2013年度
卒業論文

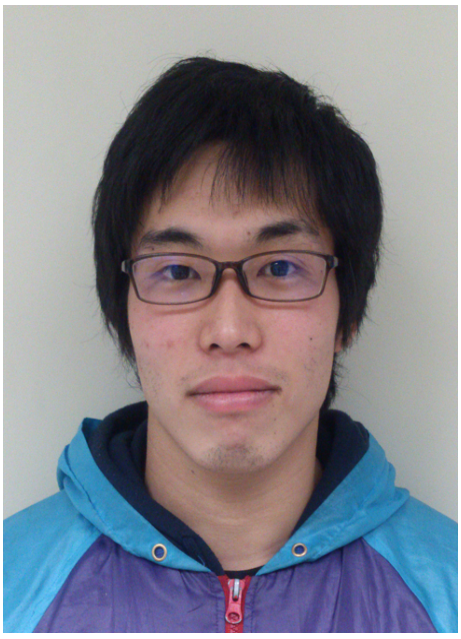
並列分散ストリームデータ処理基盤S4を用いた Incremental Rocchioによる文書フィルタリング

神戸大学工学部情報知能工学科

川原 駿

指導教員　　上原 邦昭　教授

2014年2月20日



並列分散ストリームデータ処理基盤 S4 を用いた Incremental Rocchio による文書フィルタリング

川原 駿

要 旨

今日,「知識ベース」が医学，金融，政治などあらゆる分野で重要となってきている．ここで「知識ベース」とは，Wikipedia に代表される，あるエンティティについての情報や関係といった知識を集積したもののことである．通常,「知識ベース」は人の手によって，新しい情報を収集し，編集することでページの内容を最新の情報に保っている．しかし，編集者の数に対し，エンティティ数の方が圧倒的に多いため，編集が遅れてしまい，ページの内容を常に最新の状態に保つことができないという問題がある．そこで，ニュース記事などのストリームデータをリアルタイムにフィルタリングし，自動的に新しい情報を高精度に抽出するための手法の開発が求められている．本研究では，そのような手法として，S4 と Incremental Rocchio を用いた手法を提案する．並列分散ストリームデータ処理基盤である S4 を用いることで大規模ストリームデータのリアルタイム処理に対応し，既存の高精度なフィルタリング手法として知られる Incremental Rocchio を本問題に適用する．評価実験では，Incremental Rocchio を行わない場合と精度を比較することで，Incremental Rocchio により文書フィルタリングの精度が向上していることを示す．また，1 秒間に処理できる文書数を測定し，大規模ストリームデータをリアルタイムに処理できていることを示す.

目 次

表 目 次

図 目 次

第1章　序論

近年，ストリームデータをリアルタイムに解析する研究が注目されている [6]．ストリームデータとは，際限なく到来する時刻順のデータのことで，例えば，マイクロブログ [1] や，株価情報などがある．マイクロブログとして有名なサービスとして Twitter がある．Twitter の投稿内容のことを Tweet と呼び，それをリアルタイム解析することで，今のトレンド情報を得るといったことができる [9]．多くのニュースサイトや，マイクロブログ等の SNS が存在する近年，生成されるストリームデータはビッグデータとなっている．ビッグデータとは，従来のデータ処理アプリケーションで処理することが困難なほど巨大で複雑なデータ集合の集積物である．ビッグデータを処理するには，並列分散処理基盤を用いられることが多い．並列分散処理基盤は，複数のサーバーを用いてデータ処理を並列に行うことで，多くのデータを高速に処理することを可能とする．この並列分散処理基盤では，既に蓄積されたデータを各マシンに分散させてから並列処理を行う．しかし，ストリームデータは既に蓄積されたデータではなく，際限なく到来するデータであるため，このような並列分散処理基盤では処理することができない．この問題を解決するために，大規模なストリームデータをリアルタイムに逐次処理するための技術が数多く開発されている [1, 8, 11, 18]．また，ストリームデータのリアルタイム解析の 1 つに，文書フィルタリングがある．本研究ではリアルタイムでの文書フィルタリングについて扱う．

文書フィルタリングとは，ニュース記事や Twitter などのストリームデータから，ユーザーが要求した情報を含む文書を，リアルタイムかつ自動的に抽出する技術である．ユーザーの要求であるプロファイルをシステムに入力すると，システムはプロファイルに関連した文書を抽出する．しかし，プロファイルにユーザーの要求が過不足なく含まれていないことが多く，これはフィルタリン

[1] ブログサービスの一種で，投稿内容が短いテキストであるのが特徴である．

グの精度に直接影響する．そこで，プロファイルを修正することにより，フィルタリング精度を向上させる手法が存在する．その 1 つに，適合性フィードバック [13] と呼ばれる手法がある．初期のプロファイルだけで過不足なく必要な文書だけを抽出することは難しいため，抽出された文書が必要な文書かどうかをユーザーが判断し，その結果を基にフィードバックを行い，プロファイルの修正を行うことで，フィルタリングの精度を高める手法である．

近年，この文書フィルタリングに関する研究が盛んである．TREC(Text REtrieval Conference) [2] KBA(Knowledge Base Acceleration) Task [6] がその一つである．TREC KBA Task とは，ニュース記事などの大規模ストリームデータから，指定されたエンティティ [3] についての情報を抽出する精度を競う文書フィルタリングのタスクである．TREC KBA Task では，Wikipedia に代表される，あるエンティティについての情報や関係といった知識を集積したものである「知識ベース」の問題に焦点を当てている．通常,「知識ベース」は人の手によって，新しい情報を収集し，編集することでページの内容を最新の情報に保っている．しかし，編集者の数に対し，エンティティ数の方が圧倒的に多いため，編集が遅れてしまい，ページの内容を常に最新の状態に保つことができないという問題がある [6]．そのため，TREC KBA Task では，ニュース記事などのストリームデータをフィルタリングし，自動的に新しい情報を高精度に抽出するための手法の開発を試みている．

本研究では，そのような手法として，適合性フィードバックを改良した手法に，並列分散ストリームデータ処理基盤を組み合わせた手法を提案する．並列分散ストリームデータ処理基盤を用いることで大規模ストリームデータのリアルタイム処理に対応し，また，適合性フィードバックを並列分散ストリームデータ処理基盤上に構築することで，ストリームデータのリアルタイムフィルタリングに適用する．評価実験では，適合性フィードバックを行わない場合と精度を比較することで，提案システムにより文書フィルタリングの精度が向上していることを示す．また，1 秒間に処理できる文書数を測定し，大規模ストリーム

[2]情報検索関連の研究分野に着目し開催されているワークショップ．主催はアメリカ国立標準技術研究所（NIST） とアメリカ国防総省内の研究部門の 1 つである ARDA.

[3]ここでは，Wikipedia のある 1 ページ，または Twitter のアカウントのホームページを指す.

データをリアルタイムに処理できていることを示す.

本論文の構成は以下の通りである．まず，2 章で並列分散処理と文書フィルタリングの概要を述べ，フィルタリングに用いられている一般的な手法を説明する．3 章では，並列分散ストリームデータ処理基盤 S4 を用いた Incremental Rocchio によるフィルタリングシステムを提案する．4 章では，提案システムの評価実験を行い，その結果を考察する．最後に，5 章で本論文のまとめと今後の課題を述べる.

第2章　関連研究

2.1　ストリームデータの並列分散処理

2.1.1　並列分散処理

今日，HDD の大容量化や，ネットワーク回線速度の飛躍的向上により，ビッグデータ [17] を処理する機会というのが増えている．ビッグデータとは，従来のデータ処理アプリケーションで処理することが困難なほど巨大で複雑なデータ集合の集積物である．そのため，ビッグデータは，一般的に何らかの並列分散処理フレームワークを用いて処理されることが多い．並列分散処理フレームワークとしては，Hadoop [4] が有名である．

Hadoop は Apache のトップレベルプロジェクトの 1 つであり，世界規模の開発貢献者コミュニティによって開発され，Yahoo!や楽天などの大手ポータルサイトなどにも使用されている．Hadoop は，HDFS と呼ばれる独自のファイルシステムと，MapReduce [5] と呼ばれる処理方法により，複数のマシンに大量データ処理を分散して飛躍的に性能を向上させることを容易に可能としている．しかし，Hadoop の弱点として，ビッグデータをいったんファイルシステムに蓄積し，バッチで一括処理する形態で処理を行うため，処理データが発生してから，それに対する処理結果が得られるまでに必ずタイムラグが発生してしまうことが挙げられる．このため，クレジットカードの不正アクセス検知，センサデータなどでの異常値検出のようなリアルタイムなレスポンス（低レイテンシ）が要求されるビッグデータ分野への Hadoop の適用は向いていないとされている．

本研究でも，ビッグデータの並列分散処理を行う．しかし，そのデータは同時にストリームデータでもある．ストリームデータとは，ニュース・Tweet・株価情報・交通情報のように，際限なく到来する時刻順の大量データを指す．ストリー

ムデータをリアルタイムに処理する本研究では，上記の理由により，Hadoop の利用は適さない．このため，本研究では次節に述べるような並列分散ストリームデータ処理基盤を用いる．

2.1.2 ストリームデータの並列分散処理

並列分散ストリームデータ処理基盤には様々なものがあり，代表的なものが，Storm [1] と S4[4] [11] である．Storm は，Nathan Marz 氏が開発したオープンソースのビッグデータリアルタイム分散処理システムである．主な使用例として，Twitter で Tweet の Firehose[5] から新しいトレンドを抽出するシステム（Fig. 2.1）に用いられている．一方，S4 は，米 Yahoo!で開発された分散環境でのリアルタ

Fig. 2.1: Trends of Twitter.

イムなストリーミングデータの処理エンジンで，現在は Apache Foundation で開発されている．主な使用例として，Yahoo!で検索クエリから最適な広告を配

[4]Simple Scalable Streaming System の略.

[5]Twitter の Streaming API の一つで，全公開ツイートをリアルタイムに取得することができるメソッド.

信するためのシステムに利用されている．本研究では，S4 を用いることにした．次節で S4 のより詳細なアーキテクチャについて述べる．

2.1.3 S4

S4 とは，米 Yahoo!で開発された分散環境でのリアルタイムなストリーミングデータの処理エンジンで，現在は Apache Foundation で開発されている．S4 は，入力されたストリーミングデータの 1 件 1 件を「イベント」という形で取り扱い，ストリームで繋がった「Processing Element（PE）」という計算ユニット（ユーザーアプリケーション）で順次処理を行う（Fig. 2.2）．アーキテクチャはいわゆるアクターモデルである．アクターモデルとは，1973 年に Carl Hewitt,

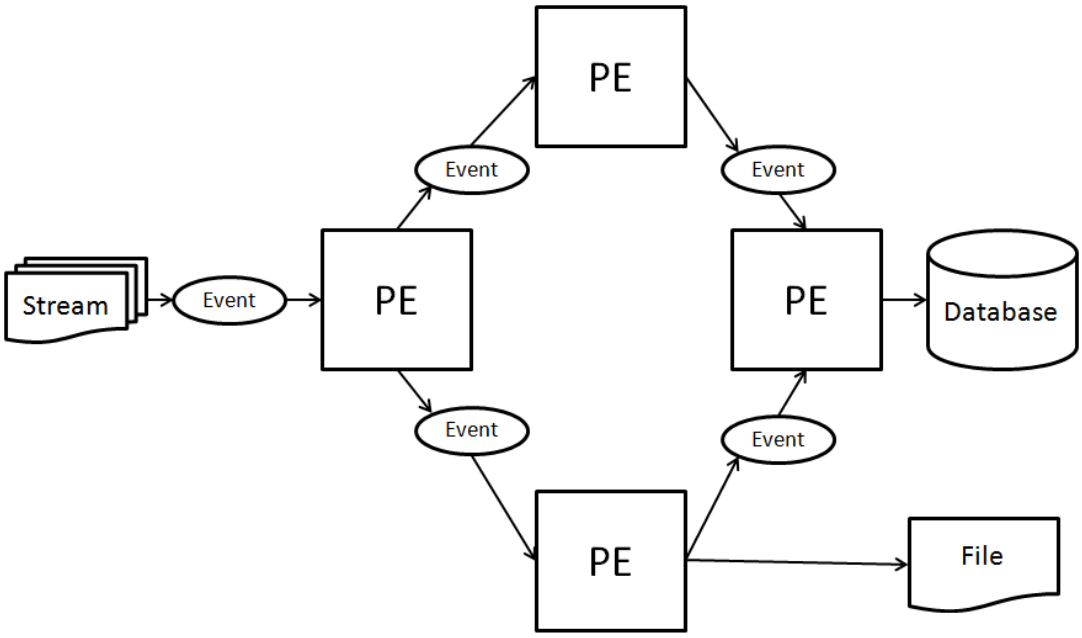


Fig. 2.2: The architecture of S4.

Peter Bishop，and Richard Steiger が 発表した並行計算の数学的モデルの一種である [7]．アクターとは，並行的に受信するメッセージに対応して,「他のアクターに有限個のメッセージを送信する」,「有限個の新たなアクターを生成する」

「次に受信するメッセージに対応する動作を指定する」という振る舞いを備えた計算実体（Computational Entity）である．これらの振る舞いは並列的に，非同期に実行される．アクターモデルでは，全てのものをこのアクターとみなすのである．

イベントの実体はJavaのオブジェクトであり，指定した型の要素を持たせることができる．また，イベントにはkeyが割り当てられており，このkeyと同じkeyを持つPEで処理される．PEはイベントのkey毎に生成され，keyごとに並行して処理を行う．イベントを処理したPEは，再びイベントを次のPEにストリームに送出したり，Databaseなどの外部システムに出力することができる．また，PEは各ノードに分散して配置される．ノード数については制限がなく，クラスタにノードを追加することでスループットがリニアに向上するとされている．

このようにして，ストリーミングデータのリアルタイム並列分散処理を可能としている．

2.2 文書フィルタリング

2.2.1 tfidf法

文書をフィルタリングする際によく用いられる手法として，tfidf法がある．tfidf法は，情報検索の分野で広く用いられているアルゴリズムの一つである．単語に重みをつけ，その重みを基に検索クエリから文書をベクトル空間で表し，文書と検索クエリのコサイン類似度を求める．求めた値が高いほど，その文書は検索クエリとの関連度が大きいとされる．tfidfは次式で現される．

$$tfidf = tf \cdot idf \tag{2.1}$$

$$tf(q, D) = \frac{n(q, D)}{|D|} \tag{2.2}$$

$$idf(q) = log\frac{N}{df(q)} \tag{2.3}$$

ここで，$n(q, D)$ は文書 D 中に含まれる単語 q の数，$|D|$ は文書 D の単語数（文書長），N は全文書数．$df(q)$ は単語 q を含む文書の数である．式 (2.2) で表される tf（Term Frequency）とは，ある文書中の単語の出現頻度を表す指標であり，文書中での出現頻度が高い単語ほど値が大きくなる．これは文書中で出現頻度が高い単語ほど，その文書において重要であると考えるためである．一方，式 (2.3) で表される idf（Inverse Document Frequency）は，逆文書頻度とも呼ばれ，複数の文書に共通して出現する単語ほど値が低くなる．そのため，idf は一種の一般語フィルタとして働き，多くの文書に出現する単語（一般的な単語）の重要度を下げ，特定の文書にしか出現しない単語の重要度を上げる役割を果たす．そして，tf と idf を積算したものが tfidf となる．単語重要度に関する tf 法と idf 法という 2 つの異なる観点を組み合わせることで，単語重要度をバランスよく評価することが可能となる．

tfidf 法は，文書中の単語数に依存して tf の値が大きく変動するという性質がある．通常，同じような単語数の文書の重要度を比較する場合には，この変動は問題にならない．しかし，大小様々な多くの文書を絶対評価的にスコアリングするような場合は，特徴語の重要度にムラができてしまう．そのような tfidf 法の欠点を補った評価指標として，Okapi BM25 がある．本研究では，この Okapi BM25 を用いて文書とクエリの関連度の評価を行う．次節でその詳細について述べる．

2.2.2 Okapi BM25

Okapi BM25 [12] は情報検索の分野で用いられている，tfidf 法よりも精度が高いとされる評価指標である．次式によって表される．

$$score(D, Q) = \sum_{q \in Q} BM25(D, q) \tag{2.4}$$

$$BM25(D, q) = idf(q) \frac{tf(q, D) \cdot (k + 1)}{tf(q, D) + k(1 - b + b\frac{|D|}{avgdl})} \tag{2.5}$$

$$idf(q) = log \frac{N}{N(q)} \tag{2.6}$$

$$tf(q, D) = \frac{n(q, D)}{|D|} \tag{2.7}$$

ここで，q は検索クエリ中に含まれる単語の 1 つ，$|D|$ は文書 D の単語数（文書長），$avgdl$ は全文書の平均文書長，N は全文書数，$N(q)$ は単語 q を含む文書の数，$n(q, D)$ は文書 D 中に含まれる単語 q の数である．また，k，b は任意のパラメータであり，一般的に $k = 2.0$，$b = 0.75$ とするのが精度がよいとされている．本研究でもこの値を用いる．Okapi BM25 は，検索クエリ Q に対する文書 D の関連度 $score(D, Q)$ を評価する指標である．BM25 は，前節で述べた tfidf 法の欠点を補うために，全文書の平均文書長 $avgdl$ を用いて，tf の値が文書中の単語数によって大きく変動するのをある程度抑えている．

2.2.3 適合性フィードバック

前節までは，検索クエリと文書との関連度を評価する手法について紹介した．しかし，これはあくまで検索クエリの質が高い場合に有効であり，一般的には初期に設定した検索クエリのみでは，本当に関連していると言える文書を抽出することが困難である．これは，自分が Google や Yahoo!などでウェブ検索する時を考えれば，容易に理解できる．初期の検索クエリでは，必要とする情報が得られず，クエリに何らかの単語を追加することで，更に絞り込んで，ようやく必要とする情報に辿り着くことができる．本研究で扱う Wikipedia のページに関する情報を抽出するために文書フィルタリングを行う場合も同様に，プロファイルに含まれる情報はエンティティのみとなるので，非常に質が低い．そのため，何らかの方法を用いてプロファイルを修正する必要がある．そのような手法として，適合性フィードバックがある．

適合性フィードバック [13] とは，検索結果に対し，ユーザーがその適合・不適合の判断を行い，その情報をシステムにフィードバックする手法である．ベクトル空間モデルの情報検索において，適合文書や不適合文書の特徴ベクトルをもとに，クエリベクトルを再定式化することによりフィードバックを行う．Rocchio は次式を用いて，クエリベクトルの修正を行っている．

$$\boldsymbol{q}(t+1) = \alpha \boldsymbol{q}(t) + \frac{\beta}{|D_r|} \sum_{\boldsymbol{d} \in D_r} \boldsymbol{d} - \frac{\gamma}{|D_{nr}|} \sum_{\boldsymbol{d} \in D_{nr}} \boldsymbol{d} \tag{2.8}$$

ここで，$\boldsymbol{q}(t)$ は時刻 t におけるクエリベクトル，$\boldsymbol{d}$ は文書の特徴ベクトル，D_r と D_{nr} はそれぞれユーザが選択した適合・不適合文書の集合である．また α，β，γ は前回のクエリベクトル，正のフィードバック，負のフィードバックの重みを定める係数である．式 (2.8) で定義される Rocchio の適合性フィードバックは，Salton と Buckley によって，ベクトル空間モデルの情報検索において非常に有効であることが示されている [15]．また，Rui と Huang は画像検索において画像特徴量に関して適合性フィードバックを行うことで検索精度が向上することを示している [14]．

また，適合性フィードバックを行う上で，式 (2.8) の第 3 項で表される負のフィードバックの扱いは非常に重要となる．負のフィードバックの重みをつけすぎる，すなわち γ の値を大きくし過ぎると，クエリベクトル中で多くの値が負の値をとってしまい，類似度の計算などに大きな影響を及ぼしてしまう．Muller らや Wing らは適合性フィードバックを行う上で，負のフィードバックを効果的に行う手法に関する研究を行っている [10, 16]．

本研究では，Rocchio の適合性フィードバックを応用した手法として，Incremental Rocchio と呼ばれる手法を用いる．

2.2.4 Incremental Rocchio

Incremental Rocchio とは，Allan によって提案された適合性フィードバックを用いた文書のフィルタリング手法である [3]．ユーザーからの検索クエリに応じてシステムによって抽出された文書が，本当に適当と言えるものかどうかをユーザー自身が判定し，その結果をもとにフィードバックを行う．Albakour らは，Incremental Rocchio を Twitter のフィルタリングに応用した手法を提案し，実際に精度が向上することを示している [2]．

Incremental Rocchio のフローチャートは Fig. 2.3 の通りである． まず，新たな文書 $\boldsymbol{d}$ が到着すると，ユーザーによって指定された検索クエリに適合するかどうかをシステムが判断する．適合しない場合はそこで終了である．適合する場合は，それをユーザーに提示し，本当にユーザーが求めていた文書であるかどうかを，ユーザー自身が判定する．その結果を用いて，適合文書セット R_t と

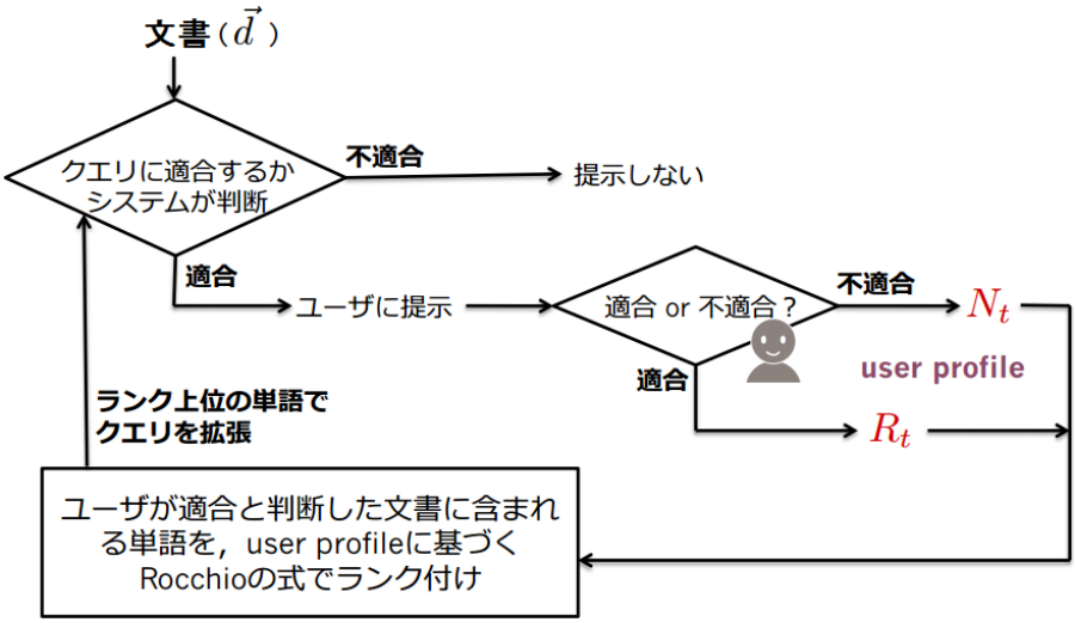


Fig. 2.3: Incremental Rocchio flowchart.

不適合文書セット N_t からなる user profile を作成する．そして，ユーザーが適合と判断した文書に含まれる全ての単語を，登場回数の多い順にランク付けする．その上位 500 件の単語を，次式の user profile にもとづく単語 t についての Rocchio の式でランク付けを行う．

$$Rocchio(t) = \omega_{query}(t) + 2\omega_{rel}(t) - \frac{1}{2}\omega_{non-rel}(t) \tag{2.9}$$

$$\omega_{query}(t) = \omega^{old}_{query}(t) \tag{2.10}$$

$$\omega_{rel}(t) = \frac{1}{|relset|}\sum_{d \in relset} bel_{t,d} \tag{2.11}$$

$$bel_{t,d} = 0.4 + 0.6 \cdot tfbel_{t,d} \cdot idf_t \tag{2.12}$$

$$tfbel_{t,d} = \frac{tf_{t,d}}{tf_{t,d} + 0.5 + 1.5\frac{len_d}{avgdoclen}} \tag{2.13}$$

$$idf_t = \frac{log(\frac{N+0.5}{docf_t})}{log(N+1)} \tag{2.14}$$

ここで，$\omega_{query}^{old}(t)$ は前回の Incremental Rocchio 時の式 (2.9) の値（値が存在しなければ 0 とする），$|relset|$ は適合文書セット内の文書数，$tf_{t,d}$ は文書 d 中に含まれる単語 t の数，len_d は文書 d の長さ，$avgdoclen$ はシステムによって関連であると判定された文書のセットに含まれる文書の単語数の平均，N はシステムによって関連であると判定された文書の数である．最終的に，そのランクの上位 100 件の単語でクエリを拡張する．これが，Incremental Rocchio の 1 サイクルの流れとなり，これを繰り返すことで，徐々に検索クエリが高精度なものとなっていく．

本研究では，この手法により初期のエンティティのみで構成されたプロファイルを拡張し，より高精度なフィルタリングを可能とする．

第3章　提案手法

3.1　S4を用いた提案システム

Fig.3.1はS4を用いた提案システムのフローチャートである．Adapterは，スト

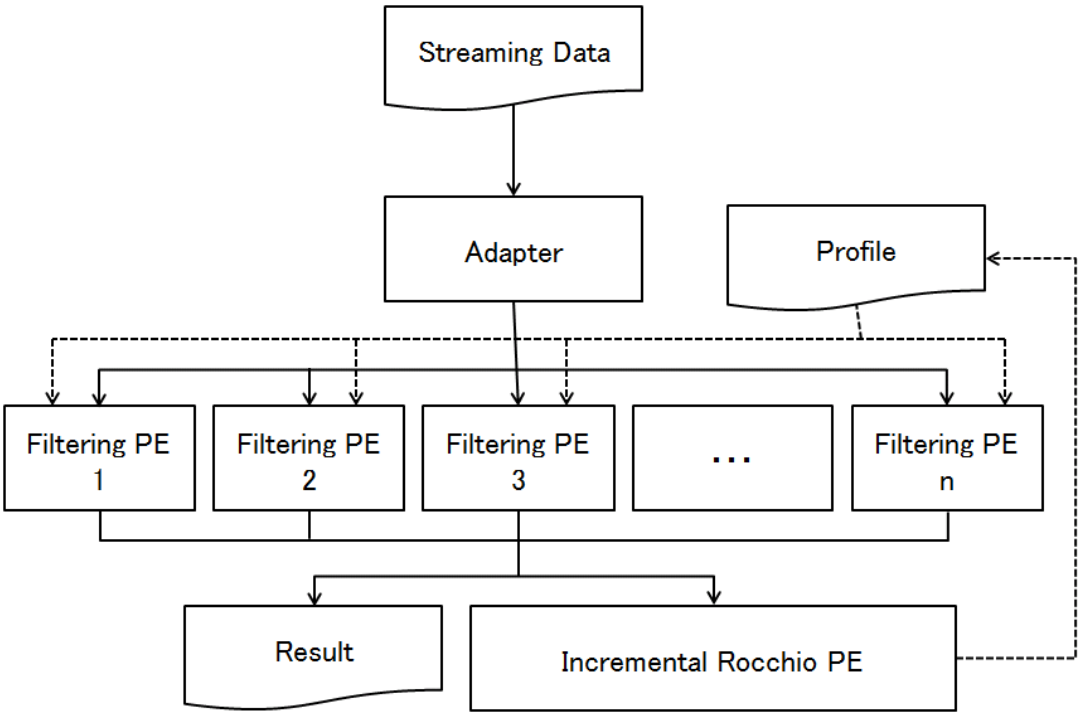


Fig. 3.1: The S4 system of proposal technique.

リームから文書を受信して，提案システムで処理できるようなフォーマットに変換した後，実際の処理を行うFiltering PEにイベントとして送出する．Filtering PEは，実際に文書のフィルタリングを行うPEであり，並列分散処理が行われ

るのもこの部分である．Adapter から送られてきた文書がプロファイルに関連するかを判定する．関連すると判定された文書はフィルタリング結果として出力される．また，同時に Incremental Rocchio PE に送出される．Incremental Rocchio PE は，Filtering PE から送られてきた，システムによって関連すると判定された文書をユーザーに提示し，本当に適合するかどうかを判定してもらう．その結果からユーザープロファイルを作成し，Rocchio の式を用いてプロファイルの拡張を行う．このようにして，絶え間なく送られてくるストリームデータのフィルタリングを行う．

3.2 Okapi BM25 の適用

Okapi BM25 は通常，検索結果の文書セット内の文書 1 件 1 件に対してプロファイルと文書の類似度スコアを算出することでランク付けを行うものである．しかし，今回は絶え間なく送られてくるストリームデータのフィルタリングに Okapi BM25 を用いるため，ランク付けはできない．そこで，本手法では，算出された類似度スコアがある一定値以上であれば，プロファイルに関連した文書として判定する．その判定には以下の閾値を用いる．類似度スコアが閾値以上の場合，プロファイルに関連した文書として判定する．

$$\alpha \times n(D, Q) \tag{3.1}$$

ここで，$n(D, Q)$ は文書 D 中に存在したプロファイル Q 中の単語の数である．Okapi BM25 の類似度スコアは，プロファイルベクトルと文書ベクトルの内積によって求められる．そのため，上記の式のように，マッチした単語の数に応じて閾値が変化するようにした．また，α は任意の実数であり，これを増減させることで閾値を変化させられる．この値には，予備実験で精度の高かった 0.0015 を用いる．また，式 (2.5) の平均文書長 $avgdl$ は，検索対象となる文書集合中に含まれる全文書の平均文書長である．しかし，今回扱う検索対象はストリームデータであるため，予め定められた文書集合は存在しない．そのため，本手法では $avgdl$ の値として 1,000 を用いることとする．さらに，Okapi BM25 の計算

に用いる idf は，予め Google N-gram と呼ばれるデータセットから算出したものを用いることとした．

3.3 フィルタリング

初期に用いるプロファイルは，エンティティの本名[6]を単語に分割したエイリアスを用いる[7]．つまり，エンティティが「Clark_Blaise」の場合，エイリアスは「Clark」「Blaise」の 2 つになる．エイリアスを全て含む文書のみをフィルタリングの対象として扱う．フィルタリングの対象となった文書は，BM25 による判定を行い，最終的にプロファイルに関連した文書かどうかを判定する．しかし，この方法では，Table. 3.1 に示すように，予備実験の際に，一部のエンティティで関連すると判定された文書（抽出数）が非常に多くなってしまった．なお，巻末の付録 A.1 の Table.A.1 に Entity とその ID との対応表を記しておいた．また，Table 3.1 の項目について，NE はシステムが抽出した文書の数（Number of Extraction documents）を，NTE はシステムが抽出した正解文書の数（Number of Extraction Truth documents）を，NT はコーパスに含まれる正解文書の数（Number of Truth documents）を表す．

Table 3.1: Part of result of preliminary experiment.

Entity	NE	NTE	NT	Precision	Recall	F
14	4295	36	37	0.008	0.972	0.016
26	4661	14	20	0.003	0.700	0.005
27	12580	19	19	0.002	1.00	0.003

[6]エンティティは Wikipedia か Twitter のページ URL の形で提供されているため，そのページに表示されるページ名を本名とする．

[7]英語圏では，本名の短縮形がよく用いられる．しかし，日本とは違い，短縮形も本名と同様に扱われることがあるため，必要に応じて短縮形をエイリアスとして扱う．例えば，正式名は「Ronald」であっても，短縮形の「Ron」を本名として使用しているのであれば，そちらをエイリアスとして扱う

このようなエンティティは，エイリアスに含まれる単語の idf がいずれも低いことが共通の特徴であった．そこで，このようなエンティティのフィルタリングを行う際には次の 2 つの処理を，フィルタリングの前に追加することとした．まず，文書中にエイリアスが，エンティティ名の順番に連続して存在するかをチェックする．つまり，エンティティが「Appleton_Museum_of_Art」の場合は，「Appleton」「Museum」「of」「Art」という各単語が文書中に点在するのではなく，「Appleton Museum of Art」を一つの単語と考えて，これが存在するかをチェックする．存在しない場合はフィルタリングの対象外とする．次に，BM25 での判定の際に使用する閾値を増加させる．Table.3.1 に示した 3 つのエンティティについて，増加させる割合を変化させたところ，Fig.3.2 に示すように 1.5 倍が最も精度がよくなったので，これを使用する．

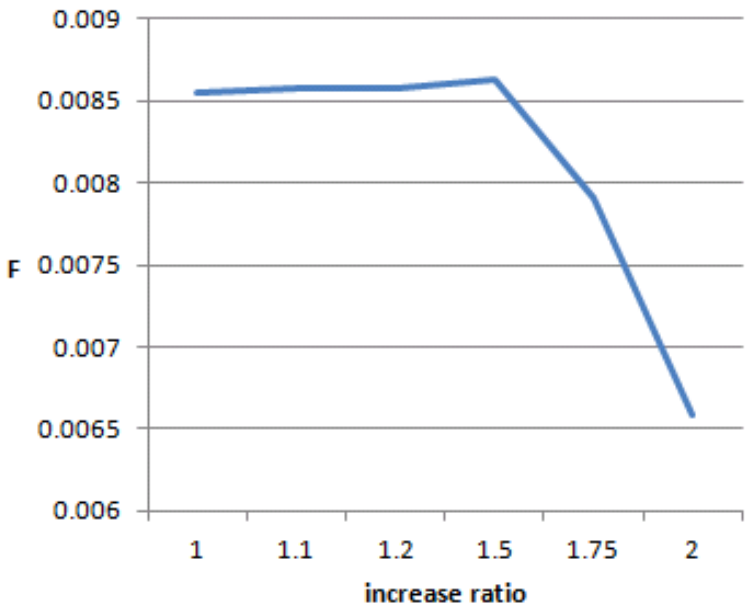


Fig. 3.2: Increase ratio of threshold.

3.4 Incremental Rocchioの適用

本手法は絶え間なく送られてくるストリームデータを対象とするため，フィードバックの結果を待っているとストリームデータの流れについていくことができなくなる．そこで，フィルタリングとIncremental Rocchioを別のプロセスで非同期で行うこととした．user profileをテキストファイルとして蓄積していき，別のプロセスでそのテキストファイルを読み込んでIncremental Rocchioによるフィードバックを行う．

第4章　評価実験

4.1　実験データ

本実験では，コーパスとして，TREC KBA task で使用された，TREC KBA Stream Corpus 2013[8] を用いた．これは，ニュース記事やウェブログなど，あらゆるジャンルの文書を含む 6TB 以上の文書データ（1,040,520,595 件）で，2011 年 10 月 5 日〜2013 年 2 月 13 日に収集されたものである．その文書が投稿された日時のデータも付与されているので，これを用いてソートを行うことで，プログラムで時系列順にデータを読み込むことが可能となる．本実験では，このようにして擬似的にストリームを作り出した．また，予め 170 個のエンティティが指定されており，それに対応する正解データも存在する．本実験ではこの正解データを用いて精度の評価を行った．

本実験では，TREC KBA Stream Corpus 2013 の一部を使用した．ニュース記事のみを扱い，期間は 2011 年 11 月 1 日〜2012 年 2 月 29 日の 4ヶ月間（31,620,376 件の文書）とした．

4.2　Incremental Rocchio による精度向上の評価

4.2.1　実験設定

本実験では，Incremental Rocchio を本手法に導入することによりフィルタリング精度が向上することを検証した．評価指標には次式で表される F 値を用い，

[8] http://s3.amazonaws.com/aws-publicdatasets/trec/kba/index.html から入手可能．

Incremental Rocchio を実行する場合としない場合の F 値を比較した.

$$F_{measure} = \frac{2 \cdot Precisiton \cdot Recall}{Precisiton + Recall} \tag{4.1}$$

$$Precison = \frac{\text{システムが抽出した正解文書の数}}{\text{システムが抽出した文書の数}} \tag{4.2}$$

$$Recall = \frac{\text{システムが抽出した正解文書の数}}{\text{コーパスに含まれる正解文書の数}} \tag{4.3}$$

F 値は Precision と Recall の調和平均で表され，Precision と Recall の値を両方考慮したスコアとなっている．また，Incremental Rocchio を実行しない場合は，Fig.3.1 における Incremental Rocchio PE の処理を実行しない.

Incremental Rocchio において，ユーザーが文書の適合・不適合を判定する部分に関しては，今回は正解データに含まれる文書を適合として扱い，user profile を作成する．また，エンティティは 30 個に限定して用いた.

4.2.2 実験結果

実験結果を Table.4.1 及び Table.4.2 に示す．なお，巻末の付録 A.1 の Table.A.1 に Entity とその ID との対応表を記しておいた.

Table 4.1: Performance of filtering for run or not run Incremental Rocchio.

	Incremental Rochhio			no Incremental Rochhio		
Entity	Precision	Recall	F	Precision	Recall	F
01	0.849	0.983	0.911	0.849	0.983	0.911
02	0.709	0.754	0.731	0.774	0.762	0.768
03	0.282	1.000	0.441	0.292	1.000	0.452
04	0.720	1.000	0.838	0.736	1.000	0.848
05	0.842	0.889	0.865	0.842	0.889	0.865
06	0.656	0.894	0.757	0.633	0.809	0.710
07	0.146	1.000	0.255	0.425	0.978	0.592

Table 4.2: Performance of filtering for run or not run Incremental Rocchio (cont).

	Incremental Rochhio			no Incremental Rochhio		
Entity	Precision	Recall	F	Precision	Recall	F
08	0.142	0.867	0.245	0.146	0.867	0.249
09	0.405	1.000	0.577	0.372	0.681	0.481
10	0.158	0.762	0.261	0.155	0.714	0.255
11	0.122	0.659	0.206	0.124	0.659	0.209
12	0.977	1.000	0.989	0.977	1.000	0.989
13	0.474	0.973	0.637	0.240	0.324	0.276
14	0.647	0.595	0.620	0.720	0.973	0.828
15	0.375	1.000	0.545	0.359	0.917	0.516
16	0.326	1.000	0.492	0.326	1.000	0.492
17	0.727	0.500	0.593	0.800	1.000	0.889
18	0.183	0.778	0.296	0.183	0.778	0.296
19	0.448	0.963	0.612	0.453	0.889	0.600
20	0.615	0.320	0.421	0.600	0.240	0.343
21	0.017	0.792	0.033	0.017	0.792	0.034
22	0.958	1.000	0.979	0.958	1.000	0.979
23	0.840	0.955	0.894	0.840	0.955	0.894
24	0.438	0.955	0.600	0.438	0.955	0.600
25	0.417	1.000	0.588	0.417	1.000	0.588
26	0.933	0.700	0.800	0.933	0.700	0.800
27	0.094	1.000	0.172	0.095	1.000	0.174
28	0.007	0.944	0.014	0.008	0.944	0.015
29	0.037	0.941	0.071	0.037	0.941	0.072
30	0.198	1.000	0.330	0.195	1.000	0.327
Average	0.458	0.874	0.601	0.465	0.858	0.603

Incremental Rocchio ありの方が，8 つのエンティティにおいて F 値が向上し，12 つのエンティティにおいて低下，10 つのエンティティでは変化がなかった．また，平均的に Incremental Rocchio ありの方が Recall は向上した一方で，Precision は低下した．しかし，いずれの値も一部のエンティティを除き，その変化は小さなものであった．

4.2.3 考察

Incremental Rocchio ありの方が，8 つのエンティティにおいて F 値が向上し，12 つのエンティティにおいて低下，10 つのエンティティでは変化がなかった．この結果から，提案手法では，ある限定された状況でのみ Incremental Rocchio が有効であることを示唆している．本節では，この結果が生じた原因について考察を行う．

12 つのエンティティにおいて F 値が低下した原因として，閾値が不適切であったことが考えられる．実際，Okapi BM25 の値は式 (2.5) より，その単語の idf によって大きく左右される．つまり，プロファイルを構成する単語の idf が小さいと，BM25 の score は絶対的に低いものとなってしまう．一方，閾値は式 (3.1) より，プロファイル中のマッチした単語の数によって変化するだけで，単語の idf による変化はない．したがって，プロファイル毎に異なる閾値を設定すれば，精度が向上する可能性がある．そこで，次のような追加実験を行い，この仮説について検証する．Incremental Rocchio ありの方が F 値が低かったエンティティのうち，特に顕著であった 6 つのエンティティについて，閾値を変化させることによる F 値の変化を測定する．それぞれの閾値について，Incremental Rocchio を実行する場合と，しない場合の 2 パターン行う．その他の実験設定は 4.2.1 と同様である．

実験結果を Fig. 4.1 に示す．Table.4.1 及び Table.4.2 に示した実験での α の値は 0.0015 であったので，いずれのエンティティでも，独自に閾値を定めることで F 値が向上した．また，6 つ中 4 つのエンティティで Incremental Rocchio ありの方が F 値が向上した．

同様にして，他のエンティティについても閾値を定め，再実験を行った結果

を Table. 4.3 及び Table. 4.4 に示す．Incremental Rocchio ありの方が，18 つのエンティティにおいて F 値が向上し，6 つのエンティティにおいて低下，6 つのエンティティでは変化がなかった．平均してフィルタリングの精度が F 値にして 1.5 ポイント向上した（有意水準 5 %で有意）．よって，適切な閾値をエンティティごとに設定すれば，Incremental Roccio による文書フィルタリングは，Incremental Rocchio を行わない文書フィルタリングよりも精度を向上させることができると言える．言い換えれば，大規模ストリームデータのリアルタイムフィルタリングに対して Incremental Rocchio を適用するには，閾値の設定が非常に重要である．

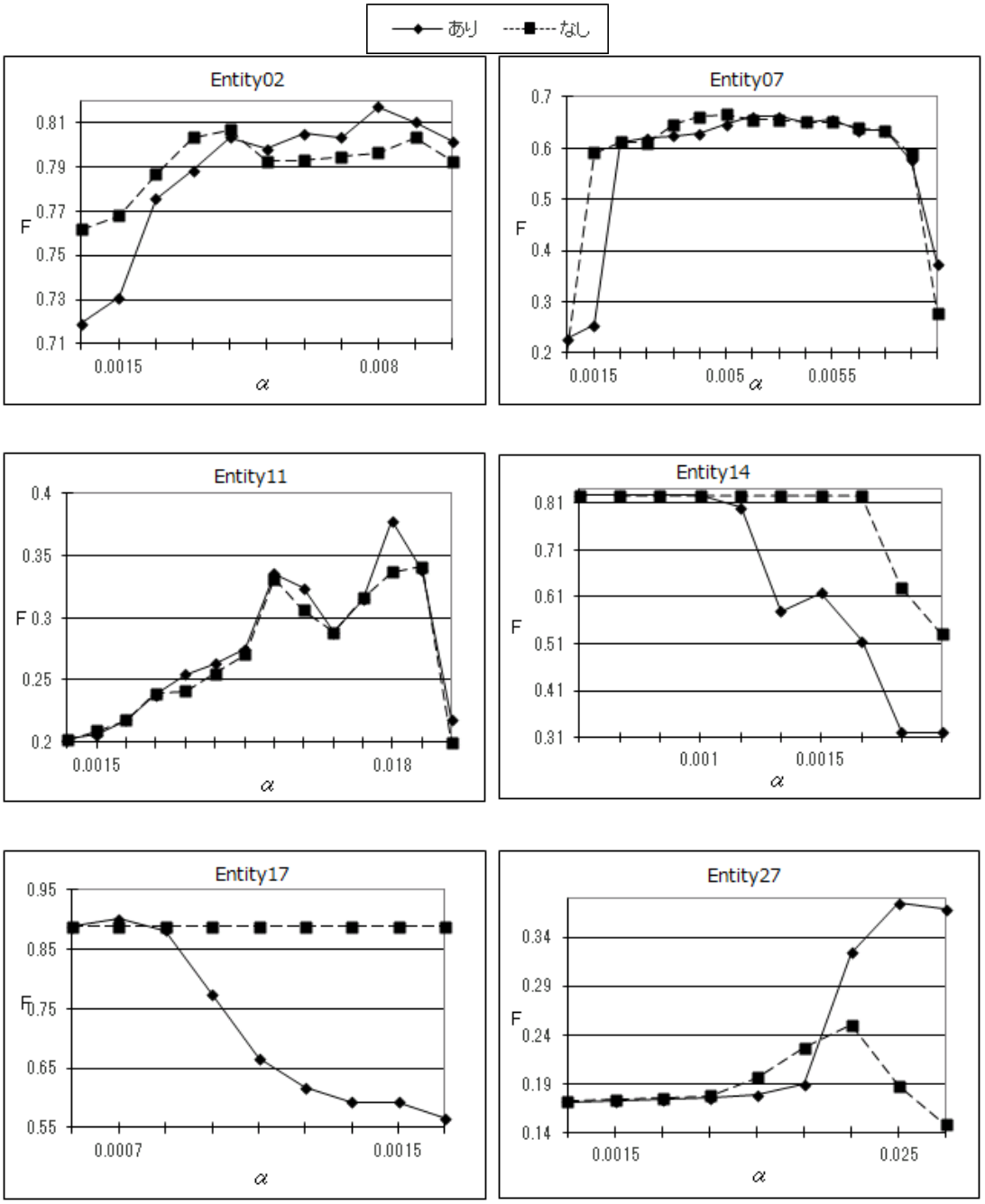


Fig. 4.1: Performance of filtering for run or not run Incremental Rocchio and for α.

Table 4.3: Performance of filtering after the α correction.

	Incremental Rochhio			no Incremental Rochhio		
Entity	Precision	Recall	F	Precision	Recall	F
01	0.915	0.936	0.925	0.954	0.971	0.963
02	0.904	0.746	0.817	0.857	0.762	0.807
03	0.303	1.000	0.465	0.304	0.966	0.463
04	0.720	1.000	0.838	0.736	1.000	0.848
05	0.855	0.981	0.914	0.855	0.981	0.914
06	0.671	1.000	0.803	0.671	1.000	0.803
07	0.512	0.913	0.656	0.512	0.957	0.667
08	0.323	0.711	0.444	0.376	0.778	0.507
09	0.439	1.000	0.610	0.376	0.681	0.485
10	0.194	0.643	0.298	0.165	0.810	0.274
11	0.424	0.341	0.378	0.341	0.341	0.341
12	0.977	1.000	0.989	0.977	1.000	0.989
13	0.480	0.973	0.643	0.456	0.973	0.621
14	0.720	0.973	0.828	0.720	0.973	0.828
15	0.379	1.000	0.550	0.359	0.917	0.516
16	0.326	1.000	0.492	0.326	1.000	0.492
17	0.727	0.500	0.593	0.800	1.000	0.889
18	0.183	0.778	0.296	0.183	0.778	0.296
19	0.553	0.778	0.646	0.680	0.630	0.654
20	0.667	0.320	0.432	0.667	0.320	0.432
21	0.017	0.792	0.033	0.017	0.792	0.034
22	0.958	1.000	0.979	0.958	1.000	0.979
23	0.840	0.955	0.894	0.840	0.955	0.894
24	0.571	0.727	0.640	0.457	0.955	0.618
25	0.500	0.950	0.655	0.487	0.950	0.644

Table 4.4: Performance of filtering after the α correction (cont).

	Incremental Rochhio			no Incremental Rochhio		
Entity	Precision	Recall	F	Precision	Recall	F
26	0.933	0.700	0.800	0.933	0.700	0.800
27	0.094	1.000	0.172	0.095	1.000	0.174
28	0.007	0.944	0.014	0.008	0.944	0.015
29	0.037	0.941	0.071	0.037	0.941	0.072
30	0.234	0.882	0.370	0.200	1.000	0.333
Average	0.561	0.782	0.618	0.536	0.797	0.603

4.3 大規模ストリーミングデータへの適応評価

4.3.1 実験設定

この実験は，提案システムにより大規模ストリームデータに対しても，リアルタイムでの文書フィルタリングが可能となっているかを評価することを目的とする．1 秒あたりに処理できる平均文書数を測定し，その値をストリームで 1 秒あたりに生成される文書数と比較することで，ストリームデータの生成に処理が追いついているかを検証した．平均文書数は,「ストリームから送られてきた文書数÷処理にかかった総時間」で算出するものとし，ストリームで 1 秒あたりに生成される文書数として，Twitter のデータ [9] を用いた．また，サーバー数を 1 台（並列化なし）の場合と，8 台（並列化あり）の場合で測定した.

4.3.2 実験結果

実験結果を Table. 4.5 に示す．並列化なしの場合では Twitter での 1 秒あたりに投稿されるツイート数を上回れなかったが，並列化することにより，上回ることができた.

Table 4.5: Number of processing documents per second for number of servers.

Server	Number
1	3031.014
8	6613.758
Twitter	4500.000

4.3.3 考察

並列化を行うことで，Twitter で 1 秒あたりに投稿されるツイート数を上回ることができたので，大規模なストリームデータに対しても，十分リアルタイム処理が可能であると言える．しかし，サーバー数を 8 倍にしたにも関わらず，処理できる文書数は 2 倍ほどしか増加しなかった．これは，ディスク IO がボトルネックとなっているためだと考えられる．本実験では，ディスク上に存在する大量の文書データを時系列順に読み込むことで，擬似的にストリームを生成していた．そのため，読み込みの度に必ずディスク IO が発生してしまったと考えられる．実際の環境では，Twitter などのストリームから直接文書を読み込むので，このようなディスク IO は発生しない．すなわち，実際の環境で本システムを運用すれば，さらに 1 秒あたりに処理できる文書数はサーバー数に応じてリニアに向上することが期待できる．

第5章　結論

本研究では，ニュース記事などの大規模ストリームデータをリアルタイムにフィルタリングし，自動的に新しい情報を高精度に抽出するための手法として，並列分散ストリームデータ処理基盤 S4 と，既存の高精度なフィードバック手法として知られる Incremental Rocchio を用いた手法を提案した．実際に大規模ストリームデータを用いた評価実験を行い，本システムにより，リアルタイム文書フィルタリングに Incremental Rocchio を適用すればフィルタリング精度を向上させることができることを示した．また，大規模ストリームデータへの適応評価も行い，実際に本システムが大規模ストリームデータに対しても運用可能であることを示した．

本手法において，文書抽出の基準となる閾値を定めることの重要性を論じた．今回の実験では，限られたエンティティ数であったため，筆者自身が閾値の調整を行った．しかし，実際には膨大な数のエンティティが存在するので，これをシステムによって自動的に変化させることで，あらゆるエンティティに対してのフィルタリング精度を向上させることができると考えられる．具体的な方法としては，プロファイルの idf によって変化する閾値を定義すればよい．このような閾値を発見することを，今後の課題とする．

また，文書抽出の基準としては，閾値だけではなく，初期プロファイルにエンティティの名前を単語に分割したエイリアスを用い，それらのエイリアスを全て含まなければ抽出を行わないこととしていた．しかし，抽出すべき文書の中にはそれらのエイリアスを含まない場合もあるため，この基準を用いることは本来好ましくない．フィルタリング精度を低下させることなく，こういった文書も抽出することができるような基準を定める必要があると考えられる．

謝辞

研究生活全般にわたるご支援を賜り，様々なご指導を頂きました神戸大学大学院システム情報学研究科上原邦昭教授に感謝いたします．日頃より丁寧なご指導を賜り，本論文の執筆にあたり多くのご助言を頂きました同研究科関和広准教授に感謝いたします．

本研究を進めるにあたり，研究の進め方など，研究内容について全般的にご指導頂きました博士後期課程３年宮西大樹氏，博士前期課程１年北口沙也加氏に感謝いたします．

参考文献

[1] "Storm," `http://storm.incubator.apache.org/`.

[2] M Albakour, Craig Macdonald, Iadh Ounis, et al., "On sparsity and drift for effective real-time filtering in microblogs," In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pp. 419–428. ACM (2013)

[3] James Allan, "Incremental relevance feedback for information filtering," In *Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 270–278. ACM, (1996)

[4] Dhruba Borthakur. "The hadoop distributed file system: Architecture and design," (2007)

[5] Jeffrey Dean and Sanjay Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, Vol. 51, No. 1, pp. 107–113 (2008)

[6] John R. Frank, Steven J. Bauer, Max KleimanWeine, Daniel A. Roberts, Nilesh Tripuraneni, Ce Zhang, Christopher Re, Ellen M. Voorhees, and Ian Soboroff. "Evaluating stream filtering for entity profile updates for trec 2013," (2013)

[7] Carl Hewitt, Peter Bishop, and Richard Steiger, "A universal modular actor formalism for artificial intelligence," In *Proceedings of the 3rd international*

joint conference on Artificial intelligence, pp. 235–245. Morgan Kaufmann Publishers Inc. (1973)

[8] Shohei Hido, Seiya Tokui, and Satoshi Oda, "Jubatus: An open source platform for distributed online machine learning," *NIPS 2013 Workshop on Big Learning, Lake Tahoe.*

[9] Richard McCreadie, Craig Macdonald, Iadh Ounis, Miles Osborne, and Sasa Petrovic, "Scalable distributed event detection for twitter," In *Big Data, 2013 IEEE International Conference on*, pp. 543–549. IEEE (2013)

[10] Henning Muller, Wolfgang Muller, Stéphane Marchand-Maillet, Thierry Pun, and David McG Squire, "Strategies for positive and negative relevance feedback in image retrieval," In *Pattern Recognition, 2000. Proceedings. 15th International Conference on*, Vol. 1, pp. 1043–1046. IEEE, (2000)

[11] Leonardo Neumeyer, Bruce Robbins, Anish Nair, and Anand Kesari, "S4: Distributed stream computing platform," In *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on*, pp. 170–177 (2010)

[12] Stephen Robertson and Hugo Zaragoza, "The probabilistic relevance framework: Bm25 and beyond," Now Publishers Inc (2009)

[13] J. J. Rocchio. "Relevance feedback in information retrieval,". In *The Smart retrieval system - experiments in automatic document processing*, pp. 313–323. (1971)

[14] Yong Rui and Thomas S Huang, "A novel relevance feedback technique in image retrieval," In *Proceedings of the seventh ACM international conference on Multimedia (Part 2)*, pp. 67–70. ACM (1999)

[15] Gerard Salton and Chris Buckley, "Improving retrieval performance by relevance feedback," *Readings in information retrieval*, Vol. 24, p. 5 (1997)

[16] Xuanhui Wang, Hui Fang, and ChengXiang Zhai, "A study of methods for negative relevance feedback," In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 219–226. ACM (2008)

[17] Tom White, "Hadoop: The definitive guide," " O'Reilly Media, Inc." (2012)

[18] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica, "Spark: cluster computing with working sets," In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pp. 10–16 (2010)

付 録A

A.1　使用したエンティティー一覧

評価実験で使用したエンティティについて，本文中で使用した Entity ID との対応表を記しておく．

Table A.1: Correspondence table of Entity and Entity ID.

Entity	ID
Sara_Bronfman	01
Benjamin_Bronfman	02
Gretchen_Hoffman	03
KentGuinn4Mayor	04
Maurice_Fitzgibbons	05
RonFunches	06
Ed_Bok_Lee	07
Ruben_J._Ramos	08
Charles_Bronfman	09
Clark_Blaise	10
George_Sinner	11
Corn_Belt_Power_Cooperative	12
Luz_del_Sur	13
Appleton_Museum_of_Art	14
Jennifer_Baumgardner	15

Entity	ID
Paul_Marquart	16
Marion_Technical_Institute	17
Atacocha	18
Richard_Edlund	19
Weehawken_Cove	20
Drew_Wrigley	21
Stevens_Cooperative_School	22
Hjemkomst_Center	23
Buddy_MacKay	24
Haven_Denney	25
Bob_Bert	26
tonyg203	27
Austral_Group	28
Jeremy_McKinnon	29
Intergroup_Financial_Services	30

質疑応答リスト

1. まつ本先生：Incremental Rocchio の適合判定を手で行うというのは，リアルタイムで行うということに矛盾するのでは.
   A. フィルタリング処理，適合判定，プロファイルの拡張は全て非同期で実行されるので，一部の処理が遅くなっても他の処理時間には影響しない.

2. 長野先生：リアルタイム性というのは何によって支配されるのか？ 並列数を増やすことで対応できる部分と出来ない部分があると思うが，その切り分けについて説明してほしい.
   A. 大量の文書を処理する必要があるフィルタリング処理のみ並列に実行する．適合判定やプロファイルの拡張は並列処理はしないが，それぞれ非同期に実行する.

3. 大西先生：プロファイルが更新された場合は，以前の文書はもう一度検索されるのか.
   A. 検索されない．リアルタイム性は失われるが，もう一度フィルタリングすれば更に精度が向上することは期待できる.

4. 長野先生：今回の実験は日本語を対象にしているのか．日本語にした場合，考慮すべきことは.
   A. 今回は英語．日本語をフィルタリングする場合は形態素解析が必要.

5. まつ本先生：PE は完全に並列か（ランダムに分散しているのか）．文書の特性に応じて特定の PE で文書を処理した方がいいのでは.
   A. 今回はランダムに分散させている．ただし，PE には key を割り当てられるので，key を文書の特性とすれば，そのような処理も可能.