# Effectively memory profiling distributed PySpark code

PyCon Taiwan 2024
September 22, 2024

Kaashif Hymabaccus
Senior Software Engineer

TechAtBloomberg.com

Bloomberg Engineering

## Topics we will cover

- Why some pandas code can’t scale to large datasets
- Migrating from pandas to PySpark - dos and don’ts
- Diagnosing and fixing PySpark memory issues
- Profiling native libraries (C++, Rust) used in Apache Spark executors

TechAtBloomberg.com

Bloomberg

Engineering

## Initial problem

- We have **Parquet files** stored in S3
- We want to compute a weighted average
- The aggregation logic comes from a **native library**
- Constraint: Each node has **1 GB** of memory

*Note: This example is scaled down*

## What does our data look like?

```
$ parquet-tools show input.parquet
+-------------+-------------+
|      weight |       value |
|-------------+-------------|
|      10.0   |        2.0  |
|      20.0   |        1.0  |
+-------------+-------------+
```

In this case, we can compute the weighted average:

$$(10 * 2 + 20 * 1) / (20 + 10) = 1.333$$

However, in practice, aggregation can be much more complex!

## What does our aggregation module look like?

```
import ctypes
import numpy as np
from numpy.ctypeslib import ndpointer
```

```
lib = ctypes.cdll.LoadLibrary("/examples/libfinance.so")
native_wavg = lib.weighted_avg
native_wavg.restype = None
native_wavg.argtypes = [
    ndpointer(ctypes.c_double, flags="C_CONTIGUOUS"),
    ctypes.c_size_t,
    ndpointer(ctypes.c_double, flags="C_CONTIGUOUS")
]
```

Binding to native shared library with C FFI

```
def weighted_avg(np_array):
    arr = np.ascontiguousarray(np_array)
    result = np.array([0.0, 0.0])
    native_wavg(arr, arr.size, result)
    return result
```

NumPy-compatible Python interface

TechAtBloomberg.com

Bloomberg Engineering

## Example #1: Typical pandas code

```
import pandas as pd
import finance

# Read a single small parquet file
df = pd.read_parquet("/examples/input.parquet")

# Returns weighted average and total weight
print(finance.weighted_avg(df.to_numpy()))
```

TechAtBloomberg.com

Bloomberg

Engineering

## Example #1: Typical pandas code

```
$ python example1.py
(np.float64(-1301.4259223484928),
np.float64(21.851472001775814))
```

TechAtBloomberg.com

Bloomberg

Engineering

## Example #2: Dataset grows larger than available RAM

```
import pandas as pd
import finance

# Read many large parquet files
df = pd.read_parquet("/examples/big_data/")

# Returns weighted average and total weight
print(finance.weighted_avg(df.to_numpy()))
```

## Example #2: Dataset grows larger than available RAM

```
$ python example2.py
Killed
```

- Exits with code 137 = SIGKILL
- Killed by the kernel’s OOM Killer (out of memory)

## Using Memray to diagnose OOM

- Memray is a Python memory profiler
- Published as open source project by Bloomberg in 2022
- Can profile native (e.g., C/C++/Rust) code
- Use `python -m memray run script.py`
- Generates readable HTML reports with detailed stats

## Example #2: Dataset grows larger than available RAM

```
$ python -m memray run example2.py
Writing profile results into
memray-example2.py.6584.bin
Killed
```

Run code using Memray

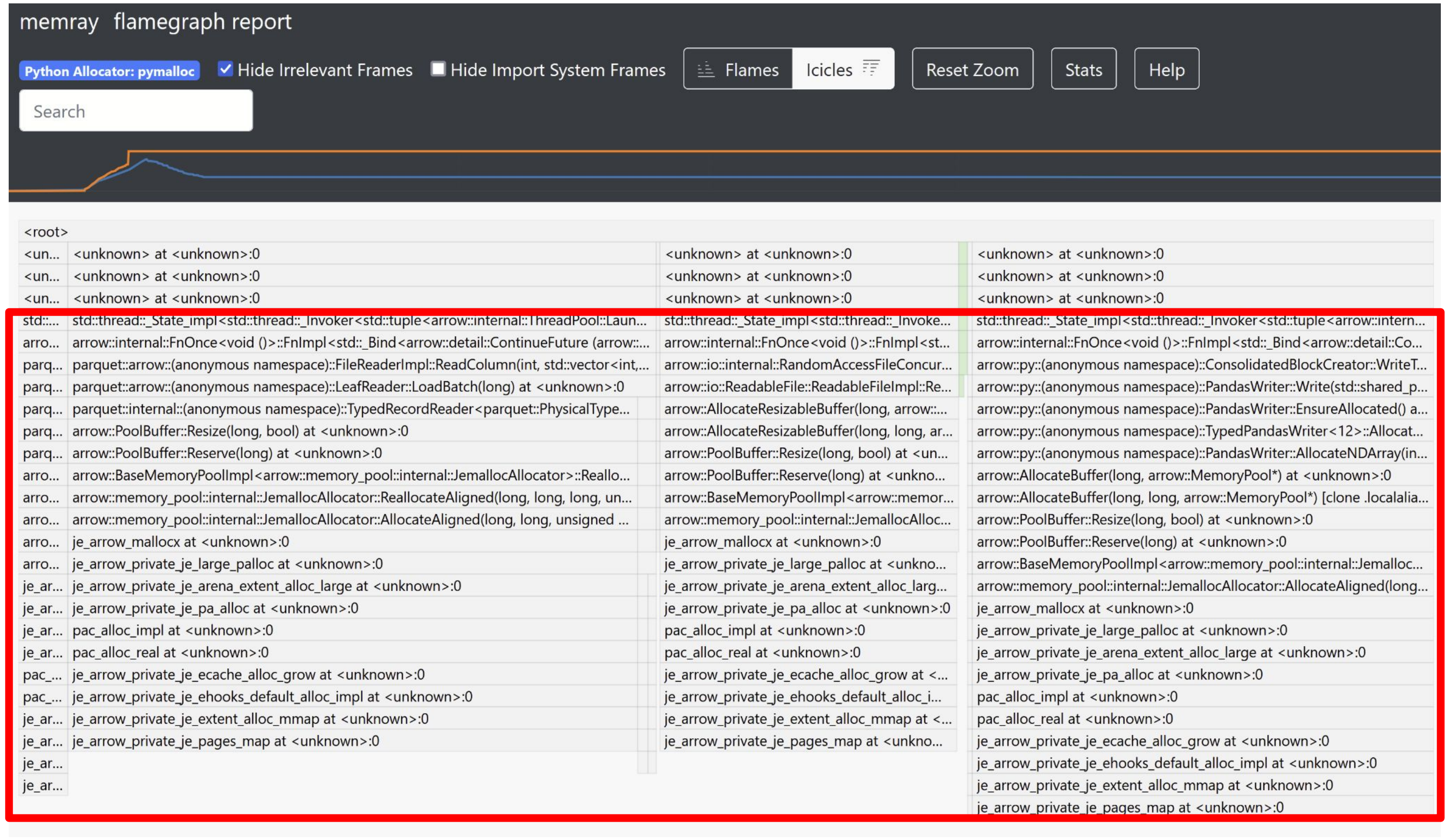
```
$ python -m memray flamegraph
memray-example2.py.6584.bin
Wrote memray-example2.py.6584.html
```

Generate HTML report of peak memory usage

TechAtBloomberg.com

Bloomberg Engineering

# Examining our Memray profile



☑ Hide Irrelevant Frames ☐ Hide Import System Frames

```
<root>
<un...  <unknown> at <unknown>:0                    <unknown> at <unknown>:0          <unknown> at <unknown>:0
<un...  <unknown> at <unknown>:0                    <unknown> at <unknown>:0          <unknown> at <unknown>:0
<un...  <unknown> at <unknown>:0                    <unknown> at <unknown>:0          <unknown> at <unknown>:0
```

```
std::...  std::thread::_State_impl<std::thread::_Invoker<std::tuple<arrow::internal::ThreadPool::Laun...
arro...   arrow::internal::FnOnce<void ()>::FnImpl<std::_Bind<arrow::detail::ContinueFuture (arrow::...
parq...   parquet::arrow::(anonymous namespace)::FileReaderImpl::ReadColumn(int, std::vector<int,...
parq...   parquet::arrow::(anonymous namespace)::LeafReader::LoadBatch(long) at <unknown>:0
parq...   parquet::internal::(anonymous namespace)::TypedRecordReader<parquet::PhysicalType...
parq...   arrow::PoolBuffer::Resize(long, bool) at <unknown>:0
parq...   arrow::PoolBuffer::Reserve(long) at <unknown>:0
arro...   arrow::BaseMemoryPoolImpl<arrow::memory_pool::internal::JemallocAllocator>::Reallo...
arro...   arrow::memory_pool::internal::JemallocAllocator::ReallocateAligned(long, long, long, un...
arro...   arrow::memory_pool::internal::JemallocAllocator::AllocateAligned(long, long, unsigned ...
arro...   je_arrow_mallocx at <unknown>:0
arro...   je_arrow_private_je_large_palloc at <unknown>:0
je_ar...  je_arrow_private_je_arena_extent_alloc_large at <unknown>:0
je_ar...  je_arrow_private_je_pa_alloc at <unknown>:0
je_ar...  pac_alloc_impl at <unknown>:0
je_ar...  pac_alloc_real at <unknown>:0
pac_...   je_arrow_private_je_ecache_alloc_grow at <unknown>:0
pac_...   je_arrow_private_je_ehooks_default_alloc_impl at <unknown>:0
je_ar...  je_arrow_private_je_extent_alloc_mmap at <unknown>:0
je_ar...  je_arrow_private_je_pages_map at <unknown>:0
je_ar...
je_ar...
```

```
std::thread::_State_impl<std::thread::_Invoke...
arrow::internal::FnOnce<void ()>::FnImpl<st...
arrow::io::internal::RandomAccessFileConcur...
arrow::io::ReadableFile::ReadableFileImpl::Re...
arrow::AllocateResizableBuffer(long, arrow::...
arrow::AllocateResizableBuffer(long, long, ar...
arrow::PoolBuffer::Resize(long, bool) at <un...
arrow::PoolBuffer::Reserve(long) at <unkno...
arrow::BaseMemoryPoolImpl<arrow::memor...
arrow::memory_pool::internal::JemallocAlloc...
je_arrow_mallocx at <unknown>:0
je_arrow_private_je_large_palloc at <unkno...
je_arrow_private_je_arena_extent_alloc_larg...
je_arrow_private_je_pa_alloc at <unknown>:0
pac_alloc_impl at <unknown>:0
pac_alloc_real at <unknown>:0
je_arrow_private_je_ecache_alloc_grow at <...
je_arrow_private_je_ehooks_default_alloc_i...
je_arrow_private_je_extent_alloc_mmap at <...
je_arrow_private_je_pages_map at <unkno...
```

```
std::thread::_State_impl<std::thread::_Invoker<std::tuple<arrow::intern...
arrow::internal::FnOnce<void ()>::FnImpl<std::_Bind<arrow::detail::Co...
arrow::py::(anonymous namespace)::ConsolidatedBlockCreator::WriteT...
arrow::py::(anonymous namespace)::PandasWriter::Write(std::shared_p...
arrow::py::(anonymous namespace)::PandasWriter::EnsureAllocated() a...
arrow::py::(anonymous namespace)::TypedPandasWriter<12>::Allocat...
arrow::py::(anonymous namespace)::PandasWriter::AllocateNDArray(in...
arrow::AllocateBuffer(long, arrow::MemoryPool*) at <unknown>:0
arrow::AllocateBuffer(long, long, arrow::MemoryPool*) [clone .localalia...
arrow::PoolBuffer::Resize(long, bool) at <unknown>:0
arrow::PoolBuffer::Reserve(long) at <unknown>:0
arrow::BaseMemoryPoolImpl<arrow::memory_pool::internal::Jemalloc...
arrow::memory_pool::internal::JemallocAllocator::AllocateAligned(long...
je_arrow_mallocx at <unknown>:0
je_arrow_private_je_large_palloc at <unknown>:0
je_arrow_private_je_arena_extent_alloc_large at <unknown>:0
je_arrow_private_je_pa_alloc at <unknown>:0
pac_alloc_impl at <unknown>:0
pac_alloc_real at <unknown>:0
je_arrow_private_je_ecache_alloc_grow at <unknown>:0
je_arrow_private_je_ehooks_default_alloc_impl at <unknown>:0
je_arrow_private_je_extent_alloc_mmap at <unknown>:0
je_arrow_private_je_pages_map at <unknown>:0
```

C++ stack frames are visible

`parquet::arrow::{anonymous}::FileReaderImpl` allocates a lot!

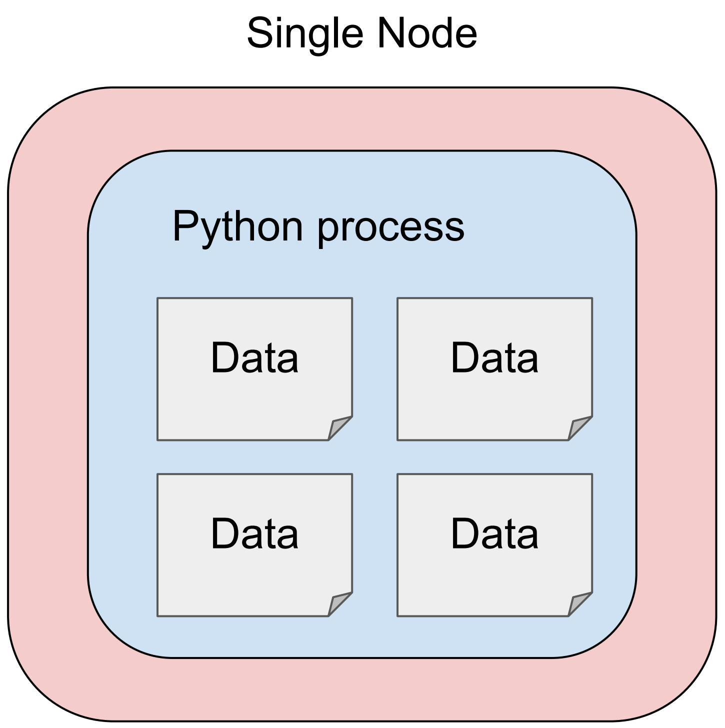
TechAtBloomberg.com

Bloomberg Engineering

## Examining our Memray profile

- More memory is required to even run `pd.read_parquet`!
- Needs more than 1 GB RAM
- We need to avoid loading all of our data into the memory on one machine

## pandas’ Computation Model



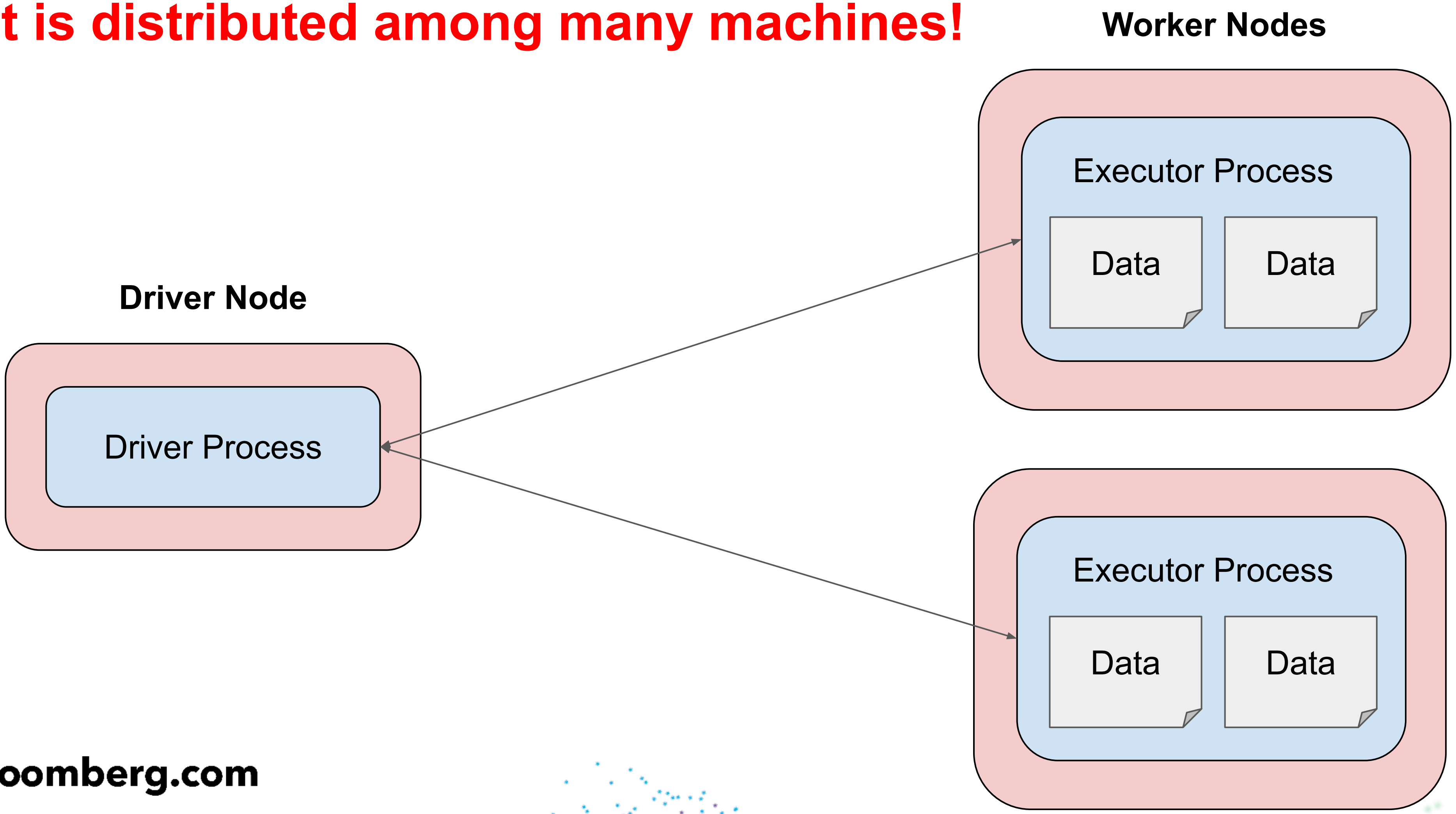
- All data **in memory on one machine**
- Doesn’t scale to arbitrarily large (>1 TB) datasets
- To handle larger datasets, we must distribute the dataset across **multiple machines**

## Solution: Distributed data processing

- Apache Spark is a framework for distributed data processing
- Distributes datasets across multiple nodes' memory
- Parallelizes computations across multiple nodes
- PySpark offers Python bindings for Spark

## Spark's Computation Model

**Dataset is distributed among many machines!**



TechAtBloomberg.com

## Example #3: Naively translated PySpark code

```
import pyspark.pandas as ps
import finance

# Read many Parquet files
# DataFrame is distributed across workers
df = ps.read_parquet("/examples/big_data/")

print(finance.weighted_avg(df.to_numpy()))
```

Attempt a drop-in replacement for `pandas`

TechAtBloomberg.com

Bloomberg

Engineering

## Example #3: Naive PySpark code

```
$ spark-submit.sh /examples/example3.py

py4j.protocol.Py4JJavaError: An error occurred
while calling o58.collectToPython.
: org.apache.spark.SparkException: Job aborted due
to stage failure: Total size of serialized results
of 6 tasks (1049.9 MiB) is bigger than
spark.driver.maxResultSize (1024.0 MiB)
```

- `df.to_numpy()` loads all data into the driver
- Spark blocks this pattern by default

## Example #3 (second attempt!)

```
$ spark-submit.sh --conf spark.driver.maxResultSize=2g
/examples/example3.py

py4j.protocol.Py4JJavaError: An error occurred while
calling o59.collectToPython.
: org.apache.spark.SparkException: Job aborted due to
stage failure: Exception while getting task result:
java.lang.IllegalStateException: unread block data
```

- Increases `spark.driver.maxResultSize`
- `collectToPython` still errors out!
- This error results from OOM in the driver

## Dangers of misusing pandas on Spark API

- Just replacing `import pandas as pd` with `import pyspark.pandas as pd` will lead to scaling issues

- Many APIs are fast with pandas on a small dataset, but slow/unusable on large datasets – even with pyspark.pandas
  - `DataFrame.to_numpy`
  - `DataFrame.shape`
  - `DataFrame.apply`

## `pyspark.pandas.DataFrame.to_numpy`

Symptoms (already seen these!):

- `An error occurred while calling o59.collectToPython.`
- `spark.driver.maxResultSize` issues
- `java.lang.IllegalStateException: unread block data`

Why?

- Loads the entire dataset into the driver
- Not how you’re supposed to use Spark!

## `pyspark.pandas.DataFrame.shape`

Issues:

- Very slow compared to `pandas.DataFrame.shape`
- Even for small datasets!

Why?

- `pandas.DataFrame.__len__` is close to free
- `pyspark.pandas.DataFrame.__len__` is very expensive!

## `pandas.DataFrame.__len__` is close to free!

```
>>> df = pd.read_parquet("examples/big_data")
>>> df.index
RangeIndex(start=0, stop=100000000, step=1)
>>> len(df)
100000000
```

- `len(df)` just takes the length of a `range` (O(1))
- We don't need to look at the data, just the index

See the pandas source code – it really *is* that simple!

## `pyspark.pandas.DataFrame.__len__` is expensive!

- Calls self.to_spark().count()
- Requires a full scan of the entire DataFrame (O(n))
- **Do not assume `df.shape` is free!**

TechAtBloomberg.com

Bloomberg Engineering

## `pyspark.pandas.DataFrame.apply(f)`

- Different from pandas when using axis=0
  - pandas: f is called once with the whole column
  - pyspark.pandas: f is called once **per batch**

- Only guarantee: Union of batches is the whole dataset

- Row batch size can be anything!

- Global aggregations aren’t possible

## Example #4: `pandas.DataFrame.apply(f)`

```
import pandas as pd
df = pd.DataFrame(range(1_000_000))
print(df.apply(sum, axis=0))
```

This prints:

```
$ python examples/example4.py
0  499999500000
dtype: int64
```

TechAtBloomberg.com

Bloomberg

Engineering

## Example #5: **`pyspark.pandas.DataFrame.apply(f)`**

```
import pyspark.pandas as ps
df = ps.DataFrame(range(1_000_000))
print(df.apply(sum, axis=0))
```

This prints:

```
$ python examples/example5.py
0   49995000
0  149995000
0  249995000
0  349995000
0  449995000
```

TechAtBloomberg.com

Bloomberg

Engineering

## `pyspark.pandas.DataFrame.apply(f)`

- Do not naively translate pandas code
- It may appear to work for small datasets (e.g., your tests!)
- Read documentation! pyspark.pandas docs warn about this prominently!

## Migration to PySpark: Conclusions

- Our dependency exposes a NumPy interface, so we must eventually convert to NumPy – it is unavoidable

- Strategy:
  - Aggregate each partition, then aggregate results
  - Use `pyspark.sql.DataFrame.mapInPandas`

- Danger! Floating point addition/multiplication is not commutative or associative – this will give slightly different answers

## `pyspark.sql.DataFrame.mapInPandas(f)`

- Takes a function `f` to map over the DataFrame
- `f` has type `Iterator[pd.DataFrame] -> Iterator[pd.DataFrame]`
- PySpark distributes data and work across nodes

## Example #6: More scalable PySpark code

```
import pandas as pd
from pyspark.sql import SparkSession
import finance

spark = SparkSession.builder.getOrCreate()
df = spark.read.parquet("/examples/big_data/")

def aggregate(dfs):
   return (pd.DataFrame(
       finance.weighted_avg(df.to_numpy())
       for df in dfs),)

averages = df.mapInPandas(aggregate, df.schema)

print(finance.weighted_avg(averages.toPandas().to_numpy()))
```

Step 1: Aggregate data into list of averages

Step 2: Aggregate into overall average

## Example #6: More scalable PySpark code

```
$ spark-submit.sh /examples/example6.py

spark-worker-2  | 24/09/02 23:40:55 INFO Worker:
Executor app-20240902234033-0000/1 finished with
state EXITED message Command exited with code 137
exitStatus 137
```

- OOM killed again!
- The executors are going OOM even on small chunks!

TechAtBloomberg.com

Bloomberg

Engineering

## How can we profile memory usage of our executor code?

- Spark 3.5 has built-in memory profilers
    - Can profile memory use of Python/Java/Scala executor code
    - Uses memory_profiler for Python
- cProfile
    - Can profile run time, but not memory
- Memray
    - Can profile memory use of any Python code
    - Can see native stack frames!

## PySpark's built-in memory profiler

1. Set `spark.python.profile.memory=true`
2. Add `sc.show_profiles()` at the end of your job
3. View your detailed memory profile!

## Spark's built-in memory profiler

```
/opt/spark/python/lib/pyspark.zip/pyspark/sql/udf.p
y:350: UserWarning: Profiling UDFs with iterators
input/output is not supported.
```

- Internally, `df.mapInPandas` uses a UDF
- UDF takes iterator input
- Not supported by PySpark profiler

## Example #7: Memray instrumented code

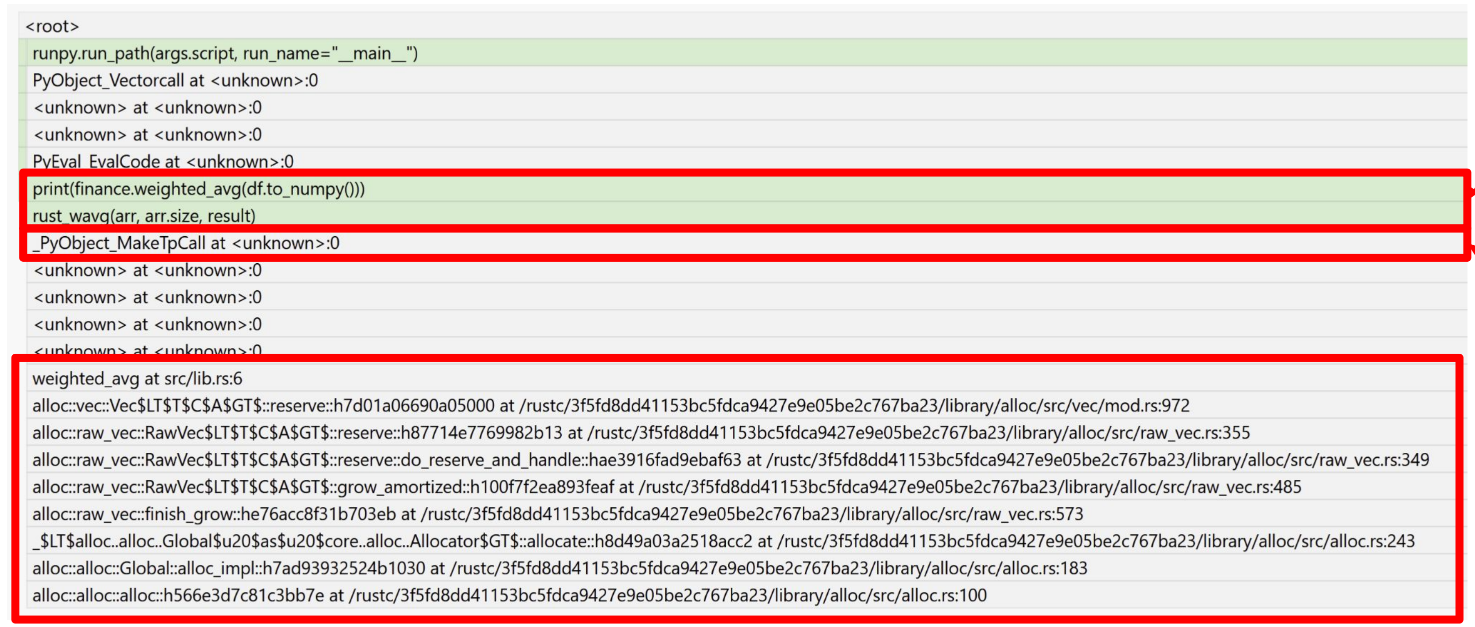
```
def aggregate(dfs):
  filename = f"/examples/memray{uuid4()}.bin"
  with memray.Tracker(filename, native_traces=True):
    return (pd.DataFrame(
      finance.weighted_avg(df.to_numpy())
      for df in dfs),)
```

TechAtBloomberg.com

Bloomberg

Engineering

# Example #7: Memray instrumented code



```
<root>
runpy.run_path(args.script, run_name="__main__")
PyObject_Vectorcall at <unknown>:0
<unknown> at <unknown>:0
<unknown> at <unknown>:0
PyEval_EvalCode at <unknown>:0
print(finance.weighted_avg(df.to_numpy()))
rust_wavg(arr, arr.size, result)
_PyObject_MakeTpCall at <unknown>:0
<unknown> at <unknown>:0
<unknown> at <unknown>:0
<unknown> at <unknown>:0
<unknown> at <unknown>:0
weighted_avg at src/lib.rs:6
alloc::vec::Vec$LT$T$C$A$GT$::reserve::h7d01a06690a05000 at /rustc/3f5fd8dd41153bc5fdca9427e9e05be2c767ba23/library/alloc/src/vec/mod.rs:972
alloc::raw_vec::RawVec$LT$T$C$A$GT$::reserve::h87714e7769982b13 at /rustc/3f5fd8dd41153bc5fdca9427e9e05be2c767ba23/library/alloc/src/raw_vec.rs:355
alloc::raw_vec::RawVec$LT$T$C$A$GT$::reserve::do_reserve_and_handle::hae3916fad9ebaf63 at /rustc/3f5fd8dd41153bc5fdca9427e9e05be2c767ba23/library/alloc/src/raw_vec.rs:349
alloc::raw_vec::RawVec$LT$T$C$A$GT$::grow_amortized::h100f7f2ea893feaf at /rustc/3f5fd8dd41153bc5fdca9427e9e05be2c767ba23/library/alloc/src/raw_vec.rs:485
alloc::raw_vec::finish_grow::he76acc8f31b703eb at /rustc/3f5fd8dd41153bc5fdca9427e9e05be2c767ba23/library/alloc/src/raw_vec.rs:573
_$LT$alloc..alloc..Global$u20$as$u20$core..alloc..Allocator$GT$::allocate::h8d49a03a2518acc2 at /rustc/3f5fd8dd41153bc5fdca9427e9e05be2c767ba23/library/alloc/src/alloc.rs:243
alloc::alloc::Global::alloc_impl::h7ad93932524b1030 at /rustc/3f5fd8dd41153bc5fdca9427e9e05be2c767ba23/library/alloc/src/alloc.rs:183
alloc::alloc::alloc::h566e3d7c81c3bb7e at /rustc/3f5fd8dd41153bc5fdca9427e9e05be2c767ba23/library/alloc/src/alloc.rs:100
```

Python stack frame

C stack frame

Rust stack frames!

TechAtBloomberg.com

Bloomberg
Engineering

## Pinpointing the issue in our Rust library

```
#[no_mangle]
pub extern "C" fn weighted_avg(indata: *const f64, size: isize, result: *mut f64) ->
() {
    let mut total_weight: f64 = 0.0;
    let mut dot_product: f64 = 0.0;
    let mut scratch: Vec<f64> = vec![];
    scratch.reserve(1000000000);

    for i in (0..size).step_by(2) {
            total_weight += unsafe { *indata.offset(i) };
            dot_product += unsafe { *indata.offset(i) * *indata.offset(i+1) };
    }

    unsafe {
            *result = total_weight;
            *result.offset(1) = dot_product / total_weight;
    }
}
```

Let's get rid of this allocation!

TechAtBloomberg.com

Bloomberg Engineering

## Success!

```
$ python final_example.py
(np.float64(-1301.4259223484928),
np.float64(21.851472001775814))
```

TechAtBloomberg.com

Bloomberg

Engineering

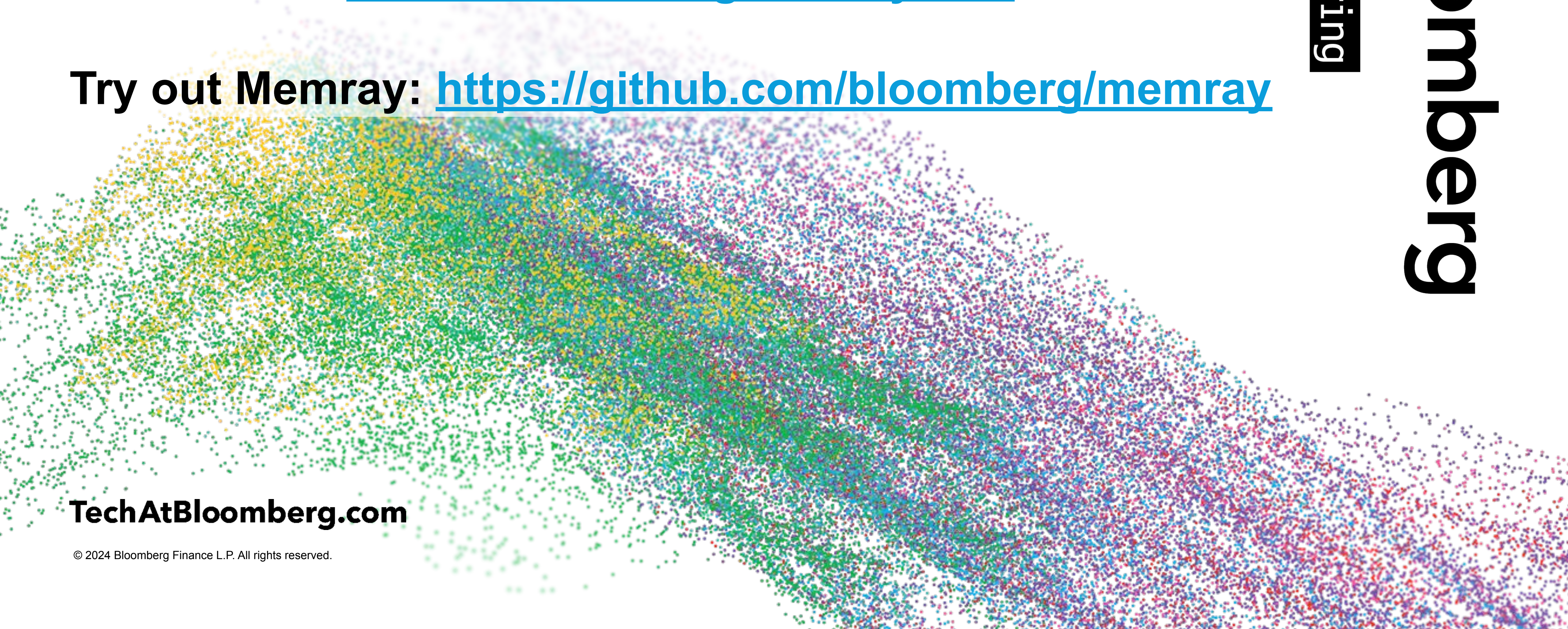
## Conclusions

- Be careful when converting pandas code to PySpark!
- pandas on Spark: While convenient, naive usage is dangerous
- Use **Memray** to diagnose memory issues in pandas, driver code, executor code, Python, C/C++/Rust
- Profile, profile, profile!

# Thank you!

**Learn more: TechAtBloomberg.com/Python**

**Try out Memray: https://github.com/bloomberg/memray**



Bloomberg Engineering

**TechAtBloomberg.com**