저지연 실시간성을 보장하는 코딩 테스트 플랫폼 설계

조현태[1] · 백병재[2] · 이호웅*

[1]호서대학교 · [2]호서대학교 · *호서대학교

Design of a Low-Latency Real-Time Coding Test Platform

Hyeon-tae Cho[1] · Byeong-jae Baek[2] · Ho-woong Lee*

[1]Hoseo University · [2]Hoseo University · *Hoseo University

E-mail : whgusxo2055@gmail.com / bottlebaek5231@gmail.com / always14@hoseo.edu

요 약

온라인 코딩 테스트 플랫폼은 코드 제출이 특정 시점에 집중되면 처리 대기열 증가와 응답 지연이 발생한다. 본 연구는 제출과 채점 작업을 메시지 큐로 분리하고, 메시지 큐 적재량에 따라 채점 워커를 동적으로 확장하는 Auto-Scaling 구조를 설계한다. 이를 통해 제출 급증 상황에서 처리 지연을 완화하고 응답성과 확장성을 높이는 방안을 제시한다.

ABSTRACT

Online coding test platforms face an increasing phenomenon where code submissions are concentrated at specific points during processing, leading to delayed responses. This study designs an auto-scaling structure that separates submissions and grading messages into queues and dynamically expands task operations based on message volume. Through this, it is necessary to accommodate processing delays in situations requiring submission while scaling response performance.

키워드

Online Coding Test Platform, Message Queue, Auto-Scaling, Dynamic Worker Scaling, Low-Latency Processing

Ⅰ. 서 론

온라인 코딩 테스트 플랫폼은 교육, 채용, 경진대회 등에서 활용되며 빠른 채점 결과 제공이 중요하다[1][2]. 실제 운영에서는 제출이 특정 시점에 집중되어 채점 요청이 누적되고 응답 지연이 발생한다. 기존 연구도 이러한 부하를 처리하기 위한 제출과 채점의 분리 구조를 제시한다[1]. 핵심은 제출 서버와 채점 워커를 분리하는 것에 그치지 않고, 그 사이의 미처리 작업량을 확장 제어에 직접 반영하는 것이다.

최근 연구에서 CPU와 메모리 외에도 애플리케이션 수준 지표가 확장 판단에 중요함을 보인다[4][5][6]. 메시지 큐 적재량은 아직 처리되지 않은 채점 수요를 직접 반영하므로, 제출 집중 상황에서는 자원 사용률보다 더 적절한 제어지표가 된다[4][5]. 본 연구는 제출과 채점 사이에 메시지 큐를 두고, 메시지 큐 적재량에 따라 채점 워커를 수평 확장하는 구조를 제안한다. 이 구조는 제출과 채점의 분리, 애플리케이션 수준 지표 기반 확장을 결합해 채점 병목을 완화하도록 설계된다.

Ⅱ. 관련연구

2.1 Online Judge 및 자동 채점 시스템

Online Judge 시스템은 사용자가 제출한 프로그램을 균일한 환경에서 컴파일, 실행, 평가하는 자동 채점 시스템으로 정의되어 왔으며, 교육, 프로그래밍 대회, 채용 등으로 활용 범위가 확장되어 왔다[2]. 최근에는 제출과 채점의 분리, 마이크로서비스 구성, 이벤트 기반 처리 구조를 갖는

* 교신저자

Cloud-Native Online Judge 시스템도 제안되고 있다[1]. 또한 신뢰할 수 없는 사용자 코드를 안전하게 실행하기 위한 격리 실행 환경과 샌드박스 안전성은 Online Judge 계열 시스템의 핵심 설계 요소로 논의된다[3].

2.2 Cloud-Native Auto-Scaling

Cloud-Native Auto-Scaling 연구는 컨테이너 오케스트레이션 환경에서 워크로드 변화에 따라 인스턴스 수를 조정하는 문제를 다룬다. 기존 연구는 수평 확장이 Cloud-Native 환경의 핵심 자원 관리 기술이며, CPU와 메모리와 같은 자원 지표뿐 아니라 애플리케이션 수준 지표도 중요한 확장 판단 기준이 될 수 있음을 정리하였다[4]. 또한 컨테이너 Auto-Scaling에서는 어떤 성능 지표를 선택하느냐가 서비스 품질 유지에 직접 영향을 주며, 쿠버네티스 환경에서는 사용자 정의 지표를 포함한 다양한 지표 기반 수평 확장이 가능하다는 점이 보고되었다[5][6]. 그러나 기존 연구는 Online Judge 시스템의 채점 구조와 Cloud-Native Auto-Scaling을 각각 다루는 데 초점이 있으며, 두 관점을 코딩 테스트 플랫폼의 제출 집중 상황에 직접 연결한 논의는 제한적이다. 이에 본 연구는 메시지 큐 적재량을 애플리케이션 수준 지표로 해석하고, 이를 코딩 테스트 플랫폼의 채점 워커 확장 문제에 적용한다.

III. 제안 시스템 설계

3.1 설계 관점

본 연구는 제출과 채점 사이의 메시지 큐 적재량을 채점 워커 확장의 핵심 제어지표로 사용하는 구조를 제안한다. 저지연 실시간성은 제출 경로와 채점 경로의 단순 분리보다, 두 계층 사이에 누적된 미처리 채점 수요를 지속적으로 관측하고 그 값에 따라 채점 워커를 수평 확장하는 방식으로 지원하고자 한다.

3.2 제안 구조

제안 구조는 사용자, 제출 서버, 메시지 큐, 채점 워커 풀, 격리 실행 환경, 결과 저장소로 구성된다. 사용자가 코드를 제출하면 제출 서버는 제출 형식, 문제 식별자, 언어, 자원 제한 조건 등 기본 검증을 수행한 뒤 채점 작업을 메시지 큐에 등록하고 접수 응답을 반환한다. 이후 채점 워커는 메시지 큐에서 작업을 가져와 격리 실행 환경에서 컴파일, 테스트케이스 실행, 출력 비교, 자원 제한 검증을 순차적으로 수행한 뒤 결과를 저장한다. 이 구조에서 메시지 큐는 제출 서버와 채점 워커를 비동기적으로 연결하는 동시에, 제출과 채점 사이에 누적된 미처리 작업량을 관측하는 지점으로 기능한다.

제출 서버는 채점 완료 여부와 독립적으로 요청을 접수할 수 있고, 채점 워커 풀은 메시지 큐 적재량에 따라 독립적으로 확장될 수 있다. 또한 사용자 코드는 격리 실행 환경에서 처리되어야 하므로, 본 연구는 개별 채점 워커 내부의 병렬성을 높이기보다 채점 워커 수를 수평 확장하여 전체 처리량을 확보하도록 설계한다.

3.3 메시지 큐 적재량 기반 채점 워커 수평 Auto-Scaling

제안 구조에서 Auto-Scaling의 핵심 제어지표는 메시지 큐 적재량이다. CPU나 메모리 사용량은 이미 발생한 자원 소모를 나타내는 반면, 메시지 큐 적재량은 아직 처리되지 않은 채점 수요를 직접 반영한다. 따라서 제출 집중 상황에서는 메시지 큐 적재량을 기준으로 채점 워커 수를 조정하는 방식이 채점 대기시간 증가를 완화하는 데 적합하다.

수식 1. $\beta = \frac{D_{target}}{S_{avg}}$

목표 적재량 β는 수식 1과 같다. 목표 대기시간을 D_{target}, 채점 워커 1개의 평균 처리시간을 S_{avg}라 한다. 각 채점 워커가 한 시점에 하나의 채점 작업을 처리한다고 가정할 때, 채점 워커 1개가 목표 대기시간 내에 감당할 수 있는 적재 목표량이다. 이에 따라 목표 채점 워커 수는 수식 2와 같다.

수식 2. $W_d(t) = \min(W_{\max}, \max(W_{\min}, \lceil \frac{B(t)}{\beta} \rceil))$

시점 t에서 메시지 큐에 대기 중인 채점 작업 수를 $B(t)$, $W_{\min}$은 채점 워커 생성 지연을 줄이기 위해 항상 유지하는 최소 채점 워커 수이고, $W_{\max}$는 클러스터 자원, 격리 실행 환경, 결과 저장소의 처리 한계를 고려하여 허용하는 최대 채점 워커 수이다. 계산된 목표 채점 워커 수 $W_d(t)$는 시점 t에서 시스템이 유지해야 할 목표 상태를 의미한다. 현재 채점 워커 수를 $W(t)$라 할 때, $W_d(t) > W(t)$이면 채점 워커를 추가하고, $W_d(t) < W(t)$이면 최소 채점 워커 수 범위 내에서 축소한다. 또한 각 채점 워커가 동시에 점유하는 작업 수를 제한하여 격리 실행 환경 간 자원 간섭을 줄인다. 작업은 처리 완료가 확인된 이후에만 메시지 큐에서 제거되며, 결과 저장은 제출 식별자 기준으로 멱등 처리하여 재시도 과정에서 중복 반영이 발생하지 않도록 한다. 결과적으로 제안 구조는 메시지 큐 적재량을 채점 워커 수평 확장의 제어지표로 사용하여 제출 집중 상황에서 채점 대기시간 증가를 완화하도록 설계된다.

Ⅳ. 설계 타당성 및 기대 효과

4.1 저지연 실시간성 측면

본 연구에서 저지연 실시간성은 모든 제출에 대해 일정 시간 안에 채점을 완료한다는 의미가 아니다. 본 논문에서는 제출이 몰리는 구간에서도 채점 대기시간이 급격히 증가하지 않도록 제어하는 운영적 특성으로 정의한다. 제안 구조는 CPU나 메모리 사용량 대신 메시지 큐 적재량을 채점 워커 확장의 제어지표로 사용한다. 메시지 큐 적재량은 아직 처리되지 않은 채점 작업 수를 직접 나타내므로, 현재 시스템이 감당하지 못한 부하를 확인하는 데 적합하다. 또한 목표 적재량은 목표 대기시간과 평균 처리시간을 이용해 계산되므로, 확장 판단을 자원 사용률이 아니라 대기시간 목표와 연결할 수 있다. 따라서 제출 집중 구간에서 메시지 큐 적재량이 증가하면 채점 워커 수를 조정하여 채점 대기시간 증가를 완화할 수 있다.

4.2 확장성 및 운영 안정성 측면

코드 채점 과정은 격리된 실행 환경을 기반으로 하므로, 하나의 워커 안에서 병렬 처리 정도를 과도하게 높이면 자원 충돌과 실행 시간 불균형이 발생할 위험이 있다. 이에 본 연구에서는 워커 내부 병렬 처리 대신 채점 워커의 수를 수평적으로 늘리는 접근을 채택하였다. 이 방법으로 각 채점 작업의 환경을 독립적으로 보장하면서 시스템 전체 처리 용량을 안정화할 수 있게 구성하였다. 제안된 구조에서 메시지 큐는 일시적 제출 트래픽 집중을 완화하는 버퍼 역할을 수행한다. 채점 워커 풀은 큐 내 작업량에 연동되어 제출 서버와 독립적으로 규모를 조정하므로, 제출 처리 경로와 채점 처리 경로 간 의존성을 최소화할 수 있다. 또한 워커 장애나 일시적 처리 실패가 발생하더라도 재처리와 제출 식별자 기반의 멱등적 결과 저장을 통해 중복 반영을 방지하고, 시스템 안정성을 유지할 수 있다.

Ⅴ. 결 론

본 연구는 코딩 테스트 플랫폼에서 제출이 특정 시간대에 몰리는 상황으로 인한 채점 대기시간 상승 문제를 해결하고자, 메시지 큐 적재량을 채점 워커 확장의 주요 지표로 사용하는 구조를 제시하였다. 제안 구조는 제출 요청을 먼저 수용한 후 메시지 큐에 쌓인 미처리 채점 작업량을 기반으로 워커를 수평 확장함으로써 제출 폭증 시 대기시간을 줄이는 방향으로 설계되었다. 나아가 격리 실행이 필수인 자동 채점 시스템과 Cloud-Native 확장 기법을 융합하여 코딩 테스트 환경에 적합한 확장성을 가진 구조를 도출하였다. 특히 CPU, 메모리 사용률 같은 자원 중심 확장과 달리 실제 미처리 채점 수요를 반영한 지표를 사용함으로써 제출 집중 환경에 맞는 저지연 설계 방향을 제시하였다.

Ⅵ. 향후 과제

향후에는 제안 구조를 실제 Cloud-Native 환경에 구현하고, 제출 집중 시나리오에서 채점 대기시간, 채점 워커 확장 반응 시간, 메시지 큐 적재량 변화 등을 측정하여 정량적으로 검증할 필요가 있다. 또한 실제 운영 환경에서는 채점 워커의 기동 시간, 메시지 큐 관측 주기, 격리 실행 환경의 초기화 비용, 제출별 처리시간 분포 등이 전체 응답성에 영향을 줄 수 있다. 따라서 이후 연구에서는 메시지 큐 적재량뿐 아니라 평균 처리시간, 채점 워커 상태 정보, 문제 유형별 처리 비용 등을 함께 반영하는 확장 정책으로 발전시킬 수 있다. 이를 통해 제출 급증 상황에서 안정적인 저지연 응답성을 제공하는 코딩 테스트 플랫폼 구조를 구체화하고자 한다.

Acknowledgement

"본 연구는 2026년 과학기술정보통신부 및 정보통신기획평가원의 SW중심대학사업 지원을 받아 수행되었음"(2025-0-00040)

References

[1] G.-C. Pan, P. Liu, and J.-J. Wu, “A Cloud-Native Online Judge System,” Proc. IEEE 46th Annu. Comput., Softw., Appl. Conf. (COMPSAC), Virtual Conf., Jun.–Jul. 2022, pp. 1293–1298.

[2] S. Wasik, M. Antczak, J. Badura, A. Laskowski, and T. Sternal, “A Survey on Online Judge Systems and Their Applications,” ACM Comput. Surv., vol. 51, no. 1, pp. 3:1–3:34, 2018.

[3] J.-Y. Kuo, Z.-J. Wen, T.-F. Hsieh, and H.-X. Huang, “A Study on the Security of Online Judge System Applied Sandbox Technology,” Electronics, vol. 12, no. 14, Art. no. 3018, Jul. 2023.

[4] B. Jeong and Y. S. Jeong, “Autoscaling Techniques in Cloud-Native Computing: A Comprehensive Survey,” Comput. Sci. Rev., vol. 58, pp. 1–26, Nov. 2025.

[5] E. Casalicchio, “A Study on Performance Measures for Auto-Scaling CPU-Intensive Containerized Applications,” Cluster Comput., vol. 22, no. 3, pp. 995–1006, Jan. 2019.

[6] T.-T. Nguyen, Y.-J. Yeom, T. Kim, D.-H. Park, and S. Kim, “Horizontal Pod Autoscaling in Kubernetes for Elastic Container Orchestration,” Sensors, vol. 20, no. 16, Art. no. 4621, Aug. 2020.