Apache Arrow

Dra. Yessenia Yari Ramos

Universidad Católica San Pablo

Departamento de Ciencia de la Computación

Contenido

- Introducción a Apache Arrow
- Por qué Apache Arrow es importante
- Apache Arrow en Python
- Estructura de datos Arrow
- Operaciones comunes con Arrow en Python
- Ventajas de utilizar Apache Arrow
- Ejemplos de uso
- Recursos adicionales

Introducción a Apache Arrow

¿Qué es Apache Arrow?

Software de código abierto desarrollado por la Apache Software Foundation (ASF)

se centra en la **interoperabilidad** y la **eficiencia** en el **procesamiento y transporte** de datos en aplicaciones de análisis de datos y big data.

Propósito

- Ecosistema de **big data** y **análisis de datos**
- Objetivo principal es proporcionar un formato de **alto rendimiento** y **estandarizada** que permita a las aplicaciones trabajar con datos de manera eficiente y sin problemas, independientemente del lenguaje de programación utilizado.

Por qué Apache Arrow es Importante

Características Claves

- **Interoperabilidad**: Permite a las aplicaciones escritas en diferentes lenguajes de programación **compartir datos** de manera eficiente sin necesidad de costosas conversiones de formato.
- **Eficiencia**: Arrow se centra en el rendimiento al **minimizar las conversiones** de datos entre formatos (memoria y acceso rápido)
- **Soporte multiplataforma**: Compatible con varios lenguajes de programación, incluidos Python, C++, Java, R y más.
- **Estructura de datos común**: Define una estructura de datos común que incluye tablas, arrays y esquemas (representación y el transporte de datos).
- **Soporte para sistemas distribuidos**: Adaptación a entornos distribuidos y sistemas de almacenamiento de big data.

“Herramienta valiosa para desarrolladores y empresas que trabajan en entornos de datos heterogéneos y complejos”.

Desafíos de la interoperabilidad de datos

Formatos de datos propietarios: Muchos lenguajes y aplicaciones utilizan formatos de datos propietarios o específicos del lenguaje, lo que dificulta la interoperabilidad.

Diferencias en la representación de datos: Cada lenguaje de programación tiene su propia forma de representar los datos en memoria. Por ejemplo, un entero en Python se almacena de manera diferente a un entero en C++. **Esto complica la transferencia de datos entre lenguajes sin una conversión** adecuada.

Desafíos de la interoperabilidad de datos

Costos de rendimiento: La conversión de datos puede ser costosa en términos de rendimiento y consumo de recursos lo cual afecta negativamente la velocidad y la eficiencia de las aplicaciones.

Problemas de precisión y redondeo: Algunos lenguajes y sistemas utilizan algoritmos de redondeo y precisión diferentes al realizar cálculos matemáticos. Esto puede llevar a resultados ligeramente **diferentes** (problemático en aplicaciones científicas y de análisis numérico).

Desafíos de la interoperabilidad de datos

Problemas de codificación de caracteres: La codificación de caracteres puede variar entre lenguajes y sistemas, lo que puede provocar **problemas de codificación al transferir datos** que contienen caracteres especiales o internacionales.

Dificultades en la serialización y deserialización: La serialización y deserialización de datos en diferentes lenguajes y plataformas puede ser complicada debido a las diferencias en la forma en que se estructuran los datos y se manejan los tipos de datos.

Desafíos de la interoperabilidad de datos

Falta de metadatos comunes: La falta de metadatos estandarizados sobre la estructura y el tipo de datos puede hacer que sea difícil interpretar y utilizar los datos correctamente en diferentes lenguajes y sistemas.

Complejidad en la administración de memoria: La administración de memoria es diferente en lenguajes de programación con y sin recolector de basura. Esto puede complicar la gestión de datos cuando se comparten entre lenguajes con enfoques de administración de memoria diferentes

Apache Arrow en Python

¿cómo utilizar Apache Arrow en Python?

Apache Arrow compatible con python 3.8, 3.9, 3.10, 3.11

1. Tener Instalado **pip**.

   !python -m ensurepip --default-pip

2. Instalar las bibliotecas necesarias de Apache Arrow y las dependencias de Python utilizando pip.

   !pip install pyarrow

3. Importar

   import pyarrow as pa

Estructura de datos Arrow

Cuatro estructuras

- Array
- RecordBatch
- ChunkedArray
- Tabla

Array

```
import pyarrow as pa

 age = pa.array([1, 12, 17, 23, 28], type=pa.int8())

name = pa.array([‘Alice’, ’Bob’, ’Charlie’])

city = pa.array([‘Brasil’, ‘Perú’,’New York’])
```

Tabla

```
import pyarrow as pa

birthdays_table =  pa.table([days, months, years], names=["days", "months",
"years"])
```

RecordBatch

Consiste en un conjunto de datos que se organiza por columnas

1. Creamos un esquema

```
schema = pa.schema([
        ('name', pa.string()),
        ('age', pa.int32()),
        ('city', pa.string())
])
```

2. Creamos RecordBatch a partir de array y esquema

```
record_batch = pa.RecordBatch.from_arrays([name, age, city], schema=schema)
```

ChunkedArray

Matriz formada por submatrices

```
arr_1 = pa.array([1, 2, 3])

arr_2 = pa.array([4, 5, 6])

chunked_array = pa.chunked_array([arr_1, arr_2])
```

Operaciones comunes con Arrow en Python

Operaciones con Arrow

Creación de Arrays

Creación de RecordBatch

Acceso a datos

Nombre: name_column = record_batch[‘name’]

Indice: name_column = record_batch(0)

Operaciones con Arrow

Filtrado de datos

```
filtro = record_batch['age'] > 25
```

Operaciones Matemáticas

```
array1 = pa.array([1, 2, 3])
array2 = pa.array([4, 5, 6])
suma = pa.add(array1, array2)
sub = pa.subtract(array1,array2)
mult = pa.multiply()
```

Operaciones con Arrow

Operaciones estadísticas

pa.sum()

pa.median()

pa.stddev()

average_age = **pa**.mean(record_batch[‘age’]).as_ap()

Operaciones con Arrow

Serialización y deserialización

```
serializado = record_batch.serialize()
deserializado = pa.RecordBatch.deserialize(serializado)
```

Conversiones de tipos de dato

```
array_enteros = pa.array([1, 2, 3])
array_floats = pa.cast(array_enteros, pa.float32())
```

Operaciones con Arrow

Concatenación de datos

Tablas y arrays

array1 = **pa**.array([1, 2, 3])

array2 = **pa**.array([4, 5, 6])

array_concatenado = **pa**.**concat_arrays**([array1, array2])

Ventajas de Utilizar Apache Arrow

Ventajas

Mitiga el problema de bajo rendimiento

Mejora la interoperabilidad

- Compartir datos de manera eficiente entre diferentes lenguajes de programación

Acceso eficiente en memoria

- Reduce la necesidad de conversiones costosas
- Mejor rendimiento en lectura y escritura

Ventajas

Menos recursos

- Minimiza las copias de datos
- Ahorra recursos de CPU, memoria
- Mejor velocidad de procesamiento

Portabilidad de datos

- Compartir datos sin pérdida de información

Ventajas

Integración con herramientas de big data

Apache parquet: un formato de archivo columnar, diseñado para almacenamiento eficiente y rápida recuperación de datos estructurados

Apache spark: un framework de computación distribuido, un motor de procesamiento y análisis de volúmenes de datos

Ejemplos de Uso

Creación de una lista y array de arrow

```python
import pyarrow as pa
# Crear una lista de Python
data = [1, 2, 3, 4, 5, 6, 10]
```

```python
arrow_array = pa.array(data)
```

```python
print(arrow_array)
```

```
[
  1,
  2,
  3,
  4,
  5,
  6,
  10
]
```

DataFrame de arrow

```
1 import pyarrow as pa
2 import pandas as pd
3 import numpy as np
```

```
4 data = {'col1': [1, 2, 3], 'col2': ['a', 'b', 'c']}
5
6 # Crear un objeto DataFrame de Arrow desde el diccionario
7 arrow_table = pa.table(data)
```

```
10 print(arrow_table)
```

```
pyarrow.Table
col1: int64
col2: string
----
col1: [[1,2,3]]
col2: [["a","b","c"]]
```

Conversión de un array arrow a DataFrame pandas

```
2 arrow_array = pa.array([1, 2, 3, 4, 5])
```

```
4 # Convertir el objeto Arrow Array a un DataFrame de Pandas
5 pandas_df = arrow_array.to_pandas()
6 print(type(pandas_df))
```

```
<class 'pandas.core.series.Series'>
```

```
8 # Convertir un DataFrame de Pandas a un objeto Arrow Table
9 arrow_table = pa.Array.from_pandas(pandas_df)
```

```python
df = pd.DataFrame({'year': [2020, 2022, 2019, 2021],
                   'n_legs': [2, 4, 5, 100],
                   'animals': ["Flamingo", "Horse", "Brittle stars", "Centipede"]})

print(type(df))

pa.Table.from_pandas(df)
```

```
<class 'pandas.core.frame.DataFrame'>
pyarrow.Table
year: int64
n_legs: int64
animals: string
----
year: [[2020,2022,2019,2021]]
n_legs: [[2,4,5,100]]
animals: [["Flamingo","Horse","Brittle stars","Centipede"]]
```

Suma de array arrow

```
 5 arrow_array = pa.array([4, 5, 6], type=pa.int32())
 6
 7 # Realizar una suma de elementos
 8 sum_result = arrow_array.sum()
 9
10 print(sum_result)
```

```
15
```

Mascara Booleana

```
1 array = pa.array([1, 2, 3, 4, 5],
2 mask=np.array([True, False, True, False, True]))
3
4 print(array)
```

```
[
  null,
  2,
  null,
  4,
  null
]
```

Recursos Adicionales

Documentación de Apache Arrow

https://arrow.apache.org/

Apache spark(PySpark) usa Apache arrow:

Mejorar el rendimiento de conversiones entre Spark DataFrame y panda DataFrame

https://spark.apache.org/


Gracias!