Software automatizador de estruturação de campanhas no Google AdWords

Mario M. T. B. de Rezende

Software automatizador de estruturação de campanhas no Google AdWords

Mario M. T. B. de Rezende

Supervisor: Rui G. Calsaverini
Empresa: Raccoon Agência de Marketing Digital LTDA.

Monografia de conclusão de curso apresentada ao Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo para obtenção do título de Bacharel em Ciências de Computação.

USP - São Carlos
Novembro de 2015

Resumo

O objetivo deste trabalho é desenvolver um sistema capaz de automatizar certas tarefas realizadas por funcionários do setor de marketing da empresa. O sistema deve ser capaz de aplicar uma lista de templates definidos pelo usuário a uma lista de produtos; após isso, o sistema deve ser capaz de utilizar a API do Google AdWords para consolidação das campanhas geradas pela primeira etapa.

Palavras-chaves: NoSQL, AdWords API, TDD

Sumário

Capítulo

1

Introdução

1.1 Sobre a Empresa

A Raccoon Marketing Digital[11] é uma empresa de publicidade online com foco em performance, atuando no mercado há pouco mais de dois anos.

Search engines utilizam diversos algoritmos e variáveis para ponderar a qualidade de sites. O valor pago para que anúncios sejam realizados no Google, por exemplo, depende da qualidade do site.

Performance é uma área do marketing digital que tem por objetivo aumentar vendas e diminuir gastos pagos por cada anúncio, ao invés de simplesmente aumentar a verba destinada aos anúncios. Performance é costumeiramente mensurada pelo conceito de *Return Over Investiment* (ROI).

Como parte da estratégia da empresa, a Raccoon também conta com um setor de TI responsável pelo desenvolvimento de softwares que possam automatizar certas tarefas rotineiramente realizadas pelo setor de marketing e softwares que possam melhorar o ROI de clientes.

Atualmente, a empresa conta com aproximadamente quinze funcionários no setor de TI e quarenta e cinco no setor de marketing digital.

Não existe plano de carreira e esse é um aspecto da empresa que necessita de amadurecimento. O valor da bolsa de estágio é consideravelmente maior do que o valor pago por outras empresas na região; porém, a ausência de plano de carreira implica que cargos não possuem responsabilidades bem definidas e a perspectiva de crescimento profissional (no sentido de promoções, não de conhecimento adquirido) é extremamente vaga.

1.2 Sobre o Processo Seletivo

O processo seletivo consistiu em três entrevistas. A primeira entrevista foi realizada por dois membros do setor de TI. A entrevista consistiu em perguntas para avaliar meu entendimento sobre conceitos de programação em geral, banco de dados, expressões regulares, Linux e familiaridade com *manpages* e sobre experiências anteriores.

As demais entrevistas foram realizadas com os dois fundadores da empresa, cada uma com um sócio. Foram entrevistas normais, e, em uma delas, precisei resolver alguns problemas estilo *guesstimate*.

1.3 Apresentação da monografia

Os próximos capítulos irão retratar de forma mais detalhada as atividades realizadas por mim durante o estágio.

No Capítulo 2, descrevo brevemente como foi o planejamento das atividades. É apresentado também como foi meu treinamento.

No Capítulo 3, apresento as atividades realizadas e ferramentas sendo utilizas para desenvolver tais atividades.

No último capítulo, são levantadas as conclusões sobre o estágio e a relação com o curso de graduação, indicando melhorias e perspectivas para o futuro, tanto para o curso quanto para o trabalho.

Capítulo

2

Planejamento do Trabalho

2.1 Atividades planejadas para o estágio

A atividade planejada foi o desenvolvimento do software chamado NapPy. NapPy é um software com o objetivo de automatizar a criação de campanhas e anúncios no Google AdWords.

O usuário pode criar templates de anúncios e aplicar tais templates em um conjunto de produtos para gerar anúncios. NapPy permite que o usuário escreva queries para selecionar produtos. NapPy então utiliza a API do Google Adwords para criar campanhas, grupos de anúncios e anúncios (forma como os anúncios são estrturados nos servidores do Google), após a aplicação dos templates.

2.2 Treinamentos planejados para o estágio

Apenas dois treinamentos foram ministrados. O primeiro, sobre o conteúdo do livro [12]. O segundo, sobre a ferramenta de versionamento *git*[4]. Ambos os treinamentos foram informais; apenas conversas sobre os assuntos.

Capítulo

3

Desenvolvimento do Trabalho

3.1 Atividades Realizadas

3.1.1 NapPy

O desenvolvimento inicial do NapPy foi realizado por mim e um colega de trabalho. Como não tínhamos familiaridade com a API do Google AdWords e nem com MongoDB, optamos por realizar o desenvolvimento em *pair programming*. Também seguimos a convenção da empresa, adotando as metodologias SCRUM e *Test Driven Development* (TDD).

3.1.2 Refatoração do NapPy

Eventualmente, meu colega decidiu sair do estágio por motivos pessoais. Assim sendo, solicitei que um sprint fosse alocado para que eu pudesse realizar benchmarks e me familiarizar com partes da code base que foram implementadas principalmente por meu colega pois eu continuaria como único desenvolvedor do software.

Os resultados do benchmark deixaram muito a desejar. NapPy, basicamente, realizava três requisições por campanha para os servidores do Google. Tais requisições eram o gargalo do sistema e, para enviar múltiplas campanhas, NapPy realizaria um número enorme de requisições.

Com tais resultados, solicitei que mais dois sprints fossem alocados para que o código fosse refatorado. A refatoração que planejei tinha por objetivo melhorar o desempenho do sistema e utilizar o serviço de labels da API do Google para simplificar a lógica de negócios do software.

3.1.2.1 Otimização

Como mencionado anteriormente, o gargalo do software era o número de requisições enviadas para o Google. NapPy realizava três requisições por campanha. A primeira requisição lidava com o serviço de campanhas da API; a segunda, com o serviço de grupos de anúncios; e a terceira, com o serviço de anúncios.

Tais requisições precisavam ser realizadas por campanha pois esta era a forma como o banco de dados foi modelado. A modelagem foi feita visando utilizar *embedded documents*, uma das formas de armazenar dados em MongoDB.

Para otimizar o desempenho, o banco foi remodelado. A nova modelagem (ainda em MongoDB) contém três tabelas. Uma tabela armazena campanhas; a segunda, armazena grupos de anúncios e o *id* da campanha à qual o grupo de anúncio pertence; a terceira, armazena anúncios, o *id* da campanha e o *id* do grupo de anúncio aos quais o anúncio pertence. Essas três tabelas são muito mais próximas da estrutura de árvore utilizada pelo Google para representar campanhas e anúncios do que a modelagem que utilizava documentos aninhados.

Com a nova modelagem, ficou trivial otimizar o software. Primeiro, uma requisição era enviada com uma lista de campanhas. Depois, uma segunda requisição contendo todos os grupos de anúncio pertencentes às campanhas enviadas anteriormente. Por fim, uma terceira requisição enviava a lista de anúncios pertinentes às entidades enviadas anteriormente. Três requisições seriam capazes de realizar toda a lógica de negócio. Na prática, porém, o número de requisições pode ser maior. A API do Google impõem certas limitações. Dessa forma, NapPy realiza requisições com, no máximo, 5000 operações.

Mesmo com a simplicidade da nova lógica de percorrer a estrutura de campanhas, a code base do sistema já tinha um tamanho considerável. Para evitar problemas durante a refatoração, segui técnicas descritas em [2]. Dessa forma, o sistema continuava funcionando, e a refatoração poderia ser comparada com a antiga implementação para validação de resultados.

Após a refatoração, executei os benchmarks novamente. A versão anterior do NapPy demorava 2 minutos para criar uma campanha. A nova versão é capaz de criar 20000 campanhas em 20 minutos.

A refatoração é considerada um sucesso. Não somente melhorou o desempenho do software como também simplificou em muito a rotina que percorre a estrutura de árvore das campanhas. Isso, por sua vez, permitiu que sub-rotinas de tratamento de erros fossem implementadas de forma concisa.

3.1.2.2 Utilização do serviço de labels

A API do Google Adwords contém um serviço específico para lidar com labels. Labels podem ser adicionadas ou removidas às entidades nos servidores do Google e utilizadas para a realização de queries em outras tabelas.

Após a execução do NapPy, toda a estrutura de campanhas e anúncios estaria concretizada nos servidores do Google, e os usuários poderiam proceder normalmente com as otimizações de campanhas e anúncios. Porém, os usuários utilizam uma ferramenta do Google para isso. Dessa forma, NapPy teria a responsabilidade extra de manter a coerência entre os dados no Google e dados em MongoDB.

Tal lógica de coerência seria extremamente complexa para implementar. Para lidar com tal problema, cogitava-se a implementação de uma interface gráfica para ser utilizada ao invés da ferramenta do Google.

Conjuntamente com a remodelagem do banco de dados, propus que labels fossem utilizadas para lidar com a coerência de dados. A ideia seria que o usuário continuasse utilizando a ferramenta do Google, mas que adicionasse labels pré-definidas para certas operações quando cabível. Por exemplo, se o usuário desejasse que um anúncio parasse de ser atualizado através das rotinas do NapPy, bastaria adicionar uma label específica ao anúncio. Labels não resolveriam completamente o problema, mas simplificariam em muito o código necessário para cobrir todos os casos de uso. Essa feature não foi implementada até o presente momento.

3.2 Problemas resolvidos

Ao término do estágio, o sistema NapPy se encontrava em estágio avançado de desenvolvimento e os requisitos de tempo de execução foram atingidos.

3.3 Técnicas, métodos e tecnologias envolvidas

3.3.1 MongoDB

MongoDB[8] é um banco de dados *NoSQL*, baseada em *documentos*. A definição de tais termos foge do escopo desse trabalho, e pode ser encontrada em [3].

MongoDB utiliza objetos JSON para representar documentos, que podem ser facilmente manipulados em Python. Queries também são descritas como objetos JSON.

MongoDB foi escolhido por sua simples utilização e pela possibilidade de ser escalado horizontalmente.

3.3.2 Python

NapPy foi implementado na linguagem Python, versão 3.

3.3.2.1 PyMongo

PyMongo[9] é o driver nativo de MongoDB para Python. A biblioteca é simples de ser utilizada. Queries em Mongo são objetos JSON, e dicionários em Python podem facilmente ser convertidos para objetos JSON. O mesmo vale para documentos resultantes de queries.

3.3.2.2 MongoEngine

MongoEngine[13] é uma ORM para MongoDB e Python. MongoEngine foi projetada para ser similar a outras ORMs utilizadas em Django, um framework em Python para sistemas *Model View Controller* (MVC).

MongoEngine é um tanto quanto mais abstrata do que PyMongo, até mesmo em como descrever queries. A biblioteca foi utilizada para implementar os modelos (inclusive os índices) utilizados pelo NapPy; mas PyMongo foi utilizada para implementar queries já que a abstração extra de MongoEngine seria desnecessária (podendo até mesmo tornar o código menos legível).

3.3.2.3 googleads-python-lib

googleads-python-lib[7] é a biblioteca cliente para Python. O repositório da biblioteca contem a implementação de diversos exemplos, inclusive como gerar credenciais de OAUTH 2.0 para se acessar a API do Google.

3.3.3 Google AdWords API

A API do Google AdWords[6] permite que dados do serviço de AdWords sejam acessados de forma programática. A API é composta por diferentes serviços, e.g. AdGroupAdService é o serviço utilizado para lidar com anúncios.

A API também disponibiliza a *AdWords Query Language* (AWQL)[5], para queries complexas. Queries simples não precisam utilizar AWQL; bibliotecas clientes possuem classes que implementam o *Query Object design pattern*.

A API impõem certas limitações em sua utilização. Existem limites nos números de requisições por minuto, no número de requisições realizadas concorrentemente e no número de operações por requisição. AWQL não implementa JOIN e nem GROUP BY.

3.3.4 Test Driven Development

O desenvolvimento foi realizado utilizando-se TDD. Tal prática consiste em escrever testes para o sistema antes que o código em si seja escrito. Dependendo de como os testes são escritos, eles podem até mesmo servir como documentação do sistema.

A fase inicial do projeto contou com uma pletora de testes de unidade. Tais testes visam especificar o comportamento de componentes do sistema de forma isolada. Como testes de unidade podem ter grande impacto na produtividade[1], optei por deixá-los sob versionamento por referência, dando apenas continuidade na escrita de testes de integração.

3.4 Impacto

Eventualmente, o desenvolvimento do NapPy foi cancelado. O projeto não possuía escopo bem definido e outros projetos com maior prioridade surgiram. Ainda assim, a code base ainda é utilizada como referência por ser escrita de forma bem estruturada e conter trechos de código que podem facilmente serem reutilizados (especialmente rotinas de benchmark).

O desenvolvimento do sistema também serviu para evidenciar certas falhas no processo de desenvolvimento de softwares da empresa como um todo.

3.5 Problemas não resolvidos

A feature que utiliza labels não foi implementada. O código que lidava com palavras-chave também não foi refatorado e incluído na versão refatorada pois o desenvolvimento já havia sido cancelado.

Capítulo

4

Conclusão

4.1 Benefícios para o crescimento profissional

O estágio me proporcionou a oportunidade de aprimorar áreas de meu conhecimento e o aprendizado de conceitos, técnicas, metodologias e linguagens novas. Estou satisfeito especialmente pela oportunidade de trabalhar com um banco de dados NoSQL.

Tive também a decepção de trabalhar utilizando a linguagem Python. Python foi projetada com foco em minimalismo. A linguagem é amplamente utilizada em trabalhos acadêmicos; também é utilizada pelo próprio Google na implementação de scripts, quando cabível. Infelizmente, ainda é comum promover a linguagem como "a linguagem que o Google utiliza". Google utiliza uma pletora de linguagens, e seus vários projetos open source podem ser vistos em seu repositório. Em minha opinião, Python está longe de ser a primeira escolha para o desenvolvimento de sistemas. Esse é um nicho melhor abordado por linguagens como Java e Ruby. Tal parágrafo pode ser considerado irrelevante, já que Python é uma linguagem *Turing Complete*, mas ainda o considero cabível pois a linguagem teve sim um impacto desmotivador em mim.

4.2 Considerações sobre o curso de graduação

O curso me forneceu fortes bases teóricas para o mercado de trabalho. Ainda assim, eu considero que o curso não foi bem planejado. Muito tempo é perdido em sala de aula, e pouco tempo é investido tarefas que realmente agregam conhecimento.

O mercado de trabalho é capaz de agregar conhecimento e experiência notavelmente mais rapidamente do que as aulas. Isso é de se esperar; contudo, a universidade deveria incentivar o estágio desde períodos iniciais da graduação. Ao invés disso, o que se tem é um foco *exagerado* em matérias de cálculo (seis créditos, mais do que as matérias de computação, se não todas) e períodos longos em sala de aula (algumas aulas podendo ter duração de até quatro horas). Infelizmente, para a maioria dos alunos, o estágio deixa de ser uma forma de complementar o próprio conhecimento e passa a ser mais uma matéria obrigatória que (de acordo com o período ideal do curso) só deveria ser cursada no último ano.

4.3 Sugestões para o curso de graduação

Ementas de disciplinas deveriam ser revisadas e atualizadas para condizer com o que se utiliza no mercado de trabalho. Infelizmente, uma série de disciplinas contém conteúdo irrelevante para o aluno. Por exemplo, a disciplina de Engenharia de Software I abrange o Modelo Cascata. Acredito que tal modelo devesse ser abordado, no máximo, em alguma disciplina específica de história da computação; a disciplina de Engenharia de Software poderia abordar, por exemplo, *workflows* adotados em sistemas de versionamento de código, como o *git*. Tal conteúdo é muito mais relevante para a vida profissional do aluno do que modelos obsoletos.

A ênfase em matérias de cálculo também deveria ser reanalisada. Tais matérias são importantes, mas a carga atual (seis créditos) é um tanto quanto exagerada. Acredito que realocar uma parcela de tal carga para matérias como estatística, matemática discreta ou uma miscelânea de matérias de computação seria algo benéfico.

O curso também deveria focar mais matérias pertinentes ao desenvolvimento de sistemas web e APIs. Existem disciplinas que abrangem tal contudo, mas tais disciplinas são optativas.

Como um todo, os trabalhos deveriam ser repensados. Não vejo motivo, por exemplo, em um trabalho onde o aluno deva implementar uma versão reduzida do *Microsoft Paint* utilizando Java e facilidades de *drag 'n drop* da IDE NetBeans. Não é algo comum, já que a maioria das disciplinas possuem trabalhos relevantes; mas ainda existe a falta de continuidade. Seria interessante, por exemplo, que o aluno implementasse um sistema inteiro ao longo do curso; um pouco em cada disciplina.

Por fim, a universidade deveria levar em conta aspectos extracurriculares dos alunos. Cursei uma disciplina onde o professor adicionou meio ponto na média final de alunos que participaram de uma determinada campanha para doação de sangue. Foi uma ideia um tanto quanto interessante; de qualquer forma, me pergunto porque nenhum professor adiciona pontos extras para alunos que contribuem para projetos open source ou para alunos ativos no site Stackoverflow[10], por exemplo.

4.4 Planos para o futuro

Pretendo continuar trabalhando na empresa. Apesar da inexistência de plano de carreira, a aquisição de conhecimentos está condizente com o que eu havia planejado para meu crescimento profissional.

Referências

[1] James O. Coplien. Why most unit testing is a waste of time. http://www.rbcs-us.com/documents/Why-Most-Unit-Testing-is-Waste.pdf. Acessado: 2015-11-09.

[2] Michael Feathers. *Working Effectively with Legacy Code*. Prentice Hall, 2004.

[3] Martin Fowler and Pramod J. Sadalage. *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley Professional, 2012.

[4] Git. Git. https://git-scm.com/. Acessado: 2015-11-08.

[5] Google. AdWords Query Language. https://developers.google.com/adwords/api/docs/guides/awql. Acessado: 2015-11-08.

[6] Google. Google AdWords API. https://developers.google.com/adwords/api/. Acessado: 2015-11-08.

[7] Google. googleads-python-lib. https://github.com/googleads/googleads-python-lib. Acessado: 2015-11-08.

[8] MongoDB Inc. MongoDB. https://www.mongodb.org/. Acessado: 2015-11-08.

[9] MongoDB Inc. PyMongo. https://api.mongodb.org/python/current/. Acessado: 2015-11-08.

[10] Stack Exchange Inc. StackOverflow. http://stackoverflow.com/. Acessado: 2015-11-08.

[11] Raccoon Marketing Digital LTDA. Raccoon Marketing Digital. http://raccoon.ag/. Acessado: 2015-11-08.

[12] Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall, 2008.

[13] MongoEngine. MongoEngine. `http://mongoengine.org/`. Acessado: 2015-11-08.