XML データベースにおけるキャッシュ管理手法の提案

朴　大一† 　遠山　元道‡

†慶應義塾大学 理工学研究科 開放環境科学専攻〒223-852 横浜市港北区日吉 3−14−1
‡慶應義塾大学 理工学部 情報工学科 〒223-852 横浜市港北区日吉 3−14−1
E-mail: †park@db.ics.keio.ac.jp, ‡toyama@ics.keio.ac.jp

あらまし 本研究では XML データベースのキャッシュ管理方法について提案する。XML データベースと関係データベースではデータ構造やクエリ構造において大きな違いがある。XML データベースの代表的なクエリ言語として XQuery が挙げられる。その一部に使われる XPath を用いて XML データの階層構造を辿って更新や検索ができる。２つの XPath 表現を比較するとき、これらの結果の包含関係を判定できる場合がある。本研究ではそれに着目し、XML データベースのキャッシュ管理を行う際、クエリをメタ情報としてキャッシュ管理する手法を提案する。関係データベースではページ単位によるキャッシュ管理が主流だったが、XML データベースでクエリ単位によるキャッシュ管理を行うことの得失を考察する。

キーワード XML、キャッシュ、XQuery、XPath

A Method of Cache Management Technique in XML Databases

Daeil Park† 　Motomichi Toyama‡

† School of Science for OPEN and Environmental Systems, Faculty of Science and Technology, Keio University
3-14-1 Hiyoshi,Kouhoku-ku, Yokohama-si, Kanagawa-ken, Japan 223-8522
‡ Department of Information and Computer Science, Faculty of Science and Technology, Keio University
3-14-1 Hiyoshi,Kouhoku-ku, Yokohama-si, Kanagawa-ken, Japan 223-8522
E-mail: †park@db.ics.keio.ac.jp, ‡toyama@ics.keio.ac.jp

Abstract We propose a cache management method of XML Databases in this research. There are differences in the data and the query structure between XML Databases and Relational Databases. XQuery is the most famous XML query language in XML Databases. The layered structure of the XML data can be traversed along XPath expressions used in the part of XQuery. Two XPath expressions being compared, the containment relationship of these results might be able to be judged. In this work, we pay attention to the relationship and propose a method of Cache management using it as meta information when doing the cache management. Although the cache management by the unit of page or tuple was a main technique in the Relational Databases, the merit and demerit of doing the cash management by the unit of query in the XML Databases are considered in this work.

Keyword XML, cache, XQuery, XPath

1. はじめに

1.1. 研究の背景

近年、XML[8]がウェブ上でデータの表現や文書のやりとりについて標準となり、XML データの格納方法や問い合わせ処理[3][6]に関する研究がより注目されている。問い合わせ処理を行う際、頻繁に使われているクエリの結果をキャッシュに格納し,それを再び用いるのは問い合わせ最適化技術[3][6]の一つである。従来、関係データベース環境でキャッシュを管理する際、キャッシュ内部のセグメントとしてページ単位[2]やタプル単位[10]で管理を行うことが多かったが、そのま

ま XML 環境で行うと無駄なデータの重複によってスペースコストが掛かってしまう。さらに、それによる非効率なキャッシュ管理が生じてヒットレートが下がるため、検索を行なう際、クエリに対して応答遅延時間が遅くなると考えられる。それは関係データベースと XML データの間、データ構造やクエリ構造が異なるにもかかわらず関係データベース環境におけるキャッシュ管理手法をそのまま使用するということから生じる問題点である。XML データは木構造で表すことができ、検索を行う際、XQuery[9]が問い合わせ言語として用いられる。XQuery ではその一部である XPath[1]を通じて階層構造を辿ることができて検索、更新や削除を行なう。XPath は XML データ内部の一部の階層構造を表すことができる。本研究では 2 つの XPath 表現を比較するとき、それらの結果の包含関係[4][7]を判明できる場合があることに着目し、クエリ単位によるキャッシュ管理手法を提案する。

1.2. 本論文の構成

本研究は次のように構成される。2 章では XML データに対する 2 つの XPath 間の包含関係について述べる。3 章では従来のキャッシュ管理手法とその問題点を挙げる。そして、XML データに適合する新たなキャッシュ管理手法について技術する。XPath 表現をキャッシュのセグメントの単位として扱うことで、より精度が高い管理手法を述べる。さらに、XPath 表現（セグメント）の間の意味的な関係とそのセグメントの統計値に基づいてキャッシュセグメントの統合や分解によるキャッシュ再構成について紹介する。4 章では本研究のキャッシュマネージャを構成するそれぞれのモジュールについて説明を行い、まとめで今後の課題を挙げる。

2. クエリの間の包含関係

キャッシュをクエリ単位で管理する場合、キャッシュされたクエリ同士の間の関係や新たなクエリとキャッシュされたクエリの間の関係を表すことによって問い合わせ処理の性能を高めることができると考えられる。まず、キャッシュされたクエリ同士の間の包含関係を判定することによってキャッシュセグメントの間に不要な重複部分をできるだけ小さくすることができてキャッシュのスペースコストを減らす。さらに、新たなクエリとキャッシュされたクエリの間の完全包含と部分包含を表すことでクエリ処理における処理の負担を減らすことによって応答遅延時間を短縮する。簡単な例を挙げて XPath 間の包含関係を述べる。

```
<bib>
   <book year = “2002”>
      <title>XML DBs and Semantic Web </title>
      <author>
         <last>Thuraisingham</last>
         <first>Bhavani</first>
      </author>
      <publisher>Addison-Wesley</publisher>
      <price>65.95</price>
   </book>
   <book year = “2003”>
      <title>XQuery from the Experts</title>
      <author>
         <last>Chemberlin</last>
         <first>Don</first>
         <last>Simeon</last>
         <first>Jerome</first>
      </author>
      <publisher> Addison-Wesley</publisher>
      <price>87.45</price>
   </book>
</bib>
```

図 1 .XML データの例

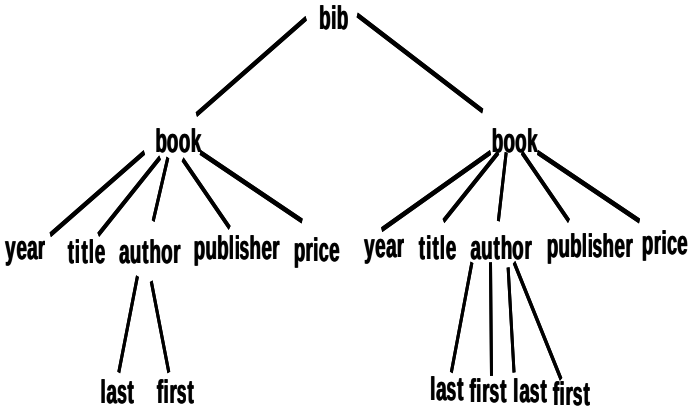

図 2. XML データの木構造

図 1 は簡単な XML データを表す。図 2 ではその XML データに対する木構造を表す。それに対するさまざまな XPath 表現のうち 2 つの関係は、図 3 の 4 つの中の何れかになることが分かる。ただし、図 3 の（a）と（b）と（d）は 2 つの単純な XPath 表現の間の関係を表し、（c）の方は複雑な XPath 表現の間の関係を表す。例えば、図 3 で XPath 2 がキャッシュのセグメントとしてすでに格納されていると仮定すると、(a)の場合はクエリ（XPath1）に対する処理を行う際、全部をキャッシュ側で処理することができ、(b)と(c)は一部だけキャッシュ側で処理することができる場合があり、応答遅延時間を短縮することでクエリ処理における効率を高める。しかし、(d)の場合、キャッシュ側には XPath 1 の情報を持っていないため、遠隔地にあるサーバやディスクまでアクセスしなければならない。XPath 表現は以下のように、連続されたロケーションステップで表す。

XPath::=/LocationStep_1/LocationStep_2/...
/LocationStep_N

上の式でそれぞれの LocationStep は以下のように構成される。

LocationStep_I::=
Axis-specifier_I::Node-test_I[Predicate_I]

XPath 表現には 13 種の軸（axis-specifier）がある。そのうちよく用いられている 2 種、‘//’, ‘/’ - それぞれ‘/Ancestor’と‘/Child’を表す - の間には以下のような関係が成り立つ。

- ‘//’ ⊇’/’
  (ただし、同じ context を基準とする)

同じ考え方から、‘/Parent’と‘/Ancestor’、‘/Following-sibling’と‘/Following’、‘/Preceding-sibling’と‘/Preceding’の関係など、すべての XPath 軸のそれぞれに対して包含関係の有無を調べることができる。

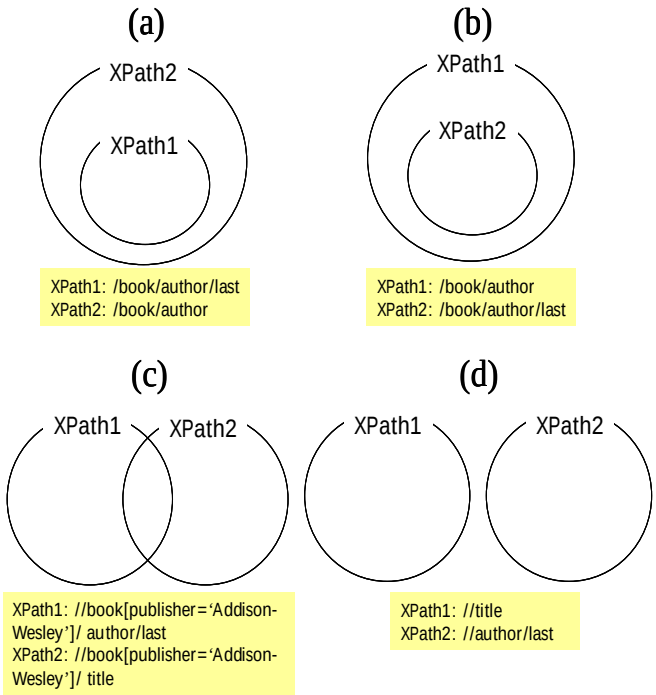

図 3 . XPath 間の包含関係

従来の研究ではその概念を用いて 2 つのノードの間の包含関係によるクエリ処理[14]やインディクス技術[13][15]に注目されたが、本研究では、2 つの XPath 間の包含関係における概念を拡張し、新たなクエリとキャッシュ側との包含関係やキャッシュされたクエリ同士の間の意味的な関係に基づいたキャッシュ管理ついて述べる。

3. キャッシュ管理手法

キャッシュ管理はクエリ最適化を目指すための 1 つの技術である。キャッシュマネージャは既存のクエリパターンを分析し、それから得られる統計値（最近アクセス時間、アクセス頻度など）に基づいて未来のクエリを予測し、それらを扱うことによってキャッシュ管理を行う。

3.1. 従来のキャッシュ管理方法とその問題点

従来のデータベース環境ではキャッシュを管理する際、ページ[2]やタプル[5] [10]を、そのセグメントの単位として管理することが多かった。関係データベース環境でも、セマンティックキャッシュ[16]という技術に注目し、新たなクエリとキャッシュされたクエリの間の包含関係に基づいてキャッシュされたクエリとその結果をクエリ処理に用いる研究が行われた。しかし、それらを XML データ側でそのまま使うと良い結果が得られない。XML データは木構造で表され、入れ子の階層構造という特徴を持っている。さらに、その構造に適合されるクエリ言語が提供されているのでそれらに合わせるキャッシュ管理が要求される。ページやタプル単位管理の代わりにクエリを単位としてキャッシュ管理に用いることによって効率を高めると考えられる。しかし、XML データ構造の複雑性から様々な制約や問題点が生じる。

DTD から DTD グラフを抽出し、それに基づいて 2 つの XPath（新たなクエリとキャッシュされたクエリ）の間の包含関係を判定する研究[7]では、モバイルクライアントの側で DTD グラフとキャッシュを用いて新たなクエリとの比較を行ってその結果がキャッシュ側で求められるかに注目する。XPath をキャッシュのセグメントとして扱うが、それに基づくキャッシュの管理などについては具体的に述べてない。ACE-XQ[17]では、XQuery の一部に使われている XPath をキャッシュセグメントの単位として扱うことによってより精密なキャッシュ管理を行う。そして、‘XPathRow’という XPath の表現ごとにアクセス頻度や最近アクセス時間などの統計値を格納する。セグメントの更新が生じると該当する XPath の統計値に更新が起こる。新たなクエリによりキャッシュのスペースが要求されると、その統計値に基づいてセグメントの交換を行う。XQuery をまるごとにキャッシュの単位として使うこ

とにより生じる無駄なデータの重複を避けることができる。しかし、キャッシュされた XPath の間の意味的な関係を見逃すことによって他の問題が生じる。簡単な例を挙げながら説明を行なう。例えば、図 1 に対するキャッシュの内容の一部が表 1 であると仮定する。'/bib/book/author'というクエリが与えられて更新を行なうと、表 2 のような状態になる。

クエリ	最近アクセス時間	アクセスカウント
/bib/book/author/last	08:33am Feb 19	11
/bib/book/publisher	08:35am Feb 19	18
/bib/book/price	08:37am Feb 19	12
/bib/book/author/first	08:39am Feb 19	8

表 1. 更新前のキャッシュ状態

'/bib/book/author'というクエリが'/bib/book/author/last'と'/bib/book/author/first'を含んでいるのにもかかわらずそのまま挿入されて、無駄な重複によるスペースのコストが掛かってしまう。

クエリ	最近アクセス時間	アクセスカウント
/bib/book/author/last	08:33am Feb 19	11
/bib/book/publisher	08:35am Feb 19	18
/bib/book/price	08:37am Feb 19	12
/bib/book/author/first	08:39am Feb 19	8
/bib/book/author	08:43am Feb 19	1

表 2. 更新後のキャッシュ状態

今度は表 1 がキャッシュフル状態を表していると仮定する。再び、'/bib/book/title'というクエリが与えられるとセグメントの交換が要求される。最も参照されていないものを選択して入れ替えるので、その結果は表 3 のようになる。その後、再び、クエリ'/bib/book/author/first'が与えられると、表 3 のキャッシュでは対応ができなくなってしまう。

クエリ	最近アクセス時間	アクセスカウント
/bib/book/author/last	08:33am Feb 19	11
/bib/book/publisher	08:35am Feb 19	18
/bib/book/price	08:37am Feb 19	12
/bib/book/title	08:45am Feb 19	1

表 3. セグメント交換後の状態

それらは、'/bib/book/author/'と'/bib/book/author/first'、'/bib/book/author/last'の間に包含関係があるにもかかわらず、その意味的な関係を考慮していない結果、生じる損である。例えば、表 1 の場合、'/bib/book/author/last' と '/bib/book/author/first'が'/bib/book/author'に含まれているという意味的な関係を考慮して、2 つのアクセスカウントを合わせてみると、'/bib/book/price'のアクセスカウントより大きいのは確実である。本研究ではキャッシュの管理を行う際、そのセグメント（XPath）の間に意味的な関係に注目しその関係を反映させる新たなアクセスカウント - セマンティックアクセスカウントと呼ぶ - を適用する。表 1 のようなキャッシュ状態に'/bib/book/title'というクエリが与えられてセグメントの交換が要求されると、キャッシュビューテーブルでは、XPath ごとに対し、あらかじめ求められたセマンティックアクセスカウント（3.2 節で定義するλ値に基いて計算する）値を用いて'/bib/book/author/last'と'/bib/book/author/first'の間にセグメントの再統合が行い、代わりに'/bib/book/author'が格納される。そして、セマンティックアクセスカウントの値が最も小さいものは'/bib/book/price'に変わって、'/bib/book/title'と入れ替えることにより次のような結果になる。

クエリ	最近アクセス時間	セマンティックアクセスカウント
/bib/book/author	08:44am Feb19	19
/bib/book/publisher	08:35am Feb19	18
/bib/book/title	08:45am Feb19	1

表 4. クエリ間に包含関係を考慮した場合

ただし、'/bib/book/author'のセマンティックアクセスカウントの値(19 = 11 + 8)は、単純化して計算を行って正確ではない。λに関する計算手法について次の章で述べる。

3.2. 本研究のキャッシュ管理手法

キャッシュを管理するとき、最も単純な方法として、それぞれのクエリに対してページ単位で管理するものが考えられる。それより、発展された手法として、1 つのクエリに対して 1 つのキャッシュセグメントを対応させることがある。この手法はクエリに対する前処理やキャッシュに対する別の処理がいらないので管理することが簡単なのが長所である。新たなクエリに対してキャッシュの更新を行うときにはそのクエリの結果と同じ値を持つセグメントの属性である‘アクセスカウント’や‘最近アクセス時間’などの統計値を増

加させる。セグメントの交換は‘最近アクセス時間’や‘アクセスカウント(頻度)’をその基準にするLRU(Least Recently Used)やLFU(Least Frequently Used)[11]に従う。LRUとLFUは従来、キャッシュの管理を行う際、ページの交換における最も知られている管理アルゴリズムである。LRU(Least Recently Used)は最近参照されたものが近いうちにまた参照される可能性が高いという発想で、ページの交換のときに基準とする。その応用としてLRU approximationがある。LFU(Least Frequently Used)[11]は最も参照されていないものを削除するとき優先にするというアルゴリズムである。これらはその単純性と利用しやすいのでよく使われている。クエリに対してページ単位で管理する場合、表2のようにキャッシュのセグメント間には重なる部分が多くなる可能性があってキャッシュのスペースコストがかかってしまう。結局、ヒットレートも低下する。さらに、キャッシュのセグメント間には包含関係による意味的な関係があるにもかかわらず、それが失われてしまう。たとえば、セグメントの交換が要求されて以下のような2つのセグメントが候補になると、アクセスカウント情報だけによって削除される方が決められる。

- XPath1: /bib/book
- XPath2: /bib/book/author

XPath1が選択され削除を行った場合、キャッシュにはXPath2が残る。XPath1のサイズが大きいので削除によってスペースには余裕を持つが、今度XPath1が要求されると、そのキャッシュでは対応できなくなる。一方、XPath2が削除されXPath1が残った場合は、再びXPath2が要求されてもXPath1に対応するセグメントを通じてXPath2に対する答えを得ることができる。しかし、XPath1ように範囲が広くてサイズが大きいものだけキャッシュに置くとキャッシュのスペース問題が生じる可能性がある。むしろ、その影響によってヒットレートが下がる場合が生じる。そして、本研究では、図4のようにキャッシュビューテーブルを提案する。キャッシュビューテーブルはキャッシュのセグメントとマッピングを行う。1つのセグメントにおいて、そのクエリとクエリ内部のすべてをプリフィックスで表す。それらを基本単位として扱う。キャッシュビューテーブルの属性の1つとして、クエリレベルの差に従ってアクセス頻度を計算するセマンティックアクセスカウントを取り入れる。セマンティックアクセスカウントは既存のアクセスカウントにλによる計算値を加えて求められる。λを用いるとあるノードに対して、その子ノードや子孫ノードへアクセスに対するローカリティー を大体調べることができる。その値を基にしてキャッシュセグメントの分解や統合を行ってキャッシュを再構成する。

Prefix path	Last Access Time	Node Number	S. A. C	Pointer to Cache
/A/B/C	10:17am Feb 19	8	λ_0	
/A/B	-	20	λ_1	Null
/A	-	50	λ_2	Null

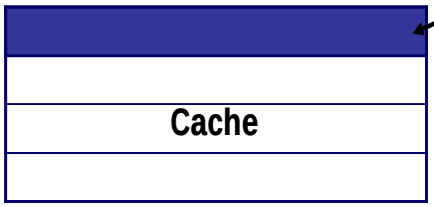

図4. キャッシュビューテーブル

図5を用いてλを求める式について説明する。図5はあるXMLデータから一部分を取って木構造で表してものである。λは以下のような式で表す。

$$\lambda_h = \lambda_0 \prod_{h=1}^{H} \beta_h \quad (h>0 \text{ 整数}) \qquad \beta_h = \frac{\text{ノード}(h-1)\text{と同種のノード数}}{\text{ノード}(h-1)\text{を含む兄弟ノード数}}$$

$$\lambda_0 = 1 \quad (h=0)$$

（XMLデータに対して）

λ_h :任意のセグメント(XPath)の中にある任意のノード(Nh)の値（N:ノード名）

λ_0 :任意のセグメント(XPath)の中にある端末ノードの値(=1)

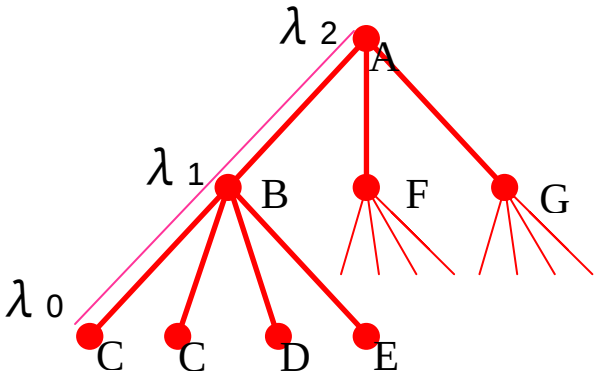

図5. XMLデータの一部分に対する木構造

λを用いると、あるセグメントに更新が生じるとき、それに含まれているすべてのプリフィックスパスにも個別にセマンティックアクセスカウントが加えられ、より精密な更新情報を得ることができる。キャッシュ

がプール状態であるとき、ある新たなクエリによりキャッシュセグメントの交換が要求されるとセマンティックアクセスカウント値が最も小さいのを選んで入れ替える。簡単な例をあけながら説明する。/A/B/C に対してキャッシュビューテーブルを以下のように表す。説明の便宜のために属性の一部だけを表す。

Prefix path	Last Access Time	Node Number	Semantic access count
/A	-	50	0
/A/B	-	20	0
/A/B/C	10:17am Feb 19	8	1

表５. /A/B/C に対するキャッシュビューテーブルの最初の状態

/A/B/C というクエリが与えられる場合、λによる計算が行われその値をそれぞれのプリフィックスに加えることによって更新を行う。

例)プリフィックス /A/B のセマンティックアクセスカウントの計算

λ１ = ２ / (2+1+1) * １ = 0.5

例)プリフィックス /A のセマンティックアクセスカウントの計算

λ２ = １ / (1+1+1) * 0.5 = 0.17

Prefix path	Last Access Time	Node Number	Semantic access count
/A	10:21am Feb 19	50	0.17
/A/B	10:21am Feb 19	20	0.5
/A/B/C	10:21am Feb 19	8	2

表６. /A/B/C による更新後の状態

それから/A/B/D というクエリが来ると､それぞれのセマンティックアクセスカウント値にλ0(/A/B/D)、λ1(/A/B)、λ2(/A)の計算値を加えて表７のように更新が行われる。

Prefix path	Last Access Time	Node Number	Semantic access count
/A	10:21am Feb 19	50	0.25
/A/B	10:21am Feb 19	20	0.75
/A/B/C	10:21am Feb 19	8	2
/A/B/D	10:31am Feb 19	4	1

表７. /A/B/D による更新後の状態

つまり、従来のアクセスカウントの代わりにλによるセマンティックアクセスカウントを取り入れるのは、あるノードがアクセスされるときそのノードだけではなくそれまで至るノードにもある程度意味を持つという発想で、意味的なアクセスを反映することである。セマンティックアクセスカウントはキャッシュのセグメント交換の基準値の１つであり、セグメント間の統合や分解にも用いられる。次にδ（ローカリティー の偏りの指標）に基づいたキャッシュセグメントの統合と分解によるキャッシュの再構成について述べる。δを求める式は以下のように定義される。

$$\delta ss' = S\text{の}S.A.C - S'\text{の}S.A.C \quad (\text{S' はSのプリフィクス})$$

キャッシュの再構成を行うのは制限されたキャッシュサイズという制約のうえ、ヒットレートが落ちない範囲でキャッシュスペースの利用率を高める最適化を目指すことである。本研究ではキャッシュを再構成する際、δを取り入れることでそれを実現する。δ値は上述のように、2 つのセマンティックアクセスカウント値の差で求められる。あるノードのδ値は、そのノードが属する親ノードや先祖ノードの範囲の中でそのノードへのアクセスの偏りを表す。δ値が高ければ高いほど、兄弟ノードへのアクセスよりそのノードへのアクセス頻度が多く、低ければ低いほど、親ノードや兄弟ノードへのアクセスが均等に分散されると考えられる。前述の場合、そのノードに対するローカリティー の偏りが強いのでそのノードをキャッシュするのが得であり、後述の場合は親ノードの以下を幅広くキャッシュした方が得になることが分かる。しかし、キャッシュサイズとのトレードオフが生じるので、それについては今後工夫が必要と考えられる。

4. 本研究のシステム構成

本研究で提案したキャッシュマネージャは次のようなモジュールで構成される。

- クエリ分析機：ユーザから与えられた XQuery が‘XQuery パーザー’を通じて、‘クエリ分析機’に伝えられると、‘キャッシュビュー’との比較を行い、そのクエリを remainder クエリと cached クエリに分解して‘クエリ Executor’に送る。
- クエリディスクリプター：XQuery に挿入されている XPath を抽出し、XPath からプリフィックスパスを抽出する。そして、それに対する統計値を‘キャッシュビューテーブル’に格納する。
- キャッシュ統合ビュー：物理的なキャッシュとの

マッピングを行い、実際キャッシュに格納されているクエリ、あるいはセグメントを木構造で提供する。クエリ分析を行ってそれに対してその答えがキャッシュで求められるかを判断する。

- キャッシュビューテーブル：キャッシュにある 1 つのセグメントに対して複数のプリフィックスを持つ。ポインタ、プリフィックスパス、セマンティックアクセスカウント、最近アクセス時間、サイズ、ノード数、応答遅延時間などの属性で構成される。
- Replacement マネージャ：λの計算によるセマンティックアクセスカウントの値に基づくセグメントの更新や交換を行う。さらに、δによるセグメントの再統合と再分解など、キャッシュの再構成について管理する。

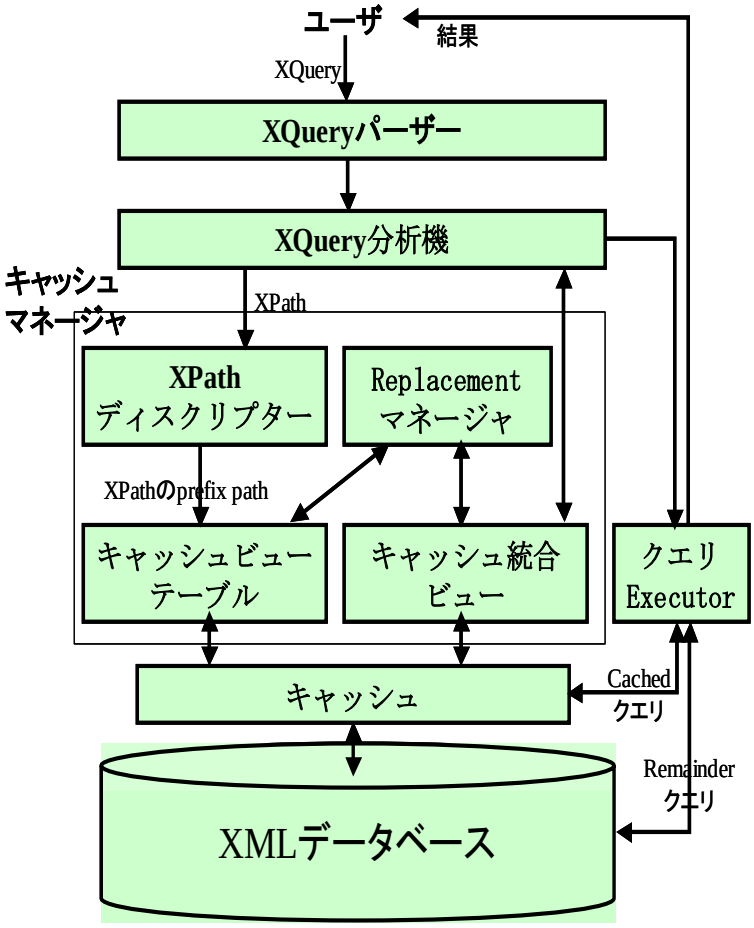

図 6. キャッシュマネージャの構成

5. まとめ

本研究では XML データに関するキャッシュ管理を行う際、そのデータ構造に適合するキャッシュ管理手法を提案した。λ値に基づいたセマンティックアクセスカウントを導入することによって、従来のアクセスカウントでキャッシュの更新を行うことより、精密なキャッシュ管理が予想される。さらに、クエリとキャッシュの関係やキャッシュセグメント同士の関係に意味的関係を生かして、セグメントの交換、再統合や再分配に用いるδを取り入れたが、実験を行って実際の環境による最も適切なδ値を求めることが重要な課題である。今後の予定として、様々なキャッシュサイズによる問い合わせ応答時間やヒットレートの測定を行い他のシステムと比較する。

文　献

[1] A. Berglund, S. Boag, D. Chemberlin, M. F. Fernandez, M. Kay, J. Robie, and J. Simon. XML path language(XPath)2.0 W3C working draft 16. Technical Report WD-xpath20-2002816, World Wide Web Consortium, Aug.2002.

[2] M. Carey, M. Franklin, and M. Zaharioudakis. Fine-grained Sharing in Page Server Databases. In Proceedings of 1994 ACM SIGMOD, Bombay, India, pages 359-370, June 1994.

[3] P. Rao and B. Moon. PRIX: Indexing And Querying XML Using Prufer Sequences. In ICDE, Boston, MA, U.S.A. pages 288-299, Mar. 2004.

[4] G. Miklau and D. Suciu. Containment and Equivalence for an XPath Fragment. In ACM PODS, Madison, Wisconsin, USA. June. 2002.

[5] D. DeWitt, P. Futtersack, D.Maier, and F. Velez. A Study of Three Alternative Workstation-Server Architectures For Object-Oriented Database Systems. In VLDB, Queensland, Australia, Pages 107-121, Aug. 1990.

[6] H. Wang, S. Park, W. Fan, and P. S. Yu. ViST: A Dynamic Index Method for Querying XML Data by Tree Structures. In Proceedings of the 2003 ACM-SIGMOD Conference, San Diego, CA, June. 2003

[7] S. Bottcher and R. Steinmetz. A DTD Graph Based XPath Query Subsumption Test. In Proceedings of 1st International XML Database Symposium, Xsym2003, Berlin, Germany, Sept. 2003.

[8] T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. Extensible markup language(XML)1.0 second edition W3C recommendation. Technical Report REC-xml-20001006, World Wide Web Consortium, Oct. 2000.

[9] S. Boag, D. Chemberlin, M. F. Fermandez, D. Florescu, J. Robie, and J. Simon. XQuery 1.0: An XML Query Language W3C working draft 16. Technical Report WD-xquery-20020816, World Wide Web Consortium, Aug. 2002.

[10] M. J. Carey and M. J. Franklin and M. Zaharioudakis. Fine-Grained Sharing in a Page Server OODBMS. In Proceedings of the 13th SIGMOD Conference, Minneapolis, Minnesota, pages 359-370, 1994.

[11] J. T. Robinson and M. V. Devarakonda. Data Cache Management Using Frequency-Based Replacement. In SIGMETRICS Conference on Measurement and

Modeling of Computer Systems, pages134-142.1990.

[12] M. Carey, M. Franklin, and M. Zaharioudakis. Fine-grained Sharing in Page Server Databases. In Proceedings of 1994 ACM SIGMOD, Bombay, India, pages 359-370, June 1994.

[13] Totsten Grust. Accelerating XPath Location Steps. In Proceedings of 2002 ACM SIGMOD, Madison, Wisconsin, USA, 109-120, June 2002.

[14] C. Zhang, J. Naughton, D. DeWitt, Q. Luo. On Supporting Containment Queries in Relational Databases. In Proceedings of 2001 ACM SIGMOD, Santa Babara, California, USA. 425-436, May 2001.

[15] Q. Li and B. Moon. Indexing and Querying XML Data for Regular Path Expression. In Proceedings of the 27th Int'l Conference on Very Large Databases(VLDB). Roma, Italy, 361-370.

[16] S. Dar, M. J. Franklin and B. Jonsson. Semantic Data Caching and Replacement. In Proceedings of the 20th Int'l Conference on Very Large Databases(VLDB). Bombay, India, 330-341.

[17] L. Chen, S. Wang and E. Cash. A Fine-Grained Replacement Strategy for XML Query Cache. WIDM'02, November, 2002, McLean, Virginia, USA.