COSC480 Project
Interim Report

Calum O'Hare
Supervisor David Eyers

1 Project goal

The aim of this project is to develop a file synchronisation tool. Similar to Dropbox (and others) its main function should be to keep data synchronised between multiple devices. What makes it different however is it should:

- Be decentralised. It will not necessarily need to be run in "the cloud", there should be no centralised server, just many cooperating client nodes.
- Allow file synchronisation between multiple clients—not just point-to-point between two clients. Clients may be running different operating systems. Clients may run on different networks, with different costs of access (including being disconnected from the Internet at times).
- Allow for fine-grained user control for the majority of the program's functions, *e.g.*, how often, and what, to replicate within different sets of files. 'What' could be file name, file type, file size *etc.*
- Show statistics about which files are being replicated, efficiency, cost (bandwidth, disk space). These statistics could also possible lead to a heuristic for when to sync a given file.

2 Background

There are already many services out there that synchronize your files. Dropbox, Google Drive, Microsoft SkyDrive, Apple iCloud which all offer cloud

based solutions for automatically synchronizing your files. The problems with these services is privacy and availability. Storing your data with a third party gives them access to your documents. If you are a commercial venture with sensitive information this might be concerning. You also can not guarantee that you will always be able to access your data, if the company who owns your data goes bankrupt or decides to shutdown their service you could lose all of your data with little or no warning.

The are other possible approaches to replicating files across multiple computers. For example you could use version control systems like git, subversion, mercurial, cvs. The potential problem here is that they are centralised, they rely on a central server should that server fail the replication will break. The same can be said for cloud based solutions, although these may be distributed on the back end a DDOS attack can still cut off or slow down the replication. Using version control for this purpose creates a bottleneck on the central server. It also means that replication is not automatic and has to be manually invoked.

Example use case

I like to keep all of the data on my laptop backed up to an external hard drive. The data on my computer that I wish to back up falls in to three main categories: documents, music, and movies. Documents are mostly scripts and programs that I am writing for University or work projects. Documents also include reports for assessment. These documents change very frequently and are very important to me. Often these are small files (but not always). My music collection changes relatively infrequently, files are around ≈5MB and I like to have a relatively current backup of this collection. My movie collection contains fairly large files but I don't need it to be backed up very often as it doesn't change very much and I don't care if I loose a couple of DVDs. Files that I work on at University would be very useful to have on my laptop at home. Files I work on at work mostly stay at work but occasionally I might want to bring something home to work on. The other device I always have with me and may be on one of any given (Wi-Fi or 3G) network at a certain time is my smartphone. I would like to have photos taken on this backed up to either (or both) my laptop and external hard drive.

Some of the files I move around are of sensitive or personal nature and I would prefer not to store them with a third party vendor. I also have different synchronisation requirements for different types of data. An effective file

synchronisation tool would be of great use to me personally.

– Insert image of personal graph, discuss coporate graph too –

3 Achievements to date

3.1 Virtual Machines, Node networks

For testing my program I needed to have a network of computers which can be linked together in different arrangements easily. I decided to use virtual machines for this job since it means I do not need to have set amount of physical machines and that I can manipulate the links between them easily and create new machines very easily.

I have used Oracle's VirtualBox for this job. I chose VirtualBox because of its easy to use command line interface. I have several scripts which call the vbmoxmange command to set up the internal network connections between machines and then start up the machine itself. This makes switching between network configurations very easy as I can just run a different script depending on what network topology I would like to test.

I have decided to use some basic topologies to test my program to start with as shown below.

– Insert network graphs –

3.2 Python

I have chosen to use python to implement my program. Python appealed to me because it supports many different platforms (Windows, Linux, OS X). This is useful because it means I will (hopefully) encounter fewer compatibility problems when running my program across different operating systems in the future.

3.3 User control

One of the main goals of my project is to allow the user to have a large amount of control over how the programs behaves. I currently have the program reading from a configuration files which allows the user to specify which directories they want to watch and where those directories should be synchronised to.

I chose to use directories as my granularity for replication as opposed to files because keeping track of a big list of files may become unwieldy and because I replicate directories recursively I can replicate large amounts of data without a cluttered configuration file.

3.4 Monitoring Directories

The application need to monitor directories for changes so that it knows when to perform a sync. The reason I have chosen to do this is because syncing a directory that has not been changed is a waste of time and my application is designed to be as efficient as possible. I do not however want to be continually polling the watched directories to see if there have been changes made. This would be a terrible waste of CPU time. Instead I have looked into ways of being notified of a change in the file system below the watched directory.

- Inotify
  - Linux kernel feature that has been included in the linux kernel since version 2.6. inotify is used to watch directories for changes and notify listeners when a change occurs. inotify is inode based and replaced dnotify an older system which provided the same function, however it was inefficient, it opened up the file descriptors for each directory it was watching which meant the backing device could not be unmounted. It also had a poor user-space interface which uses SIGIO. Inotify only uses one file descriptor and returns events to the listener as they occur. It works well and does what I need it to do. There is a python module called pyinotify which provides a python interface to inotify which I have used and tested in my program.
- FSEvents
  - FSEvents is an API in Mac OS X. It is similar to inotify in that it provides a notifaction to other applications when a directory is changed however it doesn't notify you which file in the directory was changed. This does not matter for my application since unison is smart enough not to copy unchanged files in a directory. The is a python module for FSEvents, as well.

I also looked at using the kqueue system call which is supported by OS X and FreeBSD. It notifies the user when a kernel event occurs. I decided against using kqueue as the high level approach of FSEvents, suits the applications needs.

- ReadDirectoryChangesW
  - Windows like the other operating systems I have looked up provides a nice way of doing this too. There is a function called ReadDirectoryChangesW. I have looked at how I might implement this and have come across the FileSystemWatcher Class in .NET 4 and up. IronPython might prove to be a good choice for a windows implementation although I will need to look into it further.

3.5 Point-to-Point synchronisation

After some preliminary analysis of the available file synchronisation tools I have found a tool called Unison to be a promising starting base for this project. Unison is an open source file synchronisation tool, it supports efficient (*i.e.,* it attempts to only send changes between file versions) file synchronisation between two directories (including sub folders) between two machines (or the same machine).

I decided to run some tests on using this program and the network I had set up to determine whether this would make a good base for my program or not.

I looked at three methods of files synchronisation across different networks. Naive copying, using rsync, an application for efficiently copying files in one direction by looking at the differences in the files, and unison described above.

– Insert graphs and diagrams here –

As you can see rsync and unison performed significantly better than the naive copy method (as I expected). After the initial file transfer subsequent edits to the file meant much less data had to be transmitted over the network which meant the node graph became up to date much more quickly.

4 Future work

4.1 Full node graph replication

I am going to be looking into the most efficient way to replicate data between many nodes in a graph, where nodes are machines running the program and edges in the graph are connections between machines such as over Wi-Fi,USB, or through the internet *etc.* Each connection may have different properties associated with it, for example each link may have a different cost associated with it, the two costs I am most interested in are data throughput and latency. There may be data caps to worry about (3G especially) or cost associated with usage (this could be money charged per megabyte used or the time cost of sending a lot of data over an (potentially) already busy/important channel). The link may be down temporarily or may rarely be up. The task here will be to find as near optimal algorithms as possible, given certain conditions and preferences, to shift data between all the necessary nodes efficiently and with minimal cost. Cost will most likely involve keeping the number of bytes passed over the network(s) to a minimum. Efficiency will depend on personal preference and the situation, ideally one would not want to have to be waiting on replication to occur but this should not come at the expense of extreme cost however. There will be an infinite number of use cases for this project which means an infinite number of graphs, however I will attempt to look at common use cases and as many different types of graph as possible.

4.2 Sub-Nodes

The other aspect of the multiple nodes in the graph problem is that each node will most likely be made up of many smaller nodes. Each user will be unlikely to select the root directory of the file system to synchronise which means they may select a few different directories on the file system. This gives us sub nodes that the main node is still the machine as above. The sub nodes are the individual directories that are set to be replicated. The reason this is interesting is because each of these sub nodes may have different synchronisation settings (see fine-grained controls). This leaves us with the possibility of machines being fully up to date, partially up to date, or completely out of date with all other nodes in the graph. An example of this potential situation is shown below. This sort of graph ties in with the

statistics/feedback side of the project. How out of date is the graph at any given time? If the graph contains out of date data, do we ever expect it to get back to being completely up to date? How long do we expect this to take? How do these facts reflect on the synchronisation settings we chose? Can we improve on the performance by tweaking the settings?

– insert sub node image –

4.3 Mobile nodes

Connections between nodes in the graph (edges) may change over time. This could be because one of the nodes is a laptop and joins different networks at different times or because a network/machine is unreliable and is not up at a given point in time. I have represented edges that behave in this way as grey on the diagram below. I will refer to nodes with grey edges as 'mobile' nodes

It will be interesting to see how mobile nodes effect how up to date the nodes in the graph are. We might expect that nodes that are not available for very long periods lag behind others that are. We might also expect that if a mobile node is a link between two parts of the network that these two parts fall out of sync a bit. I want to look at how nodes in the graph may get smarter about how they use unreliable edges. I will do this by having the edges up or down at different points in time and taking snapshots of the graph as it is at that time.

– insert mobile node image –

Say node two intermittently has a connection to a subset of nodes 1,3 and 4

4.4 More user control

Due to the highly unpredictable nature of the graph there should be a lot of room to tweak how the program runs. Users should be able to set what, when, where, and how when it comes to replicating their data. Which directories to replicate, how often to replicate it, where to replicate it to and any conditions on how the replication should occur. For example replicate my documents /Users/Calum/Documents on my laptop every hour to work but only when connected to the work Wi-Fi. There should be some intelligent indication of how certain options may affect performance which relates to the next section.

4.5 Feedback

Look into what statistics can be gleaned from running the program over a large period of time. Number of bits transferred total? Number of bits saved by using unison? How up to date is each part of the graph? How much is each edge (link) being used? Is this what we expect given the users preferences? What other statistics can we generate from the graph and the execution of the program? How can we display them in interesting / meaningful ways?

5 References and Related Links

inotify - www.kernel.org/pub/linux/kernel/people/rml/inotify/README