grep, ack, ag的搜索效率对比

2015-03-14 linux

前言

我经常看到很多程序员, 运维在代码搜索上使用ack, 甚至ag(the_silver_searcher), 而我工作中95%都是用grep,剩下的是ag. 我觉得很有必要聊一聊这个话题.
我以前也是一个运维, 我当时也希望找到最好的最快的工具用在工作的方方面面. 但是我很好奇为什么ag和ack没有作为linux发行版的内置部分.
内置的一直是grep. 我当初的理解是受各种开源协议的限制, 或者发行版的boss个人喜好. 后来我就做了实验, 研究了下他们到底谁快. 当时的做法也无非跑几个真实地线上log看看用时. 然后我也有了我的一个认识: 大部分时候用grep也无妨, 日志很大的时候用ag.

ack原来的域名是betterthangrep.com, 现在是beyondgrep.com. 好吧. 其实我理解使用ack的同学, 也理解ack产生的原因. 这里就有个故事.

最开始我做运维使用shell, 经常做一些分析日志的工作. 那时候经常写比较复杂的shell代码实现一些特定的需求. 后来来了一位会perl的同学. 原来我写shell做一个事情, 写了20多行shell代码, 跑一次大概5分钟, 这位同学来了用perl改写, 4行, 一分钟就能跑完. 亮瞎我们的眼, 从那时候开始, 我就觉得需要学perl, 以至于后来的python.

perl是天生用来文本解析的语言, ack的效率确实很高. 我想着可能是大家认为ack要更快更合适的理由吧. 其实这件事要看场景. 我为什么还用比较'土'的grep呢? 看一下这篇文章, 希望给大家点启示

实验条件

PS: 严重声明, 本实验经个人实践, 我尽量做到合理. 大家看完觉得有异议可以试着其他的角度来做. 并和我讨论.

- 我使用了公司的一台开发机(gentoo)
- 我测试了纯英文和汉语2种， 汉语使用了结巴分词的字典, 英语使用了 `miscfiles` 中提供的词典

```
# 假如你是ubuntu: sudo apt-get install miscfiles
wget https://raw.githubusercontent.com/fxsjy/jieba/master/extra_dict/dict.txt.big
```

实验前的准备

我会分成英语和汉语2种文件, 文件大小为1MB, 10MB, 100MB, 500MB, 1GB, 5GB. 没有更多是我觉得在实际业务里面不会单个日志文件过大的. 也就没有必要测试了(就算有, 可以看下面结果的趋势)

```
cat make_words.py
# coding=utf-8

```

扫码关注「**Python之美**」

获得博客最新内容

TOC


```
获得博客最新内容StringIO import StringIO

                                    ,E = '/usr/share/dict/words'
                                    ,E = 'dict.txt.big'
                                    :N_WORD_FILE) as f:
                                    ' = f.readlines()
                                    :N_WORD_FILE) as f:
                                    ' = f.readlines()
MB = pow(1024, 2)
SIZE_LIST = [1, 10, 100, 500, 1024, 1024 * 5]
                 FORMAT = 'text_{0}_en_MB.txt'
CN_RESULT_FORMAT = 'text_{0}_cn_MB.txt'


def write_data(f, size, data, cn=False):
    total_size = 0
    while 1:
        s = StringIO()
        for x in range(10000):
            cho = random.choice(data)
            cho = cho.split()[0] if cn else cho.strip()
            s.write(cho)
        s.seek(0, os.SEEK_END)
        total_size += s.tell()
        contents = s.getvalue()
        f.write(contents + '\n')
        if total_size > size:
            break
    f.close()


for index, size in enumerate([
        MB,
        MB * 10,
        MB * 100,
        MB * 500,
        MB * 1024,
        MB * 1024 * 5]):
    size_name = SIZE_LIST[index]
    en_f = open(EN_RESULT_FORMAT.format(size_name), 'a+')
    cn_f = open(CN_RESULT_FORMAT.format(size_name), 'a+')
    write_data(en_f, size, EN_DATA)
    write_data(cn_f, size, CN_DATA, True)
```

好吧, 效率比较低是吧? 我自己没有vps, 公司服务器我不能没事把全部内核的cpu都占满(不是运维好几年了). 假如你不介意htop的多核cpu飘红, 可以这样，耗时就是各文件生成的时间短板:

```
 # coding=utf-8

import os
import random
```

获得博客最新内容


```
    E = '/usr/share/dict/words'
    E = 'dict.txt.big'
    N_WORD_FILE) as f:
     = f.readlines()
    N_WORD_FILE) as f:
     = f.readlines()
    24, 2)
SIZE_LIST = [1, 10, 100, 500, 1024, 1024 * 5]
EN_RESULT_FORMAT = 'text_{0}_en_MB.txt'
          FORMAT = 'text_{0}_cn_MB.txt'

inputs = []

def map_func(args):
    def write_data(f, size, data, cn=False):
        f = open(f, 'a+')
        total_size = 0
        while 1:
            s = StringIO()
            for x in range(10000):
                cho = random.choice(data)
                cho = cho.split()[0] if cn else cho.strip()
                s.write(cho)
            s.seek(0, os.SEEK_END)
            total_size += s.tell()
            contents = s.getvalue()
            f.write(contents + '\n')
            if total_size > size:
                break
        f.close()

    _f, size, data, cn = args
    write_data(_f, size, data, cn)


for index, size in enumerate([
        MB,
        MB * 10,
        MB * 100,
        MB * 500,
        MB * 1024,
        MB * 1024 * 5]):
    size_name = SIZE_LIST[index]
    inputs.append((EN_RESULT_FORMAT.format(size_name), size, EN_DATA, False))
    inputs.append((CN_RESULT_FORMAT.format(size_name), size, CN_DATA, True))

pool = multiprocessing.Pool()
pool.map(map_func, inputs, chunksize=1)
```

等待一段时间后,目录下是这样的:

```
1 vagrant vagrant 2.2K Mar 14 05:25 benchmarks.ipynb
1 vagrant vagrant 8.2M Mar 12 15:43 dict.txt.big
1 vagrant vagrant 1.2K Mar 12 15:46 make_words.py
1 vagrant vagrant 101M Mar 12 15:47 text_100_cn_MB.txt
1 vagrant vagrant 101M Mar 12 15:47 text_100_en_MB.txt
1 vagrant vagrant 1.1G Mar 12 15:54 text_1024_cn_MB.txt
1 vagrant vagrant 1.1G Mar 12 15:51 text_1024_en_MB.txt
-rw-rw-r-- 1 vagrant vagrant  11M Mar 12 15:47 text_10_cn_MB.txt
-rw-rw-r-- 1 vagrant vagrant  11M Mar 12 15:47 text_10_en_MB.txt
1 vagrant vagrant 1.1M Mar 12 15:47 text_1_cn_MB.txt
-rw-rw-r-- 1 vagrant vagrant 1.1M Mar 12 15:47 text_1_en_MB.txt
-rw-rw-r-- 1 vagrant vagrant 501M Mar 12 15:49 text_500_cn_MB.txt
rw-rw-r-- 1 vagrant vagrant 501M Mar 12 15:48 text_500_en_MB.txt
rw-rw-r-- 1 vagrant vagrant 5.1G Mar 12 16:16 text_5120_cn_MB.txt
-rw-rw-r-- 1 vagrant vagrant 5.1G Mar 12 16:04 text_5120_en_MB.txt
```

TOC

确认版本

```
➜  test  ack --version # ack在ubuntu下叫`ack-grep`
ack 2.12
Running under Perl 5.16.3 at /usr/bin/perl

Copyright 2005-2013 Andy Lester.

This program is free software.  You may modify or distribute it
under the terms of the Artistic License v2.0.
➜  test  ag --version
ag version 0.21.0
➜  test  grep --version
grep (GNU grep) 2.14
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Written by Mike Haertel and others, see <http://git.sv.gnu.org/cgit/grep.git/tre
```

实验设计

为了不产生并行执行的相互响应, 我还是选择了效率很差的同步执行, 我使用了ipython提供的%timeit.

上代码

```
import re
import glob
import subprocess
import cPickle as pickle
from collections import defaultdict

IMAP = {
```


```
'', '-i', '-v')
b.glob('text_*_MB.txt')
faultdict(dict)
faultdict(dict)

': EN_RES,
': CN_RES
}
REGEX = re.compile(r'text_(\d+)_(\w+)_MB.txt')
            '{command} {option} {word} {filename} > /dev/null 2>&1'

for filename in FILES:
size, xn = REGEX.search(filename).groups()
    for word in IMAP[xn]:
        _r = defaultdict(dict)
        for command in ['grep', 'ack', 'ag']:
            for option in OPTIONS:
                rs = %timeit -o -n10 subprocess.call(CALL_STR.format(command=com
                best = rs.best
                _r[command][option] = best
        RES[xn][word][size] = _r

# 存起来

data = pickle.dumps(RES)

with open('result.db', 'w') as f:
    f.write(data)
```

温馨提示, 这是一个灰常耗时的测试. 开始执行后 要喝很久的茶...

我来秦皇岛办事完毕(耗时超过1一天), 继续我们的实验.

我想要的效果

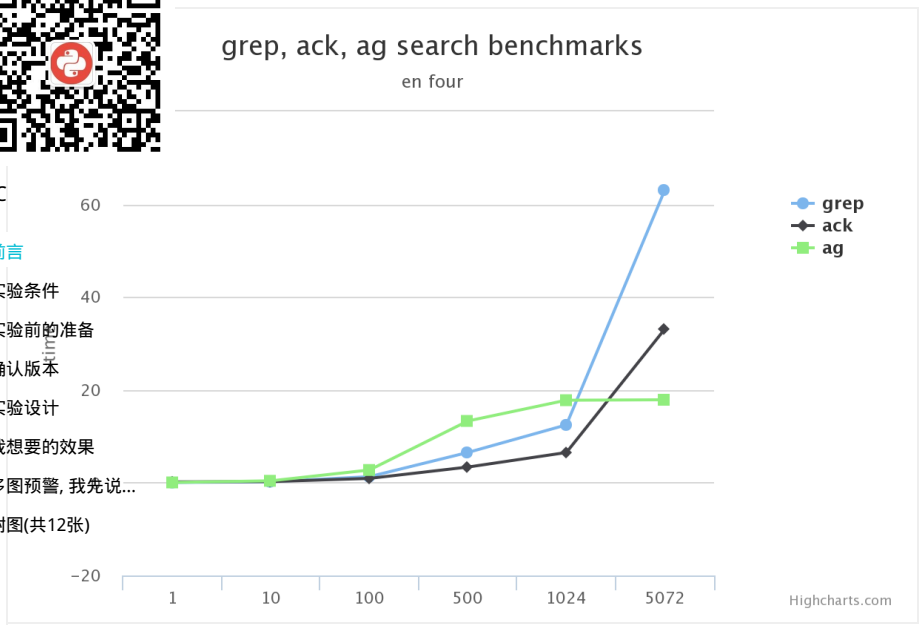
我想工作的时候一般都是用到不带参数/带-i(忽略大小写)/-v(查找不匹配项)这三种. 所以这里测试了:

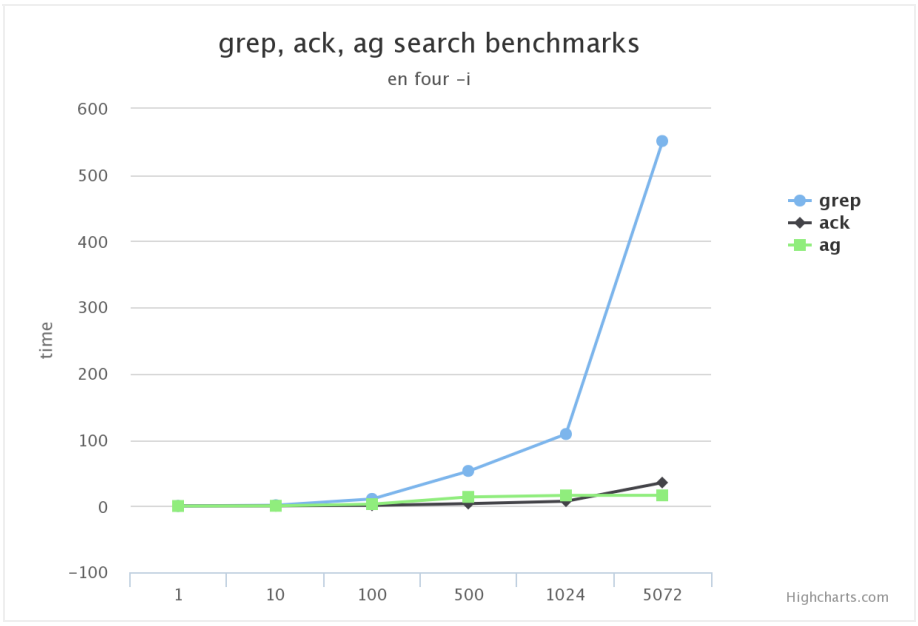
1. 英文搜索/中文搜索
2. 选择了2个搜索词(效率太低, 否则可能选择多个)
3. 分别测试''/'-i'/'-v'三种参数的执行
4. 使用%timeit, 每种条件执行10遍, 选择效率最好的一次的结果
5. 每个图代码一个搜索词, 3搜索命令, 一个选项在搜索不同大小文件时的效率对比

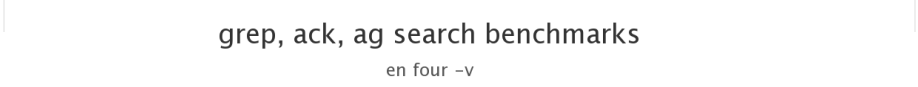
多图预警, 我先说结论

1. 在搜索的总数据量较小的情况下, 使用**grep, ack**甚至**ag**在感官上区别不大
2. 搜索的总数据量较大时, **grep**效率下滑的很多, 完全不要选
3. **ack**在某些场景下没有**grep**效果高(比如使用**-v**索索中文的时候)
4. 在不使用**ag**没有实现的选项功能的前提下, **ag**完全可以替代**ack/grep**

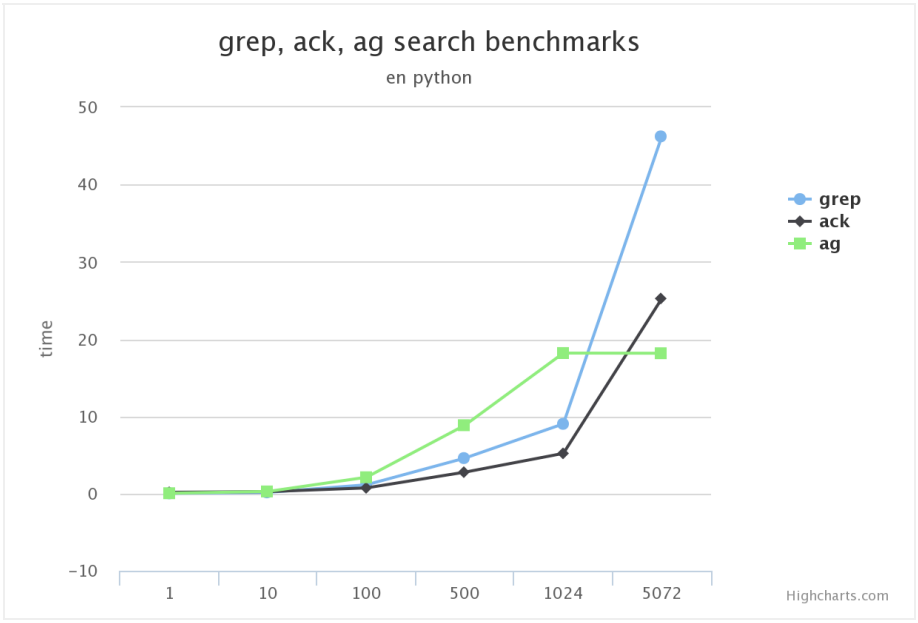
TOC

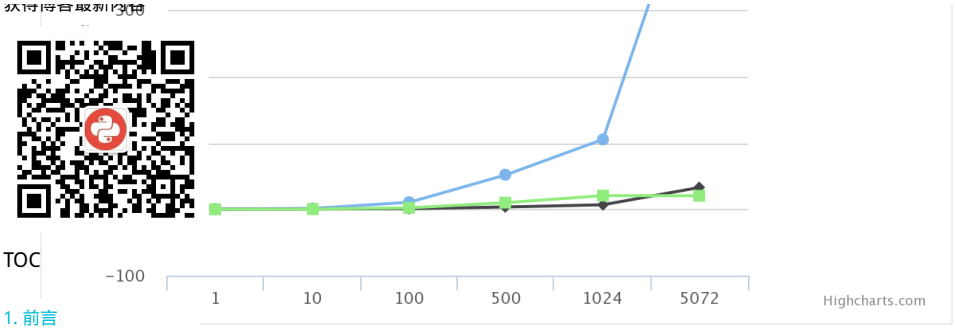




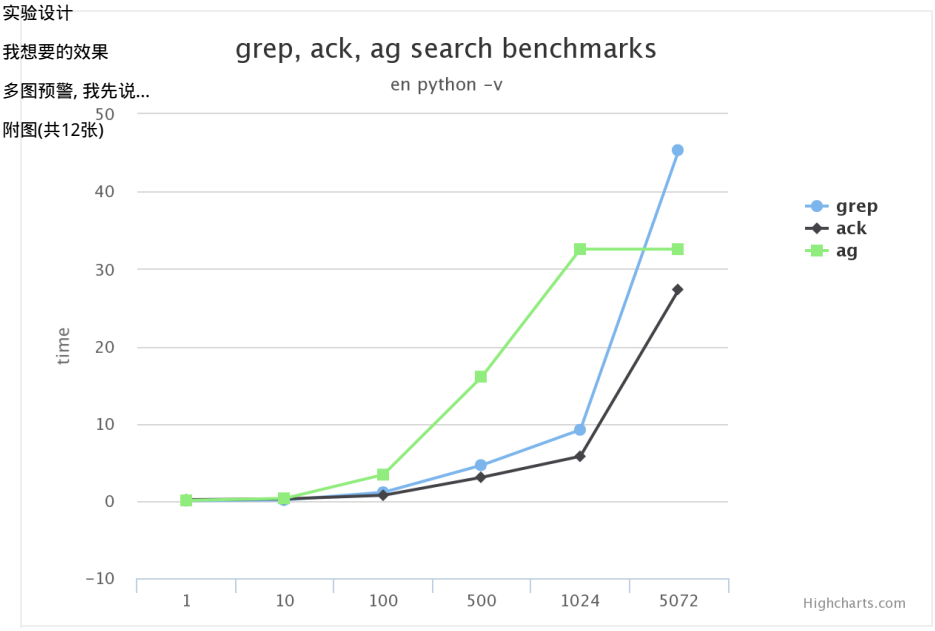


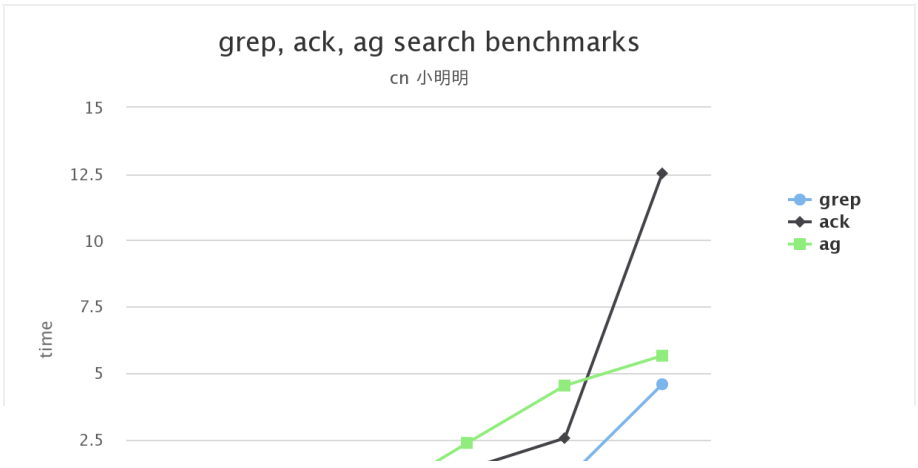
TOC

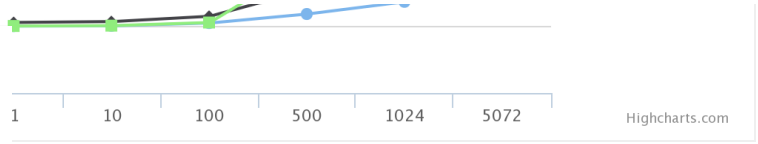




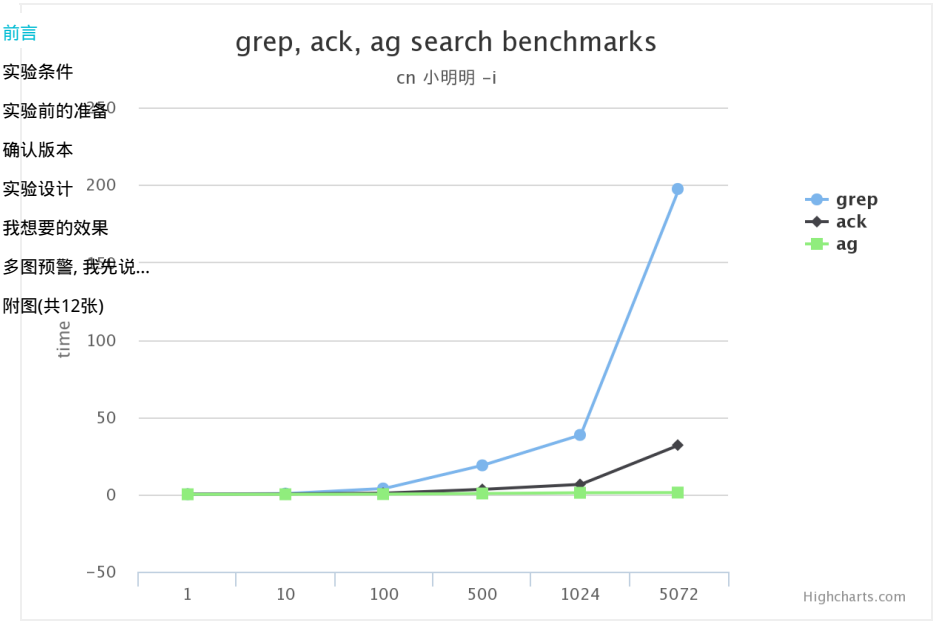
TOC

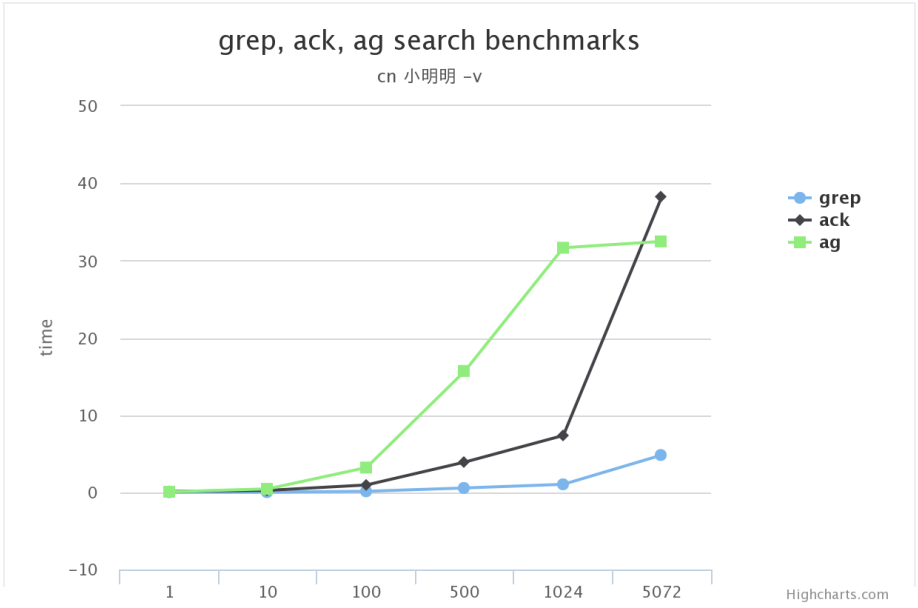


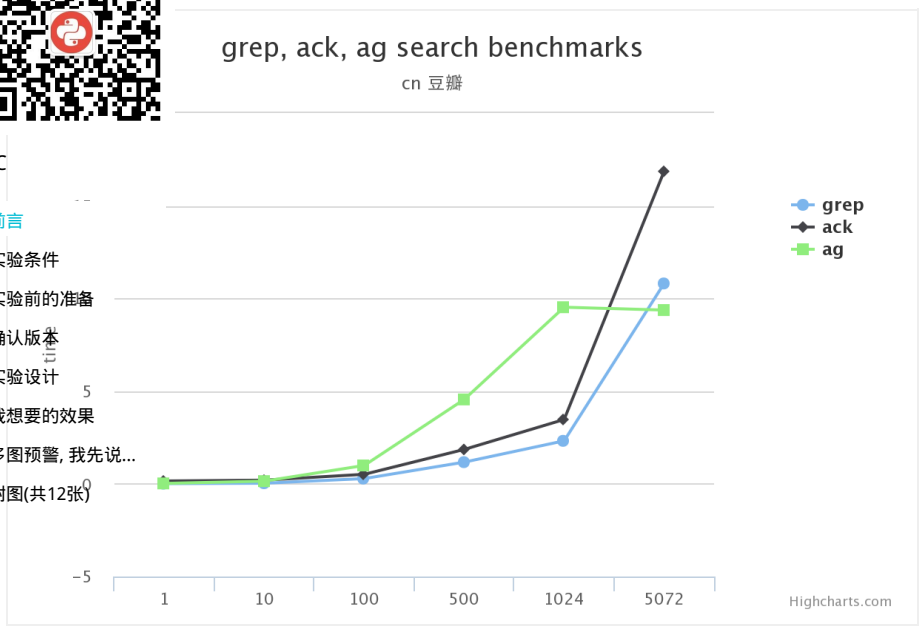




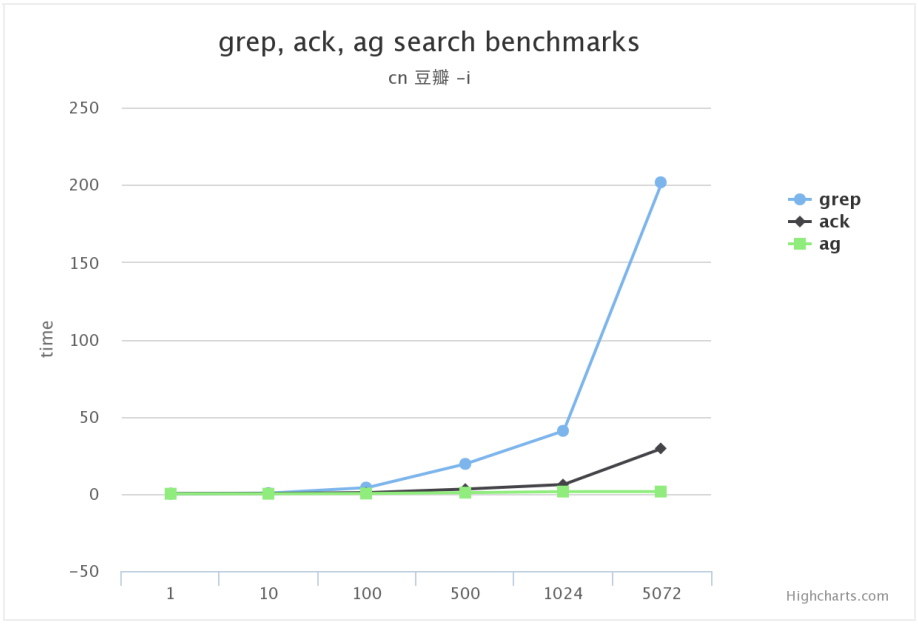
TOC

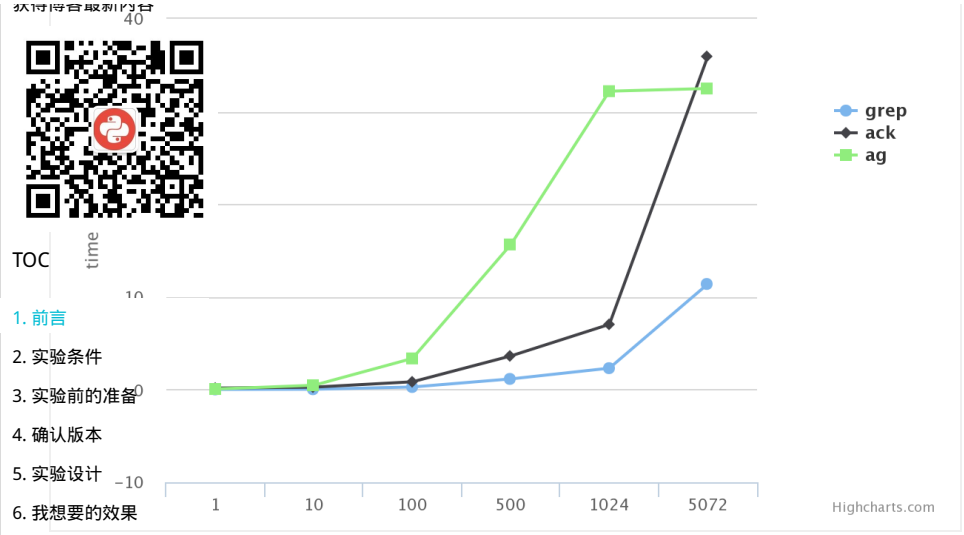






TOC





TOC

Dongweiming

ack ag grep linux

Prev
豆瓣条目组招聘

Next
IPython3时代到来


或注册一个 DISQUS 帐号
姓名
TOC
1. 前言
2. 实验条件
3. 实验前的准备
4. 确认版本
5. 实验设计
6. 我想要的效果
7. 多图预警, 我先说...
8. 附图(共12张)
依云 • 9个月前
回复 • 分享 ›
dongwm 管理员 依云 • 5个月前
咦 Rust写的，我试用一下
回复 • 分享 ›
在 DONGWM 上还有
推荐wtfPython: 一组有趣的、微妙的、复杂的Python代码片段 | 小明明s à domicile | ...
2条评论 • 2个月前
御宅暴君 — 哇，翻译辛苦了。
荐书：《Fluent Python》 | 小明明s à domicile | Python之美
16条评论 • 7个月前
Ernest He — 早读早享受
理解Python并发编程一篇就够了 - 线程篇 | 小明明s à domicile | Python之美
3条评论 • 1年前
dongwm —
选Python还是Java ？ | 小明明s à domicile | Python之美
2条评论 • 1个月前
zonghua — 还有一个选择就是Groovy，贴近Python动态语言的灵活并且和Java无缝兼容
订阅
在您的网站上使用 Disqus添加 Disqus添加
隐私