中継ノードにおけるXMLデータへの並列ストリーミング処理に関する研究

中継ノードにおけるXMLデータへの並列ストリーミング処理に関する研究

浦谷 芳幸

アブストラクト

XML 文書のネットワーク上での中間処理のモデルとして，PASS（Prefix Automata SyStem）-Node[1] が提案されている．PASS-Node では，ネットワークアプリケーションにおいて，従来は送信先の計算機（サーバ）で行われていた XML 文書への文法チェック処理の一部または全てを，各種 Web アプリケーションにおける共通の前処理として，送信先の計算機からネットワーク中に配置される中継ノードへオフロードする．このような処理のオフロードにより，最悪の処理時間の低減とサービスを利用するユーザに対してスループット等のサービスの向上が見込める．オフロードにより，送信先の計算機の処理負荷が軽減され，その分より多くのクライアントからの要求に応えられるようになるためである．さらに送信先の計算機では，サービスにとってより本質的な処理のみに専念することができる．このボトルネックの解消により，最悪の処理時間については改善される一方，中継ノードでの処理時間とオフロードによるオーバヘッドが生じ，最良の処理時間は増加する．そこで合わせて並列処理手法を採用することで，最良処理時間の低減を図る．しかし PASS-Node においては，XML 文書へのオフロード手法について，数式を用いた理論的な定義と，試験実装を用いたオフロード機能の確認のみしか行われておらず，XML 文書へのネットワーク上での中間処理技術を実現する上での課題が残されていた．

そこで本論文では，PASS-Node で残されていた課題を解決することに取り組む．まず，このような並列アプリケーションに必要な機能要素の洗い出しとシステム設計を行い，XML 文書への並列ストリーミング処理を行うアプリケーションを実装した．このアプリケーションは，XML 文書への文法チェックの一部または全てを，中継ノードの配置に応じて，パイプライン並列またはデータ分割並列で任意の中継ノードに割り当てることが可能であり，データの送信先となるノードでは，XML 文書全体にわたって検査済みの結果を得ることができる．このアプリケーションを実環境にて稼働させ，本実装による XML 文書への並列ストリーミング処

理のオフロードがうまく行われていること及び PASS-Node の実現可能性を示した．

そしてこのアプリケーションを用いて，XML 文書への並列ストリーミング処理の基本的な特性の評価と，より実際的な環境を想定した場合の評価として実データに基づく XML 文書への処理及び実環境と仮想環境における処理特性の比較を行った．さらに，中間処理に必須のデータ蓄積用のバッファに着目し，その改良を行った．加えて，本研究で得られた知見を元に，最も効率的だと思われる構成で XML 文書への並列ストリーミング処理を評価し，処理を行う中継ノードの数と処理時間，処理のオフロードの効果を議論した．

近年，ネットワークアプリケーションの重要性が増しており，クラウド・コンピューティングなどの計算機資源を用いたサービス提供や計算手法が広く用いられるようになっている．このようなネットワークアプリケーションのために，ネットワークインフラ自体が高度な処理機能を持つ高機能ネットワークの必要性が議論されるようになってきている．高機能ネットワークでは，アプリケーションに即した処理が提供され，例えば，コンテンツの内容に関するフィルタリング処理，異なる Web サービスの連携のためのデータ変換・加工・集約処理，データ転送量を減らす圧縮・伸張処理，不正アクセスの検知，フロー制御，エラー検知と訂正，統計情報の収集といったものが考えられる．通信のフィルタリングやデータの圧縮は，データの送信元に近い位置でその機能を提供することが有効である．また不正アクセスの検知等必ず提供されなければならないセキュリティ対策機能や複数の Web サービスの連携機能等，種々のアプリケーションに共通して利用可能な機能をネットワークそのものがまとめて提供することにより，送信先の計算機においては処理の一部をネットワークに移譲できる．これにより，送信先の計算機では処理負荷が軽減され，本来行いたい重要な処理のみに注力できるようになるため，コスト・機能・性能面で有用である．さらにクライアント側においては，スループット等のサービスの向上といった利点を得ることができる．またネットワークインフラを保有する回線事業者にとっては，現在単なるインフラに留まっているネットワークに付加価値を持たせ，処理機能提供による新たな利益を得る機会が生まれる．

さらに，本研究で評価の題材とする XML 文書は，汎用性に優れているが内部に構造を持つデータ形式である．そのため，適切なデータの分割が為されずデータの欠損が起こった場合，処理を進めることができなく

なるため，本質的に単純なデータ分割による並列処理には不向きである．このような並列処理の難しい XML 文書を題材とすることで，本研究で得られた知見を，より並列処理の易しい他のデータ形式への処理に対しても適用でき，データを送信元の計算機（クライアント）から送信先の計算機（サーバ）へと送り，送信先の計算機にて処理を行うようなアプリケーションへの成果の応用が期待できる．

目 次

図 目 次

表 目 次

第1章　はじめに

1.1　本研究の目的と中継ノードにおける XML 文書へのストリーミング並列処理

従来のネットワークにおけるアプリケーションサービスの提供形態では，クライアントとなる計算機より，サーバとなる計算機への処理要求とデータが送られ，サーバにおいて処理を行い何かしらのサービスを提供していた．このようなサービス形態においては，クライアント数の増加に比例してサーバでの処理量と負荷も増大していく．これに対応する方法の一つとして，サーバ以外の場所で処理を行うオフロードと呼ばれる負荷分散手法がある．オフロードにより，サーバとなるの計算機での処理負荷が軽減され，その分より多くのクライアントからの要求に応答できるようになり，サービスとしてのスループットが向上し，ユーザに対しても同様にスループット等のサービスの向上を見込める．我々はこのようなオフロードについて，クライアントとサーバの間のネットワーク中の，通信を中継するノードへの処理のオフロード（図 1.1.1）を採り上げ，XML 文書 [2] のネットワーク上での中間処理のモデルとして，提案されている PASS（Prefix Automata SyStem）-Node[1] に着目する．PASS-Node では，ネットワークアプリケーションにおいて従来，送信先の計算機（サーバ）で行われていた XML 文書への文法チェック処理の一部または全てを，各種 Web アプリケーションにおける共通の前処理として，送信先の計算機からネットワーク中に配置される中継ノードへオフロードする．このような処理のオフロードにより，最悪の処理時間の低減とサービスを利用するユーザに対してスループット等のサービスの向上が見込める．オフロードにより，送信先の計算機の処理負荷が軽減され，その分より多くのクライアントからの要求に応えられるようになるためである．さらに送信先の計算機では，サービスにとってより本質的な処理のみに専念することができる．このボトルネックの解消により，最悪の処理時間については改善される一方，中継ノードでの処理時間とオフロードによるオー

バヘッドが生じ，最良の処理時間は増加する．そこで合わせて並列処理手法を採用することで，最良処理時間の低減を図る．しかし PASS-Node においては，このような XML 文書への処理手法について，数式を用いた理論的な定義と，試験実装を用いたオフロード機能の確認のみしか行われておらず，XML 文書へのネットワーク上での中間処理技術を実現する上での課題が残されていた．

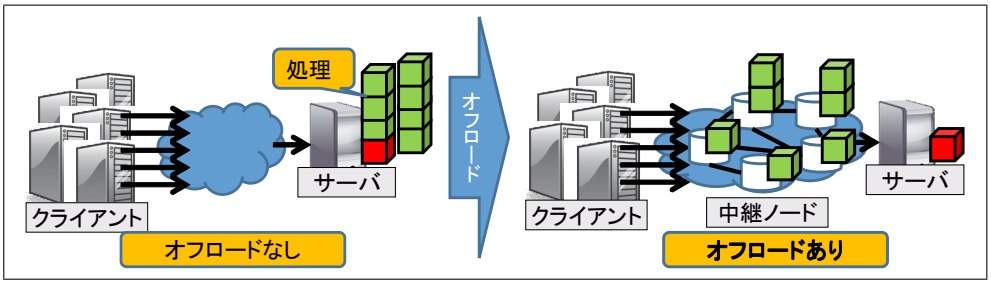

図 1.1.1: Server Processing Offload.

そこで本論文では，PASS-Node で残されていた課題を解決することに取り組む．まず，このような並列アプリケーションに必要な機能要素の洗い出しとシステム設計を行い，XML 文書への並列ストリーミング処理を行うアプリケーションを実装した．このアプリケーションは，XML 文書への文法チェックの一部または全てを，中継ノードの配置に応じて，パイプライン並列またはデータ分割並列で任意の中継ノードに割り当てることが可能であり，データの送信先となるノードでは，XML 文書全体にわたって検査済みの結果を得ることができる．このアプリケーションを実環境にて稼働させ，本実装による XML 文書への並列ストリーミング処理のオフロードがうまく行われていること及び PASS-Node の実現可能性を示した．

そしてこのアプリケーションを用いて，XML 文書への並列ストリーミング処理の基本的な特性の評価と，より実際的な環境を想定した場合の評価として実データに基づく XML 文書への処理，実環境と仮想環境における処理特性の比較を行った．さらに，中間処理に必須のデータ蓄積用のバッファに着目し，その改良を行った．加えて，本研究で得られた知見を元に，最も効率的だと思われる構成で XML 文書への並列ストリーミング処理を評価し，処理を行う中継ノードの数と実行時間，処理のオフロードの効果を議論した．

近年，ネットワークアプリケーションの重要性が増しており，クラウ

ド・コンピューティングなどの計算機資源を用いたサービス提供や計算手法が広く用いられるようになっている．このようなネットワークアプリケーションのために，ネットワークインフラ自体が高度な処理機能を持つような高機能ネットワーク（図 1.1.2）の必要性が議論されるようになってきている．

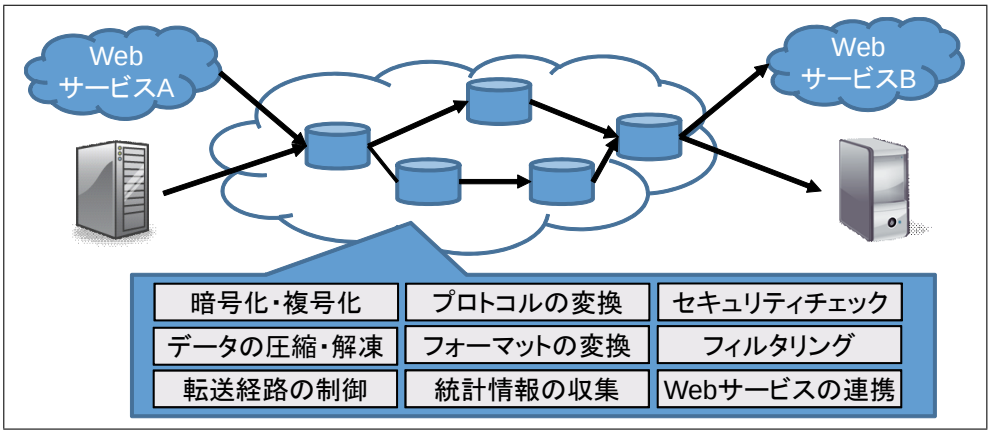

図 1.1.2: Overview of RelayNode Processing Environment.

高機能ネットワークでは，アプリケーションに即した処理が提供される．例えば，コンテンツの内容に関するフィルタリング処理，異なる Web サービスを連携して利用するためのデータ変換・加工・集約処理，データ転送量を減らすためのデータの圧縮・伸張処理，不正アクセスの検知，フロー制御，エラー検知と訂正，統計情報の収集といったものが考えられる．通信のフィルタリングやデータの圧縮は，データの送信先で行うよりも，送信元に近い位置でその機能を提供した方が有効的である．また不正アクセスの検知等必ず行われなければならないセキュリティ対策機能や複数の Web サービスを連携して利用するためのデータ変換等，種々のアプリケーションに共通して利用可能な機能をネットワークそのものがまとめて提供することにより，送信先の計算機においては処理の一部をネットワークに移譲でき，処理負荷の軽減と本来行いたい重要な処理のみに注力できるようになるため，コスト・機能・性能面で有用である．さらにクライアント側においても，スループット等のサービスの向上といった利点を得ることができる．またネットワークインフラを保有する回線事業者にとっては，現在単なるインフラに留まっているネットワークに付加価値を持たせ，遊休資源を活用した資源利用の最適化による新たな利益を得る機会が生まれる．

現在のネットワークにおける処理機能の提供は，例えば，動画像の変換を行うトランスコーディング [3, 4]，コンテンツ配信で有用なキャッシュ[4, 5, 6]，センサネットワークを対象としたセンシングデータの集約 [7, 8, 9]，転送データの圧縮と伸長 [10] 等の例が既存研究や技術として存在する．さらに，中継ノードにおける処理のためのプラットフォームとして，Active networkp[11] や VNode[12] などの研究が提案されている．この技術により，従来であればデータを送信元から送信先へと届けるという，ライフラインとしての配送機能のみを有したネットワークに，任意の位置での任意の処理機能を埋め込むことが可能となり，ネットワークインフラ自体に新たな付加価値を持たせることができるようになる．

本論文では，評価アプリケーションの実装と実環境における稼働により，本実装による XML 文書への並列ストリーミング処理がうまく分散されていることを示し，PASS-Node 並びに高機能ネットワークの具体的な実現可能性を示す．

また，評価アプリケーションでは，XML 文書への文法チェック処理の一部を，任意の中継ノードに割り当てることができる．どの処理を・いつ・どこで行うのかを決定することをスケジューリングと呼び，効率的な並列処理を実現するための重要な要素である．本評価アプリケーションを用いて，実際の計算機において，実際の XML 文書を用いて，実際のネットワークを通し，様々な条件で XML 文書に対する並列ストリーミング処理の実験と評価を行い，PASS-Node を実アプリケーションとして実現した際の処理特性を詳細に調査した．ここで得られた様々な実験結果を元に，並列ストリーミング処理に関する効率的なスケジューリング手法を実現するための様々な指標を測定し議論を行う．まず第一の実験・評価として，XML 文書への並列ストリーミング処理について，基本的な処理特性の実験・評価を行った（第 4 章）．ここでは，機械的に作成した合成 XML 文書を用いて，データ分割並列処理またはパイプライン並列処理を行う数種類のトポロジでの実験を行った．その結果，

- 概してパイプライン並列処理よりもデータ分割並列処理の方が高効率である
- ノード数を増やした場合，データ分割並列処理では処理効率が良くなるが，パイプライン並列処理における恩恵は小さなものに限定される
- XML 文書への並列処理の効率は，そのデータ構造（タグのネスト

の深さ）と分割方法と処理の割り当てが強く影響する

といった中継ノードでの処理におけるスケジューリング手法の検討に資する有用な知見を得ることができた．第二の実験・評価では，より実用的な環境を想定した評価を行った．ここでは，実データに基づくXML文書への処理，実環境と仮想環境における，処理特性の比較を行った（第5章）．現在のインターネットを取り巻く環境では，物理資源と同様に，仮想化された資源が用いられることが多くなっている．実環境・仮想環境双方の処理特性を把握することで，より実際的な環境を見据えた知見を得る．この実験の結果，

- 実環境と仮想環境では，その処理特性に関して差が小さく，一方で行った実験によって得た知見は，もう一方にも適用可能である
- XML文書への並列処理に要する時間は，ファイルサイズだけでなく，特に処理対象となるXML文書中に含まれるタグの数とタグの種類（空タグの場合は処理時間が短くなる）に強い影響を受ける

といった実験結果を得た．これらの知見は今後，中継ノードでの並列ストリーミング処理を行う場合に，どのように処理を分割し，どのように資源を割り当て，いつ処理を行えばよいのか，どのようなスケジューリング手法が効率的かを検討する際の有益な判断材料となる．これらの成果により例えば，同じ台数の中継ノードを中間処理に使用できる場合は，パイプライン並列ではなくデータ分割並列の処理形態を採る，パイプライン処理を採る場合は，段数をあまり増やさない，仮想環境と実環境のどちらでも使用できる場合は，どちらを使用しても差は小さく，両者を特別区別しない，といったスケジューリング手法を採ることが一つの方針となると考えられる．

加えてこれらの実験結果を元に第三の取り組みとして，評価アプリケーションにおけるXML文書への並列ストリーミング処理の性質を分析し，中継ノードでの処理時に，データを一時格納するために使用するバッファに関して議論した（第6章）．ここでは，このようなデータ蓄積用のバッファに求められる機能要件を整理し，これまで用いてきたバッファの改良を行った．その結果，シンプルな改良ではあっても，改良後のバッファは改良前に比べて高効率なものとなった．この改良型のバッファを元にさらにもう一段階改良を進め，複数スレッドからの同時並列アクセスに対応するバッファの実装を行った．この改良の結果，今後，複数CPUコアを用いた単一XML文書への同時並列処理を行うことが可能となった．

また，この成果は評価アプリケーションにおけるXML文書に対する文法チェックと同様に，処理前後のデータの順番が保存される必要のある性質を持つ他の種類の処理にも適用することができる．この改良は単純なものではあるが，より効率的な処理が可能となった．

本研究で評価の題材とするXML文書は，汎用性に優れるデータ形式であると同時に，内部に構造を持つデータであり，適切なデータの分割が為されずデータの欠損が起こった場合，処理を進めることができなくなるため本質的に単純なデータ分割による並列処理には不向きである．他方，動画像や音声などはデータの一部に欠損が起こった場合でも，ブロックノイズが入るなどの障害が起き得るが，視聴という観点で言えば影響は小さい．このような，並列処理の難しいXML文書を題材とすることで，本研究で得られた知見を，より並列処理の易しい他のデータ形式での処理に対しても適用でき，データを送信元の計算機（クライアント）から宛先の計算機（サーバ）へと送り，宛先の計算機にて処理を行うような様々なアプリケーションに対する，広範な成果の応用が期待できる．

さらに本論文では，上記までの実験・評価を通じて得た結果を元に，最適と思われる手法にて中継ノードで並列ストリーミング処理を行う場合と，データの送信先となる計算機単体で全ての処理を行う場合の処理時間について，比較と考察を行った結果を示す．さらに，評価アプリケーションを用いて，処理のオフロードを行った場合と行わなかった場合について，各ノードでの処理時間を測定し，オフロードの効果についても示す（第8章）．

1.2 本研究の主要な貢献

本節では，本研究における主要な貢献をまとめる．

まず第一の貢献は，PASS-Nodeに代表される中継ノードを用いた並列ストリーミング処理の実現可能性を示したことである．これまで，PASS-Nodeに関する研究においては，XML文書に対する処理のオフロードについて，数式を用いた理論的な証明と試験実装によるオフロードの確認のみが行われている段階であった．本研究では，PASS-Nodeを具体的なネットワークアプリケーション（評価アプリケーション）として実装した．評価アプリケーションでは，本来データの送信先でデータ転送後にまとめて行うべきXML文書への処理の一部または全てを，データ転送を行う中継ノードにオフロードすることが可能である．その際に並列分散

処理技術を適用し，中継ノードの配置や個数に応じてパイプライン並列処理・データ分割並列処理の 2 種類の処理形態により，中継ノードでの共通処理機能の提供を行う．我々はこの評価アプリケーションの実装と稼働とをもって，特に，中継ノードを用いた XML 文書に対する並列ストリーミング処理の実現可能性を実証した．

第二の貢献は，この評価アプリケーションを用いた，具体的な実験と評価を行って，XML 文書への並列ストリーミング処理の様々な性質を明らかにし，並列ストリーミング処理のための，スケジューリング手法の検討に資する知見を示したことである．これは，XML 文書への並列ストリーミング処理の基本的な特性の評価と，より実際的な環境を想定した状況での評価の二段階に分けられる．これらの評価の結果，データ分割並列処理の方がパイプライン並列処理よりも高効率であること，XML 文書への処理時間は XML 文書の構造と分割の仕方の影響を強く受けること．実環境と仮想環境では似た性質を持つことなどを示した．

第三の貢献は，評価アプリケーションにおける並列ストリーミング処理の性質を分析し，中継ノードにおける並列ストリーミング処理時に，各ノードでデータを一時格納するのに使用するバッファの改良を行ったことである．ここでは，このようなアプリケーションに求められるバッファの機能要求を洗い出し，簡単な改良でより効率的なバッファを実装した．

1.3 研究背景及び研究の位置づけ

近年，安価で高性能な有線・無線ネットワーク環境の整備が進んでいる．同時に，このネットワークを利用する計算機やモバイル機器などの高性能化・低価格化，ネットワークサービスの提供を受ける対象となる機器や利用者が増加し続けている．今日では，計算機端末を利用する際にネットワークを介した利用は当たり前のものとなっている．例えば，ビジネス・教育・報道・娯楽の提供・行政システムの運用などで，既に必要不可欠な社会基盤の一つとなっており，このような環境で提供されるアプリケーション・サービスが非常に重要な存在となっている．そしてこれらのアプリケーション・サービスは，単体ではなく，様々なシステムや機能部品と相互に連携して動作することが多い．例えば，電子商取引システムを例に挙げると，商品情報管理システム，顧客情報管理システム，認証システム，課金システム，暗号化システム，Web ページのレンダリング機能，などの多くのサブシステムとネットワークサービス同士の連

携が必要となる．これらによって提供される機能の中には，情報管理や認証機能，暗号化機能など，複数のサービス間でもほぼ共通の機能部品が含まれており，それらをまとめて行うことで効率化が期待できる．例えば，セールスフォース・ドットコム [13] などでは，様々な業務で共通して必要となる，顧客関係管理に特化したサービス機能をクラウド上で提供しており，顧客情報の管理とその活用（営業活動の支援，カスタマーサービス，コールセンター業務，など）を一元的かつ効率的に行うことができるようにしている．本研究では，このようなサービスで利用するような共通機能を，高機能ネットワークで提供することを考えている．

さらにネットワークサービスの普及と高度化に伴い，データ爆発（情報爆発）[14] と呼ばれる，ネットワークでやり取りされるデータ量の増大が生じている [15, 16]．それにより，今日および将来のネットワークでは，文書ドキュメント，動画像などのマルチメディアコンテンツなどで構成されるような，ビッグデータ [17] と呼ばれる多様な種類の多量のデータを取り扱うことが求められることとなる．この傾向は今後も続いていくと考えられ，多様多量なデータを効率よく処理していく必要がある．

この種のデータへの処理手法として，データを送信する側でデータの送信前に処理を行う方法と，データを受信する側でデータの受信後に処理を行う方法がある．この両者では，多量のデータへの処理を行う場合，送信側の計算機や受信側の計算機に負荷が集中するため，単純にこれらの方法を採る場合は，両者の計算機の処理能力の向上を図る必要がある．これには設備増強のための投資などが求められることとなる．さらに，大規模データへの処理や大量の計算資源を必要とする処理の場合は，二つの方法を採ることができる．一つは，一台の高性能な計算機を使って処理を行う方法である．これにはメインフレームやスーパーコンピュータと呼ばれる，非常に高価な計算機を用意する必要がある．また二つ目の方法として，安価な計算機複数台を，ネットワークで相互接続して，同時に処理を行う並列分散処理を行う方法がある．この並列分散処理は，一つの大きな処理をタスクと呼ばれる小さな処理単位に分割して，各計算機に割りつけて同時並列に処理する手法である．本研究では，並列分散処理が対象とする処理に時間のかかる大規模な処理だけでなく，処理にあまり時間のかからない小さな処理も対象とする．これらの処理の一部または全てを，中継ノードで行うことで，ネットワークの末端付近に配置される計算資源の節約や，ネットワーク資源の負担を軽減することが期待できる．またこのように多数の計算資源を用いる技術として近年で

はグリッドコンピューティング [18] やクラウドコンピューティング [19] など，ネットワーク上に存在する計算資源を必要な時に必要な分だけをオンデマンドで用いる手法が広く使われるようになっている．これらの処理はネットワークのコア網の外に計算資源を配置して処理を行うが，本研究では，高度機能ネットワークの実現のために，中間ネットワークに配置された中継ノードの計算資源での処理に着目する．加えて最終的には，並列分散処理とクラウドコンピューティングなどのネットワークの外での処理，中間処理の両者を，処理対象のデータと処理の特性に応じて使い分け，大規模な処理はクラウド計算資源へ誘導し，小規模ですぐに終わる処理は中間ノードで行ってしまうなど，適材適所で利用できるようする．

さらに，これまで述べたネットワークでやり取りされるデータへの処理要求はバッチ処理と呼ばれる時間的制約のゆるいものから，リアルタイム処理と呼ばれる時間的制約の厳しいものまで様々なレベルの処理要求がある．近年ネットワークアプリケーションにおいて，後者のリアルタイム処理に近い制約を持つ処理に対する要求が高まっている．このための処理手法として，例えばストリームコンピューティング [20] では，クラウド環境においてデータをストレージに格納せず，メモリ上に配置したまま高速に処理を行うことで処理速度と処理のリアルタイム性の向上を図っている．本研究でもこのようなストリーミング処理を対象としているが，データへの処理を行う場所として，ネットワークの通信経路の中継系計算機を対象としている．さらに，このストリームコンピューティングでの知見と，本研究によって得られる知見は相互に適用することや，中継ノードでのストリーミング処理，クラウド環境におけるストリームコンピューティングなどの処理手法を組み合わせた処理形態の実現が期待できる．例えば，あるストリームデータへの処理が，データのフィルタリングとフィルタリング後のデータに対しての処理の二段階に分けられる場合，フィルタリング処理を本研究の中継ノードが行い，中間処理ノードからの出力データをクラウド環境へ誘導し，それ以降の処理をクラウド上の計算資源でのストリームコンピューティングで行うといった連携や，特定のフォーマットで与えられるクライアントからの処理リクエストを，中間処理ノードにて各クラウド計算環境が提供する API 向けのものに逐次変換していくといった連携が考えられる．これにより，フィルタリングにより削除されたデータ分だけネットワーク資源の消費を減らし，データへの処理は重い処理に向くクラウド資源を使って効率的に行うことがで

きる．また現在の Web サービスでも，Twitter の Twitter and Streaming API[21]，Facebook の Facebook Open Stream API[22] など，多種多量なデータを生成し続ける Web サービス自身が，ストリーミング処理に特化した仕組みを提供するようになってきており，ストリーミング処理への要求が高まっている．

さらに他方では，先に述べたような増大するデータ通信要求に対して，物理回線を提供するインフラプロバイダ側でも課題が発生している．現在の多くのインフラプロバイダは，データ転送のみを行い，単純な接続性だけを提供する単純なパイプとしての役割が主であり，各インフラプロバイダ間の競争は，単純な利用コストの競争となってしまっている．この状況下で，先に述べた情報爆発の進行によって生じる，大量のネットワークトラフィックの転送要求に応えていくためには，単純には物理回線設備の増強が必要となるが，これはインフラプロバイダにとって大きなコストとなるため，ネットワーク資源の効率的な利用が求められている．例えば [23] では，この問題に対して，ネットワークの仮想化技術を用いることで，ネットワーク資源を複数のサービスで共有し，資源利用効率の向上を図りながら，サービスプロバイダの需要に応えることで，インフラプロバイダにおける問題の解決を図っている．また，インフラプロバイダ自身がデータ転送の機能だけでなく，クラウド計算資源を提供するような事例 [24, 25, 26] も増えてきている．このように，ネットワークへの付加価値を増大させるとともに，資源の利用効率の向上を図りながら利益を得る機会の創出と，インフラプロバイダ間での差別化が可能となるような技術や仕組みが，求められるようになってきている．本研究では，このようなインフラプロバイダでの付加価値を提供する手段として，ネットワークに高度な機能をもたせることを企図している．このような，ネットワーク機能をインフラプロバイダ自身が提供することで，ネットワークの状況に応じて，中間処理時に処理ノードと処理量を切り替えるといったネットワーク資源と計算資源を組み合わせた制御が期待できる．さらに，仮想化されたネットワーク環境においては，計算資源も仮想計算機として提供されることが多く，本研究においてもこの仮想計算機での処理を評価の対象の一つとして扱う．また今日では，世界中に広がったインターネット環境においても，将来の継続的な発展のために，新世代ネットワークの研究が行われている．新世代ネットワークに関する研究は，現在のインターネットのアーキテクチャやサービスの提供形態にはとらわれず，既存技術の改良や拡張では困難な課題や限界に対して，白

紙（clean slate）から新たなインターネットを検討・構築を図っていくという特徴を持つ．これは，データ爆発に対応するための単純な性能向上だけでなく，ネットワーク機器の消費エネルギーの増加への対応や，更にセキュリティ面では，各種の攻撃手法やスパムメールの問題，その他社会的に求められる問題の解決などが含まれる．NwGN[27] は，日本で行われている新世代ネットワークに関する研究であり，地球にやさしいネットワーク，価値を想像するネットワーク，生活環境を支えるネットワーク，トラスタブルネットワーク，制約を意識しないネットワークの実現をターゲットにおいている．JGN-X[28] は，研究技術の実証のためのテスドベッド環境であり，日本中に配置した物理ノードを仮想化してスライスを切ることで，例えばユーザごとに仮想化された OpenFlow ネットワークや IP ネットワークなどの実験用ネットワークを得ることができる．欧州の European Commission の主要政策である第 7 次研究・技術開発のための枠組み計画（FP7）の ICT 分野の取り組みである [29] は，特に欧州域内の大学・企業の技術力や競争力を強化していくための取り組みで，基礎研究だけでなく応用や商用に近い部分まで含めた研究開発を行っている．FIRE[30] はこのような研究開発のために構築されるテストベッド環境である．FIND[31] は，アメリカの研究ファンドプログラムであり，新世代ネットワーク実証のためのテストベッド環境である GENI[32] のような，研究プロジェクトへの助成を行っている．GENI ではコンセプトとして，様々な実験を行うための各ノードの動作を制御するプログラマビリティ，異種資源の共有とアイソレーションを提供する仮想化，GENI の異なった部分をそれぞれの組織が各々で管理・連携運用することの三つを挙げ，多数組織による様々な実証実験を同時並行して行うことを重視している．また，世界中で検討・研究されている新世代ネットワークの標準化のために ITU-T による標準化活動 [33] も進められている．

これらは，アプリケーションにとっては，動作環境となるプラットフォームであるネットワークそれ自体を高性能化・高機能化させる取り組みといえる．本研究では，これらと異なるアプローチとして，将来のネットワークでは，アプリケーションレイヤ向けの高度な機能の提供が必要と考え，このような機能を提供すること研究目標として採り上げ，アプリケーションにネットワークから機能を提供することによりアプリケーションを直接サポートすることを目指す．また，これらの新世代ネットワークの研究は長期的な戦略に立って行われているため，実現までに時間がかかることが予想されるが，本研究は，現時点のネットワーク環境ないし

は近い将来に適用可能な技術として研究している．そして本研究は，これら新世代ネットワークの研究と相反するものではなく，本研究の研究成果はこれら新世代ネットワークにおいても，高度なネットワーク機能の提供に活用することが可能である．

また，現在のネットワークにおいても処理機能の提供を行う技術が研究されている．例えば，動画像の変換を行うトランスコーディング [3, 4] がある．これは，動画像の配信サービスにおいて，クライアントに応じて動画像の解像度やコーデックを変換して届ける技術である．また，コンテンツ配信サービスで用いられる，キャッシュと言った技術がある．[4, 5, 6]．これは，クライアントがサーバよりコンテンツをダウンロードする時，サーバに格納したコンテンツをネットワーク中のキャッシュサーバにコピーしておき，クライアントは最も近いキャッシュサーバよりコンテンツを得るという手法である．この手法により，応答時間の短縮とネットワーク負荷の軽減が可能となる．また，センサネットワークを対象としたセンシングデータの集約 [7, 8, 9] では，大量に収集されるセンシング情報をまとめることで，データ転送時のネットワーク負荷の軽減やセンサデバイスのバッテリ消費を抑えることが可能となる．加えて，[10] では，送信元の計算機から送信先の計算機へデータを送信する際に，送信元に近い場所でパケットを圧縮し，送信先に近い場所でそれを伸長するという方法で途中のネットワークに流れるデータ量を小さくすることでネットワークへの負荷を軽減する．

本研究では特に，通信を行う計算機の間に複数の経路が存在し，その通信経路上にデータの中継を行う中継ノードが配置され，それらを計算資源として利用できるという環境を前提として置く．そして，これら中継ノードにてデータ転送を行うとともに，転送途中のデータに対して有益な処理を行うことで，ネットワーク自体が機能を持つような環境の実現を目指した研究を行う．このようにネットワーク上に配置される中継ノードで処理を行う処理手法として Active Network[11] がある．Active Network では，ネットワーク上に配置された，特殊なルータでの処理を行うもので，ハードウェアに処理機能を実装しているため高速な処理が可能である．本研究では，アプリケーションよりの処理とネットワークが機能を持つことによる利点を重要視し，速度よりも高汎用性と高度な処理機能を提供する．VNode[12] は，仮想化技術を用い，このような中間処理機能を仮想化資源において提供することでより柔軟な資源割り当てを可能としている．さらに PASS-Node[1] は，Web アプリケーションを対

象に，データの送信元となるクライアントより送信されたされたXML文書への処理の一部をデータの送信先となるサーバから，通信経路となるネットワークに配置された中継処理ノード（PASS-Node）に，各ネットワークアプリケーションで共通の処理機能としてオフロードするものである．これにより特にサーバとなる送信先の計算機での，処理負荷の分散・軽減や，開発・運用コスト等の削減および，サービスのスループットの向上を見込める．本研究ではこのPASS-Nodeを，中継ノードを用いた並列ストリーング処理の重要技術として着目する．

1.4 本論文の各章の構成

本論文は，全9章にて構成されている．まず，本第1章では，本研究の目的とアプローチ，背景と位置づけ，本論文の構成に関して述べた．

続く第2章では，本研究で対象とする中間処理についての利点と，本研究にて題材とした，中継ノードにおいてXML文書への並列ストリーミング処理を行う評価アプリケーションに関しての解説を行う．ここでは，この評価アプリケーションについての実装の概要，どのようなXML文書への処理を対象とし，その処理をどのようにして進めていくのか，さらに評価アプリケーションを動作させるための並列分散処理プラットフォームであるタスクスケジューラ[34]について述べる．

第3章では，本論文における実験に関して，実験環境，入力データとして使用したXML文書，タスクの構成，評価のための指標の定義と概要について述べる．

第4章から第6章までは，評価アプリケーションを用いたXML文書への中継ノードにおける並列ストリーミング処理に関する実験・評価を行った結果を述べる．

第4章は，XML文書への並列ストリーミング処理に関する特性の初期評価を行う．この時，機械的に生成した合成XML文書ならびに仮想環境を用いる．これにより，XML文書への並列ストリーミング処理に関する基本的な特性を把握する．

第5章では，第4章での評価を踏まえて，より実際的な条件を設定しての詳細な評価を行う．この時，先の合成XML文書と実データに基づくXML文書，仮想計算機で構成された実験環境と実計算機で構成された実験環境の比較といった様々な条件での評価を進める．

第6章では，評価アプリケーションのさらなる改良を進める．これは，

上記の実験評価結果の分析から得られた，中継処理ノードにおけるデータ蓄積用のバッファについて，改善可能な点が判明したため取り組んだものである．このために，評価アプリケーションに求められるバッファ構造の機能要求を整理し，それをシンプルな形で適用し性能の改善を試みた．

続く第 7 章では，本研究の関連研究についての解説を行う．

第 8 章では，これまでの実験・評価を通じて得た結果を元に，最適と思われる手法にて中継ノードで並列ストリーミング処理を行う場合と，データの送信先となる計算機単体で全ての処理を行う場合の処理時間について，比較と考察を行った結果と，評価アプリケーションを用いたオフロードの効果を示す．

最終章となる第 9 章では，本論文の各章にて得られた知見を整理し，まとめと今後の展望を示す．

第2章　中継ノードで行うストリーミング並列処理

本章では，本研究で対象とする中間処理についての解説と，このために実装した評価アプリケーションについての解説を述べる．

2.1　PASS（Prefix Automata SyStem）-Node

本研究では，XML 文書への効率的なストリーミング処理手法を提唱している PASS（Prefix Automata SyStem）-Node[1] で考案されている，通信を中継するノードにて，XML 文書に対して文法チェックやフィルタリング等の有益な前処理を行う手法に着目している．PASS-Node は，これまで XML 文書への処理に適用されていなかった，non-blocking，non-wasteful，autonomous という三つの特徴を有する prefix nature[35] と呼ばれる処理手法を XML 文書への処理に適用したものである．さらにこれまでも，ネットワークプロセッサを用いた XML 文書への処理方式が存在したが，資源の動的な再配置に対応していないなどの問題があった．PASS-Node においてはこれらの問題の解決も図ることが可能である．このような PASS-Node にて定義される前処理の実現により，送信元の計算機からサーバとなる送信先の計算機へ XML 文書を送り処理を行う際の，サーバ計算機での処理負荷のネットワークへのオフロードや，中間ネットワーク上の遊休資源を用いた資源の有効活用，高度なネットワークルーティング，さらには，ネットワーク自体に付加価値をもたせることが可能となる．しかし，[1] においてこの PASS-Node は，数学的な定式化による理論的な裏付けと，試験実装によるオフロードの評価のみしか行われておらず，具体的なシステムとしての実装は示されていなかった．本研究ではこの PASS-Node を，具体的なネットワークアプリケーションとして実現することで，その実現可能性を示し，ストリーミング並列処理についての課題を明らかにする．我々は PASS-Node を実現するための基本機

能として，

- XML 文書を分割して処理できるように複数のフラグメントに分割する機能
- 分割した XML 文書フラグメントの処理状況や処理割り当てを管理する機能
- 分割された XML 文書フラグメントを再結合し処理結果を得る機能

の三つを定義し後述する各種のノードを実装し，XML 文書へのストリーミーング並列処理を行う．

2.2 提案する中継処理とその効用

ネットワークの普及と利用端末数の増加により，今日ではネットワークアプリケーションが重要なものとなっている．そして，ネットワーク上でやり取りされるデータ量も同時に増大し，また，そのデータに対して処理を行う機会も増えている．さらには，これらのデータに対してよりリアルタイムな処理を行う要求も存在する．

このような状況下で我々は，データへの処理をコアネットワーク（インフラプロバイダ内の基幹ネットワークで，外部との直接の接続の無い環境) の外に配置される各種計算機端末やクラウド計算資源だけではなく，ネットワーク内の通信経路の途上の中継ノードにて行う手法を採り上げる．その際に，流れてくるデータに対してストリーミング処理を行うことで，データの転送と処理を同時に進めていく．インフラプロバイダの基幹ネットワーク外部との直接の接点の無いようなネットワークである．これにより，全てのデータを受けてから処理を行うような処理形態ではなく，ネットワーク資源，プロセッシング資源の両方を動的に利用するような新しいアプリケーションやサービスを，資源を効率的に使いながら実現することができる．本研究では，このような中継ノードでの処理に並列分散処理の手法を適用して効率化を図っていく．そして，中継ノードで行うストリーミングデータ処理を，特にタスクスケジューリングという観点で評価していく．以下に，中継ノードで処理を行う利点を述べる．

- データへの処理を行う場合に，その処理の一部または全てを中継ノードに割り当てるため，処理を行うサーバやクライアントなどの

エンド端末や，コアネットワークの外に計算資源を持つグラウドコンピューティング資源での処理負荷が軽減される．これにより，ネットワークのコアの外の計算資源への，過剰な設備投資などのコストを減少させることができる．

- ネットワーク越しにデータを転送する場合，転送時間と遅延が発生する．その転送時間に対してわずかな処理時間を足すだけで，コアネットワーク外の計算資源への処理負荷が軽減できるとともに，有益な処理を施すことができる

- このような機能を，物理インフラプロバイダが提供することで，インフラプロバイダは，単なる接続性を提供しデータ転送を行うだけの，単純なパイプとして以上の付加価値を提供可能となる．これにより，プロバイダ間での特徴の差別化や新しいビジネスモデルの創出が期待できる

- さらにインフラプロバイダでは，自身の保有する物理ネットワークの現在の情報を動的にモニタリングすることができる．この情報はタスクスケジューリングを考える上で非常に有用な情報である．この利点を生かして例えば，ある時は混んでいる経路を避けて通信と処理を割り当てる，転送データ量が多い通信の場合には帯域の広い経路を割り当てるなどの効率的な制御を動的に行うことができる．これにより，データへの処理とネットワークの制御とを結びつけた効率的なタスクスケジューリングの実現が期待できる

- 大規模なネットワークの持つ計算資源を利用することで，アプリケーションの処理に対するスケールメリットを得ることができる

- このような処理を，資源の余っている中継ノードの使っていない遊休資源に割り当てたり，バッファ蓄積による転送の待ち時間の間に行ったりすることで，無駄となっている資源やデータ転送の待機時間を有効活用することができる

- すべてのデータを受け取ってから処理を始めずに，データを受け取りながら処理も同時に進めることができるため，リアルタイムな処理要求に柔軟に適応可能である

本研究では PASS-Node を元に，高機能ネットワークが提供する機能の一つとして XML 文書への処理を題材とし，XML 文書に対する中継ノー

ドでのストリーミングデータ処理に関する実験・評価を行った．この評価を通して，ストリーミングデータ処理の特性を明らかにし，中間処理機能を提供する高機能ネットワークの例証と，このような中間ノードでの処理手法における，タスクスケジューリングに資する知見の蓄積を図る．

2.3 評価アプリケーション

本研究で実装を進めている評価アプリケーションはPASS-Nodeを参考に実装されており，送信元の計算機から送信先の計算機へXML文書を送る際に，途中経路上に配置した計算機で通信を中継するとともに有用な処理を施し，送信先の計算機で最終的な処理結果を得るものである．これによって送信先の計算機は，従来のように単体で処理を行った場合よりも処理に要する負荷や時間を減らすことができる．本アプリケーションでは現在，XML文書の整形式検査（Well-formedness grammar checking）と妥当性の検証（Validation grammar checking）の二つの機能が実装されており，以下に述べる4種類のノードを組み合わせてXML文書の分散処理を行う．

`StartNode`：XML文書を読み込み，データの送信を始めるノードであり，

1. ローカルストレージにファイルとして格納されているXML文書を読み込み，
2. 後続のRelayNodeへの処理割り当てを決定し，
3. タグチェック処理と割り当てのための情報を付加し，
4. XML文書を後続ノードに送る，

という機能を有する．図2.3.1にStartNodeの構成を示す．図2.3.1では，StartNodeの後ろに三つのノードがある状況を想定している．この場合，StartNodeは三つのRead/SendThreadを持つ．各Read/SendThreadは，それぞれ担当部分のXML文書を読み込んで，処理のための情報を付加し，後続のノードに送る．図2.3.1で，Read/SendThread01はXML文書の前半の1/3を担当し，Read/SendThread02はXML文書の中盤の1/3を担当し，Read/SendThread03はXML文書の後半の1/3を担当する．また，各Read/SendThreadは同時に動作し，評価アプリケーションはStartNodeにおいて，XML文書の読み込みと送信などの処理を同時に行うことができる．

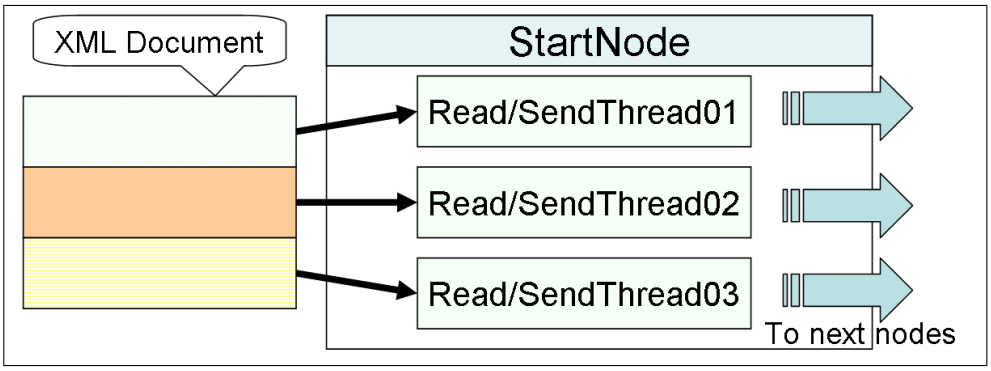

図 2.3.1: Components of StartNode (three next nodes case).

`RelayNode` : データの中継を行うノードであり，前段のノードからデータを受信し，中継するデータに対して必要に応じて処理を行いながら，後続ノードへデータを送信する．この受信・処理・送信の三つの動作は，それぞれ別のスレッドで同時に動作し，これらは共有バッファを介してデータをやり取りする．図 2.3.2 に RelayNode の構成を示す．RelayNode は，ReceiveThread，TagCheckThread，SendThread の三つのスレッドとそれらのスレッドが共有するバッファ（図 2.3.2 中の Shared Buffer）を持つ．ReceiveThread はデータの受信を行い，前段のノードからデータを受信し共有バッファに追加する．TagCheckThread は受け取った XML 文書へのタグチェック処理を行うスレッドで，バッファからデータを取り出し，そのデータへの処理がこの RelayNode に割り当てられていた場合には処理を行う．SendThread は，

- すでに処理の終わっているデータ，
- 未処理だがこのノードで処理を行うように割り当てられなかったデータ，

を共有バッファから取り出し XML 文書での順番通りに後続のノードに送る．

`MergeNode` : データ分割並列処理の場合にのみ用いられるノードであり，複数の前段のノードから受け取ったデータを中継し，正しい順番に並べ替えて後続に送るシリアライズ処理を行う．図 2.3.3 は，MergeNode の構造を図示したものである．この時，MergeNode の受信側にはデータを待ち受けるためのスレッドがそれぞれのノード

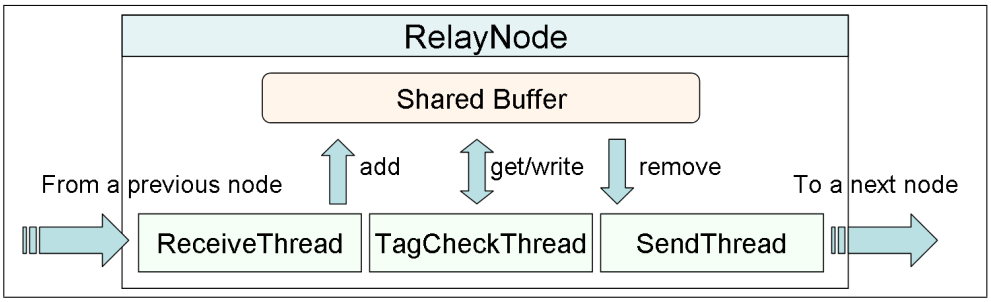

図 2.3.2: Components of RelayNode.

に対して配置され，そのスレッドのうち XML 文書の先頭に近いものから順に後続ノードにデータを出力する．その際，後続ノードにデータを送信することができない他のスレッドは，自身のデータの出力の順番を待ちながら受信処理のみを行い，各スレッドが持つバッファ（図 2.3.3 中の Temporal Buffer）にデータを蓄積する．

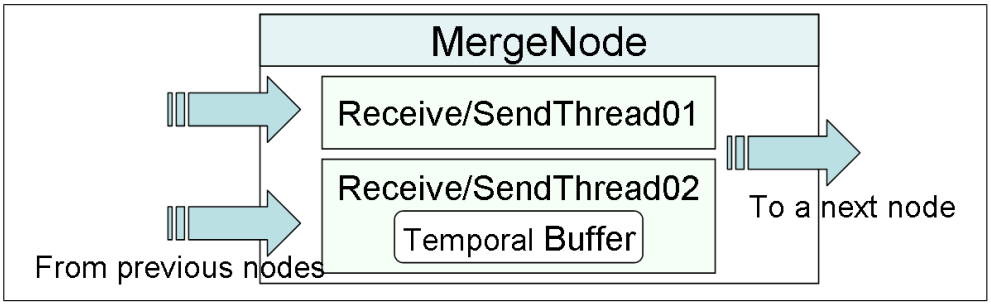

図 2.3.3: Components of MergeNode (two previous nodes).

`EndNode` : データの終着点となるノードであり，最終的な XML 文書全体にわたり処理を行った結果を得る．EndNode は，RelayNode と似た構造を持ち，処理済みのデータを送信するスレッドの代わりに，XML 文書に付加した情報を取り除き，共有バッファからデータを削除するスレッドを有する．EndNode に到着したデータの中で，この時点でまだ処理が完了していない全てのデータは，この EndNode にて処理され最終的な処理結果を得る．

2.3.1 XML 文書の文法チェック処理

評価アプリケーションでは，XML 文書の整形式検査と妥当性の検証の 2 種類の XML 文書への文法チェックを行うことができる．これを行う RelayNode 及び EndNode に配置される XML 文書へのストリーミング処理のためのスレッドである TagCheckThread では，XML 文書に対する文法チェックに XML 文書本体とタグチェック管理情報とスタックを用いる．図 2.3.4 に XML 文書の整形式判定と妥当性の検証の例を示す．タグチェック管理情報は XML 文書中のタグそれぞれに対して一つずつ用意される文字列データであり，そのタグの処理の状況 (0:未処理，1:処理済み，2:対応がとれていないタグ) を表す．XML 文書の整形式検査においては，Start タグを読んだ場合にはスタックにタグ名を積み，End タグを読んだ場合には，スタックから要素を取り出しタグ名を比較する．全てのタグ名が一致しているなら，その XML 文書は整形式であるといえ，逆にそれらが一つでも一致していなければタグの対応がとれていないと判断できる．そして TagCheckThread は，pop した Start タグと読み込んだ End タグに対応するバッファ中の要素を取得し，処理の結果を書き込む．

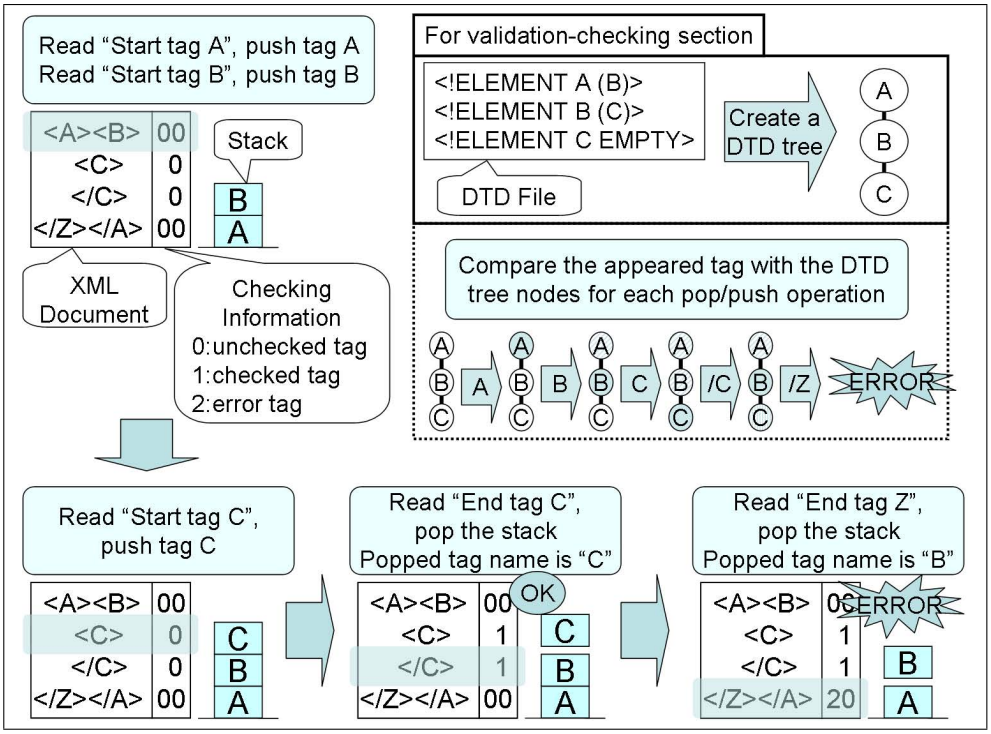

図 2.3.4: A Node Tag Checking Example.

一方，XML 文書の妥当性を検証する場合には，上記の整形式判定処理に加えて，タグや要素の出現順序についても検証を行う．このために，StartNode は XML 文書とあわせて，XML 文書の構造を決定するべく予め定義された DTD（Document Type Definition）[36] ファイルを読み込み，後続の各ノードに頒布する．各ノードはこの DTD ファイルに基づいて，構文木を生成し文法検査（整形式検査+タグの出現順序の検査）を行う．図 2.3.4 中の右上の四角の中の処理が，妥当性の検証のための処理である．このため一般に文法検査は整形式検査よりも多くの計算時間を必要とする．

評価アプリケーションでは，XML 文書本体と，タグチェック管理情報を各ノード間でやりとりすることでチェック処理を並列に行う．さらに，プロセッシング情報と呼ばれる，その行をどのノードで処理するのかの情報も付加して処理割り当てを行う．

さらに，タグの対応が取れてない場合やタグの出現順序が不正な場合などのエラー検出時には，そのエラーが後続の別のノードにて回復できる可能性があるかどうかを判断し，エラーが後続ノードで回復できる可能性がある場合は，エラー検知を意図的に無視することがある．例えば，RelayNode にて Start タグを読み込み，そのタグの名前をスタックにプッシュした後，自身に割り当てられた処理範囲だけでは，対応する End タグを見つけられなかった場合である．これは，XML 文書への文法チェック処理を適切に完了するためには，Start タグと対応する End タグの組への処理が同時に RelayNode に割り当てられねばならないが，適切な処理の分割とその割り当てが為されなかったため起こる．逆にエラーが後続ノードで回復できなことが確定した場合は即時エラー情報を書き込む．例えば，RelayNode にて Start タグを読み込んだ後，本来対応する End タグが入るべき場所に別のタグを発見した場合（図 2.3.4 はこの例に当たる）と，全てのデータの終着点であり，XML 文書全体に渡る完全な処理が可能な EndNode にてエラーを検出した場合である．このようにして，エラー検出時の状況によって適切な対応を試みる．

2.3.2 並列分散処理プラットフォーム

前述した，StartNode，RelayNode，MergeNode，EndNode の 4 種のノードを，Java 及び JavaRMI（Java Remote Method Invocation）[37] で実装されている並列分散処理プラットフォームであるタスクスケジューリング

システム（タスクスケジューラ）[34] を用いて動作させる．並列分散処理は，一つの大きな処理をタスクと呼ばれる小さな単位に分割して，ワーカと呼ばれる計算機に割りつけて同時並列に処理する手法である．さらに，どのワーカで，どのタスクを，どのようなタイミングで実行するのかを決定することをタスクスケジューリングと呼ぶ．タスクスケジューラでは処理の単位をタスクとして扱い，タスクを任意のスケジューリングアルゴリズムに基きワーカとなる計算機へと割り付ける．本研究では先に紹介した StartNode，RelayNode，MergeNode，EndNode の各ノードを，このタスクスケジューラ上で動作するタスクとして実装し，タスクをノードに割り当てる方法で XML 文書への文法チェックを構成している．タスクスケジューラは主に図 2.3.5 と下記に示すモジュールで構成される．

また，このタスクスケジューリングにおいて，最適な資源割り当てを求めることは，組み合わせ最適化問題であり，NP 困難な問題として知られている [38, 39]．そのためこれに対し，ヒューリスティックで効率的なスケジューリング手法が提案されている [40, 41, 42]．これらは各タスクが一回の通信で一度だけデータ転送を行うような，通信の場合が主たる対象である．これらに対し，[43] のように，タスク間で到着データを逐次的に処理するストリーミング通信を行う場合は，効率的なスケジューリングが難しいという課題がある．その理由としては，うまくタスクスケジューリングを行うためには，正確な資源の利用状況の収集や予測を行う必要があるが，ストリーミング通信を行うタスクをもつ場合は，その処理がいつ終わるのかの予想が難しくなるためである．このストリーミング通信を行うタスクのタスクスケジューリングは重要ではあるが，本研究では将来の研究課題とする．

タスクコントローラ ：タスクコントローラは，ユーザによって定義された並列プログラムを，タスクとして各マネージャに割り付ける役割を持つ．その際にスケジューラと連携し，スケジューラによって決定された割り付け先に従って，タスクをタスクマネージャに渡す．

スケジューラクラス ：スケジューラクラスは，あるタスクをどのワーカにどう割り付けるかを決定する方法であるスケジューリングアルゴリズムを提供するクラスである．タスクコントローラからの要求に応じて，並列プログラムの実行前に選択しておいたスケジューラのアルゴリズムに基づき，タスクを割り付けるマネージャを決定す

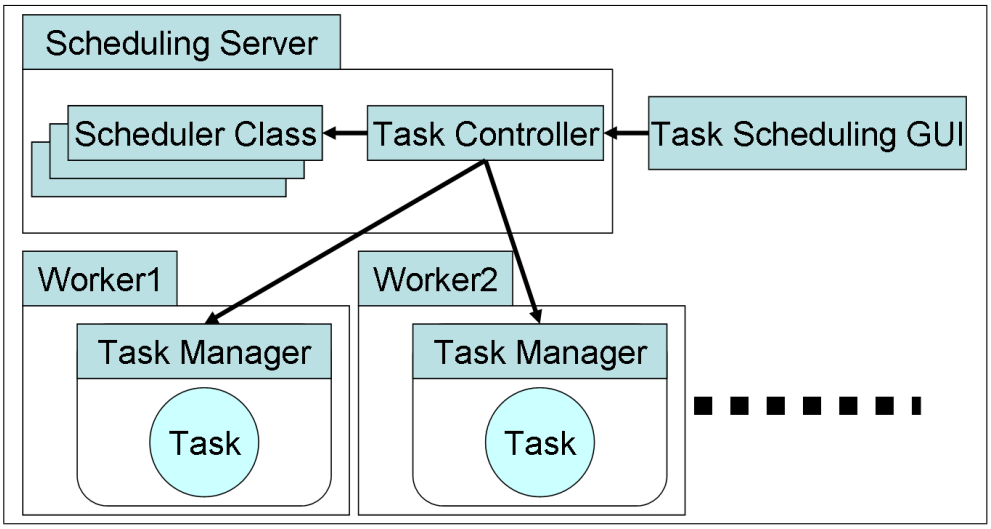

図 2.3.5: Task Scheduling System.

る．このタスクスケジューラは任意のスケジューリングアルゴリズムを定義しスケジューリング時に利用することができるが，それは [40, 41] のような静的スケジューリングだけではなく，[34, 42, 43] のような動的スケジューリングも可能である．これらのスケジューリングアルゴリズムのうちどれを利用するかは，並列プログラムの実行前に選択する．本稿では，本論文で提案・評価する手法とスケジューリングの効果とを切り離して考えるため，ワーカへのタスクの割り付けは，プログラム実行前に事前に定義することで静的に行う．

タスクマネージャ ：タスクマネージャは，ワーカとなる計算機で動作するプログラムである．タスクコントローラによって割り付けられたタスクを，ワーカとなる計算機にて動作させる役割を持つ．

タスクスケジューリング GUI ：タスクスケジューリング GUI は，タスクコントローラを制御するための GUI フロントエンドである．この GUI を用いて各タスクの配置と入出力の接続を行うことができる．そして，スケジューラを選択し並列プログラムを実行する．また，実行前・実行中・実行終了といったタスクの状態のモニタリング，データ通信の様子を表示することができる．このタスクスケジューリング GUI 画面のスクリーンショットを図 2.3.6 に示す．図 2.3.6 では，ウィンドウ中の各四角がタスクを表している．タスクにはタス

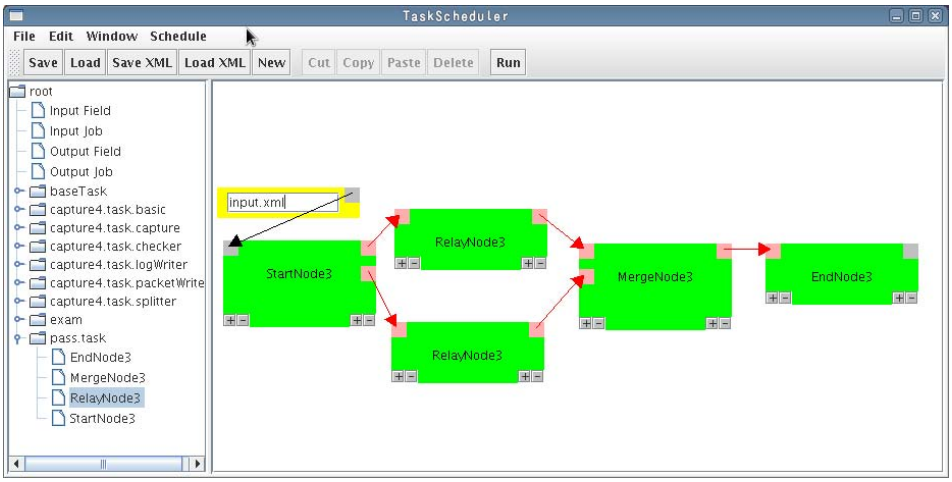

図 2.3.6: A Screenshot of Task Scheduling GUI.

クの名前とそのタスクがどの計算機に割り付けられたのかが表示され，さらに矢印で各タスク間の通信の状況が表現される．

スケジューラやタスクには，あらかじめ定義された Java インターフェースクラスが用意されている．タスクスケジューラの利用者は，このインターフェースに基づいてタスクの処理内容を実装し，タスクを作成することで任意のタスクを並列プログラムとして実行することができる．また同様に，タスクをワーカへと割り付ける順序を決定するスケジューラを作成すれば実装したアルゴリズムに基づいてタスクを自由に割り付けることができる．この様に実際の処理とスケジューリングに関する部分が完全に独立しており，タスクの作成に注目するユーザーはタスクの実装だけを，スケジューラに注目するユーザーはスケジューラの実装だけを考えることができる．

第3章　評価実験環境構成

本章では，本論文で述べる評価実験の構成の概要を述べる．本論文ではXML文書に対するストリーミングデータ処理の特性を明らかにするために

- 1種類の実環境と2種類の仮想環境を用いた合計3種類の実験環境（本章3.1節）
- 様々な特徴を持った合成XML文書（synthetic doc）と実データに基づくXML文書（realistic doc）の2種類のXML文書（本章3.2節）
- これらXML文書に対する2種類の文法チェック処理

を組み合わせた実験を行う．ここでこの合成XML文書は，実験のために機械的に作成した特定の構造を持つXML文書である．このうち，2種類の文法チェック処理についての詳細は2章2.3.1項にてすでに述べた．本章では上記について，まず3種類の実験環境について述べ，続いて2種類のXML文書の特徴を述べる．

3種類の実験環境はそれぞれ，仮想計算機・実計算機の差があるが，それぞれの比較を行うため，タスクを割り当てる実（仮想）計算機の数とその構成は同一になるように構成している．これらの環境を用いて，中継ノードでのXML文書へのストリーミング並列処理を行う評価アプリケーションを動作させる．それにより，中継ノードでのストリーミングデータ処理のスケジューリングに資する特徴を把握する．

3.1　実験環境

本説では，下記に示す本論文で使用する3種類の実験環境について述べる．

- 実験環境1:T5440_Env（仮想環境）

- 実験環境 2:PC_Env（実環境）
- 実験環境 3:X4640_Env（仮想環境）

3.1.1 実験環境 1:T5440_Env（仮想環境）

T5440_Env は，実計算機上に仮想計算機を用いて構成された仮想環境である．この環境において我々は，仮想計算機を稼働させるホスト計算機として ORACLE SPARC Enterprise T5440 Server を用いた．このサーバ機の構成を表 3.1.1 に示す．T5440 Server は表 3.1.1 に示す構成で，合計四つの CPU に計 32 個の CPU コアを持ち，最大 256 スレッドを同時稼働させることができる．また，ストレージは RAID-0 ストライピングにて構成されている．

この T5440 サーバ上で，OS レベル仮想化技術である Solaris Container[44] を用いて仮想計算機群を構築する．Solaris Container 上の仮想計算機は zone と呼ばれる．この環境の構成を図 3.1.1 に示す．Solaris Container を通じて我々は各 zone に割り当てる CPU コア数やメモリ量を動的に変更することができる．また，Solaris Container は自身が稼働している OS を特にシステム内で唯一の global zone として取り扱う．また，各 zone は直接通信を行うことができ，スケジューラは zone01 にて動作させる．

表 3.1.1: T5440 Server Specification.

CPU	Sun Ultra SPARC® T2 Plus (1.2GHz) × 4
Memory	128G bytes (FB-DIMM)
OS	Solaris™10
JVM	Java™1.5.0_17

3.1.2 実験環境 2:PC_Env（実環境）

PC_Env は，実計算機を用いて構成された環境である．この環境において我々は，2 機種 7 台の計算機を用いる．表 3.1.2 にこの環境を構成する 2 種類の計算機の性能を示す．さらに図 3.1.2 に，これらの計算機を用いて構成される実験環境を示す．

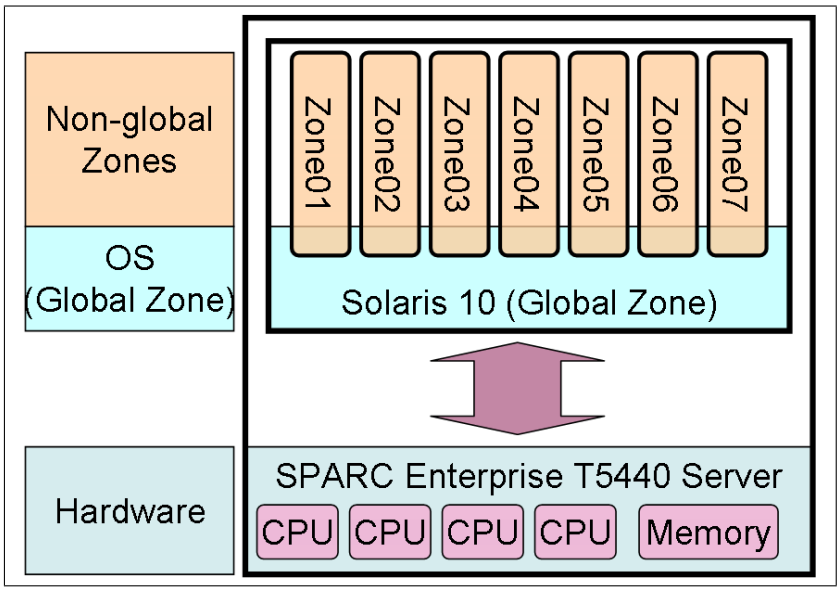

図 3.1.1: Component of Experimental Environment(T5440_Env).

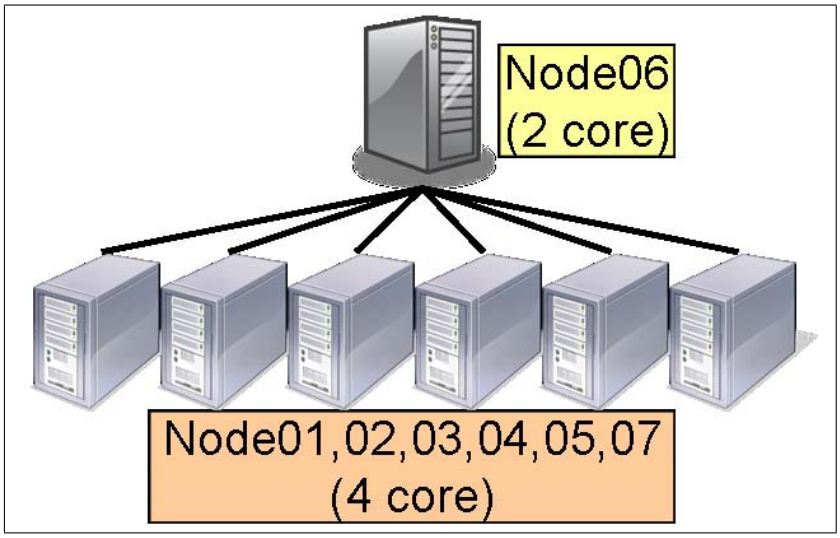

図 3.1.2: Component of Experimental Environment(PC_Env).

表 3.1.2: PC Cluster Specification.

	PC 4core	PC 2core
CPU	Intel Core 2 Quad Q965 (3GHz)	Intel Core 2 Duo E8400(3GHz)
Memory	4G Byte	
NIC	1000 BASE-T Intel 8254PI	1000 BASE-T Intel 82571 4 port × 2
OS	Fedora13_x86_64	
JVM	Java^{TM}1.5.0_22	

3.1.3 実験環境 3:X4640_Env（仮想環境）

X4640_Env は，前述の PC_Env の構成を元に実計算機上に仮想計算機を用いて構成された仮想環境である．この環境において我々は，Oracle Sun Fire X4640 Server に VMware ESX 4 を用いて構築した七つの仮想計算機（VM）を構築した．この環境の構成を表 3.1.3 及び図 3.1.3 に示す．実験には，図 3.1.3 に示される最大 4CPU コアを割り当てた六つの VM（node01–05，07）及び，最大 2CPU コアを割り当てた一つの VM（node06）を用いる．各 VM には，2Gbyte のメモリを割り当て，ゲスト OS として Fedora15_x86_64 を使用した．

本実験環境では，これらの実計算機または仮想計算機に対して第 2 章にて述べた PASS-Node の各タスクを割り当て，XML 文書への文法チェック処理を行う．

表 3.1.3: X4640 Server Specification.

CPU	Six-Core AMD Opteron Processor 8435 (2.6GHz) × 8
Memory	256G bytes (DDR2/667 ECC registered DIMM)
VMM	VMware ESX 4
Guest OS	Fedora15_x86_64
Allocated Memory	4G Byte
JVM	Java^{TM}1.5.0_22

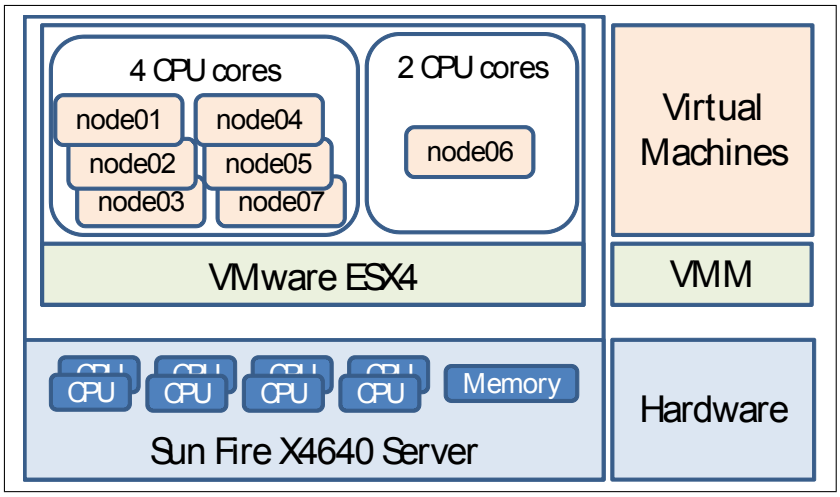

図 3.1.3: Component of Experimental Environment(X4640_Env).

3.2 評価用 XML 文書

本論文における実験では，本章冒頭で先述したように，実験用に構成した 7 種類の合成 XML 文書（synthetic doc）と実データを元に構成した 3 種類の実データを元にした XML 文書（realistic doc）を使用する．表 3.2.4 及び 3.2.5 に双方の XML 文書の特徴を示す．

表 3.2.4: XML Document Characteristics (synthetic doc).

	doc01	doc02	doc03	doc04	doc05	doc06	doc07
Width	10,000	5,000	2,500	100	4	2	1
Depth	1	2	4	100	2,500	5,000	10,000
Tag set count (Empty tag count)	10,000(0)						
Line count	10,002	15,002	17,502	19,902	19,998	20,000	20,001
File size [Kbytes]	342	347	342	343	342		

表 3.2.4 に示すように，合成 XML 文書（synthetic doc）はそれぞれファイルサイズはほぼ同じであるが，タグのネストが浅いものから深いものまである．図 3.2.4 にこれらの合成 XML 文書の構造を図示する．これらの合成 XML 文書では，このネスト構造の違いによって，XML 文書への並列処理時に処理の依存関係が変化していくこととなる．これにより，XML 文書内のタグの依存関係とその分散処理の特性の変化を調べる．

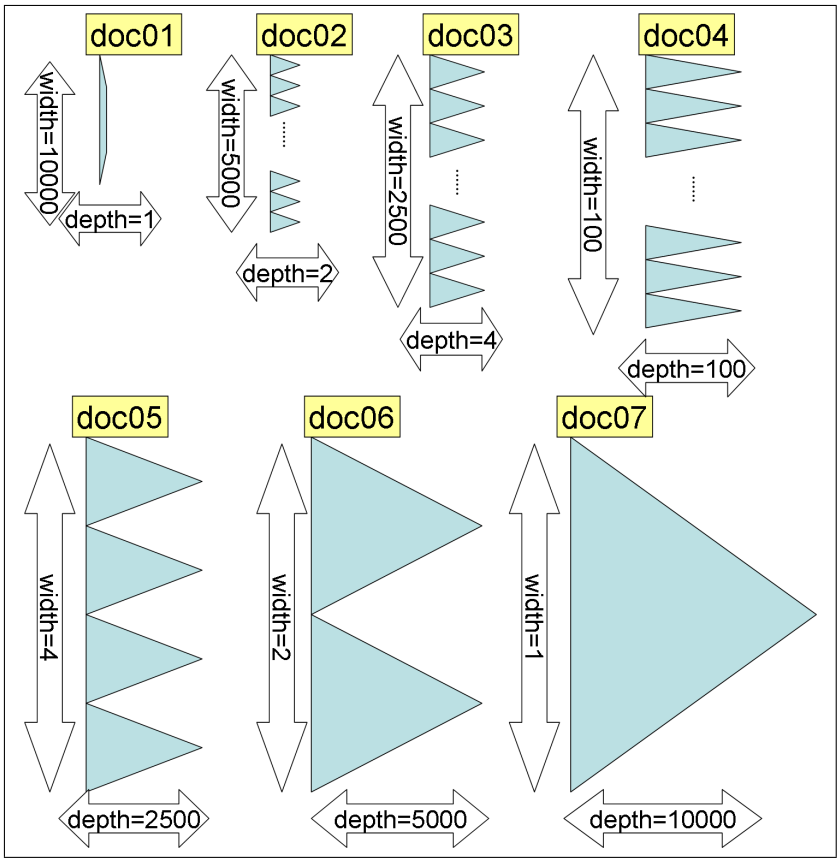

図 3.2.4: Synthetic XML Document Structure.

表 3.2.5 に示す実データに基づく XML 文書（realistic doc）は，実際に動作しているソフトウェアや，そこで用いられるデータ形式を元に構成した XML 文書であり，それぞれ以下のように構成されている．

kernel_doc: Linux kernel 2.6.39.3 のソースツリーのディレクトリ構造を XML 形式で表現したものである．kernel_doc に関しての実験結果は例えば，圧縮されたアーカイブファイルに欠損がないかを判定したり，ネットワークストレージを利用したファイルバックアップサービスにおいて，バックアップの開始前に，バックアップ元とすでに存在する過去のバックアップデータのディレクトリ構造とを比較することで差分を得，更新の必要のあるファイルのみをピックアップするなどの機能に利用可能である．

stock_doc: MySQL データベースに格納された，10,000 エントリのダミーの株式情報を XML 形式で出力したものである．stock_doc は，様々

な分野で広く利用されるデータベースに格納されるデータに対しての処理を想定して採用した．例えば，株式情報のやり取りをリアルタイムに処理していく，ネット通販のカタログデータの内，ユーザの要求に応じて，必要な物だけをピックアップしたり，適切に並び替えるなどの中間処理への応用が考えられる．

scala_doc: “The Scala Language Specification Version 2.8”（`http://www.scala-lang.org/`; 2011 年 7 月 15 日確認）を XML 形式に変換したものである．元の文書は PDF 形式であるが，それを Open Office にて OpenDocument Format[45, 46] に変換し，その中に格納されている XML 形式のデータを利用した．scala_doc での実験結果は例えば，ネットワーク上でやり取りされる文書データが，機密情報を含んでいないかどうかの検査アプリケーションや，ニュースや Web ページの要約を動的に生成するなどのサービスへの応用が期待できる．

表 3.2.5: XML Document Characteristics (realistic doc).

	kernel_doc	stock_doc	scala_doc
Tag set count (Empty tag count)	2,255 (36,708)	66,717 (146)	26,738 (1,206)
Line count	41,219	78,010	72,014
File size [Kbytes]	3,891	2,389	2,959

3.3 評価指標

本実験における XML 文書への分散処理の特徴を述べるために，下記の 6 種類の評価指標を定義する．これらの評価指標は，各ノード（の各スレッド）ごとの性質を示す Node performance Indicator（Node I）と，各ノードも含めたシステム全体の性質を表す System performance Indicator（System I）の 2 種類に分類される．

Job Execution Time（System I）: アプリケーションが，ある XML 文書の処理に要する時間である．システム内ではいくつかのノードが並列に動作するが，Job Execution Time は，最初のノードが処理

を始めた時（StartNodeがXML文書の読み込みを始めた時）から，最後のノードが処理を終えるまで（EndNodeで処理結果を受け取るまで）の時間である．このJob Execution Timeは，並列分散処理における最も基本的な評価指標であり，この値を小さくすることが重要な目標の一つである．しかし，本評価アプリケーションでは，データ転送のオーバーヘッドの影響により，単体の計算機でXML文書を処理する場合よりも，総合的にはXML文書の処理に多くの時間がかかることがわかっている．ただしその場合でも，エンド付近に配置されるサーバやクライアントなどの計算資源で行うべき処理を，中継ノードに割り当てることとなるので，これらの計算機が処理に要するコストは減少する．

Node Thread Working Time（Node I）： 各ノードでタグチェック処理やデータの転送，ファイルの読み込みと分割処理などの処理に費やした時間を表している．Node Thread Working Timeは，データ転送時における待ち受け時間などのスレッドが待機している時間を含まない．例えば，第2章にて述べたRelayNode（図2.3.2）においては，各スレッドが受信・処理・送信に要した時間の合計となる．この時，ReceiveThreadがデータの受信に200msecを費やし，TagCheckThreadがデータの処理に100msecを，SendThreadがデータの送信に300msec費やす場合，Node Thread Working Timeはそれらを合計した600msecとなる．これらのスレッドが同時に動いていてもNode Thread Working Timeは600msecのままとなる．また各ノードごとのNode Thread Working Timeを合計して，**System Thread Working Time（System I）**を定義する．この評価指標は，中継ノードでのストリーミング処理に関して，処理量を定義するための指標である．

Node Active Time（Node I）： Node Active Timeは，各ノードの各スレッドの動作時間を表す．これは各ノードにおける，あるスレッドが最初に処理を始めた時間から，全てのスレッドが最後の処理を終了するまでの時間である．例えば，RelayNodeでは，ReceiveThreadが最初のデータ受信を始めた時間から，SendThreadが最後のデータを送信するまでの時間である．Node Active Timeは，データ転送時における待ち受け時間などによって，スレッドが待機している場合はこの待機時間を含む．先と同じく第2章にて述べた図2.3.2

に示す RelayNode を例に挙げる．ReceiveThread がデータの受信に 200msec を費やし，TagCheckThread がデータの処理に 100msec を，SendThread がデータの送信に 300msec 費やすとする．この時，TagCheckThread が ReceiveThread が全てのデータの受信をしてから動き，SendThread が TagCheckThread が全てのデータを処理してから動く場合は Node Active Time は 600msec となる．一方，これら全てのスレッドが同時に動くなら Node Active Time は 300msec となる．また，各ノードごとの Node Active Time を平均して **System Active Time（System I）**を定義する．この評価指標は，中継ノードでのストリーミング処理に関して，処理に要する時間を定義するための指標である．

Node Processing Time（Node I）： 各 RelayNode と EndNode の Processing Thread にて，XML 文書への文法チェック処理に要した時間である．また，各ノードごとの Node Processing Time を平均して，**System Processing Time（System I）**を定義する．この評価指標は，データの送信・受信などの転送時間を除いた，XML 文書への処理時間のみを評価するための指標である．

Parallelism Efficiency Ratio（System I）： このインジケータは，“$SystemThreadWorkingTime/SystemActiveTime$” で定義され，処理効率比率を示す．この評価指標では，ある処理を行うために要した処理量を，その処理のために要した時間で割ることで，XML 文書の処理のために，効率的に資源を使用することができたかを評価する．

Node Buffer Access Time（Node I）： RelayNode や EndNode，MergeNode はデータの転送途中に一時的にそれらを格納するのに用いるバッファを有する．Node Buffer Access Time は，各ノードで XML 文書の処理の際に，バッファへの要素の追加，バッファからの要素の取得や削除などに費やした時間を示す．RelayNode（第 2 章，図 2.3.2）のように，ノード中のバッファには複数のスレッドが同時にアクセスするが，それらが競合する場合には，他のスレッドは待機しなければならない．Node Buffer Access Time は，バッファアクセスが競合した時のロックによる待ち時間も含む．また，各ノードごとの Node Buffer Access Time を合計して，システム全体で一つの XML 文書を処理する際にバッファアクセスに費やした時間を表す **Sys-**

tem Buffer Access Time（System I）を定義する．これらのバッファに関する評価指標は，第 6 章にて行うバッファ性能の改善に関する実験の評価に用いる．

第4章　XML文書へのストリーミングデータ処理の基本特性の評価

4.1　はじめに

XML技術はあらゆる場所で用いられる汎用的な技術となっており，特にWebサービス間の柔軟な連携などの分野での活用が進んでいる．これに伴いこれらのWebサービスを提供するサーバにおけるXMLデータへの処理機会が増加し，とりわけ大規模なXMLデータ処理のためには効率的な分散処理技術が必要となる．近年，このような分散XMLデータ処理技術について様々な方法が研究されているが，ネットワークサービスにおける分散XMLデータ処理について，整形式判定（well-formedness grammar checking）・妥当性検証（validation grammar checking）・フィルタリングなど，様々なWebシステムで共通の処理に着目したPASS（Prefix Automata SyStem）-Node[1] を用いた処理方法が提案されている．このPASS-Nodeは送信元となるクライアントから送信先となるサーバへXML文書を送信する際，その通信経路の途上となるネットワーク配置される．この時XML文書は，いくつかの処理単位に分割されて送信され，PASS-Nodeの配置や処理の割り当ての状況に応じてパイプライン並列またはデータ分割並列で前処理を施す．これにより，従来データの送信先となるサーバで行われていた処理を，ネットワークの途上の処理ノードにオフロードすることで，サーバでの処理負荷の軽減などの恩恵を得ることが可能となる．我々は過去にこのPASS-Nodeを実装し，特にパイプライン並列処理時のデータの分割単位とその処理割り当てに関して調査を行った [47, 48]．

本章ではこれらの調査を更に発展させ，XML文書へのストリーミングデータ処理について，

- 仮想環境にて合成XML文書（synthetic doc）を用いた場合の特徴

の評価（本章第 4.2 節）

- 実データを元にした XML 文書（realistic doc）及び実環境を用いた場合の初期評価（本章第 4.3 節)

の二つについて述べる．この時，XML 文書の妥当性検証（validation grammar checking）と XML 文書の整形式判定（well-formedness grammar checking）という二つの性質の異なる処理についての特徴を比較する．さらに，これらの評価実験をデータ分割並列処理（paralell）とパイプライン並列（pipeline）双方において処理を行うノードの数を増減させつつ行い XML 文書へのストリーミング並列処理の特徴を明らかにする．

また本章にて述べる内容は [49, 50] に対応するものである．

4.2 基本的な特性の評価

XML 文書への分散処理の特性を明らかにするために，実験環境とノードの構成を様々に変えて評価を行う．

実験環境

本節にて用いる実験環境は

- T5440_Env（仮想環境）

の 1 種類である．この環境にて，

- XML 文書の整形式判定（Well-formedness grammar checking）
- XML 文書の妥当性検証（Validation grammar checking）

の二種類の処理を行う．また，この時用いる XML 文書は合成 XML 文書（synthetic doc）である（第 3 章 3.2 節）．

ノードの配置パターン

本章で用いるノード構成は下記に示す 4 種類である．

- Two stages pipeline (図 4.2.1)

- Two path parallelism (図 4.2.2)
- Four stages pipeline (図 4.2.3)
- Four path parallelism (図 4.2.4)

本章では，これら 4 種類のノード構成で，処理効率を評価する．

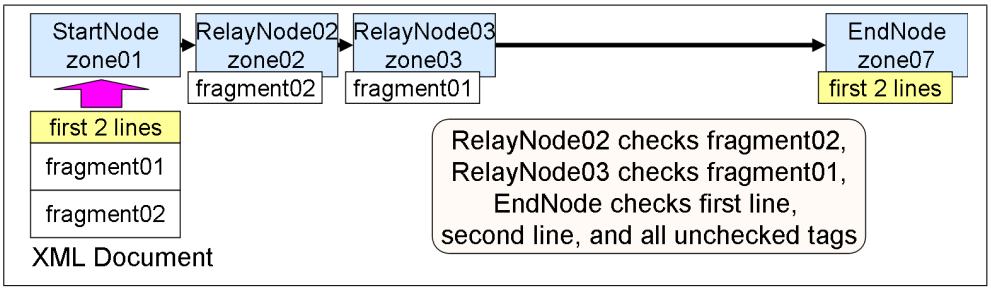

図 4.2.1: Two Stages Pipeline.

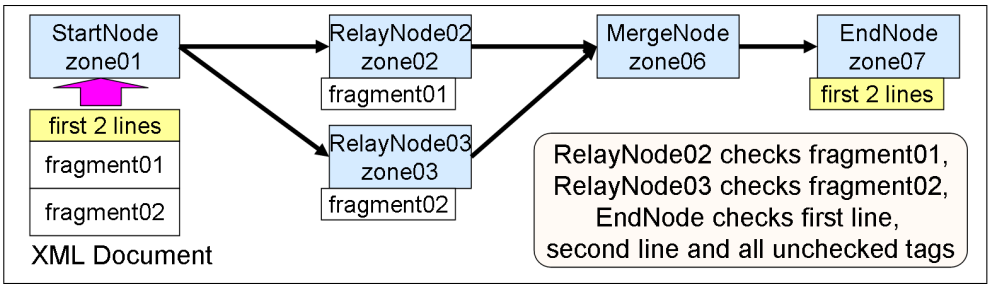

図 4.2.2: Two Path Parallelism.

まず二つの RelayNode(=二つの中間処理ノード) を用いる場合のノード構成（図 4.2.1 と図 4.2.2）について述べる．図 4.2.1 と図 4.2.2 に示すように StartNode が後続に二つの RelayNode を持つ場合，入力された XML 文書は，最初の 2 行（first two lines），fragment01，fragment02，の三つに分割される．この各フラグメントはそれぞれが，ほぼ同一のデータサイズとなるように分割される．first two lines は XML 文書におけるメタタグと，root タグを含んでいる行である．

図 4.2.1 は二つの RelayNode を用いた 2 段のパイプライン並列処理の構成である．この時分割された XML 文書とその処理に関わる情報は，StartNode から二つの RelayNode を通って，EndNode まで送られる．その際に，RelayNode02 は fragment02 を処理し，RelayNode03 は fragment01

を処理するように割り当てられる．そして，first two lines への処理と，二つの RelayNode で処理が完了しなかった全てのデータは EndNode にて処理が行われ，EndNode は最終的な処理結果を得る．

図 4.2.2 は，二つの RelayNode を用いた 2 経路のデータ分割並列処理の構成である．この時，first two lines と fragment01 及びこれらに関わる処理情報は，StartNode-RelayNode02-MergeNode の経路を，fragment02 は StartNode-RelayNode03-MergeNode の経路を通り EndNode に送られる．その際に，RelayNode02 は fragment01 を処理し，RelayNode03 は fragment02 を処理するように割り当てられる．そしてパイプライン並列処理の例と同じように，first two lines への処理と，二つの RelayNode で処理が完了しなかった全てのデータは EndNode にて処理が行われ，EndNode は最終的な処理結果を得る．

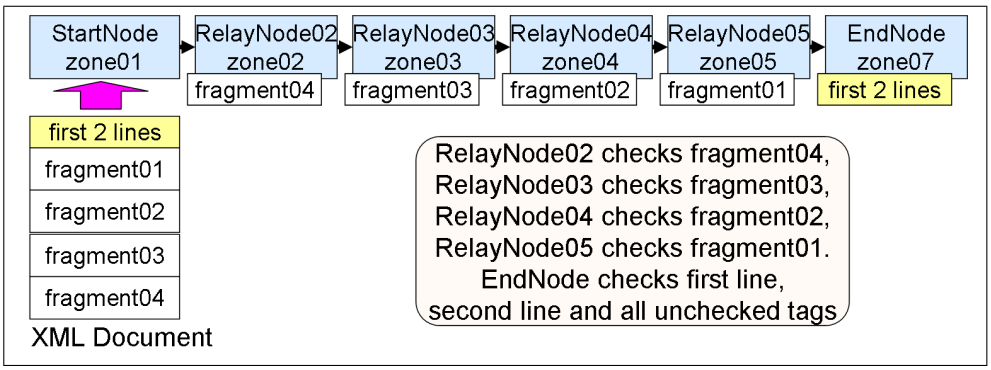

図 4.2.3: Four Stages Pipeline.

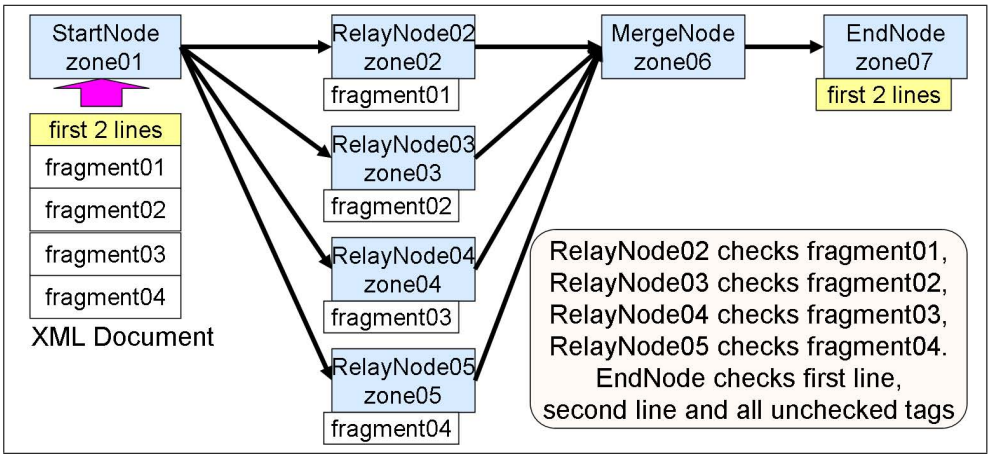

図 4.2.4: Four Path Parallelism.

図 4.2.3 と図 4.2.4 に示す四つの RelayNode(=四つの中間処理ノード) を用いる場合のノード構成では，StartNode は四つの後続 RelayNode を持つ．この時 StartNode に入力された XML 文書は，最初の 2 行（first two lines），fragment01，fragment02，fragment03，fragment04，の五つに分割される．

図 4.2.3 は四つの RelayNode を用いた 4 段のパイプライン並列処理の構成である．この時分割された XML 文書は，StartNode から四つの RelayNode を通って，EndNode まで送られる．その際に，RelayNode02 は fragment04 を処理し，RelayNode03 は fragment03 を処理し，RelayNode04 は fragment02 を処理し，RelayNode05 は fragment01 を処理するように割り当てられる．そして，first two lines への処理と，四つの RelayNode で処理が完了しなかった全てのデータは EndNode にて処理が行われ，EndNode は最終的な処理結果を得る．

図 4.2.4 は，四つの RelayNode を用いた 4 経路のデータ分割並列処理の構成である．この時，first two lines と fragment01 は StartNode-RelayNode02-MergeNode の経路を，fragment02 は StartNode-RelayNode03-MergeNode の経路を，fragment03 は StartNode-RelayNode04-MergeNode の経路を，fragment04 は StartNode-RelayNode05-MergeNode の経路を通り EndNode に送られる．その際に，RelayNode02 は fragment01 を処理し，RelayNode03 は fragment02 を処理し，RelayNode04 は fragment03 を処理し，RelayNode05 は fragment04 を処理するように割り当てられる．そしてパイプライン並列処理の例と同じように，first two lines への処理と，四つの RelayNode で処理が完了しなかった全てのデータは EndNode にて処理が行われ，EndNode は最終的な処理結果を得る．

XML ストリーミング処理の基本的な特性

7 種類の合成 XML 文書（synthetic doc）について T5440_Env を実験環境として XML 文書への並列ストリーミングデータ処理を行った結果を示す．これらのグラフはぞれぞれ

- 図 4.2.5 と図 4.2.6 は Job Execution Time を，
- 図 4.2.7 と図 4.2.8 は System Active Time を，
- 図 4.2.9 と図 4.2.10 は System Processing Time を，
- 図 4.2.11 と図 4.2.12 は Parallelism Efficiency Ratio

を示す．これらは XML 文書への処理を 22 回行った値の平均を取っている．それぞれのグラフにおいて，横軸はノード構成と XML 文書へ行った文法チェック処理の種類を表す．横軸で用いられている各表記は下記を意味する．

- `PIP_wel` :Pipeline and Well-formedness grammar checking
- `PAR_wel` :Parallel and Well-formedness grammar checking
- `PIP_val` :Pipeline and Validation grammar checking
- `PAR_val` :Parallel and Validation grammar checking

図 4.2.5 と図 4.2.6 に示す Job Execution Time において，全ての合成 XML 文書（synthetic doc）についてパイプライン並列処理（PIP_wel，PIP_val）よりもデータ分割並列処理（PAR_wel，PAR_val) の方が早く処理が終了していることがわかる．さらに処理ノードが増えた場合の，処理時間の減少についても，パイプライン並列処理よりもデータ分割並列処理の方が大きな効果が出ていることがわかる．データ分割並列処理においては，StartNode は分割した XML 文書のそれぞれを同時に読み込み，後続の RelayNode に送信し，各 RelayNode はこの分割された XML 文書を受信する．同様に，MergeNode の持つ Receive/SendThread もそれぞれの RelayNode から分割された XML 文書とそれに関わるデータを受信し，順番通りに並べ替えるシリアライズ処理を行った後 EndNode に送る．図 4.2.7 と図 4.2.8 に示す System Active Time の減少にも見られる通り，これらのノードのスレッドの動作は同時に進むため，データ分割並列処理はパイプライン並列処理よりも短い時間で処理を進めることができるといえる．

図 4.2.7 と図 4.2.8 の System Active Time によると，XML 文書のタグのネスト構造が深くなればなるほど，より多くの処理時間を必要とすることがわかる．これは，各 RelayNode において自身に割り当てられた XML 文書の部分において，スタックに積んだものの対応を取ることのできないタグが増加し，それらが無駄な処理となってしまうためである．そして，この完了することができなかった処理は，最終的に EndNode で行われるため，同時に EndNode の処理時間も増加することとなる．これらの無駄となってしまう処理を，XML 文書の構造や文法チェックのルールの事前解析を用いた XML 文書の効率的な分割によって解消することが期待できる．また，System Active Time では，データ分割並列処理の方がパ

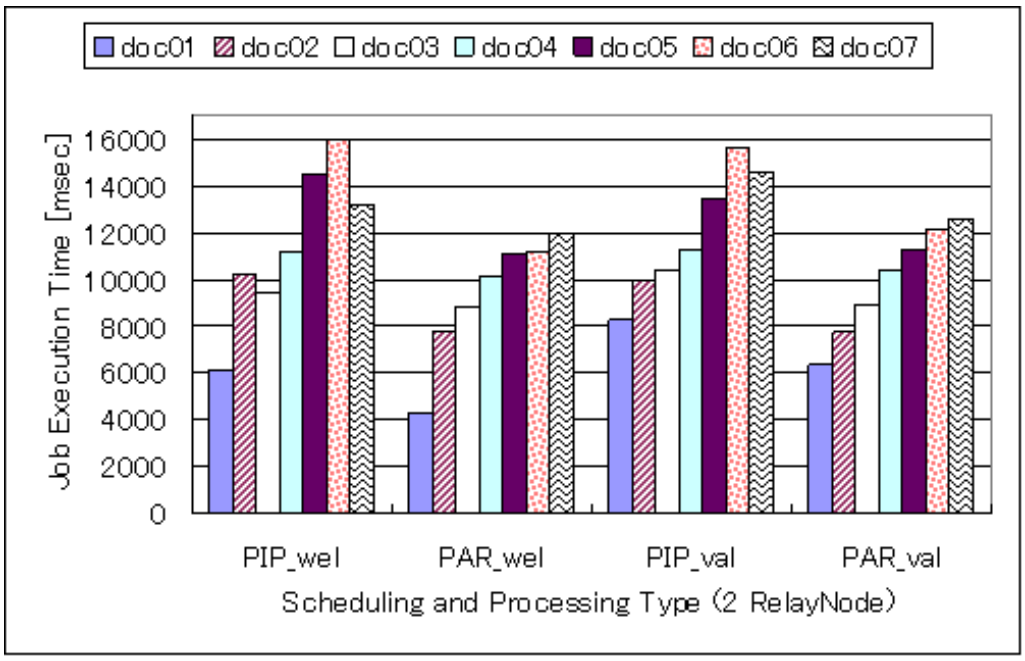

図 4.2.5: Job Execution Time (Two RelayNodes).

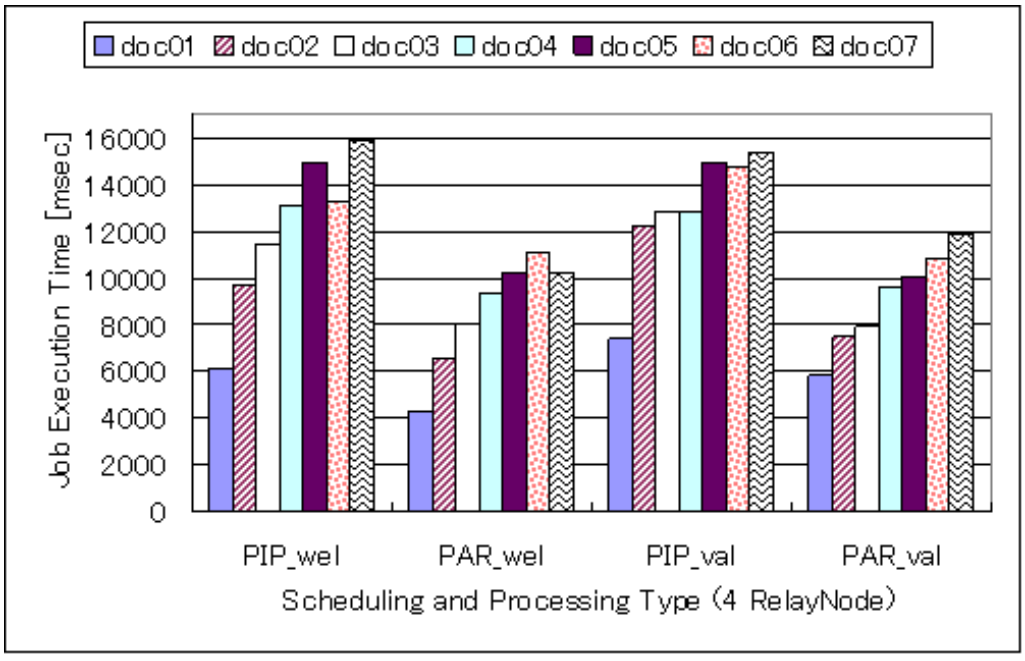

図 4.2.6: Job Execution Time (Four RelayNodes).

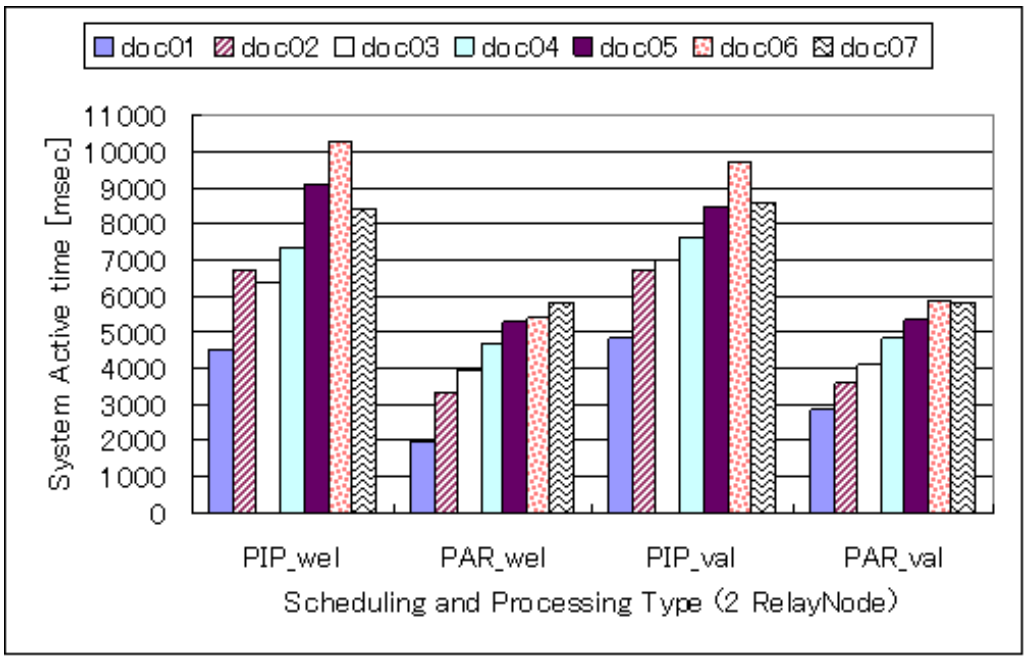

図 4.2.7: System Active Time (Two RelayNodes).

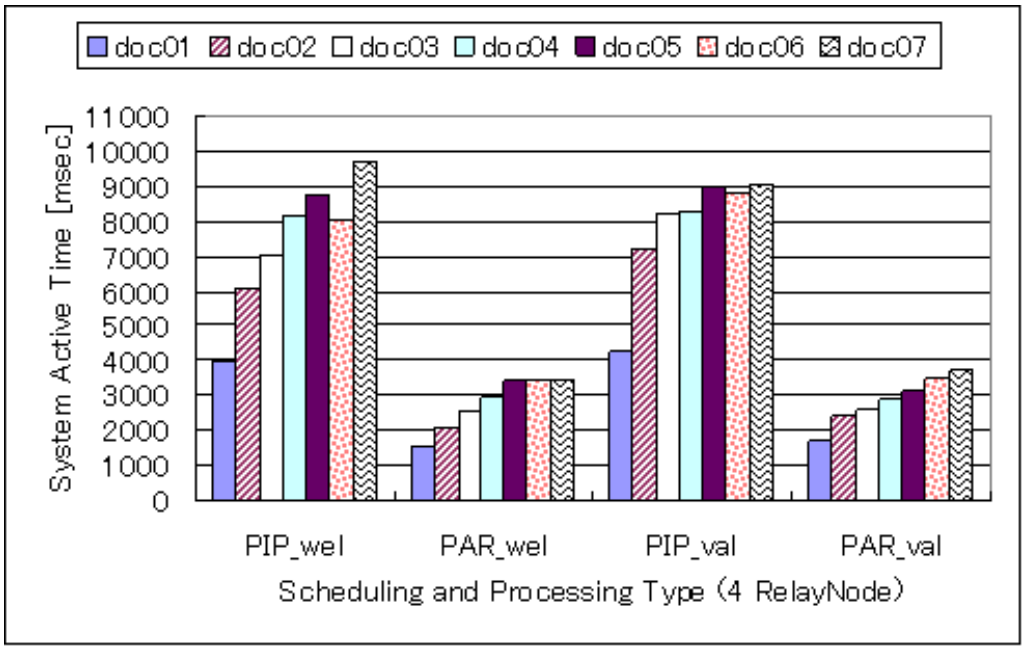

図 4.2.8: System Active Time (Four RelayNodes).

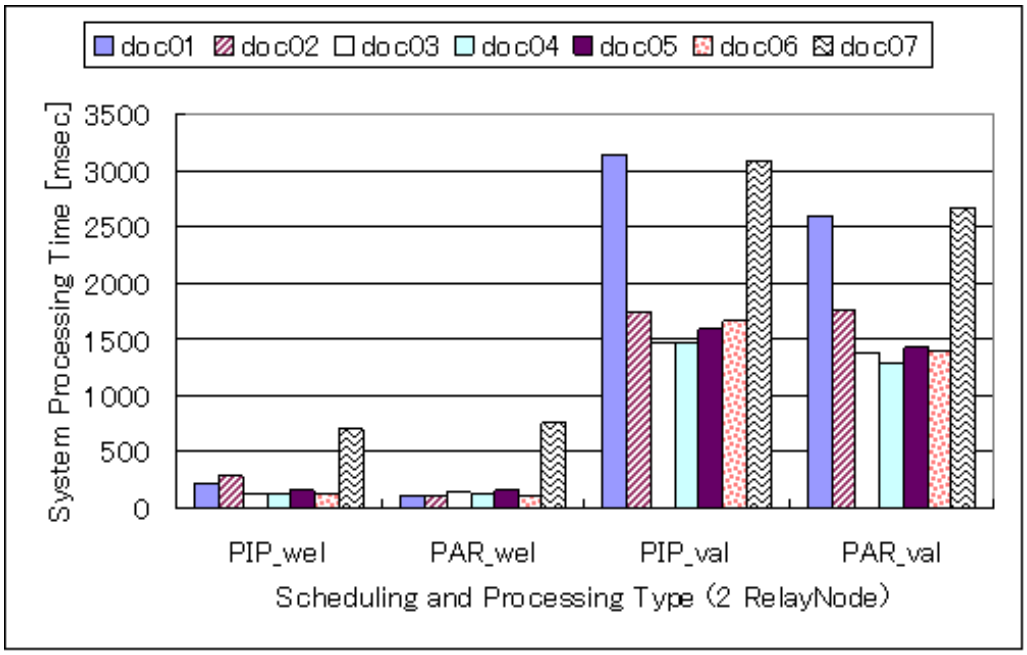

図 4.2.9: System Processing Time (Two RelayNodes).

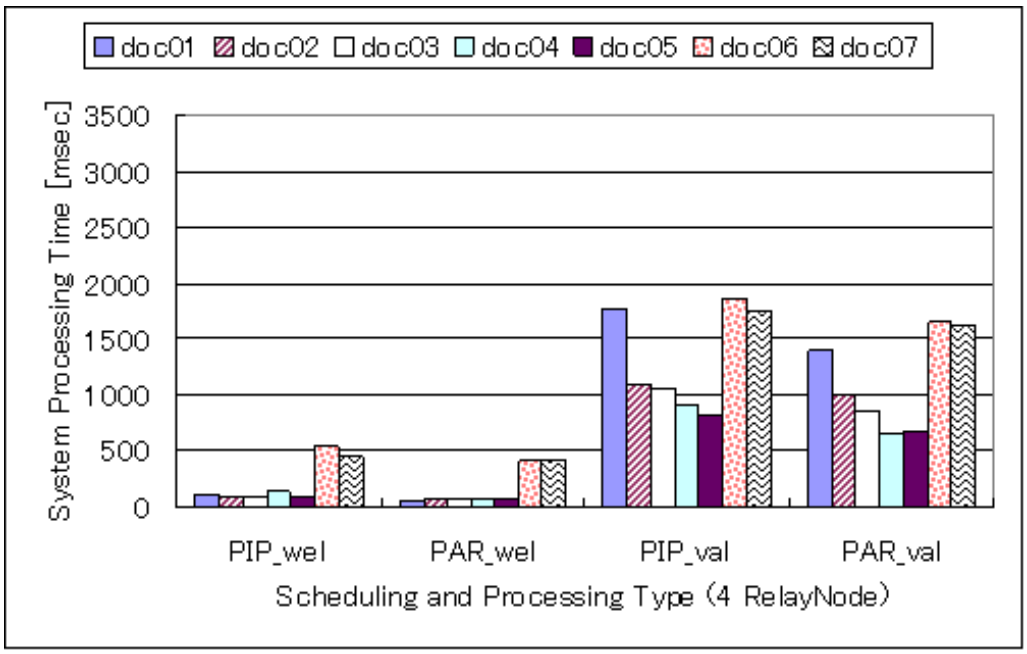

図 4.2.10: System Processing Time (Four RelayNodes).

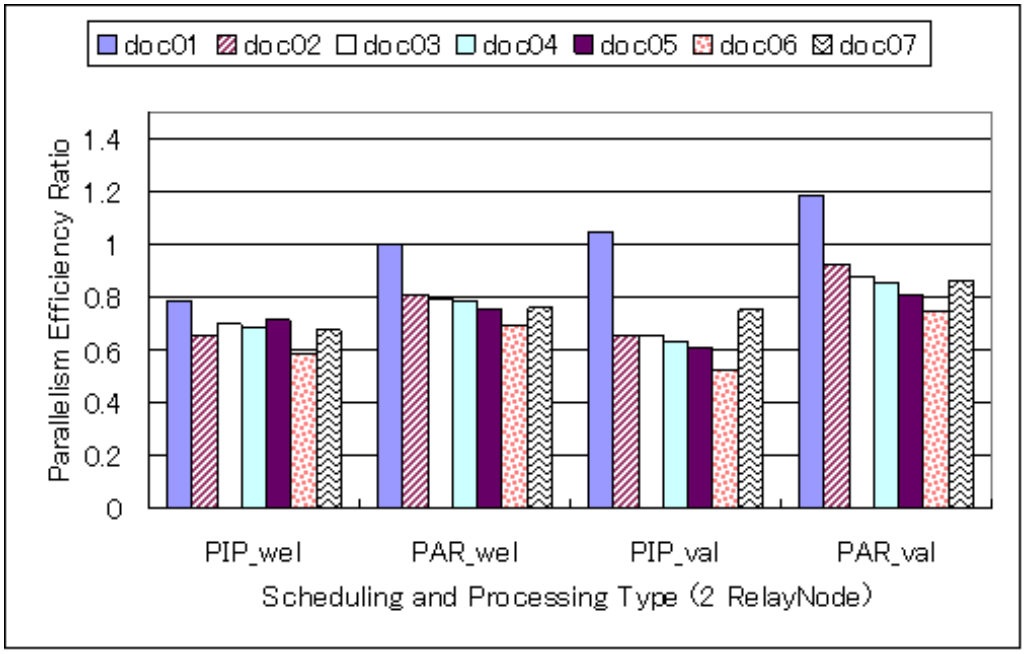

図 4.2.11: Parallelism Efficiency Ratio (Two RelayNodes).

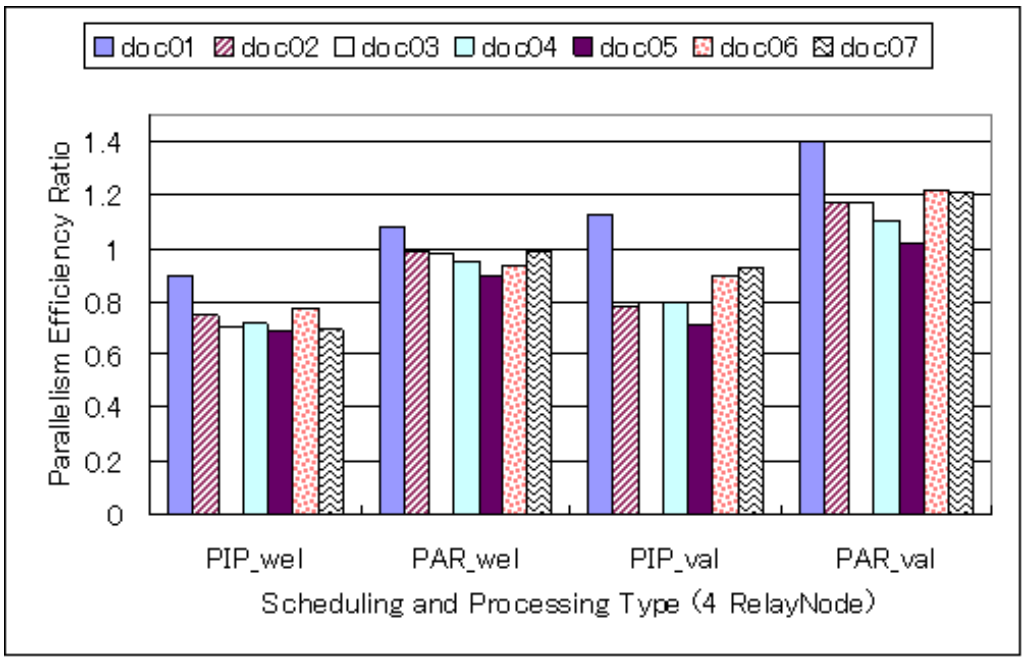

図 4.2.12: Parallelism Efficiency Ratio (Four RelayNodes).

イプライン並列処理よりも処理ノードの数の影響を強く受けていることがわかる．加えて，パイプライン並列処理において，タスクの構成によらず System Active Time は似た特徴を示す．

図 4.2.13 と図 4.2.14 は，合成 XML 文書（synthetic doc）のうち，doc01 を処理した場合の Node Active Time の詳細である．これらのグラフの中では，**SN** は StartNode を，**RN** は RelayNode を，**MN** は MergeNode を，**EN** は EndNode を意味している．図 4.2.13 は RelayNode を二つ用いた場合の図 4.2.14 は RelayNode を四つ用いた場合の Node Active Time を示している．これらの図によると，Node Active Time はパイプライン並列処理の場合よりも，データ分割並列処理の場合の方が小さくなることが分かる．

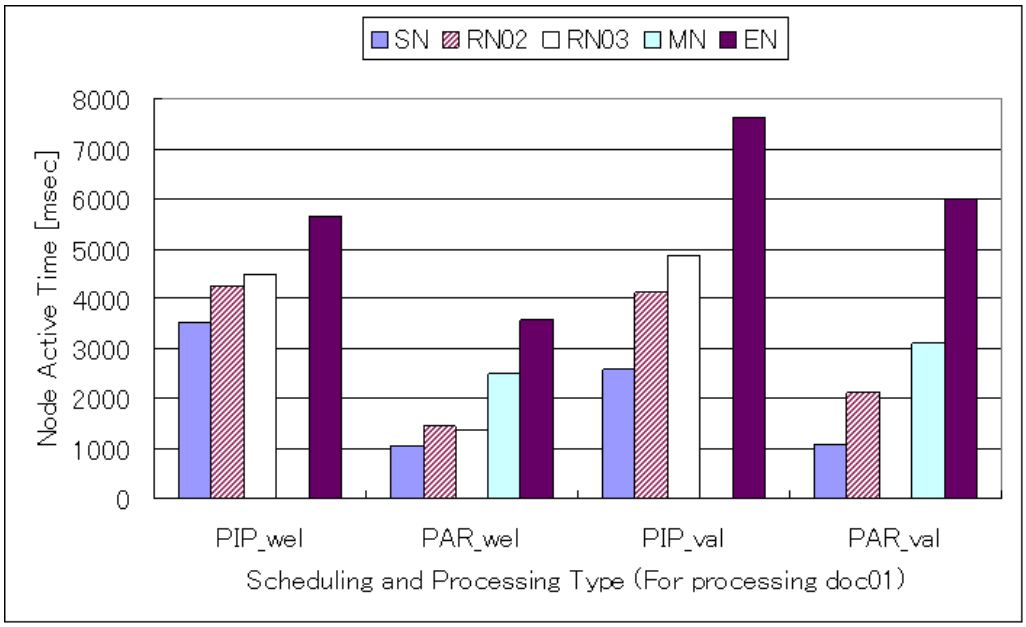

図 4.2.13: Node Active Time for Processing doc01 (Two RelayNodes).

図 4.2.9 と図 4.2.10 に示す，System Processing Time においては，わずかにデータ分割並列のほうが良いが，パイプライン並列処理とデータ分割並列処理がそれぞれ近い値を示している．これにより，先に示した Job Execution Time と System Active Time における，パイプライン並列処理とデータ分割並列処理における時間の差は，XML 文書への処理ではなく，データの送受信などに原因があることが分かる．加えて，XML 文書の整形式判定の方が XML 文書の妥当性の検証よりも早く終了している

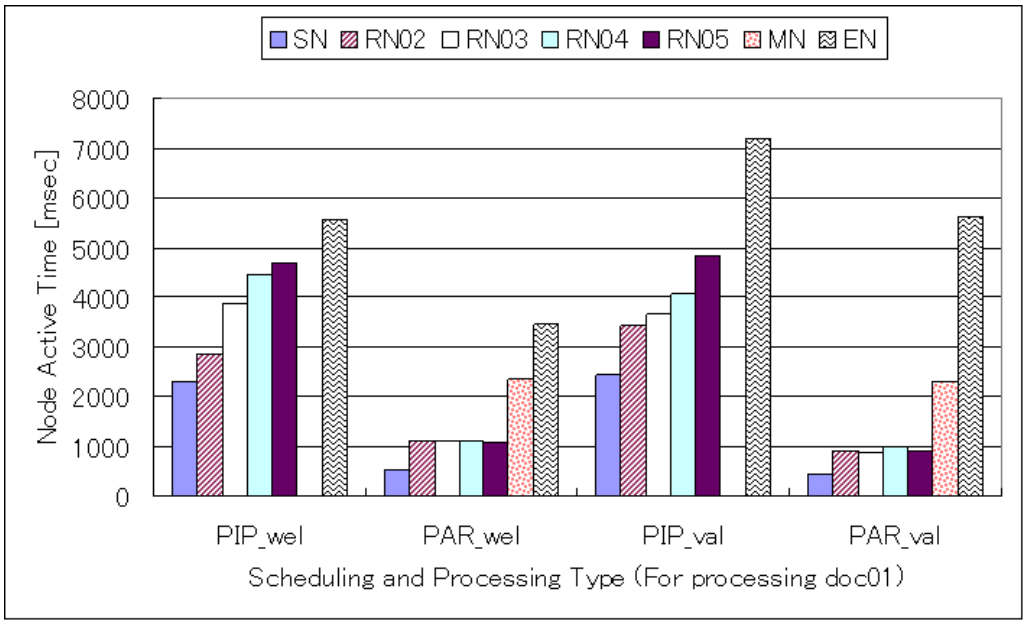

図 4.2.14: Node Active Time for Processing doc01 (Four RelayNodes).

が，これは予想された結果である．またここで示した XML 文書の処理に要する時間は，XML 文書の分割とその処理割り当てを効率的に行うことができるかどうかに強く依存している．

測定した Parallelism Efficiency Ratio の結果を図 4.2.11 及び図 4.2.12 に示す．図 4.2.11 及び図 4.2.12 より，データ分割並列処理の方がパイプライン並列処理よりも効率的に処理できていることが分かる．これは，データ分割並列処理においては，複数のスレッドが，個々に分割された XML 文書の処理を同時に行うことができるからである．

図 4.2.15 と図 4.2.16 に，二つまたは四つの RelayNode を用いて合成 XML 文書（synthetic doc）のうち doc01 を処理した場合の Node Thread Working Time の詳細を示す．これらのグラフより，XML 文書の妥当性検証の方が整形式判定よりもより多くの時間を必要とすることやパイプライン並列処理では各ノードでの処理により多くの時間が必要な事が分かる．さらに，パイプライン並列処理（**PIP**）とデータ分割並列処理（**PAR**）とを比較すると，パイプライン並列処理では各ノードでの処理により多くの時間が必要な事が分かる．そのため，XML 文書へのストリーミング並列処理において，パイプライン並列処理はデータ分割並列処理よりも非効率であると言える．この傾向は，ここで示していない他の合成 XML

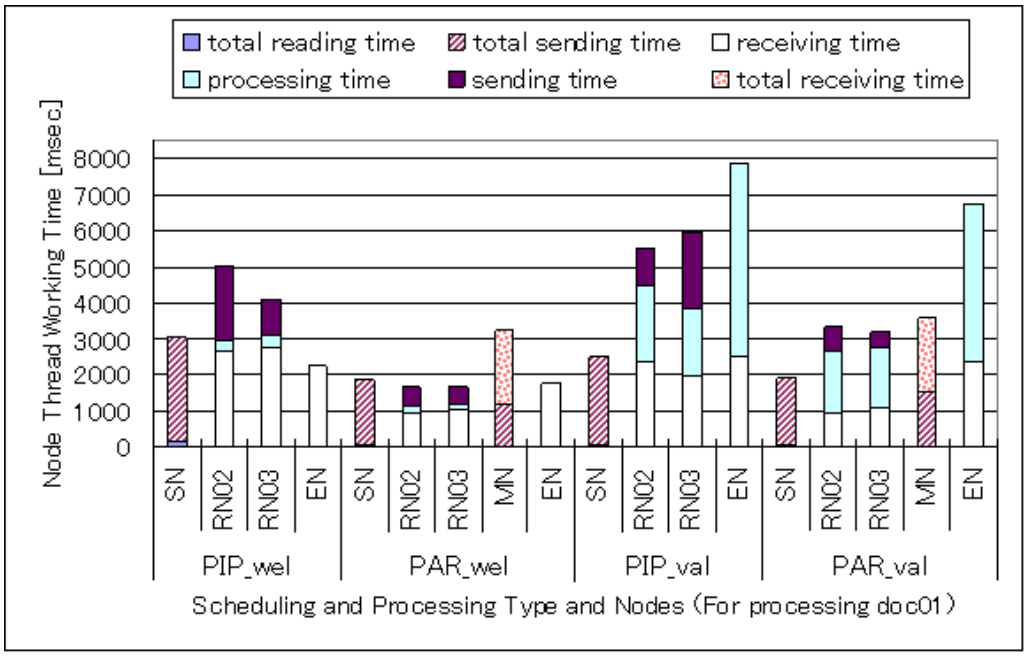

図 4.2.15: Node Thread Working Time for Processing doc01 (Two RelayNodes).

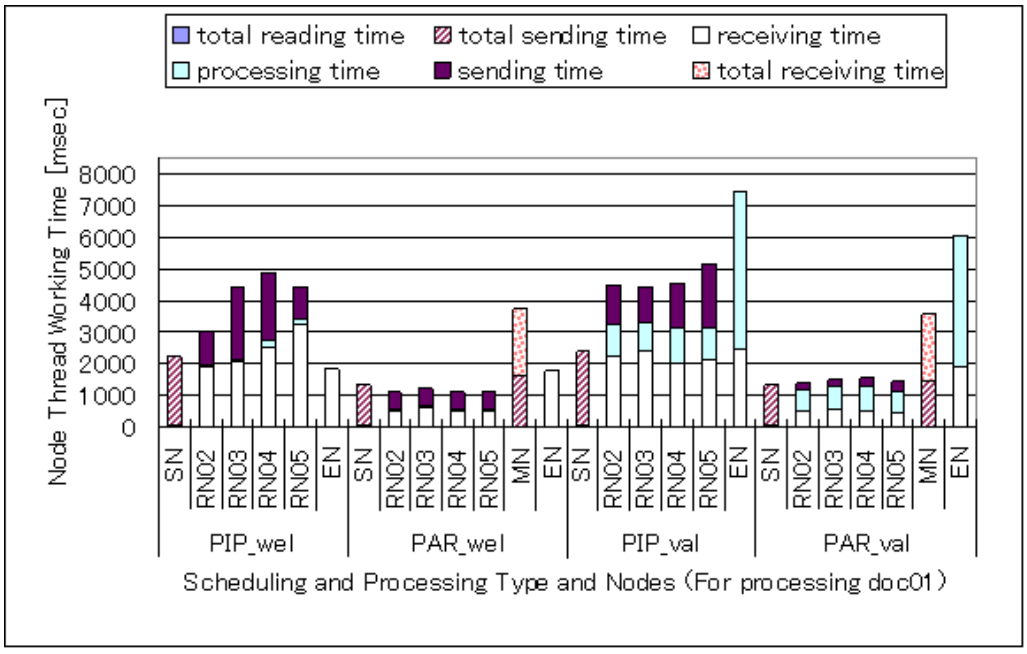

図 4.2.16: Node Thread Working Time for Processing doc01 (Four RelayNodes).

文書（synthetic doc）においても同様であった．最後に，ここまでの結果を表 4.2.1 にまとめる．

表 4.2.1: Distributed XML Processing Characterization Summary.

<table>
<tr><th rowspan="2"></th><th colspan="4">Two RelayNodes</th><th colspan="2">Four RelayNodes</th><th colspan="2">Number of RelayNodes</th></tr>
<tr><th colspan="2">Pipeline</th><th colspan="2">Parallel</th><th>Pipeline</th><th>Parallel</th><th>Pipeline</th><th>Parallel</th></tr>
<tr><td>Job execution time</td><td colspan="6">Parallel is better</td><td>No change</td><td rowspan="2">Reduces</td></tr>
<tr><td>System active time</td><td colspan="3">Parallel is better</td><td>Similar to two RN</td><td>Smaller than two RN</td><td>No effect</td></tr>
<tr><td>System processing time</td><td colspan="6">Parallel is better</td><td colspan="2">Reduces</td></tr>
<tr><td>Parallelism efficiency ratio</td><td colspan="3">Parallel is slightly better</td><td colspan="3">Parallel is better</td><td>Slightly better</td><td>Better</td></tr>
</table>

4.3 実データ・実環境での初期評価

本節では，前 4.2 節において行った仮想環境で合成 XML 文書（synthetic doc）を用いた実験に加えて実環境で実データに基づく XML 文書（realistic doc）を用いた場合の初期評価を行う．

本節にて用いる実験環境は

- X4640_Env（仮想環境）
- PC_Env（実環境）

の二種類である．これらの環境にて，

- XML 文書の整形式判定（Well-formedness grammar checking）及び，
- XML 文書の妥当性検証（Validation grammar checking）

の二種類の処理を行う．さらに，

- 合成 XML 文書（synthetic doc）
- 実データに基づく XML 文書（realistic doc）

の二種類の XML 文書を用いた実験を行い，前節との比較を行う．

評価:実データ・実環境での初期評価

本項では，前節の実験に加えて実データに基づく XML 文書を用いた評価と，性能の異なる実験環境（仮想環境:X4640_Env，実環境:PC_Env）を用いた場合の初期評価を行う．図 4.3.17 は PC_Env にて二つの RelayNode を用いて合成 XML 文書（synthetic doc）を処理した場合の Job Execution Time を示している．図 4.3.17 と図 4.2.5 を比較すると，T5440_Env は PC_Env の約 10 倍の時間を示している．詳細な検証を行ったところ，この差は CPU の性能（クロック数）によるものであり，この検証においても PC_Env と T5440_Env ではやはり約 10 倍の性能差があった．

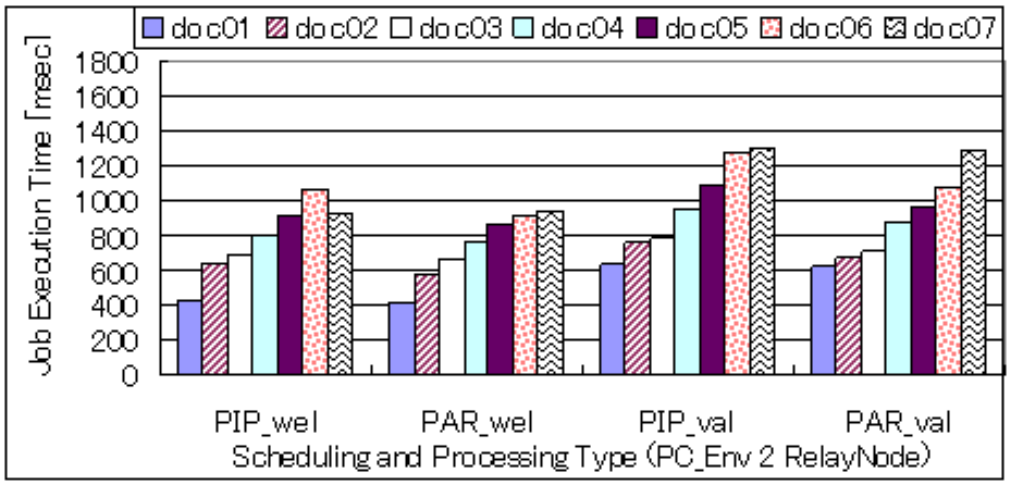

図 4.3.17: Job Execution Time of Processing Synthetic Documents (2 RelayNode in PC_Env).

これとは対照的に，図 4.3.18 に示す X4640_Env と PC_Env で二つの RelayNode を用いて，実データに基づく XML 文書（realistic doc）を処理した場合の Job Execution Time は，X4640_Env と PC_Env とでそれぞれ近

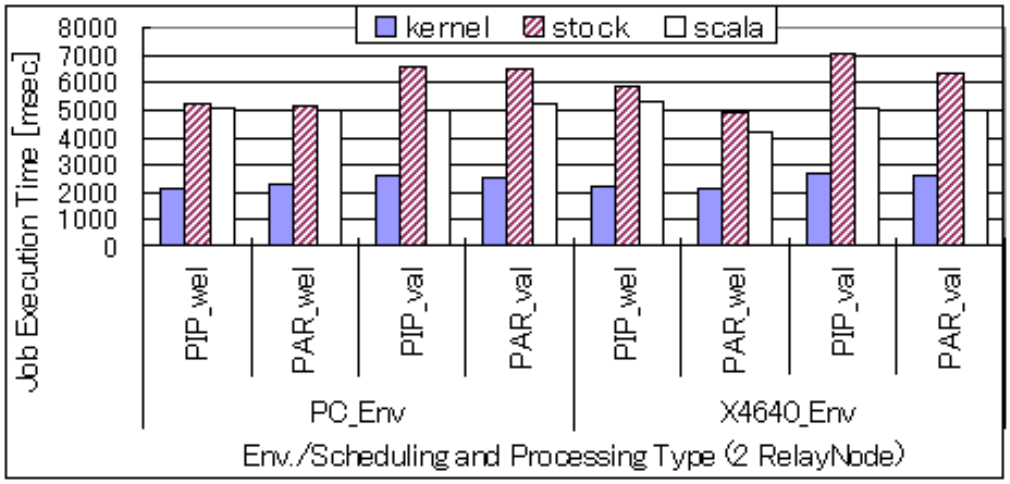

図 4.3.18: Job Execution Time of Processing Realistic Documents (PC_Env and X4640_Env).

い値を示していた．System Processing Time や System Node Active Time などの他のインジケータでも同様の傾向を示していた．これらは，実環境と仮想環境における XML 文書へのストリーミングデータ処理においてその差異が小さいことを表している．加えて，実データに基づく XML 文書（realistic doc）のうち stock doc がもっとも高い値を示しているが，これは stock doc が最も多くのタグを有しているためである．すべての実験において，Job Execution Time，System Active Time，System Processing Tiem は，処理すべきタグの数に比例して増加していく．kernel_doc は stock_doc よりも大きなファイルであるが，処理に要する時間は少ない．これは XML 文書への分散処理の効率が，そのファイルサイズよりも含まれるタグの数やその構造に影響されることを示している．

4.4 まとめ

本章では，パイプライン並列とデータ分割並列の 2 種類の XML 文書への分散処理について評価を行った．各ノードにおいて処理を行わない部分の XML 文書についても転送を行う必要がありオーバーヘッドが発生するため，概して，パイプライン並列処理はデータ分割並列処理よりも非効率である．また，両方の処理形態に共通して，XML 文書への分散処理の効率は XML 文書の構造とその分割の仕方に影響を受けることがわかっ

た．そのため，非効率な XML 文書の分割は非効率な結果を生じるが，最適な XML 文書の分割とその処理割当今後の研究課題である．加えて，本論文では XML 文書に対する整形式判定と妥当性の検証の二つの処理を評価した．一般的に妥当性の検証は整形式判定よりも多くの処理時間を必要とし，実験においても同様の結果であった．また我々はさらに，仮想環境と実環境を用いた場合の初期評価も行った．これらは仮想環境・実環境ともに似た特性を示すことが予想される．これに付いては次章にてより詳細な評価を行う．

第5章　より実際的な環境を想定した特性の評価

5.1　はじめに

XML 技術は汎用的な技術であり，互いに異なる環境が当たり前ともいえるネットワーク上の Web サービス間の連携など様々なシーンで活用されており，大規模な XML データ処理のためには効率的な分散処理技術が期待され研究が進んでいる．また，仮想化技術もまた急速な発達を見せており，資源管理とその利用の効率化・電力使用の低減・サービス運用コストの縮小・耐故障性の向上などに重要な貢献をもたらしている．この結果，Web サービスを展開するプラットフォームとして，これまでのような物資資源上でのものだけでなく，仮想化技術を適用した仮想資源上で展開されるものとの双方が想定されるようになっている．近年，XML データへの分散処理技術について様々な方法が研究されているが，ネットワークサービスにおける分散 XML データ処理について，整形式判定（well-formedness grammar checking）・妥当性検証（validation grammar checking）フィルタリングなど，各種 Web サービスで共通した XML 文書処理に着目した PASS（Prefix Automata SyStem）-Node[1] が提案されている．この PASS-Node を用いた処理手法では，クライアントからサーバへ XML 文書を送信する際，XML 文書をいくつかの処理単位に分割して送り，通信経路の途上に配置された PASS-Node にて，本来は送信先となるサーバで行うべき処理の前処理を行うことで，データの送信先となるサーバでの処理負荷の軽減などを実現している．この前処理の形態としては，ノードの配置や処理の割り当てなどの状況に応じてパイプライン並列またはデータ分割並列を組み合わせた方法が選択可能である．

本章では前章となる第 4 章（[49, 50] の内容）にて初期評価を行った

- 実環境及び仮想環境を用いた場合の特性の比較
- 合成 XML 文書及び実データに基づく XML 文書を用いた場合の特

性の比較

の二つを組み合わせたより詳細な評価を行ことで，XML 文書のストリーミング並列処理について，より実際的な環境を想定した特性を明らかにする．

また本章にて述べる内容は [51] に対応するものである．

5.2 実験環境

本章にて用いる実験環境は

- X4640_Env（仮想環境）
- PC_Env（実環境）

の 2 種類である．これらの環境にて，

- XML 文書の整形式判定（Well-formedness grammar checking）及び，
- XML 文書の妥当性検証（Validation grammar checking）

の 2 種類の処理を行う．また，この時用いる XML 文書は

- 合成 XML 文書（synthetic doc）
- 実データに基づく XML 文書（realistic doc）

である．これらの特徴の比較を前章と同様に，データ分割並列またはパイプライン並列，処理ノード数の増減，2 種類の文法チェック（整形式判定または妥当性の検証），を変更し評価を行う．

5.3 ノードの配置パターン

本章の X4640_Env と PC_Env におけるノード配置のパターンを下記に示す．これらの構成は前章とほぼ同一である．

二つの RelayNode(=二つの中間処理ノード) を用いる場合のノード構成（図 5.3.1 と図 5.3.2）では StartNode が後続に二つの RelayNode を持つ場合，入力された XML 文書は，最初の 2 行（first two lines），fragment01，fragment02，の三つに分割される．この各フラグメントはそれぞれが，ほ

ぼ同一のデータサイズとなるように分割される．first two lines は XML 文書におけるメタタグと，root タグを含んでいる行である．図 5.3.1 は二つの RelayNode を用いた 2 段のパイプライン並列処理の構成である．この時分割された XML 文書とその処理に関わる情報は，StartNode から二つの RelayNode を通って，EndNode まで送られる．その際に，RelayNode02 は fragment02 を処理し，RelayNode03 は fragment01 を処理するように割り当てられる．そして，first two lines への処理と，二つの RelayNode で処理が完了しなかった全てのデータは EndNode にて処理が行われ，EndNode は最終的な処理結果を得る．

図 5.3.2 は，二つの RelayNode を用いた 2 経路のデータ分割並列処理の構成である．この時，first two lines と fragment01，これらに関わる処理情報は，StartNode-RelayNode01-MergeNode の経路を，fragment02 は StartNode-RelayNode02-MergeNode の経路を通り EndNode に送られる．その際に，RelayNode01 は fragment01 を処理し，RelayNode02 は fragment02 を処理するように割り当てられる．そして前例と同じように，first two lines への処理と，二つの RelayNode で処理が完了しなかった全てのデータは EndNode にて処理が行われ，EndNode は最終的な処理結果を得る．

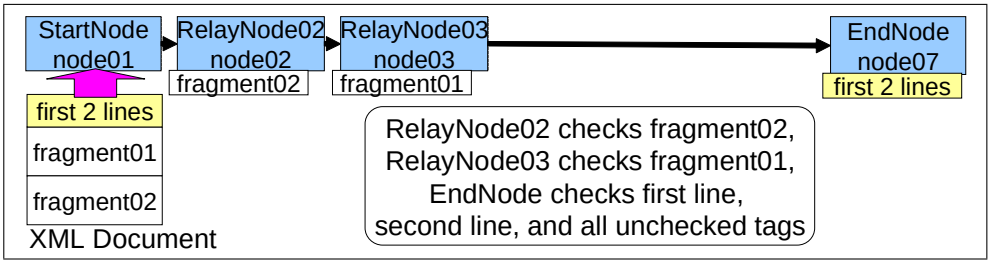

図 5.3.1: Two Stages Pipeline.

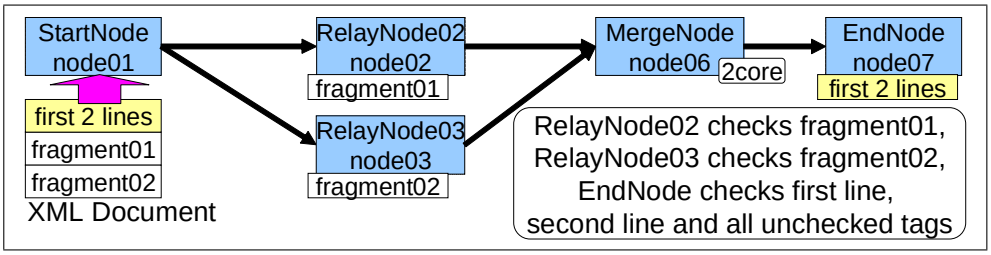

図 5.3.2: Two Path Parallelism.

また，我々は前章と同様に RelayNode を二つから四つに増やした 4 経路のデータ分割並列処理についても評価を行った．この時，XML 文書は先頭の 2 行と残り四つの fragment の計 5 つに分解され，各 RelayNode にて処理が試みられる．

5.4 実環境と仮想環境の特性の比較

7 種類の合成 XML 文書（synthetic doc）と 3 種類の実データに基づく XML 文書（realistic doc）について PC_Env と X4640_Env を実験環境として XML 文書へのストリーミングデータ処理を行った．グラフはそれぞれ，

- 図 5.4.3 と図 5.4.4 は synthetic doc に対する，図 5.4.5 と図 5.4.6 は realistic doc に対する Job Execution Time を，
- 図 5.4.7 と図 5.4.8 は synthetic doc に対する，図 5.4.9 と図 5.4.10 は realistic doc に対する System Active Time を，
- 図 5.4.11 と図 5.4.12 は synthetic doc に対する，図 5.4.13 と図 5.4.14 は realistic doc に対する System Processing Time を，
- 図 5.4.15 と図 5.4.16 は synthetic doc に対する，図 5.4.17 と図 5.4.18 は realistic doc に対する Parallelism Efficiency Ratio

を示す．これらの結果は XML 文書への処理を 22 回行った値の平均を取っている．それぞれのグラフにおいて，横軸はノード構成と XML 文書へ行った文法チェック処理の種類を表す．

図 5.4.3 と図 5.4.4，5.4.5 と図 5.4.6 に示す Job Execution Time において，データ分割並列処理はパイプライン並列処理よりも良い結果を示している．これは，パイプライン並列処理においては，処理を行わないデータの中継も行うため，それがオーバヘッドになっているためである．加えて，データ分割並列処理では RelayNode が増加した分性能は向上するが，パイプライン並列処理ではこの利点は小さくなる．さらにパイプライン並列処理において，RelayNode の増加による Job Execution Time への影響は，実データに基づく XML 文書（ファイルサイズの大きな XML 文書）の方が合成 XML 文書（ファイルサイズの小さな XML 文書）よりも強く出ている．そのため，パイプライン並列処理において大きな XML 文書を

処理する際には，多くの RelayNode を利用するほうが有利といえる．また realistic doc において，stock_doc は kernel_doc に比べてファイルサイズは小さいが，XML 文書中のタグの数は多い（表 3.2.5）．kernel_doc への処理を行った場合の Job Execution Time は，stock_doc への処理を行った場合よりも小さくなる．そのため，Job Execution Time は XML 文書のファイルサイズよりも，含まれるタグの数の影響を強く受けることがわかる．この傾向は，System Active Time や System Processing Time においても同様であった．

図 5.4.7 と図 5.4.8，図 5.4.9 と図 5.4.10 に示す System Active Time においては，パイプライン並列処理よりもデータ分割並列処理の方が良い結果を示している．また合成 XML 文書（synthetic doc）も実データに基づく XML 文書（realistic doc）も両方において，RelayNode の数を増加させると，System Active Time は減少する傾向がある．加えて，合成 XML 文書において XML 文書のタグの構造が深くなればなるほど，System Active Time は大きくなっている．これはそれぞれの RelayNode に処理を割り当てられた XML 文書中のタグについて，その処理を単一の RelayNode において完結することができなくなる事象が多くなるからである．さらに，これらの完了できなかった処理は EndNode において成されるため，EndNode の処理時間も大きくなる．このような非効率な処理を，例えば XML 文書の構造や文法チェックのルールなどを事前に参照することで効率的な処理の分割とその割当が可能になる．加えて，パイプライン並列よりもデータ分割並列処理の方が RelayNode の数の影響を受けやすく，タスクそれぞれの構成によらず System Node Active Time は似た傾向を示していた．

図 5.4.11 と図 5.4.12，図 5.4.13 と図 5.4.14 に示す System Processing Time においては，両方のタイプの XML 文書で，パイプライン並列とデータ分割並列ともに近い値を示していた．このことより，前述の Job Execution Time と System Processing Time でパイプライン並列での時間がデータ分割並列での時間よりも多くかかっていた原因が，タグに対する処理の時間ではなく各ノードでのデータの送受信に存在することがわかる．一般的に，System Processing Time は処理を行う RelayNode の数が増加した場合には減少する．しかし，XML 文書の構造とその分割方法によっては合成 XML 文書（synthetic doc）のうちの doc06 のように，RelayNode の数が少ないほうが良い場合もある．

図 5.4.15 と図 5.4.16，図 5.4.17 と図 5.4.18 に示す Parallelism Efficiency Ratio においては，データ分割並列処理がパイプライン並列処理よりも効率的であるという結果が見て取れる．これは，データ分割並列処理においては，複数のスレッドが分割された XML 文書に対して同時並列に処理を行うことが要因となっている．さらに，合成 XML 文書（synthetic doc）の方が実データに基づく XML 文書（realstic doc）よりも処理効率が高いことがわかる．加えて，合成 XML 文書（synthetic doc）においてはデータ分割並列処理に四つの RelayNode を使用するほうが，二つの RelayNode を使用する場合よりも効率が良いことがわかる．一方 realistic doc ではデータ分割並列処理もパイプライン並列処理も四つの RelayNode を使用する方が，二つの RelayNode を使用する場合よりも効率が良い．最後にここまで述べた特徴を表 5.4.1 にまとめる．

表 5.4.1: Summary of Experimental Results.

<table>
<tr><th></th><th colspan="2">Synthetic docs (smaller docs)</th><th colspan="2">Realistic docs (larger docs)</th></tr>
<tr><th></th><th>PAR vs PIP</th><th>number of RNs</th><th>PAR vs PIP</th><th>number of RNs</th></tr>
<tr><td>Job execution time</td><td rowspan="2">PAR is better</td><td rowspan="2">4 RN is better in PAR</td><td rowspan="2">PAR is better</td><td rowspan="2">4 RN is better in both PIP and PAR</td></tr>
<tr><td>System active time</td></tr>
<tr><td>System processing time</td><td>No difference</td><td>4 RN is better</td><td>No difference</td><td rowspan="2">4 RN is better</td></tr>
<tr><td>Parallelism efficiency ratio</td><td>PAR is better</td><td>4 RN is better in PAR</td><td>PAR is better</td></tr>
</table>

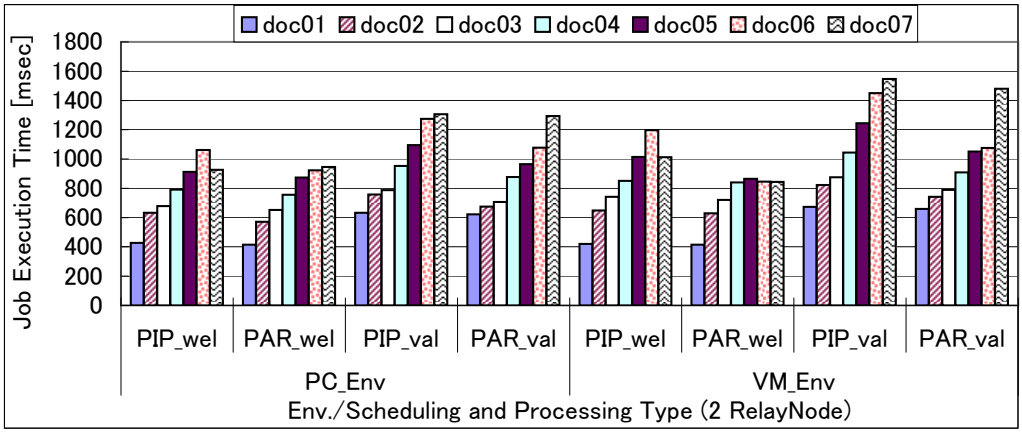

図 5.4.3: Job Execution Time (Synthetic Docs; 2RN).

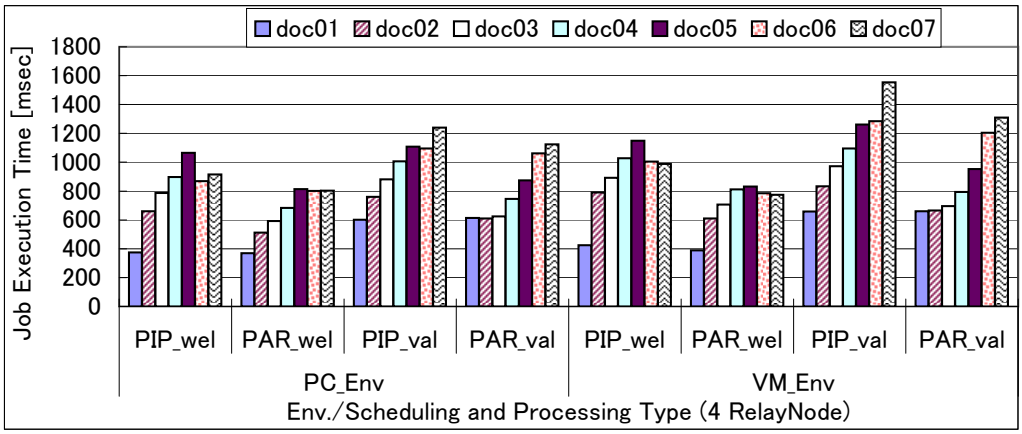

図 5.4.4: Job Execution Time (Synthetic Docs; 4RN).

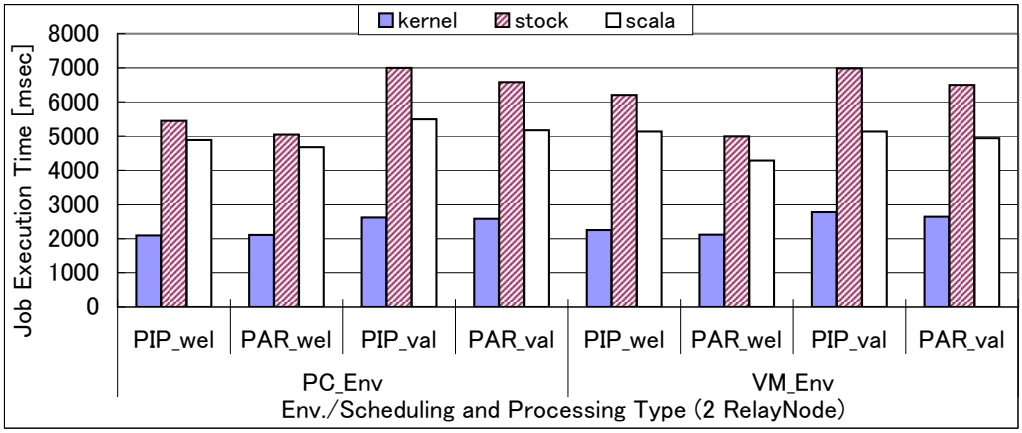

図 5.4.5: Job Execution Time (Realistic Docs; 2RN).

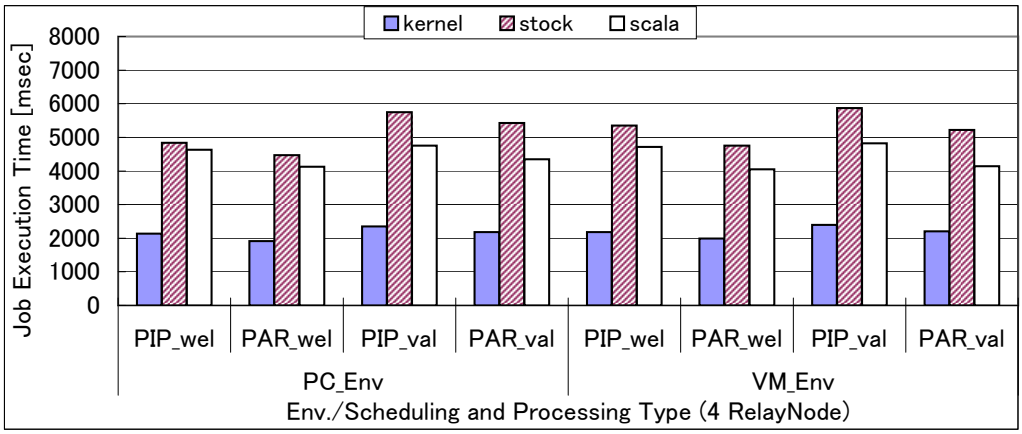

図 5.4.6: Job Execution Time (Realistic Docs; 4RN).

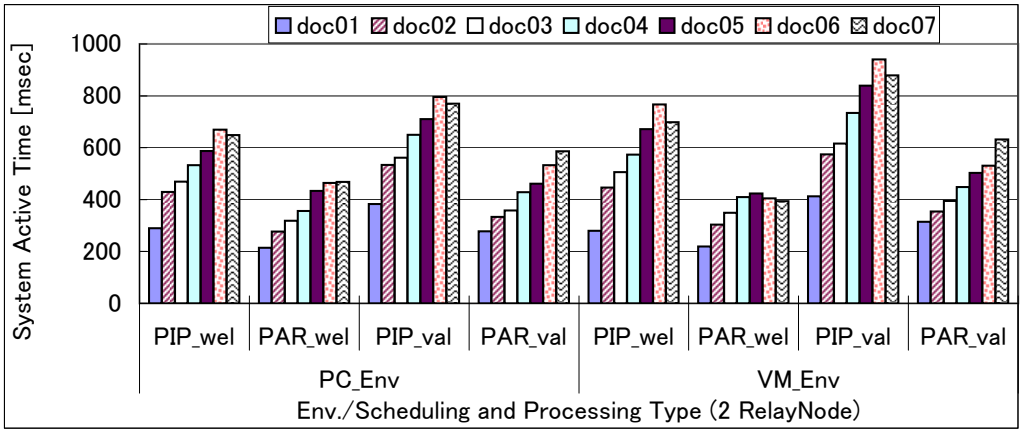

図 5.4.7: System Active Time (Synthetic Docs; 2RN).

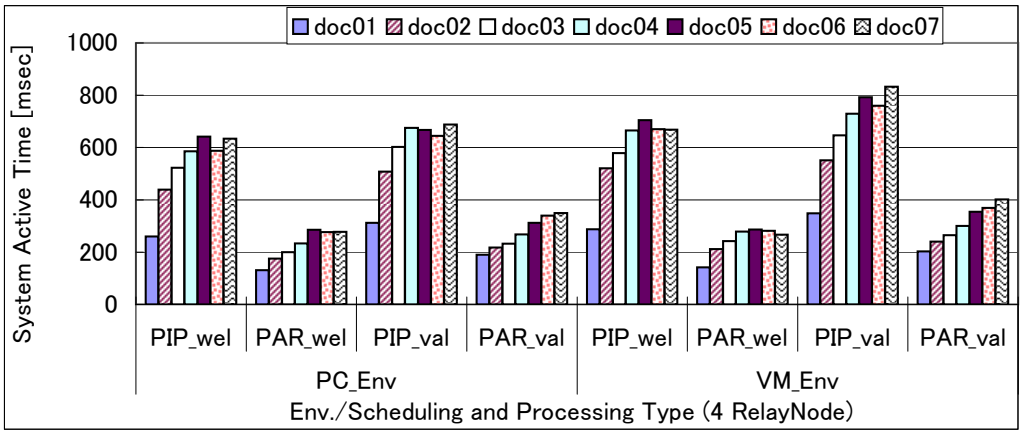

図 5.4.8: System Active Time (Synthetic Docs; 4RN).

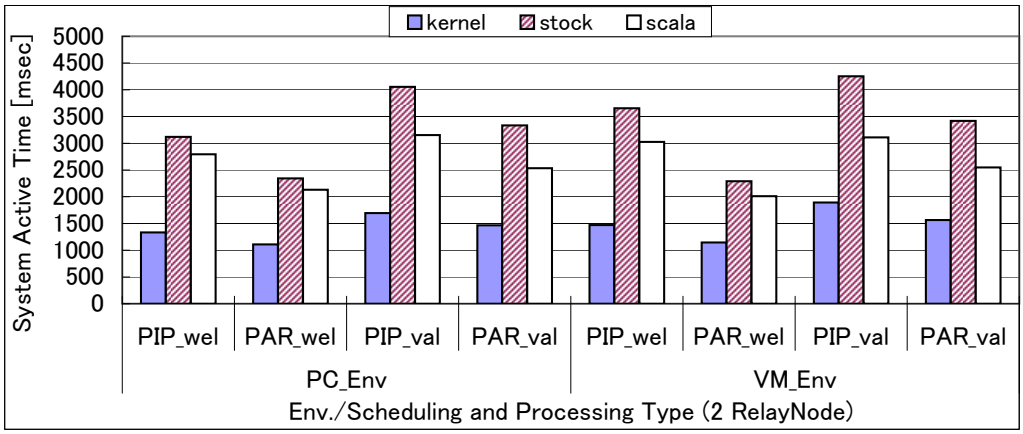

図 5.4.9: System Active Time (Realistic Docs; 2RN).

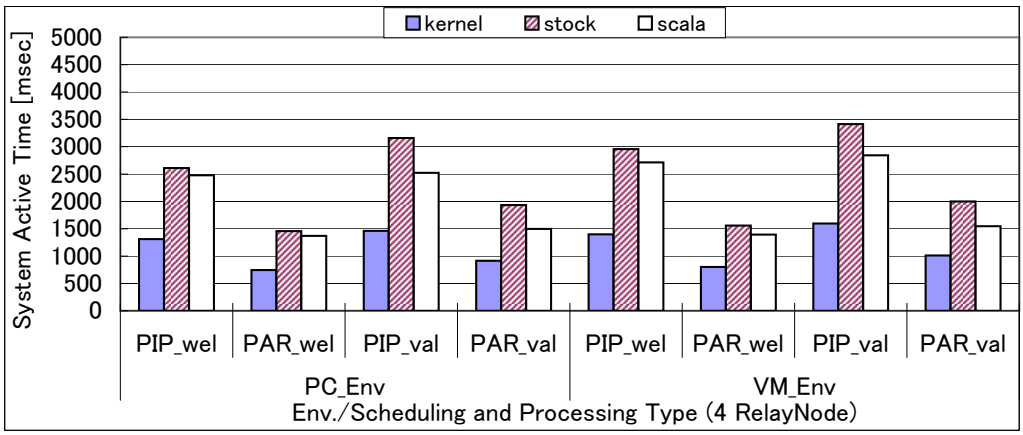

図 5.4.10: System Active Time (Realistic Docs; 4RN).

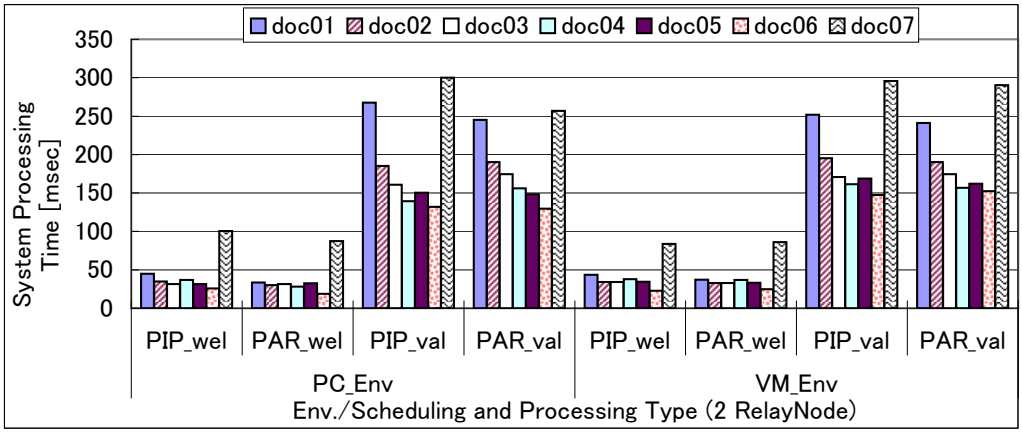

図 5.4.11: System Processing Time (Synthetic Docs; 2RN).

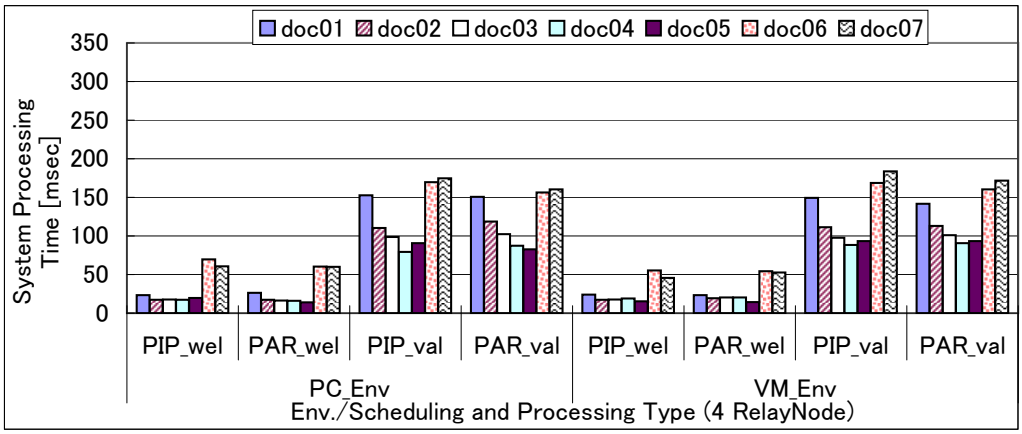

図 5.4.12: System Processing Time (Synthetic Docs; 4RN).

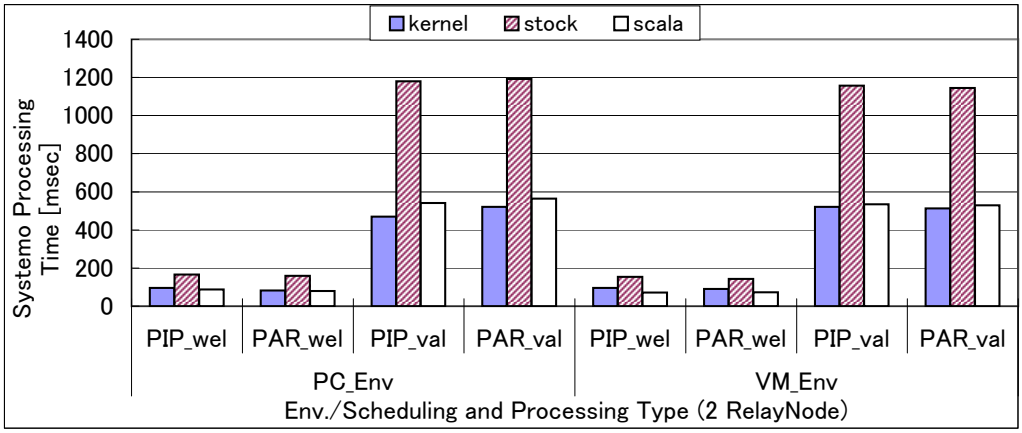

図 5.4.13: System Processing Time (Realistic Docs; 2RN).

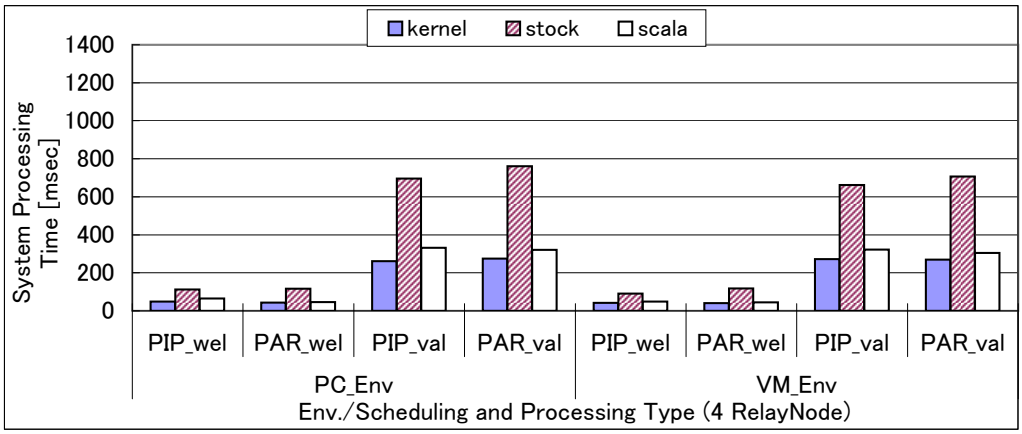

図 5.4.14: System Processing Time (Realistic Docs; 4RN).

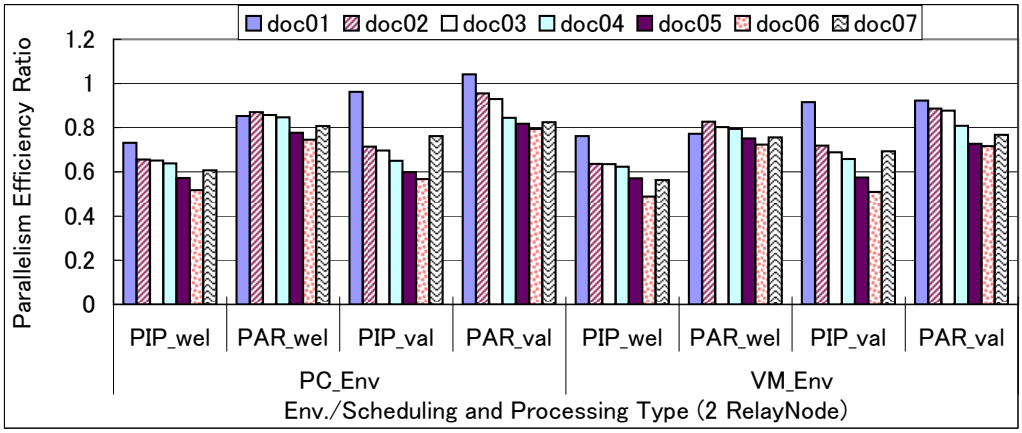

図 5.4.15: Parallelism Efficiency Ratio(Synthetic Docs; 2RN).

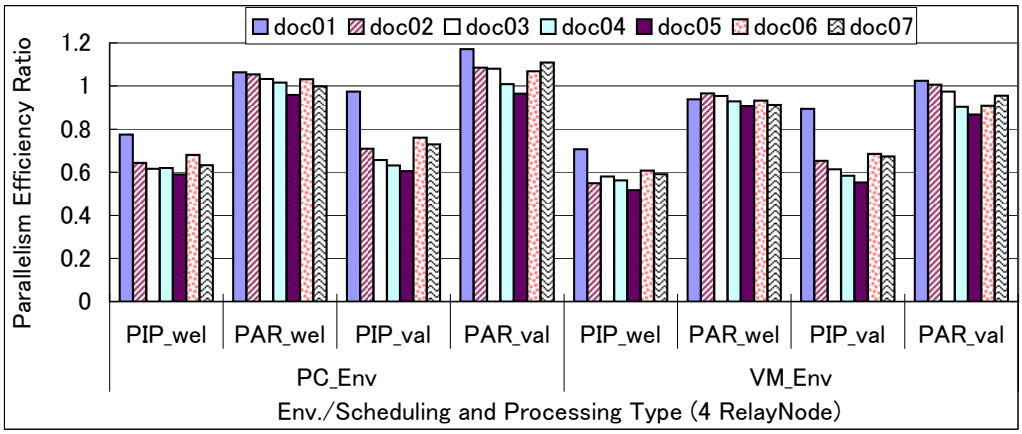

図 5.4.16: Parallelism Efficiency Ratio(Synthetic Docs; 4RN).

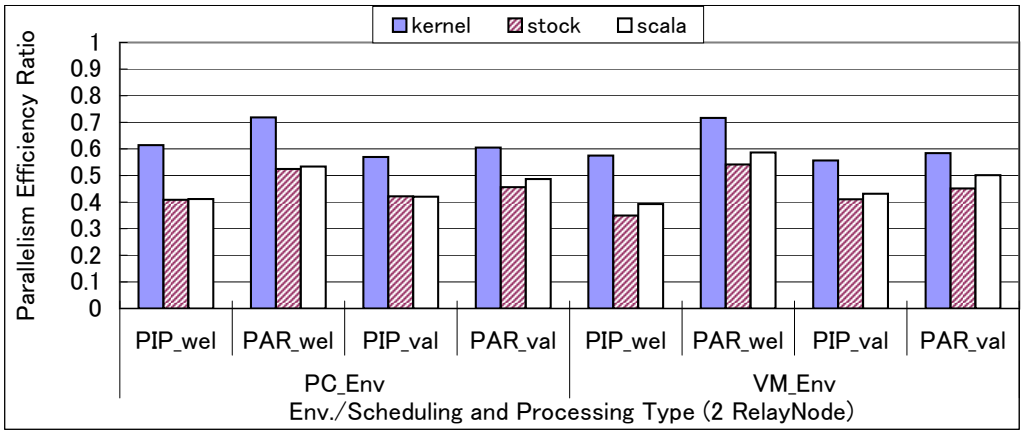

図 5.4.17: Parallelism Efficiency Ratio(Realistic Docs; 2RN).

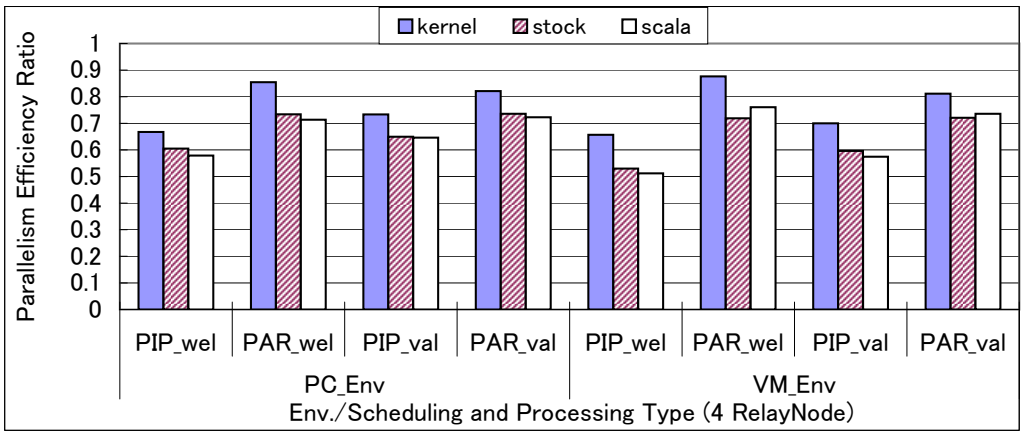

図 5.4.18: Parallelism Efficiency Ratio(Realistic Docs; 4RN).

5.5 まとめ

本章では，

- 実環境及び仮想環境を用いた場合の特性の比較
- 合成 XML 文書及び実データに基づく XML 文書を用いた場合の特性の比較

の二つの評価を行った．その結果，前章と同様に，データ分割並列処理は基本的にパイプライン並列処理の方が有利であるという点など，多くの性質が実環境と仮想環境で同一であることがわかった．また大きな XML 文書（realistic doc）と小さな XML 文書（synthetic doc）の比較においては，パイプライン並列処理においても大きな XML 文書（realistic doc）の場合は，処理ノードを増やした場合に良い影響が出ること，加えて，小さな XML 文書（合成 XML 文書：synthetic doc）の方が，大きな XML 文書（実データに基づく XML 文書：realistic doc）よりも効率的に処理できていることが分かった．

第6章　処理ノードにおける バッファ性能の改善

6.1　はじめに

計算機の高性能化・低価格化や安価で高品質のネットワーク環境の普及により，今日ではネットワークを介した計算機の利用は当たり前のものとなっており社会基盤の必要不可欠な構成要素の一つとなっている．このような状況下で，我々は，通信を行う計算機の間に複数の経路が存在し，その通信経路上にデータ転送を行う複数の計算機が配置される状況を前提とし，これら中継計算機においてデータの転送を行うとともにそのデータに対して何らかの有益な処理を行うことで，ネットワーク自体に付加的な機能を持たせるようなネットワークアプリケーションの開発と評価を行っている．また，XML 文書は，異なる環境・異なるアプリケーション間であっても影響を受けない汎用的なデータ構造であり，今日ではネットワーク通信や様々なデータの保存形式や Web サービスにおけるインタフェースとして幅広く利用されている．このため，XML 文書に対する処理を行う機会が増えており，XML 文書を効率的に処理するための研究が盛んに行われている [52, 53, 54, 55]．この内，PASS（Prefix Automata SyStem）-Node[1] は，ネットワークアプリケーションでの分散 XML データ処理に着目し，整形式判定（well-formedness grammar checking）・妥当性検証（validation grammar checking）・フィルタリングの処理機能をデータ転送途中の中継ノードへオフロードすることで，サーバでの処理負荷の軽減などを図っている．

評価アプリケーションは PASS-Node を参考に実装されており，XML 文書をネットワークを通じて送信元の計算機から送信先の計算機へ転送する際，送信先の計算機で受け取った XML 文書に対して処理を行う状況を想定している．この時，中継計算機で XML 文書に対する前処理を行う機能を持たせることで，ネットワーク自体にデータ転送以外の高度な共通機能を付加し，従来送信先の計算機で行ってきた処理をネットワーク

にオフロードすることが可能となる．そして，この種の様々なネットワークアプリケーションで必要となる共通機能をネットワークが提供することにより，複数のサービスの連携やネットワーク上のデータの健全性を保つことが容易となる．現在の評価アプリケーションではXML文書の整形式検査と文法検査の2種類のXML文書のチェック機能を実装しており，中間ノードでの前処理を行いながら，終着点となるノードではXML文書全体にわたって検査済みの最終的な結果を得ることができ，処理負荷を軽減することができる．また，これらの処理は中継ノードの配置に応じてパイプライン並列またはデータ分割並列で行う．

我々は，XML文書に対する分散処理を例として取り上げ，ストリーミングデータへの並列分散処理の特徴・課題・利点などを明らかにすることを目指している．このため今までストリーミング通信でやり取りされるXML文書に対して，通信中継ノードで処理を行う評価アプリケーションを実装し，XMLへのストリーミングデータ処理に関する様々な評価を行ってきた [48, 47, 49, 50, 51]．

本章ではこれらから得た知見と結果の分析を元に中継ノード及び送信先の計算機に配置されるノードにて使用される，転送データを一時的に保管するためのバッファについて分析を進めた．その結果，本システムに合わせた改善の余地が見られたため改良を試みた結果について述べる．

本章で述べる改良提案は2段階に分けられる．1段階目の改良では，本アプリケーションで使用するバッファに求められる特徴を分析し，LinkedListをベースにわずかな改良を施すことで，これまで最も性能の良かったArrayListによるバッファよりも高性能なバッファを実装することができた．さらに2段階目の改良では，パフォーマンスを減じること無くマルチスレッドからの同時アクセスに対応させた．この改良の結果として，システムの性能が以前よりも改善されるとともに，これまでノード内では，単体の処理スレッドで行っていたXML文書に対する文法チェック処理を，複数のCPUコアを用いて単一ノード内で同時に行う手法により，性能を損なわないまま実現できることとなった．

また本章にて述べる内容は [56] に対応するものである．

6.2 バッファ性能改善のための提案手法

各RelayNode，EndNode及びMergeNodeは，ネットワーク間のデータの受信・処理・送信のために使用する，データ格納用のバッファ（第

2 章，図 2.3.2 中の Shared Buffer）を有している．[47] ではこのバッファとして，Java 言語の標準クラスライブラリに含まれる `ArrayList（AL）` と `LinkedList(LL)` を用いた比較を行い，XML 文書への処理を行うノード (RelayNode と EndNode) で使用するバッファとしては AL の方が適し，MergeNode については LL が適するという結果を得た．一般に AL は要素の検索・取得は高速だが追加・削除は遅く，LL は要素の追加・削除は高速だが検索・取得は遅い．事実，分析を進めた結果，AL を用いた場合，バッファアクセスに要する時間の多くはバッファからの要素の削除処理であることが判明している．AL の要素の削除処理の時間を減らすことができればバッファアクセスに要する時間を低減することができるが，これは AL を用いる限り困難である．

6.2.1 共有バッファの機能要件

評価アプリケーションの処理ノードのバッファでは，下記の四つの要件を満たすことが求められる．

- 要件 1:先頭から順に要素を取り出す
- 要件 2:末尾に要素を追加する
- 要件 3:現在処理中の要素を管理・取得が可能である
- 要件 4:複数スレッドで効率よく並行アクセス可能である

要件 1 と 2 は FIFO で要素を取り扱うことであり，評価アプリケーションにおいて各ノードにおける ReceiveThread と SendThread によるバッファへの要素の追加/削除処理のためのものである．要件 3 は処理ノードにおいて，TagCheckThread による XML 文書へのストリーミング処理を行うためのインデックス管理であり，XML 文書の先頭から順に始まり末尾まで一つずつ順にアクセスされていく．この時，第 2 章でも述べた通り，このインデックスとスタックとを組み合わせることで，XML 文書のチェック処理を行う．要件 4 は評価アプリケーションにおいては複数のスレッドが同時にバッファを利用するが，その並行性と排他制御に関することである．例えば，処理スレッドがある要素の処理を継続中である場合は，SendThread がその要素をバッファから削除して，後続ノードに送らないようにする必要がある．この時，第 2 章の図 2.3.2 に示す，RelayNode を例として発生するロックは，

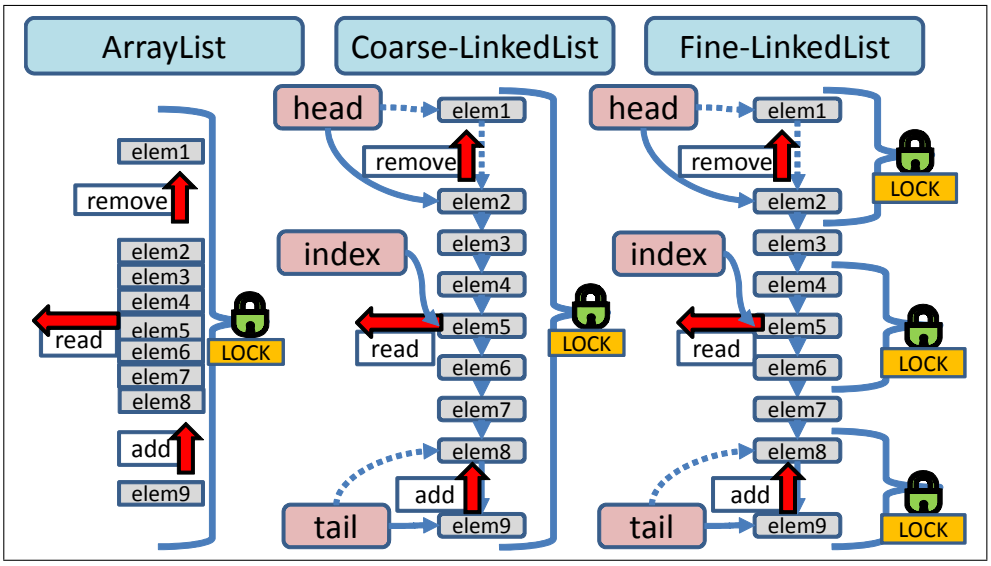

図 6.2.1: Node Buffer Component(AL; C-LL; F-LL).

- ReceiveThread による要素の追加時の書き込みロック
- SendThread による要素の削除時の書き込みロック
- TagCheckThread による要素の参照時の読み込みロック

の 3 種類に分けることができる．

前述の AL と LL では，AL は要件 2（ただし，バッファサイズの拡張が行われない場合に限定）と 3 に優れるが要件 1 で劣り，LL は要件 1 と 2 に優れるが要件 3 において劣っているといえる．さらに両者とも，要素の取得時には読み込みロック，追加時/削除時には書き込みロックをかけるなどのロック制御を行うが，バッファ全体へのアクセスをロックするグローバルロックとなってしまうため効率よく要件 4 を満たすことができない．またこの時，両者とも 3 の現在処理中の要素の管理には各要素のインデックス番号を利用している．そのため LL においては要素の取得に時間がかかるとともに，両者とも要素の削除時にインデックス番号をデクリメントする必要があり，このための同期も行う．一方，同じく Java 言語の標準クラスライブラリに含まれる ConcurrentLinkedQueue では [57] のアルゴリズムを用いて効率よく要件 1・2・4 を満たすことができる．しかし，キュー構造であるのため任意のインデックスの要素を取得することことができず，要件 3 を満たすことができないという問題がある．

6.2.2 バッファの改良

以上までを元に，LL を元にした下記の二段階のバッファの改良を行うことでバッファアクセスのボトルネックの解消を試みた．

`Coarse-LinkedList`（`C-LL`）は，LL をベースに，PASS-Node 向けの機能に特化したチューニングを施しつつ新規に実装した単方向連結リストである．現在処理中の要素を示す参照インデックス（ポインタ）を適切に管理・使用する（図 6.2.1 の index）という改良を施すことで，XML 文書に対する処理を行う際に必要な，要素の取得に費やす時間を削減している．このインデックスの管理のために必要な記憶領域の大きさは 1 オブジェクト分のポインタだけで済むため，リストの長さに比して大きくなることはない．また AL と C-LL に対して複数のスレッドが同時にアクセスする際には，先の AL や LL の場合と同様にバッファ全体をロックして，スレッド間の衝突を回避している．加えて LL ではバッファ内の要素数を管理するためのグローバル変数が存在するが，本評価アプリケーションにおいては，要素数を取得する必要が無いためこのグローバル変数を除いている．これにより要素の追加・削除時に，グローバル変数を操作するための同期処理も削減している．

`Fine-LinkedList`（`F-LL`）は，C-LL をベースに要素単位の細粒度ロックを実装し，

- 要素の追加時には末尾要素とその直前の要素への書き込みロック
- 要素の削除時には先頭要素とその直後の要素への書き込みロック
- 要素の取得時にはその直前・直後の要素への読み込みロック

のみを適切な粒度で行うことで，要素へのアクセスが競合する場合を除いてバッファアクセスを同時並列に進められるように改良した．ただし，C-LL と比べてロック操作のための時間が必要となる．

6.3 実験環境

本章にて用いる実験環境は

- X4640_Env（仮想環境）

である．この環境にて，

- XML 文書の整形式判定（Well-formedness grammar checking）
- XML 文書の妥当性検証（Validation grammar checking）

の二種類の処理を行う．また，この時用いる XML 文書は

- 合成 XML 文書（synthetic doc）
- 実データに基づく XML 文書（realistic doc）

である．また，タスクのノード割当は前章の 5.2 節と同様である．この環境において，各処理ノードで用いるバッファを変更しつつ比較・評価を行う．

6.4 評価:バッファ性能改善のための提案手法

この節では，実験結果とその評価について述べる．

6.4.1 ArrayList vs. Coarse-LinkedList

バッファとして AL 及び C-LL を用いて，synthetic doc に対して XML 文書の整形式判定（Well-formedness grammar checking）を行った時の Job Execution Time を図 6.4.2 に，System Buffer Access Time を図 6.4.3 に示す．さらにこの中から，doc07 の Node Buffer Access Time を図 6.4.4 に示す．

ジョブの実行時間（図 6.4.2）は AL よりも C-LL の方が短く，バッファアクセスに要する時間（図 6.4.3）についても C-LL の方が短い．Node Buffer Access Time（図 6.4.4）においても，AL よりも C-LL の方が短く良好な結果を示している．よって C-LL は，LL に二つのポインタを付け加えた簡単な改良を行ったものであるが，AL のように要素の削除処理に長時間を要さず，LL のように並列分散処理を含めて，全体的な性能の良さを併せ持つ．

これらの傾向は他の全ての結果でも同様であった．今回実装した C-LL は簡単な改良ではあるが，PASS-Node における各ノードで使用するバッ

ファとして AL より効率的であるといえる．つまり [47] では，LL と AL の比較を行い，AL の方が有効であるという結果を得ていたが，C-LL は更に効率的である．

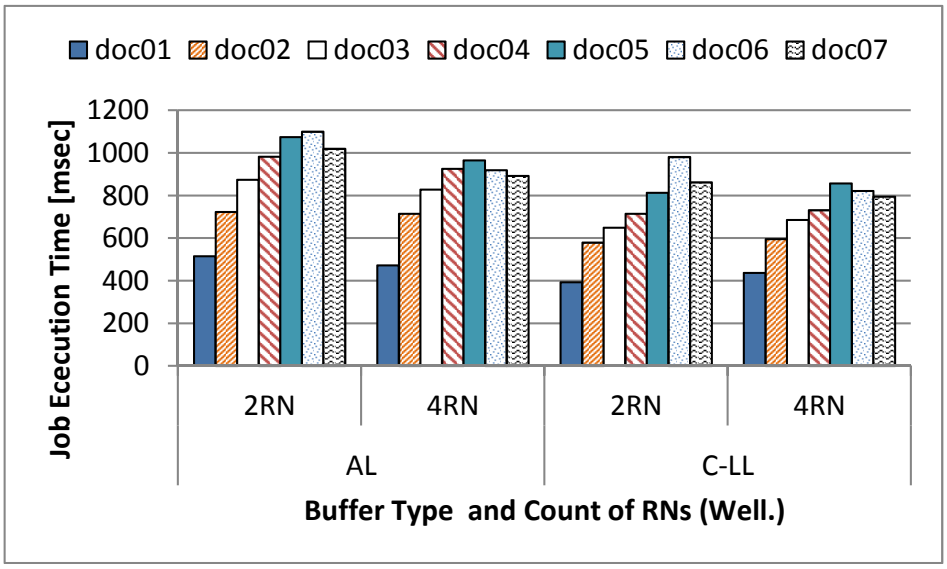

図 6.4.2: Job Execution Time（AL & C-LL; syn.; Well.）

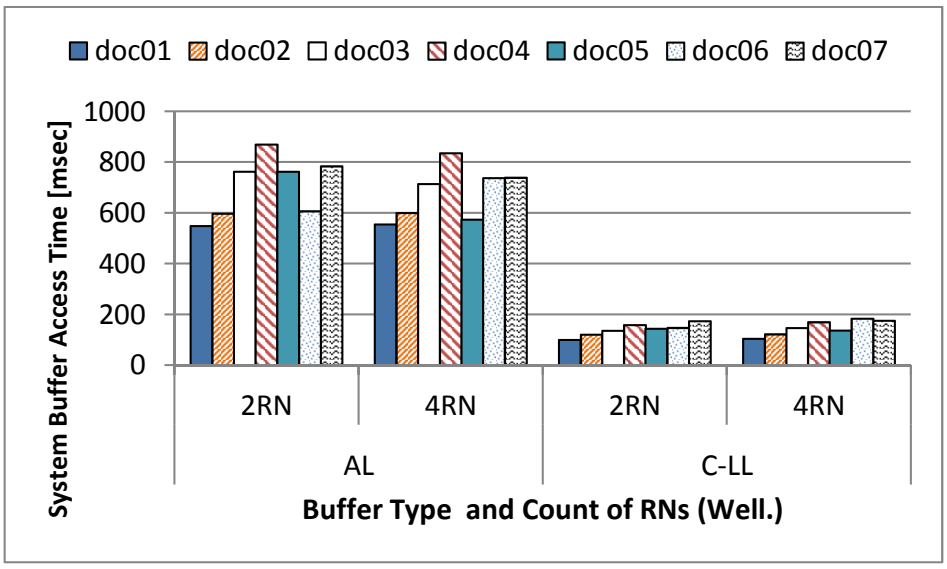

図 6.4.3: System Buffer Access Time （AL & C-LL; syn.; Well.）

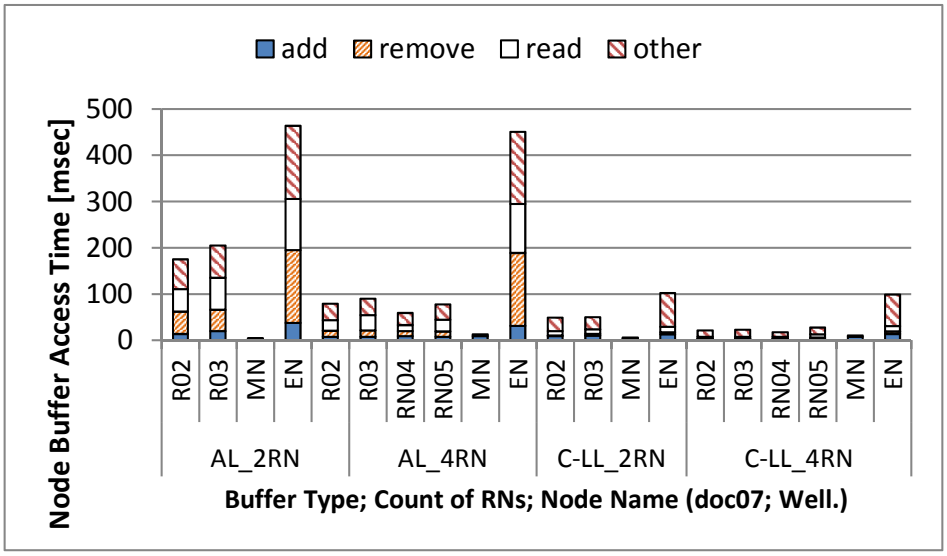

図 6.4.4: Node Buffer Access Time（AL & C-LL; doc07; Well.）

6.4.2 Coarse-LinkedList vs. Fine-LinkedList

バッファとして，C-LL または F-LL を用い，realistic doc に対して，妥当性の検証（Validation grammar checking）を施した時の Job Execution Time を図 6.4.5 に，System Active Time を図 6.4.6 に，System Buffer Access Time を図 6.4.7 に示す．さらにこの中から，kernel_doc の Node Buffer Access Time を図 6.4.8 に示し，Node Thread Working Time を図 6.4.9 に示す．

C-LL または F-LL をバッファとして用いた場合の，ジョブの実行時間（図 6.4.5）と System Active Time（図 6.4.6）には大きな差は見られない．しかし，System Buffer Access Time（図 6.4.7）に関しては，C-LL よりも F-LL の方が大きな値を示す．さらに Node Buffer Access Time（図 6.4.8）からも，F-LL は C-LL に比べて，バッファアクセスに多くの時間を要することが分かる．これは，F-LL はバッファにアクセスする際により多くのロックを管理する必要があるためである．一方，Node Thread Working Time（図 6.4.9）については C-LL と F-LL であまり差が無いため，バッファアクセス以外の評価指標の示す値はほとんど変化がないことが分かる．例えば二つの RelayNode を用いて kernel_doc を処理する場合，図 6.4.5，図 6.4.8，図 6.4.9，によると，Node Thread Working Time の合計は C-LL で 5,957msec，F-LL で 5,976msec である．Node Buffer Access Time の合

計 (=図 6.4.7 と同じ値になる) は，C-LL で 360msec，F-LL で 607msec であるが，それでも Job Execution Time は，C-LL で 2,276msec，F-LL で 2,297msec となり，ほとんど差が無い．

以上の結果より，F-LL はロックコストにより C-LL よりもバッファアクセスに多くの時間を要するが，処理に要する時間は C-LL と同等であることがわかる．そのため，F-LL ではロックのコストが必要なものの，細粒度ロックによる並列動作性により，各スレッドが同時に稼働することで，同等の性能を示していることが分かる．

また，今回の処理ノードでのスレッド数は受信・処理・送信の三つである．スレッド数が少ないときは，C-LL でも十分なパフォーマンスが得られるが，単一ノード内で一つの XML 文書に対して，複数スレッドを用いて部分毎に並列処理を行う場合は，F-LL の利用による効率化が期待できる．ただし，スレッドが増えた場合は F-LL においてもバッファアクセスの競合が起こる機会が増加するため，処理の割り当て方法について検討する必要があるといえる．

なお，本例証アプリケーションにおいて，実計算機または仮想計算機を用いた場合に，大きな違いがないことが確かめられているため [51]，本実験結果は実環境においても仮想環境においても有効である．

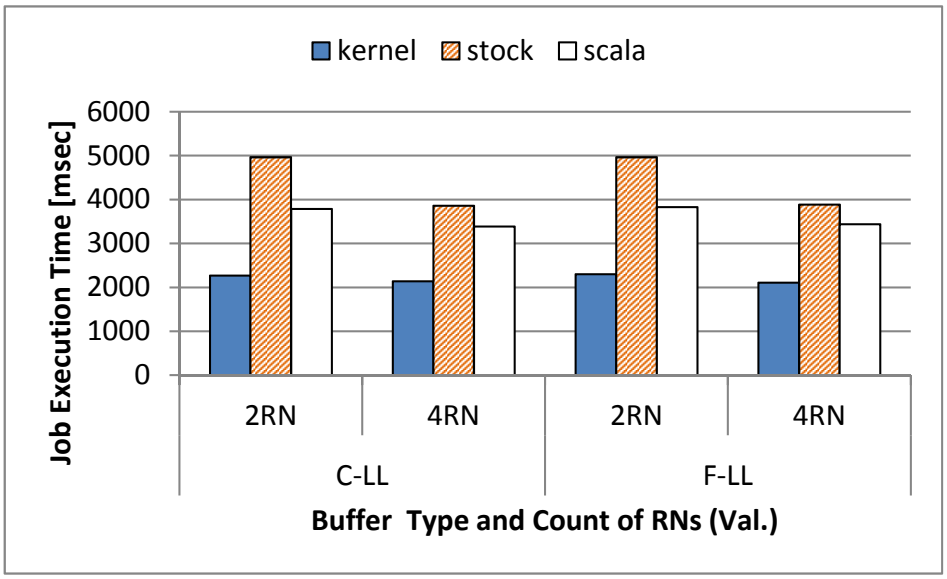

図 6.4.5: Job Execution Time（C-LL & F-LL; real.; Val.）

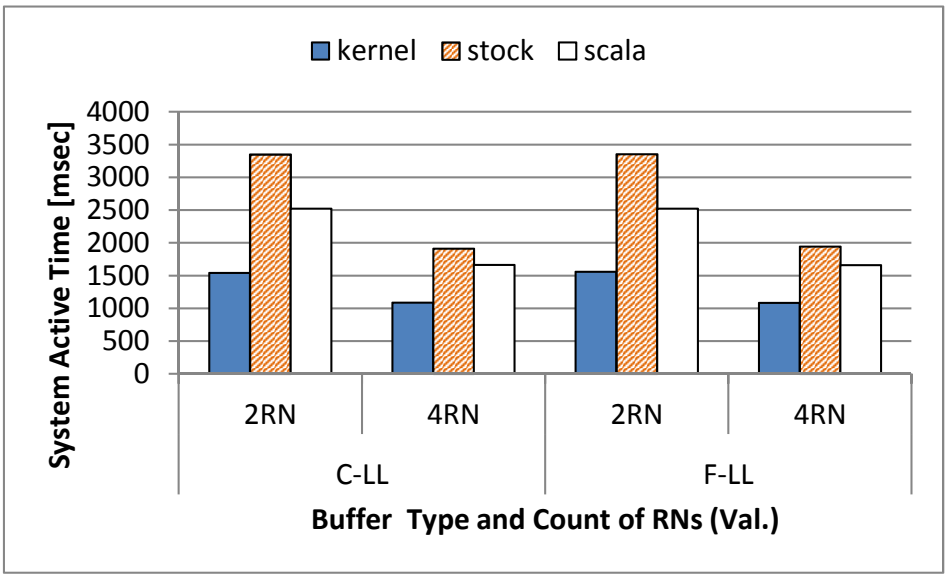

図 6.4.6: System Active Time（C-LL & F-LL; real.; Val.）

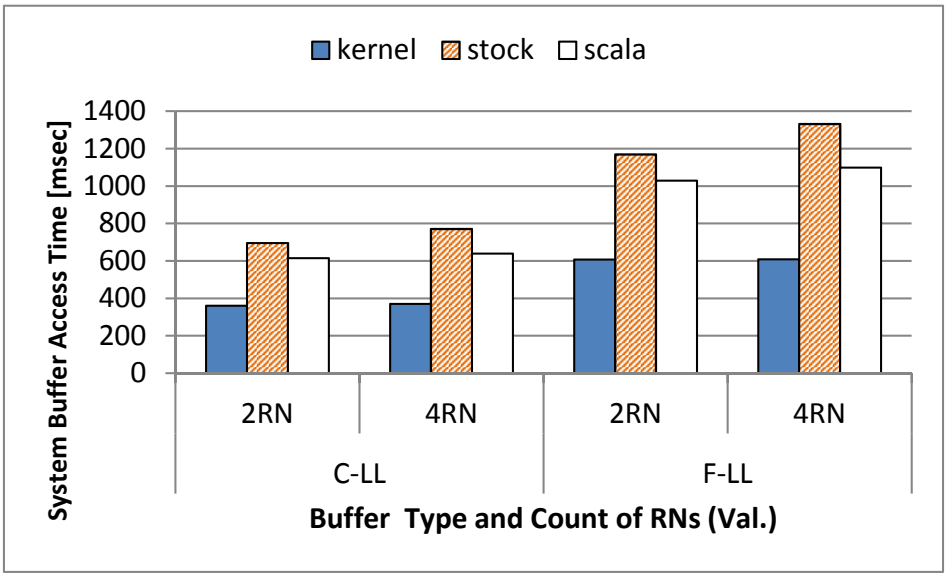

図 6.4.7: System Buffer Access Time (C-LL & F-LL; real.; Val.)

6.5 まとめ

本章では，ストリーミングデータに対する並列分散処理手法に関連して，XML 文書を取り挙げ，ネットワーク上でやり取りされるストリーミ

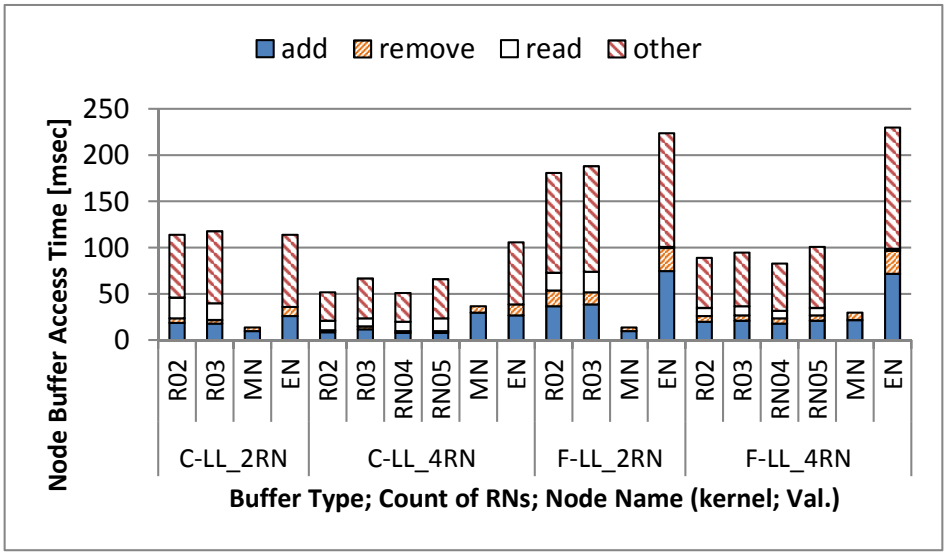

図 6.4.8: Node Buffer Access Time（C-LL & F-LL; kernel; Val.）

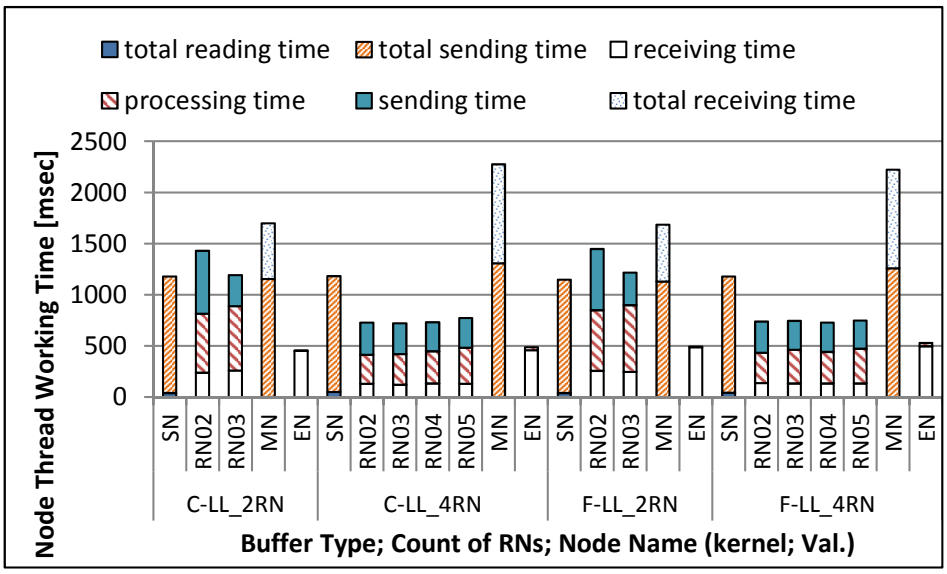

図 6.4.9: Node Thread Working Time（C-LL & F-LL; kernel; Val.）

ングデータへの並列分散処理の効率化のため，ノードにてデータを保管するバッファの改良を試みた．

評価アプリケーションにおける，ストリーミングデータ処理の性質を分析し，これまで使用してきたバッファを改良し，簡単な実装ながらより効率的なバッファを実装した．さらに，これまでは同時には単一の処

理スレッドにしか対応していなかったバッファをさらに改良し，複数のスレッドからの同時アクセスに，パフォーマンスを損ねることなく対応させた．これにより今後，一つのノード内で複数スレッドを用いた単一 XML 文書への並列処理を行うことが可能となった．また，この成果は評価アプリケーションにおける XML 文書に対する文法チェックと同様の性質を持つ別の処理を行う場合にも応用が期待できる．以上により，我々は，ノード内共有バッファの改良により経路上のノードで行うストリーミングデータ処理の効率化を実現した．

第7章　関連研究

本章では，本研究の関連研究について述べる．

本研究で取り組むネットワーク上での処理機能の提供の類似研究として，現在のネットワークにて提供されるような処理機能を紹介する．動画像の変換を行うトランスコーディング [3, 4] は，動画像の解像度やコーデックなどをユーザに配信する途中で変換する技術である．コンテンツ配信で用いられるキャッシュ [4, 5, 6] は，ユーザがサーバよりコンテンツをダウンロードする状況において，事前にサーバに格納したコンテンツのコピーをネットワーク中のキャッシュサーバに配置しておき，ユーザは一番近いキャッシュサーバからコンテンツを得るというものである．このような例は，動画像の変換や蓄積・配信機能をネットワークが有する例といえる．他にも，センサネットワークを対象としたセンシングデータの集約 [7, 8, 9] や，転送データの圧縮と伸長 [10] 等の例が既存研究や技術として存在する．

昨今，様々な局面で XML 文書に対する処理を行う機会が増えており，XML 文書を効率的に処理することは重要な課題となっており，このために様々な研究が行われている．

[52] は，シングルスレッドを対象とし，XML 文書を一旦，より高速に処理できる形式へ変換することで，XML 文書の構文解析を高速化する手法を提案している．また，[55] では，データサイズの大きな XML 文書の処理のための並列分散処理ライブラリが提案されている．この研究では決定性有限オートマトン（Deterministic Finite Automata:DFA）で表される処理を，DFA+非決定性有限オートマトン（Non-Deterministic Finite Automata:NFA）に分解し，これらを用いて投機的な同時並列実行を行っている．

さらに，[53] では，XML 文書の構文解析を同時並行に行うために，マルチスレッドで動作するパーサーのパフォーマンスとそのオーバヘッドについての議論を行っている．[54] では，XML 文書を複数のスレッドを用いて，同時並列に処理する際に，work stealing と呼ばれる方法でロー

ドバランシングを図っている．[58] も，マルチコア環境において XML 文書を同時並列に処理する研究であり，処理開始前に XML 文書を事前解析し，XML 文書の構造の情報を生成し，処理の分割やマージに利用している．これらは単一ノードでの処理ではあるが，今後，提案手法における処理性能・効率の向上に応用することが期待できる．

また本研究では，XML 文書に対する整形式検査と文法検査という基本的な機能を実装し評価を行った．この XML 文書への処理はその他にも様々なものが存在している．[59] はパソコン向けに構成された HTML コンテンツを，プロキシを通じて一旦 XML 形式に変換し，eXtensible Stylesheet Language（XSL）[60] と XML 文書の変換処理（XSLT）[61] を利用して，モバイル機器向けに情報を変換するものである．[62] はセキュリティに関連して XML 文書に対する暗号化・復号化を取り扱った研究である．[62] では，XSL と XSLT を応用して，本体となる XML 文書に暗号化・復号化用の XSL 文書を組み合わせて，XML 技術を軸にセキュアなデータ交換を提案している．評価アプリケーションでは XML 文書に対する新たな処理を追加できるように，十分な拡張性を持った設計を行っており，これらの成果を適用し，新たな処理機能を付加することも可能である．

次に，ネットワークに対してデータ転送以外の機能を持たせる取組みについて述べる．

Active Network[11] は，ネットワークの経路上に存在するルータにて処理を行うものである．これらは基本的に専用のネットワークアプライアンスを用意し，i) 事前に処理内容を記述したプログラムを配置する，ii) パケット自体にプログラムを内蔵する，という 2 種類の方法でルータにて行う処理機能を定義する．前者の，事前に処理プログラムをルータに配置する方式では，より高機能な処理を定義できる．後者のパケット自体に処理を記述する場合では，処理内容はペイロードに内蔵できる比較的単純な処理内容に限られるが，その分高速な処理が可能であり，エッジノード付近など高速処理が必要な部位での処理に向くという特徴がある．本研究での処理内容の記述は，前者の事前にプログラムを配置する形態に近いが，配置するのは，個々のパケットに対して処理を行うプログラムではなく，ストリーミングデータ処理を行うタスクとなる．また本研究では，処理速度では劣るものの，よりアプリケーションに近いレイヤでの処理を行うことで，アプリケーションを直接サポートすることを重視している．また VNode[12] では，このような任意の処理の割当を，仮想化技術を用いることで実現しており，より柔軟な資源割り当てが可能

である．このような研究はネットワークに任意の機能を付加するための，プラットフォームとして位置づけることができる．

同じく，ネットワーク上の計算資源を利用するという点ではグリッドコンピューティング [18] やクラウドコンピューティング [19] などの技術がある．これらは，コアネットワークの外の何処かに配置された計算資源を用いて処理を行うものであり，これらは大規模な処理に対して，スケールすることで適応し，この種の処理では，本研究よりも優位に立つと思われる．また，中継ノードを利用する利用という点では，グリッドコンピューティングやクラウドコンピューティングでは，自身の管理外である中継ネットワークの情報を知ることは難しく，それらの情報を処理に活かすことはできない．

さらにストリームプロセッシング [20] は，本研究と同様，様々なデータ形式を対象に，ストリーミングデータ処理を行い，種々の Web サービスへの適用を行っている研究である．ストリームプロセッシングでは，クラウド環境においてデータをストレージに格納することを回避し，メモリ上に配置したデータに逐次的に処理を施すことで，処理速度を上げることを狙っている．本研究では，このようなストリーミングデータ処理を，中継ノードで行うことで，ネットワークに機能を付加した高機能ネットワークを実現することを目指す．またこれらをタスクスケジューリングの課題として捉えている．

[63] では，本研究よりも下位のレイヤを対象とし，中継ノードにパケットの圧縮・伸張機能を実装し，パケットが中継ノード内のバッファで待機する待機時間と，空きリソースを用いながら処理を行い，ネットワーク品質の改善を行っている．

また [64] は，OpenTag[65] というパケットごとに ID を振りフロー単位で，経路制御などを行う技術を元にクラウド環境上のプラットフォームを利用する際，アイソレーションを提供している．これらと評価アプリケーションを連携させることで，我々の主眼であるタスクスケジューリングと切り分けて，プログラマブルなネットワーク上でのフロー制御等を盛り込むことが期待できる．

このように，ネットワーク接続された計算資源を用いた処理とネットワークの制御技術について様々な研究が行われている．本研究では，高機能ネットワークの実現を目指し，特に中継ノードでの処理を対象としたストリーミング並列処理に着目し，汎用性と他のデータ形式への応用が期待可能な XML 文書を題材とした．様々な技術を状況に応じて使い分

け，適材適所で利用することが重要だといえる．このようなことを実現するスケジューリング手法を通じて，将来のインターネット環境における資源利用の効率化を図る事が可能である．

さらに我々は過去に，本アプリケーションを用いたXML文書に対するストリーミングデータ処理に関するいくつかの評価を行っている．[49, 50]では，パイプライン並列とデータ分割並列，処理を行うXML文書の特徴，処理ノード数の増減，処理内容が異なる場合の差の評価を行い，概してデータ分割並列がパイプライン並列よりも優れていること，XML文書の構造によって効率的な処理の分割の仕方が異なること等を明らかにした．[47]では，各ノードで使用するバッファの性能について評価した．さらに[51]では，実環境と仮想環境の性質の違いを評価し，両者で明確な性質の違いがなく仮想環境で得た知見を実環境に応用できることを確かめた．

第8章　単体の処理ノードのみ/複数の処理ノードでのストリーミング並列処理の比較

本研究にて採り上げた，中継ノードで処理を行う並列ストリーミング処理においては，データの分割と割当処理，データ転送，データのバッファへの格納と取り出しなどに伴うオーバーヘッドが発生する．特にデータ転送やバッファアクセスが関わる部分のオーバーヘッドは，すべての処理ノードで生じるため，中間処理ノードの増加とともに積み重なっていく．このオーバヘッドにより，複数台の処理ノードを用いた並列ストリーミング処理を行う場合は，送信元から宛先へデータを送り，宛先の計算機単体でのみでストリーミング処理を行う場合に比べて非効率となり，より多くの時間が必要であった．本章ではこれまで本論文で述べた成果を元に，

- 最も効率的だと思われる構成で，複数台の中継ノードを用いた並列ストリーミング処理を行う場合，および，
- 単体の処理ノードのみを用いたストリーミング処理を行う場合

を比較し，中継ノードを用いた並列ストリーミング処理手法における，処理性能と処理時間に関する議論と処理のオフロードの確認を行う．この比較実験を通して，データ量が多く，その処理が重い場合は，複数台の処理ノードを用いた並列ストリーミング処理を行う場合の方が効率的となるケースがあることがわかった．この評価結果により，高機能ネットワークの実現において，処理性能における利点を考慮した，より効率的なタスクスケジューリング手法の検討への見通しを得ることができた．

8.1　実験条件及び結果と評価

まず，実験環境とその際の条件について下記に述べる．

- 実験環境は，仮想環境である X4640_Env（第 3 章 3.1 節）
- 実験に用いる XML 文書は，合成 XML 文書（synthetic doc）と実データに基づく XML 文書（realistic doc）の両方
- XML 文書への処理は，XML 文書への整形式判定と妥当性の検証の両方
- 先の第 6 章で述べた，改良型のバッファである F-LL を用いる

さらに，この実験においてノードの配置パターンは，

noRN StartNode と EndNode を RelayNode を介さず直接つなぎ，全ての処理を EndNode に割り当てる（図 8.1.1）

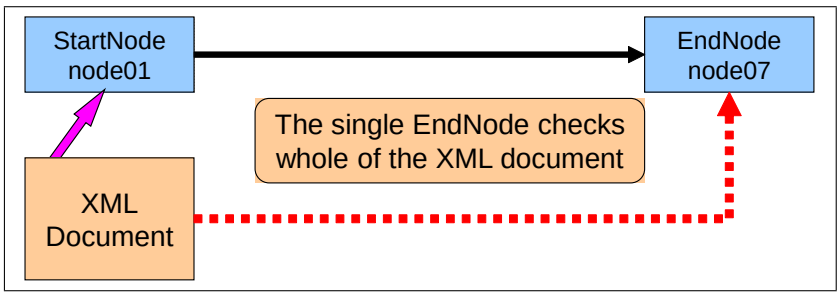

図 8.1.1: Single Processing Instance.

2RN StartNode と EndNode の間に二つの RelayNode を配置する（第 5 章，図 5.3.1 と図 5.3.2 に示したノード構成と同一）

4RN StartNode と EndNode の間に四つの RelayNode を配置する（第 5 章にて述べたノード構成と同一）

の 3 種類である．この 3 種類のノード配置パターンについて，上述の実験環境及び実験条件に基づいて，それぞれのジョブの実行時間（Job Execution Time）の比較を行う．また，2RN と 4RN ではこれまでの述べた実験と同様に，パイプライン並列処理（PIP）またはデータ分割並列処理（PAR）の処理形態をとる．これらの実験結果を図 8.1.2 と図 8.1.3 に示す．

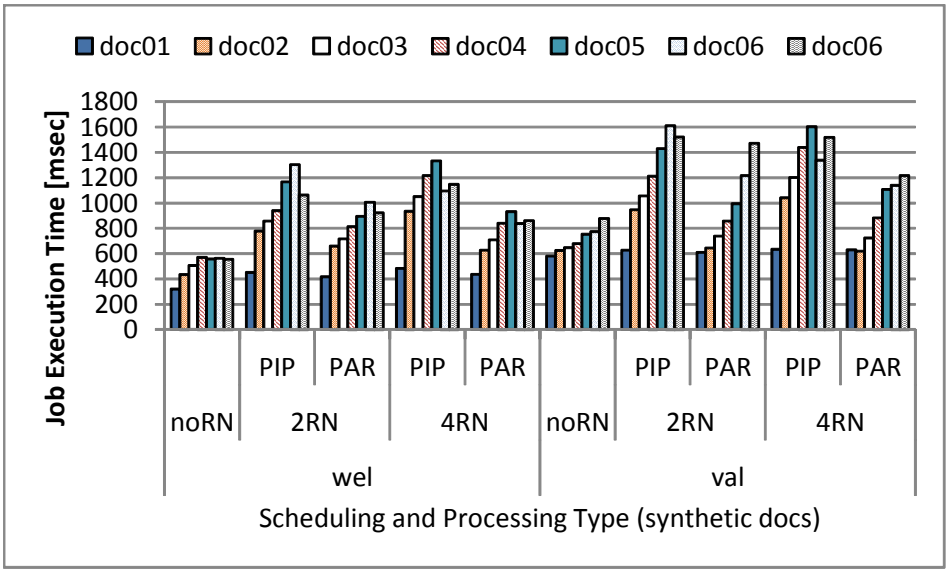

図 8.1.2: Job Execution Time (Synthetic Docs: noRN; 2RN; 4RN).

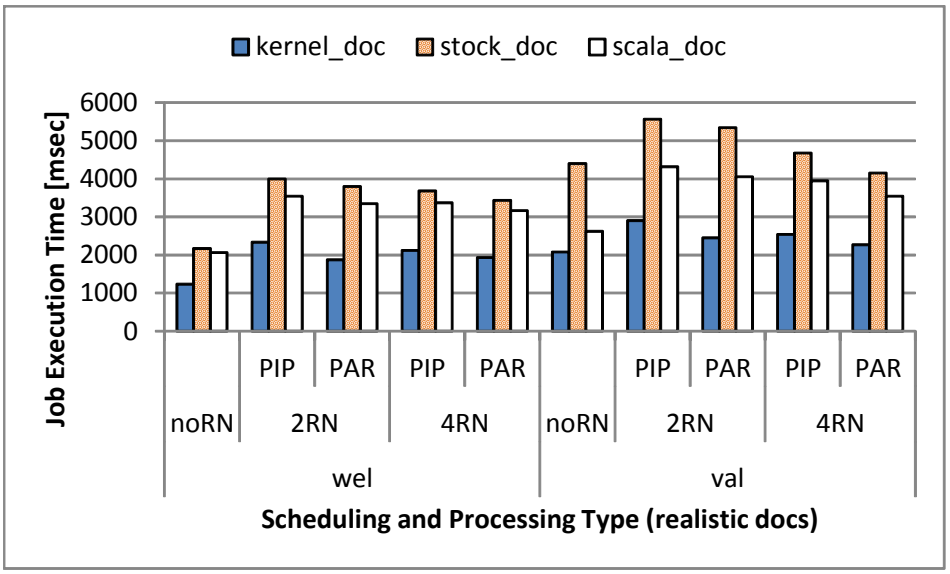

図 8.1.3: Job Execution Time (Realistic Docs: noRN; 2RN; 4RN).

図 8.1.2 と図 8.1.3 によると，多くの例で，単一ノードでのみ処理を行う noRN の実験結果が最速であることがわかる．これは，中継ノードで処理を行う場合には，データ転送とデータ分割等により，オーバーヘッドが余分に発生することが原因である．しかし，図 8.1.3 の stock_doc の処理で，noRN よりも 4RN でデータ分割並列処理を行った方が短い処理時間で処理が終わっているケースがある（図中右側，それぞれ stock_doc について，val-noRN の場合で 4,393msec，val-4RN-PAR の場合で 4,151msec）．これは本実験条件の中で，最も処理量（タグの数）が多く，処理が重い（妥当性の検証）場合である．このため，本研究が対象とする中継ノードを利用した並列ストリーミング処理においては，処理量が多く，さらにその処理が重い場合，前述のオーバーヘッドを加味してなお，単体の処理ノードで集中して処理を行う場合よりも，複数台の中間処理ノードを用いて処理を行った場合のほうが，より高速に処理できることが見て取れる．

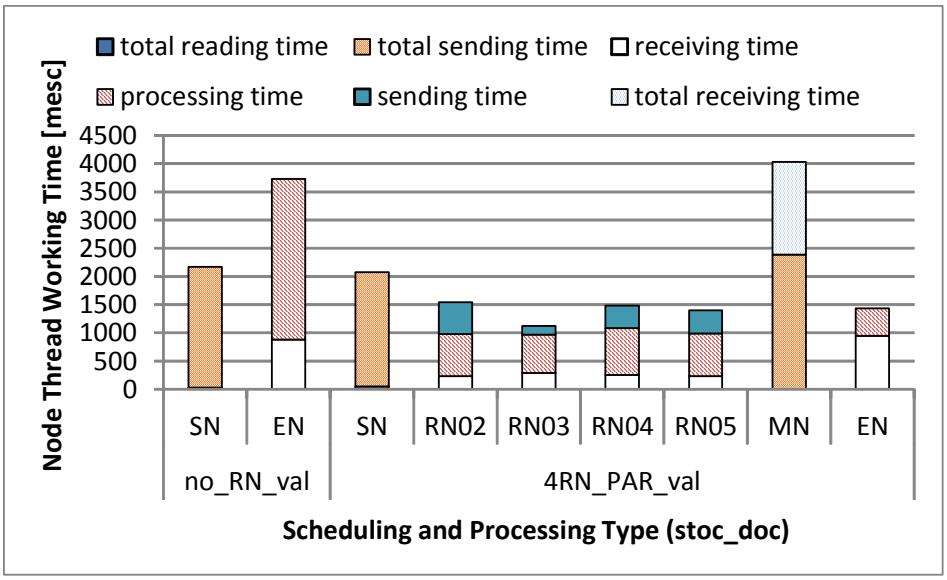

図 8.1.4: Node Thread Working Time (Stock doc: val.: noRN ;4RN_PAR).

さらに図 8.1.4 は，この時の詳細な Node Thread Working Time である．図 8.1.4 は，それぞれのノードで，

- XML 文書の読み込み (total reading time)，

- 分割された XML 文書と処理結果の送信（sending time，total sending time），
- 分割された XML 文書と処理結果の受信（receiving time，total receiving time），
- 文法チェック処理（processing time）

に要した時間を示す．図 8.1.4 によると，複数の処理ノードを用いて処理を行う場合，EndNode の処理時間は，単一ノードでのみ処理を行う場合（noRN）のそれと比べて減少することが見て取れる（XML 文書の処理のみに要した時間（図 8.1.4 中の processing time）はそれぞれ，noRN_val の場合 2,851msec，4RN_PAR_val の場合 488msec）．減少した分の処理は，それぞれの RelayNode（RN02–05）に分散され，EndNode にて行われるのは，各 RelayNode で処理を完了することができなかった処理のみとなる．このことより，本来送信先となる計算機で行われるべき処理が，ネットワーク中に配置される中継ノードにオフロードされ，最終的な処理時間が短縮できていることが確認できた．

これらの知見は，ネットワークに機能をもたせるという本研究方針において，処理時間におけるアドバンテージを得ることにつながる．今後この実験結果も活用しつつ，入力データサイズ，処理内容，などを考慮し，どのような構成で中継ノードでの処理を行うのかを判断するより効率的なタスクスケジューリング手法の検討を進める．

第9章　おわりに

本章では，XML 文書への並列ストリーミング処理について，本論文の各章の実験・評価によって得た知見を整理し，まとめ及び今後の展望を示す.

9.1　本論文で得た知見のまとめ

本論文では，中継ノードにて並列ストリーミング処理を行う例として PASS-Node の実装を行い，XML 文書へのストリーミング処理に関しての様々な性質の調査と，ストリーミング処理アプリケーションにおいて必須となる，中継ノードにてデータを蓄積するためのバッファの機能要件をまとめ，その改良を行った．本節ではこれらについて本論文で得，各章で述べた知見をまとめる．

まず，本論文において得られた XML 文書への並列ストリーミング処理についての知見をまとめる（第 4 章・5 章）.

1. 同じ条件であればパイプライン並列処理よりもデータ分割並列処理の方が処理時間が短い．ジョブの実行時間には，XML 文書自体への処理時間とネットワークを通じた XML 文書の転送に要した時間とが含まれる．この内，XML 文書自体への処理に要した時間はパイプライン並列・データ分割並列の差は小さい．そのためこの特性は，両者の差はデータ転送の時間に大きく起因するといえ，パイプライン並列では，処理対象ではない分割データについても，データの転送処理を行う必要があるが，データ分割並列処理の方ではその必要が無いという点が強く影響している

2. 本論文では，整形式判定と妥当性の検証という 2 種類の XML 文書の文法チェク処理を比較した．全体を通して妥当性の検証の方が整形式判定よりも多くの時間を要した．これは，妥当性の検証は整形

式判定に加えてタグの出現順序などを比較するという処理を含むためである

3. 本論文では，特徴の異なる XML 文書として，7 種類の合成 XML 文書（ファイルサイズ：小）と 3 種類の実データに基づく XML 文書（ファイルサイズ：大）への処理を比較した

   (a) ジョブの実行時間は，合成 XML 文書よりも実データに基づく XML 文書のほうが大きく，そのため，XML 文書の処理時間はファイルサイズに比例する傾向がある

   (b) 7 種類の合成 XML 文書は，ほぼ同一のファイルサイズであり含まれるタグの数も同一であるが，そのデータ構造が異なる．XML 文書内のタグのネストが深いほど処理に必要な時間は大きくなる傾向がある．これはある処理ノードで各タグへの文法チェック処理を完了するためには，同一処理ノード内で対応するタグも同時に処理する必要があるが，それを完了できない場合が増加するためである

   (c) 実データに基づく XML 文書においては，異なるファイルサイズ，異なるタグの数，異なるタグの性質にて比較を行った．これらから，ある程度近いファイルサイズであれば，XML 文書の処理時間はファイルサイズよりもたタグの数の影響が強いことがわかった．さらに，通常のタグのセットと空タグとでは，空タグの方が処理時間を必要としないことがわかる

4. 本論文では，XML 文書への並列ストリーミング処理に際して使用する処理ノードの数を増減させて比較を行った

   (a) パイプライン並列処理とデータ分割並列処理では，パイプライン並列処理においては処理ノードの数が増えても全体の処理時間が小さくなりにくい．一方，データ分割並列処理においては，処理ノードが増えた場合には全体の処理時間は小さくなる傾向がある．これは，パイプライン並列処理においては，すべての処理ノードにおいて処理を行わないデータの転送のオーバヘッドが発生するが，データ分割並列処理の場合にはそれが発生しないためである

   (b) 特定の構造を持つ XML 文書に対してデータ分割並列処理を施す場合，処理ノード数が少ない方が全体の処理時間が小さく

なることがあった．このことより，XML 文書への処理時間は，XML 文書の特徴（ファイルサイズ，タグの数，ファイルの構造）に加えて処理の分割の仕方とその割り当てにも強く影響を受けることがわかる

次に，XML 文書への並列ストリーミング処理のための，中間処理ノードで必要となるデータ蓄積用のバッファについての知見をまとめる．本論文では，ArrayList（AL），Coarse-LinkedList（C-LL），Fine-LinkedList（F-LL）の 3 種類のバッファの比較を行った（第 6 章）．

1. AL と C-LL では，C-LL の方がより効率的である．これは特に，AL においては，バッファからの要素の削除に多くの時間が必要なことに起因するものである．C-LL は LinkedList に対して格納要素へのアクセス時のロック管理に関して単純な改良を施したものではあるが,，本論文にて採り上げた XML 文書への並列ストリーミング処理においては，C-LL のようなバッファ構造が適していることがわかった

2. C-LL と F-LL においてそのパフォーマンスの差は小さいことがわかった．

3. バッファアクセスに要した時間については C-LL よりも F-LL の方がより多くの時間を要していた．このことは，F-LL で実装した細粒度ロックのための時間に起因する．

4. 以上の 2 点より，F-LL はロックコストにより C-LL よりもバッファアクセスに多くの時間を要するが，処理に要する時間は C-LL と同等であることがわかる．そのため，F-LL ではロックのコストが必要なものの，細粒度ロックによる並列動作性により，各スレッドが同時に稼働することで，同等の性能を示しているといえる

5. また，本論文で採り上げたの処理ノードの持つスレッドは受信スレッド・処理スレッド・送信スレッドの三つである．スレッド数が少ないときは，C-LL でも十分なパフォーマンスが得られるが，単一ノード内で一つの XML 文書に対して，複数スレッドを用いて部分毎に並列処理を行う場合は，F-LL の利用による効率化が期待できる．

さらに，第 8 にて，中継ノードで処理を行う並列ストリーミング処理において，処理量が多く処理が重い場合，分散実行によるオーバーヘッ

ドが生じたとしても，単体の処理ノードで集中して処理を行う時よりも，複数台の中間処理ノードを用いて処理を行った時の方がより高速に処理を完了できる場合があることを示した．加えて，単体の処理ノードと複数の処理ノードで処理を行う場合の，オフロードの確認とその効果を示し，本来送信先となる計算機で行われてきた処理を，ネットワーク中にオフロードすることの利点を示した．

これらの知見は今後，中継ノードを用いた並列ストリーミング処理についての，効率的なスケジューリング手法を検討するための重要な情報として応用が期待できる．

9.2 まとめと今後の展望

従来のネットワークにおけるアプリケーションサービスの提供形態では，クライアントとなる計算機より，サーバとなる計算機への処理要求とデータが送られ，サーバにおいて処理を行い何かしらのサービスを提供していた．このようなサービス形態においては，クライアント数の増加に比例してサーバでの処理量と負荷も増大する．これに対応する方法の一つとして，サーバ以外の場所で処理を行うオフロードと呼ばれる負荷分散手法がある．オフロードにより，サーバとなるの計算機での処理負荷が軽減され，その分より多くのクライアントからの要求に応答できるようになり，サービスとしてのスループットが向上し，ユーザに対しても同様にスループット等のサービスの向上を見込める．小尿な処理形態の中で，XML 文書のネットワーク上での中間処理のモデルとして，PASS-Node[1] が提案されている．PASS-Node は，送信元の計算機（クライアント）から送信先の計算機（サーバ）へ，XML 文書を送り，送信先の計算機にて XML 文書への処理を施す場合において，送信先となる計算機における XML 文書への文法チェック処理の一部または全てを，ネットワーク中に配置されるにオフロードする．このような処理のオフロードにより送信先の計算機の処理負荷が軽減され，最悪の処理時間の低減とサービスを利用するユーザに対してスループット等のサービスの向上，送信先の計算機がサービスにとってより本質的な処理へ専念可能となるといった利点を得ることができる．ただし，最悪の処理時間については改善される一方，中継ノードでの処理時間とオフロードによるオーバヘッドが生じ，最良の処理時間は増加する．そこで合わせて並列処理手法を採用することで，最良処理時間の低減を図る．これまで，PASS-Node に関する研究

おいては，XML 文書に対する処理のオフロードについて，数式を用いた理論的な証明と試験実装によるオフロードの確認のみが行われている段階であり，これを実現するための実証実験が不足しておりいくつかの課題が残されていた

本研究では，この種の並列アプリケーションに求められる機能要件を整理し，PASS-Node を具体的なネットワークアプリケーション（評価アプリケーション）として実装した．評価アプリケーションにおいても PASS-Node と同様に，データの送信先でデータ転送後にまとめて行うべき XML 文書への処理の一部または全てを，データ転送を行う中継ノードにオフロードすることが可能である．その際に並列分散処理技術を適用し，中継ノードの配置や個数に応じてパイプライン並列処理・データ分割並列処理の 2 種類の処理形態により，中継ノードでの共通処理機能の提供を行う．我々はこの評価アプリケーションの実装と稼働とをもって，中継ノードを用いた XML 文書に対する並列ストリーミング処理の実現可能性を実証した．

そして本論文では，この評価アプリケーションを用いた，具体的な実験と評価を行って，XML 文書に対する並列ストリーミング処理の性質を調査した．まず，XML 文書への並列ストリーミング処理についての基本的な特性を知るために，合成 XML 文書を用いたデータ分割並列処理・パイプライン並列処理についての実験を行った．この結果として，データ分割並列処理の方がパイプライン並列処理よりも高効率であること，XML 文書への処理時間は XML 文書の構造と分割の仕方の影響を強く受けることなどを示した．加えて，より実際的な環境を想定した状況での実験を行った．ここでは実環境と仮想環境を用いた場合の性質の変化の検証と，実データを元にした XML 文書を用いた評価を行い，XML 文書への並列ストリーミング処理について，実環境と仮想環境では似た性質を持つことや，XML 文書への処理時間はデータサイズと構造だけでなくタグの数と種類の影響を強く受けることを示した．これにより，本評価アプリケーションにおける成果は実環境・仮想環境どちらにも適用できることとなった．これらの成果により，XML 文書へのストリーミングデータ処理の様々な性質が明らかにされた．これらの知見は今後，同数の中間処理ノードを利用できる場合には，パイプライン並列処理よりもデータ分割並列の構成を取るなどの，並列ストリーミング処理のための，スケジューリング手法の検討に資する重要な材料となる．

さらに，評価アプリケーションにおけるストリーミングデータ処理の

性質を分析し，中継ノードにおけるストリーミングデータ処理時に，各ノードでデータを一時格納するのに使用するバッファに関して，その機能要求を洗い出し改良を行った．改良型のバッファはシンプルなデータ構造ではあるが，これまで使用してきたものよりも，効率的なものとして実装することができた．さらなる改良により，これまでは同時には単一の処理スレッドにしか対応していなかったバッファを，複数のスレッドからの同時アクセスに，パフォーマンスを損ねることなく対応させた．この成果により今後，ある単一のノード内で，単一 XML 文書への複数スレッドを用いた並列処理を効率的に行うことが可能となった．この成果は本研究における XML 文書に対する文法チェックと，同様に，処理前後でデータの順番が変化しないという性質を持つような，別の処理を行う場合にも応用が期待できる．以上により，我々は，ノード内共有バッファの改良により経路上のノードで行うストリーミングデータ処理の効率化を実現した．

また，XML 文書は高い汎用性を持つデータ形式であり，様々な場面で広く利用されているが，同時に，構造を持つデータのため本質的に並列処理に向かないデータ構造である．そのため，並列処理の難しい XML 文書を用いて得た知見は今後，より難易度の易しい，他のデータ形式を対象とした処理について，本研究にて採り上げたような，データを送信元の計算機から送信先の計算機に送り，送信先の計算機にて処理を施すようなアプリケーションへの応用が期待できる．

加えて，これまでの実験結果を元に，最も効率的だと思われる構成にて行った並列ストリーミング処理と，並列処理を適用せず単一のノードのみで処理を行った場合の比較を行った．これまでは，中継ノードにおける並列ストリーミング処理には，データ転送や処理分割などのオーバーヘッドが含まれるため，単一ノードでの処理よりも多くの実行時間を必要としていた．しかし，処理データのサイズが大きく，処理が重く，多数のノードでのデータ分割並列処理を行う場合に，分散化によるオーバヘッドを加味してなお，単一ノードでのみ処理する場合よりも，オフロードによる処理の高速化が可能となる場合があることがわかった．さらにここでは，オフロード時と非オフロード時のそれぞれのノードの処理負荷を計測し，オフロードによる処理負荷の軽減・分散について具体的な効果を示した．

近年，ネットワークアプリケーションの重要性が増しており，現在，単なるデータを送るだけとも言えるライフラインとしてのネットワークに

対して，ネットワーク自体が高度な処理機能を持つことが期待されるようになっている．このようなネットワークは高機能ネットワークと呼ばれ，我々は，種々のアプリケーションの連携のサポートや効率的な資源利用，ネットワーク自体の付加価値の向上につながると考えている．高機能ネットワークにおいては，フィルタリングやデータ変換などの様々な処理機能の提供が期待される．高機能ネットワークの実現により，ネットワークインフラを保有する物理回線事業者にとっては，現在単なるライフラインに留まっているネットワークに付加価値を持たせ，遊休資源を活用した資源利用の最適化による新たな利益を得る機会が生まれる．我々が今回実装・評価を行った，中継ノードを用いたXML文書への並列ストリーミング処理を行う評価アプリケーションは，このような高機能ネットワークの一例といえ，高機能ネットワークの実現可能性を示すものである．

最後に今後の展望について述べる．今後，複数の処理スレッドを用いた単一XML文書への並列処理についての評価を行う予定である．この評価においては，本論文では優位点の小さかった複数スレッドでの同時アクセスに適応させた改良型のバッファの特徴を活かすことが可能である．さらに，現在の評価アプリケーションでは，単一XML文書への処理のみを対象としているが，複数のXML文書を同時に処理する場合に有効なスケジューリング手法の考察を検討している．加えて，XML文書への文法チェック以外の異なった性質を持つ他の有用な機能（e.g. フィルタリング，XSLやXSLTを用いたフォーマット変換）の評価などを通じて，並列ストリーミング処理の特徴をさらに明らかにしていく．我々は将来の目標とする高機能ネットワークにおいては，XML文書だけでなく他のデータ形式に対しても中継ノードで処理を行う[63]などの環境も視野に入れている．この種の環境では，多数のWebサーバ，モバイル機器，ネットワークアプリケーションなどが，中継ノードでストリーミングデータ処理を行う機能を持ったネットワークによって相互に接続される．この際に処理されるデータはXML文書以外にも様々なデータ形式が対象となり，処理対象となるデータも膨大となることが予想される．最終的には，これら異なる処理要求を持つ多様・多量のデータを，様々な特徴を持つ計算資源を適材適所で利用し，高機能ネットワークにおける効率的なスケジューリング手法の検討を目指す．

謝辞

本論文を執筆するにあたり，有益なご教示・ご助言を賜りました九州工業大学情報工学部八杉 昌宏教授，江島 俊朗教授，吉田 隆一教授，鶴正人教授，小出 洋准教授に深く御礼申し上げます．

特に小出 洋准教授には，学部生時代の研究室配属時より学部，博士前期課程，博士後期課程以後から現在に至るまで様々な面でお世話になりました．

また，本研究を遂行するにあたって，研究開始当初から議論いただき，様々なご助言を賜りました，九州工業大学ネットワークデザイン研究センター客員教授及びカリフォルニア大学ロサンゼルス校の Dirceu Cavendish 先生にもお礼申し上げます．

さらに，九州工業大学情報工学部小出研究室にて同じ時間を過ごし，様々な手助けを頂いた研究室の仲間にも同じく御礼申し上げます．中でも特に，同研究室の先輩である吉永 一美氏におきましては，研究室配属から博士後期進学後にまで渡って公私両面で多くのご助言を賜りました．

最後に，これまで寛容さを持って暖かく見守ってくれた両親並びに弟・妹，祖母らの家族に対して感謝の言葉を送りたく存じます．

参考文献

[1] Dirceu Cavendish and K. Selçuk Candan. Distributed XML processing: Theory and applications. *Journal of Parallel Distributed Computing*, Vol. 68, No. 8, pp. 1054–1069, August 2008.

[2] World Wide Web Consortium (W3C). Extensible Markup Language (XML) 1.1 (Second Edition). `http://www.w3.org/TR/xml11/`.

[3] Seung Hun Kim, Keunsoo Kim, Changmin Lee, and W.W. Ro. Offloading of media transcoding for high-quality multimedia services. *Consumer Electronics, IEEE Transactions on*, Vol. 58, No. 2, pp. 691–699, 2012.

[4] A. Singh, A. Trivedi, K. Ramamritham, and P. Shenoy. Ptc : proxies that transcode and cache in heterogeneous web client environments. In *Proc. of the 3rd International Conference on Web Information Systems Engineering (WISE) 2002*, pp. 11–20, 2002.

[5] Zhenyun Zhuang and Chun Guo. Optimizing cdn infrastructure for live streaming with constrained server chaining. In *IEEE 9th International Symposium on Parallel and Distributed Processing with Applications (ISPA) 2011*, pp. 183–188, May 2011.

[6] Akamai. Cloud computing, enterprise, mobile security, solutions — akamai. `http://www.akamai.com/`（2013 年 10 月 7 日確認）.

[7] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. *SIGOPS Oper. Syst. Rev.*, Vol. 36, No. SI, pp. 131–146, December 2002.

[8] Yao-Chung Fan and A.L.P. Chen. Energy efficient schemes for accuracy-guaranteed sensor data aggregation using scalable counting. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 24, No. 8, pp. 1463–1477, 2012.

[9] Ignacio Solis and Katia Obraczka. In-network aggregation trade-offs for data collection in wireless sensor networks. *Int. J. Sen. Netw.*, Vol. 1, No. 3/4, pp. 200–212, January 2006.

[10] Masayoshi Shimamura, Takeshi Ikenaga, and Masato Tsuru. Advanced Relay Nodes for Adaptive Network Services - Concept and Prototype Experiment. In *Proc. of International Conference on Broadband, Wireless Computing, Communication and Applications (BWCCA) 2010*, pp. 701–707, Los Alamitos, CA, USA, November 2010.

[11] David L. Tennenhouse and David J. Wetherall. Towards an active network architecture. *SIGCOMM Comput. Commun. Rev.*, Vol. 37, No. 5, pp. 81–94, October 2007.

[12] Y. Kanada, K. Shiraishi, and A. Nakao. Network-virtualization nodes that support mutually independent development and evolution of node components. In *IEEE International Conference on Communication Systems (ICCS) 2012*, pp. 363–367, 2012.

[13] セールスフォース・ドットコム. `http://www.salesforce.com/jp/`.

[14] J.L. Brickell and A.M. Langer. Adapting to the data explosion: Ensuring justice for all. In *Proc. of IEEE International Conference on Systems, Man and Cybernetics (SMC) 2009*, pp. 86–90, October 2009.

[15] John Gantz and David Reinsel. Extracting Value from Chaos, June 2011. International Data Corporation (IDC) IVIEW.

[16] JPIX. テクニカル情報:トラヒック. `http://www.jpix.ad.jp/jp/technical/traffic.html`（2013 年 2 月 24 日確認）.

[17] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers. *Big data : The next frontier for innovation , competition , and productivity*, Vol. 364. May 2011.

[18] Ian Foster. What is the Grid ? A Three Point Checklist. *GRID today*, Vol. 1, No. 6, pp. 32–36, 2002.

[19] I. Foster, Yong Z., I. Raicu, and S. Lu. Cloud Computing and Grid Computing 360-Degree Compared. In *Proc. of Grid Computing Environments Workshop (GCE), 2008*, pp. 1 –10, November 2008.

[20] Toyotaro Suzumura and Tomoaki Oiki. StreamWeb: Real-Time Web Monitoring with Stream Computing. In *Proc. of IEEE International Conference on Web Services (ICWS) 2011*, pp. 620–627, July 2011.

[21] Twitter. Twiter and Streaming API, Search Service. `http://twitter.com`.

[22] Facebook. The Facebook Open Stream API. `http://www.facebook.com/`.

[23] Eric Keller and Jennifer Rexford. The "Platform as a service" model for networking. In *Proc. of the 2010 internet network management conference on Research on enterprise networking*, INM/WREN'10, pp. 4–10, April 2010.

[24] NTT Communications. Cloud^n. `http://www.ntt.com/cloudn/`.

[25] KDDI. KDDI クラウドサーバサービス. `http://www.kddi.com/business/cloud/`.

[26] Internet Initiative Japan(IIJ). IIJ GIO. `http://www.iij.ad.jp/GIO/`.

[27] 新世代ネットワーク推進フォーラム. NWGN : New Generation Network. `http://forum.nwgn.jp/index.html`.

[28] 情報通信研究機構（NICT）. JGN-X. `http://www.jgn.nict.go.jp/`.

[29] European Commission. ICT Challenge 1: Pervasive and Trusted Network and Service Infrastructures. http://cordis.europa.eu/fp7/ict/programme/challenge1_en.html.

[30] European Commission. FIRE – Future Internet Research and Experimentation. http://cordis.europa.eu/fp7/ict/fire/.

[31] National Science Foundation. NSF NeTS FIND Initiative. http://www.nets-find.net/.

[32] National Science Foundation. Global Environment for Network Innovation (GENI). http://www.geni.net/.

[33] ITU-T. Focus Group on Future Networks (FG FN). http://www.itu.int/en/ITU-T/focusgroups/fn/Pages/default.aspx.

[34] H. Koide, T. Hirayama, A. Murasugi, T. Hayashi, and H. Kasahara. Meta-scheduling for a Cluster of Supercomputers. In *Proc. of ICS'99 Workshop*, pp. 63–69, June 1999.

[35] Richard E. Ladner and Michael J. Fischer. Parallel prefix computation. *Journal of ACM (JACM)*, Vol. 27, No. 4, pp. 831–838, October 1980.

[36] ISO/IEC. ISO/IEC 19757 - Document Schema Definition Languages:DSDL (part 9).

[37] ORACLE. Java Remote Method Invocation API (Java RMI). http://docs.oracle.com/javase/7/docs/technotes/guides/rmi/index.html.

[38] J. D. Ullman. NP-complete scheduling problems. *Journal of Computer and System Sciences*, Vol. 10, No. 3, pp. 384–393, June 1975.

[39] M. R. Garey and D. S. Johnson. Tutorial: hard real-time systems. chapter Complexity results for multiprocessor scheduling under resource constraints, pp. 205–219. 1989.

[40] James E. Kelley Jr and Morgan R. Walker. Critical-path planning and scheduling. In *Proc. of IRE-AIEE-ACM '59 (Eastern)*, pp. 160–173, December 1959.

[41] Tarek Hagras and Jan Janecek. A static task scheduling heuristic for homogeneous computing environments. In *Proc. of 12th Euromicro Conference on Parallel Distributed and Network-Based Processing, 2004*, pp. 192–198, February 2004.

[42] Manimaran G. and Murthy C. Siva Ram. An Efficient Dynamic Scheduling Algorithm for Multiprocessor Real-Time Systems. *IEEE Transactions on Parallel Distributed System*, Vol. 9, No. 3, pp. 312–319, March 1998.

[43] K. Yoshinaga, Y. Uratani, and H. Koide. Utilizing Multi-Networks Task Scheduler for Streaming Applications. In *Proc. of International Conference on Parallel Processing - Workshops*, pp. 25–30, September 2008.

[44] ORACLE. Oracle Solaris Containers. `http://www.oracle.com/technetwork/server-storage/solaris/containers-169727.html`.

[45] OASIS Advancing open standards for information society. OASIS Open Document Format for Office Applications (OpenDocument) TC. `https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=office`.

[46] ISO/IEC. ISO/IEC 26300 Information technology – Open Document Format for Office Applications (OpenDocument) v1.0.

[47] 浦谷芳幸, カベンディッシュジルセウ, 小出洋. [奨励講演] 中継計算機で行うストリーミングデータ処理の特性. 電子情報通信学会 NS 研究会技術研究報告, 第 110 巻, pp. 81–86, 11 月 2010 年.

[48] 浦谷芳幸, 小出洋. 中継計算機でストリーミングデータ処理を行う並列アプリケーションの実装と評価. 電子情報通信学会 NS 研究会技術研究報告, 第 109 巻, pp. 133–138, 10 月 2009 年.

[49] Y. Uratani, H. Koide, D. Cavendish, and Y. Oie. Characterizing Distributed XML Processing – Moving XML Processing from Servers to Networking Nodes. In *Proc. of 7th International Conference on*

Web Information Systems and Technologies (WEBIST) 2011, pp. 41–50, May 2011.

[50] Y. Uratani, H. Koide, D. Cavendish, and Y. Oie. Distributed XML Processing over Various Topologies: Characterizing XML Document Processing Efficiency. In *Web Information Systems and Technologies*, Vol. 101 of *Lecture Notes in Business Information Processing*, pp. 57–71. Springer Berlin Heidelberg, January 2012.

[51] Y. Uratani, H. Koide, D. Cavendish, and Y. Oie. Distributed XML Processing over Various Topologies - Pipeline and Parallel Processing Characterization. In *Proc. 8th International Conference on Web Information Systems and Technologies (WEBIST) 2012*, pp. 116–122, April 2012.

[52] Wei Zhang and Robert van Engelen. A Table-Driven Streaming XML Parsing Methodology for High-Performance Web Services. In *Proc. of the IEEE International Conference on Web Services*, ICWS '06, pp. 197–204, July 2006.

[53] Michael R. Head and Madhusudhan Govindaraju. Approaching a parallelized XML parser optimized for multi-coreprocessors. In *Proc. of the 2007 workshop on Service-oriented computing performance*, SOCP '07, pp. 17–22, June 2007.

[54] Wei Lu and Dennis Gannon. Parallel XML processing by work stealing. In *Proc. of the 2007 workshop on Service-oriented computing performance*, SOCP '07, pp. 31–38, June 2007.

[55] Michael R. Head and Madhusudhan Govindaraju. Performance enhancement with speculative execution based parallelism for processing large-scale XML-based application data. In *Proc. of the 18th ACM international symposium on High performance distributed computing*, HPDC '09, pp. 21–30, June 2009.

[56] 浦谷芳幸, カベンディッシュジルセウ, 本間伸雄, 小出洋. XML データへのストリーミング処理における共有バッファ機構の改善. 電子情報通信学会 和文論文誌 D インターネット技術とその応用特集号, 第 J96-D 巻, pp. 1447–1457, 6 月 2013 年.

[57] Maged M. Michael and Michael L. Scott. Simple, fast, and practical non-blocking and blocking concurrent queue algorithms. In *Proc. of the 15th annual ACM symposium on Principles of distributed computing*, PODC '96, pp. 267–275, May 1996.

[58] Rongxin Chen and Husheng Liao. ParaParse: A parallel method for XML parsing. In *Proc. of IEEE 3rd International Conference on Communication Software and Networks (ICCSN) 2011*, pp. 81 –85, May 2011.

[59] J. G. S. Gonzalez, V. J. S. Sosa, A. R. Montes, and J. C. R. Olivares. Multi-Format Web Content Transcoding for Mobile Devices. In *Proc. of th Mexican International Conference on Computer Science, 2006*, pp. 109–115, 2006.

[60] World Wide Web Consortium (W3C). Extensible Stylesheet Language (XSL) Version 1.1. `http://www.w3.org/TR/xsl/`.

[61] World Wide Web Consortium (W3C). XSL Transformations (XSLT) Version 2.0. `http://www.w3.org/TR/xslt20/`.

[62] R.G. Bartlett and M.W. Cook. XML security using XSLT. In *Proc. of the 36th Annual Hawaii International Conference on System Sciences, 2003*, pp. 122–127, January 2003.

[63] M. Shimamura, T. Ikenaga, and M. Tsuru. Advanced Relay Nodes for Adaptive Network Services - Concept and Prototype Experiment. In *Proc. of International Conference on Broadband, Wireless Computing, Communication and Applications (BWCCA), 2010*, pp. 701 –707, November 2010.

[64] R. Furuhashi and A. Nakao. Applications of wide-area network slicing for improving cloud platform access by OpenTag. In *Proc. of 4th International Conference on Communication Systems and Networks (COMSNETS), 2012*, pp. 1 –7, January 2012.

[65] R. Furuhashi and A. Nakao. OpenTag: Tag-Based Network Slicing for Wide-Area Coordinated In-Network Packet Processing. In *Porc.*

of IEEE International Conference on Communications Workshops (ICC), 2011, pp. 1 –5, June 2011.

著者発表論文

査読付き雑誌

1. 浦谷芳幸, カベンディッシュジルセウ, 本間伸雄, 小出洋. XMLデータへのストリーミング処理における共有バッファ機構の改善, 電子情報通信学会 和文論文誌D インターネット技術とその応用特集号, vol.J96-D, No.6, pp.1447–1457, 2013年6月.
（6章の内容）

2. **Yoshiyuki Uratani**, Hiroshi Koide, Dirceu Cavendish, and Yuji Oie. Distributed XML Processing over Various Topologies: Characterizing XML Document Processing Efficiency. *Lecture Notes in Business Information Processing* of *Web Information Systems and Technologies*, pp. 57–71. Springer Berlin Heidelberg, 2012.
（4章の内容）

査読付き会議

3. **Yoshiyuki Uratani**, Hiroshi Koide, Dirceu Cavendish, and Yuji Oie. Distributed xml processing over various topologies - pipeline and parallel processing characterization. In *Proc. of 7th International Conference on Web Information Systems and Technologies (WEBIST) 2012*, pp. 116–122, April 2012.
（short paper 採択率31%）（5章の内容）

4. **Yoshiyuki Uratani**, Hiroshi Koide, Dirceu Cavendish, and Yuji Oie. Characterizing Distributed XML Processing – Moving XML Processing from Servers to Networking Nodes. In *Proc. of 7th*

International Conference on Web Information Systems and Technologies (WEBIST) 2011, pp. 41–50, May 2011.
（full paper 採択率 9%）（4 章の内容）

その他

5. 浦谷芳幸, カベンディッシュジルセウ, 小出洋. [奨励講演] 中継計算機で行うストリーミングデータ処理の特性. 電子情報通信学会技術研究報告, 第 110 巻, no.286, pp. 81–86, 京都府, 2010 年 11 月.

6. 浦谷 芳幸, 小出洋. 中継計算機でストリーミングデータ処理を行う並列アプリケーションの実装と評価, 電子情報通信学会技術研究報告, 第 109 巻, no.228, pp. 133–138, 熊本県, 2009 年 10 月.

7. 浦谷芳幸, 小出洋. 並列分散処理を用いたリアルタイムネットワーク監視システム, 情報処理学会 火の国情報シンポジウム 2007 CD-ROM, 鹿児島県, 2007 年 3 月.

共著論文

8. Dirceu Cavendish, Kazumi Yoshinaga, Shohei Washizu, **Yoshiyuki Uratani**, Hiroshi Koide and Yuji Oie, :Characterizing Transactions with Data Transfer on Multi-Server Systems, Proc. of the 3rd International Workshop on Information Network Design (WIND2010), pp. 404-408 November 2010.

9. Kazumi Yoshinaga, **Yoshiyuki Uratani**, and Hiroshi Koide. Utilizing Utilizing Multi-Networks Task Scheduler for Streaming Applications. In *Proc. of International Conference on Parallel Processing - Workshops*, pp. 25–30, September 2008.