XML 高速処理技術 SPlitDOM の機能拡張と評価

Enhancement and Evaluation of High-Performance XML Parser, SPlitDOM

中島 哲♥ 小田切 淳一♦
井谷 宣子◆ 吉田 茂♠

Satoshi NAKASHIMA Junichi ODAGIRI
Noriko ITANI Shigeru YOSHIDA

XML はメモリ消費や CPU 負荷が大きく性能が出しにくいという弱点がある。これを解決するため、我々は XML 処理を高速化する技術「SPlitDOM」を開発してきた。本稿では、SPlitDOM の改良と評価を行い、実用化に向けての課題であったアプリケーションへの導入容易性、及び実システムでの性能を確認した結果を報告する。

XML has the weak point where memory consumption and CPU load are large. In order to solve this problem, we have developed a technology named "SPlitDOM" which accelerates XML data processing. We have improved and evaluated SPlitDOM by applying it to a real application.

1. はじめに

近年、様々なソフトウェア開発において、XML が広く使われるようになってきている。XML は国際標準のデータ形式であり、今後はさらに幅広い分野に普及すると期待される[1][2][3]。しかしながら、XML はタグでデータを記述するため、従来の CSV 形式などと比べるとデータサイズが大きくなるため、処理時間やメモリ消費が弱点である。XML が今後、大規模・高トランザクション・システムにも適用される場合、性能が大きな問題になると予想される。

我々は、この弱点を解決する技術として、XML高速処理技術「SPlitDOM」を開発してきた[4],[5],[6]。実用化に向けたSPlitDOMの機能拡張を行い、Webアプリへの適用実験を通して効果を確認した。

2. 背景

XMLのデータ処理にはW3C標準のAPIであるDOM(Document Object Model)[7]が広く使われている。DOMは、XMLデータをメモリ上に展開する方式であり、アプリからは参照・更新処理が簡単にできるという利点がある。その反面、データ全体をメモリ展開するため、メモリ消費が大きく性能が出にくいという弱点がある。

別の標準 API として SAX(Simple API for XML)[8]がある。SAX は小メモリ消費で高速だが、XML データを読み込みながら処理する方式のため、単純な参照処理にしか適さない。

このため、一般に便利さからDOMが望まれるが、性能要件が厳しい環境では利用できないことが多い。その場合、SAXを使わざるを得ず、開発者の負担が大きい。この問題を解決する技術として、これまでに次の技術が提案されている。

- PDOM(Persistent DOM)[9]
  XML の解析情報をディスクに格納し、アクセスする部分をメモリ展開するキャッシュをもつことで、大容量でも小メモリで処理できる。ディスクベースのためランダムアクセスする用途は厳しいと考える。
- DDOM(Dictionary based DOM)[10]
  XML の解析情報をメモリ上に圧縮して保持し、DOM としてアクセスできる技術である。アクセス部分のみを復元するため小メモリで処理できるが、復元のオーバヘッドがあるため、処理速度を大きく上げるのは難しいと思われる。
- バイナリ XML 技術[11],[12],[13],[14]
  XML をバイナリ形式で表すことで、メモリ消費の削減や、データ処理速度を上げる。しかし、テキスト形式という XML の利点を損なう欠点がある。

これらに対し、広い分野で利用でき、高速で処理できる方式として、我々は SPlitDOM を開発してきた。

3. SPlitDOM の概要

実アプリでは XML の一部のみ処理することが多い。SPlitDOM は、これに注目し、XML の部分処理を対象として DOM のメモリ消費削減と処理高速化を行った技術である。図 1 のように、DOM は XML データ全体をメモリに展開するのに対し、SPlitDOM はアプリで必要な部分のみ展開する。

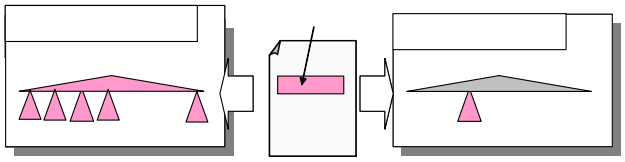

図 1 処理概要
Fig.1 Outline of Processing

図 2 に DOM と SPlitDOM のアーキテクチャを示す。双方とも、アプリから XML を渡し、DOM ツリーを結果として受け取る。この際、SPlitDOM では、設定ファイル内で指定された処理対象部分を XML から抽出し、それを DOM パーサに渡す。その解析結果がアプリに返る。アプリからは標準 API である JAXP(各社 DOM パーサを同一インタフェースで利用可能にする API)を介して処理する。これにより、アプリを基本的に変更せずに SPlitDOM を適用できる

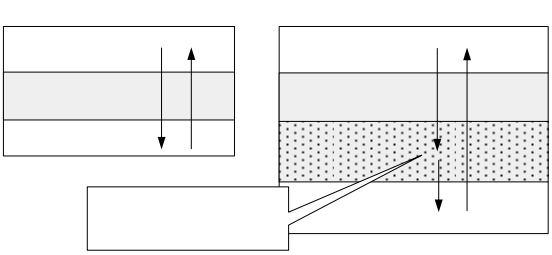

図 2 アーキテクチャ
Fig.2 Architecture

♥ 正会員 (株)富士通研究所s-nakasima@jp.fujitsu.com
♦ 非会員 (株)富士通研究所odagiri@jp.fujitsu.com
◆ 非会員 (株)富士通研究所ita2@jp.fujitsu.com
♠ 非会員 (株)富士通研究所yoshida.shig-04@jp.fujitsu.com

4. SPlitDOM の課題

① アプリへの適用容易化
一般にアプリでは、複数の箇所に DOM 処理が存在するため、DOM と SPlitDOM を各箇所で使い分けできる機能が必要となる(例えば全文検索には従来の DOM、特定タグの抽出には SPlitDOM を使用)。しかし、JAXP に単に準拠するだけでは後述の問題があり、このようなケースでアプリ適用が簡単ではなかった。

② 実システムを想定した性能評価
従来の SPlitDOM の評価は基礎性能が中心であったため、実システムを想定した性能確認が必要である。

5. SPlitDOM 選択適用方式

上記課題①を解決するために追加した新方式を示す。

5.1 目標

JAXP 準拠の上で、DOM と SPlitDOM の各パーサを選択・利用できる仕組みを目指す。図 3 に利用イメージを示す。

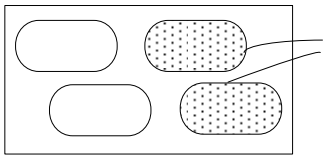

図 3 パーサ選択
Fig.3 Parser Selection

5.2 実現上の問題点

目標実現には、JAXP 準拠の上で以下の解決が必要である。

(1) 複数種の DOM パーサの混在利用
JAXP は利用パーサを JavaVM 全体で１つ指定するものであり、同時に複数を指定して使い分けることをアプリ無変更で行うことができない。

(2) SPlitDOM を利用する箇所の選択
ユーザがアプリ中の DOM 処理箇所を個別に指定できる必要がある。JAXP でなく専用の API を設ければ実現は容易だが、その場合、アプリを変更する必要が生じる。

5.3 実現方式

上記問題を解決する方式を導入した。 図 4 にアーキテクチャを示す。従来の SPlitDOM パーサの上位レイヤに、DOM パーサと SPlitDOM パーサを選択し利用する機構を設けた。これにより、JAXP を介してアプリから入力される XML データは、後述の方式により指定されたパーサに渡される。

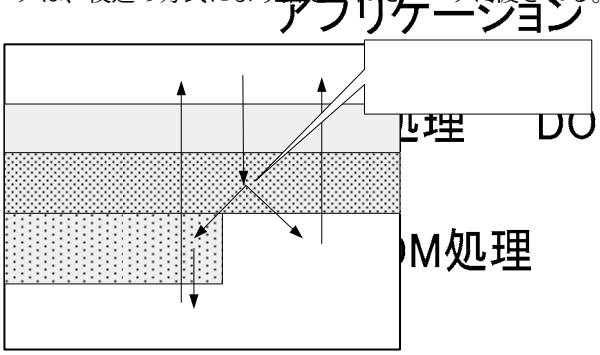

図 4 新型のアーキテクチャ
Fig.4 New Architecture

本アーキテクチャにより、上記問題点を次のように解決した。

(1) 複数種の DOM パーサ混在利用
選択機能をもつ FactoryWrapper を JAXP の実装クラスとして追加した。本クラスは利用パーサを表す DocumentBuilderFactory へのラッパーとして、次のように機能する。(a)後述の方式で SPlitDOM の使用・不使用を判断し、(b)使用の場合は SPlitDOM、そうでなければ DOM の DocumentBuilderFactory を生成し保持し、(c)それ以降の FactoryWrapper に対するメソッド呼出しは、保持する Factory に委譲する(図 5)。

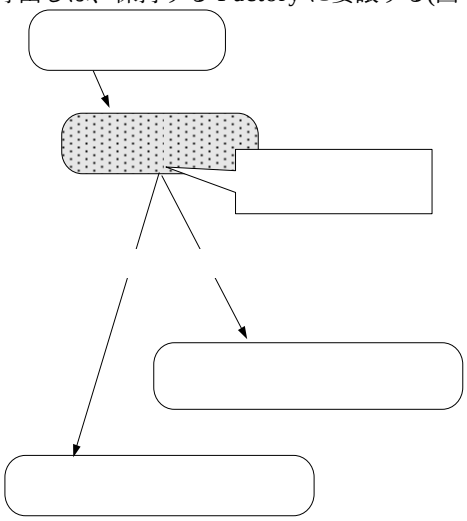

図 5 FactoryWrapper の構造
Fig.5 Structure of FactoryWrapper

(2) SPlitDOM を利用する箇所の選択

- 呼出し元メソッド名による指定
ユーザは事前に設定ファイルにメソッド名を記述しておく。例えば、アプリのクラス C1 にメソッド method1 があり、その中で DOM 処理を行っているとする。これに SPlitDOM を適用したい場合には、設定ファイルに「C1.method1」と記述する。
- 上記指定に基づく DOM／SPlitDOM 切換え
実行時に FactoryWrapper の newDocumentBuilder では、呼出し元のメソッド名を、Java の例外処理のスタックトレース情報を利用して取得する。メソッド名と設定ファイルでの指定を照合し、同一であれば、SPlitDOM の DocumentBuilder を、そうでなければ DOM の DocumentBuilder を生成する。

以上の仕組みによる、利用例を図 6 に示す。この例では、アプリ中に 2箇所ある DOM 処理のうち、1箇所(メソッド method1)で SPlitDOM を利用する。この場合、図 6 のように設定ファイルで method1 を指定することにより、SPlitDOM が method1 内の DOM 処理に適用される。

特定のDOM処理を選択し、アプリ無変更でSPlitDOMを適用

6. Web アプリでの適用評価

前記の機能追加を行った SPlitDOM を、検証用に作成した Web アプリに適用した。これを通して、課題であったアプリへの導入容易化、及び実システムを想定した多重性能について、それぞれ評価を行った。

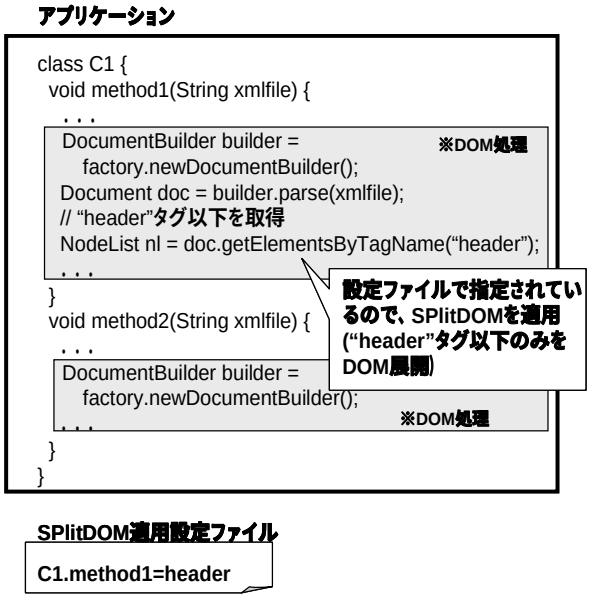

図 6 SPlitDOM の適用例
Fig.6 Example of Application Code

6.1 対象アプリ

PC カタログの検索アプリを対象とした。本アプリは、検証用に構築したものであり、Web サーバと Servlet コンテナ、RDB からなる Web アプリとして典型的な構成をもつ(図 7)。Web ブラウザから入力された商品型名や商品名をキーとして、RDB に格納された PC カタログを JDBC を介して検索し、検索結果を HTML で返す。

図 8 は、本アプリで使用する XML データの一部を示したものである。全体は CatalogHeader タグと CatalogBody タグで表されたヘッダ部とボディ部からなる。ヘッダ部には、その XML データが表す PC の基本情報が記述される。ボディ部には、その PC の詳細情報が記述される。

本アプリでは、RDB の BLOB(Binary Large Object)型カラムに XML データ全体を格納し、商品 ID など検索キーとなるデータ項目は通常のカラムにも格納している。

6.2 アプリへの SPlitDOM 適用

本アプリでは、PC カタログのうち Web で表示するのは、PC の基本情報(仕様概要)だけである。このため、XML データの参照個所は CatalogHeader タグ内のみであり、この抽出に SPlitDOM を適用した。既存アプリに適用する際、一般に以下の手続きが必要になる。

(1) 適用対象とする XML の部分処理の抽出
(2) SPlitDOM を利用する設定

6.3 評価１:SPlitDOM 導入コスト

6.3.1 評価方法

(2)の設定は数行記述するだけと単純のため、(1)に要したコストが SPlitDOM の導入コストに相当する。このコストを評価する指標として以下を用いた。

指標１：適用個所判断のために調べたソースコードの行数。
指標２：適用のために変更したソースコードの行数。

6.3.2 評価結果

以下に収集したデータを示す。アプリは、Java、JSP、XSL(XSLT の変換定義)のファイルからなる。それぞれについて、全体コード行数と各指標に該当する行数を比較する。

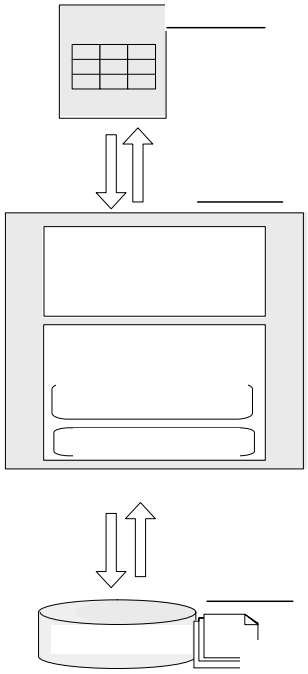

図 7 システム構成
Fig.7 System Structure

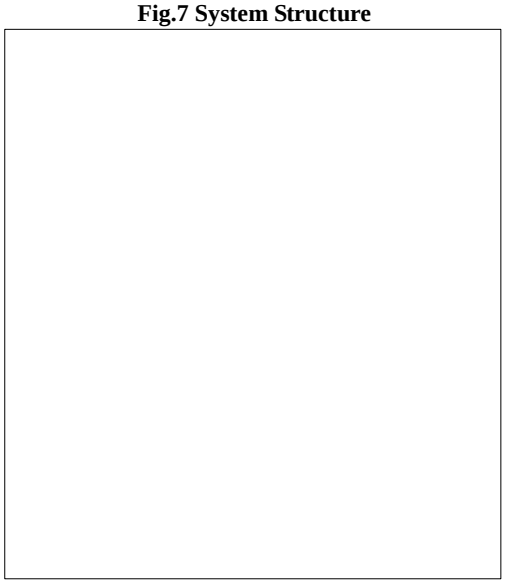

図 8 XML データの例
Fig.8 Example of XML data

表 1 ソースコード行数
Table 1 Number of Lines of Source Code

ファイル種別	コード行数	指標1	指標2
Java	736	87	0
JSP	426	0	0
XSL	35	35	0
全体	1197	122	0

指標 1 の結果は、SPlitDOM 適用の判断のために、Java ファイルと XSL ファイルを 122 行を調べたことを意味する。全体から見ると 1 割弱である。実際の作業も DOM 処理を含

む Java ファイル一つを目視確認した程度であり、少ない工数で済んだといえる。 指標 2 の結果は、SPlitDOM の適用においてアプリは無変更で済んだことを示す。

6.4 評価２:性能改善効果

次に、SPlitDOM の適用による性能改善の効果を示す。

6.4.1 評価方法

アプリに SPlitDOM を適用する前と後のそれぞれに対し、負荷テストツール[15]を用いて、擬似的に多数のユーザからアクセスする状況を作り測定した。ユーザ数(多重度)が 5,10,50 の３パターンで行った。ここで対象とした XML データのサイズは 50KB 強(ヘッダ部は約 2 割)である。なお、本測定で用いたマシン環境は次の通りである。

サーバ：Pentium4, 1.7GB, 1CPU, メモリ 512MB, Windows2000 Professional
クライアント：同上、ネットワーク：100BASE-T
ソフト:Apache2.0、Tomcat4.1、MySQL4.0、JDK1.4、XML パーサ Crimson

6.4.2 評価結果

図 9 にアプリ全体の多重処理性能(スループット)を示す。

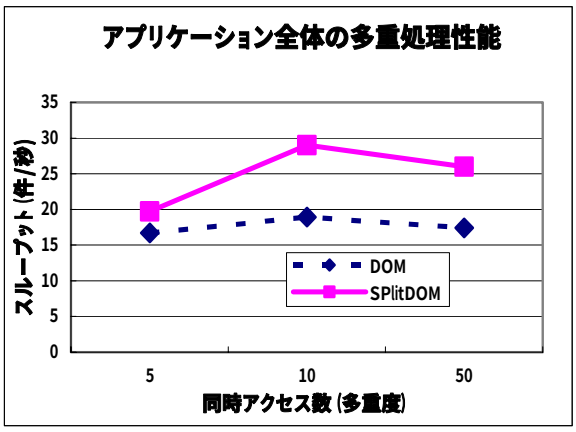

図 9 アプリの多重処理性能
Fig.9 Application Performance

本アプリに SPlitDOM の適用により、５多重時に全体で 1.17 倍、10 多重,50 多重時に全体で約 1.5 倍に向上し、有効な性能が出ることを確認できた。5 多重時に効果が小さいのは、XML 処理がボトルネックになっていないためと思われる。10 多重,50 多重では XML 処理がボトルネックになることにより、SPlitDOM の効果が大きく出ていると考えられる。

一般に、システム構築では高負荷になる業務ピーク時に十分耐えられることを考慮して、性能目標値を設定し、サーバマシンなどの見積りを行う。性能が 1.5 倍向上すれば、その分、マシンのスペック、台数を低く押させることができ、従来よりも低コストでシステムが構築できることを意味する。

本アプリでは、通信や DB アクセスなど様々な処理を含み、DOM 処理はその一つである。DOM 処理に要する時間は数十ミリ秒程度であり、通常は性能が問題にはならない。しかし、高多重処理時は、DOM 処理が多数同時に実行するため負荷が高くなる。この状況で、DOM 処理性能を改善したことが、全体の性能向上につながったと考えられる。

7. まとめ

SPlitDOM の実用化においては、アプリへの導入容易化、及び実システムを想定した性能評価が課題であった。これに対し、導入容易化のための機能拡張と、Web アプリへの適用を通した導入コストと性能の評価を行い、課題を解決した。今回の性能評価は、典型的な構成の Web アプリ 1 件で行った。今後は他にもアプリを増やして効果確認を行う。

[文献]

[1] 岡部恵造監修, “XML ビジネス白書 2003”, 翔泳社, 2003
[2] 岡部恵造監修, “XML ビジネス白書 2004”, 翔泳社,2004
[3] JP Morgenthal, Bill la Forge, "Enterprise Application Integration With XML and Java", PRENTICE HALL,2000
[4] 井谷宣子, 吉田茂, “大容量 XML 文書に対する XML パーサの処理性能改善の検討 2, 電子情報通信学会ソサイエティ大会, 2002.
[5] 小田切淳一, 井谷宣子, 吉田茂, “大容量 XML 文書に対するデータ処理効率化手法の評価”, 電子情報通信学会ソサイエティ大会, 2003.
[6] SPlitDOM,富士通 XML 技術紹介サイト, URL: http://xml.fujitsu.com/jp/tech/split/index.html
[7] Document Object Model, URL: http://www.w3.org/DOM/
[8] Simple API for XML , URL: http://www.saxproject.org/
[9] Persistent(DOM), InfoNyte, URL:http://www.infonyte.com/en/prod_pdom.html
[10] Mathias Neumuller, John N.Wilson, “Compact In-Memory Representation of XML – Design and Implementation of a compressed DOM for data-centric documents”, “Internal Report”, University of Starathclyde”,2002.
[11] Marc Girardot, Neel Sundaresan,”Millau: an encoding format for efficient representation and exchange of XML over the Web”, 9th International World Wide Web Conference.
[12] Liefke, H. and Suciu, D.,”XMILL: An Efficient Compressor for XML Data”, SIGMOD, 2000.
[13] “WAP Binary XML Content Format”, June 1999. URL: http://www.w3.org/TR/wbxml.
[14] Binary XML: Position paper for The W3C Workshop on Binary Interchange of XML Information Item Sets.
[15] E-SUP APTest/Web, URL:Test/Webhttp://www.fasol.fujitsu.com/services/esup/apt/index.html

中島 哲 Satoshi NAKASHIMA
1997 年 筑波大学理工学研究科卒、同年（株）富士通研究所入社、現在に至る。ソフト開発技術、XML データ処理の研究に従事。日本データベース学会、情報処理学会各会員。

小田切 淳一 Junichi ODAGIRI
1998 年 東京大学大学院工学系研究科卒、同年（株）富士通研究所入社、現在に至る。画像処理, データ圧縮および XML データ処理の研究に従事。電子情報通信学会会員。

井谷 宣子 Noriko ITANI
1992 年 大分大学工学部組織工学科卒、同年（株）富士通研究所入社、現在に至る。主として SLCA などロスレス圧縮関連の研究に従事。

吉田 茂 Shigeru YOSHIDA
1975 年 岩手大学工学部電子工学専攻修士卒, 同年(株)富士通研究所入社, 以降, ２値画像圧縮, 静止画像圧縮, ロスレス圧縮, XML データ圧縮の各研究に従事し, 現在に至る。電子情報通信学会, 画像電子学会各会員。