21世紀の最新エンジン達
(初出:オープンソースマガジン 2006年4月号)

(有)未来検索ブラジル

2006/06/20

近年、それもここ 1 年ほどの間に、Rast、Hyper Estraier、Senna といった新しいオープンソースの国産全文検索エンジンが相次いで公開されています。

これまで、日本語が問題なく扱えるオープンソースの全文検索エンジンと言えば、需要が急激に高まった Web の普及期に開発されたものが大半でした。SSE、SGSE、Namazu、Freya、Sufary などの優れた全文検索エンジンは、いずれも 1996〜1999 年くらいの Web の普及期に公開されています。そのあと、数年を経て再び大きな盛り上がりを迎えているようなのです。なぜいまになって新たなエンジンが精力的に開発されているのでしょうか？

1　はじめに

新世代のエンジンは、従来の多くのエンジンと同様、転置インデックスという古くから知られている方式をベースに作られていますが、いずれも従来とは異なる方針に基づいて設計されており、それを反映して性能面でも異なる特徴を持っています。面白いことに本特集で取り上げる 3 つの新世代エンジンは、設計方針や基本性能に関して、共通する傾向が見受けられるのです。

Part 1 では、全文検索エンジンの基本性能に関する技術的なポイントを整理して、新しいエンジンと従来のエンジンとの特徴の違いについて解説します。

2　検索エンジンの性能要素と設計

ユーザーの視点から見た検索エンジンの評価ポイントは、対象とするファイルの種類から、ほかのアプリケーションとの連携機能、Web との親和性など多岐におよびますが、本稿ではエンジンの基本性能に的を絞って述べます。

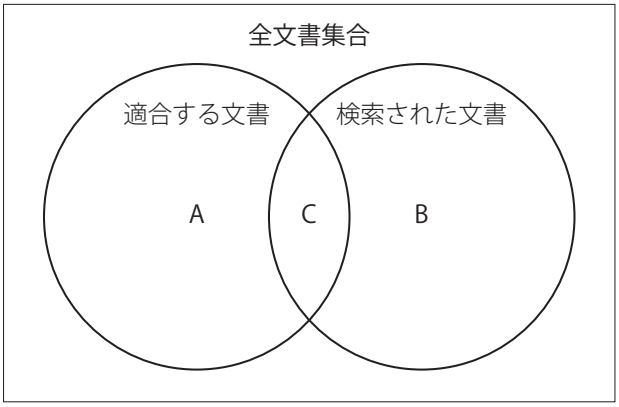
まず、全文検索エンジンの基本性能を評価する主な尺度として、検索速度、更新速度、文書容量、適合率、再現率の 5 つが挙げられます。

- 検索速度
  検索速度は、クエリを投入してから結果が表示されるまでにかかる応答時間で示されることが多いでしょう。多くの人が同時に使用するシステムでは、単位時間にいくつのクエリをこなせるかを示す「検索スループット」も、検索速度を表す重要な尺度になります。

- 更新速度
  更新速度は、インデックスを新規作成する際や、既存のインデックスに文書を追加/変更/削除する際の処理速度です。単位時間に処理できる文書のサイズを示す「更新スループット」がその尺度として用いられます。

- **文書容量**
  文書容量は、1 台のコンピュータで実効的に処理可能な文書の総容量です。

- **適合率と再現率**
  適合率と再現率は、ともに検索エンジンの精度を表す尺度です。すべての文書集合の中で、あるクエリに対して実際に適合する文書集合を A、検索エンジンが結果として返す文書集合を B、A と B の交わりにあたる文書集合を C としたとき、図 1 のように定義されます。
  適合率は、検索結果に含まれるノイズの少なさを示す尺度、再現率は検索漏れの少なさを示す尺度と考えることができます。



$$\text{適合率} = \frac{C(\text{適合する文書} \bigcap \text{検索エンジンが返す文書})}{B(\text{検索エンジンが返す文書})}$$

$$\text{再現率} = \frac{C(\text{適合する文書} \bigcap \text{検索エンジンが返す文書})}{A(\text{適合する文書})}$$

図 1: 適合率と再現率

2.1　性能要素のトレードオフ

以上の 5 つの要素については、相互にトレードオフの関係があり、すべてを兼ね備えたエンジンを作り上げることは非常に困難です。したがって、エンジンの設計に際してはどの要素を優先するかを決め、それに相応しい方式を選択する必要があります。

たとえば、検索速度と更新速度は、ほかの条件を固定した場合、トレードオフの関係となり、どちらを優先するかによって相応しい方式を選択することになります。更新速度を最優先にするのであれば逐次検索方式が、検索速度を最優先にするのであれば索引検索方

式が適しています（図 2）。

逐次検索方式とは、grep と同様に対象文書を逐次スキャンして検索する方式です。事前にインデックスを準備する必要がないので、（当然ながら）更新性能が最も優れていますが、検索するたびに対象文書をすべて参照するので、検索速度は最も遅い水準になります。

索引検索方式は、転置インデックスと呼ばれる構造を更新時に準備する方式です。転置インデックスとは、本の索引と同様の構造、すなわち、各語を含む文書や文書内での出現位置情報を、検索語をキーとして取り出しやすい形式で格納したもので、これをたどることによって非常に高速に検索処理が実行できます。

また、これらの中間に位置する方式として、シグネチャ方式という選択肢も存在します。シグネチャとは、検索語に対応する情報を、ハッシュ関数などを用いて圧縮し、検索語が文書に含まれるかどうかを小さいサイズのデータで絞り込めるようにしたものです。

シグネチャ方式は、転置インデックスと比較してサイズが小さく、短い時間で作成できます。しかし、シグネチャ情報がマッチしても検索語を含まない場合（フォルスドロップ）が存在するため、検索時にはシグネチャで絞り込んだすべての文書に対して逐次検索を実行し、実際に検索語を含む文書を特定する処理（フォルスドロップレゾリューション）が必要になります。

シグネチャのサイズを大きく取るほどフォルスドロップの発生頻度を抑えられるので、検索速度は速くなりますが、その分更新速度は遅くなってしまいます。フォルスドロップレゾリューションを省略してしまえば検索速度と更新速度を両立させられますが、検索結果にノイズが混入するため、今度は適合率が低下してしまいます。

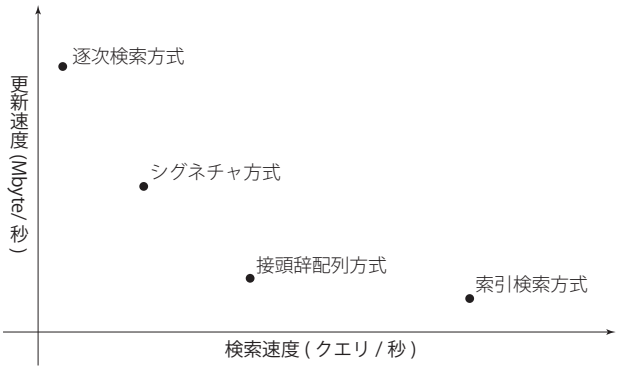


図 2: 検索速度と更新速度の関係

2.2　適合率と再現率

もう一点、検索エンジンの特性にかかわる非常に重要なトレードオフが、適合率と再現率にかかわるものです。図 1 に示す検索結果の文書集合（A）と適合する文書集合（B）とがぴったりと重なるのが理想的なのですが、適合性の判断が自明ではない場合が多いため、これを実現するのは非常に困難になります。

いくつかの例について考えてみましょう。

クエリ	→	検索対象文字列
打合せ資料	→	打ち合せ資料
渋谷の郵便局	→	渋谷郵便局
アンダルシアの風	→	アンダルシア風の料理

これらの例で、左のクエリに対する検索結果として右の文字列は相応しいといえるでしょうか?

一番上は、表記の揺らぎであって、左右は同じものを指していますので相応しいといえるでしょう。真ん中の例も（状況によりますが）ほぼ相応しいといえるでしょう。最後の例は、まったく違うものを指しています。

一番上の例のように、言葉には揺らぎが付きものです。こういった細かい言い回しの差や表記の揺らぎによる検索漏れは、付属語を検索対象から除外したり、活用形を正規化して扱ったりすることで防ぐことができます。しかし、「アンダルシアの風」に対して「アンダルシア風の料理」などのように意味が異なる（適合しない）文書をヒットさせてしまう場合も発生します。つまり、この操作は再現率を高める一方で適合率を下げてしまうのです。

では、文字列に操作をまったく加えず、クエリをそのままの形で含む文書を取り出した場合はどうでしょうか。

クエリ	→	検索対象文字列
君が代	→	それなら君が代表監督だ
相撲	→	アフガン航空相撲殺される
先生	→	この先生きのこるには

上記の例は、いずれもクエリに使われた文字を正確に含む文字列を検索していますが、利用者が期待したような意味でその文字列が使用されているわけではない（適合しない）と考えて良いでしょう。日本語は単語をわかち書き[1]して表記する習慣がないため、字面上の

[1]わかち書きとは、単語と単語の境界を（空白を入れるなどの方法で）明示的に表記することをいう。日本語は通常わかち書きでは表記されないが、形態素解析と呼ばれる処理によって単語境界を推定できる。日

一致だけでは適合性が判断できない場合があります。

この問題に対処するための方法として、検索エンジン内部で文字列を以下のように単語ごとにわかち書きする方法が考えられます。

クエリ	→	検索対象文字列
君が代	→	それ/なら/君/が/代表/監督/だ
相撲	→	アフガン/航空/相/撲殺/さ/れる
先生	→	この/先/生き/のこる/に/は

このように、単語単位で一致する文書のみをヒットさせれば、ノイズを抑えることができます。しかし、わかち書きが常に成功するとは限りません。たとえば、

ここではきものを脱ぐ

という文章は、複数の解釈ができます。

ここ/で/は/きもの/を/脱ぐ
ここ/で/はきもの/を/脱ぐ

後者のように解析された場合、「きもの」ではヒットしなくなってしまいます。このように、単語単位で処理することによって適合率が向上する一方で、再現率が低下してしまうというジレンマがあるのです。

適合率と再現率のどちらを重視するかによって、検索エンジンの方式の選択が左右されます。適合率を重視するなら単語索引方式が適していますし、再現率を重視するなら N-gram インデックス[2]や接尾辞配列[3]のような、文字単位で正確に検索する方式が適しています。どちらの尺度が優先されるかは、用途や文書の規模によって変わってくるので、一概に言えません。たとえば、特許の文書を検索する場合には再現率が重視されますし、大規模な Web 文書を検索する場合には適合率が重視されます。

2.3　フレーズ検索の実現方式に関するトレードオフ

転置インデックス方式のエンジンの場合は、フレーズ検索をどのように実現するかも、設計上非常に重要な選択肢です。フレーズ検索とは、クエリに指定された複数の語が、そのままの順番で正確に現れる文書を取り出す検索です。例えば単語索引方式のエンジンでは、

本語形態素解析エンジンとして茶筅（ちゃせん）や和布蕪（めかぶ）が有名。
茶筅（Chasen）：http://chasen.naist.jp/hiki/ChaSen/
和布蕪（MeCab）：http://mecab.sourceforge.jp/

[2]N-gram インデックスは、文書を n 文字の文字要素毎に分解して処理することによって、文字単位で一致する文書を高速に検索できる手法のこと。詳しくは、Part 2(Rast), Part 3(Hyper Estraier) を参照。

[3]接尾辞配列（suffix array）は、任意の部分文字列を高速に検索できるデータ構造。接尾辞配列を用いた検索ツールとして SUFARY、sary などがある。
SUFARY：http://nais.to/%7eyto/tools/sufary/
saryhttp://sary.sourceforge.net/

クエリ	→	検索対象文字列
日本大学	→	日本大学の
日本大学	→	日本の大学

上記の例で、右の文字列はいずれも「日本」と「大学」という語を含んでいますが、クエリと同じ順番で正確に現れる上の方だけを取り出すためにはフレーズ検索が必要です。

クエリに指定された文字列が以下の条件に当てはまるなら、それを正確に含む文書を検索するためにフレーズ検索が必要になります。

(1) 単語ベースのエンジンを採用した場合、クエリに指定された文字列が複数の単語にわかち書きされる場合、複数フレーズ検索のための仕組みが必要になる

(2) N-gram ベースのエンジンを採用した場合、クエリに指定された文字列が n 文字、つまり文字成分表の作成単位よりも長い場合、複数フレーズ検索のための仕組みが必要になる
日本語においては、複合名詞がクエリとして頻繁に使用されますが、複合名詞は多くの場合、上記の条件に当てはまってしまうので、フレーズ検索の重要度は非常に高いといえます。
フレーズ検索を正確に実行するためには、

(2-1) 完全転置インデックス方式（full inverted index）：文書内で単語が出現する位置情報を転置インデックスに記録しておき、検索時にはこれを使って、各単語が隣接して現れたことを調べる

(2-2) 転置文書インデックス方式（inverted file index）：検索結果の候補となる文書に対して、逐次検索を実行してフレーズが一致する文書を絞り込む

などの手段があります。しかし完全転置インデックス方式は転置インデックスのサイズが大きくなり、更新速度と文書容量が犠牲になる、一方の転置文書インデックス方式は検索速度が低下する、といった問題があります。

3　新世代エンジンの技術ポイント

以上のように、検索エンジンの設計にはさまざまな選択肢があり、それぞれにトレードオフが存在します。各エンジンはその設計方針にしたがって方式を選択し、それがエンジンの個性や特徴となって現れるのです。

ここでは、以上で説明した性能要素に注目して、従来のエンジンと新しいエンジンのそれぞれの特徴を概観してみましょう。

3.1　Namazu

Namazu は単語ベースの転置インデックス方式を採用しており、大枠では検索速度を重視するタイプのエンジンであるといえます。基本構成における Namazu の最大の特徴は、ハッシュ値を用いたフレーズ検索にあります。

Namazu は、転置インデックスに位置情報を持たせる代わりに、2 単語のフレーズの情報をハッシュ関数を使って圧縮し、そのハッシュ値を文書と対応付けて管理することでフレーズ検索を実現しています。これによって、完全転置インデックス方式よりもずっと小さなサイズのインデックスで、転置文書インデックス方式よりもずっと高速なフレーズ検索を実行できます。

ただし、ハッシュ値が衝突した場合や 3 単語以上のフレーズを検索した場合には、シグネチャ方式と同様に、フォルスドロップが発生します。Namazu ではフォルスドロップレゾリューションを行いませんので、そのままノイズとして検索結果に出力され、適合率は低下します。Namazu は単語ベースの転置インデックスで適合率を一定のレベルに保ちながら、検索速度・更新速度・文書容量の性能要素を重視したエンジンだといえるでしょう。

3.2　Rast

Rast は転置インデックスの内部に位置情報を記録する、先ほどの分類でいえば完全転置インデックス方式を採用したエンジンです。Rast の特徴は、トークナイザを切り替えることで、転置インデックスの単位を単語ベースと N-gram ベースとで切り替えられることです。

単語ベースでインデックスを構成した場合は、正確なフレーズ検索が高速に実行できる、適合率に優れたエンジンとして使用できます。一方、N-gram ベースでインデックスを構成した場合は、再現率に優れたエンジンとして使用できます。用途によって精度のどちらの側面を重視するかを決定できるのは大変便利です。Rast は更新速度・文書容量の面でやや不利ですが、適合率や再現率の精度面を重視したエンジンといえます。

3.3　Hyper Estraier

Hyper Estraier は N-gram ベースの転置インデックスをメインに使用しています。Hyper Estraier の基本構成上の最大の特徴は、転置インデックス上で N 文字の文字成分の後に続く M 文字の情報のハッシュ値を、文書と対応付けて管理している点です（N.M-gram 法）。この手法は Namazu におけるフレーズ検索の実現手法と同様の効果を生み出します。つまり、位置情報を持つ方式に比べてずっと小さなサイズのインデックスで、高速に N 文字よ

り長い文字列を検索できるのです。

しかし、ハッシュ値が衝突した場合や N + M 文字より長い文字列を検索した場合には、やはり Namazu と同様にフォルスドロップが発生し適合率が低下してしまいます。そこで、この問題や、単語境界に一致しない結果を返す N-gram 方式特有の問題に対処するために、単語ベースのサブインデックスを構成し、N-gram インデックスと併用することによって適合率を向上させています。Hyper Estraier は、検索速度・更新速度・文書容量を追究しつつ、適合率や再現率の精度面にもこだわったエンジンといえます。

3.4　Senna

Senna は完全転置インデックス方式を用いたエンジンで、Rast と同様、転置インデックスの単位として単語ベースと N-gram ベースのどちらかを選択可能になっています。Senna の最大の特徴は、単語ベースでインデックスを構成したときに、単語境界よりも短い単位でのマッチングを可能とし、単語インデックス方式特有の検索漏れによる再現率の低下に対処している点です。

たとえば「成田空港/リムジンバス」のようにわかち書きされてしまった場合、通常の単語インデックス方式のエンジンは「空港リムジン」というクエリではヒットしません（検索漏れになってしまう）が、Senna はこれをヒットさせるかどうかを検索時に制御することが可能です。Senna は適合率を重視しつつ、再現率にもこだわったエンジンといえます。

4　オープンソース Google が生まれる日

以上のように、各エンジンの基本構成上の特徴を概観して分かることは、新世代のエンジンは共通して精度面（適合率と再現率）を重視している、ということです。いったい何が各エンジンの設計方針に影響を与えているのでしょうか？

端的に言えば Google の影響だと筆者は考えています。Namazu を含めたオープンソースソフトウェアの検索エンジンが盛んに開発された時期は、商用の Web サーチエンジンでも各社が群雄割拠していた時代に重なります。そのあと、Google が台頭し、多くのエンジンが淘汰されてしまいました。

Google が世界を席巻するに至った理由の 1 つは、圧倒的な精度の高さにありました。その精度の高さを支えていたのは、PageRank などの新たな手法もさることながら、徹底して精度面（とくに適合率）にこだわったエンジンの基本性能にあると筆者は考えています。

2000 年当時で、デフォルトでフレーズ検索を実行し、KWIC ベースのスニペット[4]を生成する商用 Web サーチエンジンは、Google 以外にはありませんでした。

Google がこれほど普及した後で登場した新世代のエンジンたちは、Google の性能を十分に踏まえて設計されています。さまざまな性能要素の中で、精度面にとくにこだわったエンジンがそろってこの時期に現れているのは、偶然ではないと考えられます。

検索エンジンのアプリケーション開発者にとってみれば、さまざまな性能特性を持つエンジンが選択可能になることは朗報といって良いでしょう。「なるべく低負荷でデスクトップの文書を網羅的に検索したい」、「Google なみの高精度な検索サービスを作りたい」などの要件に合わせて、最適なエンジンを選ぶことができます。

それでは、オープンソースのパーツを使って Google そのものを組み上げることは可能でしょうか？　Google のシステムは、Linux などのオープンソースソフトウェアと独自開発の検索エンジンや分散処理機構などを組み合わせて構築したインフラの上に、さまざまなスコアリング手法とそのバランス、悪質 SEO 対策などのさまざまなノウハウを含むアプリケーションが載って構成されていると考えられます。

このアプリケーションレイヤーの部分は、実際にサービスを運営しないと得られない膨大なノウハウの集積であり、一朝一夕には開発できないでしょう。しかし、インフラを支えるソフトウェア部品については、近い将来オープンソースのパーツによってほとんどを満足できるようになるだろうと筆者は考えています。

OS や DBMS のように、多くの人にとって価値を持つ共通財として機能するソフトウェア部品は、Linux や MySQL のような優れたオープンソース製品が育ちやすいのです。全文検索エンジンもこうしたポジションに該当する部品です。Google を見て育ったオープンソースソフトウェアによって、Google に匹敵するサービスが構築できるようになる日がやがてやってくるのではないでしょうか。

[4] スニペットは、検索結果のリストに表示される検索対象文書の抜粋部分を指す言葉。検索対象文書の中から、クエリに指定された文字列を実際に含んでいる個所の近傍を抜き出す方式を KWIC（KeyWord In Context）と呼ぶ。