G-04

メディアデータを含む半構造データの効率的な管理方法の検討

A study on the efficient management of semi-structured data with media data

朴 徳一†　　宝珍 輝尚†　　野宮 浩揮†
Deokil Park　　Teruhisa Hochin　　Hiroki Nomiya

1. はじめに

近年、テキストデータをベースとした半構造データが様々な環境で使用される中、Web やネットワーク技術の向上により扱われるデータ量も爆発的に増えている。中でも、Extensible Markup Language（XML）[1]は、World Wide Web Consortium（W3C）[2]でデータ交換の標準形式と定められてから、地図情報、計測データ等のデータ記述、論文やデジタルコンテンツ文章のコンテキスト記述をはじめ、多くのアプリケーションの中で用いられるようになっている。しかしながら、XML データは基本的にテキストの集合であり、その中から必要なデータを取得することは容易ではなく、その管理および検索には工夫が求められる。こうしたデータを効率良く利用するために、データベースに格納し管理したいという要求がある。

さらに、XML データのような半構造データはメディア情報の表現方法として活用されている一方で、テキスト以外のデータ、例えば画像をはじめとしたメディアデータをエンコードし半構造データの中に直接記述する場合も増えている。しかしながら、こうしたメディアデータを内包すると、データサイズの肥大化に繋がり、結果としてデータ挿入時間が長くなり管理コストが増大してしまう。

そこで本論文では、こうしたデータを対象とし、格納および検索の面から効率の良い管理方法の提案を目的として、データの構造的な特徴からその最適な分割やデータベースチューニングの考察を試みる。本稿では、大規模なデータを対象とするが、ひとまずデータベースへのデータ挿入や検索の特徴を把握するため、比較的小さなテストデータを作成して処理時間を計測し、その結果をもとに検討する。

以降、2.では、XML データの格納手法と半構造データとメディアデータ表現を説明する。3.では、この研究における背景と動機を述べ、4.では、テストデータによる予備実験と考察を述べる。最後に 5.でまとめる。

2. XML データと格納

2.1 XML データの格納手法

大規模な XML データ（XML 文書）が扱われる状況が増える中、必要な情報を効率よく取得するため、データベースに格納し必要に応じてデータの更新や検索を行いたいという要求がある。

XML データを関係データベースに格納する方法は、XML の構造指向（文書指向やデータ指向）を基に次の 3 つアプローチに大きく分けられる[3]。

(a) テーブルのタプルに属性値として文書全体を文字列として格納する方法

(b) XML データを要素単位で分割し、テーブルの属性値として格納する方法

(c) ネイティブな XML データベースに格納する方法

また、XML データを関係データベースに格納する方法として、格納アプローチ（b）はさらに構造写像アプローチとモデル写像アプローチに分けることができる[4]。

2.1.1 構造写像アプローチ

構造写像アプローチは XML データの論理的な構造（DTD や XML Schema）を表現する関係スキーマを定義する方法で、XML データのスキーマをテーブルの構造に反映させる（図 1）。

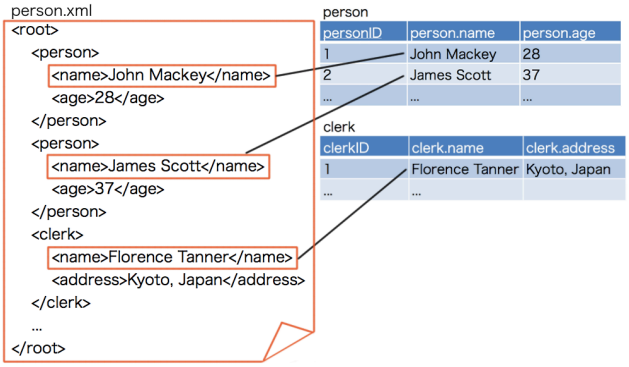

図 1 構造写像アプローチの例

このアプローチは少数の論理構造に従うものに有利であるが、XML データの論理構造が不明な整形式データや、高い頻度で論理構造が変更されるような XML データに対して不利である。

2.1.2 モデル写像アプローチ

モデル写像アプローチは、XML データの構成要素を表現する関係スキーマを定義する方法で、固定された関係スキーマを持つ 1 つ以上の関係テーブルで XML データの各要素を表現する方法である（図 2）。

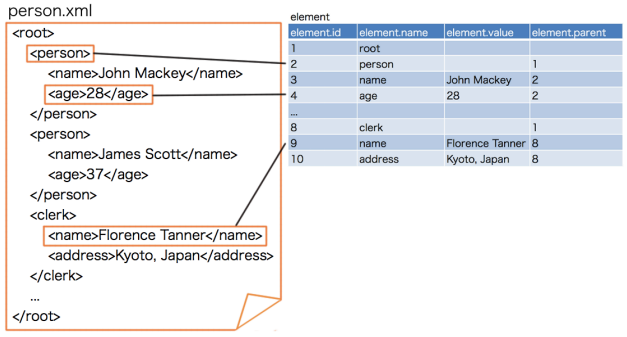

図 2 モデル写像アプローチの例

† 京都工芸繊維大学, Kyoto Institute of Technology

このアプローチは DTD や XML Schema 毎にスキーマを作成する必要がないため、XML データの扱いになれていないエンドユーザに対しても比較的扱いやすい利点がある。

2.2 半構造データとメディアデータ

マルチメディアデータを表現する一方法として、XML や SMIL、HTML といった半構造データが用いられている。特に、１）メディアデータ名やデータサイズをはじめとしたデータの概要を表す方法と、２）半構造データにリンクとして記述する方法がある。さらに２）に関しては、外部ファイルとしてリンクさせる方法と、半構造データに直接埋め込む方法の２つに大きく分けることができる。

2.2.1 外部ファイルとして参照する方法

通常、マルチメディアデータは外部データのリンクとして記述され、複数のファイルデータで構成されることが多い（図３）。

```
<author>
  <bio>This is my first photograph.</bio>
  <image src = “img.png”
       width = “100” height = “180”
         alt =  “2012/03/23” />
  <name>
    <person>John Mackey</person>
  </name>
</author>
```

図３ 外部ファイルの例

上記の例では、image 要素の属性として外部ファイルへのリンクや http リンクを記している。図３の例のようにメディアデータを外部ファイルとして記述すると、半構造データのデータサイズが小さくなり、可読性が高まる。一方で、HTTP 通信により Web サーバーにコンテンツを取得しにいくことになることから、メディアデータが多いような半構造データでは HTTP リクエストが増加し、ネットワークやサーバー負担につながることが指摘されている。

2.2.2 直接埋め込む方法

外部ファイルの例とは逆に、データ URI スキームを用いてメディアデータを半構造化データ内部に記述することができる（図４）。

```
<author>
  <bio>This is my first photograph.</bio>
  <image src = “
               SUhEUgAAABMAAAATCAYAAAByUDbMAAAABmJLR0QA
               ...
               6+RRxqe84wCQAwX2PZRPaNbt/marZsqjLxCNDaLN
               bfj0Zrvr0IRxcgrU0pN6YZwroPdCriaYFg3d8AAA
               AASUVORKSCYII=”
       width = “100” height = “180”
         alt =  “2012/03/23” />
  <name>
    <person>John Mackey</person>
  </name>
</author>
```

図４ 内部ファイルの例

データ URI スキームとは外部データなしに XML にデータを埋め込むための URI スキームのことであり、メディアデータを 64 種類の印字可能な英数字を用いた Base64 エンコードを用いて変換し格納する。外部リンクとしてメディアデータを記述する際、メディアデータが多いと複数の HTTP リクエストが発生するが，外部データとして埋め込むことで１つの HTTP リクエストで済み、効率化できる。また、メディアデータをテキスト形式データで保持することから、異種計算機環境でのデータ交換においても利点がある。しかしながら、メディアデータを直接書き込むのでデータサイズが増加し、ストレージ容量の圧迫やデータベース I/O などの処理速度の面で問題がある。

３．背景と動機

3.1 問題と解決

XML データが様々な環境で用いられる中で、天文学や核融合科学のような科学分野においては、実験での数値計測データを XML 形式で保持するのとは別に、測定時に観測された画像データ等のメディアデータも同時に保持、保存している。これらのデータを管理する上で、一元管理による管理そのものの手間の削減や、長期的な保存かつ異種環境でのデータ交換の際のテキストデータの利点を目的として、観測したメディアデータを XML データに直接埋め込んで管理する方法が採用されつつある。

しかしながら、そうしたデータが大量にかつ連続して生成される状況では、局所的なデータ量の大きさや構造的な特徴がデータベースへの挿入処理や検索処理時間に影響を与える。

本研究では、そうした問題に対して効率の良い格納管理方法を検討する。メディアデータを含んだ XML データを高速に格納し、欲しい情報を検索しユーザに提示できれば、問題は解決すると考える。

3.2 解決アプローチ

ここでは関係データベースに XML データを格納するアプローチを採用する。その理由として、

（1）現時点で、多くのシステムにおいて関係データベースが利用されていること
（2）大量のデータが既にそうした関係データベース上に蓄積されていること
（3）関係データベースのクエリ最適化やトランザクション管理など、これまでの多くの技術研究があること

以上のことから、関係データベースを用いた管理が最適であると考えた。

次に、メディアデータを格納する方法を考える。関係データベースに格納する方法は、2.1 節を参考に次の様に設定した。

（1）部分文書全体を属性値として挿入する方法
（2）部分文書のメディアデータ部分とそれ以外の部分を各々別テーブルに挿入する方法
（3）（2）において、部分文書だけ要素毎にテーブルにマッピングする方法

（3）に関しては、DTD や XML Schema による論理構造の指定がない整形式文書を考慮して、モデル写像アプローチを採用する。（3）のためのテーブル document のスキーマを表 1 に示す。また、（2），(3）においてメディアデータを格納するテーブル media のスキーマを表 2 に示す。

表 1 テーブル document のスキーマ

属性名	型	説明
doc_ID	Integer	unique object ID
element_ID	Integer	unique element ID
element_name	Text	element name
depth	Integer	hierarchial depth of element
attr_name	Text	attribute name
attr_value	Text	the value of attribute
doc	Text	the value of element

表 2 テーブル media のスキーマ

属性名	型	説明
doc_ID	Integer	unique object ID
media_ID	Integer	unique media ID
media_doc	Text	the content of media data

4． 予備実験

ここでは、小規模なデータで種々の格納方法における処理時間の特徴を把握する。

4.1 テストデータ

テストデータには、Wikipedia[5]が提供する jawiki-latest-abstract.xml を使用した。このファイルは Wikipedia のページのタイトルや概要、リンクを記した XML 部分文書が複数個記載されたものである。ページ単位の部分文書毎のひとまとまりでノード数にあまり変化がなく、かつ実ページにも画像が記載されているものがあるのでこのデータファイルを選択した。さらに、以下の設定でテストデータを作成した。

（1） 実ページに画像データが３つ以上記載されているページの部分文書をランダムに５つ選択
（2） 上記のページからデータサイズができるだけ異なる画像データをランダムに１～３個選択し、エンコードし、各部分文書に挿入
（3） ５つの部分文書を連結して１つの XML を生成

こうして生成されたデータをテストデータとした。データサイズは約１ MB である。

4.2 実験方法

XML データをデータベースに格納するに当たり、その性能を定量的に比較する上での比較指標として、次の３つを考える。

（1） 格納処理時間
（2） 特定部分文書の検索処理時間
（3） 部分文書の再構築処理時間

使用する関係データベースは、オープンソースで多機能なサポート、また汎用的に使用されていることを理由に PostgreSQL を選択した。

上述の XML データの格納方法を用いて、作成したテストデータを関係データベースへ挿入し、検索し、XML データを再構築する処理を Java を用いて実装し、処理時間を計測する。PostgreSQL への接続は JDBC を介して行い、XML データのパーサーは SAX[1] を用いる。これは、比較的データサイズが大きいものを扱うのでメモリ上に全てを展開できず、XML データを逐次処理していく必要があるためである。計測環境は表３の通りで、初回を除く３回の平均を算出する。

ここでは、１）XML データを読み込んで全ての XML データを走査し終わるまでの処理時間を格納処理時間として計測し、２）任意のキーワードを用いたパターン検索による部分文書の特定までの時間を特定部分文書の検索処理時間として計測し、３）特定したカラムから該当する XML データへの再構築処理時間を部分文書の再構築処理時間として計測している。また検索処理に関しては、現時点では各カラムに対して索引付けを行わず、入力したキーワードを用いて、文書部分に対して逐次的な全文検索処理を行っている。

表 3 実行計測環境

項目	値
CPU	Intel Core i5 1.6GHz
Memory	2GB
Storage	Apple ssd TS064C
RDB	PostgreSQL 9.3.4

表 4 挿入時間の比較

格納方法	挿入時間 [ms]
SubDoc	423.0
Fragment	354.3
Element	572.0

表 5 検索時間の比較

格納方法	検索時間 [ms]
SubDoc	28.0
Fragment	20.3
Element	20.7

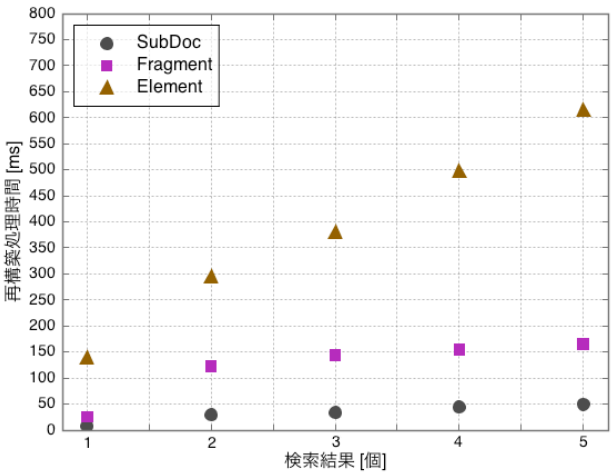

図 5 検索結果の個数毎の再構築処理時間

4.3 実験結果と考察

4.3.1 挿入ならびに再構築処理について

挿入時間、検索時間、ならびに再構築時間をそれぞれ、表４、表５、図５に示す。ここで、表４、表５、ならびに図５における SubDoc、Fragment、Element は格納方法を表しており、それぞれ、4.2 節における格納方法（1）から（3）に対応している。

[1]SAX : Simple API for XML. XML に対して逐次処理を行う.

表 4 における XML データの挿入処理時間をみると、Element が最も遅く、また Fragment が速い結果になっていることが分かる。Element に関しては、要素毎に関係テーブルへマッピングを行うのでその分解処理に時間がかかっていると考えられる。また、Fragment と SubDoc を比較すると、文書部分とメディアデータ部分に大きく分割しているにも関わらず、Fragment における挿入処理時間が速い結果となった。PostgreSQL における型 TEXT では一般的に可変長データをサポートしているが、1 つのカラムに対してサイズの大きな属性値が挿入されるとき、TOAST[2] 処理機構により、属性値が自動的に圧縮され行外テーブルに分割格納される。SubDoc 処理において 1 つのカラムに挿入する最大データサイズは Fragment における最大データサイズに比べて大きいので、この部分が挿入処理に影響していると考えられる。

一方、図 5 における検索結果毎の再構築処理をみると、Fragment 処理に比べて SubDoc 処理での再構築処理時間が速くなる結果となった。SubDoc における再構築処理は実質的にカラムから XML 部分文書をクエリにより取得する処理であるのに対して、Fragment はメディアデータ部分と文書部分を各々別テーブルから取得し結合しているので、その処理の差が再構築処理時間の差につながっていることの要因として考えられる。

これらの結果から処理時間をまとめると、以下となる。

- 挿入：Fragment ＜ SubDoc ＜ Element
- 再構築：SubDoc ＜ Fragment ＜ Element

また、テストデータの内容を考慮すると、以下の項目が各処理時間の差に繋がると考えられる。

（1） メディアデータの個数
（2） メディアデータのサイズ
（3） データサイズ（もしくは、部分文書内のノード数）

これらの点を踏まえて、再構築処理の優劣関係において、SubDoc に対して Fragment の再構築処理時間を速く、あるいは同程度になることを目指すよう最適な格納処理方法を模索する必要がある。

4.3.2 検索処理について

ここでは、再構築処理、ならびに検索処理時間に着目する。表 5 の検索処理時間をみると、Fragment と Element が同程度であり、SubDoc が最も遅い結果となった。現在は、格納した XML データに対して検索に対する索引付けは行っておらず、XML データ全体を逐次検索する処理を行い、部分文書を特定している。3.2 節で説明したように、Fragment と Element の処理では、部分文書情報を格納したテーブルとは別のテーブルにメディアデータを挿入している。これにより、XML データのタグ等の情報を除いて、検索走査する対象はほぼ同じになるため、両者の検索時間もほぼ同程度となる。これに対し、SubDoc では、メディアデータ自体も含めた部分文書をカラム走査しているため、検索処理時間が最も遅い結果となる。

また、本来想定している、キーワード検索からのユーザへの部分文書提示を考慮して、検索と再構築を合わせて一連の処理として考えた場合の処理時間を表 6 と図 6 に示す。表 6、図 6 では、メディアデータの個数や大きさの影響を無視するため、部分文書内のメディアデータは同一にして個数は 2 個に設定している。

表 6 より、検索結果が 1 件であった場合は、Fragment と SubDoc が同程度の処理時間となることが分かり、また、図 6 をみると、その後処理時間において差が徐々に開く傾向となる。これは、検索結果が増える毎に Fragment においてメディアデータがあった場所（タグ）の特定、および、再び部分文書中に挿入する処理における置換処理の呼び出し回数が多くなり、処理時間に影響していることが要因であると考えられる。

また、document テーブルの属性に対して索引付けを行うことを考えた場合、メディアデータを含まない分、Fragment では索引付けする対象が SubDoc よりも少なくなる。このため、今後、最適な索引付けを考えたとき、検索処理時間は同程度、あるいは、速くなると考えられる。

表 6 検索＋再構築処理時間 [ms]

	Element	Fragment	SubDoc
処理時間	119.7	45.7	45.3

※ 1 件の場合

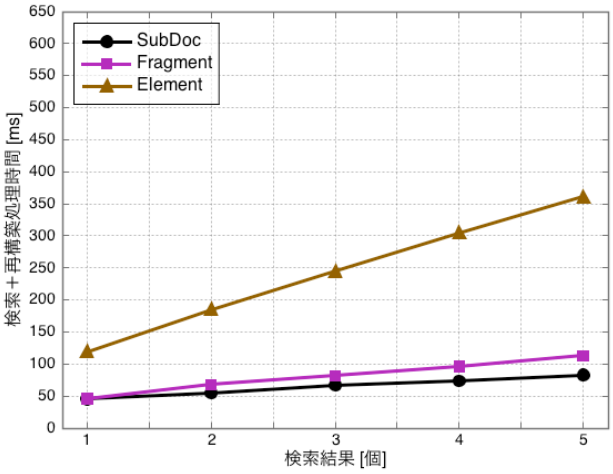

図 6 検索結果の個数毎の検索＋再構築処理時間

5. おわりに

本論文では、メディアデータをエンコードし XML データのような半構造データに直接内包させたデータに対して、関係データベースを用いた効率的な管理手法の提案を目指し、予備実験を行い、それに基づいて考察を行った。

SubDoc、Fragment、ならびに、Element という 3 手法について、挿入、検索、ならびに、再構築という観点で性能比較を行った。この結果、挿入では Fragment が最速となったものの、再構築では SubDoc が最速となった。

また、検索と再構築を一連の処理として考えたとき、検索結果が増える毎に徐々に差が開くことが分かり、規模の大きなデータを想定すると、この課題を解決しなければならない。

現時点では格納と再構築処理に注目し、検索における索引付け等は特に行っていない。今後は最適な索引付けを考えていく必要がある。さらにテストデータではなく、さらに大きなデータを用いたときの処理時間の特徴把握も行っていきたい。

6. 参考

[1] T. Bray, J. Paoli and others, “Extensible Markup

[2]TOAST : The Oversized-Attribute Storage Technique

Language (XML) 1.0 (Fifth Edition),"
http://www.w3.org/TR/xml/, Nov 2008, W3C Recommendation 26 November 2008.
[2] World Wide Web Consortium, http://www.w3.org/
[3] C. J. Date, "An Introduction to Database Systems," pp.925-927, Addison-Wesley, 2003.
[4] 油井誠，森嶋厚行：“PostgreSQL を用いた多機能な XML データベース環境の構築”, 情報処理学会論文誌, vol. 43, pp.11-22, 2003.
[5] Wikipedia Foundation，”Wikipedia,"
http://ja.wikipedia.org/