Performance tests for different Roary implementations

Lukas Gruler Nikolas Rank Steffen Knoblauch

June 2021

1 Preliminaries

We decided to compare the implementation based on Django and Node.js, because these were the only implementations that we don't run in an VM and are therefore better to compare, because the VM doesn't influence the performance.

In order to be able to test the implementations we added a new API interface for both implementations that will drop all existing roaries and create a specified amount of new (random) roaries. For example: we used it to create 1000 roaries.

2 Client-Performance

For the client side evaluation we used Lighthouse with different amounts of roaries. We executed the measurement for both implementations with 10 and 1000 roaries with the following results.

2.1 Node client performance

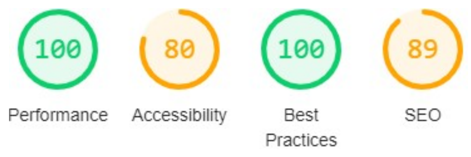


Figure 1: Node implementation with 10 roaries

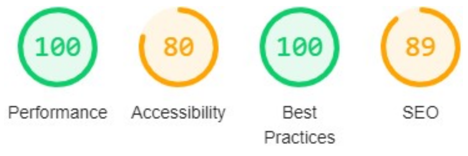
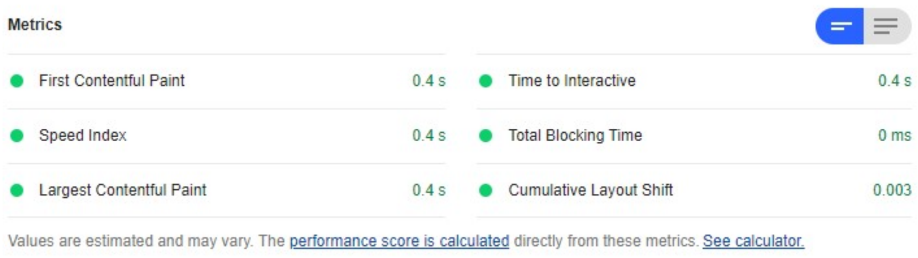


Figure 2: Node implementation with 1000 roaries

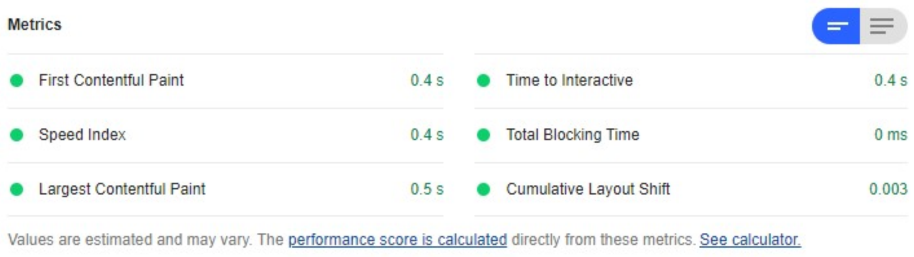


Metrics

● First Contentful Paint	0.4 s	● Time to Interactive	0.4 s
● Speed Index	0.4 s	● Total Blocking Time	0 ms
● Largest Contentful Paint	0.4 s	● Cumulative Layout Shift	0.003

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

Figure 3: Node implementation client performance with 10 roaries



Metrics

● First Contentful Paint	0.4 s	● Time to Interactive	0.4 s
● Speed Index	0.4 s	● Total Blocking Time	0 ms
● Largest Contentful Paint	0.5 s	● Cumulative Layout Shift	0.003

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

Figure 4: Node implementation client performance with 1000 roaries

2.2 Django client performance

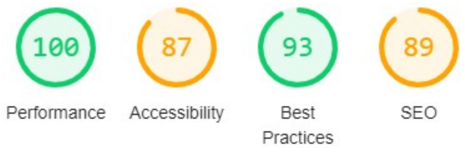


Figure 5: Django implementation with 10 roaries

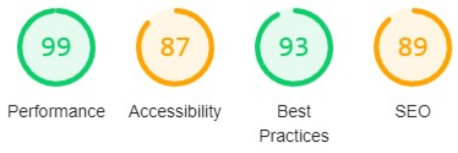
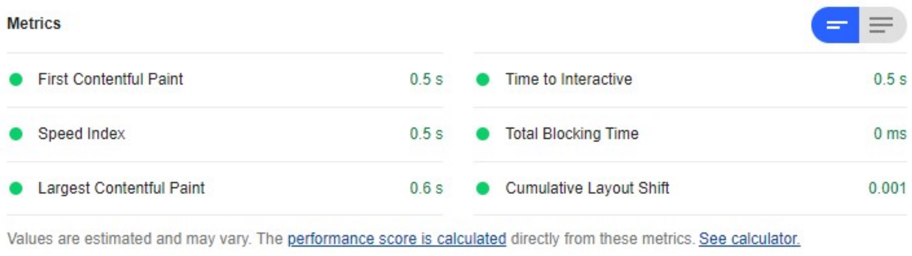


Figure 6: Django implementation with 1000 roaries

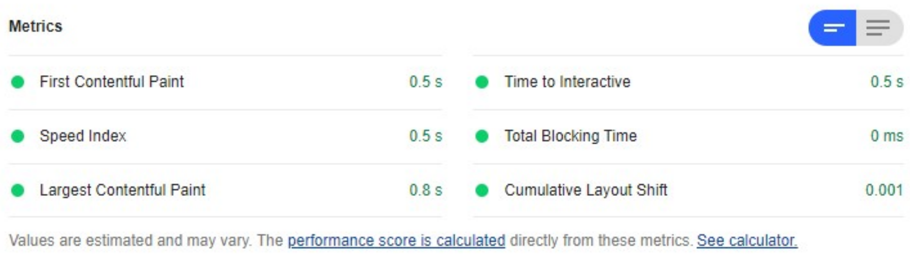


Metrics

●	First Contentful Paint	0.5 s	●	Time to Interactive	0.5 s
●	Speed Index	0.5 s	●	Total Blocking Time	0 ms
●	Largest Contentful Paint	0.6 s	●	Cumulative Layout Shift	0.001

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

Figure 7: Django implementation client performance with 10 roaries



Metrics

● First Contentful Paint	0.5 s	● Time to Interactive	0.5 s
● Speed Index	0.5 s	● Total Blocking Time	0 ms
● Largest Contentful Paint	0.8 s	● Cumulative Layout Shift	0.001

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

Figure 8: Django implementation client performance with 1000 roaries

2.3 Evaluation

In general we could only see minor differences between both implementations (with Django and Node). We could observe that the amount of roaries has a small impact on both of the implementations: By comparing loading the page with 10 and 1000 roaries the Node implementation performed 0.1s slower, the Django implementation 0.2s slower, but this could also be due to inaccurate measurements.
In total the implementation with Django was a little less performant than the Node approach, but both implementation scored very high in regards to the performance.

3 Server-Performance

For the server side evaluation we use wrk2. We executed all tests on the same machine with different parameters and different amounts of roaries.
In the first step we measured the time to load the index page for both implementations. In order to have a set of different measurements we altered the different parameters like the number of connections or the amount of requests.
In the second step we measured the time to load the roars that are going to be displayed on the page.
The diagrams contain two curves - the blue one for the Django implementation (the filenames shown on the x-axis start with "dj") and the red one for the Node implementation (the filenames shown on the x-axis start with "n").

3.1 Index loading

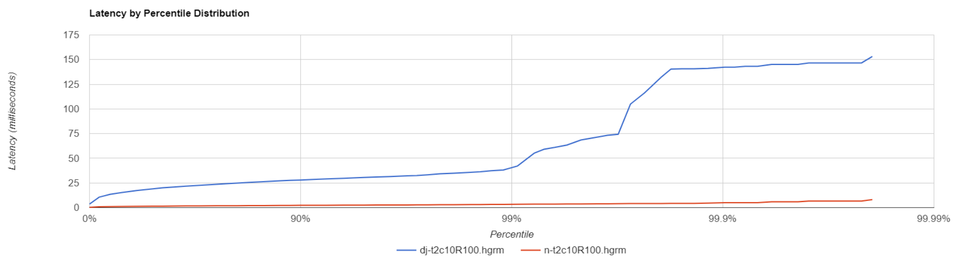


Figure 9: Index measured with: 2 threads, 10 connections and 100 requests per second

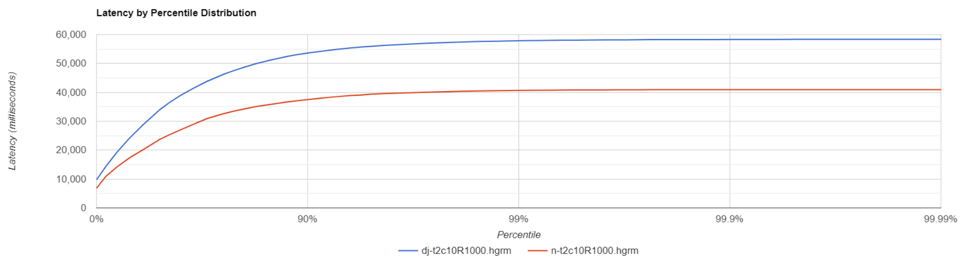


Figure 10: Index measured with: 2 threads, 10 connections and 1000 requests per second

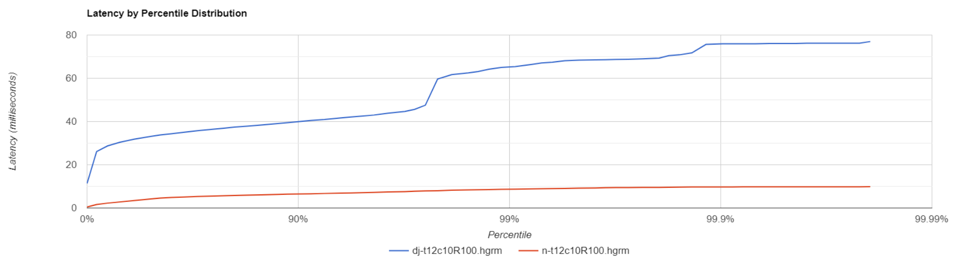


Figure 11: Index measured with: 12 threads, 10 connections and 100 requests per second

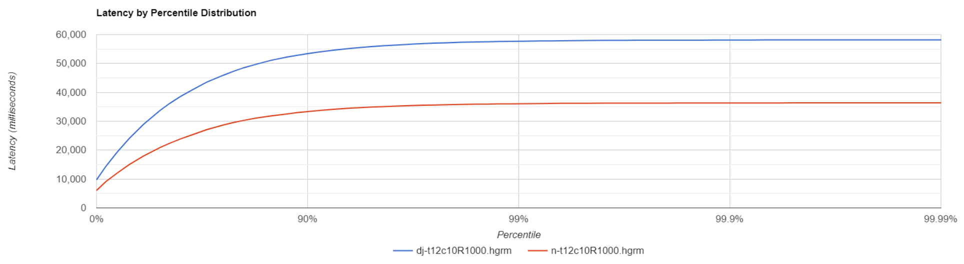


Figure 12: Index measured with: 12 threads, 10 connections and 1000 requests per second

3.2 Roar loading

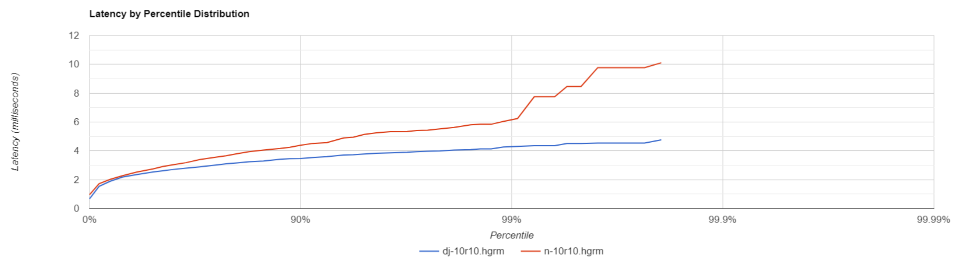


Figure 13: Roars measured with: 10 roars and 10 requests per second

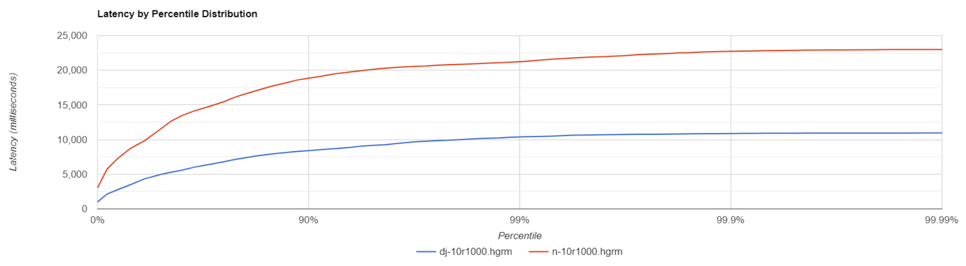


Figure 14: Roars measured with: 10 roars and 1000 requests per second

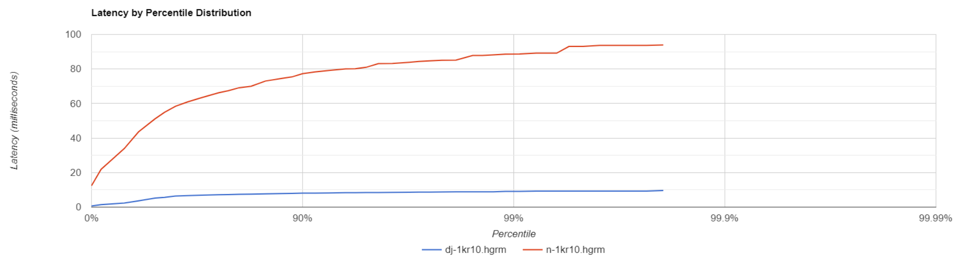


Figure 15: Roars measured with: 1000 roars and 10 requests per second

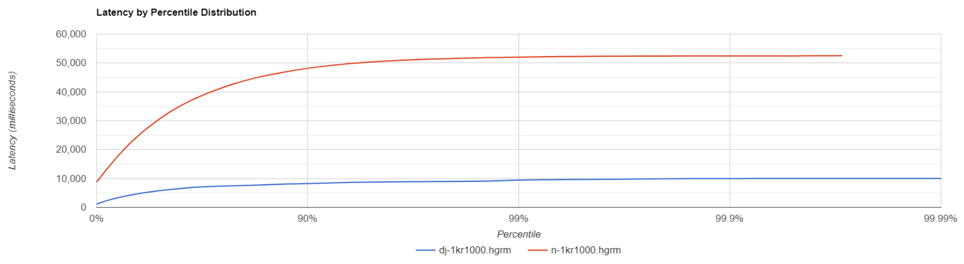


Figure 16: Roars measured with: 1000 roars and 1000 requests per second

3.3 Evaluation

For the loading of the index page the implementation of Node performed better than the Django implementation. A reason for this might be that we used the development server for the Django implementation which might negatively impact the performance. In addition to this, we didn't use a production build which might perform better for both implementations.
In contrast to this, for the roary loading the Node implementation performed worse than the Django implementation (partly up to about nine times as bad). A possible reason for this could be the SQL query/database access we used in the Node implementation because this is the most significant difference between the two implementations. The more roaries we used in the test the worse the Node implementation performed compared to the Django implementation.

3.4 Notes and Problems

The creation of test cases/data and a consistent database that can be used to compare the different measurements required to create the same data for each test again. In order to do this we added some additional functionality to our implementations because the creation by hand was too time consuming.
In general there are several problems for the comparison because the implementations are not really comparable. Reason for this is that we use different database designs (ORM Django and custom SQL code) and in general the structure of the website differs.
A problem we faced quiet fast was that we made some typos in the parameters of the test cases which made some of our measurements not usable anymore, because we couldn't compare them.

4 DISCLAIMER

IT WAS SO HOT THAT THE TEST MIGHT BE INFLUENCED BY THIS ! ;)