Visualizing Evolving Service Performance in Python

NAMES ORDER TBA

Johann Bernoulli Institute of Mathematics and Computer Science
University of Groningen
Groningen, the Netherlands
Email: {v.andrikopoulos,m.f.lungu}@rug.nl, {t.klooster.1,p.p.vogel}@student.rug.nl

***Abstract*—The abstract goes here.**

I. Introduction

Every system is a distributed system nowadays []. ML: Vasilios: I can't find anymore the reference to the paper that you sent me a while ago about this topic :(Do you have it's bibtex?)

Python is currently one of the most popular programming languages for API web service development. It was the Todo: nth most popular programming language in the year Todo: 2017/16?

Flask is one of the two most popular web frameworks for Python. It is a lightweight alternative to web site and service development. This is why it is advertised as a *micro-framework*. The popularity of Flask is hinted at also by the large number ML: Thijs, Patrick: can we find an estimate on GH? of open source projects that are hosted on GitHub which are based on it.

However, there is no dedicated solution for monitoring the performance of Flask web-applications. Thus, every one of those Todo: NNNN has one of the following options when confronted with the need of gathering insight into the runtime behavior of their service:

1) Use a heaviweight professional API monitoring setup they require setting up a different server ML: Thijs, Patrick: can we find a few examples of professional but overkill tools? ideally they require setting up a bunch of servers, and writing configs in XML!
.
2) Implement their own analytics tool
3) Live wihout analytics insight into their services [1]

ML: For the first point in the list, we can also argue that analytics solutions like Google Analytics can be used, but they have no notion of versioning/integration with the development lifecycle. Feel free to cite [1] for service evolution purposes

For projects which are done on a budget (e.g. research projects) the first and the second options are often not available due to time and financial constraints.

To avoid these projects ending up in the third situation, in this paper we present a low-effort, lightweight service monitoring API for Flask and Python web-services.

To start using our Python library for service visualization solution one needs to add exactly three lines of code that connect their Flask application object with the dashboard:

Listing 1. TBA

```
import dashboard

dashboard.config.from_file('d.cfg')
dashboard....
```

ML: Thijs, Patrick: fix this example pls

And a small configuration file:

ML: Thijs, Patrick: example of minimal (the most minimal possible, we'll introduce later extra configs) config file. Or, if we can we do completely w/o config that would look even better. And we'll introduce the extra stuff later

ML: Thijs, Patrick: a small description of how the dashboard automatically intercepts the calls to the various API calls

ML: Thijs, Patrick: metion also that the dashboard automatically is available at /dashboard endpoint

ML: Thijs, Patrick: a small screenshot of how the dashboard allows one to select the interesting

II. Case Study

Zeeguu case study description to be used as running example throughout the rest of the paper [2]

Architecture: series of web and mobile applications built aroud a core web service implemented in Python and Flask which provides:

- contextual translations
- reading recommendations
- exercises

We have this system for helping learners read texts that they like, and enable them to practice with exericses generated on their past readings.

ML: we should consider adding also one section in which the architecture/implementation and main features of the dashboard are presented before going on with discussing them in more depth in the following sections — this should include a rundown on which views are provided from where (overview or per endpoint)

III. Overall Endpoint Utilization

The most basic insights that a service maintainer must answer regard the utilization of their service. Dashboard presents two perspectives that provide insight into this question.

Figure 1 shows the usage of the services on a daily basis while highlighting the contribution of the individual endpoints[2].

[1]This is very real option: and is exactly what happened to the API that will be presented in this case study for many months.

[2]We recommend obtaining a color version of this paper for better readability

One can see that the service at its peak has about 2500 hits per day. Moreover, one can see the way the users are using the platform since the different endpoints are indicators of different activity types, e.g.:

- **api.get_possible_translations** is an indicator of the amount of reading the users are doing
- **api.report_exercise_outcome** is an indicator of the amount of vocabulary practice the users are doing

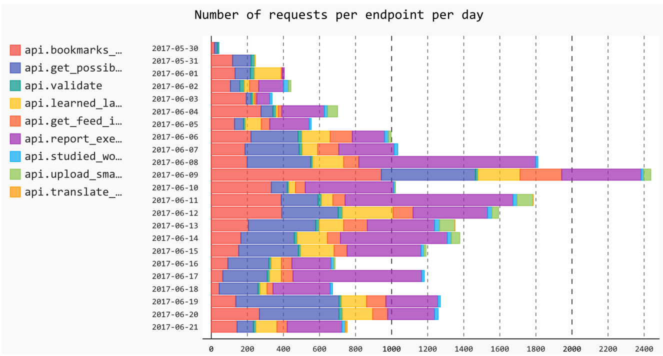

Fig. 1. The number of requests per endpoint per day view shows the overall utilization of the monitored application

ML: Thijs, Patrick: Could we regenerte this graph with the same colors as in the previous one? Otherwise the Vissoft people will surely complain

Besides showing which endpoints are used, this view is informative for the maintainer since it provides insight into which endpoints are used. In our case study, the maintainer of the API has decided to not remove several endpoints when they saw that that were being used [3]

A second question that an API maintainer must answer regards cyclic patterns of usage.

ML: Vasilios: can we provide some support for this claim? a reference maybe?

ML: Thijs, Patrick: can we add vertical lines that highlight the beginng of a new week (e.g. before Sunday):

Figure 2 shows only one clear pattern: the users of our API do not practice foreign languages as night. Besides that, the API is used around the clock with usually a few hundred hits per hour. So the load currently is not that high.

IV. Visualizing Service Performance

Another type of information that Dashboard can collect regards endpoint performance. Figure 3 shows a summary of the response times for various endpoints by using boxplot graphs.

After investigating this view it became clear to the maintainer that three of the endpoints had very large variation in performance. One of the most important endpoints that they decided to improve the performance of is the second from the top in Figure 3: the endpoint that returns translations. This is particularily critical since it is part of a live interaction loop.

[3]Usage information can also be used to increase the confidence of the maintianer that a given endpoint is not used

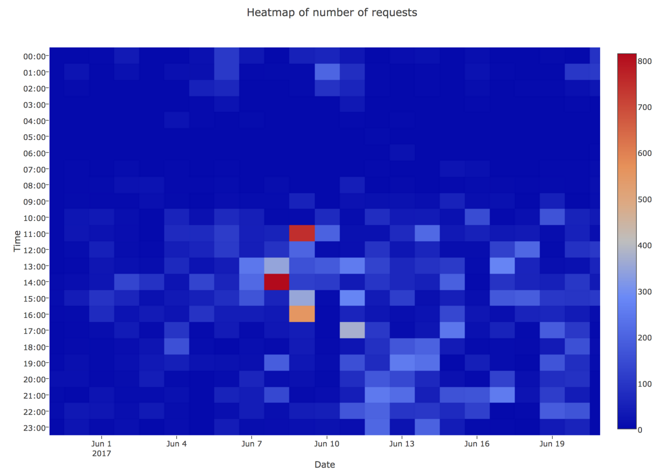

Fig. 2. Usage patterns become easy to spot in the requests per hour heatmap

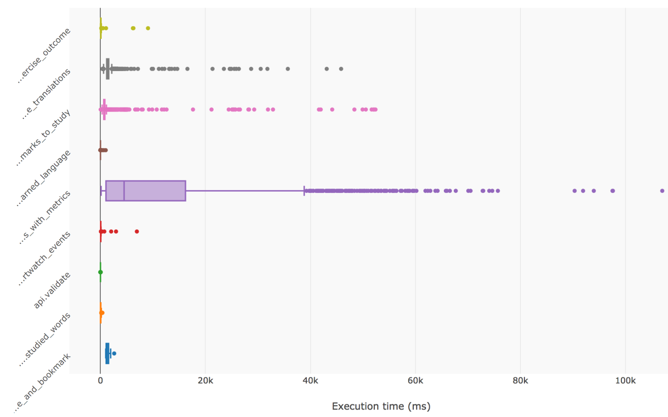

Fig. 3. The response time (in ms) per monitored endpoint view allows for identifying performance variability and balancing issues

Once they increased the performance of the endpoint, the maintainer realized that this graph was not that useful anymore. Because they could not see the improvements.

For such situations, Dashboard provides an extra configuration directive:

ML: Thijs, Patrick: Add extra config line here

If the directive is enabled, the performance data is grouped by commit and commits are detected by analyzing the .git folder on the machine.

Figure 4 shows the performance of the give endpoint

This way can confirm that our performance improves: in the latest version (bottom-most box plot in Figure 4) we see the entire boxplot move to the left.

One requirement that the maintainer requested at this point was **extra informaiton about outliers**: Python stack trace, CPU load, request parameters, etc. in order for them to

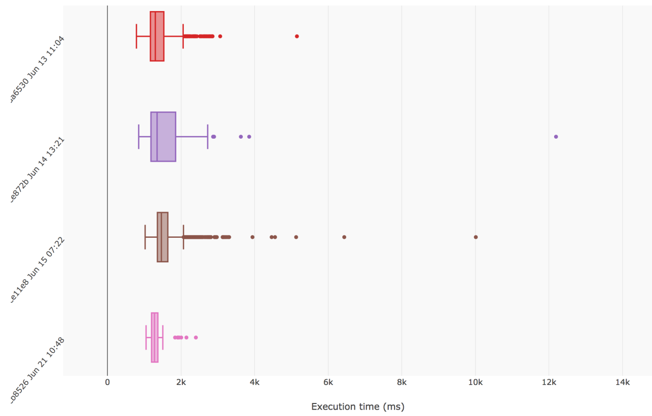

Fig. 4. Visualizing The Performance Evolution of the **api.get_possible_translations** endpoint

Listing 2. TBA

```
# app specific way of extracting the user
# from a flask request object
def get_user_id(request):
  sid = int(request.args['session'])
  session = User.find_for_session(sid)
  return user_id

# attaching the get_user_id function
dashboard.config.get_group_by = get_user_id
```

investigate the causes of these exceptionally slow response times.

In order to address this, but not degrade the usual performance the Dashboard tracks for every endpoint a running average value. When it detects that a given endpoint is an outlier with respect to this value, it triggers an exceptional data collection routine which stores extra information about the current execution environment.

V. User Centered Visualization

For service endpoints which run computations in real time as they are called, there might be very different timings based on the different loads that are sent to the endpoint.

In our cases study, one of the slowest endpoints, and one with the higest variability is **api.get_article_difficulties**: it retrieves a list of recommended articles for a given user. However, since a user can be subscribed to anything from one to three dozen article sources, and since the computation of the difficulty is personalized and it is slow, the variability in time among users is likely to be very large.

Dashboard provides a way of grouping information on a per user basis. However, to do this, the developer must specify the way in which a given API call can be associated with a given user. There are multiple ways, the simplest takes again advantage of the strengths of the Flask framework which offers a global request object which contains session information:

Sometimes, grouping the service calls per endpoint it is not sufficient. Figure 5 shows some of the results of calling the **api.get_article_difficulties** endpoint for various users.

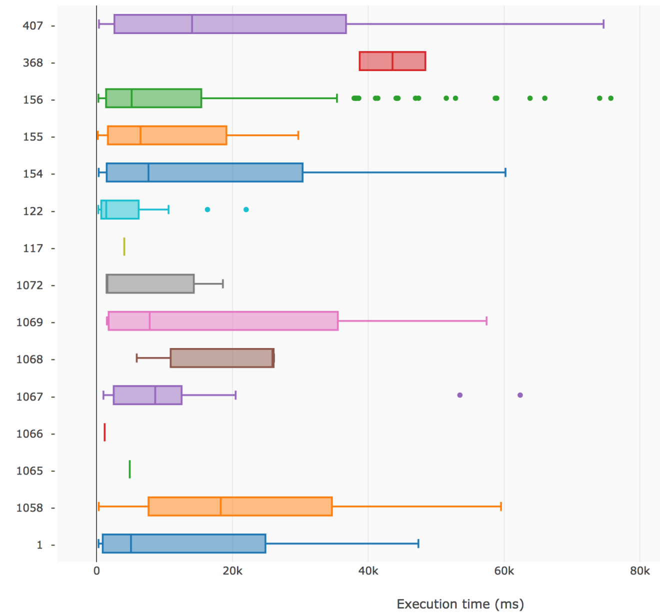

Fig. 5. The **api.get_article_difficulties** shows a very high variability across users

Different users might have different experiences: - a user has 10K emails one has 10 emails - a user is subscribed to 20 feeds one to 2 feeds

The system will have different processing times. It is important for the DevOps-er to be able to understand the difference in performance on a per user basis.

If we try to show also per version:

Fig. 6. Caption here

VI. Tool Availability

The code of Dashboard is available online at ...

The images in this paper are screenshots of the actual deployment of the tool which can be found at https://zeeguu.unibe.ch/api/dashboard. For the readers of this paper to be able to see the tool in action, they can login with the username and password: guest, dashboardguest!. ML: Thijs, Patrick: we should add a guest/guest username password which is allowed to only visualize things, but not modify anything, and not export anything!

VII. Related Work

Java Visualization [3]

Run-time monitoring of services [4]

VIII. Conclusion

The conclusion goes here.

Acknowledgment

The authors would like to thank...

References

[1] M. Papazoglou, V. Andrikopoulos, and S. Benbernou, “Managing evolving services,” *Software*, vol. 28, no. 3, pp. 49–55, 2011.

[2] M. F. Lungu, “Bootstrapping an ubiquitous monitoring ecosystem for accelerating vocabulary acquisition,” in *Proccedings of the 10th European Conference on Software Architecture Workshops*, ser. ECSAW 2016. New York, NY, USA: ACM, 2016, pp. 28:1–28:4. [Online]. Available: http://doi.acm.org/10.1145/2993412.3003389

[3] W. De Pauw, E. Jensen, N. Mitchell, G. Sevitsky, J. M. Vlissides, and J. Yang, “Visualizing the execution of java programs,” in *Revised Lectures on Software Visualization, International Seminar*. London, UK: Springer-Verlag, 2002, pp. 151–162.

[4] C. Ghezzi and S. Guinea, “Run-time monitoring in service-oriented architectures,” in *Test and analysis of web services*. Springer, 2007, pp. 237–264.