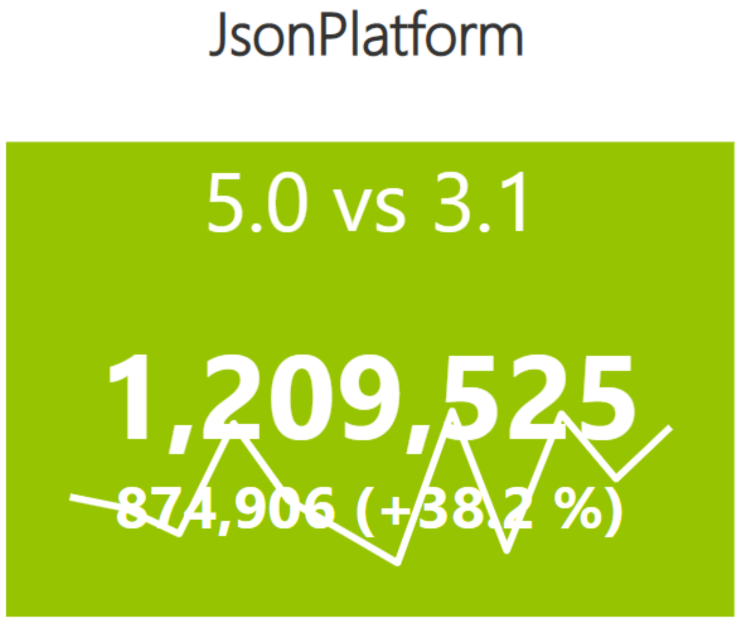
Running .NET Core performance investigation on Linux

Adam Sitnik

About Myself

- BenchmarkDotNet maintainer
- Bibliotecario #dotnet #Microsoft
  - Building Performance Culture
  - Preventing|Detecting|Solving regressions in .NET Core
  - Making .NET Core even faster
  - Making various .NET libraries faster: ML.NET, .NET for Apache Spark
  - Closing Windows ⬄ Linux gap
  - Improving ASP.NET Core performance on Linux
  - Not a Linux expert (yet).

.NET 5 Sneak Peek

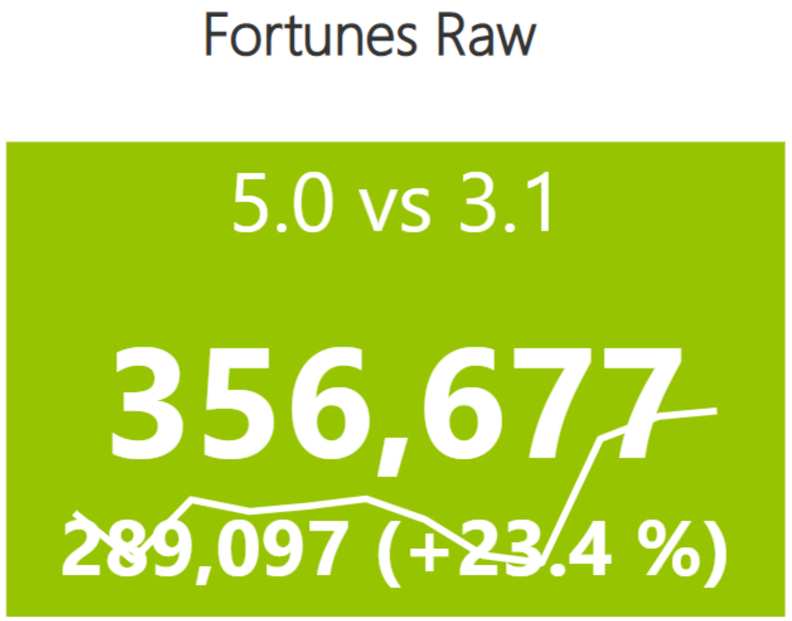


JsonPlatform

5.0 vs 3.1

1,209,525

874,906 (+38.2 %)



Fortunes Raw

5.0 vs 3.1

356,677

289,097 (+23.4 %)

Measure, measure, measure.

Without data you’re just another person with an opinion

— W. Edwards Deming, a data scientist

Benchmark? Profiler?

„In computing, a benchmark is the act of running a computer program, a set of programs, or other operations, in order to assess the relative performance of an object, normally by running a number of standard tests and trials against it”

Wikipedia

„In software engineering, profiling ("program profiling", "software profiling") is a form of dynamic program analysis that measures, for example, the space (memory) or time complexity of a program, the usage of particular instructions, or the frequency and duration of function calls. Most commonly, profiling information serves to aid program optimization.”

Wikipedia

Recommended Settings

- Release (not Debug)
- Symbols:

```
<DebugType>pdbonly</DebugType>
<DebugSymbols>true</DebugSymbols>
```

- Disable Tiered JIT (or warmup the code)

```
<TieredCompilation>false</TieredCompilation>
```

Small Repro: ML.NET regression

	Before	**After**
.NET Core	2.2	3.0
Tiered JIT	Disabled by default	Enabled by default
Vectorized Math	Native library	Managed library

- Narrow down:
  - Disable Tiered JIT and run the .NET Core 3.0 benchmarks
  - Run version with native dependency as .NET Core 3.0

Problem:
  - Vectorized Math library!

Choose the right Profiler

- dotnet trace
  - Works always and everywhere with .NET Core 3.0+!
  - if you don’t need native call stacks
  - if you can’t run as Admin/sudo
- PerfCollect
  - if you need native call stacks and can run as sudo
  - very powerful, small overhead
- VTune
- Rider

dotnet trace

- Cross platform
- .NET Core 3.0+
- No need to run as Admin|sudo
- Lacks native call stacks

Simple commands

- dotnet tool install --global dotnet-trace
- dotnet trace list-processes
- dotnet trace collect -p $pid
- dotnet trace convert $inputFile --format speedscope
- dotnet trace collect -p $pid --format speedscope

#profileURL

- Speedscope allows us to download profile info from given URL
- GIST + Speedscope: https://www.speedscope.app/#profileURL=https://gist.githubusercontent.com/adamsitnik/299f66845a3733514c613f8ac00fefd4/raw/def280919d17928001431c157c0812c6f8605332/after.speedscope.json

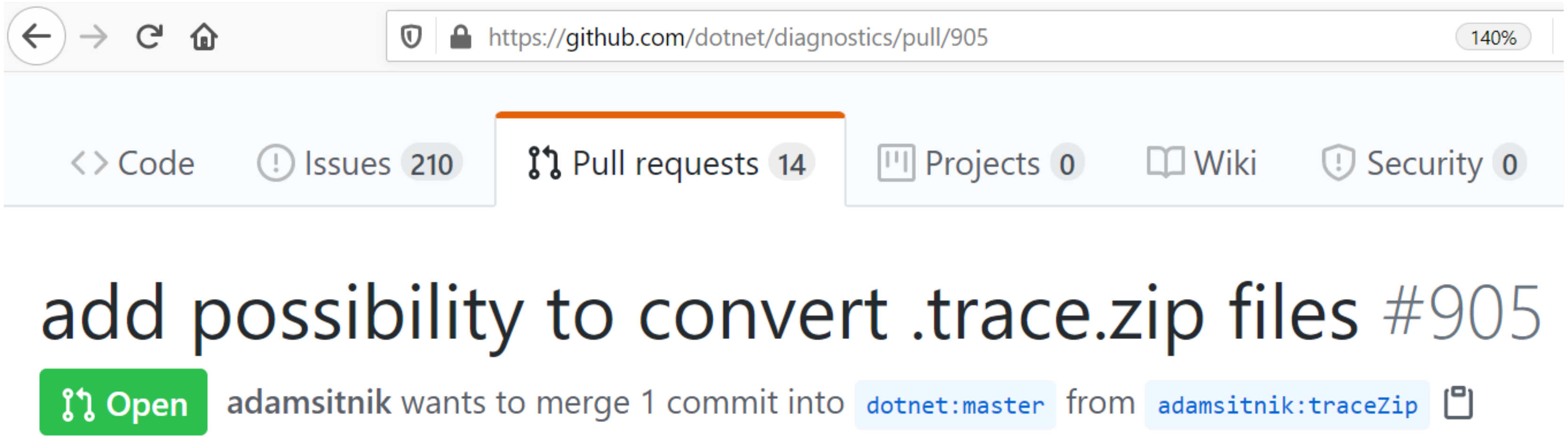
PerfCollect

- Script, located at https://aka.ms/perfcollect
- Has an excellent docs
- **PerfCollect uses perf, which gives you native callstacks. dotnet-trace can only give you managed callstacks**.
- Knows how to install its dependencies
- Has a machine-wide scope
- PerfCollect can be started prior to the process start, whereas dotnet-trace can only be attached to a running process.
- Produces a zip file that can be opened with PerfView

Installation

```
curl -OL https://aka.ms/perfcollect
chmod +x perfcollect
sudo ./perfcollect install
```

We need your input


https://github.com/dotnet/diagnostics/pull/905

add possibility to convert .trace.zip files #905

Open adamsitnik wants to merge 1 commit into `dotnet:master` from `adamsitnik:traceZip`

Vtune

BenchmarkDotNet

- EventPipeProfiler
- Cross platform disassembler

- dotnet run -c Release --filter '*' --job dry
  - --disasm --disasmDepth 5
  - --profiler EP

Rider

When you find the bottleneck

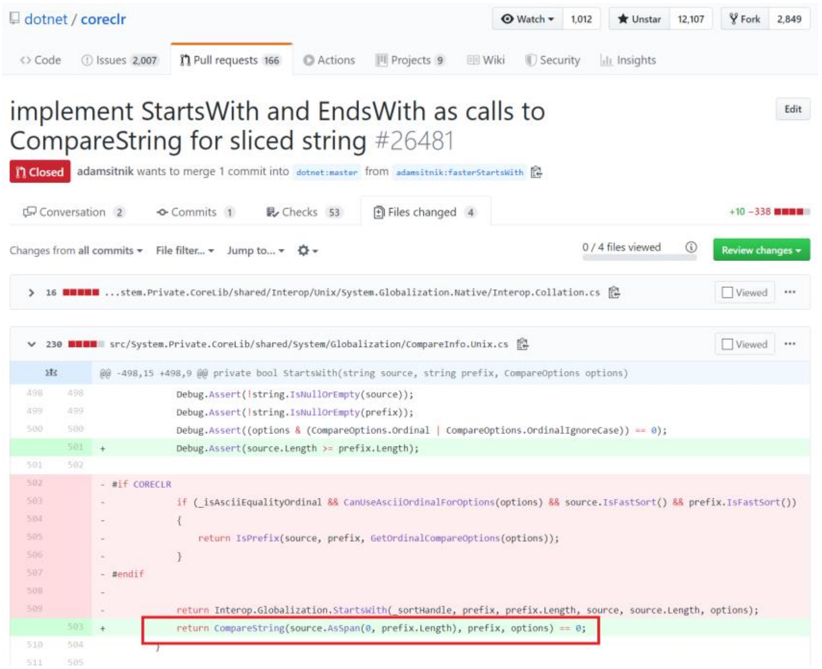
- Try to get the big picture.
- Question the current design:
  - Why do we do that?
  - Can we use it less frequently?
  - Does faster alternative exist?
- Architecture changes require more effort but can boost the perf more than any micro-optimizations.

You have an idea.
What is the next step?

Correctness > Performance

- Make sure the code has good test coverage before you try to tune the perf.
- Ask for a detailed code review:
  - Explain your decisions, share the perf knowledge.
  - Make sure your changes don’t affect the results.
- Don’t push on the reviewers.

Example: string.StartsWith



dotnet / coreclr

Watch 1,012 · Unstar 12,107 · Fork 2,849

Code · Issues 2,007 · Pull requests 166 · Actions · Projects 9 · Wiki · Security · Insights

implement StartsWith and EndsWith as calls to CompareString for sliced string #26481

Closed adamsitnik wants to merge 1 commit into `dotnet:master` from `adamsitnik:fasterStartsWith`

Conversation 2 · Commits 1 · Checks 53 · Files changed 4 · +10 −338

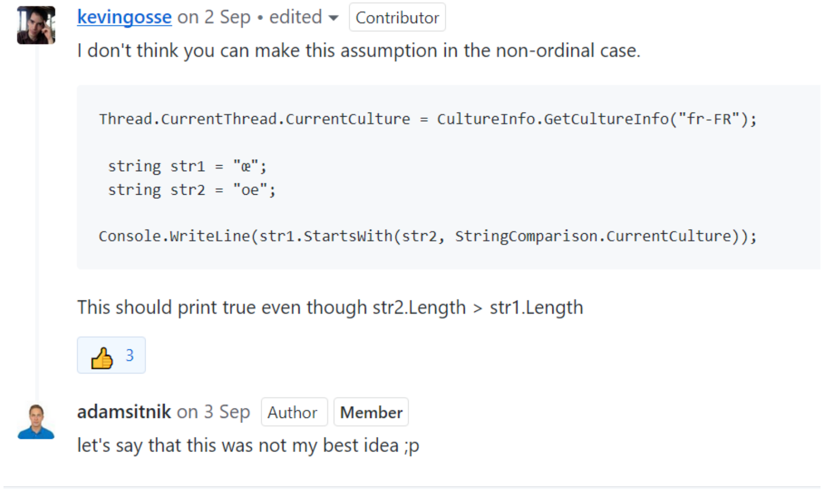
Changes from all commits · File filter... · Jump to... · 0 / 4 files viewed · Review changes

16 ...stem.Private.CoreLib/shared/Interop/Unix/System.Globalization.Native/Interop.Collation.cs

230 src/System.Private.CoreLib/shared/System/Globalization/CompareInfo.Unix.cs

```
@@ -498,15 +498,9 @@ private bool StartsWith(string source, string prefix, CompareOptions options)
              Debug.Assert(!string.IsNullOrEmpty(source));
              Debug.Assert(!string.IsNullOrEmpty(prefix));
              Debug.Assert((options & (CompareOptions.Ordinal | CompareOptions.OrdinalIgnoreCase)) == 0);
+             Debug.Assert(source.Length >= prefix.Length);

- #if CORECLR
-             if (_isAsciiEqualityOrdinal && CanUseAsciiOrdinalForOptions(options) && source.IsFastSort() && prefix.IsFastSort())
-             {
-                 return IsPrefix(source, prefix, GetOrdinalCompareOptions(options));
-             }
- #endif
-
-             return Interop.Globalization.StartsWith(_sortHandle, prefix, prefix.Length, source, source.Length, options);
+             return CompareString(source.AsSpan(0, prefix.Length), prefix, options) == 0;
          }
```



kevingosse on 2 Sep • edited · Contributor

I don't think you can make this assumption in the non-ordinal case.

```
Thread.CurrentThread.CurrentCulture = CultureInfo.GetCultureInfo("fr-FR");

 string str1 = "œ";
 string str2 = "oe";

Console.WriteLine(str1.StartsWith(str2, StringComparison.CurrentCulture));
```

This should print true even though str2.Length > str1.Length

👍 3

adamsitnik on 3 Sep · Author · Member

let's say that this was not my best idea ;p

I <3 Unicode

Culture	Source	Prefix	Windows	Unix	Comment
fr-FR	œ	oe	True	False	
hu-HU	dz	d	False	False	
pl-PL	cz	c	True	False	
	o\u0308	o	False	False	Combining character
	o\u0000\u0308	o	True	True	NULL (0) char
	\uD800\uDC00	\uD800	True	False	Surrogates
	b	new string('a', UInt16.MaxValue + 1)	False	True	18y old bug in ICU

You have an idea and good tests. What is the next step?

Benchmarks!

- Write benchmarks to validate the gains.
- Keep them and measure the perf over time!

while (perf == bad)

- Apply the optimizations
- Run the tests, verify the correctness
- Run the benchmarks, verify the gains
- Profile and analyze the data

Summary

- Have a small repro, try to narrow down the problem.
- Release + debug symbols
- Profilers
  - dotnet trace => default choice
  - PerfCollect – if you really need native call stacks or machine-wide
  - VTune – very powerful profiler, available for free!
  - Rider – an alternative
- Analyze
- Correctness over performance
- Use benchmarks to validate the gains

Questions?

Thank you!

@SitnikAdam

Adam.Sitnik@microsoft.com