分散XML処理をための複数経路を用いたルーティングアルゴリズムの提案と評価

Longjian Ye[1]　小出 洋[2]　Yaokai Feng[1]　櫻井 幸一[1]　Dirceu Cavendish[3]

概要：ウェブサービスを利用するときサーバとクライアントの間で XML 文書を用いた情報の伝達が多く行われている．サーバで処理する処理の一部分をネットワークの中継ノードで分散処理すれば，サーバの負荷を軽減できる．本研究では，レイテンシとノードの XML 処理能力により複数の経路を選択するアルゴリズムを提案した．提案したアルゴリズムを評価するためにシミュレータに複数経路選択の機能の実装を行い評価を行った. その結果，マルチルーティングアルゴリズムによって，XML 処理率を向上することができる．

キーワード：XML 処理, オフロード, マルチルーティングアルゴリズム, シミュレーション

Proposal and Evaluation of Multiple Routing Algorithm for Distributed XML Processing

Longjian Ye[1]　Hiroshi Koide[2]　Yaokai Feng[1]　Kouichi Sakurai[1]　Dirceu Cavendish[3]

Abstract: Transmission of information using XML document is widely used between the servers and clients in web service. To alleviate the overhead of the terminal, some processing of XML documents can be offloaded to the intermediate nodes of the net such as format checking and grammar validation. In our research, a multiple routing algorithm based on latency and capacity of nodes is proposed. Also, an evaluation method is implemented by simulation supporting multi-route. XML document process rate is well promoted as result.

Keywords: XML Processing,Offloading,Multiple routing algorithm,Simulation

1. はじめに

Web サービスは企業活動等も含めて多く利用されている．同時に多数の Web サービスが実行されると，CPU 時間やメモリ等の計算機資源が限られていることから，処理速度は低下する [4]．Web サービス間や Web サーバと Web クライアント間の情報交換には XML 文書が良く利用されている．XML 文書はサービスそのもののために利用される他に，XML 文書の構文の正しさの検査，整形式検査，妥当性検査，不要な情報のフィルタリング等の XML 文書処理を行う必要がある．これらの処理とそれによる負荷は利用者の増加，情報交換される XML 文書の量に比例して増加する．

Web サーバの負荷を軽減するため，これらの XML 文書の処理の一部はオフロードすることが可能である．例えば Cavendish らが提出した [1] では，受信側サーバにより処理すべき作業はネットワーク上の中間ノードにオフロード可能できることが前提とした場合，結果としてサーバの処理を軽減できる手法を提案している．サーバの負荷を軽減できるため，より多くのクライアントの要求に応えることが可能になる．わずかな転送速度を犠牲に，全体的に処理速度を向上することができる [2]．また，XML 文書の処理

[1] 九州大学　大学院システム情報科学府
Graduate School of Information Science and Electrical Engineering, Kyushu University
[2] 九州大学情報基盤研究開発センター
Information Infrastructure Research and Development Center, Kyushu University
[3] 九州工業大学
Kyushu Institute of Technology

のような基本的な機能はネットワークが備えるべき機能のひとつとして考えている．

オフロード先のノードの処理能力はそれぞれ異なることが一般的である．そのため，異なる通信経路を利用するとXML 文書の処理される率が異なる．しかし，いままで著者らが提案してきたルーティングアルゴリズムでは，転送の際の遅延だけを考慮してきた．XML 文書の処理をオフロードする目的においては，さらにそれぞれの経路におけるノード処理能力を考慮するべきである [3].

本研究の目的はネットワークトポロジーでのノード処理能力を基づき，XML データがターミナルに至るまでより多くデータが処理されるようにすることである．そのために，複数の通信経路を選択するマルチルーティングアルゴリズムを提案する．このアルゴリズムを使用することにより，XML 処理能力の低いノードより処理能力の高いノードを優先的に使用する経路を選択することができる．そして，シミュレーションを用い提出したアルゴリズムの実用性を評価した．

本稿では，シングルルーティングアルゴリズムとマルチルーティングアルゴリズムについて説明した後，シミュレーション環境で行った実験結果の報告を行う．

2. XML 分散処理

本研究における XML 分散処理は以下の基礎処理から構築されている．

(1) ドキュメントの分割　：XML ドキュメントはノードに処理させるため，すべて同じサイズの断片に分割する．
(2) ドキュメントのマーキング　：全ての断片はマークされている，そのマークはノードから別のノードの転送される時点の状態が記録されている．
(3) ドキュメントのマージ　：すべての断片が集められ，もとのドキュメントどおりの順番にマージされる．

XML 文書は情報をタグで囲まれたデータの集合であり構造を持つ (図 1)．開始タグがあり対応する終了タグが欠落するかどうかを検査するためスタックを利用する必要がある．ノードが開始タグを読み取ると，タグ名がスタックにプッシュされる．ノードが終了タグを読み取ると，スタックから先頭の要素がポップされ，終了タグ名とポップタグ名を比較する．両方のタグ名が同じ場合はタグが一致していると判断できる．すべてのタグが一致した場合，その XML 文書は整形式であると判定できる．図 2 は整形式判定の流れである．文法検証を実行するため，各ノードは DTD ファイルを読み取り，検証チェックのための文法規則を生成する．ポップ，プッシュされたタグと文法規則を比較しながら，各ノードが文法検証と整形式性を同時に処理することができる．処理した XML ドキュメントだけ内容の解析 [5] を行え，結果的に効率が向上できる．

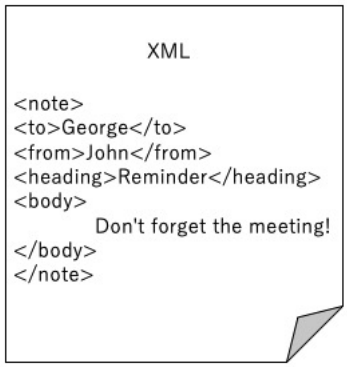

図 1　XML ファイル．
Fig. 1　XML File.

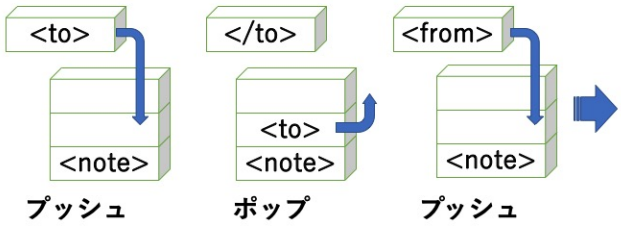

図 2　整形式チェック．
Fig. 2　Check whether well-formed or not.

3. Single Routing アルゴリズム

ダイクストラルーティングアルゴリズムは最短経路問題を解くための最良優先探索アルゴリズムである. より処理能力を得るため, 普段ネットワークの遅延最小経路を選ぶではなく, 新たな重みの計算方法を探さなければならない.

経路を選択する時, 処理能力を持つノードを選びやすいように, 我々は式 (1) の重み計算方法を提出した.weight は重みであり，procSpeed は処理能力が持つノードの処理速度である, 処理能力が持てない場合 procSpeed は 0 とする.

$$weight = \frac{1}{1 + procSpeed} \tag{1}$$

ノードの重みは式 (1) で計算され. この方法を用いる場合, 処理能力がないノードは 1 になり, 処理堪能のノードと比較して大きなコストを持つことになる. 結果として処理能力の高いノードは選択しやすくなる. この重み計算方法を使うアルゴリズムは Capacity Routing と呼びこととする.

Capacity Routing を用いることで XML 処理能力の高い経路を選ぶことができる. しかし,XML 処理能力のある経路が複数存在する場合，Capacity Routing ではどちらか一方のみ経路しか選択することができない. こちらはノードの負荷を考慮し，負荷の変化に応じて経路を選択できるように Available Capacity Routing を提案した. このアルゴリズムでの重み計算方法は式 (2) で表される.

$$weight = \frac{1}{1 + procSpeed \times A_n} \quad (2)$$

$$A_n = \frac{t_{total} - t_{busy}}{t_{total}} \quad (3)$$

ここで, t_{total} は初めから現時点まで経った時間, t_{busy} はノードが XML 処理を行った時間の累計である. 負担が重いノードの t_{busy} はだんだん上がると共に, このノードの重さも上がっている. コストが高くなるため, 処理能力の小さいノードを選択するようになる.

もう一つの要素はノード間のレイテンシーである. ノードは XML 文書の処理中に新し XML 文書を受けった際, 処理していた文書を次のノードへ転送するため, 絶えずデータの通信が行われている場合, ノード n の処理量はノード n の XML 処理能力と任意のノードからノード n へのデータ送信遅延に依存する. そこで, ノードの XML 処理能力およびノードの負担に加え, エッジのレイテンシーを考慮して Latency Available Capacity Routing を提案した. 重み計算方法は式 (4) で表される.

$$weight = \frac{1}{1 + procSpeed \times A_n \times l} \quad (4)$$

$$A_n = \frac{t_{total} - t_{busy}}{t_{total}} \quad (5)$$

ここで,l はノード n に向かうエッジのレイテンシーである. これによって,XML 処理に当てられる時間がより長い経路を優先的に選択するようになる. この方法で, 少し転送時間を増加し, 処理率が上げられる.

4. Multiple Routing アルゴリズム

前に紹介した三つのアルゴリズム cap(Capacity Routing)，acap(AvailableCapacity Routing) と lacap(Latency Available Capacity Routing) を用いれば，XML 処理率が著しく向上できることが期待できる．しかし，その三つのアルゴリズムは単一経路に関するポテンシャルを引き出す工夫のみ行われているため，大量の XML データが絶えず通信されているような状況では，通信回線がいっぱいになりそこが律速になることが推測される．acap と lacap はシステム実行中に経路の変更が可能であるが，現状ではひとつの経路しか利用していない．そこで，われわれは各時点単一経路の負荷を軽減するため Multiple-routing アルゴリズムを提案する．

われわれは Multiple-routing アルゴリズムを二種類提案する．一つ目はダイクストラ法の改善である．最短経路だけではなく，二番目に短い経路，三番目に短い経路の結果も得て，それらの複数の経路を同時に使って XML 文書の処理と転送を行う．二つ目は最短経路を計算した後，この経路にある XML 処理能力の持つノードを利用しないようにネットワークトポロジから削除し，再び最短経路を計算する．手に入れた複数経路を用いて XML 処理と転送を行う．この二つの Multiple-routing アルゴリズムは 4.1 と 4.2 に詳しく説明する．

4.1 Simple Multiple Routing

二番目や三番目およびその次の経路を探すため，ダイクストラ法を用いたルーティングアルゴリズムは以下の流れで行った．開始ノードと目的ノードを含めて全 n 個ノードのあるトポロジーを想定する．開始ノードは 1 で目的ノードは n である．

(1) ダイクストラアルゴリズムで開始ノードから各中間ノードまでの最短距離を全部計算する．そこで S(1,2) から S(1,n-1) まで n-2 個要素あるの数列が得られる．
(2) ダイクストラアルゴリズムで各中間ノードから目的ノードまでの最短距離を全部計算する．そこで D(2,n) から D(n-1,n) まで n-2 個要素あるの数列が得られる．
(3) 中間ノード t を経由する最短経路 L(t)=S(1,t)+S(t,n). L(2)L(3)....L(n-1) を小さい目から順番で並ぶとその順番は最短経路，二番最短経路，三番最短経路... になる．

以上のアルゴリズムで計算した最短経路，二番最短経路，三番最短経路... 複数の経路を同時に用いて XML 文書の転送と処理を行う．

4.2 Respective Multiple Routing

4.1 で紹介した Simple Multiple Routing が XML 文書の処理に利用された通信経路は最短経路である．このとき選択された複数の経路で処理ノードが重複している可能性がある (図 3)．XML 処理能力の持つノードが多数の経路で重複すると処理率の上昇は見込めない．XML 処理能力の持つノードの重複を回避するため，以下の Respective Multiple Routing を提案する．

(1) ダイクストラアルゴリズムで最短経路を計算する．
(2) 最短経路にある XML 処理能力の持つノードをトポロジーから削除する．
(3) 処理したトポロジーでダイクストラアルゴリズムを用いて最短経路を計算する．

この方法で二つの処理ノードが重複しない経路を得ることができる．(2)(3) を繰り返せばさらに新しい別の経路を得ることができる．得られた複数の経路を同時に用いて XML 文書の転送と処理を行う．

5. シミュレーションと結果評価

提案するルーティングアルゴリズムの実際の環境での挙動を評価するため，私たちは実際の環境を模擬するシミュレータ上に提案するルーティングアルゴリズムを実装した．

5.1 シミュレーション概要

今回のシミュレーションは Scala 言語で記述されている．シミュレーションに実装したトポロジーは図 4 で表される．

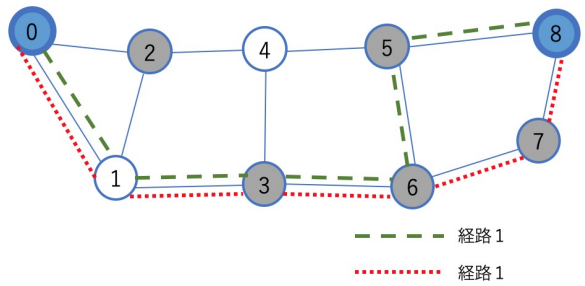

図 3　重ねたノード：3,6
Fig. 3　overlapped node：3,6.

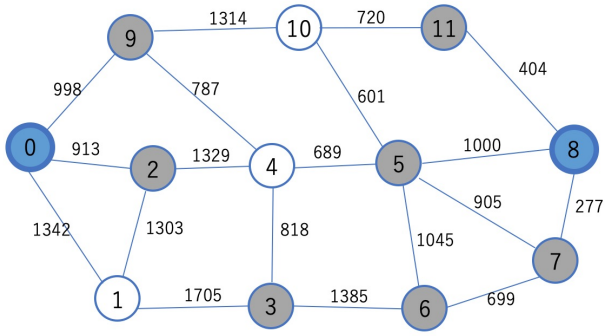

図 4　シミュレーショントポロジー.
Fig. 4　Simulation topology.

網掛けされたノードは XML 処理能力の持つノードである．ノード 0 は開始ノード，ノード 8 は目的ノードである．ノード間の数字はノード間の距離を示している．ノード間の距離に依存する遅延は距離 1000 で 10ms 程度とする．

今回のシミュレーションは小さい負荷から，大きい負荷までの環境を全部模擬できるため，ドキュメントのサイズ (document size) と時間あたりの XML 文書の生成数 (document generate frequence) を変数として実験を行う．また，XML 処理能力の持つノードの処理能力も変数として実験を行った．提案するアルゴリズムの性能を確認するため，処理能力には関係の無い，通常のネットワークルーティングに良く使われる最小ホップ法もシミュレータに実装した．最小ホップ法は出発点から終点まで経由するホップ最小経路を計算するルーティングアルゴリズムである．

本シミュレーションで評価するアルゴリズムは以下の 10 個である．

(1) min(最小ホップ法)
(2) cap(Capacity Routing)
(3) acap(Available Capacity Routing)
(4) lacap(Latency Available Capacity Routing)
(5) SimMulcap(Simple Multiple cap)
(6) SimMulacap(Simple Multiple acap)
(7) SimMullacap(Simple Multiple lacap)
(8) ResMulcap(Respective Multiple cap)
(9) ResMulacap(Respective Multiple acap)
(10)ResMullacap(Respective Multiple lacap)

5.2　結果

実験の結果評価は以下の流れで行う．

(1) 最小ホップ法と提出した cap，acap，lacap との比較
(2) ドキュメントサイズ，時間当たりのドキュメントの生成数，ノードの XML 処理能力の異なる cap，acap，lacap 間の比較
(3) 二つの複数経路アルゴリズムとシングル経路アルゴリズムの比較

5.2.1　min，cap，acap，lacap との比較

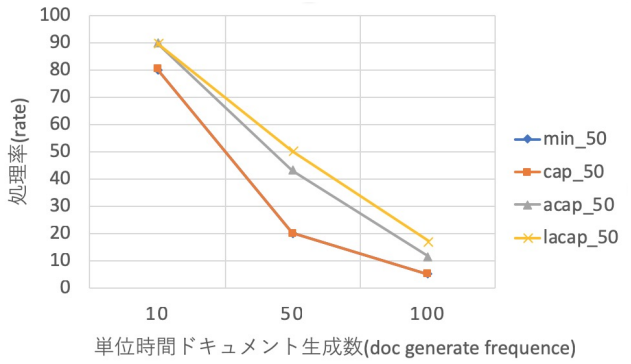

図 5　documentSize = 10.
Fig. 5　documentSize = 10.

図 5 の documentSize は 10(tag) である．XML 処理能力の持つノードの処理能力をすべて 50(tag/s) と設定し，横軸は時間あたりのドキュメントの生成数 (個)，縦軸は中継ノードで処理された XML 文書の割合 (/100) とした．

図 5 に示している min と cap の処理率はほぼ同じである．一方 acap と lacap の処理率は min と cap と比べて明らかに向上することが分かった．

5.2.2　cap，acap，lacap 間の比較

図 5，図 6，図 7 の documentSize は 50(tag) である．図 6 に示している XML 処理能力の持つノードの処理能力はすべて 50(tag/s) に設定した．図 7 の XML 処理能力の持つノードの処理能力はすべて 100(tag/s) に設定した．

図 5，図 6，図 7 に示す通り，XML 処理能力の持つノードの処理能力と documentSize と時間あたりのドキュメントの生成数に関係無く，XML 処理能力は $lacap > acap > cap$ の順番となった．この結果によって，オフロードするため，我々が提出したシステム実行中動的に通信経路の再選択は確実に XML 処理率の向上に役に立つことが分かった．

5.2.3　複数経路アルゴリズムとシングル経路アルゴリズムの比較

図 8，図 9，図 10 に cap，acap，lacap と複数経路を用い

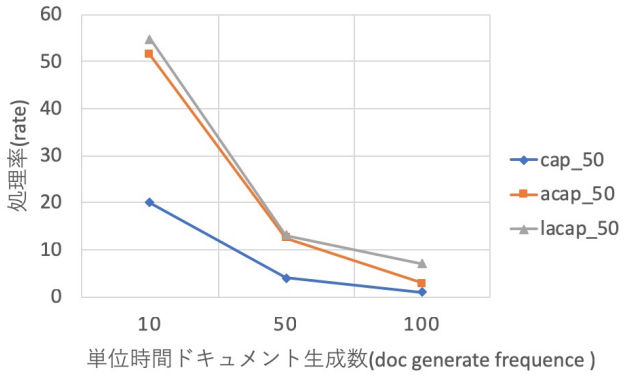

図 6 documentSize = 50.
Fig. 6 documentSize = 50.

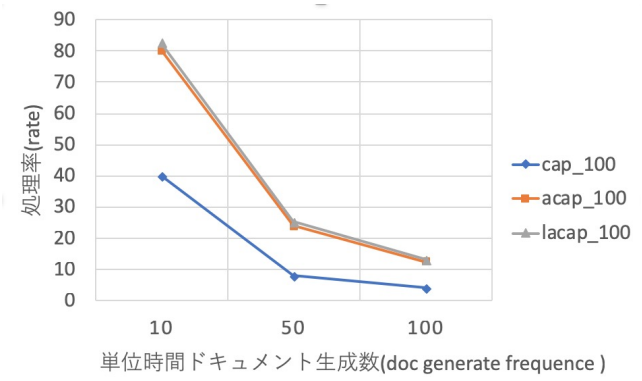

図 7 documentSize = 50.
Fig. 7 documentSize = 50.

る cap，acap，lacap の比較を示した．

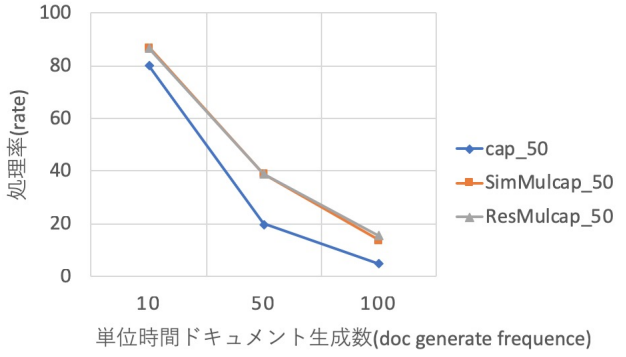

図 8 documentSize = 10.
Fig. 8 documentSize = 10.

図 8，図 9，図 10 の documentSize はすべて 10(tag) である．XML 処理能力の持つノードの処理能力をすべて 50(tag/s) と設定した．これらの比較により，二つの複数経路のパフォーマンスはほぼ同じであることが分かった．ただし単一経路と比べて複数経路アルゴリズムの方が性能が良い．特に単一経路の処理率が低い点において処理率が著しく向上した．二つの複数経路アルゴリズムには特定の場合 Respective Multiple

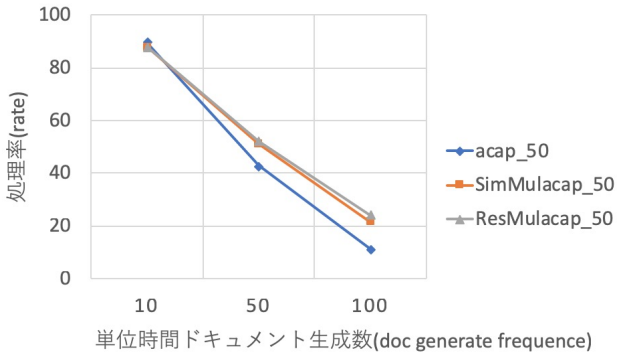

図 9 documentSize = 10.
Fig. 9 documentSize = 10.

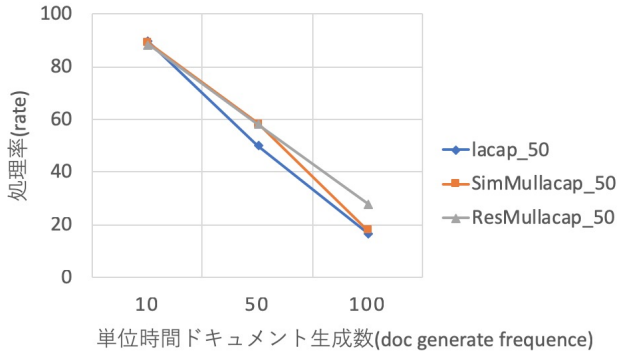

図 10 documentSize = 10.
Fig. 10 documentSize = 10.

Routing が Simple Mutiple より優れた点が挙げられる．単一経路の処理率低い場合全体的に XML 処理能力は $RespectiveMultiple > SimpleMultiple > singlerouting$ の順番となった．

6. まとめ

本研究では，効率よく分散処理を行うための複数経路アルゴリズムを提案し，シミュレーションによる評価を行って，その性能に関する考察を行った．提案したアルゴリズムは単一経路の高負荷を避けることができる．シミュレーションによる評価の結果，本アルゴリズムを用いることにより，従来の単一ルーティングアルゴリズムより高い XML 処理率を得られることが分かった．

問題点としては，本研究は大量な XML 文書処理するための提案であるが，XML データ量が少ない，ノードが XML 処理する十分余裕がある場合処理率はあげられない，処理率が下がってしまう点が挙げられる．本提案手法を XML 文書が少ない場合にも対応することは今後の課題とする．

謝辞 本研究は，国立研究開発法人科学技術振興機構 (JST) 戦略的国際共同研究プログラム (SICORP) および JSPS 科研費 JP18K11295 の助成を受けたものである．

参考文献

[1] Cavendish, D. and Candan, K. S. (2008). Distributed XML processing: Theory and applications.Journal of Parallel and Distributed Computing, 68(8):1054-1069.

[2] Barbera, M. V. , Kosta, S. , Mei, A. , Stefa, J. . (2013). To Offload or Not to Offload? The Bandwidth and Energy Costs of Mobile Cloud Computing. INFOCOM, 2013 Proceedings IEEE. IEEE.

[3] Kim, S. H., Kim, K., Lee, C., and Ro, W. (2012). Offloading of Media Transcoding for High-quality Multimedia Services.Consumer Electronics, IEEE Transactions on, 58(2):691-699.

[4] Uratani, Y., Koide, H., Cavendish, D., and Oie, Y. (2012).Distributed XML Processing over Various Topologies: Characterizing XML Document Processing Efficiency. InWeb Information Systems and Technologies, volume 101 ofLecture Notes in Business InformationProcessing, pages 57-71. Springer Berlin Heidelberg.

[5] Lam, T. , Ding, J. J. , Liu, J. C. . (2008). Xml document parsing: operational and performance characteristics. Computer, 41(9), 30-37.