

# 3

## 13장 검색어 자동완성 시스템

---

13장에선 검색어 K를 입력하여 자동완성 시키는 시스템을 설계한다.

### 1단계 문제 이해 및 설계 범위 확정

언제나 적절히 요구사항을 명확히 하는 것이 좋다.

- 자동완성될 부분은 검색어의 첫 부분(앞 부분)이다.
- 최대 5개의 검색어가 표시되어야 한다.
- 검색어가 표시될 기준은 질의 기준에 따른 인기 순위이다.
- 질의는 다국어가 지원되면 좋다.
- 모든 질의는 소문자를 기준으로 한다.
- 하루 천만명(DAU)를 수용할 수 있어야 한다.

위 기준을 통해 정리된 사항은 다음과 같다.

#### 1. 빠른 응답 속도

예를들면, 페이스북 자동완성의 경우 100ms이내여야 한다.

#### 2. 연관성

출력되어야하는 검색어는 사용자가 입력한 것과 연관성이 있어야 한다.

#### 3. 정렬

계산 결과는 인기도 등 순위 모델에 의해 결정된다.

#### 4. 규모 확장성

#### 5. 고가용성

따라서 개략적 추정 규모는

- DAU 천만명
- 한 사용자는 매일 10건의 검색 수행
- 질의할 때마다 평균 20바이트
  - ASCII를 사용한다고 가정하기 때문에 1문자 = 1바이트
  - 질의문 평균적으로 4단어로 이루어진다.
  - 따라서  $4 * 5 = 20$
- 검색어를 입력창에 입력할 때마다 클라이언트에서 서버로 요청을 보낸다. 1회 검색당 20건의 요청이 서버로 전달된다. dinner라고 입력하면 다음과 같은 질의 요청이 서버로 가게 된다.
  - search?q=d
  - search?q=di
  - search?q=din
  - search?q=dinn
  - search?q=dinne
  - search?q=dinner
- 따라서 초 당 24,000건의 질의(QPS)가 발생할 것이다. ( $= 10,000,000 \text{ 사용자} * 10 \text{ query} / \text{day} * 20 \text{ char} / 24 / 3600$ )
- 최대  $\uparrow$  QPS = QPS \* 2 = 대략 48,000
- 질의 가운데 20% 정도는 신규 검색이라고 가정한다. 따라서, 대략 0.4GB 그렇기 때문에 매일 0.4GB의 신규 데이터가 시스템에 추가 된다.

### 2단계 개략적 설계안 제시 및 동의 구하기

시스템은 두 부분으로 나뉜다.

- 데이터 수집 서비스

입력한 질의를 실시간으로 수집하는 시스템이다. 보통 데이터를 많이 사용하는 기능을 실시간으로 만드는 것은 바람직하지 않지만 설계안의 초안으로는 괜찮을 것 같다는 게 저자의 평이다.

일단 먼저, 질의 문과 사용빈도를 저장하는 frequency\_table이 있다고 가정하면, 처음엔 이 테이블이 비어있을 것이다.

사용자가 twitch , twitter, twit-ter, twillo를 순서대로 검색하면 다음과 같이 바뀐다.

		질의: twitch		질의: twitter		질의: twitter		질의: twillo	
질의	빈도	질의	빈도	질의	빈도	질의	빈도	질의	빈도
		twitch	1	twitch	1	twitch	1	twitch	1
				twitter	1	twitter	2	twitter	2
								twillo	1

그림 13-2

- 질의(query service)

주어진 질의에 다섯 개의 인기 검색어를 정렬해 보여주는 서비스다.

다음과 같은 사용빈도가 저장된 테이블이 있다고 치면,

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

표 13-1

- query : 질의문을 저장
- frequency : 질의문이 사용된 빈도를 저장

만약, rw를 검색하면 위로부터 상위 5개의 검색어가 검색된다.

```
SELECT * FROM frequency_table
WHERE query Like `prefix%`
ORDER BY frequency DESC
LIMIT 5;
```

tw
twitter
twitch
twilight
twin peak
twitch prime

그림 13-3

데이터 양이 많지 않을 때는 크게 나쁘지 않다. 하지만 데이터가 많아지면 병목이 발생할 수 있다.

상세설계안을 만들면서, 이 병목을 해결해보자

### 3단계 상세 설계

상세 설계안에서는 몇 가지 컴포넌트를 골라 최적화하는 방안을 논의한다.

논의할 컴포넌트는 trie 자료구조 / 데이터 수집 서비스 / 질의 서비스이다.

- 트라이(trie) 자료구조

위에서 소개한 rdbms를 통해 쿼리를 실행시켜 검색어를 가져오는 것은 크게 효율적이진 않다. 이 문제는 트라이(trie, 접두어 트리 prefix trie)를 사용해 해결할 것이다. 트라이가 시스템의 핵심 요소가 될 것이다.

트라이는 문자열들을 간략하게 저장할 수 있는 자료구조다. 트라이 자료구조의 핵심은 다음과 같다.

1. 트리 형태의 자료구조
2. 이 트리의 루트 노드는 빈 문자열을 나타낸다.
3. 각 노드는 글자 하나를 저장하며, 26개의 자식 노드(해당 글자 다음에 등장할 수 있는 모든 글자의 개수)를 가질 수 있다.
4. 각 트리 노드는 하나의 단어, 또는 접두어 문자열을 나타낸다.
5. 각 트리 노트는 하나의 단어, 또는 접두열 문자열을 나타낸다.

tree, try, true, toy, wish, win이 보관된 트라이를 다음 처럼 보여줄 수 있다.

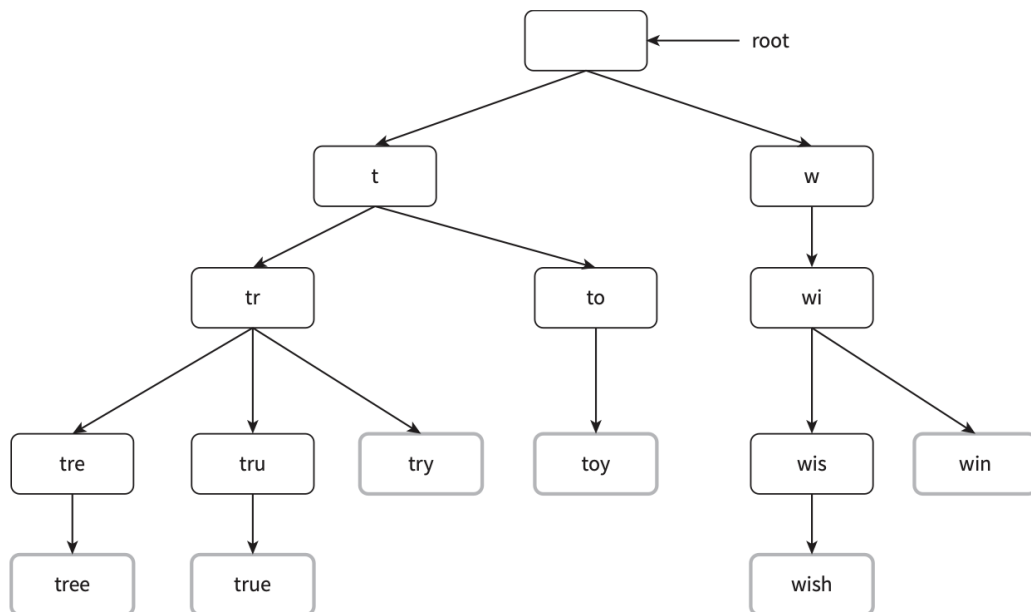


그림 13-5

기본 트라이 자료구조에서 노드에 문자들을 저장한다. 이용 빈도에 따라 저장된 결과를 내놓기 위해서는 노드에 빈도 정보까지 저장할 필요가 있다.

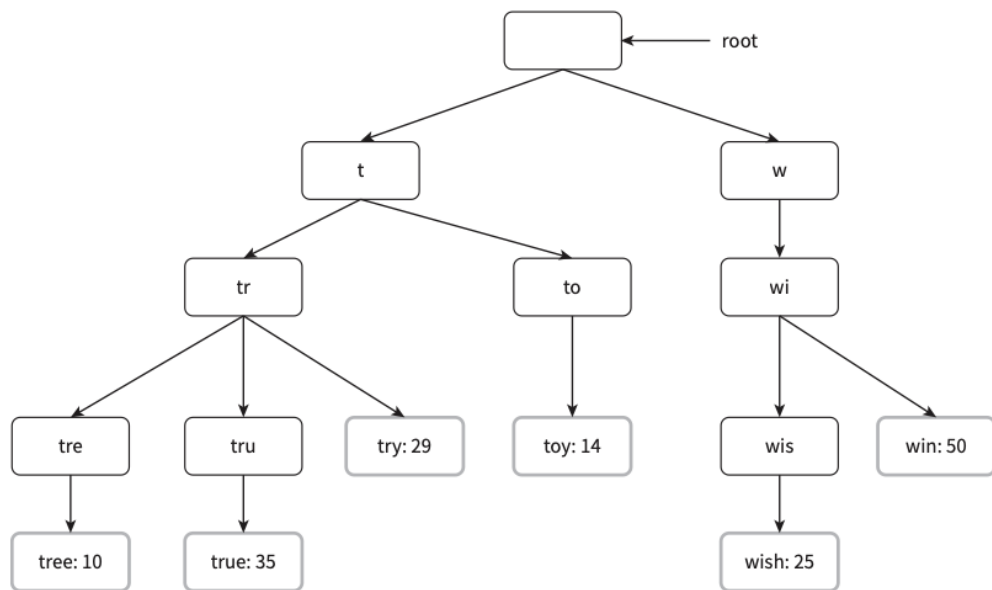


그림 13-6

자 여기까지 트라이의 기본 구조에 대해서 살펴볼 수 있었다. 이 트라이로의 자동완성은 어떻게 구현할 수 있을까?

그 전에 먼저 용어부터 정리해야 한다.

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

가장 많이 사용된 질의어 k는 다음과 같이 찾을 수 있다.

1. 해당 접두어를 표현하는 노드를 찾는다. 시간복잡도는  $O(p)$ 이다.
2. 해당 노드를 시작하는 하위 트리를 탐색하여 모든 유효 노드를 찾는다. 유효한 검색 문자열을 구성하는 노드가 유효 노드다. 시간 복잡도는  $O(c)$ 이다.
3. 유효 노드들을 정렬하여 인기있는 검색어 k를 찾는다. 시간 복잡도는  $O(c \log c)$ 이다.

만약, k = 2 이고 검색창에 be를 입력했다고 하면 다음과 같이 동작할 것이다.

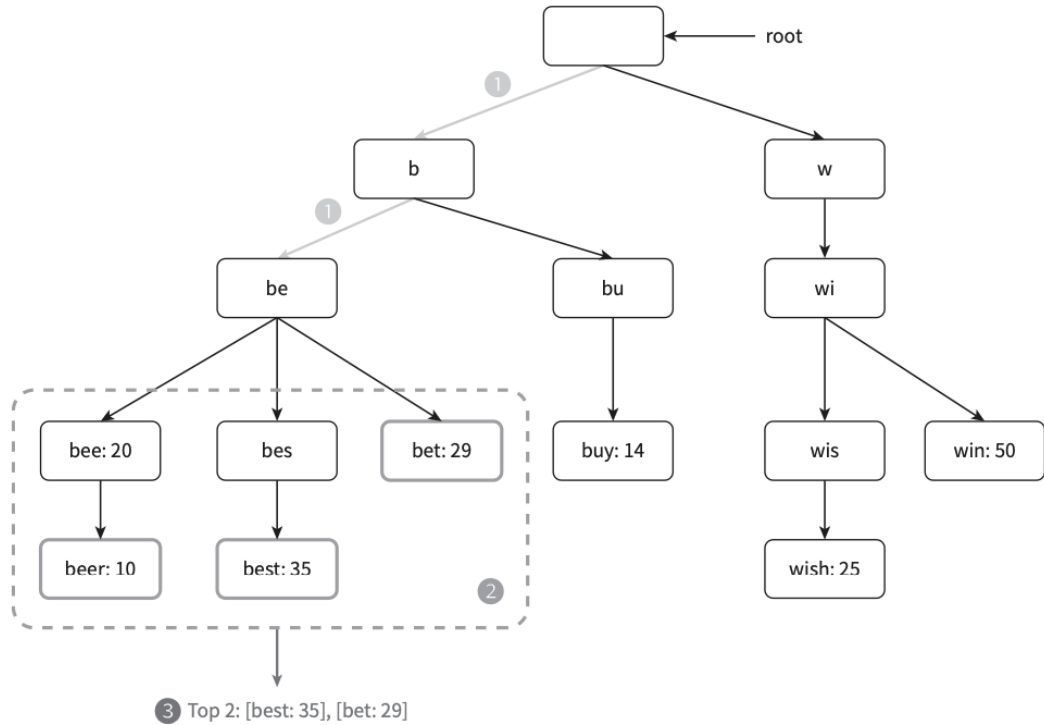


그림 13-7

1. 접두어 노드 be를 찾는다.
2. 해당 노드부터 시작하는 하위 트리를 탐색하여 모든 유효 노드를 탐색한다. 이 경우 유효 노드는  
beer: 10 / best: 35 / bet: 29이다.
3. 유효 노드를 정렬하여 2(=k)개만 골라낸다. best 35 / bet 29가 k값에 의하여 검색된 be에 대한 인기 검색어 2개이다.

이 알고리즘의 시간 복잡도는 각 단계마다 소요된 시간으로  $O(p) + O(c) + O(c \log c)$ 이다.

직관적이고 좋은 알고리즘이지만, 만약 k개 결과를 얻으려고 전체 트라이를 다 검색해야 할 수도 있다. 따라서 몇 가지 방안을 모색해야 하는데, 해결책으로는

- 접두어 최대 길이 제한
- 각 노드에 인기 검색어 캐시

가 있다. 두 가지 최적화 방안을 마저 탐색하면,

### 접두어 최대 길이 제한

검색창에 긴 검색어를 입력하는 일은 잘 없다. 따라서  $p$  값은 작은 정수 값(고정된 임의의 값)으로 설정해도 괜찮을 것이다.

시간 복잡도는  $O(p)$ 에서  $\Rightarrow O = O(1)$ 로 바뀔 것이다.

### 노드에 인기 검색어 캐시

각 노드마다 인기 검색어  $k$ 를 저장해 둔다면, 전체 트라이를 검색하는 비효율을 막을 수 있다. 대신 전체 트라이를 막는 효율을 낼 수 있지만 노드 당 질의어를 캐시할 공간이 많이 필요하게 된다는 단점도 있다.

이 부분은 빠른 정답속도와 저장 공간 관리의 비용 중 효율을 고려하여 사용하면 좋다.

아래의 그림은 개선된 트라이 구조다. 각 노드 당 인기 검색어 다섯 가지를 저장하도록 구성했다.

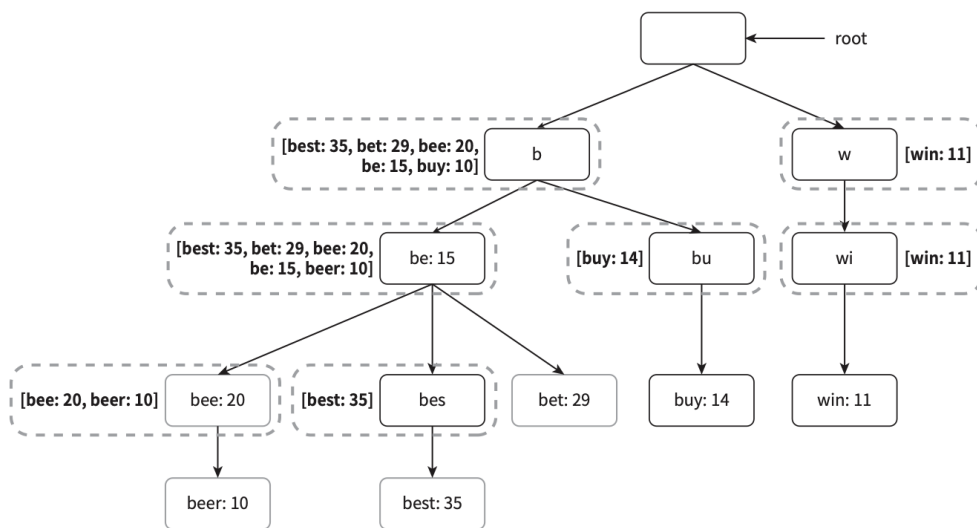


그림 13-8

각 단계의 시간 복잡도가  $O(1)$ 로 바뀐 덕분에, 최고 인기 검색어를 찾는 전체 알고리즘의 복잡도도  $O(1)$ 로 변경된다.

- 데이터 수집 서비스

위 설계안들은 검색창에 타이핑 할 때마다 실시간으로 데이터를 수정했다. 이 방법은 다음과 같은 두 문제가 존재한다.



- 매일 수천만 건의 질의가 검색되는 데 그때마다 트라이를 갱신하면 서비스의 사용성이 나빠진다.
- 트라이가 만들어지고 나면 인기 검색어는 그다지 자주 바뀌지 않을 것이다. 그렇기 때문에 트라이는 자주 갱신될 필요가 없다.

실시간 성이 짙은 트위터라면 제안 검색어를 항상 신선하게 유지할 필요가 있을 수 있겠지만, 구글 검색 같은 Application은 그렇게 자주 바뀌질 필요가 없다.

따라서 용례에 따라 서비스를 설계하면 된다. 아래는 수정된 데이터 분석 서비스의 설계안이다.

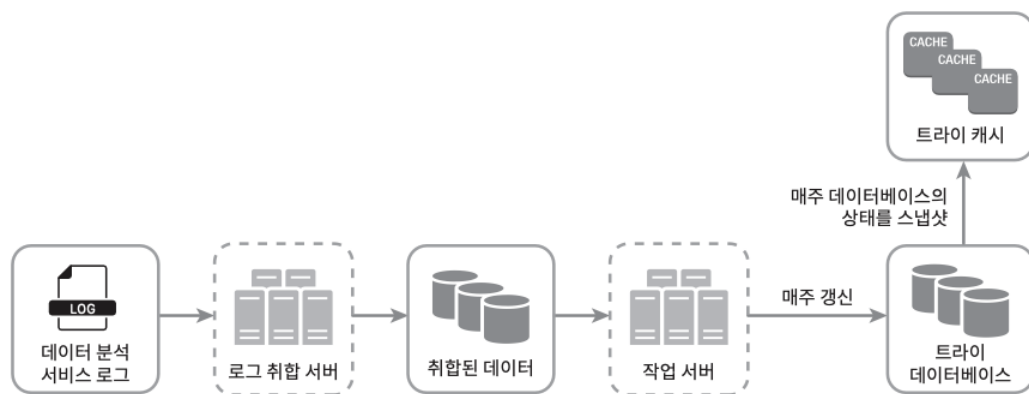


그림 13-9

컴포넌트를 하나씩 살펴보면,

- 데이터 분석 서비스 로그

query	time
tree	2019-10-01 22:01:01
try	2019-10-01 22:01:05
tree	2019-10-01 22:01:30
toy	2019-10-01 22:02:22
tree	2019-10-02 22:02:42
try	2019-10-03 22:03:03

**표 13-3**

질의에 관한 원본 로그가 기록된다. 새로운 insert만 있고 update는 없다. 로그 데이터에 인덱스를 걸지도 않는다.

- 로그 취합 서버

보통 로그는 그 양이 엄청나고, 데이터 형식도 제각각인 경우가 많다. 따라서 데이터를 잘 취합하여 시스템이 소비할 수 있도록 해야 한다.

데이터 취합의 경우 어떤 주기로 진행할 지는 시스템에 따라 다르기 때문에 요구사항에 따른 스펙을 정하는 것이 좋다.

- 취합된 데이터

취합된 데이터는 데이터베이스에 보관된다.

- 작업 서버

주기적으로 비동기 작업(batch)을 실행하는 서버 집합이다. 트라이 자료구조를 만들고 데이터베이스에 저장한다.

○ 트라이 데이터베이스

분산 캐시 시스템으로서, 트라이 데이터를 메모리에 유지하여 읽기 연산 성능을 높이는 구실을 한다. 매주 스냅샷을 떼서 갱신한다.

○ 트라이 캐시

지속적 저장소다. 트라이 데이터베이스로 사용할 만한 선택지는 두 가지다.

1. 문서 저장소(document store) : 몽고 디비나 다이نام오 디비같은 문서기반 저장소에 주기적으로 트라이를 직렬화하여 저장한 후 사용할 수 있다.
2. 키-값 저장소: 아래 로직을 적용하면 해시 테이블 형태로 변환할 수 있다.
  - a. 트라이에 보관된 모든 접두어를 해시 테이블 키로 변환
  - b. 각 트라이 노드에 보관된 모든 데이터를 해시 테이블 값으로 변환

예를 들면, 이런식이다.

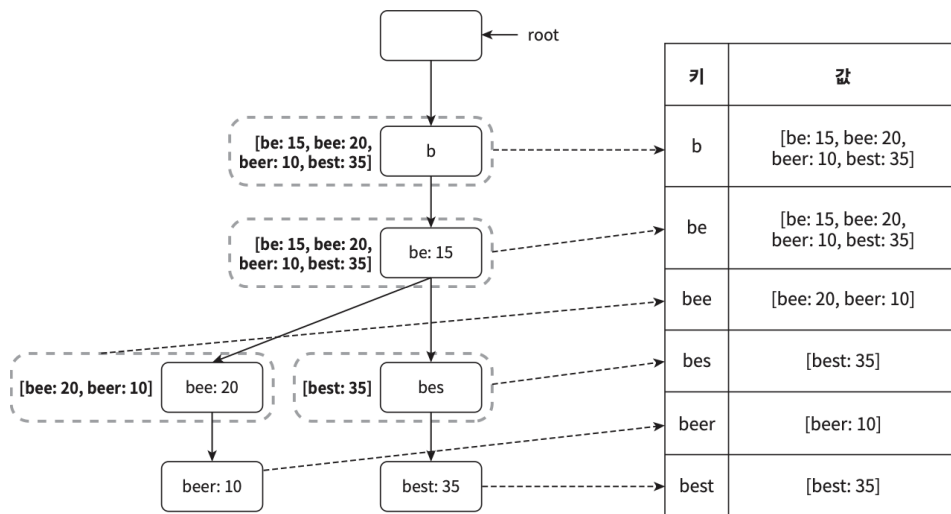


그림 13-10

● 질의 서비스

다음은 여태껏 살펴본 과정을 통해 나온 설계도다.

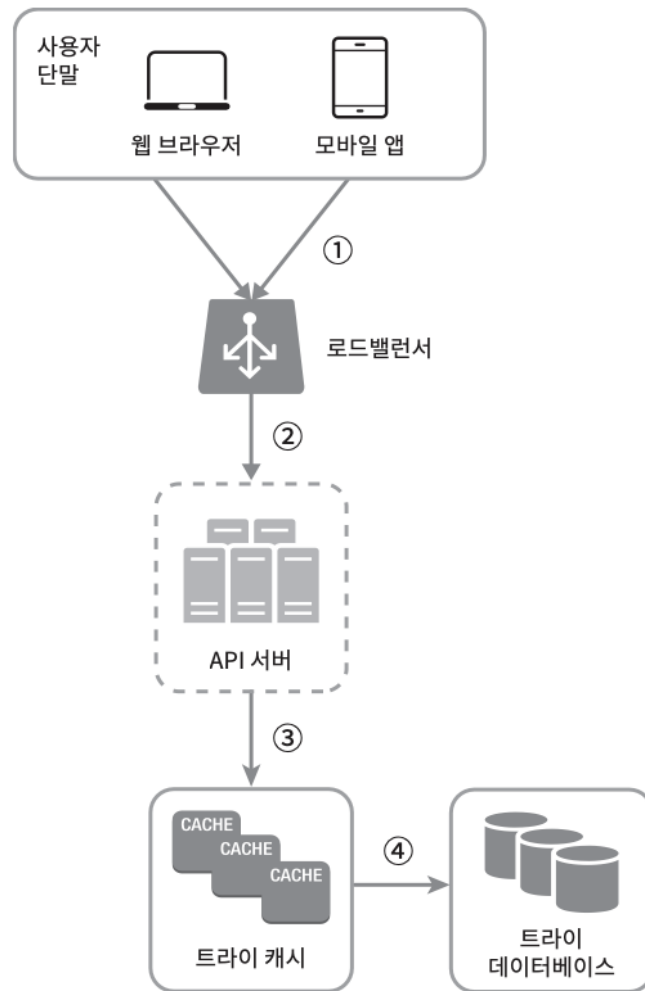


그림 13-11

순서를 살펴보면 이렇다.

1. 검색 질의가 로드 밸런서로 전송된다.
2. 로드 밸런서는 해당 질의를 API 서버로 보낸다.
3. API 서버는 트라이 캐시에서 데이터를 가져와 해당 요청에 대한 자동완성 검색어 제안 응답을 구성한다.
4. 데이터가 트라이 캐시에 없는 경우 데이터를 데이터베이스에서 가져와 캐시에 채운다. 이렇게 해야만 다음 같은 질의가 들어왔을 때 캐시에 보관된 데이터를 사용할 수 있다.

검색 질의는 정말 빨라야 하는데, 다음과 같은 방법들은 최적화를 위해 사용될 수 있다.

- Ajax 요청

ajax요청의 장점은, 브라우저를 새롭게 새로고침할 필요가 없다는 점이다. 이게 무슨 소리냐면, 리액트처럼 렌더링 새롭게 하지 않고 필요한 데이터만 비동기로 가져와 업데이트 한다는 것이다.

- 브라우저 캐싱

대부분의 경우 자동완성 검색어 제안 결과는 짧은 시간에 자주 바뀌지 않는다. 따라서 제안된 검색어들을 브라우저 캐시에 넣어두면, 후속 질의 결과는 해당 캐시에서 바로 사용할 수 있다. 구글 검색 엔진이 이런 캐시 매커니즘을 사용한다.

- 데이터 샘플링

대규모 시스템에서 모든 질의 결과를 로깅하도록 해 놓으면 CPU 자원과 저장공간 및 리소스 사용이 극심해진다. 데이터 샘플링 기법은 유용하다.

즉, N개 요청 가운데 1개만 로깅하도록 하는 것이다.

- **트라이 연산**

- 트라이 생성

트라이는 검색어 자동완성 시스템의 핵심 컴포넌트다.

트라이 생성은 작업 서버가 담당하며, 데이버 분석 서비스의 로그나 데이터베이스로부터 취합된 데이터를 사용한다.

- 트라이 갱신

갱신은 다음과 같은 두 가지 방법이 있다.

1. 매주 한 번 갱신
2. 트라이 각 노드를 개별적으로 갱신 ⇒ 성능이 좋지 않다. 트라이가 작을 때는 고려해볼 만 하다.

- 검색어 삭제

악의적인 질의어를 검색어에서 제거해야 한다. 한 가지 좋은 방법은 필터 계층을 만들어, 필터 규칙에 따라 질의 결과를 정렬하는 것이다.

데이터 베이스 상의 물리적 삭제는 다음 번 업데이트 사이클에 비동기적으로 진행하면 된다.

- 저장소 규모 확장

간단하게는 영어만 지원하기 때문에 첫 글자 기준으로 샤딩하는 방법을 생각할 수 있다.

알파벳 첫 글자 순으로 샤딩을 진행하게 되면, 최대 26개의 서버가 생길 수 있다.

## 4단계 마무리

상세 설계를 마치고, 조금 더 고민해볼만 한 주제는

- 다국어 지원

다국어를 지원하기 위해서는 트라이에 유니토크들 저장해야 한다.

유니코드는 고금을 막론하고 세상에 존재하는 모든 문자체계를 지원한다.

- 국가별 인기 검색어가 다르다면.. ?

국가별로 다른 트라이를 사용해 CDN으로 응답 속도를 높일 수 있다.

- 실시간으로 변하는 검색 추이를 반영하려면 .. ?

위에서 설계한 검색 시스템은 실시간에 적합하지 않은데, 그 이유는 작업 서버가 배치 스케줄링으로 트라이를 관리하는 것도 있고 새롭게 트라이를 구성하는 것이 어렵기 때문이다.

다만 도움이 될 만한 몇 가지 아이디어는 있다.


1. 샤딩을 통한 작업 데이터 양 줄이기
2. 데이터를 스트림 형태로 가져올 수 있다는 점으로, 한 번에 모든 데이터를 동시에 사용할 수 없을 가능성이 있다는 점을 고려해야 한다.

데이터가 지속적으로 생성된다는 의미로서, 비디오 스트리밍하는 데이터도 스트리밍 해야 한다.

아파치 하둡 맵리듀스, 아파치 스파크 스트리밍, 아파치 카프카 등이 이런 시스템이다.

[가상 면접 사례로 배우는 대규모 시스템 설계 기초] 13장 검색어 자동완성 시스템

구글 검색 또는 아마존 웹 사이트 검색창 등..단어를 입력하면 k개의 검색어가 자동으로 완성 되는 검색어 자동완성 시스템을 설계해보자1시스템 설계 면접 문제는 의도적으로 어떤 정해진 결말을 갖지 않도록 만들어진다. 따라서 면접장에서 시스템을 성공적으로 설계해 내려면

 <https://velog.io/@tkdals0978/가상-면접-사례로-배우는-대규모-시스템-설계-기초-13장-검색어-자동완성-시스템>

