DEIM Forum 2009 C7-6

半構造データベースにおける検索速度向上のための Index Fabric 構造の拡張

尾川　広樹†　　上土井陽子††　若林　真一††

† 広島市立大学 情報科学部
†† 広島市立大学大学院　情報科学研究科

あらまし　近年，XML で記述されたデータの利用が普及するに従い，複数の異なる機関の持つデータを統合して扱おうという要求が高まってきた．その際，１つのデータベースにデータを集めるのではなく，データを分散して格納したまま XML データを効率よく管理するために，Index Fabric というインデックス構造が提案されている．Index Fabric では，リレーショナルデータベースで行われているように明確に構造を定義してデータを集中管理するのではなく，構造などを詳細には定義せず，構造もデータと一緒に文字列として格納する．しかし，Index Fabric では，属性が決定されていない曖昧な検索を行う際，検索速度が遅くなるという問題がある．本論文では，曖昧な検索の際の Index Fabric での検索速度をさらに向上させるために，Index Fabric 構造を拡張する．

キーワード　Index Fabric，パトリシアトライ, XML

Extending an Index Fabric Structure for Search Speed Improvement on Semistructured Databases

Hiroki OGAWA†, Yoko KAMIDOI††, and Shin'ichi WAKABAYASHI††

† Faculty of Information Sciences, Hiroshima City University
†† Graduate School of Information Sciences, Hiroshima City University

Abstract　In recent years, it is needed to manage the XML data efficiently as the use of data described by XML spreads. Ones want to efficiently integrate and treat datasets on multiple peers. In order to search data efficiently, a data structure called“ Index Fabric ”had been suggested. Index Fabric is not to define the structure explicitly as relational databases and control the data intensively, but to store the structure as a character string with data without defining the structure. In Index Fabric, however, there is an issue that the search speed becomes slowly for ambiguous searching. In this papen, we extend the data structure of Index Fabric to improve performance of ambiguous searching.

Key words　Index Fabric, Patricia trie, XML

1. ま え が き

近年，XML で記述されたデータの利用の普及に伴い，複数の異なる機関の持つデータベース上のデータを統合して扱おうという要求が高まってきた．XML とは，文書やデータの意味や構造を記述するためのマークアップ言語の一つで，木構造を持つ情報をテキストで記述するための構文である．XML で記述されたデータは，厳密には定義されていない構造を持つ半構造データに分類される．従来までに用いられてきた方法として，データを集めてリレーショナルデータベースとして保存し，管理するという方法がある．リレーショナルデータベースでは，データはタプルの集合に変換されテーブルに保存されるので，データのためのスキーマが必要となる．半構造データにおいて，スキーマを決定することは困難であるため，無理にリレーショナルデータベースに詰め込もうとすると無駄に領域を使ってしまう．そこで，データを分散して格納したまま，XML で記述されたデータを効率よく管理する方法として，パトリシアトライに基づく Index Fabric というインデックス構造が提案されている [1]．Index Fabric では，リレーショナルデータベースで行われているように明確に構造を定義してデータを集中管理するのではなく，構造などを詳細には定義せず，構造もデータと一緒に文字列として格納する．しかし，Index Fabric では，属性が決定されていない曖昧な検索を行う際，検索速度が遅くなるという問題がある．本稿では，曖昧な検索の際の Index Fabric

での検索速度をさらに向上させるために，Index Fabric 構造を拡張する．そして，提案手法の有効性を調べるために，検索ブロックの数に注目して従来法との比較を行う．実験結果を用いて検索ブロック数減少の観点での提案手法の有効性を示す．

2. パトリシアトライ

パトリシアトライは，基数探索法の一種で，一方向分岐を作らず，各節点にキーの比較位置情報を格納したデータ構造である [5]．つまり，キーの間で異なる遷移のみで木を構築するため，根以外の節点には，不要な遷移情報が存在せず，空間効率が良い．よって，パトリシアトライのサイズは，挿入されたキーの長さに依存せず，たとえキーが長くとも，新しいキーの挿入は多くても１つのリンクと節点をインデックスへ加えるだけで済む．すべての接頭辞を表現していないため，パトリシアトライ上の葉には，葉に対応したキーを格納する．パトリシアトライの探索処理は，パトリシアトライ上を根から葉まで遷移し，葉に対応したキーと探索キーを比較する．

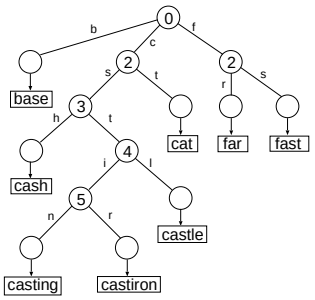

図 1　パトリシアトライの例

キー集合 K={“ cash ”,“ casting ”,“ castiron ”,“ castle ”,“ cat ”,“ far ”,“ fast ”,“ base ”} に対するパトリシアを図 1 に示す．ここで，節点の中の値はキーの比較位置を意味する．

大規模データベースではパトリシアトライは，一定のブロックサイズの部分トライに分割して格納される．

大規模なデータベースを構築する場合，ブロック単位のランダムアクセスが可能な補助記憶装置（ハードディスクドライブなど）上に木構造を実装するためには，根から葉までの深さが等しくなければキーによって探索時間が変化してしまう．従って，パトリシアトライは，不均衡なメインメモリ構造であり，このため，ハードディスクなどに保存されるようなディスクベースのデータ探索には滅多に使用されない．

3. Index Fabric

Index Fabric [1] は，パトリシアトライの不均衡を調整し補足するため，階層的に部分トライへインデックス付けする．インデックス付けによって，(部分トライによって）垂直に順にたどれるだけでなく，(Layer を使うことで）水平レベルにもアクセスできる，二次元のツリー構造である．ここで，垂直とは部分トライ内で枝をたどることを言い，水平とは Layer をまたいで枝をたどることを言う．

3.1 パトリシアトライから Index Fabric への変換法

Index Fabric では，パトリシアトライの不均衡を調整するために，新しい水平層（Layer）を作り，それによってパトリシアトライの部分トライに対してインデックスを付ける．

パトリシアトライから Index Fabric 構造へ変換するための手順について，以下に簡単に述べる．

（1）`Layer0` において，与えられた文字列を基にパトリシアトライを構成する．`i` ← `0` とする．

（2）`Layer i` のパトリシアトライを，ブロックサイズの部分トライに分割し，それぞれのブロックに対して共通の接頭辞を定める．

（3）`Layer i+1` において，共通の接頭辞に対してインデックスを付けた部分トライを構成する．

・部分トライの構成方法

1．ブロックを横断する枝を持つ親節点を抜き出す．

2．親節点を持つ部分トライの接頭辞に対してパトリシアトライを構成する．もしあるブロックに複数の親節点があった場合，それぞれの節点の持つ接頭辞に対してトライを構成する．

3．`Layer i+1` において作られた節点は，ラベルを持たない直接枝（破線矢印）によって自身の部分トライを指す．また，ラベルを持った参照枝（実線矢印）によって抜き出した親節点とつながっている部分トライを指す．

（4）`Layer i+1` に構成された部分トライに対し，`i` ← `i+1` として（2）（3）を繰り返す．

（5）`Layer i` のブロックが１つになると終了．

上記の手順に従って，図 1 のパトリシアトライを Index Fabric に変換する例を以下に示す．

[実行例]

（1） Layer0 におけるパトリシアトライは図 1 である．

（2） 図 1 を，ブロックサイズに分割したものを図 2 に示す．このとき，図 2 における各ブロックでの共通の接頭辞は“　”,“ ca ”,“ cas ”,“ cast ”,“ casti ”,“ fa ”である．

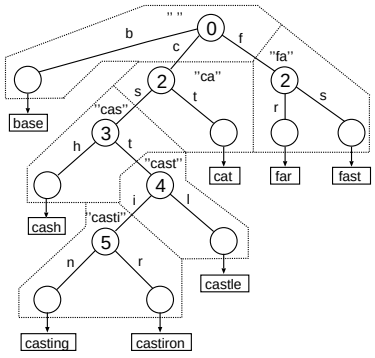

図 2　ブロック分割されたパトリシアトライ

（3） Layer $i+1$ において，共通の接頭辞に対してインデックスを付けた部分トライを構成する．図 2 に対し，部分トライを構成したものを図 3 に示す．

（4） Layer1 に構成された部分トライに対し，(2)，(3）を繰り返す．節点が１つになるまで繰り返したものを，図

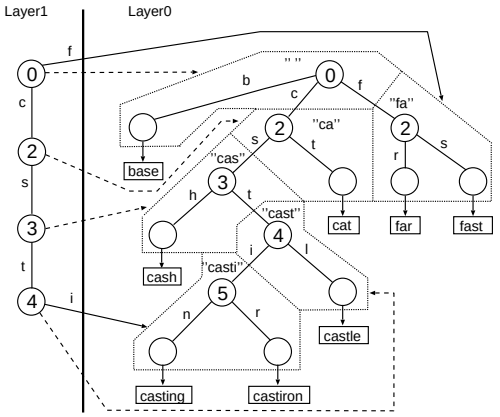

図 3　ブロックに対するインデックス付け

4 に示す.

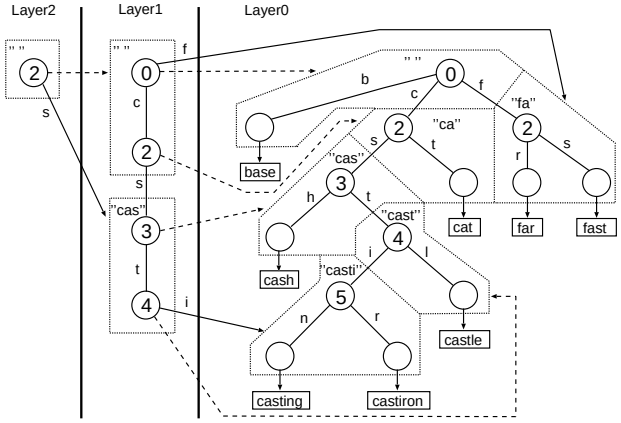

図 4　Index Fabric

この構造では，木自体は均衡が取れていないが，Layer を使用する水平な木と組み合わせることで仮想的に均衡をとる．

例えば，キー“ far ”と“ castiron ”を検索する．そのとき，パトリシアトライでは,“ far ”の検索のために 2 つのブロック検索,“ castiron ”の検索のために 5 つのブロック検索を必要とする．Index Fabric では,“ far ”と“ castiron ”のどちらも 3 つのブロックを検索をする．これにより，Index Fabric は均衡がとれていることがわかる．

3.2　キーの検索手順

Index Fabric におけるキーの検索手順を以下に示す．ここで，Layer をまたぐ枝をたどることを，水平に進むと定義する．

Step1:　一番左の Layer のブロックの根節点から検索を始める．

Step2:　特定のブロック内では，キーの比較位置情報によって，検索キーの文字列を比較する．一致した場合，それらの枝へ探索を続ける．

Step3:　もし，ラベル付けられた枝が参照枝ならば，検索は次の Layer の異なるブロックへ水平に進む．異なるブロックへ進んだ後，そのブロックの接頭辞とキーの接頭辞を比較し，一致するならばそのブロック内の根節点に遷移する．一致しないならば，前のブロックの節点へ戻り，次の Layer の新しいブロックへ直接枝をたどって水平に進む．

Step4:　もし，ラベル付けられた枝が検索キーの適切な文字列に一致しないならば，検索は次の Layer の新しいブロックへ直接枝たどって水平に進む．

Step5:　Layer 0 に達し，要求されたデータ要素が見つかるまで，Step2 と Step3 または Step4 を繰り返す．Layer 0 での検索において，ラベル付けられた枝が検索キーの適切な文字列に一致しないならば，キーが存在しないことを示し，検索終了．

Step6:　データ要素が見つかった場合，見つけられたデータ要素と検索キーが一致しているかを確認し，一致した場合，キーが存在することを示し検索終了．

これらの検索プロセスは，一つの Layer あたり 1 ブロックを調べるので，常に Layer と同じ数のブロックを調べる．ブロックがディスクブロックに対応しているならば，必要なブロックがキャッシュにない限り検索は一つの Layer につき 1 回の IO を必要とする．パトリシア構造を使う利点の 1 つは，キーが非常に詰まって格納され，一つのブロックで，多くのキーをインデックス付けることができることである．そして，Layer 0 はディスク上に格納されなければならないが，それ以外の Layer はメインメモリに格納することができるため，どんなに長い半構造データに通ずるようなインデックスをつけられた経路をたどるとしても，多くても一つのインデックス IO で十分である．

4.　Index Fabric による XML へのインデックス付け

ここでは，Index Fabric への挿入のためのキーとして，XML の経路をコード化することについて述べる．

図 5 のように，二つの納品書のドキュメントがあるとき，これらの経路をコード化する．

まず，図 6 のようにそれぞれのタグに指示子を割り当てる．タグを指示子に置き換えると，文字列「IBN ABC Corp」等が得られる．これは，

```
<invoice>
  <buyer><name>ABC Corp</name></buyer>
</invoice>
```

と同じ意味を持つ.

ここで，タグに属性が含まれている場合，タグ付けられた子供のようにタグを扱う．例えば，図 5 のタグ<item count=3>…は，<item><count>3</count>…のように，<item>の属性が<item>の中に入れ子にされた，他のタグの兄弟として現れる．

タグと属性を区別するために，タグと属性には，異なる指示子が割り当てられる．図 6 では，タグに C，属性に C’ がそれぞれ割り当てられている．

次に，図 7 に示されるキーを作るために，XML の根から葉への経路をコード化する．

最後に，図 8 に示されるトライを生成するために，Index Fabric にこれらのキーを挿入する．インデックス付けられたキーは，文書ファイルや，リレーショナルシステムの組，または，XML データベースの対象物を参照するポインタと関連付けることができる．簡単化のため，図 8 では，水平層とトライ

Doc1:

```
<invoice>
  <buyer>
    <name>ABC Corp</name>
    <address>1 Industrial Way</address>
  </buyer>
  <seller>
    <name>Acme Inc</name>
    <address>2 Acme Rd.</address>
  </seller>
  <item count=3>drill</item>
</invoice>
```

Doc2:

```
<invoice>
  <buyer>
    <name>Oracle Inc</name>
    <phone>555-1212</phone>
  </buyer>
  <seller>
    <name>ABC Corp</name>
  </seller>
  <item>
    <name>nail</name>
    <count>4</count>
  </item>
</invoice>
```

図 5　文　書　例

```
<invoice> = I
<buyer> = B
<name> = N
<address> = A
<item> = T
<phone> = P
<count> = C
count(attribute) = C'
```

図 6　指　示　子

Document1	Document2
I B N ABC Corp	I B N Oracle Inc
I B A 1 Industrial Way	I B P 555-1212
I S N Acme Inc	I S N ABC Corp
I S A 2 Acme Rd.	I T N nail
I T drill	I T N 4
I T C' 3	

図 7　経路のコード化

のいくつかを省略する．

XML の経路をコード化し，パトリシアトライにキーとして挿入する時，半構造データでよく扱われる属性を持つデータはより深い場所に位置し，あまり扱われないような属性を持つデータは浅い場所に位置する．しかし，検索においては，よく扱われる属性を探すことの方が多いと予測される．Index Fabric では，水平層を使うことで均衡が保たれ，さらに，深い位置にあるデータから検索するように作られているので，半構造データを扱うのに向いている．

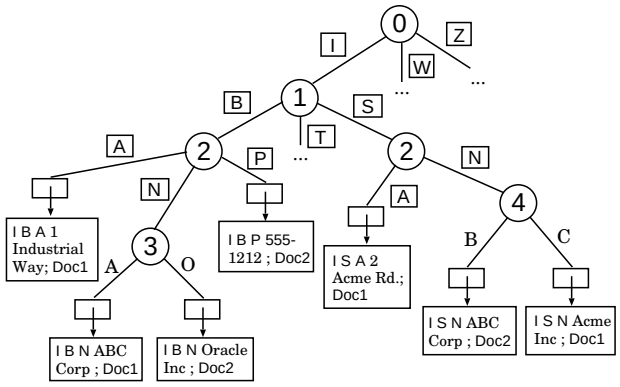

図 8　Index Fabric へのキーの挿入

5. Index Fabric の提案拡張法

Index Fabric のキー検索では，参照枝によって分岐したものの，ブロックの接頭辞と比較することによって Layer を戻らなければならないことがあるが，分岐したということは何文字目が何であるかという情報を得たことになる．第 3 節で述べた検索手順では，参照枝をたどった時点で，検索キーの何文字目が何であるという情報を得たことになる．例えば，図 4 において，Layer 2 の参照枝をたどった場合，3 文字目が s であるとわかる．Layer を戻って直接枝をたどるということは，これらの情報を無駄にしている．そこで，“ **s* ”の‘ s ’という決定している文字を頼りに検索ができるよう新しい枝（連結枝）を作り，Layer を戻ることなくデータを検索できるよう検索速度の向上を目的とした Index Fabric の拡張を提案する．

5.1　ブロック分割方法

本稿では，パトリシアトライのブロック分割に関しては，深さ優先探索 [4] を用いてキーからの距離が一定以上になると分割する方法を用いた．上記の例を用いて，ブロック分割を説明する．

図 1 のパトリシアトライをブロック分割する例を，図 9 に示す．

1. まず，深さ優先探索によってキー“ casting ”,“ cast-iron ”が最も深い位置にあることがわかる．それらのキーからの距離が 3 となる節点までを一つのブロックとすると，比較位置 3 の節点と比較位置 4 の節点の間が分断される．このとき，ブロックの接頭辞“ cast ”が定まる．(図 9（a)）
2. 次に一番深い位置にあるキーは“ cash ”であるので，同じ操作を行うと，比較位置 0 と比較位置 2 の節点の間が分断され，ブロックの接頭辞“ ca ”が定まる．（図 9（b)）
3. 最後に，一番深い位置にあるキーは“ far ”と“ fast ”である．同じ操作を行うと，根節点に達するので，一つのブロックとしてまとめる．ここで，接頭辞“　”が定まる．(図 9（c)）

上記の例では説明のため距離が 3 以内の節点をまとめブロッ

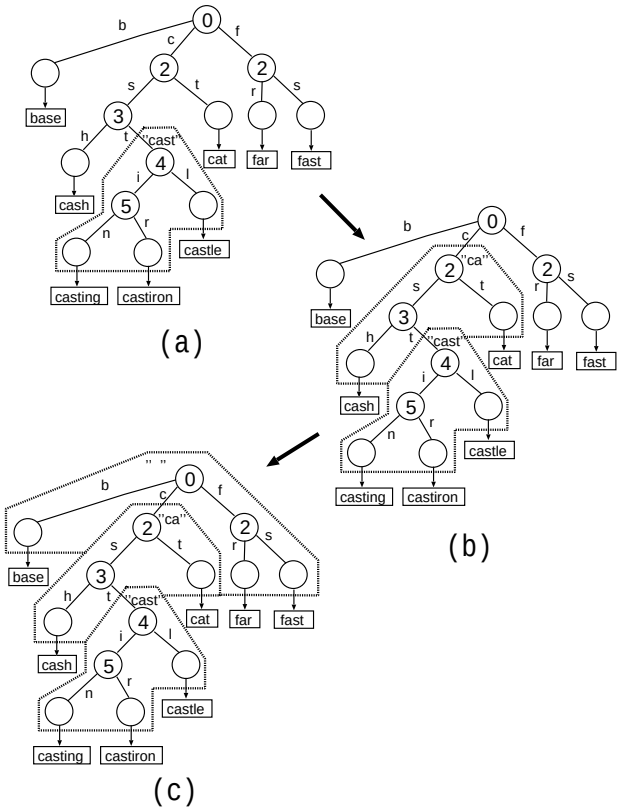

図 9　パトリシアトライのブロック分割

クとしたが，実際のディスクに格納する 1 ブロックあたりの節点数は，1000 ポインタ程度である [2].

5.2　提案拡張方法

Index Fabric の提案拡張方法を以下に述べる．

Step1.　参照枝を持つ節点を調べ，その節点の比較位置, 比較文字を構造体に保存する．その際，比較位置が 0 ではない節点のみを対象とする．

Step2.　構造体に保存された各節点の比較情報を用いて, 比較位置を比較文字にし，それ以外を*として Index Fabric で検索する．

Step3.　保存された節点以降に検索された初めのキー（ファーストキー）に対し，参照枝によって指される節点から連結枝をつなぐ. ファーストキーをラストキーとして構造体に保存する．

Step4.　次に発見されたキーに対して，ファーストキーから連結枝をつなぐ．最後のキーをラストキーとして記憶し，以前のラストキーと新しいラストキーを連結枝でつなぐ．

Step5.　構造体に保存された全ての参照枝に対して，連結枝を作り終えると終了．

5.3　拡張の実行例

上記の手順に従って，図 4 の Index Fabric を拡張する例を以下に示す．

1. 参照枝を持つ節点を調べると，その節点の比較位置，比較文字を構造体に保存する．Layer1 の（比較位置 0，比較文字 f）は該当しないため，Layer2 の節点（比較位置 2，比較文字 s）と Layer1 の節点（比較位置 4，比較文字 i）が保存される．構造体は，図 10 のようになる．

label	character
2	s
4	i

図 10　拡張に用いる構造体の利用例 1

2. 構造体に保存された節点の比較情報（label：2，character：‘ s ’）より，2 文字目を s，その他の文字を*として，Index Fabric で検索を行う．
3. Index Fabric の検索方法に従うと,“ casting ”,“ castiron ”,“ castle ”,“ cash ”が発見された後，Layer1 の“ cas ”ブロックの根節点から Layer を戻って，Layer1 の“　”ブロックの根節点に進む．その後,“ fast ”を最初に見つける.“ fast ”に対して，参照枝によって指される節点（Layer1 の“ cas ”ブロックの根節点）から連結枝をつなぐ．また,“ fast ”をラストキーとして構造体に保存する．このときの構造体を図 11 に，Index Fabric の拡張途中を図 12 に示す.

label	character	last key
2	s	fast
4	i	NULL

図 11　拡張に用いる構造体の利用例 2

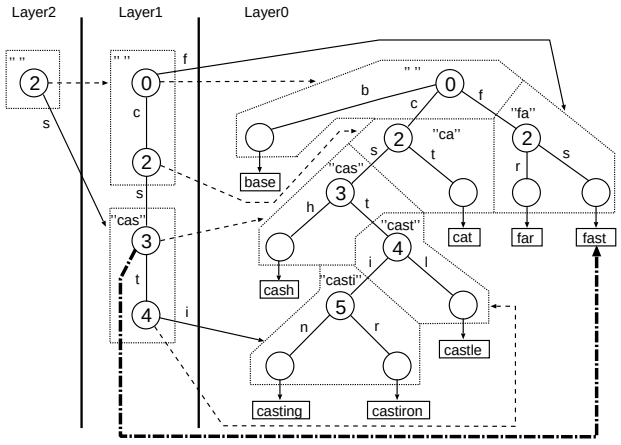

図 12　Index Fabric の拡張途中

4. 検索を進めていくと，次に“ base ”を見つける．よって,“ base ”に対してファーストキーから連結枝をつなぐ．また,“ base ”をラストキーとして構造体に保存する．このときの構造体を図 13 に，Index Fabric の拡張途中を図 14 に示す．同じように検索を進めていき，全てのキーを見つけ終えるまで，最後のキーをラストキーとして記憶し，以前のラストキーと新しいラストキーを連結枝でつなぐ.“ **s* ”に一致するキーは他にないため，これで連結枝を作り終えたことになる．

label	character	last key
2	s	base
4	i	NULL

図 13　拡張に用いる構造体の利用例 3

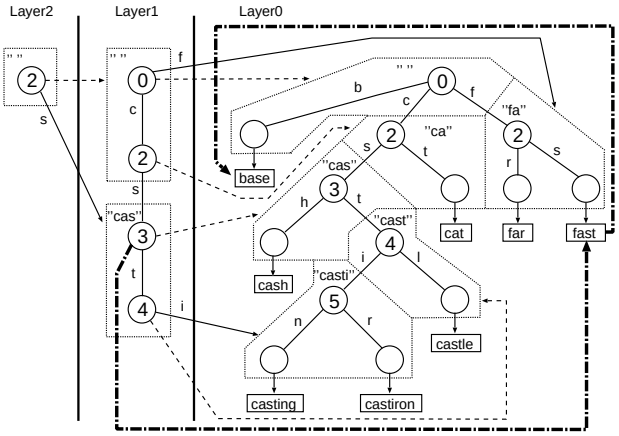

図 14　Index Fabric の拡張結果

5. 次に，構造体に保存された節点の比較情報（label：4, character：ｉ’）に対しても連結枝を作るが，今回は一致するキーがないため，連結枝は作れない．よって，図 14 が Index Fabric の拡張結果となる．

すでに連結枝につながれているキーに，さらに連結枝をつながなくてはならない場合，キーの内容をコピーしたキーを新たに作り，そのキーに対して連結枝をつなぐ．そうすることで，節点とキーは，一つのポインタのみを必要とする．作成された連結枝を使うことで無駄なブロック検索が減少し，検索速度の向上が見込める．

先に述べたように，従来の Index Fabric の検索は，パトリシアトライのより深い位置にあるデータに該当するかどうかを見ていくという性質を持っている．XML の経路をコード化し，キーとしてパトリシアトライに挿入するとき，よく扱われる属性を持つデータ群は深い場所に位置し，あまり扱われないようなデータは浅い場所に位置する．従って，Index Fabric での検索では，よく扱われる属性を持つデータに該当するかどうかを優先的に見ていく．そのため，深い位置にあるデータ要素ほど一般性が高いと定義すると，一般性の高い順番に検索を行っていることとなる．

拡張法では，Index Fabric に曖昧なキーワード検索を行い，発見された順に連結枝を作った．よって，キーは一般性の高い順に連結されている．これは，一般性の高い順に検索を行うという Index Fabric の性質を失っていないことを意味する．

6. キーの検索手順

提案拡張法におけるキーの検索手順を以下に述べる．Step3 と Step7 以外は，従来の Index Fabric と同じである．

Step1: 一番左の Layer のブロックの根節点から検索を始める．

Step2: 特定のブロック内では，キーの比較位置情報によって，検索キーの文字列を比較する．一致した場合，それらの枝へ探索を続ける．

Step3: もし，ラベル付けられた枝が参照枝ならば，検索は次の Layer の異なるブロックへ水平に進む．異なるブロックへ進んだ後，そのブロックの接頭辞とキーの接頭辞を比較する．

1. ブロックの接頭辞とキーの接頭辞が完全に一致した場合，そのブロック内の根節点に遷移する．
2. ブロックの接頭辞とキーの接頭辞が曖昧に一致した場合（キーの接頭辞に*を含む場合），一致したとして探索を続ける．一致した節点の子孫の節点への探索を終了した後，連結枝をたどる．(Step7 へ)
3. ブロックの接頭辞とキーの接頭辞が一致しない場合，前のブロックの節点へ戻り，次の Layer の新しいブロックへ直接枝をたどって水平に進む．

Step4: もし，ラベル付けられた枝が，検索キーの適切な文字列に一致しないならば，検索は次の Layer の新しいブロックへ直接枝をたどって水平に進む．

Step5: Layer 0 に達し，要求されたデータ要素が見つかるまで，Step2 と Step3 または Step4 を繰り返す．Layer 0 での検索において，ラベル付けられた枝が，検索キーの適切な文字列に一致しないならば，キーが存在しないことを示し，検索終了．

Step6: データ要素が見つかった場合，見つけられたデータ要素と検索キーが一致しているかを確認し，一致した場合，キーが存在することを示し検索終了．

Step7: 連結枝をたどり終えるまで全てのデータ要素に対して，データ要素と検索キーが一致しているかを確認し，一致した場合キーが存在することを示す．連結枝をたどり終えると，検索終了．

7. 評価と考察

7.1 実験環境

Index Fabric の提案拡張法を評価するために実験を行った．比較対象の従来法は Index Fabric とした．本研究の実験環境を以下に示す．提案法と従来法を CPU:Core2 Duo CPU E4600 2.40GHz，メインメモリ:2GB，OS:WindowsXP Professional の計算機上にC言語を用いて実装した．

7.2 実験結果

実在のデータベース DBLP [3] の XML 文書からデータ要素をランダムに抜き出し作成した入力データを用いた．従来法と提案法に対し，検索をした際の実験データを表 1 に示す．検索ワード‘ O ’,‘ I ’,‘ A ’は，DBLP のタグ<booktitle>, <inproceedings>, <article>をそれぞれ示す．キーとして入力したデータ要素数は 3452 個，Index Fabric における総ブロック数（総部分トライ数）は 319 個である．

7.3 考　　察

検索ワード“ I* ”と“ A* ”の例は，連結枝を使わない場合の検索である．連結枝を使わないため，従来法，提案拡張法のどちらも同じ手順で検索を行う．よって，検索ブロック数も変わっていない．

表 1 実験結果

検索ワード	ヒットするkey の数	検索ブロック数（キー比較を含む）		検索ブロック数		差	改善率 (%)
		従来法	提案法	従来法	提案法		
O	32	70	41	38	9	29	76
**k*	92	176	115	84	23	61	73
**m*	280	387	326	107	46	61	57
**s*	275	386	318	111	43	68	61
**t*	198	295	243	97	45	52	54
***a*	717	994	765	277	48	229	83
***e*	432	697	506	265	74	191	72
***i*	431	687	483	256	52	204	80
***o*	452	725	496	273	44	229	84
****c*	167	459	376	292	209	83	28
I*	1935	2116	2116	181	181	0	0
A*	1283	1405	1405	122	122	0	0

検索ワード“ I* ”と“ A* ”以外は，連結枝を使用する曖昧なキーワード検索の例である．結果としては，提案拡張法の方が，従来法よりも全体的に検索ブロック数が減少している．従来法では，決定している文字の前の*が多くなるにつれ，検索ブロック数が増えていることがわかる．これは，ブロックの接頭辞とキーの接頭辞を比較する際に，*の数が多いほど一致しやすいため，より多くのブロックを検索しなければならないからだと考えられる．提案法では，検索ブロック数は検索ワードによってばらつきが大きい．これは，連結枝に達するまでに検索したブロック数に依存するからだと考えられる．

実験結果より，提案法は，曖昧なキーワード検索において検索ブロック数の減少の面で，有効であると言える．

ここで，提案法を採用した場合に増加するメモリ量について述べる．まず，構造体に使用したメモリ量が増加する．増加量は，参照枝の数に依存するが，1 つの参照枝に対して 1 つの整数と 1 文字分のメモリを必要とする．また，連結枝に使用する節点とキーに 1 つずつポインタが必要となる．さらに，今回の実装ではすでに連結枝につながれているキーに連結枝をつながなければならない場合，キーの内容をコピーしたキーを別に作り，そのキーに連結枝をつなげているため，新たに作られたキーの数だけメモリ量が増加する．

8. おわりに

本論文では，Index Fabric での曖昧なキーワード検索に対し，検索ブロック数を減少させる手法を提案し，計算機実験により評価を行った．提案手法においては，深さ優先探索を用いてパトリシアトライをブロック分割し，Index Fabric を構成した．Index Fabric に対して，曖昧なキーワード検索を行ったときの検索ブロック数を求め，従来法との検索ブロックの数を比較することで，実験的に評価した．その結果，曖昧なキーワード検索において，検索ブロック数が減少することがわかった．

今後の課題として，他のベンチマークデータについての実験と考察を行い，提案手法の有効性を検証することが挙げられる．また，さらなる拡張として，今回は 1 文字に対して連結枝を作ったが，参照枝を複数回たどった場合の連結枝の作成が挙げられる．この拡張により複数個の文字が確定した上で，複数の文字に対して連結枝を作ることができ，さらなる検索ブロック数の減少が期待できると考えられる．

文　献

[1] B. F. Cooper, N. Sample, M. J. Franklin, G. R. Hjaltason and M. Shadmon,“ A fast index for semistructured data, ” Proceedings of the 27th VLDB Conference, 2001.

[2] B. F. Cooper, N. Sample and M.Shadmon,“ A parallel index for semistructured data, ” Proceedings of the ACM Symp. on Applied Computing (SAC2002), pp.890-896, 2002.

[3] DBLP Computer Science Bibliography. At http://www.informatik.uni-trier.de/~ley/db/.

[4] 茨木俊秀,“ アルゴリズムとデータ構造, ” 昭晃堂, 1989.

[5] D. R. Morrison,“ Patricia practical algorithm to retrieve information coded in alphanumeric, ” Journal of the ACM, Vol.15, pp.514-534, 1968.