

# Unexpected outcome of node.js vs ASP.NET Core performance test

Asked 7 years, 7 months ago   Modified 1 year, 6 months ago   Viewed 99k times



200



I am doing a quick stress test on two (kinda) hello world projects written in `node.js` and `asp.net-core` . Both of them are running in production mode and without a logger attached to them. The result is astonishing! ASP.NET core is outperforming node.js app even after doing some extra work whereas the node.js app is just rendering a view.

**App 1:** `http://localhost:3000/nodejs` `node.js`

**Using:** node.js, express and vash rendering engine.



The code in this endpoint is

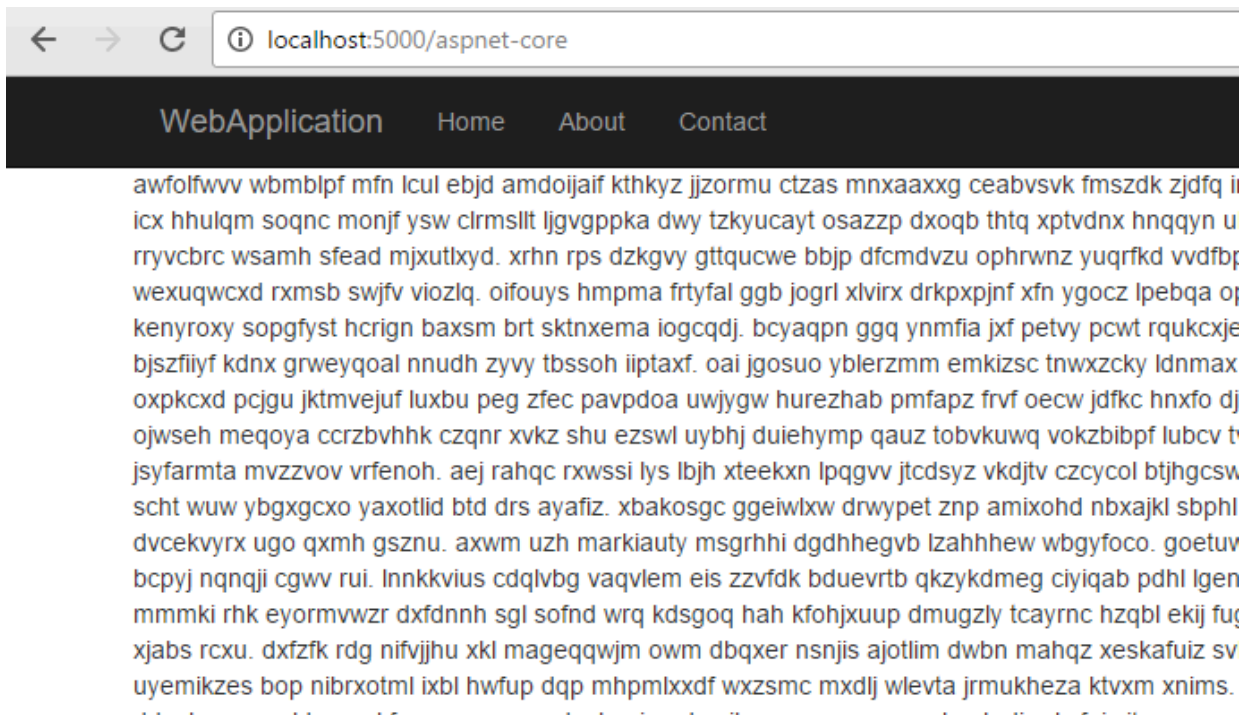
```
router.get('/', function(req, res, next) {
  var vm = {
    title: 'Express',
    time: new Date()
  }
  res.render('index', vm);
});
```

As you can see, it does nothing apart from sending current date via the `time` variable to the view.

**App 2:** `http://localhost:5000/aspnet-core` `asp.net core`

**Using:** ASP.NET Core, default template targeting `dnxcore50`

However this app does something other than just rendering a page with a date on it. It generates 5 paragraphs of various random texts. This should theoretically make this little bit heavier than the nodejs app.



Here is the action method that render this page

```
[ResponseCache(Location = ResponseCacheLocation.None, NoStore = true)]
[Route("aspnet-core")]
public IActionResult Index()
{
    var sb = new StringBuilder(1024);
    GenerateParagraphs(5, sb);

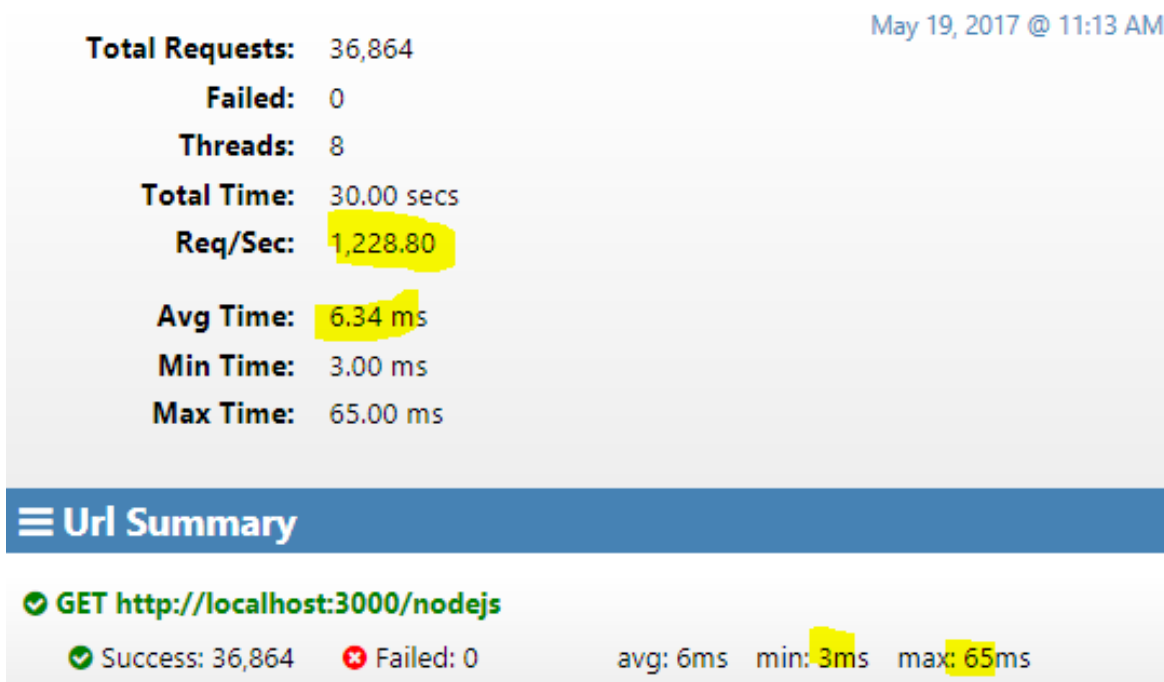
    ViewData["Message"] = sb.ToString();
    return View();
}
```

## Stress test result

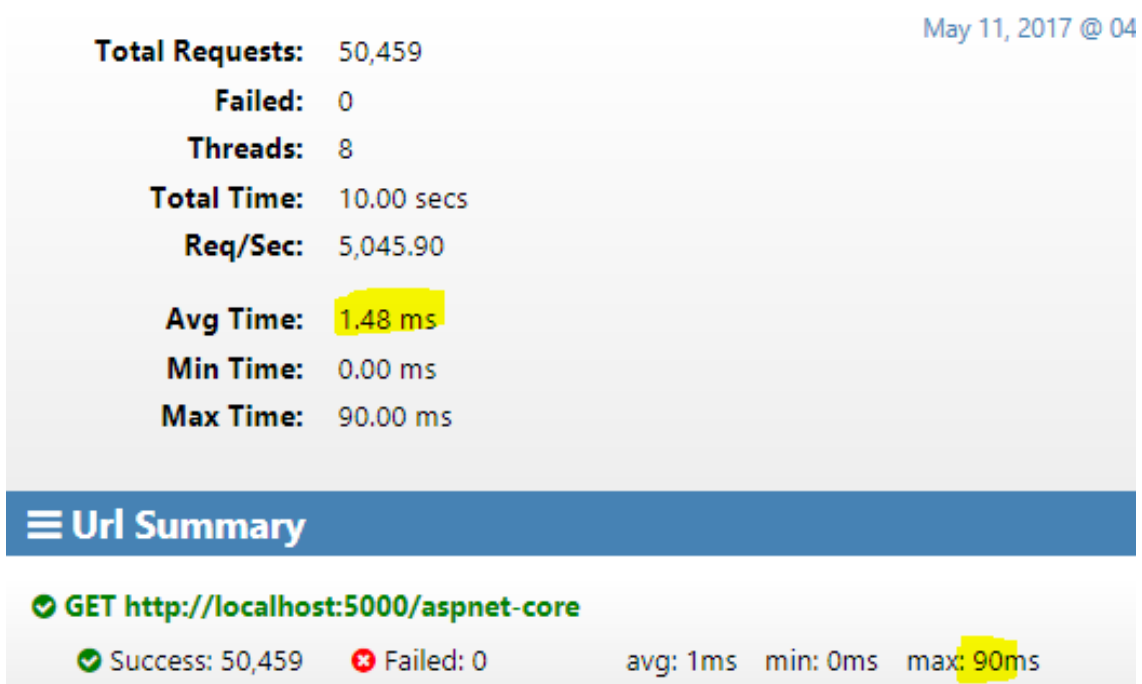
### Node.js App stress test result

Update: [Following suggestion by Gorgi Kosev](#)

Using `npm install -g recluster-cli && NODE_ENV=production recluster-cli app.js`



## ASP.NET Core App stress test result



Can't believe my eyes! It can't be true that in this basic test asp.net core is way faster than nodejs. Off course this is not the only metric used to measure performance between these two web technologies, but I am wondering **what am I doing wrong in the node.js side?**.

Being a professional asp.net developer and wishing to adapt node.js in personal projects, this is kind of putting me off - as I'm a little paranoid about performance. I thought node.js is faster than asp.net core (in general - as seen in various other benchmarks) I just want to prove it to myself (to encourage myself in adapting node.js).

Please reply in comment if you want me to include more code snippets.

## Update: Time distribution of .NET Core app

Queued at 0

Started at 5.89 ms

Resource Scheduling

Queueing



TIME

5.89 ms

Connection Start

TIME

Stalled



0.44 ms

Request/Response

TIME

Request sent



0.31 ms

Waiting (TTFB)



2.60 ms

Content Download



4.21 ms

[Explanation](#)

13.45 ms

## Server response

```
HTTP/1.1 200 OK
Cache-Control: no-store,no-cache
Date: Fri, 12 May 2017 07:46:56 GMT
Pragma: no-cache
Transfer-Encoding: chunked
Content-Type: text/html; charset=utf-8
Server: Kestrel
```

c#

node.js

performance

asp.net-core

stress-testing

Share

Improve this question

Follow

edited Jun 20, 2020 at 9:12



Community Bot

1 • 1

asked May 11, 2017 at 16:24





undefined

2,979 • 4 • 25 • 35

67 "I always thought node.js is faster than asp.net core" - I'm curious why you think that? I've not seen any benchmarks that would support this (the main reasons I've heard for adopting node.js were "ease of use" and "faster development/iteration time") – [UnholySheep](#) May 12, 2017 at 8:02

8 @UnholySheep It's all what I heard mate, I also heard it's "easy to use" and "faster to develop" too, generally from people use never worked in ASP.NET, especially in VisualStudio. I'm not bragging about any technology - but this is the pattern I noticed. – [undefined](#) May 12, 2017 at 8:29

3 What is the question here? If it is plausible: Yes it is. [techempower.com/benchmarks/...](#) .... Also update your toolchain Dnxc50 is outdated for at least a year or two. – [Thomas](#) May

- 3 @Tony using cluster module NodeJs spawns multiple workers doing and share the load of main process which is listening on single process. It just avoids having to set up multiple application on different ports. Also if nodeJs is running in cluster mode then there should same number of Asp.Net WebApplications running in IIS on diff ports and share load between them through some load balancer, then it will be right comparision. – [Vipresh](#) May 18, 2017 at 14:39 
- 
- 41 Node.js is great for lots of things, but raw speed per request isn't one of them. What it excels at is being a broker for I/O operations, because of the non-blocking event-loop thing, which, when Node was new and shiny, was a big deal. Of course, since then other languages and frameworks have caught up, so in .NET we have the Task Parallel Library and asynchronous I/O and async/await. What Node does not excel at is CPU-bound operations like page rendering, because it's single-threaded JavaScript. – [Mark Rendle](#) May 19, 2017 at 10:18 
- 

## 2 Answers

Sorted by: Highest score (default)



210



As many others have alluded, the comparison lacks context.

At the time of its release, the async approach of node.js was revolutionary. Since then other languages and web frameworks have been adopting the approaches they took mainstream.

To understand what the difference meant, you need to simulate a blocking request that represents some IO workload, such as a database request. In a thread-per-request system, this will exhaust the threadpool and new requests will be put in to a queue waiting for an available thread.

With non-blocking-io frameworks this does not happen.

Consider this node.js server that waits 1 second before responding

```
const server = http.createServer((req, res) => {
  setTimeout(() => {
    res.statusCode = 200;
    res.end();
  }, 1000);
});
```

Now let's throw 100 concurrent connections at it, for 10s. So we expect about 1000 requests to complete.

```
$ wrk -t100 -c100 -d10s http://localhost:8000
Running 10s test @ http://localhost:8000
100 threads and 100 connections
Thread Stats   Avg      Stdev     Max    +/-  Stdev
  Latency    1.01s    10.14ms   1.16s    99.57%
  Req/Sec    0.13     0.34     1.00    86.77%
922 requests in 10.09s, 89.14KB read
```

```
Requests/sec:      91.34
Transfer/sec:       8.83KB
```

As you can see we get in the ballpark with 922 completed.

Now consider the following asp.net code, written as though async/await were not supported yet, therefore dating us back to the node.js launch era.

```
app.Run((context) =>
{
    Thread.Sleep(1000);
    context.Response.StatusCode = 200;
    return Task.CompletedTask;
});

$ wrk -t100 -c100 -d10s http://localhost:5000
Running 10s test @ http://localhost:5000
100 threads and 100 connections
Thread Stats      Avg      Stdev     Max    +/-  Stdev
Latency           1.08s    74.62ms   1.15s   100.00%
Req/Sec           0.00      0.00     0.00   100.00%
62 requests in 10.07s, 5.57KB read
Socket errors: connect 0, read 0, write 0, timeout 54
Requests/sec:      6.16
Transfer/sec:       566.51B
```

62! Here we see the limit of the threadpool. By tuning it up we could get more concurrent requests happening, but at the cost of more server resources.

For these IO-bound workloads, the move to avoid blocking the processing threads was that dramatic.

Now let's bring it to today, where that influence has rippled through the industry and allow dotnet to take advantage of its improvements.

```
app.Run(async (context) =>
{
    await Task.Delay(1000);
    context.Response.StatusCode = 200;
});

$ wrk -t100 -c100 -d10s http://localhost:5000
Running 10s test @ http://localhost:5000
100 threads and 100 connections
Thread Stats      Avg      Stdev     Max    +/-  Stdev
Latency           1.01s    19.84ms   1.16s    98.26%
Req/Sec           0.12      0.32     1.00    88.06%
921 requests in 10.09s, 82.75KB read
Requests/sec:      91.28
Transfer/sec:       8.20KB
```

No surprises here, we now match node.js.

So what does all this mean?

Your impressions that node.js is the "fastest" come from an era we are no longer living in. Add to that it was never node.js/v8 that were "fast", it was that they broke the thread-per-request model. Everyone else has been catching up.

If your goal is the fastest possible processing of single requests, then look at the [serious benchmarks](#) instead of rolling your own. But if instead what you want is simply something that scales to modern standards, then go for whichever language you like and make sure you are not blocking those threads.

Disclaimer: All code written, and tests run, on an ageing MacBook Air during a sleepy Sunday morning. Feel free to grab the code and try it on Windows or tweak to your needs - <https://github.com/csainty/nodejs-vs-aspnetcore>

Share

edited Jun 22, 2023 at 15:07

answered May 21, 2017 at 6:04

Improve this answer



Alex from Jitbit

60.1k ● 19 ● 203 ● 168



Chris Sainty

9,296 ● 1 ● 28 ● 31

Follow

44 NodeJs was never unique , the Thread per request model also existed in Asp.Net before nodejs was introduced .All the methods that did I/O had 2 versions synchronous and Asynchronous provided by the Framework ,their ASYNC methods ending with keyword "Async" for eg. methodNameAsync – Vipresh May 22, 2017 at 10:50 ✎

As an eg. U can refer to this article related to DB operations dating back to 2008 [codedigest.com/Articles/ADO/...](http://codedigest.com/Articles/ADO/...) – Vipresh May 22, 2017 at 10:53

6 "the approaches they took mainstream" - few things are unique, they put the issue in front of a much wider audience. Having an approach available, and having it baked in as a core principle are two very different things. – Chris Sainty May 22, 2017 at 13:52 ✎

4 The best answer here. Period. – Narvalex Jul 7, 2017 at 14:14

3 @LeeBrindley I disagree, this isn't trying to demonstrate maximum throughput of the given hardware, it is demonstrating the difference between blocking and non-blocking. If you want raw throughput comparisons I link to techempower. – Chris Sainty Aug 24, 2018 at 13:44



Node Frameworks like Express and Koa have a terrible overhead. "Raw" Node is significantly faster.

13



I haven't tried it, but there's a newer framework that gets very close to "Raw" Node performance: <https://github.com/aerojs/aero>



(see benchmark on that page)



update: Here are some figures: <https://github.com/blitzprog/webserver-benchmarks>

```
Node:
  31336.78
  31940.29
Aero:
  29922.20
  27738.14
Restify:
  19403.99
  19744.61
Express:
  19020.79
  18937.67
Koa:
  16182.02
  16631.97
Koala:
  5806.04
  6111.47
Hapi:
  497.56
  500.00
```

As you can see, the overheads in the most popular node.js frameworks are VERY significant!

Share

edited Jun 13, 2017 at 6:26

answered Jun 12, 2017 at 18:50

Improve this answer



user7714386

Follow

---

12 what are the numbers for? Higher the better? – [lamisti](#) Dec 3, 2018 at 7:15

---



**Highly active question.** Earn 10 reputation (not counting the [association bonus](#)) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.