러너스 하이 1기 경력기술서 - 김민우

목표

I/O bound, 주기적으로 처리 지연이 발생하는 API를 호출하는 배치 시스템의 수행시간 단축과 안정성 향상

성과

- API 호출 구조 개선으로 수행시간 단축(1시간 22분 → 23분)

- API 처리 지연 발생 시 timeout이 발생하는 요청 35% 감소로 배치 수행 안정성 향상

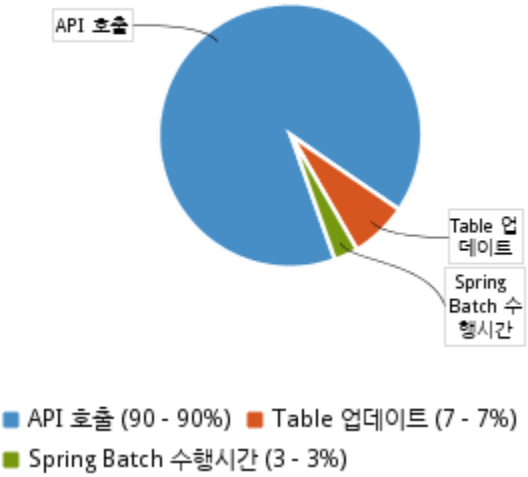
기술 스택

Java, Spring Batch, HTTP

작업내용

1) 문제정의

신규 배치를 테스트하면서 병목지점을 인지했고, 우리 시스템에서 직접 제어할 수 있는 부분과 제어할 수 없는 부분으로 구분해 문제 해결 방안을 도출했습니다.



구체적인 병목지점은 총 배치 수행시간의 90%를 차지하는 외부 API 호출입니다. 외부 API 호출에 대해 IntelliJ Profiler로 구조를 파악해보니 네트워크 I/O 대기시간이 그 중 99%를 차지하고 있었습니다.

따라서 API 호출과 관련된 부분은 제어할 수 있는 지점으로 봤고, API 호출 대기시간은 제어할 수 없는 부분으로 구분했습니다.

1) 제어할 수 있는 부분 - API 호출 구조

제어할 수 있는 부분(=API 호출)은 우리 시스템의 API 호출 구조를 개선하여 배치 수행시간을 줄일 수 있도록 했습니다.

2) 제어할 수 없는 부분 - API 대기 시간, 외부 서버의 처리 지연

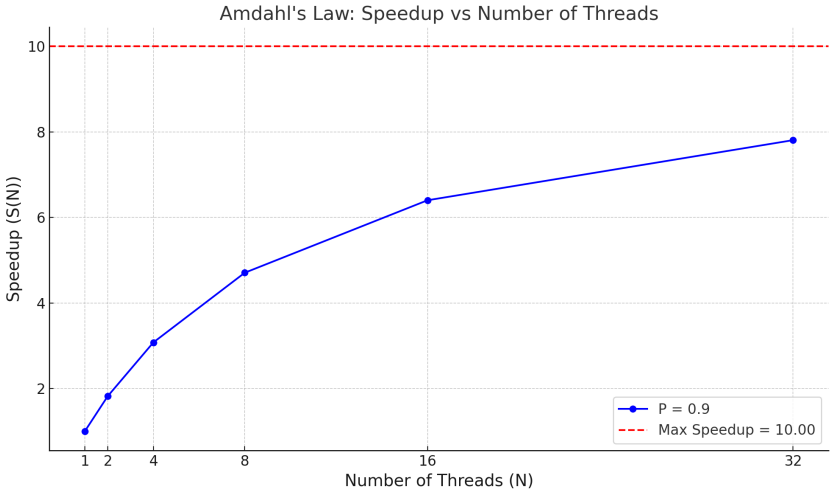
제어할 수 없는 부분에 대해서는 배치 수행의 안정성을 높이는 방향으로 개선을 진행했습니다.

2) 수행 시간 단축 - API 호출 구조 개선

1) API 호출 병렬 처리

외부 API를 호출하는 경우 응답 시간 자체를 줄이기는 어렵습니다.

API 호출(기프티콘 발급)이 서로 독립적인 요청이므로, Spring Batch의 Flow를 통해 병렬 처리를 구현했습니다.



스레드에 I/O 작업을 독립적으로 분배할 수 있을 때 암달의 법칙을 이용해 대략적인 수행 효율 향상을 그래프로 그려볼 수 있습니다.

병렬 처리 단위(스레드 개수)를 계속 늘려도 그에 비례하는 수행시간 향상을 기대하기는 어려울 것이라고 생각하여, 리소스 대비 효율적인 수행시간 향상을 실험을 통해 찾아내 적용했습니다.

수행 향상과 리소스(네트워크 처리량, DB connection 등) 사용량 사이의 trade-off를 고려해 수치를 결정했습니다.

스레드를 10개 사용하는 경우 간헐적으로 DB connection 부족 현상이 발생하여 스레드를 7개로 결정했습니다.

호출 방식	처리 단위	데이터 건수	수행시간	건 당 수행시간
직접 호출	단일 스레드	10,000	1시간 22분	0.49 초/건
직접 호출	멀티 스레드(5개)	10,000	35분	0.21 초/건
직접 호출	멀티 스레드(7개)	10,000	23분	0.14 초/건
직접 호출	멀티 스레드(10개)	10,000	-	-

2) 그 외 시도한 것들

-http client 최적화: keep-alive header, connection pool 설정으로 connection 재사용 효율을 높이려고 했지만, 외부 서버의 keep-alive: 5초로 제한되어 현재 최대 효율이라고 판단
-비동기 요청으로 변경: 쿠폰 발급이라는 업무 특성 상 API 호출 안정성이 데이터 정합성과 직결된다고 판단하여 동기 방식으로 개발, 외부 서버의 주기적인 처리 지연으로 인해 비동기 방식에서는 트래픽 초과를 유발할 것으로 예상하여 폐기
=> 후술할 timeout 도입으로 비동기 방식과 유사한 구현 채택, 과도한 요청 트래픽을 줄이기 위해 요청량 제한기 도입 아이디어

3) 안정성 향상 – API 응답 시간 제한

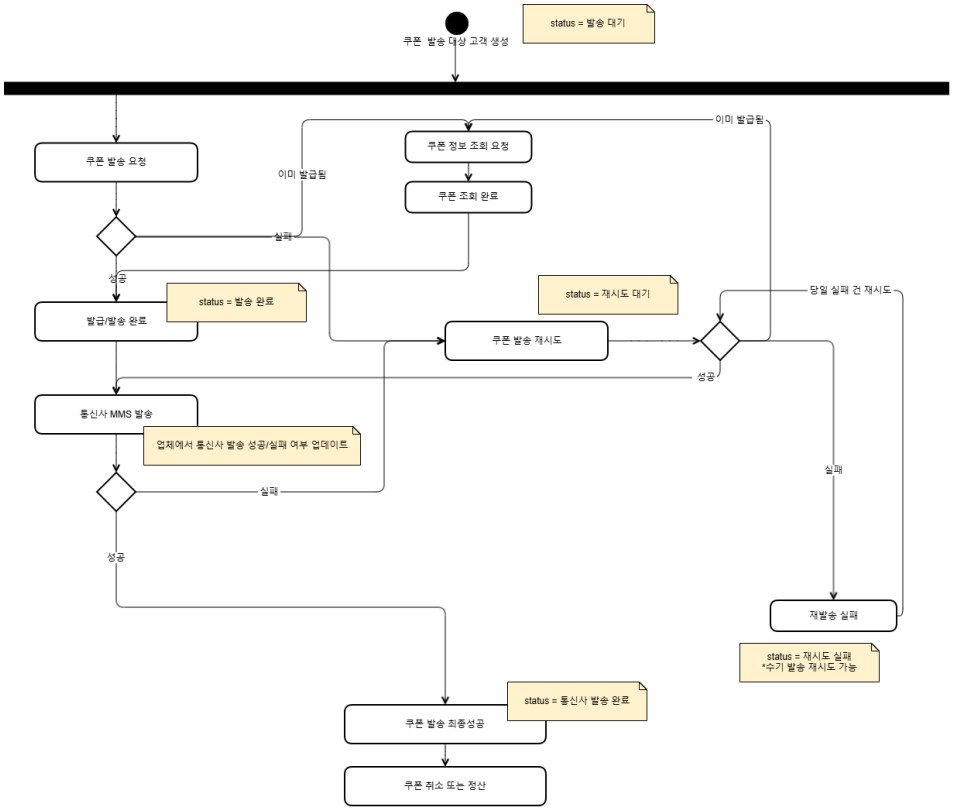
병렬처리를 통해 건 당 수행시간은 감소했지만, 단위 시간 당 트래픽 증가로 인해 외부 시스템의 처리 지연이 주기적으로 발생하게 되었습니다. 외부 API의 응답 시간은 0.1초~60초 수준으로 응답 시간이 너무 길어지는 경우 이를 제한 해야 할 필요성이 생겼습니다.

외부 API 호출 시간을 직접 줄일 수는 없지만, API timeout 설정으로 개별 API의 응답 시간을 제한할 수 있습니다.

timeout을 사용하게 되면 일방적인 연결 해제로 인해 요청이 성공적으로 처리되었더라도 응답을 받을 방법이 없습니다. 그래서 재처리 로직을 추가해 향후 데이터를 재확인할 수 있도록 했습니다.

timeout 도입은 응답 시간 제한(안정성 향상)과 재처리 로직 추가로 인한 추가 리소스 소모 사이의 trade-off를 고려해야 합니다.

리소스 소모 값보다 응답 시간 제한으로 인한 개선이 더 크다고 생각되어 timeout을 도입하고 그에 따른 추가적인 재처리 로직을 추가했습니다.

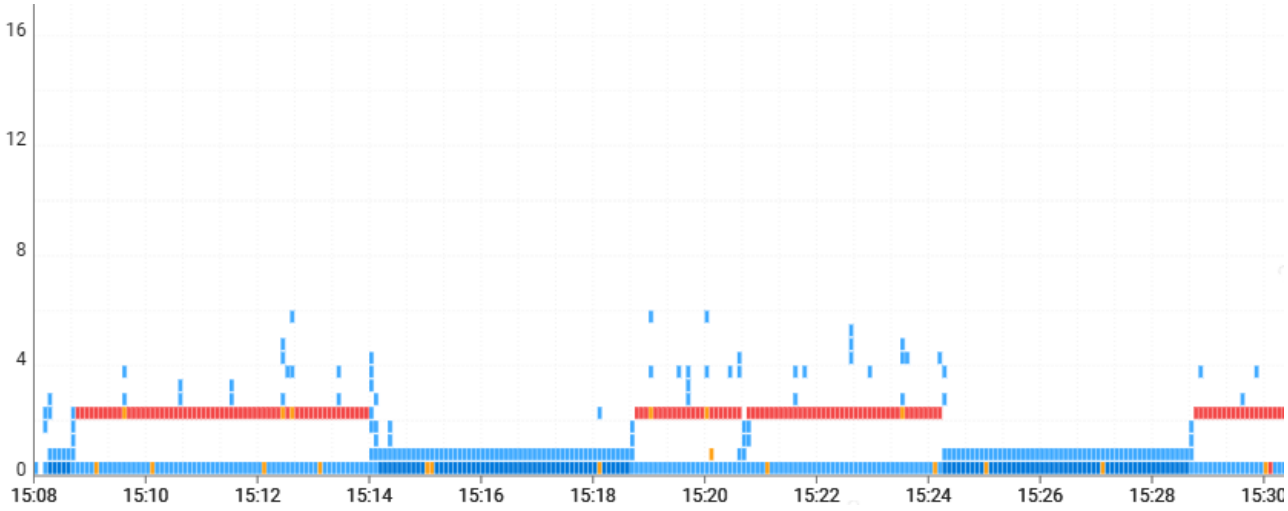


4) 안정성 향상 – 외부 API 서버 처리 지연 대응

3번 개선을 통해 timeout을 도입하고 재처리 로직을 추가했습니다. 하지만 재처리 로직은 (쿠폰 발급 요청 → 이미 발급 응답 → 쿠폰 조회 요청 → 쿠폰 데이터 응답)의 과정으로, 두 번의 API 요청이 필요하므로 리소스 낭비가 발생합니다.

그래서 timeout을 통해 능동적으로 처리 지연에 대응하면서도, timeout 되는 요청 횟수는 줄이기 위해 요청량을 제한하는 모듈에 대한 아이디어를 생각했습니다.

아래 표와 그래프를 보면 timeout은 처리 지연 구간에서 집중적으로 발생하고 그 외 기간에는 1% 이하로 발생하는 양상을 보입니다. 이에 대한 대비를 통해 배치의 수행 안정성을 높일 수 있을 것으로 기대했습니다.



< 요청량 제한기 미적용 >

< 요청량 제한기 적용 >

요청량 제한기 미적용 상태의 그래프 처럼 주기적인 처리 지연 현상 대응을 위해 요청량 제한기를 개발하여 적용했습니다. (github repository)

Sliding Window 알고리즘을 기반으로 처리 지연이 발생한 구간에서 추가 요청량을 제한하여 요청의 안정성을 향상 시키려고 했습니다.

아래 표는 실험 결과를 정리한 것이고, 결과 값을 통해 요청 안정성이 향상된 것을 확인할 수 있습니다. (**다음 장에 표 첨부**)

요청량 제한기	총 수행 시간	timeout 건 수	처리 지연 지속시간	처리 지연 중 timeout 발생 비율	처리 지연 구간
미적용	18분 54초	1285건	6분 40초	3.21 개/초	09:18:39 - 09:21:58 09:28:38 - 09:31:59
	19분 12초	1373건	7분 11초	3.19 개/초	09:48:38 - 09:52:05 09:58:39 - 10:02:23
	23분 33초	2409건	13분 36초	2.95 개/초	14:10:29 - 14:13:13 14:18:39 - 14:23:40 14:28:44 - 14:33:45
	23분 44초	2520건	13분 22초	3.14 개/초	14:40:30 - 14:43:23 14:48:40 - 14:53:46 14:58:40 - 15:04:00
	25분	2847건	15분 2초	3.15 개/초	15:08:39 - 15:13:51 15:18:40 - 15:24:05 15:28:40 - 15:33:05
	22분 53초	2249건	12분 3초	3.11 개/초	16:58:39 - 17:04:43 17:08:42 - 17:14:41
적용	20분 31초	916건	8분 45초	1.74 개/초	11:00:30 - 11:02:04 11:08:38 - 11:12:30 11:18:38 - 11:20:57
	20분 34초	1050건	8분 10초	2.14 개/초	11:28:39 - 11:32:36 11:38:38 - 11:42:51
	23분 48초	1492건	11분 50초	2.10 개/초	13:09:40 - 13:12:36 13:18:38 - 13:23:07 13:28:39 - 13:33:04
	21분 34초	1096건	8분 38초	2.11 개/초	13:38:39 - 13:43:00 13:48:42 - 13:52:59
	27분 10초	1904건	16분 17초	1.95 개/초	15:39:25 - 15:43:47 15:48:39 - 15:54:07 15:58:39 - 16:04:06
	23분 19초	1395건	11분 2초	2.11 개/초	16:18:39 - 16:24:10 16:28:39 - 16:34:10

< 요청량 제한기 적용/미적용 결과>

이 기회를 통해 제일 중요하게 생각한 점

데이터 기반 의사 결정

의사 결정에 필요한 데이터를 뽑아내기 위해 최대한 독립적인 실험을 목표로 함

문제 정의를 잘하는 것에 대한 고민 => 해결할 수 없을 것 같은 문제도 다른 관점에서 해결해보려고 노력.

어디까지 알아야 확신을 가지고 아이디어를 적용할 수 있을까에 대한 고민