

백엔드 & 인프라 스터디 3 회차 문제

2025-10-11

작성자 : 김호중

백엔드 – 논리적 사고 문제: http

상황

당신은 전 세계 수백만 명의 학생들에게 비디오 강의, 인터랙티브 퀴즈, 그리고 커뮤니티 포럼을 제공하는 글로벌 플랫폼의 백엔드 개발자로 면접을 보고 있습니다.

최근 플랫폼은 아시아 태평양 지역으로 서비스를 확장했지만, 해당 지역 사용자들로부터 "사이트가 너무 느리다", "강의 비디오 로딩에 한 세월이 걸린다"는 불만이 폭주하고 있습니다. 모든 인프라는 미국 동부 리전에 집중되어 있으며, 현재 아키텍처는 다음과 같은 기술적 문제점을 안고 있습니다.

극심한 지연 시간(Latency): 한국의 한 학생이 강의 페이지를 열면, 수십 개의 작은 이미지, CSS, JS 파일 요청이 미국 서버까지 왕복해야 합니다. 현재 시스템은 HTTP/1.1 을 사용하고 있어, 브라우저의 제한된 동시 연결 수와 프로토콜 자체의 한계로 인해 페이지 로딩이 끔찍하게 느립니다.

비효율적인 데이터 전송: 사용자가 이미 시청했던 강의 페이지를 다시 방문할 때마다, 거의 모든 데이터를 매번 새로 다운로드하여 불필요한 트래픽과 비용을 발생시키고 있습니다.

불안정한 퀴즈 제출: 네트워크가 불안정한 모바일 환경에서 학생이 퀴즈 답안을 제출 (POST /api/quizzes/{id}/submit)할 때, 간헐적인 타임아웃이 발생합니다. 학생이 재시도 버튼을 누르면, 동일한 답안이 중복으로 제출되어 채점 시스템에 오류를 일으키는 문제가 발생하고 있습니다.

보안 취약점: 최근 보안 감사에서, 악의적인 사이트가 iframe 을 통해 Cognita 의 로그인 페이지를 그대로 복제하여 사용자의 자격 증명을 탈취할 수 있는 '클릭재킹(Clickjacking)' 취약점과, 사용자가 실수로 http://로 접속했을 때 암호화되지 않은 통신에 노출될 수 있는 위험이 발견되었습니다.

당신에게 주어진 임무는, HTTP 프로토콜에 대한 깊이 있는 이해를 바탕으로 이 모든 문제의 근본 원인을 진단하고, 이를 해결하기 위한 현대적이고 견고한 아키텍처 개선 방안을 제시하는 것입니다.

문제

첫째, 아시아 태평양 지역 사용자들이 겪는 극심한 지연 시간 문제의 근본 원인을 HTTP/1.1 프로토콜의 내재적 한계와 TCP 의 동작 방식과 연관 지어 설명하십시오. 특히, 'Head-of-Line Blocking'이 무엇이며, 이것이 왜 여러 개의 작은 리소스를 로드해야 하는 현대 웹 환경에서 치명적인지 분석해야 합니다. 이어서, 이 문제를 해결하기 위해 HTTP/2 를 도입하는 것이 왜 효과적인지, 멀티플렉싱(Multiplexing)의 관점에서 설명하고, HTTP/2조차 해결하지 못하는 근본적인 TCP 계층의 Head-of-Line Blocking 문제를 HTTP/3 (QUIC)가 어떻게 UDP 를 기반으로 해결하는지 그 진화 과정을 심층적으로 논하십시오.

둘째, 비효율적인 데이터 전송 문제를 해결하기 위한 다계층 캐싱 전략을 설계하시오. 강의 비디오 파일과 같이 거의 변하지 않는 대용량 정적 자산과, 사용자의 프로필 정보처럼 개인화된 동적 API 응답에 대해 각각 어떤 캐싱 전략을 적용할 것인지 설명해야 합니다. 이 전략에는 브라우저 캐시, CDN 엣지 캐시, 그리고 서버 측에서의 검증 토큰을 활용하여 데이터 재전송을 최소화하는 방안이 모두 포함되어야 합니다.

셋째, 불안정한 네트워크 환경에서 퀴즈 답안이 중복 제출되는 문제를 방지하기 위해, POST 요청을 어떻게 멍등성을 가지도록 만들 수 있는지 설명하시오. 클라이언트가 타임아웃 발생 시 안전하게 재시도할 수 있도록, Idempotency-Key HTTP 헤더를 활용하는 표준적인 패턴을 통해 서버 측에서 중복 요청을 식별하고 처리하는 상세한 로직과 그 아키텍처적 구현 방안을 제시해야 합니다.

넷째, 보안 감사에서 발견된 두 가지 취약점을 해결하기 위한 방어 메커니즘을 HTTP 헤더 수준에서 어떻게 구현할 것인지 설명하시오. 사용자의 브라우저가 항상 HTTPS 로만 사이트에 접속하도록 강제하여 중간자 공격(MITM)을 방지하는 HSTS(HTTP Strict Transport Security) 헤더의 역할과 설정 방법을 설명하고, 클릭재킹 공격을 방지하기 위해 브라우저에게 페이지가 프레임 내부에서 렌더링되는 것을 제어하도록 지시하는 X-Frame-Options 또는 CSP(Content Security Policy) 헤더의 사용법을 논하시오.

다섯째, 당신이 제안한 여러 개선안 중, 가장 효과가 클 것으로 기대되는 HTTP/3로의 전환을 추진한다고 가정해 봅시다. 이 기술적 결정이 가져올 수 있는 예상치 못한 위험과 운영상의 도전 과제는 무엇일까요? 단순히 기술의 장점만을 나열하는 것을 넘어, 기업의 방화벽이나 오래된 네트워크 장비(Middleboxes)가 UDP 트래픽을 차단할 수 있는 문제, 로드 밸런서 및 CDN 공급자의 지원 여부, 그리고 새로운 프로토콜을 디버깅하고 모니터링하기 위한 기술적 준비 등, 실제 프로덕션 환경에 새로운 기술을 도입할 때 고려해야 할 현실적인 트레이드오프와 마이그레이션 전략을 당신의 통찰력을 담아 제시하시오.

백엔드 – 논리적사고문제: 데이터베이스 설계

상황

당신은 전 세계 수천 개의 기업 고객을 대상으로 웹사이트의 가용성과 성능을 모니터링하는 SaaS 플랫폼 **SitePulse**의 백엔드 개발자로 면접을 보고 있습니다.

SitePulse의 핵심 기능은 간단합니다. 고객이 자신의 웹사이트 URL을 등록하면, 우리 시스템은 전 세계 5개 주요 도시(버지니아, 프랑크푸르트, 싱가포르 등)에 위치한 워커(Worker)를 통해 **매분** 해당 웹사이트에 HTTP 요청을 보내 응답 시간, 상태 코드, 성공 여부를 측정합니다

초기 버전의 데이터베이스 스키마는 극도로 단순하게 설계되었습니다.

websites (id, customer_id, url)

health_checks (id, website_id, checked_at, location, response_time_ms, status_code)

서비스가 빠르게 성장하여 현재 10,000 개의 고객 웹사이트를 모니터링하고 있습니다. 이로 인해 health_checks 테이블에는 매일 약 **7,200 만 건** ($10,000 \text{ sites} \times 5 \text{ locations} \times 60 \text{ mins} \times 24 \text{ hours}$), 매달 **20 억 건** 이상의 레코드가 폭발적으로 쌓이고 있습니다.

이 거대한 데이터의 무게로 인해 시스템은 다음과 같은 심각한 문제에 직면했습니다

(Write Contention): 수만 개의 워커가 매분 동시에 health_checks 테이블에 INSERT를 시도하면서, 테이블과 인덱스에 대한 락(Lock) 경합이 발생하여 쓰기 성능이 저하되고 있습니다.

(Query Timeout): 고객 대시보드의 핵심 기능인 '지난 30 일간의 99 퍼센타일(p99) 응답 시간'을 계산하는 쿼리는 수십억 건의 데이터를 스캔해야 하므로, 실행에 수 분이 걸리거나 데이터베이스 타임아웃으로 실패합니다

운영 불가능성: health_checks 테이블에 새로운 인덱스를 추가하거나 스키마를 변경하는 작업은 테이블 전체에 락을 걸어 몇 시간, 혹은 며칠이 걸릴 수 있어 사실상 어떠한 유지보수 작업도 불가능한 상태입니다

당신에게 주어진 임무는, 이 총체적 난국을 해결하고 초당 수만 건의 쓰기와 복잡한 분석 쿼리를 동시에 처리할 수 있는 확장 가능한 v2 데이터베이스 아키텍처를 설계하는 것입니다.

문제

첫째, 현재의 단일 health_checks 테이블 설계가 왜 이토록 심각한 성능 저하를 유발하는지 데이터베이스의 **물리적 저장 구조와 인덱스의 동작 원리** 관점에서 심층적으로 분석하시오. 데이터가 계속 추가될 때 B-Tree 인덱스의 깊이가 어떻게 증가하며, 이것이 쓰기 성능(Write Amplification)과 읽기 성능(Cache Efficiency)에 어떤 연쇄적인 악영향을 미치는지 그 내부 메커니즘을 설명해야 합니다.

둘째, 이 시스템의 워크로드는 명백히 쓰기(OLTP)와 읽기(OLAP)의 요구사항이 충돌하고 있습니다. 이 문제를 해결하기 위해, 쓰기 경로와 읽기 경로를 분리하는 **CQRS(Command Query Responsibility Segregation)** 패턴을 적용한 아키텍처를 제시하시오. 쓰기 전용으로는 어떤 종류의 데이터베이스(예: 시계열 데이터베이스)나 스키마 설계가 적합할 것이며, 대시보드와 같은 읽기 전용으로는 어떻게 데이터를 사전 집계(Pre-aggregation)하여 별도의 분석용 저장소에 보관할 것인지 그 전체적인 데이터 파이프라인과 각 컴포넌트의 역할을 설명해야 합니다.

셋째, '지난 30 일간의 p99 응답 시간'과 같은 분석 쿼리를 밀리초 단위로 응답 가능하게 만들기 위한 **사전 집계(Roll-up) 전략**을 구체적으로 설계하시오. 원본(raw) 데이터를 어떤 시간 단위(예: 분, 시간, 일)로 집계할 것이며, 평균값(average)만 저장해서는 안 되는 이유는 무엇인지, 그리고 백분위수(Percentile)와 같은 비가산적(non-additive) 지표를 정확하게 계산하기 위해 어떤 종류의 통계적 데이터 구조(예: **T-Digest**, **HyperLogLog**)를 집계 테이블에 함께 저장해야 하는지 그 근거를 논하시오

넷째, 이 시스템은 수천 개의 고객사가 함께 사용하는 **다중 테넌트(Multi-tenant)** 환경입니다. 한 명의 대형 고객사(수천 개의 웹사이트를 등록한)가 복잡한 대시보드 쿼리를 실행할 때, 다른 소규모 고객사들의 쿼리 성능에 영향을 미치는 '**시끄러운 이웃(Noisy Neighbor)**' 문제를 어떻게 방지하겠습니까? 데이터베이스 **파티셔닝(Partitioning)**이나 **샤딩(Sharding)** 전략을 customer_id 를 기준으로 어떻게 적용할 수 있는지, 그리고 이러한 물리적 데이터 격리가 성능뿐만 아니라 보안과 데이터 관리 측면에서 어떤 이점을 제공하는지 설명해야 합니다

다섯째, 수십억 건의 원본 health_checks 데이터는 영원히 뜨거운(hot) 데이터베이스에 보관할 수 없습니다. 데이터의 가치와 접근 빈도에 따라 데이터를 차등 관리하는 **데이터 생명주기 관리(Data Lifecycle Management)** 전략을 수립하시오. '지난 30 일'의 원본 데이터는 빠른 조회를 위해 유지하되, 그보다 오래된 데이터는 어떻게 저비용의 **콜드 스토리지(Cold Storage, 예: Amazon S3)**로 안전하게 **아카이빙**하고, 규제 준수나 특별 분석 요청 시 해당 데이터를 어떻게 다시 조회할 수 있는지 그 전체적인 프로세스와 아키텍처를 설계하시오.

백엔드 – 논리적사고문제: 트러블 슈팅

상황

당신은 전 세계 수백만 명의 게이머를 대상으로 희귀 아이템을 실시간 경매 방식으로 판매하는 플랫폼 Mythic Marke*의 백엔드 개발자로 면접을 보고 있습니다.

지난 주말, 가장 기대가 컸던 아이템의 경매 종료 1 분을 앞두고 시스템에 전례 없는 문제가 발생했습니다. 경매 마감을 앞둔 수십만 명의 사용자가 폭발적으로 입찰(Bid)을 시도했고, 이때 약 1%의 사용자로부터 "입찰이 처리되지 않고 계속 로딩 중이다" 혹은 "알 수 없는 오류가 발생했다"는 치명적인 불만이 접수되었습니다.

사후 분석 결과, 상황은 더욱 미스터리했습니다.

증상: 해당 사용자들의 입찰 요청(POST /api/auctions/{id}/bids)은 평균 30 초 이상 소요되다 결국 HTTP 504 Gateway Timeout 으로 실패했습니다.

모니터링: APM(Application Performance Monitoring) 대시보드에서는 해당 엔드포인트의 p99 응답 시간이 수십 초까지 치솟았지만, CPU 와 메모리 사용률은 60% 수준으로 여유가 있었습니다. 애플리케이션 서버나 데이터베이스(RDS for PostgreSQL) 인스턴스가 다운되지는 않았습니다.

로그: 애플리케이션 로그에는 특별한 에러가 기록되지 않았습니다. 단지 로드 밸런서의 액세스 로그에 504 응답 기록만 남아있을 뿐입니다.

데이터베이스: RDS 의 Performance Insights 를 확인해보니, 특정 시점에 lwlock:buffer_content 와 관련된 대기 이벤트(Wait Event)가 급증한 기록이 발견되었습니다.

당신에게 주어진 임무는, 이 흩어진 단서들을 바탕으로 미스터리한 성능 저하의 근본 원인을 진단하고, 재발을 방지할 수 있는 견고한 해결책을 제시하는 것입니다.

문제

첫째, 당신은 이 사건의 근본 원인을 찾기 위한 조사를 시작해야 합니다. "CPU 와 메모리는 여유로운데, 응답은 극도로 느려졌다"는 이 모순적인 상황은 무엇을 시사합니까? 당신이 가장 먼저 의심할 만한 세 가지 가설을 제시하고, 각 가설이 왜 이 현상을 설명할 수 있는지, 그리고 그 가설을 검증하거나 반증하기 위해 어떤 구체적인 데이터(메트릭, 로그, 시스템 상태)를 어떤 도구(예: APM, pg_stat_activity, perf)를 사용하여 확인할 것인지 당신의 체계적인 진단 과정을 설명하시오.

둘째, 가장 유력한 단서인 데이터베이스의 lwlock:buffer_content 대기 이벤트에 대해 심층적으로 분석하시오. 이 대기 이벤트가 무엇을 의미하며, 이것이 데이터베이스의 어떤 내부 메커니즘과 관련되어 있는지 설명해야 합니다. 이어서, 높은 동시성 환경에서 '핫 스팟(Hot Spot)', 즉 단일 데이터 블록(페이지)에 대한 극심한 경합이 어떻게 전체 데이터베이스의 처리량을 마비시키고, CPU 는 놀고 있는데도 애플리케이션은 응답을 받지 못하는 이 상황을 만들어낼 수 있는지 그 내부 동작 원리를 논하시오.

셋째, 이 '핫 스팟' 문제가 경매의 최고 입찰가를 기록하는 auctions 테이블의 단일 행(Row)에서 발생했다고 가정해 봅시다. 전통적인 SELECT FOR UPDATE 를 사용한 비관적 락(Pessimistic Locking) 방식이 어떻게 이 문제를 더욱 악화시킬 수 있는지 설명하고, 이 문제를 해결하기 위한 두 가지 서로 다른 데이터베이스 패턴을 제시하고 비교하시오. 하나는 낙관적 락(Optimistic Locking)을 사용하여 락의 범위를 최소화하는 방안이며, 다른 하나는 입찰 기록을 별도의 로그 테이블에 먼저 빠르게 INSERT 하고, 실제 최고가 갱신은 백그라운드에서 비동기적으로 처리하여 쓰기 경로를 분리하는 아키텍처적 변경 방안입니다.

넷째, 이 문제를 해결하기 위해 애플리케이션, 데이터베이스, 그리고 인프라 전반에 걸쳐 당신이 구현할 다계층 방어(Multi-layered Defense) 전략을 설계하시오. 애플리케이션 레벨에서는 어떻게 연결 풀(Connection Pool)과 스레드 풀(Thread Pool)의 크기를 정교하게 튜닝하여 데이터베이스에 가해지는 압력을 제어할 것이며, 데이터베이스 레벨에서는 어떤 인덱싱 전략이나 스키마 변경을 고려할 수 있는지, 그리고 인프라 레벨에서는 어떻게 읽기 전용 복제본(Read Replica)을 활용하여 읽기/쓰기 워크로드를 분리하고 시스템의 전체적인 복원력을 높일 것인지 종합적인 해결책을 제시해야 합니다.

다섯째, 당신은 이 사건의 재발을 막기 위해 무엇을 할 것입니까? 단순히 문제를 해결하는 것을 넘어, 미래의 잠재적인 병목을 사전에 식별하고 방지하기 위한 선제적인 시스템을 구축해야 합니다. 어떻게 부하 테스트(Load Testing) 시나리오를 설계하여 이번과 같은 '핫 스팟' 경쟁 조건을 재현하고 시스템의 한계점을 측정할 것이며, 어떤 핵심 지표(예: 트랜잭션당 DB Lock 대기 시간)에 대한 경고(Alerting)를 설정하여 문제가 발생하기 전에 이상 징후를 감지할 것인지, 당신의 SRE(Site Reliability Engineering) 관점에서의 장기적인 안정성 확보 계획을 논하시오.

백엔드 – 논리적사고문제: 트러블 슈팅

상황

당신은 여러 사용자가 동시에 데이터 분석 작업을 실행하는 고성능 컴퓨팅 플랫폼 'Synapse Analytics'의 백엔드 개발자로 면접을 보고 있습니다. 이 플랫폼의 핵심은 256GB의 대용량 RAM을 탑재한 단일 Ubuntu 22.04 서버로, 사용자가 제출한 메모리 집약적인 Python 스크립트(Pandas, NumPy 사용)를 워커 프로세스 풀에서 병렬로 처리합니다.

평상시에는 시스템이 안정적으로 운영되지만, 여러 사용자가 동시에 대용량 데이터셋을 처리하는 작업을 제출하여 시스템의 메모리 사용량이 임계치에 가까워지면 예측 불가능한 문제가 발생합니다. 갑자기 시스템 전체가 몇 분간 극심하게 느려지며 SSH 접속조차 끊기는 현상이 발생하고, 잠시 후 특정 사용자의 워커 프로세스 하나가 아무런 애플리케이션 로그도 남기지 않고 갑자기 종료됩니다. 사용자에게는 그저 '작업 실패'라고만 통보되어 불만이 쌓이고 있습니다.

팀의 주니어 엔지니어는 `free -h` 명령의 출력에서 free 메모리가 수십 GB 남아있는 것을 보고 메모리 누수가 아니라고 판단했으며, 원인을 파악하지 못해 혼란에 빠져 있습니다. 하지만 당신은 `dmesg` 로그에서 다음과 같은 Out Of Memory Killer의 흔적을 발견했습니다.

```
dmesg | grep "Out of memory"
```

```
[... snip ...] Out of memory: Killed process 12345 (python3) total-vm:85943212kB, anon-rss:62345678kB, file-rss:0kB, shmem-rss:0kB
```

문제

첫째, 주니어 엔지니어가 `free -h` 명령어의 출력을 어떻게 오해하고 있는지, 리눅스 커널의 메모리 관리 철학의 관점에서 심층적으로 분석하십시오. 출력에 표시되는 free와 available 메모리의 근본적인 차이점은 무엇이며, 커널이 유휴 메모리를 페이지 캐시(Page Cache)로 적극적으로 활용하는 이유가 무엇인지 설명해야 합니다. 왜 시스템의 실제 메모리 압박 상태를 파악하려면 free가 아닌 available 지표를 신뢰해야 하는지 그 근거를 논리적으로 제시하십시오.

둘째, 리눅스 커널의 OOM Killer 는 왜 그리고 어떤 기준으로 희생양(victim) 프로세스를 선택하는지 그 내부 메커니즘을 설명하시오. 커널이 각 프로세스에 대해 계산하는 oom_score 의 의미를 설명하고, 어떤 요소들(예: 프로세스가 사용하는 메모리 양, 자식 프로세스의 메모리, nice 값 등)이 이 점수에 영향을 미치는지 분석해야 합니다. 또한, 시스템 운영에 필수적인 sshd 나 데이터베이스 같은 중요 프로세스가 OOM Killer 의 희생양이 되지 않도록 관리자가 oom_score_adj 값을 조정하여 어떻게 커널의 결정에 개입할 수 있는지 그 방법을 설명하시오.

셋째, OOM Killer 의 사후 대응이 아닌 사전 예방을 위해, 컨트롤 그룹(Control Groups, cgroups)을 어떻게 활용하여 자원을 격리하고 관리할 수 있는지 구체적인 아키텍처를 제시하시오. 여러 사용자의 워커 프로세스들을 사용자별 cgroup 으로 묶고, 이 cgroup 에 memory.max 와 같은 하드 리밋(hard limit)을 설정하는 것이 어떻게 전체 시스템의 안정성을 보장하는지 설명해야 합니다. 이 cgroup 내에서 메모리 한계를 초과하는 작업이 발생하면, 시스템 전역 OOM Killer 대신 cgroup 내부의 OOM Killer 가 동작하여 문제의 범위를 해당 그룹 내로 국한시키는 '피해 범위 최소화(Blast Radius Reduction)' 원리를 분석하시오.

넷째, 시스템의 안정성을 높이기 위해 스왑(Swap) 공간을 활성화하는 방안이 제시되었습니다. vm.swappiness 라는 커널 파라미터가 메모리 관리 정책에 어떤 영향을 미치는지 설명하시오. 이 값을 100 으로 설정했을 때와 1 로 설정했을 때, 커널이 파일 기반 페이지(File-backed Pages)를 회수하는 것과 익명 메모리 페이지 Anonymous Pages 를 스왑 아웃하는 것 사이에서 어떤 선택을 하게 되는지 그 차이점을 분석해야 합니다. Synapse Analytics 와 같이 인메모리 데이터 처리가 성능에 지대한 영향을 미치는 워크로드 환경에서, 스왑을 사용하는 것의 성능상 트레이드오프와 당신이 추천하는 swappiness 값의 근거를 논하시오.

다섯째, OOM Killer 가 호출되기 직전 시스템이 극심하게 느려지는 현상의 원인인 '스래싱(Thrashing)'에 대해 설명하시오. 시스템의 가용 메모리가 거의 고갈된 상태에서, 새로운 메모리 할당 요청이 계속 들어올 때 커널이 어떻게 페이지 캐시 회수와 스왑 아웃을 반복하며 시스템 자원의 대부분을 실제 연산이 아닌 메모리 페이지 교체 작업에만 소모하게 되는지 그 악순환의 과정을 설명해야 합니다. 이 현상이 어떻게 시스템 전체의 응답 불능 상태를 초래하는지 그 기술적인 원리를 분석하시오.