ログ構造に構成された分割インデックス

Partitioned Indexes Formed in Log-structure

佐藤 隆士♥
Takashi SATO

新聞，特許，WEB，XMLなどの大規模文書集合に効率的にアクセスするためには，インデックス作成が不可欠なものとなっている．これら文書集合に対する更新は，新しい文書の追加が主で，修正や削除は比較的少ないため，この特性に着目してインデックスを構成することが考えられる．一方，コンピュータの当面の性能向上は，マルチコア化，クラスタ化などの分散並列処理にたよらざるを得ない状況である．本論文では，大規模文書集合の特性と最近のコンピュータの構成に適合した，分割されたインデックスシステムの提案を行う．(1)保守を容易にし且つ分散並列処理に適合するよう複数に分割されたインデックスシステムとする，(2)文書の更新パターンの相違に着目して作成順による文書グループごとの小インデックスを作成する，などの特徴を有す．文書の追加の際は，最終文書グループ用のインデックスに限定することからログ構造の分割インデックスと呼ぶ．

Indexing is indispensable in order to access huge set of documents. In most cases, updating of these documents is almost insertion, and deletion and modification are rare. So indexing focused on these updating properties should be considered. This paper discusses partitioned indexes which are made by the document groups divided in the order of creation. We call this index system log-structure index because key words from new documents are always added to the last partial index.

1. はじめに

我々は，新聞，特許，WEB，XMLなど大量のオンライン文書から，目的とする文書にアクセスために検索の必要性が増している．このような大量の文書に効率的にアクセスするためには，文書に対して，インデックスを作成するのが一般的である．しかし，文書集合の巨大化に伴い，インデックスの作成並びに文書の更新に連携したインデックスの保守が以前にも増して負担になっている．

一方，これら大規模文書集合の更新には，新しい文書の追加が主で修正や削除は比較的少ない，古い文書の修正はほとんどないなどの特徴的パターンが認められる．従って，これらの特性に着目したインデックスの構成法があるものと考えられる．

インデックスの処理装置となるコンピュータの性能は向上しているが，単一処理装置の飛躍的な性能向上は期待できない状況である．当面の性能向上は，CPUのマルチコア化，コンピュータのクラスタ化などであり，分散並列処理にたよらざるを得ない[1][2]．

このような状況から，本論文では，大規模文書集合の特性と最近のコンピュータの構成に適合した，以下の特徴をもつインデックスシステムの提案を行う．

(1)保守を容易にし且つ分散並列処理に適合するよう複数に分割されたインデックスシステムとする．

(2)文書の更新パターンの相違に着目して作成順による文書グループごとの小インデックスを作成する．

文書の追加の際は，最終(新)文書グループ用のインデックスに対して行う．ハードディスクのアクセス特性に合わせて，ファイルの書き込みを特定部分に集中することにより性能向上を目指したログ構造ファイルシステム[3]になぞらえ，提案のインデックスシステムをログ構造の分割インデックスと呼ぶことにする．

インデックスの分割は，商用データベースにも実装されている[4][5]．これらでは，分割の方法をデータベースの値に基づいて指定するが，本論文で提案の方法は，データ到着順で分割が行われるため分割方法を指定する必要がない．

オープンソースの検索ソフトLucene[6]では，インデックスがセグメントと呼ばれる複数の独立した部分インデックスから構成されている．動的に，且つスケーラブルにインデックスを保守するための構成[7]で，最適化の際にはこれらは１つのより大きなセグメントにマージされる．本論文提案の方法が分散処理による高速化を目的として積極的に分割しているに比べ，分割の目的は異なるが関連研究と言える．

筆者らは，XMLデータベース用の分割インデックスを提案している[8]が，ログ構造の分割インデックスは，これを一般の場合に拡張したものになっている．

以下，2.では提案のインデックスシステムの構成を説明する．3.では，分割しないインデックスとの比較をしながら，検索と更新の処理方法を説明する．4.は実験結果である．インデックスを作成した環境と性能測定の結果を示す．

2. インデックスの構成と作成コスト

2.1 インデックスの構成

文書集合をコーパス$\boldsymbol{C}$とし，その分割を$C_i(i=1,2,\ldots,m)$とする．C_iの転置ファイル$T_i(=C_i^t)$の集合$\boldsymbol{T}=\{T_1,T_2,\ldots,T_m\}$が提案の分割インデックスである．ここで，コーパス$\boldsymbol{C}$は要素となる文書の作成時間順(到着順)にグループ化されC_iに分割されるものとする．分割インデックスはこのグループごとの小インデックスである．図1に構成を示す(図中Sは，追加文書用の分割インデックスである)．通常のインデックスを転置ファイル(Inverted File)とも言うが，分割インデックスは部分転置ファイル(Partial Inverted File)に相当する．

転置の方法は応用によって異なる．例えば全文検索の場合は，文書を構成する語を見出しに文書IDを求める，リレーショナルデータベースでは属性値を見出しにタプルIDを求めるなどである．

2.2 インデックスの作成コスト

文書の作成順に一定数のグループごとに小インデックスを作成する．グループごとに独立にインデックス作成が可能なため，容易に分散並列で処理できる．コーパス全体の索引件数をD，分割数をmとする．分割しない場合の作成コストを$M=c\,D\log D$と見積もる．但し，cは比例定数であり，log

♥ 正会員　大阪教育大学情報処理センター
sato@cc.osaka-kyoiku.ac.jp

の底は 2 とする．log D は，転置のための比較によるソーティングの項である．

一方，分割数 m の分割インデックスの場合，各インデックスの作成コストは D の代わりに D/m と置くことにより，$M_m = c\ (D/m) \log (D/m)$ を得る．分割インデックスを m 台の処理装置で分散処理を行う場合，処理コストの比は，$M/M_m = m \log m$ となる．1 台の処理装置で行う場合は，m 倍かかるため，コスト比は $\log m$ に縮小するが，依然として分割インデックスの方がコストが小さい．この $\log m$ 倍は，部分転置ファイルから完全転置ファイルを作成するためのコストでもある．

インデックス中には索引語情報よりむしろ検索結果となる文書 ID を記憶するスペースが大きくなる．文書 ID は検索対象の文書数の 2 を底とする対数の bit 長で表現可能なため，検索対象の文書数の少ない分割インデックスはスペースの面でも有利となる．即ち，各分割インデックス中では，文書 ID は，ローカルに一意な値とすることでスペースを節約している．検索結果のマージ時点など，グローバルに一意にする必要がある際には，処理装置 ID を付加（連接）して扱う．

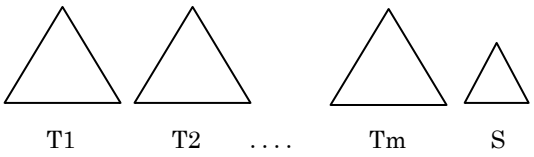

図 1　ログ構造分割インデックスの構成
Fig.1　Partitioned Indexes

3. 検索と更新処理

3.1　検索処理

分割インデックスでは，分割しない単一のインデックスに比べサイズが小さいため高速な探索が期待できる．各分割インデックスを担当する複数の処理装置に，クエリをマルチキャストして探索させ，目的の文書 ID を取得する．複数の分割インデックスの探索が必要となるが，分散処理で同時に探索可能である．

分割インデックスの探索結果のマージ時点で特定処理装置に負荷が集中することになるが，通常のクエリでは探索結果の集合がコーパスに比較して格段に小さいため，問題になることは少ない．また，探索後の処理も引き続き分散して行う場合は，検索段階処理でのマージは不要になる．

3.2　更新処理

更新操作は，追加，修正，削除からなるが，本論文では議論を単純にするため，修正は削除と追加の組み合わせで実現することとし，扱わない．

3.2.1. 追加

追加は文書の最終グループ用のインデックスに対して行う．分割しない場合に比べ，インデックスサイズが小さいため高速である．

インデックスを分割しない場合，全索引件数 D に対して，索引 1 件あたりの追加コストを，$c \log D$ と見積もる．2 分探索で比較を行いながら挿入場所を決めるため，log D 回の比較を行うためである．一方，m 個に分割したインデックスでは，最終グループの索引件数は D/m 以下とする．このため追加コストは c log (D/m) 以下と見積もることができる．この場合もコスト比が $\log m$ 倍以上となる．

なお，追加により最終グループが他のグループと同じ程度の大きさになった場合は，新グループを作成して索引を追加する．その結果，今までの最終グループは，最後から 2 番目のグループになる．

また，最終文書グループ以外は別の分割インデックスで索引付けられているため，これらインデックスを使用した検索は，文書追加で発生するインデックスの更新処理を待つことなく実行可能である．即ち，検索結果に必ずしも最新の文書グループを必要としない場合は，更新操作と検索が並行に処理できる．データの到着順でなく，値で分割した場合は，このような検索と更新の独立性が保障されないため，並行処理は困難である．

3.2.2 削除

文書の削除時には，削除表に文書 ID を記入するのみで，インデックスに反映しない方式とした．削除表は各分割インデックスを担当する処理装置ごとに用意して，メモリ中に置く．追加が主で，修正や削除は比較的少ない場合を想定しているので，使用スペースの増加は緩やかである．検索時には，インデックスを用いた検索後，削除表に含まれる文書 ID は除外する必要がある．この処理は検索時のオーバヘッドになるが，削除表をメモリ中に置くため，他の処理に比べ無視できる程度となる．また，長期間にわたる削除で削除表が負担になった場合は，計算リソースの空きのタイミングで，分割インデックスを更新して削除表を空にする．この処理は稀なため，割合からすると，日常の検索，追加の処理に比較し，無視できる程度である．

4. 実験結果

国立情報学研究所の情報検索システム評価用テストコレクション(NTCIR)から NTCIR-6 言語横断検索(CLIR)タスク[9]に用いられた新聞 2 紙の 2 年間(2000-2001)，計 4 年分 858,400 件の記事の本文(約 1Gbyte)をコーパスとしてインデックス作成実験を行った．索引語には可変長グラム[10]を使用しており，その総数は約 4 億(398,933,412)である．使用コンピュータと OS は以下のとおりである．

CPU　：AMD Opteron 250 × 2
Memory：16GB
Disk　：Hitachi HDS724040KLAT80 × 2
OS　：FreeBSD 6.1

なお，インデックス格納に Berkeley DB 4.4.20[11]をアクセス法 B-tree で利用している．

比較のため分割しない単一インデックスと，本論文で提案のログ構造分割インデックスの両方を作成し，検索，追加および削除の性能を測定した．

4.1 作成

単一インデックスと分割インデックスの作成時間とサイズを表 1 に示す．分割インデックスの行には 9 分割された部分インデックスの総和が記載されている．また，分割は索引件数約 45,000,000 ごとに行っている．作成時間，サイズとも分割インデックスの方が小さい．分割インデックスは，本実験では直列に作成した時間である．分散並列で作成すれば，更に分割数分の 1 に時間短縮されることになる．

4.2 検索

NTCIR6 CLIR の 140 題の検索課題(Topic)から 1,027 語を検索した際の検索時間を表 2 に示す．分割インデックスは 9 組からなるが，1 個あたりの検索の平均と最大値と，全分割インデックスを順に直列に検索した時間を示している．並列

分散環境で行えば，分割インデックスの検索のうち最長(27秒程度)での検索が期待できる．

表1　インデックス作成
Table 1　Indexing

	時間(分)	サイズ(GB)
単一	101	8.90
分割	40.5	6.70

表2　検索時間
Table 2　Retrieval Time

	時間(秒)
単一	68.9
分割(平均)	21.6
分割(最大)	26.8
分割(全体)	117.4

4.3 追加

追加は，100 記事の索引(総数 86,974)をインデックスに挿入している．分割インデックスでは最終の部分インデックスに挿入している．表 3 に測定時間を示す．分割インデックスの方が 5 倍以上高速であることが分かる．

表3　追加時間
Table 3　Addition time

	時間(秒)
単一	330
分割	59.0

関連して，単一インデックス作成時に得られた索引件数と追加時間の関係を図 2 に示す．各分割インデックスの追加性能は図中の左端の棒に，単一インデックスの場合は右端の棒に相当する．

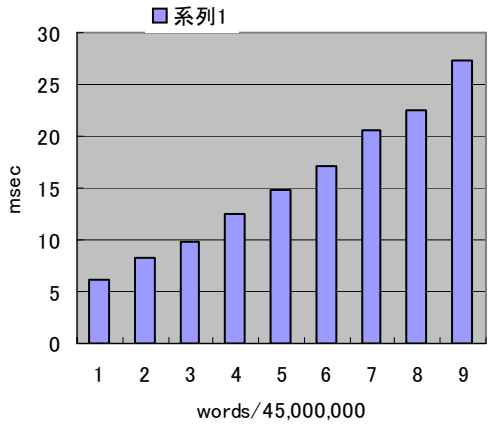

図2　インデックス中の索引件数と
1000 語あたりの追加時間の関係
Fig. 2　Number of index terms vs.
addition time per 1000 terms

4.4 削除

ランダムに選択した 100 記事の索引(総数 89,652)を削除した．単一インデックスでは，354 秒を要した．一方，提案の分割インデックスでは削除は **3.2.2** で述べたようにメモリ中の削除表(各分割インデックスあたり 9 ないし 11 記事)の管理だけで行うためほとんど時間を要しない．測定誤差範囲内の秒数(0 秒)であった．

5. まとめ

本論文では，大規模文書集合の特性と最近のコンピュータの構成に適合するよう，作成順の文書グループごとに小インデックスを作成する分割されたインデックスシステムの提案を行った．文書の追加の際は，最終文書グループ用のインデックスに限定することからログ構造の分割インデックスと命名した．インデックス構造を説明し，その作成，検索および更新コストについて解析を行い，従来の単一インデックスの場合と比較した．更に NTCIR6 の文書データから実際にインデックスを作成することにより，作成，検索，更新性能の測定を行い，有効性を確認した．

今後，より広範な応用を目的とした，実装実験並びに改良が必要であると思われる。

[文献]

[1]サン・マイクロシステムズ-UltraSPARC T1 プロセッサ : http://jp.sun.com/products/processors/UltraSPARC-T1/.
[2] インテル® Core™2 Extreme プロセッサー : http://www.intel.co.jp/jp/products/processor/core2XE/index.htm.
[3]M. Rosenblum and J. K. Ousterhout : The Design and Implementation of a Log-structured File System, ACM Trans. Computer Systems, Vol. 10, No. 1, pp.26-52 (1992).
[4]Partitioning in Oracle Database 10g Release2 : http://www.oracle.com/technology/products/bi/db/10g/pdf/twp_general_partitioning_10gr2_0505.pdf.
[5]Special Guidelines for Partitioned Indexes : http://msdn2.microsoft.com/en-us/library/ms187526.aspx.
[6]Apache Lucene - Index File Formats : http://lucene.apache.org/java/docs/fileformats.html.
[7]D. Cutting, J. Pedersen : Optimizations for Dynamic Inverted Index Maintenance, Proc. ACM SIGIR Conf. (SIGIR' 90), pp.405-410 (1990).
[8]佐藤, 張: 検索と更新が並行可能な分割された XML-DB 索引, 電子情報通信学会第 16 回データ工学ワークショップ (DEWS2005), 2A-i5(2005-03).
[9]NTCIR-6 : http://research.nii.ac.jp/ntcir/ntcir-ws6/ws-ja.html.
[10]T. Sato, K. Han : NTCIR-3 CLIR Experiments at Osaka Kyoiku University -Compression of Gram-based Indices-, Proc. NTCIR-3 CLIR, pp.149-151(2002).
[11]Berkeley DB | Oracle Embedded Database : http://www.oracle.com/database/berkeley-db/index.html

佐藤 隆士　Takashi SATO
大阪教育大学情報処理センター教授．1978 年岡山大学工学研究科修士課程修了．同年詫間電波工業高等専門学校助手．1985 年工学博士．1990 年大阪教育大学助教授．全文検索，XML 検索などの研究に従事．