論文輪講
Towards Federated Learning at Scale: System Design

杉浦 圭祐

慶應義塾大学理工学部情報工学科 松谷研究室

May 7, 2019

目次

目次

Federated Learning とは

- Federated Learning とは
  - 分散機械学習の新手法
    ⇒ 複数のデバイス間に分散したデータを利用し、共通のモデルを学習
  - 既存の分散機械学習の手法とは異なり、プライバシー等の問題を解決
    ⇐ 学習は各デバイス上で行われ、その結果が共通のモデルに反映される

Federated Learning とは

- 一般的な機械学習との違い
  - 学習に使用する訓練データは、クラウド上には保存されない
    ⇒ 全てのデータは、各デバイスに残されたままである
    ⇒ プライバシーや、データの所有権の問題に対処できる
  - クラウド上では、モデルの学習を行わない
    ⇒ モデルの学習は、各デバイス上で (オンデバイスで) 行われる
    ⇒ 学習時には、自身のデバイス上のデータを用いる
  - 各デバイスは、モデルを使用した推論だけでなく、学習も行う
    ⇒ 学習後、クラウド上にある共通のモデルの、パラメータを更新
    ⇒ 各デバイスからクラウドへは、モデルの更新情報のみ送信
  - 各デバイスで学習されたモデルを、即座に利用できる
    ⇒ クラウド上のモデルをベースとして、各デバイス向けにカスタマイズ可能

Federated Learning とは

- Federated Learning の大まかな流れ
  1. 各デバイスが、クラウド上にある現在のモデルをダウンロードする
  2. デバイス上のデータを使って、モデルを学習する
  3. 学習が終わったら、モデルのパラメータの変更内容 (差分) をまとめる
  4. 差分をクラウドに送信し、クラウド上の共通のモデルに反映させる
  5. (1) から (4) までを、繰り返し行う

Federated Learning とは

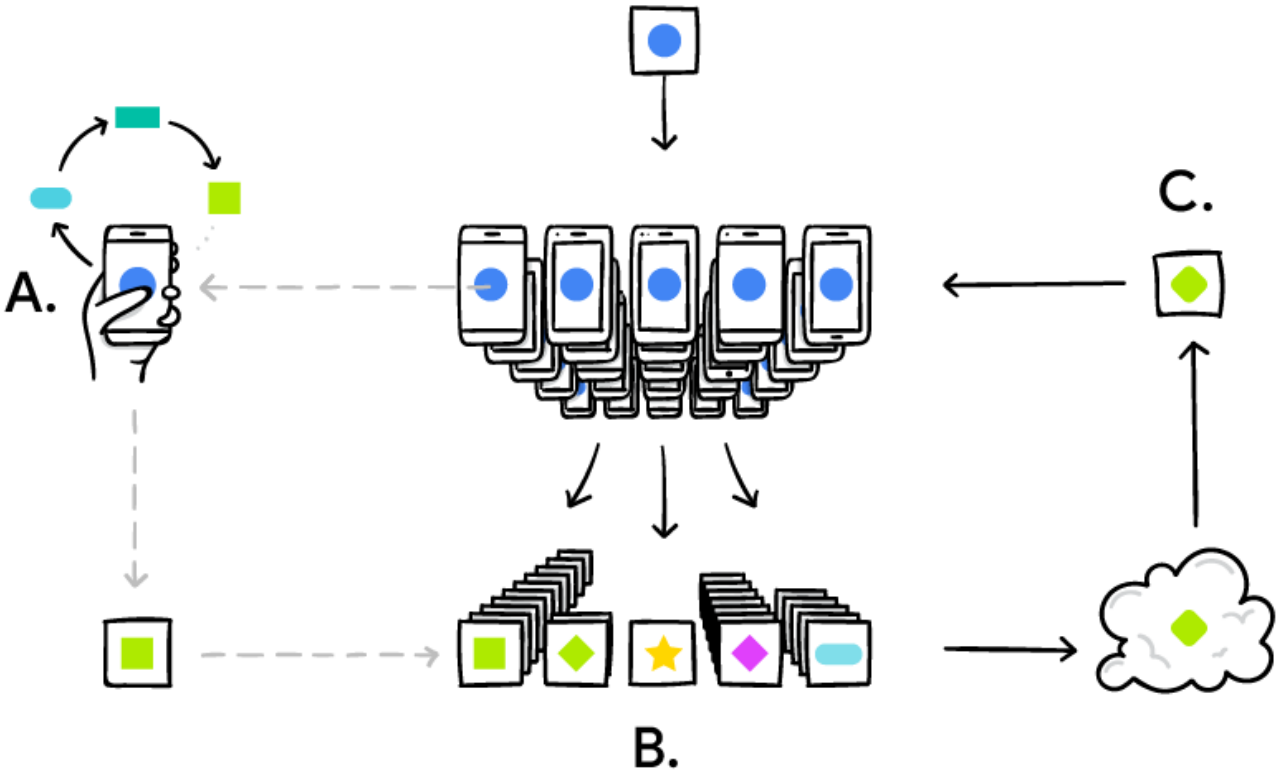

図 1: Federated Learning の流れ [2]

目次

システムの概要

- システムの概要
  - モバイル端末 (Android のスマートフォン) を対象としたシステム
  - スケーラブルかつ、実際の製品にもデプロイ可能なレベル
    ⇐ Gboard(Google Keyboard) というキーボードアプリで実際に使用
  - 実装には、TensorFlow が用いられている
    ⇐ 深層ニューラルネットの学習も可能である
  - 同期型の訓練アルゴリズム (Federated Averaging) を採用
  - セキュリティ向上のための手法 (Secure Aggregation) を利用可能

同期型の訓練アルゴリズム

- 同期型の訓練アルゴリズムが採用された
  - 具体的には、Federated Averaging というアルゴリズムを使用
    ⇐ SGD(Stochastic Gradient Descent) とよく似ている
    ⇐ SGD を、重み付きの更新によって拡張したような手法

- クラウド上のモデルが更新されるまでの流れ
  1. 各デバイスから、差分データ (モデルのパラメータの更新情報) を受信
  2. Federated Averaging を用いて、クラウド上で差分データを一つに集約
  3. 集約された差分を、モデルのパラメータに反映 (モデルの更新)
  4. 更新されたモデルを、各デバイスが取得できるようになる

同期型の訓練アルゴリズム

Algorithm 1 `FederatedAveraging` targeting updates from K clients per round.

Server executes:

initialize w_0

for each round $t = 1, 2, \ldots$ **do**

Select $1.3K$ eligible clients to compute updates

Wait for updates from K clients (indexed $1, \ldots, K$)

$(\Delta^k, n^k) = \text{ClientUpdate}(w)$ from client $k \in [K]$.

$\bar{w}_t = \sum_k \Delta^k$ *// Sum of weighted updates*

$\bar{n}_t = \sum_k n^k$ *// Sum of weights*

$\Delta_t = \Delta_t^k / \bar{n}_t$ *// Average update*

$w_{t+1} \leftarrow w_t + \Delta_t$

ClientUpdate(w)**:**

$\mathcal{B} \leftarrow$ (local data divided into minibatches)

$n \leftarrow |\mathcal{B}|$ *// Update weight*

$w_{\text{init}} \leftarrow w$

for batch $b \in \mathcal{B}$ **do**

$w \leftarrow w - \eta \nabla \ell(w; b)$

$\Delta \leftarrow n \cdot (w - w_{\text{init}})$ *// Weighted update*

// Note Δ is more amenable to compression than w

return (Δ, n) to server

同期型の訓練アルゴリズム

- 同期型の訓練アルゴリズムが採用された
  1. 近年、同期型の訓練アルゴリズムを採用する動きがみられるため
     ⇐ 同期処理の負担が大きなデータセンタですら、同期型の訓練アルゴリズムを採用する傾向
  2. プライバシーを強化する手法を適用するため
     ⇐ Differential Privacy や Secure Aggregation などの手法がある
     ⇐ これらは原則として、同期型のアルゴリズムでないと適用不可能
  3. サーバ側での処理が単純になるため
     ⇐ 多数のユーザ (デバイス) からの更新データをまとめて、モデルに適用

  - 但し、同期処理のオーバヘッドを軽減するための対策が必要 (後述)

セキュリティ向上のための手法

- セキュリティ向上のための手法を利用可能
  - 具体的には、Secure Aggregation という手法を利用可能
    ⇐ 各デバイスから送信される差分データは、外部から隠される
  - Federated Learning では、訓練データは各デバイス上に留まる
    ⇐ 訓練データには、個人を特定するに足る情報が含まれるかもしれない
    ⇐ クラウド上にデータを保存しないことで、プライバシーが確保される
  - データは送信しない代わりに、データを元に算出した差分データを送信
    ⇐ 差分データには、各デバイスを特定するのに十分な情報が、依然として含まれる可能性
    ⇐ 差分データをも隠蔽することで、セキュリティを更に向上させられる

システムを実装する上での課題

- システムを実装する上での課題が非常に多い
  1. デバイスが常に接続されているとは限らない
     ⇐ デバイス (スマートフォン) の接続状態は不安定になりがち
  2. デバイスが常に計算可能とは限らない
     ⇐ 計算が途中で中断させられるかもしれない
     ⇐ デバイスは世界中に散らばって存在するため、地理的な要因 (タイムゾーンなど) を考慮する必要がある
  3. 複数のデバイスの同期処理を取るのが困難
     ⇐ 前述の通り、これらのデバイスは接続状態が不安定で、常に利用できるかどうかも分からない
  4. デバイスの計算能力とストレージの制限が厳しい
     ⇐ 深層ニューラルネットの場合はパラメータ数が多く、メモリと計算資源の消費が特に大きい

システムを実装する上での課題

- これらの課題を 3 つの構成要素で解決
  - 通信プロトコル、デバイス、サーバ
  - この 3 つの動作について、これからみていく

  - 論文の著者によれば、このシステムは数百万、あるいは十億台のデバイス上で利用可能だとしている

システムを実装する上での注意点

- デバイス上で Federated Learning を無条件に開始してはならない
  - デバイスのパフォーマンスを劣化させてはいけない
  - Federated Learning が実行されるのは、デバイスがアイドル状態で、充電中で、かつ Wi-Fi に接続されているときのみ
    ⇒ Federated Learning のために、ユーザがデバイスを使えなくなってしまうのでは、本末転倒
    ⇒ 次の図 2 を参照

システムを実装する上での注意点

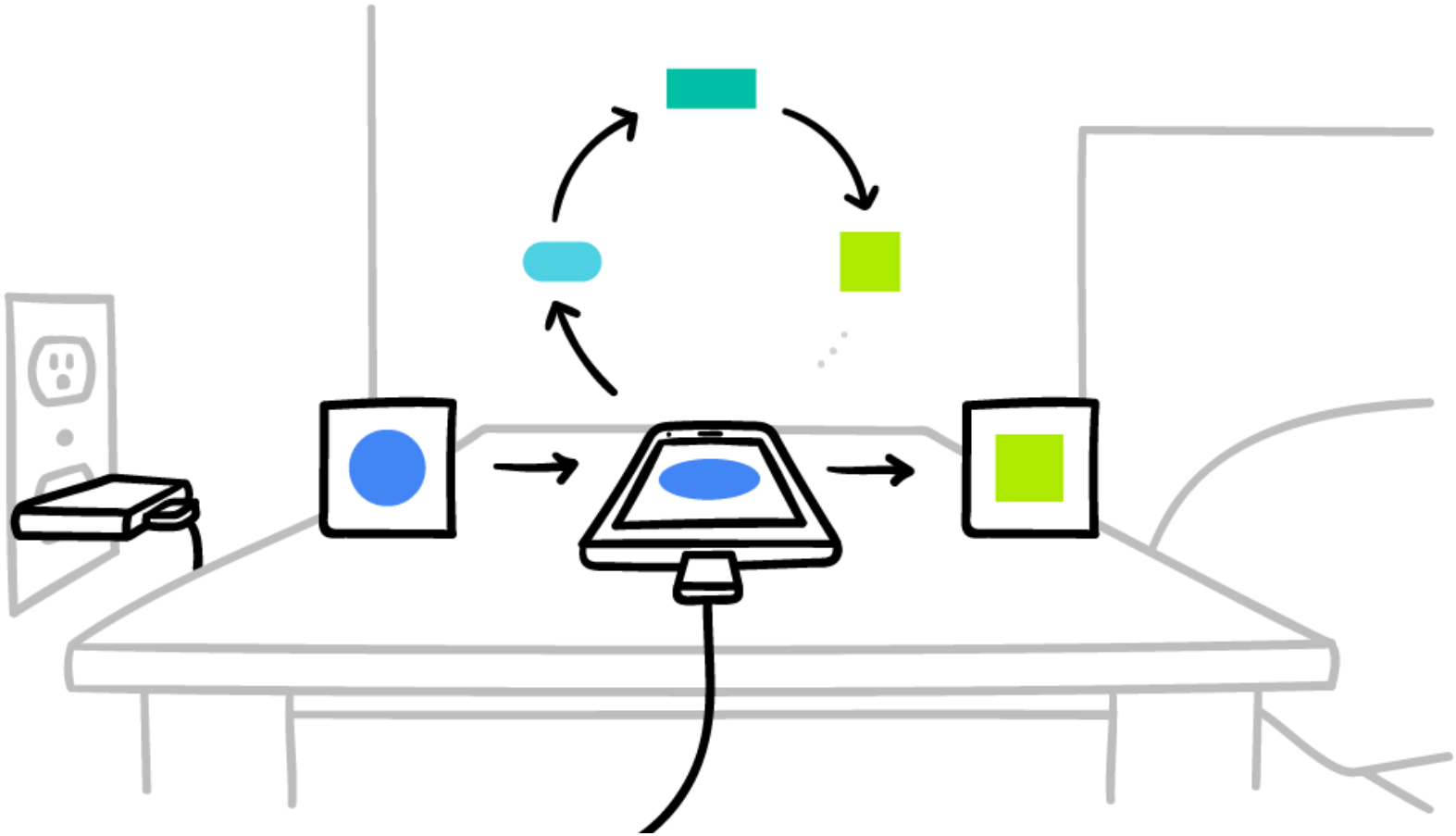

図 2: Federated Learning が実行される条件 [2]

目次

通信プロトコルの用語整理

- プロトコルの主人公
  - デバイス (ここでは Android のスマートフォン)
  - FL Server (クラウドベースの分散サービス)

- FL Population
  - 学習アルゴリズムで解こうとしている問題

- FL Task
  - 特定の計算タスク
    ⇐ あるハイパーパラメータが与えられた下での、モデルの訓練
    ⇐ デバイス上のローカルなデータを用いた、モデルの評価

  - FL Population は、複数の FL Task によって構成される
    ⇒ FL Task は、必ず何らかの FL Population に属する

通信プロトコルの用語整理

- FL Plan
  - FL Task において、デバイスが行うべき具体的な処理内容
  - 例えば、モデルの訓練方法や評価手順を表す
  - TensorFlow の計算グラフや、タスクの実行方法を格納するデータ構造
- FL Checkpoint
  - クラウド上の現在のモデルのパラメータ
  - その他の状態 (シリアライズ化された TensorFlow のセッション)
- Round
  - FL Server とデバイスとの一連の通信 (後述)
  - Selection、Configuration、Reporting の 3 段階で構成される
  - 次の図 3 を参照

Federated Learning のプロトコル

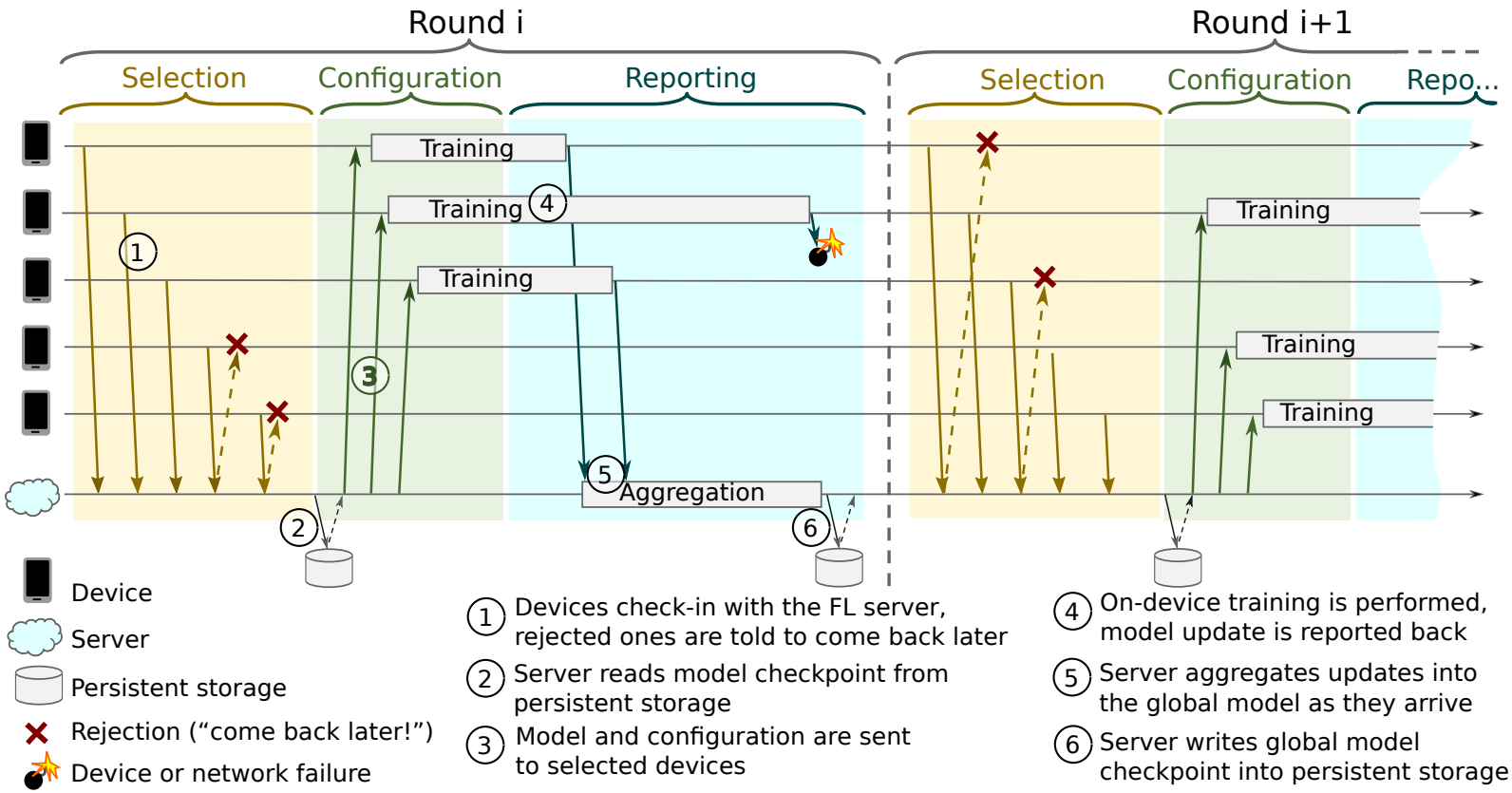

Figure 1: Federated Learning Protocol

図 3: Federated Learning のプロトコル [1]

Federated Learning のプロトコル

- Federated Learning のプロトコルの大まかな流れ
  1. FL Server は、ある決められた時間だけ、デバイスからの報告を待機
     $\Leftarrow$ ある特定の FL Task が実行可能であることの報告を待つ
  2. 多数のデバイスが、指定された FL Task を実行可能であることを、FL Server に伝達
  3. FL Server は、報告してきた数千のデバイスの中から、数百程度のデバイスを選択
  4. 選ばれた数百のデバイスで、FL Task を実行する
  5. (1) から (4) までを繰り返す
     $\Rightarrow$ この繰り返しの単位を Round という
     $\Rightarrow$ (1) から (3) までが Selection フェーズ
     $\Rightarrow$ (4) が Configuration と Reporting フェーズ

Federated Learning のプロトコル

- Federated Learning のプロトコルの Round の流れ
  - Round は Selection、Configuration、Reporting の 3 段階で構成される
  - Round の間は、選択されたデバイスは FL Server との通信を継続する

  - Round の実行途中で、時間内に応答しないデバイスは単に無視される
  - Federated Protocol のプロトコルは、このようなデバイスの脱落を考慮に入れて設計されている

Federated Learning のプロトコル

- Selection フェーズの流れ
  1. FL Task の実行に適したデバイスは、周期的に FL Server にアクセス
     ⇐ 充電中で、かつ Wi-Fi に接続されているデバイス
     ⇐ 従量課金制のネットワークに接続されたデバイスは、アクセスしない (Federated Learning には参加しない)
  2. FL Server にアクセスしたデバイスは、双方向のコネクションを確立
     ⇐ Round の間は、コネクションを維持する必要がある
  3. FL Server は、接続してきた数千のデバイスの中から、数百程度のデバイスを選択
     ⇐ 1 つの Round には数百程度のデバイスが参加
     ⇐ 選択に使用するアルゴリズムは何でもよい (溜池サンプリング)
  4. 選ばれなかったデバイスに対して、FL Server は次にアクセスすべき時刻を送信 (適当な時間の経過後に、再接続させる)

Federated Learning のプロトコル

- Selection フェーズで指定可能なパラメータの例
  - FL Task の実行に協力して欲しいデバイスの数 (希望)
  - FL Task の実行に最低限必要なデバイスの数 (閾値)
  - FL Server がデバイスからの接続を待つべき時間 (タイムアウト)

  - 希望通りの数のデバイスが接続してきた時点で、Round の実行が開始
  - タイムアウトになるまでは、接続デバイス数が希望通りになるまで待機

  - タイムアウト時に、接続デバイス数が閾値を超えていなければ、Round は実行されない

Federated Learning のプロトコル

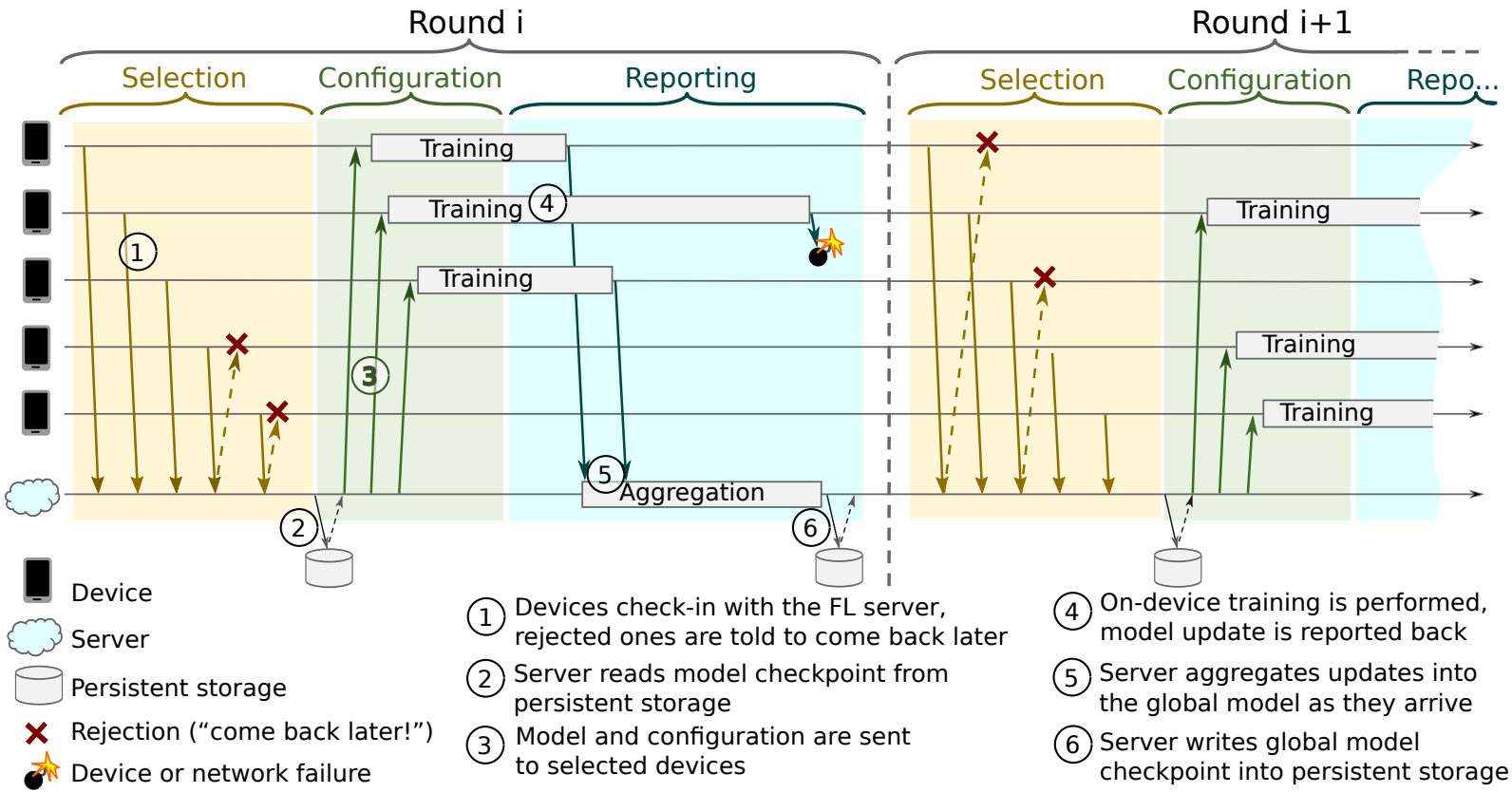

Figure 1: Federated Learning Protocol

図 4: Federated Learning のプロトコル (再掲) [1]

Federated Learning のプロトコル

- Configuration フェーズの流れ
  1. 選択されたデバイスに対して、FL Plan を送信
     ⇐ FL Plan は、TensorFlow の計算グラフや、FL Task の実行方法を格納
  2. 続いて、選択されたデバイスに対して、FL Checkpoint を送信
     ⇐ FL Checkpoint は、モデルのパラメータや、FL Task の実行に必要な様々な情報を格納
     ⇐ シリアライズ化された TensorFlow のセッションオブジェクトなど
  3. デバイスは、FL Server から渡された情報を元に、FL Task を実行
     ⇐ デバイス上に保存されたデータを用いた、モデルの訓練や評価

Federated Learning のプロトコル

- Reporting フェーズの流れ
  1. FL Server は、デバイスから結果が送信されるのを待機
     ⇐ FL Task がモデルの訓練であれば、差分データ (モデルのパラメータの更新情報) の送信を待機
  2. FL Server は、Federated Averaging を使って、差分データを一つに集約
     ⇐ 集約された差分を、モデルのパラメータに反映 (モデルの更新)
  3. FL Server は、タスクを実行し終えたデバイスに対して、次にアクセスすべき時刻を送信
     ⇐ 適当な時間の経過後に、デバイスが FL Server に再接続するように指示

  - 十分な数のデバイスが結果を報告すれば、モデルの更新が実行される
  - それ以外の場合は、Round の実行は失敗 (無かったことにされる)

Federated Learning のプロトコル

- デバイスの FL Server への接続頻度の調節
  - FL Population(解こうとしている問題) の大きさに応じて、デバイスの接続頻度 (一度に FL Server に接続してくるデバイス数) を調節
  - FL Server は、次に再接続すべき時間を、各デバイスに対して指示する
    ⇒ この時間をうまく調節することで、デバイスの接続頻度を調節可能
  - FL Task に協力可能 (アクティブ) なデバイスの数は、周期的に変動
    ⇒ 充電中で、Wi-Fi に接続されていて、アイドル状態であればアクティブとみなす
    ⇒ 昼の時間帯は人間が使用するので、アクティブなデバイスが減少
    ⇒ それ以外の時間帯 (深夜) では、逆にアクティブなデバイスが増加
  - これらの周期的な変動も考慮して、再接続までの時間を指定
    ⇒ 例えば、昼間の時間帯は、FL Server にアクセスする頻度を落とす

Federated Learning のプロトコル

- 小さな FL Population の場合 (解こうとしている問題が小さい)
  - FL Task に参加するデバイスの数も少なくて済む
  - 十分な数のデバイスが、FL Server にほとんど同時に接続できるように、接続頻度を上手く調節する
    ⇒ Selection フェーズに掛かる時間が短縮
    ⇒ 単位時間に実行可能な Round の数が増加
    ⇒ 学習が速やかに進行する

Federated Learning のプロトコル

- 大きな FL Population の場合 (解こうとしている問題が大きい)
  - 一般的に、FL Task に協力してくれるデバイスが多数存在する
  - 但し、一度の Round に参加するデバイスは、せいぜい数百程度である

  - 多数のデバイスが、一度に FL Server に接続しないようにする
    ⇒ 一度に多数のデバイスが FL Server に接続しても、そのうちのごく一部のデバイスが選択され、他の多数のデバイスの接続が無駄になるかもしれない (Thundering Herd と呼ばれる問題)

  - 各デバイスが、FL Server にアクセスする時間を、ランダムに決める
    ⇒ デバイスの接続を時間的に分散させる
    ⇒ 必要なときに、必要な数のデバイスだけが接続する

目次

デバイス上のソフトウェア

- デバイス上のソフトウェア
  - 今回のシステムは、Android のスマートフォンが対象
  - 但し、それ以外のプラットフォームでも実装可能

  - アプリケーションプロセス、FL Runtime、Example Store の 3 つが連携して動作
  - この 3 つは、Android の AIDL IPC(Inter Process Communication; プロセス間通信) を使って互いに通信を行う
  - 次の図 5 を参照

デバイス上のソフトウェアのアーキテクチャ

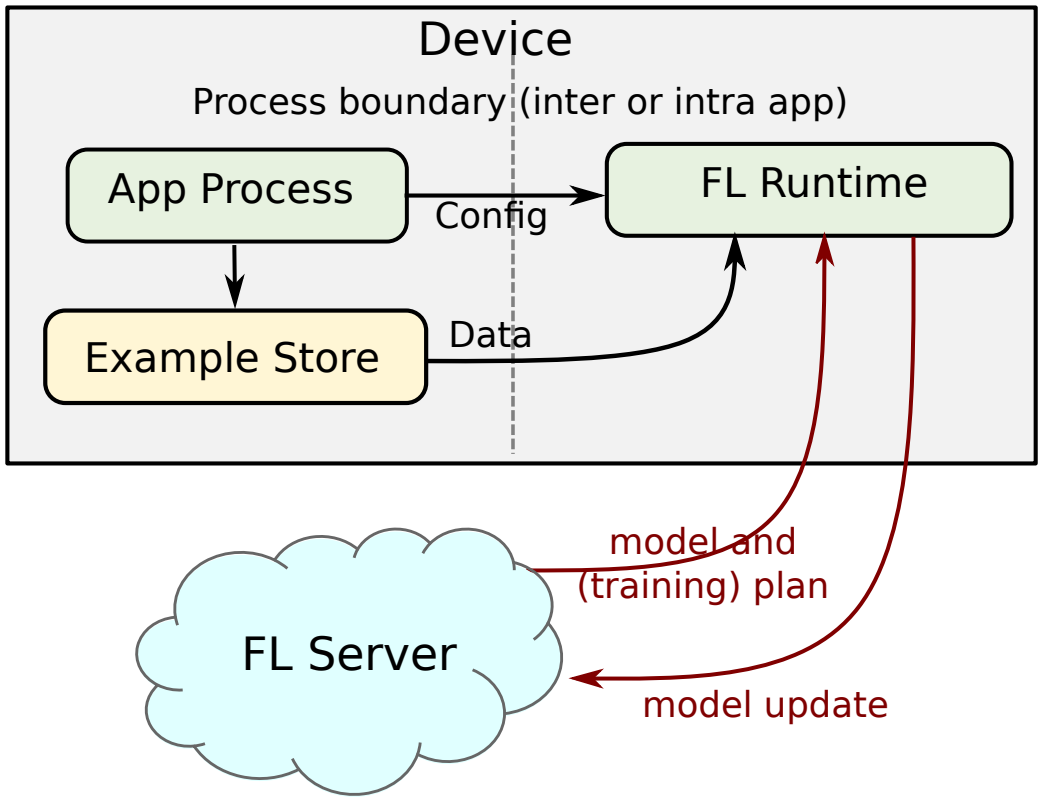

Figure 2: Device Architecture

図 5: デバイス上のソフトウェアのアーキテクチャ [1]

デバイス上のソフトウェア

- Example Store
  - Federated Learning で使用するデータを保存しておくデータベース
  - クラウド上にあるモデルの学習と、改良に用いられる訓練データ

  - 以下のような推奨事項がある

1. ストレージを圧迫しないように、データベースの最大容量を、予め決めておく
2. 各データの保存期間を決めておき、期間を過ぎたデータが自動的に削除されるようにする
3. マルウェアによる不正アクセスを防止するため、各データを適切に暗号化したうえで保存する

デバイス上のソフトウェア

- FL Runtime
  - FL Server とのやり取りを行うソフトウェアのコンポーネント
  - FL Task を実行する際、Example Store からデータを取得
  - 取得したデータを用いて、FL Task で指定された処理 (モデルの訓練や評価) を実行する

デバイス上のソフトウェア

- デバイス上での Federated Learning の流れ
  1. アプリケーションによる FL Runtime の設定
  2. FL Runtime が FL Server にアクセス
  3. FL Runtime が FL Plan を実行
  4. FL Runtime が差分データやその他の統計情報を FL Server に送信

デバイス上のソフトウェア

1. アプリケーションによる FL Runtime の設定
   1. アプリケーションが、FL Population と Example Stores を設定する

   2. Android の JobScheduler により、FL Runtime が周期的に起動される
      ⇒ 起動されるのは、デバイスがアイドル状態で、充電中で、Wi-Fi に接続されているときのみ
      ⇒ Federated Learning のためにユーザがデバイスを使えなくなってしまっては本末転倒

   3. FL Runtime は必要ならば、実行途中であっても、強制終了する
      ⇒ 強制終了するのは、デバイスが上記の 3 条件を満たさなくなったとき

デバイス上のソフトウェア

2. FL Runtime が FL Server にアクセス
   1. FL Runtime がスケジューラによって、アプリケーションとは別のプロセスで起動される
   2. FL Runtime は、FL Task が実行可能であることを、FL Server に伝達
      ⇐ この FL Task は、アプリケーションによって設定された FL Population に属する
   3. FL Server は、FL Population に対応した FL Task があれば、FL Plan を返す
   4. FL Task がない、あるいはデバイスが選択されない場合は、FL Server は、再接続すべき時間を返す
      ⇒ このとき、FL Runtime は終了する
      ⇒ 指定された時間後に、スケジューラによって FL Runtime が再度起動される

デバイス上のソフトウェア

3. FL Runtime が FL Plan を実行
   1. FL Task があり、かつデバイスが選択された場合は、FL Plan が FL Server から送信される
      ⇐ FL Plan は、FL Task において、デバイスが行うべき具体的な処理内容が格納されている
   2. FL Runtime は、Example Store からデバイス上のローカルなデータを取得
   3. 取得したデータを使って、FL Plan で指示された処理を実行
      ⇒ 例えば、データを用いたモデルの訓練
      ⇒ 或いは、データを用いたモデルの評価値の計算
      ⇒ これは、教師あり学習におけるバリデーションに相当
   4. モデルのパラメータの差分や、あるいはモデルの評価結果などを算出

デバイス上のソフトウェア

4. FL Runtime が差分データやその他の統計情報を FL Server に送信
   1. FL Plan の実行後、FL Runtime が差分データやその他の情報を FL Server に送信
   2. 一時的なリソースを解放して、FL Runtime が終了
      ⇒ 次の FL Runtime も、スケジューラによって起動される

デバイス上のソフトウェア

- ソフトウェアの特長
  1. 1 つのアプリケーションが複数の FL Population を設定可能
     ⇒ 1 つのアプリケーションが、複数のモデルに対するタスクを行える
     ⇒ 但し、デバイスに負荷が掛からないよう、同時に複数の FL Plan が実行されることはない

  2. Federated Learning には匿名で参加する
     ⇒ 但し、ユーザの情報を FL Server に送信せずに、正規のユーザであることを確認する必要
     ⇒ モデルの学習に影響を与えようとする、不正なデバイスを防止するため

     - Android の SafetyNet Attestation API により実装
     - 正規のデバイスやアプリケーションのみが Federated Learning に参加する
     - 悪意のあるデータを学習データとして利用し、学習結果を恣意的に操作する Data Poisoning も防止

目次

サーバ上のソフトウェア

- サーバ上のソフトウェア
  - デバイス数や通信データ量は、学習しようとしている問題によって全く異なる
  - FL Server と通信する可能性のあるデバイス数は、FL Population に応じて様々に変化し得る
  - 数百台程度の小規模なものから、数億台程度の大規模な問題を扱わなくてはならない
  - 各 Round に参加するデバイスの台数も、数十から数千程度になり得る
  - 学習されるモデルのサイズ (パラメータ数) や、モデルの差分データのサイズも様々
  - 各 Round において、FL Server とデバイス間でやり取りされるデータ量も、数 KB から数十 MB の範囲を取り得る
  - 時間帯によってアクティブなデバイス数が変動するので、トラフィック量も変動

サーバ上のソフトウェア

- サーバ上のソフトウェア
  - サーバ上で動作するソフトウェアは、アクターモデルに則って実装されている
  - アクターモデルは、並列計算モデルの一種である
  - ソフトウェアの各コンポーネントは、全て Actor として扱う
  - Actor 間の通信手段は、(到着順の保証されない) メッセージパッシングのみである
  - 各 Actor は、到着したメッセージを順番に処理してゆく (内部状態が変化する)
  - メッセージを受信すると、内部状態の変更、他の Actor へのメッセージの送信、新たな Actor の生成などを行う

サーバ上のソフトウェア

- サーバ上のソフトウェア
  - 同じ種類の Actor を、複数のプロセッサやマシン上で並列に動作させれば、容易にスケールできる
  - Actor は、同一のプロセッサやプロセス上に共存してもよいし、複数のマシン上に分散させてもよい

  - アクターモデルは、スケーラビリティに優れる (スケールアップとスケールダウンが容易)
    $\Rightarrow$ デバイス数や通信データ量が大きく変動するため、スケーラビリティは必須の要件

  - FL Server への負荷に応じて、Actor の数を動的に変更させる

サーバ上のソフトウェア

- サーバ上のソフトウェア
  - Coordinator、Master Aggregator、Aggregator、Selector の 4 種類の Actor が連携して動作
  - 次の図 6 を参照

  - Coordinator と Selector は常に存在する Actor である
    $\Leftrightarrow$ Master Aggregator と Aggregator は、一時的に存在する (必要に応じて作られる)

サーバ上のソフトウェアのアーキテクチャ

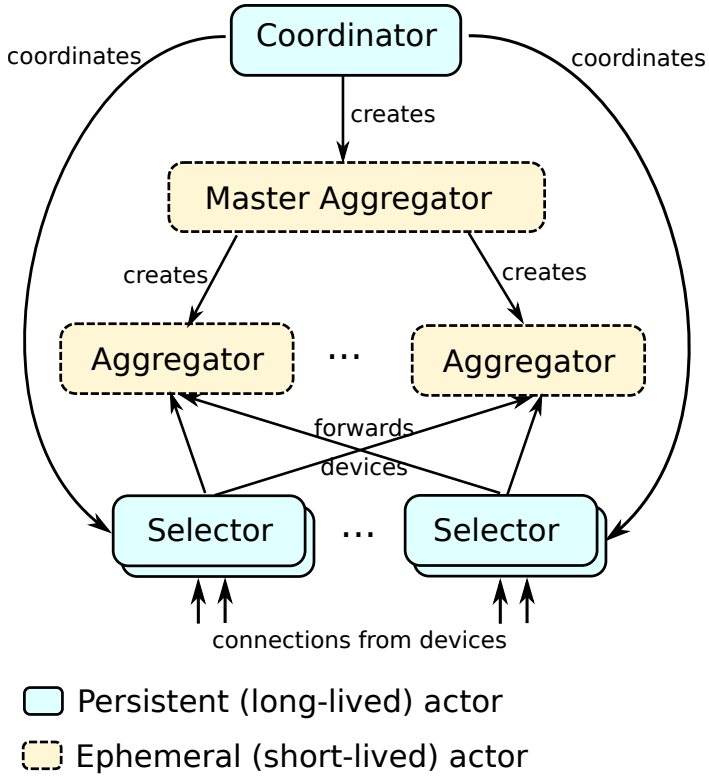

Figure 3: Actors in the FL Server Architecture

図 6: サーバ上のソフトウェアのアーキテクチャ [1]

サーバ上のソフトウェア

- Coordinator
  - 最上位に位置する Actor
  - Actor 全体の同期を取るほか、Round の進行を担う
  - Coordinator は複数存在し、各 Coordinator が、1 つの FL Population に対応する
  - 分散ロック機構によって、各 FL Population に対し、Coordinator が 1 つだけ存在することが保証される
    ⇐ 各 Coordinator が、管理対象の FL Population を登録するときに、分散ロックを使用

サーバ上のソフトウェア

- Coordinator と Master Aggregator の関係
  - Coordinator は、各 FL Task(の Round) に対して 1 つの Master Aggregator を生成
    ⇐ FL Task は、必ず 1 つの FL Population に属する
    ⇐ どの FL Task が実行されるかは、スケジューリングされている

- Master Aggregator
  - FL Task の各 Round を管理する Actor
  - スケーラビリティを確保するために、必要に応じた数の Aggregator を生成
    ⇐ モデルのサイズや、Round に参加するデバイスの台数に応じて決定

サーバ上のソフトウェア

- Coordinator と Selector の関係
  - Coordinator は、何台のデバイスが各 Selector に接続されているのかを取得
    ⇒ 各 Selector に対して、(Round に参加する) デバイスを何台選択すればよいか指示
  - 各 Selector は Coordinator から、各 FL Population に対して何台のデバイスが必要かを知る
    ⇒ この情報をもとに、各 Selector に現在接続しているデバイスの中から、何台を選択するか (Round に参加させるか) を決定する

サーバ上のソフトウェア

- Selector
  - 各デバイスとの接続を担当する Actor
  - 複数の Selector は、地理的に分散して存在できる
    ⇒ 各 Selector が、その地域のデバイスとの通信を担当する
    ⇒ Selector とデバイスとの距離が小さくなる

サーバ上のソフトウェア

- Selector (続き)
  - Coordinator は、各 FL Task に対して 1 つの Master Aggregator を生成
    ⇒ Master Aggregator は、必要に応じて幾つかの Aggregator を生成
    ⇒ その後、Selector は Aggregator に、選択された (Round に参加する) デバイスに関する情報を伝達
    ⇒ 以後、Aggregator が、Round に参加するデバイスを管理

  - Coordinator は、FL Task をデバイスに効率的に割り振ることが可能
    ⇒ 何台のデバイスがアクティブかということは、Coordinator には関係ない
    ⇒ FL Server はアクターモデルを採用することで、スケーラビリティを達成

サーバ上のソフトウェア

- FL Task の実行によって、モデルが更新されるまでの流れ
- Round の開始
  1. Coordinator が、Round に応じた Master Aggregator を 1 つ生成
  2. Master Aggregator は、必要に応じた数の Aggregator を生成
- Selection フェーズ
  1. Selector は Coordinator に対し、デバイスの接続台数を伝達
  2. Coordinator は Selector に対し、何台のデバイスを選択すべきか指示
  3. Selector が、Round に参加させるデバイスを、指定された分だけ選択し、Aggregator に伝える

サーバ上のソフトウェア

- Configuration フェーズ
  1. Aggregator は各デバイスに対し、FL Plan を配布してタスクの実行を指示
- Reporting フェーズ
  1. 各デバイスがタスクを実行し終えたら、Aggregator に結果 (差分データ) をアップロード
  2. Aggregator 上で結果を集約し、Master Aggregator に再度アップロード
  3. Master Aggregator 上で、各 Aggregator から送られてきた結果を集約して、最終的な結果を生成
  4. 最終的な結果 (差分データ) をモデルに適用する

サーバ上のソフトウェア

- システムの利点
  - 各 Actor は、内部状態をメモリ上に保持する
    ⇒ スケーラビリティの確保と、分散ストレージを使わないことによる低レイテンシの実現
  - ストレージには、各デバイスから送信された結果は保存しない
    ⇒ データセンタが攻撃されても、各デバイスに関するデータが漏洩する危険性がない
  - プロトコルのパイプライン化を達成
    ⇒ 他の Round が Configuration や Reporting フェーズであるときも、Selector は次の Round に参加するデバイスを募集できる
    ⇒ Selection フェーズは、以前の Round とは何の関係もないため
    ⇒ アクターモデルの採用によって得られる利点

サーバ上のソフトウェア

- クラッシュした場合のシステムの動作
  - 一部の Actor がクラッシュしても、システムは動作を継続できる
  - Aggregator や Selector がクラッシュすると、それらが管理していたデバイスだけが、システムからは消失
    ⇒ 他の Aggregator や Selector と通信するデバイスには、影響がない
  - Master Aggregator がクラッシュすると、管理されていた Round も消失
    ⇒ その Round の実行は失敗する
    ⇒ その後 Coordinator によって、Round を再度実行するための、Master Aggregator が生成される
  - Coordinator がクラッシュすると、Selector がそのことを検知して、再度生成する
    ⇒ Coordinator に関する情報は、分散ロックによって管理されている
    ⇒ 同一の FL Population に対応する Coordinator が、複数個生成されることはない

目次

アナリティクス

- 動作状況の把握のため、次のようなデータを解析した
  - Federated Learning が開始したときのデバイスの状態 (使用中かどうかなど)
  - Federated Learning の実行頻度と、継続時間
  - メモリの使用量
  - Federated Learning の実行中に発生したエラー
  - スマートフォンの製品データ、OS、FL Runtime のバージョン

  - これらの情報には個人を特定できるものは含まれない (本当か?)

- サーバ側では次のようなデータを解析した
  - Round に参加したデバイスの台数、リジェクトされたデバイスの台数
  - Round の各フェーズ (Selection、Configuration、Reporting) の実行時間
  - データのアップロードやダウンロード時間、発生したエラー

アナリティクス

- 発生した問題の種類
  - 適切ではないタイミングで Federated Learning が開始された
    ⇐ デバイスが充電中でない、アイドル状態でない、Wi-Fi に接続されていないなど
    ⇐ この類の問題は、ユーザがデバイスを使えなくなるので、発生させてはならない
  - Round に参加できないデバイスの台数が、予想を遥かに上回ることがあった
    ⇐ ユーザのデバイスに直接影響が及ぶことはない
  - ユーザのデバイスの性能に影響を及ぼしているかどうかを、検出するのは困難である

目次

Secure Aggregation

- Secure Aggregation の概要
  - Federated Learning のセキュリティを向上させるための手法
  - 各 Round の Reporting フェーズで利用することが可能

  - モデルの更新を繰り返して得られる、複数の差分データを集約する
    ⇐ 各デバイスの、個々の差分データは隠蔽される

  - Honest but Curious タイプの攻撃者を防止
    ⇐ プロトコルに沿った動きをするが、プロトコル上でやり取りされる
    メッセージから、様々な情報を抜き取ろうとする攻撃者

  - Prepare、Commit、Finalization フェーズの 3 つに分かれる

Secure Aggregation

- Prepare フェーズ
  - FL Server とデバイス間での暗号化された通信を確立
- Commit フェーズ
  - デバイスが、暗号によりマスクされたモデルの差分データを、FL Server にアップロード
  - FL Server は、これらのマスクされた差分データを集約
- Finalization フェーズ
  - FL Server が、デバイスから送信された差分データのマスクを外す
    ⇐ 各デバイスから、マスクを外すことの許可を受ける必要がある
    ⇐ 但し、デバイスは許可を与えなくてもよい

Secure Aggregation

- スケーラビリティとの兼ね合い
  - デバイスの数の二乗に比例して、計算コストが増大
    ⇒ 最大でも、数百台程度のデバイスでしか利用できない
    ⇒ Round に参加できるデバイスの台数が、制限されてしまう
  - Secure Aggregation を、各 Aggregator(Actor の一種) で動作させることで対処
    ⇒ 各 Aggregator が、それぞれに接続されたデバイスから、モデルの差分データを受信して、一つに集約 (ここで Secure Aggregation を使用)
    ⇒ 各 Aggregator が集約した差分データを、Master Aggregator が更に一つにまとめ上げる (Secure Aggregation は未使用)
  - Master Aggregator に計算負荷が集中しにくい
    ⇐ 複数の Aggregator が Secure Aggregation の処理を分担するため
  - Round に参加するデバイスの台数が制限されない

目次

モデル設計者の作業

- 中央 (クラウド上) にデータを集めて行う、モデルの学習とは大きく異なる
  - モデルの設計者は、個々の訓練データを直接見ることができない
    $\Leftarrow$ そこで、設計者側で、テストとシミュレーション用の、何らかの仮データを用意する必要がある
  - モデルを学習させるためには、一度 FL Plan を作成して、FL Server から各デバイスに配布しなければならない
    $\Leftarrow$ 設計者が、モデルを直接手元で試すだけでは、十分なテストができない
  - 学習はデバイス上で動作する
    $\Rightarrow$ デバイス上のリソース消費量や、デバイスとの互換性に配慮する必要がある

モデル設計者の作業

- モデル設計者の作業の流れ
  1. モデルの設計とシミュレーション
  2. デバイスにタスクを割り振るための FL Plan の作成
  3. バージョン管理、テスト、デプロイ

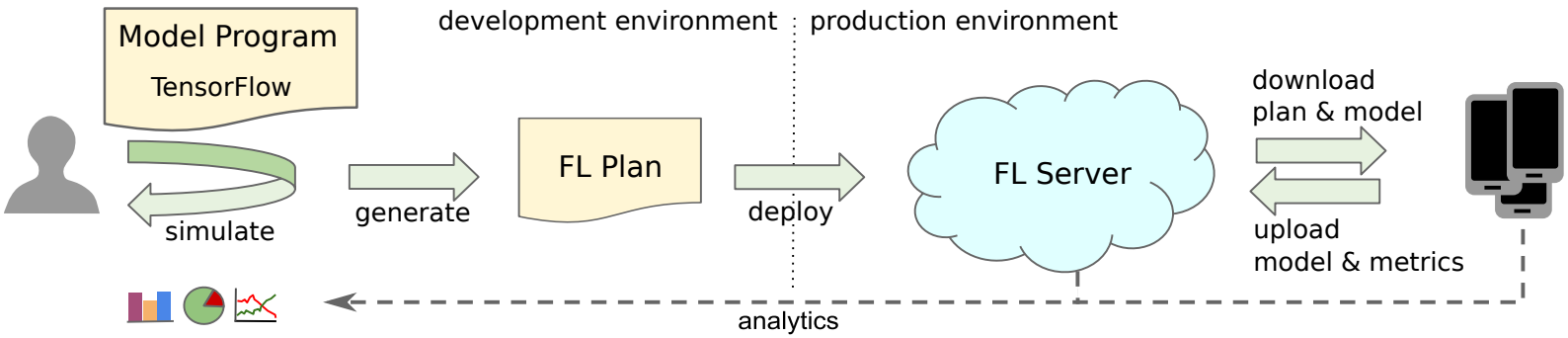

Figure 4: Model Engineer Workflow

図 7: モデル設計者の作業の流れ [1]

モデル設計者の作業

1. モデルの設計とシミュレーション
   - モデルの定義を行う
   - モデルの学習と評価を行うための FL Task を作成
     ⇐ TensorFlow ベースの専用のライブラリが用意されている
     ⇐ 入力のテンソルを、Loss や Accuracy 値に変換する計算グラフの構築
   - FL Task を、モデルの開発者側が用意したテストデータを使って評価
   - スケール可能な、シミュレーションツールを用いて行う
     ⇒ シミュレーションでは、同一のコードが実行されるほか、デバイスと FL Server との通信も完全に再現される
     ⇒ キーボードアプリであれば、ユーザの入力の代わりに、Wikipedia のコーパスなどをテストデータとする
   - シミュレーションで、モデルの事前学習を行うこともできる

モデル設計者の作業

- FL Task には、学習率などのハイパーパラメータも設定
- 複数の FL Task が 1 つのグループを形成してもよい
  ⇒ ハイパーパラメータのグリッドサーチなどが例
  ⇒ 1 つの FL Population に、複数の FL Task が含まれていてもよい

モデル設計者の作業

2 デバイスにタスクを割り振るための FL Plan の作成

- 設計者が用意した複数のモデルと、設定内容から、FL Plan は自動生成される

- FL Plan は 2 つの部分 (FL Server 用とデバイス用) に分けられる
  ⇒ Federated Learning 用のライブラリでは自動的に分割される

- デバイス用の部分には次のような情報が格納される
  ⇒ TensorFlow の計算グラフ
  ⇒ Example Store(データベース) から訓練データを選別する基準
  ⇒ 訓練データのバッチ化の手順
  ⇒ 実行するエポック数
  ⇒ モデルのパラメータの保存と読み込みのタイミング

- FL Server 用の部分には次のような情報が格納される
  ⇒ デバイスからの差分データを集約する手順

モデル設計者の作業

3 バージョン管理、テスト、デプロイ

- Federated Learning のシステムは、FL Plan のバージョン管理、テスト、デプロイを容易にしてくれる
  ⇒ Federated Learning は、様々なデバイス上で動作する
  ⇒ あるデバイス上ではメモリ使用量が大き過ぎるかもしれない
  ⇒ あるいは、(FL Server とデバイス上で動作する)TensorFlow のバージョンに、ずれ (非互換性) が生じるかもしれない

- TensorFlow のバージョンアップによって、計算グラフに非互換性が生じるかもしれない
  ⇒ デバイス上では、依然として古いバージョンの FL Runtime が動作している可能性
  ⇒ 同じノードであっても、計算の中身が変わっているかもしれない

モデル設計者の作業

- FL Plan にはバージョン情報が付加されている
  ⇒ ある 1 つの FL Task から、バージョンの異なる複数の FL Plan が生成される
  ⇒ バージョンが異なっても、(意味的に) 同一の計算が実行されるように、適切に計算グラフが書き換えられる
  ⇒ 同一のテストデータを使って、複数のバージョンの FL Plan がテストされる

- FL Plan が、FL Server で受け入れられるための条件
  ⇒ FL Plan に含まれるコードが、十分に検証されていること
  ⇒ FL Plan はテストデータを含み、そのテストにパスすること
  ⇒ FL Plan が消費するリソース量が、許容範囲内であること
  ⇒ FL Task がサポート対象とした、全ての TensorFlow のバージョンで、実際に動作すること

目次

アプリケーション

- Federated Learning が適している状況
  - 学習データがサーバ上ではなく、個々のデバイス上にある場合
    ⇐ 学習データが生成されるのが、サーバ上ではなくデバイス上であるような場合
  - 学習データが、プライバシーに関わるような場合
  - 学習データをサーバに送信するのが好ましくないか、不可能であるような場合
  - Federated Learning は、教師あり学習のタスクで主に使用されている
    ⇐ 教師データは、何らかのユーザの反応であることが多い
    ⇐ 例えば入力データは、アプリが提示した次の入力単語の候補であり、教師データは、ユーザが実際に入力した単語

アプリケーション

- Federated Learning が採用されたアプリケーションの例
  1. デバイス上でのアイテムのランク付け
     ⇒ 例えばスマートフォンの設定画面において、ユーザの検索単語から、設定内容の適切な候補を選び出して提示する機能

  2. キーボードアプリでのサジェスト機能
     ⇒ 入力された文章に応じた、様々なコンテンツのサジェスト機能
     ⇒ Gboard (Google Keyboard) というキーボードアプリで実装
     ⇒ 次の図 8 を参照

  3. 次の入力単語の予測
     ⇒ Gboard で実装された機能で、RNN を用いて次の入力単語を予測

アプリケーション

図 8: キーボードアプリでのサジェスト機能 [2]

アプリケーション

- 次の入力単語の予測
  - モデルは巨大であり、140 万のパラメータを持っていた
  - パラメータが収束するまでに、約 $3,000$ 回の Round が実行された
  - 5 日間にわたって、1.5×10^6 台のデバイス上で、6×10^8 個の文章が処理された (各 Round の実行時間は 2、3 分)
  - 1 番目に表示した単語の選択率が、13.0% から 16.4% に上昇
  - Federated Learning によって得られたモデルは、従来のモデルに比べて性能が良くなった
  - サーバ上にデータを集めて、モデルを学習させるのと比べて、パラメータが収束するのに約 7 倍の時間が掛かった

目次

実際の動作状況

- データの不正確さ
  - Federated Learning を用いたアプリケーションの数が少ない
  - 動作状況を調べるための統計データは、実際に動作するアプリケーションから得たものである
    ⇐ 精密な統計データを得るための環境は、用意されなかった

- スケーラビリティ
  - Federated Learning のシステムは、デバイスの台数や FL Population に応じて、適切にスケールする
  - 現在は、幾つかのアプリケーションと、約 1,000 万台のデバイスで、Federated Learning のシステムが動作している

実際の動作状況

- 接続されたデバイスの台数
  - デバイスは適切なときに起動されるほか、接続頻度も FL Population に応じて調節される
    ⇒ FL Server に同時に接続しているデバイスは、1 万台程度
  - Round に参加するデバイスの台数は、1 日の間で変動する
    ⇒ 夜間は、アイドル状態かつ充電中であることが多いため、Round に参加するデバイスが増加する傾向
    ⇒ 参加するデバイス数が、最も少ないときと、最も多いときの差は 4 倍

実際の動作状況

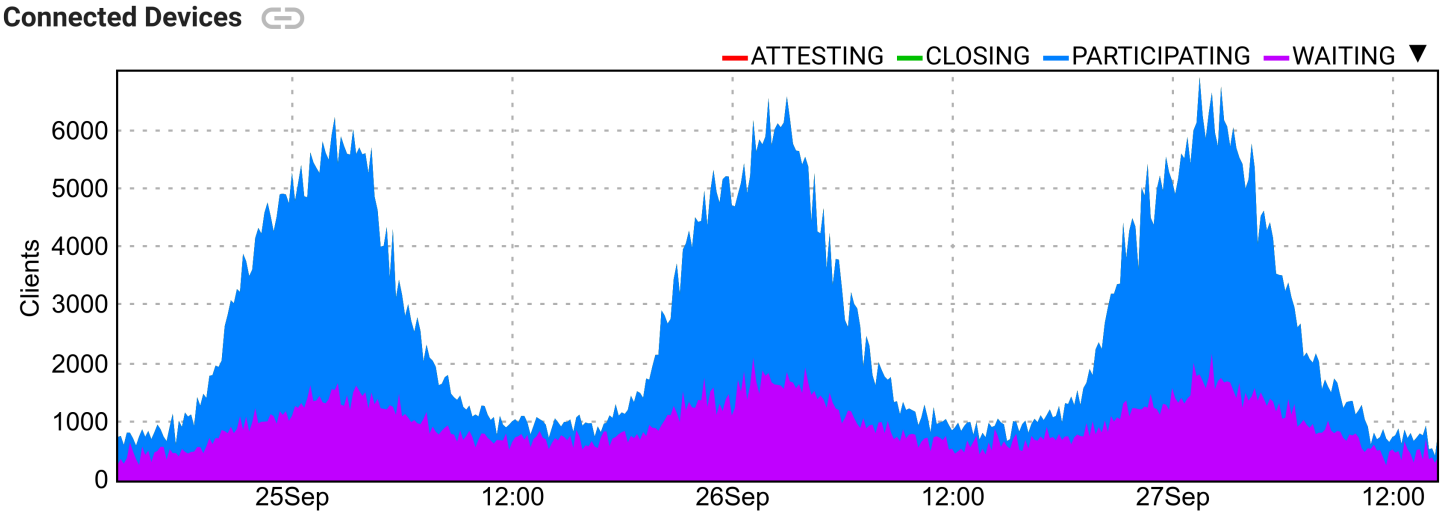

図 9: 接続されたデバイスの台数 [1]

実際の動作状況

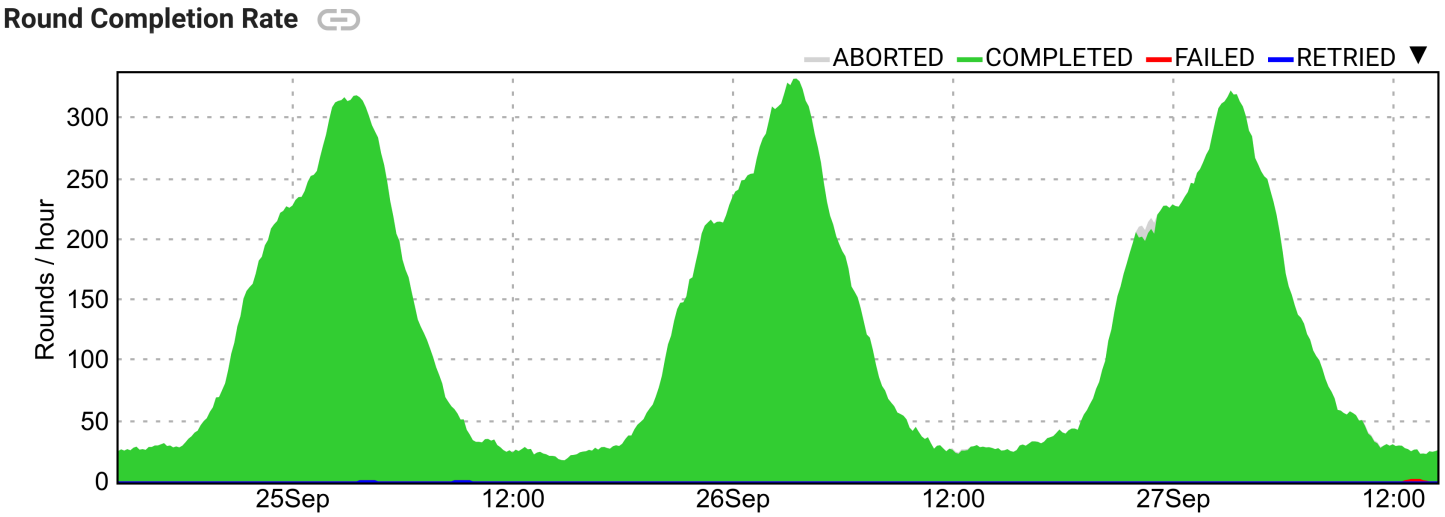

Figure 5: Round Completion Rate

図 10: 1 時間あたりの Round の実行回数 [1]

実際の動作状況

- Round に参加したデバイスの台数
  - 各 Round には数百台のデバイスが参加すれば、モデルの学習には十分であった
    $\Leftarrow$ それ以上の台数のデバイスが参加することの利点がなかった (パラメータの収束時間に変化がない)

  - ネットワークの問題、計算時のエラー、計算の中断 (ユーザがデバイスを使用したなど) によって、6% から 10% のデバイスが Round から欠落した
    $\Rightarrow$ FL Server は、デバイスの Round からの欠落を最初から見越して、実際よりも多めのデバイスを Round に参加させた
    $\Rightarrow$ 各 Round の Selection フェーズにおいて、1.3 倍の台数のデバイスを選択した
    $\Rightarrow$ 昼の方が、夜に比べて欠落するデバイスの割合が高くなった (昼の時間帯は、ユーザによってデバイスが使用される)

実際の動作状況

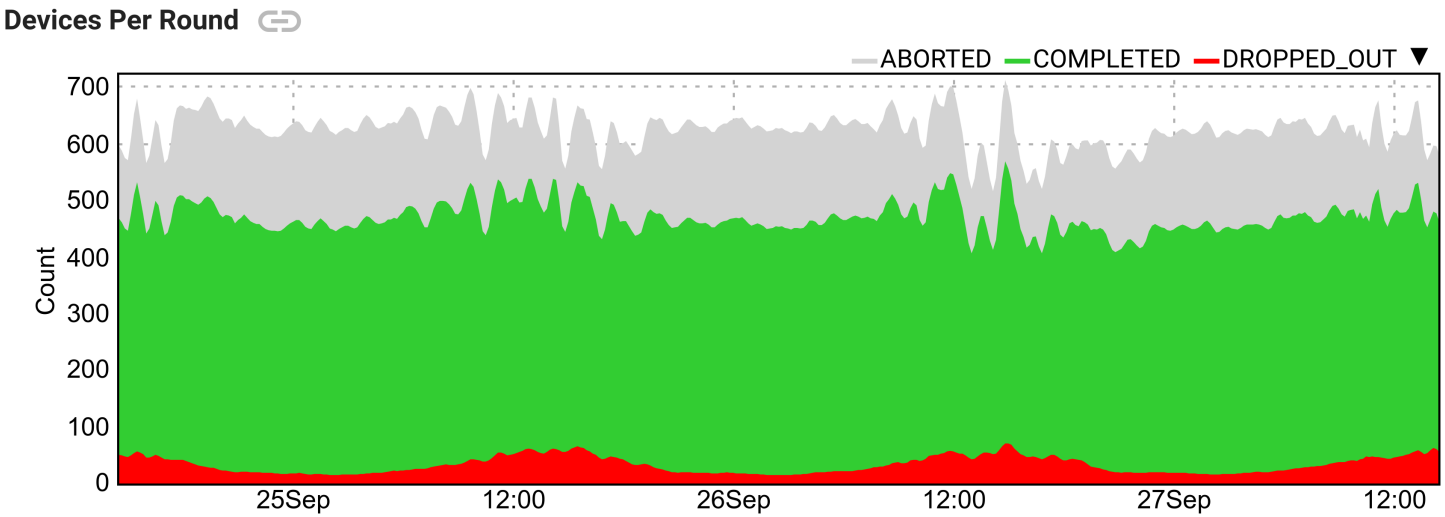

Figure 7: Average number of devices completed, aborted and dropped out from round execution

図 11: 各 Round に参加したデバイスの台数の平均 [1]

実際の動作状況

- Round の実行時間とデバイスの参加時間
  - 殆どのデバイスの Round への参加時間は、Round の実行時間と大体等しい
    ⇐ FL Server は、必要な数より少し多めのデバイスを選択する
    ⇐ そして、必要な数のデバイスから計算結果が返されたら、Round の実行が終了する
    ⇐ 従って、殆どのデバイスの Round への参加時間は、似たようなものになる
  - デバイスの Round への参加時間には上限がある
    ⇐ FL Server は、結果をなかなか返さない (もたついている) デバイスを単に無視する

実際の動作状況

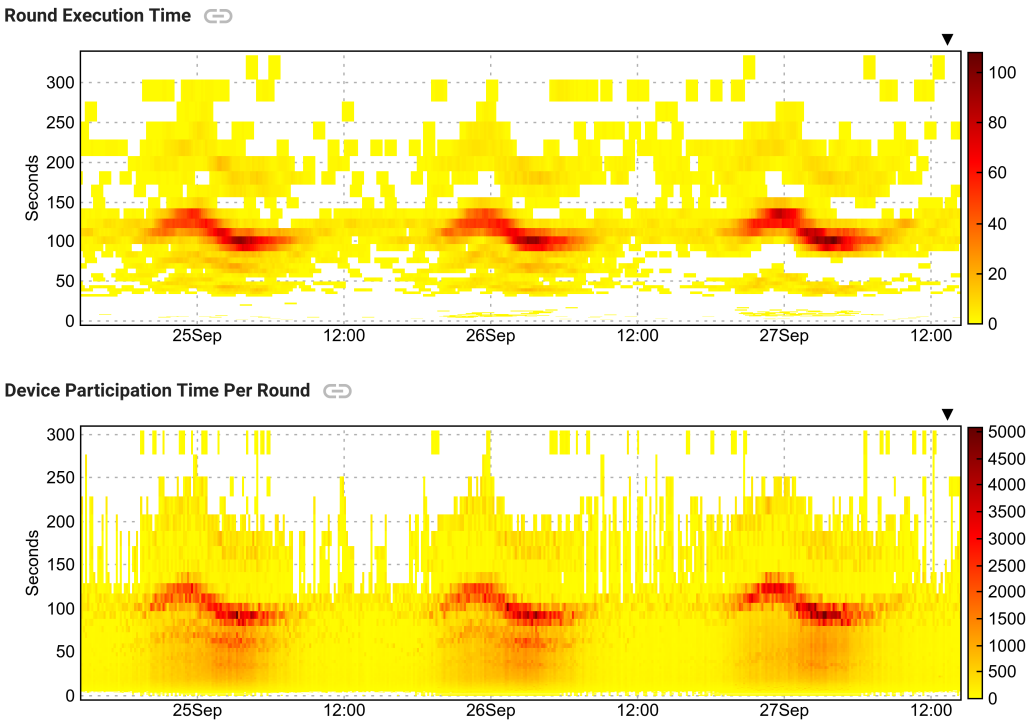

Figure 8: Round execution and device participation time

図 12: Round の実行時間とデバイスの参加時間 [1]

実際の動作状況

- ネットワークのトラフィック
  - サーバからのダウンロードの方が、サーバへのアップロードよりも多い
    ⇐ サーバからダウンロードするのは、FL Plan と、現在のモデル (いずれも同程度のサイズ)
    ⇐ サーバへアップロードされるのは、モデルに適用する差分データのみ
    ⇐ 差分データは、現在のモデルや FL Plan に比べて圧縮しやすい

実際の動作状況

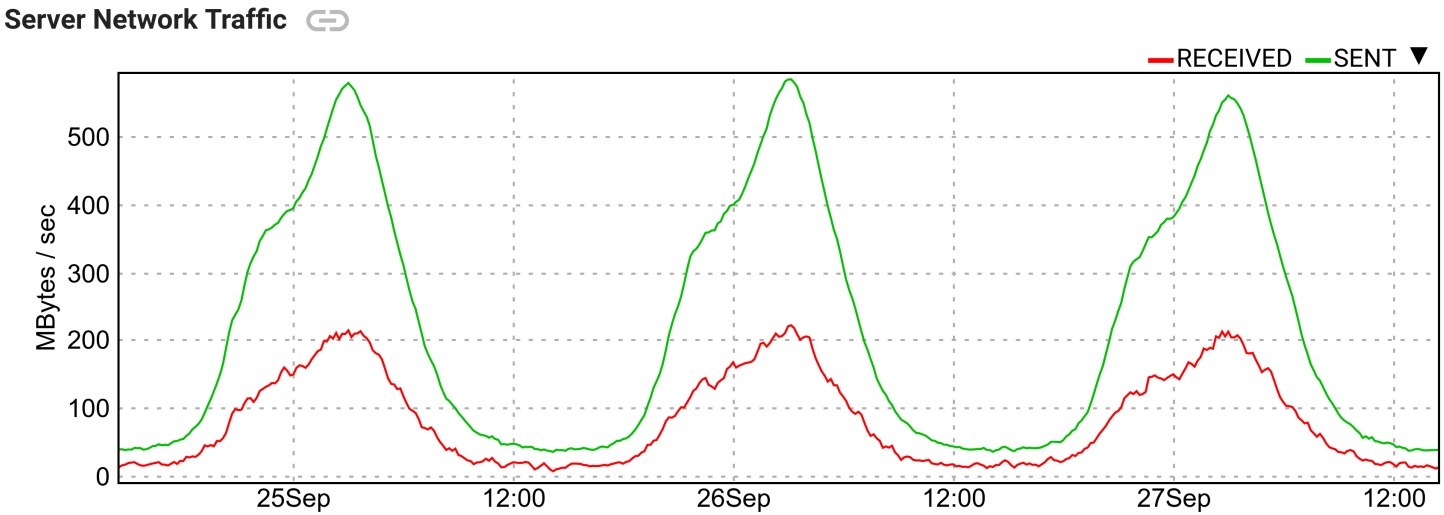

Figure 9: Server network traffic

図 13: ネットワークのトラフィック [1]

実際の動作状況

- デバイスでの学習状況
  - 75% のクライアントが、モデルを訓練し、サーバにアップロードできている
  - 22% のクライアントが、モデルの訓練は成功しているが、結果をサーバからリジェクトされている
    ⇐ 結果のアップロードまでに時間が掛かり、サーバから単に無視されている
  - 2% のクライアントが、モデルの訓練の途中で強制終了している
    ⇐ モデルの訓練途中に、ユーザがデバイスを使用し始めたことなどが原因

実際の動作状況

Session Shape	Count	Percent
-v[]+^	1,116,401	75%
-v[]+#	327,478	22%
-v[!	29,771	2%

Table 1: Distribution of on-device training round sessions. Legend: - = FL server checkin, v = downloaded plan, [= training started,] = training completed, + = upload started, ^ = upload completed, # = upload rejected, ! = interrupted.

目次

関連研究

1. デバイス上のデータを用いたモデルの学習手法
   - Pihur et al. (2018) で提案された手法
     ⇒ デバイス上で算出した差分データを集約する必要がない
     ⇒ プライバシーの対策も取られている
     ⇒ 一般化線形モデルに特化している

     ⇒ 非同期型のアルゴリズムであり、デバイスから送信された差分データをサーバに保存する必要がないので、スケーラブルであるとしている

     ⇔ Federated Learning は同期型であるが、スケーラブルであり、Federated Averaging などの幾つかの訓練アルゴリズムを使えば、デバイスからの送信データをオンラインに処理でき、サーバ上に保存する必要がない

   - Smith et al. (2017) や Kamp et al. (2018) で提案された手法
     ⇒ Federated Learning と、システムの設計が近い

関連研究

- Nishio & Yonetani (2018) による手法
  ⇒ Federated Learning と同様に、同期型のアルゴリズム
  ⇒ サーバが全てのデバイスにアクセスできるという前提
  ⇒ 各デバイスからの差分データは、同期的に受け取る
  ⇒ Federated Learning でも実装可能

- Federated Learning と似た概念は、以下のような分野で既に提唱されていた
  ⇒ 自動車同士の通信 [Samarakoon et al., (2018)]
  ⇒ 医療分野 [Brisimi et al., (2018)]

関連研究

2 分散機械学習 (Distributed Machine Learning)

- Federated Learning は、デバイスでの学習に特化したシステムである
  ⇒ データセンタのノードと比べると、通信速度が低く、また信頼性にも劣る

- 分散機械学習とは異なり、あらゆる分散処理の形態に対応しているわけではない
  ⇒ Federated Learning では、同期型のプロトコルが採用される
  ⇒ デバイス上での学習のために、最適化されている

- パラメータサーバは Federated Learning には取り込めない
  ⇒ パラメータサーバでは、モデルのパラメータに各ワーカーがアクセスし、非同期に更新していく
  ⇔ Federated Learning では、FL Server が、Round に参加するデバイスを選択する手続きがある (パラメータを同期的に更新するため)

関連研究

3 MapReduce [Dean & Ghemawat (2008)]

- MapReduce は、機械学習には適さないほか、根本的に異なる
  ⇐ MapReduce と Federated Learning との、枠組みは似ている
  ⇐ FL Server が Reducer で、各デバイスが Mapper と対応付けられる

- MapReduce とは前提条件が異なる
  ⇒ 各デバイスがデータを保持しており、完全に独立して動く Actor である
  ⇒ FL Server は、適切な (eligible) デバイスを選択するほか、選択されたデバイスの一部から送信されてきた結果を集約する (選択されたデバイスの全てが結果を送信するわけではない)

  ⇒ FL Server は、多くのデバイスが計算を中断すること、デバイスの可用性が極端に変動することを考慮して動作

  ⇒ MapReduce ではなく、これらの独特の環境に合致したフレームワークを採用すべき

目次

今後の課題

1. バイアスへの対処
   - Federated Learning では、全てのデバイスが均等に参加し、学習を行うことが前提
     ⇒ 学習されるモデルが、バイアスを生む可能性がある
   - デバイスは、Wi-Fi に接続され、かつ充電されているときにのみ Federated Learning に参加
     ⇒ 幾つかの国では、依然として Wi-Fi が普及していない
   - 今回のシステムでは、Android で、メモリを 2GB 以上搭載した、高性能のデバイスのみ参加
     ⇒ これもバイアスを生む原因となる可能性

今後の課題

- 現在のシステムでもバイアスの発生にはある程度対処
  $\Rightarrow$ デバイス上での学習を実行後、アプリケーションに応じた複数の指標を使って、リアルタイムでモデルを評価
  $\Rightarrow$ デバイス上での学習が、モデルの性能に悪影響を与えていないかどうか検出

- 現時点ではバイアスによる影響は生じていない
  $\Leftarrow$ 但し、アプリケーションによって、バイアスがどの程度出現するのかは分からない
  $\Leftarrow$ 様々なアプリケーション上での評価や、バイアスの影響を軽減するようなアルゴリズムの開発が、今後も必要

今後の課題

2 パラメータの収束速度

- データセンタでの学習と比べて、パラメータの収束が遅い

- 現在の Federated Learning では、数百のデバイスが Round に参加して、モデルの学習を実行
  ⇒ 実際には沢山のデバイスが、モデルの学習を実行できる状態にある
  ⇒ アルゴリズムの改良によって、より多くのデバイスが Round に参加できるようにする

- 各 Round の Selection フェーズや Reporting フェーズの時間は、現在は静的に決定される
  ⇒ 動的に決定することで、脱落するデバイスの割合や、単位時間あたりの Round の実行数を改善できる可能性がある
  ⇒ オンラインの機械学習アルゴリズムによって、パラメータを学習する方法などが考えられる

今後の課題

3 スケジューリングの改善

- デバイス上では、複数の FL Task を同時に扱うことが可能 (1 つの FL Task のみが実行される)
  ⇒ FL Task の実行順に関しては特に考慮されていない (キューに入った順番で決定)
  ⇒ ユーザが、どのアプリケーションを頻繁に使用するかなどを考慮できる

- FL Task が、古いデータを使って何度も学習を実行する、あるいは新しいデータを学習に用いないといった処理も考えられる

今後の課題

4 バンド幅

- モデルが RNN であれば、学習データに対して、パラメータの容量が大きくなるかもしれない
  ⇒ 単に学習データをサーバにアップロードして、学習を進めた方が、通信量の観点では有利かもしれない
  ⇒ これはユーザのプライバシーの確保とのトレードオフである

- サーバとデバイス間の通信量を削減するための手法が考えられる
  ⇒ モデルの圧縮、パラメータの量子化などが挙げられる
  ⇒ 通信量の削減と、モデルによる推論の効率化を同時に達成可能

今後の課題

5 Federated Learning の一般化 (Federated Computation)

- ここで紹介したシステムを応用できるアプリケーションは、まだまだある
- 特定の機械学習の手法に特化したものではない
  ⇐ FL Plan、モデル、差分データといった抽象化された用語で表現したに過ぎない
- Federated Learning の概念を、Federated Computation に拡張できる
  ⇒ TensorFlow を用いた機械学習に限らず、MapReduce のようなアプリケーションにも適用
- Federated Analytics を検討している
  ⇒ デバイス固有の統計情報をクラウドにアップロードすることなく、それらを集約したデータのみをアップロードする
  ⇒ プライバシーを保護しつつ、開発者は (集約された) デバイスの統計情報を得られる

参考文献

[1] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konecný, Stefano Mazzocchi, H. Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander.
Towards federated learning at scale: System design.
CoRR, abs/1902.01046, 2019.

[2] Brendan McMahan and Daniel Ramage.
Federated learning: Collaborative machine learning without centralized training data, Apr 2017.