データベースシステム　125－72
（２００１．７．１９）

インクリメンタルな更新機構を持つ全文検索インデックスの分散並列処理方式

高橋 慎　　吉原 潤　　加藤 和彦
筑波大学大学院　筑波大学大学院　筑波大学電子・情報工学系
理工学研究科　理工学研究科
(2001 年 3 月まで)
E-mail : {makoto, yosiwara, kato}@osss.is.tsukuba.ac.jp

概 要

suffix array はテキストの接尾辞のポインタを辞書順に並べかえたもので，任意の部分文字列を高速に検索できるが，静的なデータ構造のため，更新のオーバーヘッドが大きい．我々は以前，インクリメンタルな更新方式を提案したが，この方式が残す問題の一つは，差分情報を用いて作成した suffix array を一つにまとめる再構成処理のオーバーヘッドが大きいことである．本論文では suffix array を分散配置することで suffix array のサイズを小さくし，再構成処理の高速化を図る分散並列処理方式について述べる．実装を用いた実験結果により，再構成処理の高速化と検索時の性能の向上についての評価を行なう．

Distributed Parallel Processing Scheme of a Full-Text Index Structure with Incremental Updating

Makoto Takahashi　　Jun Yoshiwara　　Kazuhiko Kato
Master's Program in Science and Engineering, University of Tsukuba　　Master's Program in Science and Engineering, University of Tsukuba　　Institute of Information Science and Electronics, University of Tsukuba
(Until 2001 March)
E-mail : {makoto, yosiwara, kato}@osss.is.tsukuba.ac.jp

Abstract

Suffix array is a full-text index structure efficient to retrieve any substring of the indexed text, but requires significant overheads to update. Previously we proposed an incremental updating scheme for suffix arrays. One of the remaining problems is the overheads to reconstruct large suffix arrays. Frequency of the reconstruction operation is reduced in the incremental updating scheme, but requires considerable overheads. This paper presents a scheme to incorporate parallel and distributed processing into the incremental updating scheme. In the scheme, decomposed suffix arrays are distributed to several machines, so that the reconstruction overheads are reduced and throughput for the retrieval operations is increased. We show some experimental results performed to evaluate the proposed scheme.

1 はじめに

大規模テキストに対する文字列の検索は，あらかじめ切り出された単語を検索対象とする方法と，任意の部分文字列を対象として検索する方法とに大別される．suffix array [7] はテキストの接尾辞のポインタを接尾辞の辞書順に並べたもので，任意の部分文字列検索を高速に行うことができる．また，単語の切り出しが不要なため，日本語のように単語の切り出しが難しい言語の場合でも検索洩れがない．しかし suffix array は静的なデータ構造のため、更新のオーバーヘッドが大きく，頻繁に情報が更新される場合に不向きである．

この問題を解決するため，我々は suffix array のインクリメンタルな更新方式 [9] を提案した．この方式では，単一の巨大な suffix array を更新するのではなく，差分のテキストをもとに差分の suffix array を作るため更新が高速である．しかし更新を続け，数の増えた差分 suffix array から単一の suffix array を再構成する処理のコストが大きいという問題がある．

再構成処理の中で最もコストのかかる suffix array のマージ処理は，処理する suffix array の大きさに依存しているため，再構成処理を高速化する方法には扱う suffix array のサイズを小さくすることが考えられる．本研究では再構成処理を高速化して効率的な更新処理を行なうために，suffix array を複数のマシンに分散配置し，処理を分散並列化する方式を提案する．この方式の検証のため，頻繁なデータの更新を必要とする WWW 検索エンジンを提案方式を用いて実装し，更新処理の効率と同時に検索処理の効率を測る実験を行なう．

2 基本概念

2.1 Suffix array[7]

長さ n のテキスト $T[1,n]$ の任意の位置 $i(=1,\ldots,n)$ から文末までの文字列 $T[i,n]$ を接尾辞 s_i という．接尾辞 s_i を辞書順に並べ替え，それを s_i へのポインタ i で置き換えた配列 A をテキスト T に対する suffix array という (図 1)．A を二分探索することで T に現われる任意の部分文字列の検索ができる．

複数のテキストに対する suffix array を作るには，個々のテキスト (要素テキスト) を連結して一つのテキスト (連結テキスト) を作り，それに対して suffix array を作成する．

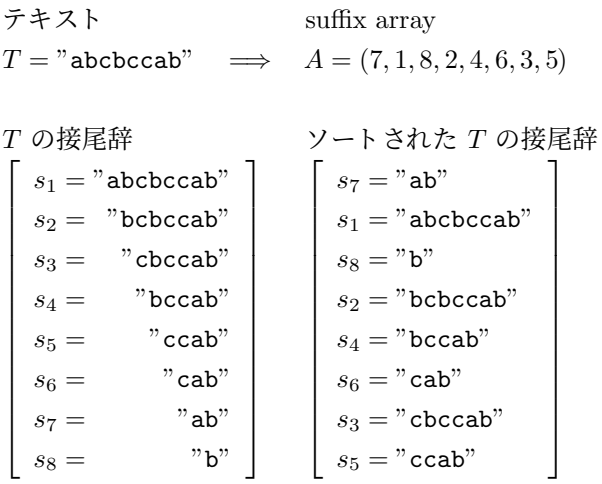

図 1: suffix array の例

2.2 インクリメンタルな更新方式 [9]

この方式では，最初のテキストの集合を連結して一つの文字列とし，これに対し suffix array を作成する．これを主インデックスとする．更新が行なわれてテキストの追加を行なう場合には，二つの処理方法がある．一つ目の方法では，差分の追加テキストから新たな suffix array を作る．これを差分インデックスと呼ぶ．もう一つの方法では，最新の差分インデックスに追加分のテキストをマージする．この二つの操作をどのような順序で行なうかは，最新の差分インデックスに差分情報をマージする回数の最大値や作成される差分インデックスの個数の最大値などのパラメータを与えることで決定される．更新処理の様子を図 2 に示す．

主インデックスが十分大きな量のテキストに対して作成された場合，保持している全テキストの量に対し，更新される差分のテキストの量は十分少ないため，主インデックス全体を更新するより別個に差分インデックスを作成する方がコストが小さい．また，差分インデックスは主インデックスより小さいので，最新の差分インデックスに差分情報をマージするコストも主インデックスに直接マージする場合より小さい．

テキストを削除する場合には，削除されたという印だけをつけ，実際の削除は後でまとめて行なう．こ

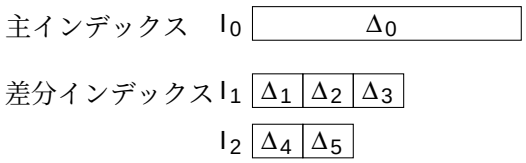

図 2: 更新処理

の場合，一つのテキストに対して古い情報と新しい情報の両方が存在することになるので，最新の情報のみを取り出せるようにする必要がある．そこで，テキストの最新の内容がどの suffix array に含まれているかを識別するための有効インデックス番号を記録しておくことにし，最新の情報を参照できるようにする．

検索処理を行なう場合は，更新によって作成された複数の差分インデックス全てに対して検索をし，その結果をマージすることで結果を得る．ここで有効インデックス番号を用いることで，検索結果の中から古くなって無効な情報を取り除くことができる．

更新を続けていくと，差分インデックスの個数とともに検索対象の suffix array の数が増えていくので，検索コストが増大していく．そこで差分インデックスの個数がある値を越えたら，再構成処理を行ない全体をマージして単一の suffix array を作成する．

3 提案分散並列処理方式

本章では，インクリメンタルな更新方式を分散並列化する方式について説明する．この方式では処理を分散並列化するために複数のマシンを用いる．役割によってそれぞれマスターノード とスレーブノード と呼ぶ．マスターノード は更新処理で追加された差分の情報に対して suffix array を作成し，スレーブノード の数に合わせて分割してスレーブノードに送信する．スレーブノード はマスターノード から受け取った suffix array などの差分情報を更新する．この時の更新の方法は 2.2 章で述べたインクリメンタルな更新方式を用いる．

3.1 Suffix array の分散配置法

suffix array は辞書順に並んだ接尾辞へのポインタの配列であるため，途中で切断することが可能である．suffix array をある文字列の範囲で分割し，複数のノードに分散して配置することで更新処理を並列に行うことができる．また，一つの検索要求はその先頭の文字列から担当しているスレーブノード が一つに決まるため，別のスレーブノード が担当する検索要求を並列に処理することができる．しかし suffix array を文字列の範囲で分割する場合，各スレーブノード では担当範囲の単語が含まれるテキストは複数存在するため，全てのテキスト情報が必要となる．マスターノード は，suffix array を分割した境界の位置にある接尾辞を記録しておく．これを分割文字列 d と呼び，分割文字列によって各スレーブノードの担当している区間を識別する．

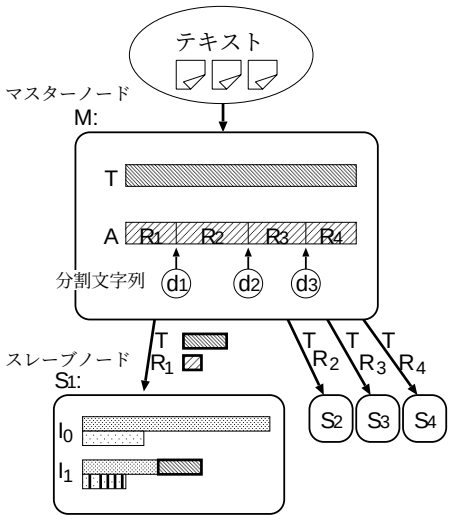

図 3: suffix array の更新処理の分散並列化

3.2 更新処理

図 3 は更新処理の様子を示したものである．テキストの追加を行なう場合，マスターノード は差分のテキストを受け取り一つのテキストに連結し，連結されたテキストに対する suffix array を作成する．次にマスターノード は，分割文字列 $d_1, d_2, ...$ を用いて各スレーブノード の担当区間を識別し，作成した suffix array を各スレーブノード の担当区間ごとに分割する．その後，分割した suffix array と追加された差分情報のテキスト全体を各スレーブノードに並列に送信する．同時に，マスターノード はあらかじめ決められたパラメータによって新規の差分インデックスを作成するか，既存の差分インデックス

にマージするかを判断してスレーブノード に命令を出す．

スレーブノード では，受け取った suffix array とテキストをインクリメンタルな更新方式を用い，マスターノード によって指定された方法で更新処理する．

テキストの削除を行なう場合もインクリメンタルな更新処理と同様に処理を行なう．マスターノードは削除するテキストのリストを各スレーブノード に送信し，スレーブノード はそれぞれテキストが削除されたという情報のみを保存しておく．実際の削除は，後の再構成処理と同時に行なわれる．

3.3 検索処理

検索要求はすべてマスターノード が受け付ける．マスターノード は分割文字列を用いて検索のキーワードがどのスレーブノード の担当している区間にあるかを調べ，該当するスレーブノード に検索の要求を出す．スレーブノード が実際の検索処理を行ない，その結果をマスターノード が受け取ってユーザに返す．

suffix array はある文字列の範囲で分割されているので，ひとつの検索要求には必ずただ一つのスレーブノード が対応するようになっている．そのため担当範囲の異なる検索要求は，一つのスレーブノードの検索結果を待たずに別のスレーブノード で並列に検索することができる．

検索にはマスターノード とスレーブノード の間の通信のオーバーヘッドがかかると予想されるが，検索を並列に実行することにより検索処理のスループットを向上させることが可能である．また，通信のオーバーヘッドを減少させるために，マスターノード に検索結果のキャッシュの機能を持たせる．キャッシュには検索要求をキーとして，その検索要求に対する URL や HTML ファイルの情報等の検索結果を格納する．キャッシュの容量は限られているので，容量を越えるときは，古い情報から捨てていく．マスターノード は検索要求を受け付けたら，最初にキャッシュに結果があるかどうか検索を行ない，ヒットすればそれを結果として返す．キャッシュに結果がない場合には，スレーブノード に検索要求を出す．検索要求に局所性がある場合にはキャッシュが有効に働くと期待できる．

4 実装と実験

本章では提案した分散並列処理方式の実装と，その有効性を検証するために行なった実験について述べる．

実装は Sun Solaris2.6 上で行なった．コードの量は約 28,000 行で，C および C++言語を用いた．マスターノード のキャッシュは，ハッシュおよび LRU アルゴリズムを用いて主記憶上に実装した．マスターノード とスレーブノード は Solaris thread を用いてマルチスレッド化した．また，スレッドプールを作成し，スレッドは実行直後に全て作成してスレッドプールで検索要求が到着するのを待機する．スレッドは検索要求が到着すると検索処理を行い，スレッドプールに戻って次の検索要求を待つ．検索要求の度にスレッドの生成消滅を行なわずに再利用できるので，スレッドの生成時のオーバーヘッドを抑えられる．マスターノード とスレーブノード の間の通信は TCP/IP を用いて実装した．

使用したマシンは Sun Netra t1(プロセッサ 440MHz UltraSPARC-IIi，メモリ 512MByte) で，各マシン間は 100Mbps のイーサネットで接続されている．使用したデータは実際の Web サーバに置いてある is.tsukuba.ac.jp ドメイン以下の HTML ファイルのうち，トップページから辿れたもの全てとした．

実験の初期状態として主インデックスと一つの差分インデックスがある状態を作った．主インデックスは Web サーバにある HTML ファイルから合計 256.0 メガバイト (31,014 ファイル，1 ファイル平均 8.75 キロバイト) を用いて構成し，次に差分インデックスは，主インデックスの中に含まれている HTML ファイルのうち内容が更新されて新しくなっていた 10,000 件の情報を反映させて作成した．

4.1 既存のインデックスにマージする更新処理

インクリメンタルな更新方式において，追加する差分の情報を最新の差分インデックスに直接更新する実験を行なった．マスターノード は 1 台とし，スレーブノード の数は 0, 2, 4, 6, 8 と変化させた．スレーブノード 数が 0 は，スレーブノード を用いずにマスターノード が全てのデータとインデックスを持ち，スレーブノード の代わりに更新処理を行なう場合のことである．実験はそれぞれ 10 回行って要し

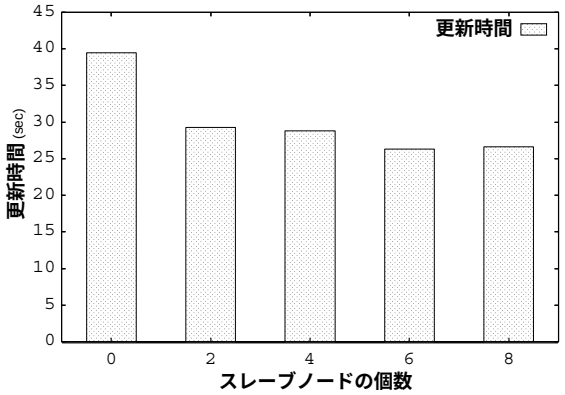

図 4: 既存のインデックスに直接更新する場合

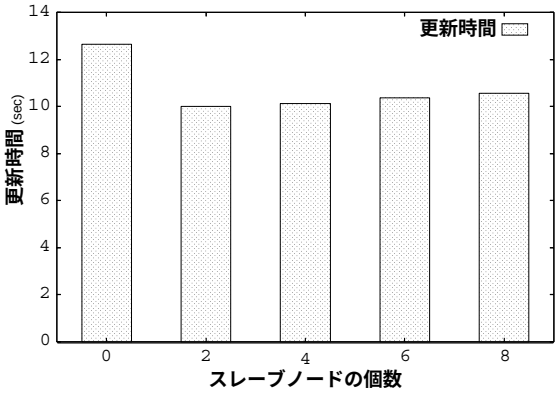

図 5: 新しく差分インデックスを作成する場合

た時間の平均を計算した．処理内容はデータの削除 100 件，すでに持っているデータの更新 100 件，新しいデータの追加 100 件で行なった．図 4 にその実験の結果を示す．

更新時間とは，マスターノード が差分情報を用いて連結テキスト および suffix array ，その他付属情報を作成するところから始まり，スレーブノード に suffix array など各情報を送信し，一番遅いスレーブノード の更新処理が終了したことをマスターノードが知るまでの時間である．

スレーブノード を用いた場合は，スレーブノードを用いずにマスターノード のみで更新した場合より更新時間が短縮されている．また，スレーブノード数が増えるに従って，更新時間は減少している．これは直接更新を行う場合，suffix array のマージ処理が更新処理の大部分を占めるため，suffix array を分割して分散配置することで，suffix array のサイズを小さくし更新処理のオーバーヘッドを減少させたためである．

4.2 新しく差分インデックスを作成する更新処理

インクリメンタルな更新方式を用いて差分情報から新しい差分インデックスを作成する場合の実験を行った．使用したデータ，実験方法や回数，それぞれ既存のインデックスにマージする場合の実験と同様にして実験した．図 5 にその実験の結果を示す．

スレーブノード を用いたどの場合も，スレーブノード を用いずにマスターノード のみで更新した場合より更新時間が短縮されている．しかしスレーブノード の個数が増えるにつれ，更新時間は増加している．これは，差分インデックスを作成する場合は suffix array のマージが行われず更新のオーバーヘッドが大きく減少されるので，時間を多く費す処理がマスターノード とスレーブノード の間の通信などになり，スレーブノード の数の増加に伴って通信のオーバーヘッドが増加したためである．suffix array 以外の情報は各スレーブノード に同じものが送信されるので，suffix array のマージ処理を行わないインクリメンタルな更新方式はスレーブノード を用いて分散並列化してもあまり処理時間に変化が現れなかった．

4.3 再構成処理

インクリメンタルな更新方式を分散化することで，再構成処理の時間がどのように変化するかを調べる．使用したマシン，データは更新の実験と同様である．マスターノード の数は 1 台，スレーブノード 数は更新の時と同様に 0，2，4，6，8 と変化させた．再構成の実験では，初期状態からさらに 1,000 件のファイルの内容を更新する処理を行なって，主インデックス 1 つ，差分インデックス 2 つの計インデックス 3 つにした状態から実験を始めた．実験は各スレーブノード 数の場合で 4 回ずつ行い，平均時間を計算した．図 6 にその実験の結果を示す．

スレーブノード を用いずマスターノード のみで更新した場合には多くの処理時間がかかっている．スレーブノード 数が増えていくにつれ，処理時間は 1/2.08，1/3.75，1/4.76，1/6.12 となり，スレーブノード の数の増加に伴って大幅に減少していくこと

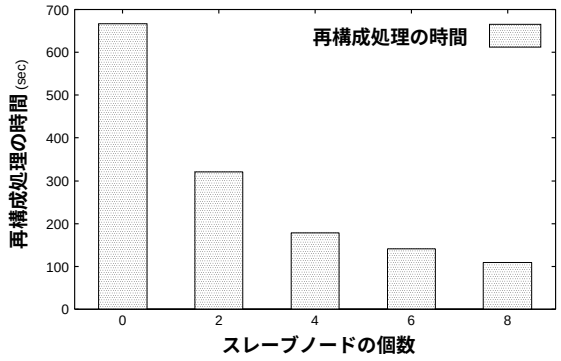

図 6: 再構成処理にかかる時間

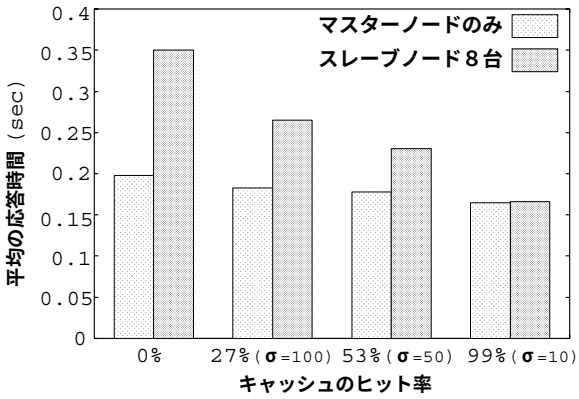

図 7: 1 件の検索処理に対する平均の応答時間

が分かった．

再構成処理はその処理の大部分を suffix array のマージ処理が占めている．よって suffix array を分割して分散配置する利点が大きく現れるため，m 個のスレーブノード を用いた場合，処理時間が 1/m となることが理想である．実際，全体の処理時間が長いスレーブノード 数 0 と 2 の場合は処理速度が 1/2 となっている．スレーブノード 数が増え，全体の処理時間が短くなると，suffix array のマージ処理以外の処理時間も無視できなくなってくるため，使用したスレーブノード の台数に見合う台数効果が得られなくなっていく．

4.4 検索処理の応答時間

分散並列方式でマスターノード とスレーブノードの間の通信のオーバーヘッドが検索処理にどのような影響を与えるかを検証するため，1 件の検索要求に対する平均の応答時間を測る実験を行なった．マシンは更新の実験と同じく Sun Netra t1 を用いた．マスターノード が 1 台，スレーブノード は 8 台，さらに検索要求を出すクライアントマシンを 1 台用いた．検索要求は毎日新聞の記事から単語を切り出し，その中で検索結果が 1 件以上必ず返ってくるキーワードだけを用いた．

1 台のクライアントマシンを用いてマスターノード に検索要求を 10,000 件出し，1 件の検索要求にかかる平均の応答時間を求めた．検索要求は 1 つの答えを受け取ってから次の検索要求を出すこととする．その際 3.3 節で述べたマスターノード とスレーブノード の通信オーバーヘッドを減少させるためのキャッシュの効果を測定するため，検索要求に局所性を持たせない場合と局所性を持たせる場合を実験した．局所性を持たせる場合は，検索要求の分布が正規分布となるようにし，偏差 σ を 100, 50, 10 と変化させた．それぞれの実験は 3 回ずつ行ない平均を計算した．図 7 に実験の結果を示す．

検索要求に局所性を持たせず，キャッシュのヒット率が 0%の時は，スレーブノード を用いた場合の方がマスターノード のみの場合より約 1.75 倍の時間がかかった．これは通信のオーバーヘッドが原因である．検索要求に局所性を持たせた場合は，キャッシュのヒット率があがるにつれ，応答時間の差は小さくなり，偏差 σ を 10 としてキャッシュのヒット率が 99%となった場合は，マスターノード のみとスレーブノード を用いた場合の差はほとんどなくなった．

実際の Web 検索エンジンでは，検索要求には偏りがある，すなわち局所性があるものと予想される．この実験によって検索結果のキャッシングは有効であることが分かったので，キャッシュの領域を十分に確保することでキャッシュのヒット率が上がれば，マスターノード と スレーブノード の通信のオーバーヘッドを大きく減少させることができる．

4.5 検索処理のスループット

分散並列方式によって検索要求に対するスループットをどれだけ向上させられるのかを検証するため，マスターノード が複数のクライアントから検索要求を受け付けた場合のスループットを測る実験を行なった．マシンは Sun Netra t1 を用い，マスターノード 1 台，スレーブノード 8 台，クライアントマシンは 2 台用いた．検索要求は新聞から切り出したキーワー

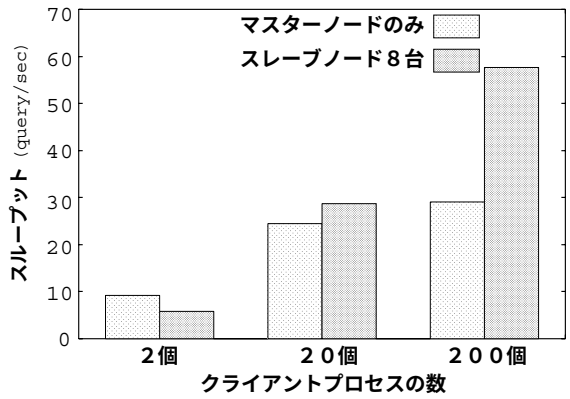

図 8: クライアントの数とスループットの変化

ドで，必ず 1 件以上の結果が見付かるもののみを用いた．

クライアントマシンを 2 台用いてマスターノードに検索要求を出すプロセスを複数実行することで，多数のクライアントから検索要求を受け付ける状況にした．各プロセスは，1 つの検索要求の結果を受け取ってから次の検索要求を出すものとする．すべてのプロセスの検索要求の合計は 10,000 件となるようにし，各クライアントマシンでプロセスの数を 1, 10, 100 と変化させ，かかった処理時間から単位時間あたりのスループットを求めた．また，検索要求に局所性は持たせず，キャッシュが有効になることがないようにした．図 8 に実験結果を示す．

3.3 節で述べたように，suffix array はある文字列の範囲で分割するようにしているので一つのスレーブノード で検索処理を行なっている間に，他のスレーブノード が担当する検索要求は並行して検索することができる．スレーブノード を用いずマスターノード のみで検索処理を行なう場合は，一つの検索要求を処理している間，他のクライアントプロセスは前の検索要求の処理が終るのを待たなくてはならない．

図 8 をみると，クライアントプロセスの数が 2 の場合は，スレーブノード を使用する場合よりマスターノード のみの場合の方がスループットが大きい．これはプロセス数が少ないために，クライアントプロセスがマスターノード で検索要求が処理されるのを待つ時間よりも，スレーブノード から検索結果が返って来る時間の方が長いからである．クライアントプロセスの数を 2 よりも増加させたとき，マスターノードのみを用いた場合は，検索処理の応答時間が長くなるため，スループットはあまり向上しない．それに対してスレーブノード を用いる場合は，検索要求を担当するスレーブノード が検索処理を実行していなければ他のスレーブノード と並列に検索処理を実行できるので，検索処理が受け付けられるまでの待ち時間が短く，多数のクライアントプロセスから検索要求が来ても並列に処理できる分，スループットが向上することが確認できる．

5 おわりに

以前提案したインクリメンタルな更新方式を分散並列化した方式を提案し，その効率について実験を行った．実験によって今回提案した分散並列処理方式が，インクリメンタルな更新方式の問題点であった再構成処理の高速化を実現し，更新や検索処理においても効果的であることを示した．

今後の課題としては，suffix array の分割の範囲を動的に変更することで，更新や検索の処理の負荷が特定のスレーブノードに集中しないようにする動的な負荷分散があげられる．また，文献 [4] で提案されているモバイル Web サーチロボットと組み合わせることにより，頻繁な情報更新を可能とする Web サーチシステムの検討を進めている．また，現在 suffix array の構成法は doubling technique [2] を用いている文献 [8] で提案されているアルゴリズムであるが，現在ではさらに効果的な構成法 [6, 5, 1] が提案されているので，suffix array の構成法について検討することも考えている．さらに今回の実験ではマスターノードが持つキャッシュはメインメモリ上に作成したが，2 次記憶上の高速インデックス構造 (例えば [3]) を用いることも検討している．

参考文献

[1] 伊東秀夫: Suffix array の効率的な構築法, 情報処理学会論文誌, Vol. 41, No. SIG1(TOD5), pp. 31–39 (2000).

[2] Karp, R., Miller, R. and Rosenberg, A.: Rapid identification of repeated patterns in strings, trees and arrays, *Proc. 4th Annual ACM Symp. on Theory of Comput.*, pp. 125–136 (1972).

[3] Kato, K.: Persistently Cached B-Trees, *IEEE Transactions on Knowledge and Data Engi-*

neering (2001). To appear.

[4] Kato, K., Someya, Y., Matsubara, K., Toumura, K. and Abe, H.: An Approach to Mobile Software Robots for the WWW, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 11, No. 4, pp. 526–548 (1999). Special Issue on Web Technologies.

[5] Kitajima, J. and Navarro, G.: A Fast Distributed Suffix Array Generation Algorithm, *Proceedings of the 5th International Symposium on String Processing and Information Retrieval (SPIRE'99)*, IEEE CS Press, pp. 97–104 (1999).

[6] Larsson, N. J. and Sadakane, K.: Faster Suffix Sorting, Technical Report LU-CS-TR:99-214, LUNDFD6/(NFCS-3140)/1–20/(1999), Department of Computer Science, Lund University, Sweden (1999).

[7] Manber, U. and Myers, G.: Suffix arrays: A new method for on-line string searches, *Proceedings of the 1stAnnual ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, pp. 319–327 (1990).

[8] Sadakane, K.: A Fast Algorithm for Making Suffix Arrays and for Burrows-Wheeler Transformation, *IEEE Data Compression Conference*, pp. 129–138 (1998).

[9] 吉原潤, 加藤和彦: WWW 検索エンジンのためのインクリメンタルな全文検索インデックス更新方式, 情報処理学会論文誌：データベース, Vol. 40, No. SIG8(TOD4), pp. 112–125 (1999).