copy.deepcopy vs pickle

Asked 15 years, 3 months ago Modified 8 years, 9 months ago Viewed 19k times

33

I have a tree structure of widgets e.g. collection contains models and model contains widgets. I want to copy whole collection, `copy.deepcopy` is faster in comparison to 'pickle and de-pickle'ing the object but cPickle as being written in C is much faster, so

1. Why shouldn't I(we) always be using cPickle instead of deepcopy?
2. Is there any other copy alternative? because pickle is slower then deepcopy but cPickle is faster, so may be a C implementation of deepcopy will be the winner

Sample test code:

```
import copy
import pickle
import cPickle

class A(object): pass

d = {}
for i in range(1000):
    d[i] = A()

def copy1():
    return copy.deepcopy(d)

def copy2():
    return pickle.loads(pickle.dumps(d, -1))

def copy3():
    return cPickle.loads(cPickle.dumps(d, -1))
```

Timings:

```
>python -m timeit -s "import c" "c.copy1()"
10 loops, best of 3: 46.3 msec per loop

>python -m timeit -s "import c" "c.copy2()"
10 loops, best of 3: 93.3 msec per loop

>python -m timeit -s "import c" "c.copy3()"
100 loops, best of 3: 17.1 msec per loop
```

python pickle deep-copy

Share edited Feb 28, 2016 at 0:41 asked Sep 11, 2009 at 12:39

Improve this question

Follow

Anurag Uniyal
88.6k ● 41 ● 180 ● 221

2 This is a very useful observation. – bayer Sep 11, 2009 at 12:56

hm, shouldn't you be comparing `pickle` with `copy.copy` ? – SilentGhost Sep 11, 2009 at 12:59

Does it depend at all on the structure of what you're copying? As in, does it make a difference if, say, the memory allocated to the copied objects is noncontiguous or the underlying pointers have a long chain to follow? – Gordon Seidoh Worley Sep 11, 2009 at 13:24

2 @SilentGhost, pickle does a deepcopy – Anurag Uniyal Sep 11, 2009 at 14:33

5 Answers

Sorted by: Highest score (default)

37

Problem is, pickle+unpickle can be faster (in the C implementation) because it's *less general* than deepcopy: many objects can be deepcopied but not pickled. Suppose for example that your class `A` were changed to...:

```
class A(object):
  class B(object): pass
  def __init__(self): self.b = self.B()
```

now, `copy1` still works fine (A's complexity slows it downs but absolutely doesn't stop it); `copy2` and `copy3` break, the end of the stack trace says...:

```
  File "./c.py", line 20, in copy3
    return cPickle.loads(cPickle.dumps(d, -1))
PicklingError: Can't pickle <class 'c.B'>: attribute lookup c.B failed
```

I.e., pickling always assumes that classes and functions are top-level entities in their modules, and so pickles them "by name" -- deepcopying makes absolutely no such assumptions.

So if you have a situation where speed of "somewhat deep-copying" is absolutely crucial, every millisecond matters, AND you want to take advantage of special limitations that you KNOW apply to the objects you're duplicating, such as those that make pickling applicable, or ones favoring other forms yet of serializations and other shortcuts, by all means go ahead - but if you do you MUST be aware that you're constraining your system to live by those limitations forevermore, and document that design decision very clearly and explicitly for the benefit of future maintainers.

For the NORMAL case, where you want generality, use `deepcopy` !-)

Share Improve this answer Follow

answered Sep 11, 2009 at 14:27

Alex Martelli

880k • 174 • 1.2k • 1.4k

8

You should be using deepcopy because it makes your code more readable. Using a serialization mechanism to copy objects in memory is at the very least confusing to another developer reading your code. Using deepcopy also means you get to reap the benefits of future optimizations in deepcopy.

First rule of optimization: don't.

Share Improve this answer Follow

answered Sep 11, 2009 at 13:18

wds

32.2k • 11 • 62 • 84

9 Second rule of optimization: Don't yet – Esteban Küber Sep 11, 2009 at 14:21

4 Forget the outdated rules, replacing deepcopy by cpickle, makes my project rendering 25% faster and makes my customer happy :) – Anurag Uniyal Sep 11, 2009 at 14:31

1 @wds, also i don't think it would be confusing if wrapper function to copy object is named deepcopyObject with a good comment – Anurag Uniyal Sep 11, 2009 at 14:31

1 @anurag if you're having such speed problems, by all means. Though it still looks like more of a band-aid to me. – wds Sep 14, 2009 at 8:34

2 Instead of optimizing with `cPickle`, I suggest implementing `__deepcopy__` (see my separate answer). – hans_meine Mar 24, 2014 at 20:34

2

It is *not* always the case that cPickle is faster than deepcopy(). While cPickle is probably always faster than pickle, whether it is faster than deepcopy depends on

- the size and nesting level of the structures to be copied,
- the type of contained objects, and
- the size of the pickled string representation.

If something can be pickled, it can obviously be deepcopied, but the opposite is not the case: **In order to pickle something, it needs to be fully serialized**; this is not the case for deepcopying. In particular, you can implement `__deepcopy__` very efficiently by copying a structure in memory (think of extension types), without being able to save everything to disk. (Think of suspend-to-RAM vs. suspend-to-disk.)

A well-known extension type that fulfills the conditions above may be `ndarray`, and indeed, it serves as a good counterexample to your observation: With `d = numpy.arange(100000000)`, your code gives different runtimes:

```
In [1]: import copy, pickle, cPickle, numpy

In [2]: d = numpy.arange(100000000)

In [3]: %timeit pickle.loads(pickle.dumps(d, -1))
1 loops, best of 3: 2.95 s per loop

In [4]: %timeit cPickle.loads(cPickle.dumps(d, -1))
1 loops, best of 3: 2.37 s per loop

In [5]: %timeit copy.deepcopy(d)
1 loops, best of 3: 459 ms per loop
```

If `__deepcopy__` is not implemented, `copy` and `pickle` share common infrastructure (cf. `copy_reg` module, discussed in Relationship between pickle and deepcopy).

Share Improve this answer Follow

edited May 23, 2017 at 12:10
Community Bot
1 • 1

answered Mar 24, 2014 at 20:27
hans_meine
1,921 • 1 • 18 • 30

1

Even faster would be to avoid the copy in the first place. You mention that you are doing rendering. Why does it need to copy objects?

Share Improve this answer Follow

answered Sep 11, 2009 at 14:41
Ned Batchelder
375k • 77 • 578 • 673

1 yes ideally it wouldn't need copy, as view(rendering) and model will be de-coupled but in my case, rendering does modify the model hence i need to copy model before rendering so original doesn't get modified – Anurag Uniyal Sep 11, 2009 at 15:04

2 I don't mean to beat a dead horse, but fixing the problem where rendering modifies the model will make you very happy. – Ned Batchelder Sep 11, 2009 at 17:12

I agree but it would be a costly affair to change so much code, as discussion here is not possible I have added a question stackoverflow.com/questions/1414246/… – Anurag Uniyal Sep 12, 2009 at 4:08

Short and somewhat late:

- If you have to cPickle an object anyway, you might as well use the cPickle method to deepcopy (but document)

e.g. You might consider:

```
def mydeepcopy(obj):
    try:
       return cPickle.loads(cPickle.dumps(obj, -1))
    except PicklingError:
       return deepcopy(obj)
```

Share

Improve this answer

Follow

edited Sep 29, 2013 at 19:50

Anurag Uniyal

88.6k ● 41 ● 180 ● 221

answered Sep 28, 2013 at 9:54

Lars

1,949 ● 2 ● 15 ● 26

why are my newlines disappearing, bug in stov? – Lars Sep 28, 2013 at 10:01

but that was not the question, I was already using cPickle in similar way – Anurag Uniyal Sep 28, 2013 at 18:51

1 Fastest way i could think of, apart from writing your own python c extension – Lars Sep 29, 2013 at 19:48