CS654 : Assignment 2

Pankaj More

Y9227402

Failures

1. Process dies either due to some bug in the code or killed by some other process
2. The VM fails by crashing or getting killed by a host process
3. The Host which is running the VM fails either due to power failure or a system crash
4. The file is deleted or corrupted by some other process
5. The network between host and turing goes down
6. The hardisk storing the file fails or turing goes down

Failure Detection

1. Any failure due to a logical bug which does not crash the program needs to be detected manually by the programmer. If the bug leads to a program crash , it will give inconsistent results. Proper exception handling might help in detecting such bugs. In case the program gets killed , a process monitor can be used to detect such an event.
2. If the VM gets killed, we can simply monitor the VM process as in the above case and detect a VM crash.
3. If the host goes down , we can use the heartbeat method discussed in class to detect it. We can continuously ping the host to check if it is alive. When there is no response , the host must have gone down.
4. A system call to check if the file exists can detect file deletion. File corruption can be checked by comparing the state of the file at the end of the last write and the state of read in the next iteration. If these 2 states are different , the file has been corrupted.

5. If the network goes down, our counter process will throw an exception while open sftp connection to turing.

6. Harddisk failures or turing crash is not under our control and the responsibility of its uptime might be delegated to CSE Lab Admins.

Recovery from Failures

1. Manual intervention in case of logical bugs. In case the process gets killed, there are 2 ways for recovering from a process crash.

   i. Process Migration/Failover - Using a heartbeat approach , a backup process possible on a different vm running on a different node can detect the death of the process and take over as the primary process. Any state of the process need to continuously mirrored in real time to the backup process. This would also require additional backup processes in case the new process also dies.

   ii. Process Restart - A much simpler way is to just restart the process using another process which monitors our counter process. But the obvious flaw would be what if this monitor gets killed. No worries! Process Supervision is built into linux. Using the kernel level supervisors such as init or upstart , it would be impossible to kill these process monitors without crashing the system. A SIGTERM would simply cause them to re-exec. What the means is if the process has not restarted by Upstart, the VM must have crashed. We don't need to persist the state of the process in this scenario. The state can retrieved from the file assuming it does not get corrupted during the small time window while the process is being restarted(usually less than a sec).

2. We can manage the VM process similarly using Upstart. This would guarantee that if the VM crashes , it will simply be re-spawned. The trade-off here is the time taken to spawn the VM which can lead to a downtime of a few seconds(which is quite reasonable in such a non-critical system). Also since we are not storing the state information of our counter process, if the file gets corrupted during downtime , we wont be able to start the counter from its old value.

3. If the host goes down , we can perform a live migration of the VM to another node and make that node the primary host.Implementing this using open-source apis to virtualBox is quite non-trivial. A very simple solution would be to monitor the host and then notify the system-administrator to restart the host server manually in case of a crash. In case we want to use of-the-shelf automatic solutions, VMWare has proprietary services to take care of live migration and automatic-failover during host crash.

4. In normal situations file deletion can be handled by appropriate File::Open calls which creates the file if necessary. In case file is corrupted during uptime, we can restore it from the memory variable. If corruption occurs during downtime, no recovery would be possible due to lack of state information. But the time window of such an event is very small. We will restart from 0.

5. Our program can detect network failures and notify us by mail that there is some problem with the network. Then proper action can be taken to bring the network up.

Implementation Details

The counter process counter.rb running inside a VM handles file corruption, file deletion, network failures as discussed above. Process termination is handled by an Upstart daemon running inside the VM. VM termination is handled by a corresponding upstart job running on the host OS. Host failure can be checked by a simple ping application which mails the admin to restart the host.

How to Run

1. Create a ubuntu-server 12.04 virtual machine using Virtualbox

2. Install ruby, gem , ruby-bundler on the VM and do a bundle install inside the code repository to download the required ruby gems : > sudo apt-get install ruby, rubygems, ruby-bundler; sudo bundle install

3. Set the HOME path inside counter.conf to the folder where counter.rb is to be deployed.

4. Deploy the counter.rb script to the home folder inside the VM, counter.conf to /etc/init/ inside the vm, countingMachine.conf to /etc/init/ on the host. Also give permissions to counter.rb. > chmod a+x ./counter.rb

5. Power-off the VM and start countingMachine service on host : > sudo start countingMachine

6. Check the file on the cse host and test various possible failure scenarios discussed above