Understanding & Profiling C# Async Await Tasks

Microsoft and the .NET community have made asynchronous programming[1] very easy with their implementation of async await in C#. The latest versions of ASP.NET heavily utilize it to improve performance. Many performance monitoring and profiling tools struggle to support and visualize the performance of asynchronous C# code.

Stackify's Prefix & Retrace products both have excellent support for applications using C# async await. In this article we will show how you can use Stackify's free profiler, Prefix[2], to understand the performance of your async .NET code.

How async await Works in C#

To start, I thought it would be useful to show how code that uses async await actually works. When your code gets compiled, the compiler does a lot of magic under the covers that are really interesting.

Let's use this basic usage of the HttpClient as an example.

```
HttpGetRoute"api/HttpClient/GetGWB"public async <bool>GetGWBHttpClientHttpClient
    await hcGetAsync"http://geekswithblogs.net/Default.aspx"return
```

By using ILSpy[3], you can see how the compiler converts this code to use an AsyncStateMachine. The state machine does all the complicated code under the covers that makes it easy for us developers to write asynchronous code. Check out this CodeProject article[4] for an even more detailed example.

```
DebuggerStepThroughAsyncStateMachinetypeofHttpClientController.<GetGWBHttpGetRoute"api/HttpClient/GetGWB"public<bool>GetGWBHttpClientController.<GetGWBGetGWBH
```

Why Profiling Async C# Code is Hard

Profiling async code is complicated because it **jumps across threads**. Traditionally, a method and all of its child method calls all happen on the same thread. That makes it easy to understand the relationship between parent and child methods.

With asynchronous code, that is an entirely different story. A parent method starts on one thread. When an I/O operation starts, code on that thread ends. When the I/O operation completes, the code continues to run on a new thread. Associating code across those threads as part of a larger transaction is complicated.

To give you an idea of the complexity of what happens under the covers, here is a high-level overview of what actually happens for our simple HttpClient example.

```
Note: This is actually even more complex than this. I have simplified it down some for brevity.

Thread #1
Starting IIS pipeline steps & HTTP modules
HttpControllerHandler.ProcessRequestAsyncCore (Web API parent method)
  -> HttpClient.SendAsync (Call the HttpClient method)

Thread #2
HttpWebRequest.BeginGetResponse (Internal class that executes HTTP request)

Thread #3
_IOCompletionCallback.PerformIOCompletionCallback
  -> HttpWebRequest.ProcessResponse (Internal class the handles the response headers of HTTP request)
     -> HttpClient.StartContentBuffering (Start downloading response body stream of HTTP request)

Thread #4
HttpControllerHandler.CopyResponseAsync (Start sending response of Web API action)
```

```
Thread #5
IHttpAsyncHandler.EndProcessRequest (End method of primary handler)

Thread #6
Final IIS pipeline steps & HTTP modules (End of request)

Note: There were dozens of other threads also used as part of downloading segments of the HTTP response body.
```

Based on this simple example, I hope you can appreciate why profiling asynchronous code is so hard.

Visualizing Asynchronous C# Code

One of the key features of Prefix[5] and Retrace[6] both is our excellent code-level trace views. They give developers exactly what they are looking for to understand what their code is doing and how long it takes. Our products have excellent support for C# async await code.

Prefix & Retrace

Prefix perfectly handles this async code and even shows how long it took to receive the response headers versus downloading the entire response stream body. It also shows things like the HTTP status code and length of the response.

Prefix View of HttpClient Request

Application Insights

How do Application Insights handle asynchronous HttpClient calls?

If you were using Application Insights, here is what you would see. They track HTTP calls as "Remote Dependencies." Notice below it says "No calls to remote dependencies were found."

Application Insights View

Application Insights doesn't support HttpClient. Weird huh? This is one of many reasons why developers prefer Retrace[7].

Visualizing Parallel Async Methods

If your code has to do multiple things and then collect the results at the end, you can take advantage of running them in parallel. This is perfect for running multiple SQL queries, web service calls, etc.

For this simple example, we will use the HttpClient again and do 3 separate calls at the same time. When doing this, you don't want to do an await on each method. You want to do an await on the Task.WhenAll at the end.

```
HttpGetRoute"api/HttpClient/TestHttpGetsCrossed"public async <bool>TestHttpGetsCrossedHttpClientHttpClientGetStringAsyncGET_URL//task #1GetByteArrayAsyncGET_L


    await WhenAll//await on all of them!!return
```

Our products perfectly support these sort of scenarios as well. Below you can see what the 3 parallel web requests look like.

Parallel Async Requests

Each one takes about 1 second each. If you did them in a series, the request would take roughly 3 seconds. By doing them in parallel, it only took about 1 second. **Utilizing parallel async tasks can be really powerful**.

OK, how about a really crazy example?

In this crazy example, we are doing a weird mix of operations in parallel and in series.

```
SystemHttpGetSystemActionName"KitchenAsyncParallel"public async HttpResponseMessageKitchenAsyncParallel task1 SERedisAsync task2 AzureServiceBus
        await DBTestAsync

        await task1//Wait for Redis to finish

        await WebClientAsync
        await task2//wait for service bus to finishreturnRequestCreateResponseHttpStatusCodeNewGuidToString
```

No problem! Prefix & Retrace can help you visualize it.

Crazy Async Example from Prefix

Summary

Profiling C# async await code is complicated. Trying to visualize what it is doing can be even more complicated.

All the examples in this article were done via our **free product**, Prefix[8]. It runs on your workstation and helps you understand what your code is doing and how it performs. Our server based product, Retrace[9], provides the same visualization for your production and non-production applications. Retrace starts at just $10 a month.

About Matt Watson

Matt is the Founder & CEO of Stackify. He has been a developer/hacker for over 15 years and loves solving hard problems with code. While working in IT management he realized how much of his time was wasted trying to put out production fires without the right tools. He founded Stackify in 2012 to create an easy to use set of tools for developers.

Links

1. https://stackify.com/asynchronous-programming-easier-think/
2. https://stackify.com/prefix/
3. http://ilspy.net/
4. https://www.codeproject.com/Articles/535635/Async-Await-and-the-Generated-StateMachine
5. https://stackify.com/prefix/
6. https://stackify.com/retrace/
7. http://stackify.com/microsoft-application-insights-alternative/
8. https://stackify.com/prefix/
9. https://stackify.com/retrace/
10. http://gtm.stackify.com/NET-profiler-guide