情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2009-DBS-149 No.2
2009/11/20

全文検索におけるスケーラブルな動的索引構築手法の提案

山田浩之[†1]　遠山元道[†1]

転置索引は、現在の全文検索において非常に重要な構成要素である。転置索引は、検索可能になるまでの遅れが許容できない環境においては動的に構築され、常に検索可能であり、最新でなければならない。近年、効率的な動的索引構築手法が多く提案されているが、それらはマルチコア CPU などの近代的ハードウェアにおけるスケーラビリティへの配慮が足りないという問題点が挙げられる。本論文では、マルチコア CPU を効率的に利用するスケーラブルな動的索引構築手法を提案する。30GB のウェブ文書での実験により、本手法の効率性を実証し、検索速度を落とすこと無く動的な索引構築時間を大幅に削減できることを示す。

Scalable Online Index Construction with Multi-core CPU

Hiroyuki Yamada [†1] and Motomichi Toyama[†1]

Inverted index is a core element of current text retrieval systems. They can be dynamically constructed using online indexing approaches in the environment which even a small delay in timeliness cannot be tolerated, and the index must always be queryable and up to date. Recently, efficient online index construction schemes are proposed, however, previous works have not focused on scalability with the modern commodity hardware resources such as multi-core CPUs. In this paper, we propose a scalable online index construction method that better utilizes multi-core CPUs. Using experiments on 30 GB of web data, we demonstrate the efficiency of our method in practice, showing that it dramatically reduces online index construction time without sacrificing query performance.

†1 慶應義塾大学
Keio University

1. はじめに

転置索引は、現在の全文検索において非常な重要な構成要素である。転置索引では、単語とその単語を含む文書集合との対応付けを行う。この対応付け情報をポスティングと呼び、各ポスティングは文書に単語が出現するかどうかを表現する。また、ブーリアン検索以外の用途では、文書内での単語の出現頻度や出現位置などの付加情報を含む場合もある。この転置索引の構築は、静的な手法、または、動的な手法のいずれかにより行われる。静的な手法は、入力データに対して構築処理を行い、それらの処理の完了時に検索可能となるような構築手法である。一方、動的な手法は、検索可能になるまでの待ち時間を少しも許容できない場合に使われ、索引を常に検索可能、かつ、最新となるように構築する手法である。現在の環境においては、動的な手法による索引の構築がより多く使われ、近年、様々な手法が提案されてきている。

動的な索引の構築において、既存の索引に対して新たな文書を追加する基本的な方法は、メモリ上とディスク上の各々に 2 つの索引を管理することである。新しい文書に対するポスティングは可能な限りメモリ上に蓄えられ、その後、ディスク上の既存の索引と統合される。この統合操作は、既存の索引にデータを追加するインプレイスな手法による場合[17] と、既存の索引と新しいデータをマージして新しい索引を作成するマージベースの手法[9] とが存在する。近年は、マージベースの手法が主流となり、より効率的な手法が提案されてきている。[3),10)] それらの特徴は、検索処理速度を大幅に低下させない程度にディスク上の索引を複数個保持することを許容し、索引の構築パフォーマンスを向上させることである。

近年、急速なハードウェアの進化は未だ衰えず、ほんの数年前よりも多くの計算資源を簡単に手に入れることができるようになった。CPU に関しては、主要なベンダーではデュアルコア、クアッドコア CPU を製品化し、また、いくつかのベンダーにおいては 8 コア CPU の製品化を行っている。また、メモリの価格低下と 64 ビットの OS の影響もあり、大量のメモリが利用可能となっている。現在のデスクトップマシンにおいては、デュアルコア CPU と 4GB のメモリは一般的な構成となっており、サーバ用途では、複数のデュアルコアやクアッドコア CPU とさらに多くのメモリを搭載したマシンが広く使われている。しかし、このような状況にも関わらず、既存の動的な索引構築手法は、そのような近代的なハードウェアにおいてスケールするように設計されていない。つまり、現在のデータベースシステムにおいて、スケーラビリティが最も重要な要素の一つであるにも関わらず、利用可能なコア数が増加しても索引の構築時間が減少しないという問題がある。

© 2009 Information Processing Society of Japan

Vol.2009-DBS-149 No.2
2009/11/20

本論文では、マルチコア CPU を効率的に利用したスケーラブルな動的索引構築手法を提案する。新しい手法の動作原理は、複数のコアにより複数のメモリ上の索引を並列に構築することにより、全体の構築時間を削減させるというものである。このような動的な索引構築においてマルチコアの利用に重点を置くのは、我々が知る限り初めての試みである。評価実験では、本提案手法による動的な索引構築により、検索速度の低下を伴わずに索引構築時間を大幅に削減できることを示す。

本論文の構成は次のとおりである。2 章では、動的な索引構築における既存の研究について述べる。3 章では、提案手法の実現のための新しい転置索引のデータ構造について、また、効率化のためのいくつかの最適化手法について述べる。4 章では、実験結果とその評価について、続く 5 章では、関連研究と本提案手法との関連について述べる。最後に、6 章では本論文のまとめと今後の展望について述べる。

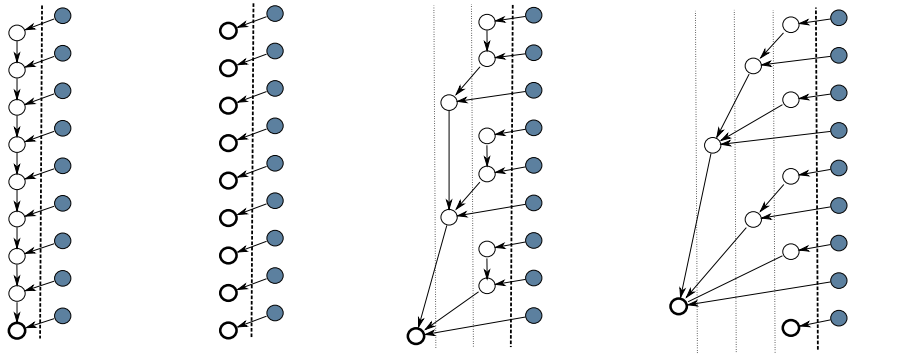
2. 技術的背景

2.1 転置索引の構築

転置索引は 2 つの主要な部分から構成される。一つは、文書集合中のすべての単語 (ターム) のリストである辞書と、もう一つは、単語ごとの転置リストである。各転置リストはポスティング (またはポインタ) の列であり、文書内の出現頻度や出現位置などの補足情報などと共に格納されている。[12] 辞書部分は B+木などの木構造で実現する場合が多く、転置リストに対しては様々な圧縮手法が開発されており[1),12),13),18),20)]、位置情報を含んだ索引でも元文章の 25%程度のサイズで保存することが可能となっている。また、転置リストの圧縮により、検索時間も削減することが知られている。

転置索引の標準的な形式では、各転置リストのポスティングは文書 ID 順で格納されていることが多い。本論文でもこの索引構成を前提に話を進める。その他の形式としては、単語の出現頻度順や単語との関連度順などが存在するが、本論文で紹介する動的な構築手法においては、これらの他の索引構成は適用していない。

検索処理では、各検索語に対してポスティングを検証することにより行われ、各文書が検索語とどの程度適合しているかを評価する類似度スコアを計算する。この処理を行うには、ストップワードを除いた文書中のすべての単語がインデックスされる必要がある。また、フレーズ検索は単語の出現位置を保持する場合のみ処理可能となる。この場合は、クエリ中の各フレーズを単語として扱い、フレーズを構成する単語の転置リストを結合することにより、フレーズの転置リストを構築する。また、フレーズ検索を効率的に処理するには、ストップワードもインデックスされる必要がある。



(a) Immediate Merge (b) No Merge (c) Geometric Partitioning(r=3) (d) Geometric Partitioning(r=2)

図 1 動的な索引構築における主要なマージ手法。色のついたノードはメモリ上で構築された部分索引を表す。また、太い線のノードは最終的な部分索引を表す。

転置リストはディスクの連続した領域に格納されることが多い。これは、一旦辞書が探索されたら、クエリ単語ごとに 1 回のシークと 1 回の連続ディスク読み取りが必要となることを意味している。

既存の索引に新しい文書を追加することは、文書が含む単語数分の（場合によっては何千という）新しい文書ポインタを転置リスト追加することを意味する。各リストの更新の度にディスク上でシークを行うことは、非常に高価であり、実際のシステムでは、一連の更新によりディスクが高負荷になりすぎてしまう。このため、動的な検索システムにおける転置索引は、2 つの索引に分けて格納されている。それは、メモリ上の索引とディスク上の索引からなり、メモリ上の索引は直近に追加された文書に対しての索引を提供し、ディスク上の索引は定期的にメモリ上の索引とマージされ、ディスクに書き戻される。この手法は効果的である。 なぜなら、ディスクベースのマージ処理はシーケンシャルに行うことができ、また、すべてのランダムアクセス処理はメモリ上で行われ、文書は追加されて間もなく検索可能となるからである。しかし、これにより、検索においてはメモリ上の転置リストとディスク上の転置リストを論理的に結合しなければいけないため、検索処理は多少複雑になる。

2.2 索引のマージ戦略

前章で述べたように、メモリ上の索引とディスク上の索引における様々なマージ手法が提

© 2009 Information Processing Society of Japan

案されている。以下では、主要なマージ手法について述べる。また、図 1 にそれぞれの手法による、索引の構築過程を示す。

2.2.1 Immediate Merge 戦略

最初のマージ戦略は、Lester ら[9] により提案された。この戦略では、1 つのディスク上の索引と 1 つのメモリ上の索引を管理する。メモリ上の索引のサイズがある閾値に達した場合、メモリ上のポスティングは既存のディスク上の索引とマージされ、新しい索引を作成する。このとき、古い索引は削除される。この戦略は、転置リストを取り出すのに必要なディスクシークの数を最小化しているが、マージの度に、すべての索引をスキャンする必要があるという欠点がある。従って、すべての文書をインデックスするのに必要なディスク操作は、文書サイズの 2 乗に比例してしまう。

2.2.2 No Merge 戦略

2 つ目の戦略では、マージ操作を全く行わない。メモリ上の索引がある閾値を超えた場合、ポスティングは並び換えられ、ディスクに書き出され、新しい部分索引としてディスクに書き出される。このように、ディスク上の索引は全くマージされない。与えられた単語の転置リストを取得したい場合は、部分リストをすべての部分索引から取得する必要がある。No Merge 戦略は、高いインデックス構築速度を達成するが、転置リストの取得に $O(n)$ 回のディスクシークが発生してしまう欠点がある。(n は部分索引の数)

2.2.3 Geometric Partitioning 戦略

上で述べた 2 つの戦略は、共に非常に極端な戦略である。3 つ目の戦略は、これらの中間的な手法となる。新しく作られたディスク上の部分索引は、毎回ではなく定期的に、既存の索引とマージされる。

Lester ら[10] は、索引を制限された数のパーティションに分割する手法を提案している。彼らは、キーとなるパラメータ r を導入する。(通常、r=2 または r=3 が選ばれる。) もし、メモリ上に b 個のポスティングを保持できるとしたら、レベル k(k=1,2,3...) のパーティションは $(r-1)r^{k-1}b$ 個以下のポスティングを保持する。それに加えて、レベル k のパーティションは、空または少なくとも $r^{k-1}b$ のポスティングが含まれている。新しい索引の作成が、レベル k のパーティションにおいて、上限の $(r-1)r^{k-1}b$ より多くのポスティングを含むような場合は、それらは新しいインデックスにマージされ、適切なパーティションに割り当てられる。この戦略は、Geometric Partitioning と呼ばれ、Lester ら[10] は、この戦略のコストの解析を行っている。また、Büttcher ら[3] は、Logarithmic Merge という同様の戦略を同時期に提案し、それは r=2 の場合の Geometric Partitioning と同じインデックス構築過程となる。[11]

Geometric Partitioning や Logarithmic Merge などは増え続ける文書集合に対して設計された戦略であり、Immediate Merge よりも高速な索引の構築が可能となる。これらの戦略の欠点は、各単語の転置リストがディスク上の複数の部分索引に分散されているため、検索処理のパフォーマンスが若干悪くなることである。

2.2.4 その他の戦略

インプレイスな更新は、新しいポスティングを既存の転置リストの最後に書き加えることにより、各ステージにおける索引への変更を最小に抑える戦略であり、この関連手法は広く研究されている。[2),5),8),14),15)] Büttcher ら[4] は、短い転置リストと長い転置リストを区別し、短いリストにはマージベースの戦略を、長いリストにはインプレイスでの更新を行う手法を提案している。また、Guo ら[6] は、索引の構築における部分索引のマージの順番を動的に調整する手法を提案した。その手法では、さらに文書の即時削除をサポートすることにより、検索処理において既存手法より良い性能を得ている。

3. メモリ上における索引構築処理の並列化

前章では、増え続ける文書集合に対する索引の更新戦略について述べてきた。Geometric Partitioning 戦略による索引構築処理はディスク I/O の観点で非常に効率的であり、現在の動的な環境においては、もっとも有効な選択肢の一つである。しかし、メモリ上での索引構築処理は、構文解析、圧縮処理、転置処理などを含み、非常に長い時間を要する処理である。また、近代的なハードウェア資源に対するスケーラビリティが考慮されていないという問題点も挙げられる。これはつまり、ハードウェア資源が増えても、索引構築処理のパフォーマンスは向上しないことを意味する。これらの状況において、我々はマルチコア CPU における Geometric Partitioning 戦略に着目し、メモリ上での索引の構築を並列に処理することが可能となるような新しいデータ構造を提案する。そして、パフォーマンス向上のためのいくつかの最適化手法と、提案するデータ構造における検索アルゴリズムの紹介をする。

3.1 メモリ上における複数の索引の構築

マルチコア CPU における索引構築処理のスケーラビリティを向上させるために、各コアによりメモリ上に複数の索引を構築する手法が考えられる。これにより、長い時間を要するメモリ上での構築処理を複数のコアに分散させることが可能となる。つまり、各メモリ上のプロセスが各々の索引を構築し、その構築が完了したタイミングでディスク上の既存の索引とマージ処理が行われる。図 2 に、メモリ上での複数の索引の並列構築処理とディスクへの

© 2009 Information Processing Society of Japan

マージの流れをタイムライン形式で示す。しかし、ナイーブな手法によるメモリ上での複数の索引の並列構築処理は、最終的に文書 ID 順に並んだ転置リストの構築が困難になるという問題が起きてしまう。本論文では、その問題の解決策を提案する。

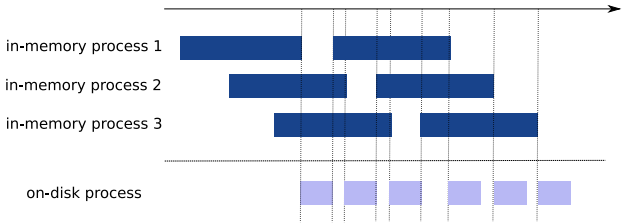


図 2 メモリ上での複数の索引の並列構築処理とディスクへのマージ

3.2 2 段階文書 ID による転置索引

上述の問題に対処するために、我々は文書 ID を 2 段階で管理する新しい転置索引のデータ構造を提案する。そのデータ構造では、文書 ID を大域的な連番 (Global Sequence Number: GSN) と局所的な文書 ID を用いることで、全体の文書 ID を管理する。メモリ上の索引に重複のない GSN を付与することにより、メモリ上でのプロセスは局所的な文書 ID で独立して処理することが可能となる。これにより、最終的に構築された転置リストは大域的には GSN 順に、局所的には局所文書 ID 順に並ぶようになる。この 2 段階の文書 ID を用いた転置リストを図 3 に示す。メモリ上での並列処理において一つ注意が必要なのは、マージ処理をプロセス間で競合しないように協調して動作させることである。これにより、転置リストが GSN の昇順に並ぶことが保証される。

この並列処理において、特定のシステムによるメモリ要求を満たすには、指定されたメモリサイズをプロセス数に分割することで対応する。例えば、メモリ上の索引のバッファサイズを 1GB とし 8 プロセスを並列で動作可能にするには、各プロセスに 128MB のメモリを割り当てるようにする。

3.3 アウトオブオーダによるマージ処理

メモリ上の索引に対してマージ処理の直前に GSN を付与することにより、マージ処理をアウトオブオーダで実行可能となる。つまり、メモリ上の索引間には順番は無くなり、メモリ上での索引構築処理の完了順に、(他のプロセスがマージ中でない場合は) マージ処理を待ち時間なく実行することが可能となる。これによりさらなるパフォーマンスの効率化が見込める。以下に、アウトオブオーダによるマージ処理を利用した転置索引の構築の基本ステップを示す。

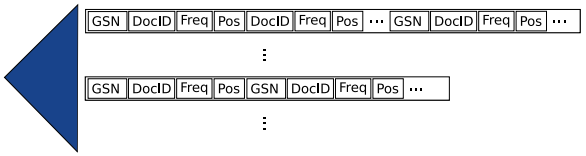


図 3 2 段階文書 ID による転置索引

(1) メモリ上の各プロセスに対して
  (a) 新しい文書のポスティングをメモリ上の索引に追加
  (b) メモリ上の索引が閾値に達したら
    (i) メモリ上の索引に GSN を割り当てる
    (ii) メモリ上の索引をディスク上の索引とマージする (もし他のプロセスがマージ中の場合は、その完了を待つ)
  (c) マージ処理完了後、ステップ 1a に戻る

3.4 遅延マージ

上で述べたように、各プロセスはメモリ上の索引用に分割されたバッファを保持している。それぞれのサイズは、単独でプロセスを動作させた場合のバッファサイズと比べて小さくなり、大量のディスク I/O を伴うマージが頻繁に発生するという問題が起こる。この問題に対応するために、遅延マージと名付けた最適化手法を提案する。この手法では、まず最初にキューを用意する。あるプロセスはメモリ上の索引の構築が完了したら、キューにその索引を追加し、そのプロセスは新しいバッファで別の索引の構築を開始する。そして、キューが満たされると、キュー上の索引と既存のディスク上の索引をマルチウェイでマージし、ディスク上に新たな索引を作成する。図 4 は遅延マージの概念図を示す。(実際の実装では、不必要なメモリのコピーや未使用バッファ領域の消費などは行われていない。) 遅延マージは、広く知られたダブル・バッファリング手法に少し類似している。しかし、我々の手法は I/O 処理を遅らせ、マージ処理を後で一気に行う。そして、バッファとメモリ上のプロセスは独立しており、バッファプールに空きバッファがある限り、メモリ上のプロセスは索引の構築を開始できるという点で異なる。

© 2009 Information Processing Society of Japan

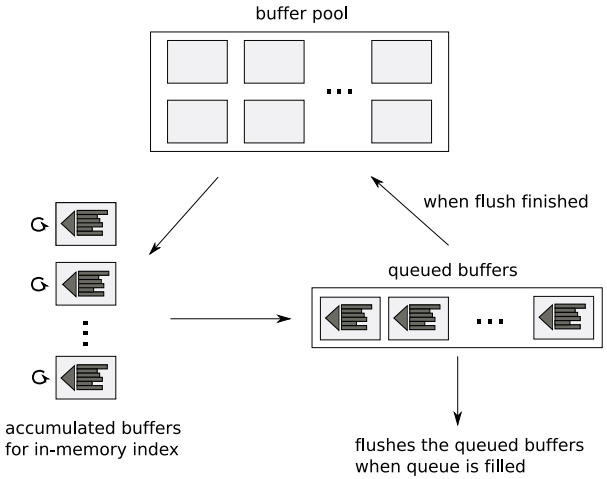


図 4 遅延マージの概念図。キューに溜められた複数のバッファはキューの空きが無くなった場合マルチウェイでマージされ新しいインデックスを作成する。

システムのメモリ要求を満たすには、メモリ領域を使用するバッファ数に分割する必要がある。それにより、この手法は余分なメモリを使用せずに効率的に動作することが可能となる。我々の実験から、n 個のメモリプロセスが存在する場合、索引用のバッファを $2n$ 個に分割し、キューのサイズを n にすることにより、最良のパフォーマンスが得られた。以下に、遅延マージとアウトオブオーダのマージによる索引構築のステップを詳細に述べる。

(1) メモリ上の索引用のキューを初期化する
(2) 各メモリ上のプロセスにおいて
  (a) 新しい文書のポスティングをメモリ上の索引に追加
  (b) メモリ上の索引が閾値に達した場合、その索引に GSN を付与し、キューに追加
    (i) キューに空きがあるならば、ステップ 2a に戻る
    (ii) キューに空が無い場合は、次のステップへ
  (c) キューにある索引と既存のディスク上の索引をマルティウェイでマージし、新しい索引をディスク上に書き出す
  (d) マージが完了した後、キューは溜められたバッファを解放し、新しいバッファが追加されるのを待つ。そして、ステップ 2a に戻る

3.5 検索処理における Intersection 処理

ブーリアン検索やフレーズ検索で使われる Intersection 処理は、提案手法により構築された 2 段階文書 ID をもつ転置索引では若干修正が必要となる。アルゴリズム 1 は 2 つの転置リストに対する、ブーリアン検索における Intersection アルゴリズムの疑似コードを示す。

主な違いは、文書 ID の同一判定の前に GSN の同一判定を行うことである。(フレーズ検索では、GSN と局所文書 ID が同一である場合は、さらに連接判定を行う必要がある。)

Algorithm 1 INTERSECT$(p1, p2)$

```
 1: answer ←<>
 2: while p1 ≠ NIL and p2 ≠ NIL do
 3:   if GSN(p1) = GSN(p2) then
 4:     ADD(answer, GSN(p1))
 5:     while hasDoc(p1) and hasDoc(p2) do
 6:       if docID(p1) = docID(p2) then
 7:         ADD(answer, docID(p1))
 8:         p1 ← nextDoc(p1)
 9:         p2 ← nextDoc(p2)
10:       else
11:         if docID(p1) < docID(p2) then
12:           p1 ← nextDoc(p1)
13:         else
14:           p2 ← nextDoc(p2)
15:         end if
16:       end if
17:     end while
18:     p1 ← nextGSN(p1)
19:     p2 ← nextGSN(p2)
20:   else
21:     if GSN(p1) < GSN(p2) then
22:       p1 ← nextGSN(p1)
23:     else
24:       p2 ← nextGSN(p2)
25:     end if
26:   end if
27: end while
```


© 2009 Information Processing Society of Japan

Vol.2009-DBS-149 No.2
2009/11/20

4. 実　　験

クローラにより収集した 30GB、2270 万文書のウェブ上の文書集合に対して実験を行い、索引の構築速度と検索速度を測定した。すべての実験は図 1 に示す環境で行い、また使用するコア数の調整は OS のパラメータ'/sys/devices/system/cpu/cpu[*CPU ID*]/online' を設定することにより行った。本論文では、転置リストの構築手法は Geometric Partitioning 戦略 (r=2) を対象とし、実験で構築した転置リストは単語の出現頻度、位置情報を各文書 ID ごとに含み、文書 ID、位置情報、GSN は前の値との差分値を保存する。転置リストは多くの検索エンジンで使われている最も普及した効率的な Variable-byte 符号[12),20)] により圧縮している。索引構築時間は、構文解析、圧縮処理、メモリ上での転置処理、リストのマージ処理を含む。また、検索時間は辞書の検索、ディスクからのリストの取得、そして Intersection 処理を含むが、検索結果のスコアリングや並び換え、元文章の取得やスニペットの作成時間は含まない。(それらはこの提案するデータ構造に依存しない処理のため。)

CPU	dual-processor Quad-Core Xeon 5345
メモリ	8GB
ディスク	SATA 7200 rpm
OS	openSUSE 11.0 (Kernel 2.6.25)

表 1　実験環境

システムは C++で実装し、メモリ上の索引は STL の std::map を、ディスク上の索引は Berkeley DB と類似した Lux IO[⋆1] という B+-木のデータベースを利用した。Lux IO は Key-Value 型のデータベースであり、Value に長いリストを保存する用途に特化しているため、本実験では Key に辞書を、Value に転置リストを格納している。並列のメモリ上のプロセスは POSIX スレッドにより実装し、本実験ではスレッド数を使用可能なコア数の数と同じにした。本システムの入力となる文書集合は文書キューに保存され、各メモリプロセスが一文書ずつキューから取り出すことで動作する。そして、前章でも述べた通り、メモリ上の構築された索引はディスク上の索引とマージされ、新しい索引が作成される。本システムの全体像を図 5 に示す。

本システムにおけるバッファは、辞書と転置リストのデータ部分のみを対象としている。

⋆1 http://luxio.sourceforge.net/ - Yet Another Fast Database Manager.

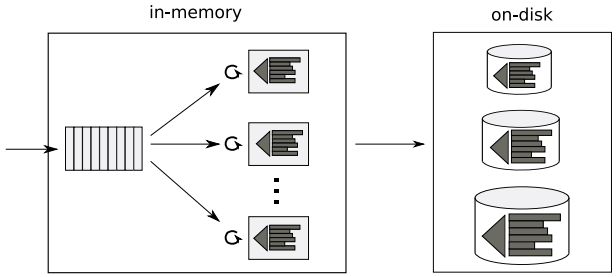


図 5　システムの全体像

よって、ライブラリによって管理されている木構造や構造体などのライブラリ依存のデータはバッファの消費としてはカウントしない。つまり、1GB のバッファによる索引の構築においては、1GB よりも多くのメモリが使われる。

図 6 に、バッファを 1GB に指定し、アウトオブオーダによるマージを有効にした状態での、各コア数における提案手法による索引の構築時間を示す。図には 3 つの線があり、1 番上部の線は既存手法による構築時間、下部点線がアウトオブマージを有効にした提案手法による構築時間、そして下部実線がさらに遅延マージを有効にした提案手法による構築時間となる。これらの結果から、既存手法は 8 コアが利用可能な状態でも構築時間に全く影響が無い反面、提案手法は利用可能なコア数が増えるに従い索引構築時間を劇的に削減できていることが確認できる。また、アウトオブオーダによるマージのみが有効である場合に提案手法による効率性が徐々に劣化しているが、遅延マージによりこの劣化は緩和され、期待通りのパフォーマンスの向上が確認できた。この劣化は、メモリでの構築処理を終えたスレッドが他のスレッドによるマージ処理を待ってしまうことが原因であると考えられる。これにより、マージに伴うディスク I/O を減らす遅延マージは、コア数が増えるに従い効果的な手法であると結論付けられる。

図 7 は、2 段階文書 ID による転置索引と通常の転置索引における 2 単語のフレーズ検索による検索時間を示す。提案手法により構築された転置索引は、8 スレッドにより構築されたものとなっている。図に示されている値は、ディスク上のすべての索引への検索時間の合計であるが、それは検索時の索引数に依存するため、10GB, 20GB, 30GB それぞれをインデックスした時の検索時間を示している。この実験では、メモリ上の索引に対しての検索時間がディスク上のそれに対する検索時間に比べ非常に小さいため、結果には含めない。また、30GB の文書集合における出現頻度に応じて、高頻出、中頻出、低頻出の 3 つの種類のクエ

© 2009 Information Processing Society of Japan

リでの検索を行った。これは、高頻出の単語の転置リストが、文書 ID 数に比べて比較的少ない数の GSN を含むのに対し、低頻出単語の転置リストは GSN をより多く含み、結果として転置リストのサイズが大きくなる可能性があるからである。しかし、結果として、2 段階文書 ID による転置索引において検索時間は増加しないことが確認できた。また、本提案手法により作成された転置索引のサイズは通常のものとほぼ同じになることが確認できた (表 2)。これは、GSN の間に存在する文書 ID が通常の文書 ID よりも小さくなるため、通常小さい値に対して小さく圧縮する符号化がうまく働いたことが原因であると考えられる。

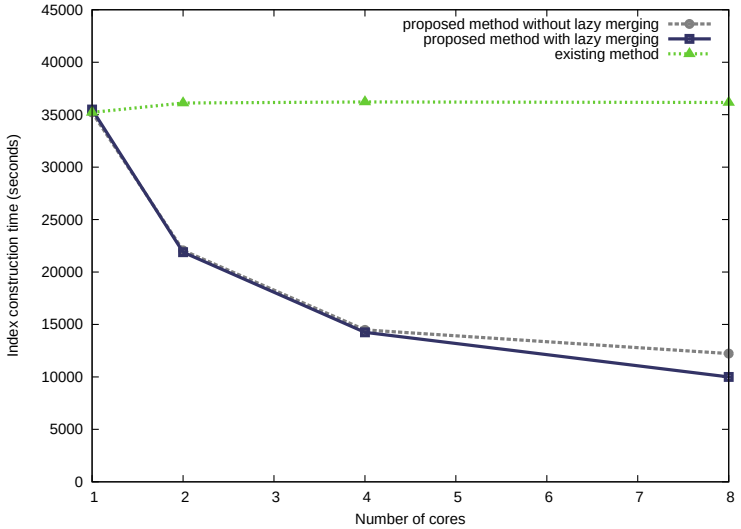


図 6 30GB の文書集合に対する、各コア数における索引構築時間

5. 関連研究と議論

本論文では、マルチコア CPU と 1 つのディスクに対する動的な索引構築手法に重点を置いた。今後の課題として、本提案手法を複数ディスクに対して適用し、さらにスケールする動的な索引構築を目指すことが挙げられる。

マルチコアやマルチ CPU と複数ディスクを利用することは、並列情報検索 (Parallel Information Retrieval)[7),16),19)] や、また、仮想化技術の利用した複数ホストによる分散情報検索 (Distributed Infromation Retrieval) とも考えることができる。つまり、マルチコアと複数ディスクのマシンは、複数の処理装置やホストと考えることができ、スケーラビリティを確保するために、文書集合をいくつかの独立な部分集合に分ける文書分割などの索引分散手法が適用できる。

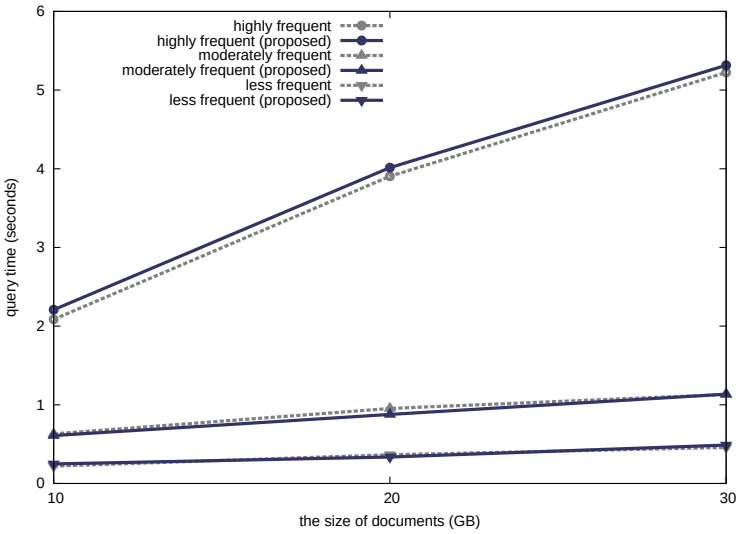


図 7 各文書集合のサイズに対する 2 単語のフレーズ検索による検索時間

	2 段階文書 ID による転置索引のサイズ (a)	通常の転置索引サイズ (b)	増加率 (a/b)
高頻出単語	184307990 (bytes)	184306856 (bytes)	0.00062 (%)
中頻出単語	34019764 (bytes)	34018979 (bytes)	0.0023 (%)
低頻出単語	13444111 (bytes)	13441946 (bytes)	0.016 (%)

表 2 提案手法のおける転置索引のサイズと通常の転置索引のサイズの比較

文書分割により、転置索引は各リソース上で独立して構築することが可能となる。しかし、独立した動的な索引の構築は、CPU 数 (コア数) に対して索引数が増加してしまう。例えば、クアッドコア (4 コア)CPU と 4 つのディスクを持つマシンがあり、100GB のデータに対して 1GB のバッファで索引を構築する場合を考えてみる。上で述べた通り、上記のマ

© 2009 Information Processing Society of Japan

シンは 1 コアと 1 ディスクを持つ 4 つの論理的に独立した処理装置として扱うことができる。この状況での Geometric Partitioning 戦略による索引の構築は、各処理装置で 25GB のデータに対して 256MB のバッファで並列に処理され、索引構築時間は 1 コアで行った場合と比べて 1/4 倍となる。しかし、構築される索引数は 4 倍 (4 * log(25GB/256MB) = 28) となり、索引構築時間の削減と引き替えに、検索時間の大幅な劣化が起こる。本研究は、この問題に対する一つの解決策となる。今後は、本提案手法を複数ディスクに適用することにより、検索速度を劣化させることなく、よりスケーラブルな索引の構築を目指す。つまり、1 コアで構築する場合の索引数を上限としつつ、マルチコアと複数ディスクによる動的な索引構築を行うことにより、検索時間を劣化させずに索引構築時間を削減することが目標となる。

6. ま と め

本論文では、マルチコア CPU を利用した新しい動的な索引構築手法を提案した。本提案手法は、スケーラブルな動的索引構築のためにメモリ上の複数の索引を並列に構築するという原理に基づいている。評価実験ではいくつかの最適化手法を合わせた手法により、索引構築速度を 8 コアで 4 倍近くにすることができた。また、それに伴う検索速度の低下がほぼゼロになることを確認した。

今後の課題として、本提案手法のさらなる最適化と、複数ディスクへの適用によるさらなるスケーラビリティの確保を進めていきたい。

参 考 文 献

1) V. N. Anh and A. Mat.: Inverted index compression using wordaligned binary codes, Information Retrieval. pp.151-166 (2005).
2) A. Biliris. The performance of three database storage structures for managing large objects, Proc. SIGMOD 1992, pp.276—285 (1992)
3) S. Büttcher and C. L. A. Clarke.: Indexing time vs. query time trade-offs in dynamic information retrieval systems, Proc. CIKM 2005, pp.317-318 (2005)
4) S. Büttcher, C. L. A. Clarke and Brad Lushman.: Hybrid Index Maintenance for Growing Text Collections, Proc. SIGIR 2006, pp.356-363 (2006)
5) D. R. Cutting and J. O. Pedersen.: Optimizations for dynamic inverted index maintenance, Proc. SIGIR 1990, pp.405—411 (1990)
6) R. Guo, X. Cheng, H. Xu and B. Wang.: Efficient On-line Index Maintenance for Dynamic Text Collections by Using Dynamic Balancing Tree, Proc. CIKM 2007, pp.751-759 (2007)
7) B. Jeong and E. Omiecinski.: Inverted File Partitioning Schemes in Multiple Disk Systems, IEEE Transactions on Parallel and Distributed Systems (TPDS), Vol 6, pp.142-153 (1995)
8) T. J. Lehman and B. G. Lindsay.: The Starburst long field manager, Proc. VLDB 1989, pp.375—383 (1989)
9) N. Lester, J. Zobel, and H. E. Williams.: In-Place versus Re-Build versus Re-Merge: Index Maintenance Strategies for Text Retrieval Systems, Proc of the 27th Conference on Australasian Computer Science (ACSC), pp.15-23 (2004)
10) N. Lester, A. Moffat, and J. Zobel.: Fast on-line index construction by geometric partitioning, Proc. CIKM 2005, pp.776-783 (2005)
11) N. Lester, A. Moffat and J. Zobel.: Efficient online index construction for text databases, ACM Transactions on Database Systems (TODS), Vol 33, No.19, (2008)
12) C. D. Manning, P. Raghavan and H Schütze.: Introduction to Information Retrieval, Cambridge University Press. (2008)
13) F. Scholer, H. E. Williams, J. Yiannis, and J. Zobel.: Compression of inverted indexes for fast query evaluation, Proc. SIGIR 2002, pp.222—229 (2002)
14) W. Y. Shieh and C. P. Chung.: A statistics-based approach to incrementally update inverted files, Proc. Int. Conf. on Information and Knowledge Engineering, pp.38—43 (2003)
15) K. Shoens, A. Tomasic, and H. Garcia-Molina.: Synthetic workload performance analysis of incremental updates, Proc. SIGIR 1994, pp.329—338 (1994)
16) C. Stanfill.: Partitioned posting files: a parallel inverted file structure for information retrieval, Proc. SIGIR 1989, pp413-428 (1989)
17) A. Tomasic, H. Garcia-Molina, and K. Shoens.: Incremental updates of inverted lists for text document retrieval, Proc. SIGMOD 1994, pp.289—300 (1994)
18) I. H.Witten, A. Moffat, and T. C. Bell.: Managing Gigabytes: Compressing and Indexing Documents and Images. Morgan Kaufmann, San Francisco, California, second edition (1999)
19) B. Yates and R. Neto.: Modern Infromation Retrieval, Addison Wesley (1999)
20) J. Zobel and A. Moffat.: Inverted files for text search engines, ACM Computing Surveys, Vol.38(2) (2006)

© 2009 Information Processing Society of Japan