

Melhoria de Desempenho de Sistemas Legados Críticos

Eduardo Teixeira¹, Mateus Manuel¹, Felipe Evangelista¹,
Renan Costa¹, Tiago Medina¹, Everton Agilar¹, Jacir Bordim¹

¹ Centro de Informática – CPD
Universidade de Brasília (UnB) – Brasília, DF – Brazil

{eduhenr, mateusmanuel, felipesantos}@unb.br

{renanfc, tiagomedina, evertonaguilar, bordim}@unb.br

Resumo. *Nesse trabalho é apresentada uma mudança arquitetural realizada no sistema legado de matrículas da Universidade de Brasília visando a melhoria de desempenho. Também é apresentado um gerador de tráfego (TRAFICGEN) construído para que fosse possível submeter o sistema a uma carga semelhante à que ocorre durante os períodos de matrícula e dessa forma fossem realizadas diversas melhorias. Foi verificado que o sistema modificado alcançou uma melhoria de desempenho de mais de 5× com relação à versão de produção com a metade dos servidores utilizados e que ainda suporta um aumento de carga de pelo menos 215% sem degradação do desempenho.*

1. Introdução

Ao longo dos últimos, em particular após o REUNI [Brasil 2007], a Universidade de Brasília (UnB) aumentou consideravelmente o quantitativo de cursos de graduação e, consequentemente, o número de alunos. Por outro lado, os sistemas apoio, como o MatrículaWeb (MW), não acompanharam este ritmo de evolução e hoje apresentam deficiências no que tange ao desempenho durante a execução de atividades críticas como o processo de matrícula.

Neste processo, em uma das etapas finais, os alunos concorrem pelas vagas existentes obedecendo o critério *First Come First Served* (FCFS) [Tanenbaum and Wetherall 2010], tendo em vista haver um número maior de alunos do que de vagas disponíveis em alguns cursos. Este critério, por sua vez faz com que haja um contingente de demanda ao servidor que supera 5.000 acessos concorrentes, o que por sua vez acarreta em problemas de indisponibilidade e lentidão no atendimento destas demandas. Este trabalho apresenta em detalhes as ações e técnicas utilizadas com o intuito de melhorar o desempenho deste sistema legado e permitir não apenas a sobrevivência do sistema mas principalmente atender com maior celeridade as demandas em momentos de pico como o que ocorre durante o período de matrícula.

Foi realizado o estudo da arquitetura de produção do MW, e foram identificados alguns pontos críticos. Um deles está relacionado ao tratamento da fila de requisições, que após o *round robin* entre os servidores Web (WebServers), é realizado por meio de um único servidor, o ServerApp, conforme mostrado na Figura 1. O ServerApp é um servidor de aplicação que contém o WebService *ConnectionRemote* (ConRem) que se comunica com os WebServers. O ConRem possui diversos componentes ActiveX que realizam o acesso ao banco de dados por meio de *stored procedures* que contém as regras

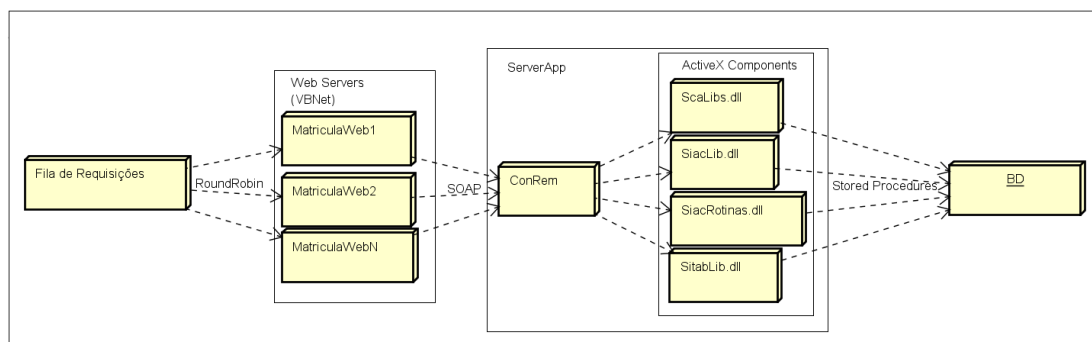


Figura 1. Arquitetura de Produção do MW.

de negócio da aplicação. Dessa forma, para que as operações sejam realizadas no servidor de banco de dados é necessário enviar as requisições ao ConRem por meio do protocolo SOAP, que por sua vez acessa os componente ActiveX que finalmente realizam chamadas para as *stored procedures*. Ou seja, existem limitações na escalabilidade da aplicação devido ao processamento de todas as requisições serem realizadas por um único servidor e também devido ao alto grau de *overhead* para que a aplicação persista os dados em banco. Além disso, caso ocorra uma queda ou um alto volume de requisições no ServerApp ocorre o comprometimento da qualidade de serviço e até mesmo do funcionamento para os usuários que estejam utilizando o sistema.

Um outro ponto crítico encontrado nessa arquitetura se deve ao fato de que todas as regras de negócio do sistema se encontram em *stored procedures* escritas na linguagem *Transact-SQL* (T-SQL). Essas *stored procedures* são chamadas em cadeia e utilizam tabelas temporárias para a comunicação entre elas. O acesso concorrente de muitas chamadas a *stored procedures* gera um alto consumo de CPU no ServerApp e conseqüentemente lentidão no processamento da fila de requisições. Além disso, o uso de *stored procedures* para regras de negócio dificulta o entendimento e a depuração do código, pois as ferramentas de *debugging* para a linguagem T-SQL são muito limitadas, podendo produzir resultados diferentes dos ambientes próprios de linguagens de programação. Devido a esse modo de funcionamento, é frequente o estouro de vagas em ofertas de disciplinas sendo necessária a intervenção manual dos coordenadores dos departamentos para realizar os ajustes necessários para cada disciplina ofertada.

2. Métodos

Foi proposta uma arquitetura alternativa ao modelo atual mostrado na Figura 1. Nessa arquitetura foi retirada da aplicação a comunicação com o Webservice ConRem, por ser identificado arquiteturalmente como um ponto único de gargalo e de falha. Por meio dessa retirada existe uma maior abertura para o paralelismo nas execuções das consultas ao banco de dados que podem ser realizadas diretamente pela aplicação. Além disso, caso ocorra a falha de um WebServer não ocorre o comprometimento de todo o sistema, afetando apenas os usuários conectados no IIS em que ocorreu a queda [Coulouris et al. 2011].

Além da retirada do ConRem foi realizada a conversão do projeto e de suas dependências para o compilador Visual Studio 2015 com o objetivo de melhorar o desem-

penho do binário executável gerado e da sua utilização com a versão mais recente do IIS (Internet Information Server 8.5). O IIS 8.5 realiza o gerenciamento de múltiplas conexões TCP de forma mais otimizada e pode ser utilizado com o sistema operacional Windows 2012 R2, que também possui um melhor desempenho para gerenciamento de recursos de rede e de CPU.

Também foi realizada a conversão do código de T-SQL para VB.NET por ser a mesma linguagem utilizada no *site* e com o objetivo de melhorar o entendimento das regras de negócio e da depuração do código, estudar potenciais pontos de melhoria e efetuar manutenções evolutivas e corretivas mais rapidamente. Entretanto, o uso de tabelas temporárias foi mantido pois a sua conversão implicaria em um alto risco devido a grande quantidade de regras de negócio envolvidas. Dessa forma, todas as funcionalidades do MW foram testadas por meio da depuração das *stored procedures* já existentes e do código convertido para VB.NET, porém ainda com o uso de tabelas temporárias, levando em conta a criticidade envolvida. Foram realizados testes com o uso de *stored procedures* e código VB.NET utilizando as mesmas variáveis de entrada e os resultados foram comparados para validar o funcionamento correto de cada funcionalidade.

Foi realizada a modificação de todas as consultas em banco (*select*) utilizando a cláusula *with(nolock)* para diminuição das serializações em função da grande quantidade de consultas realizadas pelo processo de matrícula. Também foi utilizada a estratégia de *locks* de linha *with(updlock)* transacional para atualização das tabelas de oferta e reserva de vagas em disciplinas para evitar o estouro de vagas.

3. Resultados

A arquitetura proposta foi implantada em produção no segundo período de 2016 (2016/2). Foram utilizados 12 WebServers com Windows Server 2012 R2, 16 *gigabytes* de memória e 16 núcleos cada. Devido a conversão de código existente de T-SQL para VB.NET ocorreram 2 *bugs* no período de matrícula, por conta do comportamento específico da linguagem T-SQL ao tratar variáveis não inicializadas. Entretanto, esses problemas foram corrigidos rapidamente e as atualizações em ambiente de produção foram efetuadas no mesmo dia da requisição, o que comprovou o alcance da melhoria almejada com relação ao melhor entendimento e correções rápidas. Além disso, também verificou-se que não ocorreram mais os estouros de vagas, pois a crítica da matrícula para a aquisição de vagas é realizada atômicamente.

Na Figura 2a o eixo *x* corresponde à hora do dia a partir da liberação de funcionamento do sistema que ocorre às 00:00hs e o eixo *y* corresponde ao número de matrículas efetuadas. A melhoria de desempenho do sistema com a "Arquitetura Proposta" foi $2.59 \times$ superior nas primeiras 2 horas. Entretanto, ocorreram problemas de *timeout* de banco na primeira hora, o que gerou lentidão e erros de acesso. Isso foi constatado pelo comportamento linear da "Arquitetura Proposta" nesse período de 1 hora com relação à quantidade de matrículas efetuadas no tempo.

Devido a esse problema de desempenho foi desenvolvido um programa escrito na linguagem C++ para gerar tráfego com um alto volume de requisições concorrentes para o sistema. Esse gerador de tráfego (TRAFICGEN) realiza a carga para a memória de todas as matrículas do período 2016/2 e realiza os POSTs HTTP utilizando várias conexões TCP concorrentes em várias *threads* e processos que se comunicam por meio de *Message Pas-*

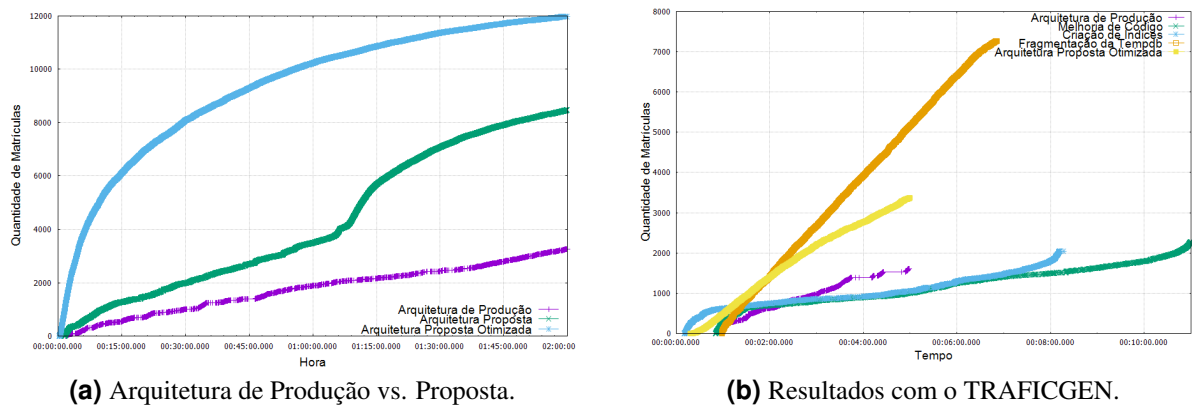


Figura 2. Comportamento da Quantidade de Matrículas no Tempo.

sing Interface (MPI) [Cameron Hughes 2003]. Os *buffers* dos POSTs foram capturados com o *sniffer Wireshark* durante a operação das funcionalidades *login*, obtenção de dados pessoais, remoção e adição de disciplinas em um *browser*. Dessa forma, esses *buffers* podem ser enviados mudando apenas as informações relativas ao número de matrícula, disciplina e turma. Esses testes tem por objetivo identificar possíveis gargalos por meio da análise do tempo médio, mínimo, máximo, desvio padrão e número de matrículas dessas funcionalidades submetendo 1440 conexões concorrentes para o *pool* de WebServers. Foi utilizado o valor 1440 pois este é o valor aproximado do pico de conexões durante o intervalo de 1 hora em que ocorreu a contenção da "Arquitetura Proposta" na Figura 2a. O envio de requisições é realizado em rajada durante 5 minutos, sendo encerrada a geração de tráfego após esse intervalo, mas é esperado o resultado das requisições enfileiradas até o término de todas as *threads* e processos iniciados pelo TRAFICGEN.

Com o uso do TRAFICGEN foi possível encontrar diversos problemas de desempenho e realizar vários ajustes para resolvê-los como a melhoria do código-fonte, a criação de novos índices e a fragmentação do banco de dados responsável pelo tratamento de tabelas temporárias. Entre as melhorias de código-fonte foram realizadas o *refactoring* da rotina de *login*, a retirada do uso de *views* e a escrita da consulta especificando as colunas necessárias ao invés de todas com o uso de "*", sendo esta última um uso recomendado pelas melhores práticas no que tange ao SQL. Além disso, foi modificado o acesso aos *resultsets* para o modo sem serialização e sem cópia. Cada uma dessas melhorias foi realizada cumulativamente na ordem: melhoria do código-fonte, criação de novos índices e a fragmentação das tabelas temporárias. Todas essas modificações foram testadas mediante a carga submetida pelo TRAFICGEN para analisar o ganho obtido.

Na Figura 2b o eixo x corresponde ao tempo em minutos da geração de tráfego (5 min.) e espera pelo término das *threads* e processos iniciados pelo TRAFICGEN, enquanto que o eixo y corresponde ao número de matrículas realizadas. Como pode ser visto, o maior ganho obtido ocorreu com o uso de fragmentação das tabelas temporárias, pois apesar da simplificação arquitetural efetuada, o sistema modificado ainda utiliza intensamente tabelas temporárias para a realização das regras de negócio. No ambiente do TRAFICGEN esses ajustes alcançaram uma melhoria $7.45\times$ superior para a funcionalidade mais crítica (adição de disciplina) e em média $2.6\times$ superior para as demais funcionalidades. Dessa forma, para a realização das matrículas no primeiro semestre

letivo de 2017 (2017/1) foi realizada a diminuição dos WebServers de produção pela metade, por ser essa a menor média de ganho encontrada.

As otimizações realizadas com ajuda do TRAFICGEN foram aplicadas na "Arquitetura Proposta" e foi obtido o resultado da "Arquitetura Proposta Otimizada", conforme mostrado na Figura 2a. É possível observar que o sistema após as alterações efetuadas teve um comportamento logarítmico ao invés de linear com relação ao número de matrículas no tempo, levando a uma melhoria de desempenho $2.92\times$ superior na primeira hora se comparado com a "Arquitetura Proposta", e $5.41\times$ em relação à "Arquitetura de Produção".

Durante o tempo em que o TRAFICGEN gerou carga foram realizadas 7.264 matrículas para a modificação "Fragmentação da TempDb" (em laranja na Figura 2b), enquanto que na "Arquitetura Proposta Otimizada" foram realizadas 3.368 matrículas (em amarelo na Figura 2b), sendo este último obtido dos primeiros 5 minutos de funcionamento real em ambiente de produção. Dessa forma, o sistema ainda suporta um aumento de carga de pelo menos 215%, sem degradação do desempenho. O aumento de carga suportado pode até ser maior que os 215%, pois além de o ambiente de produção possuir um poder computacional maior do que o do TRAFICGEN, a carga real a qual o sistema foi submetido foi inferior à do ambiente do gerador de tráfego, sendo este submetido a 1440 conexões simultâneas, enquanto que o sistema real alcançou o pico de 520 conexões simultâneas.

4. Conclusão

A melhoria de desempenho em sistemas distribuídos nem sempre está associada ao aumento da capacidade de processamento do *hardware*, sendo que os maiores ganhos geralmente ocorrem em mudanças no modo de funcionamento da arquitetura desses sistemas [Tanenbaum and Steen 2008]. Este trabalho mostrou ser possível simplificar a arquitetura e diminuir pela metade o número de servidores do sistema legado MW e mesmo assim obter um ganho de desempenho $5.41\times$ maior do que a antiga versão. Além disso, o sistema ainda suporta um aumento de pelo menos 215% na carga sem comprometer a qualidade no desempenho. Dessa forma, foi possível dar uma sobrevida a um sistema legado crítico sem necessariamente reescrever toda a aplicação. Esses resultados alcançados geraram uma grande satisfação da comunidade acadêmica com o Sistema de Matrículas da Universidade de Brasília no primeiro período letivo de 2017.

Referências

- Brasil (2007). Decreto nº 6.096 de 24 de abril de 2007. http://www.planalto.gov.br/ccivil_03/_ato2007-2010/2007/decreto/d6096.htm.
- Cameron Hughes, T. H. (2003). *Parallel and Distributed Programming Using C++*. Editora Addison Wesley, first edition.
- Coulouris, G., Dollimore, J., Kindberg, T., and Blair, G. (2011). *Distributed Systems: Concepts and Design*. Addison-Wesley Publishing Company, USA, 5th edition.
- Tanenbaum, A. S. and Steen, M. V. (2008). *Distributed systems: principles and paradigms*. Editora Pearson Prentice Hall, second edition.
- Tanenbaum, A. S. and Wetherall, D. J. (2010). *Computer Networks*. Editora Pearson Prentice Hall, fifth edition.