「マルチメディア，分散，協調とモバイル
(DICOMO2019) シンポジウム」 令和元年7月

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

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

概要：ビデオデータのリアルタイム解析は，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，クラウドでのリアルタイム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が実現可能であることを示す．また，予備評価によりリアルタイム処理基盤を構成するメッセージング基盤の性能特性を示す．

Consideration of Real-time Video Processing Framework using SINET Wide-area Data Collection Infrastructure

Atsuko Takefusa[1] Jingtao Sun[1] Masaru Nagaku[1] Hiroshi Yoshida[1] Yoshinobu Masatani[1] Kento Aida[1]

1. はじめに

ビデオデータのリアルタイム解析は，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．高度なリアルタイム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，クラウドの大規模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ネットワークの帯域と安全性
(2) 大量データの収集性能
(3) クラウドでのリアルタイムビデオ解析処理の性能

(1) は，大量のセンサから送付されるデータをクラウドの計算資源に対して送信可能であることや，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．(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，リアルタイム処理をするにはストリーミング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．

本研究では，SINET 広域データ収集基盤 [2] 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て実証実験を行うとともに予備評価を行い，リアルタイム処理基盤を構成するメッセージング基盤の性能特性を示す．

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

[1] 国立情報学研究所
National Institute of Informatics

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 ノードを用いたリアルタイム処理を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 [3] 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 [4], [5], [6] を採用した．また，提案するリアルタイムビデオ処理機構のアプリケーションとして，収集したストリーミングデータからオブジェクト抽出するライブラリ YOLO v3 [7], [8] の PyTorch 実装 [9] と，人のキーポイント抽出ライブラリ OpenPose [10], [11], [12], [13], [14] 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，ほぼリアルタイムに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」[15], [16] 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 [17]．

予備評価では，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と，Kafka と MQTT ブローカ実装の 1 つである Eclipse Mosquitto [18], [19] 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 の適していること，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．

2. リアルタイム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，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 [20] 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「広域データ収集基盤」の実証実験を開始した．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．

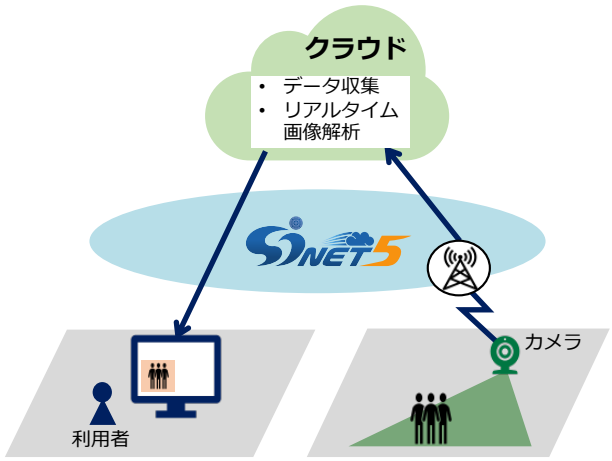

図 1　SINET モバイル網を用いたリアルタイムビデオ処理実証実験の概要．

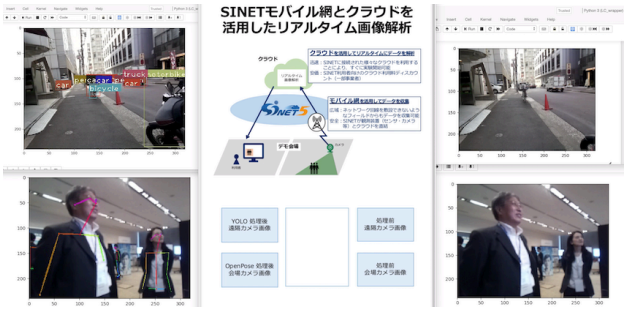

図 2　実験結果のスナップショット．

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 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．ここで，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 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 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，64KB 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 6fps 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，6fps のデータに対するリアルタイム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リアルタイム処理を行った．

2.2 プロトタイプシステムのソフトウェア構成

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

実験では，動画像は静止画のストリーミングデータとして配信・収集する．そのためのメッセージング基盤には，Apache Kafka（以降，Kafka と呼ぶ）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ミングデータを，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．データをどう利用するか予め想定し，データを格納するパスを設計しておく必要がある．

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 枚ずつリアルタイムに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 30fps 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リアルタイム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

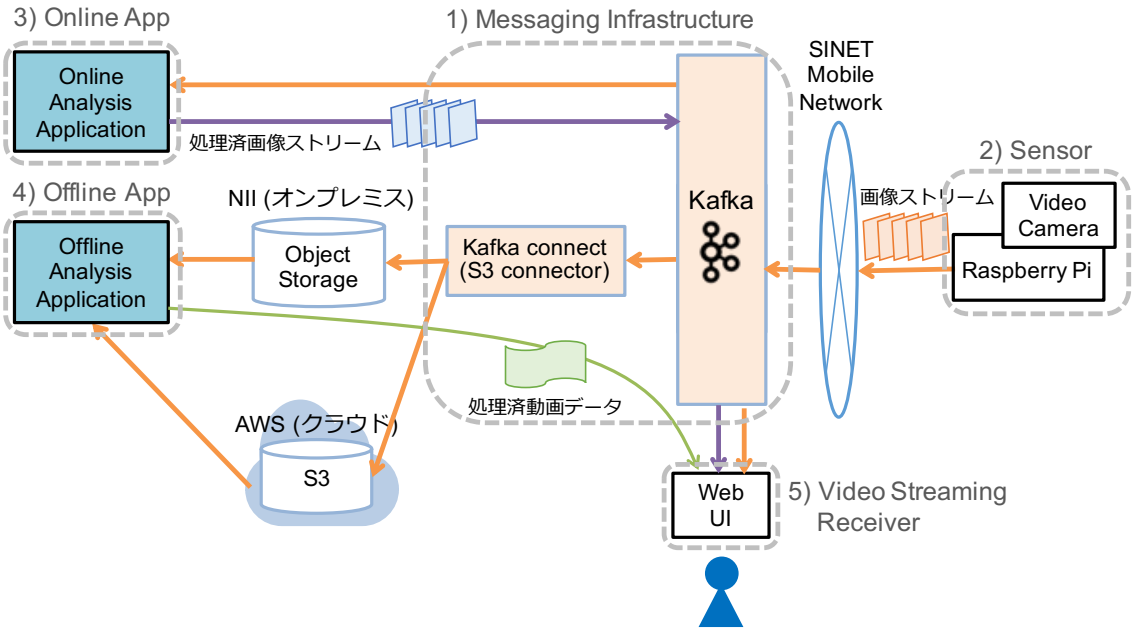

図 3　リアルタイムビデオ処理機構プロトタイプシステムのソフトウェア構成.

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．リアルタイム処理のため，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

本研究ではリアルタイム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 [21] 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 [21]．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 1 つの動画を生成するようにした．

5）ストリーミング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ミングデータを動画像として表示させる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．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 [22] で実装しており，表示したい Topic や表示レートなどがその場で適宜

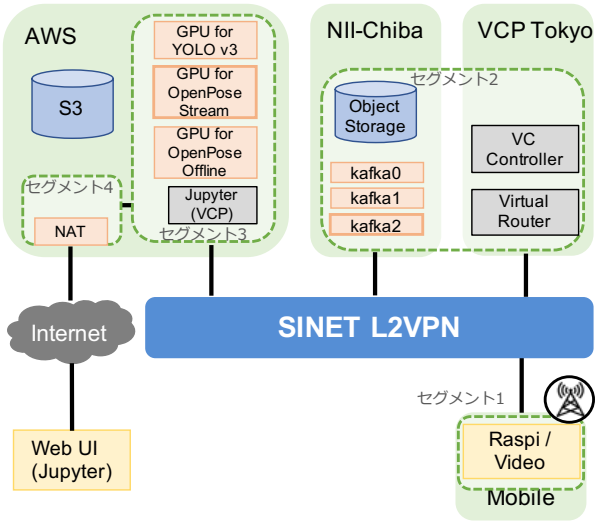

図 4　実証実験環境.

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

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.

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

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) [23] 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用いて環境構築する必要があるため，本サービスを用いた．

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 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．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，「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．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

SINET 広域データ収集基盤においてリアルタイム処理基盤を構築するための性能指針を得ることを目的とし，オブジェクトストレージの IO 性能とメッセージング基盤の性能を調査する．

3.1 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，864TB（実効容量 580TB）構成であり，Inter-rack ネットワーク帯域は 40Gbs，Intra-rack ネットワーク帯域は 10Gbs 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 1 つである COSBench [24] 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 64KiB，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 16GiB メモリの VM 3 台，AWS では vCPU 2 コア，8GiB 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 10Gbps 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 10Gbps 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わかる．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方が大きいが，リクエスタ数が多くなると S3 の方が大きくなり，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，1Gbps 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．

3.2 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 1 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

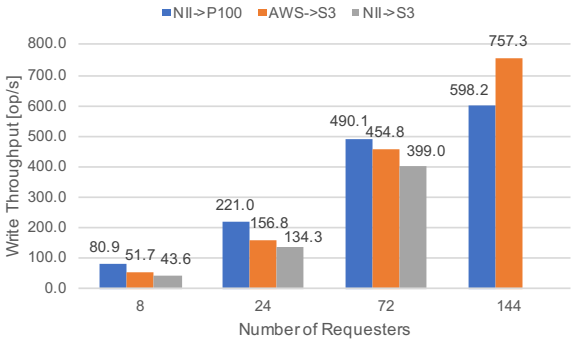

図 5 Write スループット [op/s].

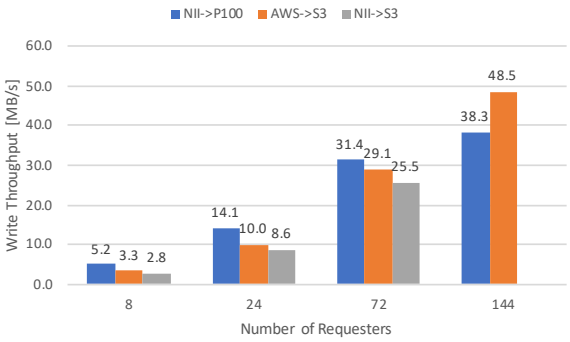

図 6 Write スループット [MB/s].

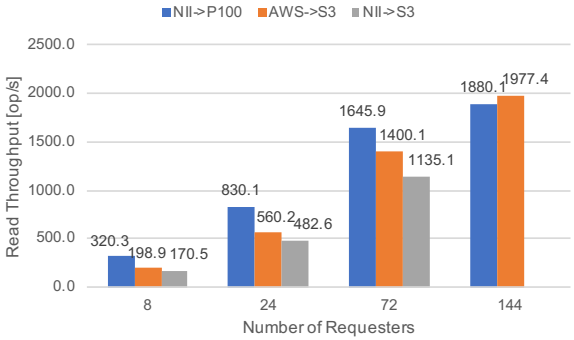

図 7 Read スループット [op/s].

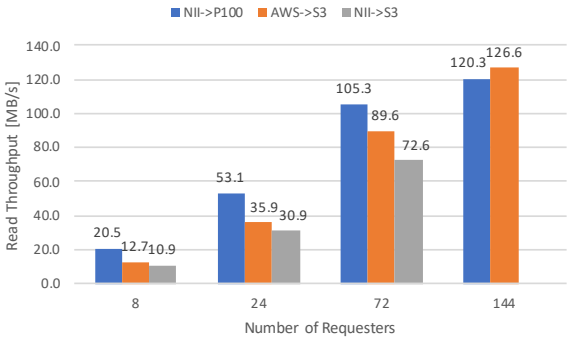

図 8 Read スループット [MB/s].

表 2 メッセージング基盤の評価環境.

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

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．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．

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[25] 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 の結果を表す．2 つめの数字は送信データサイズで，1KiB と 64KiB の結果を比較する．3 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 は 3 台構成の Broker クラスタを用いているため高スループットとなることは明らかであるが，QoS および ack の有無で比較すると Kafka ではあまり性能劣化が見られないのに対し，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 の方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

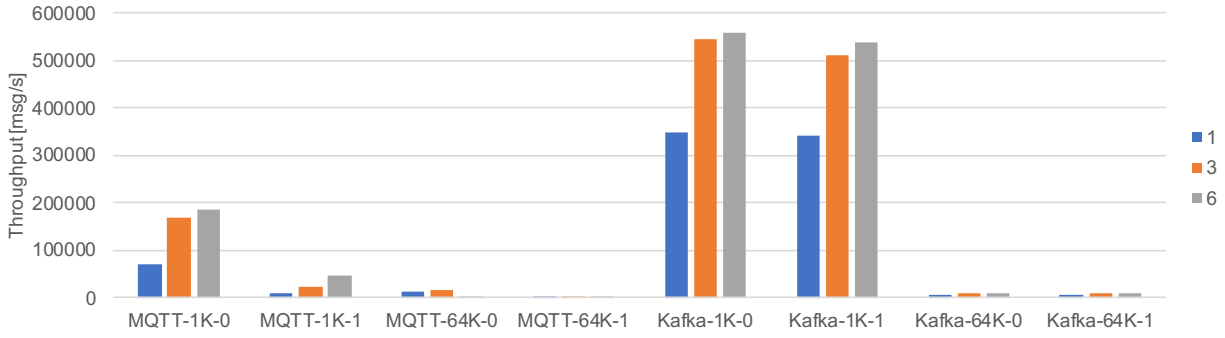

図 9 メッセージング基盤の Write スループット [msg/s].

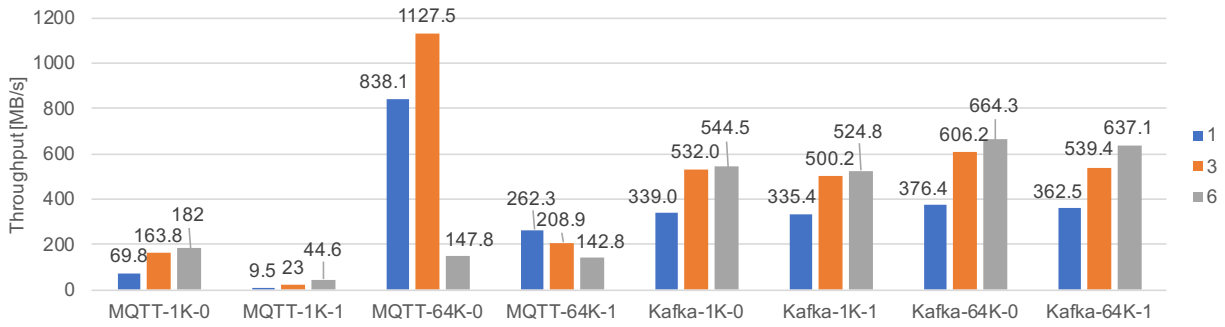

図 10 メッセージング基盤の Write スループット [MB/s].

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．

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 [26] やウェブサイト [27], [28] 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 [29] 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 [30], [31]．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 [32] 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

リアルタイムビデオ処理では，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，クラウドでのリアルタイム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「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 の適していること，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．

今後は，Broker クラスタ構成が可能な MQTT 実装の評価や，より高負荷な処理を可能にする Flink や Heron 等の

ストリーミングデータ処理基盤を活用する際の性能要件の明確化，Android 等をターゲットとしたクライアントプログラムの開発等を進め，SINET 広域データ収集基盤を活用したリアルタイム処理基盤を構築するための設計指針を示す．

謝辞　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．本研究にご協力いただいた数理技研の小泉敦延様に深く感謝いたします．

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] A. Ichinose and A. Takefusa and H. Nakada and M. Oguchi: 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).
[4] Kreps, J., Narkhede, N. and Rao, J.: Kafka : a Distributed Messaging System for Log Processing, *NetDB workshop 2011*, pp. 1–5 (2011).
[5] 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).
[6] Apache Kafka, `https://kafka.apache.org/`.
[7] Redmon, J. and Farhadi, A.: YOLOv3: An Incremental Improvement, *CoRR*, Vol. abs/1804.02767 (online), available from ⟨http://arxiv.org/abs/1804.02767⟩ (2018).
[8] YOLO, `https://pjreddie.com/darknet/yolo/`.
[9] A PyTorch implementation of the YOLO v3 object detection algorithm, `https://github.com/ayooshkathuria/pytorch-yolo-v3`.
[10] Wei, S.-E., Ramakrishna, V., Kanade, T. and Sheikh, Y.: Convolutional pose machines, *CVPR* (2016).
[11] Simon, T., Joo, H., Matthews, I. and Sheikh, Y.: Hand Keypoint Detection in Single Images using Multiview Bootstrapping, *CVPR* (2017).
[12] Cao, Z., Simon, T., Wei, S.-E. and Sheikh, Y.: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields, *CVPR* (2017).
[13] 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).
[14] OpenPose, `https://github.com/CMU-Perceptual-Computing-Lab/openpose`.
[15] 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).
[16] 学認クラウド，`https://cloud.gakunin.jp/`.
[17] 広域データ収集基盤デモパッケージ，`https://github.com/nii-gakunin-cloud/wadci-demo`.
[18] 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).
[19] Eclipse Mosquitto, `https://mosquitto.org/`.
[20] AI/SUM SHOWCASE, `https://aisum.jp/showcase/`.
[21] goofys, `https://github.com/kahing/goofys`.
[22] Jupyter Notebook, `http://jupyter.org/`.
[23] S. Yokoyama and Y. Masatani and T. Ohta and O. Ogasawara and N. Yoshioka and K. Liu and K. Aida: Reproducible Scientific Computing Environment with Overlay Cloud Architecture, *Proc. 9th IEEE Cloud*, pp. 774–781 (2016).
[24] COSBench - Cloud Object Storage Benchmark, `https://github.com/intel-cloud/cosbench`.
[25] MQTT-Bench: MQTT Benchmark Tool.
[26] 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).
[27] Wikipedia: Comparison of MQTT implementations, `https://en.wikipedia.org/wiki/Comparison_of_MQTT_implementations`.
[28] MQTT community website, server support, `https://github.com/mqtt/mqtt.github.io/wiki/server-support`.
[29] 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).
[30] 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).
[31] 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).
[32] 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).