13장. 검색어 자동완성 시스템

1단계 | 문제 이해 및 설계 범위 확정

요구사항

- 빠른 응답 속도 : 사용자가 검색어를 입력함에 따라 자동완성 검색어도 충분히 빠르게 표시되어야 한다. 페이스북의 경우 응답속도를 100 밀리초 이내라고 규정한다. 그렇지 않으면 시스템 이용이 불편해진다.
- 연관성 : 검색어는 사용자가 입력한 단어와 연관성이 있어야 한다.
- 정렬 : 시스템의 계산 결과는 인기도 등의 순위 모델에 의해 정렬되어야 한다.
- 규모 확장성 : 시스템은 많은 트래픽을 감당할 수 있도록 확장 가능해야 한다.
- 고가용성 : 시스템의 일부에 장애가 발생하거나 느려지거나, 예상치 못한 네트워크 문제가 발생해도 시스템은 계속 사용 가능해야 한다.

개략적 규모 추정

- DAU는 천만 명으로 가정
- 평균적으로 한 사용자는 매일 10건의 검색을 수행한다고 가정
- 질의 할 때 평균적으로 20바이트의 데이터를 입력한다고 가정
  - 문자 인코딩 방법은 ASCII를 사용한다고 가정하면 1문자 = 1바이트이다.
  - 질의문은 평균적으로 4개의 단어로 이뤄진다고 가정, 각 단어는 평균적으로 다섯 글자로 구성된다.
  - 따라서 질의당 평균 4 x 5 = 20 바이트이다.
- 검색창에 글자를 입력할 때마다 클라이언트는 백엔드 서버로 요청을 보낸다. 따라서 평균적으로 1회 검색당 20건의 요청이 백엔드로 전달된다. 예를 들어 검색창에 dinner라고 입력하면 아래처럼 순차적으로 6개의 요청이 생긴다.
  - search?q=d
  - search?q=di
  - search?q=din
  - 등등..
- 대략 초당 24,000건의 질의 발생 = (10,000,000 x 10 질의 / 일 x 20자 / 24시간 / 3600초)
- 질의 가운데 20% 정도는 신규 검색어라고 가정. 따라서 대략 0.4GB 정도

2단계 | 개략적 설계안 제시 및 동의 구하기

개략적으로 보면 시스템은 두 부분으로 나뉨

- 데이터 수집 서비스: 사용자가 입력한 질의를 실시간으로 수집하는 시스템이다. 데이터가 많은 애플리케이션에 실시간 시스템은 그다지 바람직하지 않지만 설계안을 만드는 출발점으로는 괜찮을 것이다.
- 질의 서비스: 주어진 질의에 다섯 개의 인기 검색어를 정렬해 내놓는 서비스이다.

데이터 수집 서비스

질의문과 사용빈도를 저장하는 빈도 테이블이 있다고 가정한다. 처음에 이 테이블은 비어 있는데, 사용자가 twitch, twitter, twitter, twillo를 순서대로 검색하면 아래와 같이 바뀌어 나간다.

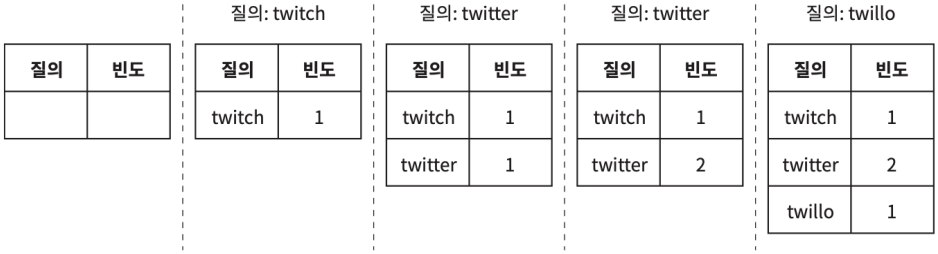

그림 13-2

질의 서비스

아래와 같은 빈도 테이블이 있는 상태라고 하자. 두 개의 필드가 있음을 볼 수 있을 것이다.

- 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

표 13-1

이 상태에서 사용자가 "tw"를 검색창에 입력하면 아래의 "top5" 자동완성 검색어가 표시되어야 한다.

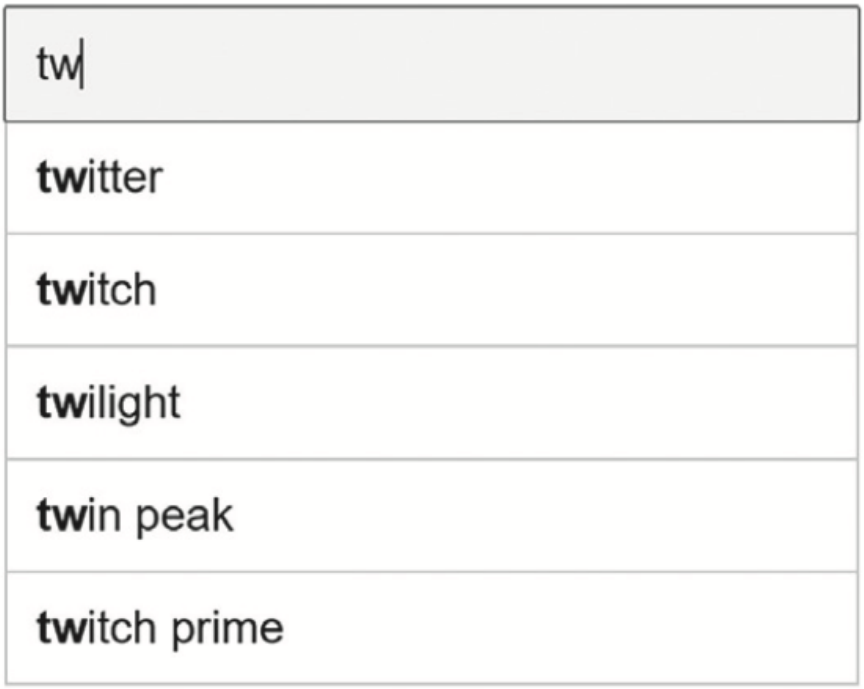

그림 13-3

- 가장 많이 사용된 5개 검색어는 아래의 SQL 질의문을 사용해 계산할 수 있다.

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

데이터 양이 적을 때는 나쁘지 않은 설계안이다. 하지만 데이터가 아주 많아지면 데이터베이스가 병목이 될 수 있다. 상세 설계안을 준비하면서 이 문제를 해결할 방법을 알아보겠다.

3단계 | 상세 설계

개략적 설계안에서 다음 순서로 최적화 방안을 논의할 것이다.

- 트라이(trie) 자료구조
- 데이터 수집 서비스
- 질의 서비스
- 규모 확장이 가능한 저장소
- 트라이 연산

트라이 자료구조

개략적 설계안에서는 관계형 데이터베이스를 저장소로 사용했다. 하지만 관계형 데이터베이스를 이용해 가장 인기 있었던 다섯 개 질의문을 골라내는 방안은 효율적이지 않다. 이 문제는 트라이(trie) 를 사용해 해결할 것이다.

트라이 자료구조란?

트라이는 문자열들을 간략하게 저장할 수 있는 자료구조다.
트라이라는 이름은 "retrieval"이라는 단어에서 온 것인데, 문자열을 꺼내는 연산에 초점을 맞추어 설계된 자료구조임을 미루어 짐작할 수 있다.

- 자료구조의 핵심 아이디어
  - 트리 형태의 자료구조
  - 루트 노드는 빈 문자열을 나타냄
  - 각 노드는 글자 하나를 저장하며 26개의 자식 노드를 가질 수 있음
  - 각 트리 노드는 하나의 단어, 또는 접두어 문자열을 나타냄
  아래는 `tree`, `try`, `true`, `toy`, `wish`, `win` 가 보관된 트라이

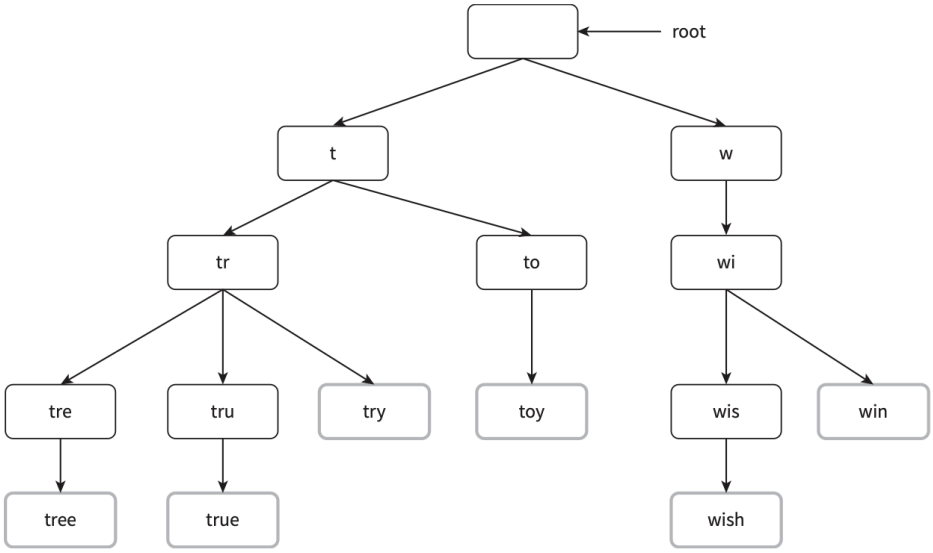

그림 13-5

이용 빈도에 따라 정렬된 결과를 내놓기 위해서 노드에 빈도 정보까지 저장할 필요가 있다.

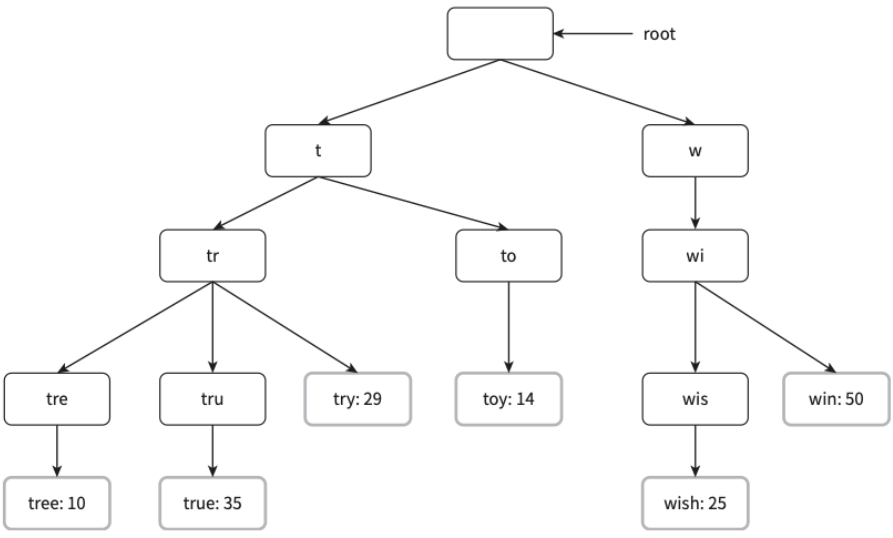

그림 13-6

해당 트라이로 검색어 자동완성을 어떻게 구현할 수 있을까?

- p: 접두어(prefix)의 길이

- n: 트라이 안에 있는 노드 개수
- c: 주어진 노드의 자식 노드 개수
  가장 많이 사용된 질의어 k개는 다음과 같이 찾을 수 있다.
- 해당 접두어를 표현하는 노드를 찾는다. 시간복잡도 O(p)
- 해당 노드부터 시작하는 하위 트리를 탐색하여 모든 유효 노드를 찾는다. 시간복잡도 O(c)
- 유효 노드들을 정렬하여 가장 인기 있는 검색어 k개를 찾는다. 시간복잡도 O(clogc)
  예시) k=2이고 사용자가 검색창에 'be'을 입력

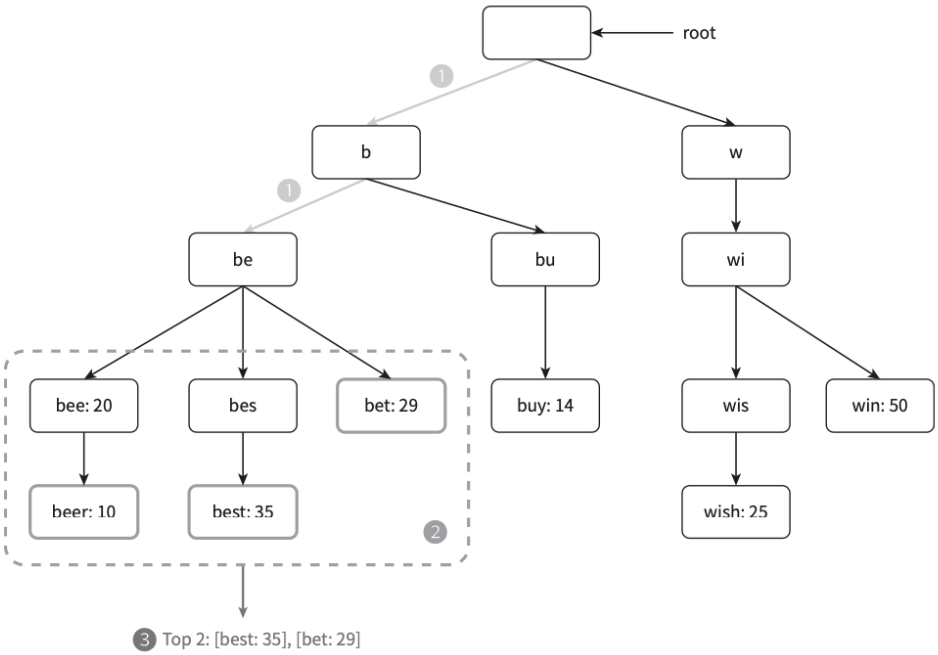

그림 13-7

1. 접두어 노드 `be`를 찾는다.
2. 해당 노드부터 시작하는 하위 트리를 탐색하여 모든 유효 노드를 찾는다. [beer: 10], [best: 35], [bet: 29]
3. 유효 노드를 정렬하여 2개만 골라낸다. [best: 35], [bet: 29]
   이 알고리즘의 시간 복잡도: **O(p) + O(c) + O(clogc)**
   이 알고리즘은 직관적이지만 최악의 경우에는 k개 결과를 얻으려고 전체 트라이를 다 검색해야 하는 일이 생길 수 있다. 이 문제를 해결할 방법으로는 다음의 두 가지가 있다.
4. 접두어의 최대 길이를 제한
5. 각 노드에 인기 검색어를 캐시

접두어 최대 길이 제한

사용자가 검색창에 긴 검색어를 입력하는 일은 거의 없다. 따라서 p값은 작은 정숫값이라고 안전하다. 검색어의 최대 길이를 제한할 수 있다면 "접두어 노드를 찾는" 단계의 시간 복잡도는 **O(p)에서 O(작은 상숫값) = O(1)** 로 바뀔 것이다.

노드에 인기 검색어 캐시

각 노드에 k개의 인기 검색어를 저장해두면 전체 트라이를 검색하는 일을 방지할 수 있다. 각 노드에 질의어를 저장할 공간이 많이 필요하게 된다는 단점이 있지만, 빠른 응답 속도가 아주 중요할 때는 희생

할 만한 가치가 있다.

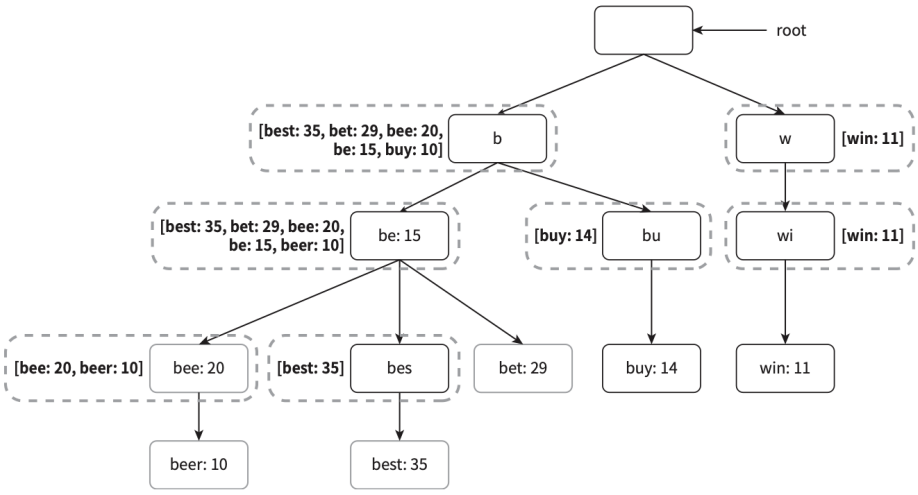

그림 13-8

1. 접두어 노드를 찾는 시간 복잡도는 O(1)로 바뀐다.
2. 최고 인기 검색어 5개를 찾는 질의의 시간 복잡도도 O(1)로 바뀐다. 검색 결과가 이미 캐시되어 있어서다.
   각 단계의 시간 복잡도가 O(1)로 바뀐 덕분에, 최고 인기 검색어 k개를 찾는 전체 알고리즘의 복잡도도 O(1)로 바뀌게 된다.

데이터 수집 서비스

지금까지의 설계안은 사용자가 검색창에 뭔가 타이핑을 할 때마다 실시간으로 데이터를 수정했다.

- 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 트라니느 그렇게 자주 갱신할 필요가 없다.
  트위터같은 실시간 애플리케이션이라면 항상 신선하게 유지할 필요가 있지만, 구글과 같은 검색 애플리케이션의 경우 그렇게 자주 바꿔줄 이유는 없다.
  아래는 데이터 분석 서비스의 수정된 설계안이다. 지금부터 각 컴포넌트를 차례대로 살펴보겠다.

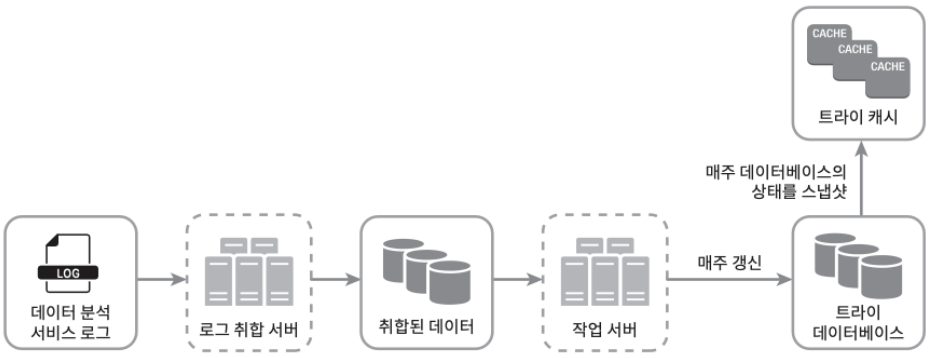

그림 13-9

데이터 분석 서비스 로그

검색창에 입력된 질의에 관한 원본 데이터가 보관된다.
데이터가 추가될 뿐 수정은 이루워지지 않으며 로그 데이터에는 인덱스를 걸지 않는다.

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

로그 취합 서버

트위터와 같은 실시간 애플리케이션 → 데이터 취합 주기를 짧게 가져가는 것이 좋음
대부분의 경우 → 일주일에 한 번 정도로 로그를 취합

취합된 데이터

매주 취합한 데이터의 사례다. time 필드는 해당 주가 시작한 날짜를 나타낸다. frequency 필드는 해당 질의가 해당 주에 사용된 횟수의 합이다.

query	time	frequency
tree	2019-10-01	12000
tree	2019-10-08	15000
tree	2019-10-15	9000
toy	2019-10-01	8500
toy	2019-10-08	6256
toy	2019-10-15	8866

표 13-4

작업 서버

작업 서버는 주기적으로 비동기적 작업을 실행하는 서버 집합이다. 트라이 자료구조를 만들고 트라이 데이터베이스에 저장하는 역할을 담당한다.

트라이 캐시

트라이 캐시는 분산 캐시 시스템으로 트라이 데이터를 메모리에 유지하여 읽기 연산 성능을 높이는 구실을 한다. 매주 트라이 데이터베이스의 스냅샷을 떠서 갱신한다.

트라이 데이터베이스

트라이 데이터베이스는 지속성 저장소다. 트라이 데이터베이스로 사용할 수 있는 선택지로는 다음의 두 가지가 있다.

1.
   - 문서 저장소
   - 새 트라이를 매주 만들 것이므로 주기적으로 트라이를 직렬화하여 데이터베이스에 저장할 수 있음
   - 몽고디비 같은 문서 저장소를 활용 가능
2. 키-값 저장소
   - 해시 테이블 형태로 변환 가능
   - 트라이에 보관된 모든 접두어를 해시 테이블 키로 변환
   - 각 트라이 노드에 보관된 모든 데이터를 해시 테이블 값을 변환

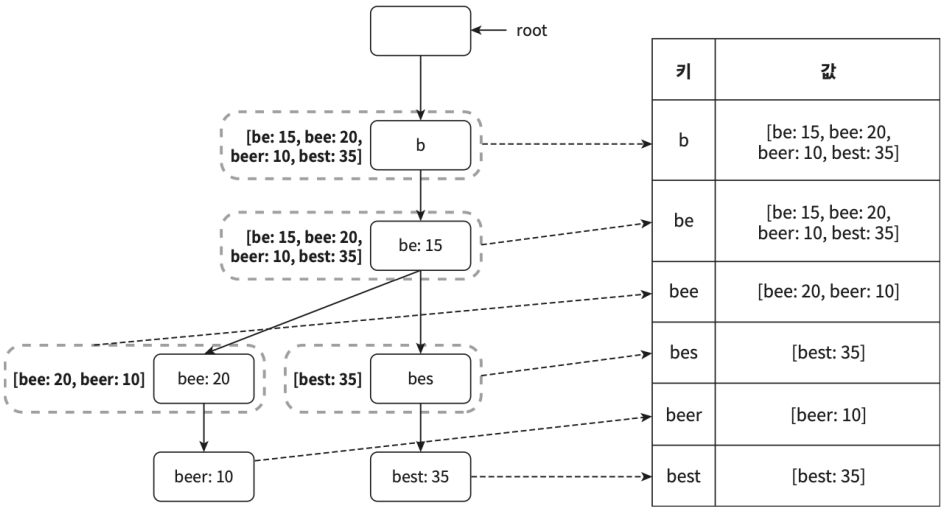

그림 13-10

질의 서비스

개략적 설계안에서 살펴본 질의 서비스는 데이터베이스를 활용하여 최고 인기 검색어 다섯 개를 골라냈다. 아래 그림은 해당 설계안의 비효율성을 개선한 새 설계안이다.

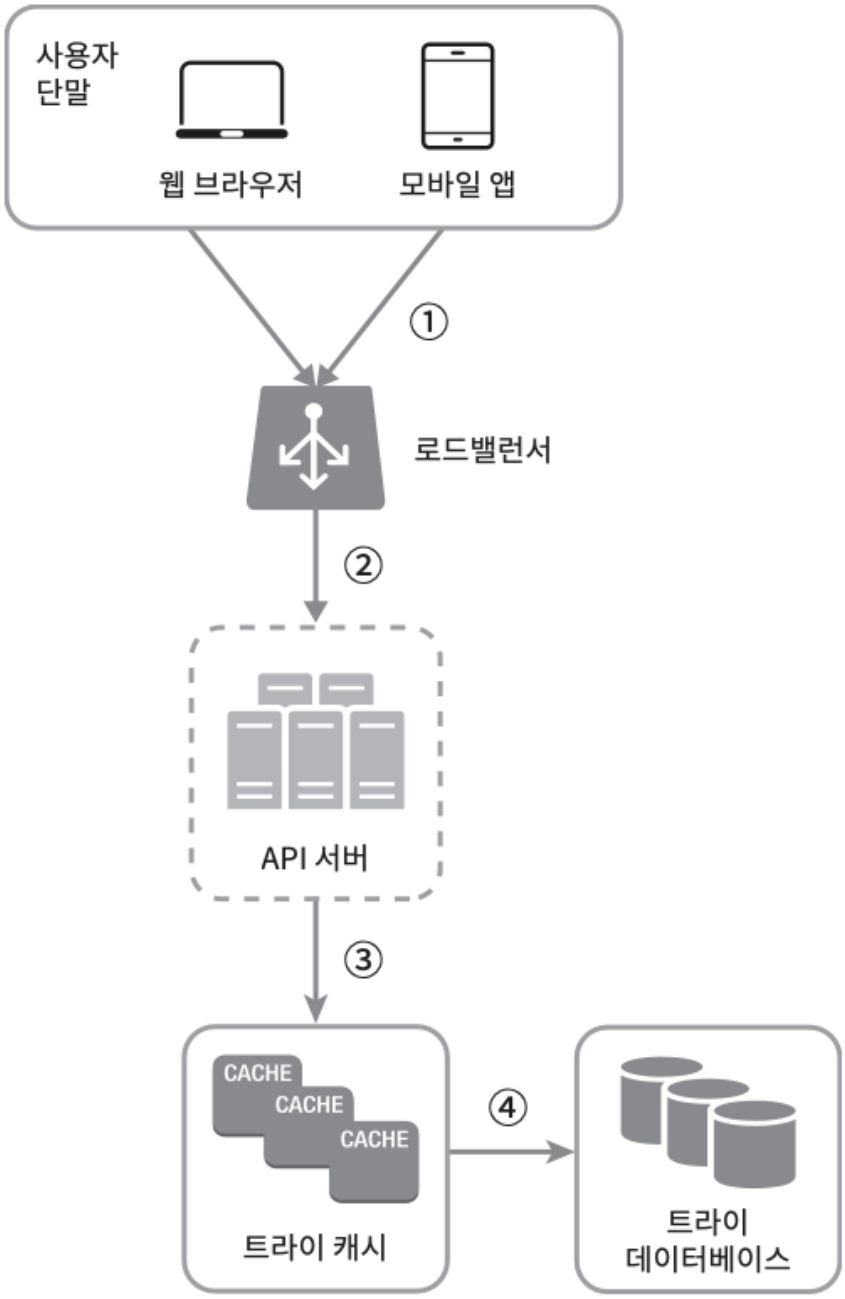

그림 13-11

1. 검색 질의가 로드밸런서로 전송된다.
2. 로드밸런서는 해당 질의를 API 서버로 보낸다.
3. API 서버는 트라이 캐시에서 데이터를 가져와 자동완성 검색어 제안 응답을 구성한다.
4. 데이터가 트라이 캐시에 없는 경우 트라이 데이터베이스에서 가져와 캐시에 채운다. 그래야 다음에 같은 접두어에 대한 질의가 오면 캐시에 보관된 데이터를 사용해 처리할 수 있다. 캐시 미스는 캐시 서버의 메모리가 부족하거나 캐시 서버에 장애가 있어도 발생할 수 있다.

질의 서비스는 번개처럼 빨라야 한다. 이를 위해 다음과 같은 최적화 방안을 생각해 보기를 제안한다.

- AJAX 요청
  - 웹 애플리케이션의 경우 브라우저는 보통 AJAX 요청을 보내어 자동완성된 검색어 목록을 가져온다. 이 방법의 장점은 요청을 보내고 받기 위해 페이지를 새로고침 할 필요가 없다는

것이다.

- 브라우저 캐싱
  - 대부분 애플리케이션의 경우 자동완성 검색어 제안 결과는 짧은 시간 안에 자주 바뀌지 않는다. 따라서 제안된 검색어들을 브라우저 캐시에 넣어두면 후속 질의의 결과는 해당 캐시에서 바로 가져갈 수 있다.
- 데이터 샘플링
  - 대규모 시스템의 경우, 모든 질의 결과를 로깅하도록 해 놓으면 CPU 자원과 저장공간을 엄청나게 소진하게 된다. 데이터 샘플링 기법은 그럴 때 유용하다. 즉, N개의 요청 가운데 1개만 로깅하도록 하는 것이다.

트라이 연산

트라이는 검색어 자동완성 시스템의 핵심 컴포넌트다. 지금부터 트라이 관련 연산들이 어떻게 동작하는지 살펴보자.

트라이 생성

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

트라이 갱신

트라이를 갱신하는 데는 두 가지 방법이 있다.

1. 매주 한번 갱신하는 방법
   - 새로운 트라이를 만든 다음에 기존 트라이를 대체한다.
2. 트라이의 각 노드를 개별적으로 갱신하는 방법
   - 본 설계안에서는 이 방법을 채택하지 않았는데, 성능이 좋지 않아서다. 트라이 노드를 갱신할 때는 그 모든 상위 노드도 갱신해야 하는데, 상위 노드에도 인기 검색어 질의 결과가 보관되기 때문이다. 따라서 트라이가 작을 때는 고려해봄직한 방안이다.

검색어 삭제

혐오성, 폭력적 등.. 여러 가지로 위험한 질의어를 자동완성 결과에서 제거해야 한다. 트라이 캐시 앞에 필터 계층을 두고 부적절한 질의어가 반환되지 않도록 할 수 있다.

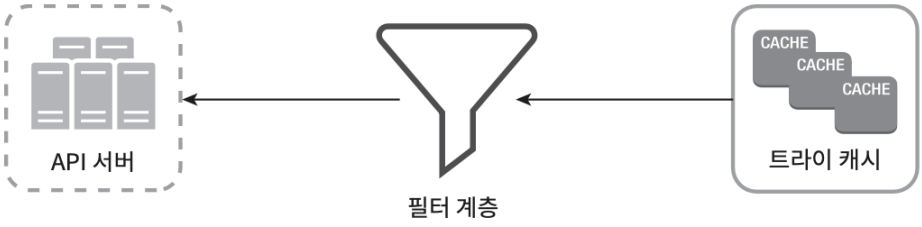

그림 13-14

저장소 규모 확장

트라이의 크기가 한 서버에 넣기엔 너무 큰 경우 대응할 수 있도록 규모 확장성 문제를 해결해보자.
영어만 지원하면 되기 때문에, 간단하게는 첫 글자 기준으로 샤딩하는 방법을 생각해 볼 수 있다.

- **두 대의 서버가 필요한 경우**
  'a'부터 'm'까지로 시작하는 검색어는 첫 번째 서버에 저장하고, 나머지는 두 번째 서버에 저장한다.
- **세 대의 서버가 필요한 경우**
  'a'부터 'i'까지는 첫 번째 서버에, 'j'부터 'r'까지는 두 번째 서버에 저장하고, 나머지는 세 번째 서버에 저장한다.
  이 방법을 쓰는 경우에는 사용 가능한 서버는 최대 26개로 제한되는데, 영어 알파벳에는 26자 밖에 없기 때문이다. 이 이상으로 서버 대수를 늘리려면 샤딩을 계층적으로 해야 한다.
- 'a'로 시작하는 검색어를 네 대의 서버에 나눠서 보관하고 싶다면
  `'aa'~'ag'`, `'ah'~'an'`, `'ao'~'au'`, `나머지`로 4개의 서버로 나누어 보관하면 된다.
  하지만 'c'로 시작하는 단어가 'x'로 시작하는 단어보다 많다는 것을 감안하면 좋은 방법이 아니다.

이 문제를 해결하기 위해, 과거 질의 데이터의 패턴을 분석하여 샤딩하는 방법을 제안한다.

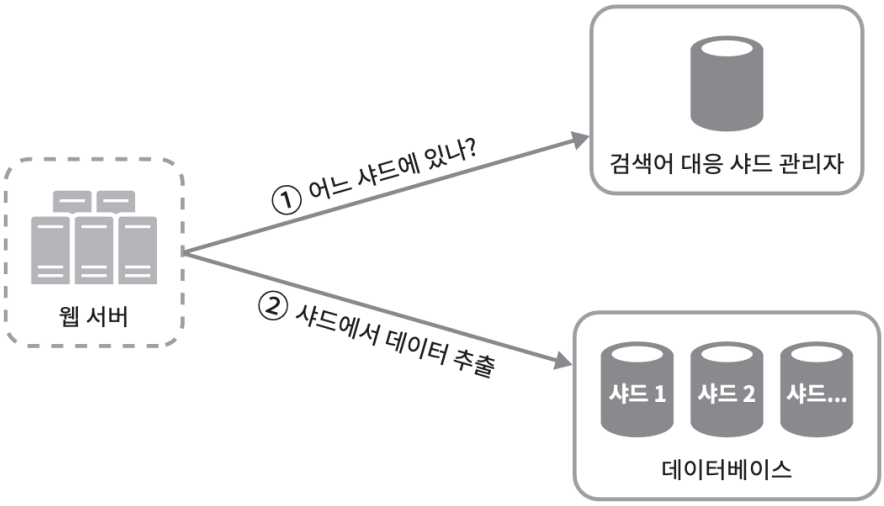

그림 13-15

검색어 대응 샤드 관리자는 어떤 검색어가 어느 저장소 서버에 저장되는지에 대한 정보를 관리한다.
's'로 시작하는 검색어의 양이 'u', 'v', 'w', 'x', 'y', 'z'로 시작하는 검색어를 전부 합친 것과 비슷하다면 `'s'`에 대한 샤드 하나와 `'u'~'z'`까지의 검색어를 위한 샤드 하나를 두어도 충분할 것이다.

4단계 | 마무리

- 실시간 검색어 자동완성 시스템을 구축할 때 도움될 만한 아이디어
  - 샤딩을 통하여 작업 대상 데이터의 양을 줄인다.
  - 순위 모델을 바꾸어 최근 검색어보다 높은 가중치를 주도록 한다.

- 데이터가 스트림 형태로 올 수 있다는 점, 즉 한번에 모든 데이터를 동시에 사용할 수 없을 가능성이 있다는 점을 고려해야 한다. 데이터가 스트리밍된다는 것은, 즉 데이터가 지속적으로 생성된다는 뜻이다. 아파치 하둡 맵리듀스, 아파치 스파크 스트리밍, 아파치, 스톰, 아파치 카프카 등이 그런 부류의 시스템이다.