스터디 트래커 부하 테스트 보고서

목차

1. Executive Summary

스터디 트래커는 Slack 메시지를 수신하여 학습 데이터를 저장하고, OpenAI 를 통해 피드백을 생성한 뒤 사용자에게 전달하는 시스템입니다. 본 테스트는 해당 시스템을 고부하, 병목 상황에서 관찰하고 아키텍처 개선 방향을 도출합니다.

시스템 개요 및 테스트 환경

기능 흐름: Slack 메시지 수신 → DB 저장 → 스케줄러 실행 → OpenAI 요약 및 피드백 → Slack 전달

로컬: JMeter + Prometheus + Grafana

클라우드: AWS EC2 (Docker 기반)

테스트 시나리오 개요:

- SC1: 대량 Slack 이벤트 수신 – TPS 한계 측정
- SC2: 단일 사용자 집중 부하 – 스케줄러 병목 및 동기 구조 확인
- SC3: 다수 사용자 + 스케줄러 – 병목 유도 및 구조적 한계 분석
- SC4: 현실적 하루 흐름 – 정상 사용 시 시스템 안정성 확인

핵심 결과:

- SC1 에서 1600 Threads 이상 시 TPS 급감 및 BindException 발생
- 클라우드 환경에서 스케줄러 병목 발생, CPU 사용률 100%
- 정상적인 흐름(SC4)에서는 안정적으로 작동
- 전체적으로 비동기 아키텍처 개선 필요

2. 테스트 시나리오 상세 분석

2.1 SC1 – 대량 Slack 이벤트 수신 (TPS 한계 측정)

목표: 대량의 Slack 이벤트가 동시에 수신될 때, 서버의 TPS 한계와 처리 성능을 측정

테스트 구성:

- Thread 수를 100 에서 5000 까지 점진적으로 증가시키며 총 33,600 건 전송

로컬 결과:

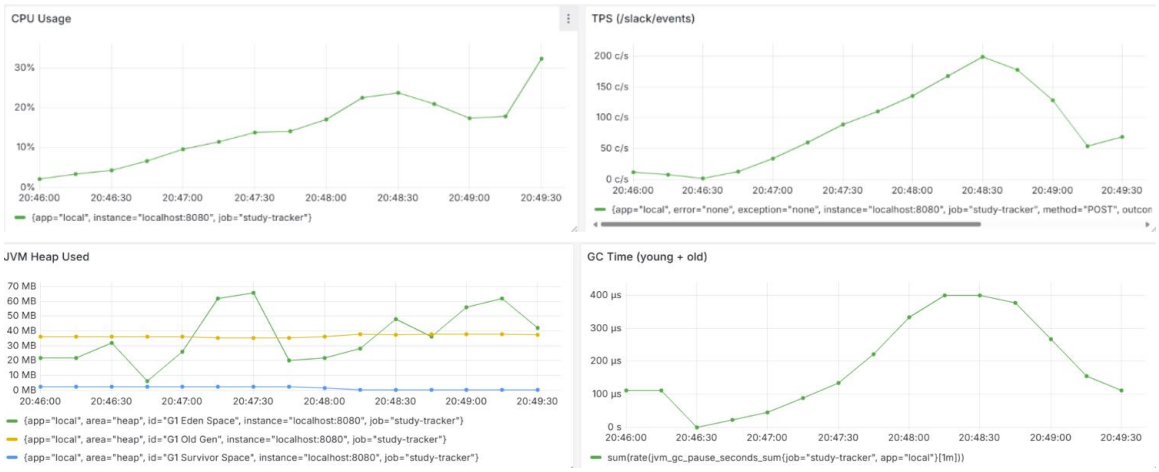
- 병목 시작 시점: Thread 약 1600 개부터
- 평균 응답 시간 185ms, 에러율 30%

- BindException 발생 → 포트 고갈

클라우드 결과:

- 평균 응답 시간 58ms 로 개선되었으나 에러율은 32%로 유사
- 여전히 BindException 발생

로컬



클라우드

차이 분석

CPU Usage : 로컬은 JMeter 와 서버가 동일 머신에 있어 자원 경합 발생

TPS(/slack/events) : 클라우드는 자원 여유로 성능 저하가 완만하게 발생

GC Time, Heap 사용량 : 클라우드는 메모리 여유로 객체 누적이 많지만 GC 안정적

추가 테스트 : BindException 원인 분석

목표 : BindException 의 발생 원인이 포트 고갈인지 확인

포트 고갈 유도 테스트 구성

- 시스템 레벨에서 ephemeral port range 설정은 유지
- 대신 JMeter 내 Thread 수를 5000 으로 고정하고 Loop 증가
- KeepAlive 비활성화해서 요청마다 새로운 포트 사용 -> 포트 고갈 유도

로컬 결과

- 에러율 65.5%, 평균 응답시간 1,970ms
- Throughput: 1430 TPS(실패한 요청이 빠르게 끝나서 비정상적으로 높을 뿐임)

클라우드 결과

- 에러율 92.6%, 평균 응답시간 163,007ms
- BindException 급증 → 클라우드는 포트 자원 한계 더 빠르게 도달

결론

- BindException 의 직접 원인은 ephemeral 포트 자원 고갈
- 운영 환경에서도 다음과 같은 조치 필요:
    - KeepAlive 및 커넥션 풀 관리
    - 리버스 프록시 및 부하 분산 적용
    - 비동기 또는 큐잉 아키텍처 설계 고려

로컬 테스트

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	33600	10395	30.94%	185.21	1	3352	7.00	934.90	2247.90	3058.98	163.48	132.91	50.39
슬랙 이벤트 전송	33600	10395	30.94%	185.21	1	3352	7.00	934.90	2247.90	3058.98	163.48	132.91	50.39

클라우드 테스트

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	33600	11054	32.90%	58.89	3	1042	35.00	200.00	271.00	361.00	161.97	138.39	48.72
슬랙 이벤트 전송	33600	11054	32.90%	58.89	3	1042	35.00	200.00	271.00	361.00	161.97	138.39	48.72

포트 고갈 유도 테스트(로컬)

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	60000	39313	65.52%	1970.72	3	25314	272.00	13804.00	18342.95	20629.96	1430.79	2194.93	217.75
슬랙 이벤트 전송	60000	39313	65.52%	1970.72	3	25314	272.00	13804.00	18342.95	20629.96	1430.79	2194.93	217.75

포트 고갈 유도 테스트(클라우드)

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	18930	17521	92.56%	163007.72	76	704137	21046.00	700506.90	701846.45	702959.69	26.82	64.12	0.89
슬랙 이벤트 전송	18930	17521	92.56%	163007.72	76	704137	21046.00	700506.90	701846.45	702959.69	26.82	64.12	0.89

※ 주의: 본 테스트의 주요 실패 응답은 HTTP status code 가 없는 BindException 으로, 일반적인 4xx/5xx 기반 에러율 그래프에는 반영되지 않음. 추후에는 포트 자원을 보다 효율적으로 재사용할 수 있는 **비동기 I/O 기반 도구**(예: **Netty 기반 Gatling**, **Coroutine 기반 K6**)를 활용하여 **보다 신뢰성 높은 부하 테스트**를 할 필요있음.

2.2 SC2 – 단일 사용자 집중 부하 + 스케줄러

목표: 단기 집중 부하 상황에서의 트랜잭션 안정성과 스케줄러의 동작 안정성 확인

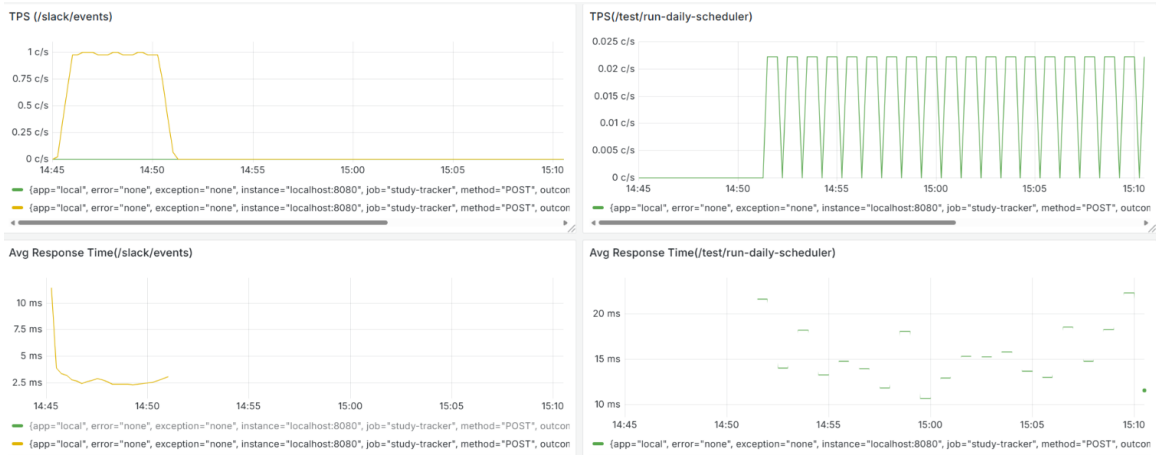
테스트 구성:

- 한 명의 사용자가 5 분 동안 300 개의 Slack 메시지를 전송
- 이후 1 분 간격으로 20 회의 스케줄러가 작동하여 총 25 분간 테스트

로컬 결과:

- Slack 평균 응답 시간: 4.6ms, 스케줄러 평균 응답 시간: 17.9ms

- 에러율 0%



클라우드 결과:

- 너무 느려서 수동 종료함. 일부 스케줄러 트리거 미실행됨

- 스케줄러 응답 시간 최대 36 초, TPS 0.025 c/s 이하 유지

- CPU 사용률 100% 도달 → 병목 지점과 일치

- 메모리 누수가 아닌, 트리거 처리의 동기 구조가 병목 원인으로 추정됨

결론

-클라우드는 EC2 의 네트워크 지연과 I/O 속도 한계로 인해 트리거 누락이 발생

2.3 SC3 – 다수 사용자 + 스케줄러 병목 유도

목표: 구조적 병목을 유도해 한계를 확인

테스트 구성:

- 200 명의 사용자가 각 300 건 Slack 메시지를 전송 (총 60,000 건)
- 동시에 1 분 간격으로 10 회의 스케줄러 실행

로컬 결과:

- 초기 테스트에서 Slack API 429 Too Many Requests 발생 → 서버 다운
- WebClient 재시도 폭주 → 커넥션 고갈, GC thrashing 발생
- 개선 후 Slack 호출 우회 → 서버는 안정화되었으나 스케줄러 응답 지연 발생

Requests	Executions			Response Times (ms)						
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct
Total	**60003**	**0**	**0.00%**	**5.43**	**1**	**52606**	**3.00**	**7.00**	**8.00**	**16.00**
스케줄러 트리거	3	0	0.00%	31784.67	12861	52606	29887.00	52606.00	52606.00	52606.00
슬랙 이벤트 수신	60000	0	0.00%	3.84	1	175	3.00	7.00	8.00	16.00

클라우드 결과:

- 유사한 구조적 병목 발생, 응답 시간 최대 28 초

- TPS 0.025c/s 로 제한, 트리거 실행 누락 지속

위 그래프에서 CPU 사용률 급등, TPS 제한 시점, 응답 시간 증가의 시간대가 일치함

결론:

- Slack 호출, WebClient 재시도, Blocking 구조의 병목이 명확히 드러남
- 메시지 큐 기반 비동기 처리 구조로 전환 필요

2.4 SC4 – 현실적인 하루 흐름

목표: 현실적인 흐름에서 시스템의 기본 성능과 안정성을 검증

테스트 구성:

- 50 명의 사용자가 1 분 간격으로 각 3 건의 Slack 메시지를 전송 (총 151 건)
- 15 분 후 스케줄러 1 회 작동

로컬 결과:
- Slack 응답 시간 4.6ms, 스케줄러 797ms
- CPU 사용률 2~6%, 에러율 0%

클라우드 결과:
- Slack 응답 시간 86ms, 스케줄러 1232ms
- CPU 사용률 2% 이하, 에러율 0%

결론:
- 네트워크 레이턴시와 EC2 성능 차이로 인한 응답 시간 차이가 있을 뿐 로컬, 클라우드에서 안정적으로 작동

3. 개선 방향 정리

영역	개선 방향
SLACK 이벤트 수신	Kafka, RabbitMQ 등 비동기 메시지 큐 도입
스케줄러 처리	@Async, Job Queue 기반 비동기 처리 구조 적용
리소스 관리	EC2 인스턴스 수평 확장, 스레드/메모리 풀 최적화, ephemeral 포트 고갈 방지
네트워크 커넥션	KeepAlive 활성화, 커넥션 풀 설정 조정, 포트 재사용 극대화
인프라 아키텍처	리버스 프록시 도입, 부하 분산 적용
모니터링	Prometheus 알림(Alert) 설정 도입, 포트 사용량/에러율 트래킹 추가 고려

4. 운영 인사이트 요약

1. TPS 성능 보장 조건

- EC2 **t2.micro** 기준으로, OpenAI 호출을 제외한 /slack/events API 는 스레드 1,600 개 이하에서 안정적인 TPS 를 유지함

2. 성능 한계 시 고려할 개선 방향

- 트래픽 급증과 짧은 간격의 스케줄러 반복 실행으로 인해 TPS 급감 및 트리거 누락 현상이 발생함. (※ 이는 실제 운영 환경(예: 일 1 회 실행)과는 다르며, 병목 유도 실험이라는 점을 고려해야 함)
- 이 경우, **EC2 인스턴스 스케일 아웃**, **Job Queue 도입**, **WebClient 재시도 설정 최적화** 등이 병목 해소에 효과적
- **메시지 큐 기반 아키텍처 전환**이 구조적 확장성과 장애 대응 측면에서 유리

3. 리소스 영향 분석

- 시나리오 2 에서는 TPS 급감과 CPU 사용률 100% 도달이 동시에 나타났으며 클라우드 환경에서의 I/O 성능 한계, 네트워크 레이턴시, JVM 내부 처리 부담이 복합적으로 작용한 결과로 해석됨
- 특히 요약 생성 과정에서의 **순차적 처리 방식**이 병목을 유발하며, 리소스 경합이 성능 저하에 영향을 미친 것으로 보임

4. 장애 징후 및 원인 추적 방법

- 장애 발생 시, 다음과 같은 지표 패턴이 반복적으로 관찰됨:
    - GC Time 증가 ↔ TPS 급감
    - WebClient 재시도 폭주 ↔ 커넥션 수 증가

- 스케줄러 지연 ↔ CPU 100% 도달 및 Thread 수 증가

- 따라서 Prometheus/Grafana 에서 다음 항목의 실시간 모니터링이 중요:
  - TPS 추세, GC Time, 커넥션 수, Active Thread 수, 스케줄러 대기 큐 길이

5. 부하 테스트 도구의 한계

- JMeter 는 고강도 테스트 환경에서 ephemeral 포트 고갈로 인한 BindException 이 자주 발생함.
- 이로 인해 **HTTP 요청이 아예 생성되지 못하는 상황**이 반복되었으며, 이는 애플리케이션 자체의 병목이 아닌 **부하 생성 도구의 구조적 한계**로 판단됨.
- 특히 BindException 은 HTTP status code 로 수집되지 않아 **에러율/성능 지표 분석에서 누락**되는 문제가 있었음.
- 신뢰성 있는 부하 테스트를 위해, 비동기 커넥션을 지원하는 **Gatling, K6** 등의 도구 이용을 고려할 필요가 있음.