서버리스 컴퓨팅 성능 최적화 동향: 콜드 스타트 변동성 관리 및 지능형 기반 방법 분석 동향

이성준[1], 김종국[2]
[1]고려대학교 전기전자공학과 석사과정
[2]고려대학교 전기전자공학과 교수
ssungjoon@korea.ac.kr, jongkook@korea.ac.kr

Trends in Serverless Computing Performance Optimization: Intelligent Methods for Cold Start Variability Management

Sung-Joon Lee[1], Jong-Kook Kim[2]
[1]School of Electrical Engineering. Korea University
[2]School of Electrical Engineering. Korea University

요 약

서버리스 컴퓨팅(Serverless Computing)은 비용 효율성과 운영 단순성을 제공하지만, Scale-to-Zero 정책으로 인해 발생하는 콜드 스타트(Cold Start) 지연 시간이 SLA (Service Level Agreement)를 저해하는 핵심 과제로 대두된다. 본 논문은 콜드 스타트 문제 해결의 동향을 단순한 지연 시간 단축을 넘어, SLA 만족 및 자원 효율성의 균형을 달성하는 지능형 시스템 운영 전략의 관점에서 분석한다. 주요 최신 연구 동향으로 확률적 모델링, 기계 학습 기반 동적 제어, 딥러닝 기반 선제적 관리을 활용하였고 이를 기반으로 작성된 최신논문을 본문에서 설명한다. 결론적으로, 서버리스 성능 변동성 관리 연구는 AI 및 ML 기술을 통해 워크로드와 환경 변화에 적응하는 자율적이고 지능적인 운영 전략의 영역으로 발전 중이다.

1. 서론

최근 클라우드 컴퓨팅 환경에서 서버리스 컴퓨팅(Serverless Computing), 특히 FaaS(Function-as-a-Service) 모델은 클라우드 활용의 새로운 패러다임으로 빠르게 자리 잡았다. 서버리스는 개발자가 서버 관리나 인프라 운영에 대한 부담 없이 비즈니스 로직에만 집중할 수 있게 하며, 트래픽에 따라 자원이 자동으로 확장되고 사용한 만큼만 지불(Pay-as-you-go)하는 경제적 이점을 제공한다. 그러나 서버리스의 핵심 비용 효율성 전략인 Scale-to-Zero 정책은 근본적인 성능 문제를 야기한다. 유휴 상태의 함수 인스턴스를 메모리에서 해제하여 자원을 절약하는 과정은, 새로운 요청이 들어왔을 때 실행 환경을 재초기화해야 하는 콜드 스타트 (Cold Start) 지연 시간을 필연적으로 발생시킨다. 이 지연은 특히 응답 시간에 민감한 애플리케이션의 SLA (Service Level Agreement) 를 위반할 수 있는 심각한 성능 저하의 주요 원인이 된다[1].

콜드 스타트 문제와 더불어, 서버리스 환경의 고밀도 배치(High-density Colocation)로 인해 다수의 단기 실행 함수들 사이에 자원 경합이 발생하며 성능 변동성(Performance Volatility)이 더욱 증폭된다. 이러한 성능 저하 현상은 단순히 함수 실행 환경의 초기화 시간을 줄이는 기술적 접근만으로는 해결될 수 없으며, 성능(Quality of Service, QoS)과 자원 효율성(비용) 사이의 균형을 맞추는 시스템 차원의 운영 전략이 필요함을 시사한다.

기존의 콜드 스타트 완화 정책들(Time-to-Live, TTL 기반 캐시, 고정 Warm-up)은 동적인 서버리스 환경에 대한 적응성(Adaptability)이 부족하고, 자원 과잉 할당(Over-provisioning)을 초래한다.

본 논문은 서버리스 컴퓨팅 환경에서 콜드 스타트 변동성을 관리하고 SLA 기반의 성능-비용 균형을 달성하기 위한 최신 연구 동향을 체계적으로 분석하는 데 목적이 있다. 이러한 분석을 통해 서버리스 성능 최적화 연구가 단순한 기술적 문제 해결을 넘어 기계학습/심층학습 (Machine Learning/Deep learning, ML/DL) 지능형 기반 자율 운영 시스템으로 나아가는 학술적 진화 과정을 제시한다.

2. 서버리스 성능 변동성 및 콜드 스타트 메커니즘 분석

2.1. 콜드 스타트 발생 단계 및 지연의 정량화

2.1.1. 자원 할당 및 인스턴스 생성: 클라우드 인프라가 요청을 수신하고 새로운 가상 머신(Virtual Machine, VM) 또는 컨테이너를 할당 및 준비하는 단계이다.

① 런타임 환경 초기화: 자바 가상 머신(Java Virtual Machine, JVM)이나 파이썬 인터프리터 등 함수 코드를 실행할 런타임 환경을 로딩하고 초기화하는 단계이다.

② 함수 코드 로딩: 함수 코드와 필요한 종속성(Dependencies)을 스토리지에서 인스턴스로 다운로드하는 단계이다.

③ 사용자 함수 초기화: 사용자 정의 로직에 따른 초기화 작업(데이터베이스 연결, 모델 로딩 등)을 수행하는 단계이다. 이러한 지연 시간은 수백 밀리초에서 수 초에 이르며, 특히 Java와 같은 무거운 런타임에서 더욱 길게 관찰된다.

2.2. 성능 저하의 시스템적 원인 확장

① 다중 함수 간의 자원 경합 및 부분 간섭: 서버리스 플랫폼은 자원 활용률을 높이기 위해 하나의 물리적 서버에 다수의 함수 인스턴스를 배치한다. 이로 인해 인스턴스 간에 CPU 캐시, 메모리 대역폭, 네트워크 자원 등의 경합이 발생하며 성능이 저하된다. 이러한 간섭은 핫스팟 전파 효과를 일으켜 시스템 균형을 해친다.

② 복잡한 함수 종속성: 서버리스 애플리케이션은 종종 여러 함수가 순차적으로 호출되는 서비스 그래프(Service Graph) 또는 워크플로우 형태로 구성된다. 함수들 간의 호출 관계, 공유 자원 사용 등의 공간적(Spatial) 종속성과 시간 경과에 따른 요청 패턴의 시간적(Temporal) 종속성이 복잡하게 얽혀 있어, 한 함수에서의 콜드 스타트나 지연이 전체 워크플로우의 QoS에 연쇄적인 영향을 미친다.

2.3. 성능과 비용의 근본적인 Trade-off

① Warm-up의 비용 증대: 콜드 스타트 지연을 방지하기 위해 컨테이너를 Warm 상태로 유지하는 Pre-Warming 또는 Keep-Alive 전략은 유휴 자원을 소비하게 되어 클라우드 운영 비용을 직접적으로 증가시킨다. 기존의 고정된 TTL 정책은 워크로드 변동성을 반영하지 못해 자원을 과도하게 낭비하는 과잉 할당을 초래한다.

② Scale-to-Zero의 성능 희생: 반대로, 엄격한 Scale-to-Zero 정책은 콜드 스타트 발생 확률을 높여 SLA 위반의 위험을 증가시키고 서비스 품질(QoS)을 희생시킨다.

따라서 최신 연구 동향은 QoS 제약을 충족하면서 자원 과잉 할당비용을 최소화할 수 있는 지능형, 적응형 제어 전략을 개발하는 데 집중하고 있다. 단순히 지연 시간을 줄이는 기술을 넘어, 시스템의 상태와 워크로드 패턴을 예측하여 최적의 운영 결정을 내리는 관리 과학의 영역으로 진화하고 있다.

3. 콜드 스타트 완화 및 지연 시간 단축을 위한 근본적 기술 동향

콜드 스타트 문제 해결의 초기 연구들은 주로 함수의 실행 환경을 미리 준비하거나, 초기화 과정 자체를 최적화하여 지연 시간을 단축하는 근본적인 기술에 집중한다. 이러한 기법들은 크게 사전 준비(Warm-up) 기반과 시작 시간 최적화(Startup Optimization) 기반으로 분류된다.

3.1. 사전 준비 기반 기법 (Pre-Warming and Container Reuse)

사전 준비 기법은 요청이 도착하기 전에 함수 인스턴스를 실행 환경에 유지함으로써 콜드 스타트의 발생을 예방하는 것을 목표로 한다.

① 컨테이너 재사용 (Container Reuse): 함수 실행이 완료된 후에도 컨테이너 인스턴스를 일정 시간 동안 유휴(Idle) 상태로 유지하는 가장 일반적인 방법이다. 이 유휴 기간 내에 새로운 요청이 들어오면 웜 스타트(Warm Start)가 발생하며 지연이 최소화된다. AWS Lambda나 OpenWhisk와 같은 상용 플랫폼들은 컨테이너 회수 시간을 설정하여 이 전략을 기본적으로 사용한다.

② 사전 Warm-up (Pre-Warming) 및 Provisioned Concurrency: 예상되는 트래픽 패턴을 기반으로 함수에 주기적인 더미 요청을 보내거나, 일정 수의 인스턴스를 항시 활성화 상태로 유지하는 Provisioned Concurrency를 활용한다. 이는 콜드 스타트 발생 확률을 낮추지만, 유휴 자원 소비가 증가하여 비용 효율성(Trade-off) 문제가 발생한다.

3.2. 시작 시간 최적화 기법 (Startup Optimization)

이 기법들은 콜드 스타트가 불가피하게 발생했을 때, 초기화에 소요되는 시간을 단축하는 데 집중한다.

① 스냅샷 및 복원 (Snapshotting and

Restoration): 컨테이너 초기화, 런타임 로딩, 종속성 로딩까지 완료된 상태의 메모리와 커널 상태를 스냅샷으로 저장해 둔다. 요청 발생 시 이 스냅샷을 빠르게 복원함으로써 초기화 단계를 건너뛰어 지연 시간을 줄인다. AWS Lambda SnapStart가 대표적인 예이다.

② 지연 로딩 (Lazy Loading): 함수 코드 내의 모든 라이브러리를 초기화 시점에 로드하는 대신, 실제로 사용되는 시점에 로드하여 초기 부하를 줄인다.

③ 패키지 크기 최소화: 함수 배포 패키지 크기를 줄여 다운로드 및 메모리 로딩 시간을 단축한다.

④ 런타임 선택 및 최적화: Go, Rust와 같이 컴파일된 언어는 일반적으로 Python이나 Node.js 같은 인터프리터 언어보다 런타임 시작이 빠르므로, 런타임 선택을 최적화하여 지연을 줄이는 방법이다.

⑤ 함수 융합 (Function Fusion): 워크플로우를 구성하는 여러 함수를 하나의 큰 함수로 논리적으로 결합한다. 이로 인해 워크플로우 내에서 개별 함수 호출 시 발생할 수 있는 콜드 스타트 횟수 자체를 구조적으로 제거하여 전체 QoS를 개선한다.

4. SLA 기반 성능-비용 균형을 위한 지능형 운영 전략 동향

최근의 연구는 수학적 모델링 및 ML을 활용하여 콜드 스타트 발생을 예측 및 제어하고, SLA 제약 하에서 자원 효율성을 극대화하는 지능형 운영 전략 개발에 집중하고 있다.

4.1. 확률적 모델링 기반 용량 계획 및 유휴 시간 최적화

***COCOA* (Cold Start Aware Capacity Planning)**[2]: *COCOA*는 기존 TTL 캐시 (Time-to-Live Cache) 기반의 컨테이너 유휴 시간 설정이 자원 과잉 할당을 유발하는 문제를 지적한다. 이 논문은 콜드 스타트 지연을 FaaS 응답 시간에 통합적으로 반영하기 위해 LQN (Layered Queueing Network)과 M/M/1/setup/delayedoff와 같은 대기열 이론(Queueing Theory) 기반 모델을 활용한다. 주요 기여점으로는 함수가 유휴 상태일 때와 실행 중일 때의 메모리 소비량 차이를 모델에 반영하고, 지정된 SLA를 만족시키는 최소 CPU 및 메모리 용량과 함수별 최적 유휴 시간을 산출하는 것이다. 이를 통해 *COCOA*는 자원 과잉 할당을 효과적으로 줄이면서도 QoS를 보장한다.

4.2. 학습 기반 동적 자원 구성 및 성능 간섭 예측

4.2.1. 비용 효율적인 동적 자원 구성 (COSE)

***COSE* (Configuration and Placement of Serverless Applications)**[3]:

*COSE*는 서버리스 애플리케이션의 메모리, CPU 할당 및 배치(Placement)와 같은 구성 파라미터가 QoS와 비용에 복합적으로 미치는 영향을 분석한다. 이 프레임워크는 Bayesian Optimization (BO) 통계 학습 기법을 사용하여, 최소한의 실제 샘플링 비용만으로 미지의 구성 값(Unseen Configuration)에서의 실행 시간과 비용을 예측한다. 주요 목표는 QoS 제약 조건 내에서 총 비용을 최소화하는 최적의 구성과 배치를 동적으로 찾아내는 것이다.

4.2.2. 증분 학습 기반 성능 간섭 예측 (Gsight)

***Gsight* (Understanding, Predicting and Scheduling Serverless Workloads under Partial Interference)**[4]:

*Gsight*는 서버리스 컴퓨팅에서 고밀도 배치에서 발생하는 자원 경합 문제를 다루며, 이를 부분 간섭 (Partial Interference)이라는 서버리스의 고유한 현상으로 특성화한다. 부분 간섭은 높은 변동성과 공간-시간적 변화를 보이며 핫스팟 전파 효과를 야기한다. 이 논문에서는 증분 랜덤 포레스트 회귀 모델을 사용하여 함수 호출 경로와 자원 중첩 정보(Overlap Codes)를 입력으로 받아, 실제 QoS 저하를 높은 정확도로 예측하는 것이다. 예측된 간섭 정보를 바탕으로 *Gsight*는 SLA를 침해하지 않으면서 함수 밀도를 18.79% 이상 향상시키는 고밀도 스케줄링을 가능하게 하여 자원 활용률을 극대화한다.

4.3. 딥러닝 기반 선제적 Warm-up 및 전체론적 제어

4.3.1. 딥러닝 기반 전체론적 오토스케일링 (FuncScaler)

FuncScaler (Cold-Start-Aware Holistic Autoscaling)[5]:

기존 오토스케일링이 개별 함수에 초점을 맞추어 함수 간의 상호 의존성을 무시한다는 문제를 해결한다. GR-GCN (Gated Recurrent Graph Convolutional Network) 모델을 사용하여 함수 간의 공간-시간적 종속성을 통합적으로 학습하고, 미래의 워크로드를 정확하게 예측한다. 주요 특징은 이 예측을 기반으로 Cold-Start-Aware JQN (Joint Queueing Network) 모델을 활용하여 전체론적이고

선제적인 Warm-up 및 스케일링을 수행하는 것이다. 이는 함수 호출의 병목 현상을 예측하고 미리 Warm 컨테이너를 준비함으로써, 콜드 스타트 지연과 핫스팟 전이를 효과적으로 완화한다.

4.3.2. 강화 학습 기반 적응형 Warm-up 정책

강화 학습(RL) 기반 접근[6]:

고정된 Warm-up 정책의 비효율성(메모리 낭비) 문제를 해결하기 위해, TD-A2C (Temporal Difference Advantage Actor-Critic) 강화 학습 알고리즘을 활용한다. 주요 목표는 에이전트가 함수 호출 패턴을 학습하고, 콜드 스타트 횟수와 유휴 메모리 소비 페널티 사이의 균형을 최대화하는 최적의 Idle-container window를 동적으로 결정하는 것이다. 이는 콜드 스타트 완화의 적응성과 메모리 효율성을 동시에 달성한다. LSTM 네트워크는 요청 버스트에 대비하여 필요한 Pre-warmed 컨테이너 수를 예측하는 보조적인 역할을 수행한다.

4.3.3. TCN 기반 시스템 정책 및 통합 관리

TCN (Temporal Convolutional Network) 기반 전체론적 관리[7]:

이 연구는 이전 ML/DL 접근법들이 여전히 사일로(Silos) 형태로 운영되어 상위 SLA 정책과 통합되지 못한다는 한계를 지적한다. TCN 모델의 장기 예측 능력 5~15분을 활용하여 미래 함수 도착을 대규모로 예측한다. 핵심 기여는 이 예측 결과를 바탕으로 상위 스케줄러(자원, 비용, 워크플로우)와 통신하며, 서비스 우선순위 (SP)와 자원 우선순위 (RP)를 계산하여 요청에 대한 최적화 적용 여부를 결정하는 정책 프레임워크이다. 이는 인프라 계층의 선제적 프로비저닝과 함수 계층의 코드 최적화 및 Warm Node 배정을 통합적으로 조율하는 최종적이고 전체론적인 관리 전략을 제시한다.

5. 결론

본 논문은 서버리스 컴퓨팅의 광범위한 채택을 저해하는 핵심 과제인 콜드 스타트 문제와 이로 인한 성능 변동성을 관리하기 위한 최신 연구 동향을 분석하였다. 콜드 스타트 완화 연구의 흐름은 단순히 지연 시간 자체를 물리적으로 줄이는 근본적인 기술(스냅샷, 함수 융합)을 넘어, SLA와 운영 비용의 균형을 지능적으로 달성하는 시스템 차원의 운영 전략으로 진화하고 있음이 확인된다. 결론적으로, 서버리스 환경에서 성능과 효율성의 Trade-off를 관리하는 것은 더 이상 고정된 규칙이 아니라, ML 및 DL 기술을 통해 워크로드와 환경 변화에 실시간으로 적응하는 지능적 운영 전략의 영역으로 확장되었다.

참고문헌

[1] Eunyoung Lee. "Survey on the Performance Enhancement in Serverless Computing: Current and Future Directions" The Transactions of the Korea Information Processing Society 13, no.2 (2024) : 60-75.doi: 10.3745/TKIPS.2024.13.2.60

[2] Alim Ul Gias and Giuliano Casale, "COCOA: Cold Start Aware Capacity Planning for Function-as-a-Service Platforms," in Proc. 28th Int. Symp. Model. Anal. Simul. Comput. Telecommun. Syst. (MASCOTS), Nov. 2020, pp. 1-8.

[3] Ali Raza, Nabeel Akhtar, Vatche Isahagian, Ibrahim Matta, and Lei Huang, "Configuration and Placement of Serverless Applications Using Statistical Learning," IEEE Trans. Netw. Service Manage., vol. 20, no. 2, pp. 1065-1077, Jun. 2023.

[4] Shijie Song, Haogang Tong, Chunyang Meng, Maolin Pan, and Yang Yu, "FuncScaler: Cold-Start-Aware Holistic Autoscaling for Serverless Resource Management," in Proc. IEEE Int. Conf. Web Services (ICWS), Jul. 2024, pp. 1036-1047.

[5] Laiping Zhao, Yanan Yang, Yiming Li, Xian Zhou, and Keqiu Li, "Understanding, Predicting and Scheduling Serverless Workloads under Partial Interference," in Proc. Int. Conf. High Perform. Comput. Netw. Storage Anal. (SC '21), Nov. 2021, pp. 1-13.

[6] Parichehr Vahidinia, Bahar Farahani, and Fereidoon Shams Aliee, "Mitigating Cold Start Problem in Serverless Computing: A Reinforcement Learning Approach," IEEE Internet Things J., vol. 10, no. 5, pp. 3917-3927, Mar. 2023.

[7] Tam n. Nguyen, "Holistic cold-start management in serverless computing cloud with deep learning for time series," Future Gener. Comput. Syst., vol. 153, pp. 312-325, Mar. 2024.