DEIM2022 K23-2

ネットワーク帯域幅の不確実性を考慮したエッジ・クラウドスケジューリング

深見　健太郎†　　有次　正義††
† 熊本大学大学院自然科学教育部　〒860−8555 熊本県熊本市中央区黒髪 2 丁目 39 番 1 号
†† 熊本大学大学院先端科学研究部　〒860−8555 熊本県熊本市中央区黒髪 2 丁目 39 番 1 号
E-mail: †fukami@st.cs.kumamoto-u.ac.jp, ††aritsugi@cs.kumamoto-u.ac.jp

あ ら ま し：近年, モバイル端末や IoT センサの増加に伴い大量のデータが生成され, リアルタイム処理を必要とするアプリケーションの需要が高まっている. パブリッククラウドは従量課金制で豊富なリソースを提供するが, 実際の性能や帯域幅はデータセンタの物理ハードウェアや同じマシンに割り当てられた他のユーザのリソースの使用状況によって変動する. そのため, アプリケーションのリアルタイム処理が困難な場合がある. 一方でエッジコンピューティングは, IoT デバイスやユーザの近くにあるエッジサーバで分散処理を行うことでアプリケーションのリアルタイム処理をサポートする. ここでエッジサーバの性能が均一ではない場合がある. 本研究では, Apache Storm を用いてエッジとパブリッククラウドで構成された Storm クラスタを構築し, ベイズ推定によりネットワーク帯域幅の不確実性を考慮したスケジューリングを行った. そして, 既存の Apache Storm のスケジューラと比較してリアルタイム処理を必要とするアプリケーションのエンドツーエンドレイテンシの減少を確認した.

1　はじめに

近年, AWS[1] などのパブリッククラウドは, 従量課金制で豊富なストレージや計算リソースを提供しており, アプリケーションなどの主要なタスクをクラウド上で処理することは効果的であると考えられている [2], [3], [4]. また, 近年モバイル端末や IoT センサの増加に伴い大量のデータが生成されるようになり, リアルタイムに処理を行う必要のあるアプリケーションの需要が高まっている [5]. しかし, パブリッククラウドの実際の性能や帯域幅は, データセンタの物理ハードウェアや同一のマシン上に割り当てられた他のユーザのリソースの使用状況によって変動するため, アプリケーションのリアルタイム処理が困難な場合がある. 一方で, IoT センサやユーザの近くに物理的に配置されたエッジーサーバで分散処理を行いリアルタイムな処理をサポートするエッジコンピューティングが提案されている [6].

パブリッククラウドでは, 豊富なストレージや計算リソースを利用できるが実際の性能や帯域幅の不確実性を考慮する必要がある. エッジサーバでは, アプリケーションの低遅延処理が期待できるが計算ノードの性能が均一であるとは限らない. そのため, クラウドとエッジ両方の利点を兼ね備えた環境においてアプリケーションのエンドツーエンドレイテンシを削減するスケジューラを考える必要がある.

本研究では, Apache Storm(以降, Storm) を用いてエッジとパブリッククラウドで構成された Storm クラスタを構築し, リアルタイムな処理を必要とする Real-Time Edge Vision Application[7] を実装する. さらに, ベイズ推定を用いてネットワーク帯域幅の不確実性を考慮したスケジューラを実装する. そして, アプリケーションのエンドツーエンドレイテンシを測定し, 既存の Storm のスケジューラとの比較を行う.

本論文は以下のように構成する. 第 2 章では, 本研究で使用している技術の関連研究について述べる. 第 3 章では, 本研究で使用しているリアルタイム分散処理システムである Storm について述べる. 第 4 章では, 本研究で実装しているアプリケーションについて述べる. 第 5 章では, 提案手法について述べる. 第 6 章では実験, 第 7 章では, 本論文のまとめについて述べる.

2　関連研究

エッジコンピューティング [6] とは, IoT センサやモバイル端末のデータ全てをクラウドに集めて処理するのではなくデータの一部または全てのデータの処理を IoT デバイスに近い場所で分散処理することである. しかし, エッジコンピューティングに用いるエッジサーバは必ずしも計算ノードの性能が均一であるとは限らない [7]. そのため, エッジコンピューティングは, 計算ノードの性能が不均一な環境でのレイテンシの削減を考える必要がある.

Storm には, Round Robin と Resource Aware Scheduler(以降, RAS)[8] の 2 つのスケジューラが実装されているがどちらのスケジューラも Storm クラスタの計算ノードの性能が全て均一であることを前提として設計されている. RAS は Topology 内の Spout と Bolt が必要とする CPU とメモリのリソースを Storm のユーザが登録し, 最も多くの利用可能な CPU とメモリのリソースを持つノードに優先してタスクを割り当てる. Round Robin はリソースに関する考慮はせずに各計算ノードにラウンドロビン方式でタスクの割り当てを行う.

Zhang ら [7] は, Storm を用いて計算ノードの性能が不均一であるエッジサーバで分散処理を行う Storm クラスタを構築した. そして, 計算ノードの性能が不均一である Storm クラスタにおいて, リアルタイム処理を

要求するアプリケーションのレイテンシを削減することを目的としたスケジューラである Latency aware Task Scheduler(以降, LaTS) を提案した．しかし, Zhang らは, Storm クラスタを全てエッジで構築しており, クラウドの利用を考慮していない．そのため, LaTS ではスケジューリングに用いるネットワーク帯域幅は変動しないことを前提としている．本研究では, LaTS のスケジューリングに用いるネットワーク帯域幅の不確実性に対処するためにベイズ推定によるネットワーク帯域幅の推定を行うベイズ推定を用いた LaTS を実装する．

また．Muhammad-Bello ら [9] は, IaaS クラウドにおけるリソースや性能の変動による不確実性に対処したロバストな期限制約付きワークフロースケジューリングアルゴリズムを提案した．Muhammad-Bello らは, IaaS クラウドの不確実性は, IaaS クラウドにおいてスケジューリングの際に考慮しなければならない重要な特異性であると述べている．そのため, タスクの実行時間の予測とクラウドリソースのプロビジョニングの遅延に関連する不確実性に対処している．タスクの実行時間の不確実性を考慮する際には, 実行時間の推定に正確な推定値を使用するのではなく, 不確実性係数を用いて実行時間の推定値に上限と下限を決定し求めた実行時間の区間を使用する．また, ワークフローのメイクスパンが期限制約を守るようにスケジューリングする際に, クラウドの VM(Virtual Machine) の起動時間を考慮している．本研究では, Storm クラスタを構築するインスタンスのネットワーク帯域幅の不確実性を考慮する．そして, ネットワーク帯域幅の推定には, ベイズ推定を用いて作成した事後分布を用いる．

3 Apache Storm

Storm[10] はオープンソースのリアルタイム分散処理システムである．Storm は, Spout, Bolt そして Topology の 3 つで構成される．Topology とは, Spout と Bolt から構成される有向非巡回グラフである．

Topology の構成の例を図 1 に示す．

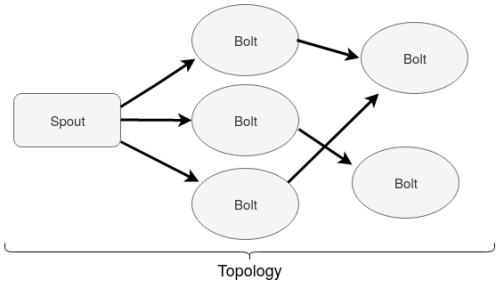

図 1　Topology の構成

Spout は Topology 内のストリームデータのソースであり, 外部ソースからタプルを読み取り, 後続の Bolt へ出力する．ここでのタプルとは処理されるデータの基本単位である．Bolt は, Spout または他の Bolt から受け取ったタプルに対して決められた処理を行い, 後続の Bolt へ新たなタプルを出力する．Storm は図 1 のような Topology を Storm クラスタに配置してストリームデータの処理を行う．

Storm クラスタは, Master Node, Slave Node, Zookeeper から構成される [11]．図 2 に Storm クラスタの構成の例を示す．

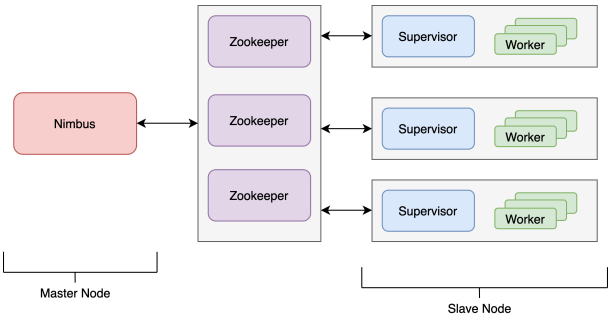

図 2　Storm クラスタの構成

Master Node は Slave Node へのタスクのスケジューリングを行う Nimbus というデーモンを実行する．Slave Node は Supervisor というデーモンを実行するノードであり, Supervisor が立ち上がった際に, Worker スレッドを Java Virtual Machine(以降, JVM) に立ち上げる．そして, Topology が実行される際に, Nimbus が各 Slave Node 上の Worker スレッドに Spout, Bolt のタスクの割り当てを行い Topology の処理が実行される．Zookeeper は, Master Node と Slave Node の間で Nimbus, Supervisor の状態を記録し, 保持することで, Storm クラスタのノード間の状態管理と同期を行う．

4 Real-Time Edge Vision Application

Real-Time Edge Vision Application とは, モバイル端末や IoT 機器で撮影された画像, 動画データを入力とし, フレームごとに処理を行うリアルタイムアプリケーションである．本研究で Topology として実装するのは, ステレオカメラから取得した 2 枚の画像データを入力とし視差画像を出力するアプリケーション [7] である．図 3 にアプリケーションの Topology の構成を示す．

以下に, 実装する Topology の Spout と Bolt について記述する．

- Spout: ステレオカメラで撮影した 2 枚の画像をタプルとして後続の Bolt へ出力する．
- Rectification and Partition Bolt: ステレオカメラのレンズによる歪みを取り除くため画像のキャリブレーションと平行化を行った後, 画像の分割を行う．
- Disparity Bolt: 分割後の画像から視差の計算を行う．
- Merge and Reprojection Bolt: 分割された画像の

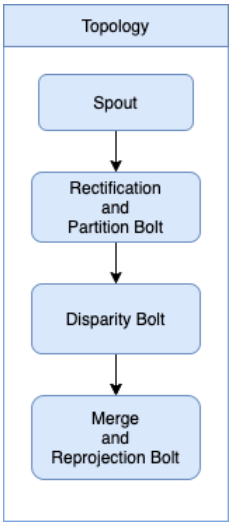

図 3　Real-Time Edge Vision Application の Topology の構成

マージと視差画像の出力を行う.

図 4 にステレオカメラで撮影した画像, 図 5 にステレオカメラのレンズによる歪みを取り除いた画像そして図 6 に出力された視差画像の例を示す.

図 4　ステレオカメラで撮影した画像

図 5　レンズによる歪みを取り除いた画像

5　提案手法

本章では, 本研究で実装するベイズ推定を用いた LaTS の詳細について述べる.

5.1　ベイズ推定を用いた LaTS

LaTS は, Storm クラスタの計算ノードの性能が不均一である環境においてレイテンシを削減することを目的としたスケジューラである [7].

LaTS は以下の 3 つの要素で構成されている.

図 6　出力された視差画像

- タスクの要求リソースと実行時間の推定.
- Storm クラスタ内の利用可能なリソースの監視.
- Latency Aware Task Scheduling.

5.1.1　タスクの要求リソースと実行時間の推定

タスクのスケジューリングのために, タスクの実行時間, タスクが要求するメモリ, ネットワーク使用量, CPU 使用率の 4 つを利用する.

メモリ使用量とネットワーク使用量はタスクがどの計算ノードで実行されても一定なので JVM の ThreadMXBean[12] を用いて取得する. タスクの実行時間の推定には推定モデルを作成する. まず, CPU 使用率を 5% から 100% まで 5% 毎に変動させながら各タスクの実行時間を測定する. 実行時間の測定には JAVA Timer API[13] を使用する. そして, 測定値を用いて予測平均二乗誤差が最小となるよう 3 次の多項式曲線を導出しタスクの実行時間を推定する推定モデルを作成する. CPU 使用率は ThreadMXBean で取得する.

5.1.2　Storm クラスタ内の利用可能なリソースの監視

Storm クラスタの各計算ノードで利用可能なリソースを取得する. 取得するのは以下の 4 つである.

(1) lscpu ユーティリティを使用して計算ノードの CPU クロック周波数と使用率を取得する.
(2) Storm のデーモンから Worker プロセスのメモリ使用量を取得する.
(3) ThreadMXBean を使用して計算ノードの CPU 使用率を取得する.
(4) ベイズ推定を用いて計算ノードのネットワーク帯域幅を推定し取得する.

5.1.3　Latency Aware Task Scheduling

割り当てられるタスクのプールが与えられた場合に, CPU 時間の降順にソートを行い 1 つずつ割り当てを行う. その際, 計算ノードの CPU 使用率を測定し, 5.1.1 節で作成した推定モデルを使用してタスクの実行時間を推定する. 次に, ベイズ推定を行い取得したネットワーク帯域幅と Bolt の出力するタプルのデータサイズを用いてデータ転送時間を推定する. そして, 推定した 2 つの推定値の合計時間が最小となる計算ノードへ Bolt の割り当てを行う. その後, 割り当てられた Bolt によって消費される CPU 使用率, メモリ, そして帯域幅を利用可

能なリソースから削除する．全ての Bolt の割り当てが完了するまで上記の処理を繰り返す．

5.2 ネットワーク帯域幅のベイズ推定

本研究では, Storm クラスタにクラウドを利用することで発生するネットワーク帯域幅の不確実性に対処するためにベイズ推定を用いて帯域幅の推定を行う．

帯域幅の推定には, 下記の式 [14] を用いる．表 1 に主な記号と定義を示す．

$$\bar{\mu} = \frac{\frac{n}{\sigma^2}\bar{x} + \frac{1}{\tau^2}\eta}{\frac{n}{\sigma^2} + \frac{1}{\tau^2}} \tag{1}$$

$$\bar{\sigma^2} = \frac{\frac{\sigma^2\tau^2}{n}}{\frac{\sigma^2}{n} + \tau^2} \tag{2}$$

表 1　主な記号と定義

記号	定義
μ	正規母集団の母平均
σ^2	正規母集団の母分散
n	正規母集団から抽出したデータ数
$\bar{x}$	標本平均
η	事前分布に用いる正規分布の平均
τ^2	事前分布に用いる正規分布の分散
$\bar{\mu}$	事後分布の平均
$\bar{\sigma^2}$	事後分布の分散

(1) 式, (2) 式は, 母平均 μ, 母分散 σ^2 に従う正規母集団から n 個の標本を抽出し, 標本平均を $\bar{x}$, 事前分布を平均 η, 分散 τ^2 の正規分布とする場合の事後分布の平均 $\bar{\mu}$, 分散 $\bar{\sigma^2}$ を導出する式である．本研究では, 事前実験を行い Storm クラスタの帯域幅を計測する．そして, 計測した帯域幅を用いてベイズ推定を行う．事前実験については 6 章にて示す．

6 実験

本章では, 提案手法であるスケジューラと既存の Storm のスケジューラを用いて Topology を実行する．そして, エンドツーエンドのレイテンシを測定した実験の結果について示す．

6.1 実験 1

本実験では, Storm に 4 章で述べた Topology を実装する．Storm クラスタは, 4 台の実マシンで構成したエッジと AWS の EC2 提供の 3 台のインスタンスで構築する．そして, エッジサーバにおいてアプリケーションのエンドツーエンドレイテンシの削減が見込める環境で Topology を実行する．また, 実験には, 提案手法であるベイズ推定を用いた LaTS, LaTS, RAS, そして Round Robin の 4 つのスケジューラを使用する．Topology は, ステレオカメラで撮影した解像度 640 × 480 のグレースケール画像を入力として, スケジューラ毎に 1fps で 3 分間実行する．そして, 1 フレーム毎に発生するエンドツーエンドレイテンシを測定し比較を行う．実験に使用したマシンとインスタンスのスペックを表 2, 表 3 にそれぞれ示す．

6.1.1 実験 1 におけるベイズ推定のための事前実験

スケジューリングに用いる帯域幅の推定のために, 5.2 節の (1) 式, (2) 式に用いる母集団と標本平均の収集を行う．実験環境は前述の通りである．

母集団の収集のために, Storm クラスタを構成する各マシン間の帯域幅を 1 秒毎に 600 秒間分の測定を行う．帯域幅の測定には, iperf を用いる．そして, 各マシン毎に収集された 3000 件の帯域幅から 5 件ずつのランダムサンプリングを 600 回行う．次に, 中心極限定理よりサンプリングした帯域幅 5 件毎の平均からなる正規分布を作成し, 母平均と母分散を決定する．標本平均の収集には, Storm クラスタを構成する各マシン間の帯域幅を 1 秒毎に 180 秒間分の測定を行う．そして各マシン毎に収集された帯域幅の平均を標本平均としてベイズ推定に用いる．事前分布は平均 500, 標準偏差 150 の正規分布を使用する．

収集された母集団の母分散, 事前分布の平均, 分散, そして, 標本平均を 5.2 節の (1) 式, (2) 式に用いて得られた事後分布の平均を帯域幅の推定値としてスケジューリングに用いる．推定した帯域幅を表 4 に示す．また, iperf を用いて測定した LaTS のスケジューリングに使用する帯域幅を表 5 に示す．

6.1.2 実験 1 の実験結果

6.1 節の実験結果を, 図 7, 表 6 に示す．図 7, 表 6 より最小値から最大値までの値全てにおいて, ベイズ推定を用いた LaTS のレイテンシが最も小さい値を示した．RAS は, ベイズ推定を用いた LaTS の次に低いレイテンシを示した．Round Robin は最小値はベイズ推定を用いた LaTS, RAS と近い値を示しているが最大値が他のスケジューラと比較して最も大きい値を示しており, 図 7 を見るとレイテンシの値のばらつきが大きいことがわかる．また, LaTS は, 中央値から最大値は Round Robin より小さい値を示したが最小値から第 1 四分位数のレイテンシでは他のスケジューラと比較して最も大きい値を示した．

また, Storm UI[15] を用いて各スケジューラの割り当てを確認した結果, ベイズ推定を用いた LaTS では, エッジである i9-9900K のマシンに全てのタスクの割り当てが行われていた．そのため, マシン間のデータ通信による遅延が発生せずレイテンシが削減されたと考えられる．RAS は, i9-9900K と i7-3930K の 2 台にタスクの割り当てが行われており, ベイズ推定を用いた LaTS と比較するとマシン間のデータ通信による遅延が発生したため, ベイズ推定を用いた LaTS の次に小さいレイテンシを示したと考えられる．Round Robin は全てのマ

表 2　実験 1 に使用した実マシンのスペック

CPU	クロック周波数 (GHz)	コア数	メモリ (GB)	台数	ノード
i9-9900K	3.60	16	32	1	Slave Node
i7-3930K	3.20	12	16	1	Slave Node
i7-3770K	3.50	8	8	1	Master Node
i7-860	2.80	8	8	1	Slave Node

表 3　実験 1 に使用したインスタンスのスペック

インスタンスタイプ	CPU	クロック周波数 (GHz)	コア数	メモリ (GB)	台数	ノード
m4.xlarge	Xeon E5-2686	2.30	4	16	3	Slave Node

表 4　実験 1 における帯域幅の推定値

マシン	帯域幅 (Mbps)
i9-9900K	725
i7-3930K	718
i7-860	720
Xeon E5-2686	502
Xeon E5-2686	503
Xeon E5-2686	476

表 5　実験 1 における LaTS の帯域幅の測定値

マシン	帯域幅 (Mbps)
i9-9900K	943
i7-3930K	942
i7-860	943
Xeon E5-2686	881
Xeon E5-2686	879
Xeon E5-2686	880

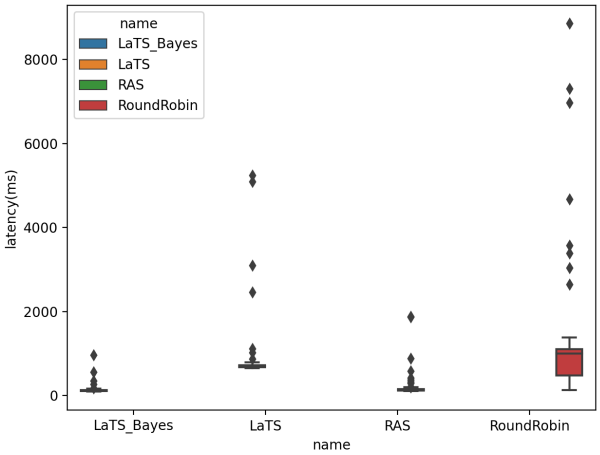

図 7　実験 1 におけるスケジューラ毎の実験結果

表 6　実験 1 における 1 フレームに発生するエンドツーエンドレイテンシ

スケジューラ	LaTS Bayes	LaTS	RAS	Round Robin
最小値 (ms)	91	648	106	135
第 1 四分位数 (ms)	106	675	124	481
中央値 (ms)	118	698	137	988
第 3 四分位数 (ms)	131	720	154	1104
最大値 (ms)	560	6617	1884	8857

シンにタスクの割り当てが行われておりエッジとクラウド間でデータ通信が非常に多く発生したためレイテンシの値が大きくなったと考えられる．LaTS はベイズ推定を用いた LaTS と違い i9-9900K と Xeon E5-2686 の 2 台にタスクの割り当てが行われており，エッジとクラウド間のデータ通信による遅延が発生したためレイテンシの値が大きくなったと考えられる．また，LaTS は，スケジューリングに使用する Xeon E5-2686 の帯域幅に iperf で測定した帯域幅の最大値を使用したためベイズ推定を用いた LaTS と異なるスケジューリング結果になったと考えられる．

6.2　実験 2

本実験では，3 台の実マシンで構成したエッジと AWS の EC2 提供の 3 台のインスタンスで Storm クラスタを構成する．Topology には，4 章で述べた Rectification and Partition Bolt, Disparity Bolt, そして, Merge and Reprojection Bolt の 3 つの Bolt に標準正規分布に従う乱数を用いて作成した 2 つの 400×400 の正方行列の積を計算する処理を追加したものを実装する．そして，インスタンス側で処理を行うことでエンドツーエンドレイテンシの削減が見込める環境で Topology を実行する．実験には，6.1 と同様に提案手法であるベイズ推定を用いた LaTS, LaTS, RAS, そして Round Robin の 4 つのスケジューラを使用する．そして，ステレオカメラで

表 7　実験 2 に使用した実マシンのスペック

CPU	クロック周波数 (GHz)	コア数	メモリ (GB)	台数	ノード
i7-3770K	3.50	8	8	1	Master Node
i7-3930K	3.20	12	16	1	Slave Node
i7-860	2.80	8	8	1	Slave Node

表 8　実験 2 に使用したインスタンスのスペック

インスタンスタイプ	CPU	クロック周波数 (GHz)	コア数	メモリ (GB)	台数	ノード
c5a.2xlarge	AMD EPYC 7R32	3.30	8	16	3	Slave Node

表 9　実験 2 における帯域幅の推定値

マシン	帯域幅 (Mbps)
i7-3930K	643
i7-860	641
AMD EPYC 7R32	1951
AMD EPYC 7R32	1926
AMD EPYC 7R32	1954

表 10　実験 2 における LaTS の帯域幅の測定値

マシン	帯域幅 (Mbps)
i7-3930K	941
i7-860	943
AMD EPYC 7R32	4760
AMD EPYC 7R32	4750
AMD EPYC 7R32	4800

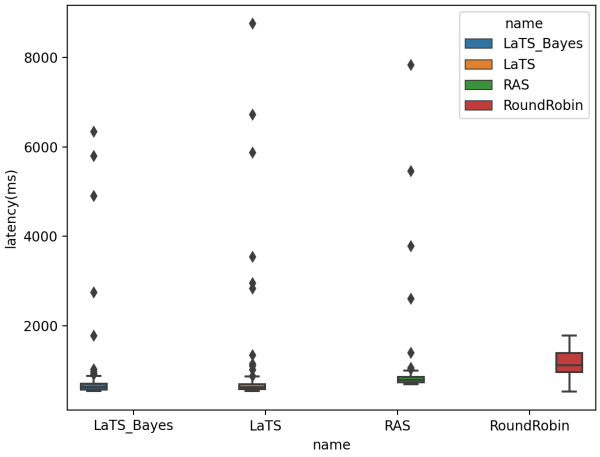

図 8　実験 2 におけるスケジューラ毎の実験結果

表 11　実験 2 における 1 フレームに発生するエンドツーエンドレイテンシ

スケジューラ	LaTS Bayes	LaTS	RAS	Round Robin
最小値 (ms)	536	538	691	536
第 1 四分位数 (ms)	577	585	734	963
中央値 (ms)	629	624	786	1116
第 3 四分位数 (ms)	701	691	849	1329
最大値 (ms)	6347	8755	7835	1779

撮影した解像度 640 × 480 のグレースケール画像を入力として, Topology をスケジューラ毎に 1fps で 3 分間実行し, 1 フレーム毎に発生するエンドツーエンドレイテンシを測定し比較を行う. 実験に使用したマシンとインスタンスのスペックを表 7, 表 8 にそれぞれ示す.

6.2.1　実験 2 におけるベイズ推定のための事前実験

6.1.1 節と同様にスケジューリングに用いる帯域幅の推定のために, 5.2 節の (1) 式, (2) 式に用いる母集団と標本平均の収集を行う. 実験環境は 6.1.1 節とは異なるためエッジサーバの実マシンが 1 台減っているが実験方法は 6.1.1 節と同じである. 推定した帯域幅を表 9 に示す. また, iperf を用いて測定した LaTS のスケジューリングに使用する帯域幅を表 10 に示す.

6.2.2　実験 2 の実験結果

実験結果を, 図 8, 表 11 に示す. 図 8, 表 11 より, ベイズ推定を用いた LaTS と LaTS は最小値から第 3 四分位数までほとんど同じ値を示した. RAS は, ベイズ推定を用いた LaTS, LaTS と比較して最小値から第 3 四分位数まで高いレイテンシを示した. Round Robin は, 最小値においてベイズ推定を用いた LaTS, LaTS とほとんど同じ値を示した. また, 最大値においては最も低い値を示したが第 1 四分位数から第 3 四分位数において他のスケジューラと比較して最も高いレイテンシを示した.

また, Storm UI を用いて各スケジューラの割り当てを確認した結果, ベイズ推定を用いた LaTS と LaTS は

どちらもクラウド側のマシンである 3 台の AMD EPYC 7R32 にすべての Bolt が割り当てられておりスケジューリング結果は同様のものとなっていた．RAS は, 全ての Bolt がエッジである i7-3930K のマシンに割り当てられており, ベイズ推定を用いた LaTS と LaTS が割り当てを行なった AMD EPYC 7R32 との計算能力の差によってレイテンシが高くなったと考えられる．Round Robin は, 全てのマシンに割り当てを行なっておりエッジとクラウド間でデータ通信が非常に多く発生したためレイテンシの値が大きくなったと考えられる．また, 図 8, 表 11 においてレイテンシの最大値が最も低い値を示していたが, 3 分間の Topology の実行時間中に処理を終えていないタプルが存在しており, 実際には最小値から最大値のレイテンシ全てにおいて他のスケジューラと比較して高いレイテンシとなっていると考えられる．

7 終わりに

本研究では, パブリッククラウドを利用する際に発生するネットワーク帯域幅の不確実性に対処するためにベイズ推定によるネットワーク帯域幅の推定を行うスケジューラを実装した．実験の結果, 既存の Storm のスケジューラと比較してベイズ推定を用いた LaTS は, エッジサーバにおいてアプリケーションのエンドツーエンドレイテンシの削減が見込める環境とクラウド側においてエンドツーエンドレイテンシの削減が見込める環境の両方においてレイテンシの削減を確認できた．

今後の展望として, アプリケーションを複数同時に実行する場合やアプリケーションを実行する際の fps 数を上昇させた場合など高負荷な環境においてもレイテンシを削減できるかを検討する．

参考文献

[1] AWS. `https://aws.amazon.com`. (2021.12.26 参照).

[2] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, pp. 301–314, 2011.

[3] Yasuhiro Yamasaki and Masayoshi Aritsugi. A case study of iaas and saas in a public cloud. In *2015 IEEE International Conference on Cloud Engineering*, pp. 434–439. IEEE, 2015.

[4] Jashwant Raj Gunasekaran, Prashanth Thinakaran, Mahmut Taylan Kandemir, Bhuvan Urgaonkar, George Kesidis, and Chita Das. Spock: Exploiting serverless functions for slo and cost aware resource procurement in public cloud. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pp. 199–208. IEEE, 2019.

[5] Kiryong Ha, Zhuo Chen, Wenlu Hu, Wolfgang Richter, Padmanabhan Pillai, and Mahadev Satyanarayanan. Towards wearable cognitive assistance. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, pp. 68–81, 2014.

[6] Pedro Garcia Lopez, Alberto Montresor, Dick Epema, Anwitaman Datta, Teruo Higashino, Adriana Iamnitchi, Marinho Barcellos, Pascal Felber, and Etienne Riviere. Edge-centric computing: Vision and challenges. *ACM SIGCOMM Computer Communication Review*, Vol. 45, No. 5, pp. 37–42, 2015.

[7] Wuyang Zhang, Sugang Li, Luyang Liu, Zhenhua Jia, Yanyong Zhang, and Dipankar Raychaudhuri. Hetero-edge: Orchestration of real-time vision applications on heterogeneous edge clouds. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pp. 1270–1278. IEEE, 2019.

[8] Apache Storm Resource Aware Scheduler. `https://storm.apache.org/releases/1.2.3/Resource_Aware_Scheduler_overview.html`. (2021.12.26 参照).

[9] Bilkisu Larai Muhammad-Bello and Masayoshi Aritsugi. A robust algorithm for deadline constrained scheduling in iaas cloud environment. *Ieice Transactions on Information and Systems*, Vol. 101, No. 12, pp. 2942–2957, 2018.

[10] Apache Storm. `https://storm.apache.org`. (2021.12.26 参照).

[11] Storm cluster. `https://storm.apache.org/releases/1.2.3/Setting-up-a-Storm-cluster.html`. (2021.12.26 参照).

[12] java.lang.management インタフェース ThreadMXBean. `https://docs.oracle.com/javase/jp/8/api/java/lang/management/ThreadMXBean.html`. (2021.12.26 参照).

[13] java.util.timer. `https://docs.oracle.com/javase/jp/8/docs/api/java/util/Timer.html`. (2021.12.26 参照).

[14] 姜興起. ベイズ統計データ解析. 東京:共立出版, 2010 年 7 月. 金明哲 (編).

[15] Storm UI. `https://storm.apache.org/releases/1.2.3/STORM-UI-REST-API.html`. (2021.12.26 参照).