Elasticsearchで検索システムを作った話

内容

- 話すこと
  - Elasticsearchの概要
    - インデックスの概要
    - 実際にどのようなマッピングにしたか
  - データ同期の仕組み
    - Logstash関連
  - 同義語ファイル生成

- 話さないこと
  - 詳細な検索クエリの解説について
  - 集計系のクエリの話

検索対象

- 対象システム内の特定のテーブルについて検索
  - テーブル1: 約500万件
  - テーブル2: 約200万件
  - テーブル3: 約1万8千件
  - テーブル4: 同上
- その他要件
  - 集計処理などはなく、単純に検索処理のみが必要だった。
  - また、各種データに紐づく添付ファイル (pdf, word, excel等)の中身からも検索することになった。

Elasticsearchとは

- 分散型のオープンソースな検索・分析エンジン
  - 検索とは文字通りなにかを抽出するような処理
  - 分析とは各種集計の処理を指す。
  - 検索、分析はどちらか一つではなく、検索したものをさらに集計したり、集計したものをベースにさらに集計したり、いろいろ組み合わせることが出来る。

- Elasticsearchで出てくる用語
  - インデックス
    - 過去はタイプがあるせいでRDBからの比喩がしづらかったが、タイプがなくなった現在、実質RDBでいうテーブルの単位と考えて構わない
  - タイプ
    - 現在では、タイプは実質廃止されており気にしなくて良い。
  - ドキュメント
    - RDBでいうテーブル(=elasticsearchのインデックス)の中の1行の単位

Elasticsearchのインデックス

インデックスは基本的には事前に定義され、RDBのテーブルのように

　　どのような名前のフィールドがあり、それはどのような型なのか？

というような定義がされている

(設定によってはデータ投入時に動的に設定も出来るが今回は使っていない)

型には数値や、日付などの基本的なもののほか、全文検索という意味ではtext型の理解が重要

全文検索

全文検索とは、wikipediaによると

全文検索（ぜんぶんけんさく、英: Full text search）とは、コンピュータにおいて、複数の文書（ファイル）から特定の文字列を検索すること。「ファイル名検索」や「単一ファイル内の文字列検索」と異なり、「複数文書にまたがって、文書に含まれる全文を対象とした検索」という意味で使用される。

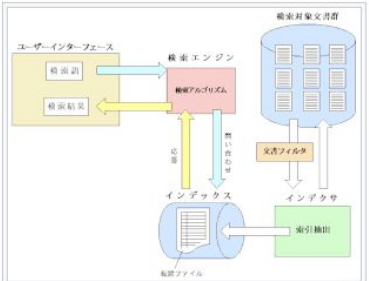
全文検索技術 [編集]

grep型 [編集]

順次走査検索、逐次検索ともいう。「grep」とはUNIXにおける文字列検索コマンドであり、複数のテキストファイルの内容を順次走査していくことで、検索対象となる文字列を探し出す。一般に「grep型」と呼ばれる検索手法は、事前に索引ファイル（インデックス）を作成せず、ファイルを順次走査していくために、検索対象の増加に伴って検索速度が低下するのが特徴である。ちなみに「grep型」とは実際にgrepコマンドを使っているという意味ではないので注意のこと。

索引（インデックス）型 [編集]

検索対象となる文書数が膨大な場合、grep型では検索を行うたびに1つ1つの文書にアクセスし、該当データを逐次検索するので、検索対象文書の増加に比例して、検索にかかる時間も長くなっていってしまう。そこであらかじめ検索対象となる文書群を走査しておき、高速な検索が可能になるような索引データを準備することで、検索時のパフォーマンスを向上させる手法が取られている。事前に索引ファイルを作成することをインデクシング（英: indexing）と呼ぶ。インデクシングにより生成されるデータはインデックス（インデックス）と呼ばれ、その構造は多くの場合、「文字列 | ファイルの場所 | ファイルの更新日 | 出現頻度…」といったようなリスト形式（テーブル構造）を取り、文字列が検索キーとなっている。検索時にはこのインデックスにアクセスすることで、劇的に高速な検索が可能となる。



インデックス作成型全文検索システム

索引文字列の抽出手法 [編集]

インデックスする手法(日本語の考慮) - 形態素解析

- 形態素解析
  - すももももももものうち
    - というような文に対して
      - すもも/も/もも/も/もも/の/うち (“すもも”も”桃”も桃のうち)
    - というような形式(「わかち書き」という)に変換し、この / で挟まれた言葉をインデックスのキーの単位とするような索引
    - ただし形態素解析を正しく行うためには検索したドメインに合わせた独自の辞書が必要になる(上記の「分かち書き」作成の際に何を単語とみなすか？の単位が辞書に依存する)

インデックスする手法(日本語の考慮) - N-Gram

- 形態素解析と違い、日本語の文節、区切りとは関係なくN文字ずつで区切っていく方式
  - あいうえお
    - をN=3でN-Gramのインデックスを作成すると
    - 「あいう」「いうえ」「うえお」
    - の3語にインデックスされる。
    - また用語として、Nが特定の値の場合に
      - N=1 の場合 ユニグラム
      - N=2 の場合 バイグラム
      - N=3 の場合 トリグラム
    - と呼ぶ場合がある

形態素解析とN-Gramの違い

形態素解析の場合、辞書に検索対象ドメインの単語が含まれていないと一見マッチしそうな単語なのにマッチしない場合があり、単純な部分一致が必要な場合N-Gramのほうが良い場合が多そうだった。

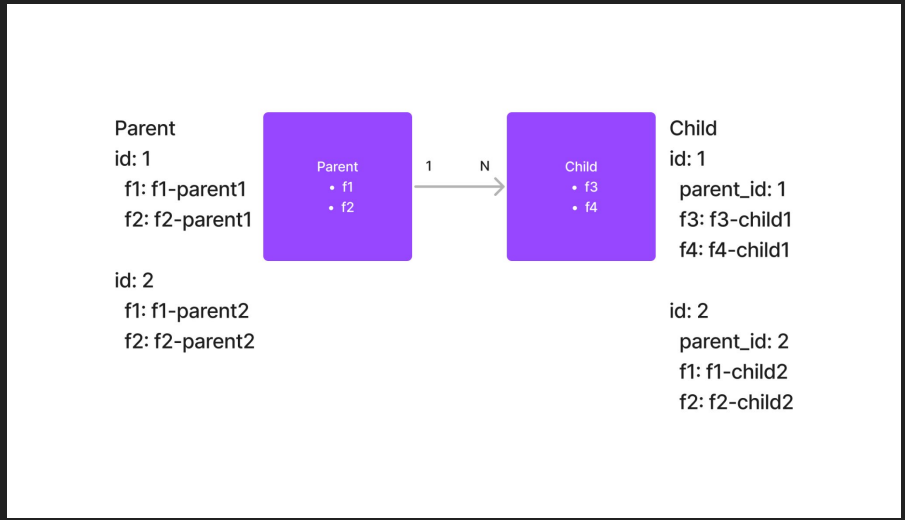
Elasticsearch自体に「部分一致」という検索手法もあるが、それは転置インデックスではなく、全ドキュメント検索での部分一致になるためまた違ってくる。

本システムではN-Gramを選択した。

語順通りの部分一致だけをしたかったのでユニグラム(N=1)にしている

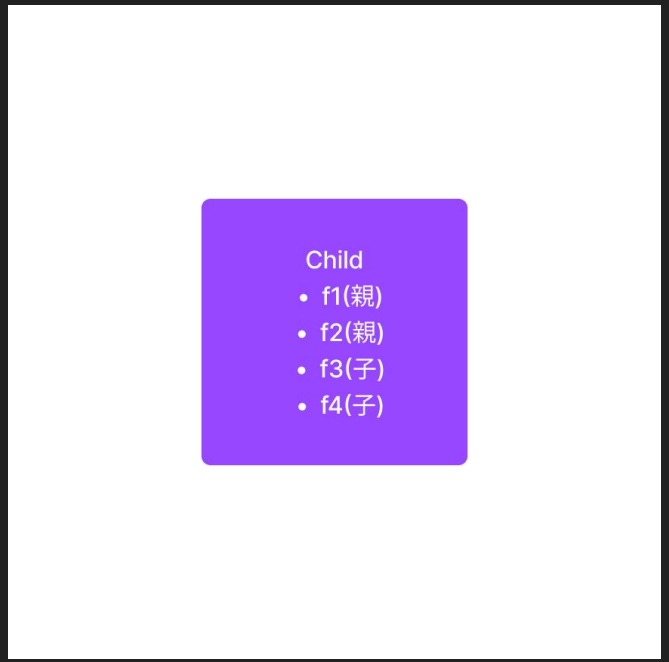
Elasticsearchのインデックス(のマッピング)設計

Elasticsearchでは転置インデックスなどにより同じ件数、内容であればRDBよりかなり高速に検索できるが、RDBが得意とするような検索時に複数テーブルをJoinして検索するようなことは不得意なため、検索対象の単位(=インデックスの1行)に、検索対象の項目(親や子テーブルの内容)を上手く保持する工夫が必要になる。



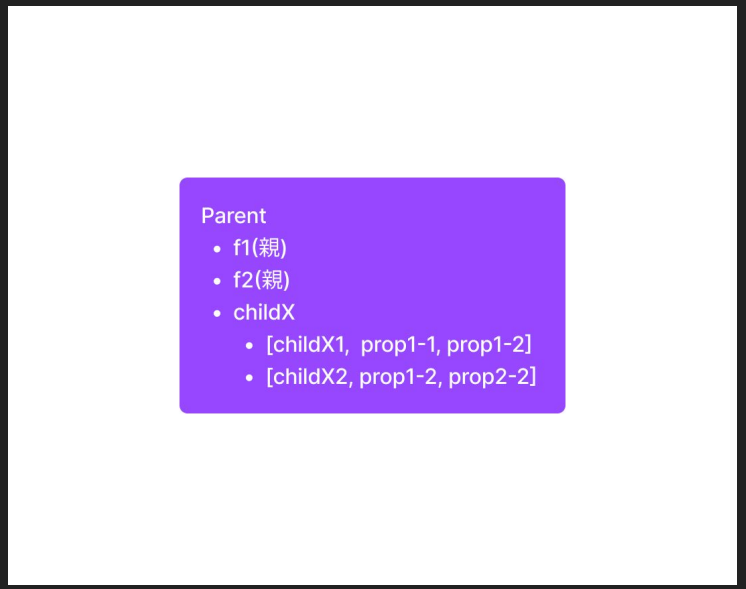
Elasticsearchのインデックス(のマッピング)設計2

先程のような親子テーブルの子側については、子供に親の情報もすべてもたせるようなmappinngにした。



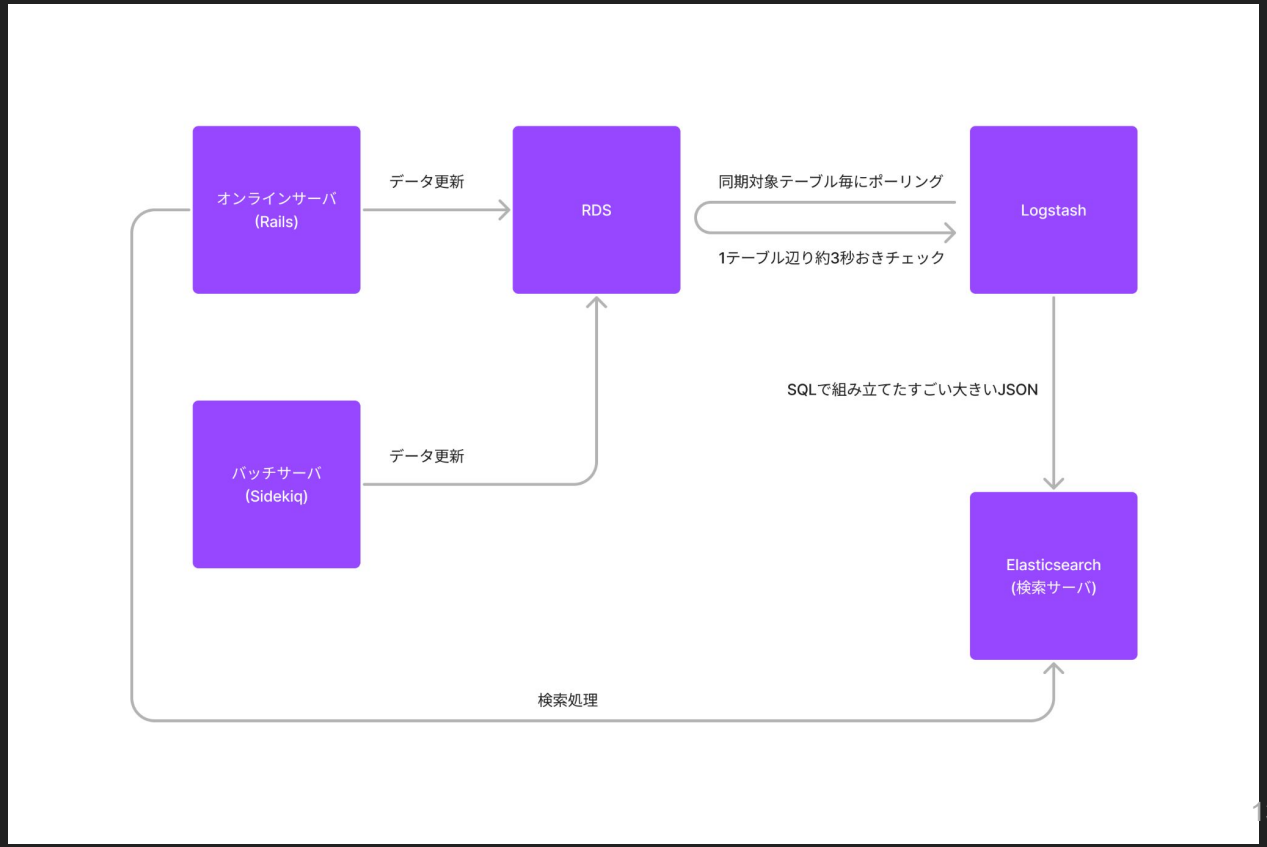
Elasticsearchのインデックス(のマッピング)設計3

また一部のデータについては、子供の情報もnested mappingという形で保持することにし、子供がマッチするような条件で親を抽出するケースも対応した。これについては、子が数件以内に収まるようなもののみを対象とした。



データ同期アーキテクチャ

- RDBがマスタ
  - そこからES同期



Logstashとは

- Elasticsearchを開発しているElastic社が開発しているETLツール

  (ETL ・・・ 抽出(Extract)、変換(Transform)、書き出し、連携(Load))

  様々な形式に対応しており、本プロジェクトでデータ抽出元になったRDBを入力とするだけでなく、csv, webサーバのアクセスログファイル等pluginという形でいろんな入力をソースとすることが出来る。

  同様に、出力にも様々な形式(連携先の一つとしてElasticsearchがあるというだけでそれ以外の様々な出力形式)を指定できる

Logstashの設定例

- 本システムでのLogstashの設定
  - cron形式で三秒に一回同期処理をスケジューリング
  - RDBの1テーブルの1行の単位と同じ単位で同期するESのインデックスが複数ある
    - この一つ一つをpipelineと呼んでいる
    - 今回はRDB側の5テーブルを対応する5インデックスに同期している

Logstash連携の参考資料

elastic社の記事でほぼ同じ構成の資料があったので、下記をかなり参考にしている。

LogstashおよびJDBCを使用してElasticsearchとリレーショナルデータベースの同期を維持する方法

https://www.elastic.co/jp/blog/how-to-keep-elasticsearch-synchronized-with-a-relational-database-using-logstash

同義語ファイルの生成

- 同義語ファイルとは？
  - 字面上はことなるがおなじものを指す単語集

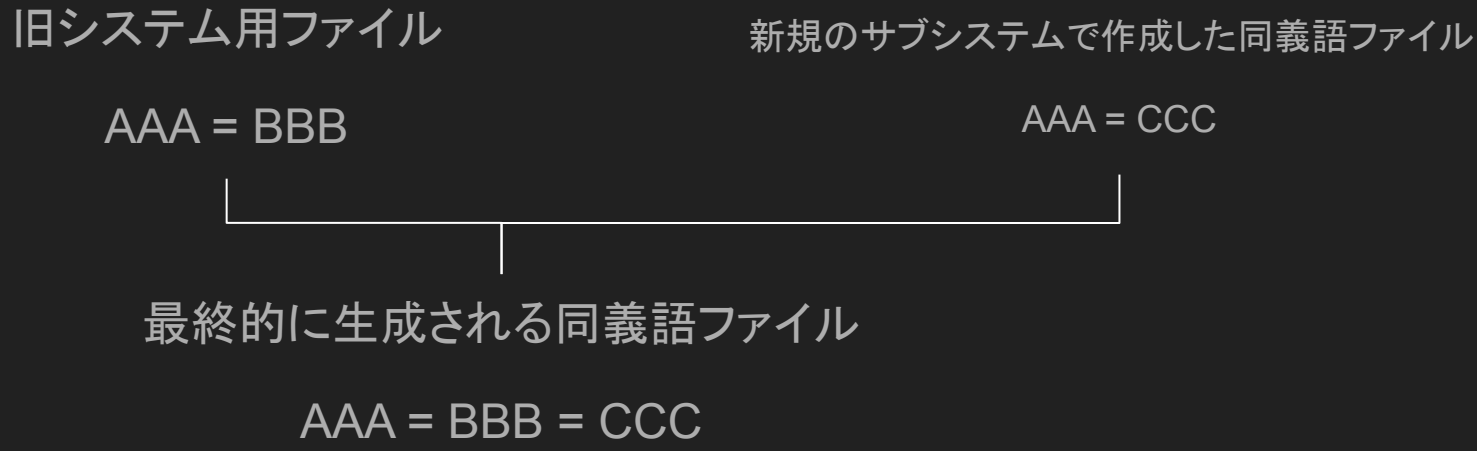
    例) ビートたけし = 北野武 = 立川錦之助(=立川談志一門に実は入っており、その高座名)

    このような紐付けを設定しておくとユーザが「ビートたけし」で検索したときに、これらいずれかが含まれていればマッチする

同義語ファイルの生成要件

旧システムでもともと使っていた（別の検索システム用の)同義語ファイル(xml)が存在しており、それに加えて、今回サブシステムの一つで追加されたシステムの情報をもとに、それらをすべてマージした一つの同義語ファイルを作ることになった。

例)



同義語ファイルの生成方式

旧システムの同義語ファイルは基本変更されないため、S3にベタでおいておくことに。新規に作成したサブシステムからの同義語ファイルは毎日一回、最新データを元に出力し、そのファイルと旧システムの同義語ファイルをマージするバッチをlambda(Golang)で実装した。

マージのアルゴリズムにはUnionFindを使ったが、かなり効率的で全く実行時間は問題なかった(マージ部分は約1〜2秒ぐらい)。参考までに最終的な単語数は1万2千単語ぐらいが含まれている。

参考) https://algo-method.com/descriptions/132

工夫した点1

logstashでupdated_atをトリガーにデータ連携することを書いたが、対象テーブルのupdated_atはデータ連携不要なカラムの更新についても変更されて、無駄に同期が走っていたため、あるテーブルXに対応して、Xの「キー項目, updated_at」のみを保持した(実際はrailsが付与するidカラムはある)データ連携用テーブルを作りそこにキー項目でupdate(or insert)することで抽出対象にした。

このおかげで、オリジナルのテーブルXを何も変更しなくても、このテーブルのレコードの変更で、トラブル時の再連携や、初期データ移行時の一括移行の制御に便利だった。

工夫した点2-1

実際には複雑なのであまり活用されていないが、実はユーザが入力した検索クエリはグーグル検索のように

hoge and piyo and not fuga

のように論理式やカッコによる優先順位を設定できるミニ言語になっており、ユーザのクエリをElasticsearchの検索クエリ用jsonにコンパイルしている

Elasticsearch自身にも「Query string query」というほぼ同様の機能があるが

https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-query-string-query.html

組込みの機能でまかなえるか不安だったので自分で作ることにした。

工夫した点2-1

右記のようなBNFを元に

parslet gem

を使ってクエリをパースしクエリ用のASTに変換し、それをElasticsearchのmustクエリ(=and条件)や、shouldクエリ(=or条件)、matchクエリ(実際にマッチさせるクエリ)などに変換している

```
module Search::Query
  # query           := or_exp
  # or_exp          := and_exp (or_op and_exp)*
  # or_op           := "or"
  #                  | "||"
  # and_exp         := not (and_op? not)*
  # and_op          := "and"
  #                  | "&&"
  # not_exp         | not_op ? term
  # not_op          := "not"
  #                  |  "-"
  #                  |  "!
  # term            := "(" query ")"
  #                  | search_word
  # search_word     := word_with_field
  #                  |  simple_word
  # word_with_field := field_name ":" simple_word
  # field_name      := identifier
  # identifier      := [_0-9a-zA-Z.]+
  # simple_word     = string_literal
  # string_literal  := quotede_string
  #                  | plain_string
  # plain_string    := [^():\s]+

```

parslet gemについては、日本語でかかれた日本一わかりやすい &詳細と自負しているQiitaの記事も参考に
https://qiita.com/pocari/items/048b878575bb28a40732

工夫した点2-1

「ワードA ワードB 」のような検索クエリが右のような検索クエリ(抜粋)にコンパイルされる。(一部モザイクをかけています。)

クエリのjson全体では、インデントや改行なしで15KBぐらいの大きさのjson(それでも検索時間は数ミリ～数十ミリ秒ぐらい)

発生した問題1

- OSSと大企業問題
  - プロジェクト開始4ヶ月後、
    - https://www.elastic.co/jp/blog/licensing-change

    と題しまさかのElastic社とAWS間でライセンス問題発生

    AWSはElasticsearch(7.10.0)互換のOpenSearchというサービスを開始

- このときはあまり判断つかず本システムではOpenSearchに乗り換えはせず当時awsで使えたElasticsearchのバージョンを採用(7.9.3)
- 最新機能などは必要なかったので現状問題は発生していない。

発生した問題2

- 同期が遅い問題
  - 同期の具体的な方式
  - Logstash側で同期対象となるテーブルの最終同期日時を保存しておき、次回データ抽出時に、
    - updated_at > 前回同期した日時
  - というような条件をいれて更新対象を絞りこむ。ここで最初はかなり時間がかかっていた。

  - 今回のデータ数(一番大きいテーブルで約500万件)ぐらいであれば、updated_atにインデックスを貼ることで特に問題なく抽出できるようになった。

終わり