Avaliando ASP.NET Web API no âmbito de Alto Desempenho

Lucas Pfeiffer Salomão Dias[1], **Thiago Roberto Lima Lopes**[1], **Rodrigo da Rosa Righi**[1]

[1]Bacharelado em Ciência da Computação
Universidade do Vale do Rio dos Sinos (UNISINOS) – São Leopoldo – RS – Brazil

lucaspfsd@gmail.com, lolthiago@gmail.com, rrrighi@unisinos.br

Resumo. *Este artigo menciona a importância das APIs no cenário tecnológico atual, e como algumas técnicas de otimização são fundamentais para criação de API de alto desempenho. São descritos os passos adotados e frameworks utilizados para criar um cenário de teste prático e faz comparações dos beneficios da aplicação destas técnicas numa ASP.NET Web API.*

1. Introdução

No cenário tecnológico atual há uma grande convergência para o conceito de internet das coisas, onde todos os tipos de dispositivos estão conectados e consumindo informações, o que por consequência aumenta de forma escalar a demanda sobre as Web APIs. Este artigo surge para demonstrar as principais vantagens de empregar algumas técnicas de otimização, como as citadas por [Zec 2014], afim de reduzir o tempo de resposta, melhorando a experiência de uso e consumo de banda necessário para a comunicação dos dispositivos, podendo também impactar numa economia financeira, haja vista que muitos dispositivos valem-se de redes móveis tarifadas por pacotes baseados na quantidade de dados trafegados.

2. Estrutura da ASP.NET Web API avaliada

Para que se possa comprovar a eficácia da otimização ou não dos métodos, criou-se uma ASP.NET Web API, daqui para frente citada simplesmente como API, na qual encontram-se 3 métodos com as seguintes caracteristicas:

1. **/api/categories**: Chamada RESTful[1] que retorna recursos sem nenhuma otimização;
2. **/api/categoriespad**: Chamada RESTful que retorna recursos combinando *caching* servidor e cliente, compressão gzip[2];
3. **/api2/categoriespadnonrestful/getwithgzipjiljsonserializercachetotal**: Chamada REST (não-RESTful) que retorna recursos combinando *caching* servidor e cliente, compressão gzip;

Para alguns recursos utilizou-se *frameworks* de terceiros, que possuem grande aceitação da comunidade .NET. Os *frameworks* padrão de aplicações ASP.NET utilizados foram o **.NET Framework 4.5**, o **MVC 4** e o **.NET Web API 2**. Para utilizar os

[1]RESTful é um padrão para construção de Web APIs onde uma das premissas é a implementação de todos os Verbos do protocolo HTTP.

[2]gzip é a abreviação de GNU zip (descrito na RFC 1952). Este é o método mais usado para compressão sem perda de dados.

benefícios de *caching* na API, utilizou-se o *framework* **Strathweb CacheOutput Web API 2**, para serializar manualmente as mensagens retornadas pela API foi utilizado os *frameworks* **Json.NET 6.0.6** e **Jil 2.4.0**, e na compressão das mensagens o *framework* **DotNetZip 1.9.3**.

O Servidor utilizado para hospedar a API está localizado em São Paulo/BR, com as seguintes configurações:

- **Sistema Operacional:** Windows Server 2012 64 bits;
- **Serviço Web:** IIS 8
- **CPU:** Intel Xeon E5645 2.4 Ghz;
- **Memória:** 4 GB;
- **Placa de Rede:** Ethernet.

3. Metodologia de Avaliação

Para efetuar os testes foi usada a ferramenta Load Impact [LoadImpact 2014], a qual fornece uma simulação realista de usuários através da criação de usuários virtuais (VU)[3], sendo possível definir a localização, a quantidade de usuários simultâneos e o navegador usado na simulação. O site da ferramenta Load Impact destaca os seguintes recursos: Uso de múltiplas conexões concorrentes quando carregando recurso da mesma URL; A capacidade de emular os navegadores: IE, Firefox, Chrome, Safari e Opera; Uso de DNS como usuários reais (cache de acordo com TTL, por exemplo); Extensivo suporte ao HTTP incluindo comportamentos de redirecionamento, *custom headers*, todos os verbos do HTTP, *cookies* e compressão. E por fim a facilidade de um *software* como serviço, ou seja, não há necessidade de instalação.

Os testes se deram através da metodologia chamada Teste de Carga, aplicada para cada chamada disponível na API, a qual sempre retorna exatos 9280 itens[4]. Os testes foram executados no ambiente padrão disponibilizado através da ferramenta Load Impact, possuindo até 50 usuários virtuais e com duração de 5 minutos. Devido aos testes serem realizados na versão gratuita, o ambiente disponibilizado segue as seguintes configurações: as requisições são feitas apenas da localização de Ashburn/US, o navegador sendo uma versão proprietária da Load Impact e quantidade de tráfego na rede ilimitada, más com latência influenciada pela distância entre São Paulo e Ashburn.

4. Resultados

Dados os três métodos disponíveis na API, sendo dois otimizados para um melhor desempenho, os testes chamando o método não otimizado, Figura 1, demonstram que o tempo de carregamento (linha azul) fica em média 15s, tendo seu pior tempo em 35s aproximadamente. A largura de banda (linha laranja) deste mesmo teste ficou entre 2Mb a 3Mb, observando-se que o tempo de carregamento é afetado diretamente pela quantidade de usuários virtuais simultâneos (linha verde), e a largura de banda é afetada pela quantidade de requisições por segundo (linha amarela). Nos testes realizados chamando os métodos otimizados, Figura 2, nota-se que o tempo de carregamento ficou entre 1s e

[3]VU (Virtual User) é um ambiente de simulações de usuários que torna possível a simulação do tráfego exatamente como ele é na vida real.

[4]Os itens são objetos simples que representam categorias compostas por ID (inteiro), Nome (*string*) e TipoMidia (inteiro). O número de 9280 itens foi escolhido para que os resultados fossem mais expressivos.

1,5s. Constata-se então um ganho devido ao *caching* tanto do lado servidor, quanto do lado cliente, que segundo [Hossam Hassanein 2002] é uma técnica popular para obter-se desempenho. Conforme citam [Kevin J. Ma 2006] e [Xiaohui Chen 2009] compressão de mensagens HTTP é o maior responsável pela diminuição de uso de banda no retorno de mensagens, as quais nos testes se obteve com tamanhos entre 500Kb e 750Kb após a compressão com gzip.

Note também que diferentemente das chamadas aos métodos não otimizados, as chamadas para os métodos otimizados não tiveram o tempo de carregamento prejudicado conforme a quantidade de usuários virtuais aumentava, isso se dá grande parte devido a utilização de cache, tanto do lado cliente quanto do lado servidor. Outro ponto a ser destacado é a utilização do padrão RESTful em comparação com o padrão REST, o qual não obteve-se ganho significativo na utilização entre um ou outro.

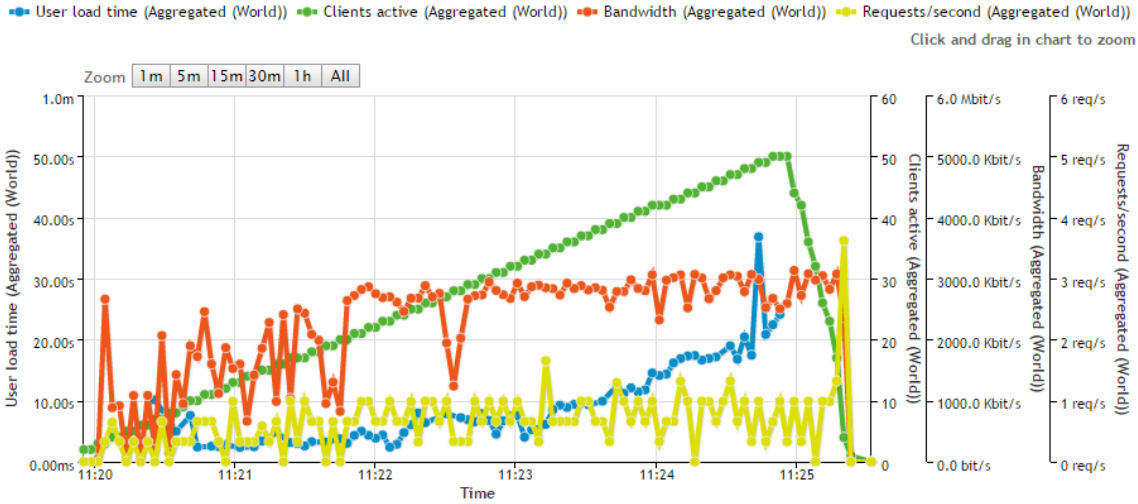

Figure 1. Resultados da API sem otimização no padrão RESTful.

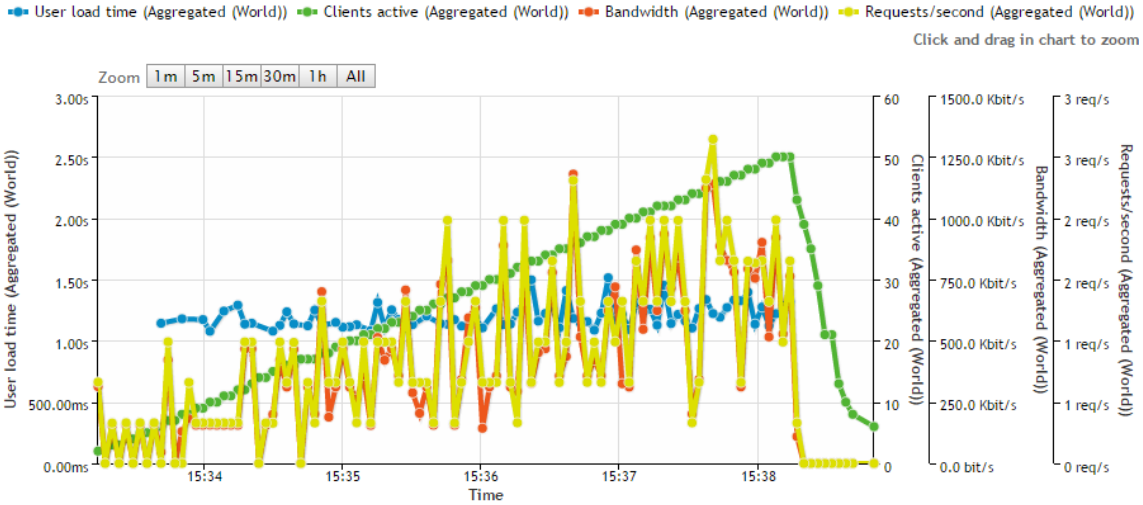

Figure 2. Resultados da API otimizada no padrão RESTful.

5. Conclusão

Este artigo, através da elaboração de um ambiente de teste, apresentou resultados iniciais sobre *caching* e compressão de mensagens HTTP que demonstraram um ganho médio de dez vezes no tempo de resposta e quatro vezes na redução do tamanho da mensagem para o cenário proposto. Estes quesitos de otimização se mostram então efetivos e de grande importância na elaboração das APIs no cenário evolutivo tecnológico atual.

References

Hossam Hassanein, Zhengang Liang, P. M. (2002). Performance comparison of alternative web caching techniques. *Proceedings of the Seventh International Symposium on Computers and Communications (ISCC'02)*, pages 213–218.

Kevin J. Ma, R. B. (2006). Analysis of transport optimization techniques. *IEEE International Conference on Web Services (ICWS'06)*, pages 611–620.

LoadImpact (2014). Simulate real user behavior, http://loadimpact.com/features.

Microsoft (2015). Asp.net web api, http://www.asp.net/web-api/.

Xiaohui Chen, Weidong Wang, G. W. (2009). Impact of http compression on web response time in asymmetrical wireless network. *2009 International Conference on Networks Security, Wireless Communications and Trusted Computing*, pages 679–682.

Zec, R. (2014). 8 ways to improve asp.net web api performance, http://blog.developers.ba/8-ways-improve-asp-net-web-api-performance/.