Vol.2019-OS-147 No.11
2019/7/25

SINET広域データ収集基盤のための基盤ソフトウェアの検討

孫 静涛[1,a)]　藤原 一毅[1,b)]　竹房 あつ子[1,2,c)]　長久 勝[1,d)]　吉田 浩[1,e)]　合田 憲人[1,2,f)]

概要：モバイルネットワークを介して IoT (Internet of Things) デバイスから収集された大量の情報をクラウドに収集，蓄積，解析する高度なデータ処理への期待が高まっている．国立情報学研究所では，モバイル網を SINET の足回りとして活用する「広域データ収集基盤」の実証実験を開始し，各種携帯端末から SINET に直接接続できるようにした．我々は，SINET 広域データ収集基盤を活用するための基盤ソフトウェアの開発とパッケージの公開を進めている．しかし，既存パッケージでは Linux ベースセンサ端末とメッセージング基盤 Apache Kafka を前提としたもので既存の多様なセンサに対応していない，また，クラウドでの処理は分散ストリーム処理に対応しておらず大量データの処理には対応できないという課題がある．そこで，本研究では，Android 端末を用いて様々なセンサ情報に対応した IoT システムプロトタイプを新たに構築するとともに，既存分散ストリーム処理基盤のリアルタイム処理性能を調査する．

キーワード：広域データ，インフラストラクチャ，クラウドコンピューティング，ストリーム処理，モバイルアプリケーション，IoT

A Study of Foundation Software for SINET Wide-area Data Collection Infrastructure

Jingtao Sun[1,a)]　Ikki Fujiwara[1,b)]　Atsuko Takefusa[1,2,c)]　Masaru Nagaku[1,d)]
Hiroshi Toshida[1,e)]　kento Aida[1,2,f)]

1. はじめに

機械学習技術の急速な普及と第 5 世代移動通信システム「5G」の実用化を目前に控え，モバイルネットワークに接続された各種センサから取得された大量な情報をクラウドに収集，蓄積し，活用する IoT (Internet of Things) のための高度なデータ処理への期待が高まっている．国立情報学研究所（NII）では，日本国内の大学，研究機関等の学術情報基盤として高性能な学術情報ネットワーク SINET を提供しており，SINET を活用した高度な IoT 研究の促進を目的として 2018 年度に「広域データ収集基盤」の実証実験を開始した [1]．SINET 広域データ収集基盤とは，SINET への新たなアクセス環境として，モバイル網を SINET の足回りとして活用した広域的な基盤であり，SINET 接続用の SIM をセンサ端末に装着することで，その端末から商用インターネットを介することなく SINET に直接接続することができる．SINET では，SINET SIM を装着したセンサ端末群と大学等のオンプレミス計算環境や商用クラウド内の計算資源間を L2VPN (Virtual Private Network) の閉域網で接続することが可能となり，安全かつ大容量のデータ通信環境を提供することができる．

我々は，SINET 広域データ収集基盤を活用した IoT 研究の実現可能性を示すため，リアルタイムビデオ解析処理のプロトタイプシステムを構築して実証実験を行った [2], [3].

[1] 国立情報学研究所, 101-8430, 東京都千代田区一ツ橋 2 − 1 − 2
[2] 総合研究大学院大学, 240-0193, 神奈川県三浦郡葉山町湘南国際村
[a)] sun@nii.ac.jp
[b)] ikki@nii.ac.jp
[c)] takefusa@nii.ac.jp
[d)] mnagaku@nii.ac.jp
[e)] h-yoshida@nii.ac.jp
[f)] aida@nii.ac.jp

実証実験では，カメラを搭載したセンサ端末と商用クラウド，オンプレミス環境を SINET のモバイル網と L2VPN で接続して，センサ端末で収集された画像データをまずオンプレミス環境および商用クラウド環境に収集し，オブジェクトストレージに蓄積するとともに，クラウドの GPU ノードを用いてリアルタイム解析処理を実施した．実験環境の構築には，学認クラウドオンデマンド構築サービス [4], [5] を用いた．また，実証実験で用いたソフトウェアパッケージは Github で公開している [3].

しかしながら，本実証実験では Linux ベースセンサ端末とメッセージング基盤 Apache Kafka[6], [7], [8]（Kafka）を前提としたもので既存の多様なセンサに対応していない，クラウドでの処理は分散ストリーム処理に対応しておらず大量データの処理には対応できないという課題がある．分散ストリーム処理基盤としては，Apache Flink[9]（Flink）や Apache Storm[10]（Storm）等複数開発されているが，それらでどの程度のリアルタイムデータ処理が可能であるか明らかでない．

本研究では，Android 端末で利用可能な様々なセンサ情報に対応した IoT システムプロトタイプを広域データ収集基盤上に新たに構築するとともに，既存分散ストリーム処理基盤のリアルタイム処理性能を調査した．Android ベース IoT システムプロトタイプでは，Android で取得可能なセンサ情報を軽量な Pub/Sub 型メッセージングプロトコルの MQTT (Message Queue Telemetry Transport) [11] でオンプレミス環境に送信，収集し，Kafka を介してクラウド上のデータベースに収集する仕組みを実装した．これにより，様々なセンサ情報の活用が可能であることを示す．また，既存分散ストリーム処理基盤の評価では，Flink と Storm を用いたクラスタ環境をそれぞれ構築し，その上でセンサデータ量と処理遅延とスループットの関係を明らかにする．

2. SINET 広域データ収集基盤を用いたリアルタイム処理実験の概要

SINET 広域データ収集基盤を用いたリアルタイムビデオ処理実証実験を行った [2]．本節では，実証実験の概要と実験に用いたソフトウェアの構成，実験環境について述べる．

2.1 実証実験

学術情報ネットワーク SINET5 とモバイル通信を直結したサービス「広域データ収集基盤」では，SINET 接続用の SIM を装着したセンサ等が取得したデータを，SINET の L2VPN で接続されたクラウド等の計算機を用いて安全に収集，解析することができる．モバイル通信を活用することで，これまで SINET が接続できなかった場所での研究データの収集や IoT 関連研究が可能になる．

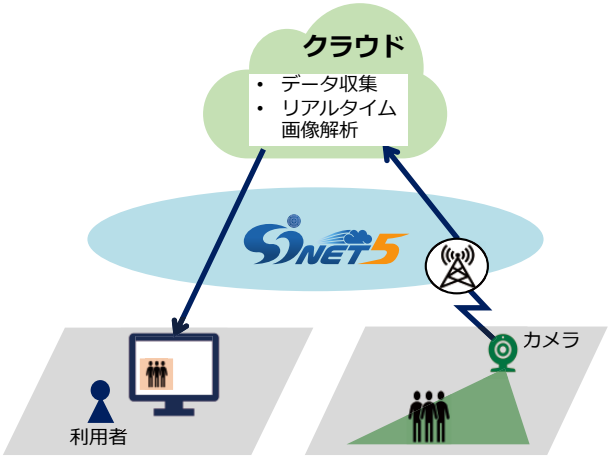

図 1: SINET モバイル網を用いたリアルタイムビデオ処理実証実験の概要．

図 2: 実験結果のスナップショット．

図 1 に実証実験の概要を示す．実験では，センサ端末のカメラで取得した動画像から複数の静止画を生成し，その静止画を順次 SINET のモバイル網経由でクラウド上のメッセージング基盤に送信・収集し，そのリアルタイム画像解析処理ができることを示した．センサ端末，クラウド上のメッセージング基盤，リアルタイム画像解析処理を行う GPU ノードはすべて SINET の L2VPN を用いて安全に接続することができている．

図 2 に実証実験実施時の利用者端末のキャプチャ画面を示す．図 2 の右側上下に並んでいる画像は，センサ端末のカメラで取得した動画がクラウド上のメッセージング基盤を介して利用者端末にストリーミング配信された結果が表示されている．右側上には NII 周辺の屋外を移動しているセンサ端末のカメラ画像，右側下には会場のセンサ端末のカメラ画像が表示されている．また，左側上下に並んでいる画像は，処理前画像をメッセージング基盤から受け取り，クラウドで何らかの処理をした処理後画像を，メッセージング基盤を介して配信して利用者端末で出力された結果となっている．ここで，左側上の画像は YOLO v3[12], [13] を用いてオブジェクト抽出を行った結果，左側下の画像は OpenPose[14] を用いて人のキーポイントを抽出した結果が表示されている．

モバイル網を利用して画像を送信する場合，その実効通

信帯域が課題となる．実験では，センサ端末で 320×240 画素，64KB 弱の静止画に変換し，フレームレートを 6fps (flame per second) とした．クラウドでは，YOLO v3 および OpenPose の各処理に対してそれぞれ GPU ノードを 1 台ずつ用いた．

2.2 プロトタイプシステムのソフトウェア構成

リアルタイムビデオ処理実験で用いたプロトタイプシステムの概要を図 3 に示す．図 3 の右側にあるセンサ端末から SINET のモバイル網を経由して静止画のストリーミングデータをメッセージング基盤に送信する．メッセージング基盤に到着したデータは，オンプレミスおよびクラウドのオブジェクトストレージにも格納することができる．図 3 左上にあるオンラインアプリケーションプログラムでは，メッセージング基盤に到着したストリーミングデータを順次受け取り，静止画を処理してその結果をメッセージング基盤に送信する．図 3 左下のオフラインアプリケーションプログラムでは，適宜オブジェクトストレージに格納されたデータをまとめて画像処理する．右下の利用者端末で，メッセージング基盤に格納された処理前および処理後の静止画ストリーミングデータを出力する．

本システムを構成するソフトウェアを，以下に示す．

1） メッセージング基盤プログラム
2） センサ用プログラム
3） オンラインアプリケーションプログラム
4） オフラインアプリケーションプログラム
5） ストリーミング画像出力プログラム

各ソフトウェアについて以下で説明する．

1）メッセージング基盤プログラムには，Apache Kafka（Kafka）を用いた．Kafka は，Pub/Sub 形式のメッセージング基盤の一つであり，センサ等の Producer から送信されるメッセージを Broker で一時的に収集、永続化し、データ処理プログラム等の Consumer に提供する．Broker はクラスタ構成をとることで，スケールアウトが可能になっている．個々のストリーミングデータは Topic と呼ばれるカテゴリ単位で管理され，Topic 名，値，タイムスタンプからなるレコードとして送信，格納される．また，Kafka Connect と呼ばれるツールによりオブジェクトストレージやストリーム処理系，バッチシステム等との連携が可能である．実験では，S3 Connector を利用して収集したデータを Amazon Web Services (AWS) の Simple Storage Service (S3) および NII のオンプレミス環境の S3 互換オブジェクトストレージにそれぞれ格納している．

2）センサ用プログラムは，Kafka の Producer として動作する．センサ端末には Raspberry Pi とカメラモジュールを用い，USB 接続したモバイルルータを用いて SINET L2VPN 経由で Kafka Broker に静止画を繰り返し送信するようにした．

3）オンラインアプリケーションプログラムでは，YOLO v3 の PyTorch 実装と OpenPose を用いてそれぞれ GPU ノード上で画像を処理した．各オンラインアプリケーションプログラムでは，Kafka の Consumer および Producer の機能を利用している．また，オブジェクトストレージに格納された複数静止画をまとめて処理する 4）オフラインアプリケーションプログラムも用意した．

利用者端末では，処理前および処理後静止画のストリーミングデータを動画像として表示させる 5）ストリーミング画像出力プログラムを用意した．ストリーミング画像出力プログラムもまた Kafka Consumer プログラムであり，Broker から受け取ったデータを順次利用者端末に出力することができる．

2.3 実証実験環境

図 4 に実証実験環境を示す．実験では, SINET の L2VPN で SINET モバイル網，VCP 東京サイト，オンプレミスの NII 千葉サイト，AWS 東京サイトを接続し，閉域網を構築している．図中のセグメント 1 から 4 は，閉域網内のネットワークセグメントを表しており，すべてプライベートアドレスが設定されている．VCP 東京サイトの仮想ルータを用いて，これらのネットワークのルーティング設定を行った．今回利用した仮想ルータは，「学認クラウドオンデマンド構築サービス」で提供しているものである．

実証実験では，2.2 節のソフトウェアを以下のように配備した．センサ端末の Raspberry Pi 2 台は，SINET のモバイル網に接続させた．NII 千葉サイトでは，3 台クラスタ構成の Kafka Broker と S3 互換オブジェクトストレージを配置した．AWS サイトでは，S3 を利用するとともに GPU ノードを必要に応じて確保してオンラインおよびオフラインアプリケーションプログラムを配備した．一方，操作性を考慮して利用者端末はパブリックインターネット経由で利用できるようにした．

3. Android 実装

実証実験では Linux ベースセンサ端末とメッセージング基盤 Kafka を前提としたもので，既存の多様なセンサに対応していない．本稿では，手軽に利用できる Android 端末をモバイル IoT デバイスとして活用するため, Android 端末から利用可能な様々なセンサ情報を Eclipse Paho[15] という MQTT 通信ライブラリを用いて，IoT システムプロトタイプを広域データ収集基盤上に新たに構築した．プロトタイプシステムでは，収集されたセンサ情報を Elasticsearch 上に蓄積し，それらのデータを Kibana というツールで可視化した結果を示す．

3.1 システム構成

前章に述べたように本研究では，SINET の L2 VPN で

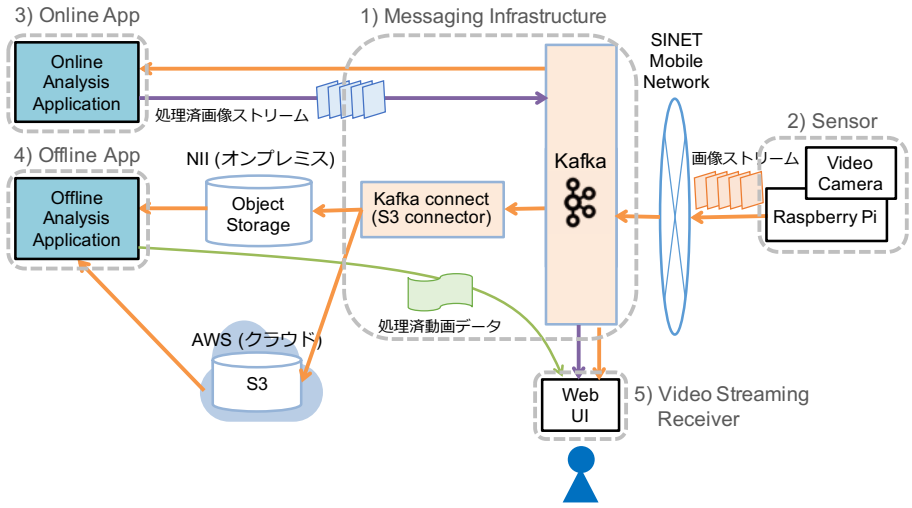

図 3: リアルタイムビデオ処理機構プロトタイプシステムのソフトウェア構成.

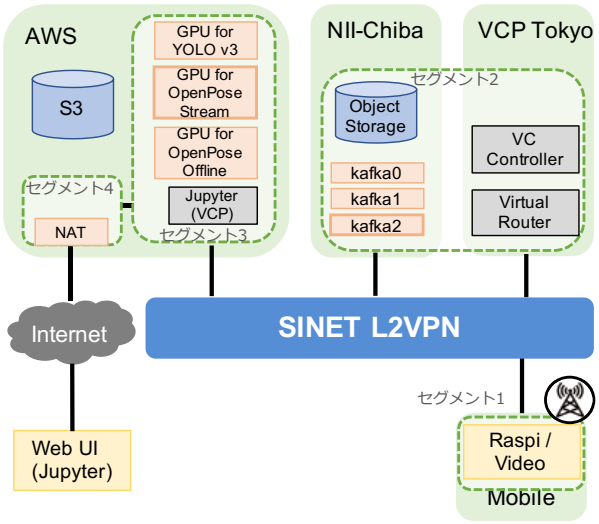

図 4: 実証実験環境.

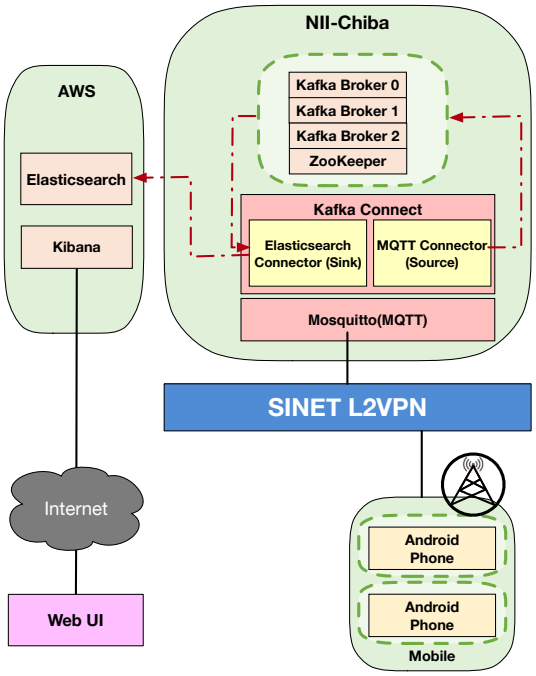

図 5: Android 端末を用いた IoT プロトタイプシステムの構成図

閉域網を構築し，安全かつ高性能なネットワーク環境下で kafka メッセージング基盤を実現した．しかし，我々の知る限り Android 端末向けの Kafka ライブラリは用意されておらず，Android 端末から Kafka を利用するにはアプリケーション開発者の負担が大きい．そこで，本稿では Android 端末からのセンサデータを軽量な MQTT Mosquitto ブローカ [16], [17] を Kafka のフロントエンドに配備して新たに用いてプロトタイプシステムを構築した．分散メッセージング基盤として，MQTT ブローカのみを利用する方法も考えれられるが，性能面や他のストリーム処理基盤やデータベースとの連携を想定し，Kafka をバックエンドで利用することとした．

図 5 にプロトタイプシステムの構成を示す．図 5 では，VCP 東京サイトは省略する．プロトタイプシステムでは，図の右下に示された Android 端末から SINET モバイル網経由でセンサデータを MQTT Mosquitto ブローカ基盤に送信する．Android 端末で収集されたセンサデータは，そのままの数値データで送るのではなく，各種センサから収集された数値データをトピックごとに分け，タイムゾーンを追加して JSON 形式に変換してから Mosquitto ブローカに送信される．Mosquitto ブローカに到着したセンサデータは，Kafka Connect を用いて Kafka クラスタに送信される．ここで利用した Kafka コネクタは，MQTT コネクタ [18] と Elasticsearch[19] コネクタから構成されている．前者は，Mosquitto ブローカに到着したセンサデータを Kafka システムに転送する．後者は，Kafka クラスタ経由で到着したセンサデータを Amazon Web Services (AWS) 上に構築された Elasticsearch にデータを格納する．蓄積されたデータは，Kibana で可視化した．

ソースコード 1: Paho で SINET Mobile にアクセス

```
1 private void connect() {
2   // SINET Mobile にアクセスするため，IP,Host,
        clientID の設定
3   MqttAndroidClient mClient = new
        MqttAndroidClient(myContext, serUrl,
        clientId.getText().toString(), new
        MemoryPersistence());
4   // コールバックの設定
5   mClient.setCallback(mqttCallbackExtended);
6 }
```

3.2 Android の MQTT クライアントプログラムの実装

Android での MQTT クライアントプログラムは，主に MQTT のパブリッシャ用プログラムとセンサプログラムから構成されている．

3.2.1 MQTT パブリッシャ用プログラム

本稿では，Google 社で発売されている Pixel 3 という Android 端末に SINET SIM に着装させ，Android クライアントの実装を行った．Pixel 3（OS バージョンは 9.0）から収集されたセンサデータを収集するため，Eclipse Paho を用いた．Paho は，オープンソースの MQTT ライブラリであり，多くの言語向けにクライアントライブラリを提供している．MQTT プロトコルの 3.1, 3.1.1 及び 5.0 のバージョンもサポートしており，比較的簡単に Android ベースでの MQTT パブリッシャ／サブスクライバ機能を実装することができる．Pixel 3 端末から Mosquitto ブローカにアクセスするため，Android クライアントプログラムでは Paho のパッケージに提供された `MqttAndroidClient` クラスを用いる．ソースコード 1 に示すように，`MqttAndroidClient` クラスに IP アドレス，ポート番号，クライアント ID とブローカにアクセスするための暗号化パラメータを指定することで Mosquitto ブローカに送信・受信することを確認できた．

3.2.2 センサプログラム

Android は様々なセンサタイプをサポートしているが，本研究では主に照度センサや加速度センサを用いて情報の送受信を行った．ソースコード 2 に，コードの一部を示す．センサプログラムを実装するには，まずセンサマネージャを取得した後，照度センサ及び加速度センサをセンサマネージャに登録する．これにより，Android スマホの周囲の環境変化を Pixel 3 に内蔵されてセンサから捉える度に `onSensorChanged()` というメソッドが呼び出される．

更新されたセンサデータをクラウドのデータベースに送信するため，センサデータが更新されるごとにその値を数値データからソースコード 3 のようなタイムゾーン付きの JSON 形式に変換して送信した．JSON 形式への変換には，Jackson というライブラリを用いた．変換された JSON オブジェクトは，ソースコード 4 で示すように `MqttAndroidClient` クラスの publisher メソッドで Mosquitto ブローカにセンサ情報を送信する．また，センサデータを送信する際に，トピック名，センサの種類の指定や，QoS と Retain 機能の設定が必要となる．Retain とは，パブリッシャからブローカに対して送られた最後のメッセージを保存しておく機能で，パブリッシャ側がメッセージを送る際にフラグをセットすることで実現される．サブスクライバ側はブローカに接続した際に，Retain されたメッセージを受け取ることができる．その他，利用者のニーズに応じてタイムアウトやキープアライブなどの複数

ソースコード 2: センサデータのパブリッシャー

```
1  public class SensorActivity extends Activity
       implements SensorEventListener {
2    private final SensorManager mSensorManager;
3    private final Sensor mLight;
4    @Override
5    protected void onCreate(Bundle
         savedInstanceState) {
6      super.onCreate(savedInstanceState);
7      // SensorManager のインスタンスを取得する
8      mSensorManager = (SensorManager)
           getSystemService(SENSOR_SERVICE);
9      // Sensor タイプの指定
10     mLight = mSensorManager.getDefaultSensor(
           Sensor.TYPE_LIGHT);
11   }
12   protected void onResume() {
13     super.onResume();
14     // イベントリスナーの登録
15     mSensorManager.registerListener(this,
           mLight, SensorManager.
           SENSOR_DELAY_NORMAL);
16   }
17   protected void onPause() {
18     super.onPause();
19     // イベントリスナーの登録解除
20     mSensorManager.unregisterListener(this);
21   }
22   public void onAccuracyChanged(Sensor sensor,
         int accuracy) {
23     // センサの精度が変更されると呼ばれる
24   }
25   public void onSensorChanged(SensorEvent event
         ) {
26     // センサの値が変化すると呼ばれる
27   }
28 }
```

ソースコード 3: JSON 情報

```
1 {
2   "sensor" : "mqtt-android-light",
3   "time" : "2019-06-11T15:18:48+0900”,
4   "value" : 457.78528
5 }
```

ソースコード 4: Paho でセンサ情報を Publish

```
1  private MqttAndroidClient mClient;
2  private Button publish = findViewById(R.id.
       publish);
3  publish.setOnClickListener(new View.
       OnClickListener() {
4    @Override
5    public void onClick(View view) {
6      try {
7        if (mClient.isConnected()) {
8          //Sensor データを publish する.
9          mClient.publish("mqtt-android-light".
               toString(), json_object.getBytes(),
               1, true);
10       }
11     }catch (MqttPersistenceException e) {
12     }catch (MqttException e) {
13     }
14   }
15 });
```

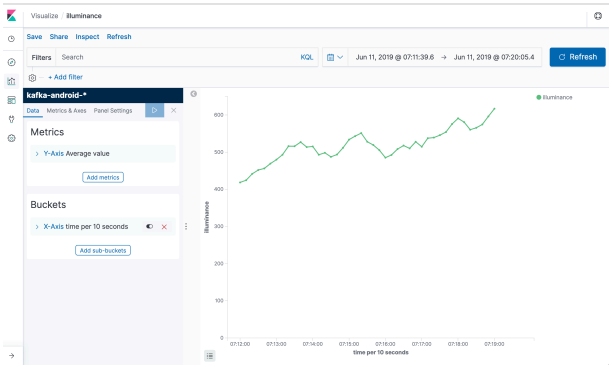

図 6: 照度センサの実行結果

のパラメータをソースコード 1 の `MqttAndroidClient` インスタンスを生成時に `MqttConnectOptions` クラスに追加して指定することができる．

3.3 実行結果

本稿では，Android 端末から収集されたデータを NII 千葉にある kafka クラスタ経由で AWS 上に構築した Elasticsearch v. 7.0 にデータを格納する．また，Elasticsearch に蓄積された Android 端末のセンサデータを Kibana v7.0 で可視化した結果を図 6 に示す．利用者がトピック名及び時間を変えることで異なるセンサの可視化結果を分析できるようにした．図 6 から，NII が提供している広域データ収集基盤に接続された Android 端末から蓄積された照度センサデータの結果を設定された時間単位で照会することができることを確認した．

4. 分散ストリーム処理基盤の予備評価

ストリーミングデータに対し複雑な分散処理を行うアプリケーション（アプリ）では，ノード間のデータ交換と同期，流量の変化に対するスケーリング，障害時のリトライといった課題に対処しなければならない．近年発達してきた分散ストリーム処理フレームワークは，これら共通の課題にフレームワーク側で対応し，プログラマが個々の処理とそれらの順序関係（パイプライン）を記述するだけで，スケーラブルな分散ストリーム処理アプリを迅速に開発・運用できる基盤を提供する．

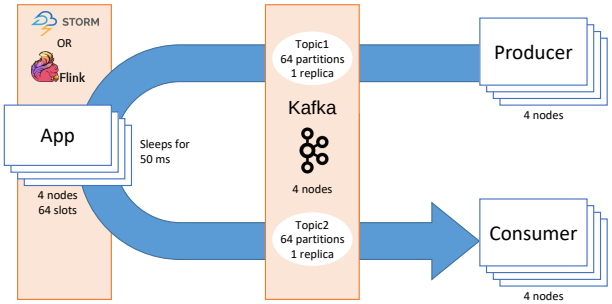

図 7: 実験構成

本稿では，分散メッセージキューを実現するミドルウェアである Kafka と，分散ストリーム処理フレームワークである Flink および Storm について，SINET 広域データ収集基盤におけるリアルタイム処理性能を検証するため，簡単なアプリによるストレステストを実施した．

4.1 評価方法

図 7 のように，メッセージを Kafka → Flink → Kafka もしくは Kafka → Storm → Kafka という経路で流し，レイテンシとスループットを測定する．メッセージを Kafka に流し込む Producer は単体の Python プログラムで，送信時刻と z KB のペイロードを含む JSON 形式のメッセージを $1/f$ 秒ごとに 1 個，180 秒間にわたって生成し，Kafka のトピック $Topic1$ に書き込む．この際，メッセージの文字列長は 72〜85 バイト＋ペイロード長となる．また，Kafka のメッセージ圧縮機能は無効とした．

メッセージを折り返す Flink アプリと Storm アプリはそれぞれ，各フレームワーク上に実装された Java プログラムで，Kafka のトピック $Topic1$ からメッセージを読み出し，一定時間待機した後，同じメッセージを Kafka のトピック $Topic2$ に書き込む．待機時間は，アプリ内で何らかの処理がされることを想定している．アプリ内では JSON を解釈せず，文字列としてデシリアライズ／シリアライズする．メッセージを Kafka から取り出す Consumer は単体の Python プログラムで，Kafka のトピック $Topic2$ からメッセージを読み出し，受信時刻とメッセージに含まれる送信時刻をローカルファイルに書き込む．

評価環境として，表 1 に示すクラスタを AWS 上に構築した．Kafka の各トピックのパーティション数は 64，レプリケーション数は 1，Flink アプリと Storm アプリの並列度はともに 64，その他の設定はデフォルトとした．Producer は，

表 1. 評価環境

役割	Kafka Broker, Consumer, Producer	Flink job manager	Flink task manager	Storm master	Storm worker
ノード数	4	1	4	1	4
インスタンス	m5.2xlarge	m5.large	m5.4xlarge	m5.large	m5.4xlarge
vCPUs	8	2	16	2	16
メモリ	32GB	8GB	64GB	8GB	64GB
ネットワーク	10Gbps	10Gbps	10Gbps	10Gbps	10Gbps

表 2. 実験で用いたパラメータ

パラメータ	実験で用いた値
ペイロードサイズ z	1, 16, 64, 256, 1024 [KB]
送信頻度 f	1, 30, 60, 240, 320 [FPS]
アプリ内での待機時間	50 [msec]

Kafka Broker と同じインスタンス上で 1 インスタンスにつき 1 つ，全体で 4 つのプロセスを走らせた．Consumer も同様である．ペイロードは $z \in \{1, 16, 64, 256, 1024\}$ KB，送信頻度は $f \in \{1, 30, 60, 240, 320\}$ 個/秒（以後 FPS と書く）とした．4 台の Producer を合わせると，Kafka Broker に流し込まれるペイロードの流量は全体で $4zf$ KB/秒となる．アプリ内での待機時間は 50 ミリ秒とした．表 2 に実験で用いたパラメータをまとめた．

評価値として，往復レイテンシとスループットを算出した．往復レイテンシは，各メッセージの送信時刻と受信時刻の差から，折り返し待機時間を引いたものである．ただし，Producer と Consumer で JSON をシリアライズ／デシリアライズする時間は含まれている．スループットは，4 台の Consumer が受信したペイロード量を 4 秒ごとに集計し，1 秒あたりのバイト数に換算したものである．

4.2 評価結果

往復レイテンシの累積分布を図 8 に示す．今回の評価ではペイロード量によってレイテンシが大きく変化しなかったため，ここではすべてのペイロード量を合わせた分布を示す．送信頻度が 1FPS〜240FPS（全体で 4FPS〜960FPS）の範囲では，90%のメッセージが 240 ミリ秒以下，99%のメッセージが 500 ミリ秒以下のレイテンシで往復していることが確かめられた．一方，送信頻度が 320FPS（全体で 1280FPS）に達すると遅延が徐々に累積し，最大で 8228 ミリ秒の遅延が観測された（Kafka+Flink で $z = 256$ KB のとき）．メッセージの処理に 50 ミリ秒かかるアプリが 64 並列で動作しているので，理想的には全体で $\frac{1}{0.05} \times 64 = 1280$FPS まで遅延なく処理できるはずだが，実際には種々のオーバーヘッドが加わるため，これは妥当な結果といえる．

スループットの測定結果を図 9 に示す．送信頻度が高くなるにつれてジッターが大きくなる傾向が認められるものの，全体として $4zf$ KB/秒を安定的に維持できていることが確かめられた．始端と終端が落ち込んでいるのは，経過時間ではなく絶対時刻で 4 秒ごとに区切って集計したためである．ここでは $z = 1024$KB の結果のみを示したが，他のペイロード量でも同様の結果が得られた．なお，このグラフには現れていないが，実験を繰り返す中で，スループットが瞬間的に大きく上下する現象が観測されたケースもあった．何らかの要因で数秒間メッセージが滞留し，その後一気に吐き出されたものと考えられる．このような秒単位のジッターがまれに発生することから，ストリーム処理基盤にミリ秒単位のリアルタイム性を期待するべきではないといえる．

5. 関連研究

多数の MQTT ベースメッセージング基盤が実装されている．文献 [20] やウェブサイト [21], [22] で複数 MQTT 実装の比較が行われている．VerneMQ や Apache ActiveMQ などクラスタ構成可能な Broker を提供するソフトウェアもあり，今後比較していく．

文献 [23] では，Kafka と RabbitMQ の機能や性能を比較している．性能評価では，RabbitMQ と 1 台構成の Kafka を比較し，メッセージサイズが 2KB までの条件で同程度の性能を示している．本研究では，画像データ等も考慮した 64KB までの比較と，クラスタ構成の Kafka の性能について調査している．

Apache Spark，Apache Storm，Apache Heron，Apache Flink, Apache Samza など，複数のストリーミングデータ処理基盤も開発されている．これらはメッセージング基盤と連携し，複数の計算ノードを活用して高度な処理を高スループットで行うことを目指して設計されている．ストリーミングデータ処理基盤を利用したシステムの構築では，性能面から Kafka と連携する試みが多い [24], [25]．また，オープンソースソフトウェアではなく商用クラウドのストリーミングデータ処理基盤を利用する選択肢もある．文献 [26] では，Kafka と Amazon Kinesis を比較して，コストと性能のトレードオフを議論している．

6. おわりに

SINET 広域データ収集基盤により，モバイル網を SINET の足回りとして SINET SIM を装着したセンサ端末群と大学等のオンプレミス計算環境や商用クラウド内の計算資源

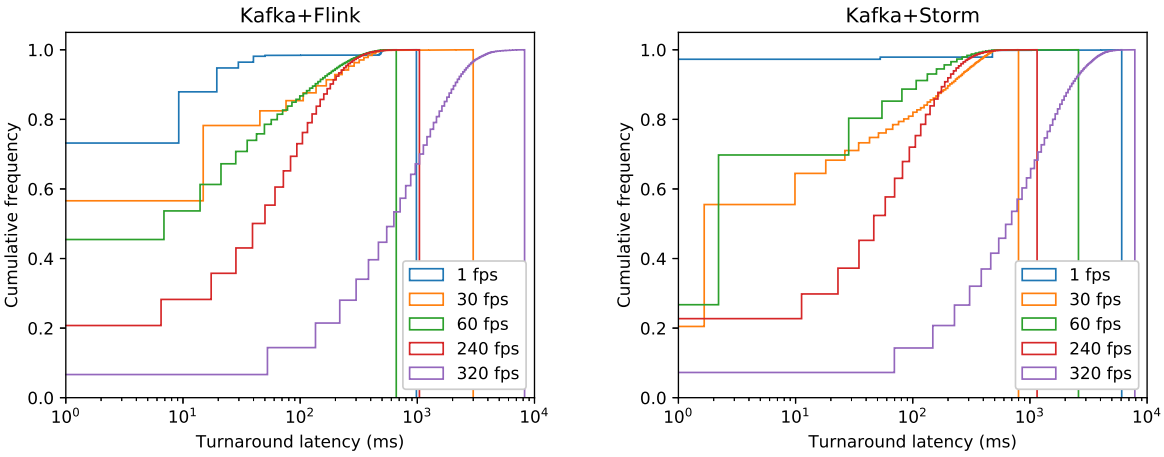

図 8: 往復レイテンシの累積分布．すべてのペイロード量の合計．右端の垂線は最大レイテンシを示す．

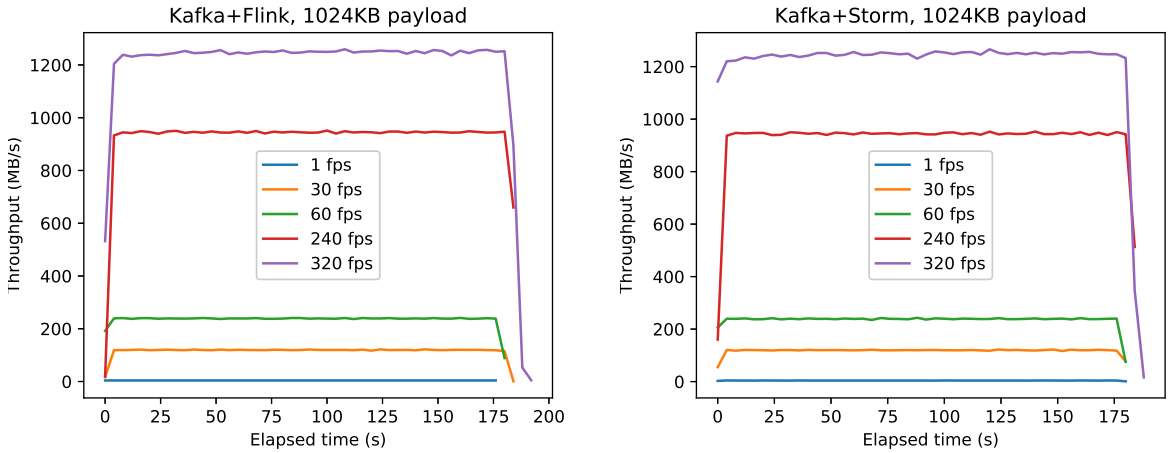

図 9: スループット．ペイロード量 1024KB の場合．

間を L2 VPN の閉域網で接続することが可能になった．既発表研究において，SINET 広域データ収集基盤を用いたリアルタイムビデオ処理機構のプロトタイプシステムを構築し，安全かつ高性能なネットワーク環境下でメッセージング基盤を利用した大量データの収集と，クラウドの GPU ノードを活用したリアルタイム処理の実証実験を行った．しかしながら，多種センサの利用や大量データ処理という点で課題があった．

本研究では，既存の多種センサに対応するため，Android 端末から利用可能な様々なセンサ情報を Eclipse Paho という MQTT 通信ライブラリと Mosquitto MQTT ブローカを用いて，IoT システムプロトタイプを広域データ収集基盤上に新たに構築した．また，分散メッセージキューを実現するある Kafka と，分散ストリーム処理フレームワークの Flink および Storm についてその性能を調査し，いずれのフレームワークも多少のオーバヘッドがあるものの低遅延で処理できることがわかった．

今後は，Android 端末からの取得される様々なセンサデータストリームを収集し，クラウドの GPU ノードと分散ストリーム処理基盤を用いたリアルタイム解析処理の開発等を進める．また，各種ストリーム処理基盤に対して統一された API でアクセスを可能にするライブラリを構築することで，SINET 広域データ収集基盤の活用を推進する．

謝辞 本研究を進めるにあたり，実験環境構築および評価実験の実施にご協力いただいた数理技研の小泉敦延様，鯉江英隆様に深く感謝いたします．

参考文献

[1] SINET 広域データ収集基盤，`https://www.sinet.ad.jp/wadci`.

[2] 竹房あつ子，孫　静涛，長久　勝，吉田　浩，政谷好伸，合田憲人：SINET 広域データ収集基盤を用いたリアルタイムビデオ処理機構の検討，マルチメディア，分散，協調とモバイル (DICOMO2019) シンポジウム論文集，pp. 1581–1588 (2019).

[3] 広域データ収集基盤デモパッケージ，`https://github.com/nii-gakunin-cloud/wadci-demo`.

[4] 学認クラウドオンデマンド構築サービス，`https://cloud.gakunin.jp/ocs/`.

[5] Takefusa, A., Yokoyama, S., Masatani, Y., Tanjo, T., Saga, K., Nagaku, M. and Aida, K.: Virtual Cloud Service System for Building Effective Inter-Cloud Applications, *Proc. CloudCom 2017*, pp. 296–303 (2017).

[6] Kreps, J., Narkhede, N. and Rao, J.: Kafka : a Distributed Messaging System for Log Processing, *NetDB*

workshop 2011, pp. 1–5 (2011).
[7] Shree, R., Choudhury, T., Gupta, S. C. and Kumar, P.: KAFKA: The modern platform for data management and analysis in big data domain, *2017 2nd International Conference on Telecommunication and Networks (TEL-NET)*, pp. 1–5 (online), DOI: 10.1109/TELNET.2017.8343593 (2017).
[8] Apache Kafka, `https://kafka.apache.org/`.
[9] Apache Flink, `https://flink.apache.org/`.
[10] Apache Storm, `https://storm.apache.org/`.
[11] MQTT (Message Queue Telemetry Transport), `http://mqtt.org/`.
[12] YOLO, `https://pjreddie.com/darknet/yolo/`.
[13] A PyTorch implementation of the YOLO v3 object detection algorithm, `https://github.com/ayooshkathuria/pytorch-yolo-v3`.
[14] OpenPose, `https://github.com/CMU-Perceptual-Computing-Lab/openpose`.
[15] Eclipse Paho, `https://www.eclipse.org/paho/`.
[16] Light, R.: Mosquitto: server and client implementation of the MQTT protocol, *The Journal of Open Source Software*, Vol. 2, pp. 1–2 (online), DOI: 10.21105/joss.00265 (2017).
[17] Eclipse Mosquitto, `https://mosquitto.org/`.
[18] Lenses.io MQTT Connector, `https://docs.lenses.io/connectors/source/mqtt.html`.
[19] Confluent Elasticsearch Connector, `https://docs.confluent.io/current/connect/kafka-connect-elasticsearch/index.html`.
[20] Patro, S., Potey, M. and Golhani, A.: Comparative study of middleware solutions for control and monitoring systems, *2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, pp. 1–10 (online), DOI: 10.1109/ICECCT.2017.8117808 (2017).
[21] Wikipedia: Comparison of MQTT implementations, `https://en.wikipedia.org/wiki/Comparison_of_MQTT_implementations`.
[22] MQTT community website, server support, `https://github.com/mqtt/mqtt.github.io/wiki/server-support`.
[23] Dobbelaere, P. and Esmaili, K. S.: Kafka Versus RabbitMQ: A Comparative Study of Two Industry Reference Publish/Subscribe Implementations: Industry Paper, *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems*, DEBS '17, New York, NY, USA, ACM, pp. 227–238 (online), DOI: 10.1145/3093742.3093908 (2017).
[24] Javed, M. H., Lu, X. and Panda, D. K. D.: Characterization of Big Data Stream Processing Pipeline: A Case Study Using Flink and Kafka, *Proceedings of the Fourth IEEE/ACM International Conference on Big Data Computing, Applications and Technologies*, BDCAT '17, New York, NY, USA, ACM, pp. 1–10 (online), DOI: 10.1145/3148055.3148068 (2017).
[25] Kato, K., Takefusa, A., Nakada, H. and Oguchi, M.: A Study of a Scalable Distributed Stream Processing Infrastructure Using Ray and Apache Kafka, *2018 IEEE International Conference on Big Data (Big Data)*, pp. 5351–5353 (online), DOI: 10.1109/BigData.2018.8622415 (2018).
[26] Nguyen, D., Luckow, A., Duffy, E. B., Kennedy, K. and Apon, A.: Evaluation of Highly Available Cloud Streaming Systems for Performance and Price, *Proceedings of the 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, CCGrid '18, Piscataway, NJ, USA, IEEE Press, pp. 360–363 (online), DOI: 10.1109/CCGRID.2018.00056 (2018).