情報処理学会論文誌　デジタルプラクティス　Vol.1　No.1　45–57 (Oct. 2020)

推薦論文

SINET 広域データ収集基盤を用いたオンラインビデオ処理実証実験

竹房 あつ子[1]　孫 静涛[1]　藤原 一毅[1]　長久 勝[1,2]　吉田 浩[1]　政谷 好伸[1]　合田 憲人[1]

受付日 2020年3月11日, 再受付日 2020年6月3日,
採録日 2020年7月15日

概要：ビデオデータのオンライン解析は，自動運転における物体認識や防犯カメラでの異常検知等の用途でその需要が高まっているが，センサとクラウド間のネットワーク帯域と安全性の確保，大量データの収集，クラウドでのオンラインビデオ解析処理の性能が課題となる．本研究では，SINET 広域データ収集基盤を用いたオンライン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 ノードを活用したオンライン処理が実現可能であることを示す．SINET 広域データ収集基盤は，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，SINET の VPN に直接接続できる．また，IoT アプリケーションの設計指針を得ることを目的とし，予備評価によりオンライン処理システムを構成するメッセージング基盤および分散ストリーム処理基盤の性能特性を示す．

キーワード：オンラインビデオ処理，分散ストリーム処理，IoT，パブリッシュ・サブスクライブシステム，モバイルネットワーク

Demonstration of an Online Video Processing Framework using the SINET Wide-area Data Collection Infrastructure

Atsuko Takefusa[1]　Jingtao Sun[1]　Ikki Fujiwara[1]　Masaru Nagaku[1,2]　Hiroshi Yoshida[1]
Yoshinobu Masatani[1]　Kento Aida[1]

Received: March 11, 2020, Revised: June 3, 2020,
Accepted: July 15, 2020

Abstract: Online analysis of video data captured by sensor devices is increasingly required for applications such as object recognition in autonomous driving and abnormality detection by security cameras. However, there are several issues such as ensuring secure and stable communication environment between sensors and a cloud, collecting large amounts of various data and processing online video analysis efficiently. In this study, we have developed a prototype system of an online video processing mechanism using the SINET wide area data collection infrastructure called "mobile SINET" and show that the prototype system can collect a large amount of data using messaging infrastructure software in a secure and high-performance network environment, and online video processing is feasible using GPU nodes in a cloud. SINET is a wide-area academic network infrastructure and the mobile SINET is a service that provides mobile networks as an access network of SINET and enables direct connection to a SINET VPN. Each sensor can use the mobile SINET by installing a SIM for the SINET connection. In addition, we investigate the performance characteristics of messaging infrastructure and distributed stream processing infrastructure that constitute an online processing system in order to obtain design guidelines for IoT applications.

Keywords: online video processing, distributed stream processing, IoT, publish subscribe system, mobile network

[1] 国立情報学研究所
National Institute of Informatics
[2] ライフマティックス株式会社
Lifematics Inc.

本稿の内容は 2019 年 7 月 5 日の DICOMO2019 シンポジウムにて報告され，IOT 研究会主査により論文誌トランザクションデジタルプラクティスへの掲載が推薦された論文である．

1. はじめに

ビデオデータのオンライン解析は，自動運転における物体認識や防犯カメラでの異常検知等の用途でその需要が高まっている．高度なオンライン解析処理を行うには，センサで取得したビデオデータを広域網を用いてクラウドに収集し，クラウドの高性能な計算資源を用いて大量のビデオデータの解析処理を行い，その結果を利用者またはセンサ側に送信する必要がある．しかしながら，このような処理では以下のような技術的課題がある．

(1) センサとクラウド間のモバイル網の帯域と安全性
(2) 大量データの収集性能
(3) クラウドでのオンラインビデオ解析処理の性能

（1）は，モバイル網を用いることで各種センサへの接続が容易になるが，既存の 4G/LTE 回線を介して大量のセンサから連続的に生成されるデータをクラウドの計算資源に送信する場合，モバイル網の帯域を考慮した IoT アプリケーション設計が必要とされる．また，サイバー攻撃等からセンサやクラウド計算資源を守るために外部ネットワークから隔離されていることが望ましいが，特にモバイル網ではそのような隔離は難しい．（2）は，データサイズの小さい大量のセンサデータをクラウドのストレージに直接書き込むには，オブジェクトストレージの書き込み性能や HTTP プロトコルのオーバヘッドなどが課題となる．IoT（Internet of Things）通信向けに MQTT（Message Queue Telemetry Transport）[1] のような軽量プロトコルを用いたメッセージング基盤が複数開発されているが，個々のアプリケーションの要件を満たすのに必要な構成などが明らかでない．（3）は，オンライン処理をするにはストリームデータの到着速度とビデオ解析処理速度とのバランスを取る必要があるが，その性能要件も明らかでない．特に，広帯域，低遅延の第 5 世代移動通信システム「5G」の利用が可能になると，性能ボトルネックはクラウドでの解析処理へと移行するため，その検証が必要不可欠である．

本研究では，SINET 広域データ収集基盤 [2] を用いたオンラインビデオ処理機構のプロトタイプシステムを構築して実証実験を行うとともに予備評価を行い，オンライン処理システムを構成するメッセージング基盤および分散ストリーム処理基盤の性能特性を示す [3], [4]．画質およびフレームレートを調整可能なシステムを構築するとともに，SINET 広域データ収集基盤を用いてモバイル網を含むネットワーク全体の隔離を実現することで，（1）の課題を解決する．また，予備評価により，（2），（3）の課題について明らかにする．

SINET 広域データ収集基盤は，国立情報学研究所（NII）が提供する学術情報ネットワーク 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，2018 年度から実証実験を開始している．提案するプロトタイプシステムでは，本モバイル網と SINET L2 VPN（Virtual Private Network）の閉域網を用いて，安全かつ広帯域なネットワークを介してオンプレミス環境および商用クラウド環境にデータを収集する．また，オンプレミスおよびクラウドでメッセージング基盤の構築，オブジェクトストレージとの連携，クラウドの GPU ノードを用いたオンライン処理が実現可能であることを示す．

プロトタイプシステムのメッセージング基盤には，既発表研究 [5] で高スループットで画像データの収集ができることを確認している Apache Kafka [6], [7], [8] を採用した．また，提案するオンラインビデオ処理機構のアプリケーションとして，収集したストリームデータからオブジェクト抽出するライブラリ YOLO v3 [9], [10] の PyTorch 実装 [11] と，人のキーポイント抽出ライブラリ OpenPose [12], [13], [14], [15], [16] を用いる．これらの画像処理ライブラリでオンライン処理するプログラムを開発し，オンラインで処理できることを示す．

プロトタイプシステムの構築には，「学認クラウドオンデマンド構築サービス」[17], [18] を用いた．これにより，クラウド上に短時間でアプリケーション環境を構築することができる．また，構築および実証実験実施の際に用いたプログラムはオープンソースとして公開した [19]．

予備評価では，オンライン IoT アプリケーションを構築するための設計指針を得ることを目的とし，オブジェクトストレージとメッセージング基盤の IO 性能と，既存分散ストリーム処理基盤を用いた処理性能を調査する．オブジェクトストレージの評価では，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の基本性能を比較する．メッセージング基盤の評価では，Kafka と MQTT ブローカ実装の一つである Eclipse Mosquitto [20], [21] の性能を調査した．また，既存分散ストリーム処理基盤の評価として，Apache Flink [22]（Flink）と Apache Storm [23]（Storm）を用いたクラスタ環境をそれぞれ構築し，そのうえ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．分散ストリーム処理基盤では，Flink，Storm の他，Spark Streaming（Spark）[24] も一般に広く利用されているが，Spark は遅延やスケーラビリティの点で Flink や Storm に劣ることが示されているため [25], [26]，本研究では Flink と Spark を用いる．

実験から，大量のセンサデータを扱うにはオブジェクトストレージでは不十分でメッセージング基盤を活用することが望ましいこと，MQTT ベースの Mosquitto はクライアント用のライブラリが充実しているという利点もあるが，大量データのデータ収集には Kafka が優位であること，IoT アプリケーションの要件に合わせてソフトウェア構成を設計する必要があることを示す．また，分散スト

リーム処理基盤の評価では Flink，Storm いずれのミドルウェアも多少のオーバヘッドがあるものの低遅延，高スループットで処理できることを明らかにする．

以降の本稿の構成は以下のとおりである．2 章では，SINET 広域データ収集基盤を用いた実証実験の概要を述べるとともに，実験で用いたプロトタイプシステムの構成と実験環境の詳細について述べる．3 章では，オンライン IoT アプリケーションを構築するための設計指針を得ることを目的とし，オブジェクトストレージとメッセージング基盤の IO 性能と，既存分散ストリーム処理基盤を用いた処理性能を示す．4 章では関連研究，5 章ではまとめと今後に課題について述べる．

2. オンラインビデオ処理機構プロトタイプシステムの構築

SINET 広域データ収集基盤を用いたオンラインビデオ処理機構のプロトタイプシステムを構築し，2019 年 4 月に開催された人工知能の活用をテーマとしたイベント AI/SUM [27] のショーケースで実証実験を行った．以降で，実証実験の概要について述べた後，プロトタイプシステムのソフトウェア構成と構築した実証実験環境の詳細について述べる．

2.1 SINET モバイル網を用いた実証実験

NII では，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「SINET 広域データ収集基盤」の実証実験を開始した．SINET 広域データ収集基盤では，大学等の SINET 加入機関の研究者が SINET 接続用の SIM を用意してセンサ等に装着することで，それらが送信するデータを SINET に接続された計算環境で収集，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 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 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，そのオンライン画像解析処理ができることを示した．ここで，センサ端末，クラウド上のメッセージング基盤，オンライン画像解析処理を行う GPU ノードはすべて SINET の VPN 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ム配信された結果が表示されている．右側上には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 AI/SUM 展示会場に設定されたセンサ端末のカメラ画像が表示されている．また，左側上

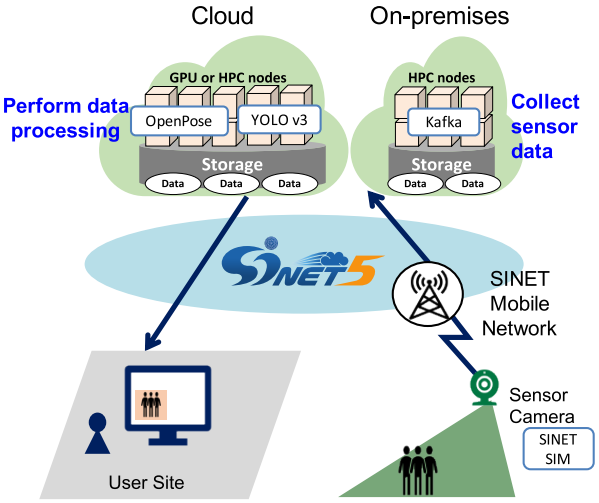


図 1　SINET モバイル網を用いたオンラインビデオ処理実証実験の概要．

Fig. 1　An overview of an online video processing demonstration over the SINET mobile network.

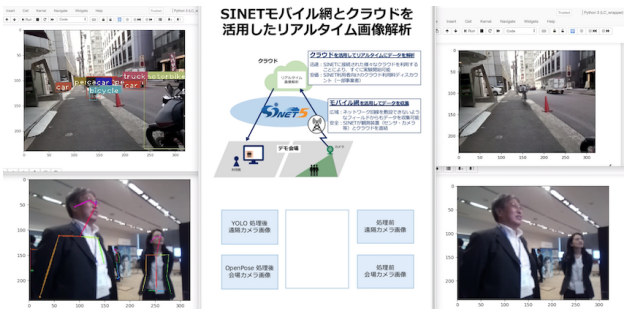


図 2　オンラインビデオ処理実証実験結果のスナップショット．

Fig. 2　A snapshot of the online video processing demonstration.

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 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 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．センサ端末のカメラで取得した動画から切り出した静止画をそのまま送ると画像サイズが大きく十分な枚数の画像が送れなくなってしまうため，センサ端末で 320×240 画素，64 KB 弱の静止画に変換してから送信するようにした．実効通信帯域は電波状況等により変化するが，実験では 6 fps 程度のフレームレートとなっていた．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．OpenPose では，6 fps のデータに対するオンライン処理ができることを確認した．一方，YOLO v3 は OpenPose より解析処理の負荷が高いため，画像数を半分程度に間引き

ながらオンライン処理を行った．

2.2　プロトタイプシステムのソフトウェア構成

オンライン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ム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ム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取得してバッチ的に画像処理する．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ムデータを出力する．

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)　ストリーム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

実験では，動画像は静止画のストリームデータとして配信・収集する．そのためのメッセージング基盤には，Apache Kafka（Kafka）を用いた．Kafka は，Pub/Sub 型非同期メッセージングモデルを採用する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．個々のストリームデータは 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になっている．プロトタイプシステムでは，Kafka Broker で収集したストリームデータを，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するため，それぞれの認証情報を設定した S3 Connector を利用した．ただし，オブジェクトストレージに POSIX ファイルシステムのパス名をマッピングする場合，同じフォルダ内に多数のファイル（オブジェクト）を格納する形にすると，フォルダ内全ファイルの検索等の処理時間が非常に長くなってしまう．具体的には，パブリッククラウドのオブジェクトストレージでファイル数が 1 万を超えると一覧取得処理のみで 5 秒から 10 秒かかることが報告されている [28]．データをどう利用するかあらかじめ想定し，データを格納するパスを設計しておく必要がある．

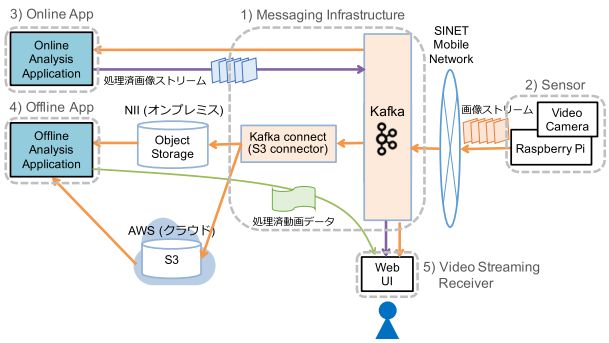


図 3　オンラインビデオ処理機構プロトタイプシステムのソフトウェア構成．

Fig. 3　Software configuration of the prototype system for online video processing.

2)　センサ用プログラム

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 VPN 経由で Kafka Broker に静止画を繰り返し送信するようにした．センサ用プログラムは，Kafka の Producer として動作する．カメラモジュールから 320×240 画素の静止画を取得し，Topic 名を指定して静止画のバイナリデータを送信する．Producer では，転送性能を高めるためにデフォルト設定では複数レコードをまとめて送信するようになっているが，プロトタイプシステムでは 1 枚ずつオンラインで画像処理ができることを示すため，1 レコードずつ送信するようにした．また，利用したカメラモジュールでは 1 秒間に 30 フレームの画像を取得することができるが，モバイル網で 30 fps で転送することはできないため，センサ用プログラムで取得するフレーム数を調整できるようにした．さらに，モバイル網を利用する場合センサ端末のデフォルト MTU（Maximum Transmission Unit）サイズが適切であるか，注意する必要がある．我々が用いた環境では，RFC2923 の Path MTU Discovery ブラックホール問題が発生して Raspberry Pi から Kafka に対して正常に通信できなかったため，MTU サイズを手動で調整する必要があった．

プロトタイプシステムの動作確認を目的とし，動画ファイルや S3 互換オブジェクトストレージに格納された複数の静止画を用いて，指定したフレームレートで Broker にレコードを送信する Producer コードも用意した．これにより，センサ端末の用意ができていない状況でもオンラインビデオ処理の動作確認が可能となる．

3)　オンラインアプリケーションプログラム

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．まず，Broker から順次静止画を受け取ると，それぞれのライブラリを用いて処理する．処理結果のストレージへの格納と利用者端末での結果確認のため，別の Topic 名を指定して Broker に送信する．オンライン処理のため，本プログラムでも必要に応じて処理する静止画の枚数を調節している．

4)　オフラインアプリケーションプログラム

本研究ではオンライン処理をターゲットとしているが，複数静止画をまとめて処理するオフラインアプリケーションプログラムも用意した．オフラインアプリケーションプログラムでは，S3 互換ストレージに格納された複数の静止画を OpenPose で処理するプログラムを開発した．S3 に格納されたデータへのアクセスでは，goofys [29] を用いた．goofys は Go で実装されており，S3 互換ストレージをファイルシステムとしてマウントすることができ，s3fs より高速なアクセスが可能となっている [29]．goofys 経由で指定された複数の静止画を読み出し，個々の静止画に対して 3）同様に画像処理を行い，生成された複数の処理済み画像からを一つの動画を生成するようにした．

5)　ストリーム画像出力プログラム

利用者端末では，処理前および処理後静止画のストリームデータを動画像として表示させるストリーム画像出力プログラムを用意した．ストリーム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．本プログラムは，ウェブインタフェースで文書やコードを記述，実行できる Jupyter Notebook [30] で実装し，表示したい Topic や表示レートなどがその場で適宜変更できるようにした．

2.3　実証実験環境構築とプログラムの公開

図 4 に実証実験環境を示す．実験では，SINET の L2 VPN で SINET モバイル網，VCP 東京サイト，NII 千葉サイト，AWS 東京リージョンを接続し，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を利用した．

学認クラウドオンデマンド構築サービスは，SINET5 とクラウドを活用したアプリケーション実行環境の構築を学術研究機関に対して支援することを目的として，2018 年 10 月に運用を開始した．本サービスでは，仮想ルータの提供の他，NII が開発した VCP（Virtual Cloud Provider）[31] と呼ばれる管理ソフトウェアを用いて様々なクラウドでの資源制御を容易にするとともに，研究・教育目的のアプリケーションの構築手順を Jupyter Notebook 形式で提供する．実証実験では，クラウドやオンプレミス環境に複数のプログラムを配備して環境構築する必要があるため，本サービスを用いた．

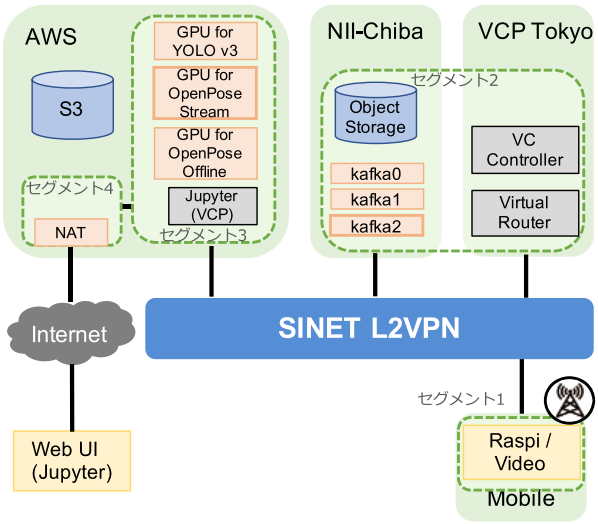


図 4　オンラインビデオ処理実証実験環境．
Fig. 4　Experimental settings for online video processing.

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 台には，3 Model B Rev 1.2，Debian v. 9.11 を用い，SINET の SIM カードを利用して SINET 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．YOLO v3 では g3.8xlarge（GPU 数 2，VCPU 数 32，244 GiB メモリ，16 GiB GPU メモリ），OpenPose では g3.4xlarge（GPU 数 1，VCPU 数 16，122 GiB メモリ，8 GiB GPU メモリ）を用いた．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，実験環境に対しては AWS の NAT インスタンスを経由してアクセスするようにした．

実証実験環境の構築および実施の際に用いたプログラムは，「SINET 広域データ収集基盤デモパッケージ」として GitHub でオープンソースとして公開している [19]．本パッケージは，クラウド資源の制御部分を除いて，学認クラウドオンデマンド構築サービスを利用しない場合にも活用できる．

3.　予備評価

オンライン IoT アプリケーションを構築するための設計指針を得ることを目的とし，有線環境でオブジェクトストレージとメッセージング基盤の IO 性能と，既存分散ストリーム処理基盤を用いた処理性能を調査する．

3.1　予備評価の目的

一般に，オンライン IoT アプリケーションを構築するには，「センサデータの収集」，「解析処理」，「結果のフィードバック」までにかかる「応答時間」を各アプリケーションの要求される時間内に収める必要がある．オンライン IoT アプリケーションにおける「応答時間」の目安は，たとえば製造現場では，製造機械の制御では 10 ミリ秒以下であるのに対し，設備生産管理では数秒から数分と，個々の用途によって異なることが報告されている [32]．また，それらで発生するデータサイズ，データ生成頻度，センサ数も大きく異なる．実験では，本稿で想定するソフトウェア構成において，どの程度のオンライン IoT アプリケーションが実現可能であるか明らかにする．

2 章では，プロトタイプシステムにより SINET 広域データ収集基盤の提供する 4G/LTE 回線を介してオンラインビデオ処理ができることを実証した．本実験では，「センサデータの収集」におけるモバイル網が性能ボトルネックとなっていたため，画質や転送レートを調整するとともにメッセージング基盤を用いてオンラインでの処理を実現した．一方で，センサデータの発生頻度が少ない IoT アプリケーションでは必ずしもメッセージング基盤を用いる必要がなく，オブジェクトストレージを利用して直接「センサデータの収集」を行い，より簡易な構成でアプリケーションを構築するほうが望ましいと考えられる．

また，複数のセンサデータを同時に扱う場合や広帯域，低遅延の第 5 世代移動通信システム「5G」の利用が可能になると，性能ボトルネックは「解析処理」へと移行する．既存分散ストリーム処理基盤の活用により処理をスケールアウトすることが期待できるが，様々なデータサイズや転送レートに対してどの程度の遅延，スループットで処理できるのか，明らかではない．

よって，本章ではオンライン IoT アプリケーションの設計指針を得るため，将来の広帯域，低遅延な環境を前提として有線環境で以下の評価実験を行う．「センサデータの収集」のための既存システムの性能特性を得るため，3.2 節ではオブジェクトストレージの IO 性能，3.3 節ではメッセージング基盤の IO 性能を調査，比較する．また，「解析処理」のための既存システムの性能特性を得るため，3.4 節では既存分散ストリーム処理基盤の処理遅延，スループットを調査する．「結果のフィードバック」については，一般的に「センサデータの収集」よりデータ量，発生頻度が少なくボトルネックになりにくいと考えられるため，本稿ではスコープ外とする．

3.2　オブジェクトストレージの性能

まず，オブジェクトストレージの評価では AWS S3 と NII オンプレミス環境の HGST 社製オブジェクトストレージ装置 ActiveScale P100（以降，P100 とする）の IO 性能を調査する．P100 は，システムノード 3 台，ストレージノード 6 台，864 TB（実効容量 580 TB）構成であり，Inter-rack ネットワーク帯域は 40 Gbs，Intra-rack ネットワーク帯域は 10 Gbs となっている．実験では，1 台のシステムノードに対して IO アクセスを行う．

評価には，クラウドオブジェクトストレージのベンチマークソフトウェアの一つである COSBench [33] を用いた．アクセスパターンは，Write を 20%，Read を 80% とし，読み書きしたデータのサイズは 64 KiB，ベンチマーク runtime は 60 秒とした．リクエスタ用のノードは，オンプレミスでは**表 1** に示す物理サーバ上に配備した 8 コア 16 GiB メモリの VM 3 台，AWS では vCPU 2 コア，8 GiB メモリの m5.large を 3 台用いた．オンプレミス環境では，リクエスタ用の物理サーバと P100 が 10 Gbps で接続されている．AWS 内では，S3 のネットワークインタフェースは明らかでないが，各リクエスタ VM は最大 10 Gbps の通信帯域となっている．

図 5，**図 6** に Write スループットを，**図 7**，**図 8** には Read スループットを 1 秒あたりの総オペレーション数 [op/s] と総バイト数 [MB/s] でそれぞれ示す．横軸はリクエスタ数で，縦軸は総スループットを示している．青いバーの“NII->P100”は NII のリクエスタから P100 へ，橙色のバーの“AWS->S3”は AWS 内部のリクエスタから S3 へ，灰色のバーの“NII->S3”は NII のリクエスタから S3 へのアクセスとなっている．NII->P100 と AWS ->S3 はクラウド内，NII->S3 はクラウド間の通信となっており，後者は一部商用インターネット回線を利用している．

図 5，図 6 の結果から，クラウド内での Write アクセスではリクエスタ数が増えるにつれて増加率は鈍るものの，スループット値が増加していくことが分かる．オンプレミスとクラウドでスループット値を比較すると，リクエスタ数が少ないときにはオンプレミスのほうが大きいが，リクエスタ数が多くなると S3 のほうが大きくなり，スケーラビリティの高さが確認できる．一方，クラウド間 Write アクセスではクラウド内アクセスより増加率は高いものの，スループット値は低く，リクエスタ数 144 のときにはベンチマークプログラムが正常に終了することができな

表 1　NII オンプレミス環境で用いたリクエスタ用物理サーバの構成．

Table 1　Specification of the physical server for requesters in the NII on-premise environment.

OS	Ubuntu 16.04.6 LTS
CPU	Intel(R) Xeon(R) CPU E5-2660 v4 @ 2.00GHz, 56 コア
メモリ	125GiB
ディスク	4023GB
NIC	10Gbps

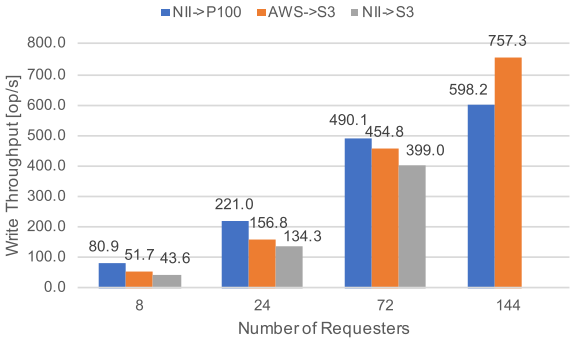


図 5　Write スループット [op/s].
Fig. 5　Write throughput in op/s.

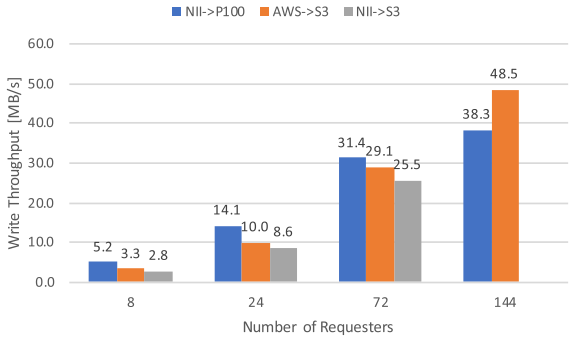


図 6　Write スループット [MB/s].
Fig. 6　Write throughput in MB/s.

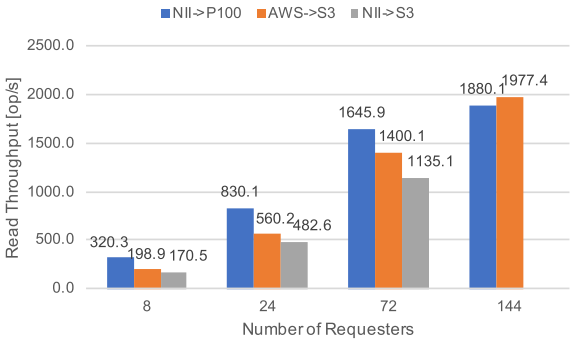


図 7　Read スループット [op/s].
Fig. 7　Read throughput in op/s.

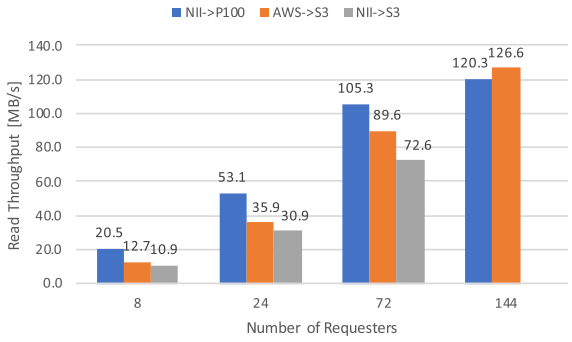


図 8　Read スループット [MB/s].
Fig. 8　Read throughput in MB/s.

表 2　メッセージング基盤の評価環境.
Table 2　Experimental settings for evaluation of messaging systems.

Broker 用ノード（Kafka 3 台，Mosquitto 1 台）	
インスタンス	m5.2xlarge
VCPU	8
メモリ	32GiB
Network	最大 10Gbps
リクエスタ用ノード（Kafka，Mosquitto 各 1 台）	
インスタンス	m5.xlarge
VCPU	4
メモリ	16GiB
Network	最大 10Gbps

かった．モバイル網を利用するような遅延が大きい環境では，一般に同一サイト内で測定される Write スループット値より低くなることを想定する必要があることが示唆された．

図 7，図 8 の Read スループットの結果でも，クラウド内・クラウド間の実行結果は Write スループット結果と同様の傾向がみられ，リクエスタ数 144 のときの実験結果は得られなかった．Read スループットは，Write スループットの 3 倍程度から 4 倍近い性能を示していたが，1 Gbps 程度であった．IoT 環境のように小さいサイズのファイルを大量に扱う場合には，通信性能よりオブジェクトストレージ側の処理がボトルネックとなることが示された．

3.3　メッセージング基盤の性能

次に，メッセージング基盤として実証実験で用いた Kafka と MQTT 実装の一つである Mosquitto の書き込み性能を比較する．Kafka はクラスタ構成を取ることでスケールアウトが容易であり，永続化も可能である．Mosquitto は 1 ノード構成で利用し機能的には Kafka に劣るものの，MQTT ベースで Android 等のセンサ側端末用クライアントライブラリが充実している，処理負荷が相対的に低いといった利点がある．実験では，Kafka は v. 2.2.0，Mosquitto は v. 1.6.2 を用いた．

メッセージング基盤の評価は AWS 内で実施した．**表 2** に評価環境を示す．Broker 用ノードにはいずれも m5.2xlarge を利用し，Kafka では 3 台，Mosquitto では 1 台用いた．リクエスタ用ノードには，m5.xlarge を 1 台ずつ用意した．ベンチマークプログラムには，Kafka では Kafka のパッケージで用意されているプログラムを利用し，Mosquitto では MQTT-Bench [34] を用いた．

図 9 と**図 10** に Kafka と Mosquitto の Write スループットを 1 秒あたりの総送信メッセージ数 [msg/s] およびバイト数 [MB/s] で示す．横軸は，実験の条件を <MQTT/Kafka>-<data size>-<qos/ack> と表記しており，“MQTT” は Mosquitto の結果，“Kafka” は Kafka の結果

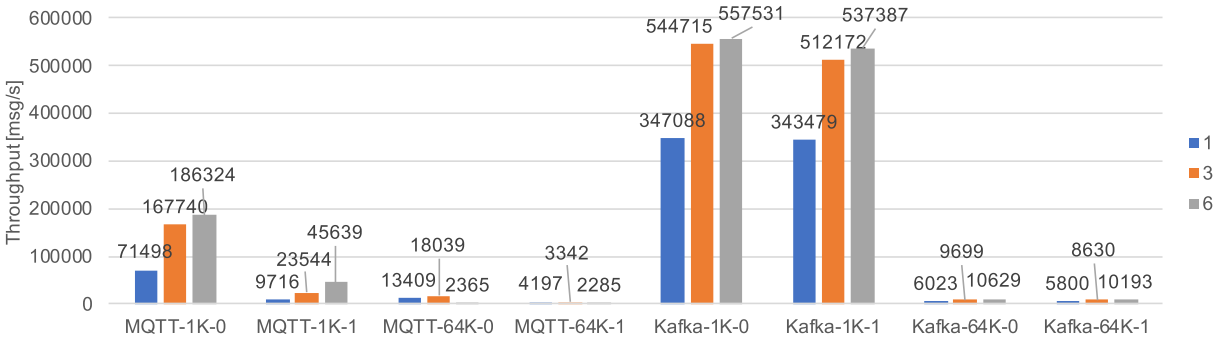


図 9　メッセージング基盤の Write スループット [msg/s].

Fig. 9　Write throughputs of messaging systems in msg/s.

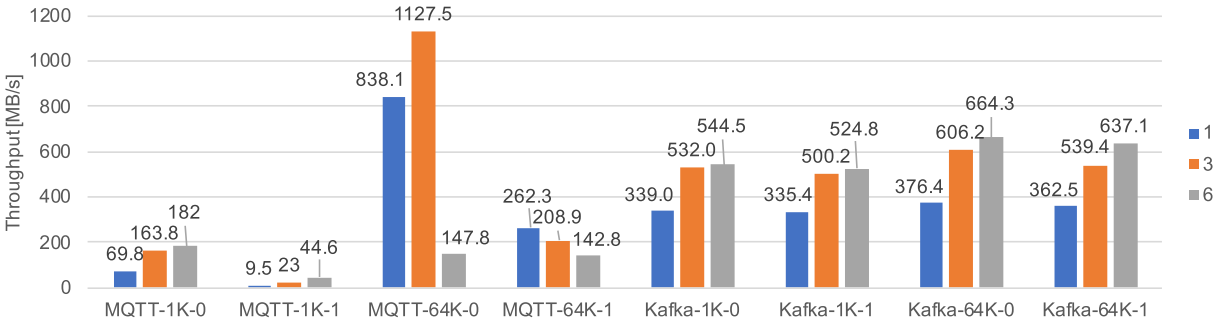


図 10　メッセージング基盤の Write スループット [MB/s].

Fig. 10　Write throughputs of messaging systems in MB/s.

を表す．二つめの数字は送信データサイズで，1 KiB と 64 KiB の結果を比較する．三つめの数字は MQTT の QoS レベルであり，0 は到着確認なし，1 は Kafka の ack 相当の QoS1 の結果を示している．Kafka では，ack の有無により 1 または 0 を表記している．青色，橙色，灰色の 1，3，6 は，リクエスタ数を表している．

図 9，図 10 で Kafka と Mosquitto の結果を比較すると，Kafka のほうが高性能であることが分かる．QoS および ack の有無の比較では，Kafka ではあまり性能劣化がみられないのに対し，Mosquitto では大幅にスループットが低下していることが分かる．図 10 の MQTT-64K-0 では高いスループット値を示しているが，MQTT-64K-1 ではスループット値が大幅に低下しており，送信メッセージの多くが Mosquitto の Broker で受信できていないと考えられる．よって，Mosquitto を用いてセンサ端末で収集したデータを欠落することなく Broker に収集したい場合には，少なくとも QoS1 の設定が必要であることが分かる．また，大量のデータを確実に収集する場合には，Kafka のほうが有望であることが示された．

一方，図 6 の最大値と図 10 の QoS/ack ありの条件で Kafka と Mosquitto の最大値をそれぞれ比較すると，Kafka で 13.1 倍，Mosquitto でも 5.4 倍のスループットとなっており，小規模データの大量処理ではメッセージング基盤の利用が非常に有効であることが明らかとなった．以上から，IoT のアプリケーションシナリオに応じて，適切なメッセージング基盤を選択することが望ましいと考えられる．

3.4　分散ストリーム処理基盤の予備評価

大量のストリーム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．よって，既存分散ストリーム処理基盤では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・運用できる機能を提供する．本稿では，分散メッセージング基盤の Kafka と分散ストリーム処理基盤の Flink および Storm についてオンライン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．

3.4.1　評価方法

図 11 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，遅延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 *Topic*1 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．

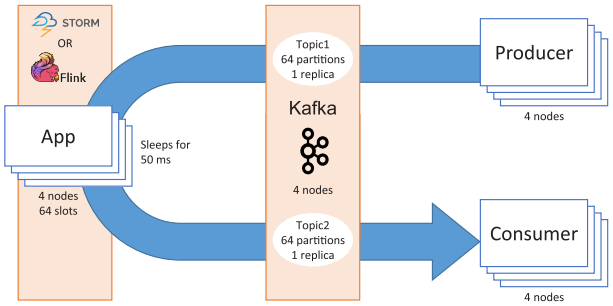


図 11　分散ストリーム処理実験の構成

Fig. 11　Configuration of the distributed stream processing experiment.

表 3　分散ストリーム処理基盤の評価環境．

Table 3　Server specifications for the distributed stream processing experiment.

Kafka Broker, Producer, Consumer（4 台）	
インスタンス	m5.2xlarge
VCPU	8
メモリ	32GiB
Network	最大 10Gbps
Flink Job Manager / Storm Master（各 1 台）	
インスタンス	m5.large
VCPU	2
メモリ	8GiB
Network	最大 10Gbps
Flink Task Manager / Storm Worker（各 4 台）	
インスタンス	m5.4xlarge
VCPU	16
メモリ	64GiB
Network	最大 10Gbps

メッセージを折り返す Flink アプリと Storm アプリはそれぞれ，各基盤上に実装された Java プログラムで，Kafka のトピック *Topic*1 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 *Topic*2 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 *Topic*2 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 3** 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，Kafka Broker と同じインスタンス上で一インスタンスにつき一つ，全体で四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 4** に実験で用いたパラメータをまとめた．

表 4　分散ストリーム処理基盤の評価実験で用いたパラメータ

Table 4　Parameter settings for the distributed stream processing experiment.

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

評価値として，往復遅延とスループットを算出した．往復遅延は，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．実験では，Kafka は v. 2.2.0，Flink は v. 1.7.2，Storm は v. 2.0.0 を用いた．

3.4.2　評価結果

往復遅延の累積分布を**図 12** に示す．左のグラフが Flink の結果，右は Storm の結果であり，縦軸は完了した処理要求の割合，横軸は遅延をミリ秒で表している．今回の評価ではペイロード量によって遅延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 1 FPS〜240 FPS（全体で 4 FPS〜960 FPS）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 ミリ秒以下の遅延で往復していることが確かめられた．一方，送信頻度が 320 FPS（全体で 1280 FPS）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．

図 13 には，Flink（左）と Storm（右）の 1 秒ごとの最大往復遅延観測値をミリ秒で示す．横軸は経過時間となっている．ここでは，ペイロード量 64 KB の場合の結果を示すが，他のペイロード量も同様の傾向を示していた．図 12 の結果で述べたように，320 FPS の結果では最大往復遅延値が大きくなっていくことが分かる．一方，1–240 FPS では最大往復遅延値が一定程度で収まっているが，最大遅延値が瞬間的に大きく上下する現象が観測された．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

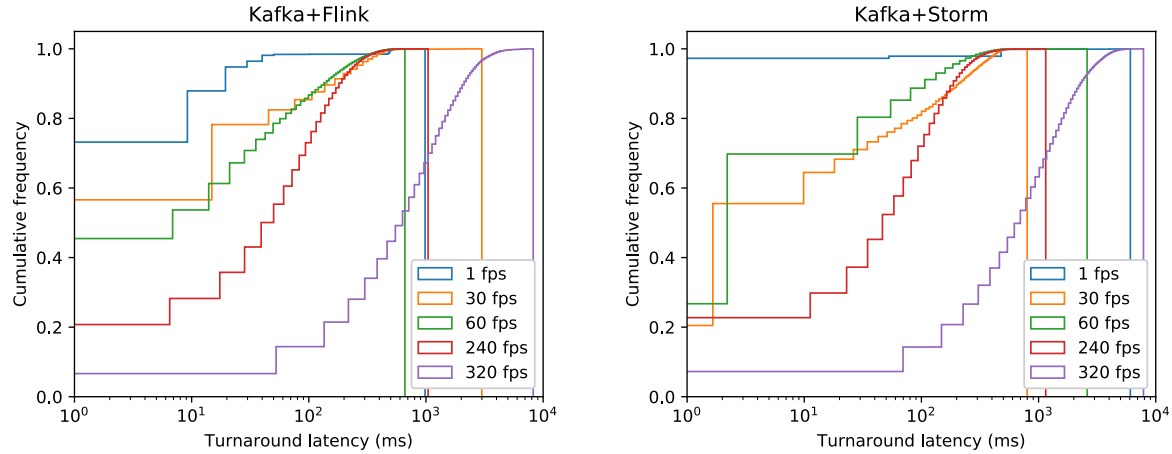


図 12　Flink（左）と Storm（右）の往復遅延の累積分布．すべてのペイロード量の合計．右端の垂線は最大遅延を示す．

Fig. 12　Cumulative distribution of round trip latencies of all payload settings using Flink (left) and Storm (right). The vertical lines at the right end show the maximum latencies of each frame rate.

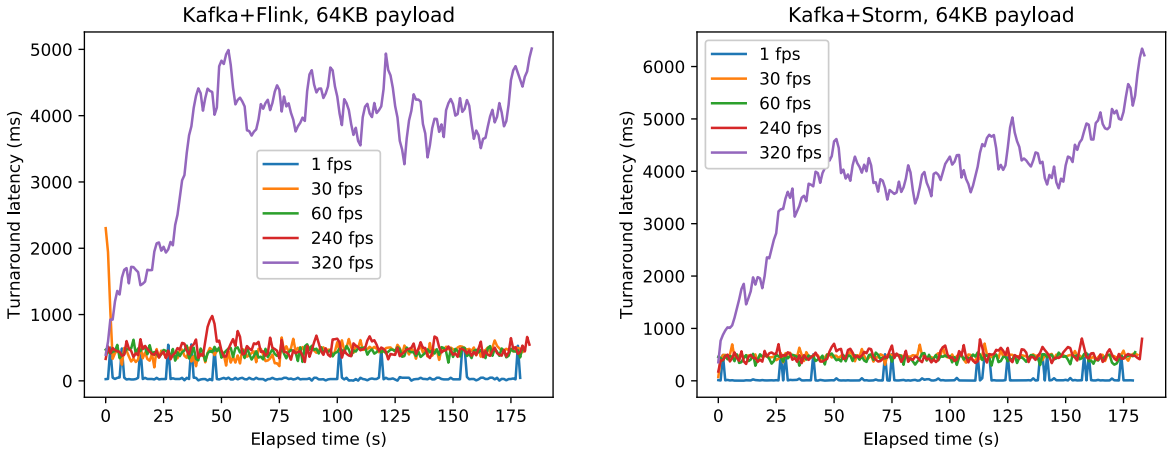


図 13　Flink（左）と Storm（右）の 1 秒ごとの最大往復遅延観測値．ペイロード量 64 KB の場合．

Fig. 13　Maximum round trip latencies per second when the payload amount is 64 KB using Flink (left) and Storm (right).

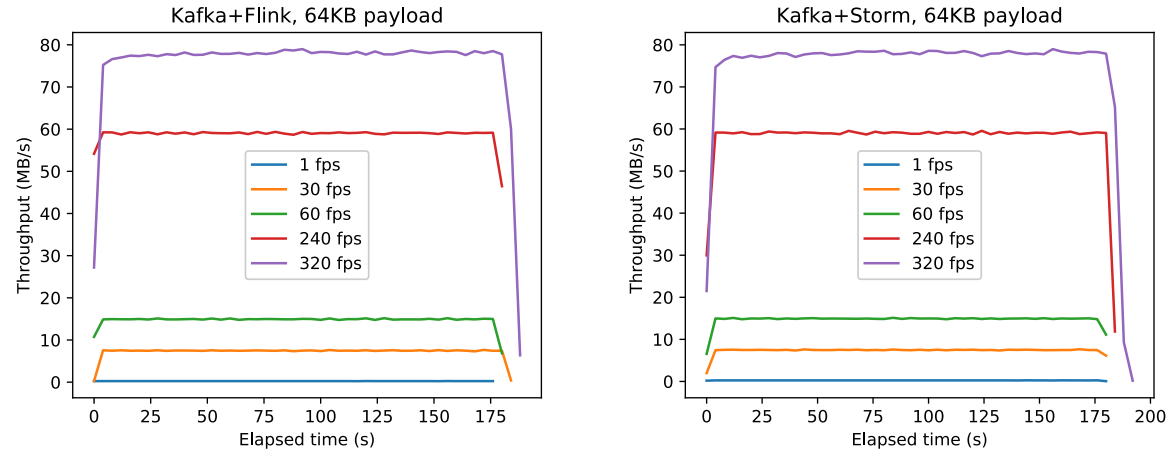


図 14　Flink（左）と Storm（右）のスループット．ペイロード量 64 KB の場合．

Fig. 14　Throughputs when the payload amount is 64 KB using Flink (left) and Storm (right).

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オンライン性を期待するべきではないといえる．

最後に，Flink と Storm のスループットの測定結果を図 14 に示す．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 $z = 64$ 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．図 14 から，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．以上から，多少のオーバヘッドがあるものの既存分散ストリーム処理基盤を用いることで低遅延，高スループットで処理が可能であることが示唆された．

4. 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 [35] やウェブサイト [36], [37] 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 [38] 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 2 KB 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 64 KB 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，複数のストリーム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．ストリーム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 [39]，[40]．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ムデータ処理基盤を利用する選択肢もある．文献 [41] 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．

5. おわりに

オンラインビデオ処理では，センサとクラウド間のネットワーク帯域と安全性の確保，大量データの収集，クラウドでのオンラインビデオ解析処理の性能が課題となる．本研究では，SINET 広域データ収集基盤を用いたオンライン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 ノードを活用したオンライン処理が実現可能であることを示し，実証実験環境の構築および実施の際に用いたプログラムを「SINET 広域データ収集基盤デモパッケージ」として GitHub で公開した．

また，予備評価により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の基本性能と，Kafka と Mosquitto の性能，Flink と Storm の性能を調査した．実験から，大量のセンサデータを扱うにはオブジェクトストレージでは不十分でメッセージング基盤を活用することが望ましいこと，MQTT ベースの Mosquitto はクライアント用のライブラリが充実しているという利点もあるが，大量データのデータ収集には Kafka が優位であること，IoT アプリケーションの要件に合わせてソフトウェア構成を設計する必要があることを示した．また，分散ストリーム処理基盤では多少のオーバヘッドがあるものの，低遅延，高スループットでの処理が期待できることが分かった．広帯域，低遅延の 5G 環境とメッセージング基盤，分散ストリーム処理基盤を適切に組み合わせることで，秒単位の応答時間を許容する IoT アプリケーションの構築が可能であることが示された．

我々は，現在広域に分散するデータを収集，解析する研究を支援するソフトウェアパッケージ SINETStream を開発し，オープンソースとして公開している [42]，[43]．本稿で示したように，メッセージング基盤ミドルウェアごとに性能特性が異なるため，IoT アプリケーション開発初期の段階で適切なミドルウェアを選択するのは難しい．しかし，メッセージング基盤ミドルウェアごとに API が異なるため，ミドルウェアの変更には大幅なプログラムの書き換えが必要となる．よって，SINETStream ではそれらを抽象化する API を提供している．また，認証・認可，暗号化といった安全性を高めるための機能も提供している．SINETStream により，安全かつ高効率な IoT アプリケーションの開発・運用を容易にする技術の開発を行っていく．

謝辞　本研究を進めるにあたり，貴重なご意見をいただいたジョージア工科大学の Calton Pu 教授，デモパッケージの開発および評価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．本研究成果の一部は，JSPS 科研費 JP19H04089 および，2019 年度国立情報学研究所公募型共同研究（19S0501）の助成を受けて実施されたものである．

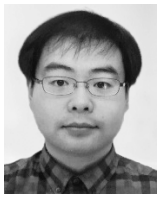
参考文献

[1] MQTT (Message Queue Telemetry Transport), http://mqtt.org/.
[2] SINET 広域データ収集基盤，https://www.sinet.ad.jp/wadci.
[3] 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).
[4] 孫　静涛，藤原一毅，竹房あつ子，長久　勝，吉田浩，合田憲人：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，情報処理学会研究報告，2019-OS-147, pp.1–9 (2019).
[5] Ichinose, A., Takefusa, A., Nakada, H. and Oguchi, M.: A Study of a Video Analysis Framework Using Kafka and Spark Streaming, *Proc. Second Workshop on*

Real-time & Stream Analytics in IEEE Big Data, pp.2396–2401 (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/TEL-NET.2017.8343593 (2017).
[8] Apache Kafka, https://kafka.apache.org/.
[9] Redmon, J. and Farhadi, A.: YOLOv3: An Incremental Improvement, *CoRR*, Vol.abs/1804.02767 (online), available from 〈http://arxiv.org/abs/1804.02767〉 (2018).
[10] YOLO, https://pjreddie.com/darknet/yolo/.
[11] A PyTorch implementation of the YOLO v3 object detection algorithm, https://github.com/ayooshkathuria/pytorch-yolo-v3.
[12] Wei, S.-E., Ramakrishna, V., Kanade, T. and Sheikh, Y.: Convolutional Pose Machines, *CVPR* (2016).
[13] Simon, T., Joo, H., Matthews, I. and Sheikh, Y.: Hand Keypoint Detection in Single Images using Multiview Bootstrapping, *CVPR* (2017).
[14] Cao, Z., Simon, T., Wei, S.-E. and Sheikh, Y.: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields, *CVPR* (2017).
[15] Cao, Z., Hidalgo, G., Simon, T., Wei, S.-E. and Sheikh, Y.: OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields, *arXiv preprint arXiv: 1812.08008* (2018).
[16] OpenPose, https://github.com/CMU-Perceptual-Computing-Lab/openpose.
[17] 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).
[18] 学認クラウド，https://cloud.gakunin.jp/.
[19] SINET 広域データ収集基盤デモパッケージ，https://github.com/nii-gakunin-cloud/wadci-demo.
[20] 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).
[21] Eclipse Mosquitto, https://mosquitto.org/.
[22] Apache Flink, https://flink.apache.org/.
[23] Apache Storm, https://storm.apache.org/.
[24] Apache Spark, https://spark.apache.org/.
[25] Karakaya, Z., Yazici, A. and Alayyoub, M.: A Comparison of Stream Processing Frameworks, *2017 International Conference on Computer and Applications (ICCA)*, pp.1–12 (2017).
[26] Nasiri, H., Nasehi, S. and Goudarzi, M.: Evaluation of distributed stream processing frameworks for IoT applications in Smart Cities, *J. Big Data*, Vol.6, No. 52, pp.1–24 (online), DOI: 10.1186/s40537-019-0215-2 (2019).
[27] AI/SUM SHOWCASE, https://aisum.jp/showcase/.
[28] 吉田　浩，合田憲人，上田郁夫，原　隆宣，小杉城治，森田英輔，中村光志：クラウドコールドストレージに対する大規模実験データ格納のケーススタディ，情報処理学会研究報告 2018-HPC-165, pp.1–10 (2018).
[29] goofys, https://github.com/kahing/goofys.
[30] Jupyter Notebook, http://jupyter.org/.
[31] Yokoyama, S., Masatani, Y., Ohta, T., Ogasawara, O., Yoshioka, N., Liu, K. and Aida, K.: Reproducible Scientific Computing Environment with Overlay Cloud Architecture, *Proc. 9th IEEE Cloud*, pp.774–781 (2016).
[32] 国立研究開発法人情報通信研究機構：製造現場における無線ユースケースと通信要件（要約版），技術報告，FLEXIBLE FACTORY PARTNER ALLIANCE (FFPA) (2017).
[33] COSBench - Cloud Object Storage Benchmark, https://github.com/intel-cloud/cosbench.
[34] MQTT-Bench: MQTT Benchmark Tool.
[35] 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).
[36] Wikipedia: Comparison of MQTT implementations, https://en.wikipedia.org/wiki/Comparison_of_MQTT_implementations.
[37] MQTT community website, server support, https://github.com/mqtt/mqtt.github.io/wiki/server-support.
[38] 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, pp.227–238 (online), DOI: 10.1145/3093742.3093908 (2017).
[39] 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, pp.1–10 (online), DOI: 10.1145/3148055.3148068 (2017).
[40] 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).
[41] 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, pp.360–363 (online), DOI: 10.1109/CCGRID.2018.00056 (2018).
[42] SINETStream, https://www.sinetstream.net/.
[43] 竹房あつ子，孫　静涛，藤原一毅，吉田　浩，合田憲人：IoT ストリームデータ処理のためのソフトウェアライブラリ SINETStream の開発，情報処理学会研究報告 2020-IOT-48, pp.1–8 (2020).

竹房 あつ子（正会員）

2000 年博士（理学）（お茶の水女子大学）取得．日本学術振興会特別研究員（DC2, PD），お茶の水女子大学大学院助手，産業技術総合研究所研究員，主任研究員を経て 2016 年より国立情報学研究所アーキテクチャ科学研究系准教授，総合研究大学院大学複合科学研究科情報学専攻准教授兼務．並列分散処理，グリッド，クラウド，エッジ，IoT に関する研究に従事．ACM，IEEE，電子情報通信学会各会員．



孫 静涛（正会員）

1987 年生．2013 年東京工科大学大学院バイオ情報メディア研究科コンピュータサイエンス専攻博士課程前期修了．2016 年総合研究大学院大学複合科学研究系情報学専攻博士課程後期修了．博士（情報学）．同年国立情報学研究所アーキテクチャ科学研究系特任研究員．分散システム，クラウドコンピューティング，IoT の研究に従事．ACM，IEEE，CCF 各会員．

藤原 一毅（正会員）

国立情報学研究所オープンサイエンス基盤研究センター特任准教授．博士（情報学）．2012 年，総合研究大学院大学複合科学研究科情報学専攻を修了．同年より，国立情報学研究所アーキテクチャ科学研究系特任研究員として計算機ネットワークを研究．2016 年より，情報通信研究機構ユニバーサルコミュニケーション研究所データ駆動知能システム研究センター主任研究員として自然言語処理向け高性能計算システムを設計．2018 年より現職にて NII Research Data Cloud を構成するデータ解析システムの開発に従事．

長久 勝（正会員）

ライフマティックス株式会社ライフサイエンス事業部リードエンジニア・主任研究員．1994 年 3 月龍谷大学理工学部数理情報学科卒業．1994 年 3 月株式会社エス・エヌ・ケイ入社．以降，コンピュータゲーム開発や動画配信などに従事．奈良女子大学，早稲田大学にて非常勤講師．国立情報学研究所にてトップエスイーやクラウド関連の業務に従事．2019 年 4 月よりライフマティックス株式会社に在籍．日本デジタルゲーム学会会員．

吉田 浩（正会員）

1978 年東京大学工学部電子工学科卒業．1980 年同大学大学院工学系研究科修士課程修了．富士通株式会社において，ソフトウェア，ストレージシステム，パブリッククラウドサービスの企画・開発に従事．2015 年より国立情報学研究所クラウド基盤研究開発センターにおいて，クラウド関連の研究・開発および大学・研究機関のクラウド導入支援に従事．IEEE 会員．

政谷 好伸

名古屋大学理学部物理学専攻科修了，NTT 電気通信技術研究所，インターネットサービスプロバイダ，システムインテグレータなど勤務．2014 年より国立情報学研究所先端 ICT センター特任研究員．クラウド関連の研究・開発・運用業務に従事．ACM 会員．

合田 憲人（正会員）

1997 年博士（工学）（早稲田大学）取得．1992 年早稲田大学情報科学研究教育センター助手，1997 年東京工業大学大学院情報理工学研究科数理・計算科学専攻助手，1999 年同大学大学院総合理工学研究科知能システム科学専攻講師，2003 年同研究科物理情報システム専攻助教授，2007 年国立情報学研究所特任教授，2015 年同教授，現在に至る．科学技術振興機構さきがけ研究員（2001 年〜2005 年），ハワイ大学 Information and Computer Sciences Department 客員研究員（2007 年），東京工業大学大学院総合理工学研究科物理情報システム専攻連携教授（2007 年〜2016 年），総合研究大学院大学複合科学研究科情報学専攻教授（2008 年〜現在）等を兼任．電子情報通信学会，IEEE，ACM 各会員．