13장. 검색어 자동 완성 시스템

구글 검색 또는 아마존 웹 사이트 검색창에 단어를 입력하다 보면 입력 중인 글자에 맞는 검색어가 자동으로 완성되어 표시되는 것을 볼 수 있다. 이번 장에서는 **가장 많이 이용된 검색어 k개를 자동완성하여 출력하는 시스템을 설계**할 것 이다.

1단계) 문제 이해 및 설계 범위 확정

요구사항

- 빠른 응답 속도: 사용자가 검색어를 입력함에 따라 자동완성 검색어도 충분히 빨리 표시되어야 한다. 약 100ms 이내에 응답되어야 하며 이후면 느리다고 판단된다.
- 연관성: 자동완성이 출력되는 검색어는 사용자가 입력한 단어와 연관된 것이어야 한다.
- 정렬: 시스템의 계산 결과는 인기도(popularity) 등의 순위 모델(ranking model)에 의해 정렬되어 있어야 한다.
- 규모 확장성: 시스템의 많은 트래픽을 감당할 수 있도록 확장 가능해야 한다.
- 고가용성: 시스템의 일부에 장애가 발생하거나, 느려지거나, 예상치 못한 네트워크 문제가 생겨도 시스템은 계속 사용가능해야 한다.

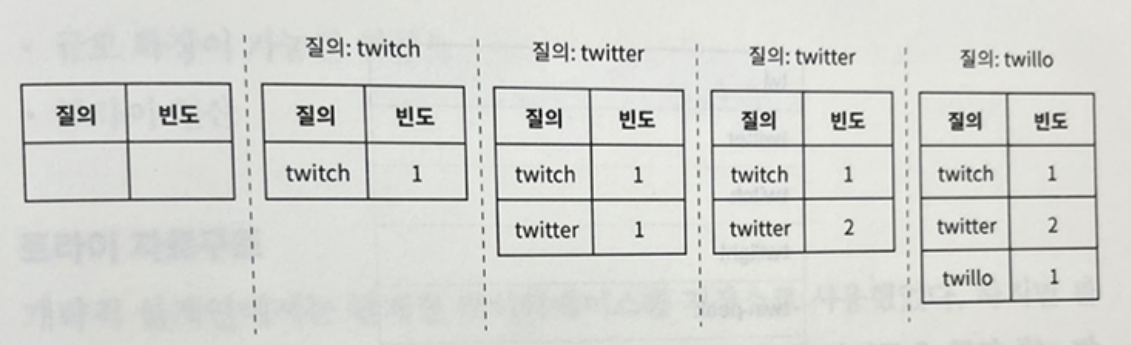
개략적 규모 추정

- 일간 능동 사용자(DAU)는 천만명으로 가정한다.
- 평균적으로 한 사용자는 매일 10건의 검색을 수행한다.
- 질의할 때마다 평균적으로 20바이트의 데이터를 입력한다고 가정한다.
    - 문자 인코딩 방법으로는 ASCII를 사용하고 1문자(1Byte)이다.
    - 질의문이 평균 4글자고 한 글자는 5문자로 구성된다고 할 때, 질의는 평균 20byte 값을 갖는다.

- 검색창에 글자를 입력할 때마다 클라이언트는 검색어 자동완성 백엔트에 요청을 보낸다.
- 대략 초당 24,000건의 질의(QPS)가 발생할 것이다.
  - 10,000,000 사용자 x 10질의 / 일x20자 / 24시간 / 3600초 = 24,000
- 최대 QPS = QPS * 2 = 약 48,0000
- 질의 가운데 20% 정도는 신규 검색어라고 가정한다. 약 0.4GB이다.
  - 10,000,000 사용자 x 10질의 / 일x20자 x 20% = 0.4GB

2단계) 개략적 설계안 제시 및 동의 구하기

- **데이터 수집 서비스 (data gathering service)**: 사용자가 입력한 질의를 실시간으로 수집하는 시스템이다.
  - 질의문과 사용빈도를 저장하는 빈도 테이블(frequency table) → key-value의 해쉬 맵 형태같다.



질의	빈도

질의: twitch

질의	빈도
twitch	1

질의: twitter

질의	빈도
twitch	1
twitter	1

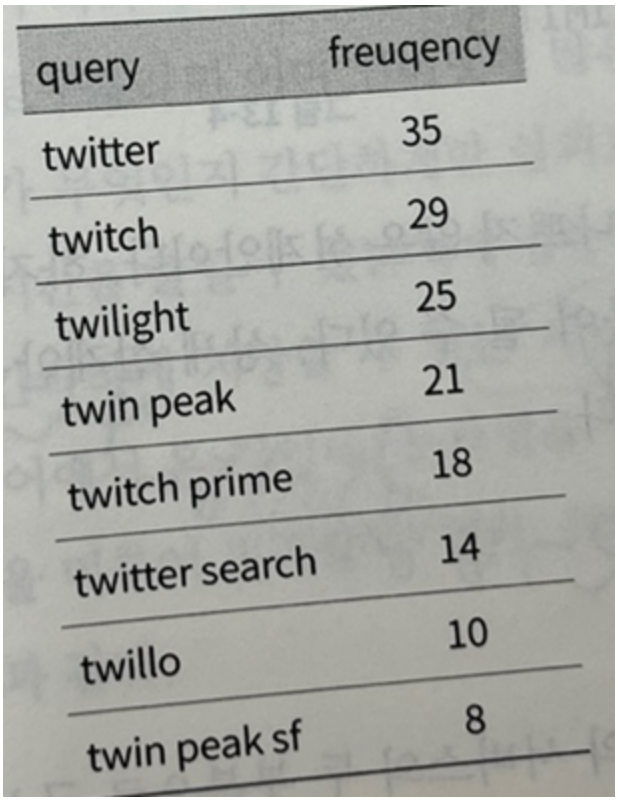
질의: twitter

질의	빈도
twitch	1
twitter	2

질의: twillo

질의	빈도
twitch	1
twitter	2
twillo	1

- **질의 서비스 (query service)**: 주어진 질의에 다섯개의 인기 검색어를 정렬해 내놓는 서비스이다.
  - 아래 빈도 테이블의 필드는 다음과 같다.
    - query: 질의문을 저장하는 필드
    - frequency: 질의문이 사용된 빈도를 저장하는 필드



query	freuqency
twitter	35
twitch	29
twilight	25
twin peak	21
twitch prime	18
twitter search	14
twillo	10
twin peak sf	8

위 설계안은 데이터 양이 적을 때는 나쁘지 않은 설계안이다. 그렇지만 데이터가 아주 많아지면 데이터베이스가 병목이 될 수 있다.

▼ **참고1. 병목현상이란?**

병의 목 부분처럼 넓은 길이 갑자기 좁아지면서 발생하는 교통 정체 현상, 컴퓨터 성능 저하 형상이다.

이는 컴퓨터 성능 저하 현상 중 하나로 엄청난 양의 데이터를 순식간에 내보내더라도 메모리가 이를 제대로 소화하지 못해 성능이 떨어지는 현상으로 주로 용량이 적은 주변기기를 사용할 때 많이 발생한다.

3단계) 상세 설계

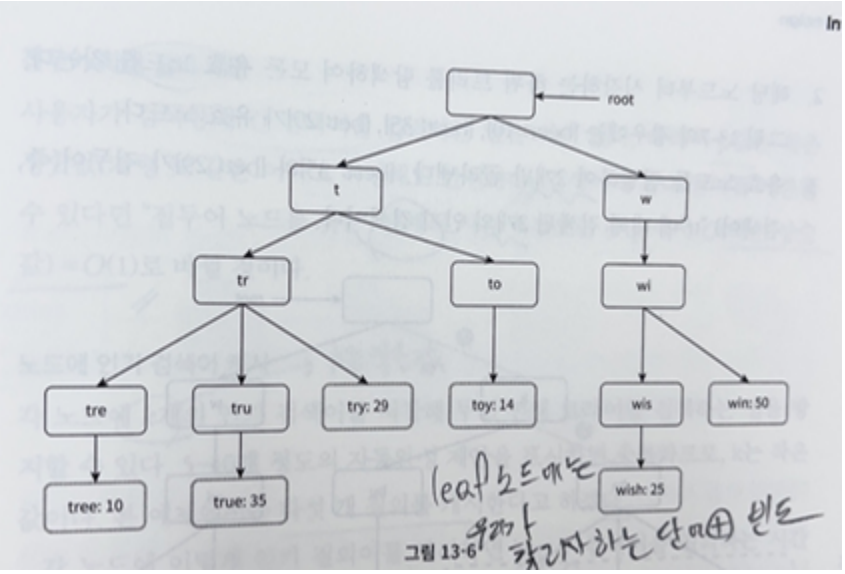
- **트라이(trie) 자료 구조**

  데이터베이스를 이용해 가장 인기 있었던 5개의 질의문을 골라내는 방안은 효율적이지 않다. 이 문제는 트라이(trie, 접두어 트리_prefix tree)를 사용해 해결해야 한다.

트라이는 문자열들을 간략하게 저장할 수 있고 문자열을 꺼내는 연산에 초점을 맞추어 설계된 자료구조이다.

트라이 핵심 정리

- 트라이는 트리 형태의 자료구조이다.
- 이 트리의 루트 노드는 빈 문자열을 나타낸다.
- 각 노드는 글자(character) 하나를 저장하며, 26개의 자식노드를 가질 수 있다.
- 각 트리 노드는 하나의 단어, 또는 접두어 문자열을 나타낸다.



용어

- p: 접두어(prefix)의 길이
- n: 트라이 안에 있는 노드 개수
- c: 주어진 노드의 자식 노드 개수

알고리즘 및 시간 복잡도 계산

1. 해당 접두어를 표현하는 노드를 찾는다. 시간 복잡도는 O(p)이다.
2. 해당 노드부터 시작하는 하위 트리를 탐색하여 모든 유효노드(유효한 검색 문자열을 구성하는 노드, 각 노드에서의 자식 노드)를 찾는다. 시간 복잡도는 O(c)이다.
3. 유효 노드들을 정렬하여 가장 인기 있는 검색어 k개를 찾는다. 시간 복잡도는 O(clogc)이다.

⇒ 총 시간복잡도 : O(p) + O(c) + O(clogc)

이 알고리즘은 직관적이지만 최악의 경우에는 k개 결과를 얻으려고 트라이를 모두 검색해야 하는 일이 생길 수도 있다.

해결책

- ***접두어의 최대 길이를 제한***

  O(p)에서 O(작은 상숫값) == O(1)로 바뀐다.

- ***각 노드에 인기 검색어를 캐시***

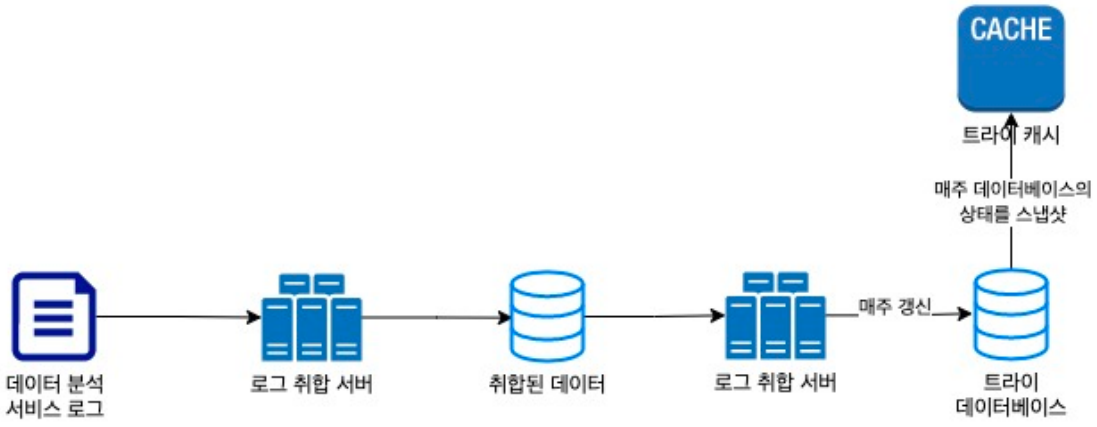
  캐시를 저장할 추가 공간이 필요하다. 그러나 빠른 응답속도가 아주 중요할 때는 저장 공간을 희생할 만한 가치가 있다.

위 두 해결책을 사용하면 O(1)로 시간복잡도가 줄어들게 된다.

- **데이터 수집 서비스**

  앞선 설계안은 사용자가 검색창에 타이핑을 하면 실시간으로 데이터를 수정했는데 이 때 발생하는 문제점은 다음과 같다.

  - 매일 수천만 건의 질의가 입력될 텐데 그때마다 트라이를 갱신하면 질의 서비스는 심각하게 느려질 것이다.
  - 일단 트라이가 만들어지고 나면 인기 검색어는 그다지 자주 바뀌지 않을 것이다. 그러니 트랑리는 그렇게 자주 갱신할 필요가 없다.



데이터 분석 서비스 로그

검색창에 입력된 질의에 관한 원본 데이터가 보관된다.

로그 취합 서버

데이터 분석 서비스로부터 나오는 로그는 보통 그 양이 엄청나고 데이터 형식도 제각각인 경우가 있는데 이 데이터를 잘 취합하여 시스템이 쉽게 소비할 수 있도록 해준다.

보통 자동검색의 경우 1주일에 한번씩 데이터 취합 기간을 갖는다.

취합된 데이터

취한한 데이터들은 QUERY, TIME, FREQUENCY 등의 정보로 정리된다.

작업 서버

주기적으로 비동기적 작업을 실행하는 서버의 잡합니다.

트라이 캐시

분산 캐시 시스템으로 트라이 데이터를 메모리에 유지하여 읽기 연산 성능을 높인다.

트라이 데이터베이스

트라이 데이터베이스로 사용할 수 있는 선택지는 두 가지가 있다.

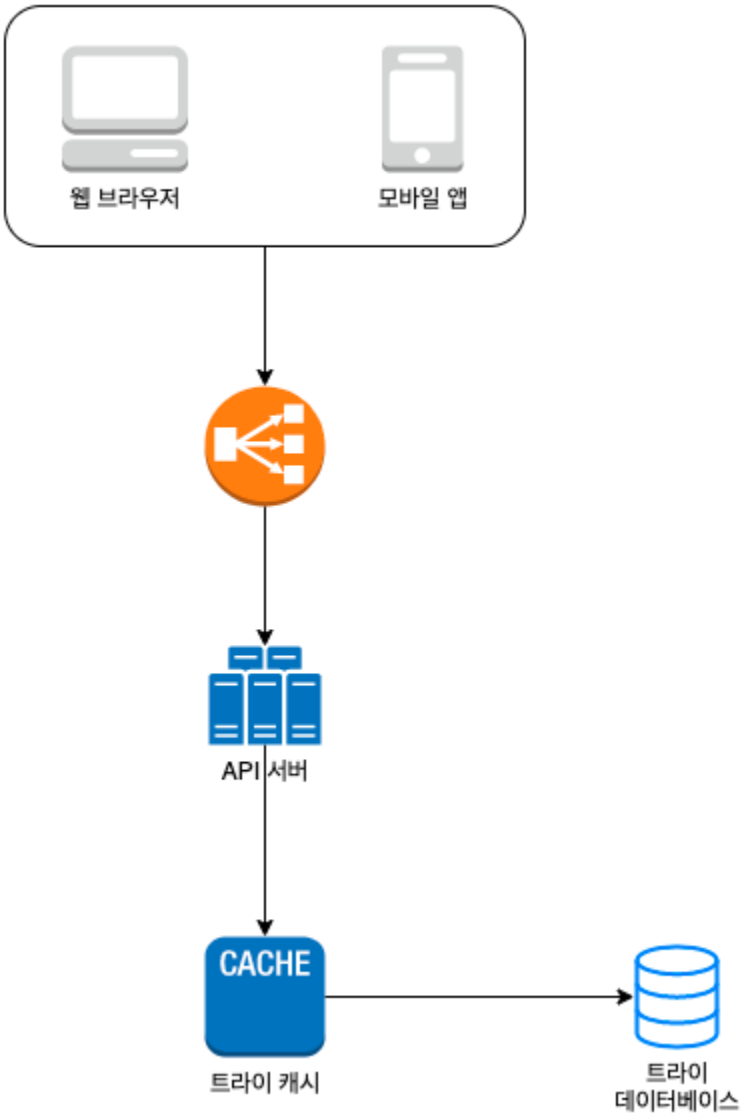
문서 저장소

새 트라이를 매주 만드는 것으로 주기적으로 트라이를 직렬화하여 데이터베이스에 저장한다.

키-값 저장소

- 트라이에 보관된 모든 접두어를 해시 테이블 키로 변환한다.
- 각 트라이 노드에 보관된 모든 데이터를 해시 테이블 값으로 변환한다.

- **질의 서비스**



1. 검색 질의가 로드밸런서로 전송된다.
2. 로드밸런서는 해당 질의를 API서버로 보낸다.
3. API 서버는 트라이 캐시에서 데이터를 가져와 해당 요청에 대한 자동완성 검색어 제안 응답을 구성한다.
4. 데이터가 트라이 캐시에 없는 경우에는 데이터를 데이터베이스에서 가져와서 캐시에 채운다.

최적화 방안

- AJAX 요청(request)
- 브라우저 캐싱(browser caching)
- 데이터 샘플링(data sampling)

- **트라이 연산**

  트라이생성

  트라이 생성은 작업 서버가 담당하며, 데이터 분석 서비스의 로그나 데이터베이스로부터 취합된 데이터를 이용

  트라이 갱신

  - 매주 한번씩 갱신
  - 트라이의 각 노드를 개별한 갱신(트라이가 작은 경우)

  검색어 삭제

  위험한 질의어는 자동 완성 결과에서 제거가 필요 (필터 계층을 사용)

- **규모 확장이 가능한 저장소**

  트라이가 너무 큰 경우, 규모 확장성 고려

  - 검색어에 따라 서버를 나눈다
  - 영어의 경우 26개 이상 늘리려면 샤딩을 한다.

4단계) 마무리

- 다음을 추가적으로 질문이 올 수 있습니다.
  - 다국어 지원이 가능하려면? 트라이에 유니코드 데이터를 저장해야 합니다.
  - 국가별로 인기 검색어 순위가 다르다면? 국가별 다른 트라이 사용, CDN등을 통한 응답속도 향상도 고려해봅니다.
- 실시간 변하는 추이를 구현하려면?
  - 샤딩을 통하여 작업 대상 데이터의 양을 줄입니다.

- 순위 모델을 바꾸어 최근 검색어에 보다 높은 가중치를 주도록 합니다.
- 데이터가 스트림 형태로 올 수 있습니다.