Ludia と Solr におけるファセットカウント取得と絞り込み検索

2008 年 5 月 14 日

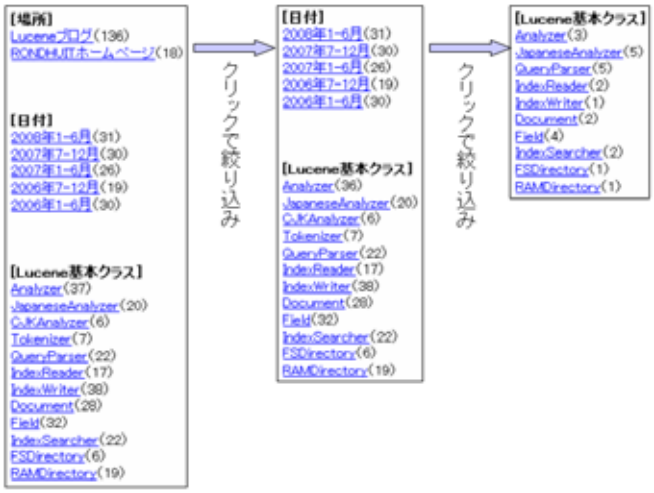
株式会社 ロンウイット　　関口宏司

はじめに

Web アプリケーションにてユーザを目的のページ（コンテンツ）に効率よく誘導する方法として、最近は「フリーワード検索」と「ナビゲーションリンクの提示」を組み合わせる手法が注目されている。まず「フリーワード検索」では、ユーザは探したいコンテンツから連想する 1 つ以上の単語を「検索窓」に入力して[検索]ボタンをクリックすることで、候補となるコンテンツの一覧表示（検索結果一覧画面）のページを得る。同じページに図 1 のようなリンクが表示されることがあるが、このリンクが「ナビゲーションリンク」である。このリンクをクリックすることでリンクに表示された条件で最初の検索結果を絞り込み検索することができる。

＜図 1＞

絞り込み検索の結果、新しい検索結果一覧と「ナビゲーションリンク」が表示される。このリンクをクリックすると、さらに絞り込み検索が行われる（図 2）。リンクをクリックするという簡単なオペレーションを繰り返すだけでユーザを目的のページに効率よく誘導できるのが特徴である。



＜図 2＞

図 1 や図 2 にあるとおり、ナビゲーションリンクの隣には括弧でくくられた数値が表示されている。この数値は当該リンクをクリックすると、検索結果を何件まで絞り込めるかを示す。ユーザはリンクのラベル文字列とこの数値情報を参考にして、どのナビゲーションリンクをクリックするかを判断する。本書ではこの数値情報を「ファセットカウント」と呼ぶ。

本書は上記のようなナビゲーション導線を実装する Web アプリケーションを想定し、高負荷状態において検索エンジンがどの程度のパフォーマンスを発揮するかを計測し、その結果を記したものである。なお、検索エンジンには Ludia と Apache Solr（以下 Solr）を取り上げることとする。

検証環境

検証環境を下表に示す：

項目		説明
ハードウェア	機種名	DELL Inspiron 1501
	CPU	AMD Athlon(TM) 64 X2 デュアルコア・プロセッサ TK-53
	メモリ	1GB(512MBx2) デュアルチャネル DDR2-SDRAM メモリ
	ハードディスク	120GB SATA HDD（5400 回転）
	ネットワーク	100MB Ethernet
OS		Windows XP Professional SP2
Web サーバ		Apache2.2 + FastCGI
Ludia (*1)	Ludia	1.4.0
	MeCab	0.96
	Senna	1.0.9
	DB	PostgreSQL 8.2.6
Solr (*1)	Java	build 1.6.0_05-b13
	Tomcat	5.5.26
	Solr	r635924
負荷試験ツール		Apache JMeter 2.2
検索対象データ		Yahoo!オークション商品データ 約 200 万件

(*1) Ludia と Solr は、本検証を開始した当時の最新バージョンを利用。

検索エンジンを実行するサーバマシンには DELL 製のノート PC を使用した（本体価格：57,124 円）。ハードディスクは低速なため今回の検証では、Ludia、Solr ともに OS のページキャッシュが十分に効いた状態で性能を測定することとした。

全検証を通じて Ludia（Senna および PostgreSQL）の主なパラメータは次のとおり設定した：

```
demo=# select * from pgs2getoption();
-[ RECORD 1 ]------+--------
max_n_sort_result   | 10
enable_seqscan      | off
seqscan_flags       | 1
sen_index_flags     | 31
max_n_index_cache   | 16
initial_n_segments | 512

demo=# show work_mem;
-[ RECORD 1 ]--
work_mem | 10MB

demo=# show shared_buffers;
-[ RECORD 1 ]--+-----
shared_buffers | 32MB
```

また Ludia で全文検索の対象となる PostgreSQL のテーブル定義は次のとおりである（紙幅の関係上、一部略称を用いた）：

```
demo=# ¥d auction_item
       Table "public.auction_item"
    Column      |  Type              | Mod
---------------+-----------------+-----------
 auction_id     | charvar(16)       | not null
 category_id    | charvar(16)       |
 category_path | charvar(256)      |
 title          | charvar(128)      | not null
 url            | charvar(128)      |
 seller_id      | charvar(24)       |
 image1_url     | charvar(256)      |
 image2_url     | charvar(256)      |
 image3_url     | charvar(256)      |
 init_price     | integer           | not null
 price          | integer           |
 bids           | integer           |
 start_time     | timestamp w/o TZ |
 end_time       | timestamp w/o TZ |
```

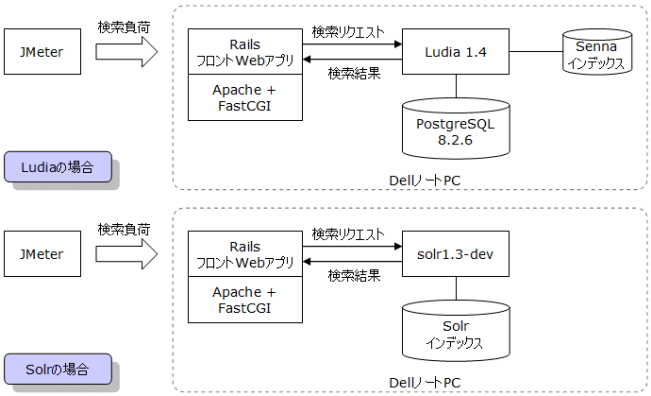
```
 description    | charvar(8192)      |
 price_group    | integer            |
 date_group     | integer            |
Indexes:
    "auction_item_pkey" PRIMARY KEY, btree (auction_id)
    "fidx" fulltext ((description::text))
    "idx_date" btree (end_time)
    "idx_date_group" btree (date_group)
    "idx_price" btree (price)
    "idx_price_group" btree (price_group)
```

Solr のスキーマ設定は次のとおりである：

フィールド名	型	概要
auction_id	string	オークション ID
category_id	string	カテゴリ ID
category_path	text_ja	カテゴリパス
title	text_ja	商品タイトル
url	string	商品 URL
seller_id	string	出品者 ID
image1_url	string	画像 1 の URL
image2_url	string	画像 2 の URL
image3_url	string	画像 3 の URL
init_price	sint	開始価格
price	sint	現在価格
start_time	date	オークション開始日時
end_time	date	オークション終了日時
description	text_ja	商品説明
timestamp	date	インデックス登録日時

上記フィールドのうち auction_id をユニークキーとし、description フィールドをデフォルトの検索フィールドに指定した。なお、text_ja はインデックスに索引付けする日本語のテキストを示し、形態素解析器 Sen により単語分割をしている。

Ludia は SQL の SELECT 文で検索し、レコードの結果セットとして検索結果を得る。一方で Solr は HTTP リクエストで検索し、XML 形式で検索結果を得る。アーキテクチャの異なる両者をなるべく公平に比較するため、検証用の簡易フロント CGI プログラムを Ruby on Rails でそれぞれの検索エンジン用に構築し、最終的な検索結果画面を HTML で得られるようにした。検索エンジンへの負荷がけはこの Rails の CGI プログラムを通じて行った（図 3）。



＜図 3＞

サーバに負荷をかけるクライアントマシンであるが、ノート PC 1 台で行った。これは事前調査の段階でほとんどの場合においてサーバの CPU がボトルネックとなることが判明したため、ノート PC 1 台の負荷で十分であったためである。クライアントからは JMeter で 100 または 30 同時ユーザ（スレッド）を起動して負荷をかけた（試験項目によりユーザ数を変更）。

フロントアプリケーションについて

図 3 のフロントアプリケーションでは次のような HTTP リクエストを受け取り、Rails のコントローラのメソッドにて検索エンジンを呼び出し、得られた検索結果をそれぞれのビュー（.rhtml）にて HTML に変換している：

```
http://localhost/contoller/method
```

ここで"controller"の部分に"ludia"または"solr"を指定することでテスト対象を Ludia または Solr に切り替えられるようになっている。また"method"には次の指定ができるようになっており、これにより実行する検索方法を切り替える：

method	意味
query	通常の検索のみを行う
facet gfacet	通常の検索に加え、ファセットカウントの取得を行う。gfacet は Ludia のみ有効（後述）
filter gfilter	通常の検索に加え、絞り込み検索を行う。gfilter は Ludia のみ有効（後述）

HTTP リクエストにはさらに下記のパラメータを指定できるようになっている：

全 method 共通パラメータ	
q	検索式を指定（必須）
start	結果表示オフセット（デフォルト＝0）
rows	結果表示数（デフォルト＝10）
hl(*2)	ハイライト on または off（デフォルト）。
facet メソッドに有効なパラメータ	
price.N.from price.N.to	N＝0,1,2,...でファセットにする価格帯を指定
date.N.from date.N.to	N＝0,1,2,...でファセットにするオークション終了日の範囲を指定
gfacet メソッドに有効なパラメータ	
facet.price	on または off（デフォルト）。on のときは price_group カラムで group by を実行しファセットカウントを一度で取得
facet.date	on または off（デフォルト）。on のときは date_group カラムで group by を実行しファセットカウントを一度で取得
filter メソッドに有効なパラメータ	
price.from price.to	絞り込む価格帯を指定
date.from date.to	絞り込むオークション終了日の範囲を指定
gfilter に有効なパラメータ	
fq.price	price_group カラムの integer 値を指定
fq.date	date_group カラムの integer 値を指定

(*2)hl パラメータのハイライトとは、検索結果表示画面で検索語の周辺文字列を検索語の強調表示付きで切り出す機能を指す。Senna では snippet と呼ばれる。

それぞれのメソッドの実装方法であるが、Solr に関しては標準的な機能であるため、ここでは説明を省略する。Ludia については各メソッドを次のように実装している：

■/ludia/query

"select 必要なカラムリスト from auction_item where 全文検索条件" で検索した後、"select pgs2getnhits()"でヒット件数を取得する。

■/ludia/facet

通常の検索とヒット件数の取得は/ludia/query と同じである。これに加えて、ファセットカウントの取得は select count(*)の where 句の全文検索条件の後に AND で価格帯や日付の範囲を指定する（この場合、Ludia の制限により pgs2getnhits()は利用できない）。

■/ludia/gfacet

通常の検索とヒット件数の取得は/ludia/query と同じである。これに加えて、ファセットカウントの取得は"group by price_group"または"group by date_group"で一度で取得する。

■/ludia/filter

"select 必要なカラムリスト from auction_item where 全文検索条件 AND フィルタ条件"で検索し、"select count(*) from auction_item where 全文検索条件 AND フィルタ条件"でヒット件数を取得する（この場合、Ludia の制限により pgs2getnhits()は利用できない）。

■/ludia/gfilter

/ludia/filter と同じであるが、フィルタ条件を price_group または date_group カラムを用いて行う。

なお、1 つの検索リクエストを処理するために複数の SELECT 文を発行するときは、1 トランザクションで処理するようにし、ISOLATION LEVEL は SERIALIZABLE を指

定した。

検索性能サマリー

負荷試験の結果を以下の表に示す：

試験種別	検索エンジン	平均応答時間（ms）	QPS
query	Ludia	1,100	45.3
	Solr	920	52.0
hl	Ludia	400	40.0
	Solr	900	17.5
facet	Ludia	計測せず	計測せず
	Solr	1,050	44.4
gfacet	Ludia	計測せず	計測せず
filter	Ludia	1,208	38.2
	Solr	420	113.8
gfilter	Ludia	1,177	38.8

すべてのテストで検索語は Solr のインデックスから任意に 100 個選択した単語を JMeter でループさせて使用した。Solr のインデックスから選択した理由は、インデックスから単語を抽出するプログラムが Lucene で簡単に書けるためである。なお、選択した 100 個の単語で Ludia の検索が問題なくできることは確認してある（しかしながら当然検索結果件数などは両者で異なる）。

query テストでは/xxx/query（xxx は ludia または solr を表す。以下同じ）を 100 同時ユーザで実行した。

hl テストでは/xxx/query に hl=on を指定して 30 同時ユーザで実行した。同時 30 ユーザは Solr がピークのスループットを示したユーザ数である（Ludia はもう少し性能がよく、同時 40 ユーザでピークのスループットを示した）。なお、Ludia と Solr でほぼ同程度の長さの断片（ハイライトスニペット）が返るように調整してある。

facet および gfacet テストは 5 個の価格帯と 7 個のオークション終了日範囲の計 12 個のファセットカウントを取得するシナリオである。このとき Ludia では 1 ユーザでもレスポンスが悪く、負荷試験が実施できなかった。よって/solr/facet のみ 100 同時ユーザで実行した。

(g)filter テストは任意に選択した価格帯とオークション終了日の同時指定による絞り込み検索のシナリオである。(g)filter テストは/xxx/filter と/ludia/gfilter を 100 同時ユーザで実行した。

考察

query テストでは Ludia と Solr で検索性能（平均応答時間と QPS）の大きな差は見られなかった。Solr は前々回のレポート（RONDHUIT REPORT Vol.1）「Solr の検索性能」で示した性能（200 同時ユーザ、200QPS）に及ばないが、これは今回のテストでは Rails のフロント Web アプリケーションが稼動しているためである。実際にフロントの Web アプリケーションが Solr のレスポンスをパースするのに忙しくなってしまい、Solr より先にフロントの Web アプリケーションがボトルネックになるのは本番システムなどでもよく見られる現象である。

hl テストでは Ludia は平均応答時間と QPS の両方で Solr のほぼ 2 倍の性能を達成した。また前述のとおり、この結果は Solr がピークを示した QPS での同時ユーザ数（30 同時ユーザ）である。Ludia は 40 同時ユーザまで QPS が伸びるので、Ludia のピーク性能はこの表の数値よりもさらによい結果となる。

hl テストにおける Ludia と Solr の性能の違いは、それぞれの検索エンジンライブラリのスニペット抽出アルゴリズムの違いからくるものである。Senna（Ludia の検索エンジンライブラリ）ではスニペット取得対象文字列の先頭から検索語文字列との単純一致を見ている。これに対し Lucene（Solr の検索エンジンライブラリ）ではハイライト取得対象文字列は再度トークナイズ処理を経て検索語トークンと比較され、スコア計算がなされてもっとも高いスコアを獲得した断片を返すという凝った処理を行っている。hl テストの Ludia と Solr の性能の差は、この両者のロジックの差が顕著に現れたものである。

facet テストでは Ludia が相当重くなってしまい、負荷試験を実施することができなかった。Solr は負荷試験を実施することができ、query と比較するとやや重くなるにとどまった。

filter テストでは Solr は平均応答時間と QPS の両方で Ludia のほぼ 3 倍の性能を達成した。Ludia の filter テストの結果は Ludia の query テストと比べるとやや重くなっている。このことから全文検索条件に RDB クエリ条件を付加して絞り込み検索を実施すると、全文検索条件単独のクエリよりも遅くなる、ということがわかる。逆に Solr の filter テストは Solr の query テストと比べると 2 倍の性能向上となっており、全文検索条件にフィルタリング条件を加えて絞り込み検索を実施すると、全文検索条件単独のクエリよりも速くなることが確認できた。Solr は過去に使用したフィルタリング条件と結果をキャッシュすることができ、また、絞り込み検索ではそうでない検索よりも検索結果一覧が小さくなるために通常の検索よりも速くなると考えられる。今回の検証により、Solr の絞り込み検索が実際に高速に処理されていることが裏付けられた。ナビゲーションリンクをクリックして検索結果を次々に絞り込んでいく場合、ユーザは絞り込むたびに速いレスポンスを期待するが、Solr のレスポンスはユーザの期待感と合致しているといえるだろう。

Ludia の gfilter は filter とあまり変わらなかった。

まとめ

facet テストが Ludia では実施できなかった。チューニングにより改善できる可能性はあるが、今回のテスト実施期間内でその方法を見つけることはできなかった。しかし、どうやら RDB データソースからの動的ファセットカウント取得のコストが高いことは一般に知られている事実のようである。このため、ファセットカウント集計のバッチを実行して、別に作成したファセットカウント保存テーブルにあらかじめ記録しておき、HTTP リクエストが来たときにそのテーブルからカウントを取得して表示する、という実装が多くのプロジェクトで採用されている。

しかしこの方法は、(a)ファセットカウント保存用の大量の管理テーブルが必要になる、(b)ファセットカウント集計用の大量のバッチ実行が必要になる、という点で管理・運用コストの増大をまねくこととなる。さらに、(c)バッチ実行時にファセットの条件が静的に決定してしまう、(d)フリーワード検索の結果と連動させるのが困難、という機動性・柔軟性に欠けるシステムを生み出す原因となる。

Solr の場合はファセットカウントを動的に集計しても十分に速く処理することができる。Solr を使用すれば RDB で必要なバッチ実行は一切不要となるため、管理・運用コストを抑えることが可能である。また動的に実行できることから、適当な絞り込み条件をセットにした「キャンペーン」を定義して新しいリンクをトップページに設置することも比較的簡単に行える。ナビゲーションの導線を増設するのに、フロントの Web アプリケーションへの少しの変更で対応できるようになる。

ファセットカウントはユーザに対し、「このリンクをクリックすると検索結果を xx 件に絞れる」ということを伝える重要な情報となっており、ファセットカウントを表示する Web サイトは今後も増えていくと考えられる。このとき、その実装方法として RDB による静的事前バッチ集計方式で構築するのか、Solr などの検索エンジンを導入して動的リアルタイム集計方式で構築するのかでその後のシステムの TCO や機動性にも大きく違いが出てくるだろう。現在は RDB による静的事前バッチ集計方式が一般に広く採用されているが、今後は検索エンジンによる動的リアルタイム集計方式が認知され、適用するプロジェクトが徐々に広まっていくと思われる。

㈱ ロンウイットについて

ロンウイットはオープンソースの全文検索エンジン Lucene/Solr を企業システムに導入する支援サービス事業を展開している。

お問い合わせ先

〒100-0005
東京都千代田区丸の内 1-1-3 AIG ビル B1F
電話：03-5288-5927　　FAX：03-5288-5353
メール：sales@rondhuit.com
ホームページ：http://www.rondhuit.com/