DEIM Forum 2018 C7-2

Flinkを用いた動的変更可能なIoT向けイベント処理基盤

中川　岳† 金政　泰彦† 山岡　久俊† 板倉　宏太† 高橋　英一†
植木　美和† 松本　達郎† 武　理一郎†

† 株式会社富士通研究所　〒 211–8588 神奈川県川崎市中原区上小田中 4-1-1
E-mail: †{nakagawa.gaku,kanemasa,yamaoka-h,itakura.kota,takahash,ueki.miwa,tatsuro,riro}@jp.fujitsu.com

あらまし　車や人などの実世界の監視対象からその状態の変化をイベントデータとして受け取り、リアルタイムに並列分散処理してサービスを提供するIoTシステムにおいては、イベントデータに対する処理内容を変更する際に、システム全体の処理を一時的に停止する必要がある。これは、システムを構成する分散ノード間で処理の一貫性を保つために同期処理が必要となるからである。数千から数百万の監視対象が数秒間隔で状態の変化を通知するような高スループットなIoTイベント処理基盤においては、このような処理の一時停止は処理のリアルタイム性を損なうため、許容されない。この問題を解決するために我々は、ストリームデータ処理エンジンであるApache Flinkを用いて、イベント処理を継続しながらイベントデータに対する処理内容を変更できるIoTイベント処理基盤を開発した。このIoTイベント処理基盤では、イベントデータに対する処理内容を取り替え可能なプラグインとして設定し、そのプラグインの変更指示をイベントデータと同じストリーム上でメッセージとして送付することにより、処理を一時停止することなくイベントデータに対する処理内容と処理フローの変更を実現する。本稿ではその設計、実装、性能評価について述べる。

キーワード　ストリームデータ処理、IoTデータ、並列分散システム

1. はじめに

自動車やスマートフォンなど実世界の多数のデバイスから高頻度で送られてくるイベントデータをリアルタイムに処理してサービスを提供するIoTシステムにおいては、イベントデータを処理するプログラムを動的に変更する必要に迫られることがある。これは、車の流れや群衆の挙動などの実世界で発生するイベントデータの内容が、システムを設計する段階においては明らかでなく、サンプル調査の結果や仮説に基づいて設計せざるを得ないからである。このようにして設計されたIoTシステムでは、システムを実際に動かし始めた後に、収集されたデータを観察した結果として、処理内容の一部を変更する必要が生じたり、新しいサービスのアイデアを発見して処理内容を追加することが行われる。

このような高スループットなイベントデータをリアルタイムに処理する基盤としては、Storm [1] や Flink [2] などの分散ストリーム処理エンジンが処理性能の面で適している。その中でもFlinkは任意のステートを分散ノードで保持できるので、過去のイベント履歴や、その結果としての現在の状態値に応じたサービスを提供する用途に特に適している。しかしながら、Flinkを始めとした既存の分散ストリーム処理エンジンでは、処理内容の一貫性を保ちながら処理内容の変更を行おうとすると、処理のリアルタイム性を保てないという問題がある。この問題は、分散ノード間において、処理プログラムの変更の一貫性を取るためにタイミング同期処理が必要となる為に発生する。ここでいう処理内容の一貫性とは、あるノードである時点でイベントデータに対する処理内容が変更された場合には、他の全てのノードでも全く同一の時点で同様に処理内容が変更されることを意味し、これはビジネス向けのIoTシステムではしばしば必要となる要件である。

この問題を解決するために我々は、高頻度で流入する大量のイベントデータの処理を停止させることなく、その処理内容を動的に変更することを可能とするIoT向けイベント処理基盤を開発した。開発したIoT向けイベント処理基盤は、イベントデータに対する処理内容をプラグインとして別途記述し、それをリストの中に登録する構造を採用している。このプラグインをイベントデータの持つ時刻情報を利用して分散ノード間で一貫性を保って有効化・無効化することで、イベントデータを処理しながらの無停止での処理内容の変更を実現している。我々はこのIoT向けイベント処理基盤をFlinkを利用して実装した。この実装を用いて、基礎的な処理性能を測るために評価を行い、100万台の自動車が1秒間隔で送ってくるGPS座標等のイベントデータ（100万イベント毎秒）をサーバ4台で遅延なく処理できることを示した。

本稿の構成は以下の通りである。2. では、本稿で議論する、IoTイベント処理基盤に関する技術的課題について述べる。3. では、我々が開発したIoTイベント処理基盤について述べる。4. では、開発したIoTイベント処理基盤の評価実験について述べる。5. では、本稿をまとめる。

2. 技術的課題

既存の分散ストリーム処理エンジンにおいて、処理内容の一貫性を保ちながら処理プログラムを動的に変更するには、データ処理の一時的な停止が伴い、イベントデータ処理のリアル

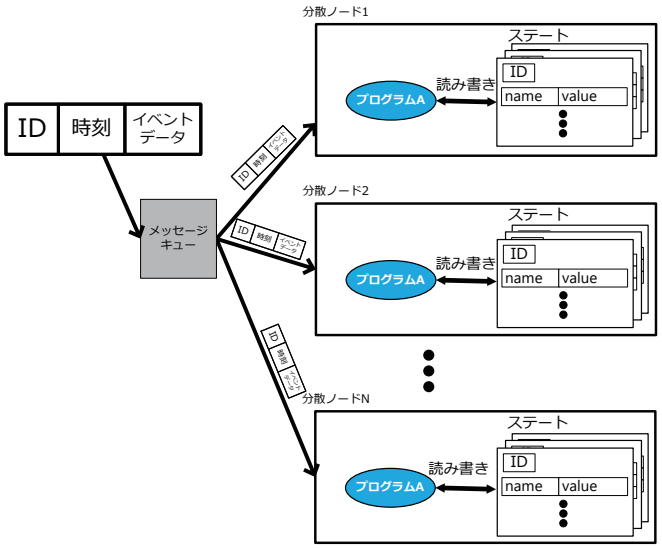

図 1　想定するストリームデータ処理システム

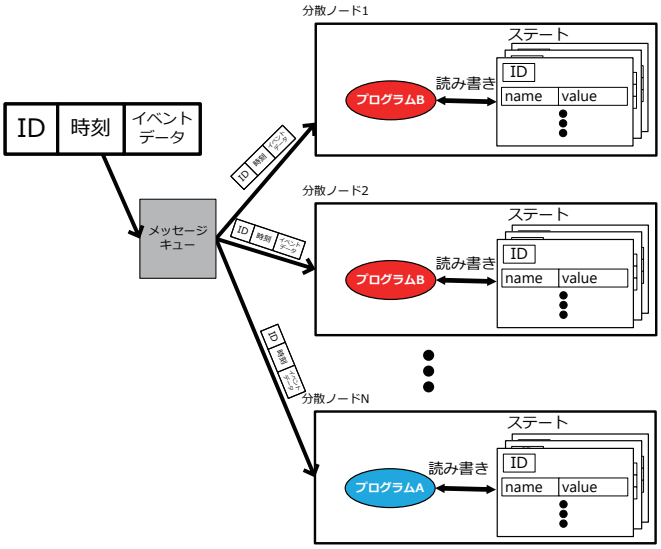

図 2　新旧処理プログラムが混在する状況

タイム性が損なわれる。ここでの処理内容の一貫性とは、あるノードである時点でイベントデータに対する処理内容が変更された場合には、他の全てのノードでも全く同一の時点で同様に処理内容が変更されることである。1. で述べたように、本稿が議論の対象とする IoT システムでは、高頻度で到着するイベントデータを処理しながら、処理内容を変更することが求められるため、これは大きな問題となる。

分散ノード間で処理内容の一貫性を保って処理プログラムを変更するためには、イベントメッセージの処理を一時的に停止して、変更するタイミングの同期を取る必要がある。ここでは例として、図 1 に示すような、N 個の分散ノードで構成されるストリーム処理システムを考える。このシステムを構成する、それぞれの分散ノードにはプログラム A が配備されており、入力されるデータを処理し、何らかの出力を得るものとする。この状況で、プログラム A をプログラム B に変更することを考える。もし、タイミングを同期せずに、分散ノードで動作するプログラムを変更すると、図 2 に示すように、変更前の処理プログラムと変更後のプログラムが分散ノードの間で混在する状況が発生する。このような混在した状況でデータを処理すると、処理の一貫性が保てなくなり、想定しない動作をする。これを防ぐためには、メッセージ処理を一時的に停止して、分散ノード間で処理プログラムを変更するタイミングを同期する必要がある。

処理プログラムを変更する方法としては、変更後のプログラム（処理プログラム B）が動作するストリーム処理システムを、変更前のプログラム（処理プログラム A）が動作するストリーム処理に対して並列に起動し、あるタイミングからイベントデータの処理を切り替える方法も考えられる。この方法では、分散ノード間での変更タイミングの同期は不要になる。しかしながら分散ノードにおける処理が、その内部的に持つ状態に依存する場合は（ステートフル処理）、そのステートを新しい環境に複製する必要がある。ステート複製の一貫性を取るためには、ステートの変更を抑制した状況で複製処理を行う必要があるため、メッセージ処理を一時的に停止する必要がある。我々が想定するイベント処理は、ステートフル処理を前提としているため、この方法でも処理の一時停止を排することはできない。

3.　動的変更可能な IoT イベントデータ処理基盤

本節では 2. で述べた問題を解決するために開発した、IoT イベントデータ処理基盤について述べる [3]。このイベント処理基盤では、処理プログラムを分散ノードで複数保持可能にし、イベントデータの持つタイムスタンプに同期した切り替えを行う。これにより、変更する処理プログラムを順次配信した後に、イベントデータ処理を一時的に停止することなく、処理の一貫性を保って処理プログラムの変更を行うことができる。

図 3 に開発したシステムの構成図を示す。システムに入力されたイベントデータはまずメッセージキューに格納される。このメッセージキューに格納されたイベントデータを、ストリームデータ処理エンジンを用いて処理する。

3.1　デバイスから届くイベントメッセージの処理

この IoT イベントデータ処理基盤は、IoT デバイスから生成された JSON 形式 [4] のイベントメッセージを受け取る。それぞれのイベントメッセージには、IoT デバイスを区別する識別子、イベントメッセージが生成された時刻、IoT デバイスの状態を表現する任意の数のイベントデータから構成される。ここでは例として、図 4 のような、自動車が生成するイベントメッセージを考える。このイベントメッセージには、自動車の識別子（id）、メッセージが生成されたタイムスタンプ（time）、自動車の速度（speed）、自動車位置の緯度と経度（lat、lon）が含まれる。

イベントデータは、それに含まれる識別子をキーとして、分散ノードに配信され、その内容に基づき、識別子ごとに区別される内部状態（ステート）を変更する。また変更したステートに、後述するプラグインが設定されている場合は、そのプラグインを実行する。識別子ごとに区別されるステートは、オブジェクトテーブルで管理される。またステートは、それぞれ独

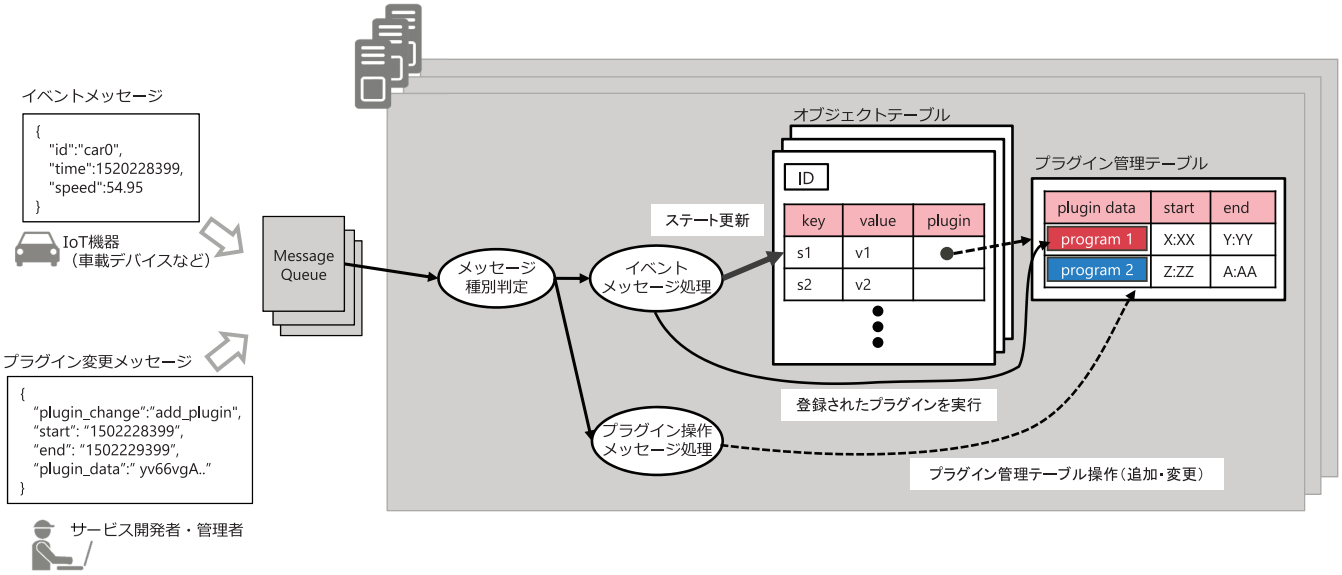

図 3　我々が開発した IoT イベントデータ処理基盤

```
{
  "id": "car0",
  "type": "event",
  "time": "1513207500",
  "lat": "43.29954323100655",
  "lon": "168.62391592425027",
  "speed": "10.0"
}
```

図 4　イベントメッセージの例

立したプラグイン管理テーブルへの参照を持つ。

3.2　プラグインの追加・変更による処理フローの動的変更

プラグインは任意のステートの変更に対応づけられて呼び出されるプログラムである。本システムでは、このプラグインをステートに対して設定することで、イベントデータを利用した任意の処理を実行する。プラグイン管理テーブルは、プラグインが適用開始される時刻（開始時刻）、適用終了される時刻（終了時刻）、プラグインの処理内容を規定するプログラムを保持している。この開始時刻と終了時刻の組み合わせを有効期間と呼ぶ。イベントメッセージを処理する際に、ステートを変更すると同時に、到着したイベントメッセージが持っている時刻情報、プラグイン管理テーブルに含まれる対象ステートと有効期間を照合し、そのイベントメッセージに対して有効であるプラグインを探索する。このとき、有効であるプラグインが登録されていれば、メッセージをパラメータとして、そのプラグインを実行する。

プラグイン管理テーブルの内容の変更は、特別なイベントメッセージ（プラグイン操作メッセージ）を分散ノードに送信することで行う。例えばプラグインの追加を行う場合は、プラグインを設定する IoT デバイスの識別子、設定するステートの名前、追加するプラグインのプログラム、その適用開始時刻、適用終了時刻を含んだイベントメッセージを送信する。プラグインの有効期限を変更する場合は、変更を適用する IoT デバイスの識別子、対象となるプラグインを特定するための情報（プラグインのプログラムのハッシュ値や一意の ID など）、その適用開始時刻、適用終了時刻を含んだプラグイン操作メッセージを送信する。これらのプラグイン操作メッセージの内容に基づいて、プラグイン管理テーブルの内容を変更する。

3.2.1　プラグインの配信

プラグインは、そのプログラムをテキスト符号化したデータを直接送信することで分散ノードに配信する。プラグインを配信する手段としては、処理系からアクセス可能なサーバにプラグインの実行形式ファイルを設置し、プラグイン操作メッセージにそのアクセスパスを含め、分散ノードからダウンロードする方法が考えられる。この方法では、複数の分散ノードからリポジトリにアクセスが集中し、プラグイン配信の遅延が大きくなる可能性がある。それを防ぐために、この IoT イベント処理基盤では、プラグインのプログラムを通常のメッセージと同じ経路で送信する。

3.2 節で述べたように、ステートに対するプラグインの設定は、プラグイン操作メッセージをイベント処理基盤に送信することで指示する。このプラグイン操作メッセージに含まれるテキスト符号化されたプラグインは、プラグイン追加処理時に復号され、実行可能な形式でメモリ上に読み込む。プラグイン管理テーブルへは、その実行可能なプログラムへのメモリ参照を登録する。

3.2.2　メッセージ時刻に同期したプラグインの切り替え

ステートに関連付けられたプラグインのうち、どのプラグインを実行するかは、それぞれのプラグインに設定された有効期間を用いて判断される。つまりこの特徴を用いると、ある特定の時刻より後に生成されたイベントメッセージの処理に関して、全ての分散ノードで同期して新しい処理プラグインを適用することが可能になる。

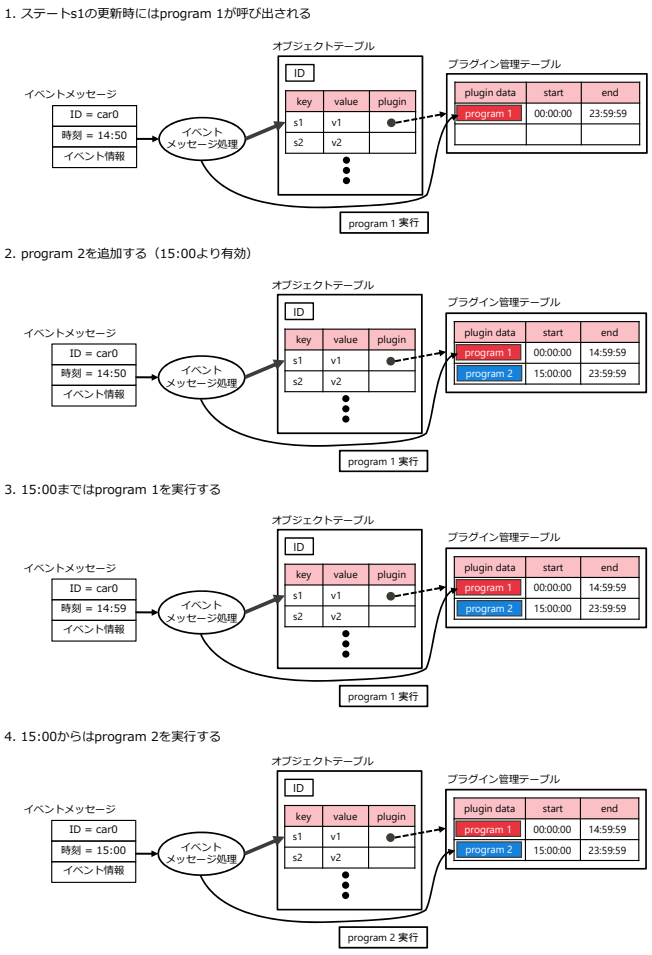

図 5　有効期間に基づいた処理内容の変更

ここでは例として、分散ノードで処理を担当するプラグインを時刻 15:00 から変更することを考える。図 5 は、IoT イベント処理基盤を構成する分散ノードのうち、任意の 1 ノードでのプラグインの変更フローについて着目したものである。なお、この例示においては、メッセージが持つタイムスタンプの最小単位は 1 秒であるとする。

現在、時刻は 14:50 であり、program 1 がステート s1 の変更に対応づけられているとする（図中番号 1）。ここで、プラグイン管理テーブルを変更して、時刻 15:00 より有効になる program 2 をプラグイン管理テーブルに登録する。このとき、program 1 の適用終了時刻を、program 2 が有効になる 1 秒前である、14:59:59 に書き換える（図中番号 2）。この状況では、14:59:59 の時刻情報を持つイベントデータの処理までは program 1 が実行される（図中番号 3）。15:00 以降の時刻情報を持つイベントデータが到着すると、program 2 が実行される（図中番号 4）。この例で示したように、分散ノードで複数のプラグインを保持し、イベントデータが持つ時刻情報を利用した同期を行うことで、イベントデータの処理を一時的に停止することなく、一貫性を保った処理プログラムの変更が可能になる。

3.3　ストリームエンジン Flink を用いた実装

我々は Apache Kafka [5] と Apache Flink [2] を用いて IoT イベント処理基盤を実装した。このシステムでは、IoT デバイスから送信されたイベントデータを Kafka で受信し、Flink を用いてその処理を行う。

3.3.1　オブジェクトの実装

我々が実装する IoT イベント処理基盤では、オブジェクトテーブルと、プラグイン管理テーブルの組み合わせを管理し、それに基づいて IoT デバイスから送信されるイベントメッセージを処理する。ここでは、IoT デバイスの識別子ごとに区別されるオブジェクトテーブルとプラグイン管理テーブルの組み合わせをオブジェクトと呼称する。

本実装では、Flink が備える KeyedStream [6] と、Process Fuction [7] を用いてこれを実現した。Flink では KeydStream を用いることで、入力するメッセージに含まれる特定のデータをキーとして、そのキーごとに異なるコンテキストを保持してメッセージ処理を行うことができる。ProcessFunction は KeyedStream に対する処理を定義するための機構である。本実装では、メッセージに含まれる IoT デバイスの識別子をキーとして KeyedStream を構成し、それに対する処理として ProcessFunction を用いた処理を実装した。この定義した処理は、3.1、3.2 節で述べた、イベントデータ処理、プラグインの実行、プラグインの操作である。

このシステムでは、IoT デバイスの識別子ごとに独立したオブジェクトテーブルとプラグイン管理テーブルを管理する必要がある。本実装では、Process Function で利用できる ValueState [6] を利用して実装した。ValueState を利用することで、IoT デバイスの識別子ごとに独立した任意のデータ構造をメモリ上に保持することができる。

3.3.2　分散チェックポイントによるステートの不揮発化

3.3.1 節で述べたように、本実装では IoT デバイスのステートをメモリ上に保持する。そのため、Flink プログラムを実行しているプロセスが異常停止したり、分散ノードがシステム障害を起こすと、ステートを失う可能性がある。

このステート消失を防ぐために、本実装では Flink が備えるチェックポイント機構を利用する。この機構は、Flink のメモリ上に保持されたコンテキストを定期的に外部システムに不揮発化し、システム異常時には、その不揮発化したコンテキストからシステム状態を復元し、処理を継続する。復元時には、システムの状態が過去に戻るため、そこから現在に至るまでのメッセージを再送する必要がある。これは Flink と Kafka の連携機能によって実現される。

Flink が備えるチェックポイント機構では、その処理を分散ノード間で非同期に行うために、分散チェックポイント方式が採用されている。この方式では、イベントメッセージの合間に、特別なメッセージ（バリアメッセージ）が挿入され、それを受け取った分散ノードは、それぞれが持つコンテキストと、バリアメッセージの識別子を指定された外部ストレージへ保存する。ある識別子を持つバリアメッセージに対して、すべての分散ノードがコンテキストの保存を完了すると、そのバリアメッセージが挿入された時点でのシステム内部状態の不揮発化が完了する。

本実装ではオブジェクトテーブルとプラグイン管理テーブルを Flink のコンテキストとして保持する。そのため、このチェッ

クポイント機構によりシステム異常時にも内部状態を復旧することができる。チェックポイントを保存する外部ストレージとしては、分散ファイルシステムである HDFS [8] を利用する。HDFS を構成する計算機は、Flink が動作する計算機と独立している。

3.3.3 分散チェックポイント処理と連動したプラグインの削除

3.2 節で述べたように、我々が開発した IoT イベント処理基盤では、プラグイン機構により、イベント処理のふるまいを動的に変更することができる。プラグインはそれぞれ有効期間を持っており、時間の経過に伴い、実行されることがないプラグインも出てくるため、これを削除する必要がある。

有効期限を超過したプラグインの削除は 3.3.2 節で述べたチェックポイント処理と同期して行う。本実装では Flink のチェックポイント機構を利用したステートの不揮発化が行われ、システム異常時にはシステムの内部状態が過去の状態に復元される。そのため、プラグインの削除を任意のタイミングで行うと、内部状態の復元時に存在すべきプラグインのプログラムがメモリ上に存在しない可能性がある。これを防ぐため、本実装ではチェックポイント処理が完了したときに、有効期限を超過したプラグインの探索と削除を行う。

4. 評　　価

開発した IoT イベント処理基盤の性能評価のために、IoT デバイスからのイベントメッセージの入力をシミュレーションし、そのイベントメッセージを処理する実験を実施した。本実験では IoT デバイスとして、自動車に搭載され、その自動車の座標、速度などを取得し、ネットワーク経由でイベント処理基盤に送信する自動車 IoT デバイスを想定する。データ送信の間隔は、1 秒とする。

実験環境を図 6 に示す。図中の Slave 1、Slave 2、Slave 3、Slave 4 はイベントデータを処理する分散ノードである。Master は Slave 1、Slave 2、Slave 3、Slave 4 を管理するためのノードである。Master はイベントデータの処理を担当しない。Slave 1、2、3、4 と Master は 1Gbps の最大帯域を持つネットワークを経由して通信を行う。Sim 1、Sim 2 では、自動車 IoT デバイスからのイベントデータ送信を模倣するシミュレータが動作する。イベントシミュレータは任意の数の自動車 IoT デバイスの動作を並行シミュレーションし、イベント処理基盤に送信する。本実験では、それぞれの自動車 IoT デバイスが 1 秒ごとにイベントデータを送信する状況を想定する。Sim 1 は Slave 1、Slave 2 に Sim 2 は Slave 3、Slave 4 へイベントデータを送信する。イベントシミュレータが動作する計算機とイベント処理基盤は、Infiniband（FDR）で接続されており、高スループットなイベントデータを遅延なく送信することができる。なお本実験では、Infiniband を用いて TCP/IP 通信を行う TCP/IP over InfiniBand を用いて通信を行った。

評価には Flink に実装されている遅延追跡機構を利用した [9]。この機構は、定期的にイベントデータの流れの中に遅延計測のための特別なデータを挿入する（遅延計測マーカー）。この遅延計測マーカーには、それぞれフローに挿入されたタイムスタンプが含まれる。挿入された遅延計測マーカーは、イベントメッセージと同じフローを通過する。遅延計測マーカーが最後段での処理に到着すると、その到着時刻とマーカーに含まれる挿入時刻の差分を記録する。遅延計測マーカーは、前後のイベントメッセージとの位置関係が変化しないように工夫されて実装されている。そのため、イベントメッセージの処理が滞れば、遅延計測マーカーの通過時間も長くなる。本実験では、遅延計測マーカーの通過時間をイベントデータの処理フローを通過するために必要な時間とみなす。遅延計測マーカーの挿入間隔は 1 秒とした。

以上の環境において、自動車 IoT デバイスの数を 100、1000、1 万、10 万、100 万として 60 秒間のイベント処理を行う実験を実施した。図 7 に実験結果を示す。このグラフは、それぞれの実験条件（シミュレーションする自動車の台数）ごとに、遅延計測マーカーの平均通過時間を示したものである。分析の結果、遅延計測マーカーの平均通過時間は最小で 201.5 ミリ秒、最大で 218.0 ミリ秒であった。この結果より、我々の開発した IoT イベント処理基盤は、100 万規模の対象から 1 秒ごとに送信されるイベントデータを処理するのに十分な性能をもっていることがわかった。

5. ま と め

ストリーミング処理エンジンを用いて IoT イベントデータを処理するシステムにおいては、処理プログラムの変更に処理の一時停止が伴うため、頻繁な処理プログラムの変更を行うと、イベントデータ処理のリアルタイム性が損なわれる課題がある。

その課題を解決するために、我々はイベントデータ処理の一時停止を伴わずに、処理内容と処理フローを変更可能な IoT イベントデータ処理基盤を開発した。この処理基盤では、分散ノードで複数の処理プログラムをプラグインとして保持すること、イベントメッセージの持つタイムスタンプを利用したタイミング同期を取ることで、データ処理の一貫性を保ちながら、処理プログラムの変更を実現する。開発したイベントデータ処理基盤を用いて、基礎的な処理性能を測るための評価を行い、100 万台の自動車が 1 秒間隔で送ってくる位置座等のイベントデータ（100 万イベント毎秒）をサーバ 4 台で遅延なく処理できることがわかった。

文　　献

[1] Apache Storm. http://storm.apache.org/.

[2] Apache Flink: Scalable Stream and Batch Data Processing. https://flink.apache.org/.

[3] 山岡久俊, 中川岳, 板倉宏太, 高橋英一, 金政泰彦, 植木美和, 武理一郎. 組み立て型のサービス開発を実現する IoT 向けイベント処理基盤の提案. DEIM2018 論文集, 2018.

[4] RFC 7159 - The JavaScript Object Notation (JSON) Data Interchange Format. https://tools.ietf.org/html/rfc7159.

[5] Apache Kafka. https://kafka.apache.org/.

[6] Apache Flink 1.4 Documentation: Working with State. https://ci.apache.org/projects/flink/flink-docs-release-1.4/dev/stream/state/state.html.

[7] Apache Flink 1.4 Documentation: The Process Fuction.

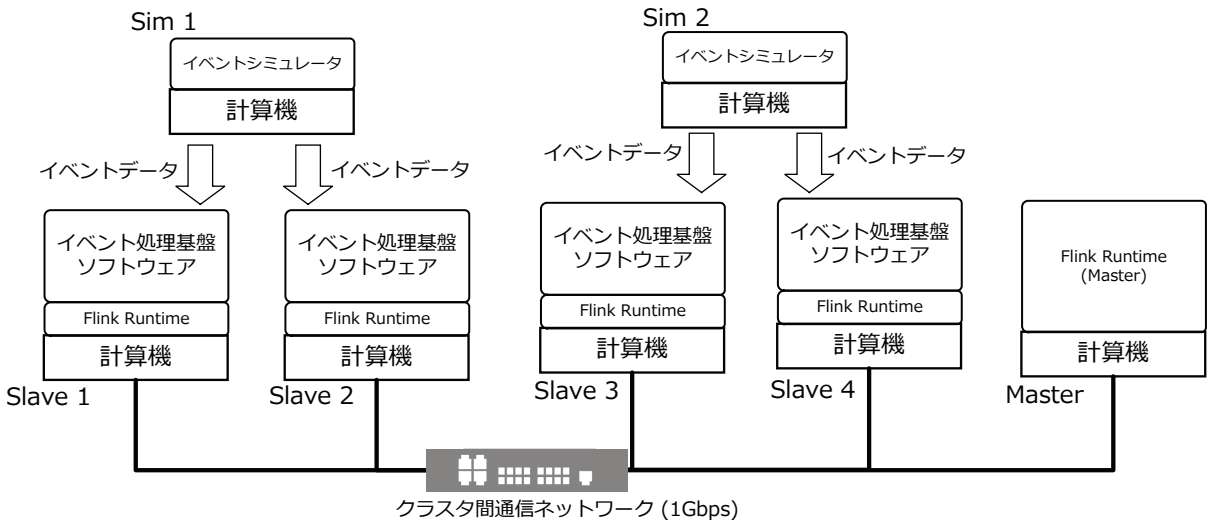

図 6　実験環境の構成図

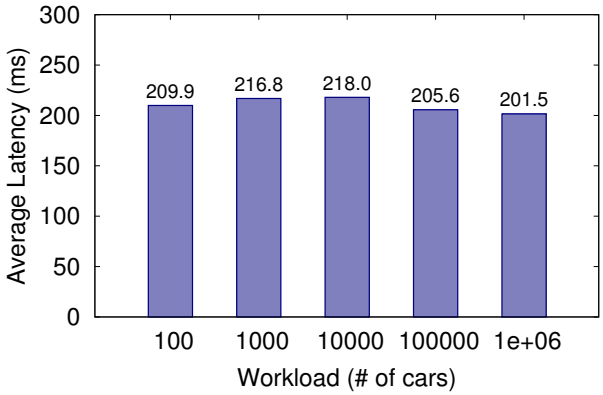

図 7　イベント処理性能の評価

https://ci.apache.org/projects/flink/flink-docs-release-1.4/dev/stream/operators/process_function.html.

[8] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, MSST '10, pp. 1–10, 2010.

[9] Apache Flink 1.4 Documentation: Metrics. https://ci.apache.org/projects/flink/flink-docs-release-1.4/monitoring/metrics.html.