Private Dropbox
Final Report
COSC480

Calum O'Hare

Supervisor: David Eyers

1 Abstract

I have written a program in python which reads user settings from a file. Synchronises the appropriate files to the appropriate machines when they have been modified. Using an efficient two way file synchronising tool called unison. I will discuss in this paper what I have done and how I have tested it.

Contents

2 Introduction

2.1 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. However it should be possible to configure the system to be centralised if the user wants to. The system should be flexible in this regard.

- Allow file synchronisation between multiple clients not just point-to-point between two clients. Although still synchronise between two Clients as this is the basis for multiple client synchronisation. 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 (time taken for the files to become fully up to date), cost (bandwidth, disk space used). These statistics could also possibly lead to a heuristic for when to synchronise a given file.

- Operate automatically, without the user having to initiate a file synchronisation themselves. The user should be able to set when and where they would like synchronisation to occur.

2.2 Background

There are already many services available that synchronize your files. Dropbox, Google Drive, Microsoft SkyDrive, Apple iCloud 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 organisation with sensitive information this might be concerning. You also cannot 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.

For example Megaupload.com a file hosting service has recently been shut down by the United States Department of Justice for alleged copyright infringement. According to the founder, 100 million users lost access to 12 billion unique files[1].

There are other possible approaches to replicating files across multiple computers. For example you could use version control systems like Subversion, Mercurial, and CVS. One problem with these is that they are centralised, they rely on a central server should that server fail the replication will break. Not only that but they create a bottleneck at the server. Cloud based solutions are also often centralised. Another problem is that even if they are decentralised like git, they won't automatically push updates to other working sets.

Example use case

Here is how I would use such a tool as an 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 into 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 do not need it to be backed up very often as it does not change very much and I do not 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 that 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 smart phone. 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 that I move around are of a 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. For example my collection of large video files does not change that often and will chew up valuable network bandwidth whenever it has to transfer a new file. I like this to be replicated only occasionally as I do not use it that much. On the other hand my document collection which I use for work and coursework changes very often, is very important, and is fairly small. I would like this to be as up to date as possible.

An effective file synchronisation tool would be of great use to me personally. Dropbox does not do enough for me. It does not give me enough control over my data. I want to know which machines my files are going to and when. I want to feel confident that I will always be able to access my data even if Dropbox closes down or my internet connection dies.

The graph of a personal network has been described above, the graph of a corporate network is another example use case. It will have many of the same basic needs as the personal graph. The coloured rings represent the need for different policies for different machines in a network. Something which dropbox will not provide but private dropbox

(a) Personal Network

(b) Corporate Network
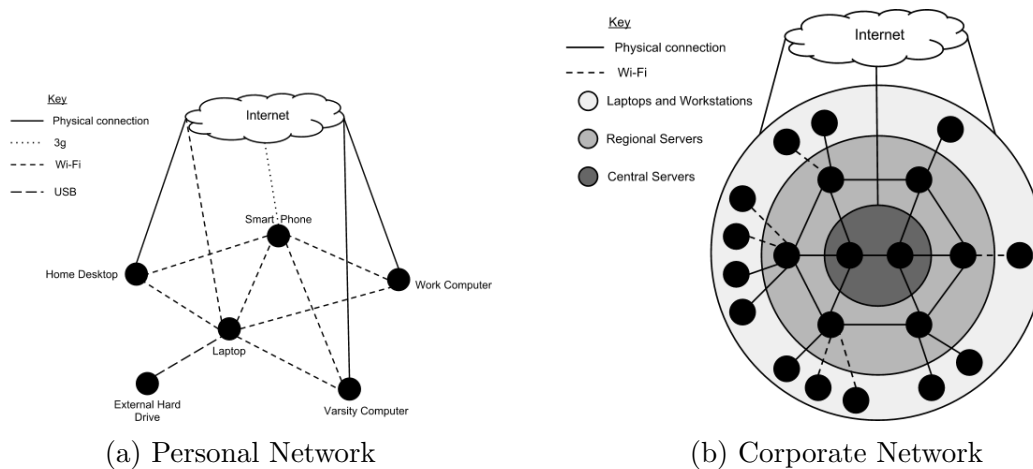
will.

3   Virtual Machines, Node networks

For testing my program I needed to have a network of computers that 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 a large number of physical machines. I can create new machines very easily, and manipulate the links between them.

I have used Oracle's VirtualBox software. I chose VirtualBox because of its easy to use command line interface. I have several scripts that 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 which network topology I would like to test.

I have decided to start testing my program with some simple topologies to see if I can gain any insight into how best to replicate data around a network with many nodes. The next step will be to use those principles and start running more complicated networks to see how the program performs.

Snippet from one of my network scripts:

```
VBoxManage modifyvm "Ubuntu-Test" --nic2 intnet
VBoxManage modifyvm "Ubuntu-Test" --intnet2 "intnet"
VBoxManage startvm "Ubuntu-Test"
```
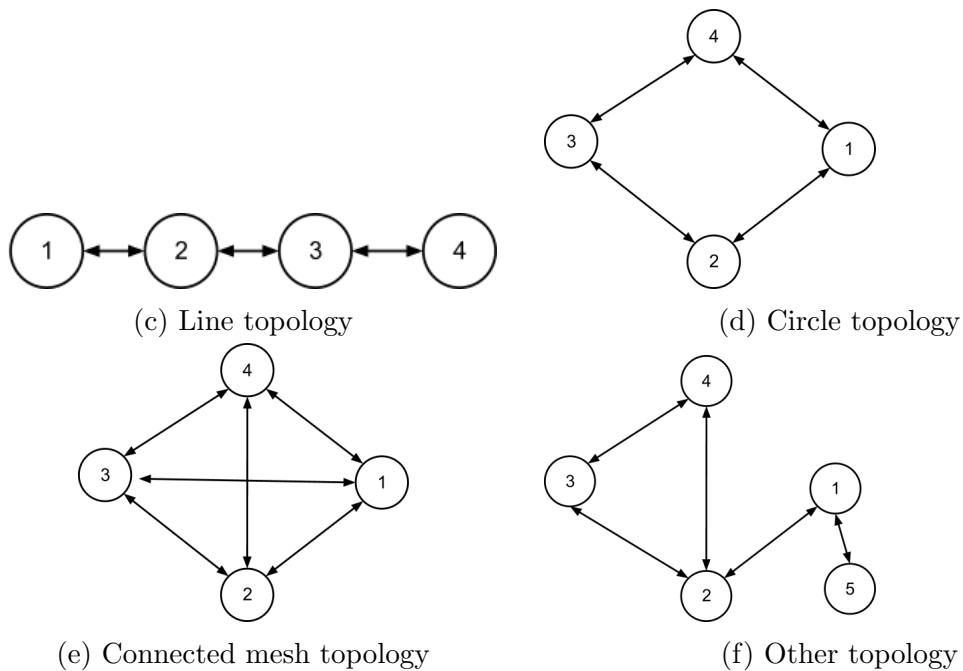
(c) Line topology

(d) Circle topology

(e) Connected mesh topology

(f) Other topology

Figure 1: Simple network topologies

4   Python

I have chosen to use Python to implement my program. Python appealed to me because it supports many different platforms (Windows, Linux, Mac 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.

5   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 program behaves. I currently have the program reading from configuration files that allow 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 large list of files may become unwieldy, and because I replicate directories recursively, I can replicate large amounts of data without a cluttered configuration file.

Another reason I chose directories as my granularity was because it may be handy to have a directory full of symlinks pointing to other directories.

6   Monitoring Directories

The application needs to monitor directories for changes so that it knows when to perform a sync. The reason I have chosen to do this is because synchronising 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 any changes made. This would be a significant 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

    - Inotify is a linux kernel feature that has been included in the Linux kernel since version 2.6. It 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. Dnotify however 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[2]. It works well and does what I need it to do. There is a Python module called pyinotify that provides a Python interface to inotify, which I have used and tested in my program. Another reason I chose inotify was because different kinds of changes triggered different inotify events. So I can differentiate between a file being deleted, created or modified *etc.*

- FSEvents

    - FSEvents is an API in MacOS X[3]. It is similar to inotify in that it provides a notification to other applications when a directory is changed however it does not inform 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. There is a Python module for FSEvents, as well.

        I also looked at using the `kqueue`[4] system call that 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 application's 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. There is a FileSystemWatcher Class in .NET version 4 and above. Iron-Python might prove to be a good choice for a Windows implementation.

I have chosen only to implement my program on linux because portability wasn't in the main scope of the project. It would have been nice to look at it further but became too time consuming and not interesting from a research perspective.

7 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 using unison 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 file synchronisation across different networks. Naive copying; using rsync, an application designed for efficiently copying files in one direction by looking at the differences in the files; and unison described above.
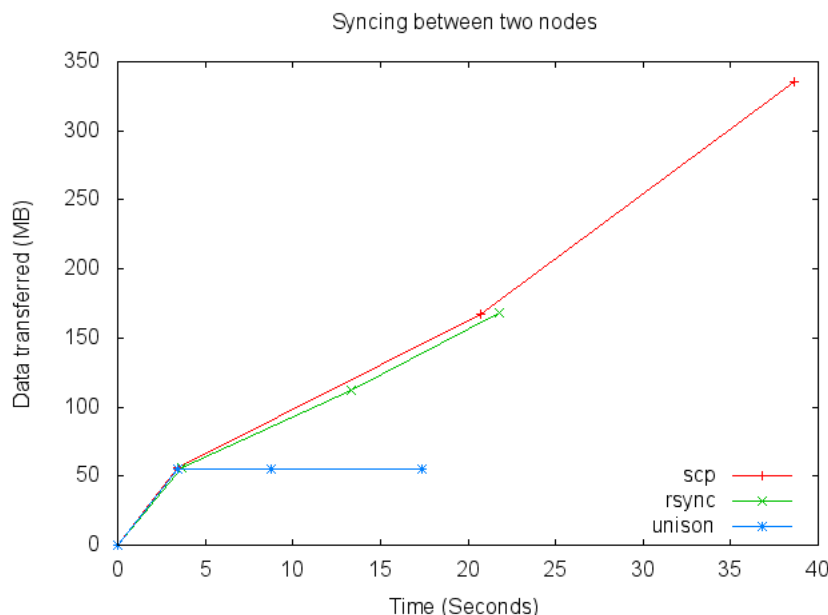

Figure 2: Comparison of scp,rsync,unison

Rsync and unison performed significantly better than the naive copy method (as 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.

The reason naive copy sent over 300Mb of data to copy three 50Mb files was because my implementation is deliberately naive, it will copy the entire directory each time it

is changed. For rsync and unison this is not a problem because they work based on the differences between the files. However copy doesn't look at the files it just copies everything in the directory tree. Hence it will copy 50Mb after file one is created, 100Mb after the second file is added and finally 150Mb when all three files are present.

50Mb + 100Mb + 150Mb = 300Mb

Rsync copies the expected 150Mb for three 50Mb files. While Figure 2 illustrates another advantage of unison over rysnc. The graph shows three zero filled binary files being copied from one node to another one after the other. Unison recognised that even though the files were named differently they were the same file. Another advantage of unison is that it handles replication in two directions without clobbering the files on the other side.

Each of the three methods I trialled had some overhead associated with them. This overhead was due to the ssh tunnel between the machines which all three methods used. Unison and rsync also require some overhead when checking the differences between the files in the directories. This is why the graph shows the three lines slightly above where you might expect them to be for the amount of data that was copied.
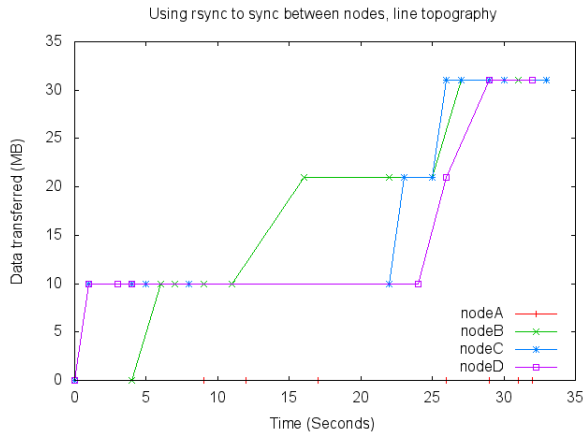
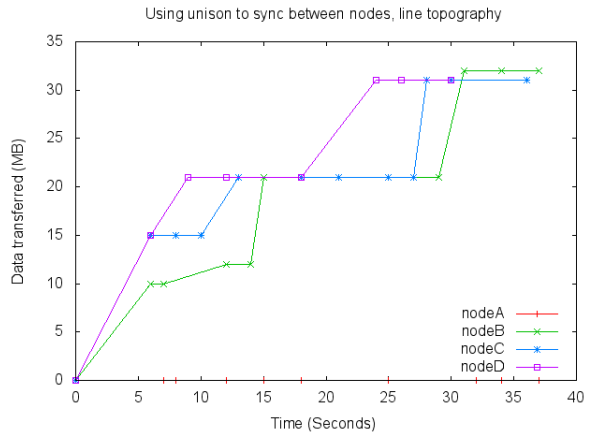# 8   Full graph replication
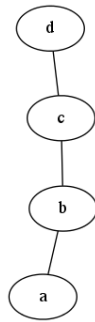
(a) Line - Generated Graph of Topology
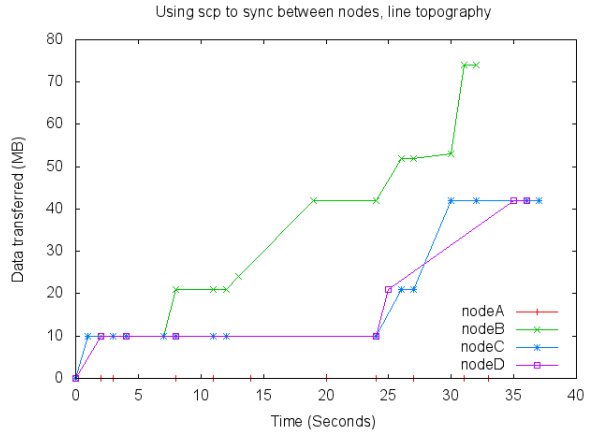
(b) SCP

(c) Rsync

(d) Unison

Figure 3: Comparison of methods over line topology

9 When to stop copying

After testing my program on some simple topologies one problem became clear. Each node would notice changes had occurred to a folder it was watching and would then try to copy these changes to other nodes that it was connected to. The problem was that if the changes came from one of its neighbour nodes this would cause an infinite loop of two nodes trying to copy changes to each other. This was particularly a problem when using scp to copy. When using Unison this was not as much of a problem because it could detect that no changes had occur between the nodes and would stop syncing after one check (which had minimal overhead).
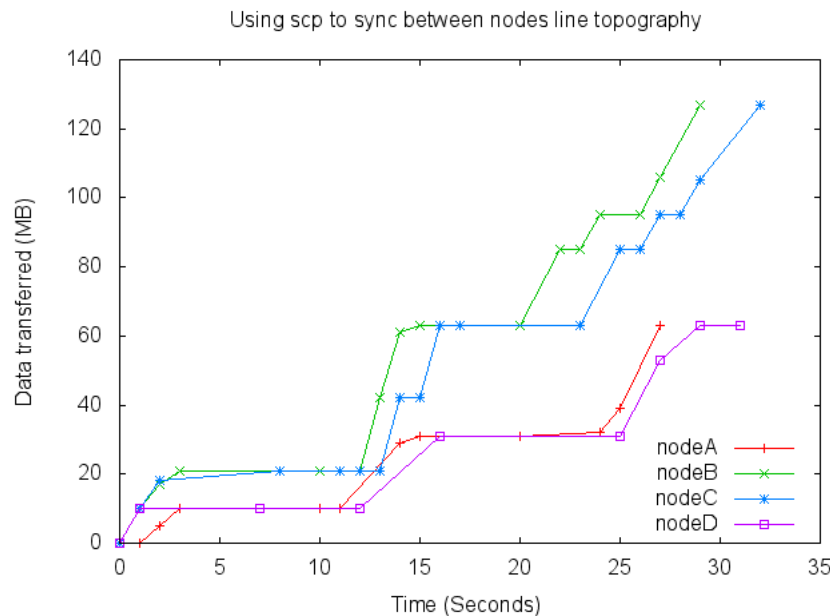
Figure 4: Line topology, using scp, nodes copying data back and forth

Figure 4 shows three 10Mb files being copied to a node in a line topology. The problem is that nodeB and nodeC continue to send data to each other even after every node has all of the files. NodeA receives a lot of data even though it was the source of the file changes.

The data points in figure 5 show that when using unison although no extra data was sent unison had to make checks to see whether there were any changes or not.

I used a configuration file to get around this problem. Each time a node synchronised with another node it would write out a configuration file telling the other node what files had been copied, who sent them and what the modification time of the files were. In this way a node could check if it was about to synchronise a file back to the node it received the file from or if it had local changes that were newer than a received file it could continue with its sync.

11
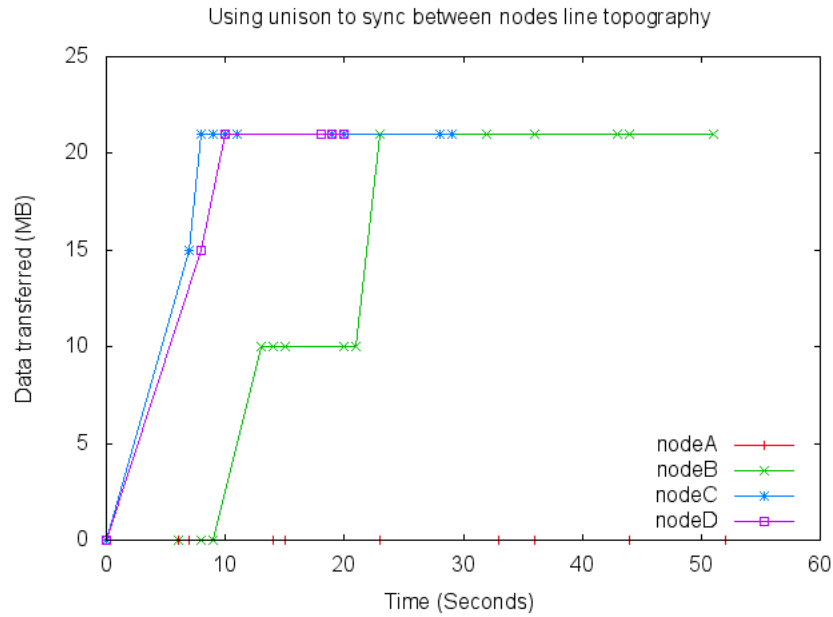
Figure 5: Line topology, using unison, unnecessary checking

10 Sub-nodes

I chose to classify directories as 'sub nodes' of a graph. The reason I choose directories is because they are easy to manage a configuration file of directories to keep in sync (from the users point of view). If we wanted to only synchronize certain files in a directory we could write a unison configuration file with exclusions/inclusions in it. The other reason directories are a good choice is because I can have different directories in different places on different file systems by using symbolic links. I wanted to see how the freshness of different sub-nodes varied between nodes when the program was running.
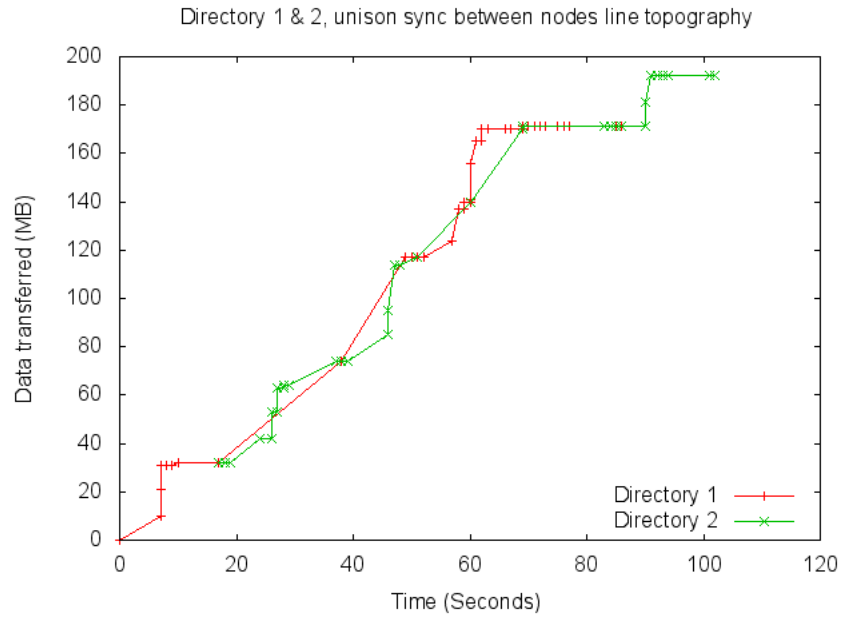
Figure 6: Line topology, using unison, two different directories being synced

11   How often to sync

So how often should I sync once I noticed a change. If lots of small changes are occurring frequently it might be more efficient to perform a synchronisation after several changes have occurred. Given that there is overhead with each synchronisation, fewer copies means less data sent over the network.
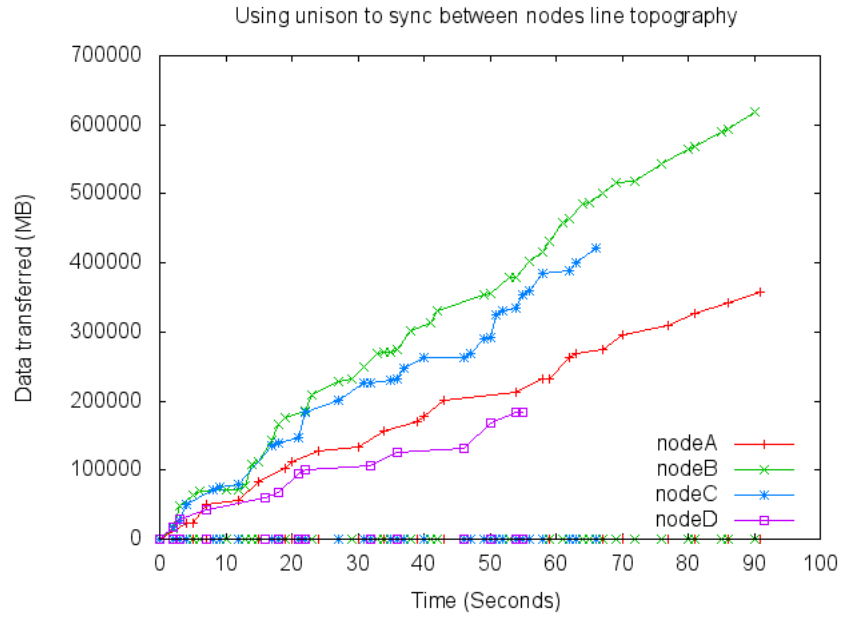
Figure 7: 2 seconds sleep text file
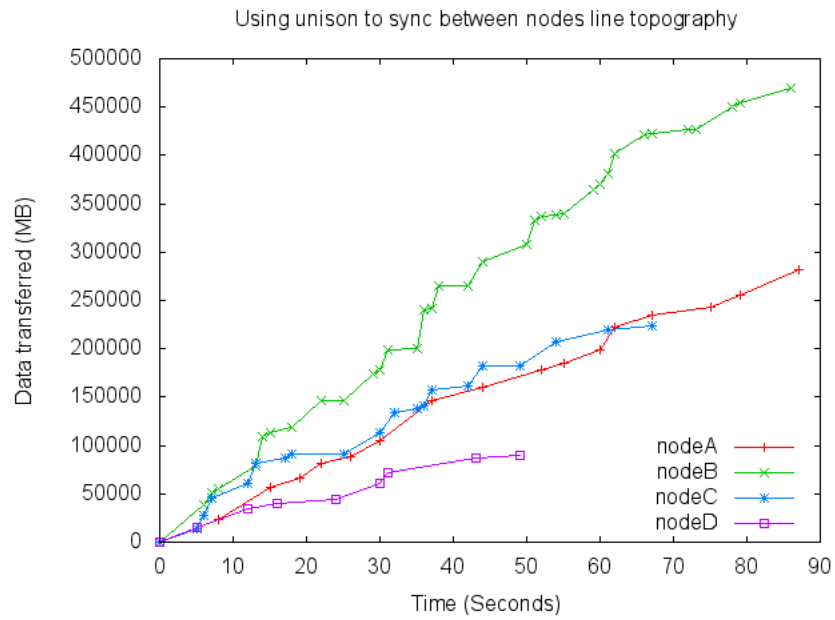
Figure 8: 5 seconds sleep text file

14

12 Unison and temporary files

I noticed that when Unison ran it created temporary files in the directory and once these files had been fully copied it renamed them to their intended name. The problem with this was that my program was picking up these temporary files as they were created and trying to copy them to the next node, only to find that these files no longer existed. To get around this problem I decided to implement a filter on the files to be copied. The program filters out files that contain ".tmp" in the filename. Unison is not the only program that uses temporary files. I decided that this should be a user set preference given that users may want to filter out different files.

My program simply reads from a file with each file pattern to exclude listed on a new line. It is easy to add to/remove from. As I said above I added .tmp to the file as a default. This could easily be extended to allow a user to omit certain files from the replication by adding all files in my programs ignore file to unisons ignore list. Or conversely by maintaining a white list of files to sync. This would allow for greater granularity when syncing nodes.

13 Results

14 Conclusion

References

[1] Foreman, Michael "Kim Dotcom v United States of America". Computerworld. 3 February 2012.

[2] www.kernel.org/pub/linux/kernel/people/rml/inotify/README, 22 September 2004.

[3] Apple Inc. `https://developer.apple.com/library/mac/#documentation/Darwin/Conceptual/FSEvents_ProgGuide/Introduction/Introduction.html`, 11 October 2011.

[4] Apple Inc. `http://developer.apple.com/library/mac/#documentation/Darwin/Reference/ManPages/man2/kqueue.2.html`

15   Bibliography

16   Glossary

17   Index

18   Appendices

Time taken for nodes to be up to date
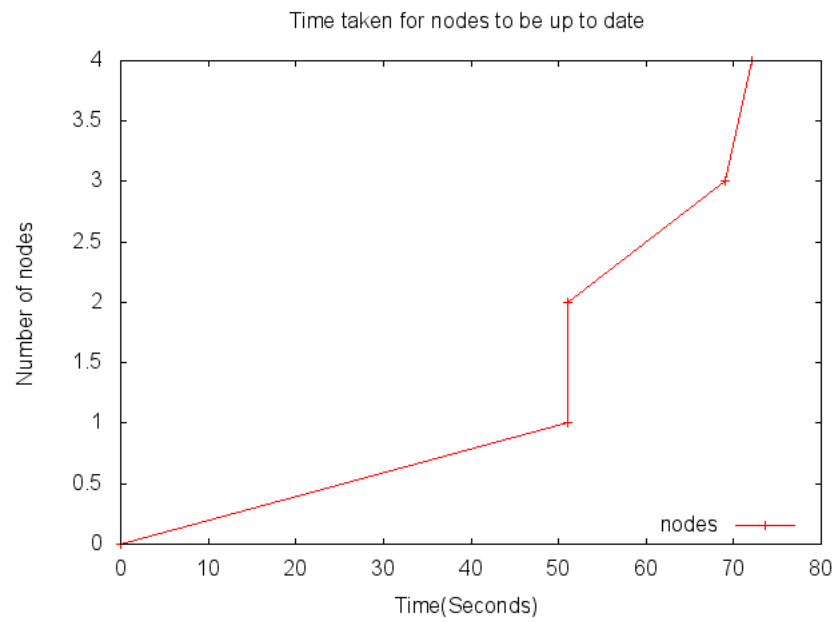
Figure 9: Unison, line, finishing times

WatchAndSync.py

```python
import pyinotify, os, subprocess, argparse, socket, time, glob, datetime
import readnet

wm = pyinotify.WatchManager()
watchedfolders = {}
homepath = "/home/cal/Documents/Private-Sync/"
#homepath = "/Users/calum/Documents/Private-Sync/"

parser = argparse.ArgumentParser()
parser.add_argument("-c","--scp",action="store_true",help="Copy using scp")
parser.add_argument("-r","--rsync",action="store_true",help="Copy using rsy
args = parser.parse_args()

class Tools():
    def updateFolderInfo(self, wfolds):
        f = open('./folders.dat','w')
        for fold in wfolds:
            f.write(fold + " ")
            for i in range(0,len(wfolds[fold])-1):
                f.write(wfolds[fold][i] + " ")
            f.write(wfolds[fold][len(wfolds[fold])-1] + "\n")
        f.close()

    def timeElapsed(self, dtstamp, diff):
        if diff == "*":
            print "Sync ASAP"
            return
        diff = int(diff)
        FMT = '%Y-%m-%d %H:%M:%S.%f'
        #FMT = '%Y-%m-%d %H:%M:%S'
        tdelta = datetime.datetime.now() - datetime.datetime.strptime(dtstan
        print   tdelta.total_seconds()
        timeDiff = tdelta.total_seconds()
        if (timeDiff >= diff):
            print "Time period reached"
        else:
            print "Time not elapsed, sleeping for " + str(diff - timeDiff +
            time.sleep(int(diff - timeDiff + 1))

class MyEventHandler(pyinotify.ProcessEvent):
    def flipIP(self,ip):
```

17

```python
        octets = ip.split(".")
        if(octets[3] == "1"):
            octets[3] = "2"
        elif(octets[3] == "2"):
            octets[3] = "1"
        else:
            octets[3] = "1"
        return ".".join(octets)


    #Get the last modified time of a file
    def getModTime(self,path):
        try:
            return time.ctime(os.path.getmtime(path))
        except Exception, e:
            return ""


    #Deprecated - Check for IP not to copy too
    def getStopInfo(self):
        stopIP = ["",""]
        try:
            o = open("./stop",'r')
            stopIP = o.read().split()
            o.close()
        except IOError, e:
            pass
        return stopIP


    def inStopFile(self,ip,path):
        stopIPs = {}
        stop = False
        modTime = self.getModTime(path)
        while True:
            tmpcount = 0
            print "Files found: " + str(glob.glob("Stop-*"))
            for files in glob.glob("Stop-*"):
                #print "File: " + str(files)
                if ".tmp" in files:
                    tmpcount += 1
                    time.sleep(5)
                    break
                f = open(files,"r");
                for line in f:
                    l = line.split()
```

18

```python
                if self.exclusions(l[1]):
                    print str(l[1]) + " was in ignore file skipping"
                else:
                    print "local " + str(path) + " modtime: " + modTime
                    print "Stop " + l[1] + " modtime: " + str(l[2:])
                    ts1 = time.strptime(modTime,"%a %b %d %H:%M:%S %Y")
                    ts2 = time.strptime(" ".join(l[2:]),"%a %b %d %H:%M
                    print "local <= stop: " + str(ts1 <= ts2)
                    #if l[0] == ip and l[1] == path and ts1 <= ts2:
                    #If IP sending to has sent data more recently don't
                    if l[0] == ip and ts1 <= ts2:
                        print "Stop = True, file: " + l[0]
                        stop = True
                    else:
                        stopIPs[l[0]] = [l[1]," ".join(l[2:])]

            if stop:
                f.close()
                #f = open(files,"w")
                #for k in stopIPs.keys():
                #    f.write(k + " " + stopIPs[k][0] + " " + stopIPs[k]
                #f.close()
                #stopIPs.clear()
                return True

            f.close()
            #stopIPs.clear()
        if tmpcount == 0:
            break

    return False

#Set flag on other server telling it not to immediately try and copy da
def setStopFileUniq(self,ip,myIP,path,folder):
    nodename = self.getNodeName()
    #print "ssh",ip,"echo " + myIP + " " + path + " " + self.getModTime
    #subprocess.call(["ssh",ip,"echo " + myIP + " " + path + " " + self
    subprocess.call(["ssh",ip,"rm " + homepath + "Stop-" + nodename + "
    for cpFile in glob.glob(folder + "/*"):
        subprocess.call(["ssh",ip,"echo " + myIP + " " + cpFile + " " +

#Sets the config files on the remote node
def beginCopy(self, ip):
```

```python
        nodename = self.getNodeName()
        print "ssh",ip,"touch " + homepath + "Stop-" + nodename + ".tmp; mv
        subprocess.call(["ssh",ip,"touch " + homepath + "Stop-" + nodename

#Moves the Stop files back into place
def endCopy(self, ip):
    nodename = self.getNodeName()
    print "ssh",ip,"mv " + homepath + "Stop-" + nodename + ".tmp " + ho
    subprocess.call(["ssh",ip,"mv " + homepath + "Stop-" + nodename + "

#Get node name from whoami file
def getNodeName(self):
    w = open(homepath + "whoami","r")
    nodename = w.read()
    nodename = nodename[0].upper()
    w.close()
    return nodename

#Deprecated stop file
def setStopFile(self,ip,myIP,path):
    subprocess.call(["ssh",ip,"echo " + myIP + " " + path + "> " + home
    print "ssh",ip,"echo " + myIP + "> " + homepath + "stop"

def rmTree(self,path):
    subprocess.call(["ssh",ip,"rm -r '" + path + "'"])
    print "ssh",ip,"rm -r '" + path + "'"

#Exclude files matching patterns in the ignore file
def exclusions(self, path):
    try:
        f = open("./ignore",'r')
        for line in f:
            if line.rstrip() in path:
                #print "Ignoring: " + path
                return True
        f.close()
    except error, e:
        print e
    return False

def initFileSync(self,event):
    if self.exclusions(event.pathname):
        #print "Excluded returning"
```

20

```
            return
        pathparts = event.pathname.split("/")
        foldName = "/".join(pathparts[0:len(pathparts)−1])
        print "Removing watch on: " + foldName
       wm.rm_watch(wm.get_wd(foldName), rec=True)
        self.fileSync(event)
        print "Putting watch back on: " + foldName
       wm.add_watch(foldName.rstrip(),pyinotify.ALL_EVENTS, rec=True, auto

   #Sync files
   def fileSync(self,event):
        t = Tools()
        if os.path.isdir(event.pathname):
            print "Watching: ",event.pathname
        for folder in watchedfolders.keys():
            print "For each folder: " + str(folder) + " in watchedfolder ke
            if folder in event.pathname:
                for i in range(0, len(watchedfolders[folder]),4):
                    ip = watchedfolders[folder][i]
                    path = watchedfolders[folder][i+1]
                    waitTime = watchedfolders[folder][i+2]
                    lastTime = watchedfolders[folder][i+3]
                    print "Wait: " + str(waitTime) + " Last: " + str(lastTim
                    print "Current ip and path: " + ip + " " + path
                    readnet.logIPtraffic(ip, event.pathname)
                    myIP = readnet.getMyIP(ip)
                    subprocess.call(["ssh",ip,"/usr/bin/python " + homepath
                    print "ssh",ip,"'/usr/bin/python " + homepath + "readne
                    fparts = folder.split("/")
                    fname = fparts[len(fparts)−1]
                    #stopIP = self.getStopInfo()
                    #print "STOP: " + stopIP[0] + " " + stopIP[1]
                    #if stopIP[0] == ip and stopIP[1] == event.pathname:
                    if self.inStopFile(ip,event.pathname):
                        print "STOPPED to " + ip + " " + path
                        #os.remove("./stop");
                    else:
                        print "CONTINUE"
                        t.timeElapsed(lastTime, waitTime)
                        watchedfolders[folder][i+3] = str(datetime.datetime
                        t.updateFolderInfo(watchedfolders)
                        self.beginCopy(ip)
                        if args.scp:
```

21

```python
                                    #print "SCP: For cpFile in " + folder
                                    for cpFile in glob.glob(folder + "/*"):
                                        #print "SCP GLOB:" + cpFile
                                        print "scp","-rp",cpFile,ip + ":" + cpFile
                                        subprocess.call(["scp","-rp",cpFile,ip + ":"
                                        #subprocess.call(["ssh",ip,"yes y | find /tr
                                        print "ssh",ip,"mv " + cpFile + ".tmp " + c
                                        subprocess.call(["ssh",ip,"mv " + cpFile +
                                        print "END SCP GLOB"
                                elif args.rsync:
                                    print "rsync","-rt",folder,ip + ":" + path
                                    subprocess.call(["rsync","-rt",folder,ip + ":"
                                else:
                                    time.sleep(5)
                                    print "unison","-batch","-confirmbigdel=false",
                                    subprocess.call(["unison","-batch","-confirmbig
                            print "Set stop files uniq: " + event.pathname
                            self.setStopFileUniq(ip,myIP,event.pathname,folder)
                            self.endCopy(ip)
                        subprocess.call(["ssh",ip,"/usr/bin/python " + homepath
                        readnet.logIPtraffic(ip, event.pathname)


    #def process_IN_CREATE(self, event):
    #    print "Create:",event.pathname
    def process_IN_DELETE(self, event):
        print "Delete: ",event.pathname
        #self.initFileSync(event)
    def process_IN_CREATE(self, event):
        print "CREATE: ",event.pathname
        self.initFileSync(event)
    def process_IN_MOVED_FROM(self, event):
        print "Move from: ",event.pathname
    #    self.initFileSync(event)
    def process_IN_MODIFY(self, event):
        #print "Modify: ",event.pathname
        self.initFileSync(event)
    def process_IN_MOVED_TO(self, event):
        print "Move to: ",event.pathname
        self.initFileSync(event)


def main():
    t = Tools()
```

```python
f = open('./folderstowatch','r')

for folder in f:
    if(folder[0] == '#'):
        pass
    else:
        info = folder.split()
        wm.add_watch(info[0].rstrip(), pyinotify.ALL_EVENTS, rec=True, a
        print "Watching: ", info[0].rstrip()
        if info[0] not in watchedfolders.keys():
            watchedfolders[info[0].rstrip()] = []
        watchedfolders[info[0].rstrip()].append(info[1])
        watchedfolders[info[0].rstrip()].append(info[2])
        watchedfolders[info[0].rstrip()].append(info[3])
        watchedfolders[info[0].rstrip()].append(str(datetime.datetime.n
f.close()

try:
    f = open('./folders.dat','r')
    for folder in f:
        if(folder[0] == '#'):
            pass
        else:
            info = folder.split()
            if info[0] in watchedfolders.keys():
                del watchedfolders[info[0].rstrip()]
                #wm.add_watch(info[0].rstrip(), pyinotify.ALL_EVENTS, re
                #print "Watching: ", info[0].rstrip()
                if info[0] not in watchedfolders.keys():
                    watchedfolders[info[0].rstrip()] = []
                watchedfolders[info[0].rstrip()].append(info[1])
                watchedfolders[info[0].rstrip()].append(info[2])
                watchedfolders[info[0].rstrip()].append(info[3])
                watchedfolders[info[0].rstrip()].append(str(datetime.da
            else:
                print "Removing: " + info[0]
    f.close()
except IOError, e:
    print "Folders.dat does not exist, skipping"

t.updateFolderInfo(watchedfolders)

#print watchedfolders
```

```python
    eh = MyEventHandler()

    notifier = pyinotify.Notifier(wm, eh)
    notifier.loop()

if __name__ == '__main__':
    main()
```

ReadNet.py

```python
import subprocess, datetime, socket, argparse

homepath = "/home/cal/Documents/Private-Sync/"
#homepath = "/Users/calum/Documents/Private-Sync/"

parser = argparse.ArgumentParser()
parser.add_argument('-i', action="store", dest='ip', help='IP address to recor
parser.add_argument('-f', action="store", dest='fold', help='Folder to record

interfacenames = []

w = open(homepath + "whoami","r")
nodename = w.read()
nodename = nodename[0]
w.close()

#Get my ip corresponding to the interface with ipaddr
def getMyIP(ipaddr):
    route = subprocess.check_output("ip route get " + ipaddr, shell=True)
    words = route.split()
    interface = ""
    for word in words:
        if word.startswith("eth"):
            interface = word
            #print interface
            break
    ifconf = subprocess.check_output("ifconfig " + interface, shell=True)
    words = ifconf.split()
    now = False
    for word in words:
        if word == "inet":
            now = True
        elif now:
            word = word.split(":")
            #print word[1]
            return word[1]

#Log interface coresponding to ipaddr
def logIPtraffic(ipaddr, folder):
    route = subprocess.check_output("ip route get " + ipaddr, shell=True)
    words = route.split()
```

25

```python
        interface = ""
        for word in words:
            if word.startswith("eth"):
                interface = word
                #print interface
                break
        writeIface(interface, folder)

def writeIface(iface, folder):
    ifs = subprocess.check_output("ifconfig -s", shell=True)
    ilines = ifs.split("\n")
    for i in range(1,len(ilines)-1):
        interfacenames.append(ilines[i].split()[0])
    output = subprocess.check_output("ifconfig", shell=True)
    splitput = output.split()
    interface = False
    interfacename = ""
    nex = ""
    count = 0
    upload = 0
    download = 0
    for split in splitput:
        if split in interfacenames:
            interface = True
            interfacename = split
            #print interfacename
        if(nex != ""):
            sp = split.split(":")
            if(sp[0] == "bytes"):
                if(nex == "RX"):
                    download = int(sp[1])
                else:
                    upload = int(sp[1])
                nex = ""
                count += 1
                if(count == 2):
                    interface = False
                    if interfacename == iface:
                        f = open(homepath + "log/" \
                        + "node" + nodename.upper() + "-" \
                        + iface + ".log",'a')
                        f.write("#D " + folder + "\n")
                        f.write(str(datetime.datetime.now()) + " " + interfa
```

26

```python
                    f.close()
                count = 0
        elif(interface):
            if(split == "RX" or split == "TX"):
                nex = split


#Log all interfaces
def main():
    ifs = subprocess.check_output("ifconfig -s",shell=True)
    ilines = ifs.split("\n")
    for i in range(1,len(ilines)-1):
        interfacenames.append(ilines[i].split()[0])
    output = subprocess.check_output("ifconfig",shell=True)
    splitput = output.split()
    interface = False
    interfacename = ""
    nex = ""
    count = 0
    upload = 0
    download = 0
    for split in splitput:
        if split in interfacenames:
            interface = True
            interfacename = split
            #print interfacename
        if(nex != ""):
            sp = split.split(":")
            if(sp[0] == "bytes"):
                if(nex == "RX"):
                    download = int(sp[1])
                else:
                    upload = int(sp[1])
                nex = ""
                count += 1
                if(count == 2):
                    interface = False
                    f = open(homepath + "log/" \
                    + str(socket.gethostname()) + "-" \
                    + interfacename + ".log",'a')
                    f.write(str(datetime.datetime.now()) + " " + interfacena
                    f.close()
                    count = 0
        elif(interface):
```

27

```python
            if(split == "RX" or split == "TX"):
                nex = split

if __name__ == "__main__":
    args = parser.parse_args()
    if args.ip != None:
        logIPtraffic(args.ip, args.fold)
        #getMyIP(args.ip)
    else:
        pass
        main()
```

onTheFly.sh

```
vm_name_arr=("Ubuntu-Pool" "Ubuntu-Silence" "Ubuntu-Wild" "Ubuntu-Spheros")
vm_addr_arr=("192.168.0.28" "192.168.0.27" "192.168.0.19" "192.168.0.14")
intnetarr=("lion" "tiger" "cat" "dog" "fish" "kiwi")
#These should all be in one big dictionary apart from inet names
letterarr=("a" "b" "c" "d" "e" "f" "g")
ifcountarr=(2 2 2 2 2 2 2 2 2)
ethcountarr=(1 1 1 1 1 1 1 1 1)
incount=1
bigncount=2
littlencount=1
folderpath="/home/cal/Documents/t03"
folderpath2="/home/cal/Documents/t02"
homepath="/home/cal/Documents/Private-Sync/"
waitTime=10

function clear_ifaces() {
    i=0
    while [ "$i" -lt "${#vm_name_arr[@]}" ]; do
        VBoxManage modifyvm ${vm_name_arr[$i]} --nic2 none
        echo "VBoxManage modifyvm ${vm_name_arr[$i]} --nic2 none"
        VBoxManage modifyvm ${vm_name_arr[$i]} --nic3 none
        echo "VBoxManage modifyvm ${vm_name_arr[$i]} --nic3 none"
        VBoxManage modifyvm ${vm_name_arr[$i]} --nic4 none
        echo "VBoxManage modifyvm ${vm_name_arr[$i]} --nic4 none"
        let "i++"
    done
}

function clear_watched_folders() {
    i=0
    while [ "$i" -lt "${#vm_addr_arr[@]}" ]; do
        ssh cal@${vm_addr_arr[$i]} "echo \"#Local folder path to watch, hos
        let "i++"
    done
}

function git_pull() {
    i=0
    while [ "$i" -lt "${#vm_addr_arr[@]}" ]; do
        echo "ssh cal@${vm_addr_arr[$i]} \"cd /home/cal/Documents/Private-S
        ssh cal@${vm_addr_arr[$i]} "cd /home/cal/Documents/Private-Sync; gi
```

```
            let "i++"
        done
}

function search_letters() {
    index=0
    while [ "$index" -lt "${#letterarr[@]}" ]; do
        if [ "${letterarr[$index]}" = "$1" ]; then
            echo $index
            return
        fi
        let "index++"
    done
    echo "None"
}

function vbmMOD {
    echo "VBoxManage modifyvm $1 --nic$3 intnet"
    VBoxManage modifyvm $1 --nic$3 intnet
    echo "VBoxManage modifyvm $1 --intnet$3 $2"
    VBoxManage modifyvm $1 --intnet$3 $2
}

function gatherLogs {
    index=0
    while [ "$index" -lt "${#vm_addr_arr[@]}" ]; do
        echo "scp cal@${vm_addr_arr[$index]}:/home/cal/Documents/Private-Sy
        scp cal@${vm_addr_arr[$index]}:/home/cal/Documents/Private-Sync/log
        let "index++"
    done
}

function clean {
    index=0
    while [ "$index" -lt "${#vm_addr_arr[@]}" ]; do
        echo "ssh cal@${vm_addr_arr[$index]} \"rm ${homepath}log/*; rm ${ho
        ssh cal@${vm_addr_arr[$index]} "rm ${homepath}log/*; rm ${homepath}
        let "index++"
    done
}

function cleanFold {
    index=0
```

```bash
        while [ "$index" -lt "${#vm_addr_arr[@]}" ]; do
            echo "ssh cal@${vm_addr_arr[$index]} \"rm -rf ${folderpath}/*;\""
            ssh cal@${vm_addr_arr[$index]} "rm -rf ${folderpath}/*;"
            let "index++"
        done
}

function sendKeys {
    index=0
    while [ "$index" -lt "${#vm_addr_arr[@]}" ]; do
        #ssh cal@${vm_addr_arr[$index]} "rm /home/cal/.ssh/authorized_keys"
        for file in /Users/calum/.ssh/*.pub; do
            #echo "$file"
            echo "cat $file | ssh cal@${vm_addr_arr[$index]} \"cat >> /home
            cat $file | ssh cal@${vm_addr_arr[$index]} "cat >> /home/cal/.s
        done
        let "index++"
    done
    #for file in /Users/calum/.ssh/*.pub; do
    #    echo "$file"
    #    cat $file | ssh cal@192.168.0.17 "cat >> /home/cal/.ssh/testfile"
    #    echo "cat $file | ssh cal@192.168.0.17 \"cat >> /home/cal/.ssh/tes
    #done
}

function ifconf {
    echo "ssh cal@$1 'sudo /sbin/ifconfig eth$2 192.168.$3.$4 netmask 255.2
    ssh cal@$1 "sudo /sbin/ifconfig eth$2 192.168.$3.$4 netmask 255.255.255
}

function ifconf2 {
    echo "ssh cal@$1 \"sudo /sbin/ifconfig eth$2 192.168.$3.$4 netmask 255.
    ssh cal@$1 "sudo /sbin/ifconfig eth$2 192.168.$3.$4 netmask 255.255.255
}

if [ $2 == "vm" ]; then
    clear_ifaces

    while read line
    do
        first=$(echo "$line" | awk '{print $1}')
        last=$(echo "$line" | awk '{print $(NF)}' | sed 's/[;]//g')
        #echo "$first and $last"
```

```bash
        index=$(search_letters $first)
        if [ "$index" = "None" ]; then
            #echo "None"
            :
        else
            vbmMOD ${vm_name_arr[$index]} ${intnetarr[$incount]} ${ifcounta
            #echo "in: $index"
            (( ifcountarr[$index]++ ))
            index=$(search_letters $last)
            vbmMOD ${vm_name_arr[$index]} ${intnetarr[$incount]} ${ifcounta
            #echo "in: $index"
            (( ifcountarr[$index]++ ))
            incount=$incount+1
        fi
    done <graphs/$1
elif [ $2 == "if" ]; then
    clear_watched_folders

    while read line
    do
        first=$(echo "$line" | awk '{print $1}')
        last=$(echo "$line" | awk '{print $(NF)}' | sed 's/[;]//g')
        echo "$first and $last"
        index=$(search_letters $first)
        if [ "$index" = "None" ]; then
            #echo "None"
            :
        else
            ifconf ${vm_addr_arr[$index]} ${ethcountarr[$index]} $bigncount
            #echo "in: $index"
            (( ethcountarr[$index]++ ))
            (( littlencount++ ))
            index=$(search_letters $last)
            ifconf ${vm_addr_arr[$index]} ${ethcountarr[$index]} $bigncount
            #echo "in: $index"
            (( ethcountarr[$index]++ ))
            incount=$incount+1
            (( bigncount++ ))
            (( littlencount-- ))
        fi
    done <graphs/$1
elif [ $2 == "if2" ]; then
    clear_watched_folders
```

```
    while read line
    do
        first=$(echo "$line" | awk '{print $1}')
        last=$(echo "$line" | awk '{print $(NF)}' | sed 's/[;]//g')
        echo "$first and $last"
        index=$(search_letters $first)
        if [ "$index" = "None" ]; then
            #echo "None"
            :
        else
            ifconf2 ${vm_addr_arr[$index]} ${ethcountarr[$index]} $bigncoun
            #echo "in: $index"
            (( ethcountarr[$index]++ ))
            (( littlencount++ ))
            index=$(search_letters $last)
            ifconf2 ${vm_addr_arr[$index]} ${ethcountarr[$index]} $bigncoun
            #echo "in: $index"
            (( ethcountarr[$index]++ ))
            incount=$incount+1
            (( bigncount++ ))
            (( littlencount-- ))
        fi
    done <graphs/$1
elif [ $2 == "key" ]; then
    sendKeys
elif [ $2 == "gather" ]; then
    gatherLogs
elif [ $2 == "clean" ]; then
    clean
elif [ $2 == "pull" ]; then
    git_pull
elif [ $2 == "clean-fold" ]; then
    cleanFold
elif [ $2 == "help" ]; then
    echo "vm          - setup vm networking"
    echo "if          - setup network addresses etc for each vm"
    echo "if2         - setup network addresses etc for each vm for two fol
    echo "gather      - gather the logs in"
    echo "clean       - clean out the logs/config files"
    echo "clean-fold  - clean out the files folder"
    echo "pull        - pull the latest code from the repository to each vm
    echo "help        - display this help message"
```

```
else
    echo "Oops try again"
fi

neato -Tpng graphs/$1 > graphs/$1-graph.png
```