Apache Arrow

データ処理ツールの次世代プラットフォーム

須藤功平

株式会社クリアコード

日本OSS推進フォーラム アプリケーション部会 第10回勉強会

2018-12-04

自己紹介：名前

- 須藤功平（すとうこうへい）
  「と」はにごらない！
- よく使うアカウント名：kou
  KOUhei
  - ↑を使えないときのアカウント名：ktou
    Kouhei suTOU

自己紹介：プログラミング

- ✓ Rubyが好き
  - ✓ 2004-01からコミッター
  - ✓ 130くらいのライブラリーをメンテナンス
    詳細はRubyKaigi 2018でのキーノートを参照
  - ✓ Rubyでプレゼンツールも作っている（このツール）
- ✓ C/C++を書いている時間も結構ある

自己紹介：C/C++を書く理由

- ✓ RubyでC/C++のライブラリーを使うため！
  Rubyを書くためにC/C++を書く
- ✓ Apache ArrowをC++で開発しているのもそう
  - ✓ テストはRubyで書いている

自己紹介：Apache Arrowの開発

- ✓ 2016-12-21に最初のコミット
- ✓ 2017-03-16にGLibバインディングを寄贈
- ✓ 2017-05-10にコミッター
- ✓ 2017-09-15にPMCメンバー
- ✓ 2018-12-02現在コミット数3位（221人中）

自己紹介：仕事

- ✓ 株式会社クリアコードの代表取締役
  - ✓ 自由なソフトウェアでビジネスをする会社
    自由なソフトウェア：OSSが参考にしたやつ
- ✓ 私の業務内容
  - ✓ Ruby/Groonga関連の開発・サポート
    Groonga：全文検索エンジン。10年くらいやっている。
  - ✓ 自由なソフトウェアの推進
  - ✓ データ処理ツールの開発事業立ち上げ（New!）

データ処理ツールの開発事業

- ✓ データ分析をする事業じゃない
- ✓ データ分析をする人たちが使うツールを開発する事業
  - ✓ Apache Arrowはそのために有用なツール
    Apache Arrowの開発に参加し始めたのはRubyで使いたかったから
    事業立ち上げのためにApache Arrowの開発に参加し始めたわけではない
    Apache Arrowの開発に参加していたら面白そうと思えてきた
  - ✓ 募集：開発して欲しい・開発したい（転職したい）

Apache Arrow

各種言語で使えるインメモリーデータ処理プラットフォーム

実現すること

データ処理の効率化
（大量データが対象）

効率化のポイント

- ✓ 速度
  - ✓ 速いほど効率的
- ✓ 実装コスト
  - ✓ 低いほど効率的

速度向上方法

- ✓ 遅い部分を速く
- ✓ 高速化できる部分を最適化

遅い部分

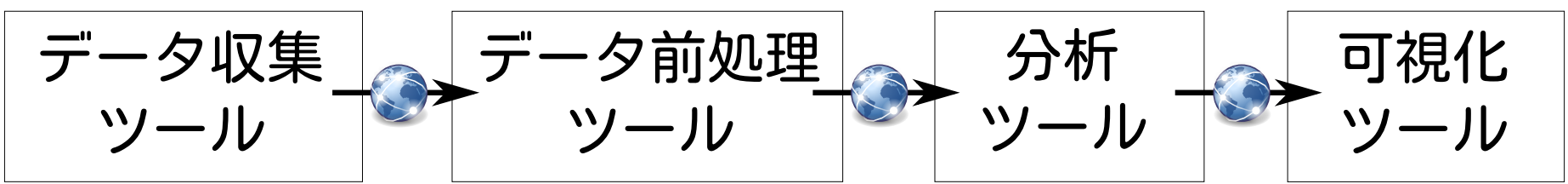
データ交換

データ交換

- ✓ データ処理ツール間で必要
- ✓ データ処理システム
  - ✓ 複数ツールを組み合わせて実現
  - ✓ データ処理システムではデータ交換が必須

データ処理システム例

- データ交換



データ交換処理

1. シリアライズ
   データをバイト列へ変換
2. 転送
   バイト列を別ツールに渡す
3. デシリアライズ
   バイト列からデータを復元

データ交換処理：必要なリソース

1. シリアライズ：CPU
2. 転送：I/O（ネットワーク・ストレージ・メモリー）
3. デシリアライズ：CPU

Ruby+JSONでデータ交換

```ruby
# 1000要素の数値配列
n = 1000
numbers = n.times.collect {rand}
# シリアライズ
JSON.dump(numbers, output)
# デシリアライズ
JSON.load(input)
```

Ruby+JSONの速度の傾向

n	シリアライズ	デシリアライズ
1000	0.011秒	0.004秒
10000	0.093秒	0.037秒
100000	0.798秒	0.369秒

注：ストレージI/Oなしで計測

データ量の増加と同じくらいの比率で遅くなる

データ交換の高速化

- ✓ データ量が増加すると
  シリアライズ・デシリアライズ速度が劣化
- ✓ 速度劣化を抑えられれば高速化可能

Apache Arrowのアプローチ

- ✓ データフォーマットを定義
  - ✓ シリアライズ・デシリアライズが速い
  - ✓ データ量増加に影響を受けにくい
- ✓ このフォーマットの普及
  - ✓ 各種言語で読み書き処理を実装

Ruby+Apache Arrowでデータ交換

```
# 1000要素の数値配列
n = 1000
numbers = Arrow::Int32Array.new(n.times.collect {rand})
table = Arrow::Table.new("number" => numbers)
# シリアライズ
table.save(output)
# デシリアライズ
Arrow::Table.load(input)
```

Ruby+Apache Arrowの速度の傾向

n	シリアライズ	デシリアライズ
1000	0.0003秒	0.0004秒
10000	0.0004秒	0.0004秒
100000	0.0015秒	0.0004秒

注：ストレージI/Oなしで計測

全体的に速い+デシリアライズ速度が一定

Apache Sparkでの高速化事例

- PySpark
  - Sparkが分割したデータをPythonで処理
- Spark⇔PySpark間でデータ交換
  - 従来：pickleでシリアライズ
    pickle：Python標準のシリアライズ方法
  - Apache Arrowを使うことで数十倍レベルの高速化

Apache Arrowフォーマットの特徴

- メモリー上でのフォーマットを変換しない
  - JSONは「数値」を「数字」に変換
  - 例：29（1バイト整数）→"29"（2バイト文字列）
- シリアライズ時：変換不要
- デシリアライズ時：パース不要

メモリーマップの活用

- メモリーマップ機能
  - ファイルの内容をメモリー上のデータのようにアクセスできる機能
  - readせずにデータを使える（データコピー不要）
- パース不要+メモリーマップ
  - デシリアライズ時にメモリー確保不要
  - 「転送」コスト削減

遅い部分の高速化まとめ

- ✓ 遅い部分を速く
  - ✓ データ交換を速く
- ✓ 高速化できる部分を最適化

高速化できる部分

大量データの計算

大量データの計算の高速化

- ✓ 各データの計算を高速化
- ✓ まとまったデータの計算を高速化

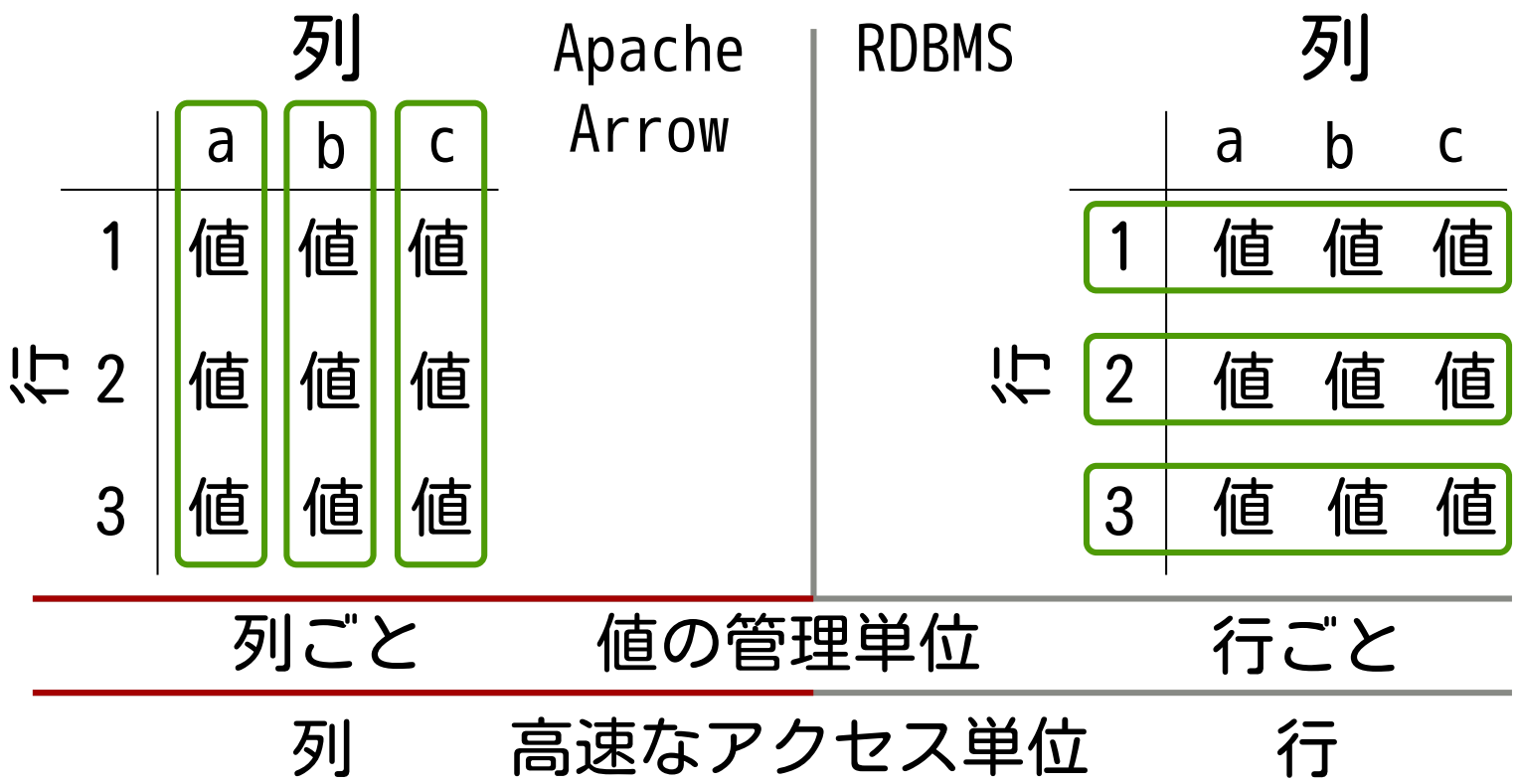
各データの計算の高速化

- ✓ データを局所化
  - ✓ CPUのキャッシュメモリーを活用できる
- ✓ 局所化に必要な知識
  - ✓ データの使われ方
  - ✓ 局所化：一緒に使うデータを近くに置く

想定ユースケース

- ✓ OLAP (OnLine Analytical Processing)
  - ✓ データから探索的に知見を探し出すような処理
- ✓ 列単位の処理が多い
  - ✓ 集計処理・グループ化・ソート…

OLAP向きのデータの持ち方



まとまったデータの計算を高速化

- ✓ SIMDを活用
  Single Instruction Multiple Data
  - ✓ CPU：データをまとめてアラインすると使える
    アライン：データの境界を64の倍数とかに揃える
  - ✓ GPUの活用
- ✓ スレッドを活用

スレッド活用時のポイント

- ✓ 競合リソースを作らない
  - ✓ リソースロックのオーバーヘッドで遅くなりがち
- ✓ アプローチ
  - ✓ リソースを参照するだけ
  - ✓ 各スレッドにコピー

Apache Arrowとスレッド

- ✓ データはリードオンリー
  - ✓ スレッド間でオーバーヘッドなしで共有可能
- ✓ データコピーは極力避けたい
  - ✓ データ交換時もスレッド活用時も

高速化のまとめ

- ✓ 速度
  - ✓ 遅い処理（データ交換処理）を高速化
  - ✓ 速くできる処理（大量データの計算）を最適化
- ✓ 実装コスト
  - ✓ 低いほど効率的

実装コストを下げる

- ✓ 共通で使いそうな機能をライブラリー化
  - ✓ メリットを受ける人たちみんなで協力して開発
  - ✓ 最適化もがんばる
- ✓ Apache Arrowの実装コストは下がらない
  - ✓ Apache Arrowを使うツールの実装コストが下がる
    実装コストが下がる：ツール開発者のメリット

今のApache Arrowが提供する機能

- ✓ 高速なデータフォーマット
- ✓ フォーマット変換機能
- ✓ 効率的なデータ交換処理
- ✓ 高速なデータ処理ロジック

これらの機能が必要ならメリットがある

Apache Arrowの向き不向き

- ✓ 向き
  - ✓ 大量データの交換
  - ✓ メモリー上での大量データの分析処理
- ✓ 不向き
  - ✓ データの永続化
    処理結果の一時的なキャッシュならアリなケースもある
  - ✓ OLAPシステムのバックエンド

実装コストのまとめ

- ✓ 速度
- ✓ 実装コスト
  - ✓ 共通で使いそうな機能をライブラリー化
  - ✓ みんなで協力して開発
  - ✓ 向いていないケースもあるけど
    メリットがある人は一緒に開発しよう！

Apache Arrowが扱えるデータ

- ✓ データフレーム
- ✓ 多次元配列

データフレーム

- 行と列で管理する2次元データ
  RDBMSでいう表
  - すべての行は同じ列を持つ
  - 各列は異なる型にできる
  - すべての型でnullをサポート
- 2次元配列との違い
  - 2次元配列はすべての値が同じ型

扱える型：真偽値・数値

- ✓ 真偽値
- ✓ 整数（{8, 16, 32, 64}ビット{非負, }整数）
- ✓ 浮動小数点数（{16, 32, 64}ビット）
- ✓ 128ビット小数

リトルエンディアンのみ対応

ビッグエンディアンに対応させようという動きもある：ARROW-3476

扱える型：文字列・バイト列

- ✓ UTF-8文字列
- ✓ バイナリーデータ（{可変, 固定}長）

扱える型：日付・タイムスタンプ

- 日付
  - UNIXエポックからの経過日数（32bit）
  - UNIXエポックからの経過ミリ秒数（64bit）
- タイムスタンプ（64ビット整数）
  - UNIXエポックからの経過{,ミリ}秒数
  - UNIXエポックからの経過{マイクロ,ナノ}秒数

扱える型：時間

- 時間
  - 深夜0時からの経過{, ミリ}秒数
    （32bit整数）
  - 深夜0時からの経過{マイクロ, ナノ}秒数
    （64bit整数）

扱える型：リスト

- ✓ 0個以上の同じ型の値を持つ
- ✓ 例：32ビット整数のリスト
  - ✓ 👍0要素：[]
  - ✓ 👍2要素：[2, 3]
  - ✓ 👎型が違う：[1, "X"]

扱える型：構造体

- ✓ 1個以上のフィールドを持つ
- ✓ 各フィールドは別の型にできる
- ✓ 例：aは32ビット整数、bはUTF-8文字列
  - ✓ 👍全部ある：`{a: 1, b: "X"}`
  - ✓ 👍nullもOK：`{a: 1, b: null}`
  - ✓ 👎bがない：`{a: 1}`

扱える型：共用体

- ✓ 1個以上のフィールドを持つ
- ✓ 各フィールドは別の型にできる
- ✓ どれかひとつのフィールドの値のみ設定
- ✓ 例：aは32ビット整数、bはUTF-8文字列
  - ✓ 👍aだけある：`{a: 1}`
  - ✓ 👎2つある：`{a: 1, b: "X"}`

扱える型：辞書

- 名義尺度なカテゴリーデータ
  （統計っぽい説明）
- 各値にIDを割り当て、そのIDで値を表現
  （実装よりの説明）
- 例：["X","X","Y"]を辞書型の列にした場合
  - 値：[0, 0, 1]
  - IDの割り当て：{"X": 0, "Y": 1}

データフレームのまとめ

- ✓ データフレーム
  - ✓ 2次元データ
  - ✓ 各列は異なる型にできる
  - ✓ 型はたくさんある
- ✓ 多次元配列

多次元配列

- ✓ n次元のデータを扱う
- ✓ 要素はすべて同じ型
- ✓ 型は整数・浮動小数点数のみ
- ✓ 密・疎ともに対応
  疎は対応中：ARROW-854 [Format] Support sparse tensor

密な多次元配列

- データの並び方
  - row-major order: C order
  - column-major order: Fortran order

疎な多次元配列

- データの持ち方
  - COO: 0以外の値の情報だけを持つ
  - CSR: 0以外の値の情報だけを圧縮して持つ

多次元配列のまとめ

- ✓ データフレーム
- ✓ 多次元配列
  - ✓ n次元データ
  - ✓ すべての要素は同じ型の値
  - ✓ 型は整数・浮動小数点数のみ
  - ✓ 疎・密両方サポート

Apache Arrowが提供する機能

- ✓ 高速なデータフォーマット
- ✓ フォーマット変換機能
- ✓ 効率的なデータ交換処理
- ✓ 高速なデータ処理ロジック

フォーマット変換機能

- Apache Arrowフォーマット
  - インメモリー用のフォーマット
  - 永続化向きではない
- 永続化されたデータを使う場合
  - 永続化に適したフォーマットで保存
  - 読み込み時にApache Arrowに変換して
    インメモリーでの処理にApache Arrowを使う

対応フォーマット：CSV

- よく使われているフォーマット
  - 亜種が多くてパースが大変

対応フォーマット：Apache Parquet

- ✓ 永続化用フォーマット
  - ✓ 列単位でデータ保存：Apache Arrowと相性がよい
- ✓ 小さい
  - ✓ 列単位の圧縮をサポート
- ✓ 速い
  - ✓ 必要な部分のみ読み込める（I/Oが減る）

対応フォーマット：Apache ORC

- 永続化用フォーマット
  - 列単位でデータ保存：Apache Arrowと相性がよい
  - Apache Parquetに似ている
- Apache Hive用に開発
  - 今はHadoopやSparkでも使える

対応フォーマット：Feather

- 永続化用フォーマット
  - 列単位でデータ保存：Apache Arrowと相性がよい
  - データフレームを保存
- PythonとR間のデータ交換用
- **今は非推奨！**
  - Apache Arrowを使ってね

対応中フォーマット：Apache Avro

- RPCフレームワーク
  - データフォーマットも提供
- 永続化にも使えるフォーマット
- ARROW-1209
  - ちょっと止まっている

非公式対応フォーマット：MDS

- Multiple-Dimension-Spread
  - Yahoo! Japan開発フォーマット
- 詳細：Apache Arrow東京ミートアップ2018
  - 2018年12月8日（土）開催

フォーマット変換機能まとめ

- ✓ 高速なデータフォーマット
- ✓ フォーマット変換機能
  - ✓ いろいろ対応している
  - ✓ これからも増えそう
- ✓ 効率的なデータ交換処理
- ✓ 高速なデータ処理ロジック

効率的なデータ交換処理

- Plasma
  - 同一ホスト上のプロセス間でのデータ共有
- Apache Arrow Flight
  - Apache ArrowベースのRPC
- DB連携
  - 各種DBのレスポンスをApache Arrowに変換

Plasma

- 同一ホストでのデータ共有システム
  - サーバー：データ管理
  - クライアント：データ登録・参照
- Apache Arrowフォーマットは使っていない
  - 任意のバイト列を共有
- ユースケース：マルチプロセス連携
  - もともとRay（分散計算システム）用に開発

Apache Arrow Flight

- Apache ArrowベースのRPCフレームワーク
  - クライント：リクエストデータフレームを送信
  - サーバー：レスポンスデータフレームを返信
- gRPCベース

DB連携

- DBのレスポンスをApache Arrowに変換
- 対応済み
  - Apache Hive, Apache Impala
- 対応予定
  - MySQL/MariaDB, PostgreSQL, SQLite
  - SQL Server, ClickHouse

効率的なデータ交換処理のまとめ

- ✓ 高速なデータフォーマット
- ✓ フォーマット変換機能
- ✓ 効率的なデータ交換処理
  - ✓ 同一ホスト内での高速なデータ交換
  - ✓ 異なるホスト間での高速なデータフレーム交換
  - ✓ DBのレスポンスをApache Arrowに変換
- ✓ 高速なデータ処理ロジック

高速なデータ処理ロジック

- ✓ 高速なデータフレーム処理
  - ✓ CPU：コンパイラーの最適化でベクトル化
  - ✓ GPU：RAPIDSが[cuDF]を開発
- ✓ 高速なクエリー処理エンジン
  - ✓ Gandiva

Gandiva

- ✓ SQLレベルのクエリーの実行エンジン
  - ✓ 四則演算だけではない
  - ✓ 集計（GROUP BY）やフィルター（WHERE）などもカバー
- ✓ 実行時に最適化

Gandiva：実行時に最適化

- クエリーを解析して最適化
  - 不要な処理を削除・処理順番を入れ替え
    やるようになるんじゃないかな。。。
- クエリーをJITコンパイルして実行
  - インタプリターが実行、ではない
  - 最適化済みネイティブコードにして実行
    実行環境のCPUに合わせてベクトル化とか

高速なデータ処理ロジックのまとめ

- ✓ 高速なデータフォーマット
- ✓ フォーマット変換機能
- ✓ 効率的なデータ交換処理
- ✓ 高速なデータ処理ロジック
  - ✓ CPUでもGPUでも最適化
  - ✓ クエリーレベルの高速な実行エンジン

対応言語

- ✓ C, C#, C++, Go, Java, JavaScript, Lua
- ✓ MATLAB, Python, R, Ruby, Rust

非公式実装：

- ✓ Julia (Arrow.jl)

実装方法

- ネイティブ実装
  - C#, C++, Go, Java, JavaScript, Julia, Rust
- C++バインディング
  - C, Lua, MATLAB, Python, R, Ruby

C#の実装状況

- ✓ 未対応の型
  - ✓ 16bit浮動小数点数・小数・構造体・共用体・辞書
- ✓ 多次元配列未対応
- ✓ 計算未対応
- ✓ Plasma・Flight未対応

C++の実装状況

- ✓ すべて実装済み
  - ✓ 一番実装が進む
  - ✓ C++実装のバインディングとして開発する言語があるため
- ✓ Apache Parquet実装も取り込んだ

Goの実装状況

- ✓ 未対応の型
  - ✓ 16ビット浮動小数点数・小数・共用体
- ✓ 計算対応（Gandivaは未対応）
- ✓ フォーマット変換はCSVのみ対応
- ✓ Plasma・Flight未対応

Javaの実装状況

- 未対応の型
  - 16ビット浮動小数点数
- 多次元配列未対応
- 計算対応（Gandivaも対応）
- フォーマット変換未対応
- JDBCを使ったDB連携対応

JavaScriptの実装状況

- ✓ TypeScript実装
  - ✓ WebブラウザーでもNode.js上でも動く
- ✓ 多次元配列未対応
- ✓ 計算対応（Gandivaは未対応）
- ✓ フォーマット変換未対応
- ✓ Plasma・Flight未対応

Juliaの実装状況

- ✓ 未対応の型
  - ✓ 16bit浮動小数点数・小数・構造体・共用体
- ✓ 多次元配列未対応
- ✓ 計算未対応
- ✓ フォーマット変換未対応
- ✓ Plasma・Flight未対応

Rustの実装状況

- ✓ 未対応の型
  - ✓ 小数・バイナリーデータ・共用体・辞書
- ✓ 計算未対応
- ✓ フォーマット変換はCSVのみ対応
- ✓ Plasma・Flight未対応
- ✓ Apache Parquet実装も取り込んだ

C・Lua・Rubyの実装状況

- ✓ C++バインディング
- ✓ 未対応の型：共用体
- ✓ 計算対応（Gandivaも対応）
- ✓ Plasma対応
- ✓ Flight未対応

MATLABの実装状況

- ✓ C++バインディング
- ✓ Featherの読み込みのみ対応

Pythonの実装状況

- ✓ C++バインディング
- ✓ pandas・NumPy相互変換対応
- ✓ 計算対応（Gandivaも対応）
- ✓ Plasma対応
- ✓ Flight未対応

Rの実装状況

- ✓ C++バインディング
- ✓ 未対応の型：小数・共用体
- ✓ 計算対応（Gandivaは未対応）
- ✓ フォーマット変換はFeatherのみ対応
- ✓ Plasma・Flight未対応

対応言語まとめ

- ✓ 基本的な型はすべての言語で対応済み
  - ✓ すでにデータ交換用途に使える
- ✓ データの高速な計算は一部言語で対応
- ✓ フォーマット変換も各言語の対応は様々

まとめ

- ✓ Apache Arrowが言語を超えて実現すること
  - ✓ データ交換・データ処理の高速化
  - ✓ 実装の共有（共同開発・ライブラリー化）
- ✓ Apache Arrowを使うとメリットがある人は Apache Arrowの開発にも参加しよう！

Apache Arrowの開発に参加を支援

- ✓ Apache Arrow東京ミートアップ2018
- ✓ 2018-12-08 13:30-
- ✓ 目的：開発者を増やす
  - ✓ 対象プロダクト：Apache Arrow・Apache Spark・Python関連・R関連・Ruby関連などなど

https://speee.connpass.com/event/103514/

OSSをITに活用

- ✓ 「使う」だけが活用かな
- ✓ 「開発」にも参加するのはどうかな
  - ✓ 参加した方が活用できることもあるかも

OSSの開発に参加を支援

- ✓ OSS Gate東京ワークショップ
- ✓ 2018-12-15 13:00-
- ✓ やること：「OSSの開発に参加」を体験
  - ✓ 対象プロダクト：好きなOSS

https://oss-gate.doorkeeper.jp/events/76042