# 動的データフロー処理のための分散ルーティング機構

寺西 裕一[1,a)]　木全 崇[1]　山中 広明[1]　河合 栄治[1]　原井 洋明[1]

**概要**：Topic-Based Pub/Sub (TBPS) によって定義されたクラウドやエッジコンピューティング環境上のデータフロー処理を，ネットワークや計算機の過負荷状態等に応じて動的にスケールアウト／オフローディングさせることを可能とする分散ルーティング機構 Locality-Aware Stream Routing(LASR) を提案する．LASR は，TBPS のトピックとは独立にルーティングの際に参照する「index」をデータに付与することで，データフローの定義を更新することなくスケールアウト/オフローディングを実行可能とする．また，LASR は，index と物理ネットワークトポロジに基づく構造化オーバレイの構成により，既存のマスタスレーブ型のルーティング方法が持つ配信遅延やスケーラビリティの課題を解決する．本稿では，構造化オーバレイ Suzaku を用いた LASR のプロトタイプ実装，および，分散クラウドテストベッド JOSE 上でのプロトタイプ実装の動作検証・評価の結果を示す．

## 1. はじめに

ネットワーク接続可能な小型デバイス，センサー，スマートフォン，ウェアラブルデバイス等の台頭にともない，Internet of Things(IoT) が注目を集めている．IoT のアプリケーションには，センサー等から連続的に生成されるデータを逐次処理した上で，その結果をスマートフォンやウェアラブルデバイス等へ通知するといった一連のデータ処理の流れがある．IoT デバイスは十分な処理能力を持つとは限らないため，生成されたデータをデータセンタ等へ送信し，クラウドコンピューティングによって処理する形態が典型的である．また，クラウドコンピューティングの発展形として，クラウド上の処理の一部をクラウド以外のネットワークエッジ等の計算機上で実行することで，応答時間の短縮や無駄なトラヒックを減少させる，いわゆるエッジコンピューティング [1], [2], [3] と呼ばれる形態も注目されている．こうした環境において IoT のアプリケーションを簡単に作成することを目指し，実行するデータ処理の流れを，データフローのパラダイムによって定義するフレームワークがいくつか提案されている [4], [5], [6], [7]．これらのフレームワークには，データフローを構成する処理のコンポーネントの詳細を知らずとも，一連の処理をフローとして定義できる利点がある．

一方，IoT では，データの生成元が自動車やスマートフォンである場合も考えられ，生成されるデータソースの数やデータ量はデータ生成元の移動や電源のオン／オフ等にともない変化する．また，例えば，映像に映った人を分析するアプリケーションでは，カメラに映る人の増減に応じて，データ量や処理量が大幅に変化する．このため，処理に必要なネットワークや計算機の資源の数・量も変動する．

しかし，既存のデータフローフレームワークは，一度データフローを定義すると，基本的に次にデータフロー定義者が更新するまで，フローの構造は変化しない．したがって，上記変動に対応するには，データフロー定義者がデータフローの定義を更新するか，ピーク時に必要なネットワークや計算機の資源を常に確保し，それらを用いるよう，あらかじめデータフローを定義しなければならない．

関連技術として，クラウド上で連続的に生成されるストリームデータを分散処理するフレームワークもいくつか提案されている [10], [11], [12], [13]．しかし，これらは，専用プログラムの記述が必要であり，データフローフレームワークのように個々の処理の詳細に立ち入らずに全体の処理フローを定義できない．また，基本的にマスタスレーブ型でシステムが構成されるため，データ送信時にクラウド上のサーバへの問い合わせが必要となり，伝送遅延によって配信遅延が大きくなる問題や，問い合わせの集中にともなうスケーラビリティの問題がある．

上記課題をふまえた本稿の貢献は次のとおりである．データの処理を行なうコンポーネント間の接続を Topic-Based Pub/Sub (TBPS) によって定義する前提のもと，データフローの定義を変更することなく，ネットワークや計算機の過負荷状態に応じてデータフローの構造を動的変化させることを可能とする分散ルーティング機構 LASR(Locality-Aware Stream Routing) を提案した．また，双方向キー順序保存型構造化オーバレイ Suzaku [16] 上に実装した LASR のプロトタイプを用いて，分散クラウドテストベッド JOSE[17] 上で実機評価を行ない，既存のマスタスレーブ型のルーティング方式よりも配信時間が短かく，配信エラー数を低減できることを示した．

[1] 国立研究開発法人 情報通信研究機構
4-2-1 Nukui-Kitamachi, Koganei, Tokyo 184–8795, Japan
[a)] teranisi@nict.go.jp

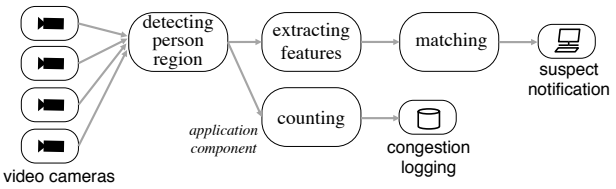


図 1　データフローのグラフ例

## 2. 想定環境

まず，本研究において想定するデータフロー定義，動的な構成変更要求，ならびに，データ配信環境を示す．

### 2.1 データフロー定義

最初に，本研究で扱うデータフローを定義する．データフローは，有向グラフ $G = (V, E)$ によって表現される．グラフノード $c \in V$ は，処理を行なうプロセスやデータ入力元，出力先に対応する (これらをコンポーネントと呼ぶ)．グラフエッジ $l \in E$ は，コンポーネント間でデータストリームを配信するリンクである．$l = (c_1, c_2)$ は $l$ がコンポーネント $c_1$ と $c_2$ を接続していることを表す．ひとつのグラフは，複数のデータフローアプリケーションを含むことがある．図 1 は，ふたつのアプリケーションを含むグラフの例である．このグラフは，映像から人の領域を切り出す「detecting person region」のコンポーネントがビデオカメラからの入力を受信し，「extracting features」のコンポーネントで特徴を取り出し，「matching」のコンポーネントがマッチングを行ない，特徴が合致した場合に通知を行なうというアプリケーション (該当者通知アプリケーション) と，「detecting person region」の出力数を「counting」のコンポーネントで数えて，時間ごとに記録するアプリケーション (混雑度記録アプリケーション) を含んでいる．

### 2.2 グラフの動的な変更

IoT アプリケーションにおける動的な状況変動に対応するためには，データフローのグラフにおいて，コンポーネントの追加および削除を行なう必要が生じる．本稿で対象とするグラフの変化はスケールアウト／スケールインおよびオフローディングとする．これらの動作を図 2 に示す．

スケールアウト／スケールインは，あるノード (計算機) で実行している処理を分割し，複数のノードに振り分ける (スケールアウト)，または，複数のノードで実行されている処理を統合し，ひとつのノードで実行するようにする (スケールイン) 動作を指す．

また，オフローディングは，エッジコンピューティング環境において，クラウド上で実行されている処理の一部または全てをエッジノード (例えば，計算能力を持つゲートウェイ) へ移動させることを指す．オフローディングでは実行中の処理を全て別のノードへ振り分けることがあるが，スケールアウトではそのようなことは無い．

スケールアウトおよびオフローディングは，$G = (V, E)$ において，$(c_s, c_d) \in E$ のとき，$V$ に $c_d'$ を追加し，$E$ に $(c_s, c_d')$ を追加することに対応する．$x'$ は $x$ の複製であることを表す (以下，複製コンポーネントと呼ぶ)．$(c_s, c_d')$ が $E$ に追加されると，$c_s$ の出力先は 2 つとなる．$c_s$ は，$c_d$ への出力の一部または全部を $c_d'$ へ振り分ける．スケールインは，上記の逆の動作であり，$V$ から $c_d'$ を，$E$ から $(c_s, c_d')$ を削除することに対応する．

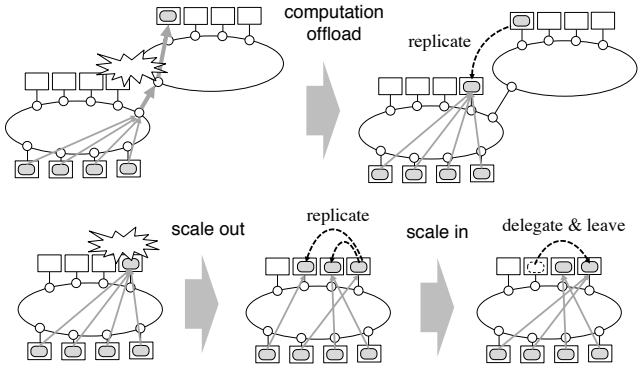


図 2　データフローのグラフの動的な変更: 上は，ゲートウェイ上の過負荷を回避するオフローディングの動作例．下はスケールアウト，スケールインの動作例．

### 2.3 エッジブローカーモデル

本研究では，グラフエッジに相当するコンポーネント間リンクは，Topic-Based Pub/Sub (TBPS) を用いる前提とする．また，TBPS はエッジブローカーモデルに基づくものとする．エッジブローカーは，TBPS において publisher と subscriber を管理するブローカーを，クラウド上ではなくエッジ上に分散して配備することで，同一エッジブローカー配下の publisher と subscriber 間のデータ配信遅延を小さくするとともに，コアネットワークに流通するデータ量を削減する．エッジブローカーモデルによる TBPS 方式としては，文献 [18], [19] 等がある．publisher や subscriber に相当するデバイスは，基本的に物理的に最寄りのエッジブローカーに接続する．たとえば，無線ネットワークにおける基地局等が，エッジブローカーに対応する．上記文献の方式では，エッジ間のデータ配信に，後述の双方向キー順序保存型オーバレイネットワークの一つである Skip Graph [14] を用いている．

## 3. LASR

本章では，提案方式である LASR の動作を示す．LASR は，既存のネットワークに手を加えずに動作するアプリケーションレベルのオーバレイネットワークに基づくアルゴリズムである．

### 3.1 双方向キー順序保存型オーバレイネットワーク

LASR は，双方向キー順序保存型オーバレイネットワークを用いる．双方向キー順序保存型オーバレイネットワークは，キーが値の順に並ぶオーバレイ構造を持つ．キーを持つ計算機 (ノード) の検索は，オーバレイ構造に基づいて行なわれるが，双方向キー順序保存型オーバレイネットワークでは，キーの値が連続するため，キーの値の範囲を

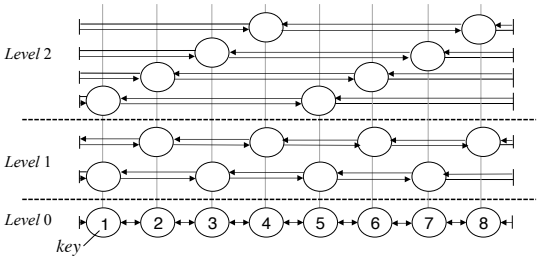


**図 3** 双方向キー順序保存型オーバレイネットワークの例

指定した範囲検索が可能である．図 3 は，双方向キー順序保存型オーバレイネットワークの例である．この例は後述の Suzaku の構造の例を示している．キーの検索は，対象キーの値を越えず最も値が近いキーへのリンクを再帰的にたどる greedy ルーティングによって行なわれる．双方向キー順序保存型オーバレイネットワークにおける greedy ルーティングでは，任意の連続した近傍ノード集合 $S$ に属するノード間のルーティングが $S$ に属さないノードを経由することは無い．

TBPS のメッセージ送受信を行なう publisher および subscriber は，それぞれトピック等に応じたキーを持つ (後述)．オーバレイへは，エッジブローカーと処理コンポーネントが参加する．エッジブローカー配下には，複数の publisher や subscriber を収容することが考えられるが，同一のエッジブローカーが仮想的に複数のキーを持つことで対応する．

### 3.2 TBPS の拡張：index の導入

LASR では，グラフエッジ $l \in E$ は，TBPS のトピックに対応する．これは既存のデータフローフレームワークと同様である．

定義されたデータフローに基づくオフローディング，スケールアウト／スケールインに対応するため，LASR は TBPS の送受信データに *index* と呼ぶ属性を追加する．index は一次元のスカラー値である．publisher は，トピックに加えて index を付与してデータを publish する．subscriber は，topic と index の範囲を指定して subscribe する．subscriber $s$ が subscribe する index の範囲は $I_s$ と表記する．$I_s = \big[min(I_s), max(I_s)\big)$ を範囲として subscribe している subscriber は，index $i$ を持つデータを，$min(I_s) \leq i < max(I_s)$ が満たされる場合のみ受信する．

LASR における基本的な負荷分散方法は，*shuffle mapping* である．この方法は，index $i$ として乱数を指定する．振り分先の subscriber 群は，index 範囲の大きさに応じた数のデータを受信する．均等の大きさの範囲を割り当てると，確率的に同数のデータを受信することになる．もう一つの負荷分散方法は，*content mapping* である．この方法は，データの内容に応じて $i$ を割り当てる．content mapping の例としては，例えば，データがユーザ ID フィールドを持つとして，そのフィールド値のハッシュ値を $i$ に割り当てる場合が相当する．この割り当てによって，同じユーザ ID を持つデータは同じノードに送信される．アプリケーションが状態を持つ場合，content mapping を適用

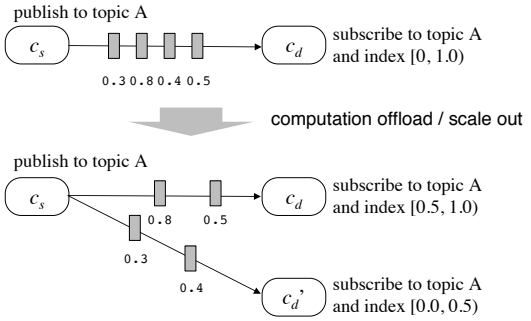


**図 4** index の割り当て変更の例

する必要がある．オフローディング，および，スケールアウトを実行する際，まず，$I_s$ を次の 2 つに分割する．

$$I_{s1} = \big[min(I_s), sp\big), I_{s2} = \big[sp, max(I_s)\big),$$

$sp$ は，index の分割点 (splitting-point) である．$sp$ の値はアプリケーションにおいて決定する．$I_{s1}$ あるいは $I_{s2}$ を，subscriber として動作する複製コンポーネントに割り当てることで，処理の一部を分担させることができる．オフロードやスケールアウトにおいて複数のノードへ処理を分担させるには，index の分割を再帰的に行なう．

図 4 は，データフローのグラフの更新動作を示している．この例では，トピック A が $(c_s, c_d)$ に対応している．$(c_s, c_d)$ 間で送受信されるストリームデータが，0.3, 0.8, 0.4, 0.5 を index として持っている．図の上の例では，$c_d$ は $[0, 1.0)$ に subscribe しているため，全てのデータは $c_d$ に到達する．図の下の例は，オフローディングやスケールアウトによって，$(c_s, c_d')$ が追加された様子を示している．この例では，$sp = 0.5$ である．$c_d$ はその担当範囲を $[0.5, 1.0)$ に変化させ，複製コンポーネント $c_d'$ は $[0.0, 0.5)$ を新たに担当している．したがって，0.3 と 0.4 は $c_d'$ へ送信されるようになる．

エッジブローカーが生成するデータに対する index は，オーバレイに参加するエッジブローカーや処理コンポーネントにおける LASR のオーバレイ転送処理モジュールにおいて割りあてる動作を基本とする．すなわち，エッジブローカーに接続するデバイスやデータフローの処理コンポーネント自体は index による構造変化を意識する必要がないようにする．content mapping では，アプリケーションに応じた index を定義するプログラムが LASR の一部となるようインストールする必要がなる．

### 3.3 LASR の動作

LASR では，双方向キー順序保存型オーバレイネットワークにおけるキーを次の通り定義する．

$$\{topic, net\_id, flag, index, unique\_id\},$$

*topic* は，トピックの文字列である．*net_id* は，物理ネットワークの識別子である．*flag* は，subscriber:1 か publisher:0 のいずれかを示すフラグである．*index* が前節で示した index に対応する．*unique_id* は subscriber, publisher が持つ一意の識別子である．

各要素は，比較可能な順序を持つ値を持つ．例えば，文字列は辞書順，識別子は数値の値の順序を持つ．$net\_id$ は文献 [19] の定義に従った順序を持つ値とし，ネットワークインフラから取得できる前提とする．双方向キー順序保存型オーバレイネットワークでは，各キーは $net\_id, flag$ の順序にしたがってソートされるため，同一のエッジネットワーク (例えば同じ LAN) に属する複数エッジブローカー，publisher と subscriber がオーバレイ上で隣接する構造となる．キー自体の値の比較は，全要素の辞書順によって決まる．

上記キーを用いて，エッジブローカーは次のとおり動作する．

(1) エッジブローカー $c_b$ は，publisher に対応するデバイスからトピック $t$ に対応するデータを最初に受信すると，$c_b$ は，キー $\{t, net\_id(c_b), 0, rand\}$ によりオーバレイに参加する．$net\_id(c)$ はコンポーネント $c$ の $net\_id$，$rand$ は一意の識別子となる乱数である．一定時間受信しなければ，オーバレイから離脱する．

(2) (1) ののち，エッジブローカー $c_b$ は，データに応じた index $i$ を割り当て，オーバレイ上でキー $k = \{t, 1, net\_id(c_b), i, MAX\}$ を越えない最大のキーに対応する subscriber へ転送する．$MAX$ は識別子に対応する数値の最大値である．$k$ を越えない最大のキーを持つノードの検索は，双方向キー順序保存型オーバレイネットワークにキーを追加する際に必要となる探索機能であり，LASR ではこの探索機能をデータ配信に用いる．

一方，処理コンポーネントは次のとおり動作する．

(1) 全ての処理コンポーネントは初期状態では，クラウド上のノード (初期ノード) で動作を開始する．処理コンポーネント $c_p$ がトピック $t$ で subcribe したとき，キー $\{t, MIN, 1, i, rand\}$ によりオーバレイに参加する．$MIN$ は識別子の最小値である．$i$ は，初期値として index の値域の最小値を持つ．すなわち，初期ノードは，動作開始時，全てのデータを受信する．

(2) 処理コンポーネント $c_p$ は，キー $k$ のメッセージを $k.net\_id$ から最初に受信したとき，キー $\{k.topic, k.net\_id, 1, i, rand\}$ をオーバレイに追加する．これにより，異なる物理ネットワークからのデータを他の物理ネットワークを経由することなく受信できるようにする．

(3) 複製コンポーネント $c'_p$ が生成されると，$c'_p$ は，$\{t, net\_id(c'_p), 1, i, rand\}$ によりオーバレイに参加する．また，複製元のコンポーネント $c_p$ は，新たに $\{t, net\_id(c_p), 1, sp, rand\}$ をオーバレイに追加し，同時に，追加済みの $\{t, net\_id(c'_p), 1, i, ID\}$ ($ID$ は追加時の識別子) をオーバレイから削除する．

(4) 複製コンポーネント $c'_p$ が動作を終了する (スケールイン) とき，$c'_p$ は，追加済みのキー $\{t, net\_id(c'_p), 1, i, ID\}$ をオーバレイから削除する．また，複製元のコンポーネント $c_p$ は，追加済みのキー $\{t, net\_id(c_p), 1, sp, ID\}$ をオーバレイから削除し，同時に，$\{t, net\_id(c'_p), 1, i, rand\}$ をオーバレイに追加する．

上記は動作の概要であり，実際にはネットワークエラーやノード障害に対応するタイムアウト処理等も必要である．なお，LASR の詳細なアルゴリズムについては既発表の文献 [15] を参照されたい．

表 1　仮想マシンのスペック

| 項目 | 値 |
|---|---|
| Virtual CPU | 2 core / 2.1 GHz |
| Memory | 8 GByte |
| Network | 1000 BASE-T |
| OS | Ubuntu 14.04 |

## 4. LASR の実装と評価

### 4.1 プロトタイプ実装

本研究では，実装に筆者らが提案している双方向キー順序保存型オーバレイネットワーク Suzaku [16] を用いて LASR のプロトタイプを実装した．Suzaku は，多数の連続したキーが挿入・削除された場合も最大検索ホップ数が $log_2\ n$ 程度に抑えられ (n はノード数)，スケーラビリティ，Churn 耐性に優れている．Suzaku はオーバレイフレームワーク PIAX [20] 上に実装中であり，LASR のプロトタイプも PIAX 上に実装した．プロトタイプは，基本機能のみ実装を完了している．

動的な負荷分散アルゴリズムは本稿の範疇外であるが，動作確認のため，次の分散アルゴリズムにより計算機の処理負荷を分散させる機能を簡易実装した．

(1) 負荷が一定以下となった物理ノードは，トピック 'underloaded' に，乱数の index により参加する (単一ノードの場合は index の最小値)．

(2) ある物理ノードが過負荷状態となった場合，それを契機にトピック 'underloaded' に，shuffule mapping の index を付与してノードを探索する．

(3) (2) の探索によって到達したノードは，元の処理コンポーネントの複製を起動する (動作確認時は，あらかじめ起動している状態とした)．複製起動時のキーの操作は LASR に従う．複製に対する index 範囲の割り当ては，元の index の範囲の 1/2 とした．

(4) (2) の探索によって到達したノードは，'underloaded' から unsubscribe する．

### 4.2 評価環境

分散クラウドテストベッド JOSE[17] 上のクラウドサーバ群を用いて，プロトタイプの動作確認，および，性能評価を行なった．評価環境における仮想マシンのスペックは表 1 に示す通りである．

評価を行なった JOSE テストベッド上の実験環境の構成は図 5 に示す通りである．神奈川 (YRP)，石川 (StarBED)，京都 (けいはんな)，それぞれのデータセンタ内に動作する仮想マシンによるサーバを合計 52 台用いた．神奈川データセンタをクラウドネットワーク，石川データセンタ，京都データセンタをそれぞれエッジネットワークにそれぞれ見立て，クラウドネットワークに 20 台の処理コンポーネ

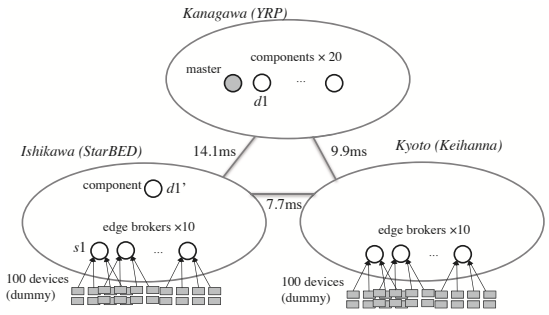


**図 5** 分散クラウドテストベッド JOSE 上の評価環境の構成

**表 2** 動作実験の設定

| パラメータ | 値 |
|---|---|
| 初期処理コンポーネント数 | 20 |
| エッジネットワーク数 | 2 |
| エッジブローカー数 | 10 / エッジネットワーク |
| (疑似) デバイス数 | 10 / エッジブローカー |
| データサイズ | 20KByte |
| デバイス上のデータ生成間隔 | 100ms |
| 実験時間 | 10s |
| 試行回数 | 10 |

ントのサーバ，北陸，京都にそれぞれ 10 台づつのエッジブローカーのサーバが動作する．データを生成するデバイスは，エッジブローカーが動作するサーバ上で擬似プロセスとした．疑似デバイスは同一サーバ上で動作するため，エッジブローカーまでの通信遅延は非常に小さい．実環境では，本実験の配信遅延に無線ネットワークの伝送遅延と転送遅延 (1ms〜) が加算される．

比較対象は，マスタスレーブ型によるルーティング方式である．スレーブノード (データの送信元) は，データを送信する際，マスターノードへ宛先を問い合わせる．マスターノードの負荷を低減させるため，マスターノードが動作するサーバを複数台用意し，ロードバランスさせる方法も考えられるが，本評価では，マスターノードの基本性能を調べるため，サーバ 1 台で動作させている．マスタスレーブ型では，神奈川データセンタ上でマスタノードが動作し，各エッジブローカーはスレーブノードとして動作する．

1 つのエッジネットワーク内に 10 個のエッジブローカーが動作し，各エッジブローカーは 10 個の疑似デバイスを収容する．すなわち，全体で合計 200 個のデバイスが動作する環境が再現されている．

実験時間は 10 秒間であり，初期状態では，各エッジブローカーの出力は，クラウド上の処理コンポーネントに配信される状態で動作を開始する．$s1$ の出力は $d1$ へ配信されるよう index が設定される．実験時間中の 5 秒目に，疑似的に $d1$ の物理ノードが過負荷状態となったと想定し，$d1$ は，エッジネットワーク上で動作する処理コンポーネントの複製 $d1'$ へ処理を委譲する動作を行なう．LASR では，$d1$ は，トピック 'underloaded' から空きノードを探索する．'underloaded' には，石川データセンタのノードの一つが subscribe しており，探索の結果 $d1'$ に $d1$ が受け持っていた index の半分が委譲される．マスタスレーブ型では，$d1$ から神奈川データセンタで動作するマスタノードへ，処理を委譲する指示が出される．すなわち，$d1'$ へは，$s1$ の出力のうち，1/4 のデータが配信される．各データセンタ間の平均 RTT (Round Trip Time) は図 5 に記載の通りである．

### 4.3 評価結果

図 6 は，各方式における $s1$ からのデータ配信遅延の累積分布である．各仮想マシンの時刻は NTP で誤差は 1ms 以下に同期されており，データ配信遅延は，送信開始時刻をデータパケットに記録し，受信時刻からパケット上の送信開始時刻を減算することで求めている．乱数によって index が決められるため多少の誤差はあるが，図のとおり，およそ 1/4 のデータの配信遅延は他と比べ短くなっている．これは上記実験設定で想定される通りの動作である．LASR では，エッジネットワーク内でオーバレイのルーティングが行なれるが，おおむね 5ms 以内で配信ができている．一方，マスタスレーブ型では，平均で RTT に対応する問い合わせ時間が最低限かかる ($s1$ とマスタノードの間の平均 RTT は 14.1ms)．図 6 によれば，エッジネットワーク内のルーティングであっても最低でも 10ms の遅延時間となっている．$s1$ から $d1$ へのデータ配信では，LASR，および，マスタスレーブ型それぞれ最大で 80ms 程度の遅延時間となった．これは，処理系が Java であるため，JIT コンパイラの影響で，配信開始直後の処理性能が比較的遅くなることに起因していると考えられる．同一の処理は，数回目の実行以降高速化される．

図 7 は，$s1$ で生成されたデータ (フレーム) が，想定される送信先に送信されなかった場合をエラー配信とみなし，各フレームにおけるエラー配信数を計測した結果である．上記実験設定に示した通り，5 秒目にデータ配信先の変更を $d1$ 上から指示しているため，50 フレーム目以降でエラー配信が生じている．LASR，マススレーブ型ともに，構成変更を反映するためにかかる遅延ののち配信が変化するため，$d1'$ へ配信されるべきデータの一部は $d1$ に送信され，エラー配信となる．LASR は 100ms 以下で構成が変更されるが，デバイスの配信タイミングによって構成変更の最中に受信されるデータがいくつかエラー配信となった．一方，マスタスレーブ型は，全デバイスの配信のたびに問い合わせが生じるための応答負荷と，エッジ-クラウド間の伝送遅延によって配信エラーの数は LASR の 2 倍以上となった．デバイス数が増加すると応答負荷が増えて，ネットワークタイムアウト，再送等が生じる可能性が高まるため，システムに参加するデバイス数に応じてエラー配信数は多くなってしまうと考えられる．一方 LASR は，システムに参加するデバイス数が増加してもエッジネットワーク内のデバイス数に変化がなければ，ルーティング経路は変わらないため，エラー配信数が増えることは無い．

## 5. おわりに

本稿では，Topic-Based Pub/Sub (TBPS) によって定義されたクラウド上のデータフロー処理を，ネットワークや計算機の過負荷状態等に応じて動的にスケールアウト／オ

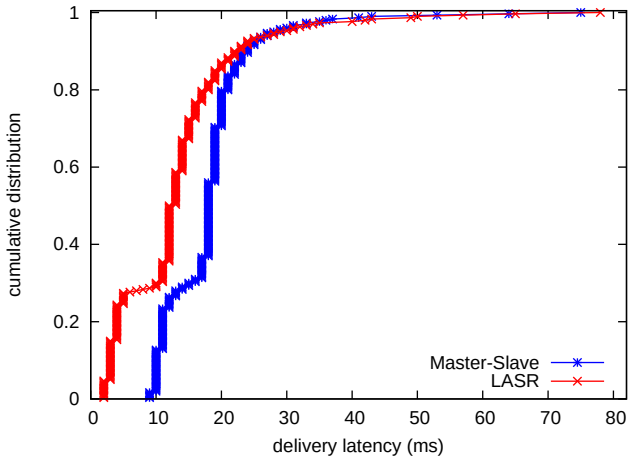


図 6　データ配信遅延の累積分布

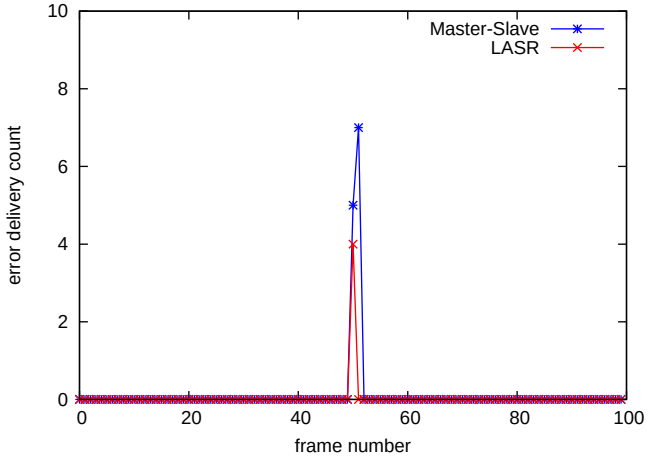


図 7　エラー配信数

フローディングさせることを可能とする分散ルーティング機構 LASR を提案した．LASR はマスターノードを持たず，ルーティングを双方向キー順序保存型構造化オーバレイによって行なうため，エッジネットワーク内のデータの送受信がクラウドや他のネットワークを経由することがなく，低遅延でのデータ配信が可能である．また，構成変更もシステム全体のデバイス数の増減に関係なく，素早く行なえる．これらの特徴を構造化オーバレイ Suzaku を用いた LASR のプロトタイプ実装，および，分散クラウドテストベッド JOSE 上での動作検証によって示すことができた．

今後，さらなる性能向上の方策，ノード故障やデータの消失があっても再送等により信頼性を向上させる方法，実際の負荷分散アルゴリズムの適用方法などの検討を進めるとともに，実アプリケーション，実デバイスへの適用等を行ない，有効性を示していく．

## 参考文献


[1] M. Patel, et al., *Mobile-Edge Computing - Introductory Technical White Paper*, ETSI MEC white paper, V1 18-09-14, 36 pages, 2014.

[2] V. Verbelen, P. Simoens, F. D. Turck, and B. Dhoedt, *Cloudlets: bringing the cloud to the mobile user*, Proc. of the third ACM workshop on Mobile cloud computing and services pp.29-36, 2012.

[3] N. Takahashi, H. Tanaka, and R. Kawamura, *Analysis of Process Assignment in Multi-tier mobile Cloud Computing and Application to Edge Accelerated Web Browsing*, Proc. of the 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud 2015), pp.233-234, 2015.

[4] Node-RED, available at: `http://nodered.org/` (2017.06.23).

[5] WoTKit, available at: `https://wotkit.readthedocs.io/` (2017.06.23).

[6] SpaceBrew, available at: `http://docs.spacebrew.cc/` (2017.06.23).

[7] A. Pintus, C. Davide, and P. Andrea, *The anatomy of a large scale social web for internet enabled objects*, Proc. of the 2nd ACM International Workshop on Web of Things, 2011.

[8] MQTT Version 3.1.1, available at: `http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.pdf` (2017.06.23).

[9] Advanced Message Queuing Protocol, available at: `http://www.amqp.org` (2017.06.23).

[10] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, *Spark: cluster computing with working sets*, Proc. of the 2nd USENIX conference on Hot topics in cloud computing, pp.1-7, 2010.

[11] Storm: Distributed real-time computation system, Available at: `http://storm-project.net/` (2017.06.23).

[12] P. Carbon, S. Ewen, and S. Haridi, *Apache Flink: Stream and Batch Processing in a Single Engine*, IEEE Data Engineering: 28, 2015.

[13] B. Satzger, W. Hummer, P. Leitner, and S. Dustdar, *Esc: Towards an Elastic Stream Computing Platform for the Cloud*, Proc. of 2011 IEEE 4th International Conference on Cloud Computing, pp.348-355, 2011.

[14] J. Aspnes and G. Shah, *Skip Graphs*, ACM Transactions on Algorithms (TALG), Vol.3, No.4, 37:1-37:25, 2007.

[15] Y. Teranishi, T. Kimata, H. Yamanaka, E. Kawai, and H. Harai *Dynamic Data Flow Processing in Edge Computing Environments*, Proc. of 2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC 2017), pp.935-944, 2017.

[16] 安倍広多，寺西裕一, 高い Churn 耐性と検索性能を持つキー順序保存型構造化オーバレイネットワーク Suzaku の提案と評価，信学技報 Vol. 116, No. 362 (IA2016-65), pp. 11–16，2016.

[17] Y. Teranishi, Y. Saito, S. Murono, and N. Nishinaga, *JOSE: An Open Testbed for Field Trials of Large-scale IoT Services*, NICT Journal, Vol. 6, No. 2, pp. 151–159, 2015.

[18] R. Banno, S. Takeuchi, M. Takemoto, T. Kawano, T. Kambayashi, and M. Matsuo, *Designing Overlay Networks for Handling Exhaust Data in a Distributed Topic-based Pub/Sub Architecture*, Journal of Information Processing, vol.23, no.2, pp.105-116, 2015.

[19] Y. Teranishi, R. Banno, and T. Akiyama, *Scalable and Locality-Aware Distributed Topic-based Pub/Sub Messaging for IoT*, Proc. of IEEE Globecom 2015, pp. 1-7, 2015.

[20] Y. Teranishi, *PIAX: Toward a Framework for Sensor Overlay Network*, Proc. of 6th IEEE Consumer Communications and Networking Conference 2009 (CCNC 2009), pp. 1–5, 2009.