최근 정보통신 기술 서버리스 컴퓨팅을 위한 복원력 있는 디버거에 관한 연구

김도균, 정현서, 김경연, 박상오*
중앙대학교, 중앙대학교, 중앙대학교, *중앙대학교

dgkim@cslab.cau.ac.kr, hyunseo0412@cau.ac.kr, ky0207@cau.ac.kr, *sopark@cau.ac.kr

A Study on the Resilient Debugger for Serverless Computing systems

Kim Do Gyun, Jung Hyun Seo, Kim Kyeong Yeon, Park Sang Oh*
Chungang Univ., Chungang Univ., Chungang Univ., *Chungang Univ.

요 약

서버리스 컴퓨팅의 폭넓은 도입으로 인해 플랫폼이 본질적으로 가지는 무상태성(stateless)과 일회성(ephemeral)으로부터 비롯된 디버깅상의 도전 과제가 대두되고 있다. 기존의 로컬 및 원격 디버깅 방식은 클라우드 환경을 정확하게 재현하지 못하거나 함수 타임아웃 시 전체 실행 컨텍스트를 잃어버려 서버리스 패러다임에 효과적으로 대응하지 못한다. 본 논문은 이러한 한계를 극복하기 위해 상태를 유지하는 연속적인 디버깅을 가능케 하는 프레임워크 CSBugger 을 제안한다. CSBugger 의 핵심은 캡처 및 복원(Capture & Apply) 메커니즘과 타임아웃 관리 기능이다. 이 시스템은 타임아웃 직전 실행 상태를 캡처해 새 인스턴스에서 복원하며, 예측 불가능한 중단을 관리 가능한 디버깅 이벤트로 전환시킨다. 이를 통해 CSBugger 은 본질적으로 무상태인 서버리스 환경에 논리적 상태를 부여하여 여러 호출에 걸친 중단 없는 디버깅 세션을 구현한다. 결과적으로 본 프레임워크는 개발자 생산성을 향상시키고, 더 복잡하고 안정적인 서버리스 애플리케이션 개발을 촉진하는 기반을 제공한다.

Ⅰ. 서 론

서버리스 컴퓨팅은 개발자가 서버 인프라를 직접 관리할 필요 없이 애플리케이션을 빌드하고 실행할 수 있도록 하는 클라우드 컴퓨팅 실행 모델이다. 클라우드 제공업체가 리소스 할당부터 확장, 운영까지 자동으로 처리하며, 코드는 특정 이벤트에 의해 트리거될 때만 실행된다. 이러한 효율적인 사용량 기반 과금 모델과 운영 부담 감소라는 명확한 이점 덕분에, 서버리스 아키텍처는 현대적인 애플리케이션 개발의 표준 중 하나로 빠르게 자리 잡았다.

이처럼 서버리스 컴퓨팅이 널리 도입되면서, 플랫폼이 본질적으로 가지는 무상태 및 일회성 특성으로 인해 디버깅 분야에서 상당한 어려움들이 부상하고 있다.

서버에 대한 제한적인 접근, 독립적인 함수 실행, 그리고 플랫폼 정책에 따른 타임아웃 발생 시 모든 실행 상태가 소멸하는 구조적 제약은 개발자가 버그의 근본 원인을 파악하는 것을 매우 어렵게 만든다. 특히, 여러 서비스가 복잡하게 얽힌 이벤트 기반 아키텍처는 전체적인 실행 컨텍스트를 파악하기 어렵게 만들어, 장애 상황을 정확히 분석하고 재현하는 것이 상당히 어려워진다.

이러한 서버리스 컴퓨팅의 특성 때문에 기존의 디버깅 방식들은 명확한 한계를 보인다. 로컬 디버깅은 클라우드 환경의 정확한 런타임 조건과 자원 제약을 재현하지 못하여 종종 운영 환경에서 예기치 못한 문제가 발생한다. 원격 디버깅은 실행 중인 클라우드 환경을 직접 눈으로 볼 수 있는 이점을 제공하지만, 함수 타임아웃 이후 실행 상태를 보존할 수 없다는 결정적인 한계를 가진다. 이와 같은 라이브 디버깅 및 컨텍스트 복원의 신뢰성 부족은 새로운 접근 방식의 필요성을 시사한다. 이러한 한계를 해결하기 위해, 우리는 클라우드 상에서 연속적이고 상태를 유지하는 디버깅 세션을 가능하게 함으로써 동작하는 원격 디버깅 시스템을 제안한다.

Ⅱ. 관련 연구

1) 로컬 디버깅

로컬 디버깅은 초기 개발 단계에서 널리 사용되는 기법으로, 개발자가 클라우드에 배포하는 복잡한 과정 없이 자신의 장비에서 코드를 신속하게 테스트하고 즉각적인 피드백을 받을 수 있다는 명확한 장점을 가진다. 이러한 편리함과 속도는 반복적인 코드 수정과

빠른 프로토타이핑을 가능하게 하여 개발 생산성을 높여준다. 하지만 이러한 접근 방식은 컨테이너나 VM 을 이용해 클라우드 환경을 모방할 뿐, 실제 운영 환경을 완벽하게 복제하지 못한다는 근본적 한계를 갖고 있다. 이러한 불일치는 IAM 정책, 이벤트 트리거, 네트워크 구성과 같은 클라우드 네이티브 서비스 차이에서 기인하며, 종종 배포 후 예기치 못한 장애를 초래한다. 로컬 환경에서는 네트워크 지연이나 콜드 스타트와 같은 운영 환경 특유의 문제를 정확히 재현하는 것이 거의 불가능하다. 그 결과 로컬 테스트를 통과한 코드가 실제 운영 환경에서는 실패할 수 있으며, 이는 실제 클라우드 환경에서 동작하는 디버깅 솔루션의 필요성을 극명하게 보여준다. CSBugger 은 이러한 한계를 극복하기 위해 디버깅 세션을 실제 클라우드 환경에서 직접 실행할 수 있도록 한다.

2) 원격 디버깅

원격 디버깅은 로컬 환경에서는 파악하기 어려운 문제들을 해결하기 위한 강력한 접근법이다. 개발자는 이를 통해 실제 클라우드 환경에 배포된 애플리케이션에 직접 연결하여, 운영 환경의 데이터와 설정을 바탕으로 코드의 동작을 실시간으로 분석할 수 있다. 이러한 방식은 환경 차이로 인해 발생하는 버그를 진단하는 데 특히 효과적이다.

하지만 이처럼 강력한 접근법도 서버리스 패러다임과는 근본적으로 상충한다. 가장 심각한 한계는 기존 디버거들이 서버리스 함수의 일시적이며 무상태적인 특성을 고려하지 못한다는 점이다. 함수 실행이 완료되거나 클라우드 제공자가 정한 타임아웃 제한에 도달하면, 해당 함수 인스턴스와 그 전체 실행 컨텍스트(메모리, 변수, 호출 스택 등)가 완전히 소멸된다. 이로 인해 연결되어 있던 디버깅 세션도 즉시 종료되며, 개발자는 타임아웃 직전의 모든 상태 정보를 잃게 된다. Lambda Live Debugger[5]와 같은 기존 도구들 역시 이러한 한계를 벗어나지 못한다. 일부 실행 가시성을 제공하지만, 해당 도구는 주로 Node.js 환경만 지원하며 타임아웃 발생 시 디버깅 컨텍스트를 보존하지 못하는 근본적인 문제를 공유한다. CSBugger 은 바로 이 문제를 해결하기 위해, 실행 상태를 보존하고 다음 인스턴스에서 복원하는 캡처 및 복원 메커니즘을 도입했다.

Ⅲ. CSBugger

1) 캡처 및 복원

캡처 단계(Capture): 디버깅 세션 동안 CSBugger 은 주기적으로, 또는 함수 타임아웃 임계 시점 직전에 함수의 전체 실행 상태를 스냅샷으로 저장한다. 이 스냅샷에는 현재 실행 위치, 전체 호출 스택, 그리고 각 스택 프레임의 모든 변수가 포함된다. 캡처된 상태 정보는 이후 JSON 과 같은 형식으로 직렬화 되어, 소켓 통신을 통해 개발자의 로컬 시스템으로 안전하게 전송 및 저장된다.

복원 단계(Apply): 함수 디버깅 인스턴스가 타임아웃으로 종료되면, 저장된 상태로부터 디버깅 세션을 재개하기 위해 새로운 함수 인스턴스가 자동으로 시작된다. CSBugger 은 가장 최근에 저장된 상태 파일을 불러와 새로운 인스턴스 내에서 해당 실행 상태를 복원한다. 이러한 캡처-복원 메커니즘을 통해 CSBugger 은 본질적으로 무상태인 서버리스 환경에 논리적 상태를 부여하여 여러 번의 함수 호출에 걸쳐 끊김 없는 디버깅을 가능하게 한다. 이러한 과정은 단순히 변수 값을 복원하는 수준을 넘어, 캡처된 코드 라인으로 정확히 실행 지점을 이동시키고, 호출 스택까지 복원함으로써 동일한 실행 컨텍스트에서 즉시 디버깅 세션을 이어갈 수 있도록 한다.

2) 타임아웃 관리

CSBugger 은 타임아웃을 예측 불가능한 실패 요인에서 예측 가능하며 관리 가능한 디버깅 흐름의 요소로 전환시킨다. 디버깅 세션이 시작되면 시스템은 해당 함수의 남은 실행 시간을 실시간으로 계산하여 개발자에게 보여준다. 이 기능은 앞서 소개한 캡처-복원 메커니즘과 연동되어 작동한다. 함수 타임아웃이 임박하면 시스템은 최신 실행 상태의 보존을 보장하기 위해 자동으로 최종 캡처를 수행한다.

Ⅳ. 결론

본 논문에서는 서버리스 컴퓨팅의 주요 디버깅 한계를 해결하기 위한 새로운 디버깅 프레임워크 CSBugger 을 제안하였다. CSBugger 은 타임아웃을 넘어 지속되는 상태 지속적 연속 디버깅 세션을 가능하게 함으로써 개발자의 생산성을 향상시키고, 보다 견고하고 복잡한 서버리스 애플리케이션의 개발을 촉진한다.

향후 연구에서는 CSBugger 의 호환성을 확장하여, 초기 Python 구현을 넘어 더 다양한 서버리스 런타임 및 플랫폼으로 적용 범위를 확대할 계획이다. 이를 통해 CSBugger 은 다양한 클라우드 환경에서 보편적으로 활용 가능한 디버깅 프레임워크로 진화할 것이다.

ACKNOWLEDGMENT

이 논문은 정부(과학기술정보통신부)의 재원으로

한국연구재단의 지원을 받아 수행된 연구임(RS-2024-

00345869).

참 고 문 헌

[1] J. Wen, Z. Chen, X. Jin, and X. Liu, 'Rise of the Planet of Serverless Computing: A Systematic Review', ACM Trans. Softw. Eng. Methodol., vol. 32, no. 5, Jul. 2023.

[2] Z. Li, L. Guo, J. Cheng, Q. Chen, B. He, and M. Guo, 'The Serverless Computing Survey: A Technical Primer for Design Architecture', ACM Comput. Surv., vol. 54, no. 10s, Sep. 2022.

[3] A. P. Cavalheiro and C. Schepke, "Exploring the Serverless First Strategy in Cloud Application Development," 2023 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW), Porto Alegre, Brazil, 2023, pp.

[4] N. Ekwe-Ekwe and L. Amos, 'The State of FaaS: An Analysis of Public Functions-as-a-Service Providers', in 2024 IEEE 17th International Conference on Cloud Computing (CLOUD), 2024, pp. 430- 438.

[5] ServerlessLife, 'Lambda Live Debugger', 2022. [Online]. Available: https://github.com/Serverless-Life/lambda-live-debugger.