성능 개선법

배경

웹 어플리케이션은 다중사용자 지원능력이 중요하다. 따라서, 이를 평가하기 위한 테스트가 필요하다.

테스트에는 두가지 종류가 있다.

1. **성능 테스트** : 시스템의 성능을 평가한다.
2. 스트레스 테스트 : 서버가 과부하일 때, 서버의 동작과 장애 복구 절차를 평가한다.

이 글에서는 성능 테스트에 집중한다.

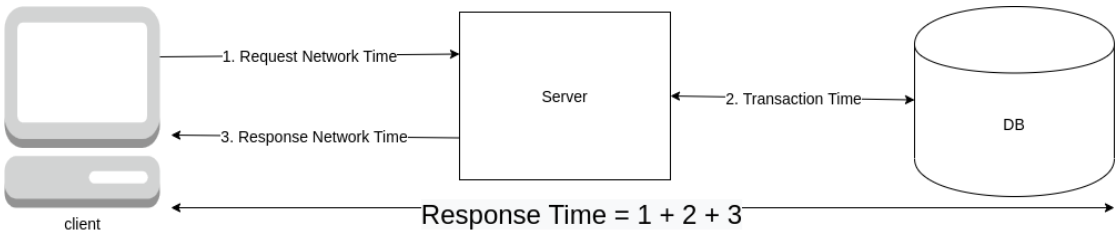
성능 테스트

시스템의 성능은 다음과 같이 3가지 지표로 표현될 수 있다.

1. **유저 수** : 얼마나 많은 사람들이 동시에 사용할 수 있는가?
2. **시간** : 서비스의 응답시간이 빠른가?
3. **TPS** : 일정시간동안 얼마나 많은 트랜잭션을 처리할 수 있는가?

$$
\begin{aligned}
TPS &= \frac{TR}{T}, \text{where TR is the number of transaction and T is the total sec} \\
&= \frac{AU}{RT}, \text{where AU is the number of active user and RT is response time}
\end{aligned}
$$

- Active User란, 실제로 트랜젝션을 요구하는 유저 수로, 서버의 쓰레드 수로 볼 수 있다.
- Response Time이란, 사용자가 서버 요청부터 응답까지 걸린 모든 시간이다. 아래 그림과 같이 3가지로 표현될 수 있다.



    1. Latency Time (Network Time, 대기시간) : 서버 요청을 보내고 응답 받는 네트워크 시간이다.
    2. Transaction Time : 서버가 요청을 처리하는 시간이다.
- TPS는 유저 수와 시간 지표를 통합하는 종합적인 지표로 볼 수 있다.

따라서, 우리가 성능 목표를 세울 때, 위와 같은 3가지 지표를 활용하여 표현할 수 있다.
(예: 초당 10만 tps를 감당하는 서비스 제공, 응답시간 10ms 이하인 API 등)

성능 개선법

성능을 개선하기 위해 다음과 같은 일련의 과정을 지킨다.

1. **목표 설정** : 지표를 활용하여 목표를 설정한다.
2. **문제 정의** : 문제를 정의한다. 이때, 해당 문제는 재현이 가능해야 한다.
3. **문제 분해** : 시나리오 기반으로 어떤 구간이 문제인지 Break down 한다.
4. **문제 확인** : 문제인 구간을 집중 분석하기 위해, 연동된 다른 부분으로부터 고립시켜 테스트한다.
5. **근원지 찾기** : 근원을 찾기 위해 Narrow Down 하여, 근원지를 발견한다.

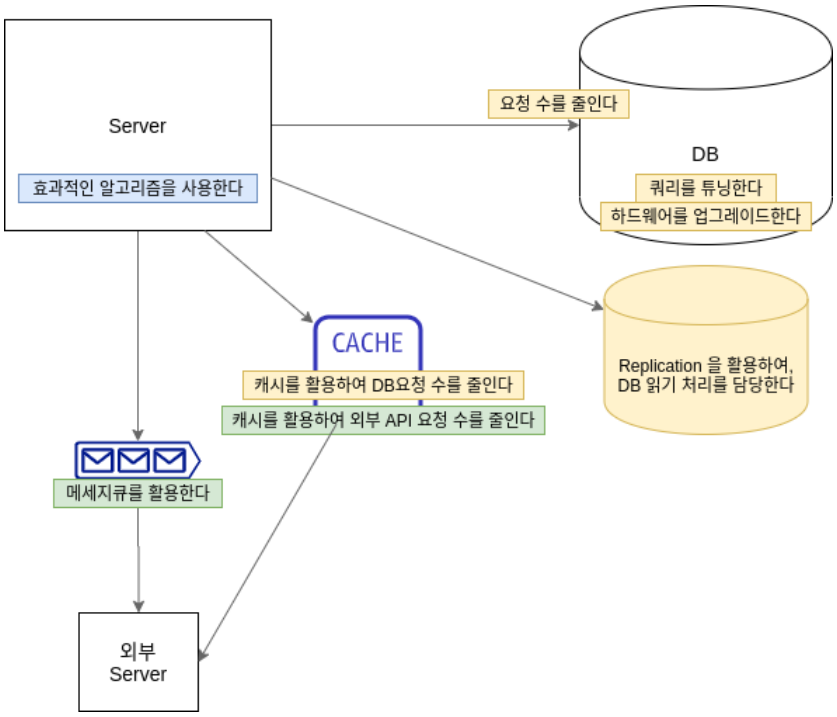
6. **해결** : 해결한다. 이때, 설계상의 문제는 아닌지 큰 관점에서도 보도록 한다.

성능 개선 결과 예시

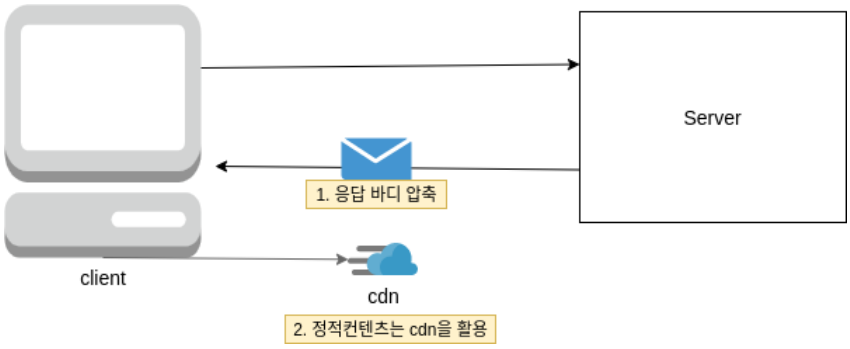
성능 개선은 응답시간을 줄이거나 액티브 유저를 늘리는 것으로 해석할 수 있다. 아래 글은 이 목적을 달성하기 위해 자주 도출되는 결과이다. 하지만 자주 사용되는 해결법일 뿐, 문제의 원인 분석과 해결방법은 꼭 개인이 해야한다는 점을 주의한다.

1. 응답시간에서 transaction time을 줄이기 위한 방법 예시

   노란 박스는 DB로 인한 처리시간을, 초록 박스는 외부 API로 인한 처리시간을 줄이기 위한 방법이다. 또한, 파란 박스는 서버 내부의 처리시간을 줄일 수 있는 방법이다.



2. 응답시간에서 latency time을 줄이기 위한 방법 예시



3. Active User 수를 늘리는 방법 : Thread Pool 늘릴 수 있다.

성능 개선 예시 : 트위터

다음은 트위터에서 공개한 성능 개선 사례를 과정에 따라 정리한 내용이다.

1. 목표 설정

트위터에서 게시물을 올렸을 때, 상대의 Timeline에 올라오는 **응답속도가 5초 이내**

2. 문제 정의

00에서 응답속도가 00초가 되는 문제를 발견하였다.

(3. 문제 분해, 4. 문제 확인)

(공개하지 않았음)

5. 근원지 찾기

30만 qps를 DB가 감당하고 있다.

6. 해결

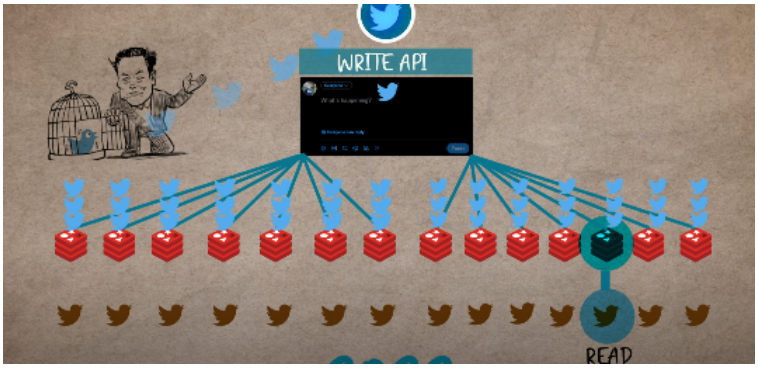
해결 방향

- **qps는 왜 많은가?** 팔로워 수만큼 읽기 요청이 있어야 한다. 트위터 구조상 요청수를 줄일 수는 없다.
- **그럼에도 qps를 줄일 수 있는 방법이 있는가?** DB컨넥션을 줄이면 되는 것이므로, 캐시를 활용한다. 이는 요청수가 qps에 큰 영향이 가지 않도록 할 수 있다.

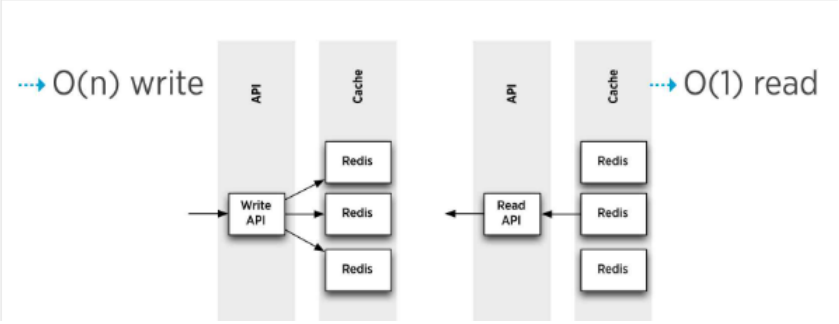
해결 방법

캐시를 도입하여 시스템 아키텍처를 다시 설계한다.

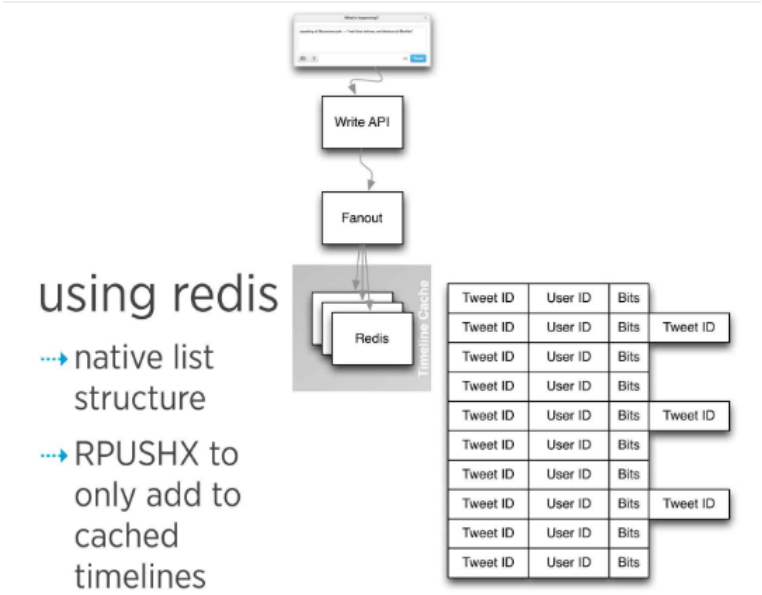
- **REDIS 선택 이유** : key-value 형태가 필요하다. 또다른 key-value형태인 Memcache는 10만 tps에서 병목현상을 보이는 네트워크 대역폭 문제가 있어서 Redis를 선택한다.
- **REDIS 스팩**
    1. 레디스 구성 : 클러스터 및 복제 구성



- 클러스터 : 데이터를 00개 서버에 분산저장한다.
- 복제 : 각 클러스터에 slave를 3개씩 두어서 데이터의 안정성을 추구한다.



2. 자료구조 선택 : 리스트



3. 알고리즘에 따른 메모리 판단 : (8byte + 8byte + 4byte) * 1000 followers * x 개수 = 20KB * x

References

https://www.infoq.com/presentations/Twitter-Timeline-Scalability/

http://highscalability.com/blog/2014/9/8/how-twitter-uses-redis-to-scale-105tb-ram-39mm-qps-10000-ins.html

https://www.youtube.com/watch?v=FEkXjNFrL1o

https://bcho.tistory.com/787

http://www.jidum.com/jidums/view.do?jidumId=562

https://grapevine9700.tistory.com/402