Vol.2019-DBS-169 No.13
Vol.2019-IFAT-136 No.13
2019/9/10

多頻度・順不同で到着するシーケンスデータの主キーごとの処理順序制約を満たすリアルタイム並列処理手法

山添高弘[†1][†2]　三輪忍[†1]　本多弘樹[†1]

概要：業務アプリケーションやセンサネットワーク等のシステムにおける，多頻度かつ順不同で到着する『主キー情報＋時系列情報＋属性情報』を有するシーケンスデータを「主キー情報」のグループごとに「時系列情報」に基づく処理順序制約を守りながら処理する問題に対し，一定のレイテンシ内でリアルタイム処理を可能とする並列処理手法を開発した．
また，実装・評価の結果，本手法がポーリング処理による無駄な待ち時間を排除し，複数台サーバでの並列処理によりスケールすることを確認した．

キーワード：ストリーミングデータ処理，シーケンスデータ，並列処理

1. はじめに

近年 IT システムは，社会インフラとしての位置づけまで高まり，金融系システムや交通系システムだけでなく，電子商取引，RFID による物流管理にも導入されている．そのため，各種 IT システムでは，その大量のデータをいかに効率よくかつスピーディーに処理する事[1][2]が社会インフラ上，より必要になってきた．

業務アプリケーションシステムにおいては，「カネモノの流れ」「各種データの状態」を表すデータは，以下の表 1 に分類することができる．

本研究で対象とするシーケンスデータはトレーサビリティを目的としたデータであるため，主キー情報と時系列情報と更新前後の情報を持つ．また，データの特性から，生成・保管・データ連携等においての処理の優先度は低く，非同期で連携されるため，処理するシステムとしては順不同で到着するデータを処理する事となる．そのため，シーケンスデータを処理対象とするシステムにおいては，データの到着順ではなく，データの内容（主キー情報および時系列情報）に応じて処理する必要がある．また，このシーケンスデータを活用した顧客アナウンスや異常検知の関連先への通知を行う場合があるため，リアルタイムに処理する必要もある．

しかし，シーケンスデータを含むストリームデータの処理において，既存のストリーミングデータ処理技術[3][4]では，個々のシーケンスデータの処理順序制約を設けることができない．そこで本研究では，このシーケンスデータに処理順序制約を設けたリアルタイム処理が可能なソフトウェアアーキテクチャを提案する．

表 1　データの種類と特性

データの種類	特性	金融取引での例	航空座席予約での例
マスタデータ	台帳，最新状態保持，少量	銀行口座マスタ	座席予約マスタ
トランザクションデータ	更新操作都度生成，同期的	入出金・振込操作データ	予約・変更・キャンセルデータ
シーケンスデータ	更新操作都度生成，非同期的，トレーサビリティ	入出金操作前後データ	予約更新操作前後データ

2. シーケンスデータの特徴と活用

シーケンスデータはトレーサビリティの観点からも，データのユニークなキーを表す『主キー情報』，データの更新順序やタイミングを表す『時系列情報』，データの更新内容を表す『属性情報』を持ち，システム利用におけるピーク時は一時点で大量に生成されるデータとなる．また，本データは顧客が直接利用するデータではない為，生成・保管・連携等においての処理の優先度は低く，非同期で連携される場合が多い事が特徴として挙げられる．

ここでは，シーケンスデータを取り扱う代表的な事例として航空系座席予約システムを用いたシーケンスデータの特徴を述べる．図 1 に示す通り，オンライン処理で生成されたシーケンスデータは連携元の制約により非同期および順不同でデータ蓄積サーバに連携される．

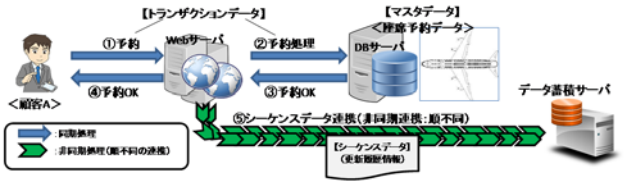

図 1　座席予約システム構成例

†1 電気通信大学　The University of Electro-Communications
†2 株式会社エーフロンティア　A-frontier Co.,Ltd.

ⓒ2019 Information Processing Society of Japan

Vol.2019-DBS-169 No.13
Vol.2019-IFAT-136 No.13
2019/9/10

シーケンスデータの具体的なデータ構造は表 2 の通りである.予約番号を主キーとする予約データは別に存在し，シーケンスデータはその予約データに対する更新操作の毎に生成されるデータである．また，大量更新が発生する状況として,「台風」や「飛行機の機材故障」などをきっかけとして，複数便の予約データが一斉に変更やキャンセルになる場合があり，関連する全ての予約データが更新されるため，結果このシーケンスデータが一時点で大量に生成されるのが特徴である．

また，このシーケンスデータを基に，予約データに対する全ての更新操作内容が把握できるため，その内容に応じた顧客アナウンス（メール通知や電話連絡)，帳票出力（領収書,案内文書),関連先への通知に活用される場合がある．

なお，連続して発生し続けるデータは広義としてストリーミングデータ[5]と呼ばれ，その内，時系列情報と属性情報を持ちデータの順序に重要な意味を持つデータをシーケンスデータと呼んでいる．

表 2　シーケンスデータの構成

分類	要素名	PK	説明
主キー情報	予約番号	○	予約番号（航空券番号）
時系列情報	更新回数		予約情報に対する更新回数（１から順番にインクリメント）
属性情報	操作区分		操作内容を表す区分　新規予約／変更／キャンセル
	更新内容		変更前後内容
	・・・・		・・・

このシーケンスデータをインプットとし，顧客アナウンスや関連先への通知を行うために，シーケンスデータの到着後なるべく早く処理を行いたい．但し，その際に重要となるのが，全シーケンスデータを主キー情報でグループ化した場合の時系列情報の順に処理する必要がある．

シーケンスデータの生成と連携方式は連携元システムの制約により非同期での連携となるため，順不同での到着となるため，データの処理順序についてはデータの受け取り側で考慮が必要となる．

台風発生時の例であるが，天候状態により刻々と状況は変わる．空港への台風接近により一度は欠航となる場合があるが，台風経路の変更により欠航キャンセル（つまり予約状態が元に戻る）となる場合がある．そのシチュエーションの場合，欠航を決定した際に欠航のアナウンスを行い欠航キャンセル時に欠航キャンセルのアナウンスを行うが,搭乗者としては，この順序を守ってアナウンスされることが重要（アナウンスが順序逆だと問題）である．

3. 従来処理手法の問題点

シーケンスデータのように不定期で絶え間なく生成されるデータを処理する方法としては，ストリーミングデータ処理による処理方式が適切と考えられる．

しかし，本研究の対象事例のように，データが順不同で到着し,『主キー情報』単位で『時系列情報』順に処理する為には，従来のストリーミングデータ処理方式やオンライン処理方式ではデータ間の処理順の依存関係を設定した処理が不可能であるため，一時的にデータベース等の記憶領域に蓄積して，定期的なポーリング処理（検索およびソート処理）を行う必要がある．そのため，リアルタイム性が損なわれると共に，処理対象データが存在しない場合でもポーリング処理を行う必要があり，検索処理等による無駄なハードウェアリソースが消費されてしまう．

そこで本研究では，シーケンスデータを「主キー情報」でグループ分けし「時系列情報」順にオンデマンドで処理するシステムを開発することを目的とする．

4. 関連研究

無限に発生し続けるデータを処理する方式としてストリーミングデータ処理や大量データの並列分散処理に関する研究が挙げられる．

以下に Apache から OSS として提供されているライブラリに関して特徴等をまとめる．

(1) Storm[6]

データの処理順序を有向グラフである Topology（トポロジー）として定義し，Topology へのデータのインプットを Spout（スパウト）として定義する．何らかのデータ処理を行うプロセスを Bolt として定義し，Bolt へは Spout あるいは Bolt からデータを受け取り処理する．

処理対象のデータ（１件）を Tuple として定義し，Spout から入力された Tuple が，Topology の構成（有向グラフのフロー）に沿ってデータが流れ各 Bolt で随時処理される．

Storm は複数台で動作し，Spout から入力された Tuple がいずれかのサーバに分散させて処理され，水平方向の拡張性を持つ．

処理対象データの Tuple 間の処理順序を定義する事は不可能であるため，順不同で到着するデータを Storm 内で並び替えての処理や，データの到着待ちを定義する事は難しい．

(2) Spark Streaming[7]

Spark では，入力データを RDD（Resilient Distributed Dataset）と呼ばれる分散データ配列（コレクション）として扱う．

Spark Streaming は, マイクロバッチ方式を採用しており，数秒から数分ほどの短い間隔（ニアリアルタイム）で繰り返しバッチ処理を行う．流れてきたデータを一定時間ごと

ⓒ2019 Information Processing Society of Japan

Vol.2019-DBS-169 No.13
Vol.2019-IFAT-136 No.13
2019/9/10

に区切って RDD として扱うことで，擬似的なストリームデータ処理を実現している．

その為, RDD 間の処理順序変更やデータ到着待ちを定義する事は出来ない．

(3) **Hadoop（MapReduce）[8]**

処理対象データを専用のファイルシステム（HDFS：Hadoop Distributed File System）にファイルとして格納し，バッチ処理を複数台のサーバで分散させて実行する．

データを Key-Value 形式で取扱い，一連の時間のかかる処理をタスクと呼ばれる小さい処理に分割し，多数のサーバでデータ間とタスク間を疎結合で実行する事が出来る．

バッチ開始時点で存在するデータを対象とするため，リアルタイム処理には向かず，処理対象データが順不同で到着するシチュエーションでは，バッチの開始タイミングが決定できない．

5. 提案ソフトウェアアーキテクチャ

本研究で開発したシステムについて説明する．提案手法における実装のポイントは以下の通りである．

- クライアントから受信したシーケンスデータの主キー情報に応じて処理するサーバを特定し転送する
- 処理するサーバ内において主キー毎に時系列情報順にシーケンスデータを保持する
- 個々のシーケンスデータに処理ステータスを設けることでリアルタイム処理を可能とする
- 時系列情報に抜けがあった場合でもタイムアウト制御をすることで全てのシーケンスデータを処理する事を可能とする
- 処理するサーバ内において異なる主キー情報であれば並列で処理する事を可能とする

5.1 前提

本研究で開発したシステムにおける前提は次の通りである．

- 汎用的なライブラリとして，ビジネスの場面でも一般的に用いられる，Apache などのオープンソースソフトウェアで動作するソフトウェア環境で実装する．
- 既存システムと接続する事や，インターネットを介したサービス公開が可能なサービスとするため，Web サービス SOAP[9]を利用した API サービスとして実装する．
- 水平方向に拡張できるよう, 一般的な L7 ロードバランサを利用して負荷分散可能な構成とする．
- シーケンスデータのピーク時の処理目標性能として「１０，０００件／秒」とする．
  - アマデウス社の扱う乗客のデータ処理件数は年 2 億 3,800 万人分：（平均）7.55 件／秒[10]
  - ピーク時は平均値の２０倍のデータ件数と想定：150.94 件／秒
  - アマデウス社の世界シェアは１１％であるため全世界の処理量を算定：1,372.17 件／秒
  - 将来におけるビジネス拡大におけるデータ量増加見込みを現在の５倍と算定: 6,860.85 件／秒　単位切り上げして「１０，０００件／秒」を目標性能に設定．

5.2 ソフトウェアアーキテクチャ構成

システム全体の構成を図 2 に示し WebAP サーバに実装した各機能の名称と機能概要を表 3 に示す．また，個々の WebAP サーバのライブラリ構成は表 4 の通りであり, Web サービス SOAP を受け付ける事を可能とする一般的なオンライン処理基盤となっている．

基本的にはデータの到着都度処理可能なオンライン処理となっており，データ転送元から随時 Web サービス SOAP でデータ受信する. ロードバランサは一般的な L7 ロードバランサを想定し，各 WebAP サーバに均等にリクエスト（負荷）を分散させることを想定している．

詳細は後述するが，＜B : 振分機能＞の処理でシーケンスデータの「主キー情報」に応じて適切なサーバに転送し，各サーバ内（＜D : データ処理機能＞の部分）にシーケンスデータの「主キー情報」でグルーピングしたグラフ構造のリスト情報を保持する．順不同で到着するデータを時系列順のリストとして保持し，各データの処理ステータスを保持することで,「時系列情報」順に処理を行う事を実現している．また，リスト構造で情報を保持することにより，時系列情報に抜けがあった場合でも順序および処理順序を保つことを可能にしている．そして，長時間もしくは永久に時系列情報の抜けが到着しない場合は，所定時間経過後にタイムアウトをすることで，対象のデータを処理する．

ⓒ2019 Information Processing Society of Japan

Vol.2019-DBS-169 No.13
Vol.2019-IFAT-136 No.13
2019/9/10

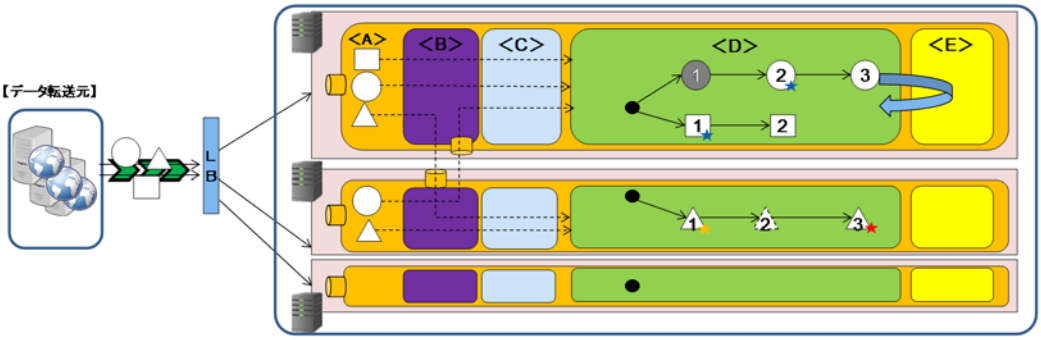

図 2　システム全体構成

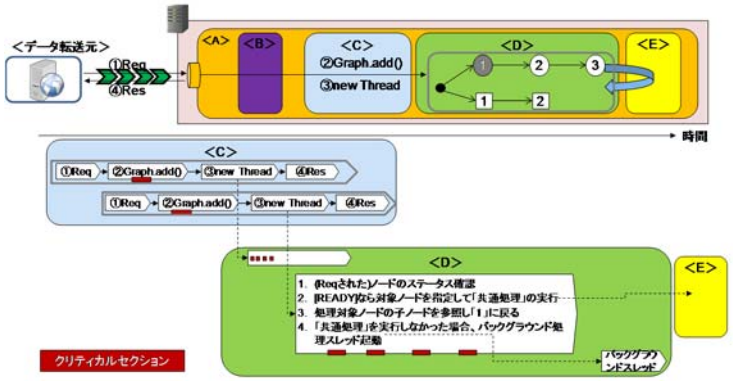

図 3　処理フロー

表 3　WebAP サーバ機能

ID	機能名	機能概要
A	Web サーバコンテナ	標準の Web コンテナ機能．Web サービスのリクエストを受信する基本的な初期処理および終了処理を行う．
B	振分機能	データ連携元から受信したデータを「主キー情報」に応じて適切なサーバを指定し＜C：データ受信機能＞に転送する．
C	データ受信機能	リクエストデータをグラフ構造（主キーのリスト）の内部データに格納し処理用スレッドの生成および起動を行う．
D	データ処理機能	生成された処理スレッドが，各ノードの処理順序条件を監視し，ステータス更新や共有処理の開始処理を行う．
E	共通処理	対象ノードをパラメータとして共通機能の呼出処理を行う．この部分に時系列情報順に処理したい処理ロジックを指定する．

表 4　WebAP サーバライブラリ環境

種類	ライブラリ
Java 実行環境	Java
Web コンテナ	Tomcat
Web サービスエンジン	Apache-AXIS2

5.3　データ処理フロー

データ処理フローについて説明する．各機能における処理フローは図 3 の通りである．

＜A：Web サーバコンテナ＞は J2EE 標準の Web コンテナの機能を用いて Web サービス SOAP のリクエストを受信する基本的な初期処理および終了処理を行う．

＜B：振分機能＞においてデータ連携元から受信したデータの「主キー情報」の値に応じてサーバ＜C：データ受信機能＞に転送する．振り分け先のサーバは，「主キー情報」の MD5 ハッシュダイジェスト値（１０進数表現）の振り分けサーバリストサイズに対する剰余を算出することで決定する．

＜C：データ受信機能＞においてリクエスト内のデータをグラフ構造（主キーのリスト）の内部データに格納し，処理用スレッドを生成および起動する．この内部データの個々のデータをノードとして表現する．

＜D：データ処理機能＞は生成された処理スレッドが，ノードの処理順序条件を監視し，ステータス更新や共有処理の開始処理を行う．生成された処理スレッドで共通処理の実施を試みるが，親ノードが未到着や処理中の状態など，対象ノードの実行開始条件が整わない場合は処理を行わず，バックグラウンド処理用のスレッドを起動する．所定時間経過しても処理開始をしない場合は，バックグラウンドスレッドが共通処理をフラグ付きで呼び出す．

＜E：共通処理＞は，個々のノードをパラメータとして実行する処理であり，この部分に汎用的な処理を指定する事が可能である．共通処理が呼び出される際に通常の処理スレッドから呼ばれたのかバックグラウンドスレッドから呼ばれた（つまり，親ノードが到着せず抜け番の状態のままタイムアウトした状態）のかをフラグ値によって判断する．

なお，本システムはマルチスレッドで動作するため，グラフ構造データへのノード追加や各ステータス変更時に不

ⓒ2019 Information Processing Society of Japan

Vol.2019-DBS-169 No.13
Vol.2019-IFAT-136 No.13
2019/9/10

整合が生じないよう，クリティカルセクションにおいては対象主キーオブジェクトのロック・アンロックによる排他制御を行う．

5.4 ノードの状態遷移

シーケンスデータはまず,「待機」状態のノードとして生成され，その後，各条件に応じて図 4 の通りにステータスが遷移していく．ノードの状態遷移における条件は以下の通りである．

- 条件１：[自ノードの「時系列情報」が１][親ノードの状態が「処理済」][自ノードの「待機」の経過時間が所定時間以上]のいずれか
- 条件２：自ノードに対する親ノードが到着
- 条件３：対象ノードが処理スレッドにおいて処理に着手する

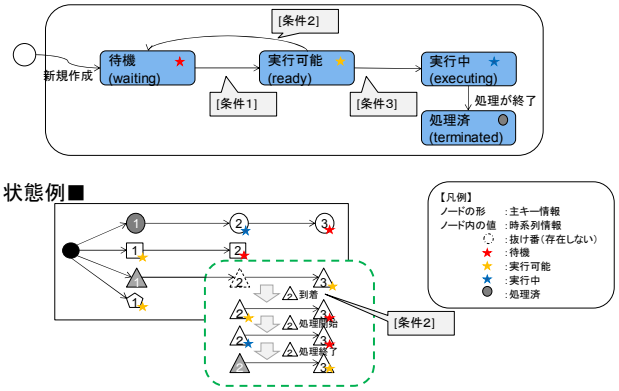

図 4　ノードの状態遷移

6. 実験

本章では，提案ソフトウェアアーキテクチャに対する機能面および性能面についての実験内容について述べる．

6.1 実験方法

データ転送元を表すクライアントサーバ上に Web サービス SOAP の投げ込み用のツール（SOAP-UI[11]）を導入し，クライアントからロードバランサ経由でサーバにリクエストを送信しサーバの受信性能を測定する．

送信するデータとしては，クライアントツール（SOAP-UI：groovy スクリプト）上で，主キー情報および時系列情報をランダムに生成し，所定時間（300 秒とした）の間送信し続ける．

実験内容として，２種類の性能実験を実施しており，実験２については，実験１で得られた性能測定結果のうち，最も性能の良かった状態を指定した実験方法としている．

実験環境の構成方針として，サーバ間（ゲスト OS 間）のネットワーク遅延を極小化するため，１台の筐体内にすべての関係サーバや機能を動作させる．各ゲスト OS やライブラリに関するスペックは表 5 の通りである．

- 実験１：クライアント台数とクライアントスレッド数の変化
  - WebAP サーバ台数：３台
  - クライアント台数：１～８
  - クライアントスレッド数：１０～１０００
- 実験２：サーバ台数の変化
  - クライアント台数：２台
  - クライアントスレッド数：２００
  - WebAP サーバ台数：１～５

表 5　実験環境

分類	種類	スペック等
筐体	モデル	Dell PowerEdge T630
	CPU	インテル® Xeon®プロセッサーE5 2600×４０個
	メモリ	３２GB
ホスト OS	OS	CentOS Linux release 7.2.1511
	CPU	４０個
	メモリ	4 GB
各ゲスト OS	OS	CentOS Linux release 7.5.1804
	CPU 割当	４個
	メモリ割当	4 GB
データ転送元クライアント	Web サービス SOAP クライアント	SoapUI-x64-5.5.0
ロードバランサ	ソフトウェアロードバランサ	Nginx-1.15.5[12]
WebAP サーバ	Java 実行環境	Java 7 （jdk1.7.0_79）
	Web コンテナ	Tomcat7.0.69
	Web サービスエンジン	Apache-AXIS2 1.7.1

6.2 実験結果

実験１および実験２における性能結果は以下の通りであり，いずれの実験においても，エラーおよびスラッシングは発生しておらず，各ノードは状態遷移に沿って処理していることを確認した．

- 実験１

  実行性能結果は表 6 の通り，サーバ３台構成で平均２，２３６件／秒・最高２，７０６件／秒の処理性能となった．

  クライアントマシン台数やクライアントスレッド数の増減による大きな変化は無い事から，本実験環境におけるサーバ処理性能の限界が確認できている事と考える．
- 実験２

  実行性能結果は表 7 および図 5 の通り，サーバ台数によりスケールする強い相関関係が確認できる．

ⓒ2019 Information Processing Society of Japan

Vol.2019-DBS-169 No.13
Vol.2019-IFAT-136 No.13
2019/9/10

表 6　実験 1 実行性能結果

		クライアントマシン台数							単位：TPS
		1	2	3	4	5	6	7	8
スレッド数/台	10	1,432.82	1,855.77	2,136.71	2,209.18	2,404.93	2,145.57	2,093.10	2,199.45
	20	1,954.56	2,098.95	2,367.38	2,365.35	2,383.17	2,450.98	2,248.17	2,194.20
	50	2,127.46	2,622.32	2,639.99	2,432.70	2,451.79	2,407.30	2,168.08	2,170.42
	100	2,345.28	2,548.73	2,418.11	2,471.96	2,354.39	2,348.98	2,111.46	1,981.77
	200	2,530.35	**2,706.88**	2,603.12	2,381.78	2,149.17	2,187.28	1,909.47	1,890.47
	300	2,574.65	2,669.74	2,541.81	2,424.57	2,263.66	2,144.97	1,891.82	1,815.19
	500	2,528.37	2,482.79	2,526.29	2,332.36	2,225.93	1,899.99	1,928.31	1,753.68
	1000	2,459.43	2,213.31	2,323.79	2,357.17	1,870.36	1,872.26	1,856.25	1,702.56

表 7　実験 2 実行性能結果

	サーバ台数				単位：TPS
計測回数	1	2	3	4	5
1 回目	1,028.15	1,919.82	2,706.88	3,002.89	3,543.02
2 回目	986.73	1,899.58	2,448.25	2,876.79	3,505.00
3 回目	1,042.04	1,969.85	2,624.88	2,888.64	3,525.16
平均	1,018.97	1,929.75	2,593.34	2,922.77	3,524.39

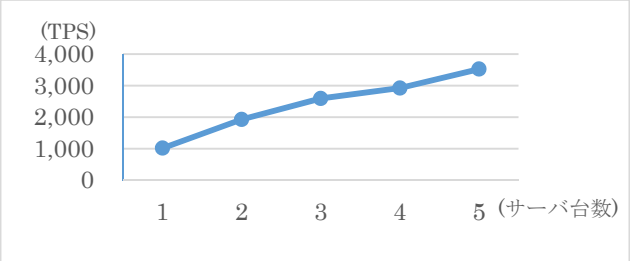

図 5　実験 2 実行結果グラフ

7. 考察

本研究で開発したシステムにより，順不同で到着するシーケンスデータを「主キー情報」のグループごとに「時系列情報」に基づく処理順序制約を守りながらリアルタイムに処理する事が可能である事を確認した．それにより，従来手法である定期的なポーリング処理（検索およびソート処理）を行う必要が無いため，無駄なハードウェアリソース消費を避けられるだけでなく，シーケンスデータのスピーディーな利活用がより広げられる事になると考える．

また，サーバ台数を水平方向に単純増加させる事で処理性能向上が見込めるため，ピーク時処理量やデータ量に応じた柔軟なソフトウェアアーキテクチャであると考える

さらに，一般的な Web アプリケーションのライブラリを用い，一般的なシステム間連携プロトコルである Web サービス SOAP を使っている事から，現在稼働中のシステムにも容易に組み込む事ができる事が明らかである．

8. おわりに

現時点では受信したシーケンスデータのキー情報をメモリ上に蓄積するため，定期的にハウスキープする処理を検討し実装する想定である．また，動的なサーバ台数の増減を可能とする仕組みにより，クラウド環境で動作させる事も可能としたい．

参考文献

[1] Dong-Hyuck Im,Cheol-Hye Cho,IlGu Jung. Detecting a large number of objects in real-time using apache storm. 2014 International Conference on Information and Communication Technology Convergence (ICTC).https://ieeexplore.ieee.org/document/6983306
[2] Dragan Stojanović ,Natalija Stojanović,Jovan Turanjanin. Processing big trajectory and Twitter data streams using Apache STORM. 2015 12th International Conference on Telecommunication in Modern Satellite, Cable and Broadcasting Services (TELSIKS). https://ieeexplore.ieee.org/document/7357792
[3] Liu Yan, Zhao Shuai, Cheng Bo. Multi-sensor data fusion system based on Apache Storm. 2017 3rd IEEE International Conference on Computer and Communications (ICCC).https://ieeexplore.ieee.org/document/8322712
[4] Özlem Hakdağli, Caner Özcan, İskender Ülgen Oğul. Stream text data analysis on twitter using apache spark streaming. 2018 26th Signal Processing and Communications Applications Conference (SIU).https://ieeexplore.ieee.org/document/8404540
[5] Streamingdata　https://aws.amazon.com/jp/streaming-data/
[6] Apache Storm　https://storm.apache.org/
[7] Apache Spark Streaming　https://spark.apache.org/streaming/
[8] Apache Hadoop https://hadoop.apache.org/
[9] SOAP https://www.w3.org/TR/soap/
[10] https://www.tjnet.co.jp/archive/17/2010-05?itemid=3180
[11] SOAP-UI https://www.soapui.org/
[12] Nginx https://nginx.org/en/

ⓒ2019 Information Processing Society of Japan