도커를 이용한 온라인 저지 시스템 자원 제한 방법

김영훈[1], 한상곤[2], 우균[3]
[1,2,3]부산대학교 정보컴퓨터공학부
{yonghun83, sangkon, woogyun}@pusan.ac.kr

A Resource Restriction Method for Online Judge Systems Using Docker

Yeonghun Kim[1], Sangkon Han[2], Gyun Woo[3]
[1,2,3]Dept. of Information Convergence Engineering, Pusan National University

요 약

온라인 저지 시스템은 학습자가 제출한 코드를 평가하기 위해 많은 시스템 자원을 사용한다. 학습자의 코드를 평가하는 방법 중 하나인 코드 효율성 측정은 시간복잡도를 기반으로 평가하기 때문에 대량의 인수를 입력 데이터로 사용한다. 본 연구에서 컨테이너 기술인 도커의 컨트롤 그룹을 활용하여 CPU 자원을 제한한다. 이를 통해 기존에 사용한 데이터보다 적은 데이터를 이용하여 코드 효율성을 측정하는 방법을 제안한다. 제안된 방법에 따르면 최단 경로 계산 문제에서 데이터 크기를 60%, 측정 시간을 33.3% 절감할 수 있는 것으로 나타났다.

1. 서론

정보통신기술이 다양한 분야에 적용됨에 따라 소프트웨어 개발 직무의 관심이 증가하고 있다. 이에 따라 프로그래밍 능력 평가 기준이 자료구조와 알고리즘을 사용하여 문제 해결에 필요한 코드를 구현하는 능력으로 바뀌었다. 코드 구현 능력을 검증하기 위해서 시간복잡도 기반의 검증 방법을 사용하고 있다.

시간복잡도를 기반으로 코드를 평가하면 대량의 입력 데이터를 사용한다. 이는 메모리 용량뿐만 아니라 계산 비용도 증가시킨다. 따라서 코드 평가에 하드웨어 성능이 뒷받침되어야 하는 문제가 있다.

본 논문은 온라인 저지 시스템에서 시간복잡도를 기반으로 학습자의 코드를 검증하는 방법의 단점 보완을 위해 도커(Docker)를 사용해서 자원을 제한하는 방법을 제안하고자 한다. 컨테이너의 CPU 자원 할당량을 제한하면 기존보다 적은 데이터를 사용해 시간복잡도 기반의 코드 효율성을 검증할 수 있다.

본 논문의 구성은 다음과 같다. 2절에서는 온라인 저지 시스템(Online judge system)과 컨트롤 그룹(Cgroups: Control groups), 도커를 알아본다. 3절에서는 자원 제한 시스템 구현 방법을 설명한다. 4절에서는 자원 제한 시스템을 사용해 CPU 자원 할당량에 따른 실행 시간차를 확인한다. 5절에서는 제안 시스템의 한계점을 살펴보고, 6절에서 결론짓는다.

2. 관련 연구

2.1 온라인 저지 시스템

온라인 저지 시스템은 대학 강의나 프로그래밍 학습 사이트에서 학습용으로 사용하거나 프로그래밍 대회에서 코드 평가를 위해 사용된다. 학습자가 문제의 정답 코드를 제출하면 코드 실행을 통해 나온 출력 데이터와 정답 데이터를 비교해 일치하면 성공으로 판단하는 동적 평가기법 중 일반 평가를 이용한다[1].

최근 온라인 저지 시스템의 평가 방법과 관련하여 다양한 시도를 하고 있다. 입력 데이터의 크기에 따라 부분 점수를 부여하는 방법, 코드 실행 단계에서 발생하는 오류의 중요도에 따라 부분 점수를 부여하는 방법이 대표적인 예라고 할 수 있다[2,3].

2.2 컨트롤 그룹

컨트롤 그룹은 프로세스가 사용할 수 있는 자원을 제한하고 격리하는 역할을 한다. 컨트롤 그룹 v1은 리소스(resource)를 기준으로 그룹을 나누고, v2에서 워크로드(workload)를 기준으로 그룹을 나눈다[4,5]. 최신 리눅스 계열 운영체제에서 별도의 설치 없이 사용할 수 있으며 v2가 기본값으로 적용된다.

도커는 컨테이너에서 실행되는 프로세스를 설정한 자원만 사용하도록 컨트롤 그룹을 활용하고 있다. 도커에서 설정할 수 있는 옵션은 컨테이너의 CPU 자원 할당량, CPU 스케줄링에 할당할 시간, 특정 CPU 코어를 지정해서 사용하도록 제한할 수 있으며, CPU 사용 가중치, 할당 메모리, 스왑 메모리 등이 있다. 도커에서 관리하는 각각의 컨테이너는 컨트롤 그룹을 사용해서 사용량을 제한할 수 있다.

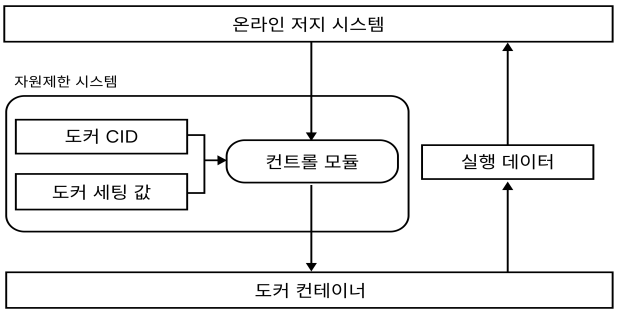
2.3 도커

도커는 하이퍼바이저(Hypervisor)를 사용하지 않고, 도커 자체 엔진을 이용하여 컨테이너를 생성하고 실행한다. 이에 따라 컨테이너를 생성하는 데 걸리는 시간이 획기적으로 짧아지며, 빠르게 시작하고 종료할 수 있다[6]. 도커 컨테이너를 사용하면 여러 개의 격리된 실행 환경을 쉽게 만들고 관리할 수 있어, 문제를 풀거나 알고리즘 테스트하는 온라인 저지 시스템에서 매우 효과적이다.

온라인 저지 시스템의 활용도를 높이기 위해서 컨테이너 기술인 도커를 사용해서 오픈소스 온라인 저지 시스템인 HUSTOJ와 DOMjudge를 도커 기반으로 구축한 사례가 있다. 도커를 이용하여 간편하게 시스템을 구축할 수 있게 됨에 따라서 온라인 저지 시스템이 다양한 곳에서 활용되고 있다.

3. 시스템 설계 및 구현

이 장에서는 도커를 이용한 온라인 저지 시스템의 자원 제한 방법을 소개한다. 도커 컨테이너는 온라인 저지 시스템과 저장소를 공유하는 기능인 바인드 마운트(bind-mounts)를 이용해 별도의 통신 없이 코드를 실행할 수 있다. 본 논문에서 제안하는 시스템 구조는 그림 1과 같다.



(그림 1) 온라인 저지 자원 제한 시스템 구조

온라인 저지 시스템은 학습자가 코드를 제출하면 컨트롤 모듈에 학습자의 코드 위치와 문제에서 설정된 CPU 자원 할당량을 전달한다. 컨트롤 모듈은 전달받은 CPU 자원 할당량으로 컨테이너의 설정값을 변경하고, 학습자의 코드를 검증하는 명령을 컨테이너에 전달한다. 명령을 전달받은 컨테이너는 준비된 입력 데이터로 학습자의 코드를 실행한다. 실행 결과는 공유된 디렉토리에 저장된다. 다음으로 온라인 저지 시스템에서 실행 데이터가 정답 데이터와 일치하는지, 코드를 실행하는데 제한된 시간을 초과하였는지 판별하여 점수를 부여한다.

4. 실험 및 결과 분석

4.1 실험 환경 구성

표 1은 실험에 사용한 호스트와 컨테이너의 환경을 나타낸다. 실험에 사용될 서버는 AWS(Amazon web service)에서 t2.xlarge 인스턴스를 사용한다. 이 인스턴스는 최대 속도가 3.0GHz인 인텔 스케일러블 프로세서에서 4개 코어를 사용할 수 있다. 호스트 OS는 Ubuntu에 도커와 코드 자동 채점 시스템 neoESPA를 설치한다[7]. 컨테이너는 Ubuntu 및 Python 3.11.5 버전을 사용한다.

<표 1> 실험에 사용한 환경

구분	Host	Container
CPU Core	4	4
Memory	16 GB	10 GB
OS Version	Ubuntu 22.04	Ubuntu 22.04
Tools	Docker 24.0.5, neoESPA	Python 3.11.5

4.2 실험 설계

실험은 파스칼 삼각형 문제와 최단 경로 계산 문제 두 가지를 이용하여 진행한다. 각 문제에서 서로 다른 시간복잡도의 코드를 준비한다. 준비된 코드를 CPU 자원 할당량을 조절한 조건에서 실행 시간(wall-clock time)을 측정한다. CPU 자원 할당량은 CPU 프로세서를 사용할 수 있는 시간을 나타낸다. 이 값은 100밀리초(milliseconds)를 곱함으로써 컨테이너가 사용할 수 있는 CPU 처리 시간을 나타낸다. 실행 시간은 코드를 3회 반복 실행하여 얻은 시간의 평균을 기록한다.

첫 번째 실험은 파스칼의 삼각형에서 N열의 K행 값을 찾는 문제이다. 입력 데이터는 N이 짝수이고, K가 N의 절반 값으로 입력한다. 문제에 사용된 코

드는 동적 프로그래밍 방식을 사용하여 값을 계산하는 시간복잡도 $O(N \cdot K)$ 코드와 이차원 배열에 파스칼의 삼각형을 생성하여 값을 출력한 시간복잡도 $O(N^2)$ 코드를 사용한다.

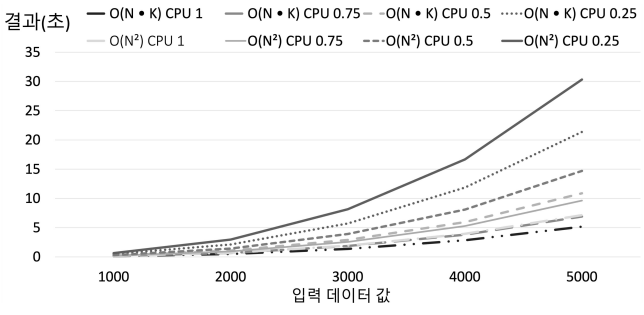
두 번째 실험은 자신을 제외한 모든 정점을 간선으로 연결한 그래프에서 1번 정점에서부터 N번 정점까지의 최단 경로를 구하는 문제이다. 입력 데이터는 각 정점에서 해당 정점과 간선으로 연결된 다른 정점의 리스트를 Python의 딕셔너리 형식으로 입력한다. 문제 해결에는 다익스트라(Dijkstra) 알고리즘을 기반으로 구현한 두 가지 코드를 사용한다. 하나는 우선순위 큐를 활용하여 시간복잡도 $O(E+ Vlog(V))$ 코드이며, 다른 하나는 배열과 반복문을 사용한 시간복잡도 $O(V^2)$ 코드이다.

4.3 실험 결과 및 분석

표 2는 첫 번째 실험에서 CPU 자원 할당량별 시간복잡도가 다른 두 코드의 실행 시간을 나타낸다. 그림 2는 표 2의 실행 시간을 그래프로 나타낸 것이다. 그림 2에서 CPU 자원 할당량이 감소함에 따라 코드 실행 시간 차이가 발생하는 것을 알 수 있다.

<표 2> 파스칼의 삼각형 문제 코드 실행 시간

시간복잡도	N	CPU 자원 할당량			
		1.00	0.75	0.50	0.25
$O(N \cdot K)$	1000	0.12	0.14	0.22	0.49
	2000	0.51	0.68	1.09	2.13
	3000	1.38	1.85	2.89	5.70
	4000	2.85	3.80	5.96	11.85
	5000	5.16	6.90	10.88	21.37
$O(N^2)$	1000	0.16	0.20	0.29	0.64
	2000	0.70	0.93	1.43	2.97
	3000	1.92	2.55	3.91	8.15
	4000	3.95	5.28	8.09	16.66
	5000	7.15	9.62	14.70	30.33



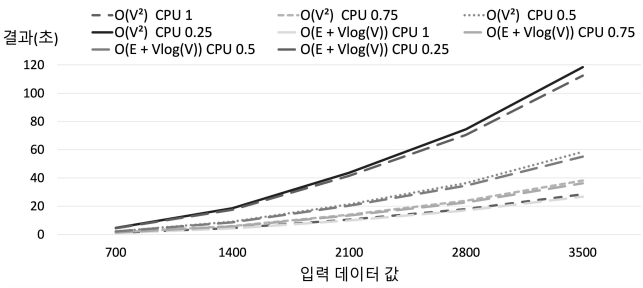
(그림 2) 파스칼의 삼각형 문제 실행 시간 그래프

CPU 자원 할당량을 제한하지 않은 경우 4,000개의 입력 데이터를 입력했을 때 두 코드의 실행 시간은 3.95초와 2.85초로 1.1초 차이가 발생한다. 따라서 해당 알고리즘의 시간제한을 3초로 설정하면 시간복잡도를 기준으로 어떤 코드가 효율성이 높은지 평가할 수 있다. 또한 CPU 자원 할당량을 0.5로 설정하면 입력 데이터(N)를 4,000개에서 3,000개로 줄일 수 있다.

표 3은 두 번째 실험에서 CPU 자원 할당량별 시간복잡도가 다른 두 코드의 실행 시간을 나타낸다. 또한, 그림 3은 표 3의 실행 시간을 그래프로 나타낸 것이다. 그림 2에서 확인한 CPU 자원 할당량이 감소함에 따라 두 코드의 실행 시간 차이가 발생하지만, 앞선 실험과 달리 입력 데이터 크기(N)가 실행 시간에 큰 영향을 미치지 않음을 확인하였다.

<표 3> 최단 경로 계산 문제 코드 실행 시간

시간복잡도	N	CPU 자원 할당량			
		1.00	0.75	0.50	0.25
$O(E+ Vlog(V))$	700	1.05	1.39	2.11	4.39
	1400	4.19	5.65	8.51	17.49
	2100	9.91	13.31	20.35	41.53
	2800	16.91	22.64	34.46	70.40
	3500	26.74	36.19	55.06	112.40
$O(V^2)$	700	1.11	1.46	2.25	4.60
	1400	4.42	5.95	9.04	18.49
	2100	10.51	14.01	21.30	43.59
	2800	17.88	23.81	36.35	74.39
	3500	28.39	38.14	58.39	118.39



(그림 3) 최단 경로 계산 문제 실행 시간 그래프

CPU 자원 할당량을 제한하지 않은 경우 3,500개의 입력 데이터를 입력했을 때 두 코드의 실행 시간을 고려하면 시간제한을 27초로 설정할 수 있다. 반면, 자원 제한을 통해 CPU 자원 할당량을 0.25로 설정하면 입력 데이터(N)를 3,500개에서 1,400개로 줄일 수 있으며 시간제한을 18초로 설정할 수 있다. 알고리즘의 특성상 입력 데이터의 크기가 큰 영향을

주지 않기 때문에 CPU의 할당량을 조절하여, 더 적은 자원을 활용해 코드의 효율성을 검증할 수 있다.

CPU 자원 할당량을 조절하여 입력 데이터 크기와 시간을 효과적으로 절감할 수 있음을 확인하였다. 파스칼의 삼각형 문제에선 입력 데이터 크기를 4,000개에서 3,000개로 약 25% 정도 줄였다. 최단 경로 계산 문제에서 입력 데이터 크기를 3,500개에서 1,400개로 약 60% 정도 줄였으며 시간제한을 통해서 검증 시간을 27초에서 18초로 약 33.3% 줄일 수 있었다.

두 코드의 시간복잡도를 비교하기 위해서는 많은 연산량이 필요하며, 이러한 연산량을 빠르게 처리하기 위해서는 CPU 자원이 필수적이다. 그러므로 두 코드의 시간 차이가 CPU 자원 할당량을 조절함으로 더 크게 드러나게 된다. 이에 따라 추가된 시간 차이만큼 더 적은 양의 입력 데이터를 사용하여 코드 성능을 판별하는 데 충분한 결과를 얻을 수 있다.

5. 고찰

본 논문에서 제안하는 방법인 CPU 자원 할당량을 작게 설정하여 코드의 효율성을 측정하면 실행 시간이 항상 일정하지 않을 수 있다. 이러한 불규칙한 결과는 프로세스가 실행될 때 서버의 상태에 따라 콜드 스타트(cold-start) 현상이 발생하기 때문이다. 따라서, 일반 평가를 먼저 시행한 후 효율성 측정을 동작하거나 프리워밍(pre-warming)을 하는 등 추가적인 동작이 필요하다.

또한, 여러 컨테이너에서 병렬로 채점하여 코드의 효율성을 판단할 수 없다. 도커의 컨테이너는 호스트 OS로부터 자원을 할당받아 동작하기 때문에 코드 실행에 필요한 CPU를 확보하기 위한 경쟁 상태가 된다. 이러한 문제를 해결하기 위한 연구가 추가로 필요하다.

6. 결론

이 논문에서는 도커를 이용하여 CPU 자원 할당량을 조절하여 시간복잡도를 사용해서 코드의 효율성을 측정할 방법을 제안하였다. 특히 시간 차이가 1초 미만인 경우 CPU 자원 할당량을 조정하면 시간복잡도를 기반으로 효율성을 확인할 수 있었다. 실험 결과로 제안한 방법을 이용하여 파스칼의 삼각형 문제에서 검증 시간을 절감할 수 없었지만, 입력 데이터 크기를 25% 절감할 수 있었으며 최단 경로 계산 문제에서 입력 데이터 크기를 60%, 검증 시간을 33.3% 절감할 수 있었다.

하지만 시간복잡도 $O(N)$과 $O(Nlog(N))$ 코드의 효율성 측정은 1코어 수준의 CPU 자원 할당량에서는 구분할 수 없는 문제가 있다. 따라서 이러한 시간복잡도를 구분하기 위해서는 1코어 수준보다 더 낮은 CPU 자원 할당량이나 CPU 스케줄러 기간과 같은 추가적인 설정을 이용하여 두 시간복잡도에서 효율성을 측정할 수 있는지에 대한 연구가 추가로 필요하다.

참고문헌

[1] 한상곤, 류사오, 우균, "교육 수준 및 목적을 기반으로 한 온라인 저지 시스템의 평가 방법 및 분류", 정보과학회 컴퓨팅의 실제 논문지, Vol.28, No.10, pp. 487-492, 2022.

[2] 정종광, 송정범, 이태욱, "한국정보올림피아드의 현황 분석 및 개선 방안", 한국컴퓨터교육학회 학술발표대회논문집, Vol.13, No.1, pp. 187-196, 2009.

[3] 이재영, 김자미, 이원규, "텍스트 기반 프로그램 평가에서 부분점수 구성에 관한 고찰", 컴퓨터교육학회 논문지, Vol.22, No.2, pp. 29-38, 2019.

[4] Paul Menage, “Control Group v1”, [Online]. URL: https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt, 2004, last visited on Sep 2023.

[5] Tejun Heo, “Control Group v2”, [Online]. URL: https://www.kernel.org/doc/Documentation/cgroup-v2.txt, 2015, last visited on Sep 2023.

[6] Dirk Merkel, “Docker: Lightweight Linux Containers for Consistent Development and Deployment,” *Linux Journal*, Vol. 2014, No. 239, pp. 76-91, 2014.

[7] 천준석, 김연어, 류샤오, 왕인서, 변석우, 우균, "QuickCheck를 사용한 neoESPA 입출력 데이터 생성 시스템", 한국정보과학회 학술발표논문집, Vol.2021, No.6, pp. 1565-1567, 2021.