社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2004－DBS－134 (I) (29)
2004／7／13

SAX-GTR：高速XMLストリーム読み込み手法

横山 昌平† 太田 学† 片山 薫† 石川 博†

†東京都立大学大学院工学研究科 〒192-0397 東京都八王子市南大沢 1-1
E-mail: †{shohei,ohta,katayama,ishikawa}@hikendbs.eei.metro-u.ac.jp

あらまし アプリケーションからXML文書を扱う仕組みは大きく分けて二つある．一つはDOMに代表されるXML木をメモリ上に展開するタイプの手法であり，もう一つはSAX等，XMLファイルを先頭からデータストリームとして読み込んでゆくタイプの手法である．前者は柔軟な処理を，後者は高速・低消費メモリを特徴としている．本論文では後者の手法に着目し，さらなる高速化に関する手法の提案を行う．具体的には，XML文書をその文書をSAXのイベントから成るストリーム単位に分解し二つのバイナリファイルに分解して保存する．一つのファイルにはイベント列の定義と走査するプログラムによって頻繁に利用される要素属性値が保存され，それ以外のデータはもう一つのファイルに保存する．前者のファイルはXMLファイルを走査する時に読み込まれ，後者はそこに保存された値が利用される時に読み込まれる．この仕組みによりデータ走査時に読み込むデータの絶対量を削減する事ができ，より高速にXMLストリームを読み込むことができる．本稿ではこのバイナリファイルのフォーマットを説明する．またXMLデータを自動でバイナリ化し，ユーザが提案手法の知識がなくとも，SAXパーサを用いて透過的にアクセス手法を提案する．さらにSAXパーサより高速に動作する事を実験により示す．

キーワード XML，SAX，XMLストリーム，高速読み込み

SAX-GTR : Fast Loading of XML Stream

Shohei YOKOYAMA† Manabu OHTA† Kaoru KATAYAMA† and Hiroshi ISHIKAWA†

† Graduate School of Engineering, Tokyo Metropolitan University.
1-1 Minami-Osawa, Hachioji, Tokyo 192-0397, Japan
E-mail: †{shohei,ohta,katayama,ishikawa}@hikendbs.eei.metro-u.ac.jp

Abstract There are two basic types of XML parser: SAX and DOM. The SAX is an event driven XML parser that parses an XML input stream. The DOM is an in-memory tree representation of the XML document. Unlike SAX, DOM is a read-write API. Documents can be searched, queried, and updated via the DOM interface. This makes DOM much more effective when randomly accessing to XML documents. However, it is quite memory exhaustive compared with SAX because SAX doesn't have to store the entire document in memory. Therefore SAX is scalable from small to very large XML documents. The time taken to parse an XML document grows only linearly with the size of the document. In this paper, we propose an efficient framework for SAX applications called SAX-GTR that divides an XML document in two binary files; one is for the event stream and frequently-accessed data, the other is for non-frequently-accessed data. The division is decided automatically with a particular SAX event handler, without the structural and semantic information such as DTDs. The purpose of this division is to decrease the cost of loading. We also present a transparent access to SAX-GTR documents using SAX event handlers. The effectiveness of SAX-GTR is demonstrated by some experiments.

Keyword XML, SAX, XML Stream, Fast Loading

1. はじめに

XML フォーマットは登場と共に爆発的に普及し，様々な分野で用いられている．アプリケーションからXMLファイルを読み込むには大きく分けて二つの手法がある．一つはXML木をメインメモリ上に全て展開する方法で，アプリケーションはそのメモリ上の木に対してアクセスを行う．もう一つの手法はXMLファイルを先頭から読み込み，XML の構成要素，例えば開始タグや終了タグなどの出現をアプリケーションに伝える，ストリーミング型の手法である．前者は柔軟な処理が特徴であり，後者は高速動作・低消費メモリ量が特徴である．本稿ではこのストリーミング型の手法に着目し，処理の効率化について議論する．

XML は企業間のデータ交換に良く用いられているが，最近はデータベース用途としての役割も注目されてきている．つまり，より大きな XML ファイルを扱う必要性が増していると言える．例えば巨大な XML ファイルで表現されたデータベースへの検索を考える．ある問い合わせに対し XML ファイルの先頭から走査し結果を得る．この過程でXMLファイル中での結果の位置というのが検索時間に対して重要な影響を与えるため，XML ファイル読み込みの速度は，そのシステム全体のスケーラビリティを左右する要因になる．

我々はXML読み込み速度の高速化の研究に取り組んできた．XML 読み込み速度を決定する要因として一つにはXMLファイルのパースに要する時間がある．XML ファイルは内部的には文字のストリームであり，アプリケーションで扱うには文字ストリームを解析し，XML ストリームへと変換しなければならない．この処理コストがXML 読み込み速度に影響を与えている．この点に関しては議論が多く，例えばストリーム型のシステムである SAX パーサには読み込み効率を高めた幾つかの実装があり，また改良されている．

一方で我々のアプローチは，読み込むデータ全体の量を削減しファイル I/O にかかるコストを削減する事により高速化を達成している．両者の技術は補完しあう関係にある．

本稿で提案するシステムSAX-GTRはSAXパーサからの利用を想定している．SAX パーサはイベントドリブン型のXMLパーサと位置づけられているシステムで，XML ファイルを先頭から走査し，XML の構成要素を発見するとイベントという形でアプリケーションに通知する手法を採っている．そのためオンライン型の処理に向いており，アプリケーションは SAX パーサが定義するイベントハンドラインタフェースを実装し，SAX のイベントを受け取る．

提案手法ではXMLファイルをバイナリ化して保存している．XML のバイナリ化は人間にも機械にもreadableな形式として登場したXMLフォーマットの存在意義と矛盾する．そこで，提案システムでは SAX パーサとの互換及びバイナリ化を意識しない透過的なアクセス手法を提供している．また，バイナリ化はXML文書が提案手法で初めて読み込まれた時に自動で行われ，ユーザはバイナリ化の有無，或いはバイナリフォーマットに関する知識なしで，通常の SAX パーサに対するプログラムとしてアプリケーションを構築できる．

本稿の構成は次の通りである．続く第2章では提案手法と関連する研究について述べる．次に提案手法の説明として第3章でSAX-GTRのバイナリフォーマットについて述べ，第4章では java 環境で透過的アクセスを実現するための実装について述べる．また SAX パーサとの比較実験を行い，結果を第5章に示す．

2. 関連研究

XML データのバイナリ化に関して多くの研究がある．特にXML圧縮技術に関して議論が多い．例えば XMill[1]はスキーマにかかわらず XML を圧縮する技術で独自のバイナリファイルを作成する．その他にも圧縮の目的でXML文書をバイナリ化する手法は幾つか提案されている[2,3]．文献[2]はWBXMLフォーマット[4]を基に，SAX と DOM のサポートを提供しており，提案システムと類似した目的を持つ．しかし DTD が必要で．それを用いて要素名などをトークン化しており，パース時にトークン表をメモリ上に展開する必要がある．また一つのトークン表に格納できるト

ークンの数は 27 種類しかなく，複雑なスキーマを持つ XML を読み込むには，頻繁にトークン表の切り替えを行う必要があるなど，汎用性に欠ける．

また XML 木を二次記憶上に配置し永続化する技術も議論が多い．XML 木を一次記憶上で扱うには主に DOM を用いる事が多いが，ozone[5]ではこれを二次記憶上に保存し，それに対してアプリケーションは DOM を用いてアクセスする事ができる．また XML データベース eXcelon は DOM インタフェースを持っている．これらは木を基にした格納手法であり，XML ストリームを効率よく再現するのは困難である．

一方で，XML ストリーム処理に関する効率化の研究も多くなされている．それらの多くは XPath 等による問い合わせの効率処理を目的としている[6,7]．例えば文献[7]の XSQ は XML の読み込みに SAX を利用する．提案手法は SAX そのものを高速化するため，これらの技術と提案手法は利益を補完しあえると考えている．

SAX パーサの実装は多数ある．我々は現在 java で提案手法の実装を進めている．java で XML 処理を行うには JAXP(Java API for XML Parsing) を利用する．JAXP の特徴は『Plugability』である．SAX の実装は多数あると述べたが，java 環境ではこれらを動的に選択する事ができる．実装に関しては第 4 章で詳述する．

提案手法の性能を評価するためのテストデータを XMark プロジェクト[8]の xmlgen ツールを用いて作成した．XMark は XML データベースの性能を測定するためのベンチマークで，xmlgen は様々なサイズのテストデータを作り出すことができる．これらを用いた実験に関する詳細は第 5 章で述べる．

我々は XML ストリームに関連して， P2P 環境を想定した XML ストリーム統合のフレームワークについて研究を行ってきた．提案手法はこのフレームワークのストレージ技術とすることを想定している．また，XML のストレージとしては関係データベースを利用する手法も我々は提案している．このサーバ型の手法に対し提案手法はより軽量なシステムを目指し，利用機会に応じて使い分ける事を考えている．

3. SAX-GTR フォーマット

本章では続く 3.1.節で提案手法の前提事項として SAX を利用した XML 処理について記し，3.2.節で提案システムの全体像，3.3.節で提案手法によって作り出されるバイナリフォーマットの詳細について述べる．

3.1. 前提

提案手法はストリーミング型の XML パーサで巨大な XML 文書の処理を効率良く行う事を目的としている．ストリーミング型のパーサには大きく分けてイベントドリブン型とプル型とある．本稿ではイベントドリブン型のパーサ SAX との連携について述べる．XML のプル型パーサには XPP(XML Pull Parser)[9]がある．イベントドリブン型とプル型の違いはイベントの受け取り方法が能動的であるか受動的であるかの違いであり，対象とする XML ストリームの構成要素は同じである．よって提案手法はプル型のパーサにも適用可能であると考えるが，ここでは議論しない．

① SAX パーサを用いた XML 読み込み手順を簡単に記すと以下の様になる．

② SAX パーサオブジェトのインスタンスを作成

③ SAX パーサにイベントハンドラを登録

④ パース開始

⑤ 先頭から文書が終了するまで順に XML 構成要素をイベントとしてイベントハンドラに渡す

⑥ パース終了

イベントハンドラは XML の構成要素毎のイベントを受けるメソッドからなる．本稿では文書開始，要素開始，属性出現，文字列，要素終了，文書終了のイベントを扱う事とする．図 1 に SAX パーサを利用して XML を読み込む例を記す．

提案手法は SAX を高速化することで，スケーラビリティを高める事を目的としている．つまり大きなサイズの XML 文書を扱うシステムからの利用を主に想定している．XML フォーマットが利用されている分野は数多くあり，利用方法や利

用機会も様々である．提案手法は大きなXML文書に問い合わせをするという利用方法を特に考慮している．

XML 問い合わせに関しては，問い合わせ言語や問い合わせの効率化など議論が多いが，本稿の範囲を超えるため詳述しない．ただし，イベントハンドラの動作を端的に表す目的で XQuery を用いる．

例えば前述したXMLベンチマークプロジェクトXMarkで用いられるベンチマークの指標として用いられる問い合わせに次のようなクエリがある．

```
1: FOR $b IN document("auction.xml")
     /site/people/person[@id="person0
     "]
2: RETURN $b/name/text()
```

このクエリではauction.xmlというファイルの/site/people/parson 要素の id 属性の値がperson0の場合，そのparson要素に属するname要素の文字列を返すという事を表している．XMarkのテストデータには77種類の要素と14種類の属性が定義されている．しかし，この問い合わせを行う処理に於いて考慮されるべきは/site/people/parson/@idの値のみである．

我々はこの点に着目した．id属性以外の値を別のファイルとして保存し，実際に値が参照された時のみそのファイルからデータを読み込む事にすれば，ファイルI/Oは大幅に削減され，ファイル読み込み終了までの時間が大幅に削減できると考えた．

3.2. 概要

提案手法はXMLを独自のバイナリ形式ファイルに変換しそれを読み込むことによって，SAXパーサの動作速度を上げる手法である．ユーザにはXMLのバイナリ化やそのフォーマットの知識無しに透過的なアクセスを提供する．

SAX-GTR のバイナリフォーマットは二つのファイルからなる．XML のイベントストリーム及び頻繁にアクセスされる値で構成されるファイルと，その他の値を格納するファイルである．前者をDisplayファイル，後者をStockファイルと呼ぶ．

SAX-GTR パーサはこれらのファイルを走査し，オリジナルのXML文書と同じSAXイベントを発生させる(図2)．Stock ファイルに値が保存されているイベントは，イベントハンドラ内でその値が参照された時点で，SAX-GTR パーサは Stock ファイルにアクセスを行い，値を得る．Stock ファイルとDisplayファイルは別の二時記憶装置に保存すれば更なる効果が期待できる．

3.3. 詳細

本節では，Displayファイル及びStockファイルのフォーマットに関して述べる．

まず用語を定義する．提案手法ではXMLの構成要素である，要素開始，属性出現，文字列，要素終了のストリームを扱う．これをXMLストリームあるいはイベントストリームと呼ぶ．各イベントはそれぞれ要素名，{属性名,属性値}，文字列，要素名のプロパティを持っている．もしこれらがStockファイルに格納されている場合，各イベントは代わりにそれらへのポインタをプロパティとして持つ．SAX-GTR パーサは提案手法の実装系であり，SAXパーサと同じインタフェースを持つ．

SAX-GTRのフォーマットを次のXML文書を用

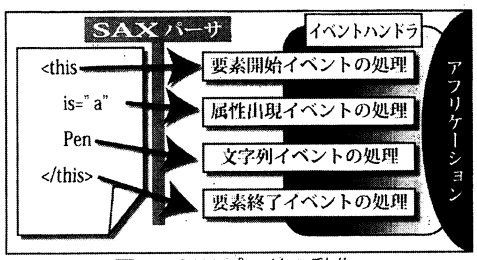

図1　SAXパーサの動作

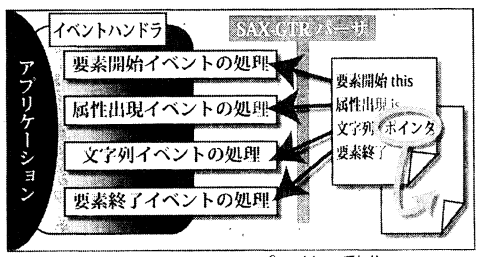

図2　SAX-GTRパーサの動作

いて説明する.

```
 3: <LIST>
 4:   <bottle material="芋">
 5:     森伊蔵
 6:   </bottle>
 7:   <bottle material="黒糖">
 8:     里の曙
 9:   </bottle>
10: </LIST>
```

このXMLで表現された焼酎のリストをイベントとそのプロパティに分解すると図3になる. 要素終了イベントのプロパティがNULLなのは, 直近の有効な要素開始イベントから予想できるためである. この手法ではパースの際に要素名をスタックに保存する等の処理が必要になるが, ファイルサイズを大幅に減少させ I/O コストの低減が期待できる.

図は可読性を高めるためイベントはアイコンで表されているが, 実際は凡例にある 3bit の値でイベントの種類を表す. さらにプロパティがDisplay ファイルにあるのか, Stock ファイルにあるのかをそれぞれ{0,1}と 1bit で表す. このイベント種類を表す 3bit とプロパティの位置を表す 1bit をあわせた 4bit のデータをイベントヘッダと呼ぶ.

プロパティは UTF-8 で保存される. これは可変長の値であるため最初の 2byte を使いプロパティの長さを byte 数で表現している. またプロパティが Stock ファイルへのポインタの場合, Stock ファイルの先頭からの位置を byte 数で表す. これには 8byte を使用する.

一つのイベントヘッダは 4bit である, Display ファイル中では 4byte のヘッダエリアを設け, 一つのヘッダエリアに 8 つのヘッダを格納する. ヘッダエリアに続いて, 各ヘッダに対応する 8 つのプロパティが配置される. ヘッダエリアには逆順でヘッダが格納されている. 上記のXML文書をSAX-GTRフォーマットで表現すると図4になる. 図のようにヘッダエリアとプロパティエリアが交互に並んでいる. ヘッダエリアにはイベント順とは逆順に 8 つのヘッダが並んでいる.

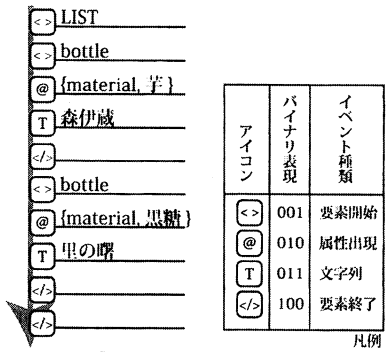

図3 XMLストリーム

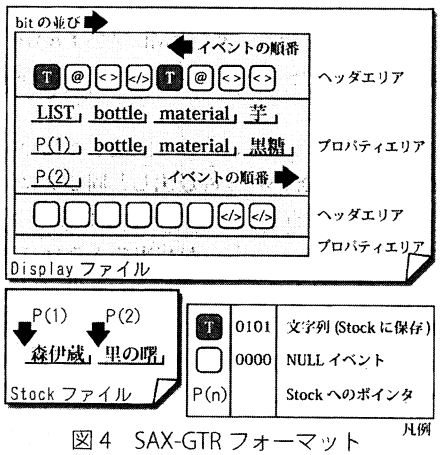

図4 SAX-GTR フォーマット

NULL イベントは何もしない事を表すイベントで, ヘッダエリアの余剰部分はそれで埋められる. SAX-GTR パーサは Display ファイルを先頭から読み込み, SAX イベントをイベントハンドラに渡す.

この例では/LIST/bottle/text()を Stock ファイルに配置したが, どのイベントのプロパティでも Stock ファイルに格納する事ができる. どの値を Stock ファイルに配置するかはあらかじめ指定する必要があるが, SAX との完全な透過性を維持するためにはこれを自動化する必要がある. その手法については実装環境との関連が深いため次章で説明を行う.

4. SAX-GTR 実装と利用

本節では SAX-GTR の実装に関して述べる. 続く 4.1.節では提案システムの透過的アクセスに

ついて述べる．4.2.節では SAX-GTR フォーマットの読み込み方法について記し，4.3.節ではイベントハンドラに関して説明する．4.4.節では XML 文書から最適な SAX-GTR フォーマットへの自動変換に関して議論を行う．

4.1. 提案システムの透過性

SAX パーサを用いて SAX-GTR フォーマットのデータへ透過的にアクセスするためにはバイナリ化ファイルを意識しないアクセスが必要である．提案手法ではあるイベントハンドラが初めて使われた時，XML ファイルを自動でバイナリ化する．それ以後はバイナリ化したファイルから SAX イベントを得る．また XML ファイルが更新された場合，新たにバイナリ化ファイルを再構成する．

この際，システムの透過性を実現するための要件として，①SAX パーサ呼び出しと同じプロシージャで提案手法が利用できなければならない，②SAX のイベントハンドラが利用できなければならない．③バイナリ化する際に Stock ファイルへ配置するイベントを動的に決定しなければならない．これらの項目について以後詳述する．

4.2. SAX-GTR パーサ

java から XML を扱うための API 群である JAXP が『Plugability』であることは前述した．以下は JAXP の SAX パーサを利用する一般的なコードである．

```
11: File xmlFile = new File("example.x
    ml");
12: SaxEventHandler handler = new SaxE
    ventHandler();
13: SAXParserFactory sf = new SAXGTRPa
    rserFactory();
14: SAXParser parser = sf.newSAXParse
    r();
15: parser.parse(xml, handler);
```

このように JAXP ではファクトリーメソッドと呼ばれる手法を用いている．プログラム内で直接利用するパーサのインスタンスを作成するのではなく，ファクトリーメソッドにインスタンスの作成を依頼しインスタンスを得ている．SAXParser は抽象クラスであり，SAX の各実装系が継承し独自のパーサを作成している．

この仕組みにより java では SAXParser を実行時に選択することができる．実行時に java ヴァーチャルマシンに指示するか，デフォルトで利用するパーサを java のプロパティファイルに記述する事ができる．提案手法の SAX-GTR パーサは JAXP の SAXParser に準拠しているため，既存の SAX プログラムを容易に提案手法へ移行させる事が可能である．

4.3. イベントハンドラ

イベントハンドラも SAX と非常に似た構成である．全く同じ構成にしていないのは幾つかの java の仕様に基づく制約からである．提案手法では Stock ファイルに格納されたデータはイベント発生時には読み込まれず，イベントのプロパティがハンドラによって参照された時点で読み込まれる．これはパーサが不必要なプロパティを読み飛ばすことによって I/O コストを低減させるためであるが，実現するにはプロパティの参照を検知し，動的に読み込みを行う必要がある．

文字列イベントのプロパティである文字列は SAX ではハンドラに char 型の配列として渡される．char 型はクラスではなくデータ型であり，java の言語仕様レベルで拡張を行わないとデータの参照を検知する事が出来ない．また，要素名などは String クラスとして渡されるが String クラスには final 修飾子が指定されており，拡張が許可されていない．これらの問題から我々は独自の String クラスを作成し，ハンドラではそれを利用している．例えば文字列イベントを受け取るメソッドは次のように記述される．

```
16: public void Characters(myString s
    tr) {
17:     System.out.println("文字列=" +
    str);
18: }
```

2 行目で str が参照されるとき，myString クラスの toString メソッドが呼ばれる．もしこのイベントのプロパティが Stock ファイルに配置されているならば，toString メソッドが呼び出され

たタイミングで Stock ファイルにアクセスして値を得る. またこのイベントの値が参照されなかった場合, Disk I/O は発生しない.

4.4. プロパティ配置の決定

SAX-GTR フォーマットに変換する際, イベントプロパティを Display ファイルに配置するか, Stock ファイルに配置するかを決定する必要がある. これには三つの方法がある.

文字列イベントのみを Stock ファイルに格納
値の参照状況に応じて動的に決定
XPath でユーザが指示

一番簡易的な手法は①であり, 柔軟な手法は③である. しかし③の手法はシステムの透過性が低くなってしまう. 以下②の手法について説明を行う.

前述の焼酎リスト XML ファイルを材料で検索するアプリケーションを考える. 例えば芋を検索キーとするなら, ハンドラは以下の問い合わせと同値である.

```
19: FOR $b IN document("example.xml")
      /LIST/bottle/name[@material ="芋
      "]
20: RETURN $b/name/text()
```

この問い合わせでは要素名と全ての /LIST/bottle/name/@material イベントが参照され, 条件にマッチした時のみ /LIST/bottle/name/text()の値が参照されている.

まずシステムは example.xml を SAX パーサで読み込み, 上記問い合わせを行うイベントハンドラへイベントを渡す. 前節で述べた手法により, イベントハンドラでどの値が参照されているかを検知し, それに基づきイベントの配置計画を立てる. この問い合わせの場合 /LIST/bottle/name/text()を Stock ファイルへ, その他を Display ファイルへ配置する. この間もユーザ定義のイベントハンドラは通常と同じくイベントを受信できるため, バイナリ化の処理は完全に隠蔽されている. このイベントハンドラを利用した次回以降のアクセスでは自動で SAX-GTR フォーマットのファイルを読み込むため, 処理速度が SAX での読み込みに比べ早くなる.

5. 実験

XML ファイルの読み込み速度を提案手法と SAX で比較した. 読み込みに用いたデータは XMark プロジェクトの xmlgen ツールを用いて 27KB から 830MB までの 8 つの文書を用意した. イベントハンドラはイベント数の数え上げのみを行っており, SAX と SAX-GTR 共に同じクラスを利用している. バイナリ化ファイルでは文字列イベントのみを Stock ファイルに配置している.

実験環境は OS に Windows XP を用い, CPU は Pentium4(1.5GHz), メモリは 640MB, ハードディスクは Maxtor 社製の Ultra ATA 100・

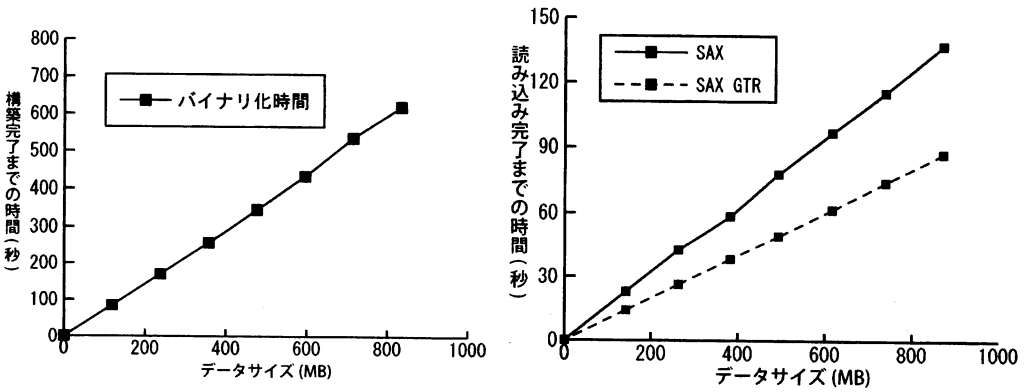

図 5 読み込み時間(左)とバイナリ化時間(右)

7200RPMのものをNTFSフォーマットで用いている．SAXパーサにはcrimsonを使用した．

実験手順は，8つのテストデータに対してSAXによるアクセス，SAX-GTRフォーマットへの変換，SAX-GTRによるアクセスの三つの項目に関して計時を行った．またこれらの実験結果を図5に示す．

バイナリ化にかかる時間が通常のSAX処理に比べ4倍かかっている．今回の実験ではDisplayファイルとStockファイルを同じハードディスク上に保存している，バイナリ化の時は同時に書き込みを行うため，時間がかかってしまったものと考えられる．バイナリ化の効率化は今後の課題としたい．

SAXとSAX-GTRによる読み込みでは，全域で処理時間を4割削減する事ができた．

この実験において，提案手法を用いる事により，SAXアプリケーションのXMLストリーム読み込み速度が改善されることを示した．また非常に大きなサイズのXML文書においても安定した性能が得られる事を示した．

6. まとめと今後の課題

本稿ではXMLストリームを高速に読み出すためのバイナリフォーマットを提案した．またバイナリ化をユーザから完全に隠蔽するための透過的アクセス手法を実装した．実験を行い，XMLストリームの読み込み時間をデータサイズにかかわらず4割程度削減できる事を示した．

今後の課題としては，動的にイベント配置を決定する最適なアルゴリズムの研究とバイナリ化のより高速な実装を考えている．また，イベントハンドラで複雑な処理を行ったときの性能を調査し，問い合わせの効率処理等の研究と統合していきたい．

謝辞

本研究の一部は，文部科学省科学研究費補助金特定領域研究(2)[情報学:A02](課題番号:16016273)による．

文　献

[1] Hartmut Liefke and Dan Suciu, "XMill: an efficient compressor for XML data," In Proceedings of the ACM SIGMOD International Conference on Management of Data, Dallas, Texas, May 2000.

[2] M. Girardot and N. Sundaresun, "Millau: An encoding format for e#cient representation and exchange of xml over the web," In Proceedings of the 9th WWW Conference, Amsterdam, Netherlands, May 2000.

[3] 大塚真吾，宮崎収兄，"二段階圧縮法のXMLへの適用，"情報処理学会データベースシステム研究会・電子情報通信学会データ工学研究専門委員会合同ワークショップ(DEWS2001)，函館，July 2001.

[4] Bruce Martin and Bashar Jano, "WAP Binary XML Content Format," W3C NOTE, June 1999, http://www.w3.org/TR/wbxml/.

[5] "Ozone," http://www.ozone-db.org/.

[6] T. J. Green, M. Onizuka, and D. Suciu, " Processing XML Streams with Deterministic Automata and Stream Indexes, " In Proceedings of IEEE Conference on Database Theory. 2003.

[7] Feng Peng and Sudarshan S. Chawathe, "XPath Queries on Streaming Data," In Proceedings of the ACM SIGMOD International Conference on Management of Data, San Diego, California, June 2003.

[8] A. R. Schmidt, F. Waas, M. L. Kersten, M. J. Carey, I. Manolescu, R. Busse, "XMark: A Benchmark for XML Data Management," In Proceedings of the International Conference on Very Large Data Bases (VLDB), pp 974-985, Hong Kong, China, August 2002.

[9] Home page of XML Pull Parser (XPP), http://www.extreme.indiana.edu/xgws/xsoap/xpp/.