RENAN DOMINGOS MERLIN GRECA

WISYNC: SINCRONIZAÇÃO DE DIRETÓRIOS EM REDE LOCAL

CURITIBA

2015

RENAN DOMINGOS MERLIN GRECA

WISYNC: SINCRONIZAÇÃO DE DIRETÓRIOS EM REDE LOCAL

Monografia apresentada para obtenção do Grau de Bacharel em Ciência da Computação pela Universidade Federal do Paraná.
Orientador: Prof. Dr. Luiz Carlos P. Albini

CURITIBA

2015

SUMÁRIO

RESUMO

É muito comum ver hoje uma pessoa que possui mais de um computador; isso causou um crescimento na demanda por softwares de sincronização de arquivos, que permitem a um usuário manter diretórios de dois ou mais computadores diferentes totalmente sincronizados. Isso gera uma conveniência e também a segurança de um *backup* fácil de realizar. Neste projeto, é proposta uma solução de sincronização usando duas cópias do mesmo programa em dois computadores. Utilizando HTTP para transferir arquivos e o formato JSON para armazenar as listas de arquivos, o WiSync consegue fazer dois diretórios em computadores diferentes, mas numa mesma rede local (LAN), se tornarem idênticos. São descritas as maiores dificuldades encontradas durante o desenvolvimento do programa e os métodos usados para superá-las.

Palavras-chave: Sincronização, rede local, diretórios, backup

ABSTRACT

It's common nowadays to see someone who owns more than one computer; this caused a growth in demand for synchronization software, which allow a user to keep directories in two or more different computers completely synchronized. This is convenient and makes it easier to perform backups. In this project, a synchronization solution is proposed, using two copies of a same program in two computers. Using HTTP to transfer files and the JSON format to store lists of files, WiSync is able to make identical two directories in different computers, connected over a local area network (LAN). The largest difficulties found during the development of the program are described as well as the methods used to overcome them.

Keywords: Synchronization, local network, directories, backup

CAPÍTULO 1

INTRODUÇÃO

Com o barateamento e a popularização de microcomputadores, é comum um escritório, uma família ou até mesmo um indivíduo ter diversos computadores pessoais à sua disposição [16]. Portanto, torna-se útil e às vezes necessário existir uma maneira de manter esses computadores "conversando" entre si, trocando informações e arquivos para que o usuário possa mudar de um para outro de forma coesa. Além disso, computadores portáteis (*laptops*) são mais vulneráveis do que computadores domésticos a roubos ou danos físicos, além de outros fatores que podem comprometer a integridade das informações neles contidas, o que torna necessária a realização de *backups* frequentes.

Atualmente existem alguns programas disponíveis na Internet que fazem operações desse tipo, mas em contextos um pouco diferentes. Entre produtos comerciais, há diversos serviços de armazenamento em nuvem[1] que permitem que o usuário sincronize seus arquivos com um servidor na nuvem e com outros dispositivos, caso seja instalado um aplicativo em cada dispositivo. Exemplos desses produtos incluem Dropbox [12], Box [11], Google Drive [13], Apple iCloud Drive [10] e Microsoft OneDrive [8]. A vantagem desses produtos é que o usuário pode acessar seus arquivos facilmente de outros dispositivos como celulares e tablets – contudo, para a sincronização ocorrer é necessário que os computadores clientes estejam conectados à Internet e todos esses serviços impõem limites na quantidade de arquivos que podem ser sincronizados. A principal diferença para o atual projeto é a existência de um servidor mantendo os arquivos na nuvem.

Outra categoria de programas que realizam operações semelhantes são os programas de *backup*. Um exemplo desses programas é o FreeFileSync [18], solução *open-source* disponível na web. Considerando um par de diretórios A e B, programas de *backup* possuem três funcionalidades básicas:

[1] Com "serviço em nuvem", entende-se que o serviço funciona através de um servidor (ou um conjunto deles) na Internet; ou seja, os dados são armazenados em um servidor remoto ao usuário. [14]

- Sincronização de dois sentidos, que torna A e B idênticos copiando as modificações de um para o outro e vice-versa;
- Sincronização de espelho, que faz o diretório B ser um clone do diretório A, incluindo arquivos removidos; e
- Sincronização de atualização, que atualiza B com os arquivos novos ou modificados de A, mas não remove arquivos excluídos em A.

Enquanto o FreeFileSync sincroniza dois diretórios no mesmo computador e os serviços em nuvem se beneficiam de um servidor remoto que gerencia a sincronização, o objetivo deste projeto é desenvolver um programa que funcione de maneira local, ou seja, fazendo dois computadores interagirem pela rede sem um servidor no meio. Para o propósito desta monografia, o projeto irá focar no primeiro tipo de sincronização mencionada acima, a de dois sentidos. Os outros dois tipos também são interessantes e possivelmente serão adicionados ao escopo futuramente. Comparado às soluções em nuvem, isso permite que o programa seja rodado sem uma conexão à Internet, não impõe limites de dados e também resulta numa velocidade maior na transferência dos arquivos. Contudo, requer que ambos computadores estejam conectados à mesma rede e torna a sincronização em si um pouco mais complicada.

Programas que dependem da comunicação em rede são naturalmente desafiadores, pois eles dependem muito de fatores externos para funcionarem como esperado. No caso deste projeto, duas instâncias do programa estarão rodando simultaneamente numa rede e devem se encontrar e se comunicar. Então, além do desafio natural de comparar diretórios e transferir arquivos pela rede local, deve-se sempre manter em mente a sinergia entre as duas instâncias do programa, garantindo que uma esteja sempre preparada para a próxima ação da outra.

1.1 Proposta

Para trabalhar em diversos computadores ou manter cópias seguras de arquivos de forma conveniente, o presente projeto propõe um programa que, através de uma rede local (daqui

em diante referida como LAN, de *Local Area Network*), compare diretórios em dois ou mais computadores distintos e realize a sincronização dos mesmos. Com sincronização, quer-se dizer que o programa irá, para cada instância rodando:

- Copiar arquivos novos (adicionados desde a última execução) à outra instância;
- Apagar arquivos removidos nas outras instância; e
- Copiar alterações aos arquivos às outras instância, incluindo arquivos renomeados.

Então, para este trabalho, foi desenvolvido o WiSync, cujo nome vem de ***Wi****reless* ***Sync****hronization* (a sincronização não é necessariamente sem fio). No capítulo 2 são descritas algumas tecnologias que foram utilizadas para possibilitar o desenvolvimento do projeto. A seguir, no capítulo 3, é explicado com maiores detalhes o funcionamento do programa, além dos desafios encontrados e as soluções encontradas para eles. No capítulo 4, é explicada a forma de utilização do programa e são mostrados testes de desempenho.

CAPÍTULO 2

REVISÃO BIBLIOGRÁFICA

Neste capítulo, são apresentados o Python, o HTTP e o JSON, três tecnologias fundamentais para o funcionamento deste projeto. Além de introduzidos, são explicadas as vantagens e desvantagens de cada um e é dito por que foram escolhidos ao invés de suas alternativas.

2.1 Python

Python [17] é uma linguagen de programação de alto nível de propósito geral. Python é fácil de aprender, permite que software seja escrito com agilidade, dispõe de uma amplitude de bibliotecas eficientes e de código aberto e há uma grande comunidade disposta a auxiliar outros programadores. É uma linguagem interpretada, então programas criados com ela não enfatizam desempenho, sendo frequentemente mais lentos do que programas similares construídos em linguagens como C. Contudo, essa lentidão só se demonstra um problema em algoritmos computacionalmente pesados. Quando esse não é o caso, um programador pode optar pelo desenvolvimento mais rápido do Python em troca de alguns segundos a mais na hora de rodar o programa.

2.2 Sobre a Transmissão dos Arquivos

A primeira etapa do desenvolvimento do programa foi definir o método e protocolo que seriam usados na hora de transmitir arquivos entre os computadores. Após alguma pesquisa, quatro alternativas foram consideradas:

- **Sockets via TCP**, que utilizaria a biblioteca padrão `socket` do Python [5] para transmitir os dados dos arquivos a nível de camada de transporte utilizando o TCP (Transmission Control Protocol). A implementação em Python é relativamente sim-

ples, mas requer muita atenção porque arquivos maiores têm que ser divididos em blocos para depois serem re-montados. O TCP cuida da confiabilidade da transmissão dos blocos de arquivo [7], mas o programador tem que se preocupar com a organização desses blocos quando recebidos.

- **Secure Copy** (SCP), método que utiliza os mesmos padrões de segurança do protocolo SSH (*secure shell*) para garantir a segurança e confidencialidade dos dados [2]. A maneira mais simples de fazer isso em Python seria invocando um comando da linha de comando, que não dá muito controle sobre a operação de dentro do programa e também requer uma configuração mais específica dos computadores (ambos precisam ter o SSH configurado e o usuário precisa autenticar em ambos sentidos com senha).

- **File Transfer Protocol** (FTP) é um protocolo da camada de aplicação específico para o envio de arquivos. Na sua especificação, entre seus objetivos constam "1) promover o compartilhamento de arquivos, 2) encorajar uso de computadores remotos, 3) proteger o usuário de variões de sistemas de arquivos e 4) transferir dados de forma confiável e eficiente" [15]. Contudo, ele se mostra mais eficaz em situações onde há servidor e clientes bem definidos: ou seja, um servidor que hospeda arquivos e clientes que podem acessar e alterar os arquivos do servidor.

- **Hypertext Transfer Protocol** (HTTP) é o protocolo padrão de envio de dados na Internet, cuja versão 1.1, definida em 1997 e atualizada em 1999 [9], é atualmente utilizada. É fácil de utilizar, mas seu principal objetivo não é o envio de arquivos. Como é o protocolo padrão da Internet, é possível acessar os arquivos por um navegador se o endereço for conhecido.

Para este projeto, o HTTP foi escolhido para realizar a troca de arquivos. Isso se deve principalmente à facilidade de implementar essa troca, e também ao fato que dois computadores podem rapidamente trocar seus papéis de cliente e servidor enquanto transmitem arquivos entre si. Como a intenção do programa é ser usado em redes locais privadas, a abertura temporária dos arquivos na rede não é uma grande preocupação.

2.3 JSON

O *JavaScript Object Notation*, ou JSON, é um formato de armazenamento e troca de dados baseado na notação de objetos do JavaScript. Com o JSON, é possível armazenar objetos de um programa em um arquivo de texto, utilizando os tipos *array*, *string*, *number* e *boolean* do JavaScript [1]. O JSON foi escolhido por sua simples integração com Python e por ser um formato facilmente legível por um humano, facilitando a verificação dos dados durante os testes. Contudo, essa facilidade de leitura também se mostra um risco, já que é possível alterar os dados armazenados por fora do programa e então gerar resultados inesperados.

CAPÍTULO 3

DESENVOLVIMENTO

Neste capítulo, são descritos o desenvolvimento e as soluções da implementação do WiSync para este trabalho. A versão do WiSync entregue com este texto foi escrita em Python, versão 2.7. Originalmente, havia como objetivo tornar o programa compatível com os três sistemas operacionais mais comuns: Microsoft Windows, Apple OS X e Linux. Contudo, devido a diferenças inerentes na forma como o Windows funciona, optou-se por manter apenas compatibilidade com OS X e Linux. Para tal, o WiSync foi desenvolvido usando os seguintes computadores para testes:

Nome	"SgtPepper"	"Packard"
Sistema Operacional	OS X 10.11	Linux Mint 17
Processador	Intel Core i5-4308U	Intel Core i7-2600
RAM	8GB DDR3L	8GB DDR3
Armazenamento	SSD 512GB	HDD 320GB
Conectividade	Wi-Fi 802.11n 5GHz	Cabo Ethernet
IP local	192.168.1.110	192.168.1.132

3.1 Bibliotecas Usadas

Para executar o WiSync, são necessárias algumas bibliotecas padrão do Python: `os`, `sys`, `argparse`, `time`, `json`, `datetime`. Além dessas, é necessária a biblioteca `python-dateutil` [4]. Também é usada uma versão modificada do programa `woof.py` [6] (distribuída sob a licença GNU General Public License), que é usada na hora de transmitir os arquivos entre os computadores. No WiSync, arquivos JSON são usados para armazenar e transmitir as listas de arquivos geradas pela leitura e comparação dos diretórios.

3.1.1 Mudanças feitas ao woof

Como citado anteriormente, o programa `woof.py` [6] foi usado como biblioteca para auxiliar no envio de arquivos por HTTP. Para se adequar às necessidades do WiSync, algumas pequenas mas importantes mudanças tiveram que ser feitas:

- Após servir um arquivo ao outro computador, o endereço IP do computador remoto é armazenado para poder ser utilizado novamente no futuro sem depender de outras buscas por endereços.
- Originalmente, o `woof.py` utilizava diversas *threads* para permitir o *download* simultâneo de arquivos. Como essa funcionalidade não é necessária para o WiSync e as múltiplas *threads* causavam consequências indesejadas, o código foi alterado para desabilitar isso.

3.2 Arquivos JSON

Como mencionado acima, arquivos JSON é utilizado para armazenar os dados que precisam ser acessados por mais de uma instância do WiSync. Isso se refere tanto a duas instâncias em computadores diferentes quanto duas instâncias no mesmo computador, mas executadas em momentos diferentes. Quando o WiSync é executado pela primeira vez, ele cria um diretório oculto chamado `.wisync`, onde armazenará os arquivos JSON necessários. Os arquivos JSON produzidos pelo WiSync, bem como suas utilidades, são os seguintes:

- `files.json`: Lista de arquivos e diretórios contidos no diretório raiz antes da sincronização, com os metadados relevantes de cada um.
- `rfiles.json`: Uma cópia do `files.json` do diretório remoto sendo sincronizado.
- `changes.json`: Lista de arquivos e diretórios que devem ser sincronizados. Define arquivos que devem ser criados, sobreescritos ou excluídos em cada um dos dois diretórios sendo sincronizados. Este arquivo é gerado pela instância rodando em modo servidor (ver capítulo 4) e é então copiado para a outra.

- `last_sync.json`: Lista de arquivos e diretórios contidos no diretório raiz depois da sincronização, para ser usada por uma instância futura do WiSync.

3.3 Situações consideradas

Tratando-se das possíveis diferenças que podem existir entre um par de diretórios (que serão chamados de A e B), as seguintes possibilidades são consideradas:

- Arquivos existem em A mas não em B e vice-versa. Como demonstrado na figura 3.1, nessa situação basta copiar os arquivos do diretório que os contém para o que não os contém.

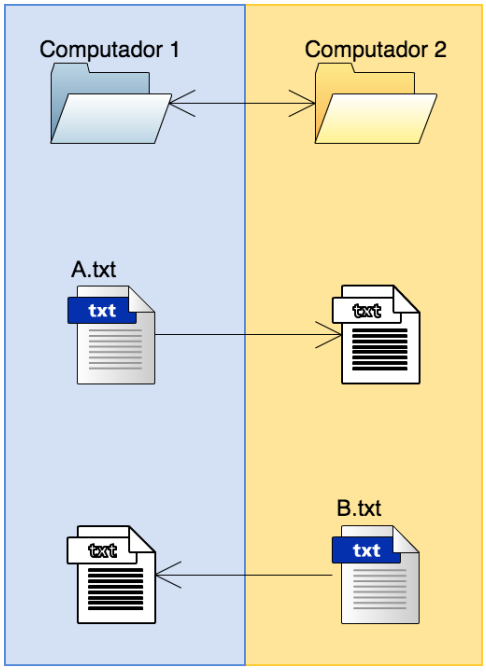

Figura 3.1: Caso em que os dois diretórios têm arquivos novos

- Arquivos com o mesmo nome existem em A e B mas têm conteúdo diferentes. Caso da figura 3.2. Nesse caso o WiSync utiliza o histórico da última sincronização antes da atual e as informações de data de modificação dos arquivos. Se o arquivo já existia antes e a data de alteração de uma das versões atuais é igual à anterior (implicando que o arquivo não foi modificado desde então), a versão mais recente substitui a mais antiga. O mesmo processo se aplica a um arquivo que foi excluído desde a última sincronização.

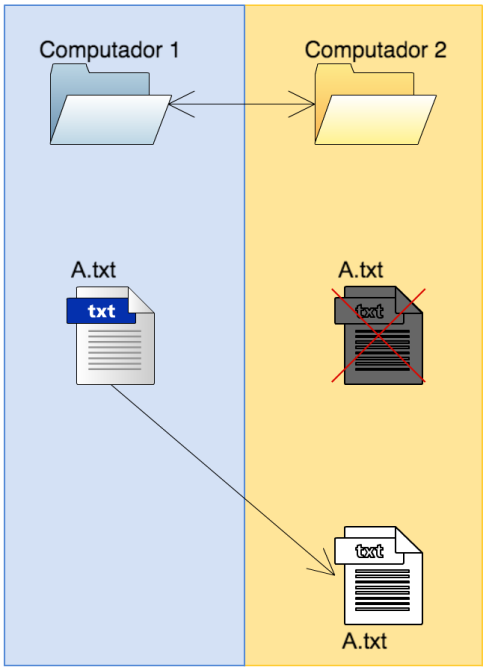

Figura 3.2: Caso em que um arquivo foi atualizado em apenas um dos diretórios

- Similar à situação acima, mas ambos arquivos têm datas de modificação diferentes às da última sincronização, ou não há informação sobre a última sincronização. Nesse caso, mostrado na figura 3.3, o WiSync copia ambas versões para o outro diretório, renomeando os arquivos para evitar conflitos e sobre-escritas.

- Caso um arquivo seja renomeado em A, o WiSync tratará essa situação como se fossem dois arquivos: um excluído e um novo. Caso o arquivo correspondente em B não tenha sido alterado desde a última sincronização, este será excluído e uma nova cópia, com o novo nome, será criada. Isso é ilustrado na figura 3.4. Caso o arquivo em B tenha sido alterado, então essa versão dele será copiada para A, mantendo o nome original, como mostra a figura 3.5.

3.4 Fluxo do programa

Durante a execução, o WiSync executa as seguintes operações:

1. Processa parâmetros e configura as classes WiFiles e WiNet, que preparam o que é necessário para lidar com arquivos e rede no programa.

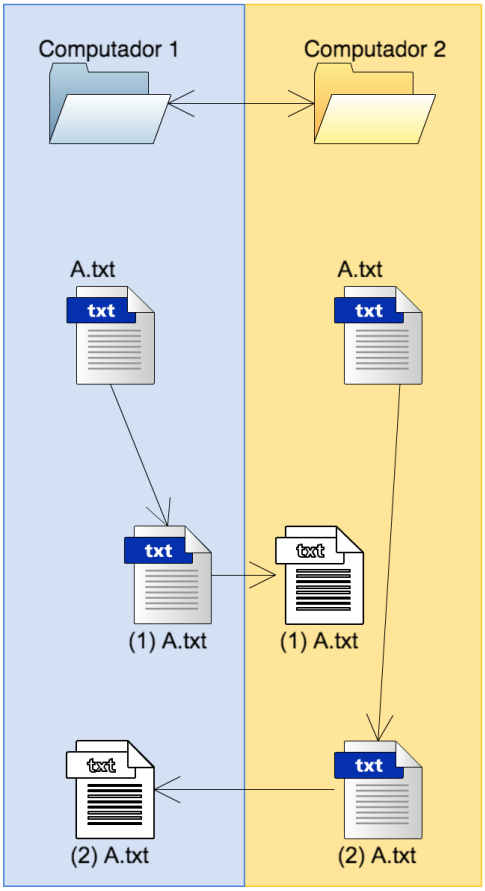

Figura 3.3: Caso em que há conflito de versões entre dois arquivos

2. As duas instâncias do programa trocam os arquivos JSON contendo informações sobre o diretório a ser sincronizado.

3. A instância que estiver em modo servidor (ver capítulo 4) utiliza esses arquivos JSON e, se existir, o arquivo contendo informações do estado anterior do diretório para gerar um novo JSON contendo a lista de arquivos que devem ser transferidos ou excluídos em cada um dos diretórios.

4. A instância que realizou a comparação começa a enviar os arquivos de acordo com a lista contida no JSON e, ao mesmo tempo, a outra instância recebe esses arquivos.

5. As instâncias trocam e a segunda passa a enviar arquivos para a primeira.

6. Ambas instâncias excluem seus próprios arquivos de acordo com o JSON.

7. Um novo JSON é gerado contendo as informações dos diretórios após a sincronização, que será usado para sincronizações futuras.

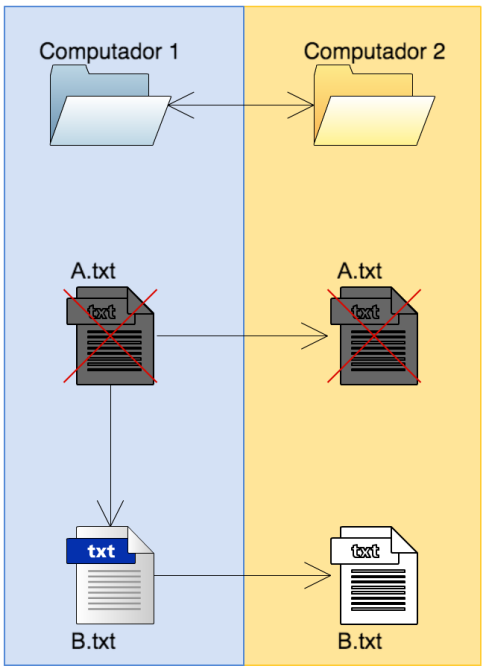

Figura 3.4: Caso em que um arquivo foi renomeado

3.5 Dificuldades encontradas

Durante o desenvolvimento do WiSync, algumas dificuldades foram encontradas. A seguir elas são descritas e são apresentadas as soluções encontradas.

3.5.1 Waits e timeouts

Talvez a parte mais difícil do desenvolvimento do WiSync tenha sido levar em consideração o fato de que o programa roda em dois computadores diferentes, com velocidades diferentes e que se comunicam apenas ocasionalmente através da troca de arquivos. Para evitar erros, é necessário garantir que ambos computadores estejam no mesmo passo ao mesmo tempo.

Um erro que aconteceu com frequência durante o desenvolvimento, por exemplo, é que um computador procurava o arquivo do outro antes deste ter preparado a hospedagem do arquivo, ou seja, o arquivo não estava disponível ainda e o primeiro computador retornava um erro de endereço inválido. Para evitar isso, foi adicionado uma pausa de 1 segundo logo antes de tentar acessar um arquivo do outro computador. Este tempo permite que o outro computador termine de preparar a hospedagem. Essa solução não é infalível, pois pode ser que o computador demore mais ainda para hospedar e, neste caso, o erro

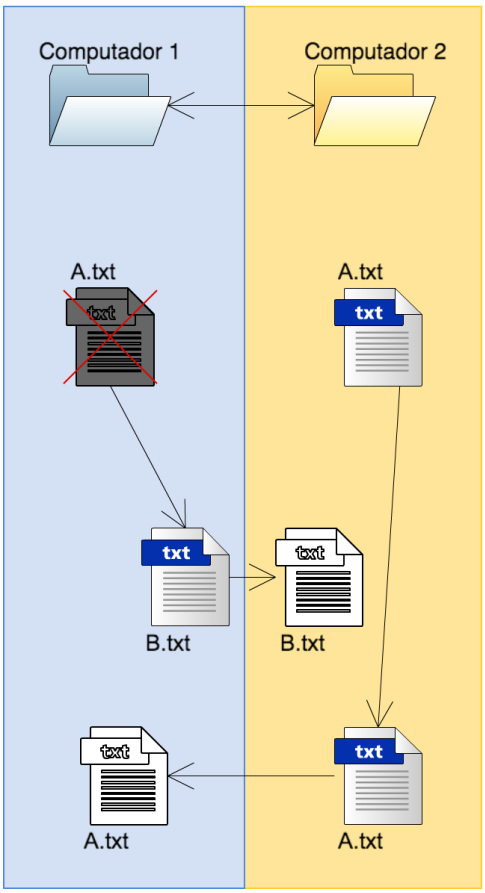

Figura 3.5: Caso em que um arquivo foi renomeado em um computador e modificado em outro

acontece de qualquer forma. Uma solução para isso seria criar um laço onde se tenta diversas vezes acessar o arquivo remoto mas, nos testes, isso teve um desempenho pior do que a pausa.

3.5.2 Datas de criação/modificação

Como explicado anteriormente, o WiSync utiliza metadados dos arquivos – a data de modificação em particular – para decidir quais devem ser copiados. Contudo, isso gerou um problema que não havia sido previsto: quando um arquivo é copiado para o outro, a data de modificação da cópia é a data de sincronização – não a data de modificação do arquivo original. Portanto, numa sincronização subsequente, a cópia é entendida como uma versão nova do arquivo e copiada novamente para o computador original.

Para resolver isso, foi usada a função `os.utime` do Python, dessa forma configurando as datas de criação e modificação do arquivo criado para serem iguais às do arquivo de

origem. Uma preocupação que surgiu em relação a isso foi a possibilidade de, devido a configurações diferentes entre os computadores, um arquivo ter data de criação e/ou modificação futura ao relógio do sistema. Contudo, testes foram realizados no Linux e no OS X e isso não se mostrou um problema.

3.5.3 Ordem dos arquivos

As listas de arquivos que o WiSync necessita para realizar a sincronização são armazenadas usando o tipo *Dictionary*, do Python, para que torne-se simples indexar os arquivos utilizando os nomes deles. Contudo, um dicionário, ao ser percorrido, não tem uma sequência definida a ser seguida. Portanto, na hora de enviar e receber arquivos, o dicionário é convertido para uma lista de tuplas, ordenada alfabeticamente com os nomes dos arquivos. Isso garante que ambos computadores enviem e recebam arquivos na mesma ordem.

CAPÍTULO 4

RESULTADOS

Na versão entregue com esta monografia, o WiSync cumpre quase todos os objetivos descritos anteriormente. Com dois computadores na mesma rede local, com duas cópias do WiSync é possível realizar a sincronização entre os diretórios.

Era desejável eliminar a relação cliente/servidor entre os dois computadores, mas isso se mostrou demasiadamente complicado considerando o escopo do programa, pois cada instância precisaria verificar se há outra instância rodando na rede e estar preparada para ser encontrada por outra instância ao mesmo tempo. Portanto, é necessário que um dos computadores rode em "modo servidor" (usando o parâmetro `-s`; veja abaixo) e o outro rode normalmente.

Ao executar o WiSync, é possível explicitar qual é o computador destino utilizando o endereço IP local ou o nome de rede mas, caso esse parâmetro não seja passado, o programa irá realizar realizar uma busca por endereço IP na rede local para tentar encontrar outra instância do WiSync.

4.1 Organização do Programa

O WiSync é composto por três arquivos: `wisync.py`, `winet.py` e `wifiles.py`. Abaixo estão as funcionalidades de cada um:

- `wisync.py`: Arquivo principal do projeto, responsável por ler os parâmetros da linha de comando e controlar a execução do processo.
- `winet.py`: Contém a classe WiNet, que inclui os métodos e atributos necessários para fazer as partes em rede do programa, como hospedar e receber arquivos. Quando o WiNet é instanciado, é decidido se o programa irá rodar em modo servidor ou modo cliente

- `wifiles.py`: Contém a classe WiFiles, que inclui os métodos e atributos necessários fazer as partes que lidam com o sistema de arquivos do programa, como ler e comparar diretórios.

O `woof.py`, mencionado acima, também é necessário para a execução. A biblioteca `python-dateutil` [4] deve ser instalada separadamente; recomenda-se utilizar o gerenciador de pacotes `pip` [3] para isso.

4.1.1 Parâmetros de Execução

Para rodar o WiSync, utiliza-se o seguinte comando:

```
python wisync.py [-h] -d DIRECTORY [-n HOSTNAME] [-s]
```

Os argumentos são os seguintes:

- `-h, --help`: Mostra o texto de ajuda do programa.
- `-d DIRECTORY, --directory DIRECTORY`: Caminho para o diretório a ser sincronizado (de preferência, o caminho completo)
- `-n HOSTNAME, --hostname HOSTNAME`: Nome de rede ou endereço IP do outro computador. Esse argumento é opcional e pode ser preenchido tanto pelo endereço IP do outro computador quanto pelo nome de rede. Por exemplo, `-n SgtPepper.local` e `-n 192.168.1.110` têm o mesmo efeito. Caso o argumento não esteja presente, o programa procura outras instâncias do WiSync na rede local.
- `-s, --server`: Modo servidor. Se definido, esta instância será o "servidor", hospedando seus arquivos antes da instância remota.

4.1.2 Bugs Conhecidos

Quando se exclui um diretório inteiro dentro do diretório sincronizado, o WiSync não tem como distinguir se o diretório em si foi excluído ou simplesmente todos os arquivos dentro

dele foram removidos. Então, no outro computador, todos os arquivos são excluídos mas o diretório vazio permanece lá.

O programa também não está preparado para retomar a sincronização após uma eventual quedas na conexão durante o processo. Se isso ocorrer, a sincronização sera abortada com um erro. Após isso o usuário pode tentar executar o WiSync novamente, assim realizando outra sincronização e finalizando com o resultado esperado.

Finalmente, devido à forma em que os arquivos JSON são armazenados, tentar sincronizar o mesmo diretório com diretórios remotos diferentes (com um terceiro computador, por exemplo) pode gerar resultados inesperados.

4.1.3 Desempenho

Como era esperado, numa rede com condições ideais a troca de arquivos se mostrou ágil. Nos testes realizados, foi possível sincronizar um diretório contendo um único arquivo de 1GB (1.073.748.731 bytes) em aproximadamente 2 minutos. Contudo, como também era esperado, a execução do programa é mais demorada quando existem vários arquivos, devido às esperas que as instâncias devem fazer para manterem-se sincronizadas. Foram realizados testes sincronizando um diretório contendo 100 arquivos de 1MB (1.048.576 bytes) cada, totalizando 100MB, e esse processo levou aproximadamente 14 minutos.

CAPÍTULO 5

CONCLUSÃO

Neste projeto, foi demonstrada a viabilidade de se criar um sincronizador de arquivos simples mas eficaz utilizando HTTP numa rede local com Python.

Utilizando arquivos temporários em JSON e com a ajuda de algumas bibliotecas Python, a sincronização se demonstrou bem-sucedida em todos os testes realizados na versão entregue.

Durante o planejamento e o desenvolvimento do software, foram verificadas as maiores dificuldades de criar um software deste tipo. Como esperado, manter duas instâncias do programa em computadores diferentes rodando de maneira organizada mostrou-se o maior desafio do desenvolvimento, enquanto a transferência de arquivos em si foi razoavelmente simples com a ajuda do `woof.py`.

Em possíveis atualizações futuras, o foco seria em reduzir as esperas adicionadas para manter a sincronia entre as instâncias, assim tornando o processo intero mais ágil. Ainda mais interessante seria possibilitar que o WiSync rode como serviço no sistema operacional, permitindo que ele verifique mudanças nos arquivos em tempo real e sincronize os diretórios configurados de acordo.

BIBLIOGRAFIA

[1] Json. Disponível em: http://www.json.org/.

[2] Linux and unix scp command. disponível em: http://www.computerhope.com/unix/scp.htm.

[3] pip. disponível em: https://pypi.python.org/pypi/pip.

[4] python-dateutil. disponível em: https://pypi.python.org/pypi/python-dateutil.

[5] Socket programming howto.
Disponível em: https://docs.python.org/2.7/howto/sockets.html.

[6] Simon Budig. Woof. Disponível em: http://www.home.unix-ag.org/simon/woof.html.

[7] Vinton G Cerf e Robert E Icahn. A protocol for packet network intercommunication. *ACM SIGCOMM Computer Communication Review*, 35(2):71–82, 2005.

[8] Microsoft Corporation. Microsoft onedrive. Disponível em: https://onedrive.live.com.

[9] R Fielding, James Gettys, Jeffrey C Mogul, H Frystyk, Larry Masinter, P Leach, e T Berners-Lee. Rfc 2616: Hypertext transfer protocol—http/1.1, 1999. https://tools.ietf.org/html/rfc2616.

[10] Apple Inc. Apple icloud drive. Disponível em: http://www.apple.com/icloud/icloud-drive.

[11] Box Inc. Box. Disponível em: https://www.box.com.

[12] Dropbox Inc. Dropbox. Disponível em: https://www.dropbox.com.

[13] Google Inc. Google drive. Disponível em: https://drive.google.com.

[14] Peter Mell e Tim Grance. The nist definition of cloud computing. 2011.

[15] Jon Postel e Joyce Reynolds. Rfc 959: File transfer protocol, 1985. https://tools.ietf.org/html/rfc959.

[16] Reineke Reitsma. How many us households have multiple pcs? Disponível em: http://blogs.forrester.com/reineke_reitsma/ 11-04-29-the_data_digest_how_many_us_households_have_multiple_pcs.

[17] Guido van Rossum. Python. Disponível em: https://www.python.org/.

[18] ZenJu. Freefilesync. Disponível em: http://sourceforge.net/projects/freefilesync.