ClearCode

Apache Arrow

須藤功平

株式会社クリアコード

RubyData Tokyo Meetup

2018-11-17

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

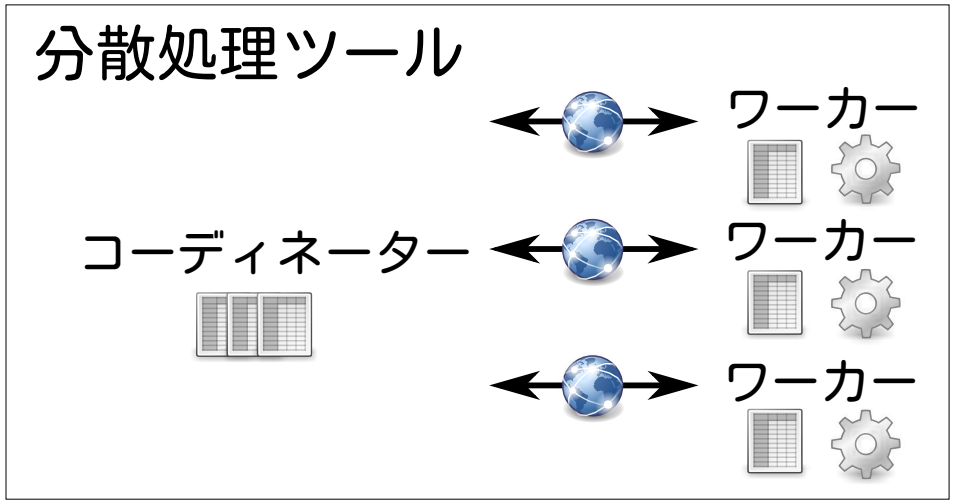
提供するもの

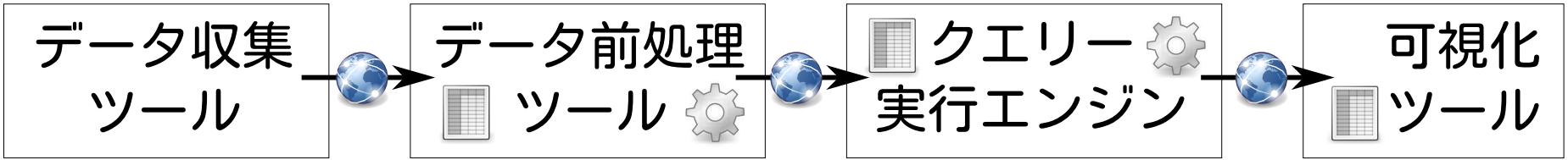
- ✓ 高速なデータフォーマット
- ✓ 高速なデータ処理ロジック
  - ✓ 各プロダクトで個別に実装するより
    一緒にいいものを実装して共有しよう！
- ✓ 効率的なデータ交換処理
- ✓ ...

利用例

Apache Arrow提供

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





大事にすること1

効率的なデータ交換

前提

イマドキの
データ処理システムは
単一コンポーネントで
完結しない

複数コンポーネント

- メリット：
  - コンポーネント毎に適した言語を使える
- デメリット：
  - データ交換が増える（オーバーヘッド）

データ交換コスト

- ✓ シリアライズコスト
- ✓ 転送コスト
- ✓ デシリアライズコスト

コスト例：JSON

```
シリアライズコスト
  [1] -#to_json-> "[1]"
転送コスト
  -output#write->
  -input#read->
デシリアライズコスト
  "[1]" -JSON.parse-> [1]
```

コスト比較例：JSON

```
n = 1000000
numbers = n.times.to_a
JSON.dump(numbers, json_file)
JSON.load(json_file)
```

コスト比較例：Apache Arrow

```
n = 1000000
numbers = Arrow::Int32Array.new(numbers)
arrow_table = Arrow::Table.new("number" => numbers)
arrow_table.save(arrow_path)
Arrow::Table.load(arrow_path)
```

コスト比較例

	実行時間	JSON比
JSON	0.099秒	1
Apache Arrow	0.002秒	1/50

データ交換コストの影響

- ✓ コンポーネント数と正の相関
  - ✓ コンポーネントが増えると無視できない
- ✓ データ量と正の相関
  - ✓ データが多くなると無視できない

まとめ

- ✓ イマドキのデータ処理システムで
  大量データを処理するなら
  データ交換コストを無視できない
- ✓ Apache Arrowはデータ交換コストが低い
  - ✓ 仕組みは後述

大事にすること2

各種言語で使えること

各種言語

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

イマドキのデータ処理システム

- コンポーネント毎に適した言語を採用
  - 採用言語でApache Arrowを使えないと
    システムでApache Arrowを使えない
- Apache Arrowに対応していれば
  コンポーネントでその言語を採用しやすい
  - Railsが活きるコンポーネントでRubyを使うとか

実現方法

- ネイティブ実装
  - Java, C++, JavaScript, Go, Rust, C#
  - メリット：扱いやすい（インストールが楽とか）
- C++実装のバインディング
  - Python, C, **Ruby**, Lua, MATLAB, R
  - メリット：高速・実装の共有

まとめ

- ✓ Apache Arrowは各種言語で使える
  - ✓ Rubyと他の言語でのデータ交換が楽になる
- ✓ Ruby実装はC++実装のバインディング
  - ✓ 速い・豊富な機能（C++実装はすごく進んでいる）

大事にすること3

速いこと

速さが必要な理由

大量のデータを
処理するため

ポイント：大量データ前提の設計

速いデータフォーマット

- パースせずに使えるデータフォーマット
  - メモリー上で効率よく扱える並びでデータを配置
  - パースしなくてよいし、そのまま使っても速い
- 既存のデータの並びと互換性あり
  - 例：NumPyの数値配列と互換
  - 互換性があると**ゼロコピー**で使える

速いデータ処理

- SIMD・キャッシュメモリー・マルチコアで高速化
  - データをアライン・局所化・リードオンリーに
- 高速な式評価器
  - 式：column1 + column2みたいなやつ
    ifとかも使える
  - Gandiva：式をJITコンパイルして実行

データ処理例：Ruby

```ruby
n = 100000
ruby_table = n.times.collect do
  {
    "number1" => rand,
    "number2" => rand,
  }
end
ruby_table.collect do |record|
  record["number1"] + record["number2"]
end
```

データ処理例：Numo::NArray

```
n = 100000
numo_number1 = Numo::DFloat.new(n).rand
numo_number2 = Numo::DFloat.new(n).rand
numo_number1 + numo_number2
```

データ処理例：Gandiva

```
n = 100000
arrow_number1 = Arrow::DoubleArray.new(n.times.collect {rand})
arrow_number2 = Arrow::DoubleArray.new(n.times.collect {rand})
arrow_table = Arrow::Table.new("number1" => arrow_number1,
                               "number2" => arrow_number2)
```

データ処理例：Gandiva

```ruby
# 次のリリースまでにいい感じに書けるようにする予定
schema = arrow_table.schema
expression =
  Gandiva::Expression.new("add",
                          [schema[:number1], schema[:number2]],
                          Arrow::Field.new("sum", :double))
projector = Gandiva::Projector.new(schema, [expression])
arrow_table.each_record_batch do |record_batch|
  projector.evaluate(record_batch)
end
```

データ処理例

	実行時間	Ruby比
Ruby	0.010247秒	1
Numo::NArray	0.000158秒	1/67
Gandiva	0.000459秒	1/25

Numo::NArrayがすごくがんばっている

速いデータ交換

- 同一マシン上での交換
  - メモリーファイルシステム上に置いてmmap
  - Plasma：データ共有サーバーを動かしてIPC
    Inter-Process Communication
- 別マシン上での交換
  - Arrow Flight：gRPCベースのRPCフレームワーク

GPUで速い

- Plasma：GPU対応
- RAPIDS：NVIDIAのGPUをデータサイエンスで活用するためのプロジェクト
  - libgdf：Apache Arrowフォーマットのデータを GPUで扱うデータフレームライブラリー
    Rubyバインディングはまだない

まとめ

- ✓ Apache Arrowは速い
  - ✓ 速いデータフォーマット
  - ✓ 速いデータ処理（もっと速くなるはず）
  - ✓ 速いデータ交換
- ✓ 今後、GPUももっと活用していく

Apache Arrowのこれから例

データフォーマットの相互変換強化

相互変換：Apache Parquet

```
# Apache Arrow→Apache Parquet
arrow_table.save("data.parquet")
# Apache Parquet→Apache Arrow
Arrow::Table.load("data.parquet")
```

相互変換：Feather

```
# Apache Arrow→Feather
arrow_table.save("data.feather")
# Feather→Apache Arrow
Arrow::Table.load("data.feather")
```

相互変換：Apache ORC

```
# Apache ORC→Apache Arrow
Arrow::Table.load("data.orc")
```

相互変換：CSV

```
# Apache Arrow→CSV
arrow_table.save("data.csv")
# CSV→Apache Arrow
Arrow::Table.load("data.csv")
```

CSV読み込み例

```ruby
# 標準ライブラリー（Ruby実装）
CSV.foreach(path) {|row| row}
# 拡張ライブラリー
Ccsv.foreach(path) {|row| row}
# C++実装
Arrow::Table.load(path, use_threads: true)
```

CSV読み込み時間

	実行時間	csv比
csv	0.818315秒	1
ccsv	0.064988秒	1/13
Apache Arrow	0.009030秒	1/90

相互変換：Rubyオブジェクト

Rubyバインディング限定

```
# Active Record→Apache Arrow
User.all.to_arrow
# Numo::NArray→Apache Arrow
narray.to_arrow
# NMatrix→Apache Arrow
matrix.to_arrow
```

相互変換の今後

- ✓ JSON→Apache Arrow
- ✓ Apache Avro→Apache Arrow

Apache Arrowのこれから（もっと）

- ✓ RDBMS連携強化
  - ✓ PostgreSQL・MySQLでの実行結果を Apache Arrowフォーマットで返す
- ✓ テンソルサポート強化
- ✓ ...

Rubyバインディングの今後

- ✓ Plasma対応
- ✓ GandivaバインディングのAPIをいい感じに
- ✓ バインディングフレームワークの高速化

一緒に開発しようぜ！

Apache ArrowとRubyまわりの今後

- ✓ libgdfのRubyバインディング開発
- ✓ gumath/xnd/ndtypesとの連携

一緒に開発しようぜ！

おしらせ1

- ✓ コード懇親会（今日の懇親会）
- ✓ 興味がでてきたプロダクトのコードを一緒に触ってみよう！
- ✓ 開発に参加したくなるかも！

https://github.com/speee/code-party/tree/master/rubydata-tokyo-meetup-2018

おしらせ2

- ✓ OSS Gate東京ミートアップ
  for Red Data Tools in Speee
- ✓ 2018-11-20 19:30-（来週の火曜日）
- ✓ Red Data Toolsメンバーが開発する集まり

https://speee.connpass.com/event/105237/

おしらせ3

- 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、Red Data Tools、Ruby/Numo、SciRubyなど

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