J. M. Hellerstein, M. Stonebraker, J. Hamilton, "Architecture of a database system," *Foundations and Trends® in Databases*, Vol.1, No.2, pp.141-259, 2007.

[62] J. C. Corbett et al., "Spanner: Google's globally distributed database," *ACM Transactions on Computer Systems (TOCS)*, Vol.31, No.3, pp.1-22, 2013.

[63] "Cockroach DB." https://www.cockroachlabs.com/docs/stable/developer-guide-overview. Online available; accessed at 2023/09/28.

[64] H. Ding, Z. Wang, Z. Shen, R. Chen, and H. Chen, "Automated verification of idempotence for stateful serverless applications," in *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*, pp.887-910, 2023.

[65] J. Ousterhout et al., "The RAMCloud storage system," *ACM Transactions on Computer Systems (TOCS)*, Vol.33, No.3, pp.1-55, 2015.

[66] A. Dragojević, D. Narayanan, M. Castro, and O. Hodson, "FaRM: Fast remote memory," in *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pp.401-414, 2014.

[67] A. Klimovic, Y. Wang, P. Stuedi, A. Trivedi, J. Pfefferle, and C. Kozyrakis, "Pocket: Elastic ephemeral storage for serverless analytics," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pp.427-444, 2018.

[68] C. Wu, V. Sreekanti, and J. M. Hellerstein, "Autoscaling tiered cloud storage in anna," *Proceedings of the VLDB Endowment*, Vol.12, pp.624-638, 2019.

[69] A. Chen, "A review of emerging non-volatile memory (NVM) technologies and applications," *Solid-State Electronics*, Vol.125, pp.25-38, 2016.

[70] M. Lipp et al., "Meltdown: Reading kernel memory from user space," *Communications of the ACM*, Vol.63, No.6, pp.46-56, 2020.

[71] P. Kocher et al., "Spectre attacks: Exploiting speculative execution," *Communications of the ACM*, Vol.63, No.7, pp.93-101, 2020.

[72] A. Pérez, G. Moltó, M. Caballer, and A. Calatrava, "A programming model and middleware for high throughput serverless computing applications," in *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pp.106-113, 2019.

[73] W. Zhang, V. Fang, A. Panda, and S. Shenker, "Kappa: A programming framework for serverless computing," in *Proceedings of the 11th ACM Symposium on Cloud Computing*, pp.328-343, 2020.

[74] S. Risco, G. Moltó, D. M. Naranjo, and I. Blanquer, "Serverless workflows for containerised applications in the cloud continuum," *Journal of Grid Computing*, Vol.19, pp.1-18, 2021.

[75] E. Jonas, Q. Pu, S. Venkataraman, I. Stoica, and B. Recht, "Occupy the cloud: Distributed computing for the 99%," in *Proceedings of the 2017 Symposium on Cloud Computing*, pp.445-451, 2017.

[76] Maurizio, S. Giallorenzo, I. Lanese, F. Montesi, M. Peressotti, and S. P. G. Zingaro, "No more, no less: A formal model for serverless computing," in *Coordination Models and Languages: 21st IFIP WG 6.1 International Conference*, pp.148-157, 2019.

[77] S. Winzinger and G. Wirtz, "Model-based analysis of serverless applications," in *2019 IEEE/ACM 11th International Workshop on Modelling in Software Engineering (MiSE)*, pp.82-88, 2019.

[78] V. Lenarduzzi and A. Panichella, "Serverless testing: Tool vendors' and experts' points of view," *IEEE Software*, Vol.38, No.1, pp.54-60, 2020.

[79] S. Winzinger and G. Wirtz, "Applicability of coverage criteria for serverless applications," in *2020 IEEE International Conference on Service Oriented Systems Engineering (SOSE)*, pp.49-56, 2020.

[80] Y. Gan et al., "An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp.3-18, 2019.

[81] T. Yu et al., "Characterizing serverless platforms with serverlessbench," in *Proceedings of the 11th ACM Symposium on Cloud Computing*, pp.30-44, 2020.

[82] P. Maissen, P. Felber, P. Kropf, and V. Schiavoni, "Faasdom: A benchmark suite for serverless computing," in *Proceedings of the 14th ACM International Conference on Distributed and Event-based Systems*, pp.73-84, 2020.

[83] N. Mahmoudi and H. Khazaei, "SimFaaS: A performance simulator for serverless computing platforms," *arXiv preprint arXiv:2102.08904*, 2021.

[84] M. Prechtl, R. Lichtenthäler, and G. Wirtz, "Investigating possibilites for protecting and hardening installable FaaS platforms," in *14th Symposium and Summer School on Service-Oriented Computing*, SummerSOC 2020, Crete, Greece, September 13-19, 2020 14, pp.107-126, 2020.

[85] "Amazon Lambda authorizer." https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-use-lambda-authorizer.html. Online available; accessed at 2023/09/28.

[86] K. Swedha and T. Dubey, "Analysis of web authentication methods using amazon web services," in *9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pp.1-6, 2018.

[87] D. S. Jegan, L. Wang, S. Bhagat, T. Ristenpart, and M. Swift, "Guarding serverless applications with seclambda," *arXiv preprint arXiv:2011.05322*, 2020.

[88] P. Datta, P. Kumar, T. Morris, M. Grace, A. Rahmati, and A. Bates, "Valve: Securing function workflows on serverless computing platforms," in *Proceedings of The Web Conference 2020*, pp.939-950, 2020.

[89] K. Alpernas, C. Flanagan, S. Fouladi, L. Ryzhyk, M. Sagiv, T. Schmitz, and K. Winstein, "Secure serverless computing using dynamic information flow control," *Proceedings of the ACM on Programming Languages*, Vol.2, pp.1-26, 2018.

이 은 영

https://orcid.org/0000-0001-8703-9730
e-mail : elee@dongduk.ac.kr
1996년 고려대학교 전산과학과(학사)
1998년 고려대학교 전산과학과(석사)
2000년 미국 Princeton University, Dept. of Computer Science(석사)
2004년 미국 Princeton University, Dept. of Computer Science (박사)
2005년 ~ 현 재 동덕여자대학교 컴퓨터학과 교수
관심분야 : 클라우드 컴퓨팅, 소프트웨어 보안, 프로그래밍 언어