4L-03

マルチXMLストリーム向けの複合イベント処理システム

松田 達希† 藤田 悟‡

法政大学大学院 情報科学研究科情報科学専攻† 法政大学 情報科学部‡

1. まえがき

近年, ストリームデータの分析手法の 1 つである複合イベント処理(CEP)システムの研究が盛んに行われている. その中でも我々は XML ストリーム処理に注目し研究を行ってきた. 既存研究として, XML 向け問い合わせ言語である XPath を XML ストリーム向けに拡張した XSeq が提案されている[1]. XSeq は処理エンジンに Visibly Pushdown Automaton(VPA)[2]を利用し, 時系列を踏まえた複雑な検索要求も高速に処理することができるものの, 処理対象は単一ストリームのみとなっている.

本研究では, 複数の XML ストリームに対して高速な検索を行うシステムの構築を目指す. これまでに, CEP システムにおいて必要とされる処理のモデル化や, モデルに沿った XML ストリーム向けの問い合わせ言語:QLMXS の設計, 処理エンジンの開発を行ってきた[3][4].

本稿では, 時系列処理及び複数ストリーム処理におけるモデルに関して計算量を踏まえた再考を行う. また, これまでのシステムにおいて処理性能のボトルネックとなっていた XML のパーサを変更し, システムの再構成及び性能向上を図る. システムの性能に関しては, 実際の株式市場で使用されたデータを用いてシミュレーション実験を行い, 性能評価を行う.

2. QLMXS

2.1. マルチストリーム処理のモデル化

シングル及びマルチストリームに対する分析処理を, 代数表現を用いてモデル化を行う. モデル化する処理は, ストリームデータに対するフィルタリング(Filtering), 複数ストリーム同士の合成(Combination), 複数ストリーム中データの合成(Activation), ストリーム中のデータの細分化(Decomposition), 単一ストリームから複数ストリームへの分割(Partitioning)の 5 つとする. ストリームを *s*, フィルタを *f*, ストリーム中のデータ数を *n*, クエリ数を *q* として, それぞれのモデル及び計算量を表 1 に示す.

Activation に関しては, 一方のストリームデータともう一方のストリームデータとの比較回数によって計算量が大きく変わるため, 各データが一度だけ他のストリーム中のデータと比較を行う場合と, 各データが他のストリーム中の一定期間のデータと比較を行う場合に分けて定義した.

表 1 代数モデルと計算量

	代数モデル	計算量
Filtering	*s	f >> s'*	*O(n*q)*
Combination	*s + s' >> s''*	*O((n+n')*q)*
Activation	*s * s' >> s''*	*O((n+n')*q)*
Activation(term)	s*t(s') >> *s''*	*O(n*n'*q)*
Decomposition	*s / key >> s'*	*O(n*q)*
Partitioning	*s[key] >> s'[]*	*O(n*q)*

2.2. 問い合わせ言語

QLMXS は XPath をマルチストリーム処理向けに拡張し, SQL ライクな記述を可能にしつつ, 2.1 節のモデルに沿うように設計した言語である.

例として, 株式売買システムより各時刻における各企業の株式売買データが XML として一つ一つ流れてくるような状況を想定し, 各 XML に対するクエリの例を示していく.

```
Ex1.  return  StockStream4768
      select  *
      from    TestStream
      where   /StockRecord[StockCode=4768]
```

Ex1 は, 各 XML から銘柄番号が 4768 であるものを抽出するクエリである. return 節に出力先ストリーム, select 節に出力フォーマット, from 節に入力ストリーム, where 節にフィルタ条件をそれぞれ記述する.

```
Ex2.  return  StockStream4768_2
      select  *
      from    StockStream4768
      where   /StockRecord/Quotes/Quote/Price
                < $N/StockRecord/Quotes/Quote/Price
      setting $counter = 0
      processing  $counter = $counter + 1
      until   $counter >= 3
```

Ex2 は, Ex1 で抽出された XML に対して, 価格の値が 3 回連続で上昇した瞬間を検出し, そのときの XML を抽出するクエリである. where 節では, $N を用いることで次に来る XML を参照してい

Complex Event Processing System for Multiple XML Streams
† Tatsuki Matsuda, Graduate School of Computer and Information Sciences, Hosei University
‡ Satoru Fujita, Faculty of Computer and Information Sciences, Hosei University

る. setting 節では変数宣言を, processing 節では where 節の条件の成立時に実行される処理を, until 節にはループの終了条件を記述することがまた, 複数ストリームを指定するための chaining 節や処理間隔の限界時間を指定するための while 節等も利用することが可能である.

3. システム概要

システムの構成を図 1 に示す. 各ストリームから到着した XML はキューに格納され, 順次処理されていく. 今回, XML パーサには VTD パーサを用いている. VTD は DOM 同様に XML 全体を読み込んでから処理を行うパーサであり, DOM や SAX よりも高速な処理が可能である. QLMXS クエリは, クエリ中の XPath の評価結果を遷移シンボルとするオートマトンへと解釈される. Ex1 と Ex2 から生成されるオートマトンの例を図 2 に示す. オートマトン上を動くトークンは, 遷移に必要となる XPath の評価結果を VTD から受け取り遷移していく. トークンが最終状態に到達した時点で, 各出力先に出力されていく. Ex2 のように until 節による条件がある場合は, 遷移後に条件判定を行う.

4. 性能実験と考察

実際の株式市場で使用されたデータを用いてシミュレーション実験を行い, 本システムの性能を検証する. 実験環境を表 2 に示す. 実行するクエリは前述した Ex1 とする.

実験結果を図 3 に示す. 処理時間は処理件数に比例して増大しており, 50 万件の時には秒間約 26000 件の XML を処理することができた.

これまでの著者らの研究では, XML のパーサとして StAX パーサを用いており, 複数ストリームに対する単純検索に関して, 秒間 1000 件程度の処理性能であった. 今回提案したシステムでは, VTD を用いたことにより, 処理性能のボトルネックとなっていた XML のパースの高速な実行が可能になり, 大幅な性能向上に繋がっている. また, XML のパース及び XPath の評価を VTD が担うような構成にしたことによって, パーサの切り替えによる他のデータ形式への対応も容易に行えるようなシステムを構築することができた.

5. まとめ

本稿では, マルチストリーム処理のモデル化の再考及びシステムの再構成を行った. Activation モデルに関して, 期間データとの比較に関して新たにモデルを定義することで, より網羅的なモデルの定義が出来た. また, システムの構成を VTD

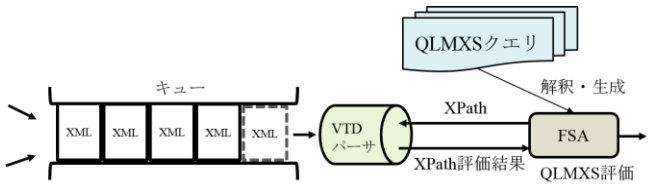

図 1　システム構成

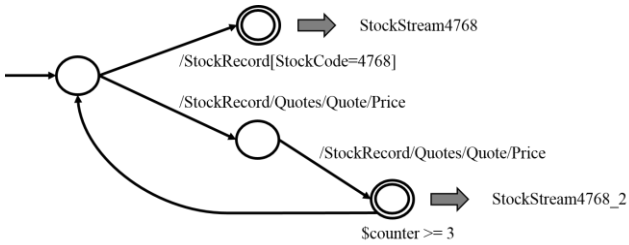

図 2　オートマトン生成例

表 2　実験環境

CPU	Intel® Core™ i5-4300U
Memory	8GB
XML	東証株式市場 2010 年 1 月 4 日

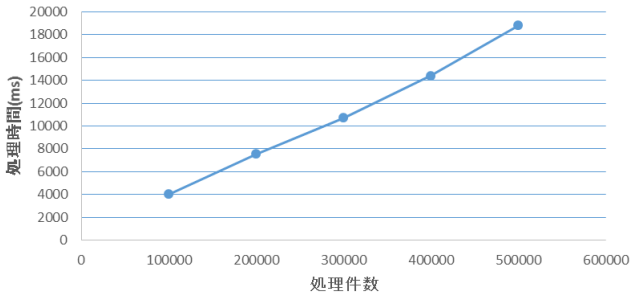

図 3　実験結果

パーサを利用する構成に変えたことで, 大幅な処理速度の向上を実現した.

今後の課題として, クエリから生成されるオートマトンの最適化や, 他のデータ形式への対応等が挙げられる.

参考文献

[1] Mozafari B., Zeng K., Zaniolo C., "High-performance complex event processing over xml streams", Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, p.253-264 (2012).

[2] Alur R., Madhusudan P., "Visibly pushdown languages", Proceedings of the thirty-sixth annual ACM symposium on Theory of computing, p.202-211 (2004).

[3] 松田達希, 内田友樹, 藤田悟, “XMLStream 処理のモデル化と検索言語の設計”, 情報処理学会 第 77 回全国大会, (2015).

[4] 内田友樹, 松田達希, 藤田悟, “XMLStream の時系列イベント処理の性能評価”, 情報処理学会 第 77 回全国大会, (2015).