RaSC: 高速なストリーム通信をサポートする言語処理プログラムの高速化・高並列化ミドルウェア

田仲 正弘 † 大竹 清敬 † 鳥澤 健太郎 † 田浦 健次朗 §
† (独) 情報通信研究機構 (NICT) 情報分析研究室
§ 東京大学大学院情報理工学系研究科
{mtnk, kiyonori.ohtake, torisawa}@nict.go.jp, tau@eidos.ic.i.u-tokyo.ac.jp

1 はじめに

近年，大規模テキストを対象としたデータ分析が注目を浴びている．(独) 情報通信研究機構で開発している大規模 Web 情報分析システム WISDOM2013[4] においても，Web から収集される数十億規模の文書に対して，各種の深い解析を行う言語処理を適用することによって，様々な分析を実現している．例として，図 1 に WISDOM2013 に組み込まれた未来分析 [1] と呼ぶ分析機能を示す．この機能は，Web 文書から「森林破壊が続く → 地球温暖化が進行する」といった因果関係の記述を抽出し，未来に起きうる様々なシナリオを仮説として提示する．また，対災害情報分析システム [3] では，災害直後に発信される大量のツイートを解析することで，救援物資の要望や付随する地理情報の発見と整理などの分析を実現している．

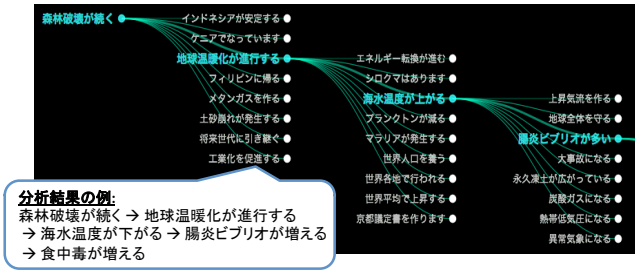

図 1: WISDOM2013 における未来分析 [1]

こうした深い解析では，複数の言語処理プログラムをパイプライン的に結合することが多く，また計算負荷も高い．そのため，大規模テキストへの適用には，多数の計算機を用いた並列処理が必須である．しかし多くの言語処理プログラムは，主に研究目的に開発されており，大規模な並列処理に適さない．

そこで我々は，既存の言語処理プログラムを高速・高並列に組み合わせて実行可能にするミドルウェア RaSC (**Ra**pid **S**ervice **C**onnector) を開発した．RaSC 上でプログラムを稼働させることで，それらのプログラムをネットワークを介して高速に呼び出すことができるようになり，複数の計算機を用いた並列処理が容易になる．RaSC 上で稼働するプログラム同士は，ストリーミングによるデータ送受信を行うことが可能となり，プログラム間のデータ交換に起因する CPU アイドル時間を小さくできる．さらに，RaSC はプログラムを起動した状態に保つため，断続的に到着するデータに対しても，起動時のデータのロード等によるオーバヘッドを避けられる．また，RaSC は並列処理の実行や結果の集約の実装を簡単化する機構を備えている．

以降では，大規模テキストに言語処理プログラムを適用するにあたっての課題と，RaSC による解決法を示す．さらに，Web 情報分析システム WISDOM2013 での利用例を紹介する．

2 大規模テキスト分析の課題

図 2 は，継続的に流入する Web 文書を入力として，複数の言語処理プログラムを実行する例と共に，大規模テキストの分析における課題を示している．以降でそれぞれの課題を説明する．なおこれらの課題はいずれも，1 台の計算機を用いて，バッチ処理的かつ直列的に実行する場合には顕在化しないものである．

2.1 データ通信に伴う CPU アイドル時間

テキストの解析では，しばしば複数の言語処理プログラムからなるパイラインを構成する．このとき，前段のプログラムの出力をファイルに書き出して，後段のプログラムの入力として与えるといった方法が取られることがある．しかしこの方法では，前段のプログ

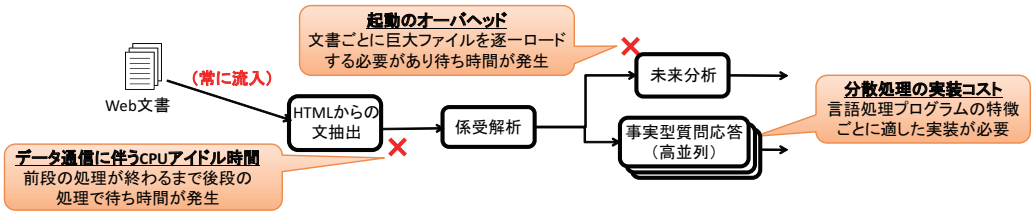

図 2: 大規模テキスト分析の課題

ラムが出力を完了するまで後段のプログラムが実行できない．今日ではほとんどの CPU はコアを複数持つが，この方法では計算機資源が有効に活用できないことになる．言語処理プログラム間をパイプによって接続する方法も頻繁に用いられるが，プログラムごとに処理速度が異なると，後段のプログラムで待ち時間が生じる．いずれの方法でも，複数のパイプラインの並列実行によってアイドル時間を減少できるが，全ての言語処理プログラムのプロセスが，並列実行数だけ起動され，アイドル状態のプログラムについてもメモリが消費される．言語処理プログラムには，巨大な辞書データやモデルデータによって，使用メモリが数 GB に及ぶものもあり，メモリの無駄が大きい．

2.2 起動オーバヘッド

言語処理プログラムの多くは，起動時に辞書データやモデルデータをディスクから読み込み，内部表現に変換してメモリに読み込む．このような方式は，あらかじめ蓄積されたデータのバッチ処理に適する．しかし，Web から定常的に収集される文書に対して順次処理を適用するなどの場合には，1 件ずつ言語処理プログラムを起動するとオーバヘッドが大きい．入力データをある程度バッファリングすることによってオーバヘッドの影響を小さくすることができるが，バッファリングしている間は後段のプログラムが実行できず，前述の CPU アイドル時間が大きくなる．

読み込むデータを RAM ディスクなど高速に読み出し可能なストレージに配置することでこのオーバヘッドを小さくすることもできるが，プロセス内でのデータ構造の構築コストを軽減することはできない．

2.3 並列処理実装コスト

1 台の計算機では十分な実行速度が得られなかったり，処理に必要なデータを収納できない場合には，言語処理プログラムを複数の計算機で並列分散実行する必要がある．しかし，複数の計算機ノードでプログラムを実行し，各計算機ノードで出力される結果を段階的に取得したり，その結果に随時ランキング等の集約処理を適用することは，さほど一般的な処理ではない．そのため，通常の設計パターンやライブラリが適用できず，開発コストが大きくなる．

3 高速化・高並列化ミドルウェア RaSC

前述の問題を解決するため，我々は言語処理プログラムの実行を高速化･高並列化するミドルウェア RaSC を開発した．以下にその主要な機構について述べる．

3.1 プログラムのストリーミングサーバ化

RaSC は，標準入出力を通じて入出力を行うプログラムを，高速ストリーミングに対応したサーバとする機構を提供する．ストリーミングによる入出力を行うことで，複数のプログラムを接続した際のアイドル時間を減少させる．また，プログラムを常時起動状態に保ち，起動時のオーバヘッドを減少させる．RaSC は Java で開発されているが，対象となるプログラムの実装言語は問わない．主な呼び出しプロトコルとして MessagePack RPC を用いるが，相互運用性を考慮し，JSON RPC, SOAP 等のプロトコルにも対応している．

図 3 に，RaSC の導入によって言語処理プログラムをサーバ化した場合の構成を示す．リクエストを受けた際，RaSC は言語処理プログラムのプロセスを新たに生成する．リクエストに対応する処理が終了しても，プロセスは終了されず，以降のリクエストの実行に再利用される．先に受けたリクエストによって起動済みのプロセスが利用中に，新たなリクエストを受けた場合には，定めたプロセス数の上限に達するまで新たなプロセスを生成する．プロセス数が上限に達していた場合は，リクエストを実行待ちキューに入れる．いずれかのプロセスが空き状態になると，実行待ちキューのリクエストを順に実行する．一定時間内にいずれのプロセスも空き状態にならなければ，エラーが返される．

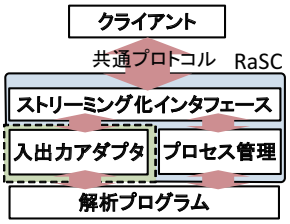

図 3: サーバの構成

表 1: WISDOM2013 におけるサーバ化事例

適用例	実装コード量	速度向上
HTML からの文抽出	12 行	2.8 倍
構文解析（J.DepP[1]）	3 行	8.1 倍
因果関係抽出 [1]	6 行	126 倍

RaSC は，1 件の入力データをプログラムの標準入力に書き込む一方で，同時に標準出力から 1 件の入力に対応する出力データを読み取る．そのため，ファイル等の標準入出力以外を用いるプログラムや，処理単位の区切りを出力しないプログラムについては，改修が必要となる．この改修はプログラムごとに必要であるが，その規模は多くの場合数行〜数十行であり，コストは限定的である．また改修の内容は多くのプログラムで定型的なものである．表 1 は，WISDOM2013 で用いた言語処理プログラムでの改修の規模と，高速化の効果の例を示している．ここでは，ストリーミングで断続的に到着する入力を処理するのに，1 つのメインプログラムから複数の言語処理プログラムをコンポーネントとして組み合わせて実行することを想定し，Java で実装されたプログラムから Web 文書 1 件ずつを言語処理プログラムを直接起動して処理する場合と，RaSC 上で実行する場合について比較している．起動時に読み込む辞書データが特に巨大な因果関係抽出では，顕著な効果が得られている．

3.2 大規模分析の並列分散化

RaSC では前節の機構でサーバ化されたプログラムの並列分散機構（図 4）を提供する．ユーザが分散配置されたサーバのアドレスとポートを指定すると，並列にリクエストを発行し，結果を収集して返す．プログラムの性質に応じて，各計算機での処理結果を得られた順に単純に返す，全ての結果が揃ってからランキング等の集約処理を実行して返す，処理結果が揃うまでの過程でも定期的にランキングを実行して順位に変更のあったデータのみを返すなどの機能を備えている．

[1] http://www.tkl.iis.u-tokyo.ac.jp/~ynaga/jdepp/

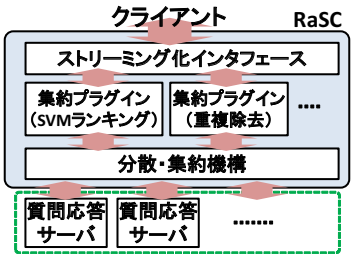

図 4: 並列分散処理の構成

ユーザは必要に応じて各種の集約アルゴリズムを実装し，プラグインとして指定できる．

4 利用例: 大規模 Web 情報分析システム WISDOM2013

（独）情報通信研究機構で開発している大規模 Web 情報分析システム WISDOM2013 では，日々収集される Web 文書に，十数種の言語処理プログラムや SVM・CRF などの分類器を適用している．そこで，RaSC の導入による高速化を図った．HTML 文書からの文抽出・構文解析・評価情報抽出等からなる処理を，無作為に選択した 100 件の Web 文書に対し，並列化を行わずに適用したところ，RaSC 導入前に 240 秒を要したものが，80 秒に短縮された．既存プログラムの改修及び入出力アダプタの実装量は，プログラムあたり数行〜数十行である．さらに，12 個あるいは 6 個の CPU コアを持つ計算機ノード 140 台を用い，日々収集される Web ページを 2200 並列で処理したところ，一日あたり約 2000 万件の処理が実現された．この処理速度は，Web ページ収集のネットワーク帯域の制限によるものであり，解析の実行だけであれば一日あたり 1 億件の文書の解析も可能であると見込まれる．

さらに，数十テラバイトに及ぶ蓄積された解析結果を 40 台の計算機ノードに分散配置し，RaSC の並列分散化機構を用いて解析結果への並列アクセスを実現した．これらの解析結果を用いる質問応答等では，多くの場合で各ノードごとの処理に数百 ms 以上を要する．一方で，この並列分散化機構によるオーバヘッドは数 ms に止まり，その影響は小さいと言える．

5 関連フレームワーク

コモディティハードウェアの低価格化によって，大規模データ処理には，1 台の超高速な計算機を用いるよりも，多数の安価な計算機ノードを組み合わせるこ

とが一般的になっている．そのため，処理の高並列化による大規模データ処理を目的としたソフトウェアが注目を浴びている．代表例は Hadoop[2] であり，簡便に堅牢な並列処理を記述できる．また，バッチ処理に特化された Hadoop では不可能なストリーミング処理を扱うことを目的に，Twitter で開発された Storm[3] や Yahoo によって開発された S4[4] などのミドルウェアも相次いで公開されている．

しかしこれらのミドルウェアは，それぞれのミドルウェアの実装言語で記述された処理を並列実行したときに効率よく実行できる．外部プログラムを使用するためのプラグインも作られているが，例えば巨大な辞書データやモデルデータへの多数のランダムアクセスを必要とするようなプログラムは，2 章に述べた理由から高速に実行できない．一方 RaSC では，既存の言語処理プログラムを実行することを目的としており，わずかの改修によって高速に実行が可能である．

より応用に特化したミドルウェアとして，手軽にオンライン機械学習の分散化を可能とする Jubatus[5] があり，最大 100 台程度のストリーミングによる並列分散実行が可能であるとされている．しかし，既存プログラムを高速に実行するための機構は持たない．

多様な自然言語処理ツールを組み合わせることを目的としたミドルウェアには UIMA[6] があり，質問応答システム Watson など高速システムでの利用実績がある．しかし UIMA で高速な解析を行うには，解析プログラムを UIMA に合わせて大幅に作り直すことが必要になる．この点で，研究の成果として作成された深い分析のためのプログラムを，低コストで高速・高並列実行可能にする RaSC とは目的が異なる．

表 2 に，これらのミドルウェアと RaSC の特徴の簡単な比較を示す．

表 2: 関連フレームワークの特徴

	既存プログラム利用	並列規模	簡便さ	ストリーミング処理
Hadoop	△	◎	◎	×
Storm	△	◎	◎	◎
Jubatus	△	○	◎	◎
UIMA	○	△	△	×
RaSC	◎	○	○	◎

[2] http://hadoop.apache.org/
[3] http://storm-project.net/
[4] http://incubator.apache.org/s4/
[5] http://jubat.us/
[6] http://uima.apache.org/

6 おわりに

本稿では，深い解析を行う言語処理プログラムを高速化・高並列化するミドルウェア RaSC を紹介した．RaSC は言語処理プログラムの高速化･高並列化によって大規模テキストの分析を可能とするものであり，すでに大規模 Web 情報分析システム WISDOM2013 に導入され，未来分析 [1]，Why 型質問応答 [2] などの高度な分析の高速化に寄与している．また，同じく（独）情報通信研究機構で開発された対災害情報分析システム [3] にも導入されており，今後災害時におけるツイートの解析 [5] などに利用される予定である．

RaSC は本稿の発表とともに，オープンソースで公開する予定であり，今後より幅広い言語処理プログラムや計算機環境に対応可能なよう，拡張を進めていく．

参考文献

[1] Chikara Hashimoto, Kentaro Torisawa, Stijn De Saeger, Jong-Hoon Oh, and Jun'ichi Kazama. Excitatory or inhibitory: A new semantic orientation extracts contradiction and causality from the web. In *EMNLP-CoNLL 2012*, pp. 619–630, 2012.

[2] Jong-Hoon Oh, Kentaro Torisawa, Chikara Hashimoto, Motoki Sano, Stijn De Saeger, and Kiyonori Ohtake. Why-question answering using intra- and inter-sentential causal relations. In *ACL 2013*, pp. 1733–1743, 2013.

[3] Kiyonori Ohtake, Jun Goto, Stijn De Saeger, Kentaro Torisawa, Junta Mizuno, and Kentaro Inui. Nict disaster information analysis system. In *IJCNLP 2013 (Demonstration Track)*, 2013.

[4] Masahiro Tanaka, Stijn De Saeger, Kiyonori Ohtake, Chikara Hashimoto, Makoto Hijiya, Hideaki Fujii, and Kentaro Torisawa. Wisdom2013: A large-scaleweb information analysis system. In *IJCNLP 2013 (Demonstration Track)*, 2013.

[5] István Varga, Motoki Sano, Kentaro Torisawa, Chikara Hashimoto, Kiyonori Ohtake, Takao Kawai, Jong-Hoon Oh, and Stijn De Saeger. Aid is out there: Looking for help from tweets during a large scale disaster. In *ACL 2013*, pp. 1619–1629, 2013.