Concurrency vs. Parallel vs. Async in .NET

#net

Scott Hannen 4 · ١٦ أبريل ٢٠١٩ min read

I read a few posts that made me question whether I understood these concepts or could explain them clearly. There were lots of diagrams, and at least for me, too many words. That's not their problem, it's mine. That's a little bit embarrassing as I've written code that applies all three.

Here's a stab at simplifying it. Feel free to offer corrections, although the intent is to make these concepts easier to understand so that someone can learn about them in more detail, not to include all of those details up front.

While *concurrency*, *parallelism*, and *multithreading* are not the same thing, I think the biggest confusion is mixing those three related concepts with *asynchronous* execution (`async` / `await`).

Another confusion is that in the context of .NET code the words "concurrent" and "parallel" differ from their use elsewhere. That's unfortunate. I've added some clarifications at the end of this post.

Concurrency

Different threads are doing different things at the same time. A simple example is a web application which may start processing one request on one thread and then, if another request comes in while it's still processing the first one, start processing the next one on another thread.

Parallelism

I need perform 100 of some task. If I divide up that work between multiple threads that work simultaneously, I'll finish faster.

That's this:

```
    Parallel.ForEach(things, thing => thing.Process());
}
```

Async

This gets mixed up with the other two, likely because it has something to do with threads. But it's entirely different.

Async describes how *individual* threads are used.

- If I execute a stored procedure using `SqlCommand.ExecuteNonQuery` and the procedure runs for ten seconds, the thread used to call that method will do nothing for ten seconds except wait.
- If I execute it using `await SqlCommand.ExecuteNonQueryAsync` in an `async` method, that thread can do other things for ten seconds. When the command is finished executing, that thread (or another one - we don't really care) will pick up where the first thread left off.

These are not comprehensive definitions or detailed technical descriptions. In my experience, sometimes we need simpler explanations before trying to process the big articles with lots of diagrams. Or maybe it's just me that needs that.

Perhaps when this is polished up I can add a second post with some of the details with which I didn't want to clutter this one.

How "Concurrency" and "Parallelism" differ when speaking of .NET

I really wanted to make this post short and simple. It wasn't meant to be.

Concurrency

Concurrency means doing multiple things at one time but does not specifically refer to the use of multiple threads. JavaScript uses an event loop to implement concurrency using a single thread.

In the context of .NET applications, concurrency is almost always associated with execution on simultaneous threads. That's probably not intentional.

> Represents a thread-safe first in-first out (FIFO) collection.

If our concurrency was not achieved using multiple threads we would not need a thread-safe collection. We would just use a `Queue`.

Use of the word "concurrent" in the namespace and classes is accurate - the word means "simultaneous, at the same time." But the result is that when working with .NET concurrency and multithreading have become intertwined.

If "concurrency" means multithreading then it's not related to `async/await`. If it means doing multiple things at once then `async/await` supports concurrency by allowing a single thread to start one process and then do something else instead of waiting for the first process to finish.

Parallelism

Parallelism broadly means achieving concurrency by distributing work across multiple CPUs. So in .NET discussions when we talk about concurrency we mean parallelism.

In .NET world when we talk about parallelism we're often referring to a subset, a particular application of parallelism. It's when we have a very specific set of computations to perform and we distribute it across multiple threads.

Why? Look at Microsoft's Task Parallel Library and you'll see that most links referring to "parallel programming" lead to `Parallel.ForEach` or other implementations which start with a set of tasks and distribute them.

This is not to imply that all .NET developers are confused about these concepts. It's just that in documentation or StackOverflow discussions we tend to use the terms differently.

Consider this paragraph from the description of a book entitled *Concurrency In .NET*, emphasis mine:

> Unlock the incredible performance built into your multi-processor machines. *Concurrent applications run faster because they spread work across processor cores*, performing several tasks at the same time. Modern tools and techniques on the .NET platform, including parallel LINQ, functional programming, asynchronous programming, and the Task Parallel Library, offer powerful alternatives to *traditional thread-based concurrency*.

Okay, now I know it's not just me. Concurrency is equated with threads executing on

Parallel Library is called an alternative to thread-based concurrency when it's explicitly a

way to implement concurrency using multiple threads. I think I know what he means, but we've blurred some meanings.

Even referring to ".NET" is a bit vague at this point because we used to equate it with software running on multi-CPU Windows computers, but now it runs on all sorts of things.

Now my clarification is longer than the original post and involves five definitions of three terms. I'm really sorry.

Discussion

Subscribe

DEV

Add to the discussion

rhymes • Apr 16 '19

I think your definition of concurrency is actually parallelism. If the system is doing two things *at the same time*, then it's doing them in parallel.

The subtle difference is that concurrency means that the system is able to advance multiple tasks indipendently, parallelism is that it's able to advance them at the same exact time.

Love this definition by Rob Pike's famous Concurrency is not parallelism slides:

> Concurrency is about dealing with lots of things at once.
> Parallelism is about doing lots of things at once.

You can see concurrency as a round table with 5 people sitting on it and a waiter. The waiter is able to mind each person's request indipendently (hence advancing the tasks) in a few minutes but he's not able to talk to two persons at the same time. To achieve that you need at least one more waiter, so that one talks to a person and another talks to another person at the same time.

Async is a programming model. It can be implemented without threads, I believe .NET implements with threading, Node.js for example uses a single thread with an event loop to achieve async.

This is always a tricky topic because we tend to conflate concurrency models with the definition of concurrency and parallelism

 10  0  11

6

Scott Hannen • Apr 16 '19

What I'm beginning to realize is that, as if this wasn't confusing enough, the common use of these terms in the scope of .NET programming is not quite the same as their common meaning. That's unfortunate because using the same terms to describe different things undermines the reason why we have terms.

Specifically, most scenarios involving concurrency in a .NET application involve multiple threads. In the broader sense concurrency does not require multiple threads, but the purpose of all the classes in the `System.Collections.Concurrent` namespace (like `ConcurrentQueue`) is to support multiple concurrent threads.

When a website handles multiple requests and uses multiple threads, that is both concurrent and parallel. The use of multiple threads really fits the definition of "parallel," but if you google ".net parallel web requests" the results all describe distributing the work of *making* requests, not receiving them.

I qualified the post as being .NET-specific, but this means I have to qualify it even more.

1

Scott Hannen • Apr 16 '19

I think where this gets mixed up is that concurrency involves *any* tasks being executed at the same time. That's the literal meaning of concurrency.

Parallelism involves taking a *specific* set of tasks and executing them across multiple threads rather than sequentially.

That lines up with the sentence just before the one you quoted:

> In programming, concurrency is the composition of independently executing processes, while parallelism is the simultaneous execution of (possibly related) computations.

In other words, concurrency means that multiple things are going on the same time (the literal meaning of "concurrent.") One user is submitting an order. Another submits an order a second later while the previous one is still processing. Another is updating profile information. All are happening at the same time.

Parallelism relates to breaking up a task into separate parts and executing them

10 0 11

I can see where there's room for confusion. Both concurrency and parallelism result in things happening at the same time on multiple threads. We could say that concurrency is

incidental. We can process two requests at once, but if we get them one at a time then we only process them one at a time.

Parallelism is deliberate. I've got to do a bunch of stuff so I'm going to distribute the load across multiple threads.

That's why I separated async from the other two. While the terms have distinct meanings, in practice there's some overlap. But async is not about multiple threads.

2

ChrisP • Apr 9

Let me summary them in a story, please correct me if I am wrong at any points.

- Bob started a restaurant and he does all the thing: Being a chef, being a waiter and cashier. this kind of system is **non concurrency**
- More and more customers come. Bob decided to hire 1 Chef, 1 Waiter and 1 Cashier. He just enjoys to do the management things. Now, at the same time, Waiter gets order while Cashier collects payment from another customer: Now we have a **concurrency* * system
- The number of customer continues being increase. 1 waiter is not enough. Bob decided to get another waiter. Waiter 1 gets order from customer table number 1 to 10, waiter 2 gets order from table number 11 to 20. Getting order is now divided between 2 waiter. Now we have **Parallel* * system.
- Waiter 1 after getting the order, he bring the order to the kitchen and wait for the food. He just standing there and keep waiting for the Chef. After the food is ready, he bring the food to his customer and getting next order. Bob doesn't want to do this way, he ask waiter 1 stop waiting, After the food is ready, the chef will inform and either waiter 1 or waiter 2 can bring it to the customer. Now we have **Async* *

2

SAYED UZ ZAMAN • Nov 18

Thanks for amazing explanation, the comments are even better.
Developer always tends to make confusion between multi threading and parralel task

10 0 11

You will complete half of para-1, then switch to para-2 and do half of para-2 and bla bla..
But in parralel fashion,

You will use your both hand two write both paragraphs at the same time. So these two paragraphs should not be related/depends on each other.
And, I am not sure, probably if you wanted to write a paragraph as para-1, and paragraph summery as para-2 then you have to ensure para-1 finishes first so that based on this you can do the next task. This should be async fashion, isn't it?

2

Code of Conduct • Report abuse

Scott Hannen

I've been developing software full time since 2003, beginning with languages I'm still embarrassed to mention.

Follow

WORK
Senior Software Engineer

JOINED
١١ أبريل ٢٠١٨

More from Scott Hannen

The Interface Segregation Principle Applied in C#/.NET

#interfacesegregation #net #solid #oop

String Interpolation Functions vs. string.Format Constants

#net #stringinterpolation #stringformat #constants

10 0 11