Apache Arrow Flight

ビッグデータ用高速データ転送フレームワーク

須藤功平

株式会社クリアコード

db tech showcase 2021

2021-11-17

モチベーション

ビッグデータを
処理したい！

ビッグデータ処理に必要なもの

- ✓ 大量データ：データがないと始まらない！
- ✓ 速度：速くないと処理しきれない！

大量データの収集に必要なもの

- ✓ データ収集ツール
  - ✓ 例：ログ：Fluentd/Fluent Bit
- ✓ ストレージ
  - ✓ 例：Amazon S3
- ✓ 効率的なデータフォーマット
  - ✓ トレードオフ：空間効率と時間効率
  - ✓ 例：CSVよりApache Parquet

高速処理に必要なもの

- ✓ 高速な分散システム
  - ✓ 1台では処理しきれない
- ✓ 高速なアルゴリズムとその実装
  - ✓ 個々の処理が速くなると全体も速くなる

今日注目すること

高速な分散システム

高速な分散システムに必要なもの

- ✓ 効率のよいタスク管理
  - ✓ より速く処理が終わるようなリソース配分
- ✓ 効率のよいデータ転送
  - ✓ ノード間でのデータ交換コストは無視できない

大量データの交換コスト

Don't Hold My Data Hostage –
A Case For Client Protocol Redesign

Mark Raasveldt
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
m.raasveldt@cwi.nl

Hannes Mühleisen
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
hannes@cwi.nl

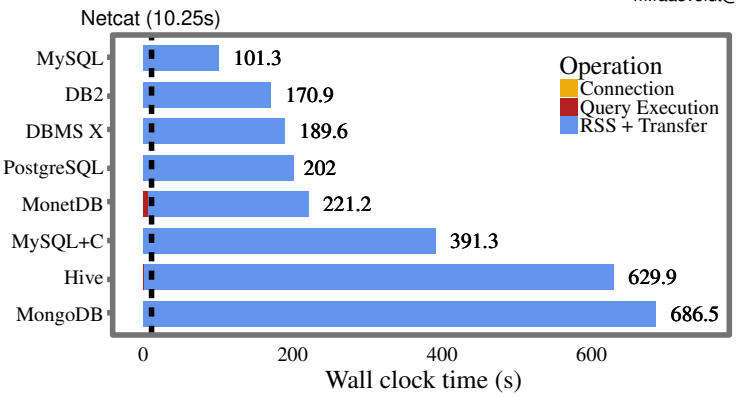


Figure 1: Wall clock time for retrieving the `lineitem` table (SF10) over a loopback connection. The dashed line is the wall clock time for netcat to transfer a CSV of the data.

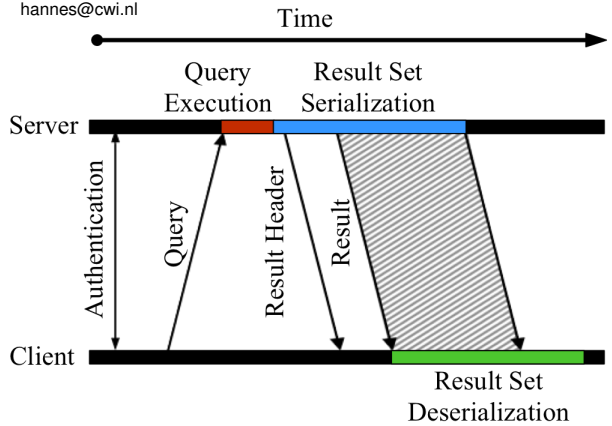
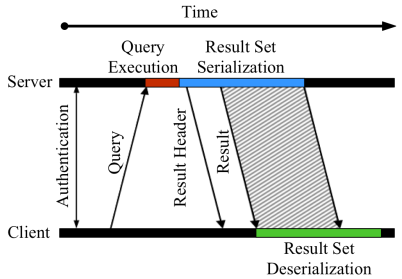


Figure 2: Communication between a client and a server

https://hannes.muehleisen.org/p852-muehleisen.pdf

大量データの交換コスト



- ボトルネックになりやすい
  a. シリアライズ・デシリアライズ
  b. ネットワーク帯域
- 目指すところ
  - メイン処理がボトルネック
    （メイン処理以外が十分速い）

解決策

Apache Arrow Flight

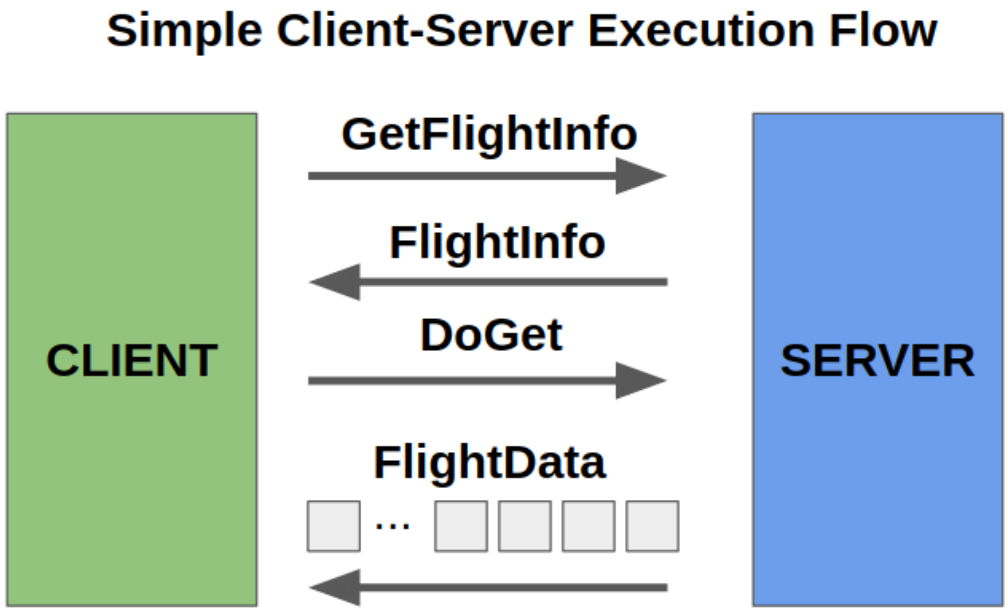
Apache Arrow Flightと私

- ✓ Apache Arrowプロジェクトの PMCメンバー
  - ✓ Apache Arrow Flightも開発しているプロジェクト
  - ✓ PMC：プロジェクト管理委員会
- ✓ コミット数2位

Apache Arrow Flight

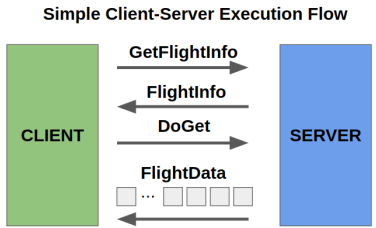
- ✓ gRPCベースのデータ転送フレームワーク
- ✓ ポイント
  - ✓ 並列転送対応
    - ✓「効率のよいタスク管理」に有用
  - ✓ ストリーム処理対応
    - ✓「効率のよいタスク管理」に有用
  - ✓ シリアライズ・デシリアライズがほぼ不要
    - ✓「効率のよいデータ転送」に有用

簡単な使い方



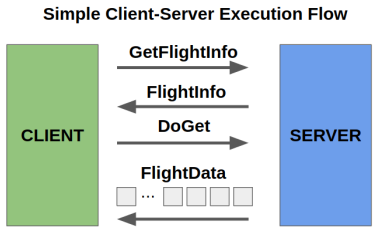
https://arrow.apache.org/img/20191014_flight_simple.png

GetFlightInfo



- ✓ クライアント→サーバー
- ✓ データの取得方法を教えてもらう
  - ✓ サーバーはFlightInfoを返す
- ✓ FlightInfoの中身
  - ✓ メタデータ：スキーマ・総レコード数…
  - ✓ 複数エンドポイント：
    データは複数ヶ所に分散しているかもしれない！

DoGet

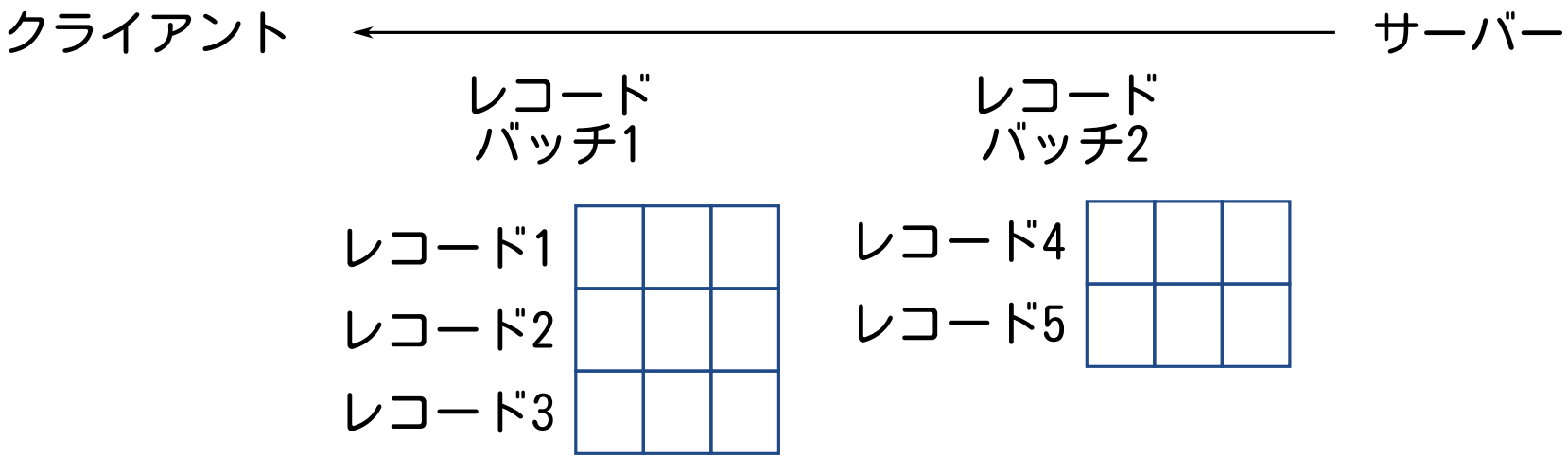


- ✓ クライアント→サーバー
- ✓ データを取得する
  - ✓ サーバーはレコードバッチをストリームで返す
  - ✓ プロトコルレベルではFlightDataと呼んでいる
- ✓ レコードバッチ
  - ✓ データ全体のうちの一部のレコードの集まり

Apache Arrow Flightが扱うデータ

- ✓ 型付きのテーブルデータ
  - ✓ RDBMSで扱うようなデータ
  - ✓ カラムごとに型がある
  - ✓ すべてのレコードは同じカラム構成
- ✓ レコードバッチ
  - ✓ テーブルデータ内の連続したレコードの集まり

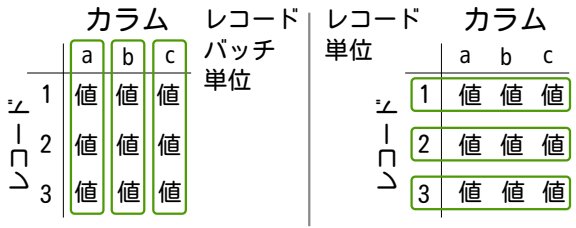
レコードバッチストリーム



なぜストリームが重要か

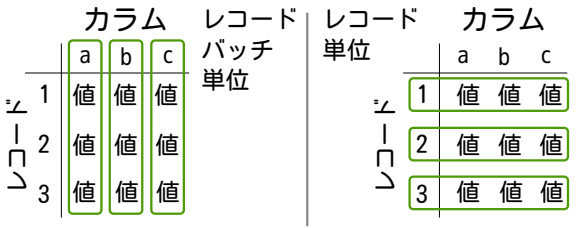
- ✓ 大量データを準備ができた順に処理できる
  - ✓ 全部準備ができるまで待たなくてよい
  - ✓ リソースを有効活用できる
- ✓ レコードバッチのストリームでいいの？
  - ✓ レコードのストリーム方がいいんじゃない？

ストリームの単位



- ✓ レコード
  - ✓ レコード単位で処理できる
  - ✓ データはレコード単位でまとめる
- ✓ レコードバッチ
  - ✓ 複数レコードをまとめて処理できる
  - ✓ データをカラム単位でまとめられる

レコードバッチと処理



- ✓ 複数レコードをまとめて処理
  - ✓ SIMDを活用すればレコード単位の処理より高速
- ✓ カラム単位でまとまったデータ
  - ✓ 分析処理が高速
  - ✓ ビッグデータ処理の多くは分析処理なはず

詳細

Apache Arrowフォーマットはなぜ速いのか

須藤功平
株式会社クリアコード

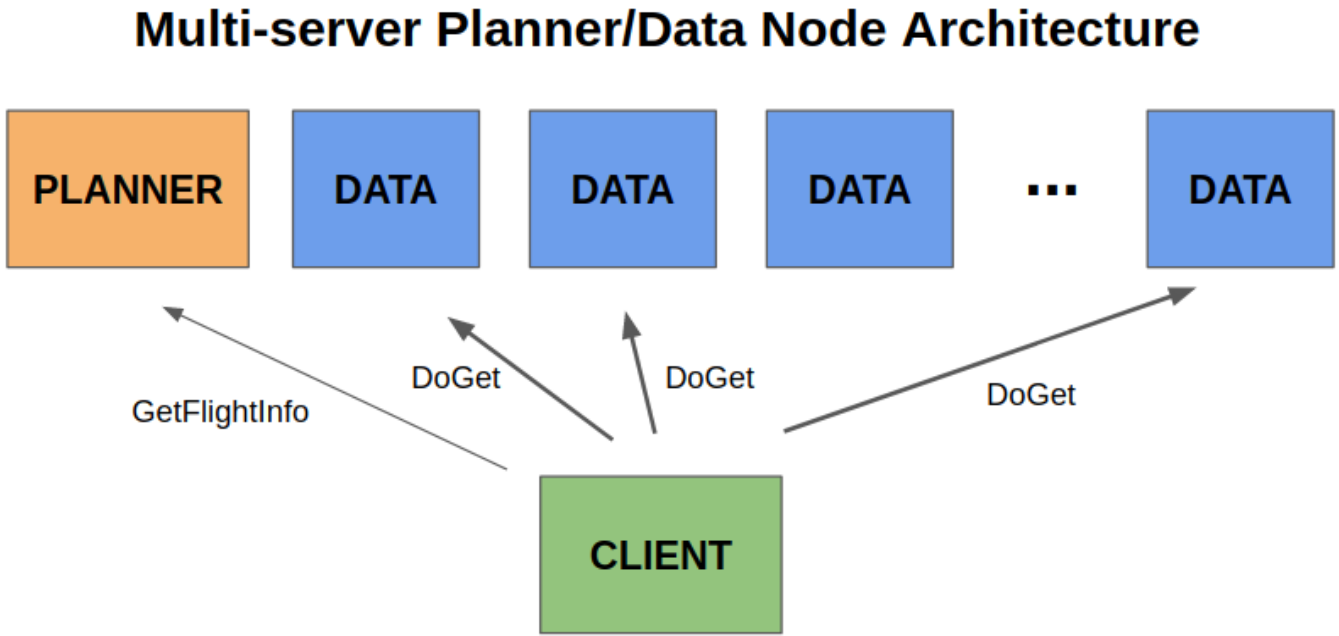
db tech showcase ONLINE 2020
2020-12-08

https://slide.rabbit-shocker.org/authors/kou/db-tech-showcase-online-2020/

高速な分散システムの実現方法

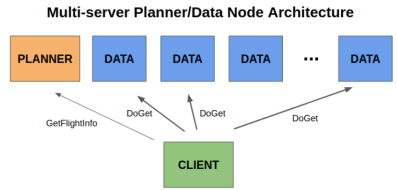
- ✓ レコードバッチのストリーム
  ここまでで説明したこと
  - ✓ 待ち時間を減らせる
  - ✓ 受け取ったデータを高速処理できる
- ✓ スケールアウト
  これから説明すること
  - ✓ データを複数ノードで分散処理
  - ✓ それを効率的に扱う

スケールアウト構成例



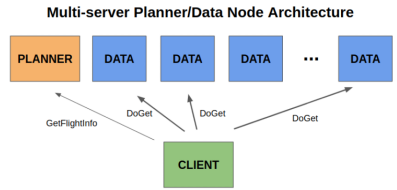
https://arrow.apache.org/img/20191014_flight_complex.png

ポイント1：ムダな転送を回避可能



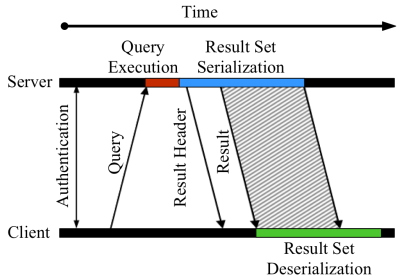
- ✓ 多くの分散システム
  - ✓ 「コーディネーター」経由で通信
  - ✓ ノード→コーディネーター→クライアント
- ✓ Apache Arrow Flight
  - ✓ 直接クライアントがデータ取得可能
  - ✓ ノード→クライアント

ポイント2：並列転送可能



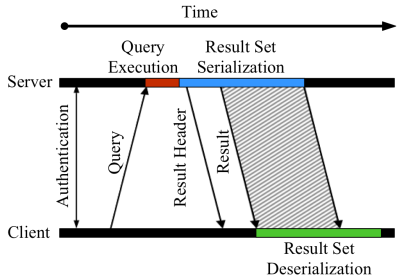
- ✓ 同時に複数ノードからデータ取得可能
  - ✓ データごとに異なるエンドポイントだから可能
- ✓ ストリームなので各データを随時処理可能

個々のデータ転送も速い



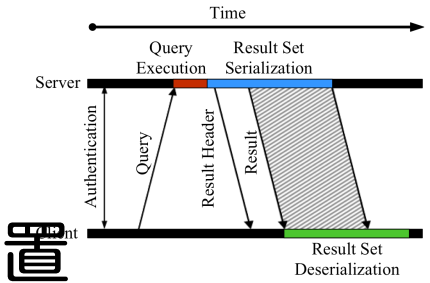
- ✓ 個々のデータ転送のボトルネック
  - ✓ シリアライズ・デシリアライズ
- ✓ どうすれば速くできる？
  - ✓ なにもしなきゃいいじゃん！

なにもしない？



- ✓ そもそもなぜシリアライズが必要？
  - ✓ メモリー上のデータの配置と通信時のデータの配置が違うから
- ✓ シリアライズしなくて済むには？
  - ✓ メモリー上のデータの配置と通信時のデータの配置を同じにすればいい！

Apache Arrowフォーマット



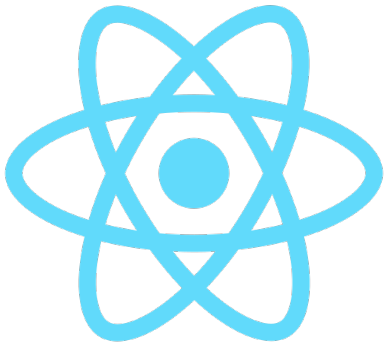
- ✓ シリアライズ不要フォーマット
  - ✓ メモリー上で効率よくデータを扱える配置
  - ✓ データ交換時も↑と同じ配置を使う
- ✓ Apache Arrow Flightが扱うデータは Apache Arrowフォーマット
  - ✓ 個々のデータ転送も速い！

Apache Arrow Flightのまとめ

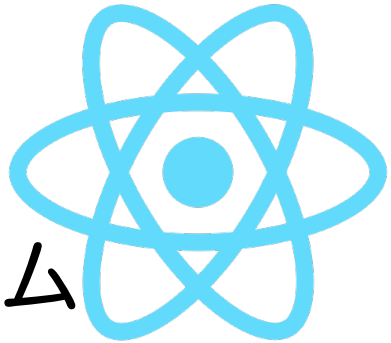
- ✓ 高速データ転送フレームワーク
  - ✓ 並列転送対応
  - ✓ ストリーム処理対応
  - ✓ シリアライズ・デシリアライズがほぼ不要
- ✓ gRPCベース
  - ✓ 既存のgRPCライブラリーでも接続可能
  - ✓ 専用ライブラリーあり：C, C++, C#, Go, Java, Python, R, Ruby, Rust

Apache Arrow Flightの利用事例

Apache Arrow Ballista

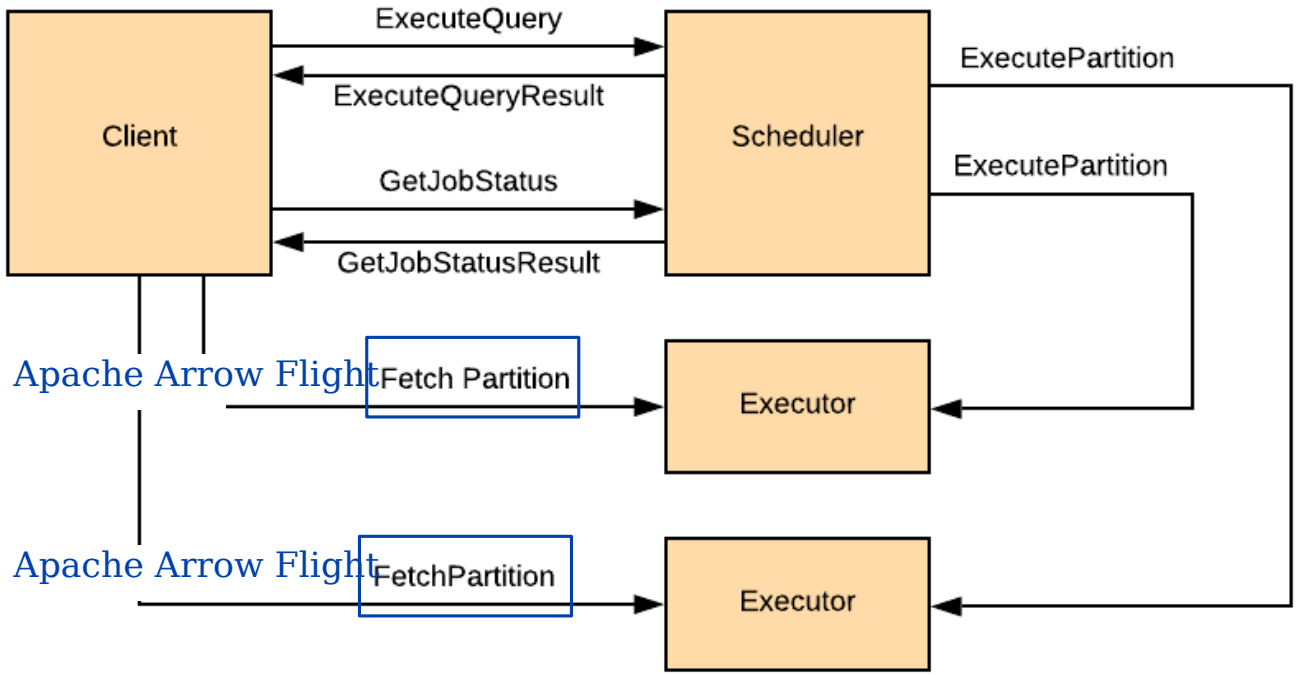


Apache Arrow Ballista

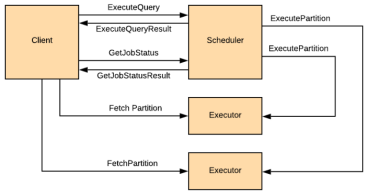


- ✓ Rust実装の分散計算プラットフォーム
- ✓ データはApache Arrowフォーマット
- ✓ 通信はgRPCとApache Arrow Flight

アーキテクチャー



Apache Arrow Ballistaでの使い方



- ✓ DoGetだけ使っている
  - ✓ 処理済みのパーティションを受け取る
  - ✓ 大量データになりうる
- ✓ その他の通信は素のgRPC
  - ✓ 各Executorへの処理の依頼など
  - ✓ 大量データにはならない

さらにApache Arrow Flight

- ✓ 利用可能なリクエスト
- ✓ ミドルウェア
- ✓ 将来の展望

利用可能なリクエスト1

- ✓ Handshake
  - ✓ 認証
- ✓ ListFlights
  - ✓ 利用可能なデータの取得方法一覧を取得
  - ✓ GetSchema/DoGetなどで使える
- ✓ GetFlightInfo
  - ✓ 指定したデータの取得方法を取得
  - ✓ GetSchema/DoGetなどで使える

利用可能なリクエスト2

- ✓ GetSchema
  - ✓ 指定したデータのスキーマを取得
- ✓ DoGet：指定したデータを取得
- ✓ DoPut：データを送信
- ✓ DoExchange：データを送受信

利用可能なリクエスト3

- ✓ DoAction
  - ✓ 任意の処理を実行
  - ✓ 普通のRPCなので素のgRPCでも十分
- ✓ ListActions
  - ✓ 利用可能なアクション一覧を取得

ミドルウェア

- プラグインみたいなもの
  - サーバー側・クライアント側両方あり
- 利用例
  - 認証機能
  - 分散トレーシング機能

将来の展望1

- gRPC以外もサポートするかも
  - https://issues.apache.org/jira/browse/ARROW-13889
  - メーリングリストで議論中
- 候補：UCX
  https://openucx.org/documentation/
  - RDMA/GPUなどハードウェアも活用した高速通信

将来の展望2

- Apache Arrow Flight SQLの追加
  - https://issues.apache.org/jira/browse/ARROW-9825
  - Apache Arrow Flight経由で各種RDBMSとやりとり
- GetFlightInfoでSQL送信
- DoGetで結果を受信

  - RDBMSとFlightアダプターの間のシリアライズがボトルネックになりそうな気がするんだけど、RDBMSのプロセス内にFlightアダプターを実装するのかな。。。

まとめ

- ✓ 大量データのやりとりは
  シリアライズ・デシリアライズが遅い
  - ✓ Apache Arrow Flightでそのコストをなくせる
- ✓ さらに高速に扱う仕組みもある
  - ✓ 並列転送やストリーム処理
  - ✓ 将来：UCXやFlight SQL

次のステップ

- ✓ もっと詳しく知りたくなったから イベント・社内・…で紹介して！
  - ✓ https://www.clear-code.com/contact/
- ✓ アンケートに答えてね！
  - ✓ ⇗のQRコードあるいは⇓ https://www.clear-code.com/surveys/db-tech-showcase-2021.html