

Re: Joining Big Files

Source: <http://coding.derkeiler.com/Archive/Python/comp.lang.python/2007-08/msg03027.html>

- *From:* vasudevram <vasudevram@xxxxxxxxxx>
 - *Date:* Sun, 26 Aug 2007 14:45:15 -0000
-

On Aug 26, 6:48 am, Paul McGuire <pt...@xxxxxxxxxxxxxxxx> wrote:

On Aug 25, 8:15 pm, Paul McGuire <pt...@xxxxxxxxxxxxxxxx> wrote:

On Aug 25, 4:57 am, mosscliffe <mcl.off...@xxxxxxxxxxxxxxxx> wrote:

I have 4 text files each approx 50mb.

<yawn> 50mb? Really? Did you actually try this and find out it was a problem?

Try this:
import time

```
start = time.clock()
outname = "temp.dat"
outfile = file(outname,"w")
for inname in ['file1.dat', 'file2.dat', 'file3.dat', 'file4.dat']:
    infile = file(inname)
    outfile.write( infile.read() )
    infile.close()
    outfile.close()
end = time.clock()
```

```
print end-start,"seconds"
```

Re: Joining Big Files

For 4 30Mb files, this takes just over 1.3 seconds on my system. (You may need to open files in binary mode, depending on the contents, but I was in a hurry.)

-- Paul

My bad, my test file was not a text file, but a binary file.
Retesting with a 50Mb text file took 24.6 seconds on my machine.

Still in your working range? If not, then you will need to pursue more exotic approaches. But 25 seconds on an infrequent basis does not sound too bad, especially since I don't think you will really get any substantial boost from them (to benchmark this, I timed a raw "copy" command at the OS level of the resulting 200Mb file, and this took about 20 seconds).

Keep it simple.

-- Paul

There are (at least) another couple of approaches possible, each with some possible tradeoffs or requirements:

Approach 1. (Least amount of code to write – not that the others are large :)

Just use `os.system()` and the UNIX `cat` command – the requirement here is that:

a) your web site is hosted on *nix (ok, you can do it if on Windows too – use `copy` instead of `cat`, you might have to add a "cmd /c " prefix in front of the `copy` command, and you have to use the right `copy` command syntax for concatenating multiple input files into one output file).

b) your hosting plan allows you to execute OS level commands like `cat`, and `cat` is in your OS `PATH` (not `PYTHONPATH`). (Similar comments apply for Windows hosts).

```
import os
os.system("cat file1.txt file2.txt file3.txt file4.txt >
file_out.txt")
```

`cat` will take care of buffering, etc. transparently to you.

Approach 2: Read (in a loop, as you originally thought of doing) each

Re: Joining Big Files

line of each of the 4 input files and write it to the output file:

("Reusing" Paul McGuire's code above:)

```
outname = "temp.dat"
outfile = file(outname,"w")
for inname in ['file1.dat', 'file2.dat', 'file3.dat', 'file4.dat']:
    infile = file(inname)
    for lin in infile:
        outfile.write(lin)
    infile.close()
outfile.close()
end = time.clock()
```

```
print end-start,"seconds"
```

You may need to check that newlines are not removed in the above code, in the output file. Can't remember right now. If they are, just add one back with:

```
outfile.write(lin + "\n") instead of outfile.write(lin) .
```

(Code not tested, test it locally first, though looks ok to me.)

The reason why this `_may_` not be much slower than manually coded buffering approaches, is that:

- a) Python's standard library is written in C (which is fast), including use of `stdio` (the C Standard IO library, which already does intelligent buffering)
- b) OS's do I/O buffering anyway, so do hard disk controllers
- c) from some recent Python version, I think it was 2.2, that idiom "for lin in infile" has been (based on something I read in the Python Cookbook) stated to be pretty efficient anyway (and yet (slightly) more readable than earlier followed approaches of reading a text file).

Given all the above facts, it probably isn't worth your while to try and optimize the code unless and until you find (by measurements) that it's too slow – which is a good practice anyway:

[http://en.wikipedia.org/wiki/Optimization_\(computer_science\)](http://en.wikipedia.org/wiki/Optimization_(computer_science))

Excerpt from the above page (its long but worth reading, IMO):

"Donald Knuth said, paraphrasing Hoare[1],

"We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil." (Knuth, Donald. Structured Programming with go to Statements, ACM Journal Computing Surveys, Vol 6, No. 4, Dec. 1974. p.268.)

Re: Joining Big Files

Charles Cook commented,

"I agree with this. It's usually not worth spending a lot of time micro-optimizing code before it's obvious where the performance bottlenecks are. But, conversely, when designing software at a system level, performance issues should always be considered from the beginning. A good software developer will do this automatically, having developed a feel for where performance issues will cause problems. An inexperienced developer will not bother, misguidedly believing that a bit of fine tuning at a later stage will fix any problems." [2]

"

HTH

Vasudev

Vasudev Ram

<http://www.dancingbison.com>

<http://jugad.livejournal.com>

<http://sourceforge.net/projects/xtopdf>

.