Build Typeahead/Autocomplete System

Requirements clarification

- **Functional requirements**
  - As the user types in their search, our system should suggest top K frequent words matching the prefix the user has typed.
- **Non-functional requirements**
  - This suggestion functionality should happen in real-time with minimal latency.

Estimation

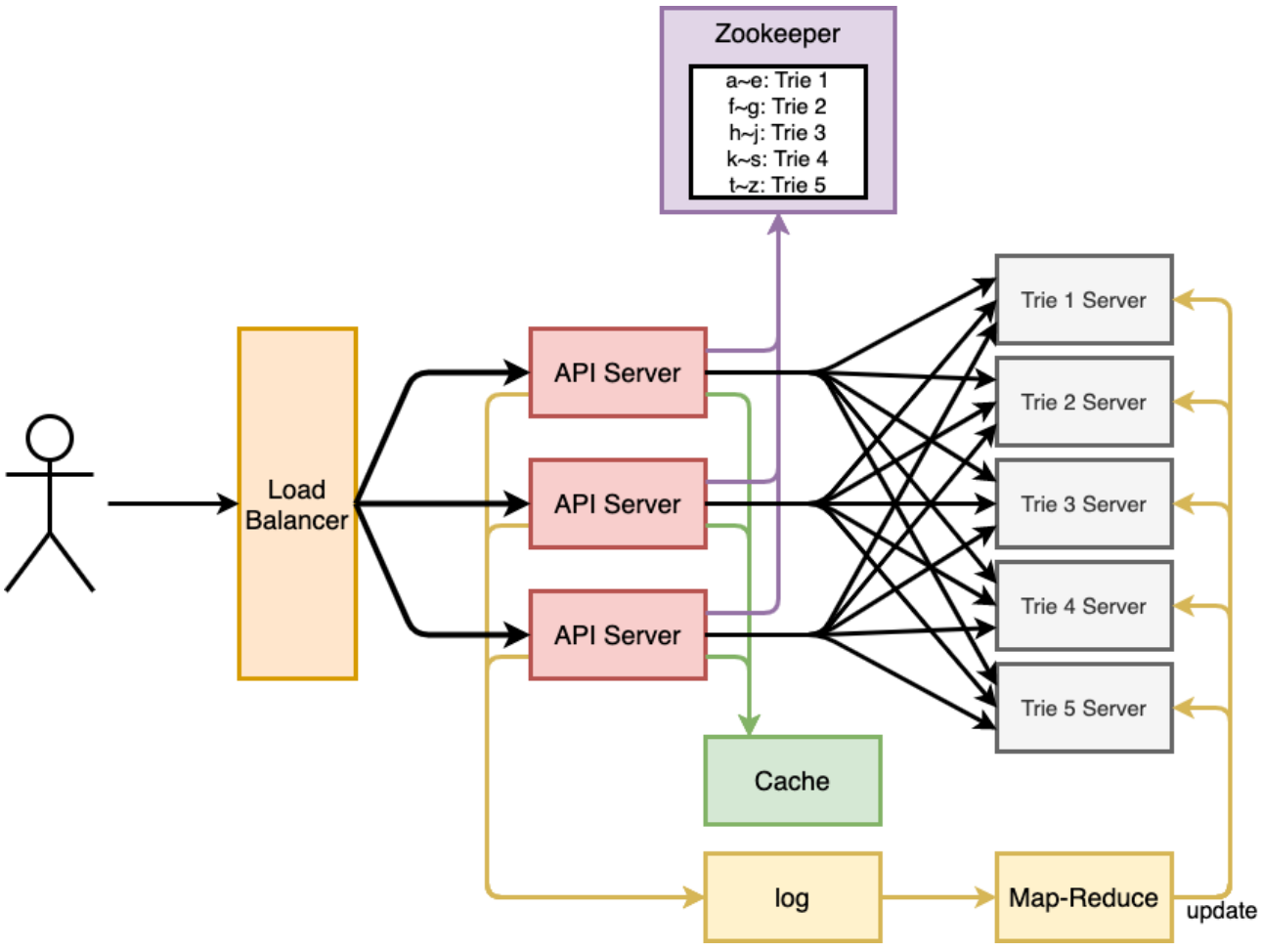
- **Traffic estimation**
  - Number of searches per day = 5 billion (Assumed)
  - Number of searches per second (QPS) = Number of searches per day / 24 hours / 3600 seconds = 60000 times/s
- **Storage estimation**
  - Types
    - Data: Yes
    - File: No

- Capacity
  - Number of terms need to build an index
    - 5 billion searches per day.
    - Only 20% of these will be unique. (Assumed)
    - We only want to index the top 50% of the search terms. (Assumed)

69 lines (62 sloc) | 2.96 KB

    - 50% = 5 million
  - Size for storing the index
    - Each search has 3 words.
    - Each words has 5 characters.
    - Each character needs 2 bytes
    - Total size for storing the index per day = 5 million x 3 x 5 x 2 = 15 GB

High-level design

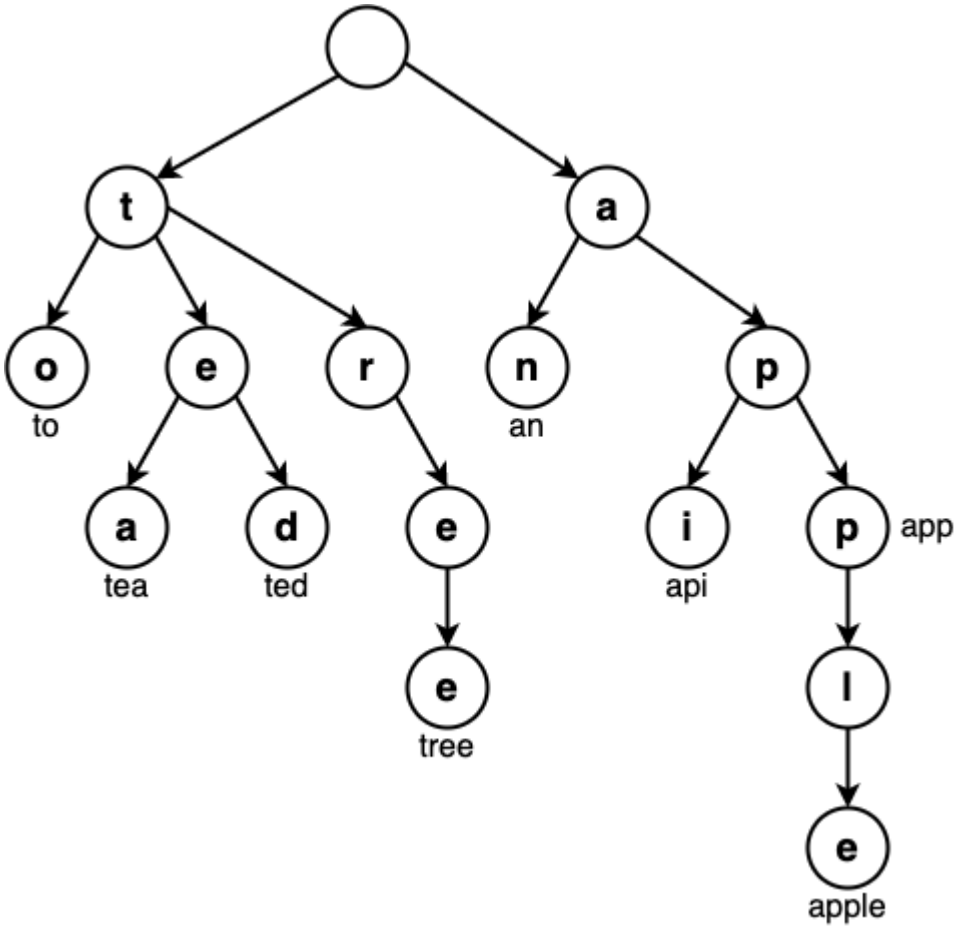


- **API Servers**
  - Handle the requests from clients.
- **Trie Servers**

    - Each trie server stores a sub-trie of the whole trie.
- **Cache**
    - Cache stores top searched words.
    - API server will try to find the result from the cache first. If there is no result from the cache, it will check tries servers later.
- **Zookeeper**
    - Zookeeper will help API servers locate the proper trie server(s) for getting the results.
- **Log**
    - Log is used to store searches and track their frequencies.
- **Map-Reduce**
    - Map-Reduce will process the logging data to update the copy of the old trie to a new trie.

Detailed design

- **Data structure**
    - Choice
        - Trie
    - General structure of Trie
        - Each node store one character.
        - Root node store an empty character.
        - The path from the root node to the leaf node can construct a word.



    - Additional features for our use case
        - Each node stores the frequency of its prefix has been searched.
        - Each node only keep top K frequent of all its children.
- **Update trie**
    - Basic idea
        - Update our trie offline after certain interval (every day or every week)
    - Normal time
        - Put new searches into a log and track their frequencies.
        - Log every search or log every 1000th search (sampling)
    - At the time to update the trie
        - Copy the old tries from servers
        - Use Map-Reduce to process the logging data to update the copy of the old tries to new tries.
        - Replace the old tries on the servers with the new tries.