

Forum:[Hack Forums](#)

Thread: [Faster methods for Python large file searching \(tried dask\)?](#)

TehTester - May 24, 2020, 2:18 pm

Hi all,

I currently have a program I've coded in python that opens large text files, and searches for keywords. It then saves those keywords into different lists.

I have tried using DASK and Pandas - however the fastest method I've found is actually just using `f.readline()` and looping through the lines one by one.

The way i've coded it, is that it takes an initial REGEX and then 3 keywords. The code then looks at the file line, applies the initial regex, then if the line is good for the regex, it then searches for the keywords and combinations on that line.

I've put the code in pastebin just because it's easier than in here: <https://pastebin.com/rYyYg4KL>

Can anyone think of anyway to speed this up? As I said I've tried dask and opening the file as a cluster but it just takes even longer.

Appreciate any help :)
Thanks.

SIM-0000001 - May 24, 2020, 4:06 pm

hmm, how big are these files? are you using a filestream (reading chunks at a time)? Regex should be find as long as you compile the statements before hand

TehTester - May 24, 2020, 4:17 pm

(May 24, 2020 04:06 PM)SIM-0000001 Wrote: [\[CITE\]](#) hmm, how big are these files? are you using a filestream (reading chunks at a time)? Regex should be find as long as you compile the statements before hand

File sizes are 5-10gb. Yeah I believe I'm using a filestream (kinda new to python sorry), as I'm not fully opening the file to memory, i'm just reading each new line. I've posted my code so please tell me I've I'm wrong.

Surely a line = insert logic is just as quick as a large regex statement. I use regex to first shape what I want found, then use line = to search what the regex returns.

I've heard `re.searchall` is very inefficient too. So staying away from that.

Thanks

SIM-0000001 - May 24, 2020, 4:21 pm

cool man, a quick way to speed it up would be to read large chunks of the file. Not every line

Start out reading 1MB at a time, it will be faster :) just do

```
Code file.read(1024 * 1024 * 1024)
```

TehTester - May 24, 2020, 4:28 pm

(May 24, 2020 04:21 PM)SIM-0000001 Wrote: [\[CITE\]](#) cool man, a quick way to speed it up would be to read large chunks of the file. Not every line

Start out reading 1MB at a time, it will be faster :) just do

```
Code file.read(1024 * 1024 * 1024)
```

Hmm okay, sorry again just maybe me being new to python but this is my read line:

```
Code with open(inputFile, buffering=MEG, encoding="utf-8") as in_f:
    line=in_f.readline().lower() #read the first line
```

How do I incorporate your file.read? I don't actually use file.read, just with open().

Also just so I can learn, what happens to this if it's only reading in a line at a time? Surely it buffers everytime a new read command is issued, but that is only reading the current line (not looking ahead)? Sorry bit confused :)

SIM-0000001 - May 24, 2020, 4:48 pm

(May 24, 2020 04:28 PM)TehTester Wrote: [\[CITE\]](#) Hmm okay, sorry again just maybe me being new to python but this is my read line:

```
Code with open(inputFile, buffering=MEG, encoding="utf-8") as in_f:
    line=in_f.readline().lower() #read the first line
```

How do I incorporate your file.read? I don't actually use file.read, just with open().

Also just so I can learn, what happens to this if it's only reading in a line at a time? Surely it buffers everytime a new read command is issued, but that is only reading the current line (not looking ahead)? Sorry bit confused :)

So put together an example to run some tests, should help you

code

Spoiler [\(Click to View\)](#)

```
Code import time
```

```
def read_chunks(chunk_size):
    start_time = time.time()
    chunks_read = 0
    with open("test.txt", encoding="utf-8", mode='r') as f:
        while True:
            chunk = f.read(chunk_size)
            if len(chunk) == 0:
                break
            chunk = chunk.lower()
            chunks_read += 1
            # now treat 'chunk' just like 'line'
```

```
    print('chunk size: ', chunk_size, 'chunks read: ', chunks_read, 'total time: ', (time.time() - start_time))
```

```
read_chunks(20)
read_chunks(1024 * 1)
read_chunks(1024 * 4)
read_chunks(1024 * 16)
read_chunks(1024 * 64)
```

```
read_chunks(1024 * 256)
read_chunks(1024 * 512)
read_chunks(1024 * 1024)
```

output

Spoiler ([Click to View](#))

```
Code chunk size: 20 chunks read: 10499950 total time: 2.6800243854522705
chunk size: 1024 chunks read: 205078 total time: 0.5359401702880859
chunk size: 4096 chunks read: 51270 total time: 0.4958314895629883
chunk size: 16384 chunks read: 12818 total time: 0.43529605865478516
chunk size: 65536 chunks read: 3205 total time: 0.47782397270202637
chunk size: 262144 chunks read: 802 total time: 0.390392541885376
chunk size: 524288 chunks read: 401 total time: 0.3886575698852539
chunk size: 1048576 chunks read: 201 total time: 0.6498758792877197
```

you can see reading 1024 characters at a time vs 20 characters at a time is 5 times faster. But there is a sweet spot, as reading 1MB at a time is actually slower than 512KB.

Test around and find the sweet spot for your system

TehTester - May 24, 2020, 5:03 pm

(May 24, 2020 04:48 PM)SIM-0000001 Wrote: [\[CITE\]](#) So put together an example to run some tests, should help you

code

Spoiler ([Click to View](#))

Code import time

```
def read_chunks(chunk_size):
    start_time = time.time()
    chunks_read = 0
    with open("test.txt", encoding="utf-8", mode='r') as f:
        while True:
            chunk = f.read(chunk_size)
            if len(chunk) == 0:
                break
            chunk = chunk.lower()
            chunks_read += 1
            # now treat 'chunk' just like 'line'

    print('chunk size: ', chunk_size, 'chunks read: ', chunks_read, 'total time: ', (time.time() -
start_time))
```

```
read_chunks(20)
read_chunks(1024 * 1)
read_chunks(1024 * 4)
read_chunks(1024 * 16)
read_chunks(1024 * 64)
read_chunks(1024 * 256)
read_chunks(1024 * 512)
read_chunks(1024 * 1024)
```

output

Spoiler ([Click to View](#))

Code chunk size: 20 chunks read: 10499950 total time: 2.6800243854522705
chunk size: 1024 chunks read: 205078 total time: 0.5359401702880859
chunk size: 4096 chunks read: 51270 total time: 0.4958314895629883
chunk size: 16384 chunks read: 12818 total time: 0.43529605865478516
chunk size: 65536 chunks read: 3205 total time: 0.47782397270202637
chunk size: 262144 chunks read: 802 total time: 0.390392541885376
chunk size: 524288 chunks read: 401 total time: 0.3886575698852539
chunk size: 1048576 chunks read: 201 total time: 0.6498758792877197

you can see reading 1024 characters at a time vs 20 characters at a time is 5 times faster. But there is a sweet spot, as reading 1MB at a time is actually slower than 512KB.

Test around and find the sweet spot for your system

Wow great thanks I'll have a look. Can I use this with `f.readline()` however? Also bit confused on the while loop?

SIM-0000001 - May 24, 2020, 5:11 pm

(May 24, 2020 05:03 PM)TehTester Wrote: [\[CITE\]](#) Wow great thanks I'll have a look. Can I use this with `f.readline()` however? Also bit confused on the while loop?

So you could use `readline()` and time that, but it'll likely be slower (unless python uses some advanced caching idk)

That while loop is sorta bad for using 'while True:' with a 'break' but to me it makes sense and its not any slower, let me re-write it

Spoiler [\(Click to View\)](#)

Code import time

```
def read_chunks(chunk_size):
    start_time = time.time()
    chunks_read = 0
    with open("test.txt", encoding="utf-8", mode='r') as f:
        bytes_read = chunk_size
        while bytes_read != 0:
            chunk = f.read(chunk_size)
            bytes_read = len(chunk)
            chunk = chunk.lower()
            chunks_read += 1
            # now treat 'chunk' just like 'line'

    print('chunk size: ', chunk_size, 'chunks read: ', chunks_read, 'total time: ', (time.time() -
start_time))

read_chunks(20)
read_chunks(1024 * 1)
read_chunks(1024 * 4)
read_chunks(1024 * 16)
read_chunks(1024 * 64)
read_chunks(1024 * 256)
read_chunks(1024 * 512)
read_chunks(1024 * 1024)
```

output - its actually a little slower this way :P

Spoiler ([Click to View](#))

```
Code chunk size: 20 chunks read: 10499951 total time: 2.7327699661254883
chunk size: 1024 chunks read: 205079 total time: 0.5368950366973877
chunk size: 4096 chunks read: 51271 total time: 0.4928779602050781
chunk size: 16384 chunks read: 12819 total time: 0.433469295501709
chunk size: 65536 chunks read: 3206 total time: 0.471912145614624
chunk size: 262144 chunks read: 803 total time: 0.40894222259521484
chunk size: 524288 chunks read: 402 total time: 0.39193272590637207
chunk size: 1048576 chunks read: 202 total time: 0.6611940860748291
```

TehTester - May 24, 2020, 5:20 pm

(May 24, 2020 05:11 PM)SIM-0000001 Wrote: [\[CITE\]](#) So you could use readline() and time that, but it'll likely be slower (unless python uses some advanced caching idk)

That while loop is sorta bad for using 'while True:' with a 'break' but to me it makes sense and its not any slower, let me re-write it

Spoiler ([Click to View](#))

Code import time

```
def read_chunks(chunk_size):
    start_time = time.time()
    chunks_read = 0
    with open("test.txt", encoding="utf-8", mode='r') as f:
        bytes_read = chunk_size
        while bytes_read != 0:
            chunk = f.read(chunk_size)
            bytes_read = len(chunk)
            chunk = chunk.lower()
            chunks_read += 1
            # now treat 'chunk' just like 'line'

    print('chunk size: ', chunk_size, 'chunks read: ', chunks_read, 'total time: ', (time.time() -
start_time))

read_chunks(20)
read_chunks(1024 * 1)
read_chunks(1024 * 4)
read_chunks(1024 * 16)
read_chunks(1024 * 64)
read_chunks(1024 * 256)
read_chunks(1024 * 512)
read_chunks(1024 * 1024)
```

output - its actually a little slower this way :P

Spoiler ([Click to View](#))

```
Code chunk size: 20 chunks read: 10499951 total time: 2.7327699661254883
chunk size: 1024 chunks read: 205079 total time: 0.5368950366973877
chunk size: 4096 chunks read: 51271 total time: 0.4928779602050781
chunk size: 16384 chunks read: 12819 total time: 0.433469295501709
chunk size: 65536 chunks read: 3206 total time: 0.471912145614624
chunk size: 262144 chunks read: 803 total time: 0.40894222259521484
chunk size: 524288 chunks read: 402 total time: 0.39193272590637207
```

chunk size: 1048576 chunks read: 202 total time: 0.6611940860748291

Sorry again newb haha - what is the while loop actually looping and what is it checking boolean wise? Did you just do an infinite loop and manually break to leave it when you want?

How would I incorporate the line = keywords in line into this (from my code)? Would it be like this?

```
Code def read_chunks(keyword,search_rgx):
    chunks_read = 0
    with open("test.txt", encoding="utf-8", mode='r') as f:
        bytes_read = chunk_size
        while bytes_read != 0:
            chunk = f.read(chunk_size)
            bytes_read = len(chunk)
            chunk = chunk.lower()
            chunks_read += 1

            out = re.match(search_rgx,chunk, re.IGNORECASE)
            if out: #if the line matches the initial regex
                if (keyWord in chunk): #if keyword2 is in the current line
                    etc etc etc
```

SIM-0000001 - May 24, 2020, 5:23 pm

(May 24, 2020 05:20 PM)TehTester Wrote: [\[CITE\]](#) Sorry again newb haha - what is the while loop actually looping and what is it checking boolean wise? Did you just do an infinite loop and manually break to leave it when you want?

How would I incorporate the line = keywords in line into this (from my code)? Would it be like this?

Sorry confused, i dont see a line

Code line = keyword

ahh nvm, looking at pastebin

oh man, is it okay if i sort of rewrite it?

TehTester - May 24, 2020, 5:28 pm

(May 24, 2020 05:23 PM)SIM-0000001 Wrote: [\[CITE\]](#) Sorry confused, i dont see a line

Code line = keyword

ahh nvm, looking at pastebin

oh man, is it okay if i sort of rewrite it?

Sure! I'm really looking at this as a learning exercise so I would appreciate if you make any changes do you mind telling me why :)???? Don't just want to take your code and not learn anything from your awesome knowledge!

EDIT: Also my logic on what keywords (AND, OR) match with each other may change a bit too.. Still need to map it out :)

SIM-0000001 - May 24, 2020, 5:34 pm

(May 24, 2020 05:28 PM)TehTester Wrote: [\[CITE\]](#) Sure! I'm really looking at this as a learning exercise so I would appreciate if you make any changes do you mind telling me why :)???? Don't just want to take your code and not learn anything from your awesome knowledge!

True, man.. idk how much effort im willing to do for this lol. Its not just a simple search and at some point you need to break on a newline :/

What is the inspiration? like what data is in the files and what are you searching for? without giving too much away

TehTester - May 24, 2020, 5:36 pm

(May 24, 2020 05:34 PM)SIM-0000001 Wrote: [\[CITE\]](#) True, man.. idk how much effort im willing to do for this lol. Its not just a simple search and at some point you need to break on a newline :/

What is the inspiration? like what data is in the files and what are you searching for? without giving too much away

Absolutely, whatever you're willing to help with I'll be grateful :)

I'll pm you :)!