オンラインパターン照合と並行オブジェクトに基づくイベントストリーム処理サーバの設計と実装

細川　威樹† 　金田　悠作†† 　有村　博紀††

† 北海道大学工学部　〒 060–0814 北海道札幌市北区北 14 条西 9 丁目
†† 北海道大学大学院情報科学研究科　〒 060–0814 北海道札幌市北区北 14 条西 9 丁目
E-mail: †ov105endeavour@gmail.com, ††{y-kaneta,arim}@ist.hokudai.ac.jp

あらまし　センサーネットワーク等の大規模ストリーム処理応用において重要な基盤技術である多ストリーム多パターン照合を考察する．本論文では，多ストリーム多パターン照合問題に対して，オンラインパターン照合アルゴリズムと並行オブジェクト技術の結合に基づく分散ストリーム検索サーバの構成方法を与える．さらに，その Erlang 言語による実装を与え，その性能について予備実験を行う．

キーワード　ストリーム処理，並行オブジェクト，Erlang 言語，文字列パターン照合アルゴリズム

Design and Implementation of Event Processing Server based on Online Pattern Matching and Concurrent Objects

Takeki HOSOKAWA†, Yusaku KANETA††, and Hiroki ARIMURA††

† Faculty of Engineering, Hokkaido University, N14 W9, Sapporo 060–0814, Japan
†† Grad. School of Inform. Sci. and Tech., Hokkaido University, N14 W9, Sapporo 060–0814, Japan
E-mail: †ov105endeavour@gmail.com, ††{y-kaneta,arim}@ist.hokudai.ac.jp

Abstract　In this paper, we report the design and implementation of an efficient event processing server for multi-pattern multi stream pattern matching based on online pattern matching and concurrent objects.

Key words　stream processing, concurrent object, Erlang, string pattern matching

1. はじめに

イベントストリームに対するパターン照合は，センサーネットワーク等の大規模ストリーム処理応用において重要な基盤技術である．現在，主に単一のパターンに対するイベントパターン照合技術が，盛んに研究されている [1]．一方，ネットワーク上のストリームデータに対する大規模応用を考えると，超多数のパターン照合タスクを，限定された計算資源のもとで効率よく実行する方法は重要である．

そこで，本研究では，多ストリーム多パターン照合問題を考察し，オンラインパターン照合アルゴリズム [8] と並行オブジェクト技術 [9] に基づく分散ストリーム検索システムの構成方法を与える．さらに，その Erlang 言語 [7] による実装を与える．

このシステムでは，各パターンに対するパターン照合プロセスを，オンラインパターン照合アルゴリズムを実装し，それ自身の記憶空間を持って，非同期メッセージ送受信により計算する並行オブジェクトとして実現する．パターン照合サーバは，入力ストリームを非同期に受けとり，多数のパターン照合プロセスを呼び出しながら，動き続けるプロセスである．

このサーバは，新規パターンの実行時の動的な登録や，異なるストリームごとの照合の実行と，ストリームの入れ替わりに応じて，並行オブジェクトの適応的に生成・消滅を行う．これにより限定された計算資源で無限長のストリームを効率よく処理可能である．さらに，既存のオンラインパターン照合アルゴリズムを，本方式のパターン照合プロセスに変換する一般的方法を与える．

最後に，性能評価のための人工データを用いた簡単な計算機実験を行った．

2. 準　　備

2.1 イベントストリーム

$\mathbf{N} = \{0, 1, 2, \ldots\}$ で非負整数全体の集合を表す．Σ を有限のアルファベットとする．アルファベットの要素 $e \in \Sigma$ を**イベント**という．以下では，イベントを文字と同一視して考える．$|\Sigma|$ で Σ の要素数を表す．Σ^* と Σ^+ で，それぞれ，Σ 上の全ての有限長イベント列と，全ての空でない有限長イベント列の集合を表す．長さ $n \geqq 0$ のイベントストリームをイベントの列 $S = s_1 \cdots s_n \in \Sigma^*$ とする．

2.2 多ストリーム多パターン照合問題

以下に，本稿で考察する多ストリーム多パターン照合問題を述べる．まず，単ストリーム単パターン照合問題を定義する．

［定義 1］ **パターン**は長さ $m \geqq 1$ のイベント列 $P = p_1 \cdots p_m \in \Sigma^*$ である．パターン P が長さ $n \geqq 0$ のイベントストリーム $S = s_1 \cdots s_n \in \Sigma^*$ 上で**位置 pos で出現する**とは，イベントストリーム S の連続する部分列 $s_{pos-m+1} \cdots s_{pos}$ とパターン P が等しくなることである．このとき，pos を P の**出現位置**と呼び，P の S 上の出現位置全体の集合を $Occ(P,S) = \{1 \leqq pos \leqq n \mid P \text{ は } S \text{ 上で位置 } pos \text{ に出現する}\}$ で表す．**単ストリーム単パターン照合問題**とは，パターン P とストリーム S を入力として受け取り，$Occ(P,S)$ を求める問題である．

続いて，この単パターン単ストリーム照合問題をもとに多ストリーム多パターン照合問題を定義する．

［定義 2］ $\mathcal{P} = \{P_1, \ldots, P_L\}$ をパターンの集合，$\mathcal{T} = \{S_1, \ldots, S_K\}$ をストリームの集合とする．パターン P_l がストリーム S_k 上で位置 pos で出現するとき，出現位置を pos から拡張し，3 つ組 (l,k,pos) で表す．このとき，P_l の S_k 上の出現位置全体の集合は $OCC(\mathcal{P},\mathcal{T}) = \{(l,k,pos) \mid 1 \leqq l \leqq L, 1 \leqq k \leqq K, pos \in Occ(P_l,S_k)\}$ と定める．**多ストリーム多パターン照合問題**とは，パターンの集合 $\mathcal{P}$ とストリームの集合 $\mathcal{T}$ を入力として受け取り，$OCC(\mathcal{P},\mathcal{T})$ を求める問題である．

2.3 オンラインパターン照合アルゴリズム

本稿のパターン照合システムの鍵が，オンラインパターン照合アルゴリズムである．**オンラインパターン照合アルゴリズム** *(online pattern matching algorithm)* [8] は，パターン P を前処理した後で，各時点 $t = 1, \ldots, t$ で，入力テキスト S から新しい文字 $\alpha = S_t$ を一文字ずつ受取り，テキストの接頭辞 $S_1 \cdots S_t$ に対する現時点での照合結果を出力するアルゴリズムである．

一般的にここでは，オンラインパターン照合アルゴリズムは，次の 3 つの演算をもつ抽象データ構造 $state$ とみなす：

- $state$.create(P): パターン P を前処理し，状態 $state$ を初期化する．
- $state$.update(α): 文字 $\alpha \in \Sigma$ を受取り，状態 $state$ を更新する．
- $state$.is_accept() $\in \{0,1\}$: 状態 $state$ が受理状態かどうか判定する．

次の補題は容易である．

［補題 1］ 上記のデータ構造 $state$ において，パターン長 m に対して，初期化および，更新，受理判定演算に要する計算時間を，それぞれ，$T_{init}(m), T_{update}(m), T_{accept}(m)$ とおき，領域量を $S(m)$ とおく．このとき，オンラインパターン照合アルゴリズムは，パターン照合問題を長さ n のテキストに対して，前処理時間 $T_{init}(m)$ と，領域 $S(m)$, 計算時間 $(T_{update}(m) + T_{accept})n$ 時間で解く．

次に，代表的なオンラインパターン照合アルゴリズムの例を示す．

```
Algorithm on-line-KMP;
    read(symbol); j := 0;
    while symbol != end-of-text do begin
        while j < m and pat[j + 1] = symbol do begin
            j :=j + 1; if j = m then write(1) else write(0);
            read(symbol);
        end;
        if Strong_Bord[j] = -1 then begin
            write(0); read(symbol); j := 0;
        end else
            j := Strong_Bord[j];
    end
```

図 1　オンライン KMP アルゴリズムの実行時処理

2.3.1 KMP アルゴリズム

KMP **アルゴリズム**は，定数文字列のオンラインパターン照合を行うアルゴリズムである（[4] の 2.2.1 節）．図 1 にオンライン KMP アルゴリズムを示す．

このアルゴリズムは，前処理 create(P) として入力パターンから，$BORDER$ 表と呼ばれる 1 次元整数配列（図 1 中では $Strong\_Bord$）を計算する．$Strong\_Bord[k]$ は照合がパターンの k 文字目で失敗した場合に，照合を再開する位置，あるいは -1 を保持する．ここで，-1 ならば次の文字を読み，パターンの先頭から照合を再開することを意味する．実行時処理 $state$.update(α) として，入力テキストから次の文字 α を受け取り，パターンの照合を 1 文字分進める．これを，パターンの照合検査 $state$.is_accepted() が成功するか失敗するまで繰り返す．パターンの照合が成功した場合，パターンが出現したことを示す 1 を出力し，$BORDER$ 表にしたがってパターンの照合位置を設定し，照合を再開する．パターンの照合が失敗した場合も $BORDER$ 表にしたがってパターン照合を再開する．テキストの終端に達したら終了する．

このアルゴリズムの計算量は，入力パターン長を $m = |pat|$ とし，入力テキスト長を $n = |text|$ として，$BORDER$ 表の計算に $O(m)$ 時間と，$O(m)$ 領域，実行時間は $O(n)$ である [4].

2.4 Shift-And アルゴリズム

Shift-And **アルゴリズム**は，ビット並列手法にもとづき，定数文字列のオンラインパターン照合を行うアルゴリズムである（[8] の 2.2.2 節）．ビット並列手法とは，汎用 CPU 上のビット演算と数値演算のレジスタ内並列性を利用し，計算を高速化する手法である [8].

入力テキスト T の長さを n で表し，入力パターン P の長さを m で表す．また，計算機ワードの長さを w 表す．

このアルゴリズムは，前処理時 create(P) に入力パターン P からマスクテーブル B を構築する．ここで，文字 $t \in \Sigma$ に対するマスクテーブルの要素 $B[t] \in \{0,1\}^m$ は，t のパターン P 中の位置を表すビットマスクである．

実行時 $state$.update(α) には，始めに各時点における状態集合を保持するビットマスク D を $D := 0^m$ で初期化する．次に，入力テキスト $T = t_1 \cdots t_n$ の各文字 $t_i \in \Sigma$ $(1 \leqq i \leqq n)$ に対して，以下の更新式で D を更新する．

$$D := ((D \ll 1) \mid 0^{m-1}1) \,\&\, B[t_i]$$

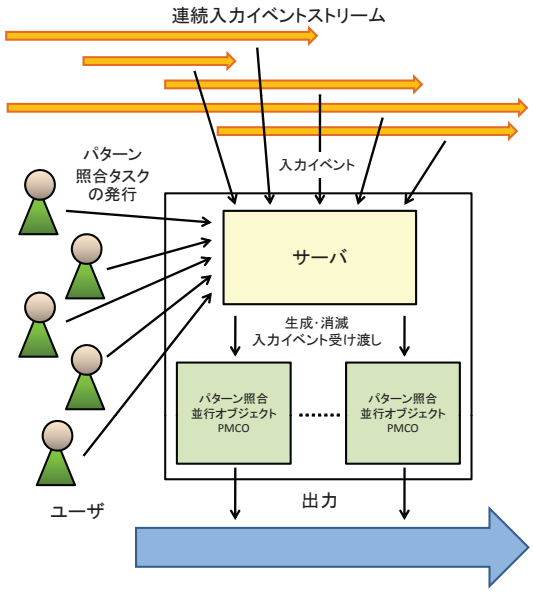

図 2　システムの概要

パターンの出現検査 $state$.is_accepted() は，更新式の適用後に D が 10^{m-1} と等しいかどうかを調べることで確認できる．

Shift-And アルゴリズムは，パターン長が $m \leqq w$ を満たす場合，$O(m + |\Sigma|)$ 前処理時間と，$O(|\Sigma|)$ 領域，$O(n)$ 時間と効率的に動作する．Shift-And アルゴリズムは，文字クラスや，有限ギャップ，オプション文字，一文字繰り返し等の限定された正規表現に拡張可能であることが知られている（[8] の 4 節）．

2.5　並行オブジェクト指向プログラミング

並行オブジェクト指向プログラミングでは，独立したメモリ空間をもち，非同期に動作する複数の並行オブジェクト（Concurrent Objects, CO）によって計算を実行する．以後，並行オブジェクトを単にプロセスと呼ぶ．並行オブジェクト間の相互作用は，共有メモリ等を用いず，メッセージ通信（message passing）によって行うことで，同期や排他等の並行プログラミング特有の困難を避けて，並行計算を扱える点に特徴がある．

特に，従来の OS が提供するスレッドを用いるスレッドプログラミングに比較して実行速度で長所があり，計算をイベント処理に分解するイベント処理プログラミングに比較して，アルゴリズムの自然で容易な実装が可能であるとされている．

例として，並行オブジェクト指向プログラミング言語には，ABCL/1 等がある [9]．ABCL/1 では，過去形（非同期送信），現在形（同期交信），未来形（非同期交信）等の各種のメッセージ通信モードをもつ [5]．Erlang や, Scala, Google Go 等の最近の並行プログラミング言語もこれらの系譜上にある．これらの言語の実行系は，ＯＳと独立な自前の仮想機械をもち，きわめて多数の非常に軽量なプロセスを，効率よく実行することができる [3], [7].

3.　提案システム

3.1　システムの概要

図 2 に，提案システムの概要を示す．このシステムは，多数のイベントストリーム $S_1, \ldots, S_K$ $(K \geqq 0)$ と，多数のパターン $P_1, \ldots, P_L$ $(L \geqq 0)$ を入力として受け取り，いずれかのパターンに出現すると，合致したパターンの番号 pid，パターンが出現したストリームの識別番号 sid とパターンの出現位置 pos の組 $\tau = (pid, sid, pos)$ 出力ストリームにを出力する．このシステムは，パターン数とイベントストリーム数が動的に増減する場合にも対応可能な柔軟な構成を持つ．

3.2　パターン照合並行オブジェクト

照合プロセスは，入力ストリームに対するパターン照合を行う並行オブジェクト π である．ストリーム ID sid とパターン P の組をプロセスの署名と呼ぶ．照合プロセス π は，以下の演算をもつ．

（1）**プロセスの生成**:
仕事： 照合プロセスを生成する．
手順： 署名 (sid, P) を設定し，それが用いるパターン照合アルゴリズムで定まる内部状態を create(sid, P) で初期化する．

（2）**照合状態の更新**:
仕事： 次の文字に対してパターン照合を行う．
手順： 照合サーバから，入力イベント (sid, tid, α) を非同期送信で受取り，イベント記号 α に対する演算 $state$.update(α) で，アルゴリズムの状態 $state$ を更新する．更新後，それが受理状態であるかの検査 $state$.is_accepted() を行い，それが真なら，照合通知 (sid, mid, P) を出力プロセスに非同期送信する．

（3）**パターンの消滅**:
仕事： 動作を停止し，消滅する．
手順： 照合サーバから，メッセージ terminate を受け取ると，作業を停止し，照合サーバに向けてメッセージ terminated を非同期送信で送信した後，消滅する．

照合プロセスは，1 文字分の処理が終了すると，次のメッセージが来るまで待機状態になる．パターン数が p 個，ストリーム数が s 個であれば，パターン照合並行オブジェクトは最大で $p \times s$ 個生成される．

3.3　パターン照合サーバ

パターン照合サーバは，メッセージを受け取りながら，パターン登録，入力イベントの受信，照合プロセスの生成・消滅，入力イベントの照合プロセスへの送信等の一連のタスクを実行し，動き続ける独立したプロセスである．

サーバは，データ構造として次の表をもつ：

- パターン表 PT：パターンの有限集合．
- 入力イベントキュー Q：イベントの有限長の待ち行列．
- 照合プロセスのディスパッチ表 DT：ストリーム ID sid をキーとし，プロセス ID の有限集合 $DT[sid]$ を値とする辞書．

サーバのタスクと，そのアルゴリズムは以下のとおりである．

（1）**ユーザからのパターン登録依頼**:
仕事： 複数のユーザから送信されたパターンを登録する．
手順： 非同期に送信されたパターン P を受け取ると，サーバは P をパターン表 PT に挿入する．

（2）**入力ストリームからのイベント受信**：
仕事： 多数のストリームから送信されたイベントを受け取る．
手順： ストリーム ID sid をもつ入力ストリームから，非同

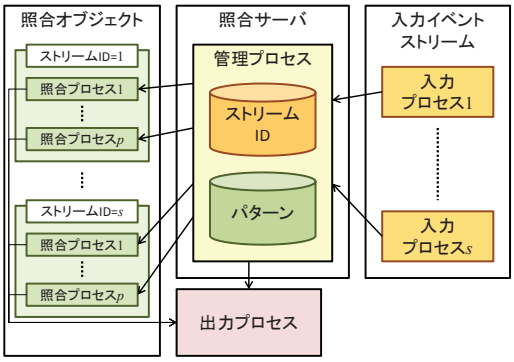

図 3　実　　装

期に送信された入力イベント $e = (sid, tid, \alpha)$ を受け取り，到着順に 1 本のキュー Q に Q.enque(e) で挿入する．これは，1 本の仮想的な入力ストリームにまとめることに対応する．イベントの時刻印 tid で，対応するストリームの到着時刻表を更新する：$TS[sid] = tid$.

（3） **照合プロセスへのイベント送信**：

仕事： 受信した入力イベント $e = (sid, \alpha)$ を，sid に対応する照合プロセスに e をメッセージ送信する．

手順： キュー Q から取り出した入力イベント $e = (sid, \alpha) = Q$.deque() に対して，ストリーム ID sid をキーとして照合プロセスのディスパッチ表 DT を引き，sid に対応する照合プロセス ID のリスト $\{mid_1, \ldots, mid_h\} = DT[sid]$ $(h > 0)$ を得る．次に，各照合プロセス mid_i $(i = 1, \ldots, h)$ に対して，イベント e を引数とするメッセージ $\langle$match$, e\rangle$ を送信する．

（4） **照合プロセスの生成・破棄**：

仕事： ストリームの増減に応じて対応する照合プロセスを生成・破棄する．

手順：（生成）サーバは，初めて到着した sid に対し，パターン表 PT に登録された各パターン $pat \in PT$ に対して，新しい照合プロセス mid を生成し，組 (sid, mid) を，ストリーム ID をキーとしてパターン表に登録する．

手順：（消滅）表 TS に対して，一定の時間ごとに $CurrTime - TS[sid] > W$ となるストリーム ID sid を見つける．対応する照合プロセス $mid = DT[sid]$ に，非同期送信で破棄命令 terminate を送り，実行を継続する．ある時点で破棄報告 terminated を受け取ったら，ディスパッチ表から mid を取り除く：$DT = DT - \{\langle sid, mid\rangle\}$.

3.4　解　　析

提案システムの計算時間を解析する．長さ m のパターンをもつ 2.3 節のデータ構造 $state$ に対して，初期化および，更新，受理判定に要する計算時間を，パターン長 m に対しそれぞれ，$T_{init}(m), T_{update}(m), T_{accept}(m)$ とし，領域量を $S(m)$ と置く．$\mathcal{P}$ をパターン集合，パターン P のパターン長を $|P|$，入力イベントの総数を N とする

［定理 1］ このとき，任意のパターン集合 P に対して，提案システムの総計算時間は，個々のプロセスの計算時間の和になる．具体的には，次の通りである．

```
%%% server プロセス
server(State) ->
   receive
    {addPattern, Pat} -> % パターン Pat の追加;
    {addEvent, Sid, Event} ->
      Pid = lookupProcess(Sid),
      if
        Pid == undefined ->
          各 Pat に対して，Spawn
          で照合プロセス Pid を生成，
          registerProcess(Sid, Pid);
        Pid /= undefined ->
          Pid ! Event   %Send 構文
      end
  end,
  server(State).
```

図 4　照合サーバの Erlang による擬似コード

```
%%% matcher プロセス
matcher(Pattern, Border, OrgBorder) ->
   receive
    {putEvent, Event} ->
      [P | Ps] = Pattern,
      [B | Borders] = Border,
      if
        Event == eof ->
          exit();
        Event == P ->
          Border1 = Borders,
        Event /= P ->
          Pattern1 = B,
          Border1  = Border
      end
  end,
  matcher(Pattern1, Border1, OrgBorder).
```

図 5　照合プロセスの Erlang による擬似コード

- 領域 $S = \sum_{P \in \mathcal{P}} S(|P|)$ かつ，
- 前処理時間 $Q = \sum_{P \in \mathcal{P}} T_{init}(|P|)$，
- 実行時間 $T = \sum_{P \in \mathcal{P}} (T_{update}(|P|) + T_{accept}(|P|))N$.

ただし，サーバの処理（イベント受け取り・送出，プロセスの生成，プロセスの表引き）は定数時間で実行可能と仮定する．

［証明］ 全体の計算時間は，高々，個々の照合プロセスの計算時間の総和であることから，全てのパターン照合プロセスと入力ストリームについて，各イベント毎の計算時間の総和に関するならし解析を行うことで結果が容易に示せる．　□

4.　実装と実験

4.1　実　　装

プログラミング言語 *Erlang* を用いて提案システムを実装した．Erlang は関数型プログラミング言語であり，言語環境が OS から独立したプロセス管理を行うため，個々のプロセスの実行効率が良い (軽量プロセス) という特徴を持つ．そのため並行分散処理を得意とし，提案システムを実装するのに適してい

る．Erlang 言語で実装したシステムの構成を図 3 に示す．

図のように，管理プロセス，入力プロセス，出力プロセス，照合プロセスの 4 種類のプロセスによってシステムが構成される．入力プロセスは入力イベントストリームに対応するプロセスであり，1 つのプロセスが 1 つのストリームである．出力プロセスはパターン照合結果等の出力を担当するプロセスである．照合プロセスが先に述べたパターン照合並行オブジェクトである．ここでは照合アルゴリズムに KMP アルゴリズムを用いた．管理プロセスはシステム全体を統括する．入力プロセスからの入力イベントを受け付け，出力プロセスや照合プロセスの生成破棄を行う．

図 4 と図 5 に照合サーバと照合プロセスの Erlang 言語による擬似コードを示す．照合サーバでは，spawn による並行プロセス生成や，Send 節と Receive 節による非同期メッセージ通信，再帰呼出によるサーバの実行ループの実現などの Erlang の機能を使用している．照合プロセスでは，Erlang でオンライン KMP を実装した．単純な実装では $O(mn)$ 時間になるが，リストと構造体の共有を使って，$O(n)$ 時間実装を行った．

4.2 実験データ

人工データを用いて実験を行った．$\Sigma = \{a, b, \ldots, z\}$ を，英文字の a から z までの 26 個のアスキー文字から成るイベント集合とする．k 個のランダムパターン $\mathcal{P} = \{P_1, \ldots, P_k\}$ と，l 個のランダムストリーム $\mathcal{T} = \{S_1, \ldots, S_l\}$ を用意した．各ランダムパターン P_i は，それぞれ Σ から一様に生成されたランダムな長さ m のイベント列である．同様に，各ランダムストリーム S_j は，それぞれ Σ から一様に生成されたランダムな長さ n のイベント列である．

4.3 方　　法

前述の Erlang 言語で実装されたシステムを用いて実験を行った．全ての実験は，PC（Intel Core i7 860 2.80GHz, RAM 8GB, Debian GNU/Linux 5.0.4, Erlang R13B）上で行い，総実行時間の測定には Linux の time コマンドを使用した．

実験では実験サーバは，k 個のランダムパターン $\mathcal{P} = \{P_1, \ldots, P_k\}$ と l 個のランダムストリーム $\mathcal{T} = \{S_1, \ldots, S_l\}$ に対して，オンラインパターン照合を検索サーバに実行させる．実験サーバは，time コマンドによって，全ての照合プロセスが対応する入力ストリームを読み終わり，終了するまでの総実行時間 T_{all} を計測する．

4.4 実　験　1

Erlang システムによるメッセージ通信の基本性能を測定する実験を行った．実験では，入力イベント数 N に対して，sender プロセスから receiver プロセスに対して，N 個のイベントを連続して送り，sender が最初のイベントを送ってから，receiver が最後のイベントを受け取るまでの総実行時間 T_{all} を計測した．

図 6 に，イベント数 N に対する総実行時間 T_{all} のグラフを示す．このグラフより，この実験では送受信の初期オーバーヘッドは 1msec 以下であり，$N \geqq 1000$ では総実行時間はイベント数に比例することがわかる．図 7 に，イベント数 N に対するスループット TH のグラフを示す．このグラフより，$N \geqq 100,000$ イベントでは，スループットは $TH = 12$ M(イベント/秒) 程度であることが分かる．

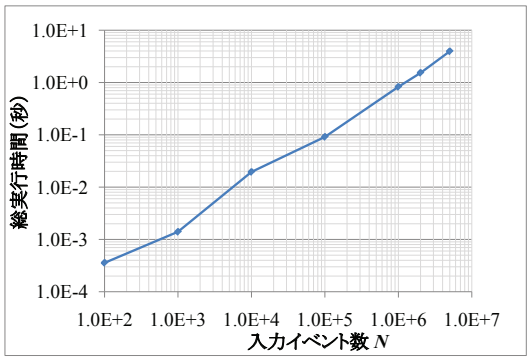

図 6　実験 1 のイベント数 l に対する総実行時間

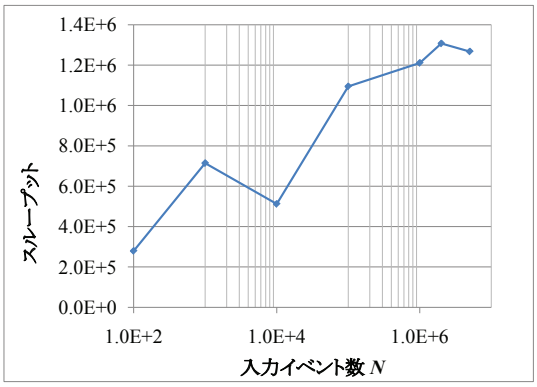

図 7　実験 1 のイベント数 l に対するスループット TH_{in}

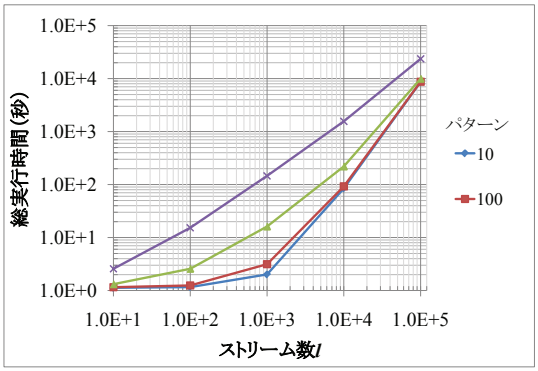

図 8　実験 2 のストリーム数 l に対する総実行時間

4.5 実　験　2

ストリーム数とパターン数に対する総実行時間の変化について調べた．図 8 に，パターン数 $k = 10 \sim 100000$ とストリーム数 $l = 10 \sim 100000$ の全ての組み合わせについて，総実行時間 T_{all} を示す．パターン長は $m = 3 \sim 5$ で，ストリーム長は $n = 5 \sim 10$ のランダム数である．図 9 に，管理プロセスが入力として受け取るスループット (個数/秒) を示す．パターン数が同じであれば，計算時間はストリーム数 l に比例することが期待されるが，実験ではそのようになっていない．

4.6 実　験　3

パターン数 $k = 10$，ストリーム数 $l = 10$，パターン長 $m = 10$ とし，ストリーム長を $n \in \{10, 100, 1000, 10000, 100000\}$ として，ストリーム長を変えながら実行時間を測定した．実行時間と

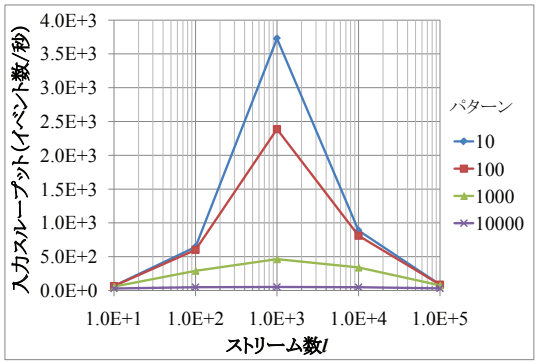

図 9　実験 2 のストリーム数 l に対する入力スループット TH_{in}

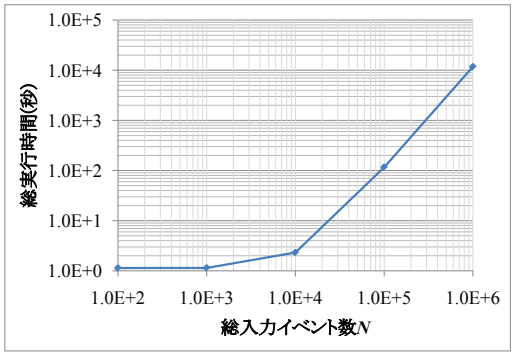

図 10　実験 3 のストリーム長 n に対する総実行時間

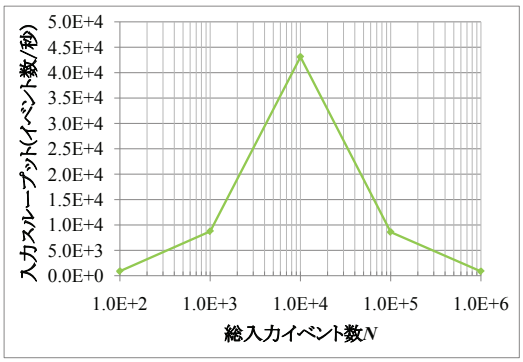

図 11　実験 3 のストリーム長 n に対するスループット

入力スループットは実験 2 と同様にして求めた．図 10 と図 11 はそれぞれ総実行時間と入力スループットのグラフである．

計算時間はストリーム長 n に比例することが期待されるが，実際の実行時間を見るとそのようになっていない．原因は現在不明だが，照合サーバのプロセス管理方法等に原因がある可能性もある．この点については，今後，詳細に調べたい．

5. おわりに

本稿では，分散ストリーム検索サーバのオンラインパターン照合アルゴリズムと並行オブジェクト技術に基づく構築方法を提案した．また，この技術に基づき，プロトタイプシステムのプログラミング言語 Erlang を用いた実装について報告した．

実験では，性能面で現在の実装にはかなり改善の余地があることがわかった．一方で，大規模多ストリーム多パターン照合システムを，既存のオンライン照合アルゴリズム技術を基盤として，性能保証を与えながら，系統的に構築するための本アプローチの有用性は示せたと思われる．

関連研究として，Agrawal, Diao ら [1] は，正規表現の部分クラスを用いたパターン照合に基づくパターン照合モデルを与え，RFID 等のセンサネットワークを応用としたイベントストリーム処理システムを提案している．今後の課題として，彼らのシステムを極く多数のパターンの並行実行に拡張することが今後の課題である．

金田らは，高速ストリーム処理のためのビット並列手法に基づく文字列パターン照合手法とその FPGA 設計について考察している [6]．本稿で議論した分散ストリーム検索サーバのハードウェア実装も，興味深い研究課題である．また，系列マイニングとの融合も可能な研究トピックである．

文　献

[1] J. Agrawal, Y. Diao, D. Gyllstrom, N. Immerman, Efficient pattern matching over event streams, *Proc.SIGMOD'08* , 147–160, 2008.

[2] J. Armstrong, プログラミング Erlang, オーム社, 2008.

[3] J. Armstrong, B. O. Dacker, S. R. Virding and M. C. Williams, Implementing a functional language for highly parallel real time applications, *Software Engineering for Telecommunication Switching Systems*, IEEE, 157-163, 1992.

[4] M. Crochemore and W. Rytter, *Jewels of Stringology*, World Scientific Publishing, 2003.

[5] 石川 裕, 所 真理雄, オブジェクト指向並行プログラミング言語, 解説, 情報処理, Vol.29, No.4, 325-333, 1988.

[6] 高速ストリーム処理のための文字列パターン照合手法とその FPGA 設計 , 金田悠作, 吉澤真吾, 湊 真一, 有村博紀, 宮永喜一, 電子情報通信学会 2009 年総合大会, D-4-18, 松山大学, 2009 年 3 月.

[7] J. Larson, Erlang for Concurrent Programming, *CACM*, Vol.52, No.3, 48–56, 2008.

[8] G. Navarro and M. Raffinot, *Flexible Pattern Matching in Strings: Practical On-Line Search Algorithms for Texts and Biological Sequences*, Cambridge University Press, 2002.

[9] A. Yonezawa, E. Shibayama, E. Takeda, and T. Honda, Object-Oriented Concurrent Language ABCL/1, *Object-Oriented Concurrent Programming*, A. Yonezawa and M. Tokoro (eds.), MIT Press, 55–89, 1987.