13장 검색어 자동완성 시스템

1단계 문제 이해 및 설계 범위 확정

- 사용자는 자동완성 될 검색어의 첫 부분을 입력
- 검색어 인기 순위 상위 5개의 자동완성 단어 표시
- 맞춤법 지원 x / 영어 질의 가정
- 일간 사용자 (DAU) 기준 천만 명 지원

요구 사항

- 빠른 응답 속도 : 페북 검색어 자동완성 관련 문서 보면 100밀리초 이내여야 한다.
- 연관성 : 출력 검색어가 입력된 단어와 연관된 것이어야 한다.
- 정렬 : 시스템 계산 결과는 인기도 등의 순위 모델에 의해 정렬되어야 한다.
- 규모 확장성 : 시스템이 많은 트래픽을 감당할 수 있도록 확장 가능해야 한다.
- 고가용성 : 시스템 일부에 장애가 발생하거나, 느려지거나, 네트워크 문제 생겨도 계속 사용 가능해야 한다.

개략적 규모 추정

- 일간 능동 사용자 (DAU) 천만 명으로 가정.
- 평균적으로 각 사용자가 매일 10건의 검색 수행 가정.
- 질의할 때마다 평균적으로 20바이트의 데이터 입력한다고 가정.
  - 문자 인코딩 방법으로는 ASCII 사용 가정. 1문자 = 1바이트
  - 질의 문은 평균적으로 4 단어, 각 단어 평균 다섯 글자로 구성된다고 가정.
  - 각 질의당 평균 4 X 5 = 20 바이트
- 검색창에 글자 입력할 때마다 클라이언트는 검색어 자동완성 백엔드에 요청을 보냄. 즉 1회당 평균 20 건의 요청이 전달됨.
  - `search?q=d`, `search?q=di` 이런 식으로.

- 대략 초당 24,000 건의 질의 (QPS) 가 발생할 것임. (천만 사용자 X 10질의/일 X 20자/24시간/3600초)
- 최대 QPS = QPS X 2 = 약 48,000
- 질의 가운데 20% 정도는 신규 검색어라고 가정. 따라서 대략 0.4GB. (천만 사용자 X 10질의/일 X 20자 X 20%). 즉 매일 0.4 GB의 신규 데이터가 시스템에 추가됨.

2단계 개략적 설계안 제시 및 동의 구하기

데이터 수집 서비스

- 질의문과 사용빈도 실시간으로 저장하는 빈도 테이블이 있다고 가정.
- 사용자가 검색하면 그에 따른 질의가 추가되거나 해당 빈도가 업데이트 됨.

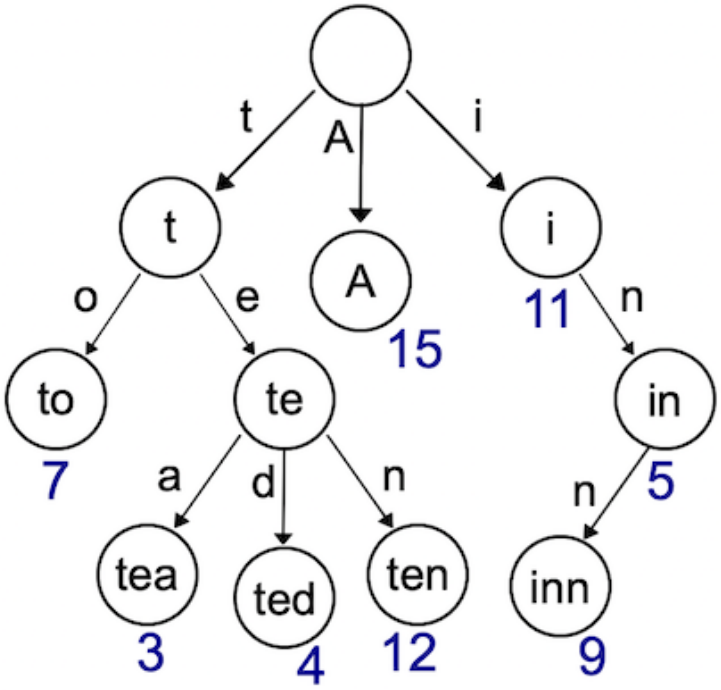
질의 서비스

- 빈도 테이블에서 top5 결과를 반환함.

3단계 상세 설계

트라이 (trie) 자료구조

(p : 접두어의 길이, n : 트리 안의 노드 개수, c : 주어진 노드의 자식 개수)



- 트리 형태의 자료구조.
- 루트 노드는 빈 문자열을 나타냄.
- 각 노드는 글자 하나를 저장, 26개의 자식 노드를 가질 수 있음.
- 각 트리 노드는 하나의 단어, 또는 접두어 문자열을 나타냄.
- 사진에는 없지만 해당 문제에서는 노드에 빈도 정보도 함께 저장.

가장 많이 사용된 질의어 k개 서치 과정

- 해당 접두어를 표현하는 노드를 찾는다. O(n).
- 해당 노드부터 시작하는 하위 트리를 탐색하여 모든 유효 노드를 찾는다. O(n).
- 유효 노드를 정렬항 가장 인기 있는 검색어 k개를 찾는다. O(clogc).

전체 트리 검색을 방지하는 해결 방법

- 접두어의 최대 길이 제한
- 노드에 인기 검색어 캐시트라이를
  - 각 노드에 k개의 인기 검색어를 저장해둠.
  - 5-10 개의 자동완성 제안 표시하면 되므로 k는 작은 값으로 충분.
- 효과
  - 접두어 노드 찾는 시간 복잡도 O(1)로 바뀜.

    - 최고 인기 검색어 5개 찾는 시간 복잡도 O(1)로 바뀜. 캐싱 덕분.
    - 따라서 최고 인기 검색어 k개 찾는 전체 복잡도가 O(1)이 된다.

데이터 수집 서비스

매일 발생하는 수천만건의 질의에 대해 매번 트라이를 갱신하는 것은 서비스 속도를 현저히 낮춘다. 우선 트라이가 만들어지면 인기 검색어는 쉽게 바뀌지 않으므로 그럴 필요가 없다. 다음은 해당 문제를 개선한 구조이다.

데이터 분석 서비스 로그

- 검색창에 입력된 질의에 관한 원본 데이터가 보관됨. 새로운 데이터가 추가될 뿐 수정은 이루어지지 않는다.

로그 취합 서버

- 서비스 특성 따라 취합 방식이 달라짐. 트위터 등 실시간 어플은 데이터 취합 주기가 비교적 짧을 수 있다. 대부분은 일주일에 한 번 정도로 충분.

취합된 데이터

- query 별로 time과 frequency 필드를 함께 저장

작업 서버

- 주기적으로 비동기적 작업을 실행하는 서버 집합. 트라이 구조 만들고 트라이 db에 저장.

트라이 캐시

- 분산 캐시 시스템으로 트라이 데이터를 메모리에 유지하여 읽기 연산 성능을 높임. 매주 트라이 db 스냅샷을 통해 갱신

트라이 데이터베이스

- 문서 저장소 : 주기적으로 트라이를 직렬화하여 db에 저장 가능. ex) MongoDB
- 키-값 저장소 : 트라이에 보관된 데이터를 모두 해시 테이블에 저장 (키 - 질의, 값 - 노드와 빈도)

질의 서비스

질의 서비스는 매우 빨라야 하므로 다음과 같은 최적화 방안을 생각해볼 수 있다.

- AJAX 요청 : 요청을 보내고 받기 위해 페이지 새로고침을 할 필요가 없다.
- 브라우저 캐싱 : 제안된 검색어들을 브라우저 캐시에 넣어둔다.
- 데이터 샘플링 : N개 중 한 개의 요청만 로깅해 cpu와 저장공간 절약.

트라이 연산

트라이 생성 : 데이터 분석 서비스의 로그나 db로부터 취합된 데이터 이용.

트라이 갱신

1) 매주 한 번 갱신.

2) 트라이 각 노드를 개별적으로 갱신. 트라이가 작을 경우 고려해볼 수 있다. 상위 노드도 함께 갱신.

검색어 삭제 : 트라이 캐시 앞에 필터 계층을 두고 부적절한 질의어 반환되지 않도록. 규칙따라 결과를 자유롭게 변경할 수 있다는 장점.

저장소 규모 확장 : 샤딩을 통해 서버 여러대에 균등한 양의 데이터를 분배할 수 있다.