WebAgent: uma abordagem prática

Bruno Melo, brunoamelo@gmail.com, *Aluno, FEEC-UNICAMP*

***Abstract*—Este documento aborda o planejamento e desenvolvimento do projeto prático de final de semestre. O projeto é um agente de internet, denominado *webagent*, para auxílio na manutenção de servidores de páginas de internet, denominados *hosts*. O documento relata as fases: de *brainstorming*, de planejamento, de arquitetura, de implementações (e otimizações) e de análise de resultados. Ao fim, cita-se mais otimizações possíveis e futuros trabalhos sobre a arquitetura desenvolvida.**

***Index Terms*—Internet, spider, webagent, website maintenance**

I. Introdução

AGENTES de internet são agentes de software que atuam no ambiente conhecido como internet e são muito utilizados na atualidade por grandes empresas para lidar com o grande excesso de informação existente no mundo virtual. Para sensoriar e atuar no ambiente virtual os agentes de internet utilizam-se de sockets de comunicação, e através desses canais que as mensagens são enviadas permitindo o agente relalizar a percepção e a atuação. Uma peculiaridade deste agente e seu método de comunicação é que os sensores e atuadores podem estar em um mesmo canal de comunicação já que um socket pode ser bidirecional.

Com a popularidade destes agentes entre 1993 e 1995, um grande número deles coexistiam em operação e por causa de configurações mal feitas, eles causaram uma grande quantidade de transtornos. Assim, o desenvolvimento destas aplicações recebeu um apelo da comunidade científica, e também do mercado de trabalho, em implementar um conjunto de boas práticas para desenvolvimento desses robôs digitais.

Existem vários tipos de robôs de internet, vejamos três dos mais conhecidos:

- Webrobots: sistemas que navegam por servidores de páginas de internet, navegando de página em página através de ligações nos documentos, ou seja, referências para outros documentos *online*;
- Robôs de email: sistemas monitores de caixa postal digital, organizando mensagens através de parâmetros, palavras ou outra informação contida na mensagem e que possa ser identificada;
- Chatterbots: sistemas capazes de monitorar discussões em linguagem natural, muitas vezes respondem a comunicação simples na mesma linguagem, tomam decisões como expulsar ou banir um participantes da discussão.

Bruno Melo é engenheiro de software da empresa Tex Tecnologia Eletrônica, sediada em Itupeva/SP (telefone: +55-11-4591-2825; fax: +55-11-4591-2825; e-mail: engenharia@tex.com.br).

O presente trabalho prático é classificado no primeiro grupo, dos *webrobots*. Mais do que ou robô que captura links e navega de página em página, o projeto proposto visa suportar: a cópia de segurança do conteúdo de um *website*, a atualização automática do *sitemap* do website e manutenção ativa das páginas de um *website*.

II. Trabalhos Correlatos

Existem vários projetos na internet, inclusive muitos de código aberto, que realizam as operações desejadas neste presente trabalho. Podemos citar entre eles, os de código aberto: jSpider, Heritrix, WebSPHINX, WebEater.

Dentro destes, o que vale uma observação interessante é o jSpider. O projeto jSpider suporta: checar erros do site, checagem de links, construção de sitemaps, análise de estrutura do site, download de site inteiros, e outras tarefas. O mais interessante, além das tarefas *built-in* é a possibilidade de programar em Java, importar os objetos do projeto e extender o motor de jSpider, e através de tokens e funções construir a sua funcionalidade, adequada as suas necessidades.

III. Funcionalidades

O presente trabalho suporta quarto modos de execução. Os modos são independentes uns dos outros.

A. Modo backup

No modo backup, o sistema é responsável por acessar um servidor, navegar até um certo diretório e executar a cópia complete daquele nível. Cada cópia é identificada com uma etiqueta de tempo, e as cópias são separadas por um tempo de espera fixado.

B. Modo sitemap

No modo sitemap, o sistema é responsável por acessar um servidor, navegar até um certo diretório e rastrear todos os arquivos de conteúdo HTML daquele nível. Após o rastreamento, o sistema deve construir um arquivo *sitemap* representando a estrutura daquele diretório lido dentro do servidor, sempre repetindo a construção a cada intervalo fixo de tempo.

Um *sitemap* é um arquivo XML padronizado que reúne as informações de todas as páginas de seu site. Esse arquivo, quando lido por um *crawler*, como o GoogleBot, informa ao robô de atualização quais são as páginas que ele deve visitar, com qual freqüência ele deve procurar voltar para as próximas atualizações, e qual a relevância de cada página dentro do seu

website. O arquivo deve ser posicionado na raiz do *website*.

No agente desenvolvido, duas simplificações são realizadas: todas os arquivos localizados com conteúdo HTML são definidos com a prioridade máxima (valor 1) e com a mesma taxa de atualização (toda hora).

C. *Modo manutenção*

No modo manutenção, o sistema é responsável por acessar um servidor, navegar até um certo diretório e criar uma cópia deste diretório no disco local. Após a cópia, o sistema realiza uma verificação periódica (tempo fixado pelo usuário) do diretório e: para cada arquivo que falta no diretório remoto, copie o local para o remoto, para cada arquivo do diretório remoto que falte no local, remova-o do diretório remoto, para cada arquivo que existe em ambos os diretórios se o remoto é diferente do local então copie o local para o remoto.

Em outras palavras, nesse modo é feita uma cópia íntegra do diretório remoto, e a cada interval de tempo garante-se que o diretório remoto esteja igual ao diretório local.

D. *Modo checagem de links*

No modo checagem de links, o sistema é responsável por acessar um link inicial de internet e navegar até seus links âncoras realizando a mesma tarefa recursivamente. Para cada link âncora encontrado, verifica-se se ele já não foi visitado antes de adicioná-lo na lista de pendências. Verifica-se também se o link está dentro das dependências do *website* raiz definido, pois não desejamos indexar toda a internet, somente o diretório de um *website* indicado. Assim, espera-se que ao fim o sistema tenha rastreado toda a árvore de ligação entre as páginas do website, e tenha as informações de links funcionais e não-funcionais. Por fim, um relatório pode ser enviado por email para o administrador contendo a relação dos documentos e disponibilidade dos mesmos.

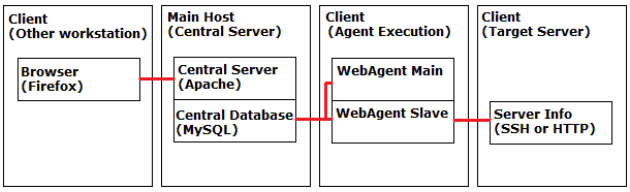

Figura 1. Arquitetura proposta

IV. Arquitetura proposta

Além das quatro funcionalidades descritas, existem outras características que foram trabalhadas no projeto. São elas:

- Administração remota: capacidade de controlar à distância a execução do bot;
- Relatório de execução: capacidade de rastrear toda e qualquer tarefa completa pelo bot;
- Acompanhamento de execução: capacidade de identificar que o bot está em operação;
- Funcionamento à prova de falhas: capacidade do bot voltar a funcionar caso seu processo seja finalizado na máquina cliente.

Com os objetivos anteriores em mente, planejei a arquitetura apresentada na Figura 1. Vamos entendê-la:

- O cliente *Client* pode controlar remotamente o agente pelo seu *browser* utilizando a *interface* de administração chamada de *Central Server*. Além disso, o cliente pode analisar os dados sobre a operação do agente nesta *interface* também;
- A *Central Server* acessa diretamente o banco de dados principal de nome *Central Database*, portanto, salva as operações e parâmetros e pesquisa as informações postadas pelos agentes;
- O *WebAgent Main*, quando inicializado na máquina do client, pesquisa no banco de dados central *Central Server* como deve funcionar e em qual modo. Assim, ele inicia um processo *WebAgent Slave*, que é independente. O agente principal é responsável por atualizar o estado no banco de dados principal e capturar os comandos que ele deve executar sobre o agente em execução, que é o escravo. No caso do escravo para de funcionar, o principal pode restartá-lo;
- O *WebAgent Slave* executa as operações necessárias dentro do modo que foi programado, ele pesquisa no banco de dados principal alguma informação que necessite e salva o histórico de suas operações. O agente acessa o servidor alvo, chamado de *Target Server* através de: ou SSH (backup, manutenção e sitemap) ou HTTP (checagem de links).

Durante a definição de prioridades de desenvolvimento, optou-se por não desenvolver a interface de administração indicada por *Central Server* dentro de *Main Host*. Essa decisão foi tomada pois este item nada agregaria ao conhecimento a ser trabalhado no agente, pois seria um conjunto de páginas dinâmicas, por exemplo em PHP, para ler e gravar no banco de dados. E, como teremos acesso ao banco de dados diretamente, durante os testes, seria um desenvolvimento que não seria bem aproveitado.

V. Implementação

A implementação deste projeto proposto foi executada evolutivamente, buscando sempre construir funcionalidades completas a cada interação de desenvolvimento.

A. *Versão 1.0*

Em um primeiro momento, optei por implementar uma aplicação altamente compatível com os mais diversos formatos de funcionamento. Considerei que o usuário poderia utilizar um servidor Windows ou Linux, e que poderia ter a interface de acesso ao servidor por FTP ou SSH+SFTP. Assim, decidi iniciar o desenvolvimento das duas funções mais simples: o backup e o sitemap.

Na função backup, dado um *website* e uma pasta, conecta-se ao servidor e faz o download arquivo por arquivo, entrando em subpastas. Após o download completo, faz a compressão em um arquivo único.

Na função sitemap, rastreia todos os arquivos do website com conteúdo HTML, guarda a URL de acesso e cria o arquivo *sitemap*. Por fim, copia o arquivo pelo FTP na raiz.

Após completar uma versão estável, testei com um website

e 10 arquivos/pastas. Após verificar o funcionamento, comecei a fase seguinte.

B. *Versão 2.0*

Nesta fase, decidi implementar a função de manutenção. Nesta função, faz-se o download de todo conteúdo no disco local. Depois, inicia um processo infinito que captura um arquivo do servidor, faz o download, verifica-se se sofreu modificação (comparando com o arquivo local espelhado) e copia o local para o servidor caso esteja diferente.

Após completar a versão estável, testei com alguns arquivos/pastas e passei para a próxima fase.

C. *Versão 3.0*

Nesta fase, fiz a implementação da função de checagem de links. Neste caso, o protocolo HTTP é utilizado, comunicando-se por sockets com o servidor alvo. Uma importante observação é que os dados dos links já visitados e o código HTTP retornado na pesquisa são armazenados no banco de dados central. Essa decisão foi tomada pois existia o desejo de tornar esta funcionalidade altamente escalável, e para isso seria necessário um ponto comum para sincronizar e controlar a execução de todos os agentes que trabalhariam conjuntamente.

D. *Versão 4.0*

Após completer uma versão estável de cada funcionalidade, iniciei alguns testes mais robustos. Inicialmente com 100 arquivos, notei um grande demora para cada função completar sua execução. Com um crescimento gradativo da quantidade de arquivos e pastas, ao chegar em +-2000 arquivos/pastas as funções demoravam até 30 minutos entre todas as operações necessárias.

Obviamente, percebi que aquilo não seria um bom resultado, e permitir tamanha flexibilidade ao usuário estava atrapalhando. Decidi descontinuar o suporte a FTP, mantendo somente o SSH+SFTP.

Mesmo assim, executando somente operações básicas como envia/recebe arquivo o tempo era muito alto. Decidi explorar características peculiares do sistema Linux, pois vislumbrei mais facilidade para otimizações explorando comandos Shell e delegando boa parte do processamento para o servidor. Assim, iniciei a reprogramação das duas primeiras funções: backup e sitemap.

Na função backup, a primeira otimização foi solicitar a compressão de todos os arquivos/pastas em um TAR.BZ2. Assim, reduziria o tráfego pela compressão e reduziria o tempo de IO entre memória e discos pelos dois servidores, pois seria somente um arquivo. A otimização foi muito bem pensada, pois reduziu o tempo de algumas dezenas de minutos para alguns minutos. Um ganho de 10 vezes.

Na função sitemap, a otimização foi pegar a lista de arquivos por uma listagem de diretório em comando Shell, utilizando somente um comando, ao invés de vários pequenos comandos de listagem diretório por diretório. O ganho foi de 2 vezes para os mesmo testes com +-2000 arquivos.

E. *Versão 5.0*

Nessa versão, conceontrei-me em otimizar a mais demorada das funções até então. A função de manutenção demorava quase 30 minutos para fechar um ciclo de funcionamento para quase 2 mil arquivos, o tornava-a impraticável pela demora em restaurar o conteúdo de um site, o que se propunha.

Nos últimos estudos com comandos Shell, descobri como executar a listagem completa do diretório, tirar o md5 de cada item da listagem e copia a resultado na saída padrão (output) todos ao mesmo tempo. Outra otimização pensada foi primeiro compactar todo o conteúdo do diretório alvo e transferir em um arquivo somente para o diretório local espelhado. Outro detalhes é que ao invés de tirar o md5 do arquivo local a cada iteração, armazenei todos os md5 em um dicionário na memória.

O conjunto de modificações gerou um ganho de 70 vezes na velocidade de início da função. A velocidade de início compreende: copiar a estrutura do diretório, tirar o md5 dos arquivos locais e armazenar em memória. Gerou um ganho de 90 vezes na velocidade de detecção de modificações no servidor remoto, considerando os quase 2 mil arquivos anteriores. Para cada diferença detectada, o tempo de restauração do arquivo remoto pelo arquivo local continuo o mesmo.

VI. Futuras Otimizações

As futuras otimizações, que chamo de versão 6.0, e ficaram pendentes pela complexidade envolvida, seriam:

- Maior velocidade na transferência local para remoto na função manutenção: para reduzir ao máximo o tempo de restauração de um arquivo/diretório, o ideal seria na fase de cópia criar uma pasta temporária no servidor remoto, chamada de espelho, e uma pasta na máquina local, chamada de cópia. As modificações detectadas entre o diretório alvo e o diretório espelho, ambos no mesmo servidor, seriam corrigidas rapidamente copiando do espelho para o alvo (modificação 1). As modificações detectadas, entre a pasta espelho e a cópia local, seriam corrigidas da seguinte maneira: listar as diferenças entre espelho e cópia local, compactar somente os arquivos locais listados, enviar o pacote compactado, descompactar o pacote no diretório espelho (modificação 2).

- Escalabilidade na checagem de links: para aumentar a velocidade de indexação, primeiramente seria importante trabalhar com threading dentro do agente (modificação 1). Segundo, com o armazenamento centralizado dos links já visitados, é possível disparar mais de um agente buscando o mesmo objetivo de mapear um determinado domínio. Porém, é necessário estudar e definir uma estratégia de divisão, ou seja, alguém (usuário ou agente) deve definir de onde o agente começa a procurar informações e como navegar para evitar menor quantidade de retrabalho. Por exemplo, um agente que controla os outros indicando para os outros de onde começar, e essa estratégia de início pode ser iniciar cada outro agente em

uma pasta ou subdomínio diferente, assim a chance de um deles buscar a mesma informação que outro já esteja buscando (ou mesmo que espere o outro buscar) é menor (modificação 2). Além disso, são necessárias, além de estruturas de sincronismo no banco de dados central, subestruturas para indexação rápida, pois com tantos processos abertos ao mesmo tempo e todos buscando e lidando com grandes strings, a lentidão será garantida, justamente por lidar com as url que são grandes strings (modificação 3).

VII. Resultados e Análises

Os resultados e análises apresentados nesta seção são referentes a última versão desenvolvida, ou seja, a versão 5.0.

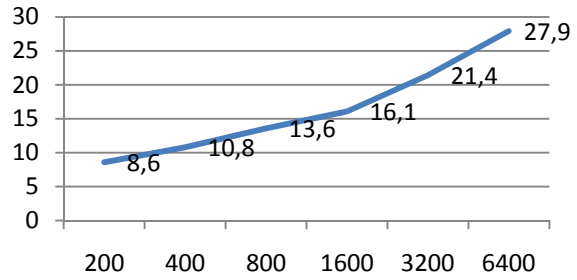

Gráfico 1. Modo backup. Relação de quantidade de arquivos e segundos consumidos para processamento.

No modo backup, temos os dados apresentados pelo Gráfico 1. O eixo esquerdo indica o tempo em segundos que a quantidade de pacotes demorou para ser processada. O tempo consumido compreende as atividades de: compressão do diretório, transferência para o disco local e identificação.

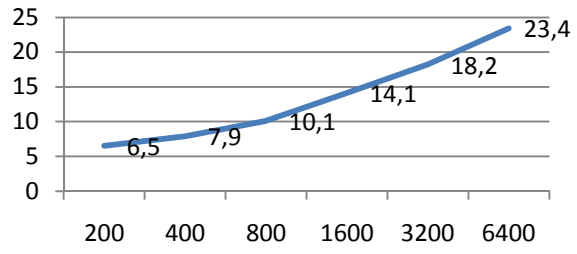

Gráfico 2. Modo sitemap. Relação de quantidade de arquivos e segundos consumidos para processamento.

No modo sitemap, temos os dados apresentados pelo Gráfico 2. O eixo esquerdo indica o tempo em segundos que a quantidade de pacotes demorou para ser processada. O tempo consumido compreende as atividades de: listagem de arquivos, seleção de arquivos com conteúdo HTML, escrita do sitemap no disco local, transferência do arquivo para o servidor remoto.

No modo manutenção, temos os dados apresentados pelo Gráfico 3. O eixo esquerdo indica o tempo em segundos que a quantidade de pacotes demorou para ser processada. O tempo consumido compreende as atividades de: listagem dos arquivos no servidor remoto, comparação com a listagem de arquivos locais.

No modo manutenção, antes de iniciar a fase de comparação do conteúdo entre diretórios local e remoto, executa-se uma fase de cópia das informações remotas. Em todas as execuções realizadas, o tempo de compressão foi muito rápido, não alcançando 1 segundo. O tempo de envio desse conteúdo sofreu pouca variação, 1,2 segundo no máximo. Isso deve-se a compressão realizada antes da transferência, pois como os testes foram executados com arquivos cheios de texto puro, a taxa de compressão é altíssima.

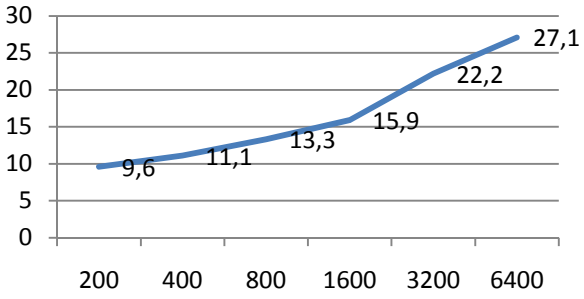

Gráfico 3. Modo manutenção. Relação de quantidade de arquivos e segundos consumidos para processamento.

Ainda analisando o modo manutenção, é importante ressaltar que o tempo de reconstrução do diretório remoto é proporcional a quantidade de arquivos e diretórios modificados. No modo atual, sem compressão, obtive o tempo de 0,5 segundo para cada arquivo a ser transferido. Por exemplo, deletando o diretório completo, com 2000 itens, teremos que aguardar 1000 segundos, ou seja, mais de 15 minutos. Isso mostra a necessidade de implementar novas otimizações, que explorem compressão e cópias temporárias no mesmo servidor, para reduzir mais o tempo de restauração.

VIII. Conclusões

Depois de projetar e implementar as funcionalidades descritas dentro da arquitetura proposta, e otimizar algumas vezes o projeto, é importante dizer que os conceitos de agentes de software foram importantes para desenvolvimento deste projeto. Podemos citar que as propriedades de agentes chamadas de independência e rastreabilidade delinearam todo o desenvolvimento do projeto.

Considerando os resultados de tempo obtidos inicialmente, e os obtidos após as otimizações que foram dependentes de sistema operacional e protocolo (SSH+SFTP) é válido afirmar que priorizando o tempo e o objetivo da funcionalidade, explorar as peculiaridades do servidor remoto foi uma ótima decisão. De nada valeria ter um agente que levasse 30 minutos para restaurar os arquivos do meu servidor se este é o tempo de sair da minha casa, plugar um pendrive na estação e copiar o conteúdo novamente.

Não foi objetivo deste trabalho dar suporte para diversos

protocolos. O grande objetivo, e alcançado, era desenvolver uma aplicação real baseada em necessidades reais (e minhas) para solução automática. Por isso, novamente a exploração de comandos em Shell (e quaisquer outros pacotes que dependam a aplicação) foi válida e importante.

IX. Trabalhos Futuros

Dentro do projeto desenvolvido, o primeiro trabalho é implementar o agente com capacidade de realizar boas decisões sobre como dividir o trabalho entre ele e mais agentes buscando mapear um endereço da internet, dando a propriedade de escalável ao projeto.

Segundo ponto, e não menos importante, é transformar o agente em um agente móvel, por exemplo, com suporte do Voyager. Aliado com a propriedade escalável, o agente pode explorar os recursos de rede para alcançar seu objetivo de mapeamento.

Outro ponto é aumentar o nível de controle sobre o agente. Hoje, só existe o comando de reiniciá-lo a distância, e devem surgir outros comandos necessários de acordo com o aumento do uso, ou o aumento de funcionalidades, do projeto.

Referências

[1] Jeffrey M. Bradshaw - "Software Agents", AAAI Press / MIT Press, 1997.
[2] Richard Murch, Tony Johnson - "Intelligent Software Agents", Prentice Hall, 1999.
[3] Fah-Chun Cheong - "Internet Agents - Spiders, Wanderers, Brokers and Bots", New Riders Publishing, 1996.