JACN 学術情報処理研究
Journal for Academic Computing and Networking
No. 26
2022年 pp. 20-31
DOI: 10.24669/jacn.26.1_20

論　文

パブリッククラウドを活用した大規模センサデータ集約システムの構築

林 豊洋[1)], 冨重 秀樹[2)], 福田 豊[1)]

1）九州工業大学情報統括本部 情報基盤センター，2）九州工業大学管理本部 技術部

Deployment of Large Scaled Sensor Data Aggregation System Utilized Public Cloud Infrastructure

Toyohiro Hayashi[1)], Hideki Tomishige[2)], Yutaka Fukuda[1)]

1) Information Science and Technology Center, Information Infrastructure Management Headquarters of Kyushu Institute of Technology, 2) Technical Support Department, Management Headquarters of Kyushu Institute of Technology

概　要：九州工業大学では，全ての講義室と多くの共用スペースに無線 LAN のアクセスポイント（AP）が整備されている．AP が学内に網羅されていることから，各 AP への接続端末数と AP のカバーする面積を用いた密集度が定義できることに着目し，地図情報と共に密集度を図示するシステムを公開している．接続端末数は密集度以外にも応用可能なセンサデータの一種であることから，より大規模なセンサデータ集約システムの構築を検討した．特に，多数のセンサ，プロトコル，フォーマットで構成されたデータを受け入れ可能であり，同様に多様なシステムへのデータ出力が可能なシステムの構築を目的とした．本稿では，上記の仕様を考慮したパブリッククラウドを活用した大規模センサデータ集約システムの構築について言及する．

キーワード：センサデータ集約，パブリッククラウド，無線 LAN

1　はじめに

IoT（Internet of Things）やデータサイエンス分野において，情報機器の持つ状態を収集し，それらを大規模なセンサデータとみなし有用な情報源とする研究や活用例の報告がなされている．九州工業大学（以下，本学と称する）においても，携帯端末電波を情報源とした空間の滞留人数可視化の実証実験等を行っている [1]．

筆者らも，2020 年より始まった新型コロナウイルス感染症予防対策として，情報機器から収集されたセンサデータの活用を検討した．学内に設置された各 AP への接続情報を SNMP を用いて収集し，接続情報から室内における人の密集度を算出・表示するシステムを構築し，2020 年 7 月より学内向けに公開している．構築したシステムによって人の密集度を事前に得ることが可能となり，感染症予防対策の一つとして寄与することとなった．

構築システムは，AP の接続情報と密集度を合わせた 984 データを 5 分間隔で収集している．しかし，システムの性能限界に達しており，収集データの追加や収集間隔の短縮が行えない状況にある．加えて，構築システムは密集度の表示に特化したシステムであり，密集度を図る指標となる新たなセンサ（CO2 センサ等）の追加が困難である．

したがって，本論文ではより広範な機能を有し，大規模化が可能な構造を有するセンサデータ集約システムを新たに構築する．具体的には，「多数の機器・センサからの入力を遅延なく受け付ける構造（従来の 3 倍規模である 3000 データを 5 分間隔で収集可能）」「分析・可視化・長期保存等の各用途に適したデータ記憶が可能な構造」「高可用性を有する基盤での稼働（センサ入力時のエラー

が生じた際の再送機能等を有する)」が可能な大規模センサデータ集約システムの構築を行う．

これらの要件を満たし，簡便な手法で開発が行えることを指向し，稼働基盤はパブリッククラウドが提供するPaaS（データ入出力，データベース等），サーバレスアーキテクチャ（機能間のデータフローを制御する関数の実行等）の各種サービスを組み合わせ構築する．本学では，Microsoft 社のパブリッククラウド基盤である Azure 上に実システムを構築した．実システムは，目標であった従来システムの 3 倍規模のデータを収集し，可視化システム（Grafana）との協調や入力データの長期保存を可能とした．本稿において，パブリッククラウドを活用した大規模センサデータ集約システムの構築について詳細を述べる．

2　先行研究：無線 LAN 接続情報を利用した密集度表示システム

本節では，先行研究である無線 LAN 接続情報を利用した密集度表示システム [2] の概要と運用後の課題について述べる．

2.1　無線 LAN 接続情報を利用した密集度表示システムの概要

本学では，新型コロナウイルス感染症予防対策の一環として，キャンパス内の入構者数と，講義室等における人の密集度を計測する手法の検討を進めていた．計測手法のひとつとして，学内の無線 LAN 接続情報を利用して人の密集度をリアルタイムに表示するシステム(以下，密集度表示システムと称する）を開発し，学内向けに公開している．

本学では，全ての講義室と多くの共用スペースに無線 LAN のアクセスポイント（AP）を設置している．また，AP はキャンパスそれぞれに設置している無線 LAN コントローラに収容・統合管理されており，無線 LAN コントローラはアクセスポイントから端末接続情報（接続時間，接続 AP 等）が取得可能である．網羅的に設置された AP 毎の接続情報に加え，設置場所や面積の情報を収集することにより，ある時間における人の密集度が算出可能である．密集度表示システムは，本学の無線 LAN 基盤の整備状況に着目した計測システムであると言える．

密集度表示システムの構成は，機能ごとに Linux OS 上にシステムを構築し，各システムが協調することにより利用者に表形式や地図形式により密集度情報を提供する（図 1）．

図 1　密集度情報の可視化

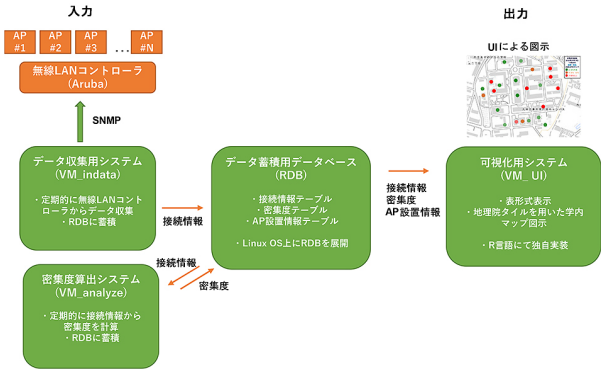

図 2　密集度表示システムの構成

図 2 に各システムが担う機能とシステム間の接続関係を示す．システムは，データ収集用システム（図 2，VM_indata），密集度算出システム（図 2，VM_analyze），データ蓄積用データベース（図 2，RDB），可視化用システム（図 2，VM_UI）で構成される．各システム上で稼働するプログラムは，データ抽出用のシェルスクリプトと可視化用の R 言語を主体として，独自実装している．

端末の接続情報および密集度の提供に必要な情報（AP

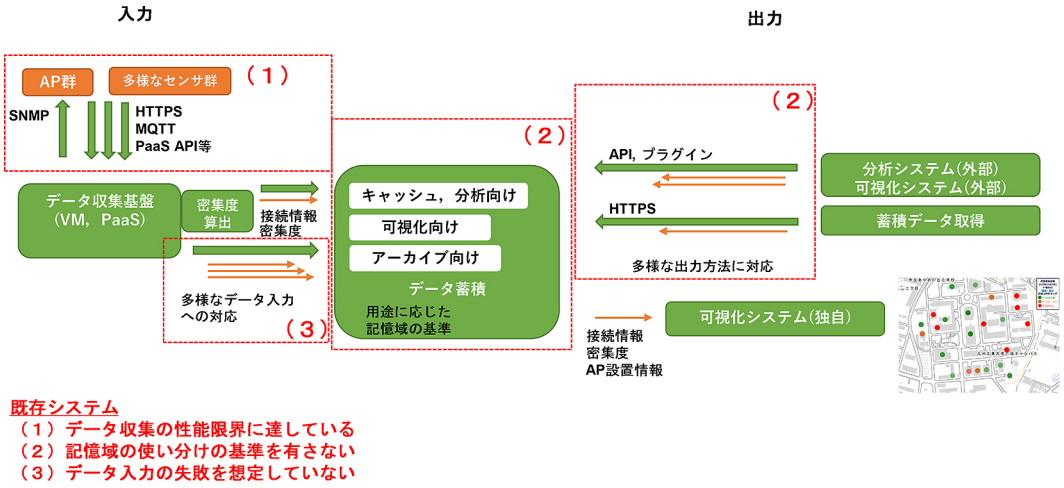

図３　密集度表示システムの機能拡張に対する課題

の名称，接続台数，計算した密集度等）は，事前に選定を行い，データ蓄積用データベース上でスキーマを定義する．

データ収集用システムは周期的（本学では 5 分毎）に無線 LAN コントローラより SNMP を用いて MIB を取得し，端末の接続情報を抽出する．抽出したデータはデータ蓄積用データベース内の接続情報テーブルに追加する．密集度算出システムは周期的（本学では 5 分毎）にデータ蓄積用データベースより接続情報と AP 位置情報を取得し，密集度を算出する．算出したデータは密集度テーブルに追加する．

可視化用システムは利用者向け UI を提供し，利用者が指定した検索条件（時間や設置場所）に応じて，データ蓄積用データベースから対応する情報を抽出し，表や地図上に密集度情報を出力する．

2.2　密集度表示システムの機能拡張に対する課題

前述の通り密集度表示システムは，無線 LAN コントローラから SNMP を用いて収集した接続情報に基づき密集度を算出し，利用者向け Web インタフェース（表形式，マップ図示），管理者向け Web インタフェース（時系列データ）による表示機能を提供している．有用なデータを集約したシステムであり，今後はより多くの無線 AP の収容や新たなセンサデータの入力，外部システムとの連携，公開を想定したデータ出力等を目的とした拡張が考えられる．

しかし，密集度表示システムは，システム構成の拡張のみではこれらに対応することが困難である．以下に懸念点を示す．

2.2.1　多数の機器・センサからの入力への対応（図３(1)）

密集度表示システムでは，学内に設置された AP から 492 台を選択し，5 分間隔で接続情報の抽出ならびに接続情報から密集度の算出を行い収集を行う．すなわち，5 分以内に 984 データを収集する状況にあるが，既存のシステム構成ではセンサ数の増加や収集間隔の短縮が行えない性能限界に達している．一方，学内に設置される AP の増加が見込まれることや，室内の状況を測定する新たなセンサ（CO2 センサ等）を設置した際のデータ収集を考慮すると，性能的な余力が必要となる．

2.2.2　用途に適したデータ記憶への対応（図３（2））

密集度表示システムでは，RDB に保存されたデータについては密集度の可視化にのみ利用する．他の活用は想定していなかったことから，保存したデータの保持期限やアクセス速度については明確な基準を有していない．なお，前述の 984 データを 5 分間隔で収集した際，保存されるデータ量は年間で 28（GB）程度となる．

可視化向けには高速なキャッシュ領域に保存されることが望ましいが，データの長期保存を行う際は低速な領域で十分である（低速なストレージは安価で維持可能である）．したがって，データの利用目的に応じた記憶域を使い分けることが望ましい．

また，密集度は VM_UI にて可視化システムを稼働させ，表形式・マップへの図示形式にて表示する．AP への接続情報，AP の設定箇所データならびに密集度の保存先は RDB となる．RDB からデータを取得と可視化システムへのデータ出力については，シェルスクリプトと R 言語を用いたハードコーディングで実装しており，外部システムとの連携は想定されていない．他の可視化システムや分析システムの API やデータ入力プラグインに対応することにより，豊富な描画機能を備えたデータの可視化やセンサデータからの傾向分析等が実現可能となる．

2.2.3 高可用性を有する基盤での稼働（図 3（3））

センサデータの収集を行う際はデータの取りこぼしのない構造を有することが重要となるが，密集度表示システムでは考慮されていない．システムへのデータ送信に失敗した際は，対象のデータを再送する仕組みが必要となる．また，システムを構成する各機能群（データ入力，記憶域等）が冗長性を有し，高い稼働率で動作することが望ましい．

前節で述べた先行研究の機能拡張に対する課題への対処には，機能群のスケールアップのみでは拡張が困難であると判断し，より広範な機能を有し，大規模化が可能な構造を有するセンサデータ集約システムを新たに検討する（図 3）．本研究では，先行研究の機能拡張に対する課題に対する対応目標を以下と定める．

多数の機器・センサからの入力への対応　先行研究に対して AP の設置個所が倍増し，加えて CO2 センサ等が追加されることを想定し，3 倍の性能（5 分以内に 3000 データを収集）を目標とする．また，センサごとに異なるデータフォーマット入力を考慮し，スキーマを意識しないデータ蓄積が可能なシステム構成を目標とする．

用途に適したデータ記憶への対応　記憶域として，入力データ蓄積用途の高速・低遅延記憶域，外部システムへの連携用途の API 連携用記憶域，アーカイブ用途の長期保存用記憶域を備えることを目標とする．

高速・低遅延記憶域については，システムへ入力されたセンサデータの保存が開始されるまでの遅延時間がミリ秒オーダであることを目標とする．API 連携用記憶域については，データの可視化システムとして広範に利用されている Grafana[5] と接続できることを目標とする．長期保存用記憶域は，データの保存期限を無期限とし，JSON 等の形式でセンサデータが保存されることを目標とする．

高可用性を有する基盤での稼働　取りこぼしなくデータ受信が行えるシステム基盤を前提とし，入力データの受信に失敗した場合は対象のデータの再送が可能であることを目標とする．

また本研究では，システムの稼働にはパブリッククラウドの活用を前提とし，データ入力やデータ蓄積等は，それぞれに特化したパブリッククラウド上の機能（PaaS）を用いる．PaaS を用いることにより，豊富な機能を簡便な手順で用いることができることや，冗長構成が前提であることから，安定性や可用性の確保が期待される．データの入出力の制御については，PaaS へのデータ入力を条件として，制御プログラム（関数）が駆動するサーバレスアーキテクチャを用いて，同様に安定性や可用性を確保する．

先行研究のような仮想サーバや IaaS 上を用いて構築する手法では，各機能を実装する要素（OS，データベースシステム，冗長化手法の選択）の検討に時間を要する事や，稼働基盤の高コスト化が予測される．従って，本研究においてはパブリッククラウドの活用が妥当であると判断する．

2.3 システム構成

大規模センサデータ集約システムの構成について示す．システムは，センサデータ入力，記憶域（目的に応じて，低遅延・高速，API 連携用，長期保存用）の機能群と，各機能間のデータ入出力の制御を行う関数の実行環境が主要な構成要素となる．図 4 にデータ入出力の流れ，各種機能群の接続関係，駆動される制御プログラムの関係について示す．

3 大規模センサデータ集約システム

以下に，各種機能群の利用目的と，対応するパブリッククラウドの機能について示す．本研究ではパブリッククラウドとして Microsoft Azure を用いており，機能は Azure 上の実装・提供となる．

なお，Azure に依らずパブリッククラウド基盤の選択肢は存在し，Amazon Web Services（AWS）の選択も候補となる．規模や機能の充実度は Azure と同等であり，構築に用いた機能と同等の機能を有する [3] ことから，AWS での実装も可能と考えられる．しかし，本学は Azure とのネットワーク接続に L2VPN を用いた専用線接続を設定し，秘匿性の高い通信を実施していること [4] に加え，可視化システムである Grafana が連携用の API を有し，正常な動作の確認が取れた記憶域（Azure Data Explorer）が Azure に整備されていることから，本学においては Azure が唯一の構築可能な基盤となる．

センサデータ入力 / IoT Hub（図 4（a））

センサデータ入力部は，センサと本システムの最初期のインタフェース部分となる．センサデータ入力部は，センサデータ数の増加に伴う同時接続を前提とし，入力時の遅延やエラーが生じないこと，データ収集の源泉であることから可用性が重要となるまた，SNMP に加え，HTTPS や MQTT 等のプロトコルへ対応できることが望ましい．これらの機能を満たすため，センサデータ入力部には Azure IoT Hub を用いる．IoT Hub は，IoT デバイスからのデータ収集に必要となる同時接続や，多様なプロトコルへの対応，高可用性を前提とした設計がなされたデータ収集基盤である [6]．なお，IoT Hub 自体は SNMP

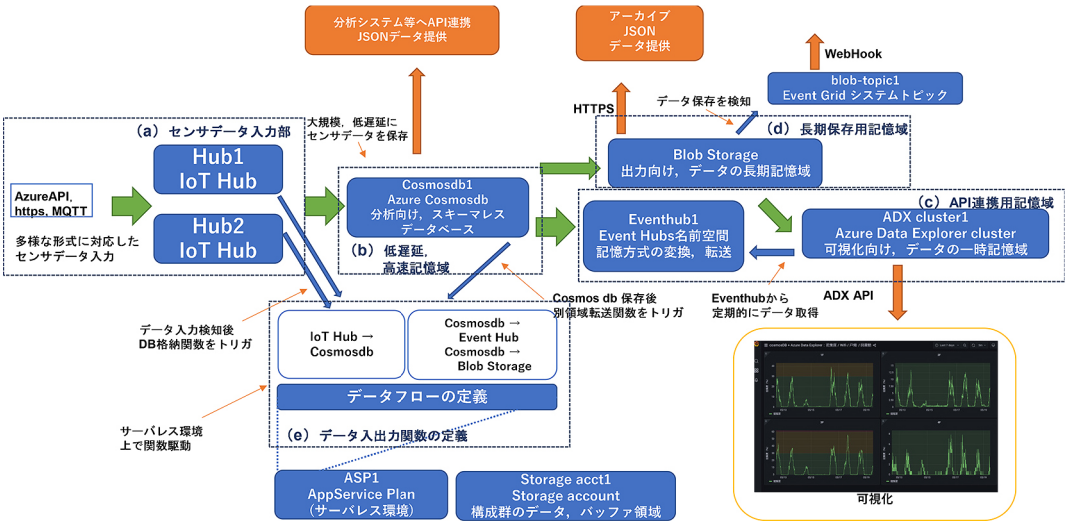

図 4　センサデータ集約システム構成図

プロトコルの入力に対応しておらず，SNMP による入力についてはデータ収集後 Azure IoT Hub API を用いて IoT Hub へ送信するシステムを構築し，集約システムに入力する構成とする．

本研究においては，先行研究で取り扱われた AP の接続情報，密集度に加え，二酸化炭素濃度（CO2 センサからの取得を想定）のセンサデータ入力を想定する．センサデータは以下に示す通り，センサが取得した計測値のほか，取得日時，地点の名称，センサの固有等で構成する．

・キャンパス名，建屋名，階数
・固有名
・データカテゴリ（接続情報 AP，密集度 Congestion，二酸化炭素濃度 CO2PPM）
・計測値
・取得タイムスタンプ

以降，入力されたセンサデータについては，データカテゴリ名 +_indata+ 番号を付し，AP_indata0，Congestion_indata0，CO2PPM_indata1 のように称して取り扱う．

記憶域

センサデータ入力部で受け付けたデータは，システム内の記憶域に保存する．本システムでは，利用目的に応じて以下の 3 種類の記憶域を使い分ける．

3.1　並列化を考慮したデータフロー，各記憶域への入力

1. **低遅延, 高速記憶域** / Cosmos DB（図 4（b））低遅延，高速記憶域はセンサデータ入力部の直後に設置する記憶域となる．

センサデータ入力部は多数のセンサの同時入力に対応した設計となるため，入力部に近い記憶域につ いても，低遅延かつ高速にアクセスできる構成とする．また，センサデータのフォーマット形式の変更や，新たなセンサデータの入力を実施する際，記憶域の軽微な仕様変更で対応できることが望ましい．

これらの機能を満たすため，低遅延，高速記憶域には Azure Cosmos DB を用いる．Cosmos DB は従来のデータベース（RDB）における記憶域の定義であるスキーマが不要であり，入力データを JSON フォーマットとして高速に保存することが可能なデータベースシステムである [7, 8]．また，機械学習を用いた分析システムとの API を有しており，センサデータからの傾向分析等への拡張も可能となる．ただし，Cosmos DB は記憶域の運用コストが比較的高額（従来の RDB の 2 倍程度）であることから，本システムではデータ保存期限は短期間（本学の構成では 7 日間）とし，後述の記憶域での中・長期の保存を実施する．

2. **API 連携用記憶域** / Data Explorer, Event Hub（図 4（c））低遅延，高速記憶域の後段に設置される記憶域となる．

本システムでは，保存されたセンサデータの外部システムへの連携（API，外部システムの連携プラグイン）を念頭に置くものの，低遅延，高速記憶域で用いる Cosmos DB に対応した外部システムは整備が進んでおらず，本システムでは Cosmos DB に保存されたデータを変換し，API 連携用記憶域に転送する構成とする．

これらの機能を満たすため，本システムでは API 連携用記憶域として Azure Data Explorer と Azure Event Hub をあわせて利用する．Azure Data Explorer は，JSON 形式等のデータ等を取り込み，データベース

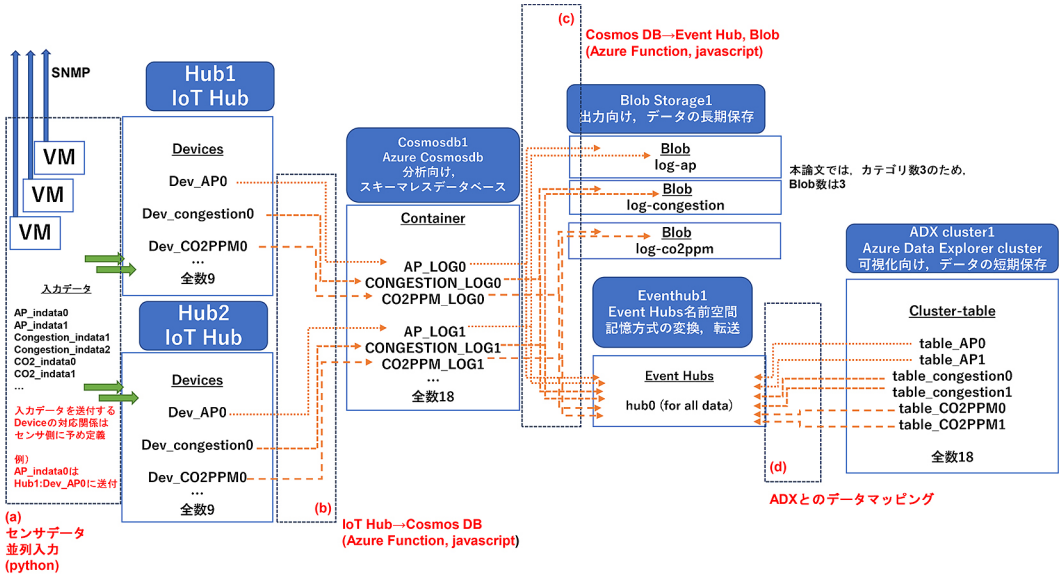

図 5　並列化を考慮したデータフロー，各記憶域の構造

として保存，多くの外部システムとの API を提供する基盤である．また，Event Hub は，Cosmos DB からデータを取得し，Data Explorer へ入力する際のデータマッピングと中継を行うために利用する [9, 10]．本学の構成では，データ保存期限は 30 日間としている．

3. **長期保存用記憶域** / Blob Storage（図 4（d））API 連携用記憶域と同様に，低遅延，高速記憶域の後段に設置される記憶域となる．

　データ入力から期間が経過したデータについてはアーカイブ用の記憶域に JSON 形式で保存する構成とする．アーカイブ用の記憶域はアクセス速度が低速であり，アクセス手段も HTTPS に限定化される．ただし，安価なアーカイブ用の記憶域を用いるため，保存期間を長期間に設定できる．

　これらの機能を満たすため，本システムでは長期保存用記憶域として，Azure Blob Storage を用いる．Blob Storage はオブジェクトストレージと称されるストレージ領域である [11]．本学の構成では，データ保存期間は無期限としている．

データ入出力関数の定義 / Function（図 4（e））

　センサデータ入力部で受け付けたデータは各記憶域にて保存を行う．この際，入力データの受信を検知し，適切な記憶域へ保存するデータフロー処理の定義を要する．データフロー処理が滞った場合，各機能群にデータが滞留しシステムの稼働が維持できなくなるため，遅延なく処理し，高信頼に稼働させる必要がある．従来のシステム開発手法では，データフロー用のプログラムをハードコーディングにより作成し，仮想サーバ等で稼働させる方法が一般的であった．しかし従来手法では，可用性の確保が困難であると考えられる．また，入力データの受信を検知する処理等が煩雑となり，作成したプログラムの保守面においても課題が生じる．

　これらの問題に対応するため，本システムではサーバレスアーキテクチャを活用する．サーバレスアーキテクチャは，仮想サーバ等を用いずに，処理関数の実行が可能な環境である．処理関数は各機能群へのデータ入力を条件としてプログラムが駆動し，データ入出力の関係性を記述するのみでデータフロー処理の定義が可能である．

　本システムでは，Microsoft Azure Function[12] を用いて，1. データ入力部と低遅延，高速記憶域間のデータフロー，2. 低遅延，高速記憶域と後段である API 連携用記憶域・長期保存用記憶域間のデータフローを定義する．前節の通り，本システムはセンサデータ入力部や各記憶域については，同時接続や並列性を考慮した構造を仕様上は有している．換言すれば，多数のセンサデータ入力を同時並列に取り扱うためには，データフロー関数の記述や記憶域の構成を機能群の特性に応じて行う必要がある．

　データフローと各記憶域への入力の流れを図 5 に示す．センサデータはデータ収集用システム（仮想マシン上に構成）から，データ入力部に送付する．データ入力部は IoT Hub 2 式（Hub1，Hub2）で構成し，記憶域は低遅延，高速記憶域（Cosmosdb1）・API 連携用（ADXcluster1，EventHub1）・長期保存用（BlobStorage1）とも 1 式で構成する．

　本論文では説明の簡略化のため，センサデータは AP 接続情報（AP_indata0，AP_indata1）の 2 式，密集度情報（Congestion_indata0，Congestion_indata1）の 2 式，二酸化炭素濃度（CO2PPM_indata0，CO2PPM_indata1）の 2 式とする．

　なお，各機能群が有するデータ入出力に関わる部分機

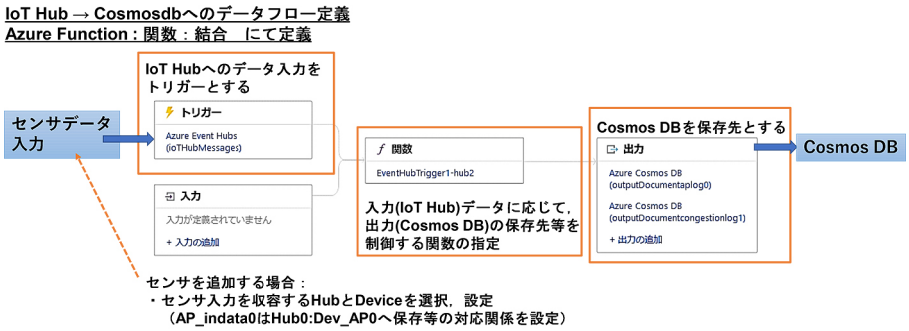

図 6　データ入力部→低遅延，高速記憶域間のデータフロー定義

能を示す際は，「:」記号で名称を接続する．例として，「Hub1 内のデータを取り扱う領域となる Device0」を示す際は，「Hub1:Device0」と記述する．また，各機能群間のデータの流れについては，「→」記号で名称を接続する．例として，「センサデータ AP_indata0 を Hub1:Device0 へ入力」を示す際は，「AP_indata0 → Hub1:Device0」と記述する．

センサデータ→データ入力部（図 5（a））

センサデータ入力部に用いる IoT Hub は，同時に 9 並列の接続に対応している．接続情報は Device と呼ばれ，登録したセンサや API のエンドポイントに対応する．本システムは 2 式の IoT Hub を用いるため，同時接続数は 18 となる．

本システムでは，Device へカテゴリごとに Dev_AP0, Dev_congestion0, Dev_CO2PPM0 等の名称を付している．センサデータをどの Device に送付すべきかの対応関係は，センサ設置時および送付先の変更を要するごとに，センサ側に設定を行う．すなわち，AP_indata0 は Hub1:Dev_AP0 へ送付といった対応関係をセンサ側プログラム等へ設定する．データ送付は Azure IoT Hub API を利用し，python 言語用の IoTHubDeviceClient パッケージ [14] を用いて入力プログラムを記述する（末尾付録内 Listing 2, python 言語）．

なお，同時接続センサ数を増やす場合は，1. IoT Hub を追加する方法，2. 入力プログラム側での繰り返し処理や分割により同時接続数を超えないよう制御する方法が考えられる．ここでは，2. の方法を採り，データ収集用システム上でセンサデータを集約し，センサデータごとに対応する IoT Hub の Device にデータを送付する．

データ入力部→低遅延，高速記憶域（図 5（b））

データ入力部である IoT Hub の Device へのデータ入力後，低遅延，高速記憶域にデータを転送し保存を行う．本システムにおいて低遅延，高速記憶域に利用する Cosmos DB では，保存領域は Container と呼ばれる．本システムではデータ入力部である IoT Hub の Device に一対一で対応する Container を作成する（すなわち，全数 18 の Container を有する）．図 5 では，Dev_AP0 へ入力されたデータ用の領域 AP_LOG0，Dev_congestion0 へ入力されたデータ用の領域 CONGESTION_LOG0，Dev_CO2PPM0 へ入力されたデータ用の領域 CO2PPM_LOG0 等を作成していることを示している．

Device へ入力されたデータに対する Container への保存に関する定義は，Azure Funciton を活用する．Azure Function では，関数をトリガーする入力，実行する関数，出力先を定義することが可能である．本システムでは，トリガーとして Hub1, Hub2 の各 Device へのデータ入力，出力先として Cosmosdb1 を指定する（図 6）．

ならびに，実行する関数に IoT Hub → Cosmos DB へのデータ転送の定義として，Device への入力データと保存する Container の対応関係を記述する（末尾付録内 Listing 3, javascript）[13]．上述の通り，Device への入力データと保存する Container の対応関係の定義となるため，新たなセンサ設置時に定義を追加する必要はない．

複数の Device にデータが並列に入力された場合においても，入力トリガーの発生と連動する Azure Function の駆動は並列に実行される．このことから，データ入力部から低遅延，高速記憶域へのデータ転送と保存も並列性を有する．

低遅延，高速記憶域→他の記憶域（図 5（c，d））

低遅延，高速記憶域への保存後，API 連携用および長期保存用にデータを転送する．

本システムに置いて API 連携用記憶域に用いる Azure Data Explorer では，保存領域は Cluster-Table と呼ばれる．本システムでは，低遅延，高速記憶域である Container（およびデータ入力元となる IoT Hub の Device）に一対一で対応する Cluster-Table を作成する．

図 5 では，対応する Container と Cluster-Table が，Event Hubs によって中継される関係を示している（低遅延，高速記憶域 AP_LOG0 に対応する API 連携用記憶域 table_AP0 等を作成し，Event Hubs によって中継する）．長期

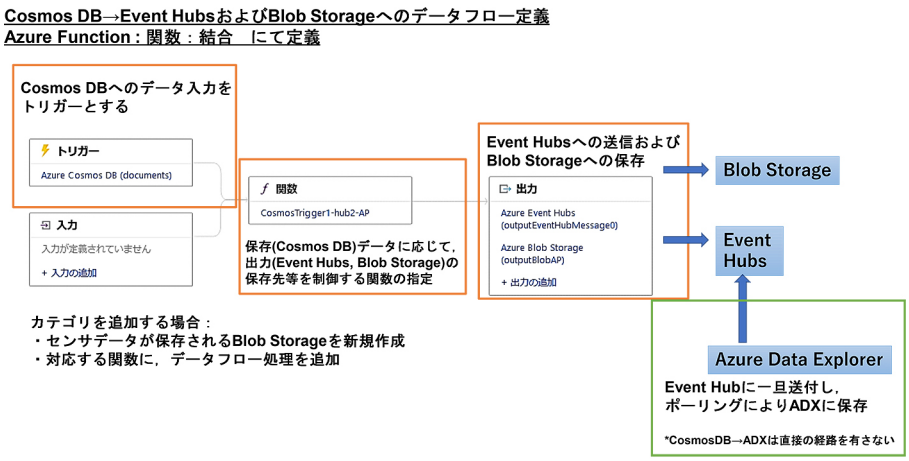

図 7　低遅延，高速記憶域→他の記憶域間のデータフロー定義

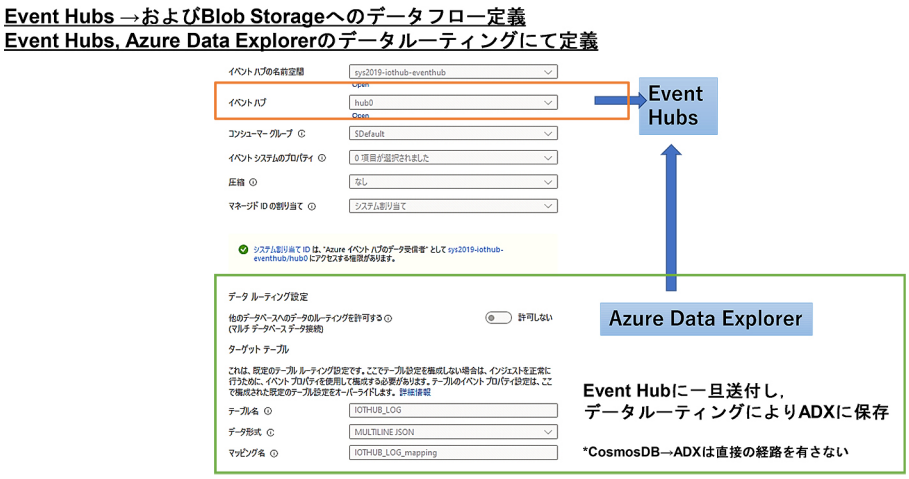

図 8　Azure Data Explorer へのデータルーティング定義

保存用記憶域である Blob Storage には，センサのカテゴリの追加に応じて，保存領域である Blob を作成する（図 5 では，接続情報用の領域 log-ap，密集度用の領域 log-congestion，二酸化炭素濃度用の領域 log-co2ppm を作成）．Blob には，システムに入力されるセンサデータ単位として JSON 形式によりデータを記憶する．

データの入出力および保存に関する定義は，データ入力部→低遅延，高速記憶域と同様に Azure Function を活用する．本システムでは，トリガーとして低遅延，高速記憶域へのデータ保存，出力先として BlobStorage1，EventHub1 を指定する（図 7）．ならびに，Event Hub に送付されたデータに対するデータルーティングを定義することにより，Azure Data Explorer へデータを中継する（図 8）．

Cosmos DB → EventHub および Blob Storage へのデータ転送の定義については，トリガーにより実行される関数に記述する（末尾付録内 Listing 4，javascript）．

ここでも，Device への入力データが源泉となる Container と，Cluster-Table と Blob の対応関係の定義となるため，新たなセンサ設置時に定義を追加する必要はない．

4　構築システムの評価，考察

本節では，構築した大規模センサデータ集約システムについて，3 節にて定めたシステムの性能の達成度を通して評価，考察する．

4.1　多数の機器・センサからの入力への対応

構築システムへのデータ入力に要する時間を測定し，評価基準とする．目標数である 3000 個のセンサデータ発生源を有していないため，AP への接続数・密集度・CO2 濃度に相当するデータを生成し，入力データ数を変化させながら収集に要した時間を計測する．ならびに，後述の高可用性を有する基盤での稼働の評価のため，収集に失敗したデータ数もあわせて計測する．

入力データ数は，100 から 3000 までとし，1000 まで

データ数	データ入力所要時間（秒）
100	6.5
200	12.9
300	19.1
400	25.2
500	31.3
600	37.6
700	44
800	50.3
900	57
1000	62.7
2000	125.9
3000	189.2

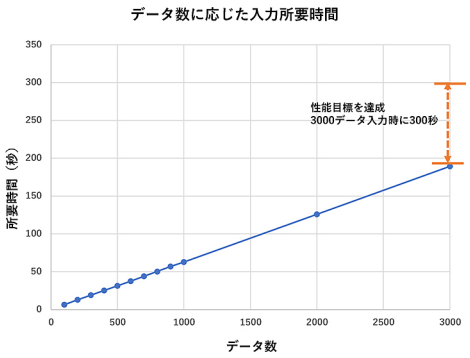

図 9　入力データ数に対する所要時間

は 100 ずつ増加，1000 以降は 1000 ずつ増加とする．試行は 4 回実施し，要した最大時間数を計測値とする．入力データ数と要した時間（秒）の推移について図 9 に示す．目標と定めた 3000 データ入力時の所要時間は 189 秒であり，目標であった 300 秒に対して余力のある性能を有することが確認された．また，入力データ数に対して線形に所要時間は推移していること，1000 データ入力に対する所要時間は 60 秒程度であることから，300 秒で 5000 データ程度の入力に対応可能であると推定される．

4.2　用途に適したデータ記憶への対応

低遅延，高速記憶域

低遅延，高速記憶域については，データ入力から記憶域が記憶を開始するまでの遅延時間を評価基準とする．記憶域である Cosmos DB が記録を開始するまでの遅延時間については，Server Side Latency と呼ばれるメトリックに記録される．記録された Server Side Latency は，4.1 節の検証における入力データ数によらず，3（ms）以下であった．データ入力部での処理後，直ちにデータ蓄積が開始される能力を有していることが確認された．

API 連携用記憶域

API 連携用記憶域については，目標と定めた可視化用システムである Grafana との API 連携を実施し，その動作を確認した．可視化用システムである Grafana と，API 連携用記憶域（Azure Data Explorer）の出力 API との連携後，取得したセンサデータの時系列情報の検索，取得地点ごとのセンサデータを表形式やグラフ形式で提示することが可能であった（図 10）．

長期保存用記憶域

長期保存用記憶域については，記憶されたセンサデータが出力（ダウンロード）可能であることを確認した．取得したセンサデータの属性名と属性値がアーカイブ保存されており，JSON 形式でのダウンロードを可能としている（Listing 1）．

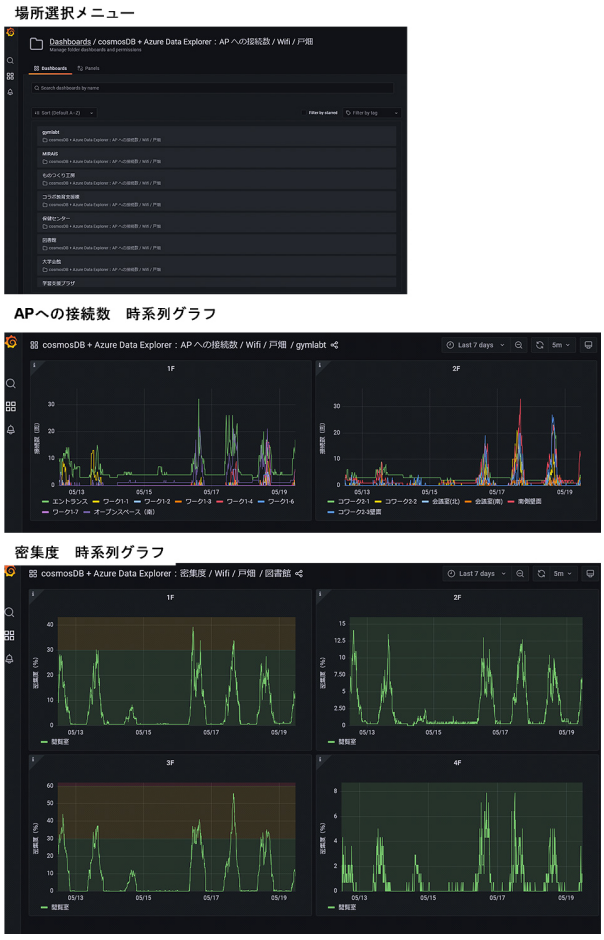

図 10　可視化システム（Grafana）との連携

Listing 1　長期保存用記憶域から取得されるセンサデータの例（JSON 形式）

```
# 接続数
  {
    "id": "49c9d522-fba9-455c-9e85-3f58d13c62ef",
    "category": "ap_count", # カテゴリ：接続数，密集度，二酸化炭素濃度等
    "campus": "iizuka", # キャンパス情報
    "floor": "3", # 階数
    "place": 課外活動施設"", # 建屋情報
    "name": "kagaii-3f-outdoor", # 固有名
    "value": "1", # 接続数
    "TimeStamp": "2022-05-15T22:05:00Z" # センサ側が記録したタイムスタンプ
  }
```

また，本システムでは 5 分毎に 3000 データを入力した際のデータ記憶容量は年間 86（GB）程度となるが，その際の記憶域である Azure Blob Storage のアーカイブ層の維持費は年間 500 円程度である．コスト面において，データ保存期間を無制限とすることが可能と判断できる．

4.3　高可用性を有する基盤での稼働

目標と定めた，入力データの受信に失敗した場合に対象のデータが再送可能であることを確認する．ここでは，

データ数	最大エラー率（%）	最大エラー数
100	0	0
200	0	0
300	0	0
400	0	0
500	0	0
600	0	0
700	0	0
800	0	0
900	0.2	2
1000	0	0
2000	0	0
3000	0	0

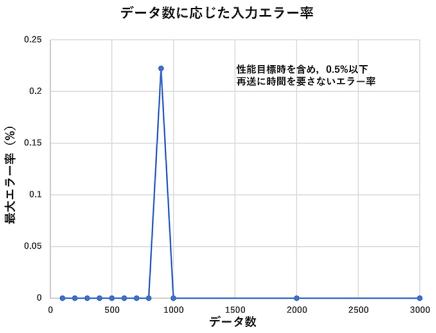

図 11　入力データ数に対するエラー率

4.1 節にて測定したデータ入力に要した時間と，入力に失敗したデータ数から，データ入力処理時の空き時間に再送が可能であるかを判断する．試行は 4 回実施し，要した最大エラー率（入力データ数に対するエラー数の割合）を計測値とする．図 11 に入力データ数と最大エラー率の推移を示す．

データ数 900 を試行した際に，0.2% のエラー率を示したが，それ以外ではエラーは生じない結果となった．目標と定めた入力データ数 3000 以内であれば，再送を要するデータ数は数個であることから，先行研究と同様の 5 分間隔でセンサデータ収集を行う運用において，再送可能な時間的余裕を持つと判断できる．

5　むすび

本稿では，パブリッククラウドを活用した大規模センサデータ集約システムを構築し，本学に設置された無線 AP 情報及び設置個所の密集度の収集での活用状況について言及した．

先行研究である密集度表示システムは，接続端末数を含む AP の動作状況と算出した密集度が蓄積され，利用者に提示する機能を有していたが，多様なセンサへの対応や，外部システムへの連携，可用性への考慮が不十分であった．これらの要件を満たすため，パブリッククラウドが有する大規模なデータ入出力が可能な機能群を組み合わせ，新たに大規模センサデータ集約システムを構築した．

大規模センサデータ集約システムは，多様なプロトコルによるデータ入力，多数のセンサの同時接続への対応，アクセス速度や長期保存等の要求毎に対応した記憶域でのデータ管理，多様な外部システムへの連携を実現した．また，機能間のデータフローの制御については，サーバレスアーキテクチャを用いることにより，ハードコーディングの排除による可用性，メンテナンス性の向上も実現している．

学内設置の全ての AP や CO2 センサの収集は未実施であることから，評価においてはセンサデータを仮想的に生成したものであった．今後は，学内の全ての AP や CO2 センサから実データを収集し，センサデータの一元管理を推進する．

参考文献

[1] 九州工業大学 IoT システム基盤研究センター，“『Project “PLATEAU”』における空間の滞留人数可視化技術の実証”，https://www.kyutech.ac.jp/whats-new/topics/entry-7968.html，（2022 年 6 月 30 日参照）．

[2] 冨重 秀樹，井上 純一，畑瀬 卓司，和田 数字郎，林 豊洋，福田 豊，“無線 LAN 接続情報を利用した密集度表示システムとその改良”，国立大学法人情報系センター協議会学術情報処理研究，Vol. 25, No. 1, pp. 1–8, 2021.

[3] Azure Documentation, “AWS to Azure services comparison”, https://docs.microsoft.com/en-us/azure/architecture/aws-professional/services,（2022 年 9 月 20 日参照）．

[4] 林 豊洋，福田 豊，佐藤 彰洋，中村 豊，“SINET クラウド接続サービスを用いた学内サーバ群のパブリッククラウドへの展開”，大学 ICT 推進協議会 2021 年度年次大会，pp. 250–256, 2021.

[5] Grafana Labs, “Grafana: The open observability platform”, https://grafana.com/,（2022 年 6 月 30 日参照）．

[6] Azure Documentation, “IoT concepts and Azure IoT Hub”, https://docs.microsoft.com/en-us/azure/iot-hub/iot-concepts-and-iot-hub,（2022 年 6 月 30 日参照）．

[7] Azure Architecture, “Azure Cosmos DB in IoT workloads”, https://docs.microsoft.com/en-us/azure/architecture/solution-ideas/articles/iot-using-cosmos-db,（2022 年 6 月 30 日参照）．

[8] Azure Documentation, “Understanding the differences between NoSQL and relational databases”, https://docs.microsoft.com/en-us/azure/cosmos-db/relational-nosql,（2022 年 6 月 30 日参照）．

[9] Microsoft Tech Community, “Integration of Azure Data Explorer with Cosmos DB for near real-time analytics”, https://techcommunity.microsoft.com/t5/azure-data-explorer-blog/integration-of-azure-data-explorer-with-cosmos-db-for-near-real/ba-p/1485099,（2022 年 6 月 30 日参照）．

[10] Azure Documentation, “Ingest data from event hub into Azure Data Explorer”, https://docs.microsoft.com/en-us/azure/data-explorer/ingest-data-event-hub,（2022 年 6 月 30 日参照）．

[11] Azure Documentation, “Introduction to Azure Blob storage”, https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction,（2022 年 6 月 30 日参照）．

[12] Azure Documentation, “Azure Functions triggers and bindings concepts”, https://docs.microsoft.com/en-us/azure/azure-functions/functions-triggers-bindings,（2022 年 6 月 30 日参照）．

[13] Azure Documentation, “Azure IoT Hub trigger for Azure Functions”, https://docs.microsoft.com/en-us/azure/azure-functions/functions-bindings-event-iot-trigger,（2022 年 6 月 30 日参照）．

[14] Microsoft Docs - Reference, “azure-iot-hub Package”, https://docs.microsoft.com/en-us/python/api/azure-iot-hub/,（2022 年 6 月 30 日参照）．

付　録

実装時のプログラムコード，蓄積データ

本節にて付録として，大規模センサデータ集約システムを実

装する際の，データ入力 API 利用プログラム，サーバレスアーキテクチャ上の関数プログラムを示す．

Listing 2　センサデータ入力プログラム（データ収集用システム上で稼働）

```
# Copy AP log (VM) to Cloud using IotHub

import asyncio
from azure.iot.device import Message
from azure.iot.device.aio import
 IoTHubDeviceClient
from datetime import datetime, timedelta
import datetime


CONNECTION_STRING = "HostName=***HOST***.azure-
 devices.net;DeviceId=device0;SharedAccessKey=***
 KEY***"

MSG_TXT = '{{"log": "no2","category": "ap_count
 ","campus": "iizuka","floor": "1","place": "1F
 ","name": "k-1f-place0","value": {value},"
 TimeStamp": "{datetime}"}}'


async def main():
  try:
    date = '***DATE***'
    day = datetime.datetime.strptime(date,'%Y/%m
     /%d %H:%M:%S')
    result = day + timedelta(hours=-9)
    a = result.isoformat(timespec='milliseconds')
    dateTime = a + 'Z'

    conn_str = CONNECTION_STRING

    device_client = IoTHubDeviceClient.
     create_from_connection_string(conn_str)

    await device_client.connect()

    with open('*** log_k-1f-place0 ***') as f:
      value = f.read()

    msg_txt_formatted = MSG_TXT.format(value=
     value,datetime=dateTime)
    message = Message(msg_txt_formatted)

    await device_client.send_message(message)

  finally:
    await device_client.disconnect()

if __name__ == '__main__':
  loop = asyncio.get_event_loop()
  loop.run_until_complete(main())
  loop.close()
```

Listing 3　データ入力部→低遅延，高速記憶域間のデータフロー関数

```
//Data in : IoTHub -> Copy : Cosmos DB

module.exports = async function (context,
 IoTHubMessages) {

  IoTHubMessages.forEach(message => {

    //no1, no2 : AP log
    if (message.log == 'no1' || message.log == '
     no2') {
      var str1 = "{" + "category: " + "'" +
       message.category + "'" + ",campus: " + "'"
        + message.campus + "'" + ",floor: " +  "'
       " + message.floor + "'" + ",place: " + "'"
        + message.place + "'" + ",name: " + "'" +
        message.name + "'" + ",value: " + "'" +
       message.value + "'" + ",TimeStamp: " + "'"
        + message.TimeStamp + "'" + "}";

      if (message.log == 'no1')
        context.bindings.outputDocumentaplog1 =
         str1;

      if (message.log == 'no2')
        context.bindings.outputDocumentaplog2 =
         str1;
    }

    //no5, no6 : congestion log
    if (message.log == 'no5' || message.log == '
     no6') {
      var str2 = "{" + "category: " + "'" +
       message.category + "'" + ",campus: " + "'"
        + message.campus + "'" + ",floor: " + "'"
        + message.floor + "'"  + ",place: " + "'"
        + message.place + "'"  + ",construction:
       " + "'" + message.construction + "'" + ",
       value: " + "'" + message.value  + "'" + ",
       TimeStamp: " + "'" + message.TimeStamp + "
       '" + ",area: " + "'" + message.area + "'"
       + ",rate: " + "'" + message.rate + "'" + "
       ,device: " + "'" + message.device + "'" +
       ",meter: " + "'" + message.meter + "'" + "
       ,ap: " + "'" + message.ap + "'" + "}";

      if (message.log == 'no5')
        context.bindings.
         outputDocumentcongestionlog1 = str2;

      if (message.log == 'no6')
        context.bindings.
         outputDocumentcongestionlog2 = str2;
    }

    });

};
```

Listing 4　低遅延，高速記憶域→他の記憶域間のデータフロー関数

```
//Data in : Cosmos DB (document for AP log1) ->
 Transfer : Event Hub, Copy : Blob Storage for AP
  log
module.exports = async function (context,
 documents) {
    if (!!documents && documents.length > 0) {

        context.bindings.outputEventHubMessage0 =
          documents;
        if (documents[0].category == "ap_count")
         {
            context.bindings.outputBlobAP =
             documents;
        }
    }
}


//Data in : Cosmos DB (document for Congestion
 log1) -> Transfer : Event Hub, Copy : Blob
 Storage for Congestion log
module.exports = async function (context,
 documents) {
    if (!!documents && documents.length > 0) {

        context.bindings.outputEventHubMessage5 =
          documents;
        if (documents[0].category == "
         ap_congestion") {
            context.bindings.outputBlobCONGESTION
              = documents;
        }
    }
}
```