デジタル・ドキュメント 7－3
（1997. 5. 23）

高速テキスト検索エンジン

松井 くにお, 難波 功, 井形 伸之
富士通研究所

本論文では，大量の文書情報を高速に検索する全文検索エンジン Teraß を開発したことを報告する。大容量情報の検索では，検索もれと検索速度に問題点があった。Teraß では，日本語文書における検索もれを排除するために文字成分抽出方式を採用した。また，検索速度を向上させるために，I/O の効率化や新たなインデックス圧縮手法を考案し，圧縮効率を３割以上高めた。これらの高速化手法により，ギガバイトを越える大容量文書の検索に対し，実用上十分な性能を得た。

Hi-speed Fulltext Search Engine

MATSUI, Kunio　　NAMBA, Isao　　IGATA, Nobuyuki
FUJITSU LABORATORIES LTD.

This paper describes the high speed fulltext search engine(Teraß) for large-scale data. In a large-scale Japanse fulltext database, the main concerns are false drop and the searching speed. We adopt the character N-gram indexing for Japanese text to avoid false drops. We design the new compressing method of inverted file to reduce I/O and to speed up decoding in searching time. Expreiment using Japanse patent abstracts(1 Giga bytes) shows these methods are quite effective in large-scale data.

1 まえがき

1990 年代に入り，インターネットが爆発的に流行し，電子化文書の流通量が増大している。これに伴い，欲しい文書を即座に検索できるシステムへの要求が急速に高まっている。現在，国内外において様々な検索システムが開発され，多くの会社および団体がインターネット上で検索サービスを行なっている。

大量の文書を検索する事例としては，*AltaVista*[1] などに代表されるインターネットホームページ検索や企業内で活用されている特許情報検索などがある。これらの検索システムで扱われるデータ量は，インターネットホームページ検索で 100-200 Gバイト，特許全文検索で 30 Gバイト と膨大な量となっている。ところが，従来の検索システムは数百Mバイト 程度の文書容量を前提として設計されていたため，大容量文書に対して十分な性能を得ることができなかった。さらに，日本語文書を検索対象とした場合，本来検索されるべき文書が検索結果から落ちてしまうといった検索もれが発生することもあった。

筆者らは検索もれがなく高速に大量の文書を検索する全文検索エンジン Teraß (TERASS: The Express Retriever Advanded for Smart Searching) を開発した。本論文では，Teraß を開発する上で優先したポイントとそのために採用した手法について解説する。

2 全文検索とは

文書内のすべての単語を検索対象としたものを全文検索という。これに対し，各文書にキーワードを付加し，キーワード部分だけを検索対象としたものをキーワード検索という。全文検索を実現する手法には，Unix の grep コマンドに代表される文字列検索と，文書中の語句をあらかじめインデックスに登録するインデックス検索がある。

検索実行時に検索対象となるすべての文書を逐一調べあげる文字列検索は，検索速度が遅く，ある程度の文書容量を検索対象とする場合には実用的ではない。そこで，大容量検索の場合にはインデックス検索を用いるのが主流となっている。

インデックスの構築例を表 -1 に示す。インデックスは，ある特徴素（検索のキーを構成する文字列）とその特徴素に対するデータ部から構成されている。このようなインデックスを文書検索に用いる場合，文書中に出現する語句を特徴素，その語句を含んでいる文書番号をデータ部としてインデックスを構築する。検索実行時には，検索する文字列と同じ文字列の特徴素のデータ部を参照し，その文字列を含む文書を検索する。

表 -1 インデックスの構築例

検索対象文書

文書番号	内容
文書 1	パソコン市場における富士通の
文書 2	富士通のパソコンＦＭＶ
文書 3	健さんも満足ＦＭＶ

インデックス

特徴素	データ部
パソコン	文書 1，文書 2
市場	文書 1
富士通	文書 1，文書 2
ＦＭＶ	文書 2，文書 3
健さん	文書 3
満足	文書 3

3 大容量全文検索の実現におけるポイント

一般に，検索システムの性能は，以下の項目によって評価できる。

1. 検索速度

2. 高機能（もれのない検索，類似文書の検索等）

3. インデックスサイズ

4. インデックス作成速度

5. インデックス追加速度

6. インデックス更新速度

しかし，これらの項目は互いにトレードオフであり，すべての項目の性能を高めるような手法は現在のところ開発されていない。

Teraß を開発する上で優先したポイントは，以下の点である。

1. 日本語文書での検索もれをなくすこと

2. 検索速度が高速であること

3.1 検索もれをなくす

3.1.1 日本語文書特有の問題点

日本語文書に対してインデックス検索を適用する場合，それぞれ文書からどのように特徴素を抽出するか，つまり文書からどのように単語を切り出すかが問題となる。日本語の単語の切り出しは，現在の形態素解析レベルでもかなり困難な処理となっている。形態素解析でうまく扱えないものの例を表 -2 に示す。

表 -2 形態素解析でうまく扱えない例

文字列	特徴素抽出例（二者択一）
例 1-1 「システム部門長谷川」	システム／部門長／谷川 システム／部門／長谷川
例 1-2 「赤色補正」	赤／色補正 赤色／補正

表 -2 の例 1-1 は，もともとの文字列があいまい性を有している例である。例 1-2 は，単語をどこで切るかの問題である。もし例 1-2 の文字列が「赤色／補正」で単語切りされインデックスに登録されたとすると，「色補正」を入力して，「赤色補正」を含む文書が検索できなくなってしまう。つまり，この単語切りの問題が，日本語文書検索時の検索もれの原因となっている。このような形態素解析による特徴素抽出に代わり，文書中の文字成分を特徴素とするものが提案されている [2),3)]。1，2 文字成分による特徴素抽出例を表 -3 に示す。

表 -3 1，2 文字成分による特徴素抽出例

文字列	特徴素抽出例（1 文字単位か 2 文字単位）
例 2-1 「システム部門長谷川」	シ／ス／テ／ム／部／門／長／谷／川 シス／ステ／テム／ム部／部門／門長／長谷／谷川
例 2-2 「赤色補正」	赤／色／補／正 赤色／色補／補正

表 -3 に示すような文字成分で特徴素を抽出した場合，「色補正」の検索は，「色補」と「補正」の AND で検索が行なわれるため，単語切りによる検索もれが発生しない。

このように，もれのない検索を目標とすると，現在のところ文字成分による特徴素抽出が有効であると考えられる。しかし，文字成分による特徴素抽出方式を用いた場合，抽出される特徴素数が形態素解析で得られる特徴素数よりも多くなり，インデックスサイズを増加させる。さらに，検索もれの問題を解決するかわりに，検索ノイズという新しい問題を発生させてしまう。検索ノイズとは，検索した文字列を含まない文書を引

いてくることである。例えば，「八戸市」を検索した場合に，「八戸市」の２文字成分は「八戸／戸市」となるため，単に「八戸」と「戸市」の AND 検索を行なうと「…松戸市に住宅八戸…」のような文書も検索してしまう。

3.1.2 検索速度を低下させずに検索ノイズを排除する

検索ノイズの問題は，インデックスに各特徴素の文書内出現位置情報を持たせて，二つの特徴素「八戸／戸市」が隣接しているかどうかをチェックすることによって解決できる。しかし，長めの文字列を検索した場合，隣接チェックがかなり重い処理となる。例えば，「クリントン」を検索した場合に，２文字成分は「クリ／リン／ント／トン」となるが，それらすべてが横一列に並んでいるかチェックするためには，３回の隣接チェックが必要となる。この隣接チェックの回数が増えるにしたがって，検索速度は低下してしまう。

隣接チェックの回数を減らすには，文字成分の長さを長くとればよい。例えば，４文字成分をとった場合，「クリントン」の特徴素は「クリント／リントン」となり，隣接チェック１回で検索できる。しかし，文字成分を長くとればとるだけ，抽出される特徴素数が増加し，インデックスサイズを巨大なものとしてしまう。インデックスサイズの増加は，検索実行時の I/O を増加させるため，せっかく隣接チェックの回数を減らしても，逆に検索速度を劣化させることになる。

そこで，Teraß で用いている文字成分抽出では，文字種を判定し，文字種の違いにより成分の長さを切替える処理を行なっている。例えば，漢字ならば１，２文字単位，カタカナならば１，２，３音節単位というように，成分の長さを変えて特徴素を抽出している。このような処理を行なうことにより，それほど特徴素数を増やさずに，隣接チェックの回数をできるだけ減少させ，検索速度の低下を防いでいる。

3.2 検索速度を高める

Teraß では，検索速度を向上させるために，以下の点を考慮した。

1. I/O 効率のよいデータ構造
2. インデックス圧縮
3. 復号処理回数の削減

これらについて具体的に記述する。

3.2.1 I/O 効率のよいデータ構造

検索速度を向上させるためには，当たり前のことだが計算機の負荷をできるだけ避けること，つまり I/O をできるだけ減少させることが重要である。

検索実行時の I/O を考えると，できる限り情報を詰め込んだインデックス構造にした方がよい。データベースのようなページ形式のインデックス構造にした場合，同じ特徴素に関する情報をいくつものページに分散してしまうと，ディスク上に分散するページを読み込むため，I/O の負荷が増大し，検索速度が一桁以上低下する。そこで，Teraß では，同じ特徴素に関する情報を１箇所にまとめることにより，I/O の効率化を図り，高速検索性能を実現している。

このように同じ特徴素に関する情報を１か所にまとめると，インデックスの各ページに更新の余地が少なくなり，インデックス更新性能を低下させることになる。しかしながら，一般に大容量文書では文書が頻繁に更新されることはなく，基本的に蓄積が主であるため，更新性能はそれほど重要ではないと考えられる。

実際，大容量文書の例として挙げられる特許や新聞では，基本的に修正が入らない文書を蓄積するＤＢであるため，更新性能が問題となることはない。また，１，２週間で全データの約半分が更新されるインターネットホームページのようなＤＢでは，更新処理を行なうよりもインデックスを再構築した方が速い。そこで，Teraß では，検索速度とは独立に性能を高められるインデックス作成速度の向上に力を入れることで，更新性能の劣化をカバーする仕様とした。

3.2.2 インデックス圧縮

前述のように，検索ノイズを排除するため特徴素数を増やすことは，インデックスサイズを巨大化させる。インデックスサイズの増加により検索実行時の I/O が増えてしまうため，インデックス内の数値データを圧縮し，I/O を減少させることが重要である。

しかし，検索実行時に圧縮されたデータを元の数値データに復元する処理が増えるため，あまり復号処理に時間がかかる圧縮手法では，十分な検索速度が得られない。そこで，筆者らは，圧縮率は高いが処理速度の遅い γ-coding [4] と呼ばれる標準的な数値の符号化手法を拡張し，同等の圧縮率で高速な 新４ビットブロックコーディングを開発した。これは、４ビット単位の差分圧縮の先頭１ビットをまとめることによって復号速度をそれほど落さずにインデックスの圧縮率を３割向上させている（図１）。

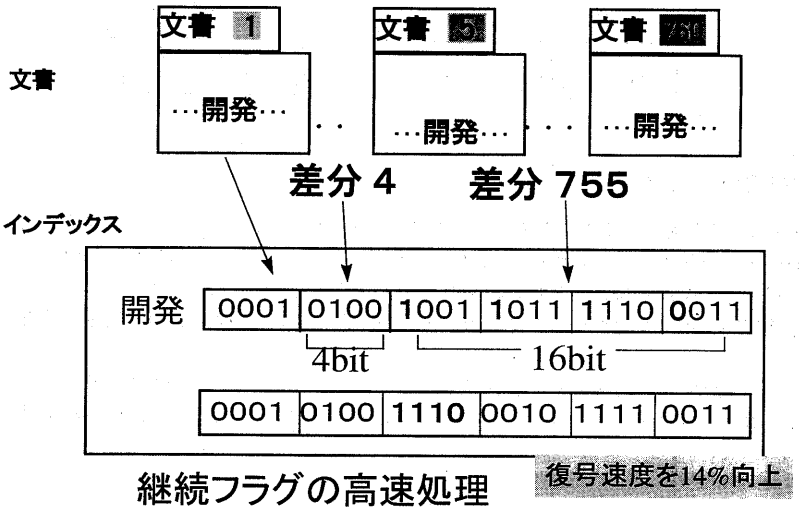

図１　新４ビットブロックコーディング

以下の条件で圧縮性能を測定したところ，Teraß では，この圧縮手法を用いることにより，商用システムで標準的に使われている圧縮手法（８ビットブロックコーディング [1),2)]）に比べて，表-4 の性能を得た。

測定マシン SUN-4/20 HS model 22(CPU: SpecInt92 133, MEM: 256 M bytes)

CC 環境 Solaris 2.5.1, gcc 2.7.2 -O2

測定対象 特許抄録 167 万件

- 原文サイズ: 約 1.0 G bytes (1107228996 bytes)
- 平均文書サイズ: 約 666 bytes
- 全文書数: 1,662,506 件
- 全 N-gram 数: 632,587 個
- 全文書番号数: 395,246,189 個
- 全文書内単語出現頻度数: 395,246,189 個
- 全文書内単語出現位置数: 759,145,192 個

表-4 圧縮性能 (M bytes)

	32ビット	8ビット	新4ビット	ガンマ	デルタ
文書番号（差分値）					
size	1507.7	449.7	362.1	386.6	**361.0**
(%)	(100)	(29.8)	(24.1)	(25.6)	(23.9)
文書内単語出現頻度					
size	1507.7	377.0	192.6	**96.9**	110.9
(%)	(100)	(25.0)	(12.8)	(6.4)	(7.3)
文書内単語出現位置（差分値）					
size	2895.9	952.6	**901.9**	1085.0	998.0
(%)	(100)	(32.9)	(31.1)	(37.5)	(34.5)
文書内単語出現位置長					
size	1507.7	377.0	**216.7**	397.1	402.0
(%)	(100)	(25.0)	(14.4)	(26.3)	(26.7)
全体の累計					
size	7419.0	2156.3	**1673.3**	1965.6	1871.9
(%)	(100)	(29.1)	(22.6)	(26.5)	(25.2)

3.2.3 復号処理回数の削減

インデックス内の数値データを圧縮すると，各数値へのランダムアクセスが不可能となる。例えば，100個並んだ数値データを圧縮してしまうと，99個目の数値にアクセスするためには，1個目のデータから99回復号処理をかけなければならない。1個の数値を復号すること自体はそれほど重い処理ではないが，数値の復号化は，検索実行時に何度も繰り返して行なわれる処理であるため，検索速度性能を決定する要因の一つとなる。

Teraß では，圧縮された数値データを等間隔にサンプリングし，それらが書き込まれているインデックス内の位置を持つスキップリスト[5),6)]を作成することにより，復号処理回数を削減し，検索速度の向上を図っている（図2）。

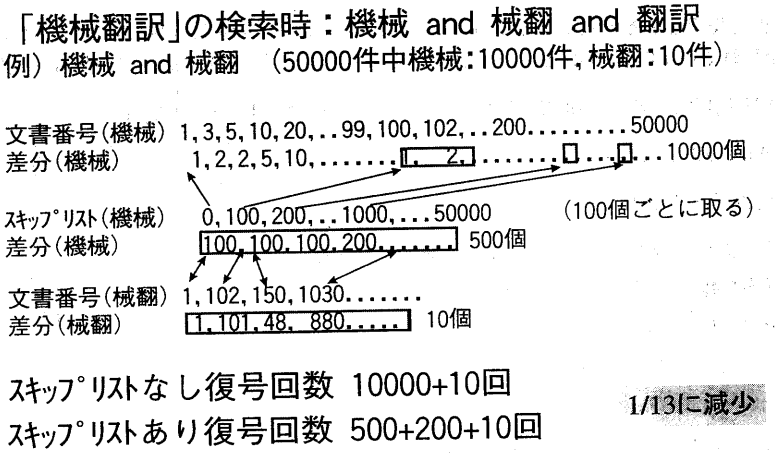

図2 スキップリストによる復号の省略の例

4 評価

Teraßの性能を表-4に示す。実験はSun4/20 HS Model22 (Solaris 2.5 SpecInt92 133) の環境下で日本語特許抄録166万件(1 Gバイト)に対して行なった。検索速度は，検索要求977個を連続して処理し，その平均(全体の検索時間/977)を表したものである。検索要求は，社内の特許検索サービスで実際に検索された検索ログから収集した。166万件の文書(1 Gバイト)に対して，1検索要求当たり0.02 ～ 0.21秒の速度性能を達成している。

表-5 Teraßの性能（日本語特許抄録166万件(1 Gバイト)を検索対象とした場合）

インデックスタイプ	文書内出現位置なし	文書内出現位置あり
インデックス作成時間	4.3時間	4.8時間
インデックスサイズ	420 Mバイト	1.5 Gバイト
ノイズ率	7%	0%
OR検索速度	0.05秒	0.21秒
AND検索速度	0.02秒	0.13秒

5 むすび

実際の検索業務において，許容できる検索速度は1検索要求当たり1～2秒であると言われている。また，特許調査の場合，少々検索ノイズが混じっても絶対に検索もれを発生させないことが一番に要求されている。これらの要求に対し，今回筆者らが開発したTeraßは，実用上十分な検索性能を達成し，検索もれをなくしている。

年々，CPUの速度性能は向上しているが，その速度向上を上回るスピードで検索対象となるデータ量やネットワークアクセス量が増加している。本論文で述べた各要素技術は，数Gバイト～十数Gバイト程度の文書容量を想定し開発を行なったため，T（テラ）バイトオーダとなるような超大容量文書情報に対し十分な性能が得られる保証はない。このような問題に対処するために，現在，筆者らは並列計算機を用いた並列検索エンジンの開発を進めている。

参考文献

1) 丹波 ほか: AltaVistaにおける大規模検索。Proceedings of Advanced Database Symposium '96，東京，情処DB研，1996, pp.19–25。

2) 赤峯 ほか: 高速全文検索のためのフレキシブル文字列インバージョン法。Proceedings of Advanced Database Symposium '96，東京，情処DB研，1996, pp.35–42。

3) 菅谷 ほか: n-gram型大規模全文検索方式の開発。情報処理学会第53回（平成8年後期）全国大会論文集，大阪，情報処理学会，1996, 3-237–238。

4) Elias P.: Universal codeword sets and representations of the integers。IEEE Trns, of Info IT21, pp.194–203 (1975)。

5) William Pugh.: Skip Lists A Probablistic Alternative to Balanced Trees。CACM, 33–6, pp.668–676 (1990)。

6) Ian H. W., Alistair M., Timothy C. B.: Managing Gigabytes。初版，New York, VAN NOSTRAND REINHOLD, 1994, pp.138–140 。