

Programação Assíncrona e Paralelismo no WebAPI

Johnathan Fercher

January 31, 2018

1. Introdução
2. Programação Síncrona e Assíncrona em C#
3. Paralelismo em C#
4. Programação Assíncrona e Paralela em C#
5. Conclusões

Introdução

- Apresentar as principais formas de aprimorar a performance de execução de requisições no WebAPI;

- **Programação Assíncrona** tem o propósito de possibilitar a execução de tarefas demoradas sem o bloqueio da execução;
- **Programação Paralela** tem o propósito de possibilitar a execução de tarefas ao mesmo tempo;

Problema assíncrono

- Consumir uma API;
- Operações de CRUD em banco de dados;
- Quaisquer outras operações que demorem;

Problema paralelo

- **Em um jogo:** Uma thread é responsável por obter os comandos do Joystick e outras N são responsáveis por controlar a renderização de objetos, comandos de adversários e etc;
- **Em um algoritmo de reconhecimento facial:** Pode-se dividir uma imagem em quadrantes, onde N threads serão responsáveis por aplicar filtros nas secções;

Programação Síncrona e Assíncrona em C#

Sintaxe de um código síncrono

```
41     public void Execute(){  
42         var value = Method();  
43     }
```

Sintaxe de um código assíncrono

```
41     public async Task<T> Execute(){  
42         T value = await MethodAsync();  
43         return value;  
44     }
```

Transformando código síncron em assíncrono

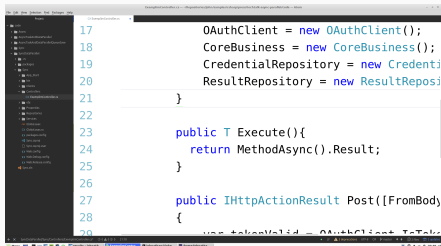
```
41     public async Task<T> MethodAsync(){  
42         return await Task.Run(() => {  
43             return Method();  
44         });  
45     }
```

Transformando código assíncrono em síncrono

```
41     public T Method(){  
42         return MethodAsync().Wait();  
43     }
```

Nota Importante: Somente utilize `Wait()` se o escopo que utiliza o código assíncrono **PRECISAR** ser síncrono.

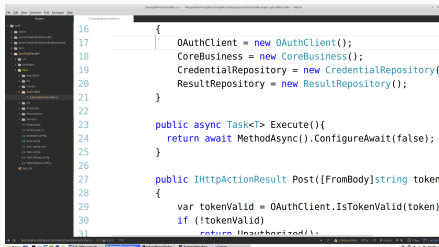
Transformando código assíncrono em síncrono



```
17 OAuthClient = new OAuthClient();
18 CoreBusiness = new CoreBusiness();
19 CredentialRepository = new CredentialRepository();
20 ResultRepository = new ResultRepository();
21 }
22
23 public T Execute(){
24     return MethodAsync().Result;
25 }
26
27 public IHttpActionResult Post([FromBody]
28 {
29     // use tokenValid = OAuthClient.TokenValid;
```

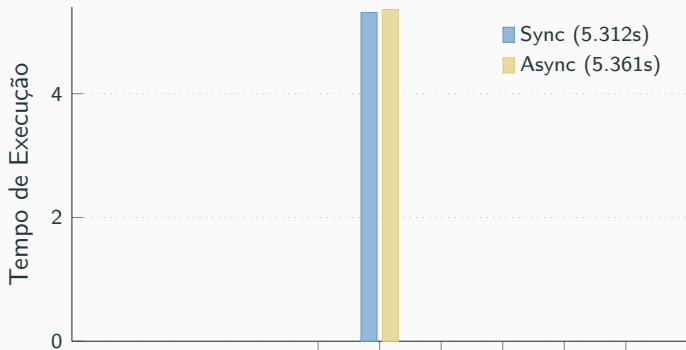
Nota Importante: Jamais utilize Result no WebAPI.

Transformando código assíncrono em síncrono



Sync x Async

Benchmark



- Sync: Síncrono;
- Async: Assíncrono;

Fui tapeado?

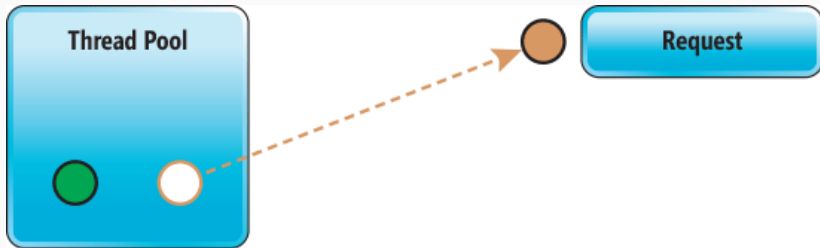
Por que não houve ganhos de performance?

Programação Assíncrona apenas é responsável por não bloquear a execução.

Então para que vou usar isso?

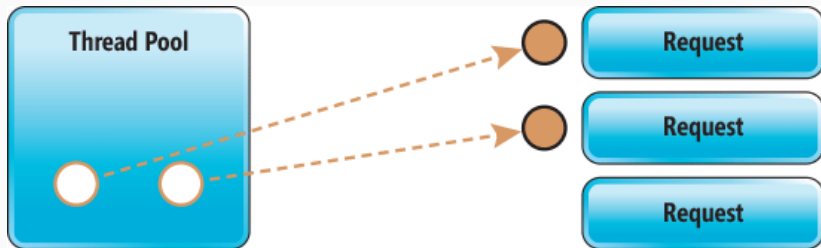
Depende da aplicação.

Uma requisição síncrona no WebAPI



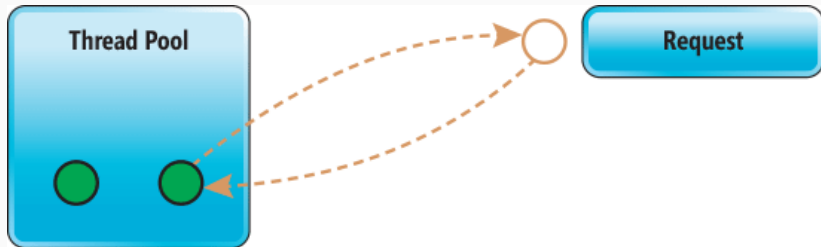
- Uma thread livre e outra thread bloqueada realizando **nada**;

Três requisições síncronas no WebAPI



- Duas threads bloqueadas realizando **nada** e uma requisição em espera;

Uma requisição assíncrona no WebAPI



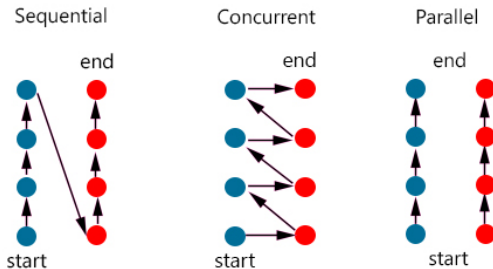
- Thread realiza uma requisição bloqueante e retorna para a espera;

- Redução de custos em relação à memória;
- Redução do tempo de CPU desperdiçado;

Paralelismo em C#

Paralelismo X Concorrência

- **Paralelismo:** Duas tarefas são executadas simultaneamente em diferentes processadores ou núcleos;
- **Concorrência:** Duas tarefas estão em progresso ao mesmo tempo, de forma alternada em um processador ou núcleo;

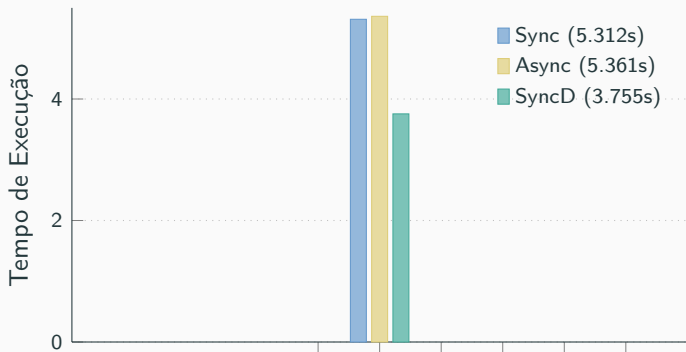


- **Em um processador de 4 núcleos:**
 - Executando 4 threads. Pode-se alocar 1 thread por núcleo;
 - Executadas 5 threads. Pelo menos 2 threads vão ser alternadas em um único núcleo;
- **Vantagens:**
 - Execução Simultânea;
 - Execução Simultânea e aumento da frequência com que as tarefas são alocadas na CPU em um SO de tempo compartilhado;

- **Paralelismo de Dados:** A mesma instrução é executada em diferentes threads.
 - Ex: Enquanto uma thread itera sobre a parte inicial de uma lista, outra itera sobre a parte final;
- **Paralelismo de Tarefas:** Diferentes instruções são executadas em várias threads;
 - Ex: Enquanto uma thread valida um token, outra busca credenciais no banco;

SyncDataParallel

Benchmark



- Sync: Síncrono;
- Async: Assíncrono;
- SyncD: Síncrono com Paralelismo de Dados;

Parallel em um método Assíncrono:

C# não consegue realizar paralelismo com *Parallel* dentro de métodos assíncronos :(

Posso aplicar isso em qualquer problema?

Depende da quantidade de transformação de dados envolvida.

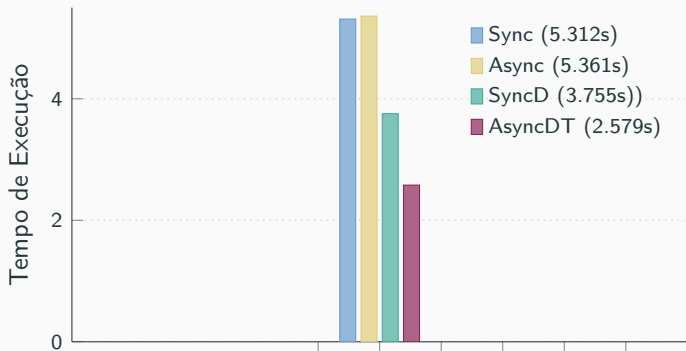
Quanto maior o número de threads melhor?

Existe um limiar onde a aplicação gasta mais tempo na comunicação entre threads do que o ganho de performance na execução paralela.

Programação Assíncrona e Paralela em C#

AsyncTaskAndDataParallel

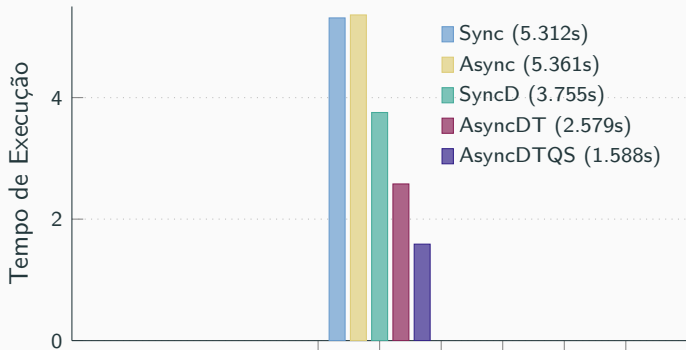
Benchmark



- Sync: Síncrono;
- Async: Assíncrono;
- SyncD: Síncrono com Paralelismo de Dados;
- AsyncDT: Assíncrono com Paralelismo de Dados e de Tarefas;

AsyncTaskAndDataParallelQueueSave

Benchmark



- Sync: Síncrono;
- Async: Assíncrono;
- SyncD: Síncrono com Paralelismo de Dados;
- AsyncDT: Assíncrono com Paralelismo de Dados e Tarefas;
- AsyncDTQS: Assíncrono com Paralelismo de Dados e Tarefas com Fila de Execução;

QueueWork != Async sem Await:

A second operation started on this context before a previous asynchronous operation completed. Use 'await' to ensure that any asynchronous operations have completed before calling another method on this context. Any instance members are not guaranteed to be thread safe.

Conclusões

- É possível criar métodos assíncronos e paralelos no WebAPI de forma simples e conveniente;
- Métodos assíncronos devem ser utilizados em APIs com finalidade de otimizar a utilização de recursos;
- Paralelismo deve ser utilizado em problemas onde a quantidade de transformações de dados é grande;

Perguntas?