D-035

系列パターン発見技術の並列分散化の実現と評価
Implementation and Evaluation of Discovery Techniques of Sequential Patterns by Parallel Distributed Processing

松岡　有希†　　合田　浩二†　　西澤　実†　　櫻井　茂明‡
Yuki Matsuoka　Koji Goda　Minoru Nishizawa　Shigeaki Sakurai

1. はじめに

近年，機器のセンサやログから取得される膨大なデータを使って，機器の故障予測や異常検知といった将来予測を行うことが注目されている．多数の系列データから特徴的なパターンを発見する，系列パターン発見技術[1]は，将来予測のための予測ルール生成手段の 1 つとして有効と考えられる．本論文では，大規模なデータに対応できるように，系列パターン発見技術を並列分散化し，性能評価をした結果について報告する．

2. 系列パターン発見技術

[1]が提案する系列パターン発見技術は，系列データ集合の中から特徴的に出現する部分系列を系列パターンとして発見する．対象とする系列データは複数のアイテム集合を時系列に並べたデータである．特徴的な部分系列を発見するために，下式(1)で定義される支持度を利用し，ユーザが指定した最小支持度以上の全ての系列パターンを発見する．

$$\text{支持度 S} = \frac{\textit{系列パターンを含む系列データの総数}}{\textit{系列データの総数}} \quad (1)$$

また，我々が提案した時間制約[2]を指定することで，アイテム間の時間間隔を考慮した系列パターンも発見できる．たとえば，系列パターンを構成する先頭のアイテム集合に含まれるアイテムと，最後尾のアイテム集合に含まれるアイテムとの時間間隔を指定することで，指定された始端と終端の時間間隔に該当する系列パターンのみを発見する．

3. 並列分散化

系列パターン分析では，最小支持度や時間制約の設定を変えて試行錯誤しながら系列パターンを発見するため，一回あたりの処理時間は短いことが望ましい．しかし，系列パターン発見技術は，全ての系列データを繰り返し探索して系列パターンを発見するため，系列データの量に応じて処理時間がかかるという特性がある．そのため，大規模データを使って系列パターンを発見すると，現実的な時間内で終わらないことが問題であった．そこで，処理時間の短縮を課題とし，系列データ集合を分割して並列に系列パターン発見を行うことで全体の処理時間を短縮する方針とした．まず，系列データを分割して系列パターンを発見する方式として，パーティショニング方式[3]を導入することとした．本方式では，以下の 4 ステップで系列パターンの発見を行う．

ステップ 1：系列データ集合を分割し，分割された系列データ部分集合ごとに，式(1)を用いて支持度を計算し，指定された最小支持度以上の系列パターン発見を行う．もし時間制約が指定されていれば，最小支持度および時間制約を満たす系列パターンを発見する．

ステップ 2：各系列データ部分集合で発見された系列パターンを統合して，重複する系列パターンを取り除く．

ステップ 3：各系列データ部分集合で発見された系列パターンは各系列データ部分集合における最小支持度を満たしたもので，系列データ集合における最小支持度を満たしたものではない．そのため，ステップ 2 で得られた系列パターンごとに，系列パターンが含まれる系列データの総数を求めて支持度を再計算する必要がある．ここでは，再度，系列データ集合を分割し，各系列データ部分集合において，系列パターンが含まれる系列データの総数 $C_i = \{C_1, C_2, \ldots, C_n\}$を求める（n は系列データ集合の分割数）．

ステップ 4：ステップ 3 で求めたC_iを利用して，支持度を計算し，指定された最小支持度以上の系列パターンを発見する．支持度は，下式(2)を用いて計算する．

$$\text{支持度 S} = \frac{\sum_{i=1}^{n} C_i}{\textit{系列データの総数}} \quad (2)$$

本方式により，系列データを分割しない場合と同じ系列パターンの発見が可能となる．

さらに，この方式を実装するため，スケールアウト型の並列分散処理を可能にする Hadoop[4]を採用した．Hadoop は，分散処理対象のファイルが保存される分散ファイルシステムの HDFS（Hadoop Distributed File System）と，並列分散処理を行うフレームワーク MapReduce で構成される．本研究では， MapReduce のフレームワークに基づいて，ステップ 1，ステップ 2，ステップ 3，ステップ 4 を適用した．図 1 に系列パターン発見技術における MapReduce の処理フローを示す．

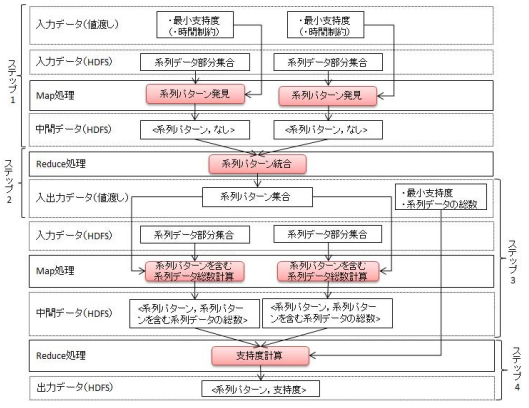

図 1 系列パターン発見技術における MapReduce 処理フロー

4. 性能評価

系列パターン発見技術を並列分散化することで大規模データに対応できているかを確認するために性能評価を行った．ここでは，時間制約がある場合にも大規模データに対応できているかを確認するために，時間制約がある場合と

† 東芝ソリューション株式会社 IT 研究開発センター
‡ 株式会社 東芝 クラウド&ソリューション社 ビッグデータ・クラウドテクノロジーセンター

ない場合で，データ量と並列数をそれぞれ変化させて処理時間を測定した．系列パターン発見の処理が最も時間がかかることから，ステップ 1 およびステップ 2 の処理を対象とした．実行環境は仮想マシンを利用し，6 台のうち 3 台が①CPU1.0GHz×6 コア，メモリ 24GB，残りの 3 台が②CPU1.0GHz×4 コア，メモリ 16GB である．Hadoop は CDH4.5，JavaVM は JRE1.7 を利用した．Hadoop の構成は，①のマシン 1 台をマスターノード，①のマシン 2 台と③のマシン 3 台をスレーブノードとした．Hadoop の設定は HDFS のブロックサイズを 64MB，1 ノードあたりの同時実行 Map タスク数を 1，1Map タスクあたりのメモリ量を 1GB とした．

4.1 スケールアウトの効果

データ量に対する処理時間と並列数に対する処理時間を測定することで，スケールアウトの効果を確認した．性能測定に使用したデータは，機器の異常ログで，系列データに含まれるアイテムは 2114 種類，アイテム集合に含まれるアイテムの個数は 1～28，系列データの系列長は 1～100 である．アイテム集合に含まれるアイテムの個数は 1 のものが最も多く，系列長は 100 のものが最も多かった．データ量が 50MB のときは 51,608 件，100MB のときは 96,765 件，500MB のときは 470,923 件，1GB のときは 967,650 件の系列データを用いた．最小支持度は 0.05，時間制約は始端-終端の時間制約とした．

データ量に対する処理時間を図 2 に示す．スレーブ台数を 5 台とし，データ量を 50MB～1GB に変化させて処理時間を計測した．この結果より，時間制約がある場合もない場合も，データ量の増加に応じて線形的に処理時間が増加していることが分かった．1Map タスクで処理するデータ量が増えたため，処理時間も増加したものと考えられる．時間制約がある場合の処理時間が速いのは，指定した時間間隔に該当する系列パターンの数が少なかったためである．

並列数に対する 1 秒あたりのデータ処理量を図 3 に示す．利用した系列データのデータ量は 1GB で，スレーブの台数を 1～5 台に変えて処理時間を測定した．時間制約がある場合もない場合も，処理量は並列数の増加に応じて増加しており，並列化の効果が見られる．

これらの結果より，データ量を増やすと線形的に処理時間が増えるため，大規模データになると現実的な時間内では終わらないことが想像される．しかし，並列数を上げると線形的に処理量が増加することから，スケールアウトによって並列数を上げることにより，現実的な時間内で大規模データの処理を終えることが期待できる．また，時間制約がある場合もない場合と同様にスケールアウトの効果があることも確認できた．

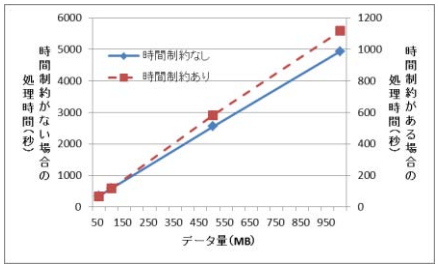

図 2 データ量に対する処理時間

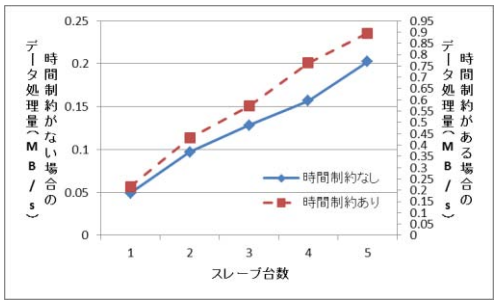

図 3 並列数に対する 1 秒あたりのデータ処理量

4.2 マシンリソースの調査

スレーブノードのマシンリソースが効率的に使われているかどうかについて調べた．データ量が 1GB，スレーブが 5 台で系列パターン発見を実行しているときの CPU 使用率，メモリ使用量，ディスク I/O について調査した．

系列パターン発見の処理が始まってから終わるまでの，プロセスごとの CPU 使用率を調べたところ，系列パターン発見のプロセスが連続してほぼ 100%利用していることが分かった．系列パターン発見のプロセスは最初に系列データをメモリに読み込んだ後，メモリ上で処理を行うため CPU 使用率が常に 100%近くになっていた．メモリに関しては，Hadoop の設定によって割り当てられたメモリ内で処理が行われており，メモリ不足は生じなかった．ディスク I/O に関しては，系列データ部分集合が書かれた系列データファイルを読み込むときと，系列パターンが書かれた系列パターンファイルを書き出すときにディスク I/O が増えていたが，ファイル入出力の処理時間は数秒で，処理時間に大きな影響を与えることはなかった．これらの結果より， 今回の実行環境や Hadoop の設定，系列パターン発見の条件では，CPU がボトルネックとなっており，CPU の性能が高いものを利用するか，系列パターン発見の処理を高速化しないと，処理時間をこれ以上速くすることはできないことが分かった．

5. まとめ

系列パターン発見技術を並列分散化し，性能評価を行った結果，時間制約の有無にかかわらずスケールアウトすることで大規模データを扱えることを確認できた．一方，現在の系列パターン発見技術の並列分散化の実装では，系列パターン発見時の CPU 使用率が高いことが分かった．今後，系列パターン発見の処理を高速化させる等のアルゴリズムの改善が必要である．

参考文献

[1] Agrawal, R. and Srikant, R.: Mining sequential patterns, Proc. 11th Int. Conf. Data Engineering, pp.3-14 (1995)
[2] Sakurai, S.,Ueno, K. and Orihara, R.: Discovery of Time Series Event Patterns based on Time Constraints from Textual Data, Int, J. Computational Intelligence, Vol. 4, No. 2, pp.144-151 (2008)
[3] A. Sarasere, E. Omiecinsky, and S. Navethe.: An efficient algorithm for mining association rules in large databases, Proc. 21St Int. Conf. on Very Large Databases, pp.432-444(1995)
[4] Hadoop, http://hadoop.apache.org/