



گزارش فاز سوم پروژه مهندسی نرم افزار

بررسی کارایی با ابزارهای Xdebug، KCacheGrind، Locust، Apache JMeter

استاد: دکتر فیضی

اعضاي گروه:

اميرمحمد عزتي - 980122680012
ستايش يوسفنيا - 1400012268002
پورياب عباسى - 980122681012
ابراهيم گلريز - 990122681014

بهار 1402

فهرست مطالب

3.....	افزونه Xdebug
3.....	نصب Xdebug
4.....	گرفتن خروجی از Xdebug profiler
6.....	آزمون بار و آزمون فشار Stress Testing و Load Testing
6.....	آزمون بار
7.....	آزمون فشار
8.....	تفاوت آزمون بار و آزمون فشار
9.....	Locust
9.....	نصب Locust
11.....	اجرای Locust روی شبکه HumHub
15.....	پیدا کردن آستانه تحمل فشار شبکه(آزمون فشار) با Locust
16.....	Apache JMeter
17.....	نصب JMeter
17.....	اجرای JMeter بر روی شبکه HumHub
21.....	پیدا کردن آستانه تحمل فشار شبکه(آزمون فشار) با JMeter و مقایسه با Locust
22.....	QCacheGrind
23.....	نصب QCacheGrind
23.....	تحلیل خروجی Xdebug در QCacheGrind
28.....	سه تابعی که بیشترین زمان را صرف کرده‌اند

افزونه Xdebug

یک افزونه برای PHP است و طیف وسیعی از ویژگی‌ها را برای بهبود تجربه توسعه PHP ارائه می‌دهد.

کاربردهای مختلف افزونه Xdebug شامل موارد زیر است:

Step Debugging -

راهی برای گذر از کد در IDE یا ویرایشگر در حین اجرای اسکریپت.

بهبود در گزارش خطای PHP -

یک تابع var_dump() بهبود یافته، ردیابی‌ها را برای Exceptions، Notices، Warnings، Errors در یک پشته اضافه می‌کند تا مسیر کد به خطا را برجسته کند.

Tracing -

هر فراخوانی تابع را با آرگومان‌ها و مکان فراخوانی روی دیسک می‌نویسد. به صورت اختیاری همچنین شامل هر انتساب متغیر و مقدار بازگشتنی برای هر تابع است.

Profiling -

به شما امکان می‌دهد با کمک ابزارهای visualization، عملکرد برنامه PHP خود را تجزیه و تحلیل کنید و گلوگاه‌ها(bottlenecks) را پیدا کنید.

Code Coverage Analysis -

برای نشان دادن اینکه کدام بخش از کد شما هنگام اجرای تست‌های واحد با PHPUnit، اجرا می‌شوند.

نصب Xdebug

(این توضیحات برای نصب در ویندوز است.)

1. ابتدا با توجه به نسخه phpمان از این [لينك](#) فایل dll مربوط به Xdebug را دانلود می‌کنیم.
2. سپس فایل dll را در مسیر C:\xampp\php\ext قرار می‌دهیم.
3. فایل php.ini موجود در C:\xampp\php را باز می‌کنیم و خط زیر را به آن اضافه می‌کنیم:
zend_extension = "C:\xampp\php\ext\php_xdebug-3.2.1-8.2-vs16-x86_64.dll"

* آدرس فایل dll در خط بالا نوشته شده است.

```

; The MIBS data available in the PHP distribution must be installed.
; See https://www.php.net/manual/en/snmp.installation.php
;extension=snmp

;extension=soap
;extension=sockets
;extension=sodium
;extension=sqlite3
;extension=tidy
extension=xsl
extension=zip

zend_extension=opcache
zend_extension = "C:\xampp\php\ext\php_xdebug-3.2.1-8.2-vs16-x86_64.dll"

;;;;;;
; Module Settings ;
;;;;;;
asp_tags=Off
display_startup_errors=On
;zend_start_up_error_level=Warning
;zend_start_up_error_level=Error

```

Ln 955, Col 73 100% Windows (CRLF) UTF-8

- .4 سرور apache برنامه xampp خود را restart می‌کنیم.
 .5 با اجرای دستور -v php shell در php shell می‌توانیم مانند شکل زیر از اجرای Xdebug مطمئن شویم.

```

Setting environment for using XAMPP for Windows.
amezz@UNREACHABLE c:\xampp
# php -v
PHP 8.2.0 (cli) (built: Dec  6 2022 15:31:23) (ZTS Visual C++ 2019 x64)
Copyright (c) The PHP Group
Zend Engine v4.2.0, Copyright (c) Zend Technologies
    with Xdebug v3.2.1, Copyright (c) 2002-2023, by Derick Rethans

amezz@UNREACHABLE c:\xampp
# |

```

گرفتن خروجی از profiler

در Xdebug Profiler به ما کمک می‌کند تا php خود پیدا کنیم و در نهایت آنها را با ابزار خارجی مانند WinCacheGrind یا KCacheGrind مصوّرسازی کنیم.

برای فعال کردن profiler ابتدا باید خطوط highlight شده در عکس پایین را به فایل php.ini اضافه کنیم تا برای request هر چیزی در دایرکتوری C:\xampp\tmp خروجی profile تولید شود.

```

;extension=snmp

;extension=soap
;extension=sockets
;extension=sodium
;extension=sqlite3
;extension=tidy
extension=xsl
extension=zip

zend_extension=opcache
zend_extension = "C:\xampp\php\ext\php_xdebug-3.2.1-8.2-vs16-x86_64.dll"

xdebug.mode = profile
xdebug.profiler_enable_trigger = 1
xdebug.output_dir = "C:\xampp\tmp"
xdebug.profiler_output_name = "cachegrind.out.%u.%H_%R"

;;;;;;
; Zend OPCache settings :

```

Ln 960, Col 56 100% Windows (CRLF) UTF-8

برای اینکه خروجی‌ها بصورت فایل compressed شده تولید نشود، در یک خط کد مانند عکس پایین این ویزگی را غیرفعال می‌کنیم تا بتوانیم از خروجی در ابزار QCacheGrind استفاده کنیم.



```
File Edit View
;extension=snmp

;extension=soap
;extension=sockets
;extension=sodium
;extension=sqlite3
;extension=tidy
extension=xsl
extension=zip

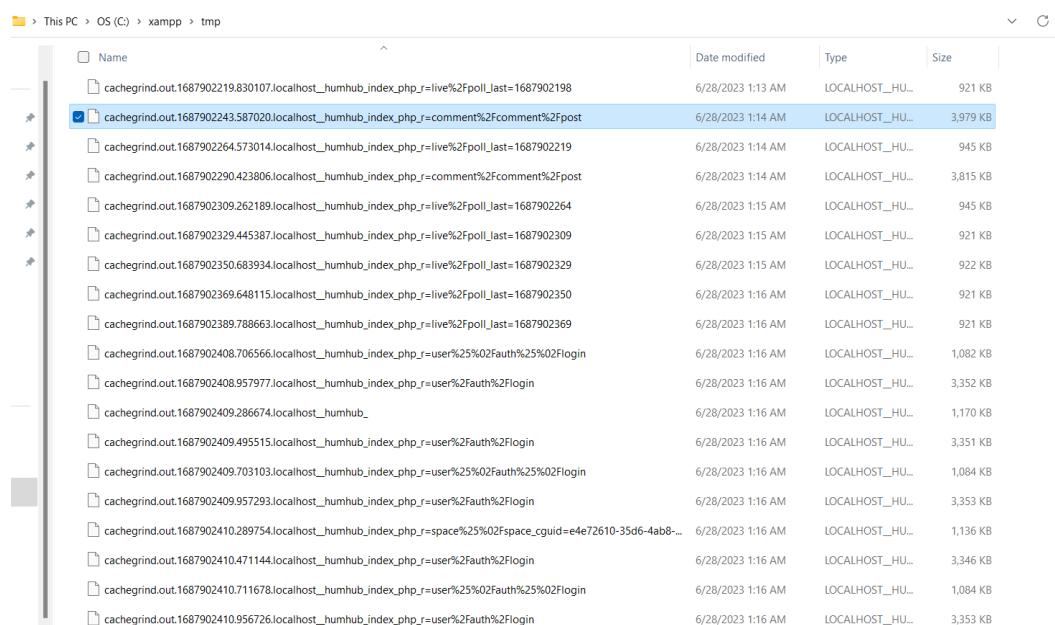
zend_extension=opcache
zend_extension = "c:\xampp\php\ext\php_xdebug-3.2.1-8.2-vs16-x86_64.dll"

xdebug.mode = profile
xdebug.profiler_enable_trigger = 1
xdebug.output_dir = "c:\xampp\tmp"
xdebug.profiler_output_name = "cachegrind.out.%u.%H_%R"
xdebug.use_compression = 0

Ln 961, Col 27
```

100% Windows (CRLF) UTF-8

حال با انجام شدن هر request یک خروجی توسط Xdebug در دایرکتوری C:\xampp\tmp مانند زیر ایجاد می‌شود.



سپس به کمک ابزار QCacheGrind می‌توانیم bottleneck‌های اجرای پروژه PHP‌مان را تحلیل کنیم.

آزمون بار Stress Testing و آزمون فشار Load Testing

آزمون بار

آزمون بار نوعی از آزمون عملکرد سیستم نرم افزاری است که عملکرد یک سیستم یا محصول نرم افزاری را در شرایط بار (load) معمولی و واقعی نشان داده و می‌سنجد. در واقع این آزمون مشخص می‌کند که زمان استفاده همزمان چندین کاربر، نرم افزار چه رفتاری از خود نشان می‌دهد. هدف این آزمون تشخیص bottleneck (گلوبگاه داده) بوده و نحوه رفتار سیستم در هنگام بارهای معمولی را بررسی می‌کند و تعیین می‌کند که آیا یک سیستم، نرم افزار یا دستگاه محاسباتی می‌تواند بارهای بالا را با توجه به تقاضای بالای کاربران نهایی تحمل کند یا خیر این کار کمک می‌کند اطمینان حاصل کنیم که سیستم ما می‌تواند میزان استفاده مورد انتظارمان را هندل کرده و هر مشکل بالقوه ای را قبل از استقرار نهایی نرم افزار در محیط عملیاتی را مشخص کنیم.

حين آزمون بار سناریوهای متفاوتی شبیه سازی می‌شوند که این سناریوها می‌توانند شامل شبیه سازی تعداد بالای کاربر همزمان، ایجاد نرخ بالای درخواست، و شبیه سازی ترافیک سنگین شبکه باشد. عملکرد سیستم تحت این شبیه سازی‌ها اندازه‌گیری و آنالیز می‌شود.

مزایای آزمون بار

از مزایای این آزمون می‌توان به کشف گلوبگاه‌ها قبل از تولید، scalability (مقیاس پذیری)، کاهش زمان از کار افتادن سیستم (system downtime)، بهبود رضایتمندی مشتری (کاربران) و کاهش نرخ شکست اشاره کرد.

• افزایش مقیاس پذیری یک سیستم

این آزمون می‌تواند به شناسایی محدودیت ظرفیت عملیاتی یک برنامه کمک کند. این می‌تواند در تعیین نیازهای زیرساختی هنگامی که مقیاس سیستم رو به افزایش است مورد استفاده قرار بگیرد.

• کاهش خطر از کار افتادن

آزمون بار برای تعیین سناریوهایی که ممکن است باعث از کار افتادن سیستم شود مورد استفاده قرار می‌گیرد که آن را به یک ابزار عالی برای پیدا کردن راه حل مشکلات ناشی از ترافیک بالا، قبل از وقوع آن در واقعیت، تبدیل می‌کند.

• بهبود رضایتمندی مشتری‌ها

اگر زمان پاسخ دهی response time یک سیستم همزمان با افزایش مقیاس خود ثابت و کوتاه بماند مشتری‌هایی که از این سیستم برای یک بار هم استفاده کرده باشند با احتمال بالاتری دوباره برای استفاده به این سیستم رجوع می‌کنند.

آزمون فشار

تست فشار نوعی تست نرم افزاری است که پایداری و قابلیت اطمینان نرم افزار را تایید می کند. هدف تست فشار اندازهگیری استحکام نرم افزار و قابلیتهای مدیریت خطا در شرایط بار بسیار سنگین و اطمینان از عدم خرابی نرم افزار در شرایط بحرانی است. این آزمون حتی فراتر از نقاط عملیاتی معمولی آزمایش می کند و نحوه عملکرد نرم افزار را در شرایط سخت ارزیابی می کند.

در مهندسی نرم افزار، آزمون فشار به عنوان آزمون استقامت نیز شناخته می شود. تحت این تست، برنامه تحت آزمایش برای مدت کوتاهی تحت فشار قرار می گیرد تا ظرفیت تحمل آن مشخص شود. مهمترین کاربرد تست استرس تعیین حدی است که در آن سیستم یا نرم افزار یا سخت افزار از کار می افتد. همچنین بررسی می کند که آیا سیستم مدیریت خطای مؤثری در شرایط سخت از خود نشان می دهد یا خیر.

انواع آزمون فشار:

• آزمون فشار توزیع شده Distributed Stress Testing

• آزمون فشار نرم افزار Application Stress Testing

این آزمایش بر روی یافتن نقص های مربوط به قفل و مسدود کردن داده ها، مشکلات شبکه و گلوگاه های عملکرد در یک برنامه مرکز است.

• آزمون فشار تراکنش/تعاملات Transactional Stress System

این تست فشار را روی یک یا چند تراکنش بین دو یا چند برنامه انجام می دهد. برای تنظیم دقیق و بهینه سازی سیستم استفاده می شود.

• Systemic Stress Testing

این یک تست استرس یکپارچه است که می تواند در چندین سیستم در حال اجرا بر روی یک سرور آزمایش شود و برای یافتن نقص در موقعیت که داده های یک برنامه، برنامه دیگر را مسدود می کند.

• Exploratory Stress Testing

این یکی از انواع تست فشار است که برای آزمایش سیستم با پارامترها یا شرایط غیرعادی که بعید است در یک سناریوی واقعی رخ دهد، استفاده می شود. برای یافتن نقص در سناریوهای غیرمنتظره مانند:

- تعداد زیادی از کاربران به طور همزمان وارد سیستم شدند
- اگر یک virus scanner در همه ماشین ها به طور همزمان شروع شود.
- اگر دیتابیس هنگام دسترسی از یک وب سایت آفلاین شده باشد.
- هنگامی که حجم زیادی از داده ها به طور همزمان به دیتابیس وارد می شود.

تفاوت آزمون بار و آزمون فشار

تست فشار به شما کمک می کند تا تعیین کنید که یک سیستم تحت یک بار شدید، مانند حمله DDoS، اثر Slashdot یا سناریوهای دیگر چگونه رفتار می کند. هدف بیشتر از شناسایی تنگناها تعیین حد حداکثری است. به این ترتیب می توان برای شرایط غیرمنتظره آماده شد.

از سوی دیگر، آزمایش‌های بار برای اطمینان از برآورده شدن انتظارات کاربر، مانند وعده‌های توافق سطح خدمات (SLA) طراحی شده‌اند. هدف، اطمینان از یک تجربه کاربری کلی قابل قبول به جای تلاش برای از کار انداختن برنامه است. این به ما امکان می دهد با اطمینان که جدید را مستقر کنیم.

Locust

Locust یک ابزار تست عملکرد آسان، قابل استفاده و مقیاس‌پذیر است. با Locust می‌توان آزمون تست بار (Load Testing) را انجام داد.

شما رفتار کاربران خود را به جای درگیری با یک رابط کاربری یا زبان خاص با دامنه محدود، در کد معمولی پایتون، تعریف می‌کنید.

این باعث می‌شود Locust بی‌نهایت قابل گسترش و برای توسعه‌دهندگان بسیار مناسب باشد.

:Locust ویژگی‌های

- سناریوهای آزمایشی را در پایتون قدیمی بنویسید.
- توزیع شده و مقیاس‌پذیر - از صدها هزار کاربر همزمان پشتیبانی می‌کند.
- UI مبتنی بر وب
- می‌تواند هر سیستمی را تست کند.
- قابل هک شدن

نصب Locust

(این مراحل روی ویندوز انجام شده است.)
برای دیدن مراحل نصب Locust می‌توان به آن در [سایت Locust documentation](#) مراجعه کرد.

Installation

0. [Install Python \(3.7 or later\)](#)
1. Install the package (check [the wiki](#) if the installation fails)

```
$ pip3 install locust
```

2. Validate your installation

```
$ locust -V
locust 2.15.1 from /usr/local/lib/python3.10/site-packages/locust (python 3.10.6)
```

3. Done! Now you can [create your first test](#)

1. ابتدا باید python را روی سیستم خود داشته باشد. اگر آن را ندارید می‌توانید از این [لینک](#) آن را دانلود کرده و نصب کنید.
2. سپس با اجرای دستور زیر در محیط CMD Locust را نصب کنید.
`pip install locust`

```

Command Prompt
-----[REDACTED]-----
Downloading certifi-2023.5.7-py3-none-any.whl (156 kB)
Collecting brotli
  Downloading Brotli-1.0.9-cp311-cp311-win_amd64.whl (333 kB)
Collecting charset-normalizer<4,>=2
  Downloading charset_normalizer-3.1.0-cp311-cp311-win_amd64.whl (96 kB)
Collecting idna<4,>=2.5
  Downloading idna-3.4-py3-none-any.whl (61 kB)
Collecting urllib3<3,>=1.21.1
  Downloading urllib3-2.0.3-py3-none-any.whl (123 kB)
Collecting MarkupSafe<2.1.1
  Downloading MarkupSafe-2.1.3-cp311-cp311-win_amd64.whl (17 kB)
Collecting pycparser
  Downloading pycparser-2.21-py2.py3-none-any.whl (118 kB)
Collecting colorama
  Downloading colorama-0.4.6-py2.py3-none-any.whl (25 kB)
Installing collected packages: roundrobin, pywin32, msgpack, brotli, zope.interface, zope.event, urllib3, typing-extensions, Six, pyzmq, pycparser, psutil, MarkupSafe, itsdangerous, idna, greenlet, ConfigArgParse, colorama, charset-normalizer, certifi, blinker, Werkzeug, requests, Jinja2, click, cffi, gevent, flask, geventhttplibclient, Flask-Cors, Flask-BasicAuth, locust
  DEPRECATION: roundrobin is being installed using the legacy 'setup.py install' method, because it does not have a 'pyproject.toml' and the 'wheel' package is not installed. pip 23.1 will enforce this behaviour change. A possible replacement is to enable the '--use-pep517' option. Discussion can be found at https://github.com/pypa/pip/issues/8559
    Running setup.py install for roundrobin ... done
  Successfully installed ConfigArgParse-2.1.3 Flask-BasicAuth-0.2.0 Flask-Cors-3.0.10 Jinja2-3.1.2 MarkupSafe-2.1.3 Six-1.16.0 Werkzeug-2.3.6 blinker-1.6.2 brotli-1.0.9 certifi-2023.5.7 cffi-1.15.1 charset-normalizer-3.1.0 click-8.1.3 colorama-0.4.6 flask-2.8.2 gevent-22.10.2 geventhttplibclient-2.0.9 greenlet-2.0.2 idna-3.4 itsdangerous-2.1.2 locust-2.15.1 msgpack-0.5.9 psutil-5.9.5 pycparser-2.21 pywin32-306 pyzmq-25.1.0 requests-2.31.0 roundrobin-0.0.4 typing-extensions-4.6.3 urllib3-2.0.3 zope.event-5.0 zope.interface-6.0
  DEPRECATION: Flask-BasicAuth is being installed using the legacy 'setup.py install' method, because it does not have a 'pyproject.toml' and the 'wheel' package is not installed. pip 23.1 will enforce this behaviour change. A possible replacement is to enable the '--use-pep517' option. Discussion can be found at https://github.com/pypa/pip/issues/8559
    Running setup.py install for Flask-BasicAuth ... done

```

3. سپس برای چک کردن اینکه Locust درست نشده شده یا خیر می‌توانید دستور زیر را در CMD اجرا کنید.

`locust --v`

```

Command Prompt
-----[REDACTED]-----
C:\Users\amezz>locust --v
locust 2.15.1 from C:\Users\amezz\AppData\Local\Programs\Python\Python311\Lib\site-packages\locust (python 3.11.3)
C:\Users\amezz>

```

همان‌طور که در عکس بالا مشخص است، locust 2.15.1 با موفقیت روی سیستم نصب شده است.

اجرای Locust روی شبکه HumHub

ابتدا یک فایل به نام locustfile.py ایجاد می‌کنیم تا سناریوهای مختلف کاربران را برای آزمون بار(Load Testing) در آن ایجاد کنیم.

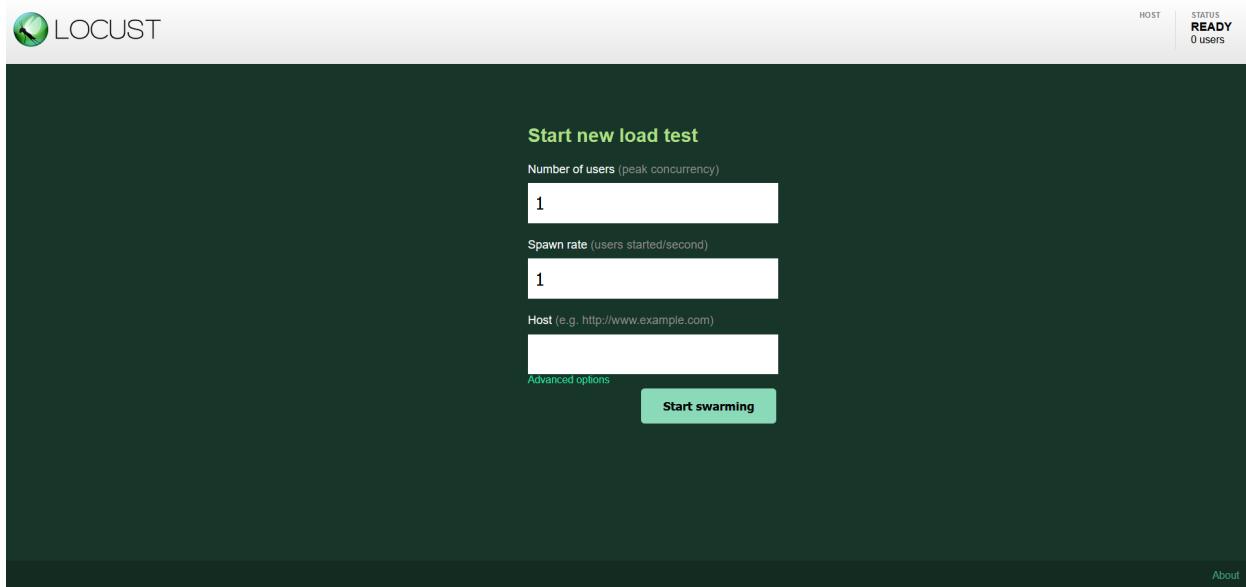
ابتدا یک کلاس برای کاربران تعریف می‌کنیم که از HttpUser ارث می‌برد و به هر کاربر یک ویژگی Client می‌دهد، که نمونه‌ای از HttpSession است که می‌تواند برای ارسال درخواست‌های HTTP به سیستم هدف که می‌خواهیم آزمون بار انجام دهیم، استفاده شود. هنگامی که یک تست شروع می‌شود، locust برای هر کاربری که شبیه‌سازی می‌کند نمونه‌ای از این کلاس ایجاد می‌کند و هر یک از این کاربران شروع به اجرا می‌کنند.

پنج function مختلف برای پنج سناریو مختلف تعریف می‌کنیم. به ترتیب برای رفتن به داشبورد، پست جدید، لايك کردن یک پست، فالو کردن، کامنت گذاشتن و یک function هم برای login کردن user هست که در ابتدای شبیه‌سازی هر user انجام می‌شود.

```
1 import time
2 from locust import HttpUser, task, between
3
4 class FlowException(Exception):
5     pass
6
7 class QuickstartUser(HttpUser):
8     wait_time = between(1, 3)
9
10    @task(1)
11    def dashboard(self):
12        self.client.get("/")
13
14    @task(2)
15    def new_post(self):
16        self.client.get("/index.php?r=space%2Fspace&cguid=e4e72610-35d6-4ab8-8be0-18489a83ee0f")
17        post_response = self.client.post('/index.php?r=space%2Fspace&cguid=e4e72610-35d6-4ab8-8be0-18489a83ee0f', json={'id': 1, 'containerClass': 'new post'})
18        if post_response.status_code != 200:
19            raise FlowException('Post is not created.')
20
21    @task(2)
22    def like_post(self):
23        post_response = self.client.post('/index.php?r=like%2Flike&contentModel=humhub%5Cmodules%5Cpost%5Cmodels%5CPost&contentId=3')
24        if post_response.status_code != 200:
25            raise FlowException('Post is not liked.')
26
27    @task(1)
28    def follow_user(self):
29        self.client.get("/index.php?r=user%2Fpeople") # go to people tab
30        self.client.get("/index.php?r=user%2Fprofile&cguid=8f3d6c9e-d163-4f69-b804-aea5c6ffd09a") # go to David Roberts Profile
31        post_response = self.client.post('/index.php?r=user%2Fprofile%2Ffollow&cguid=8f3d6c9e-d163-4f69-b804-aea5c6ffd09a') # follow David
32        if post_response.status_code != 200:
33            raise FlowException('The person is not followed.')
34
35    @task(3)
36    def leave_comment(self):
37        self.client.get("/index.php?r=space%2Fspace&cguid=e4e72610-35d6-4ab8-8be0-18489a83ee0f") # go to welcome space
38        post_response = self.client.post('/index.php?r=comment%2Fcomment%2Fpost', json={'id': 1, 'comment': 'new comment'})
39        if post_response.status_code != 200:
40            raise FlowException('The comment is not sent.')
41
42    @task
43    def on_start(self):
44        self.client.post("/index.php?r=user%2Fauth%2Flogin", json={"username": "amirezziati", "password": "12345"})
```

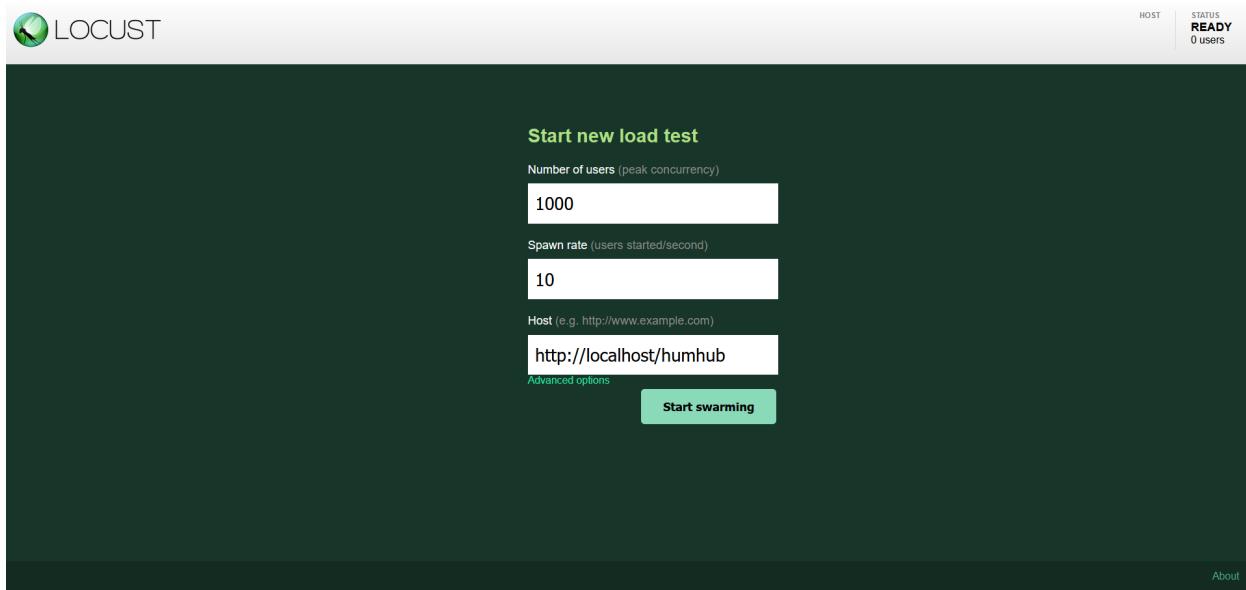
سپس بعد از نوشتن function ها برای اجرای Locust وارد دایرکتوری فایل locustfile.py شده و دستور

را اجرا می کنیم. حال با باز کردن <http://localhost:8089> می توانیم وارد بخش UI-based Locust ابزار شویم.



در اینجا باید تعداد کاربران و Spawn rate و همچنین Host را برای شبیه سازی مان مشخص کنیم و swarming را بزنیم.

در نهایت برای اجرا با 1000 کاربر با spawn rate=10، مانند عکس زیر ورودی ها را می دهیم و دقایقی منتظر می مانیم تا اجرا تکمیل شود.

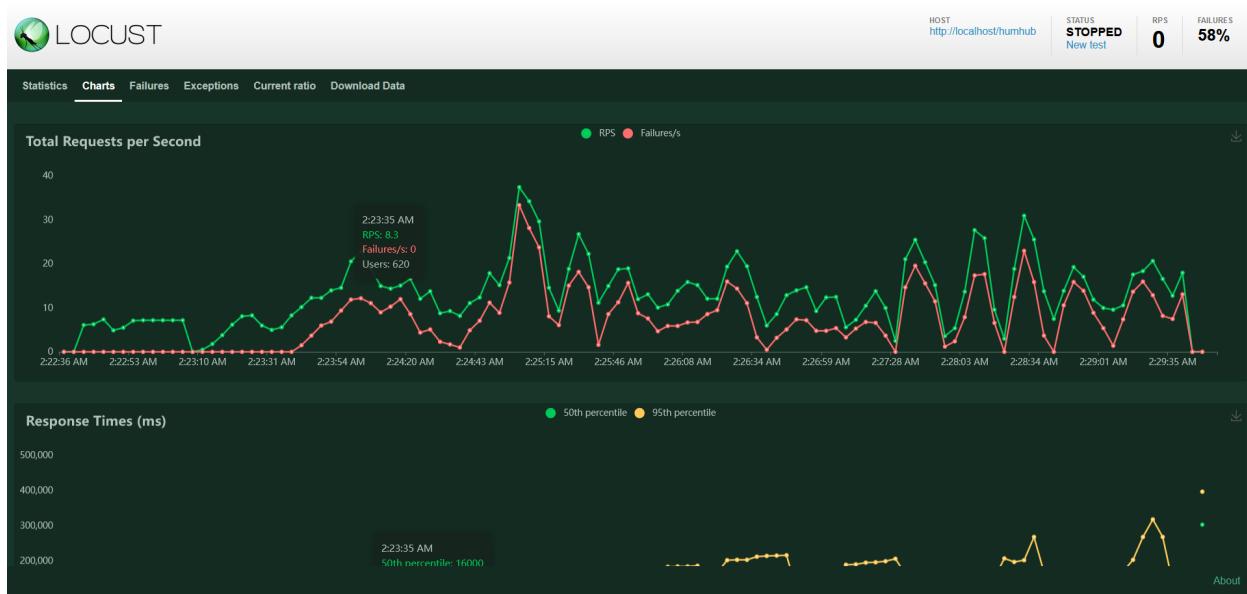


5 دقیقه بعد از اینکه تعداد userها به 1000 رسید صبر کردیم و سپس میتوانیم خروجی‌ها را در قالب چارت و جدول بشكل زیر ببینیم:

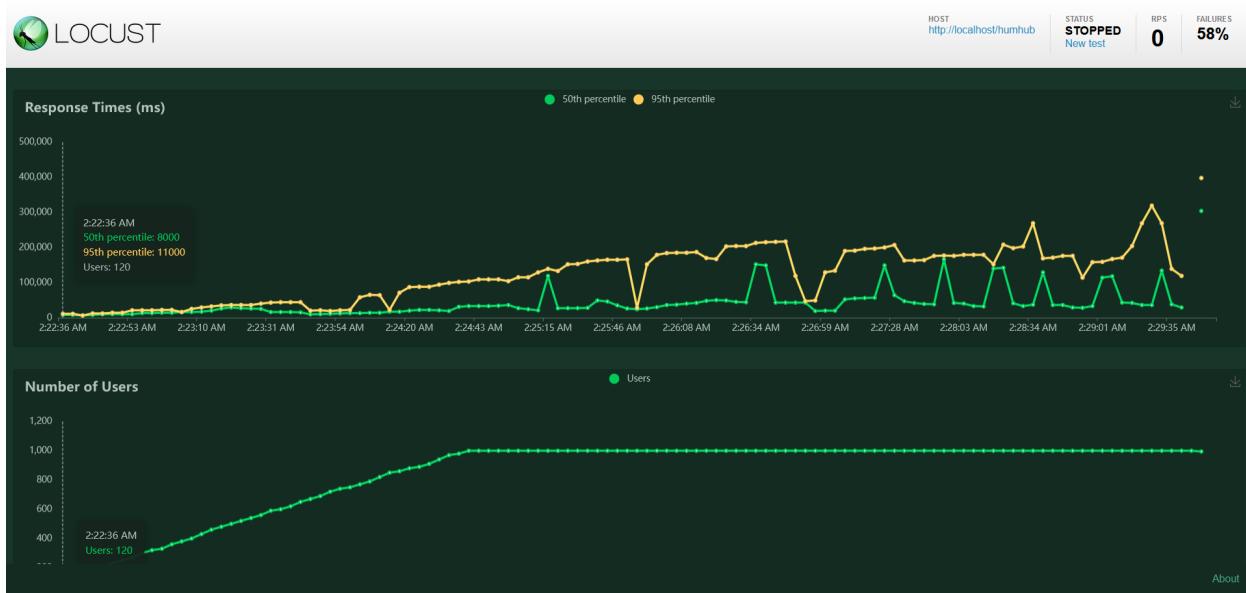
کل Http Request‌ها، همچنین تعداد request‌های fail شده و response time در جدول زیر قابل مشاهده است.

LOCUST												
Statistics Charts Failures Exceptions Current ratio Download Data												
Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/humhub/	355	213	35000	149000	307000	52645	280	398248	12966	0	0
POST	/humhub/index.php?r=comment%2Fcomment%2Fpost	811	469	31000	150000	311000	53736	308	398286	13669	0	0
POST	/humhub/index.php?r=like%25%02Flike%25%02Flike&contentModel=humhub%25Cmodules%255Cpost%255Cmodels%255CPost&contentId=3	606	370	36000	152000	270000	55219	1385	398273	12624	0	0
GET	/humhub/index.php?r=space%25%02Fspace&cguid=e4e72610-35d6-4ab8-8be0-18489ab3ee0f	1623	998	34000	145000	270000	51135	23	398603	12483	0	0
POST	/humhub/index.php?r=space%25%02Fspace&cguid=e4e72610-35d6-4ab8-8be0-18489ab3ee0f	569	310	30000	146000	311000	52818	596	398510	14755	0	0
POST	/humhub/index.php?r=user%25%02Fauth%25%02Flogin	1304	684	33000	170000	269000	65661	541	398276	15412	0	0
GET	/humhub/index.php?r=user%25%02Fpeople	344	216	37000	157000	318000	57876	222	387745	12061	0	0
POST	/humhub/index.php?r=user%25%02Fprofile%25%02Ffollow&cguid=8f3d6c9e-d163-4f69-b604-aea5c0fffd09a	243	120	29000	158000	308000	59805	2471	397244	16407	0	0
GET	/humhub/index.php?r=user%25%02Fprofile&cguid=8f3d6c9e-d163-4f69-b604-aea5c0fffd09a	292	176	32000	141000	311000	52014	4419	346839	12877	0	0
Aggregated		6147	3556	33000	154000	307000	55967	23	398603	13663	0	0

چارت مربوط به تعداد کل request‌ها و میزان fail‌شدن‌شان با افزایش تعداد userها در عکس پایین قابل مشاهده است.



به ترتیب چارت مربوط به response time و تعداد userها با گذر زمان در عکس پایین قابل مشاهده می‌باشد.



در نهایت جدول زیر نشان می‌دهد که هر کدام از request‌ها چند بار fail شده‌اند و نوع یا دلیل fail شدن‌شان را بیان می‌کند.

The table lists failed requests with their details:

# fails	Method	Name	Type
1	GET	/humhub/	ConnectionResetError(10054, 'An existing connection was forcibly closed by the remote host', None, 10054, None)
209	GET	/humhub/	ConnectionRefusedError(10061, '[WinError 10061] No connection could be made because the target machine actively refused it.')
3	GET	/humhub/	RemoteDisconnected('Remote end closed connection without response')
17	POST	/humhub/index.php?r=comment%2Fcomment%2Fpost	ConnectionResetError(10054, 'An existing connection was forcibly closed by the remote host', None, 10054, None)
439	POST	/humhub/index.php?r=comment%2Fcomment%2Fpost	ConnectionRefusedError(10061, '[WinError 10061] No connection could be made because the target machine actively refused it.')
13	POST	/humhub/index.php?r=comment%2Fcomment%2Fpost	RemoteDisconnected('Remote end closed connection without response')
5	POST	/humhub/index.php?r=like%25%02Flike%25%02Flike&contentModel=humhub%255Cmodules%255Cpost%255Cmodels%255CPost&contentId=3	ConnectionResetError(10054, 'An existing connection was forcibly closed by the remote host', None, 10054, None)
1	POST	/humhub/index.php?r=like%25%02Flike%25%02Flike&contentModel=humhub%255Cmodules%255Cpost%255Cmodels%255CPost&contentId=3	ChunkedEncodingError(ProtocolError("Connection broken: ConnectionResetError(10054, 'An existing connection was forcibly closed by the remote host', None, 10054, None)", ConnectionResetError(10054, 'An existing connection was forcibly closed by the remote host', None, 10054, None)))
361	POST	/humhub/index.php?r=like%25%02Flike%25%02Flike&contentModel=humhub%255Cmodules%255Cpost%255Cmodels%255CPost&contentId=3	ConnectionRefusedError(10061, '[WinError 10061] No connection could be made because the target machine actively refused it.')
3	POST	/humhub/index.php?r=like%25%02Flike%25%02Flike&contentModel=humhub%255Cmodules%255Cpost%255Cmodels%255CPost&contentId=3	RemoteDisconnected('Remote end closed connection without response')
13	GET	/humhub/index.php?r=space%25%02Fspace&cguid=e4e72610-35d6-4ab8-8be0-18489a83ee0f	ConnectionResetError(10054, 'An existing connection was forcibly closed by the remote host', None, 10054, None)

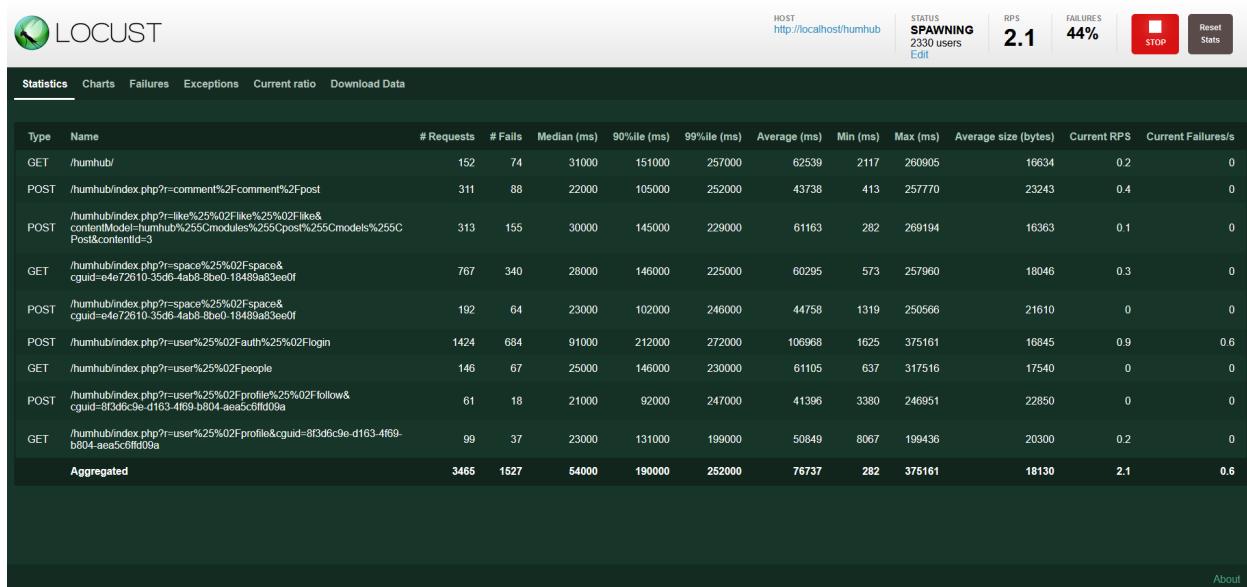
فایل locustfile.py و ریپورت‌های csv مربوط به این اجرای repository گیت‌هاب قرار گرفته است.

پیدا کردن آستانه تحمل فشار شبکه(آزمون فشار) با Locust

برای پیدا کردن آستانه تحمل فشار شبکه، یک تست با 5000 کاربر و spawn rate=10 انجام دادیم. زمانی که تعداد کاربران به حدود 2300 می‌رسد، میانگین response time درخواست صفحه از 1 دقیقه عبور می‌کند.

اگر آستانه تحمل فشار شبکه را بازگذاری شدیم زیر یک دقیقه صفحه dashboard در نظر بگیریم، در حالتی که 2300 کاربر با spawn rate=10 داشته باشیم سرویس دهی شبکه‌مان دچار اختلال می‌شود و میانگین بازگذاری این صفحه از یک دقیقه می‌گذرد.

همانطوری که در عکس زیر مشاهده می‌شود، زمان میانگین بازگذاری صفحه داشبورد 62539 میلی ثانیه با spawn rate=10 کاربر و 2330 است.



Apache JMeter

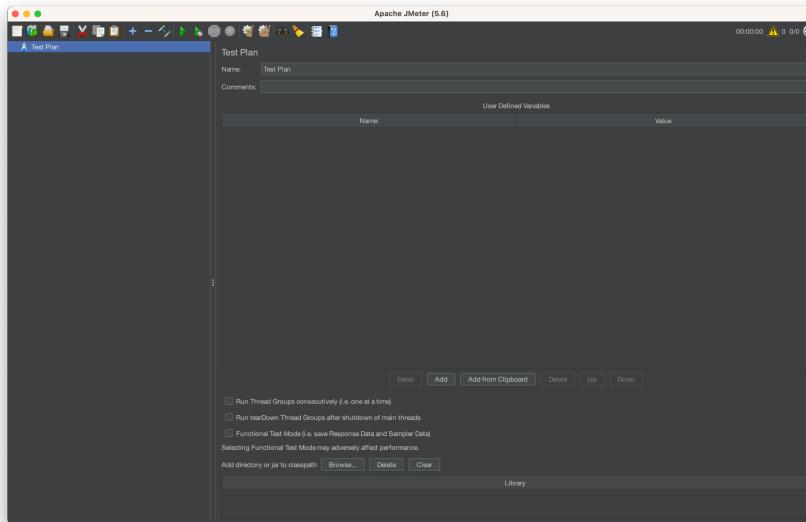
Apache JMeter ابزاری است برای آزمایش عملکرد در منابع استاتیک و پویا و برنامه های کاربردی وب پویا. می توان از آن برای شبیه سازی یک بار سنگین بر روی سرور، گروهی از سرورها، شبکه یا شیء برای آزمایش قدرت آن یا تجزیه و تحلیل عملکرد کلی تحت انواع مختلف بار استفاده کرد.

ویژگی های Apache JMeter عبارتند از:

- قابلیت بارگذاری و تست عملکرد بسیاری از انواع برنامه ها/سرور/پروتکل های مختلف:
- وب - (NodeJS، PHP، ASP.NET، جاوا، HTTP، HTTPS،)
- خدمات وب
- FTP
- JDBC
- LDAP
- میان افزار پیام گرا (MOM) از طریق JMS
- نامه - (IMAP(S، SMTP(S)، POP3(S
- دستورات بومی یا اسکریپت های پوسته
- TCP
- اشیاء جاوا
- مدیران سیستم همچنین می توانند از آن برای آزمایش عملکرد سخت افزار مانند عملکرد سرور یا سرعت شبکه استفاده کنند.
- می توانید گزارش های عملکرد شخصی سازی شده ایجاد کنید و اطلاعات عملکرد خود را به فایل های CSV صادر کنید.
- این به شما امکان می دهد آمار عملکرد و یک مدل را جمع آوری و تجزیه و تحلیل کنید و تعاملات وب پیچیده را شبیه سازی کنید.
- همچنین به شناسایی، رسیدگی و اجتناب از مشکلات عملکرد برنامه ها و سرویس های وب شما کمک می کند.

JMeter نصب

در واقع JMeter نیاز به نصب و تنظیمات خاصی ندارد. کافی است تا فایل زیپ مربوط به ریلیس این ابزار را از صفحه رسمی آن به آدرس jmeter.apache.org دانلود کرده و سپس بسته به سیستم عامل فایل اجرایی مورد نظر در پوشه bin آغاز خواهد شد و ما به صفحه این مانند صفحه زیر روبرو خواهیم شد.



اجرای JMeter بر روی شبکه HumHub

برای اجرای این ابزار بر روی شبکه مورد نظر ابتدا نیاز است تا یک تست پلن ایجاد کنیم. پس از ایجاد تست پلن برای ایجاد تست های تکرار شونده یا Distributed نیاز به یک Thread Group داریم که ما در اینجا از Custom Thread Group استفاده کرده ایم. در اینجا سعی میکنیم تا تنظیمات تست را تا حد ممکن به تنظیمات استفاده شده در Locust نزدیک کنیم. برای مثال همانطور که در تصویر زیر مشخص است تعداد رشته ها را ۱۰۰۰ و Spawn Rate را نیز معادل ۱ یوزر در ثانیه تنظیم کرده ایم.



حالا نیاز است تا در زیرمجموعه این Thread Group مراحل مختلف تست را پیاده سازی کنیم. به این منظور از ابزار های مختلفی که JMeter در اختیار ما قرار می دهد استفاده می کنیم. در این تست مراحل زیر پیاده سازی شده اند که به بررسی آن ها می پردازم.

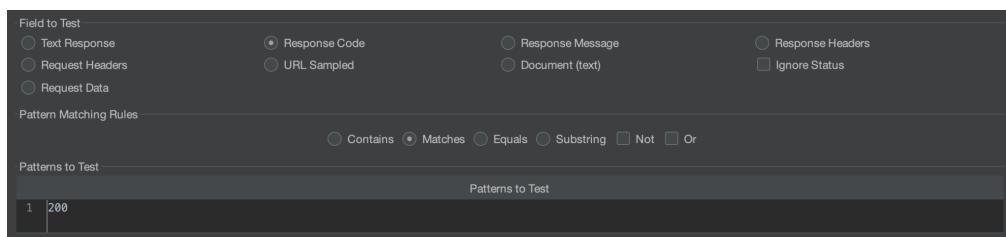


مرحله چک کردن در دسترس بودن وبسایت با درخواست به صفحه داشبورد

در این مرحله درخواستی به صفحه اصلی که شامل داشبورد و آخرين پست ها و ... زده می شود تا در دسترس بودن وبسایت به طور کلی را بررسی نماید. این درخواست یه درخواست ساده از نوع GET می باشد که هیچ سرآیند یا داده خاصی را حمل نمی کند. این درخواست با تنظیمات زیر پیاده سازی می شود.



همچنین بعد از این درخواست نیاز است تا برای این درخواست یه عملگر صحت سنجی یا Assertion ایجاد کنیم. Assertion ها بسته به درخواست های مختلف می توانند انواع مختلفی داشته باشند اما از آنجا که در این درخواست تنها در دسترسی بودن وبسایت مدنظر است می توانیم این تست را با بررسی کد پاسخ وبسایت پیاده سازی کنیم. به این صورت که اگر وبسایت پاسخ ۲۰۰ را در جواب برگرداند تست پاس شده و در غیر اینصورت تست ما ناموفق خواهد بود. پیاده سازی این عملگر نیز به صورت زیر انجام شده است و تنظیمات متفاوت آن در تصویر مشخص است.



مرحله تنظیم سرآیند ها و تست ورود و اعتبار سنجی

از این مرحله به بعد برای انجام عملیات هایی مانند ساختن پست جدید، دنبال کردن کاربر و ... نیاز است تا در سیستم اعتبار سنجی انجام دهیم. به این منظور به استفاده از ابزار مربوط به سرآیندها در JMeter مربوط به اعتبار سنجی را مشابه تصویر زیر تنظیم می کنیم.

HTTP Header Manager	
Name:	HTTP Header Manager
Comments:	
Headers Stored in the Header Manager	
Name:	Value
X-Csrftoken	9cN0 WDEi8PTV97gc04ViPhptkMWhva374hi1t9KwC2S8h3aOfTMgOcf7pkYOl5rJH3dkcsj-S...
Cookie	language=b610205b6655c929cf933c146255cc5b9c427a6636aea7f404616ca6c2822aca%
Content-Type	application/json

سپس نیاز است تا ببینیم که آیا اعتبار سنجی به درستی انجام شده است یا خیر. پس یک درخواست از نوع GET به صفحه `i` ویرایش پروفایل ایجاد میکنیم. اگر اعتبار سنجی به درستی انجام نشده باشد این صفحه ما را به صفحه ورود ریدایرکت خواهد کرد و در غیر این صورت می توانیم با بررسی اینکه کلمه My Profile در صفحه وجود دارد یا غیر عملگر صحبت سنجی مناسب را ایجاد کنیم. پیاده سازی این درخواست و همچنین Assertion آن به صورت زیر می باشد.

HTTP Request	
Name:	Testing Login
Comments:	
<input checked="" type="radio"/> Basic <input type="radio"/> Advanced	
Web Server Protocol [http]: http Server Name or IP: humhub Port Number:	
HTTP Request Method: GET Path: /user/account/edit Content encoding:	
<input checked="" type="checkbox"/> Redirect Automatically <input type="checkbox"/> Follow Redirects <input checked="" type="checkbox"/> Use KeepAlive <input type="checkbox"/> Use multipart/form-data <input type="checkbox"/> Browser-compatible headers	
Field to Test <input checked="" type="radio"/> Text Response <input type="radio"/> Response Code <input type="radio"/> Response Message <input type="radio"/> Response Headers <input type="radio"/> Request Headers <input type="radio"/> URL Sampled <input type="radio"/> Document (text) <input type="radio"/> Ignore Status <input type="radio"/> Request Data	
Pattern Matching Rules <input checked="" type="radio"/> Contains <input type="radio"/> Matches <input type="radio"/> Equals <input type="radio"/> Substring <input type="checkbox"/> Not <input type="checkbox"/> Or	
Patterns to Test Patterns to Test 1 My profile	

تست های ساخت پست جدید، لایک پست، ایجاد کامنت، دنبال کردن کاربر باقی درخواست ها و مراحل نیز با همین مفاهیم اولیه پیاده سازی می شوند. در واقع تفاوت آن ها تنها در آدرس درخواست، نوع درخواست، بدنه درخواست و تنظیمات مربوط به Assertion های مربوط به آن ها می باشد. با ترکیب کردن این تنظیمات می توانیم برای بخش اعظمی از سیستم تست هایی ایجاد کنیم.

اجرای تست و مشاهده نتایج

برای اجرای تست کافیست روی علامت Play سیز رنگ در بالای صفحه کلیک کنیم. اما برای مشاهده نتایج نیاز است تا از ابزار های مختلف مشاهده نتایج استفاده کنیم. کاربردی ترین این ابزارها که در تصاویر زیر نیز مشاهده خواهید کرد شامل Aggregate Report و Summary Report و Results Tree می باشد.

SE Test Plan.jmx (/Users/poorya/Projects/SE Test Plan.jmx) - Apache JMeter (5.6)

00:00:34 ⚠ 0/1000 ⏪

Aggregate Report

Name: Aggregate Report
Comments:
Write results to file / Read from file
Filename:

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput	Received KB/s...	Sent KB/s...
Testing If T...	2000	8495	4006	25904	25907	25912	0	25916	55.65%	44.5/sec	1196.16	15.06
Testing Lo...	1000	1131	739	3443	4280	4441	1	9005	34.90%	32.0/sec	1617.90	25.74
Making a n...	1000	776	674	909	1110	4077	1	25903	28.40%	31.6/sec	1318.28	31.90
Like a Post	1000	728	651	988	1164	2504	1	4080	21.40%	31.8/sec	555.74	30.34
Commenting ...	1000	706	649	942	1045	2002	1	3089	15.80%	31.7/sec	1256.26	32.34
Follow a user	1000	1081	1127	1365	1473	1626	369	2461	33.70%	31.5/sec	2051.46	26.27
TOTAL	7000	3060	840	8859	21422	25908	0	25916	35.07%	133.1/sec	5101.35	100.75

Include group name in label? Save Table Data Save Table Header

SE Test Plan.jmx (/Users/poorya/Projects/SE Test Plan.jmx) - Apache JMeter (5.6)

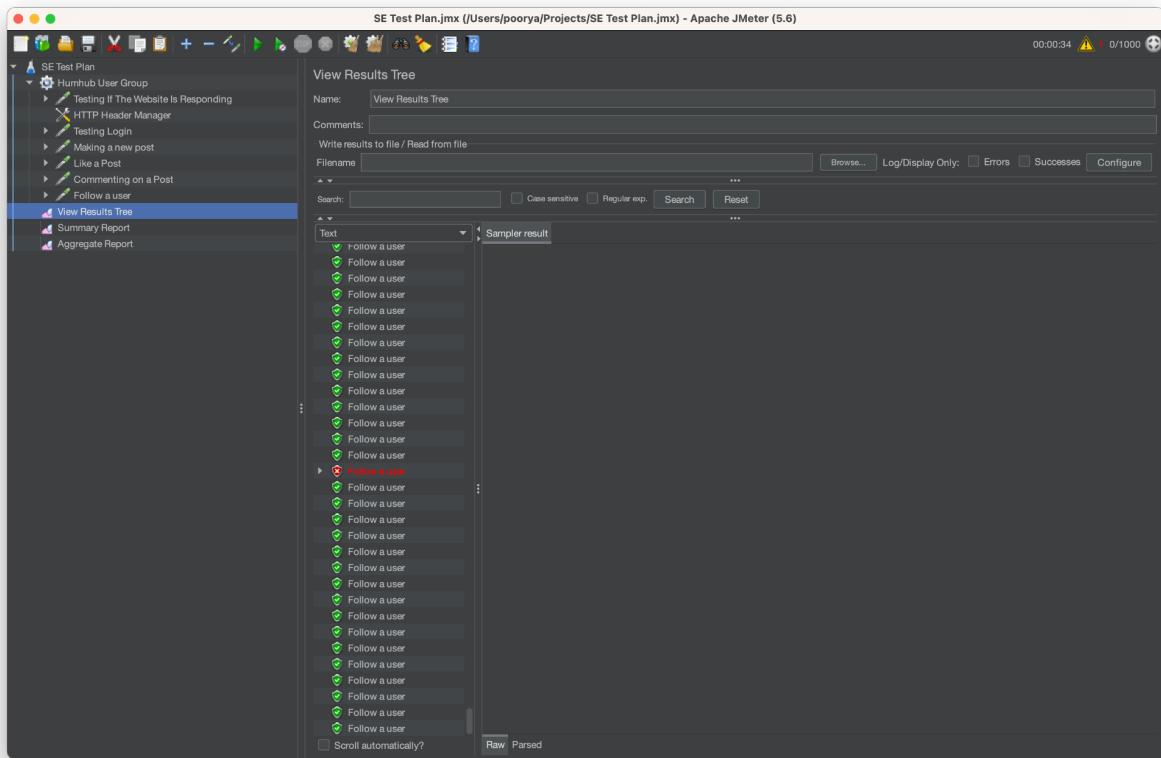
00:00:34 ⚠ 1/0/1000 ⏪

Summary Report

Name: Summary Report
Comments:
Write results to file / Read from file
Filename:

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/s...	Sent KB/sec	Avg. Bytes
Testing If T...	2000	8495	0	25916	9018.65	55.65%	44.5/sec	1196.16	15.06	27517.9
Testing Login	1000	1131	0	9005	1185.59	34.90%	32.0/sec	1617.90	25.74	51801.1
Making a new ...	1000	776	0	25903	932.06	28.40%	31.6/sec	1318.28	31.90	42722.1
Like a Post	1000	728	0	4080	367.36	21.40%	31.8/sec	555.74	30.34	17895.9
Commenting ...	1000	706	0	3089	257.44	15.80%	31.7/sec	1256.26	32.34	40552.8
Follow a user	1000	1081	0	2461	283.09	33.70%	31.5/sec	2051.46	26.27	66747.8
TOTAL	7000	3060	0	25916	5954.64	35.07%	133.1/sec	5101.35	100.75	39250.8

Include group name in label? Save Table Data Save Table Header



پیدا کردن آستانه تحمل فشار شبکه(آزمون فشار) با JMeter و مقایسه با Locust

به این منظور مشابه Custom Thread Group مقادیر Locust را تنظیم می‌کنیم. برای اینکار تعداد کاربران ۵۰۰۰ و Spawn Rate را ۱۰ یوزر در ثانیه انتخاب می‌کنیم. این بار به نتایج کمی متفاوت از Locust میرسیم و آستانه تحمل سیستم حدود ۳۰۰۰ کاربر مشخص می‌شود. البته این تفاوت می‌تواند به علت تفاوت توان سخت افزاری سیستم‌هایی که روی آن‌ها این تست‌ها انجام شده نیز باشد چون Jmeter و Locust دو عضو مختلف گروه با دو سیستم مختلف انجام شده است.

QCacheGrind

با استفاده از این برنامه، میتوانیم خروجی xdebug را که مستقیما قابل تحلیل و بررسی نیست، آنالیز کرده و از اطلاعات زیادی که در اختیار ما قرار می‌دهد استفاده کنیم.

برنامه QCacheGrind توسط نمودارها و رابطهای گرافیکی، زمان اجرا، میزان استفاده از مموری و دیگر اطلاعات آماری بخش‌های مختلف کد و توابع استفاده شده آن به همراه روابط بین آن‌ها را ارائه می‌دهد. با استفاده از این اطلاعات در می‌یابیم که چه توابعی منابع بیشتری را استفاده می‌کنند یا باعث کند شدن اجرا می‌شوند که بتوانیم آن بخش‌ها را بهبود ببخشیم.

برخی از قابلیت‌های این ابزار عبارتند از:

- اطلاعات دقیق از توابع مورد استفاده در سیستم

این ابزار، توابع استفاده شده در سیستم را ارزیابی کرده و میزان زمانی که توسط آن‌ها صرف شده و میزان مموری که استفاده کرده اند را به ما می‌دهد. میتوان توابع به دست آمده را بر اساس زمانی که صرف کرده اند اولویت بندی نمود.

- روابط بین توابع

با استفاده از گراف‌های متنوعی که این ابزار در اختیار ما می‌گذارد، میتوانیم روابط بین توابع را ببینیم. در این صورت Caller‌ها و Calee‌های هر تابع مشخص است، پس وقتی که میبینیم تابعی دارد زمان زیادی را صرف می‌کند، با بررسی توابعی که آن تابع را فراخوانی می‌کنند(به همراه تعدد فراخوانی‌ها) میتوانیم مشکل اصلی را پیدا کنیم. همچنین با داشتن داده‌ها به شکل تصویری، میتوانیم سریع‌تر و بهتر ساختار کدمان و پیچیدگی آن را درک کنیم، به همین دلیل فرایند نگهداری از کد و ریفکتورینگ، به شکل بهینه تری صورت می‌پذیرد.

- به دست آوردن گلواه‌های سیستم(Bottlenecks)

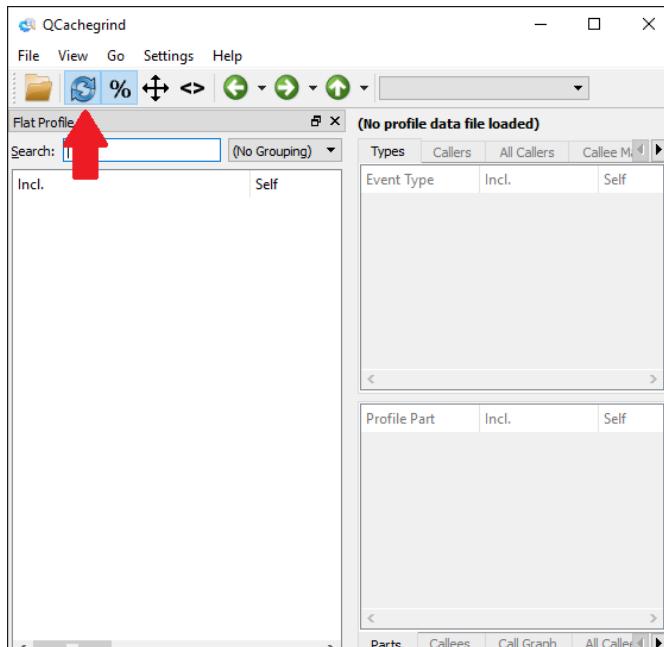
با اطلاعات به دست آمده از دو قابلیت بالا، میتوانیم بخش‌هایی از سیستم که دارند منابع زیادی را مصرف می‌کنند پیدا کنیم، دلایل احتمالی آن را به دست آوریم و قدم‌های اولیه برای بهینه‌سازی سیستم را برداریم.

- مقایسه کد در ورژن‌های مختلف

میتوان از این ابزار برای مقایسه نسخه‌های مختلف کد، استفاده کرده و میزان بهتر شدن آن و تاثیر تغییرات اعمال شده در روند اجرای آن را در مرور زمان بررسی نمود.

نصب QCacheGrind

- ابتدا آخرین نسخه این برنامه را دانلود میکنیم (در زمان نوشتن این گزارش، آخرین نسخه آپدیت شده برای سال 2014 است).
- سپس فایل زیپ دانلود شده را Extract میکنیم.
- با اجرای qcachegrind.exe، برنامه باز میشود و قابل استفاده است.
- طبق گفته صورت پروژه، برای کاهش خروجی های نامناسب و غیر کاربردی، گزینه Detect Cycles را از نوار بالایی برنامه، غیر فعال میکنیم.

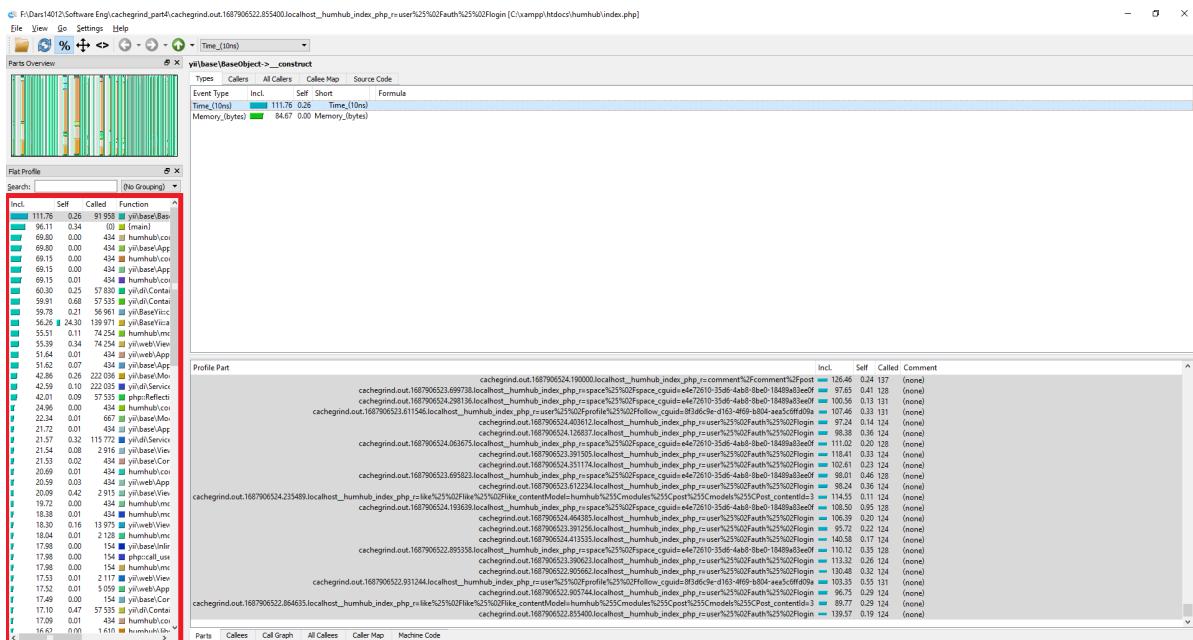


تحلیل خروجی Xdebug در QCacheGrind

بعد از اجرای Locust و با داشتن خروجی های Xdebug، میتوانیم داده ها را در QCacheGrind به تصویر در آورده و تحلیل کنیم.

- برای این بخش، از خروجی های به دست آمده Xdebug پس از اینکه در اجرای Locust به 1000 کاربر رسیدیم، استفاده میکنیم.

پس از باز کردن تمام خروجی ها توسط QCachegrind، با تصویر زیر روبرو می شویم:



بخش "Flat Profile" که در سمت چپ برنامه قابل ملاحظه است، لیستی از تمامی **function call** ها به ترتیب زمانی که مصرف کردہ‌اند به ما می‌دهد.

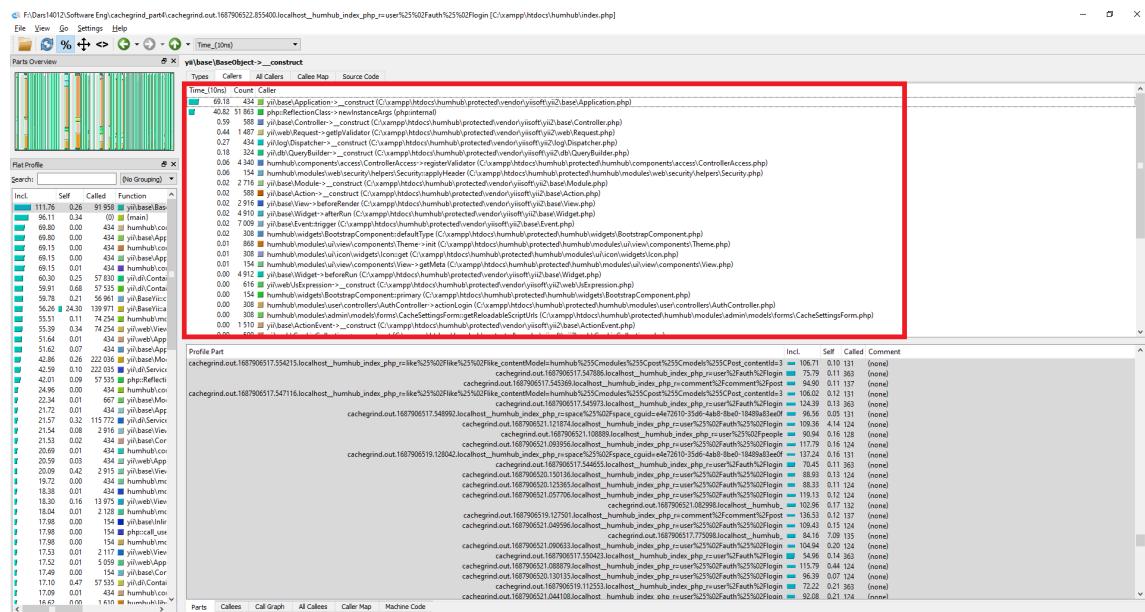
ستون **Icnl**، زمان مصرف شده توسطتابع، به علاوه زمان **callee** ها را نمایش می‌دهد.

ستون **Self**، زمان مصرف شده تابع بدون در نظر گرفتن زمان **callee** ها را نمایش می‌دهد.

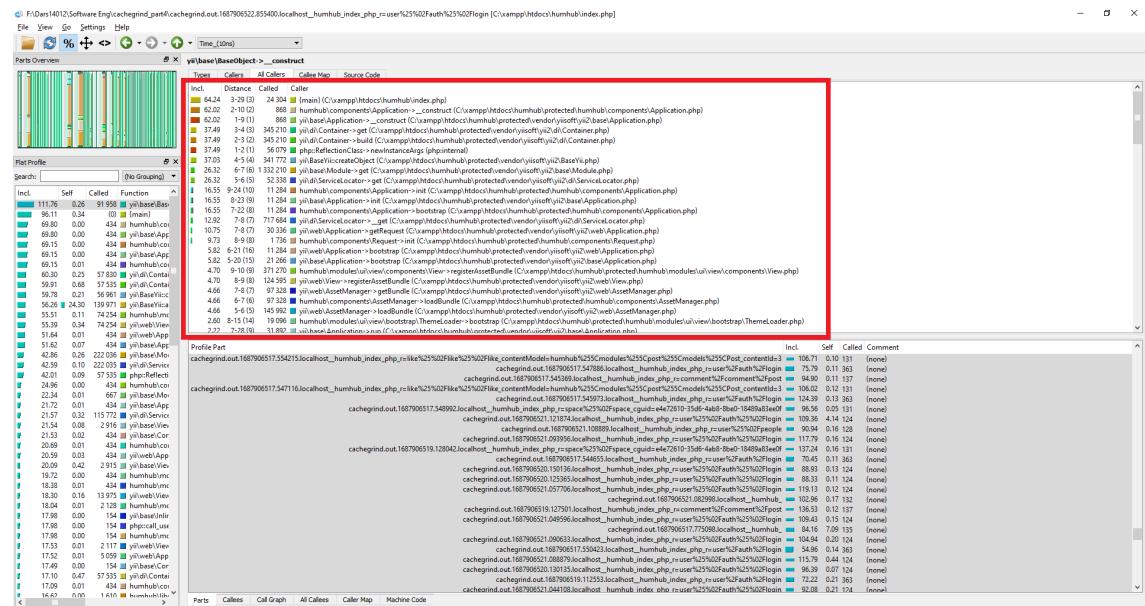
ستون **Called** نماینگر تعداد دفعاتی است که این تابع فراخوانی شده است و ستون **Function** نیز نام تابع را نشان می‌دهد.

در پنل بالایی سمت راست برنامه نیز به **view** های مختلفی دسترسی داریم.
بخش **Type**، اجزای اندازه گیری شده تابع انتخاب شده را نشان می‌دهد. مانند زمان و مموري صرف شده.

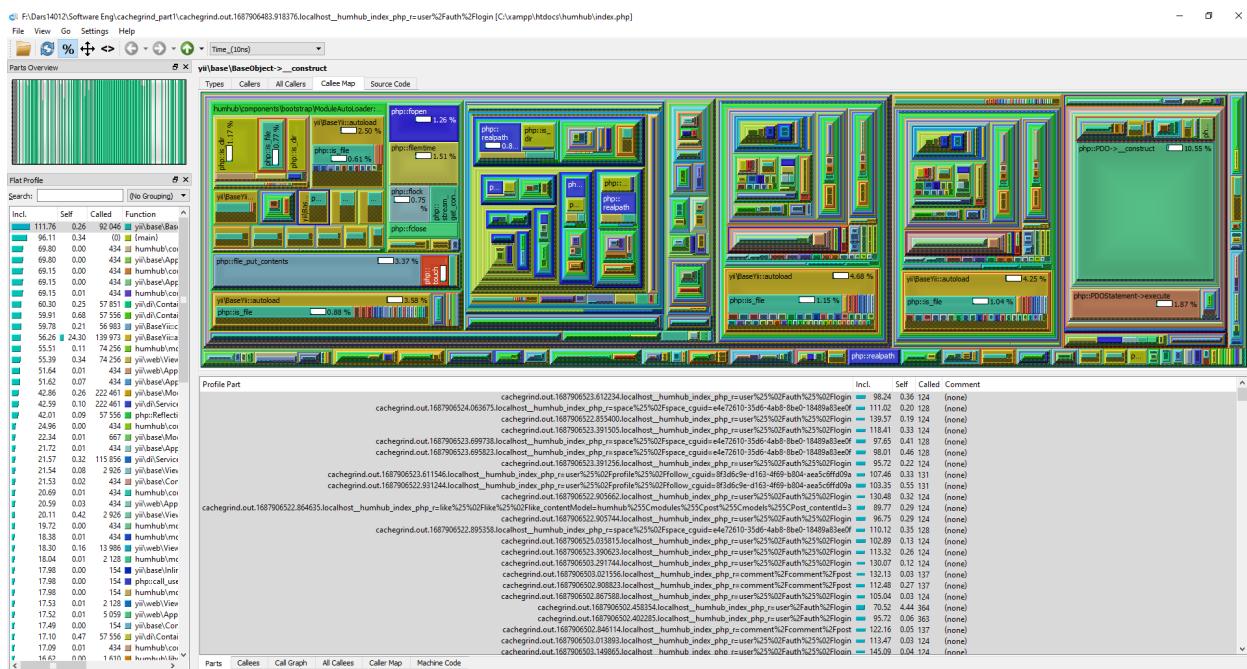
بخش Callers، توابع فراخوانی کننده این تابع را نشان می‌دهد.(آن هایی که این تابع را مستقیماً فراخوانی می‌کنند).



بخش Callers، توابع فراخوانی کننده غیر مستقیم این تابع را نیز نشان می‌دهد.

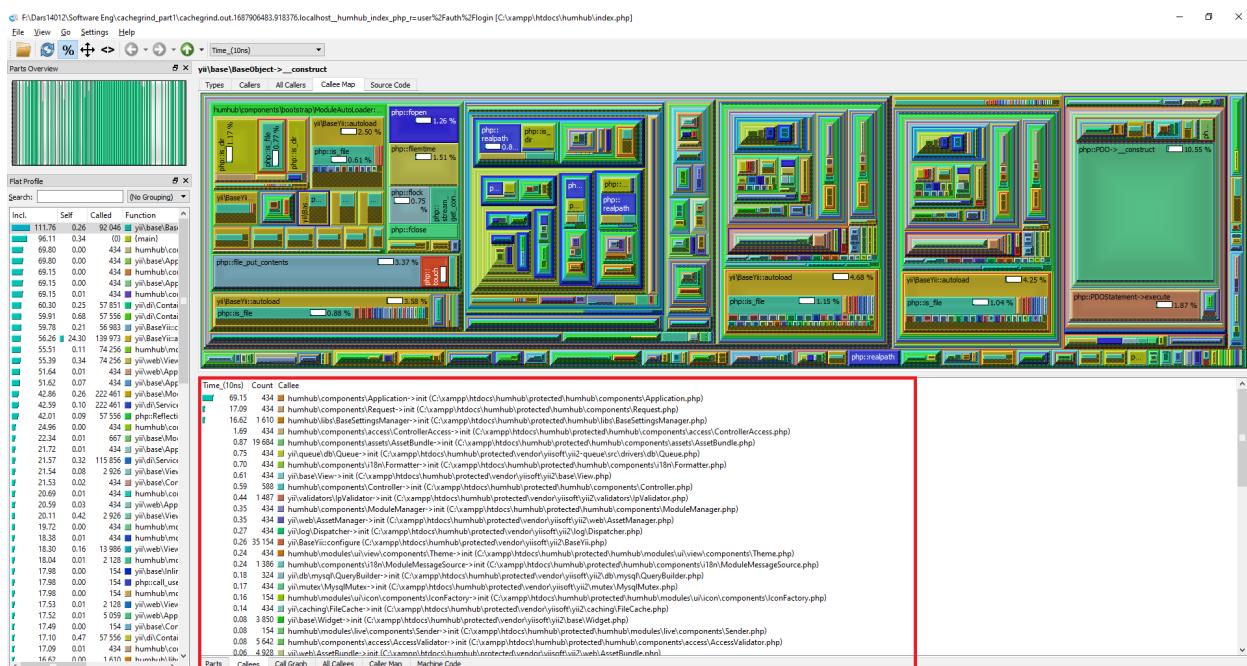


بخش Heatmap، نمایش توابع فراخوانی شونده این تابع را نشان می‌دهد.

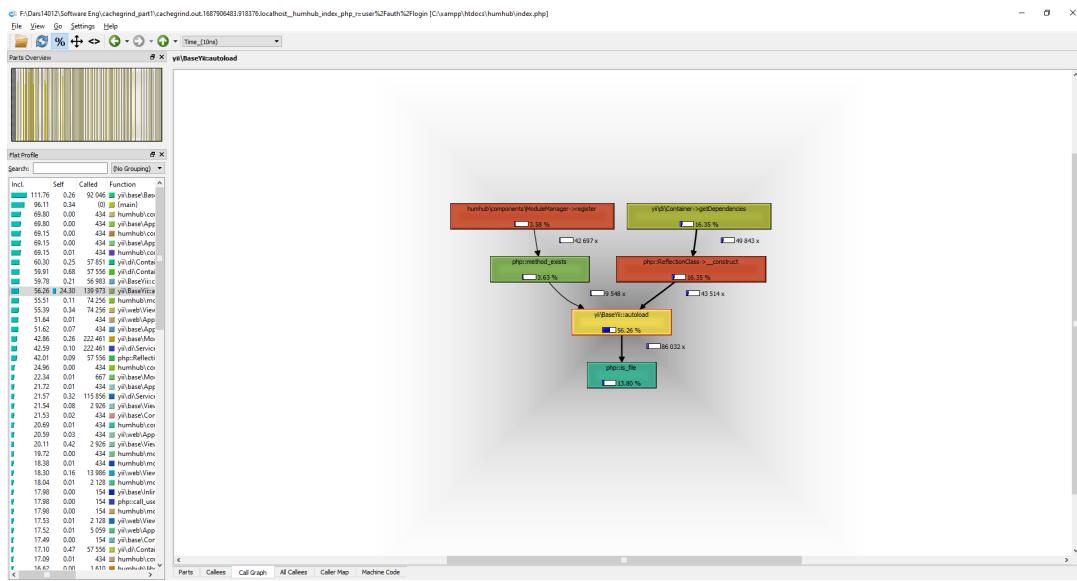


در پنل پایین سمت راست نیز بخش های مختلفی برای نمایش اطلاعات وجود دارد.

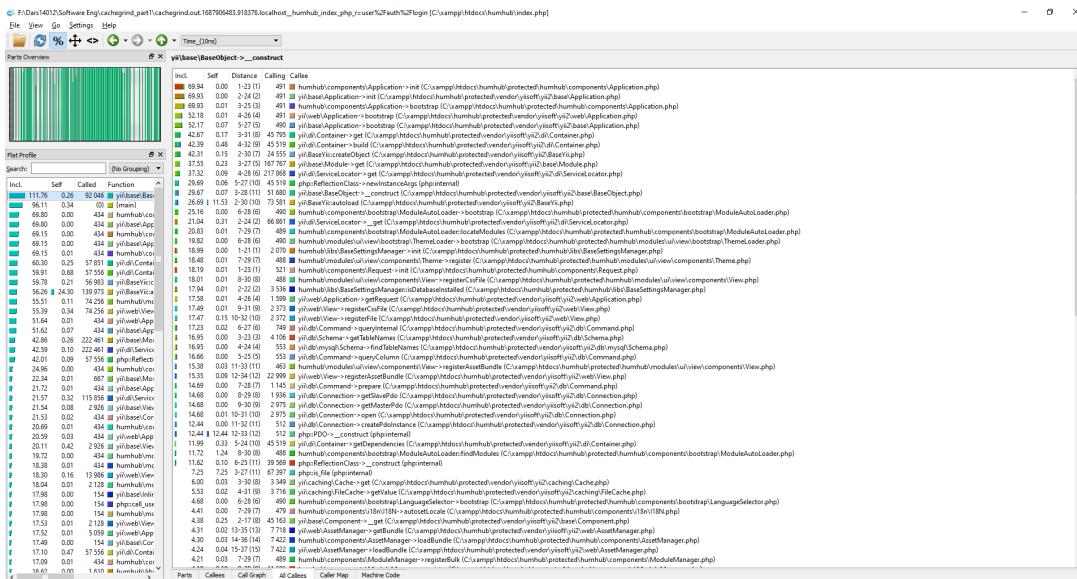
بخش Callees، نشان دهنده توابع تابع انتخاب شده، فراخوانی می‌شوند.



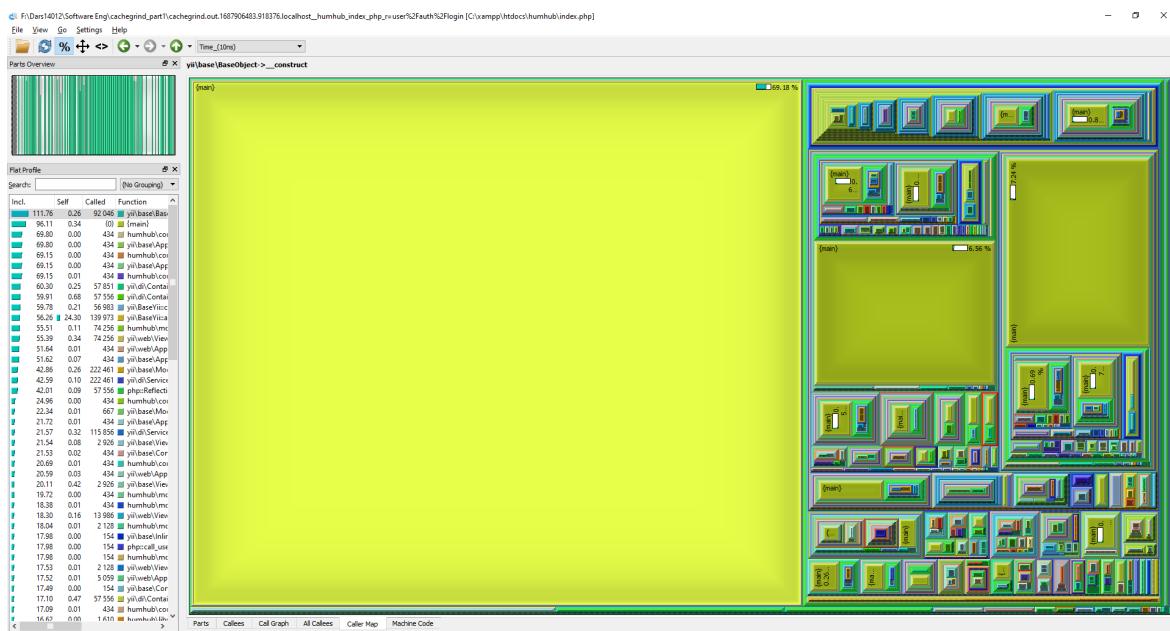
بخش Call graph، جریان فراخوانی ها را به صورت نمودار نمایش می دهد.



بخش All Callees، تمامی توابعی که توسط تابع انتخاب شده فراخوانی می شوند را نشان می دهد.

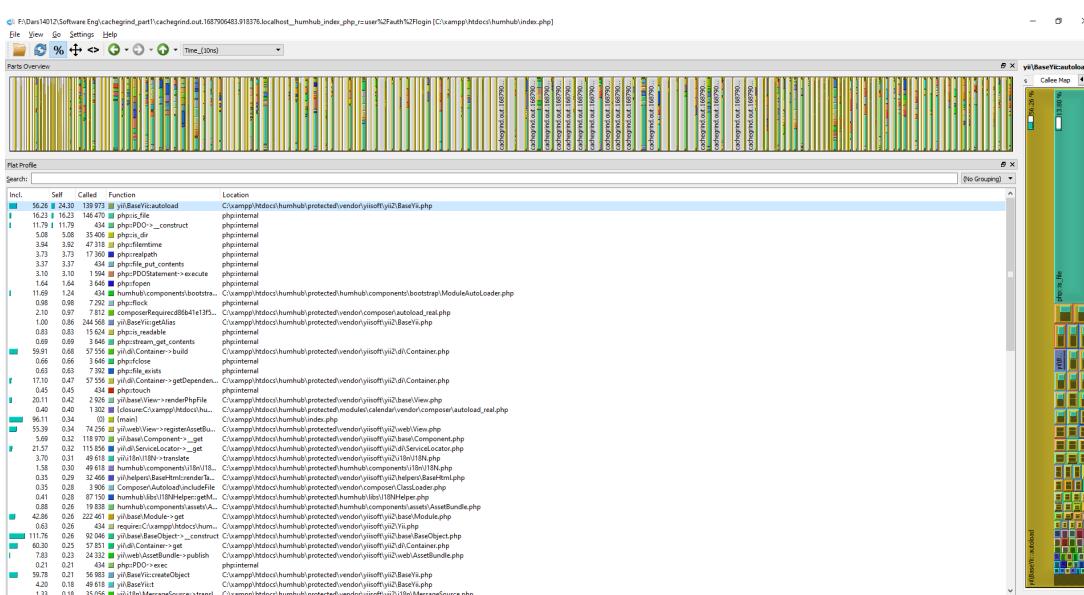


بخش **Caller Map**, نمایش Heatmap توابع فراخوانی کننده این تابع را نشان می‌دهد.



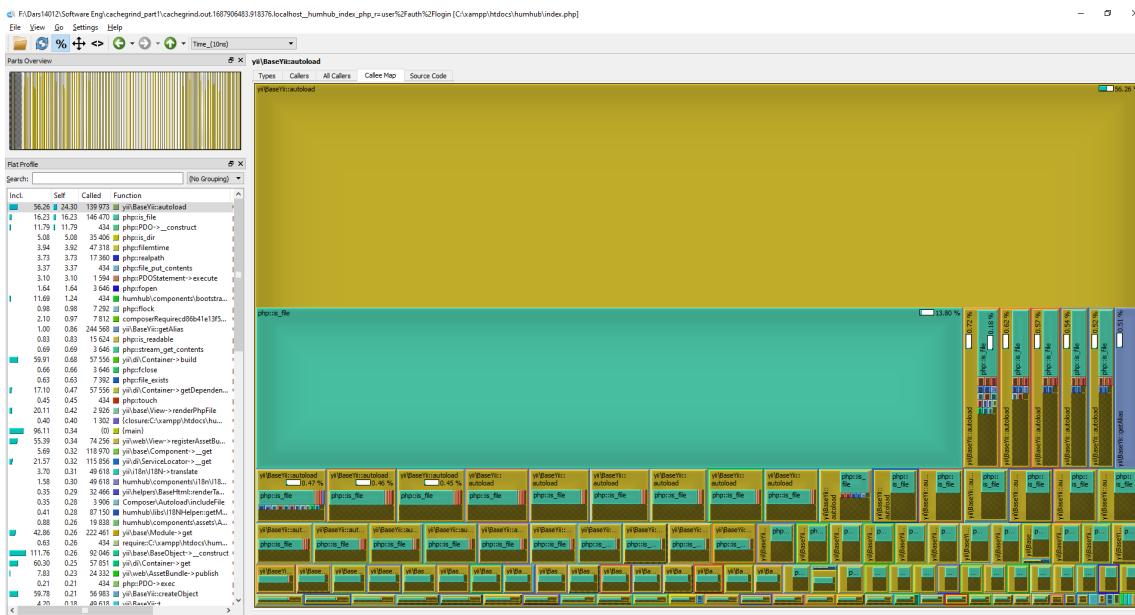
سه تابعی که بیشترین زمان را صرف کرده‌اند

- در این بخش سه تابعی را که بیشترین زمان را مصرف کرده‌اند، مورد بررسی قرار می‌دهیم.
- برای اینکار همانطور که در ابتدای گفته شد، در بخش **Flat Profile**، توابع را بر اساس **Self**، اولویت بندی می‌کنیم.

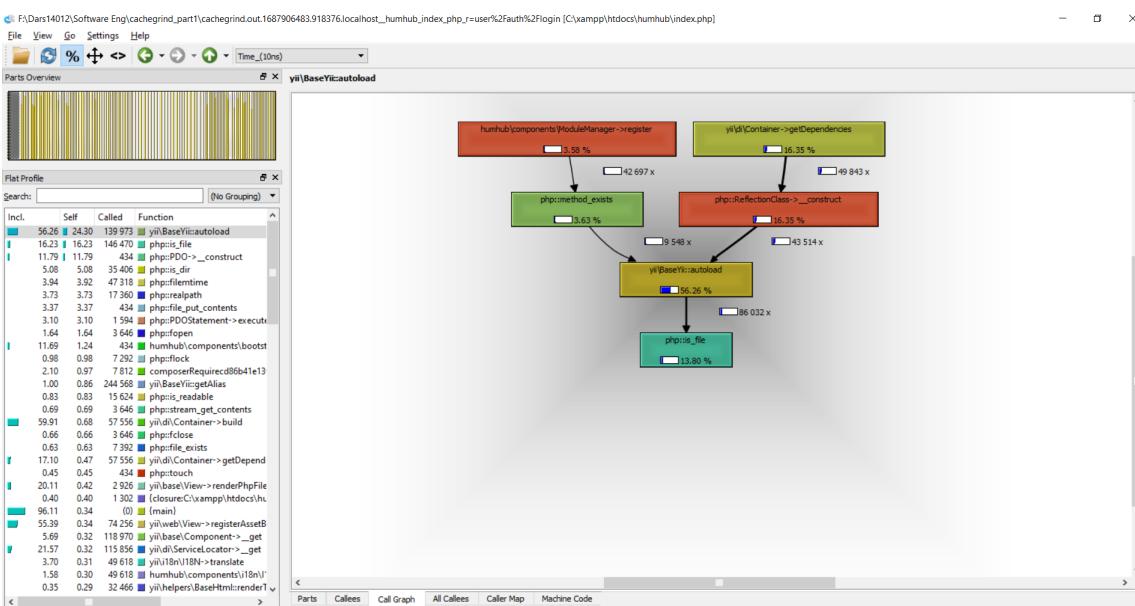


1. تابع yii\BaseYii::autoload

همانطور که در Calee Map این تابع مشاهده می شود، تابع yii\BaseYii::autoload بیشترین زمان اجرا را دارد.

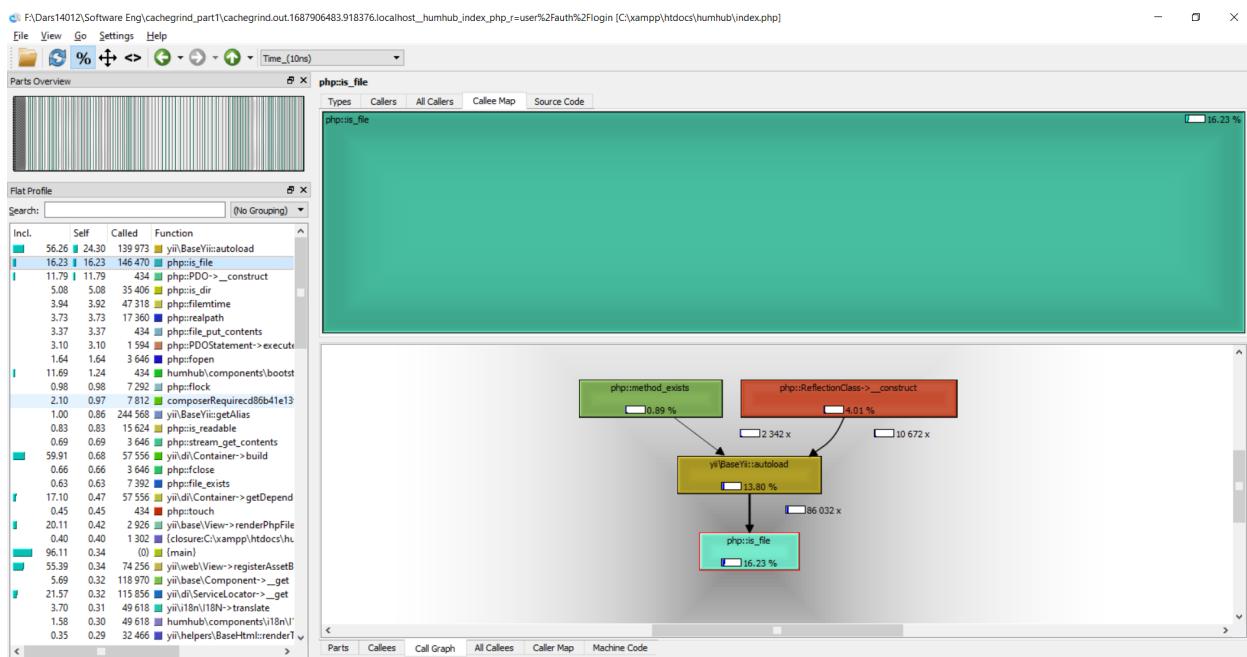


گراف call این تابع، که نمایانگر تعداد و نحوه فراخوانی آن است:



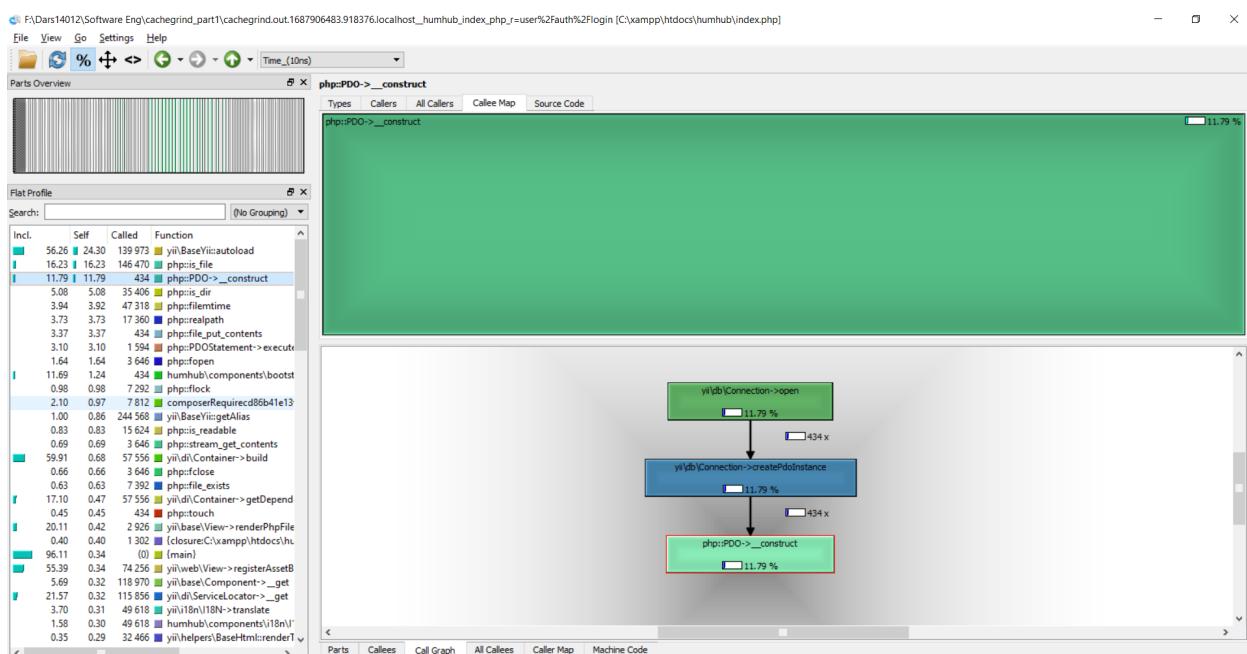
.2 تابع `php::is_file`

طبق نمودار، `php::is_file`، بیشترین زمان اجرا دارد. و گراف تماس های آن نیز مشخص است.



.3 تابع `php::PDO->__construct`

طبق نمودار، `php::PDO->__construct`، بیشترین زمان اجرا دارد. گراف تماس های آن نیز مشخص است.



پایان فاز سوم

امیرمحمد عزتی

پوریا عباسی

ستایش یوسف‌نیا

ابراهیم گلریز

بهار 1402