

A photograph of a river with a waterfall. A salmon is captured mid-jump, leaping over the white, foamy water of the falls. The surrounding rocks are dark and mossy. The overall scene is dynamic and natural.

Going Upstream

Olivier Grisel

October 2017

Random Worker Restarts with Dask Distributed



Search or jump to...

[Pull requests](#) [Issues](#) [Marketplace](#) [Explore](#)



[dask](#) / [dask](#)



Used by

7,996

Watch

200

★ Star

4,890

Fork

763

[Code](#)

[Issues](#) 371

[Pull requests](#) 46

[Projects](#) 0

[Wiki](#)

[Security](#)

[Insights](#)

[Settings](#)

Running out of memory due to task prioritization by scheduler #2470

Edit

New Issue

Closed

olivier-lacroix opened this issue on Jun 17, 2017 · 18 comments



olivier-lacroix commented on Jun 17, 2017

Hi,

I am using dask (0.14.3) (Python 3.6.1 :: Anaconda 4.4.0 (64-bit)) and have run into an issue using either the distributed or multiprocessing scheduler on a single machine.

I am applying a function through a `map_partitions` on a large dask dataframe (read from several h5 files) that returns a result with a small RAM footprint from a relatively large dask partition, and then `compute` the result.

Doing this, the memory footprint increases until the system runs out of it and the kernel kills a couple of workers. Looking at task progress with the distributed scheduler, a lot of partitions are being read in advance (2-3 per worker), and then wait in RAM for the function to be applied, and thus filling up the RAM unnecessarily.

Assignees

No one—assign yourself

Labels

None yet

Projects

None yet

Milestone

No milestone

Notifications



Search or jump to file

Pull requests Issues Marketplace Explore



dask / distributed



Used by

4,539

Watch

58

Star

813

Fork

322

Code

Issues 349

Pull requests 37

Projects 0

Wiki

Security

Insights

Settings

Aggressively GC on high memory usage in worker #1488

Edit

Merged

mrocklin merged 1 commit into `dask:master` from `ogrisel|aggressive-gc` on Oct 23, 2017

Conversation 44

Commits 1

Checks 0

Files changed 3

+67 -9



ogrisel commented on Oct 20, 2017

Member

This is a possible fix for some of the spurious worker restarts reported in the discussion of [dask/dask#2470](#).

I ran the test script of [dask/dask#2470 \(comment\)](#) with additional logs before and after the call to `gc.collect()` and I confirm that many unnecessary restarts can be avoided this way when memory is tight.



ogrisel commented on Oct 20, 2017

Author Member

I don't think there is an easy way to add a test for this.

Reviewers

mrocklin

pitrou

Assignees

No one—assign yourself

Labels

None yet

Projects

None yet

(Rarer) Random Worker Restarts with Dask
Distributed

Resilience

[User code failures](#)
[Closed Network Connections](#)
[Hardware Failures](#)
[Scheduler Failure](#)
[Restart and Nanny Processes](#)

Scheduling Policies

[Scheduling State](#)
[Worker](#)
[Work Stealing](#)

ADDITIONAL FEATURES

[Actors](#)
[Adaptive Deployments](#)
[Asynchronous Operation](#)
[Configuration](#)
[Local Cluster](#)
[IPython Integration](#)
[Joblib Integration](#)
[Publish Datasets](#)
[Data Streams with Queues](#)
[Worker Resources](#)
[Submitting Applications](#)

Resilience

Software fails, Hardware fails, network connections fail, user code fails. This document describes how `dask.distributed` responds in the face of these failures and other known bugs.

User code failures

When a function raises an error that error is kept and transmitted to the client on request. Any attempt to gather that result or any dependent result will raise that exception.

```
>>> def div(a, b):
...     return a / b

>>> x = client.submit(div, 1, 0)
>>> x.result()
ZeroDivisionError: division by zero

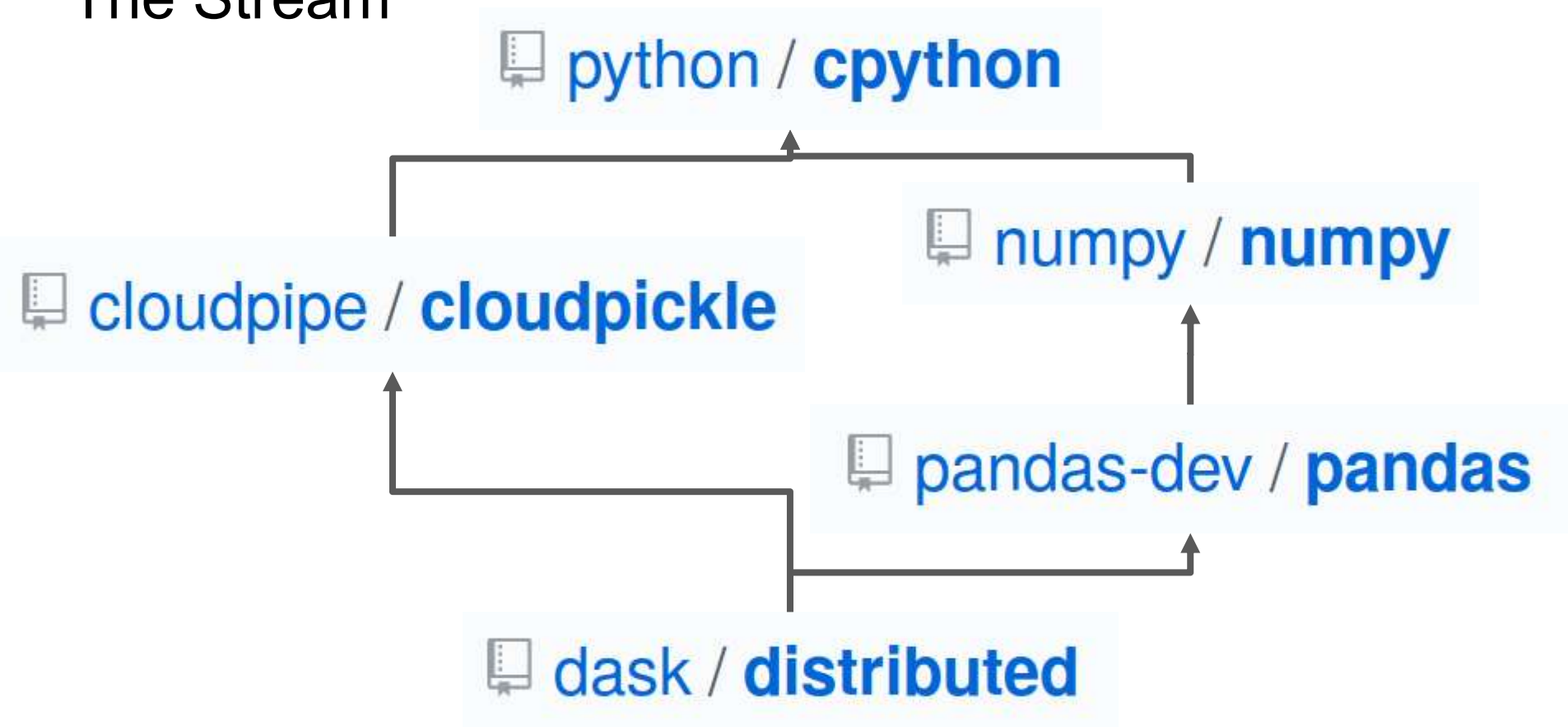
>>> y = client.submit(add, x, 10)
>>> y.result() # same error as above
ZeroDivisionError: division by zero
```

This does not affect the smooth operation of the scheduler or worker in any way.

Closed Network Connections

If the connection to a remote worker unexpectedly closes and the local process appropriately raises an `IOError` then the scheduler will reroute all pending computations to other workers.

The Stream



Root cause(s)

- Under memory pressure dask workers dumps intermediate objects to disk
 - Most data-science objects have internal data-buffers based on numpy
 - Numpy makes intermediate copies of the data buffer when pickling
 - Some of those extra memory copies would trigger excessive memory usage causing the dask worker to be killed and restarted...
-
- There no natural way to pickle read-write buffers without making extra copies using the pickle protocol and implementation in CPython
 - Even for simple readonly bytes, pickle module would make extra copies...

December 2017

Issues

[Random issue](#)

Summaries
[Issues with patch](#)
[Easy issues](#)
[Stats](#)
User
[Login \(OpenID possible\)](#)



 Remember me?

[Register](#)
[Lost your login?](#)
Administration
[User list](#)
classification
Title: [pickle.dump allocates unnecessary temporary bytes / str](#)
Type: [performance](#) **Stage:** [resolved](#)
Components: [Library \(Lib\)](#) **Versions:** [Python 3.7](#)
process
Status: [closed](#) **Resolution:** [fixed](#)
Dependencies: **Superseder:**
Assigned To: **Nosy List:** [Olivier.Grisel](#), [pitrou](#), [serhiy.storchaka](#)
Priority: [normal](#) **Keywords:** [patch](#)

Created on **2017-11-09 18:11** by [Olivier.Grisel](#), last changed **2018-01-12 22:29** by [serhiy.storchaka](#). This issue is now **closed**.

Pull Requests

URL	Status	Linked	Edit
PR 4353	merged	python-dev , 2017-11-09 18:14	
PR 5114	merged	serhiy.storchaka , 2018-01-06 19:15	
PR 5154	merged	serhiy.storchaka , 2018-01-11 11:13	

Messages (39)

[msg305975 - \(view\)](#) **Author:** [Olivier Grisel \(Olivier.Grisel\) *](#) **Date:** 2017-11-09 18:11

I noticed that both `pickle.Pickler` (C version) and `pickle._Pickler` (Python version) make unnecessary memory copies when dumping large str, bytes and bytearray objects.

This is caused by unnecessary concatenation of the opcode and size header with the large bytes payload prior to calling `self.write`.

For protocol 4, an additional copy is caused by the framing mechanism.

I will submit a pull request to fix the issue for the Python version. I am not sure how to test this properly. The `BigmemPickleTests`



bpo-31993: do not allocate large temporary buffers in pickle dump #4353

[Edit](#)**Merged** serhiy-storchaka merged 39 commits into `python:master` from `ogrisel:issue-31993-pypickle-dump-rem-optio` on Jan 6, 2018

Conversation 109 Commits 39 Checks 0 Files changed 6

+297 -50



ogri sel commented on Nov 9, 2017 • edited +

Contributor + ⌨ ...

For all protocols: avoid concatenating large bytes and str with their opcode and size header but instead issue an individual call to `self.write(data)`.

For protocol 4: if the size of the opcode + size header + data is larger than the target frame size, commit the current frame and bypass `io.BytesIO` to write the next frame directly to the underlying file object.

<https://bugs.python.org/issue31993>

Edit: this now includes changes both the Python and C implementations of the picklers.



1

Reviewers

- vstinner
- pitrou
- serhiy-storchaka

Assignees

No one assigned.

Labels

- CLA signed
- type-enhancement

Projects

None yet.

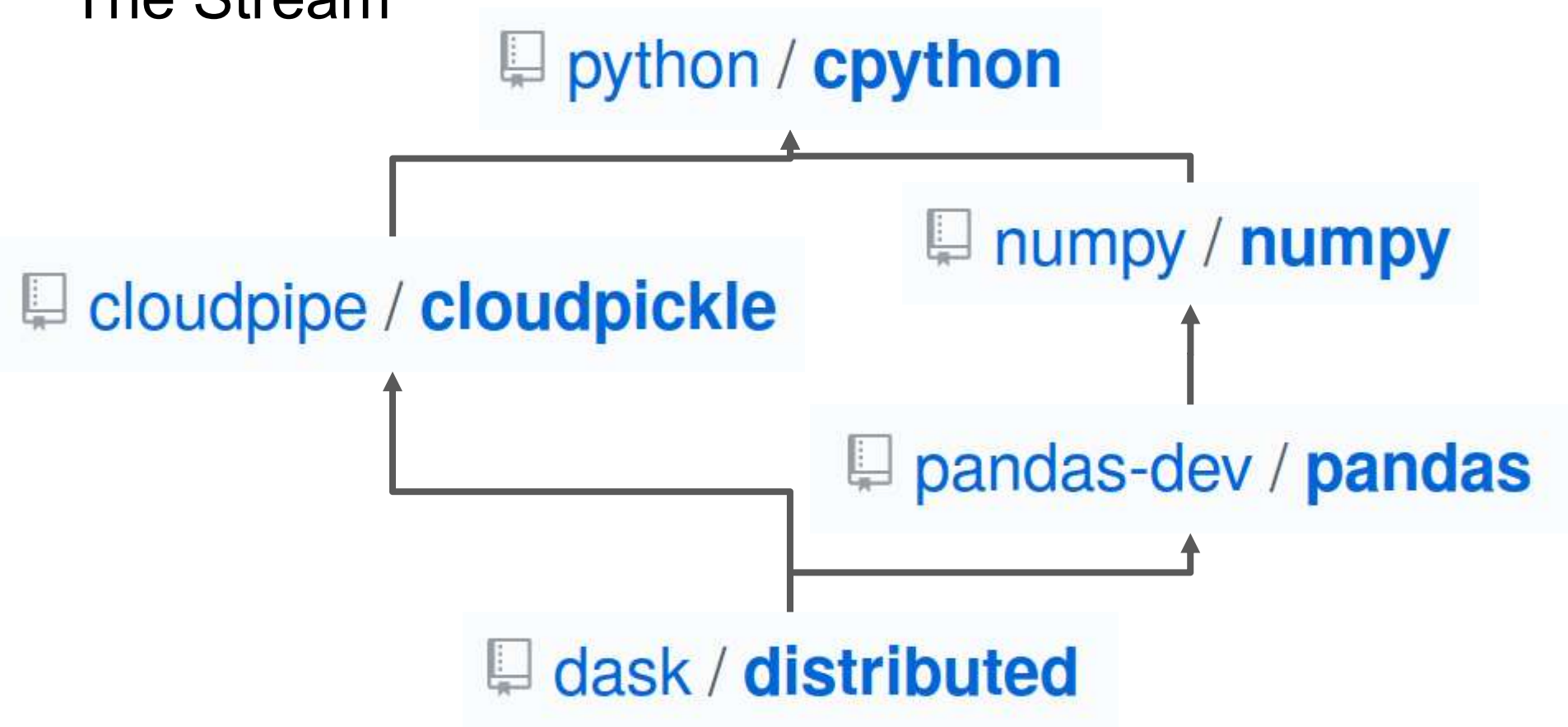


the-knights-who... commented on Nov 9, 2017

+ ⌨ ...

<https://github.com/python/cpython/pulls>

The Stream





[WIP] No-copy semantics for large memoryviews #138

Edit

Closed ogrisel wants to merge 31 commits into [cloudpipe:master](#) from [ogrisel:nocopy-memoryviews](#)

Conversation 20 Commits 31 Checks 0 Files changed 5

+81B -8



ogrisel commented on Dec 4, 2017 · edited

Member

Here is a PR that backports some of the work done for large bytes in upstream CPython 3.7 or 3.8 in the following PR: [python/cpython#4353](#).

The goal is to make the `memoryview` support of `cloudpickle` benefit from it and implement `nocopy` semantics and later any nested numpy arrays datastructures (e.g. pandas dataframes, scipy sparse arrays, large scikit-learn `RandomForestClassifier`...).

In particular, this would make it possible for dask distributed to spill any large numpy-based datastructure to disk without making any temporary in-memory copy on workers that are close to their memory usage limit.

Please do not merge this PR while the upstream CPython PR is still under review.

- prevent `_memoryview_from_bytes` to ever mutate single bytes interned bytes instances,
- add a battery of unittests to check edge cases with bytes mutability,
- take the numpy serializer out of the test suite and make it available to users,
- fix pickling for numpy arrays with the object dtype,
- wait for the final review of [python/cpython#4353](#),

Reviewers

pitrou

ogrisel

Assignees

No one—assign yourself

Labels

None yet

Projects

None yet

Milestone

0.6

```
def _is_safe_to_mutate(data_holder):
    """Helper function, see _memoryview_from_bytes for details."""
    if not RUNNING_CPYTHON:
        # sys.getrefcount and ob_sstate are not always available on other
        # platforms (e.g. PyPy): better be conservative.
        return False

    # If the following call to sys.getrefcount returns 2, it means that there
    # is one reference from data_holder and one from the temporary arguments
    # datastructure of the sys.getrefcount call. It is therefore safe to reuse
    # the memory buffer of the bytes object as writeable buffer to back the
    # memoryview: this buffer is no longer reachable from anywhere else.
    safe_to_mutate = sys.getrefcount(data_holder[0]) <= 2
    data = data_holder[0]
    if safe_to_mutate and isinstance(data, str):
        # Python 2 str instances are bytes that can be interned, make sure
        # that this is not the case.
        safe_to_mutate = not _Py2StrStruct.from_address(id(data)).is_interned()
    return safe_to_mutate
```

```
safe_to_mutate = _is_safe_to_mutate(data_holder)
data = data_holder[0]
del data_holder[:]
if readonly:
    # A memoryview of a bytes object is readonly by default.
    view = memoryview(data)
else:
    # Python 3.4 implementation of memoryviews backed by ctypes buffers
    # has a bug: # https://bugs.python.org/issue19803
    if safe_to_mutate and PY_MAJOR_MINOR != (3, 4):
        buffer = ctypes.cast(data, ctypes.POINTER(ctypes.c_char))
        addr = ctypes.addressof(buffer.contents)
        array = (ctypes.c_char * len(data)).from_address(addr)
        # Store a reference to the backing bytes object to make GC work
        # correctly. We hide the object itself in a closure to prevent
        # unsuspecting users to access it.








        def _hidden_buffer_ref():
            raise TypeError("Access to mutable buffer with id %d is unsafe"
                            % id(data))

        array._hidden_buffer_ref = _hidden_buffer_ref
    view = memoryview(array)
```


 Closed

[WIP] No-copy semantics for large memoryviews #138

ogrisel wants to merge 31 commits into [cloudpickle/master](#) from [ogrisel:nocopy-memoryviews](#) 

-  Small reorg, better wording. #41F82D
-  cosmetics #4108F8
-  Better comment  #663740
-  Fixed wrong assertions in tests for PyPy ✓ #2c598a
-  Single element bytes mutation is actually safe ✓ #10562f
-  Improve tests ✓ #3a22a3



ogrisel commented on Dec 12, 2017

Author

Member



@lmmmmmm thanks for the feedback. I think I have addressed the safety concerns in the latest batch of commits. This ctypes-based implementation should mirror all the checks done in C in pandas.




lmmmmmm commented on Dec 12, 2017

Member



This is really impressive work!

 ogrisel added some commits on Dec 12, 2017

-  cosmetics. ✓ #578844
-  Nocopy semantics for numpy arrays for Python 3 enabled by default ✓ #d0ff10

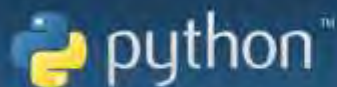
 ogrisel referenced this pull request on Dec 18, 2017

WIP Experimental integration of nocopy cloudpickle with bytelist API #1643

 Closed

There must be a better way...

March 2018

[Donate](#)[Menu](#)[A A](#)[Socialize](#)

Tweets by @ThePSF

The PSF

The Python Software Foundation is the organization behind Python. Become a member of the PSF and help advance the software and our mission.

[Python](#) >>> [Python Developer's Guide](#) >>> [PEP index](#) >>> [PEP 574 -- Pickle protocol 5 with out-of-band data](#)

PEP 574 -- Pickle protocol 5 with out-of-band data

PEP:	574
Title:	Pickle protocol 5 with out-of-band data
Author:	Antoine Pitrou <solipsis at pitrou.net>
BDFL-Delegate:	Nick Coghlan
Status:	Final
Type:	Standards Track
Created:	23-Mar-2018
Python-Version:	3.8
Post-History:	28-Mar-2018, 30-Apr-2019
Resolution:	https://mail.python.org/pipermail/python-dev/2019-May/157284.html

[Help](#)[Donate](#)[Log in](#)[Register](#)

pickle5 0.0.8

[Latest version](#)

```
pip install pickle5
```



Last released: May 7, 2019

Experimental backport of the pickle 5 protocol (PEP 574)

Navigation

[Project description](#)[Release history](#)[Download files](#)

Project links

[Homepage](#)

Project description

This package backports the features and APIs added in [PEP 574](#). It should work with Python 3.6 and 3.7.

Basic usage is similar to the `pickle` module, except that the module to be imported is `pickle5`:

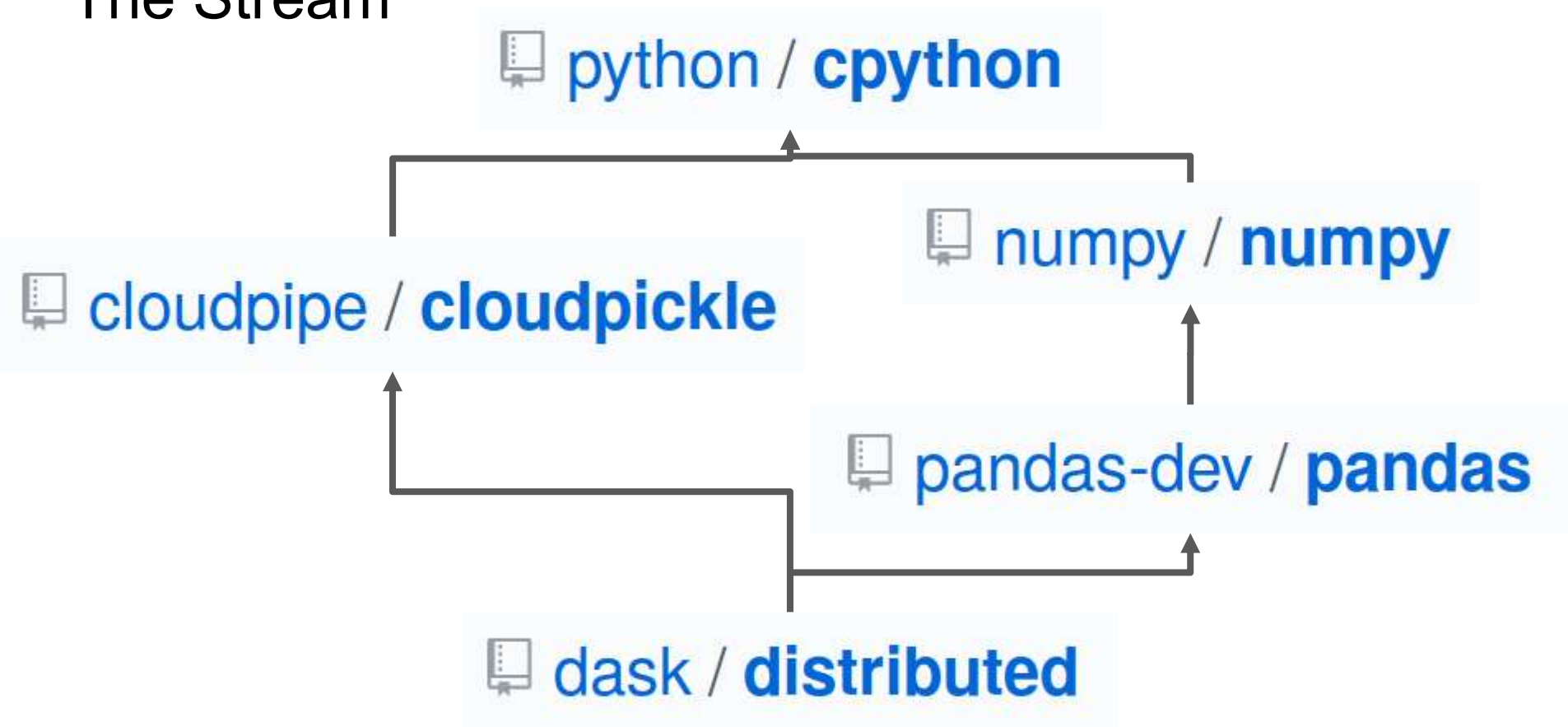
```
import pickle5 as pickle

pb = pickle.PickleBuffer(b"foo")
data = pickle.dumps(pb, protocol=5)
assert pickle.loads(data) == b"foo"
```

Detailed documentation can be found in [PEP 574](#) and the standard [pickle documentation](#).

May 2018

The Stream





Leverage the new PEP 574 for no-copy pickling of contiguous arrays #11161

Edit

New Issue



ogrisel opened this issue on May 25, 2018 · 24 comments



ogrisel commented on May 25, 2018 · edited →

Contributor



PEP 574 (scheduled for Python 3.8) introduces pickle protocol 5 with support for no-copy pickling of large mutable buffers.

I made a small proof-of-concept benchmark script using [@pitrou's pickle5](#) backport of his draft implementation of PEP 547.

See: <https://gist.github.com/ogrisel/a2b0e5ae4987a398caa719277cb3b90a>

The meat lies in the following reducer:

```
from pickle5 import PickleBuffer

def _array_from_buffer(buffer, dtype, shape):
    return np.frombuffer(buffer, dtype=dtype).reshape(shape)

def reduce_ndarray_pickle5(a):
    # This reducer assumes protocol 5 as currently there is no way to register
```

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Notifications

Unsubscribe



You're receiving notifications because you modified the open/close state.

September 2018



ENH: implementation of array_reduce_ex #12011



charris merged 2 commits into `numpy:master` from `pierrreglaser:implement-reduce-ex` on Oct 10, 2018

Conversation 73

Commits 2

Checks 1

Files changed 4

+232 -0



pierrreglaser commented on Sep 21, 2018 • edited •

Contributor

Follow up on #11161.

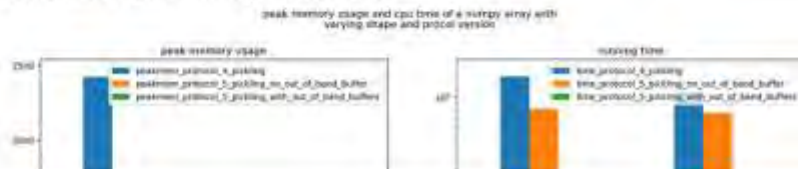
In order to enable no-copy pickling as mentioned in the [PEP 574 proposal](#), numpy arrays need to have:

- a `__reduce_ex__` method, returning metadata and a buffer on their internal data
- a method that reconstructs an array from a buffer, a `dtype` and a `shape`

The first one is mostly inspired from the `array_reduce` method in `core/src/multidarray/methods.c`

The second one is a wrapper written in python around `np.frombuffer(...).reshape(...)`

Here is the decrease in peak memory and cpu time:



Reviewers

ogriuel

mattlp

pitrou

tylerjreddy

Assignees

No one assigned

Labels

01 - Enhancement

component: numpy.cxx

component: numpy.lib

Projects



ARROW-2660: [Python] Experimental zero-copy pickling #2161

pitrou wants to merge 3 commits into apache:master from pitrou:ARROW-2660-zero-copy-pickl

```
{'a', (488890,)},  
{'b', (888888,)}
```

@mrocklin



pitrou commented on Jul 25, 2018

Author Contributor + 👤 ...

I created [ARROW-2913](#) for the issue that exported buffers lose the data type. @mrocklin your informed opinion on that one could be useful.

(note it doesn't block this PR)



wesm commented on Jul 25, 2018

Member + 👤 ...

Nice and informative benchmarks. The copying of memory in unpickling with NumPy arrays etc. has been a long-standing gripe of mine



xhochy commented on Jul 26, 2018

Member + 👤 ...

This looks really nice. @pitrou What is the best way to keep updated of the status of a PEP? Poll the website?

I guess, we wait with merging until the PEP is accepted?



June 2019



bpo-36785: PEP 574 implementation #7076



pitrou merged 5 commits into python:master from pitrou:pickle5 a day ago

Conversation 60

Commits 5

Checks 0

Files changed 19

+1,896 -240



pitrou commented on May 23, 2018 • edited

Member

<https://bugs.python.org/issue36785> the-knights-who-say-ni added the **CLA signed** label on May 23, 2018 bedevere-bot added the **awaiting merge** label on May 23, 2018

pitrou requested a review from python/windows-team as a code owner on May 23, 2018

 pitrou added **DO-NOT-MERGE** **type-enhancement** and removed **awaiting merge** labels on May 23, 2018 pitrou force-pushed the pitrou:pickle5 branch 3 times, most recently from `cef2ed0` to `8943edf`

Reviewers

ogrisel



pierreglaser



terryfreedy



vstinner



windows-team



Assignees

No one assigned

Labels

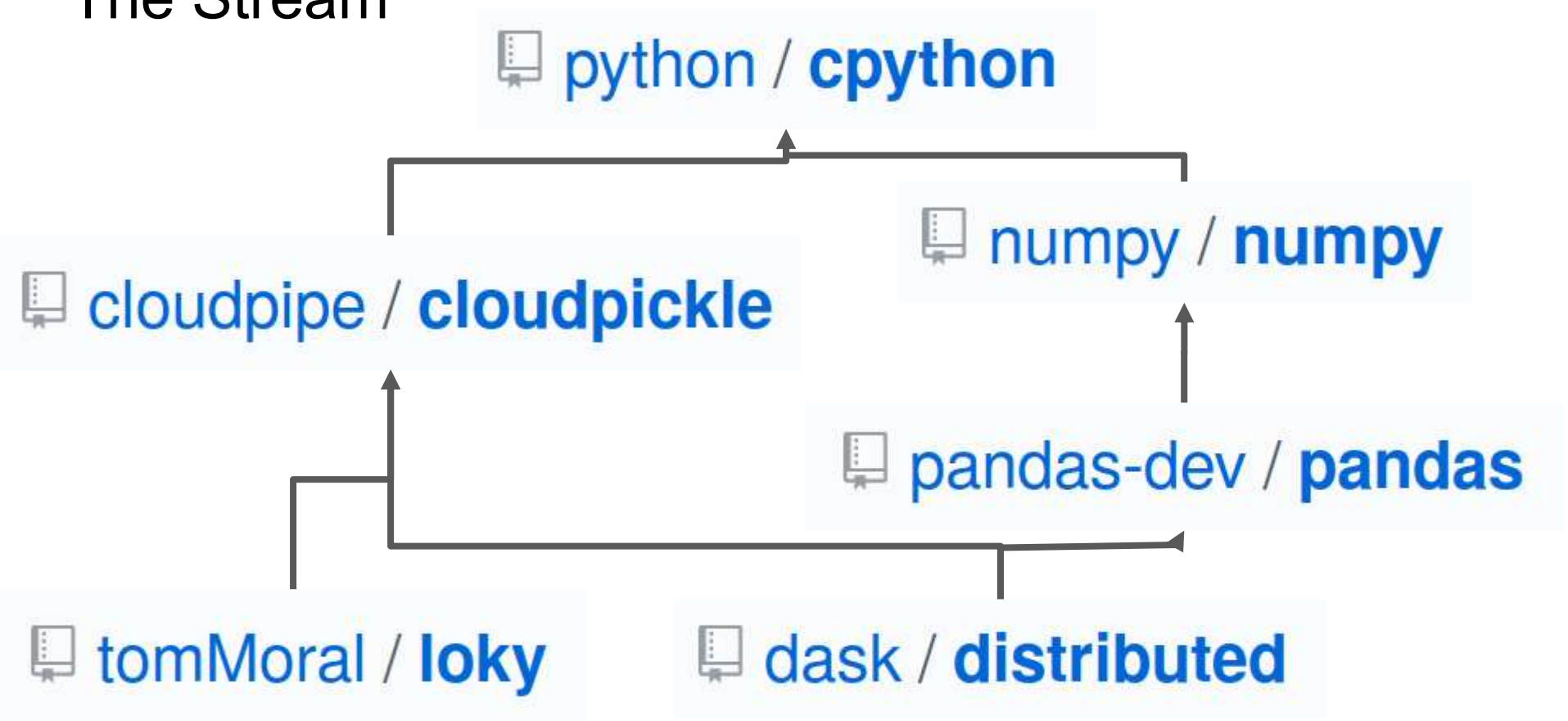
CLA signed**type-enhancement**

Projects

Fin?

Meanwhile in February 2019

The Stream





ENH: derive from C-pickler for fast serialization #253

Edit

pierreglaser wants to merge 44 commits into `cloudpipe:master` from `pierreglaser:fast-cloudpickle`

Conversation 56

Commits 44

Checks 0

Files changed 7

+742 -169



pierreglaser commented on Mar 7 • edited

Contributor

Summary:

This PR proposes a new `cloudpickler` class, that inherits from the C `_pickle.Pickler` instead of the python `pickle.Pickler`, allowing 10x+ speedups for the serialization of large builtin objects such as dicts, lists..

Disclaimer: a new start

Moving from the python to the c `Pickler` requires a fair amount of changes. For this reason, instead of simply adapting the current code to respect the new constraints, I started back from scratch. This allows a new, clean API and structure, that will be hopefully easier to understand for everyone.

I made a lot of comments, (sometimes overly verbose), to ease the review process of this PR. Eventually, I hope the information they contain can be

Reviewers

ogrisel

Assignees

No one—assign yourself

Labels

None yet

Projects

None yet

Milestone

No milestone

Notifications



bpo-35900: Enable custom reduction callback registration in `_pickle`

#12499



pitrou merged 25 commits into `python:master` from `pierreglaser:add-global-hook/cpickler` 19 days ago



Conversation 38



Commits 25



Checks 0



Files changed 6

+227 -24



pierreglaser commented on Mar 22 • edited

Contributor

The C implementation of the `Pickler` class does not allow to register custom pickling behavior for classes and functions.

This PR enables it by adding a hook that will call a user-defined callable for those types. If no callable is registered, then the `pickler` simply fallbacks the `save_global` method, as before.

See also:

[cloudpipe/cloudpickle#253](#)

<https://bugs.python.org/issue35900>

Reviewers

ZackerySpytz



ogrisel



pitrou



Assignees

No one assigned

Labels

CLA signed

Projects

None yet



the-knights-who-say-ni added the **CLA signed** label on Mar 22



bedevere-bot added the **awaiting review** label on Mar 22



bpo-35900: Add a state_setter arg to save_reduce #12588

Merged

pitrou merged 29 commits into `python:master` from `pierrreglaser:give-a-state-setter-to-save-reduce` 19 days ago

Conversation 42

Commits 29

Checks 0

Files changed 5

+114 -15



pierrreglaser commented on Mar 27 • edited

Contributor

Second PR of [bpo-35900](#).

What

The aim of this PR is to allow users to specify custom state setting methods for types they have no control on. This is done via adding a `state_setter` keyword argument in `Pickler.save_reduce`, expected to be a callable. This callable will be called at unpickling time with the following signature `state_setter(obj, state, slotstate)`.

Why

A practical use case is the one of [cloudpickle](#) that provides extended serialization procedures, especially for functions and classes. In `cloudpickle`, functions and classes are reconstructed from scratch at unpickling time. As for other built-in types reconstructed from scratch (list, dicts...), circular-references are handled by initializing a placeholder object, and then updating it in a state setting phase. This state setting phase currently

Reviewers

ogrisel ✓

pitrou ✓

Assignees

No one assigned

Labels

CLA signed

Projects

None yet

Milestone

No milestone

October 2019

PEP 569 -- Python 3.8 Release Schedule

PEP:	569
Title:	Python 3.8 Release Schedule
Author:	Łukasz Langa <lukasz at python.org>
Status:	Active
Type:	Informational
Created:	27-Jan-2018
Python-Version:	3.8

Open Source is not a
zero-sum game!

Investing in people
(via code-reviews)
is highly-fruitful!

Investment in shared
knowledge and trust!



Pierre Glaser
paraglider



Joe Jevnik
|||



Matthew Rocklin
mrocklin



Antoine Pitrou
pitrou



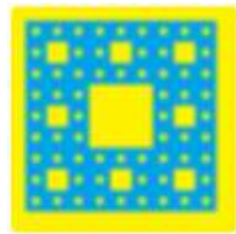
Serhiy Storchaka
serhiy-storchaka



Matti Picus
mattip



Victor Stinner
vstinner



Terry Jan Reedy
terryreedy



Tyler Reddy
tylerjoreddy

This value can only be
unlocked via long term
support...

Thank you to all employers
who support OS developers!

scikit-learn @ Inria Fondation partners:



Thank you for your
attention!

and see you next time!

Image Credits

Upstream salmon:

<https://www.flickr.com/photos/umpquawild/2493000462/in/photolist-4NihbA-21pAVYd-JaGNLz-4MzvLZ-yK3q1k>