콜드 스타트 완화를 위한 도커 환경에서의 이미지 레이어 동시성 수준에 따른 풀링 속도 성능 분석

강민우, 김동균, 유헌창, 강지훈
고려대학교 대학원 컴퓨터학과

{mwkang98, kdonggyun97, yuhc, k2j23h}@korea.ac.kr

A Performance Analysis of Pulling Rate Based on Image Layer Concurrency Level in Docker Environment for Cold Start Mitigation

Minwoo Kang, Heonchang Yu, Donggyun Kim, Jihun Kang
Dept. of Computer Science and Engineering, Korea University

요 약

최근에 Serverless 컴퓨팅은 많은 관심을 받는 기술로, 서버 프로비저닝 없이 코드를 배포하고 실행할 수 있으며 요청량에 따라 동적으로 컴퓨팅 리소스를 확장하여 애플리케이션을 안정적으로 운영할 수 있는 환경을 제공한다. Serverless 컴퓨팅의 주요 이슈 중 하나인 cold start 는 함수를 실행하기 위한 컨테이너 초기화 및 구동하는 단계이며, 해당 과정에서는 이미지 풀링이 수행될 수 있다. 이미지 풀링은 cold start 지연의 대부분을 차지하고 함수의 응답시간을 증가 시켜서 사용자 경험에 부정적인 영향을 줄 수 있다. 따라서, 본 논문에서는 cold start 지연을 줄이기 위해 도커를 활용해서 이미지 레이어 동시 풀링 개수를 조절함으로써 이미지 풀링 속도를 개선시킬 수 있는지 분석하였다. 이와 같은 분석을 통해 풀링 개수가 풀링 속도에 영향을 줄 수 있음을 확인하였다.

1. 서론

Serverless 컴퓨팅은 클라우드 컴퓨팅의 한 형태로, 애플리케이션 개발자가 서버를 직접 관리하지 않고도 코드를 배포하고 실행할 수 있는 환경을 제공하는 컴퓨팅 모델이다[1]. 개발자는 코드 작성 및 업로드만 수행하면 되기 때문에 서버 인프라 설정 및 관리에 대한 고려를 할 필요가 없으므로 개발 생산성이 향상된다.

최근에는 주요 클라우드 제공 업체가 Serverless Computing platform 을 제공하면서 관심이 증가하였다. 대표적으로 Amazon 의 AWS Lambda 와 Google Cloud Functions 들이 존재한다. Serverless 플랫폼은 대부분 컨테이너 기반 기술을 사용하여 함수 또는 애플리케이션을 실행한다. 컨테이너 기반 기술은 함수나 애플리케이션의 빠른 시작과 확장을 가능케 하고 격리된 실행 환경을 제공함으로써 안정성을 강화한다.

현재 Serverless 컴퓨팅은 몇 가지의 이슈가 존재하며 그 중에서 가장 많이 논의 되는 이슈는 cold start 현상이다. Cold start 는 함수가 최초로 호출될 때 함수 실행을 위한 컨테이너 초기화 및 구동에 필요한 지연을 의미한다[2]. Cold start 에서 로컬 레지스트리에 이미지가 캐시 되지 않았다면 원격 레지스트리로부터 이미지를 풀링해야 한다. 이미지 풀링은 cold start 로 인한 지연 중 많은 부분을 차지하므로[3], 함수의 응답 시간을 증가하게 만든다. 이는 사용자 경험에 부정적인 영향을 미치므로 해당 이미지 풀링 비용을 최적화함으로써 Cold Start 비용을 최소화하는 것이 중요하다.

본 논문에서는 도커를 활용하여 이미지 레이어의 동시 풀링 개수를 조절하면서 여러 지역에 분산된 레지스트리에 배포한 다양한 크기의 이미지를 풀링하는 실험을 진행했다. 그 후, 이미지 레이어의 동시 풀링 개수를 조절하면 이미지 풀링 속도를 개선 할 수 있는지 분석했다. 해당 실험의 결과로 인해 Serverless 컴퓨팅 환경에서 cold start 에서의 이미지 풀링 비용을 최소화하고 성능을 최적화하는 데 중요한 지침을 제공할 것으로 기대한다.

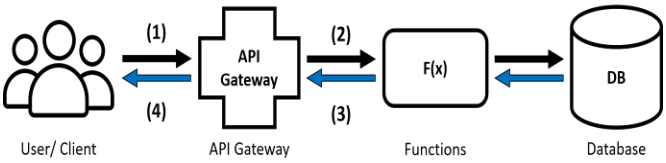
이 논문의 구성은 다음과 같다. 2 장에서는 해당 실험과 관련된 연구와 기술들을 서술한다. 3 장에서는 해당 실험 환경과 방법들을 소개하고, 4 장에서는 결

과에 따른 풀링 성능을 분석한다. 마지막으로 5 장에서는 결론과 기대효과를 서술한다.

2. 관련 연구

2.1 Serverless 컴퓨팅

Serverless 컴퓨팅은 클라우드 기반 플랫폼으로, 개발자가 서버와 인프라를 직접 관리할 필요가 없다. 클라우드 공급자가 하드웨어 관리와 서버 프로비저닝을 처리하며, 개발자가 코드 작업에만 집중할 수 있게 해준다. 이벤트가 발생하면 Serverless 플랫폼은 해당 이벤트에 대응하는 함수를 자동으로 실행한다. 이때 컨테이너가 생성되고 함수는 해당 컨테이너 내에서 실행하게 된다, 함수에 대한 요청량에 따라 컨테이너 수를 자동으로 스케일링 함으로써 탄력성과 가용성을 보장한다. Serverless 컴퓨팅은 pay as you go 모델을 채택한다. 함수가 실행되는 동안만 비용이 발생하게 되므로 사용량만큼만 결제하면 된다. 따라서 사용하지 않을 때에도 비용이 청구될 수 있는 가상머신과 비교하면, 비용 측면에서 매우 효율적이다. (그림 1)은 Serverless 컴퓨팅에서 대략적인 서비스 함수 호출 과정을 나타낸다.



(그림 1) Serverless 컴퓨팅 서비스 함수 호출 과정

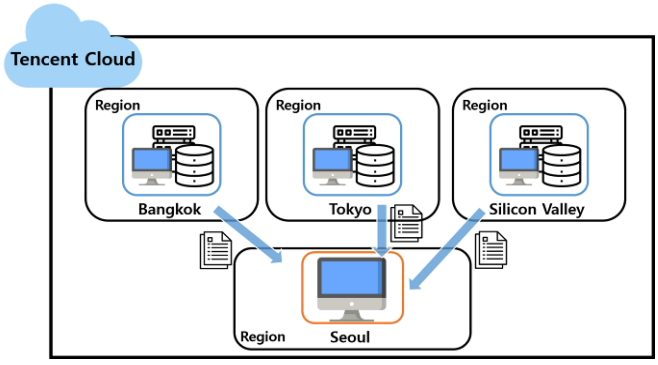
(1)에서는 클라이언트가 Serverless 함수를 호출할 때 요청을 생성하고 이를 API Gateway 로 전송된다. (2)에서 API Gateway 는 요청된 Serverless 함수를 식별하고 실행 될 함수를 호출한다. (3)에서는 호출 된 함수가 클라이언트가 요청한 작업을 수행한다. 이 때, 함수는 필요한 데이터베이스, 외부 API 와 저장소 등의 리소스에 액세스 할 수 있다. 함수 실행을 마치면 결과를 API Gateway 에 전달한다. (4)에서 API Gateway 는 전달받은 함수 결과를 클라이언트에게 넘겨준다.

2.2 Cold Start

Cold start 는 Serverless 컴퓨팅에서 발생되는 현상이다. 함수가 처음으로 실행되거나 활동이 많지 않은 함수가 다시 활성화될 때 발생한다. Cold start 에서는 해당 함수를 처리하기 위해서 컨테이너 초기화 및 활성화 작업을 거치게 된다. Cold start 를 겪게 되면 함수의 응답 시간이 증가 되고 사용자 입장에서 많은 딜레이를 초래할 수 있다. Cold start 는 Serverless 환경에서 주요한 고려 사항 중 하나로, 최근에는 cold start 를 완화하는 기법들이 많이 연구되고 있다 [4] [5]. Cold start 에서 함수 실행에 필요한 컨테이너 이미지를 원격으로부터 풀링해야 한다면 cold start 의 소요 시간은 대폭 증가하게 된다. 따라서 이미지 풀링 비용을 줄이면 cold start 의 딜레이를 최소화 할 수 있다.

3. 실험 환경 및 과정

3.1 실험 환경

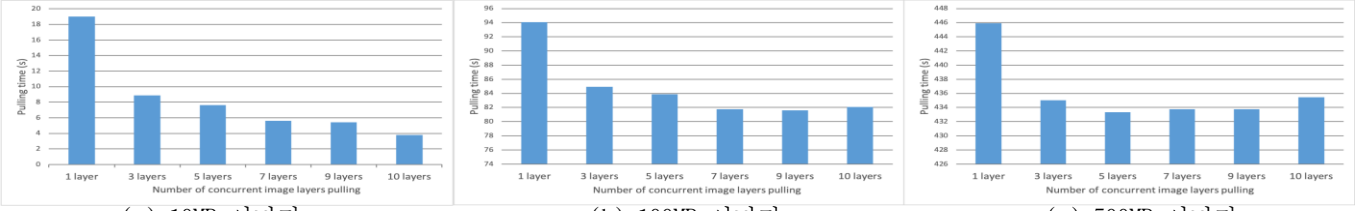


(그림 2) Tencent Cloud 가상머신 구성도

(그림 2)는 클라우드 플랫폼인 Tencent Cloud 를 이용한 가상머신 구성도를 나타낸다. 먼저 서울 지역에서 클라이언트 역할을 하는 가상머신을 생성했다. 클라이언트로부터 지리적으로 가까운 순서대로 도쿄, 방콕 그리고 가장 먼 실리콘밸리 지역에 각각 레지스트리 서버 역할을 수행할 동일한 사양의 가상 머신을 생성했다. 실험에 사용 된 가상 머신의 CPU 는 Intel Xeon Cascade Lake 8255C 로 사용했다. 메모리는 2GB 를 지니고 있고 대역폭은 모두 10Mbps 으로 적용하였다. 운영체제는 Ubuntu Server 22.04 LTS 64bit 를 사용하였다.

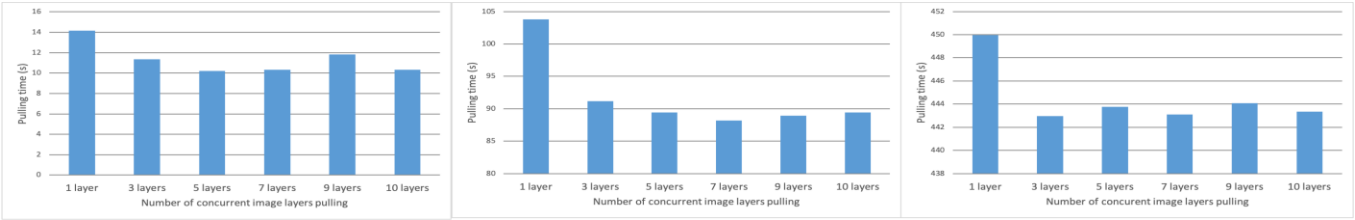
3.2 실험 과정

Docker registry 를 활용해서 도쿄, 방콕, 실리콘밸리 지역에 사설 레지스트리를 구축하였다. 크기가 10MB, 100MB 그리고 500MB 인 이미지를 빌드하였고 이들은 각각 1MB, 10MB, 50MB 크기로 이루어진 10 개의 이미지 레이어로 구성되어있다. 해당 이미지들을 각 지역의 사설 레지스트리에 배포하였다. 서울 지역에 생성한 가상머신이 레지스트리를 구축한 지역들로부터 다양한 크기의 이미지를 풀링하고 해당 소요시간을 측정했다. 측정하는 방법은 리눅스의 명령어인 'time docker pull' 을 이용했다. Docker 는 이미지 레이어 동시 풀링 개수를 기본으로 3 개로 지정 되어 있다. 이를 1,3,5,7,9,10 개로 각각 변경 하면서 이미지



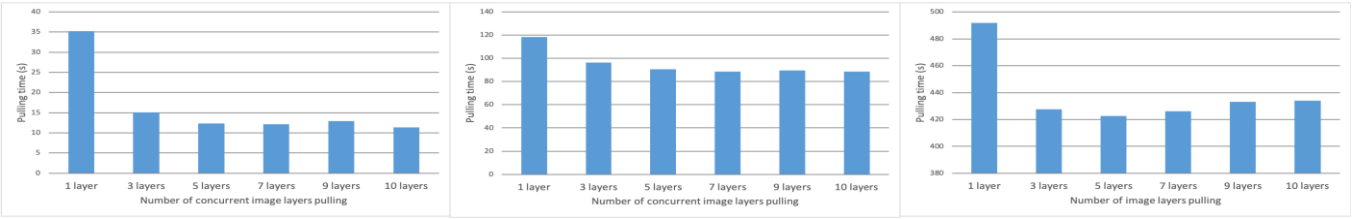
(a) 10MB 이미지 (b) 100MB 이미지 (c) 500MB 이미지

(그림 3) 일본 지역의 레지스트리로부터의 이미지 풀링 소요시간



(a) 10MB 이미지 (b) 100MB 이미지 (c) 500MB 이미지

(그림 4) 방콕 지역의 레지스트리로부터의 이미지 풀링 소요시간



(a) 10MB 이미지 (b) 100MB 이미지 (c) 500MB 이미지

(그림 5) 실리콘밸리 지역의 레지스트리로부터의 이미지 풀링 소요시간

를 풀링한다. 실험은 각 설정 값에 대해 10회씩 반복하여 수행했다. 또한 특정 시간대에 트랙픽이 급증할 수 있다는 것을 고려해, 서로 다른 시간대에 해당 과정을 2번씩 진행하고 전체 풀링 시간의 평균을 구했다. 동일한 설정에 대해서 실험을 반복 할 때, 정확한 측정을 위해 이전에 풀링했던 이미지 및 캐시 데이터를 삭제하고 풀링을 수행하였다.

4. 이미지 배치 및 동시 풀링 전략에 따른 성능 분석

서울에 위치한 클라이언트가 다른 지역에 있는 레지스트리로부터 크기가 각각 10MB, 100MB 그리고 500MB 인 이미지를 풀링한다. (그림 3), (그림 4) 그리고 (그림 5)는 이미지 레이어 동시 풀링 개수에 따른 도쿄, 방콕, 실리콘밸리로부터의 이미지 전체 풀링 소요시간을 나타낸 그래프이다. 그래프의 가로축은 이미지 레이어 동시 풀링 개수를 나타내고, 세로축은 풀링에 소요된 시간을 나타낸다. 해당 실험으로부터 공통으로 발견된 점은 이미지를 원격으로부터 풀링할 때, 하나의 이미지 레이어를 단독으로 풀링하는 것 보다 다수의 이미지 레이어를 동시적으로 풀링을 하는 것이 소요시간 면에서 크게 개선된 모습을 보였다는 것이다. 이는 단독으로 이미지 레이어를 풀링 할 경우, 대역폭을 충분히 활용하지 못하면서 레이어가 순차적으로 처리되기 때문으로 추정된다. 하나의 레이어만 풀링 하더라도 대역폭이 해당 풀링에 전부 활용되지 않을 수 있다. 남겨진 대역폭을 활용하여 다른 레이어를 동시에 풀링 할 수 있지만 레이어가 단독으로 받아짐으로써 남겨진 대역폭 자원은 낭비되는 것으로 추정된다. 이는 전체 이미지 풀링시간이 증가되는 주요 원인인 것으로 보인다.

대부분의 실험에서 이미지 레이어 동시 풀링 개수를 5 혹은 7 개로 설정했을 때 풀링 성능이 가장 최적으로 개선되었다. 레이어를 순차적으로 받지 않고 다수의 이미지 레이어를 동시에 풀링 함으로써 대역폭을 효과적으로 활용하고 낭비를 최소화한 것으로 보인다. 하지만 동시 풀링 개수를 그 이상으로 증가 시키면 성능 개선이 크게 나타나지 않거나 오히려 악화되었다. 일정 수준 이상으로 동시에 많은 풀링 작업을 수행하면 대역폭 한계에 도달하는 현상이 나타나는데 이는 네트워크 병목 현상을 발생시킬 수 있다. 병목 현상으로 인해 동시 풀링 개수를 증가 시켜도 성능 개선이 미미하거나 오히려 악화 된 것으로 보인다.

일본으로부터 10MB 이미지를 풀링할 때 동시 풀링 개수를 늘릴수록 한계점 없이 성능이 계속 개선되는 것을 확인했다. 이는 지리적으로 가까운 레지스트리로부터 작은 크기의 이미지를 풀링할 때, 데이터 경

로가 짧고 네트워크 지연이 감소하므로 이미지 레이어들이 빠르게 전송된다. 이러한 상황에서는 네트워크 병목 현상이나 대역폭 제한의 영향을 상대적으로 적게 받으므로, 계속해서 동시 이미지 레이어 풀링 개수를 증가시켜도 성능이 지속적으로 개선되는 것으로 예측된다.

<표 1>은 각 상황 별로 동시 풀링 개수를 조절함에 따라서 풀링 속도가 가장 빠른 경우와 느린 경우의 차이를 나타낸 것이다. <표 1>를 통해 이미지 크기가 작을수록 동시 풀링 개수가 풀링 속도에 큰 영향을 미친다는 것을 확인했다. 작은 크기의 이미지를 풀링할 때, 네트워크 대역폭 제한과 병목현상의 영향을 비교적 적게 받으므로 동시 풀링 개수를 조절함으로써 더 뚜렷한 성능개선을 보인 것으로 생각된다. 방콕으로부터 풀링 개수 조절에 따른 성능차이가 도쿄, 실리콘밸리 경우와 비교하면 유독 낮은 것을 확인했다. 이는 일본과 미국이 많은 인터넷 백본을 보유하고 네트워크 인프라가 우수한 국가로써 대역폭을 안정적으로 확보 할 가능성이 높지만 방콕에서는 인프라가 상대적으로 적을 수 있기 때문에 동시 풀링 개수를 조정해도 대역폭을 충분히 확보하지 못하여 크게 개선이 되지 않은 것으로 보인다.

<표 1> 동시 풀링 개수에 따른 최고 및 최저 풀링 성능 차이

이미지 크기 / 지역	10MB	100MB	500MB
도쿄	405.39%	15.21%	2.89%
방콕	38.56%	17.68%	0.39%
실리콘밸리	189.91%	33.38%	16.39%

5. 결론

본 논문에서는 도커를 활용해서 이미지 레이어 동시 풀링 개수가 이미지 풀링 성능에 영향을 미치는지 분석했다. 동시 이미지 레이어 풀링 개수를 조절하면서 여러 지역에 있는 레지스트리로부터 다양한 크기의 이미지를 풀링했다. 해당 실험에서, 하나의 이미지 레이어를 단독으로 받는 것보단 5 개에서 7 개의 레이어를 동시적으로 풀링하는 것이 효율적인 것을 확인했다. 이는, 이미지 레이어를 단독으로 받을 경우 대역폭을 충분히 활용하지 못한 것으로 추정된다. 반대로 너무 많은 개수의 레이어를 동시적으로 풀링하면 성능 개선이 제한되거나 오히려 악화되었고, 대역폭 제한과 병목현상으로 인한 결과로 보인다. 예외적으로, 지리적으로 가까운 레지스트리로부터 크기가 작은 이미지를 풀링할 때는 동시 풀링 개수를 계속 증가시켜도 지속적으로 성능개선이 된 모습을 보였다. 해당 실험에서는 이미지 레이어 동시 풀링 개수를 조절함으로써 이미지 풀링 속도를 향상시킬 수 있다는 것을 확인했다. 이는 추후에 Serverless 컴퓨팅 환경에서 발생되는 cold start 완화 연구에 기여를 할 것으로 기대된다.

Acknowledgement

이 논문은 2023 년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임 (2022R1I1A1A01063551)

이 논문은 2023 년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No.2022-0-01198, 융합보안대학원(고려대학교))

참고문헌

[1] https://www.redhat.com/ko/topics/cloud-native-apps/what-is-serverless

[2] P. Vahidinia, B. Farahani and F. S. Aliee, "Cold Start in Serverless Computing: Current Trends and Mitigation Strategies," 2020 International Conference on Omni-layer Intelligent Systems (COINS), Barcelona, Spain, 2020, pp. 1-7, doi: 10.1109/COINS49042.2020.9191377.

[3] Wang, Ao, et al. "{FaaSNet}: Scalable and fast provisioning of custom serverless container runtimes at alibaba cloud function compute." 2021 USENIX Annual Technical Conference (USENIX ATC 21). 2021.

[4] Vahidinia, Parichehr, Bahar Farahani, and Fereidoon Shams Aliee. "Mitigating cold start problem in serverless computing: a reinforcement learning approach." IEEE Internet of Things Journal 10.5 (2022): 3917-3927.

[5] Lee, S.; Yoon, D.; Yeo, S.; Oh, S. Mitigating Cold Start Problem in Serverless Computing with Function Fusion. Sensors 2021, 21, 8416. https://doi.org/10.3390/s21248416