RubyもApache Arrowでデータ処理言語の仲間入り

須藤功平

クリアコード

DataScience.rbワークショップ

2017-05-19

はじめに

私はRubyが好きだ

だからデータ分析だって

Rubyでやりたい

Rubyよりも向いている言語があるのはわかっているけどさー

Apache Arrow

データフォーマットの仕様
と
その仕様を処理する実装

Arrow：解決したい問題

- ✓ 高いデータ交換コスト
  - ✓ →低くしたい
- ✓ 重複した最適化実装
  - ✓ →実装を共有したい

Arrow：文脈

ビッグデータの分析

ビッグデータの分析

- いろんなシステムが連携
  - Java実装のもろもろとPythonとR
- システム間でデータ交換が必要
  - 交換する度にシリアライズ・パース
  - ↑に結構CPUと時間を使われる…
  - そんなのより分析処理に使いたい！

Arrow：解決方針

コストゼロの
シリアライズ・
パース

Arrow：コストゼロの実現

- そのまま使えるフォーマット
  - 例:int8の配列→int8の値を連続配置
  - 1バイトずつずらせば高速アクセス可
- Arrowのトレードオフ
  - サイズ圧縮よりシリアライズゼロ
  - 参考：Parquetはサイズ圧縮優先

Arrowがある世界

- 各システムがサクサク連携
  - 例：PySparkが高速化
  - 理由:Py↔Javaのデータ交換コスト減
- Java・Python・R以外も活躍
  - 例：Ruby・Lua・Julia・Go・Rust…
  - 理由：低コストでデータ交換可能

ArrowとRuby

チャンス！

ArrowとRubyとデータ分析

- RubyがArrowに対応
  - Rubyにデータが回ってくる！
  - →Rubyにもデータ分析の機会が！
    （今はできることは少ないだろうけど…）
- 次のステップ
  - できることを増やしていく！
  - →Rubyでもいろいろデータ分析！

ArrowとRubyの今

- ✓ RubyでArrowを使える！
  - ✓ 私が使えるようにしているから！
    コミッターにもなった
  - ✓ 公式リポジトリーにも入っている
    厳密に言うと違うんだけど公式サポートだと思ってよい
- ✓ Rubyでデータを読み書きできる
  - ✓ いくらかデータ処理もできる

今できること

- ✓ Python・R…とのデータ交換
- ✓ データ処理をいくらか
- ✓ Rubyの各種ライブラリー間でのデータ交換

Arrow：Python

```
# pandasでデータ生成→Arrow形式で書き込み
import pandas as pd
import pyarrow as pa

df = pd.DataFrame({"a": [1, 2, 3],
                   "b": ["hello", "world", "!"]})
record_batch = pa.RecordBatch.from_pandas(df)

with pa.OSFile("/dev/shm/pandas.arrow", "wb") as sink:
    schema = record_batch.schema
    writer = pa.RecordBatchFileWriter(sink, schema)
    writer.write_batch(record_batch)
    writer.close()
```

Arrow：Ruby

```ruby
# RubyでArrow形式のpandasのデータを読み込み
require "arrow"

Input = Arrow::MemoryMappedInputStream
Input.open("/dev/shm/pandas.arrow") do |input|
  reader = Arrow::RecordBatchFileReader.new(input)
  reader.each do |record_batch|
    puts("=" * 40)
    puts(record_batch)
  end
end
```

Arrow：Lua

```
-- LuaでArrow形式のpandasのデータを読み込み
-- Torchへの変換コードはArrowの公式リポジトリーにアリ
local lgi = require "lgi"
local Arrow = lgi.Arrow

local input_class = Arrow.MemoryMappedInputStream
local input = input_class.new("/dev/shm/pandas.arrow")
local reader = Arrow.RecordBatchFileReader.new(input)
for i = 0, reader:get_n_record_batches() - 1 do
   local record_batch = reader:get_record_batch(i)
   print(string.rep("=", 40))
   print("record-batch["..i.."]:")
   io.write(record_batch:to_string())
end
input:close()
```

Feather：R

```
# Rでデータ生成→Feather形式で書き込み
library("feather")

df = data.frame(a=c(1, 2, 3),
                b=c(1.1, 2.2, 3.3))
write_feather(df, "/dev/shm/dataframe.feather")
```

Feather：Ruby

```ruby
# RubyでFeather形式のRのデータを読み込み
require "arrow"

Input = Arrow::MemoryMappedInputStream
Input.open("/dev/shm/dataframe.feather") do |input|
  reader = Arrow::FeatherFileReader.new(input)
  reader.columns.each do |column|
    puts("#{column.name}: #{column.to_a.inspect}")
  end
end
```

Parquet：Python

```python
# Pythonでデータ生成→Parquet形式で書き込み
import pandas as pd
import pyarrow as pa
import pyarrow.parquet as pq

df = pd.DataFrame({"a": [1, 2, 3],
                   "b": ["hello", "world", "!"]})
table = pa.Table.from_pandas(df)
pq.write_table(table, "/dev/shm/pandas.parquet")
```

Parquet：Ruby

```ruby
# RubyでParquet形式のデータを読み込み
require "arrow"
require "parquet"

path = "/dev/shm/pandas.parquet"
reader = Parquet::ArrowFileReader.new(path)
table = reader.read_table
table.each_column do |column|
  puts("#{column.name}: #{column.to_a.inspect}")
end
```

対応データ形式まとめ

- Arrow形式
  - 各種言語（これから広く使われるはず）
- Feather形式
  - Python・R専用
- Parquet形式
  - 各種言語（Hadoop界隈ですでに広く使われている）

データ処理例

- Groongaでフィルター
- Groonga
  - 全文検索エンジン
  - カラムストアなので集計処理も得意
  - Apache Arrow対応
  - よくできたRubyバインディングあり

Groonga : Ruby

```ruby
# 空のテーブルにArrow形式のデータを読み込む
logs = Groonga::Array.create(name: "logs")
logs.load_arrow("/dev/shm/pandas.arrow")
logs.each {|record| p record.attributes}
# フィルター
filtered_logs = logs.select do |record|
  record.b =~ "hello" # "hello"で全文検索
end
# フィルター結果をArrow形式で書き込み
filtered_logs.dump_arrow("/dev/shm/filtered.arrow",
                         column_names: ["a", "b"])
```

Groonga：Python

```
# Arrow形式のGroongaでのフィルター結果を読み込む
import pyarrow as pa

with pa.OSFile("/dev/shm/filtered.arrow") as source:
    writer = pa.RecordBatchFileReader(source)
    print(writer.get_record_batch(0).to_pandas())
```

Rubyでデータ処理（現状）

- ✓ 既存のCライブラリーを活用
  - ✓ 速度がでるし機能もある
- ✓ CライブラリーをArrowに対応
  - ✓ Arrow→Ruby→Cライブラリー
    ↑から↓で高速化（オブジェクト生成は遅い）
  - ✓ Arrow→Cライブラリー

Rubyでデータ処理（案）

- ✓ Fluentdとか速くなりそう
  - ✓ 途中でメッセージを参照しないなら
- ✓ MessagePackからArrowに変える
  - ✓ Arrowのまま出力先へ送る
  - ✓ 途中でRubyオブジェクトを作らない
    シリアライズ・パースがなくなって速い！

多次元配列

- Arrowではオプション機能
  - テンソルと呼んでいる
    （traditional multidimensional array objectと説明）
- C++実装ではサポート
  - バインディングでは使える
  - Python・Ruby・Lua…では使える

Tensor：Python

```python
# NumPyでデータ生成→書き込み
import pyarrow as pa
import numpy as np

ndarray = np.random.randn(10, 6) # 10x6
print(ndarray)
tensor = pa.Tensor.from_numpy(ndarray)
with pa.OSFile("/dev/shm/tensor.arrow", "wb") as sink:
    pa.write_tensor(tensor, sink)
```

Tensor：Ruby

```ruby
# Rubyで読み込み
require "arrow"

Input = Arrow::MemoryMappedInputStream
Input.open("/dev/shm/tensor.arrow") do |input|
  tensor = input.read_tensor(0)
  p tensor.shape # => [10, 6]
end
```

Ruby：GSL

```ruby
# GSLオブジェクトに変換
require "arrow"
require "arrow-gsl"
require "pp"

Input = Arrow::MemoryMappedInputStream
Input.open("/dev/shm/tensor.arrow") do |input|
  tensor = input.read_tensor(0)
  pp tensor.to_gsl
  # tensor.to_gsl.to_arrow == tensor
end
```

Ruby：NMatrix

```ruby
# NMatrixオブジェクトに変換
require "arrow"
require "arrow-nmatrix"
require "pp"

Input = Arrow::MemoryMappedInputStream
Input.open("/dev/shm/tensor.arrow") do |input|
  tensor = input.read_tensor(0)
  pp tensor.to_nmatrix
  # tensor.to_nmatrix.to_arrow == tensor
end
```

Ruby：Numo::NArray

```ruby
# Numo::NArrayオブジェクトに変換
require "arrow"
require "arrow-numo-narray"
require "pp"

Input = Arrow::MemoryMappedInputStream
Input.open("/dev/shm/tensor.arrow") do |input|
  tensor = input.read_tensor(0)
  pp tensor.to_narray
  # tensor.to_narray.to_arrow == tensor
end
```

ここまでのまとめ1

- ✓ Arrowが実現したい世界
  - ✓ データ交換コストが低い世界
  - ✓ 最適化実装を共有している世界

ここまでのまとめ2

- ✓ RubyとArrowの今
  - ✓ ArrowはRubyを公式サポート！
  - ✓ Rubyの外の世界とデータ交換可能
    （Arrow・Feather・Parquetをサポート）
  - ✓ Rubyの各種ライブラリーとの相互変換が可能
    （メモリーコピーぐらいのコストで）

ArrowとRubyとこれから

- Arrow
  - データフレーム処理の最適化実装
  - マルチコア・GPU対応
- Ruby
  - Red Data Toolsプロジェクト

Red Data Tools

- Rubyでデータ処理したいなぁ！の実現を目指すプロジェクト
- リソース：
  - GitHub: red-data-tools
  - https://red-data-tools.github.io
  - https://gitter.im/red-data-tools

既存プロダクト

- ✓ Red Arrow（ArrowのRubyバインディング）
  - ✓ Red Arrow XXX（ArrowとXXXの相互変換）
- ✓ Parquet GLib（ParquetのGLibバインディング）
- ✓ Red Parquet（ParquetのRubyバインディング）
- ✓ Jekyll Jupyter Notebook plugin（JekyllでJupyter Notebookを表示）

ポリシー1

Collaborate
over Ruby communities

Ruby以外の人たちとも言語を超えて協力する
Apache Arrowがやっていることはまさにそう
もちろんRubyの人たちとも協力する

ポリシー2

Acting than blaming

時間は嘆き・非難より手を動かすことに使う

ポリシー3

Continuous small works than
a temporary big work

一時的にガッとやって終わりより
小さくても継続して活動する

ポリシー4

The current
lack of knowledge
isn't matter

現時点で数学や統計学などの知識が足りなくても問題ない
既存の実装を使ったりそこから学んだりできるから

ポリシー5

Ignore blames from outsiders

部外者の非難は気にしない
結果がでるまでグチグチ言われるはず :p

ポリシー6

Fun!
Because we use Ruby!

Rubyを使うんだし楽しくやろう！

Join us!

- Rubyでデータ処理したい人！
- ポリシーに同意できる人！
- リソース：
  - GitHub: red-data-tools
  - https://red-data-tools.github.io
  - https://gitter.im/red-data-tools