データセンターネットワークでの輻輳対策どうしてる？

深澤 開 / 小林 正幸

LINEヤフー

登壇者紹介

深澤 開 / Fukazawa Kai

所属: ネットワーク開発チーム兼バックボーンネットワークチーム

元Hadoopエンジニアの現ネットワークエンジニア

業務: データセンターおよびバックボーンネットワークの運用

過去のJANOG登壇: Yahoo! JAPAN アメリカデータセンタとネットワーク変遷 :: JANOG52

小林 正幸 / Masayuki Kobayashi

所属: ネットワーク開発チーム シニアネットワークエンジニア

業務: データセンターネットワーク運用, AI/HPC用ネットワーク設計などを担当

過去のJANOG登壇: LINEのネットワークをゼロから再設計した話 :: JANOG43

このプログラムの概要

Agenda

データセンターネットワークでの輻輳の課題と代表的な対策手法を整理・再考するとともに、
実環境を使ったユーザ視点での検証結果から見えてきたことを共有します

発表者自身この領域について試行錯誤中なので、皆様との議論を通じて理解を深めたいと思います

間違っている部分があれば遠慮なくお知らせください

目次

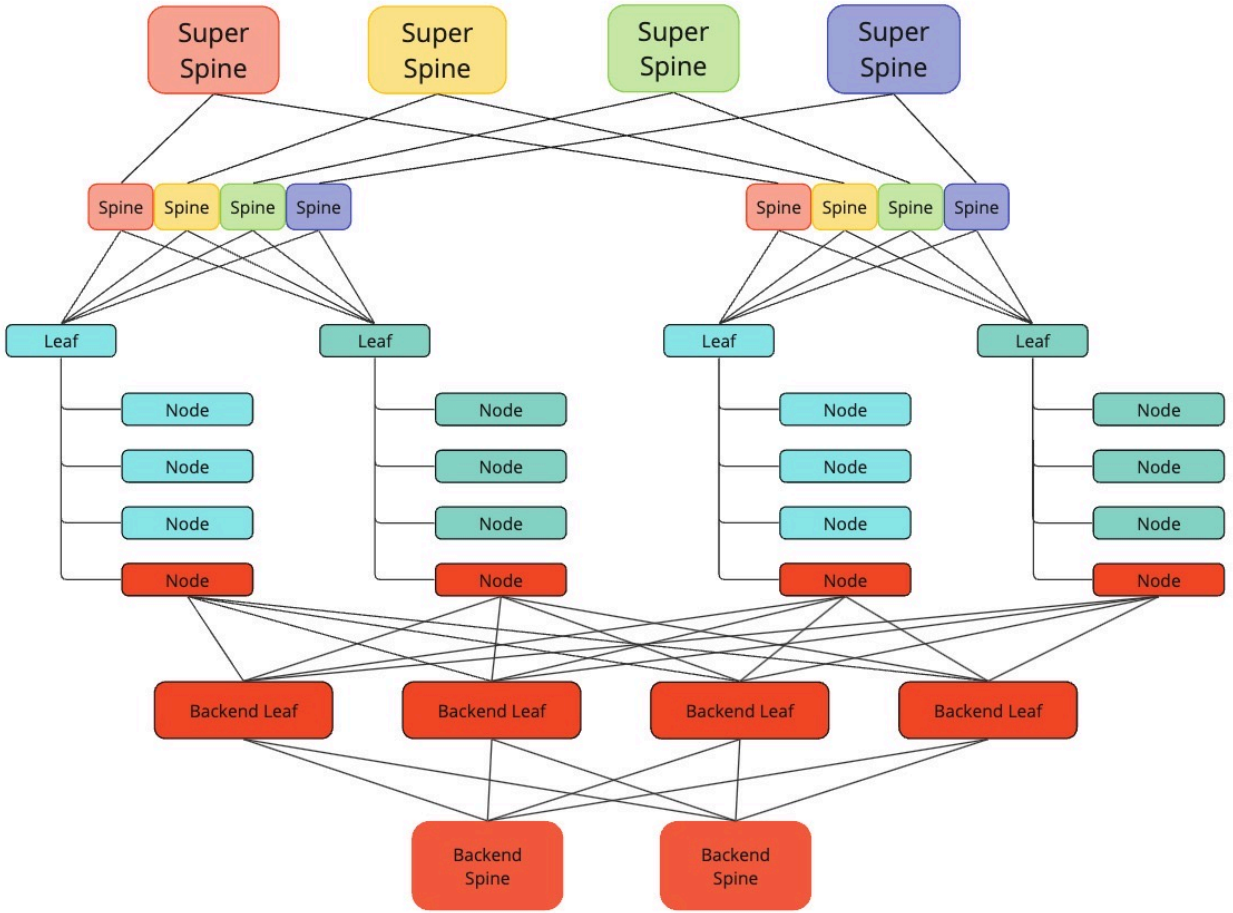
- データセンターネットワークでの輻輳と課題
- 輻輳制御手法の整理
- Hadoop環境での検証
- 考察
- 議論

データセンターネットワークと輻輳

課題とモチベーション

データセンターネットワークのいま

ワークロードの混在と複雑化する通信品質要求



Frontend Fabric

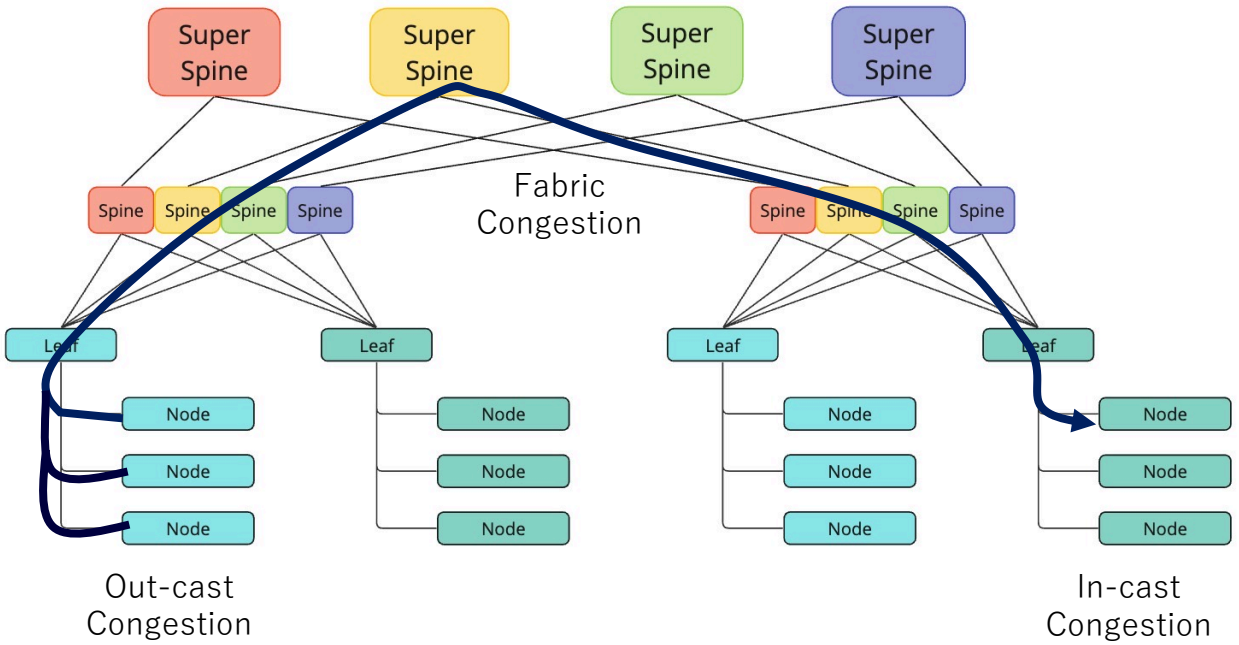
- Compute node(CPU)の通信
- NWリソースを共有する
- パケットロスが起きる前提
- 多様なアプリのflowが混在

Backend Fabric

- HPC Acceleratorの通信
- クラスタ間通信でNWを専有
- パケットロスが許容されない
- 特定アプリのflowが占有傾向

データセンターネットワークと輻輳

どこで輻輳が発生するのか



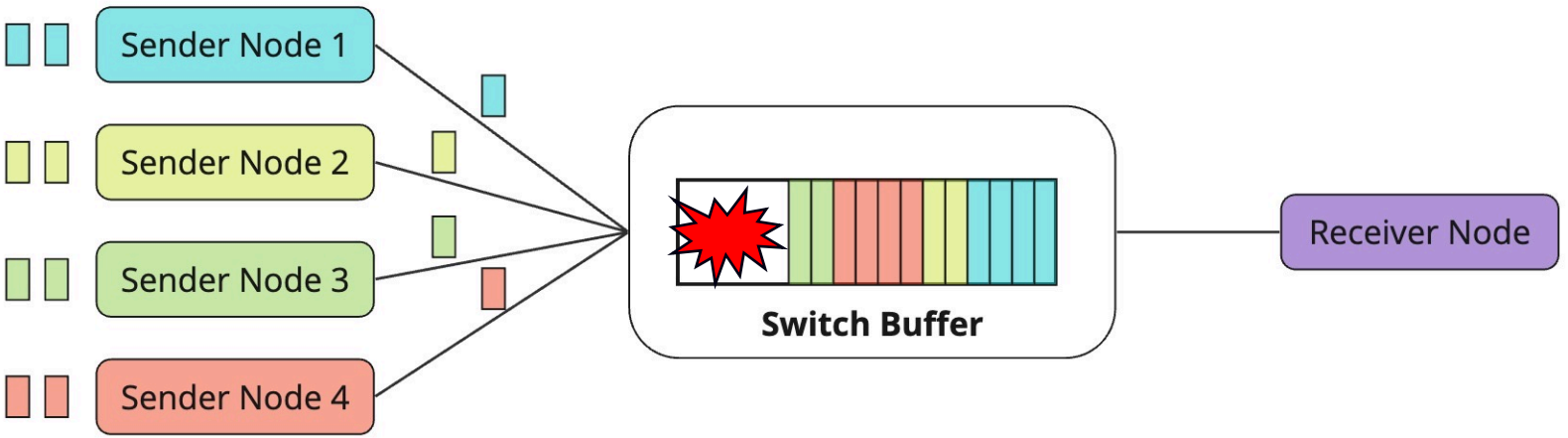
- **In-cast congestion**
  - 複数の送信元が1つの宛先にデータを同時に送信する多対一の通信
  - 受信側バッファでtail dropが発生する
  - 現在のデータセンターネットワークで問題になりやすい
  - 発生したときの対応が難しい
- Fabric congestion
  - 不均衡なトラフィック分散やフローの偏りによってファブリック内のバッファでパケットロスが発生する
  - DLBやGLB※での対策が推奨される
- Out-cast congestion
  - 特定の出力キューが総リンク帯域幅を超える速度でデータを送信するときに発生する
  - 送信元が特定されるため対応は比較的容易

※ Dynamic Load Balancing / Global Load Balancing

データセンターネットワークと輻輳

In-cast congestion

In-castはデータセンターネットワークの特性上、回避が困難



輻輳の制御 ≒ バッファの制御

Active Queue Management (AQM)

データセンターネットワークと輻輳

どんな通信が輻輳を起こすのか

Bimodalなデータセントートラフィック

Mice Flow

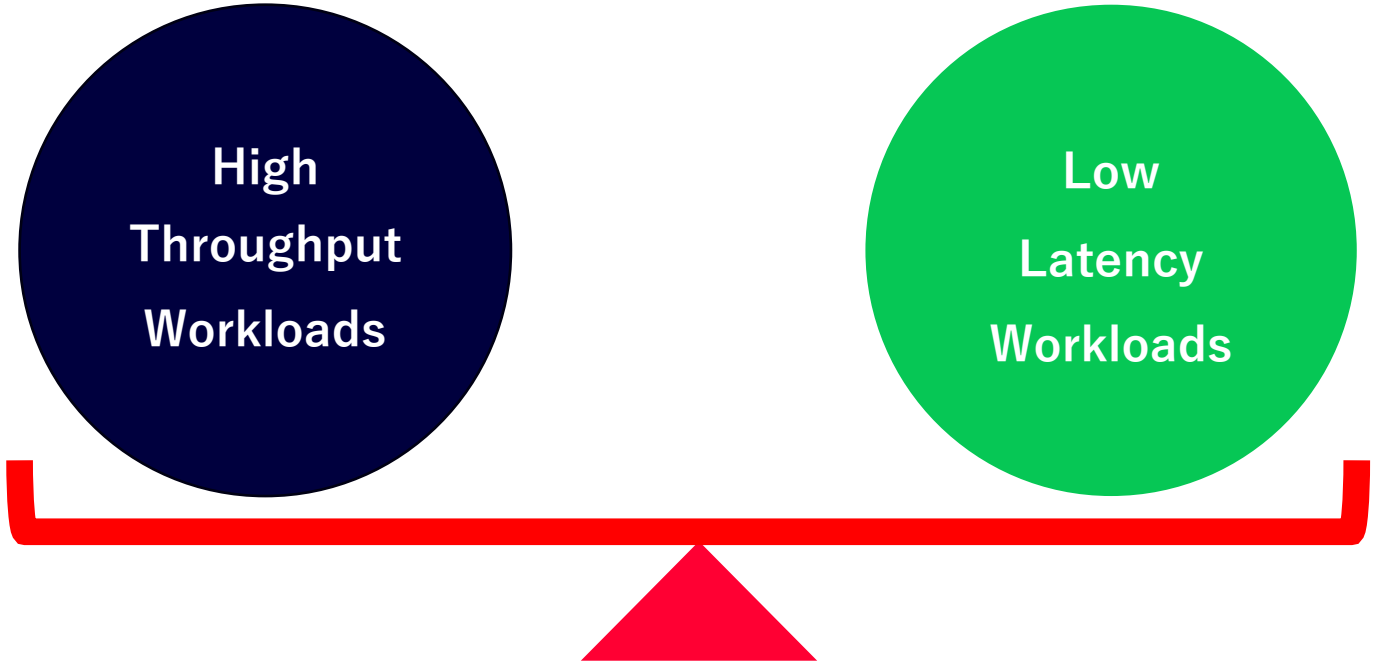
- 存続期間が短く低レートな通信
- データセンターのフローの大部分を占めるが、総トラフィック量の1~2割程度
- 遅延の影響を受けやすく、クエリや制御メッセージで構成される
- アプリケーションの性能低下を防ぐため、パケットロスを最小限またはゼロにする必要がある

Elephant Flow

- 存続期間が長く高レートな通信
- データセンターの総トラフィック量のほとんどが少数のElephant Flow
- バックアップや分散処理などの大規模なデータ転送が該当し、高いスループットを必要とする

データセンターネットワークと輻輳

公平性(fairness)の担保



可能な限り最大のスループットと低い遅延を様々なワークロードに提供したい

しかし、スイッチのバッファ内のフローを動的に区別し運用することは一般的に難しい

輻輳と回復力

Lossy vs Lossless

アプリケーションによってパケットロスへの許容度と回復力が異なる

Lossy

- 現在の多くのアプリケーションはパケットロスをTCPの再送などでカバーしている
- パケットロスが起きることを前提とした設計で、厳密なパケット順序付け制御が必要ない
- ホスト側のTCP輻輳制御アルゴリズムのスタンドアロンで運用されることが一般的

Lossless

- パケットロスを許容せず、厳密なパケット順序付け制御が必要になる
- 昨今導入が進んでいるAIのGPUクラスタネットワークなどがこれに該当する
- ホストがネットワーク機器からのフィードバックを受ける形で厳密に運用される

輻輳と回復力

Lossy vs Lossless

アプリケーションによってパケットロスへの許容度と回復力が異なる

Lossy

本プログラムの検証対象

- 現在の多くのアプリケーションはパケットロスをTCPの再送などでカバーしている
- パケットロスが起きることを前提とした設計で、厳密なパケット順序付け制御が必要ない
- ホスト側のTCP輻輳制御アルゴリズムのスタンドアロンで運用されることが一般的

Lossless

- パケットロスを許容せず、厳密なパケット順序付け制御が必要になる
- 昨今導入が進んでいるAIのGPUクラスタネットワークなどがこれに該当する
- ホストがネットワーク機器からのフィードバックを受ける形で厳密に運用される

輻輳への対応策

分類とメリット・デメリット

大きく分けて3パターン

#	実現方法	NW運用者視点のメリット	NW運用者視点のデメリット
1	専用のネットワークを構築する	NWリソースを占有可能	機器と運用のコストが増加
2	輻輳に強いハードウェアを導入する	ホスト側に設定が不要	機器コストが増加
3	輻輳制御の設定をチューニングする	機器コストを削減可能	ホスト側にも設定が必要

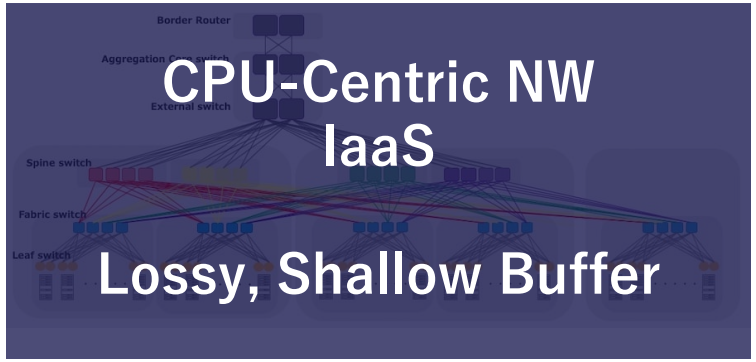
輻輳への対応策

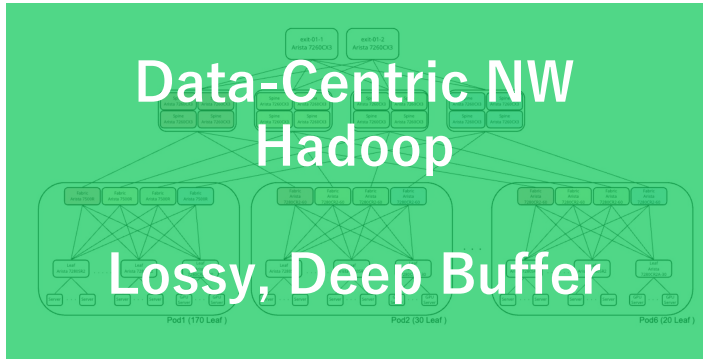
分類とメリット・デメリット

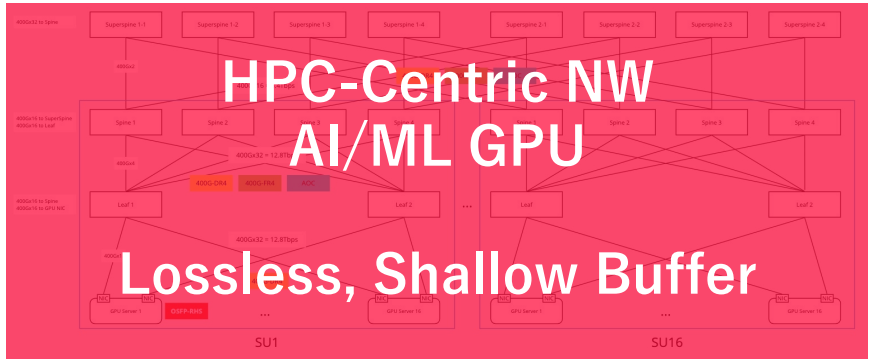
これまで旧ヤフーではシステムの特性ごとに専用構成のネットワークを構築してきた

また通常のバッファ容量の機器では輻輳が予想される環境では、大容量バッファの機器も導入してきた

#	実現方法	NW運用者視点のメリット	NW運用者視点のデメリット
1	専用のネットワークを構築する	NWリソースを占有可能	**機器と運用のコストが増加**
2	輻輳に強いハードウェアを導入する	ホスト側に設定が不要	**機器コストが増加**
3	輻輳制御の設定をチューニングする	機器コストを削減可能	ホスト側にも設定が必要







課題とモチベーション

これからのデータセンターネットワークが目指す形

課題

- 用途別の専用構成が乱立しインフラがフラグメント化、構築・運用のコストが増加している
- 大容量なバッファを持つ機器を運用しているが、 そのコストの妥当性が不明
- LossyなNWでも異なる機器や設定を運用しなければならないことの負荷が大きい

我々のモチベーション

- ハードウェアや構成・設定のバリエーションを増やしたくない
- 可能な限り構築・運用のコストパフォーマンスの高い標準構成を定義したい
- 今後の機種選定やconfiguration, コスト算出の根拠となるデータが欲しい

このプログラムの背景

まとめると

旧ヤフーのHadoop環境などでは、NW品質の安定を目的として大容量バッファの機器を導入してきた

実際にどれくらいの効果が出ているか測定が出来ておらず、適したコストで運用できているか不明だった

運用をはじめて数年経ったいま、改めて振り返ると改善点があるのではないか？

また最近はAI/MLなど様々なワークロードを収容する必要がありそれぞれ要求が異なってくる

運用の観点で、採用するハードウェアの種類や設定のバラエティを増やしたくない

可能な限り構築・運用のコストパフォーマンスの高い標準構成を定義したい

我々に本当に必要だったハードウェア要件と輻輳管理の手法を確立させるための根拠が欲しいので、

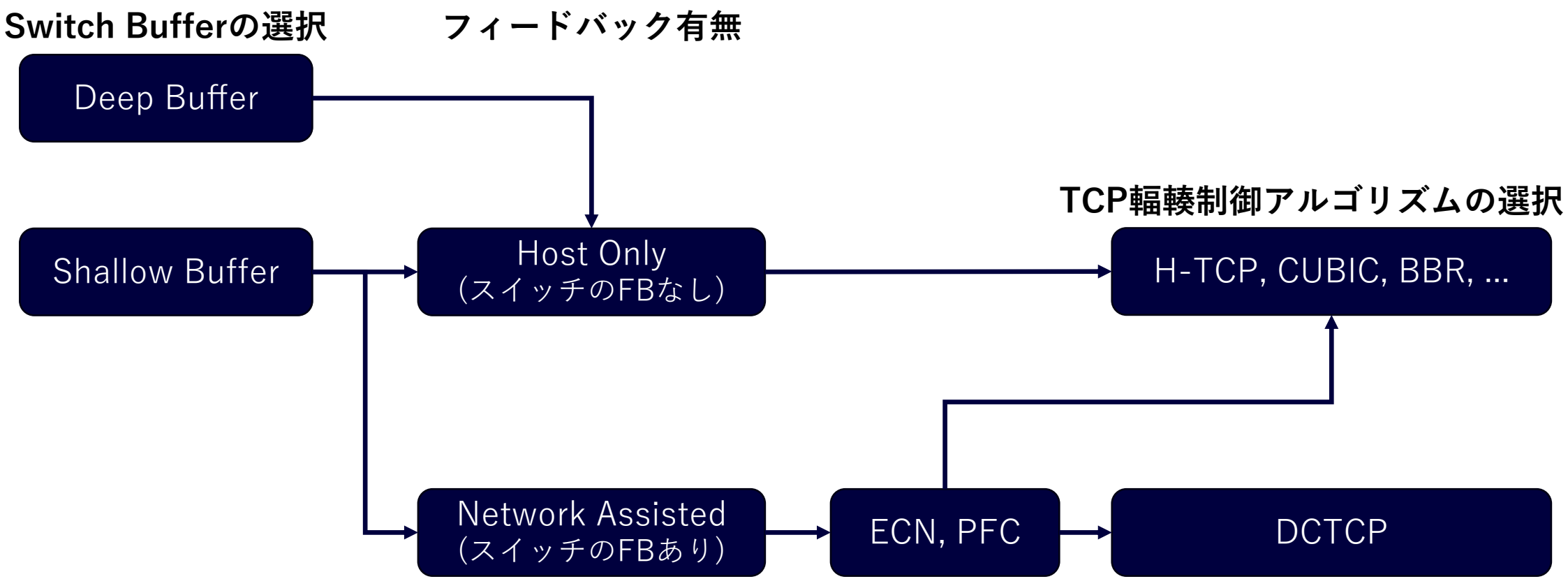
既存の課題を振り返りデータセンターネットワークでの輻輳制御手法を再考したい！！

輻輳制御の手法

代表的な技術の整理

バッファ管理手法の組み合わせ

バッファ実装と輻輳通知をフィードバックするかしないか



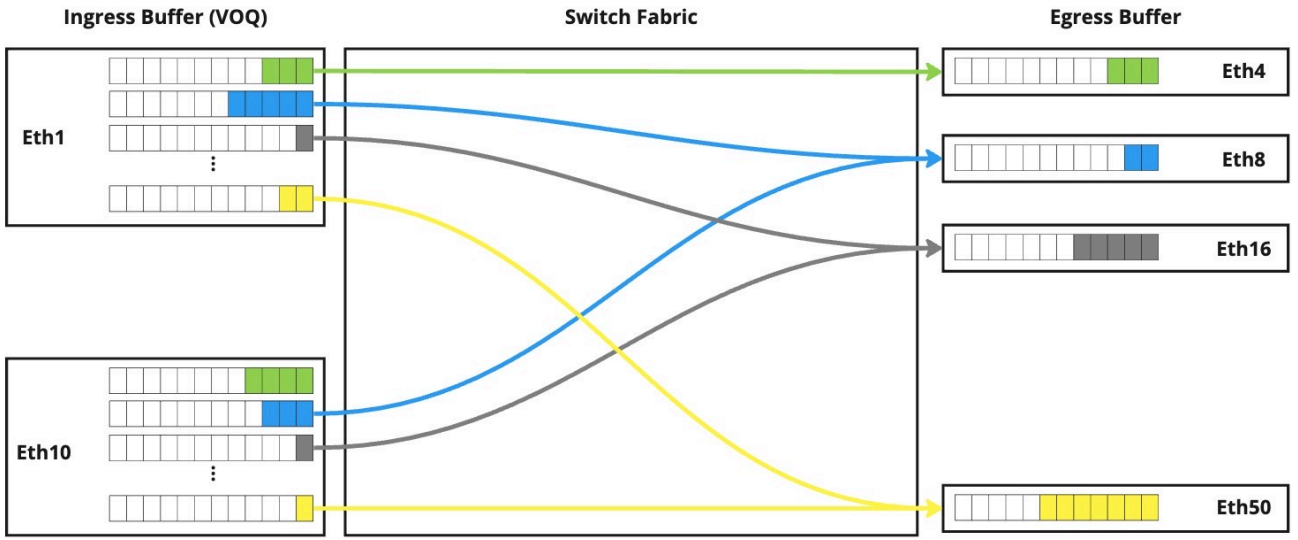
Deep Buffer

VOQ architecture と Bufferbloat

Deep Buffer

VOQ (Virtual Output Queuing)

- Switch ASICに搭載されたGB単位の大容量メモリ(HBM)でバーストトラフィックを吸収する方式
- HOLB※の対策として、各入力ポートに出力ポート分の仮想出力キュー(VOQ)が実装されている
- ポート数Nのスイッチの場合、NxNのVOQになる



※ HOLB (Head of Line Blocking)

Deep Buffer

Bufferbloat問題

- バッファサイズは大きければ良いというものではない
- パケットがバッファ内にストアされることによって生じる遅延の問題(Bufferbloat)がある
- 遅延に敏感なアプリケーションでは品質低下につながる
- Loss-basedなTCP輻輳制御アルゴリズムとの相性が良くないとされている
    - パケットロスが発生するまでバッファを埋め尽くす動作をするため
- 対策として CoDel (Controlled Delay) などのAQMアルゴリズムが考案されている
    - キューイング遅延が閾値を超えたパケットを破棄する
    - ネットワーク機器に実装されているBufferbloat対策の実装については不明(調査不足)

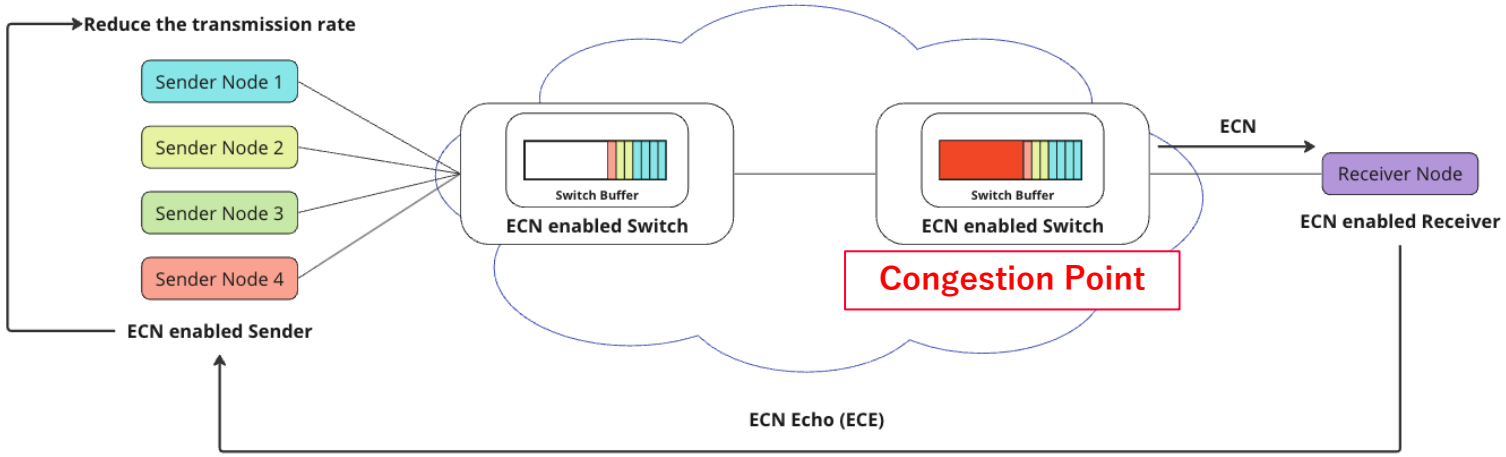
Network Assisted

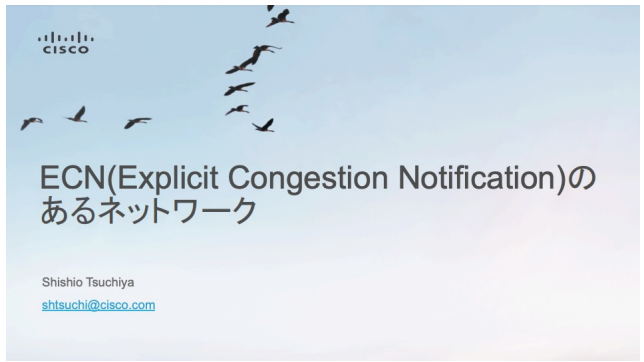
Shallow Buffer Configurations

ECN

Explicit Congestion Notification

- バッファの閾値を超えたパケットのECNビットにマーキングし受信側に輻輳を通知する仕組み
- 通知された受信側はECN Echo(ECE)を返し、それを受信した送信側は送信レートの調整を行う
- 送信側と受信側のホストで事前にECN Capabilityのネゴシエーションが必要
- Lossless Ethernet(RoCEv2)の場合は、TCP ECE相当の役割をCNP(Infiniband BTH)が担う



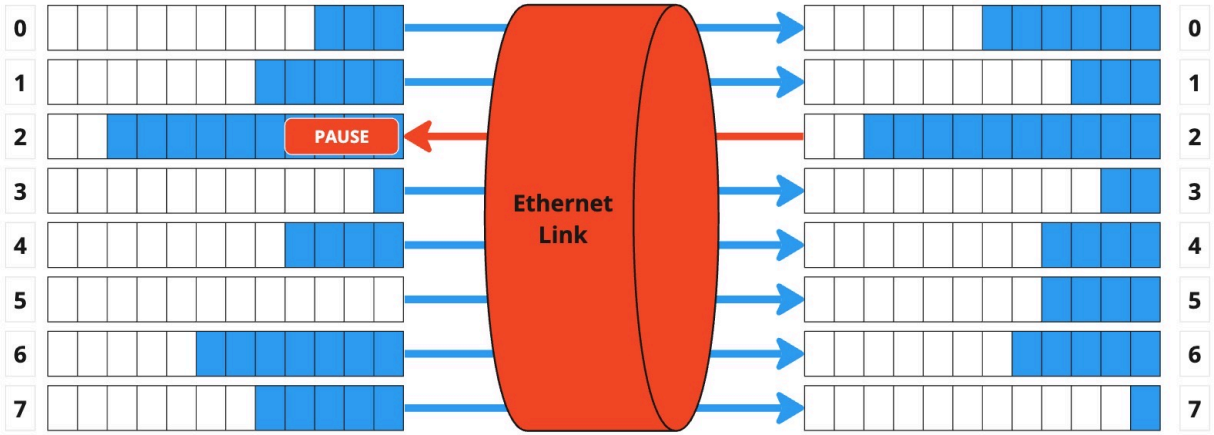


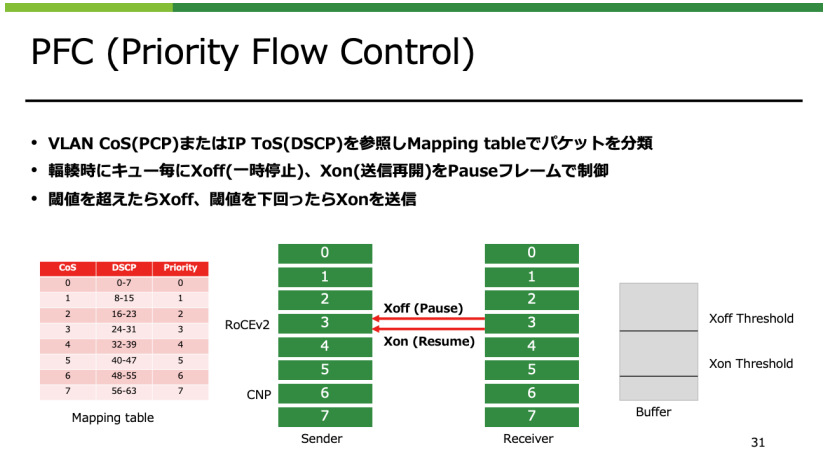
https://www.janog.gr.jp/meeting/lt-night-1/download_file/view/16/1

PFC

Priority-based Flow Control

- 輻輳が発生したTraffic Class(0~7)のキューからPAUSEフレームを隣接ノードに送信
- PAUSEフレームを受信したノードはそのTCに対応するキューからのパケットの送信を一時停止
- 特にLossless EthernetでECNと併用されることが一般的





https://www.janog.gr.jp/meeting/janog52/wp-content/uploads/2023/06/janog52-aiml400-uchida-koshoji.pdf

ECNとPFCは両方必要なのか？

輻輳通知の使い分け

- RDMAなどのLossless Ethernetであれば両方設定するのがベストプラクティス
    - DCQCN (ECN + PFC) + ETS(送信キューのQoS)
- **PFCはパケットの送信を止めてしまうため、最後の手段**と理解している
    - ジョブ完了時間が伸びてしまうと予想される
- 可能な限りECNでカバーして、それでも輻輳しているときにPFCが働くように設計したい
- HadoopなどTCPでLossyなネットワークでECNを使う場合、PFCも設定すべきなのか？
    - Losslessを要求されない環境なら不要と考えている
    - 議論したいポイントのひとつ

TCP Congestion Control

End-to-Endの輻輳制御アルゴリズム

TCP輻輳制御アルゴリズム

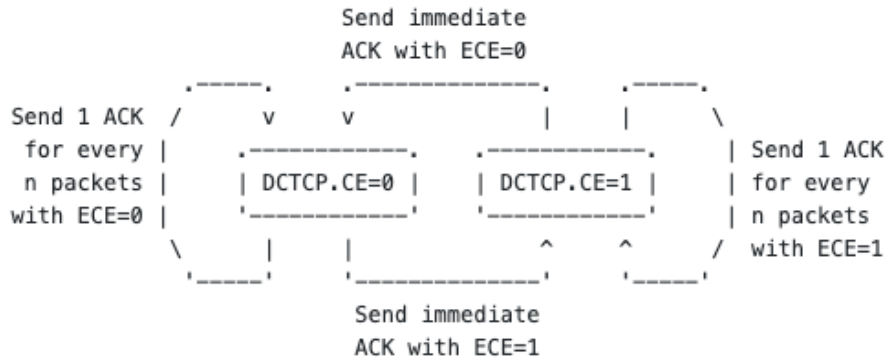
Delay-based vs Latency-based

- 動的に輻輳ウィンドウサイズ(cwnd)を調整し高いスループットを維持するための仕組み
- TCP輻輳制御アルゴリズムはフィードバック形式の違いで分類される
    - Loss-based (パケットロス)
    - Delay-based (遅延)
    - Hybrid
- 輻輳制御(ウィンドウサイズ調整)のトリガーに何を用いるかの違い
- 基本的にインターネット環境での利用を想定しているものがほとんど
- データセンター用にもっとアグレッシブな制御をしたいという需要がある
    - DCNWはend-to-endで管理された環境のため、専用の輻輳制御アルゴリズムを使いたい

DCTCP

Data Center TCP

- データセンターネットワーク向けのTCP輻輳制御アルゴリズム
  - RFC8257: https://datatracker.ietf.org/doc/html/rfc8257
- ECNに依存する技術で、ネットワークの実際の輻輳レベルに応じて緩やかに送信レート減少させる
  - 従来のECNは単純な輻輳検知しかできないが、輻輳しているデータ量を推定し制御するアルゴリズム
- Shallow Bufferのスイッチでも高いスループットを出せることを目的にしている
- 既存のTCPと同じネットワークに共存できないという課題もある



データセンターネットワーク輻輳対策手法

まとめ

方式	特徴	考慮事項
Deep Buffer	• バーストトラフィックに強い • ホスト側の設定を変更できない場合に有用	• Bufferbloatの対策が必要 • Shallow Bufferの機器より高価
Shallow Buffer – Host Only	• 従来のNW機器で運用可能 • ホスト側の設定を変更できない場合に有用	• バーストトラフィックに弱い • ホスト側の輻輳制御方式に依存
Shallow Buffer – NW Assisted	• 適切な設定で機器コストを抑えて構築可能 • 設定次第である程度の公平性を担保可能	• ホスト側の設定変更が必要 • NW側はバッファの閾値調整が必要 • ホスト側の輻輳制御が重要

結局どれを選ぶべきなのか？

↓

検証で確認

Hadoopを用いた検証

Hadoopの概要とIn-castを発生させる通信

方針

やること

1. まず既存の商用環境でどの程度の輻輳が起きているか把握する
    - Deep Bufferを採用しているHadoop環境(Lossy)を対象にバッファの使用状況を調査・可視化
    - Hadoopの分散処理の特性上、TCP In-castが最も発生していると考えられる環境のため
2. Deep Bufferによる恩恵がどの程度なのか、Shallow Bufferの機器と同一条件で比較
    - 機器をShallow Bufferのモデルに入れ替えて同じジョブを実行し比較
    - ジョブの完了時間に差があるのか、通信影響が出るのかを実際のワークロードを使って検証
3. Shallow Bufferの機器でもDeep Bufferと同等のアプリケーション性能を実現できるか検証
    - 輻輳制御の設定をスイッチとホストの両方で行うことでジョブ完了時間を比較

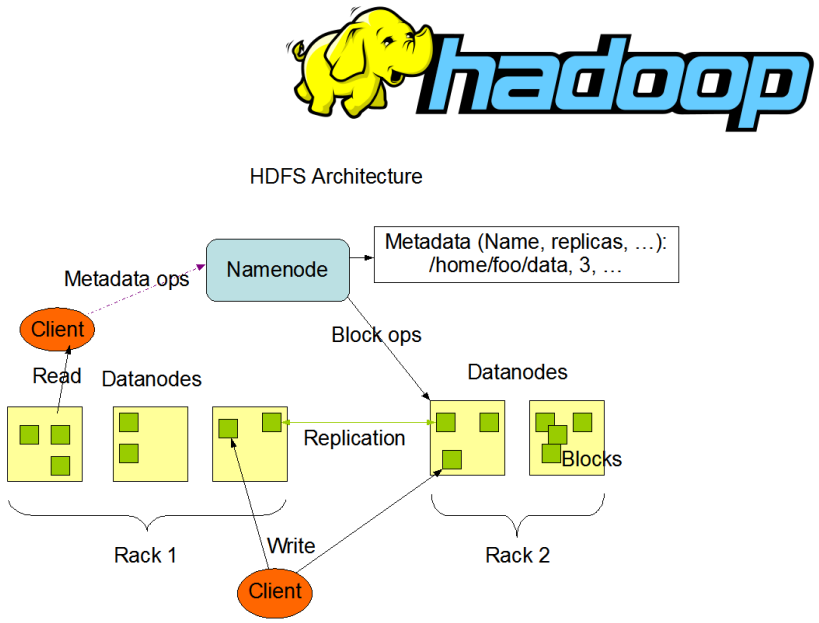
方針

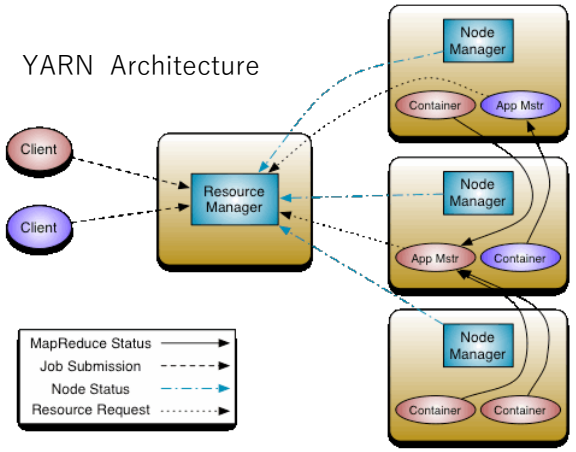
やらないこと

- Vendor Specificな製品や機能の比較
  - 可能な限り標準的な技術の組み合わせで実現したいため
- Losslessが要求されるネットワークの輻輳制御については今回の検証スコープからは除外
  - 商用と同程度の検証環境が現時点で用意できないため (GPUやNICの納期・コストの観点)
  - 要求特性が大きく異なることから専用NWで構築している
  - 専用NWで構築しているのでfairnessの観点はそこまで考慮しないため
    （他のアプリケーションのフローと混在しない）

Hadoop Overview

Architecture





- オープンソースの大規模な分散処理ミドルウェア
- 主に2つのコンポーネントで構成されている
  - HDFS
    - 分散ファイルシステム
  - YARN
    - 分散リソース制御システム
- その他にも多様なHadoopエコシステムが存在
- 参考
  - Apache Hadoop
  - 分散処理技術「Hadoop」とは

（参考）Hadoop の利用実績

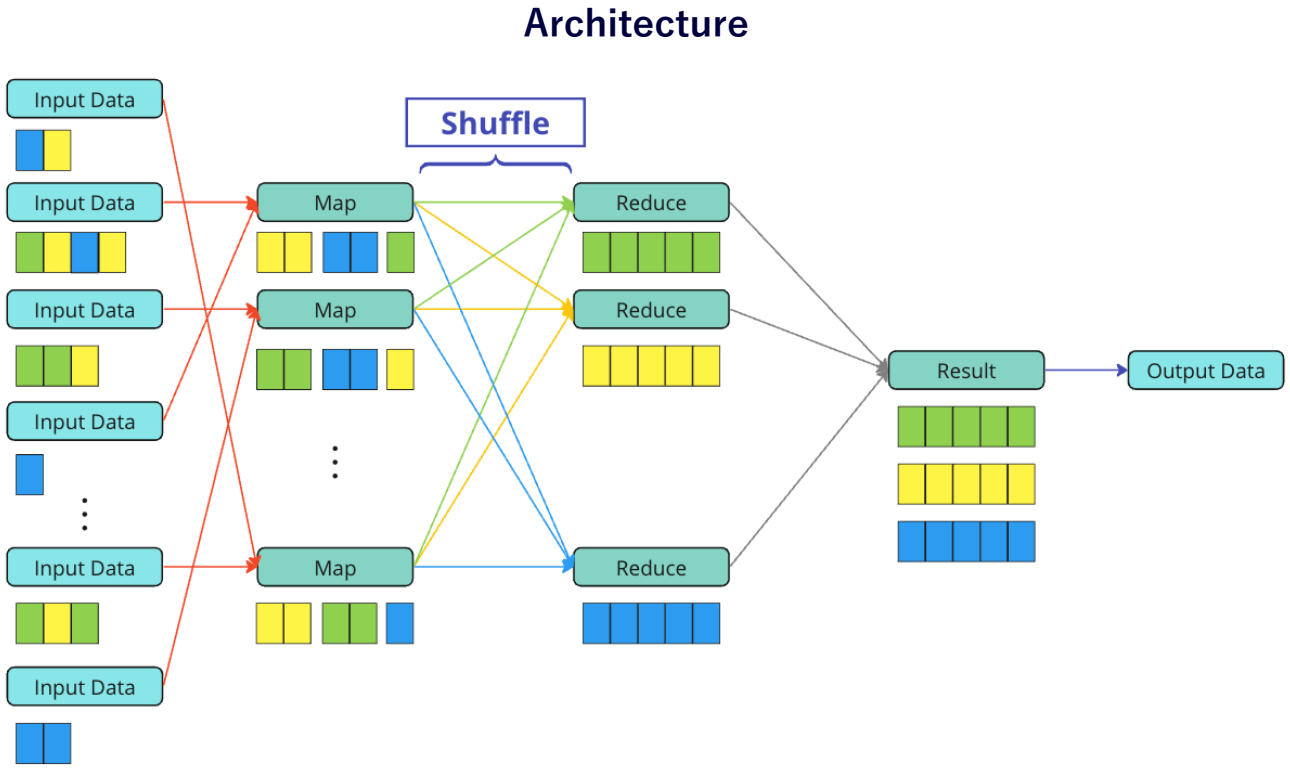
- 10年以上前から利用し、最も大きいHadoopクラスタでは120PB以上のデータを保管
- OSSコミニティのコミッター(PMC)も在籍しており、日々クラスタの性能向上へ取り組んでいる
  - Apache Hadoop - Yahoo!デベロッパーネットワーク
  - HDFSをメジャーバージョンアップして新機能のRouter-based Federationを本番導入してみた - Yahoo! JAPAN Tech Blog
  - 数千rpsを処理する大規模システムの配信ログをHadoopで分析できるようにする 〜 ショッピングのレコメンドシステム改修 - Yahoo! JAPAN Tech Blog

商用環境のHadoopでのQueueの状況

```
show queue-monitor length

Report generated at 2022-10-05 09:30:22
S-Start, U-Update, E-End, P-Polling, TC-Traffic Class
* Max queue length during period of congestion
Type       Time                   Intf(TC)          Queue          Duration    Ingress
                                                    Length                     Port-set
                                                    (bytes)        (usecs)
---------- ---------------------- ----------- --------------- ------------ ------------------------
E          0:00:27.74060 ago      Et20(1)          5750000*       2010188    Et5-8,13-16,21-24,50/1,51/1
...
E          0:12:59.96617 ago      Et5(1)           36128608*      2012219    Et29-32,41-44,52/1,54/1
...
E          0:23:38.61004 ago      Et28(1)          97254976*      5029613    Et29-32,41-44,52/1,54/1
...
S          0:48:26.13189 ago      Et7(1)           21290960       N/A        Et29-32,41-44,52/1,54/1
S          0:48:26.13498 ago      Et7(1)           35922528       N/A        Et1-4,9-12,17-20,49/1
E          0:50:20.79371 ago      Et17(1)          5417104*       1005168    Et29-32,41-44,52/1,54/1
S          0:50:21.79888 ago      Et17(1)          5417104        N/A        Et29-32,41-44,52/1,54/1
E          0:50:38.90245 ago      Et20(1)          5698704*       1005569    Et29-32,41-44,52/1,54/1
S          0:50:39.90802 ago      Et20(1)          5698704        N/A        Et29-32,41-44,52/1,54/1
E          0:52:14.46406 ago      Et30(1)          17724384*      3018536    Et5-8,13-16,21-24,50/1,51/1
E          0:52:15.46825 ago      Et30(1)          19959616*      1004884    Et29-32,41-44,52/1,54/1
E          0:52:15.47088 ago      Et30(1)          23763408*      1004764    Et1-4,9-12,17-20,49/1
S          0:52:16.47313 ago      Et30(1)          19959616       N/A        Et29-32,41-44,52/1,54/1
U          0:52:16.47556 ago      Et30(1)          11163744       N/A        Et5-8,13-16,21-24,50/1,51/1
S          0:52:16.47564 ago      Et30(1)          23763408       N/A        Et1-4,9-12,17-20,49/1
S          0:52:17.48260 ago      Et30(1)          17724384       N/A        Et5-8,13-16,21-24,50/1,51/1
E          0:52:19.49307 ago      Et3(1)           11787056*      1005883    Et5-8,13-16,21-24,50/1,51/1
S          0:52:20.49896 ago      Et3(1)           11787056       N/A        Et5-8,13-16,21-24,50/1,51/1
E          0:52:34.57773 ago      Et21(1)          17678288*      3016978    Et29-32,41-44,52/1,54/1
E          0:52:34.58029 ago      Et21(1)          18985584*      2012206    Et1-4,9-12,17-20,49/1
E          0:52:35.58547 ago      Et21(1)          10737008*      2012232    Et5-8,13-16,21-24,50/1,51/1
U          0:52:36.58931 ago      Et21(1)          17678288       N/A        Et29-32,41-44,52/1,54/1
S          0:52:36.59249 ago      Et21(1)          18985584       N/A        Et1-4,9-12,17-20,49/1
S          0:52:37.59471 ago      Et21(1)          8530064        N/A        Et29-32,41-44,52/1,54/1
S          0:52:37.59770 ago      Et21(1)          10737008       N/A        Et5-8,13-16,21-24,50/1,51/1
E          0:53:31.91133 ago      Et20(1)          38318176*      2011937    Et29-32,41-44,52/1,54/1
E          0:53:31.91389 ago      Et20(1)          9922048*       2011929    Et1-4,9-12,17-20,49/1
E          0:53:32.92011 ago      Et20(1)          22974688*      2011463    Et5-8,13-16,21-24,50/1,51/1
...
```

定常的にサーバポートで20MBを超えるBufferが使われており、100MBを超えることもある

MapReduce Overview

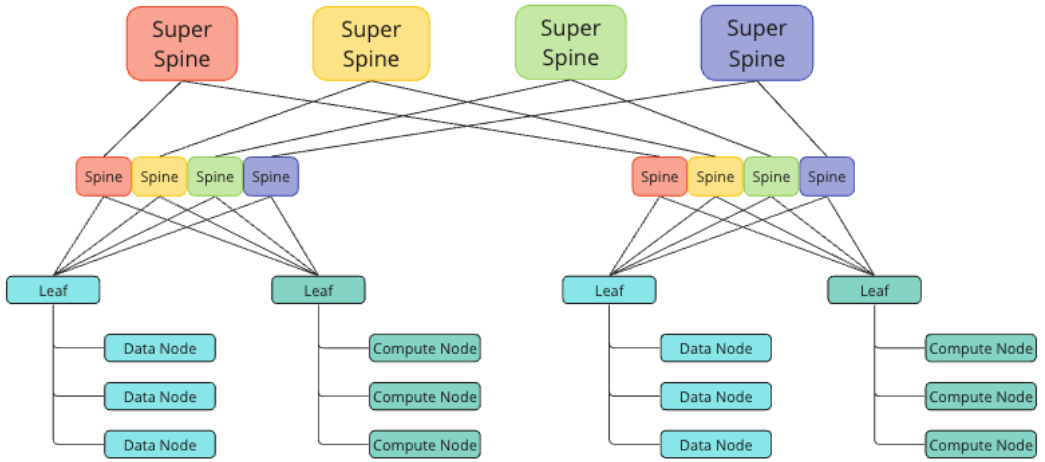


- **Map**：入力データに対して、各Compute Nodeで処理を行う
- **Shuffle**：Map Phaseで生成したデータを特定レコードでまとめるためにデータ転送を行う
- **Reduce**：Shuffle Phaseで受け取ったデータを集約する

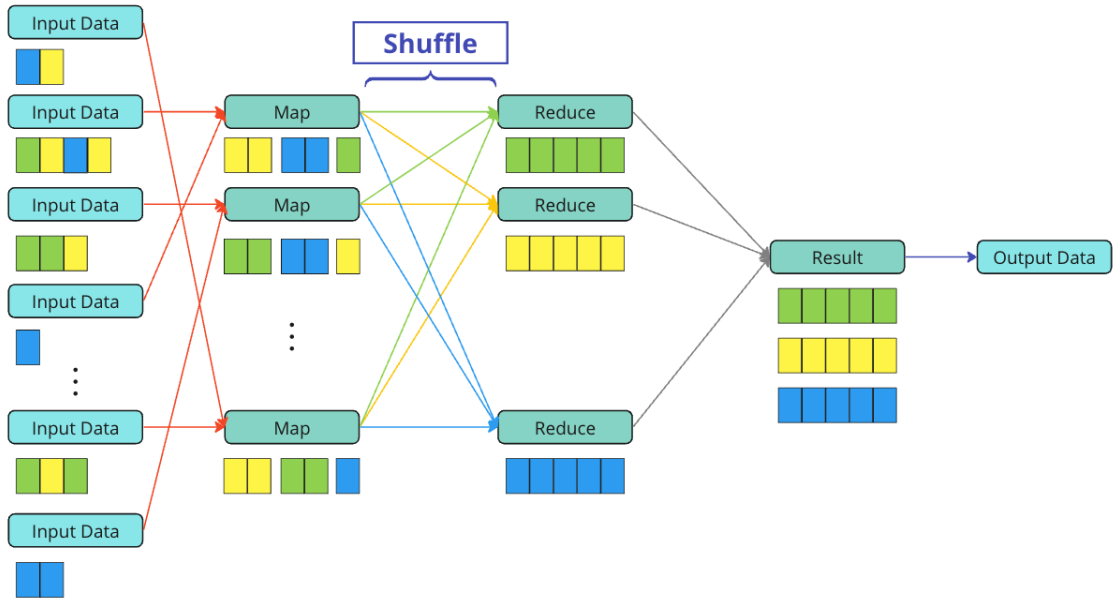
Hadoop Incast-congestion

Hadoop でどのようなときに多対一の通信が発生するのか

Network Design



MapReduce



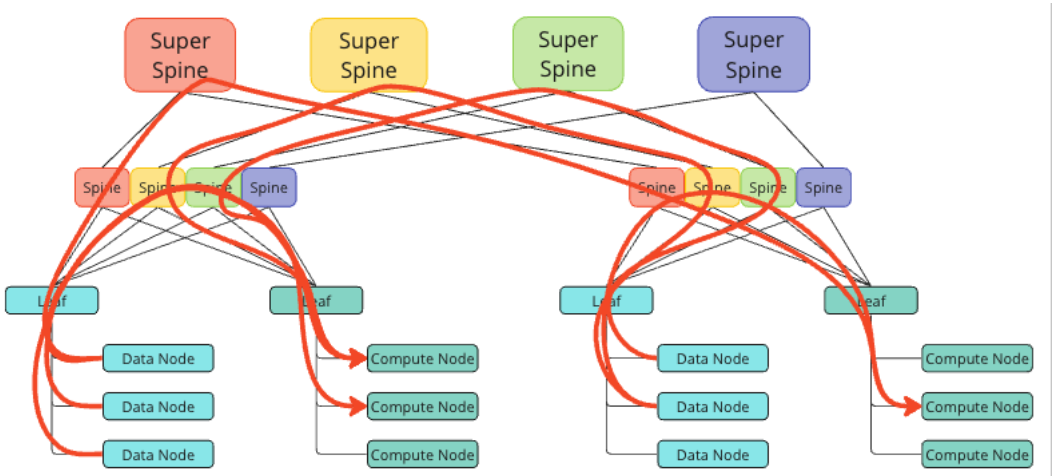
Data Node: HDFSのデータを保存しているノード

Compute Node : データ処理を行うノード

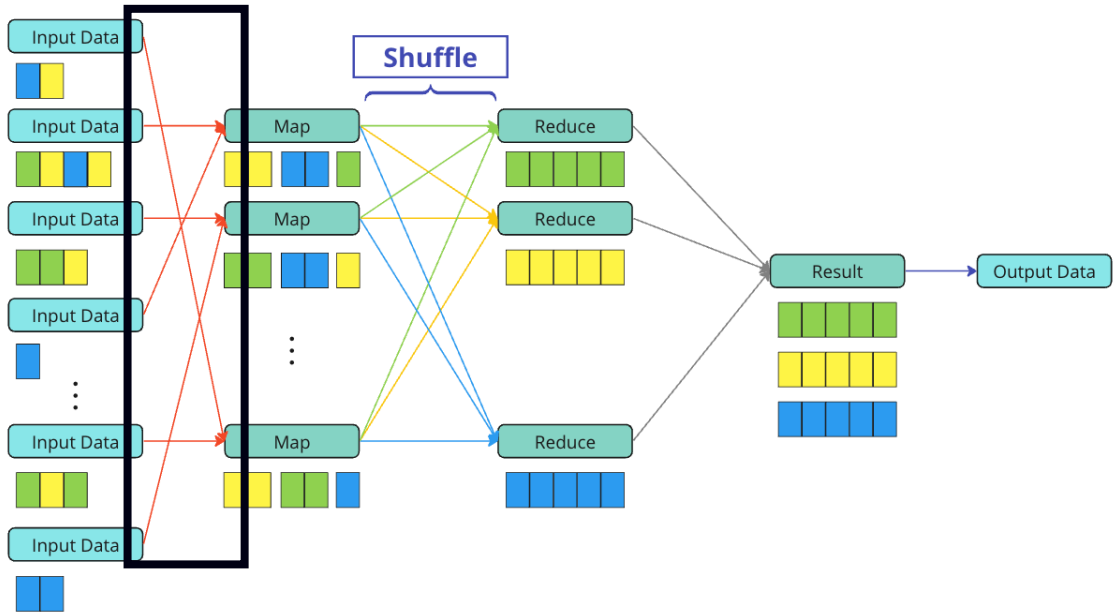
Hadoop Incast-congestion

Hadoop でどのようなときに多対一の通信が発生するのか

Network Design



MapReduce

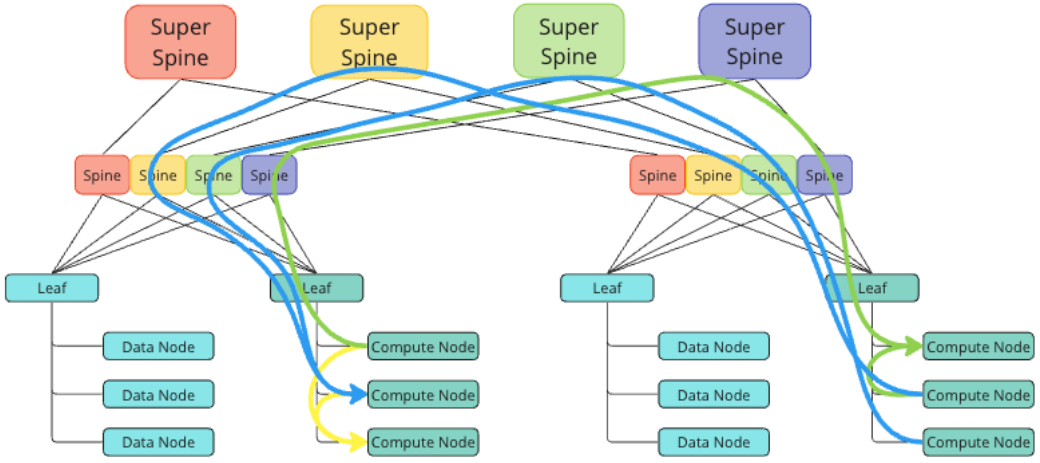


Map Phase: Data NodeからCompute Nodeへのデータ転送

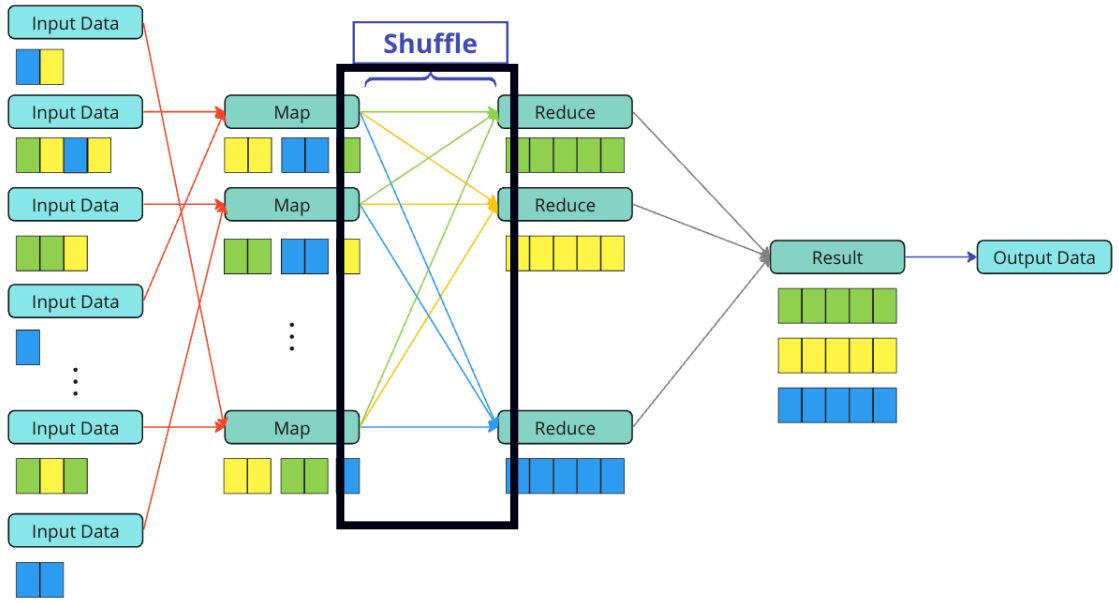
Hadoop Incast-congestion

Hadoop でどのようなときに多対一の通信が発生するのか

Network Design



MapReduce

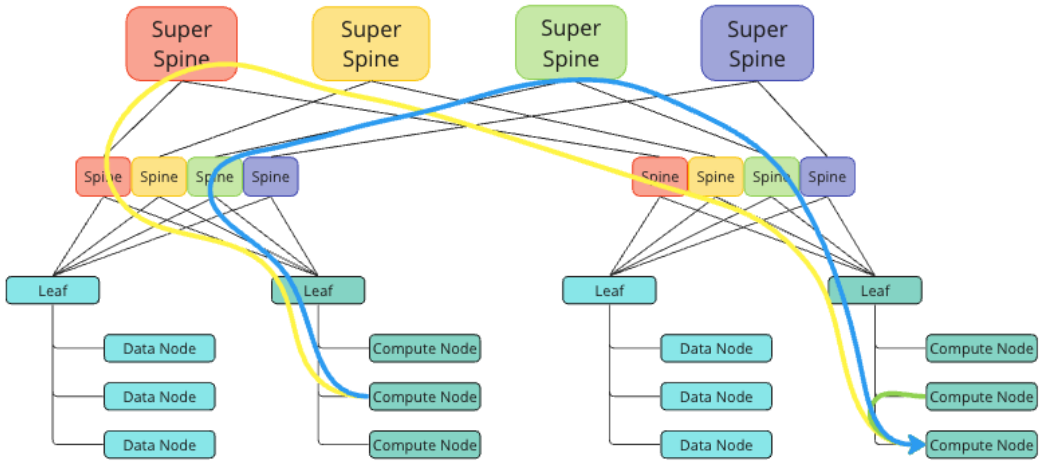


Shuffle Phase: Map処理したデータがReduce処理のためにCompute Node間で転送される

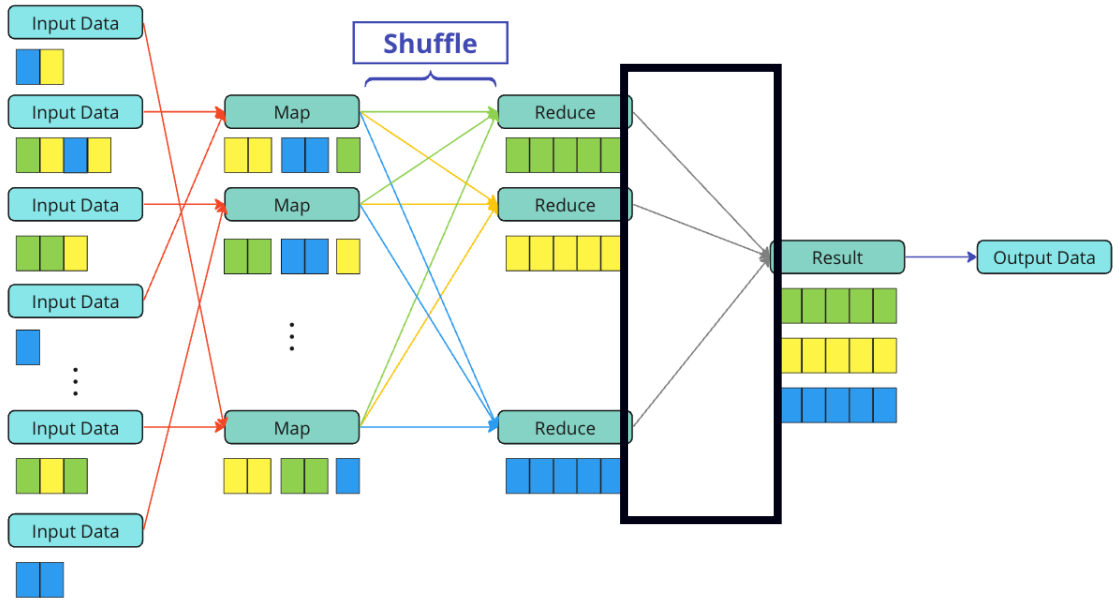
Hadoop Incast-congestion

Hadoop でどのようなときに多対一の通信が発生するのか

Network Design



MapReduce



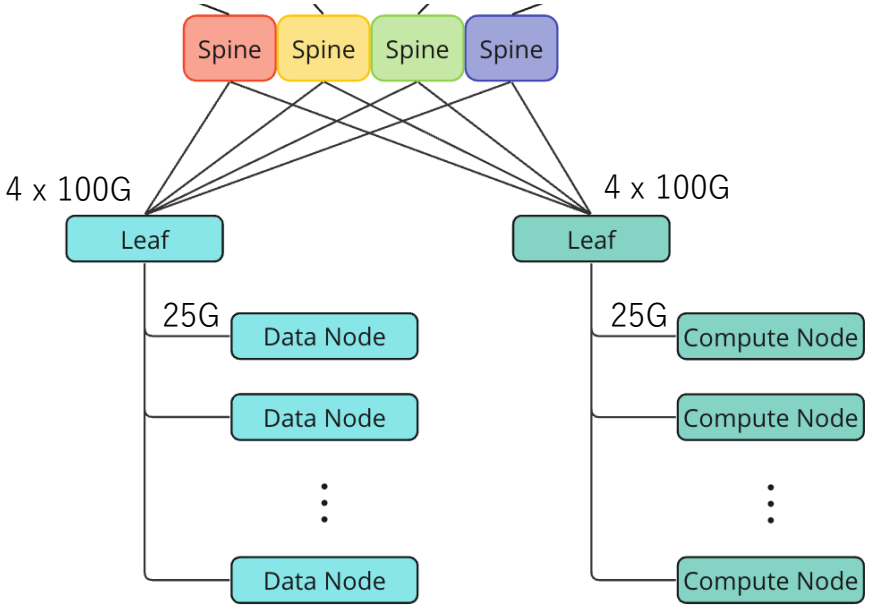
Reduce Phase: 各Compute Nodeが結果を集約するCompute Nodeへデータ転送を行う

検証構成

実行するジョブと検証パターン

検証環境

構成



下記の条件でジョブ完了時間がどうなるか検証

- LeafをBuffer仕様の異なるモデルに変更
- ECN設定の有無

DiscardsやQueueの可視化にはVender製品を利用

- Hadoop検証クラスタの一部を利用
  - 商用環境と同じハードウェアと構成
- Data Node
  - 7台
- Compute Node
  - 14台
- Data Node / Compute Nodeは
  それぞれ専用のLeaf switchに接続
- 検証で利用したスイッチのBuffer仕様
  - Deep Buffer 8GB (Jericho+)
  - Shallow Buffer 32MB (Trident 3)

検証環境

実行するHadoopジョブと環境

- 種類
  - TeraSort（Hadoopに標準搭載されているベンチマーク）
- 処理フレームワーク
  - MapReduce
  - Tez (MapReduceに比べてIn-castの通信が少ない)
- データサイズ
  - 10G (低負荷) / 200G (高負荷)
- 実行環境
  - トラフィック負荷なし
    - キューの競合のない状態で完了時間がどの程度か確認するためジョブのみ実行
  - トラフィック負荷あり - Compute Node間でトラフィック(10~25Gbps)を流しながら実行
    - 実際の環境で複数のジョブが実行されている状況を想定し計測

参考：Apache Tezの解説 | Hadoop Advent Calendar 2016 #07

検証パターン

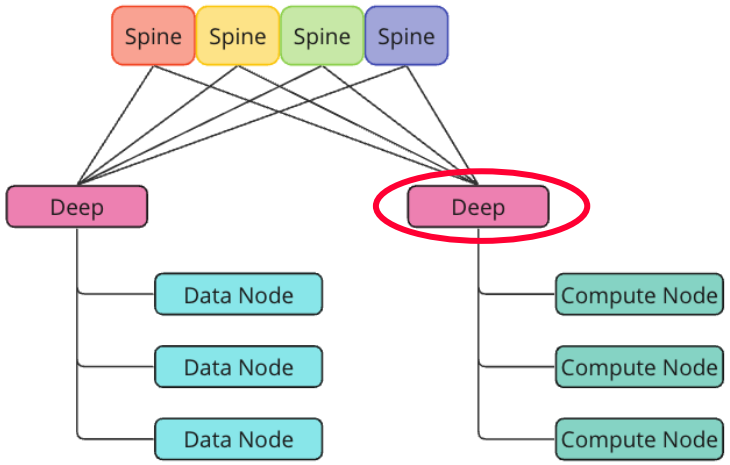
各パターンとその目的

1. Deep Buffer と Shallow Buffer の単純比較 (ECN設定なし)
    - Network Assistが無いHost Onlyな輻輳制御でDeep BufferとShallow Bufferのパフォーマンス検証
2. ECN 設定の有無の比較
    - Network AssistとしてShallow BufferでECNを設定した場合の輻輳制御の効果を検証
3. ECN+PFC と PFCのみ の比較
    - Network AssistとしてShallow BufferでECNに加えてPFCの設定の必要性を検証
4. サーバの輻輳制御アルゴリズムの違いの比較
    - サーバ側の輻輳制御アルゴリズムをNetwork Assist(ECN)と密接に連携させた場合の効果を検証

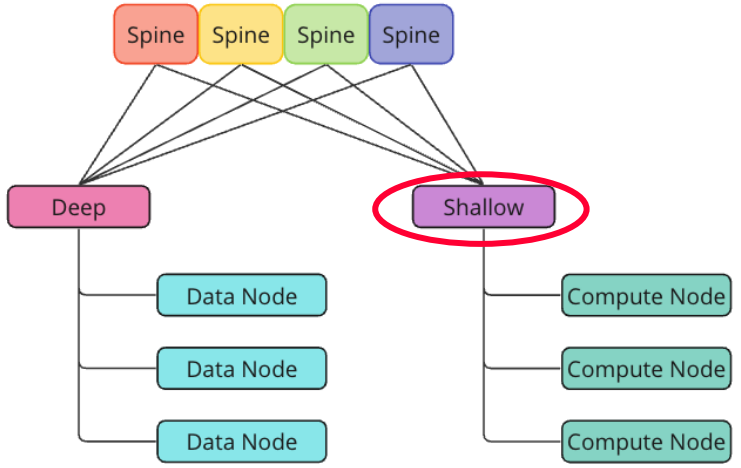
検証パターン1

Deep Buffer と Shallow Buffer の比較（ECN設定なし）

Data Node Leaf / Deep Buffer
Compute Node Leaf / Deep Buffer
ECN 設定なし



Data Node Leaf / Deep Buffer
Compute Node Leaf / Shallow Buffer
ECN 設定なし



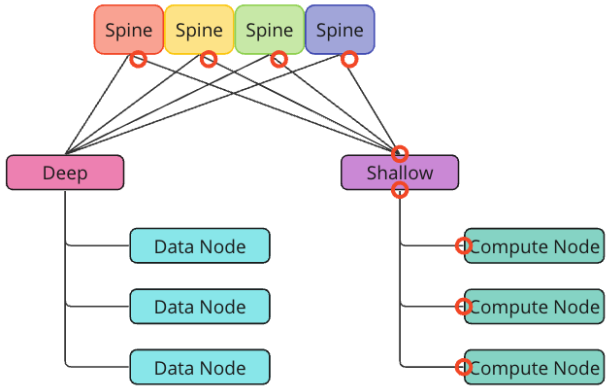
- Compute NodeのLeafをDeep Bufferの機種とShallow Bufferの機種で比較
- ECN などの輻輳制御の設定はなし

※ Data NodeのLeafをShallow Bufferの機種にしての検証は検証機器と検証環境の都合から未実施

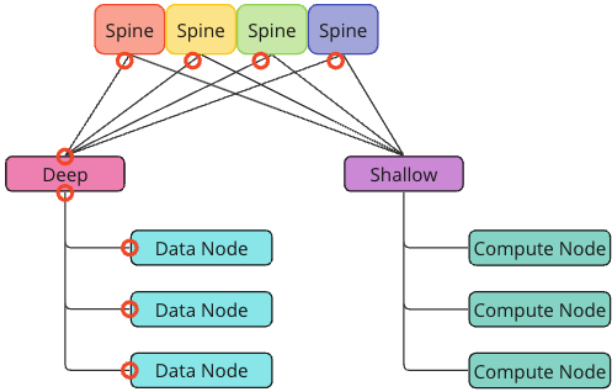
検証パターン2

ECN 設定の有無の比較

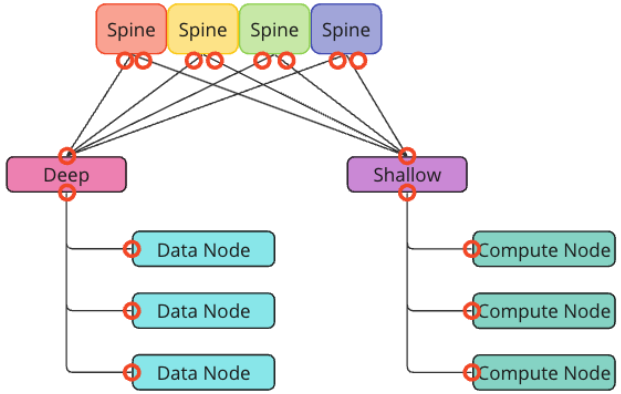
Data Node Leaf / Deep Buffer
Compute Node Leaf / Shallow Buffer
Compute Node ECNあり



Data Node Leaf / Deep Buffer
Compute Node Leaf / Shallow Buffer
Data Node ECNあり



Data Node Leaf / Deep Buffer
Compute Node Leaf / Shallow Buffer
Data Node, Compute Node ECNあり



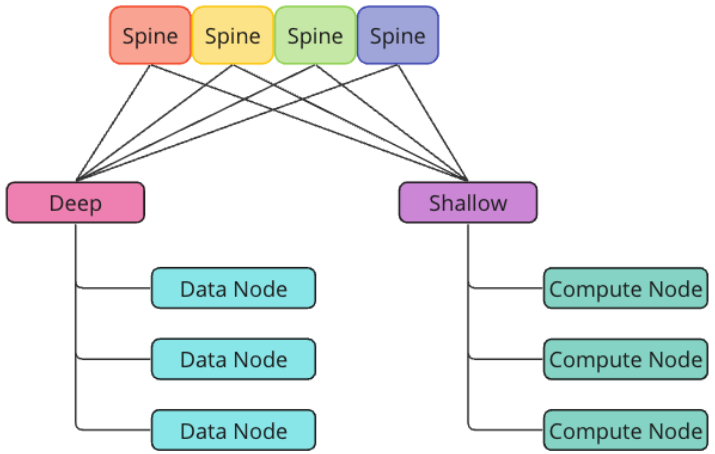
ECNを、

- Compute Nodeで有効化 → Shuffle, Reduceの通信にECNが効果を発揮する想定
- Data Nodeで有効化 → Mapの通信でのData NodeとCompute Nodeのデータ転送で効果を発揮する想定
- Compute Node とData Node の両方で有効化 → MapReduce の全ての通信で効果を発揮する想定

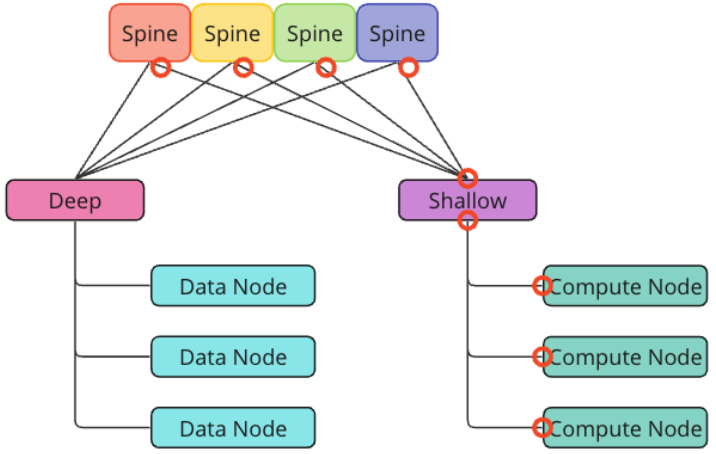
検証パターン2

ECN 設定の有無の比較

Data Node Leaf / Deep Buffer
Compute Node Leaf / Shallow Buffer
ECN 設定なし



Data Node Leaf / Deep Buffer
Compute Node Leaf / Shallow Buffer
Compute Node ECNあり



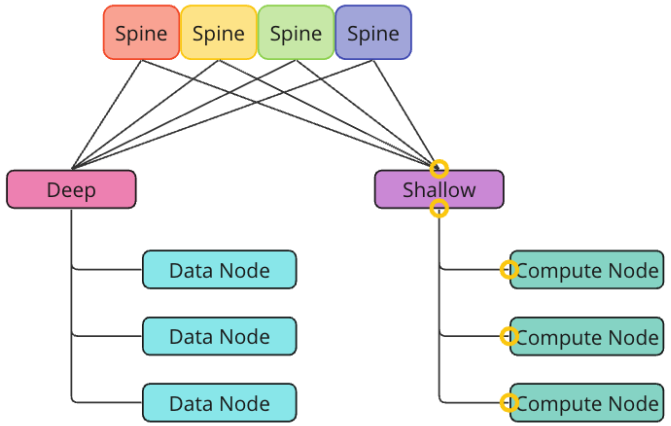
ECNを

- Compute Nodeで有効化 → Shuffle, Reduceの通信にECNが効果を発揮する想定
- Data Nodeで有効化 → Mapの通信でのData NodeとCompute Nodeのデータ転送で効果を発揮する想定
- Compute Node とData Node の両方で有効化 → MapReduce の全ての通信で効果を発揮する想定

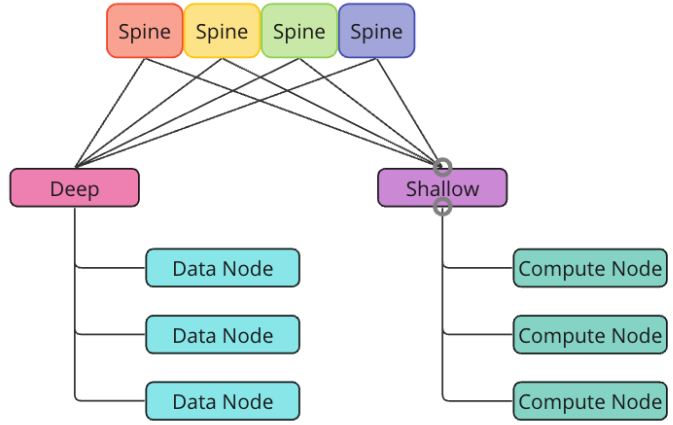
検証パターン3

ECN+PFC と PFCのみの比較

Data Node Leaf / Deep Buffer
Compute Node Leaf / Shallow Buffer
Compute Node ECN+PFC



Data Node Leaf / Deep Buffer
Compute Node Leaf / Shallow Buffer
Compute Node PFC のみ

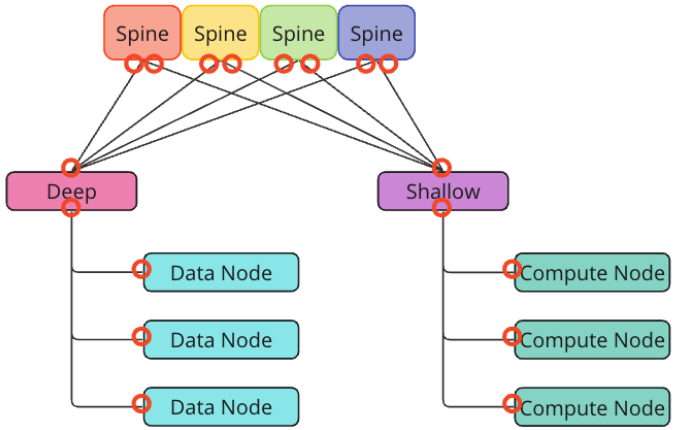


- Compute NodeでECN+PFCを有効にしたものとPFCのみを有効にしたものを比較
- ECNによってPFCのPAUSEフレームの送信を抑制できることの確認が目的

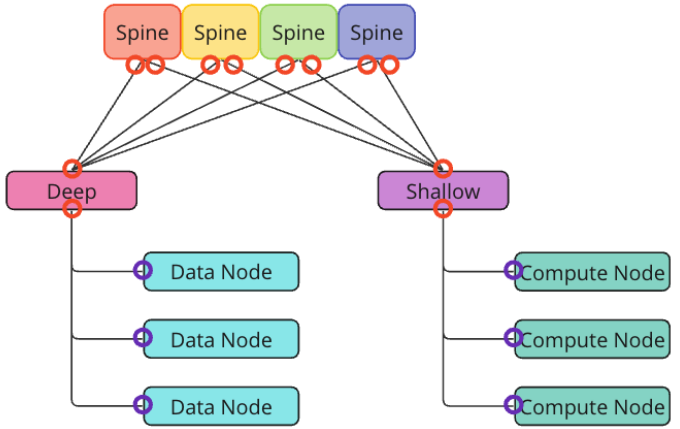
検証パターン4

サーバの輻輳制御アルゴリズムの違いでの比較

Data Node Leaf / Deep Buffer
Compute Node Leaf / Shallow Buffer
All ECN(H-TCP)



Data Node Leaf / Deep Buffer
Compute Node Leaf / Shallow Buffer
All ECN(DCTCP)



- サーバ側のECNの設定をDCTCPに変更し、H-TCPと比較
  - net.ipv4.tcp_congestion_control=dctcp
  - net.ipv4.tcp_ecn_fallback の設定はサーバのkernelバージョンが理由で設定できず
- Spine/Leaf Switch のECN設定はH-TCPの場合と同様

検証結果

Summary

期待する検証結果

Summary

	方式	Queueの利用状況	ジョブ完了時間
1	**Deep Buffer**	• 負荷が高い状態でもDiscardsやQueue Dropsなどを抑えることができている	• 短い
2	**Shallow Buffer (Host Only, Non-ECN)**	• 負荷が高い状態でDiscardsや Queue Dropsが頻繁に発生	• 長い
3	**Shallow Buffer (ECN Only)**	• 負荷が高い状態でもDiscardsやQueue Dropsなどを抑えることができる	• 1と同程度
4	**Shallow Buffer (ECN + PFC)**	• ECNを利用することでPFCカウンターの増加を抑えることができる	• 1と同程度
5	**Shallow Buffer (ECN + DCTCP)**	• 3よりもDiscardsやQueue Dropsなどを抑えることができる	• 3よりも短い

検証結果

Summary

	方式	Queueの利用状況	ジョブ完了時間
1	**Deep Buffer**	• 負荷が高い状態でもDiscardsや Queue Dropsなどを抑えることができた	~~• 短い~~ • **長い**
2	**Shallow Buffer (Host Only, Non-ECN)**	• 負荷が高い状態でDiscardsやQueue Dropsなどが頻繁に発生した	~~• 長い~~ • **短い**
3	**Shallow Buffer (ECN Only)**	• 負荷が高い状態でもDiscardsや Queue Dropsなどを抑えることができた	~~• 1と同程度~~ • **2と同程度**
4	**Shallow Buffer (ECN + PFC)**	• ECNを利用することでPFCカウンターの増加を抑えることができた • **PFCのみでもQueue Dropsを抑えることができた**	~~• 1と同程度~~ • **フレームワークによって差異あり**
5	**Shallow Buffer (ECN + DCTCP)**	• **3と比較して、Queue Dropsなどが発生**	~~• 3よりも短い~~ • **3よりも長い**

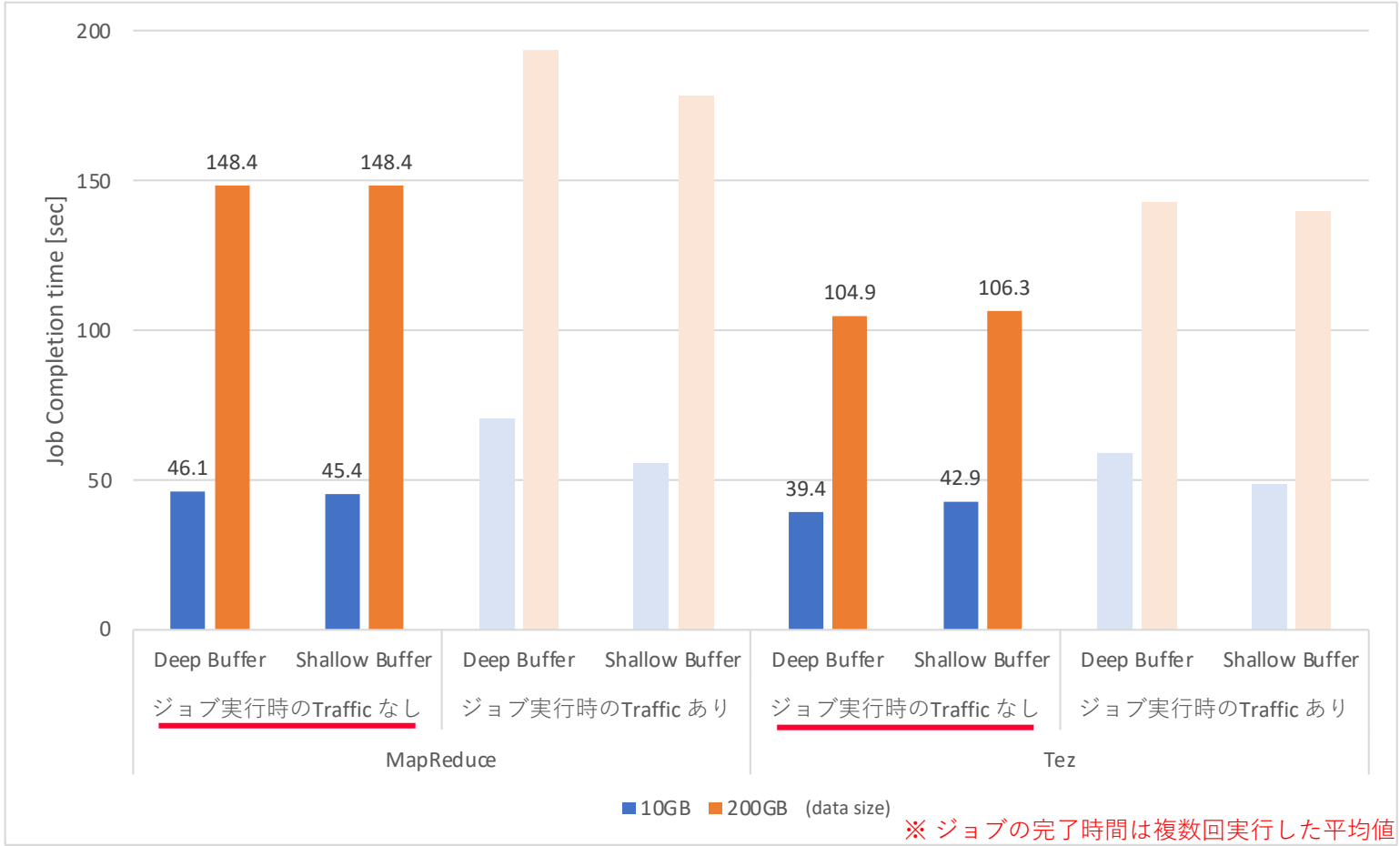
検証結果

パターン1

Deep Buffer と Shallow Buffer の比較

検証パターン1

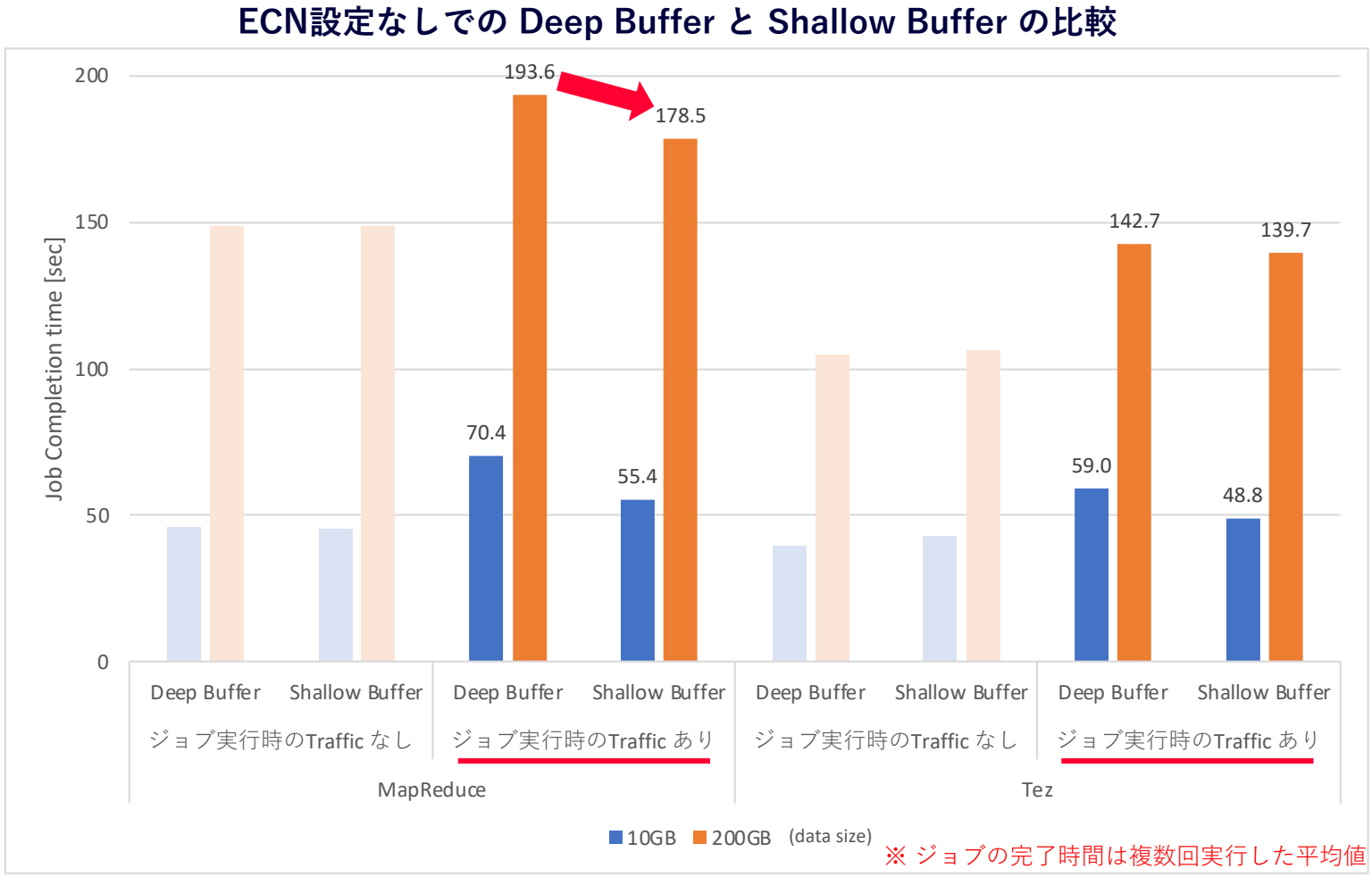
ECN設定なしでの Deep Buffer と Shallow Buffer の比較



※ ジョブの完了時間は複数回実行した平均値

トラフィックを流していない状態では、Deep / Shallowのジョブ完了時間に大きな差はない

検証パターン1



トラフィックを流している状態では、Shallowの方がジョブ完了時間が短い結果となった

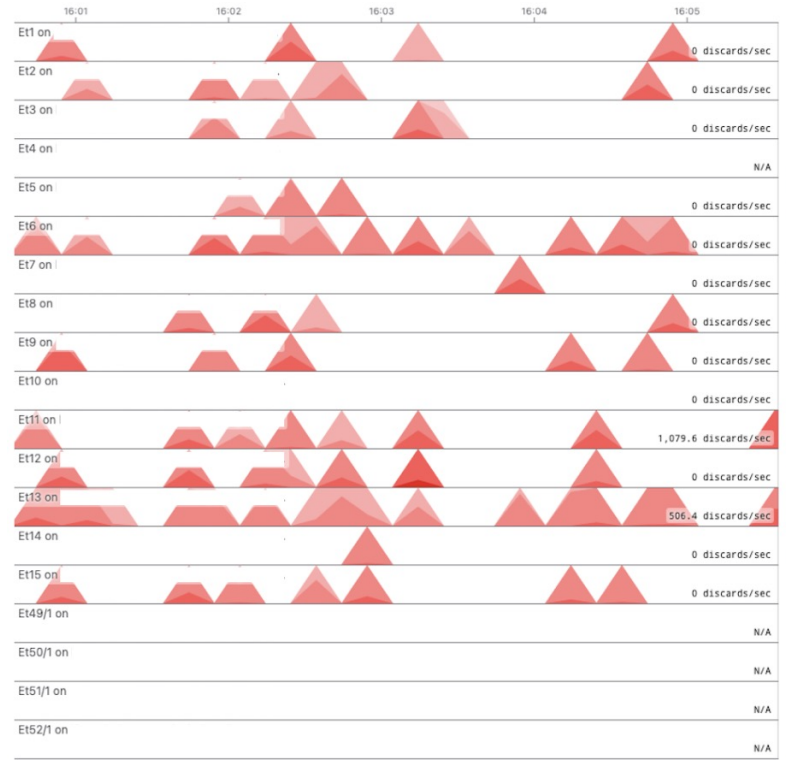
(特にMapReduceで差が出た)

検証パターン1

Discards の比較 (Trafficを流している状況でのジョブ実行時)

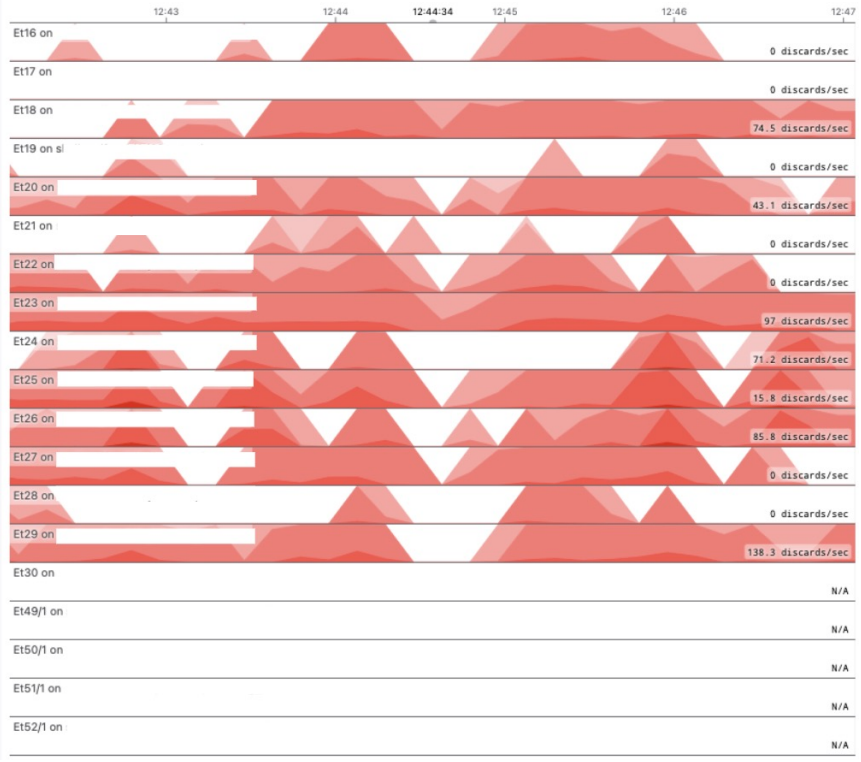
Deep Buffer

Out Discards



Shallow Buffer

Out Discards

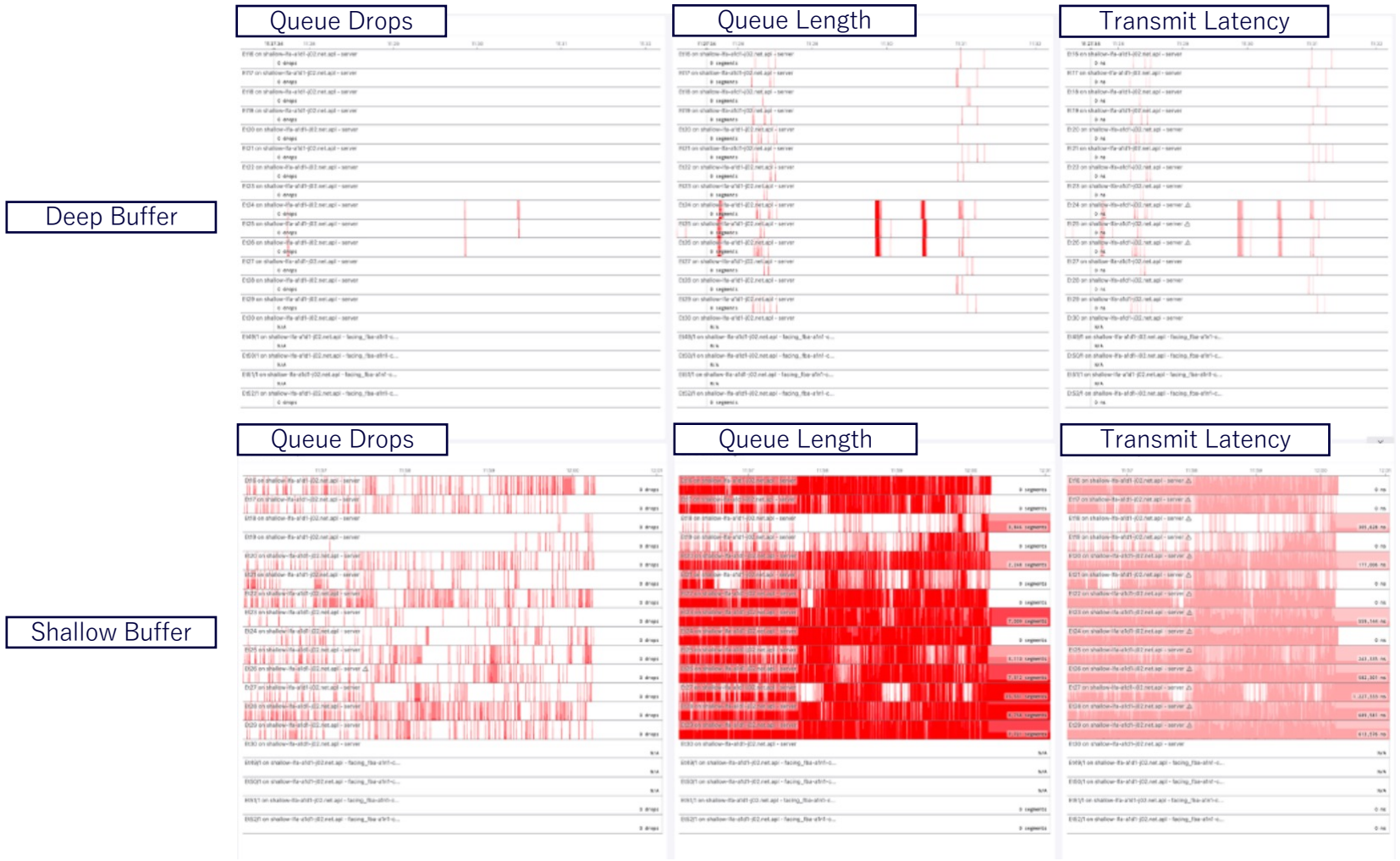


Deep Bufferでは特定のタイミングでDiscardsが発生している

Shallow Bufferでは特定のタイミングでなくても、発生してしまっている

検証パターン1

Queue の比較 (Trafficを流している状況でのジョブ実行時)



- **Queue Drops**
  発生したTail Drop
- **Queue Length**
  Queueの輻輳状況
- **Transmit Latency**
  送信遅延

ShallowではQueue Dropsなどがほぼ常に発生してしまっている

検証パターン1

結果と考察

- Deep Buffer
    - 負荷が高い状態でもDiscardsやQueue Dropsなどを抑えることができている
    - ジョブ完了時間がShallow Bufferより長い
- Shallow Buffer
    - 負荷が高い状態ではほぼ常にDiscardsやQueue Dropsなどが発生している
- Deep Bufferの方がジョブ完了時間が長い理由として、Bufferbloatが発生している可能性
    - Deep Bufferによってバーストが吸収された反面、キューイング遅延が発生している可能性
    - 今回の検証で断定はできず

検証結果

パターン2

Compute Node ECN 設定の有無の比較

検証パターン2

ECN 設定内容

- switch configuration

```
!
hardware counter feature ecn out
!
int ethX
  tx-queue 1
    random-detect ecn minimum-threshold 128 kbytes maximum-threshold 1280 kbytes max-mark-probability 100
    random-detect ecn count
!
```

閾値に関しては今回は決め打ち

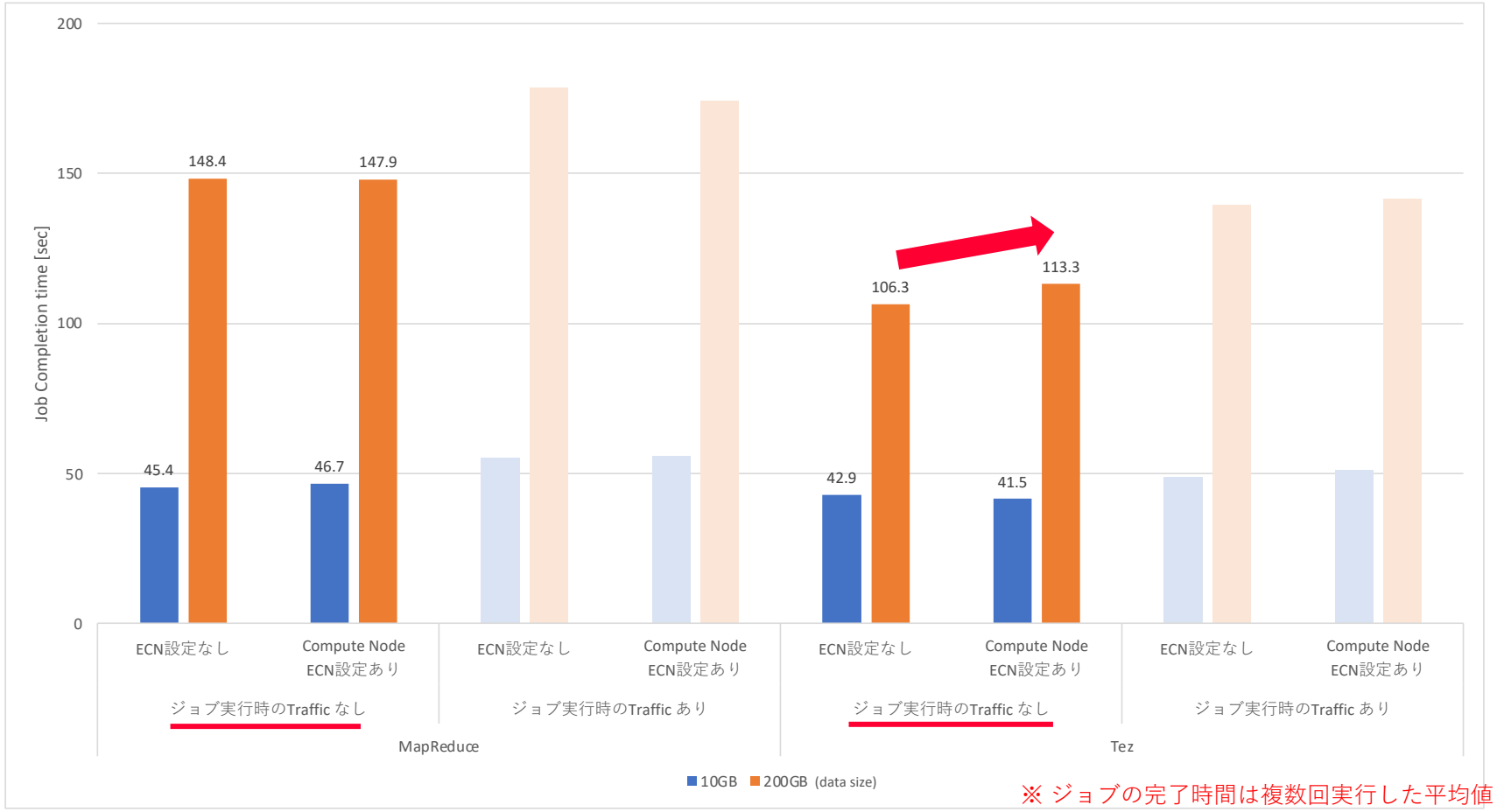
- server configuration

```
net.ipv4.tcp_congestion_control=htcp (Defaultはcubic)
net.ipv4.tcp_ecn = 1 (Defaultは2)
```

net.ipv4.tcp_ecn=1 は「有効」、2 は「ECNを通知してきた相手には有効にする」、0 は「無効」

検証パターン2

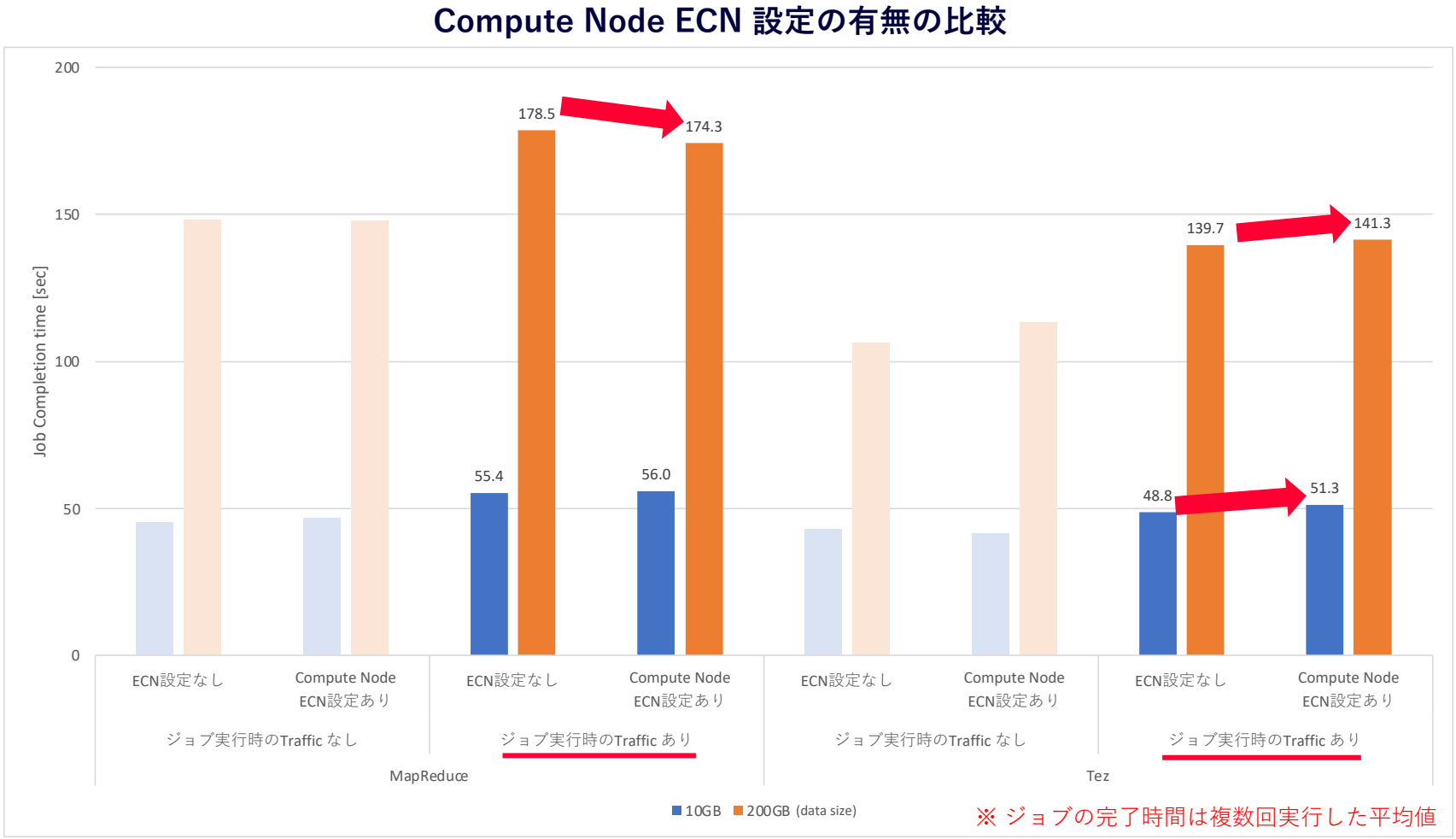
Compute Node ECN 設定の有無の比較



※ ジョブの完了時間は複数回実行した平均値

トラフィックを流していない状態での10GBではほぼ差がないが、Tez 200GBでは若干差があり、
フレームワークの違いで実行結果への影響が違う結果

検証パターン2



Traffic 流している状態では、MapReduce ではECN設定ありが短く、
Tez では少し長いという結果に

検証パターン2

Discards の比較 (Trafficを流している状況でのジョブ実行時)

ECN設定 **無**

Out Discards

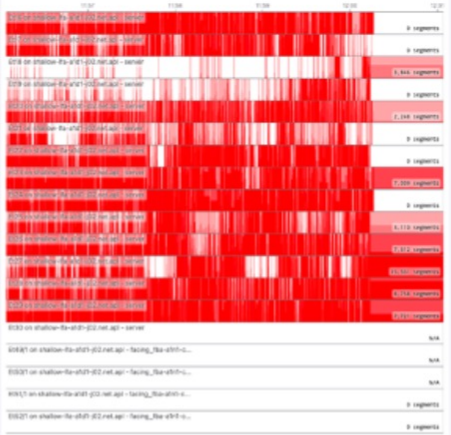
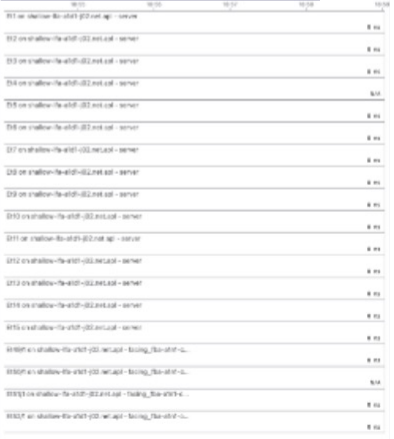
ECN設定 **有**

Out Discards

ECNの設定しない場合と比べてDiscardsの量が抑えることができている

検証パターン2

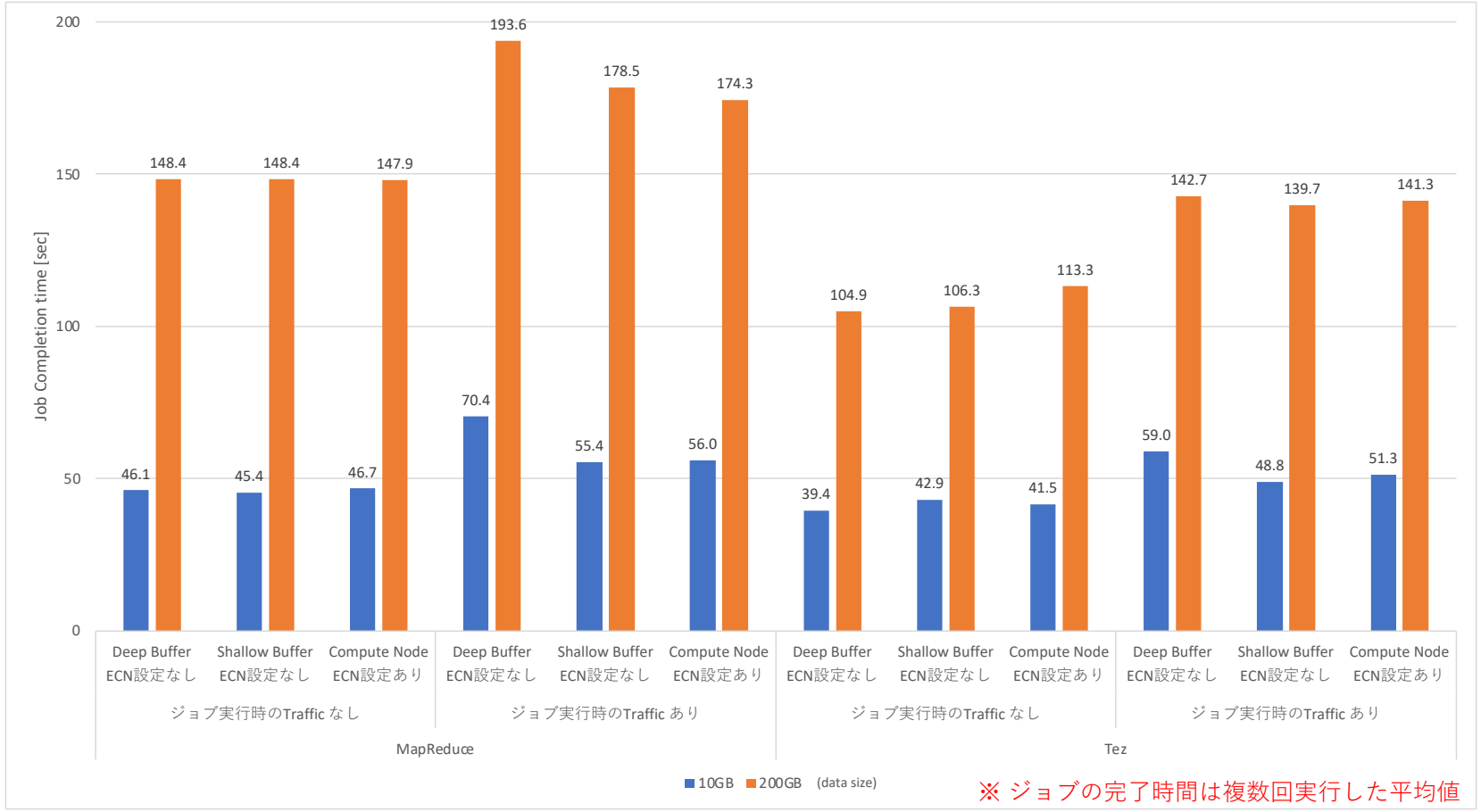
Queue の比較 (Trafficを流している状況でのジョブ実行時)

	Queue Drops	Queue Length	Transmit Latency
ECN設定 **無**			
ECN設定 **有**			

ECN設定無しの場合と比べて、明らかにQueue Dropsの発生やQueue Dropsや輻輳がなくなった

検証パターン2

Deep Buffer / Shallow Buffer / ECN 比較



Shallow BufferでECNを設定してもTraffic流している状態で

Deep Bufferよりジョブ完了時間が短い結果に

検証パターン2

結果と考察

- Shallow Buffer + ECN
    - トラフィック負荷が高い状態でもDiscardsやQueue Dropsなどが抑えることができている
    - ECNを設定してもDeep Buffer利用時よりジョブ完了時間が短い

ECNを設定することはDiscardsやQueue Dropsなどに対して有効と考えられる

- TezではECNを設定したときの、Traffic流した状態での実行時間が伸びているため、
  輻輳の制御によりジョブへ影響がでている可能性がある
    - 検証パターン1 で推測した「キューイング遅延が発生している可能性」の説明をすることができる一つの結果

検証結果

パターン3

ECN+PFC と PFCのみの比較

検証パターン3

PFC の設定

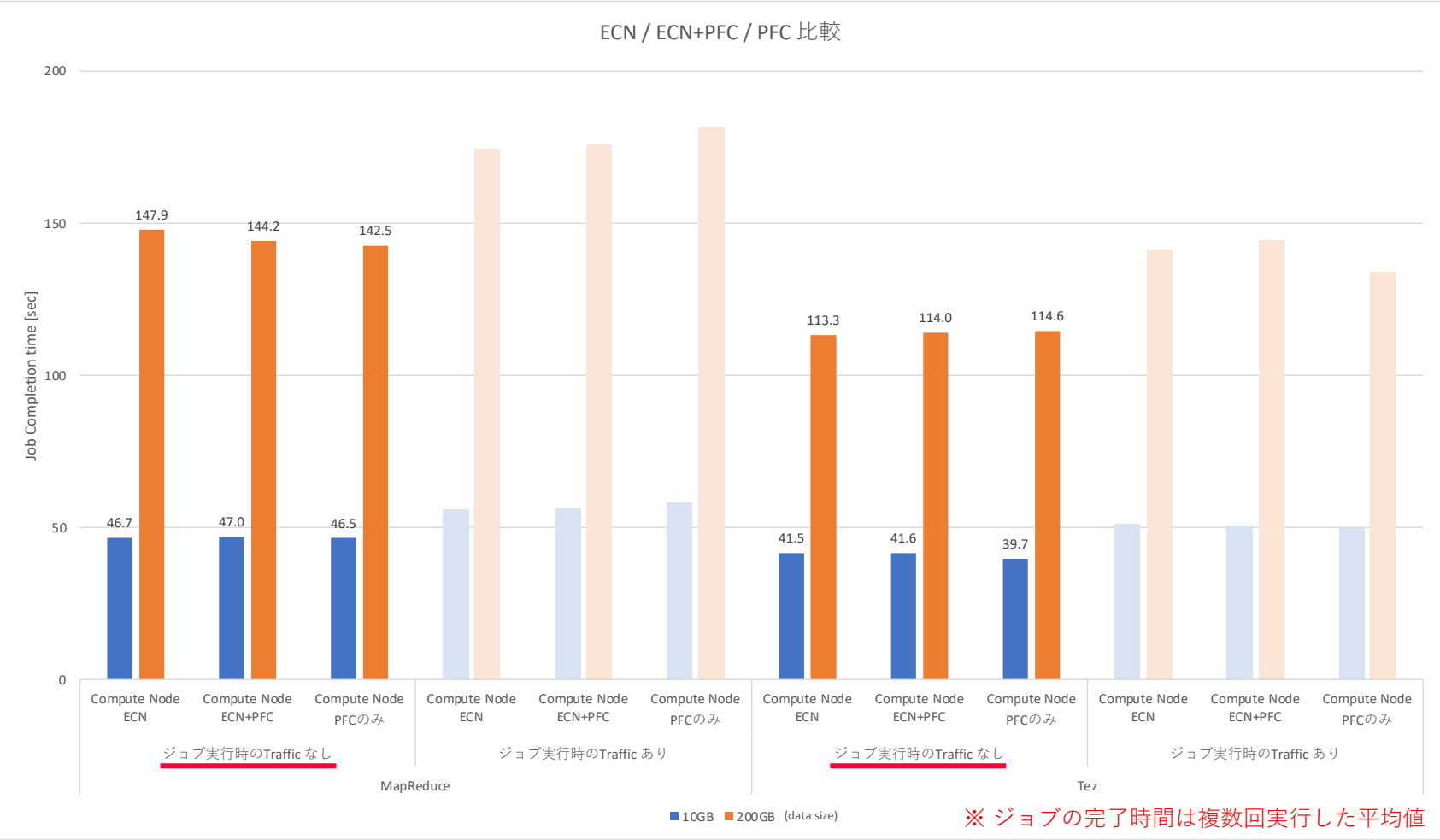
- switch configuration

```
!
int ethX
  priority-flow-control on
  priority-flow-control priority 0 no-drop
  priority-flow-control priority 1 no-drop
!
```

```
> show qos maps
  Number of Traffic Classes supported: 8
  Number of Transmit Queues supported: 8
  Cos Rewrite:   Disabled
  Dscp Rewrite:  Disabled

  Cos-tc map:
    cos:  0  1  2  3  4  5  6  7
    ----------------------------
    tc:   1  0  2  3  4  5  6  7

  Dscp-tc map:
    d1 :  d2 0  1  2  3  4  5  6  7  8  9
    ---------------------------------------
     0 :     1  1  1  1  1  1  1  0  0
     1 :     0  0  0  0  0  0  2  2  2  2
     2 :     2  2  2  2  3  3  3  3  3  3
     3 :     3  3  4  4  4  4  4  4  4  4
     4 :     5  5  5  5  5  5  5  5  6  6
     5 :     6  6  6  6  6  6  7  7  7  7
     6 :     7  7  7  7

  Tc-cos map:
    tc:   0  1  2  3  4  5  6  7
    ----------------------------
    cos:  1  0  2  3  4  5  6  7

  Tc-dscp map:
    tc:    0  1  2  3  4  5  6  7
    ------------------------------
    dscp:  8  0 16 24 32 40 48 56

  Tc - tx-queue map:
    tc:         0  1  2  3  4  5  6  7
    ----------------------------------
    tx-queue:   0  1  2  3  4  5  6  7

  Tc - PriorityGroup map:
    tc:              0  1  2  3  4  5  6  7
    ---------------------------------------
    priority-group:  1  0  2  3  4  4  5  7
```

PFCを動作させる priority は Traffic Class とのマッピングから決定

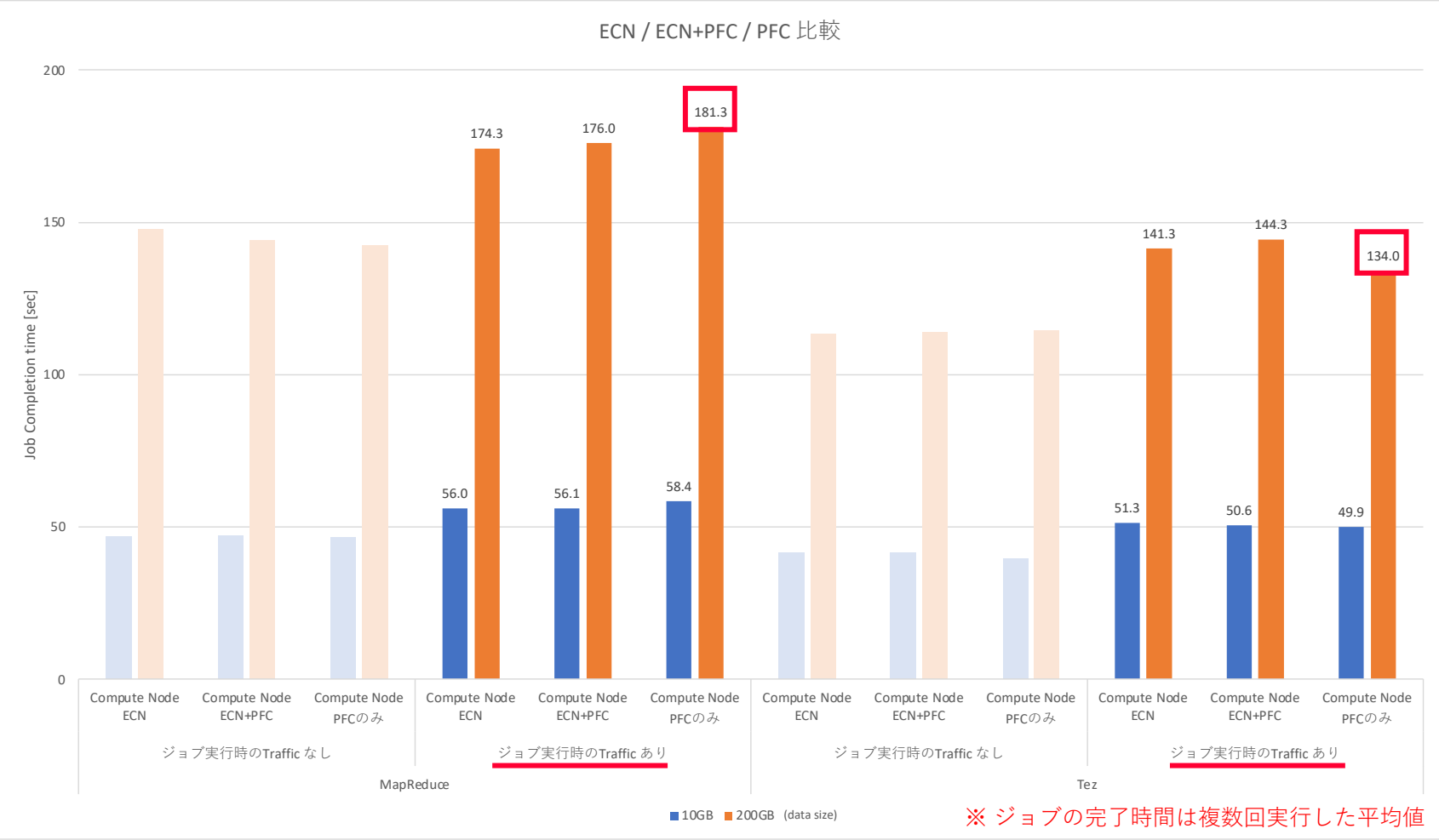
検証パターン3



トラフィックを流していない状態では

MapReduce → PFCのみが最も短い、Tez →ほとんど差はない

検証パターン3



トラフィックを流している状態ではフレームワークによって差がある

MapReduce → PFCのみが一番長い、Tez → PFCのみが一番短い

検証パターン3

ECN+PFC と PFCのみでのPFC Counterの違い (Trafficを流している状況でのジョブ実行時)

PFC のみ

```
> show priority-flow-control counters | nz
Port                 RxPfc          TxPfc
Et1                      0          67564
Et2                      0         120842
Et3                      0         171418
Et5                      0         116843
Et6                      0         318155
Et7                      0         407988
Et8                      0         313726
Et9                      0         156158
Et10                     0         208990
Et11                     0         288596
Et12                     0         314099
Et13                     0         295810
Et14                     0         197029
Et15                     0          70856
Et49/1                   0         119972
Et50/1                   0         125959
Et51/1                   0         108426
Et52/1                   0         102311
```

ECN + PFC

```
> show priority-flow-control counters | nz
Port                 RxPfc          TxPfc
Et50/1                   0              2
```

ECNを設定することで、PAUSEフレームの送信が抑えられていることを確認

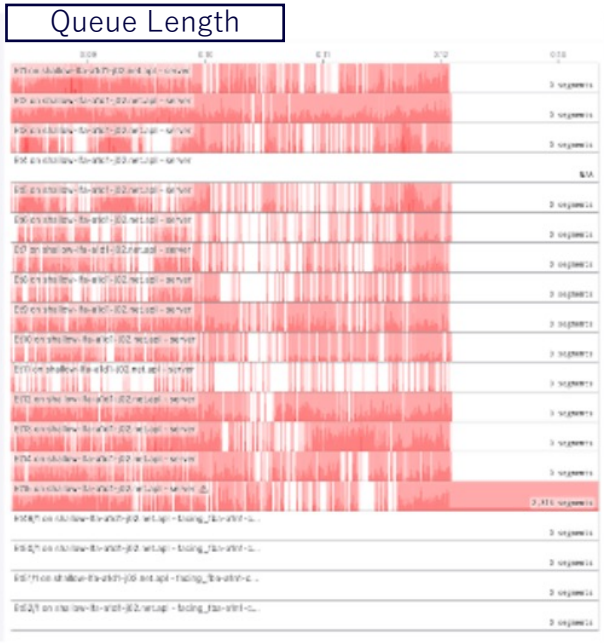
検証パターン3

PFCのみの環境で発生した事象 (Trafficを流している状況でのジョブ実行時)

In Discards

Queue Drops

Queue Length



Transmit Latency

PFCのみの環境ではOut DiscardsではなくIn Discardsが発生

輻輳は発生しているものの、Queue Dropsの発生は抑制された

検証パターン3

結果と考察

- Shallow Buffer + ECN + PFC
    - ECNを利用することでPFCのカウンター増加を抑えることができる
- Shallow Buffer + PFC
    - PFCのみでもQueue Dropsに効果があるが、In Discards が発生するようなった
    - PFC単独利用のケースは考えにくいため無視できると考える(原因未調査)
- PFCを利用する場合、ECNと組み合わせて使うのが良いと考えられる
- PFC stormによるデッドロック回避のためwatchdogの設定が必要になる可能性がある(未検証)

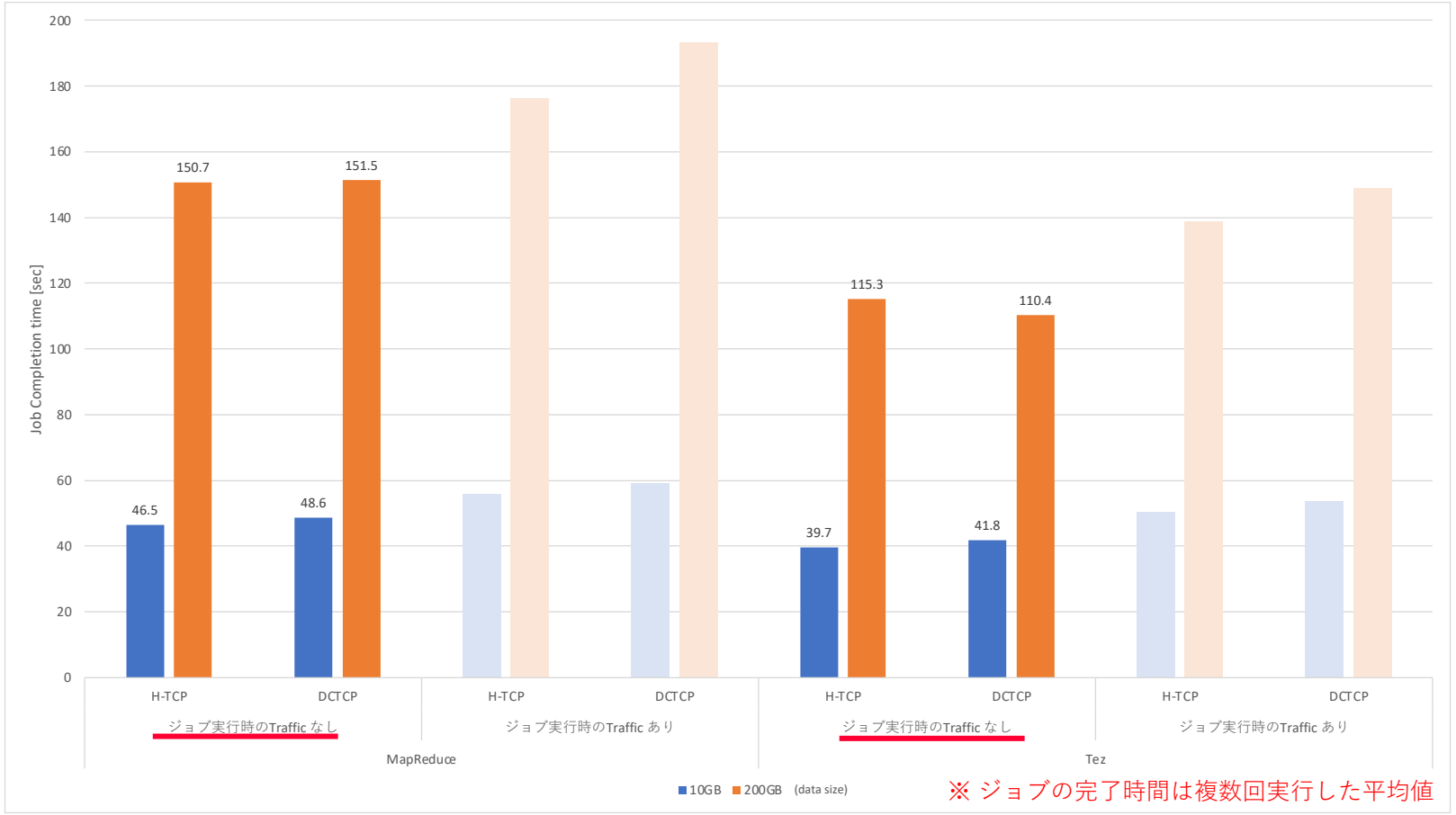
検証結果

パターン4

サーバの輻輳制御アルゴリズムの違いでの比較

検証パターン4

サーバの輻輳制御アルゴリズムの違いでの比較



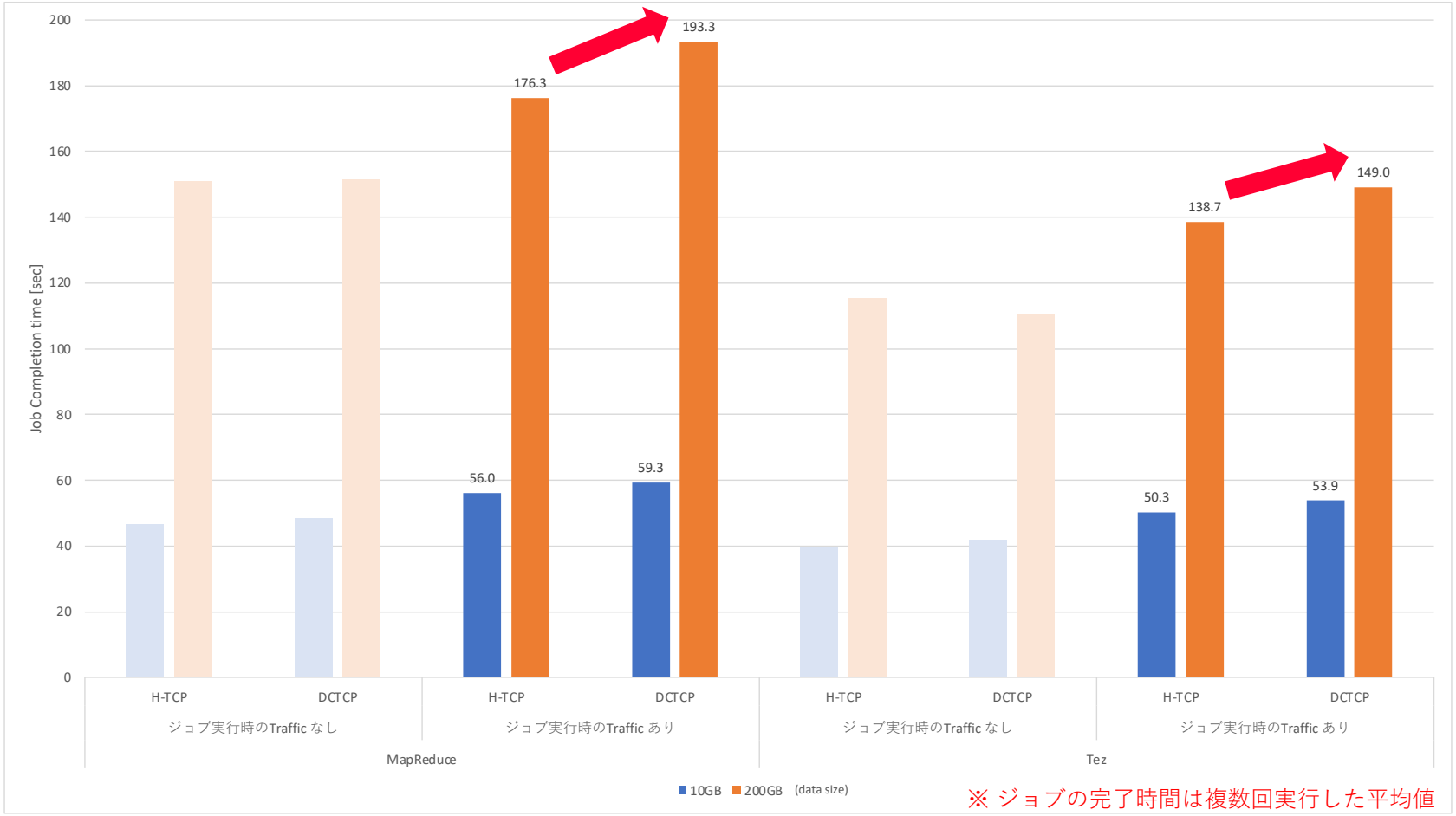
※ ジョブの完了時間は複数回実行した平均値

トラフィック流していない状態では

MapReduce → ほとんど差がない、Tez → DCTCPの方が若干短い

検証パターン4

サーバの輻輳制御アルゴリズムの違いでの比較

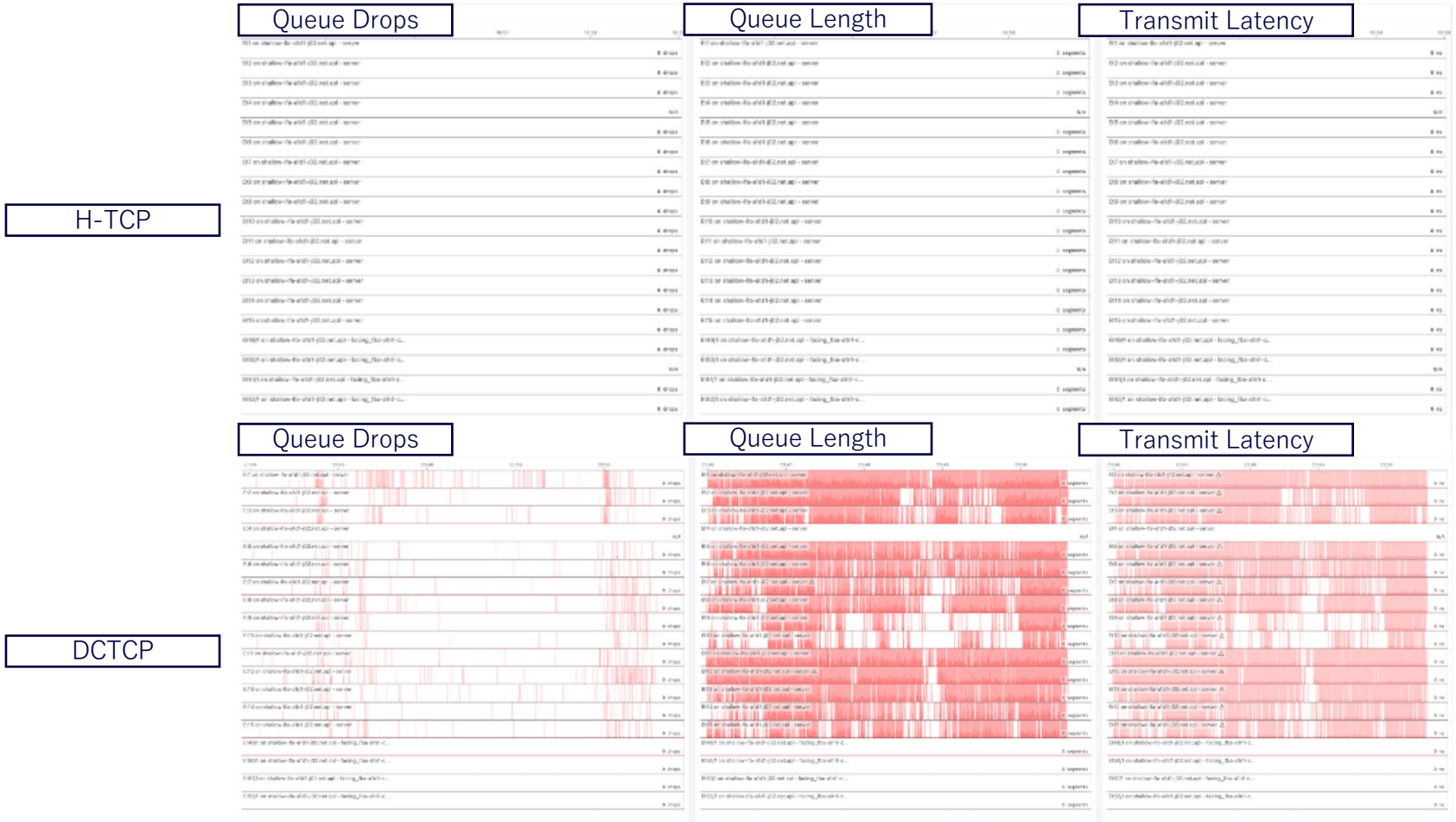


※ ジョブの完了時間は複数回実行した平均値

トラフィック流してる状態では、DCTCPの方がジョブ完了時間が長い

検証パターン4

Queue の比較(Traffic 流している状況でのジョブ実行時)



H-TCPでは発生が抑えられていた、Queue Dropsや輻輳が発生していた

検証パターン4

結果と考察

- Shallow Buffer + ECN + DCTCP
  - 検証パターン1のShallow Bufferと比較してQueue Dropsや輻輳を多少抑えることができた
  - 検証パターン2のShallow Buffer + ECN + H-TCPよりQueue Dropsや輻輳が発生している
- 仮説を立てているが調査中
  - 我々のHadoopではH-TCPの方が適している可能性
  - 想定外の通信(ECNが設定されていないノードとの通信)が発生した影響の可能性

検証パターン4

DCTCPでQueue Dropsが発生した原因の深掘り

- サーバの通信を確認したところ、DCTCPではない通信があった
    - 想定していないData Nodeとの通信
- 想定していないData Nodeとの通信
    - Hadoopジョブのログを**HDFSへアップロードするために発生した通信と判明**
    - 検証用の**ジョブのログのアップロード先を考慮にいれていなかった**
    - そのため、**ECNとDCTCPの設定がされていないData Nodeにアップロード**された

ECNの設定差異により、Queue Drops などが発生した可能性

検証パターン4

DCTCPでQueue Dropsが発生した原因の深掘り

- DCTCPを設定したCompute Node のみの通信に絞って発生させてみたところ**Queue Dropsの発生状況にあまり変化**はなかった

- 我々のHadoopでは、H-TCPの方が性能が良い可能性
- 今回のECN設定対象ではない、管理系ノードとの通信(メタデータの通信)で DiscardsやQueue Dropsなどが発生している可能性

考察

検証結果

Summary

	方式	Queueの利用状況	ジョブ完了時間
1	**Deep Buffer**	• 負荷が高い状態でもDiscardsや Queue Dropsなどを抑えることができた	~~• 短い~~ • **長い**
2	**Shallow Buffer (Host Only, Non-ECN)**	• 負荷が高い状態でDiscardsやQueue Dropsなどが頻繁に発生した	~~• 長い~~ • **短い**
3	**Shallow Buffer (ECN Only)**	• 負荷が高い状態でもDiscardsや Queue Dropsなどを抑えることができた	~~• 1と同程度~~ • **2と同程度**
4	**Shallow Buffer (ECN + PFC)**	• ECNを利用することでPFCカウンターの増加を抑えることができた • **PFCのみでもQueue Dropsを抑えることができた**	~~• 1と同程度~~ • **フレームワークによって差異あり**
5	**Shallow Buffer (ECN + DCTCP)**	• **3と比較して、Queue Dropsなどが発生**	~~• 3よりも短い~~ • **3よりも長い**

考察

検証結果と照らし合わせて

方式	検証で確認できたこと
Deep Buffer	• ジョブ完了時間が長くなる傾向あり • ホスト側に設定が不要でフローの区別を運用者で行う必要がない利点がある
Shallow Buffer – Host Only	• 輻輳時にQueue Dropsなどが発生する • job completion timeに影響はなく、Deep Bufferと比較し、実行時間が最大で8%短くなった
Shallow Buffer – NW Assisted	• ECNを有効にすることで輻輳を抑制できる • 自社の環境に合わせたチューニング(選択)をする必要がある • job completion timeに影響はなく、Deep Bufferと比較し、実行時間が最大で10%短くなった

(再掲)課題とモチベーション

これからのデータセンターネットワークが目指す形

課題

- 用途別の専用構成が乱立しインフラがフラグメント化、構築・運用のコストが増加している
- 大容量なバッファを持つ機器を運用しているが、 そのコストの妥当性が不明
- LossyなNWでも異なる機器や設定を運用しなければならないことの負荷が大きい

我々のモチベーション

- ハードウェアや構成・設定のバリエーションを増やしたくない
- 可能な限り構築・運用のコストパフォーマンスの高い標準構成を定義したい
- 今後の機種選定やconfiguration, コスト算出の根拠となるデータが欲しい

実現に向けて

これからのデータセンターネットワークが目指す形

我々のモチベーション

- ハードウェアや構成・設定のバリエーションを増やしたくない
- 可能な限り構築・運用のコストパフォーマンスの高い標準構成を定義したい
- 今後の機種選定やconfiguration, コスト算出の根拠となるデータが欲しい

検証を通して、

- 特定用途向けのハードウェアを選択する必要性がなくなる可能性が高い
- その場合、特定用途向けの設定が必要になるトレードオフが発生する
- Frontend Fabricの構成については標準化できる見込みがたった

今後

今後

データセンターネットワークのこれから

- ECNがあるネットワークの運用実績を蓄積する
- ネットワークのいまの状態をより深く・詳細に見える化する
    - バッファ使用状況、ECMP-way ..etc
- 今回は既存のTCP環境を対象にしたが次は Lossless Ethernet を対象に検証を行いたい
    - 今回の検証の知見がこれから必要な要求にも活用可能と考えている
    - Fabric側の輻輳対策技術と組み合わせる

議論

Q&A

議論

議論したいポイント

- 今後のデータセンターネットワークに求められる構成や技術について
- 複数のワークロードが混在するネットワークをどのように運用していけば良いか
- データセンターネットワークでどのような輻輳制御・対策を実施していますか？
- ネットワーク機器のバッファ監視・可視化をしてますか？
- ECNやPFCの閾値はどのように、何を根拠に決定していますか？
- サーバ(のNIC)とネットワーク機器がこれまで以上に密接に連携する時代が来ていると思いますが、運用で困ったことありませんか？
- 問題になったケースや痛い目に合った経験談など

Appendix

各Hadoopジョブの実行時間

検証パターン1

Deep Buffer と Shallow Buffer の比較（ECN設定なし）

			10GB	200GB
MapReduce	ジョブ実行時のTraffic なし	Deep Buffer	46.1	148.4
		Shallow Buffer	45.4	148.4
	ジョブ実行時のTraffic あり	Deep Buffer	70.4	193.6
		Shallow Buffer	55.4	178.5
Tez	ジョブ実行時のTraffic なし	Deep Buffer	39.4	104.9
		Shallow Buffer	42.9	106.3
	ジョブ実行時のTraffic あり	Deep Buffer	59.0	142.7
		Shallow Buffer	48.8	139.7
			(sec)	(sec)

※ ジョブの完了時間は複数回実行した平均値

検証パターン2

Compute Node ECN 設定の有無の比較

			10GB	200GB
MapReduce	ジョブ実行時のTraffic なし	ECN設定なし	45.4	148.4
		Compute Node ECN設定あり	46.7	147.9
	ジョブ実行時のTraffic あり	ECN設定なし	55.4	178.5
		Compute Node ECN設定あり	56.0	174.3
Tez	ジョブ実行時のTraffic なし	ECN設定なし	42.9	106.3
		Compute Node ECN設定あり	41.5	113.3
	ジョブ実行時のTraffic あり	ECN設定なし	48.8	139.7
		Compute Node ECN設定あり	51.3	141.3
			(sec)	(sec)

※ ジョブの完了時間は複数回実行した平均値

検証パターン3

ECN+PFC と PFCのみの比較

			10GB	200GB
MapReduce	ジョブ実行時のTraffic なし	Compute Node ECN	46.7	147.9
		Compute Node ECN+PFC	47.0	144.2
		Compute Node PFCのみ	46.5	142.5
	ジョブ実行時のTraffic あり	Compute Node ECN	56.0	174.3
		Compute Node ECN+PFC	56.1	176.0
		Compute Node PFCのみ	58.4	181.3
Tez	ジョブ実行時のTraffic なし	Compute Node ECN	41.5	113.3
		Compute Node ECN+PFC	41.6	114.0
		Compute Node PFCのみ	39.7	114.6
	ジョブ実行時のTraffic あり	Compute Node ECN	51.3	141.3
		Compute Node ECN+PFC	50.6	144.3
		Compute Node PFCのみ	49.9	134.0
			(sec)	(sec)

※ ジョブの完了時間は複数回実行した平均値

検証パターン4

サーバの輻輳制御アルゴリズムの違いでの比較

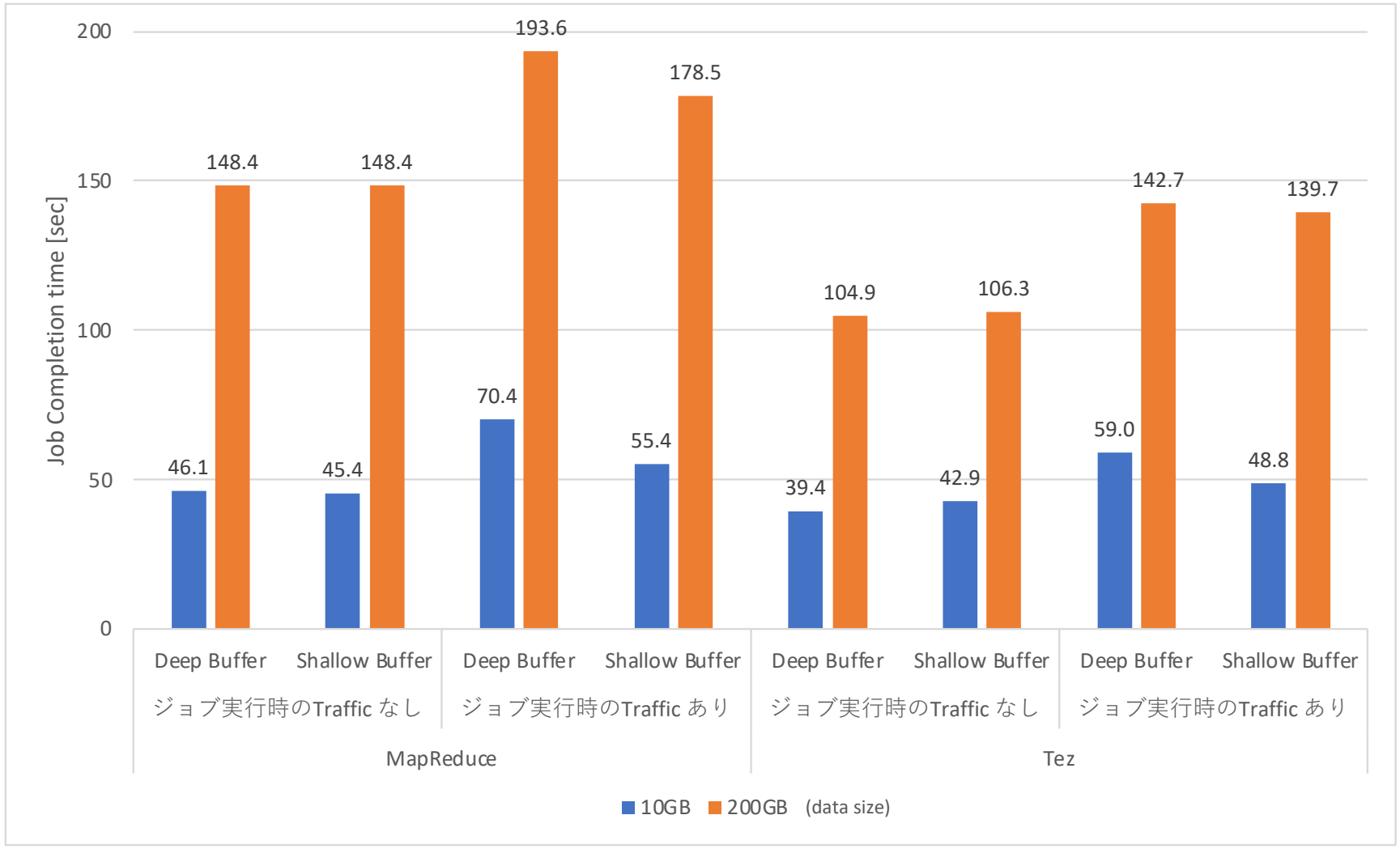
			10GB	200GB
MapReduce	ジョブ実行時のTraffic なし	H-TCP	46.5	150.7
		DCTCP	48.6	151.5
	ジョブ実行時のTraffic あり	H-TCP	56.0	176.3
		DCTCP	59.3	193.3
Tez	ジョブ実行時のTraffic なし	H-TCP	39.7	115.3
		DCTCP	41.8	110.4
	ジョブ実行時のTraffic あり	H-TCP	50.3	138.7
		DCTCP	53.9	149.0
			(sec)	(sec)

※ ジョブの完了時間は複数回実行した平均値

Appendix

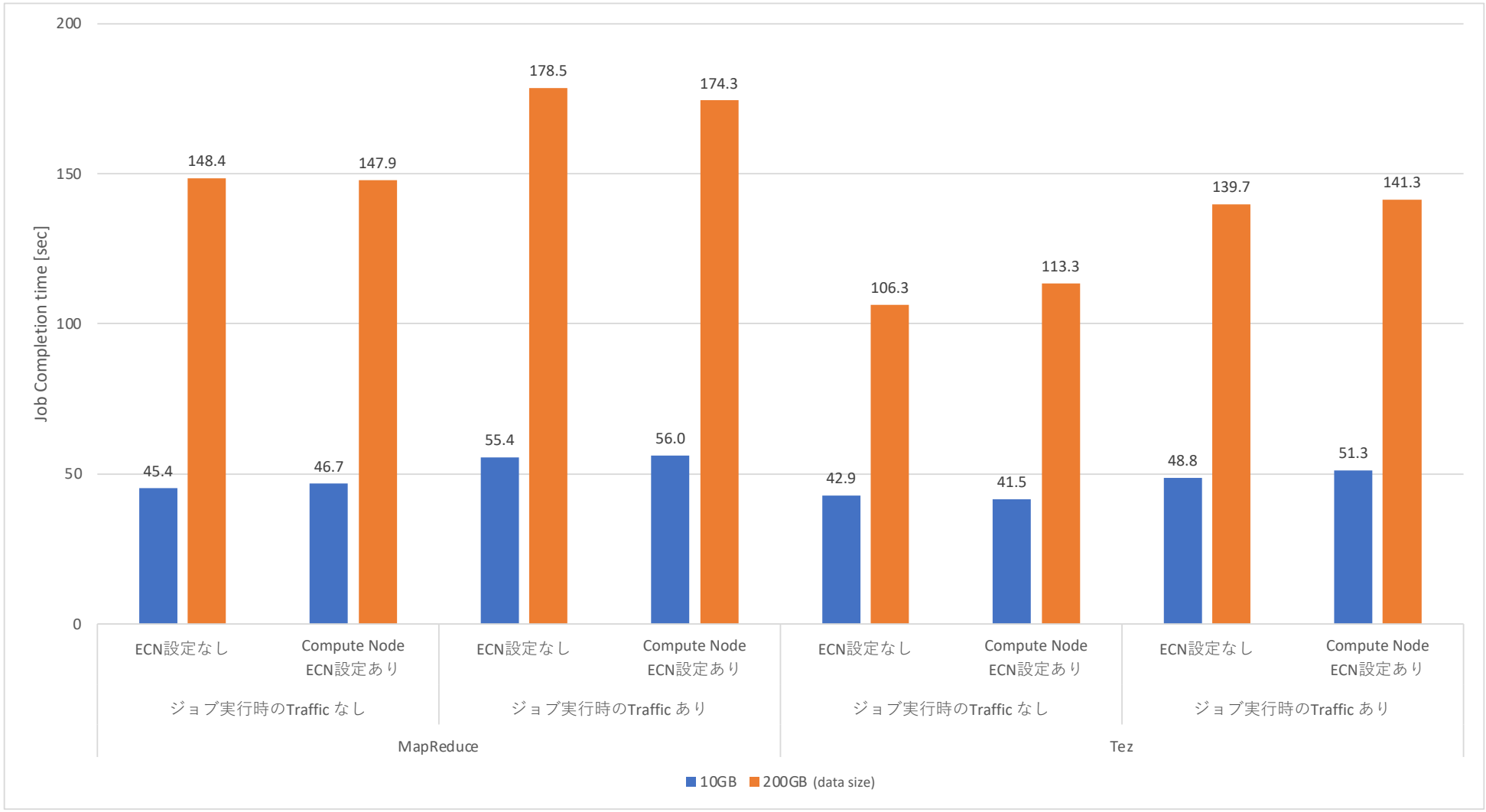
グラフ

検証パターン1



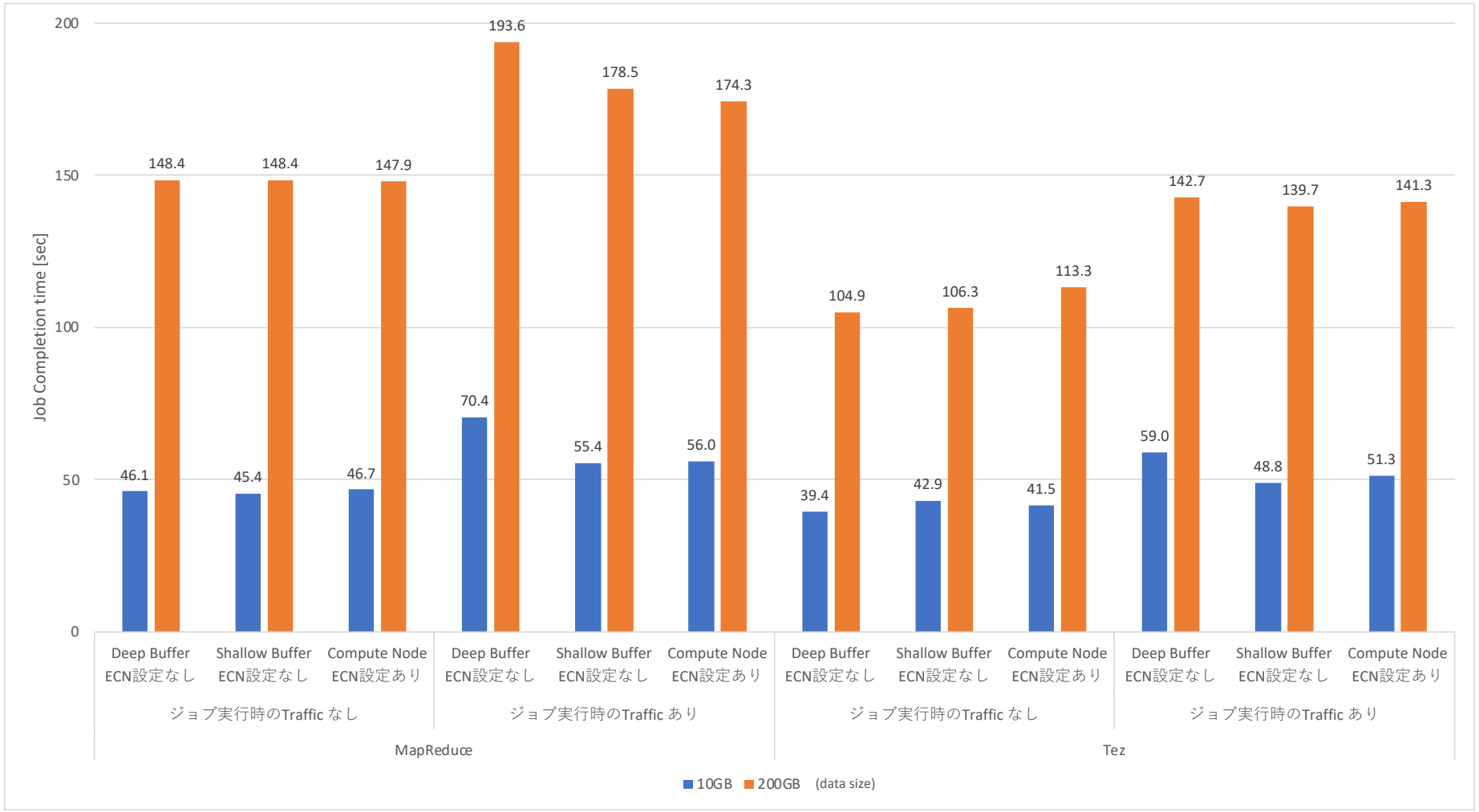
検証パターン2

Compute Node ECN 設定の有無の比較



検証パターン3

ECN+PFC と PFCのみの比較



検証パターン4

サーバの輻輳制御アルゴリズムの違いでの比較