13장. 검색어 자동완성 시스템

Created	@November 24, 2022 11:40 AM
Progress	In Progress

📃 **학습 TODO list**

☐

8.1. 1단계: 문제 이해 및 설계 범위 확정

> *질문을 통해 요구사항을 알아내고 설계 범위를 좁힌다.*

👩‍💻 사용자가 입력하는 단어는 자동완성될 검색어의 첫 부분이어야 하나요? 아니면 중간 부분이 될 수도 있습니까?

👨‍💻 첫 부분으로 한정하겠습니다.

👩‍💻 몇 개의 자동완성 검색어가 표시되어야 합니까?

👨‍💻 5개 입니다.

👩‍💻 자동완성 검색어 5개를 고르는 기준은 무엇입니까?

👨‍💻 질의 빈도에 따라 정해지는 검색어 인기 순위를 기준으로 삼겠습니다.

👩‍💻 맞춤법 검사 기능도 제공해야 합니까?

👨‍💻 아뇨. 맞춤법 검사나 자동수정은 지원하지 않습니다.

👩‍💻 질의는 영어입니까?

👨‍💻 네. 하지만 시간이 허락한다면 다국어 지원을 생각해도 좋습니다.

👩‍💻 대문자나 특수 문자 처리도 해야 합니까?

👨‍💻 아뇨. 모든 질의는 영어 소문자로 이루어진다고 가정하겠습니다.

👩‍💻 얼마나 많은 사용자를 지원해야 합니까?

일간 능동 사용자(DAU) 기준으로 천만(10 million) 명입니다.

8.1.1. 요구사항

- 빠른 응답 속도
- 연관성
- 정렬
- 규모 확장성
- 고가용성

8.1.2. 개략적 규모 추정

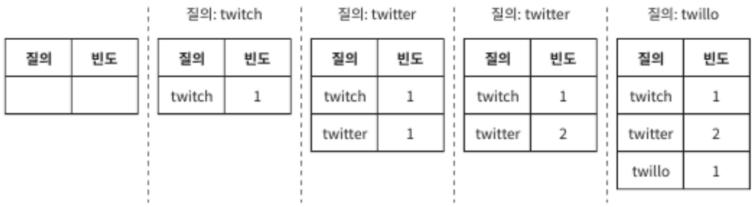
- 일간 능동 사용자(DAU)는 천만 명으로 가정한다.
- 평균적으로 한 사용자는 매일 10건의 검색을 수행한다고 가정한다.
- 질의할 때마다 평균적으로 20바이트의 데이터를 입력한다고 가정한다.
  - 문자 인코딩 방법: ASCII 가정(1문자=1바이트)
  - 질의문은 평균적으로 4개 단어로 이루어진다고 가정. 각 단어는 평균적으로 다섯글자로 구성된다고 가정
  - 질의당 평균 4×5=20바이트
- ex. 검색어 자동완성 백엔드에 질의 요청: 검색창에 dinner 입력
  - `search?q=d`
  - `search?q=di`
  - `search?q=din`
  - `search?q=dinn`
  - `search?q=dinne`
  - `search?q=dinner`
- 대략 초당 24,000건의 QPS QPS 발생
  - 10,000,000사용자×10질의/일×20자/24시간/3600초
- 최대 QPS=QPS×2=대략 48,000
- 질의 중 20% 정도는 신규 검색어라고 가정

- 대략 0.4GB 정도=10,000,000사용자×10질의/일×20자×20%
- 매일 0.4GB의 신규 데이터가 시스템에 추가된다.

8.2. 2단계: 개략적 설계안 제시 및 동의 구하기

8.2.1. 데이터 수집 서비스(data gathering service)

- 사용자가 입력한 질의를 실시간으로 수집하는 시스템



사용자가 ‘twitch’, ‘twitter’, ‘twillo’를 순서대로 검색할 때 빈도 테이블

- 빈도 테이블(frequency table): 질의문과 사용빈도를 저장

8.2.2. 질의 서비스(query service)

- 주어진 질의에 다섯 개의 인기 검색어를 정렬해 내놓는 서비스

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

사용자가 ‘tw’를 검색창에 입력하면 표시되는 top 5 자동완성 검색어

- `query`: 질의문을 저장하는 필드
- `frequency`: 질의문이 사용된 빈도를 저장하는 필드
- 빈도 테이블에 기록된 수치를 사용해 top 5 계산

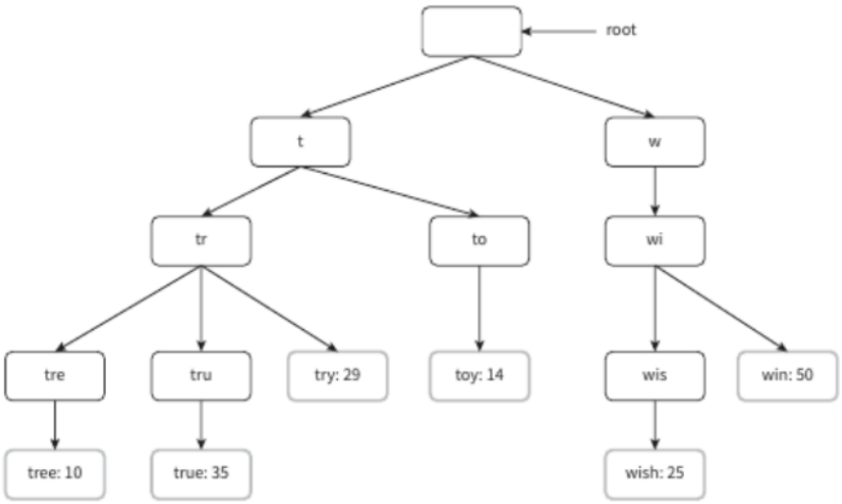
```
SELECT * FROM frquency_table
WHERE query Like `prefix%`
```

```
ORDER BY frequency DESC
LIMIT 5
```

8.3. 3단계: 상세 설계

8.3.1. 트라이(trie) 자료구조

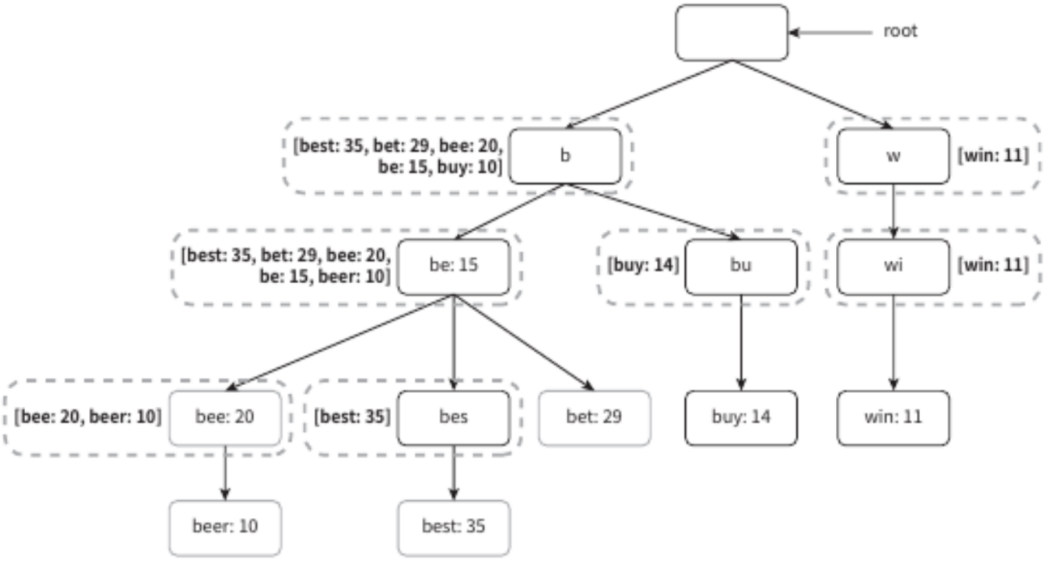
- 관계형 데이터 베이스로 질의문을 골라내는 방법은 효율적이지 않다. → `트라이(trie, 접두어 트리 prefix tree)`로 문제 해결
  - 문자열들을 간략하게 저장할 수 있는 자료구조. 문자열을 꺼내는 연산에 초점을 맞추어 설계되었다.(retrieval)
    - 트리 형태의 자료구조
    - 루트 노드는 빈 문자열을 나타낸다.
    - 각 노드는 글자(character) 하나를 저장하고 26개(해당 글자 다음에 등장할 수 있는 모든 글자 개수)의 자식 노드를 가질 수 있다.
    - 각 트리 노드는 하나의 단어, 또는 접두어 문자열(prefix string)을 나타낸다.
  - p: 접두어(prefix) 길이
  - n: 트라이 안에 있는 노드 개수
  - c: 주어진 노드의 자식 노드 개수



1. 접두어 노드 'be'를 찾는다. `O(p)`

2. 해당 노드부터 시작하는 하위 트리를 탐색하여 모든 유효 노드를 찾는다. `O(c)`
    - 유효노드: `[beer: 10]`, `[best: 35]`, `[bet: 29]`
3. 유효 노드를 정렬하여 2개만 골라낸다. `O(clogc)`
    - 인기 검색어: `[best: 35]`, `[bet: 29]`

- 최악의 경우, k개의 결과를 얻기 위해 전체 트라이를 다 검색해야 한다. → 해결방법
    1. **접두어 최대 길이 제한** → `O(1)`
    2. **노드에 인기 검색어 캐시** → `O(1)`



질의어가 보관된 트라이: k=2, 사용자가 검색창에 'be' 입력
접두어 'be'를 나타내는 노드에 `[best: 35]`, `[bet: 29]`, `[bee: 20]`, `[be: 15]`, `[beer: 10]`의 다섯 개 검색어를 캐시해 두었다.

8.3.2. 데이터 수집 서비스

- 사용자가 타이핑 할 때마다 실시간으로 데이터를 수정하는 경우 문제점
    - 매일 수천만 건의 질의가 입력되는데 그때마다 트라이를 갱신하면 질의 서비스는 심각하게 느려진다.
    - 일단 트라이가 만들어지면 인기 검색어는 자주 바뀌지 않을 것이다. 그러므로 트라이는 자주 갱신할 필요 없다.

- **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

로그 파일 예제

  - 검색창에 입력된 질의에 관한 원본 데이터가 보관된다.
  - 새로 데이터가 추가될 뿐 수정은 이루어지지 않으며 로그 데이터에 인덱스를 걸지 않는다.

- **로그 취합 서버**
  - **실시간 애플리케이션**: 결과를 빨리 보여주는 것이 중요하므로 데이터 취합 주기를 보다 짧게 가져가야 한다.
  - 그 외: 일주일에 한 번 정도로 로그를 취합해도 충분하다.

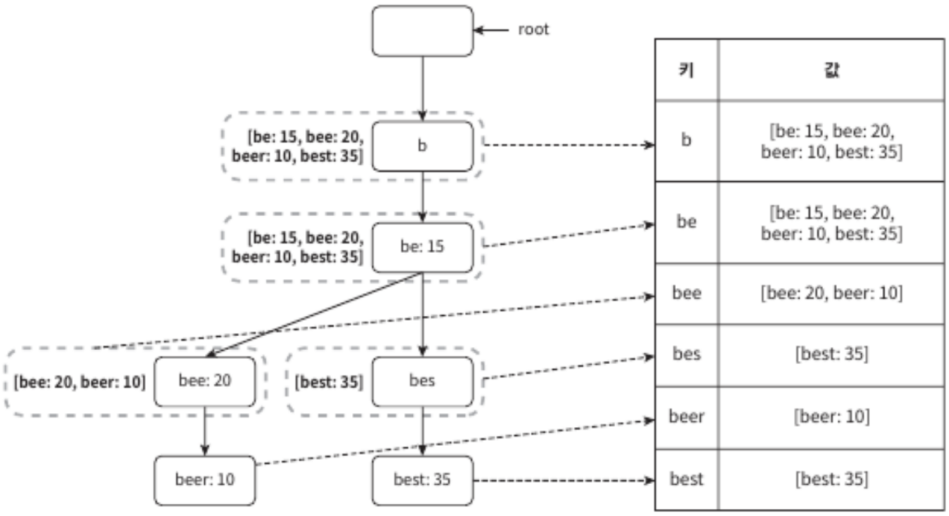
- **취합된 데이터**

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

취합된 데이터 예제

- `time`: 해당 주가 시작된 날짜
- `frequency`: 해당 질의가 해당 주에 사용된 횟수의 합

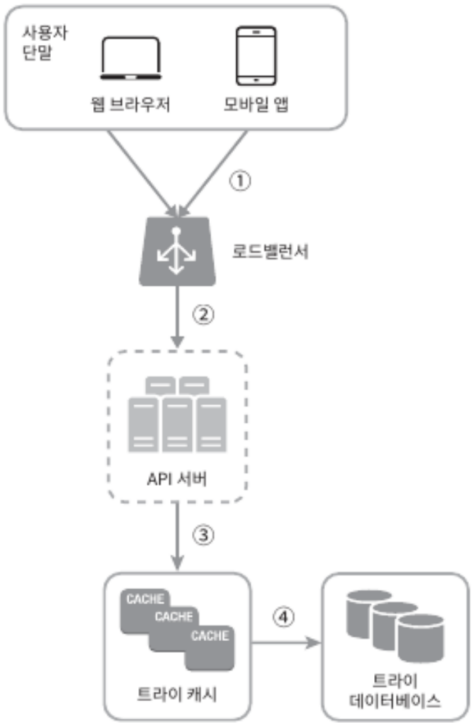
- **작업 서버(worker)**
  - 주기적으로 비동기적 작업(job)을 실행하는 서버 집합
  - 트라이 자료구조를 만들고 트라이 데이터베이스에 저장하는 역할을 담당
- **트라이 데이터베이스**
  - 분산 캐시 시스템으로 트라이 데이터를 메모리에 유지하여 읽기 연산 성능을 높인다.
  - 매주 트라이 데이터베이스의 스냅샷을 떠서 갱신한다.
- **트라이 캐시**
  - 지속성 저장소



트라이를 해시 테이블로 대응시키는 예

1. **문서 저장소(document store)**: 새 트라이를 매주 만들 것이므로 주기적으로 트라이를 직렬화하여 데이터베이스에 저장할 수 있다.
   - ex. 몽고디비(MongoDB)
2. **키-값 저장소**: 아래 로직을 적용하여 트라이를 해시 테이블로 변환할 수 있다.
   - 트라이에 보관된 모든 접두어를 해시 테이블 키로 변환
   - 각 트라이 노드에 보관된 모든 데이터를 해시 테이블 값으로 변환

8.3.3. 질의 서비스



비효율성을 개선한 새 설계안

1. 검색 질의가 로드밸런서로 전송한다.
2. 로드밸런서는 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 경우에는 데이터를 데이터베이스에서 가져와 캐시에 채운다.
    - 다음에 같은 접두어에 대한 질의가 오면 캐시에 보관된 데이터를 사용해 처리할 수 있다.
    - 캐시 미스(cache miss)는 캐시 서버의 메모리가 부족하거나 캐시 서버에 장애가 있어도 발생할 수 있다.

- **질의 서비스 최적화 방안**
    - **AJAX 요청(request)**
        - 요청을 보내고 받기 위해 페이지를 새로고침 할 필요가 없다.
    - **브라우저 캐싱(browser caching)**

        - 제안된 검색어들을 브라우저 캐시에 넣어두면 후속 질의의 결과는 해당 캐시에서 바로 가져갈 수 있다.
    - **데이터 샘플링(data sampling)**
        - 모든 질의 결과를 로깅하도록 해 놓으면 CPU 자원과 저장공간을 엄청나게 소진한다. → N개 요청 가운데 1개만 로깅하도록 한다.

8.3.4. 트라이 연산

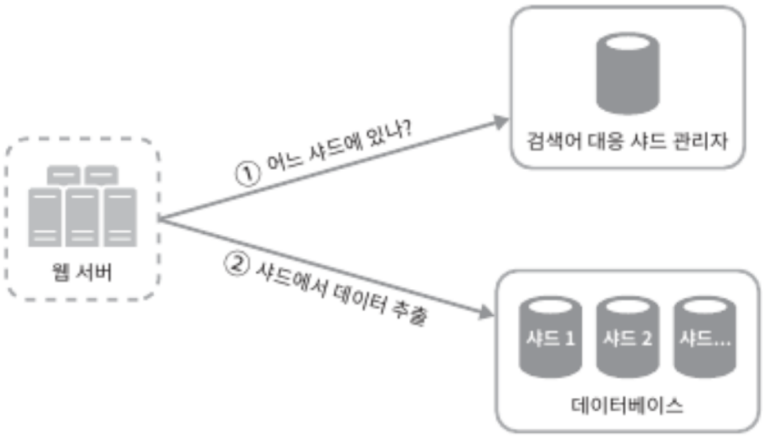
- **트라이 생성**: 작업 서버가 데이터 분석 서비스의 로그나 데이터베이스로부터 취합된 데이터를 이용한다.
- **트라이 갱신**
    - **매주 한번 갱신하는 방법**: 새로운 트라이를 만든 다음 기존 트라이를 대체한다.
    - **트라이의 각 노드를 개별적으로 갱신하는 방법**
        - 트라이가 클 경우 성능이 좋지 않다.
        - 트라이 노드를 갱실할 때 상위 노드에도 인기 검색어 질의 결과가 보관되므로 모든 상위 노드(ancestor)도 갱신해야 한다.
- **검색어 삭제**

부적절한 질의어가 발환되지 않도록 필터링

8.3.5. 저장소 규모 확장

- **첫 글자를 기준으로 샤딩(shrading)** → 가능한 서버는 최대 26대로 제한된다.
    - 검색어를 보관하기 위해 두 대의 서버가 필요하면 'a'부터 'm'까지 글자로 시작하는 검색어는 첫 번째 서버에 저장하고, 나머지는 두 번째 서버에 저장한다.
    - 세 대의 서버가 필요하면 'a'부터 'i'까지는 첫 번째 서버에, 'j'부터 'r'까지는 두 번째 서버에, 나머지는 세 번째 서버에 저장한다.
    - 데이터를 각 서버에 균등하게 배분하기가 불가능하다.

- **과거 질의 데이터의 패턴을 분석하여 사딩(shrading)**



과거 질의 데이터의 패턴을 분석하는 방법

  - `검색어 대응 샤드 관리자(shard map manager)`가 어떤 검색어가 어느 저장소 서버에 저장되는지에 대한 정보를 관리한다.

8.4. 4단계: 마무리

👨‍💻 추가 논의 사항

- 다국어 지원이 가능하도록 시스템을 확장하는 방법
  - 비연어권 국가에서 사용하는 언어를 지원하려면 트라이에 유니코드(unicode) 데이터를 저장해야 한다.
- 국가별로 인기 검색어 순위가 다를 경우
  - 국가별로 다른 트라이를 사용한다.
  - 트라이를 CDN에 저장하여 응답속도를 높일 수 있다.
- 실시간으로 변하는 검색어 추이를 반영하는 방법
  - 현 설계안은 갑자기 인기가 높아지는 검색어를 지원할 수 없다.
    - 작업 서버가 매주 한 번씩만 돌도록 되어 있다.
    - 때맞춰 서버가 실행되도 트라이를 구성하는 데 너무 많은 시간이 든다.
  - 샤딩을 통하여 작업 대상 데이터의 양을 줄인다.
  - 순위 모델(ranking model)을 바꾸어 최근 검색어에 보다 높은 가중치를 주도록 한다.
  - 데이터가 스트림 형태로 올 수 있다. 한번에 모든 데이터를 동시에 사용할 수 없을 가능성이 있다. 데이터가 스트리밍되므로 스트림 프로세싱을 위한 시스템이 필요하다.
    - 아파치 하둡 맵리듀스(Apache Hadoop MapReduce)
    - 아파치 스파크 스트리밍(Apache Spark Streaming)
    - 아파치 스톰(Apache Strom)
    - 아파치 카프카(Apache Kafka)