Algoritmos de seleção de web services baseada em técnicas de mineração de dados aplicada em arquiteturas orientadas a serviços

Lucas Junqueira Adami
Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo
lucasadami@grad.icmc.usp.br

Julio Cezar Estrella
Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo
jcezar@icmc.usp.br

RESUMO

O objetivo deste artigo é apresentar os resultados oriundos de experimentos nos quais são utilizadas técnicas de mineração de dados, especificamente algoritmos de aprendizado de máquina, para realizar a seleção de serviços em arquiteturas orientadas a serviço. Tais técnicas são componentes de um módulo acoplado a um *Broker* de uma arquitetura denominada *WSARCH*. Para a mineração dos dados são considerados informações de acessos de clientes da arquitetura provenientes de um *LogServer*. O resultado do treinamento no *LogServer* é usado para classificação de futuras requisições selecionando provedores aptos a atender os diferentes clientes. Esta forma de classificar provedores de serviços pode ser útil em caso de falhas em registros de serviços como o *UDDI*. A análise dos resultados após as fases de planejamento de experimentos e análise de desempenho mostram a eficiência das técnicas de classificação acopladas à arquitetura.

1. INTRODUÇÃO

A interoperabilidade entre os sistemas computacionais pode, nos dias atuais, ser alcançada por meio do paradigma orientado a serviços utilizando para essa finalidade os chamados *web services*. No entanto, para garantir que a interoperabilidade seja adequada, a qualidade do serviço envolvida neste processo precisa ser melhor explorada. Para estudar e avaliar essa questão, a *WSARCH* (*Web Services Architecture*) [3] foi desenvolvida. Trata-se de uma arquitetura para prover *web services* com qualidade de serviço. A arquitetura *WSARCH* é uma arquitetura cuja meta é ser uma referência para tratar de questões de desempenho na integração de aplicações distribuídas utilizando o paradigma orientado a serviços. O propósito da arquitetura é sempre buscar o melhor desempenho possível na integração de aplicações presentes em empresas e órgãos governamentais. A arquitetura é separada em 5 módulos: aplicação cliente, provedores, *Broker*, *UDDI* e o *LogServer*. A interação entre eles é mostrada na Figura 1.

WEBMEDIA '12 São Paulo, São Paulo Brazil

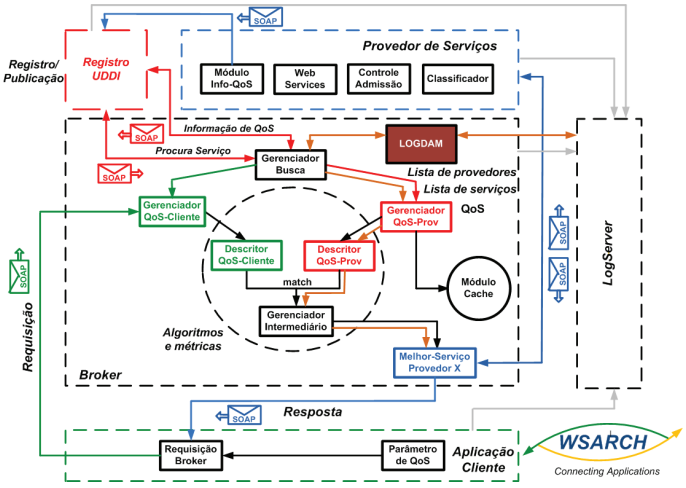

Figura 1: WSARCH - Modelo de interação [3]

Os clientes fazem requisições diretamente ao *Broker* da arquitetura. Eles especificam alguns parâmetros no envio da mensagem de requisição para que possam ser atendidos corretamente e receberem respostas adequadas. Os clientes podem ser desenvolvidos em qualquer linguagem de programação que se encaixe no padrão dos *web services*. Os provedores contém as aplicações que os clientes necessitam. Eles são servidores de aplicações que lidam com as requisições e respostas. Um provedor pode conter várias aplicações e cada aplicação pode apresentar várias operações. Assim como os clientes, os provedores de serviços podem ser codificados em qualquer linguagem de programação que esteja em conformidade com o padrão dos *web services*. O *Broker* é o elemento central da arquitetura. Ele é responsável pelo escalonamento das mensagens para os provedores de acordo com as exigências dos clientes. O *UDDI* armazena informações funcionais e não-funcionais dos serviços presentes nos provedores, tais como localização e especificações técnicas. O *Broker* da arquitetura faz requisições ao *UDDI* para selecionar o melhor provedor disponível para seus clientes. O *LogServer* armazena os registros de testes de desempenho realizados na arquitetura. Dentre esses registros, constam as informações de carga do *Broker* e do *UDDI* e as informações de acesso durante a requisição de um cliente. Os *logs* ficam armazenados em uma base de dados relacional do

LogServer. Esses dados contêm informações como horário do acesso, tempo de execução e endereço *IP* do provedor que atendeu a requisição [3].

Uma análise mais criteriosa e seletiva desses dados pode levantar questões quanto aos pontos críticos do sistema e servir de histórico para que decisões futuras possam ser tomadas com base no passado, permitindo realizar diversas abordagens. Nesse sentido, é proposto um módulo de análise de dados, o qual foi inicialmente descrito em [1], composto de algoritmos de seleção de *web services* que utilizam técnicas de aprendizado de máquina. Ele engloba um novo seletor na arquitetura *WSARCH* que utiliza dados locais para decidir qual provedor atenderá o cliente. O módulo entra em ação quando o seletor tradicional do sistema, que utiliza o *UDDI* para obter a lista de provedores disponíveis, não consegue buscar a informação de onde a requisição do cliente deve ser escalonada.

2. TRABALHOS RELACIONADOS

Guo et al. [4] definem um modelo de redes *Petri - WS_QPN* para processar os valores de *QoS* em composição de serviços. Uma vez encontrada que a solução para seleção de serviços com qualidade dirigida é NP-complexo, estratégias baseadas em algoritmos genéticos são mais adequados do que técnicas tradicionais de otimização. Foi usado um algoritmo evolutivo multiobjetivo, chamado *NSGA-II* para resolver este problema e avaliar o esquema de codificação e o objetivo das funções obtidas do *WS_QPN*. Shi [8] discute um modelo de *WSDL* estendido com o intuito de aumentar o detalhamento de *QoS*. Além disso, são propostos serviços com algoritmos baseados nesse modelo, e o algoritmo de seleção de *QoS* é baseado no modelo de programação inteira não-linear. O trabalho propõe o algoritmo de serviço correspondente com base neste modelo e do algoritmo de seleção *QoS* com base em NLIP. Depois de encontrar o solução do modelo avançado, o documento tem recebido as sequências de plano de serviços Web que cumpram as tarefas complexas de requisitos do usuário. Shen et al. [9] formalizam o problema de seleção *web services* como um processo de máquina de estados finitos. Propõem um algoritmo *backward* para criar a árvore de composição de *web services* (*WSCT - Web Services Composition Tree*), bem como um algoritmo heurístico para completar a seleção do *web service* com base no *WSCT*. Os resultados do experimento mostram que o algoritmo heurístico proposto é eficaz. Uma abordagem personalizada para prever *QoS* com base em filtragem colaborativa, para melhorar a eficácia é proposta em [7]. A ideia básica da abordagem é descobrir uma semelhança entre os consumidores, com os dados coletados de *QoS* e, em seguida, fazer a previsão para os serviços não utilizados, com base na similaridade. Os resultados experimentais mostram que a abordagem pode melhorar significativamente a eficácia da previsão de *QoS* em *web services*. Quando não há dados suficientes disponíveis para supor a semelhança, a abordagem ainda apresenta um bom potencial para prever a qualidade de serviço. Para selecionar um *web service* funcionalmente similar, os critérios não funcionais, tais como os de *QoS* são considerados em [5]. De acordo com o estudo, a maioria dos clientes não são experientes o suficiente para adquirir a melhor seleção de *web service* baseado em suas propriedades de *QoS*. Assim, propõem uma arquitetura baseada no *QoS Broker* o qual utiliza um mecanismo eficiente para encontrar o melhor *web service*.

3. MÓDULO DE ANÁLISE DE DADOS

Como é mostrado na Figura 2, o *Broker* é o responsável por receber requisições dos clientes e usar o seletor inteligente para escolher um provedor de serviços apto a atender o pedido do cliente. Originalmente, a arquitetura possui apenas um seletor, chamado de seletor tradicional. Esse seletor consulta o *UDDI* para obter a lista de provedores aptos a atender a requisição e utiliza a distância euclideana para escolher o melhor provedor para o atendimento. O seletor de serviços inteligente é um novo seletor do *Broker* da arquitetura. Ele utiliza o treinamento para realizar uma classificação usando alguns parâmetros obtidos do *LogServer*, retornando o *IP* do provedor que atenderá o cliente.

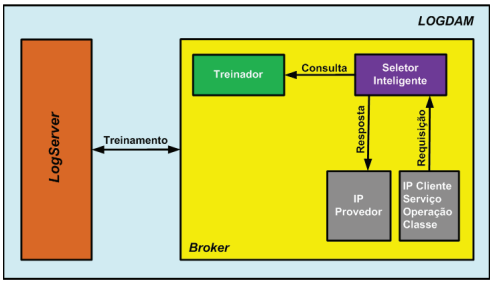

Figura 2: Módulo inteligente [1]

3.1 Desenvolvimento do treinador

Para realizar os treinamentos usando a base de dados, o treinador utiliza algoritmos oferecidos pelo *WEKA* [10]. O treinador é uma *thread* que executa junto ao *Broker*, ao qual são passados quatro parâmetros. O primeiro é o algoritmo que será responsável pela mineração. Há dois disponíveis: *Naïve Bayes* [2] e *IBk* [6]), implementados no *WEKA*. O segundo parâmetro é o número máximo de amostras recuperadas do banco de dados usadas no treinamento. Esse limite foi criado porque o número de acessos ao *Broker* pode ser muito grande e o treinamento para enormes quantidades de dados pode ficar inviável e lento. O terceiro parâmetro, a idade máxima das instâncias, permite ao treinador ignorar registros antigos que estejam presentes no *LogServer*. Registros antigos não são interessantes porque não refletem o comportamento atual da arquitetura. Caso o treinador utilize-os, o comportamento do seletor inteligente pode ficar inadequado e não obter um resultado adequado. O último parâmetro é o intervalo de treinamento. Não é necessário realizar um novo treinamento a todo instante. Por isso, essa variável foi criada para delimitar uma faixa de tempo em que o treinador fica inativo. Após a inicialização do treinador, sua *thread* treina um conjunto de dados que é recuperado na base de dados de acordo com os parâmetros passados. Os atributos utilizados para o treinamento são listados na Tabela 1.

Tabela 1: Atributos utilizados no treinamento

Atributos	
ServiceName	Nome do serviço
Operation	Operação associada ao serviço
ClientIP	IP do cliente que efetuou a requisição
Class	Classe do cliente (*gold*, *silver* ou *bronze*)
Provider	IP do provedor que atendeu a requisição

A classe a ser classificada de acordo com os atributos da Tabela 1 é o *Provider* e todos estão presentes em uma tabela

do *LogServer* denominada *AccessLog*. Para entender melhor como funciona o seletor inteligente, a Figura 3 mostra a interação entre as classes que compõem o módulo.

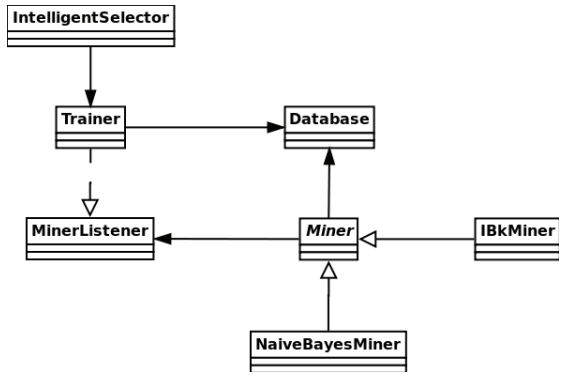

Figura 3: Diagrama de classes do módulo

A classe *Miner* é uma classe abstrata e pai de todas que utilizam os algoritmos de aprendizado de máquina. Ela é responsável por recuperar as instâncias para treinamento do *LogServer* e chamar o algoritmo de aprendizado implementado nas classes filhas *IBkMiner* e *NaiveBayesMiner*. Através da interface *MinerListener*, ela informa ao ouvinte o término do treinamento. A classe *Trainer* é o treinador do módulo. Ela é encarregada de administrar a conexão com o banco de dados através da classe *Database* e passá-la à classe mineradora. O treinador também carrega a classe mineradora especificada e chama-a periodicamente, sendo sua ouvinte. Além disso, a classe *Trainer* classifica novas requisições, pedido determinado pela classe *IntelligentSelector*.

4. PLANEJAMENTO DE EXPERIMENTOS

Para a execução dos testes dos algoritmos implementados, as seguintes configurações para o *UDDI*, *Broker* e provedores foram utilizadas de acordo com a Tabela 2.

Tabela 2: Infraestrutura

UDDI	
Processador	4 núcleos - Intel Core 2 Quad 8400
Memória	4GB - DDR3
HD	HD Virtual de 50GB
SO	Linux Ubuntu 11.10 Server
Qtde	1
Provedor	
Processador	1 núcleo - Intel Core 2 Quad 8400
Memória	6 com (512MB) e 6 com (1GB) - DDR3
HD	HD Virtual - 50GB
SO	Linux Ubuntu 11.10 Server
Qtde	12
Broker	
Processador	6 núcleos - AMD Phenom II X6 1090T
Memória	16GB - DDR3
HD	500GB
SO	Linux Ubuntu 11.10 Server
Qdte	1

Para avaliar os algoritmos usando o seletor inteligente e o seletor tradicional, dois cenários de avaliação de desempenho foram considerados conforme a Tabela 3.

No total, foram 1600 requisições para o primeiro cenário (16 clientes com 10 processos cada, repetindo 10 vezes) e 3200 requisições para o segundo (16 clientes com 10 processos cada, repetindo 20 vezes). Na primeira metade das requisições, o seletor tradicional com o algoritmo de distância euclidiana foi utilizado. Na segunda metade, o *Broker* utilizou o seletor inteligente com o algoritmo de classificação *Naïve Bayes* ou *IBk*.

Tabela 3: Cenários de avaliação de desempenho

Cenário 1	
Processos	10
Repetições	10
Qtde e tipo de cliente	8 bronze e 8 gold
Intervalo entre requisições	3 segundos
Cenário 2	
Processos	10
Repetições	20
Qtde e tipo de cliente	8 bronze e 8 gold
Intervalo entre requisições	3 segundos

5. RESULTADOS

Na avaliação dos resultados, três medidas foram usadas para analisar o desempenho do módulo e dos algoritmos de seleção: tempo de busca, tempo de serviço e tempo de requisição. O tempo de busca indica o intervalo que o seletor da requisição leva para conseguir uma lista de provedores capaz de fornecer o serviço adequado ao cliente. O tempo de serviço representa o intervalo da execução do serviço em uma requisição. O tempo de requisição indica o intervalo total da requisição. Ele é registrado no *LogServer* por meio da aplicação cliente.

Na Figura 4, cada segmento representa uma faixa de valores que contém os tempos de busca obtidos em cada item do experimento. Um item caracteriza-se pelo número de repetições (10 ou 20), o tipo do seletor (*IBk* ou tradicional) e o tipo do cliente (*gold* ou *bronze*). Observando a Figura, o tempo de busca para o seletor utilizando o algoritmo *IBk* é sempre próximo de zero, com uma faixa de valores pequena. Para o seletor tradicional, esse tempo varia, dependendo da carga imposta ao sistema. Essa diferença ocorre porque o seletor inteligente utiliza informações locais para classificar uma nova requisição enquanto o seletor tradicional busca no *UDDI* a lista de provedores aptos a atender o cliente.

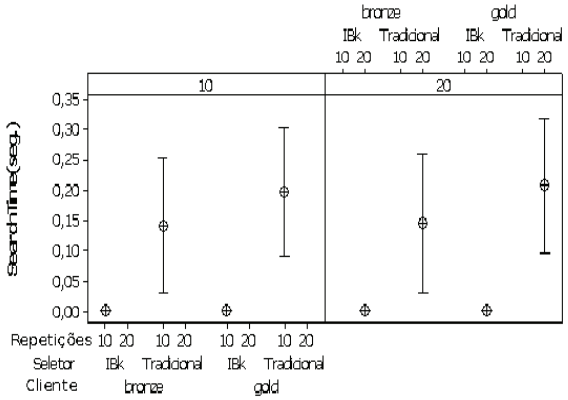

Figura 4: Intervalos de confiança do tempo de busca para o seletor *IBk* e o tradicional [1]

Na Figura 5, de modo similar à Figura 4, cada eixo vertical representa valores que representam os tempos de serviço do seletor inteligente utilizando o algoritmo *Naïve Bayes* e o *IBk* em cada item do experimento. O algoritmo *Naïve Bayes* é menos sensível à mudança da carga se comparado com o *IBk*. Esse comportamento é notado quando o tipo do cliente é *gold*.

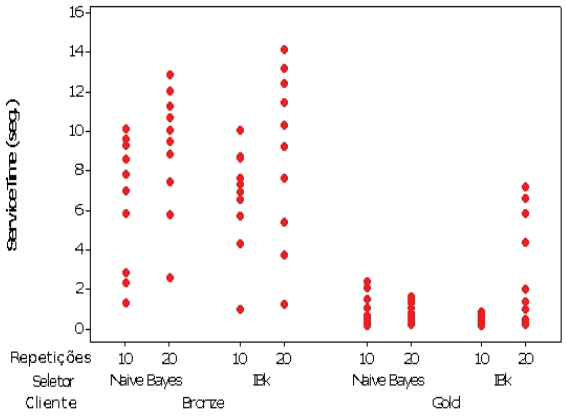

Figura 5: Concentração do tempo de serviço para os seletores *Naïve Bayes* e o *IBk*

Ao analisar a Figura 6 é possível observar que o algoritmo *Naïve Bayes* resulta em tempos que variam pouco na média quando a carga imposta ao sistema aumenta. Verifica-se também que os tempos para clientes *gold* e *bronze* são consistentes, uma vez que para o primeiro tipo, a duração da requisição é menor que para o segundo. Adicionalmente, o algoritmo observado possui um tempo de requisição menor na média se comparado com o seletor tradicional.

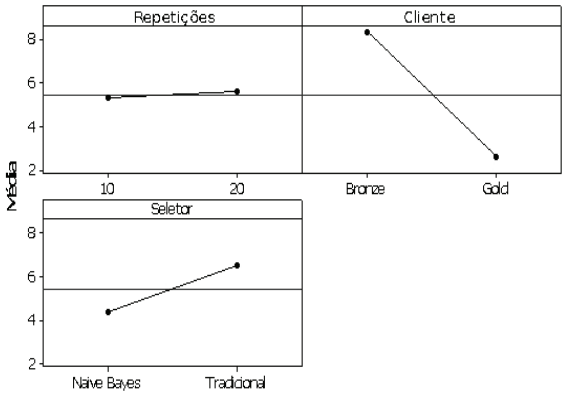

Figura 6: Efeitos sobre o tempo de requisição para o seletor *Naïve Bayes* [1]

6. CONCLUSÕES E TRABALHOS FUTUROS

Os resultados obtidos mostram que o objetivo de usar as informações do passado existente no *LogServer* para classificar novas requisições dos clientes por meio de aprendizado de máquina e atendendo aos padrões de qualidade de serviço foi alcançado. Como o método de seleção proposto não consulta o *UDDI* por provedores (o que é feito pelo seletor tradicional), mas se baseia na classificação do treinador, os resultados dos tempos de busca apresentados neste trabalho foram próximos de zero. Ainda, foi possível observar que o seletor inteligente foi melhor que o seletor tradicional para as configurações expostas. Se comparados os algoritmos de treinamento, ambos obtiveram resultados similares. O algoritmo *Naïve Bayes* foi menos sensível à mudança de carga nos testes. Adicionalmente, o trabalho mostra que é interessante pensar em um mecanismo onde o seletor tradicional possa consultar um *cache* local com informações provenientes do *UDDI* sem a necessidade de realizar um novo acesso a cada requisição de um cliente. Isso permitirá uma rápida consulta (tempo de busca), além de diminuir a carga no *UDDI*, melhorando assim o tempo de serviço e o tempo de resposta final. Como trabalhos futuros novos algoritmos, incluindo um mecanismo de inteligência artificial por meio de uma rede neural serão desenvolvidos, bem como um estudo comparativo entre as técnicas de mineração de dados e de reconhecimento de padrão/classificação.

7. REFERÊNCIAS

[1] L. J. Adami and J. C. Estrella. A data analyzer module for logs of service oriented architecture. In *III Escola Regional de Alto Desempenho - ERAD-SP*, jul. 2012.

[2] D. Aha and D. Kibler. Instance-based learning algorithms. *Machine Learning*, 6:37–66, 1991.

[3] J. C. Estrella, R. H. C. Santana, and M. J. Santana. *WSARCH: An Architecture for Web Services Provisioning with QoS Support - Performance Challenges*. Saarbrücken : VDM Verlag Dr. Müller GmbH & Co, 2011. http://www.amazon.com/WSARCH-Architecture-Provisioning-Performance-Challenges/dp/3639378245.

[4] F. Guo, L. Zhao, Y. Wang, and M. Zhang. Research on the web services selection problem. In *Education Technology and Computer Science (ETCS), 2010 Second International Workshop on*, volume 2, pages 284 –287, march 2010.

[5] P. Harshavardhanan, J. Akilandeswari, and R. Sarathkumar. Dynamic web services discovery and selection using qos-broker architecture. In *Computer Communication and Informatics (ICCCI), 2012 International Conference on*, pages 1 –5, jan. 2012.

[6] G. H. John and P. Langley. Estimating continuous distributions in bayesian classifiers. In *Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 338–345, San Mateo, 1995. Morgan Kaufmann.

[7] H. Liu, F. Zhong, and B. OuYang. A web services selection approach based on personalized qos prediction. In *Parallel and Distributed Computing (ISPDC), 2011 10th International Symposium on*, pages 199 –206, july 2011.

[8] S. Na. The study of web services selection based on qos. In *Computer Application and System Modeling (ICCASM), 2010 International Conference on*, volume 13, pages V13–109 –V13–112, oct. 2010.

[9] H. Shen, Z. Ding, and H. Chen. Reliable web services selection using a heuristic algorithm. In *Grid and Cooperative Computing (GCC), 2010 9th International Conference on*, pages 290 –295, nov. 2010.

[10] WEKA. Weka wiki. http://weka.wikispaces.com/Properties+file, 2012.