

How to install Cotson

Cotson was made to run in Unix environment, mainly in Debian and AMD machine. Before installing Cotson install Debian and certificate that the machine has the minimum requisites to run Coston.

- 1- Install SimNow in the PC. This is the link were you can find <http://developer.amd.com/cpu/simnow/Pages/default.aspx>
- 2- Install Cotson in PC. Please download the version for development
Put in the prompt the following command to install Cotson:

```
svn co https://cotson.svn.sourceforge.net/svnroot/cotson/trunk cotson
```

- 3- ./configure --simnow (the address where is installed simnow)
- 4- Make

Running experiments on Cotson

There are 8 examples that you can use to see cotson running.

The examples are inside cotson/src/examples.

There is two way to run the examples. If the example is described in the Makefile, to run is needed to write on the prompt:

```
#make run_exemplename
```

The Makefile has some instructions to call cotson and the script file. The makefile is responsible to lunch the example on coston.

```
../bin/cotson twonodes.in > run_twonodes.log 2>&1 ;
```

This line is calling coston and the example of two_nodes and saving data in run_twonodes.log.

The other way to run the example is by calling in the prompt cotson and the example. This is done when the example is not inside the make file. The user should call the example in the following way:

```
#../bin/cotson example_name.in
```

As coston uses some graphic to show the results, make sure that the graphic environment is working. Otherwise examples which shows the simnow will not work properly

Examples of Cotson

Two-nodes(two_nodes.in)

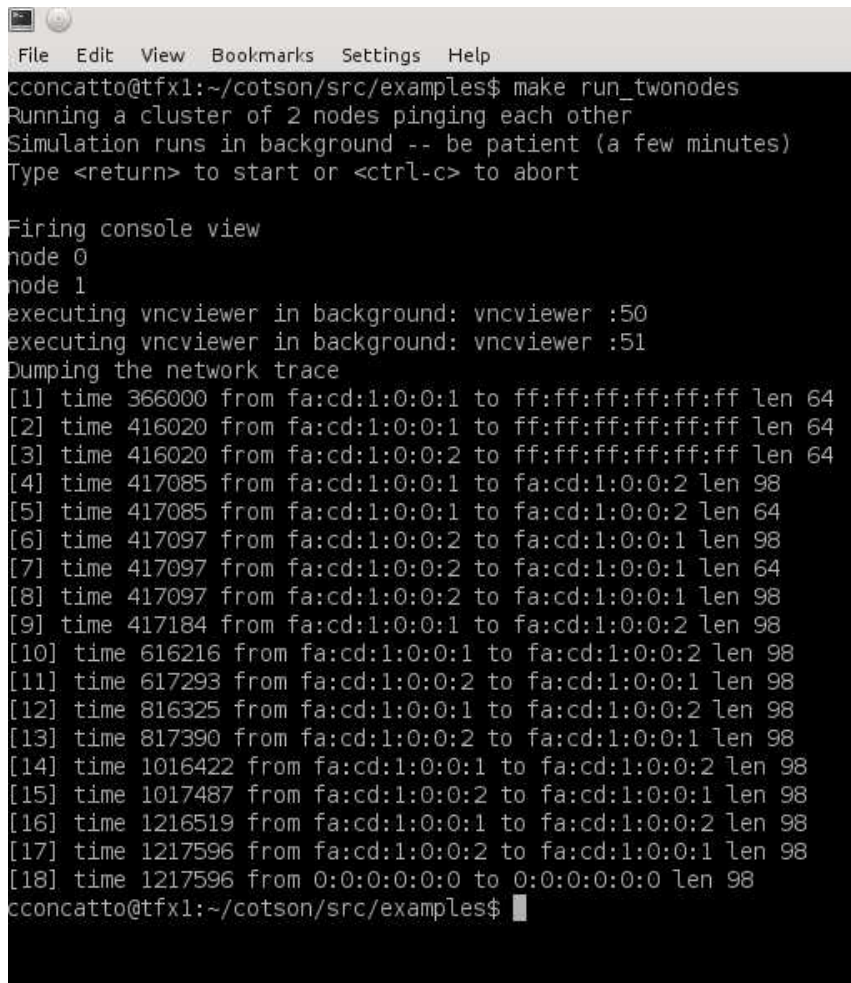
If you look inside `twonodes.in` you will see that besides the typical build, options, and `simnow.commands`, there is also a mediator table and plenty of individual commands floating around. Basically are there to instruct cotson of how to run the two instances of SimNow and the mediator and how to hook them up together to form a cluster simulation. Nevertheless, in order to run a cluster simulation you also need to slightly modify the `simnow.commands` function. Notice how instead of using `send_keyboard` function we are using the 'execute' function. This function instructs cotson to load the `ping.sh` script into each node and execute it there. Since this node must work for each node, when running it, it is passed two parameters, the node id (running from 1 to N) and the total amount of nodes. As you can see in the `ping.sh` script, the first node basically pings the second one while the other one does nothing (just waits for those pings). The `simnow.command` also has two new functions, `set_network` and `subscribe_result`. The function 'set_network' instructs cotson to prepare the particular bsd to have network. The function 'subscribe_result' will let you describe as many elements as you wish, and it will be responsible of gathering all that output files for you and placing them outside the virtual environment once the simulation has finished.

```
simnow.commands=function()  
    use_bsd('1p.bsd')  
    use_hdd('karmic64.img')  
    set_journal()  
    set_network() -- this is responsible to call the network  
    execute('ping.sh') --executes the shell  
    subscribe_result('/tmp/ping.log') -- store the information here  
end
```

```
MCAST=os.date('%d')+os.date('%H')+os.date('%M')+os.date('%S')  
mediator = {  
    multicast_ip="239.200.1"..MCAST,  
    quantum_min=10,  
    quantum_max=1000,  
    heartbeat_interval=10000, -- 10msec  
    heartbeat={ type="file_last" },  
    heartbeat1={ type="sqlite", dbfile=DBFILE, experiment_id=EXP },  
    tracefile="/tmp/net-trace.gz", -- dump a trace  
    slirp=false, -- don't NAT with the external world  
}
```

Figure 1 shows the prompt after running `two-nodes` as one can see, it shows the number of nodes and the ping activity that node0 did on node1. In this example we did 10 pings, that is 10 experiments. Figure 2 shows the `two-nodes` examples running, it shows the GUI of SimNow and inside shows a node doing ping in other node, and others just waiting. The example

generates some log files at the end. It is possible to check all features used to run the example, as MAC address, number of experiments, errors, and others.



```
cconcatto@tfx1:~/cotson/src/examples$ make run_twonodes
Running a cluster of 2 nodes pinging each other
Simulation runs in background -- be patient (a few minutes)
Type <return> to start or <ctrl-c> to abort

Firing console view
node 0
node 1
executing vncviewer in background: vncviewer :50
executing vncviewer in background: vncviewer :51
Dumping the network trace
[1] time 366000 from fa:cd:1:0:0:1 to ff:ff:ff:ff:ff:ff len 64
[2] time 416020 from fa:cd:1:0:0:1 to ff:ff:ff:ff:ff:ff len 64
[3] time 416020 from fa:cd:1:0:0:2 to ff:ff:ff:ff:ff:ff len 64
[4] time 417085 from fa:cd:1:0:0:1 to fa:cd:1:0:0:2 len 98
[5] time 417085 from fa:cd:1:0:0:1 to fa:cd:1:0:0:2 len 64
[6] time 417097 from fa:cd:1:0:0:2 to fa:cd:1:0:0:1 len 98
[7] time 417097 from fa:cd:1:0:0:2 to fa:cd:1:0:0:1 len 64
[8] time 417097 from fa:cd:1:0:0:2 to fa:cd:1:0:0:1 len 98
[9] time 417184 from fa:cd:1:0:0:1 to fa:cd:1:0:0:2 len 98
[10] time 616216 from fa:cd:1:0:0:1 to fa:cd:1:0:0:2 len 98
[11] time 617293 from fa:cd:1:0:0:2 to fa:cd:1:0:0:1 len 98
[12] time 816325 from fa:cd:1:0:0:1 to fa:cd:1:0:0:2 len 98
[13] time 817390 from fa:cd:1:0:0:2 to fa:cd:1:0:0:1 len 98
[14] time 1016422 from fa:cd:1:0:0:1 to fa:cd:1:0:0:2 len 98
[15] time 1017487 from fa:cd:1:0:0:2 to fa:cd:1:0:0:1 len 98
[16] time 1216519 from fa:cd:1:0:0:1 to fa:cd:1:0:0:2 len 98
[17] time 1217596 from fa:cd:1:0:0:2 to fa:cd:1:0:0:1 len 98
[18] time 1217596 from 0:0:0:0:0:0 to 0:0:0:0:0:0 len 98
cconcatto@tfx1:~/cotson/src/examples$
```

Figure 1: Report of Two-nodes example.

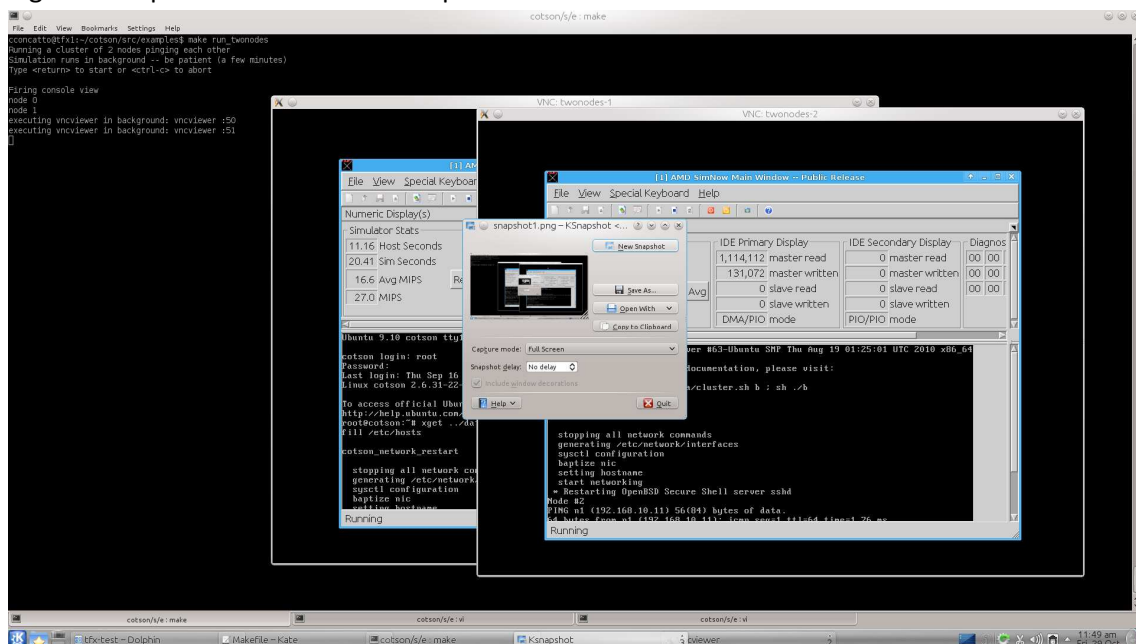


Figure 2: Two-nodes running.

Mem_tarcer

All load/store in the main memory is saved in a DB file. At the end informs the IPC of the application

```
1 exec> keyboard.key 16 96
1 exec> keyboard.key 1F 9F
1 exec> keyboard.key 12 92
1 exec> keyboard.key 13 93
1 exec> keyboard.key 35 B5
1 exec> keyboard.key 14 94
1 exec> keyboard.key 12 92
1 exec> keyboard.key 1F 9F
1 exec> keyboard.key 14 94
1 exec> keyboard.key 34 B4
1 exec> keyboard.key 17 97
1 exec> keyboard.key 1C 9C

1 exec> go
TIME=0.0625 ms IPC ( 0.33881 )
TIME=3.1875 ms IPC ( 1 )
TIME=6.3125 ms IPC ( 1 )
TIME=9.4375 ms IPC ( 1 )
TIME=12.5625 ms IPC ( 1 )
MAX SAMPLES: 5
cconcatto@tfx1:~/cotson/src/examples$
cconcatto@tfx1:~/cotson/src/examples$
```

Figure 3: Report of Mem_trace

Sqlite (sqlite.in)

It is a dump from a data based. It shows the simnow running and after graphs with the IPC and data packets. Figure 4 shows the results of dumping the data from data base and figure 5 is the report given by the prompt when running sqlite. The sqlite does not use mediator, the data is from one simnow instance.

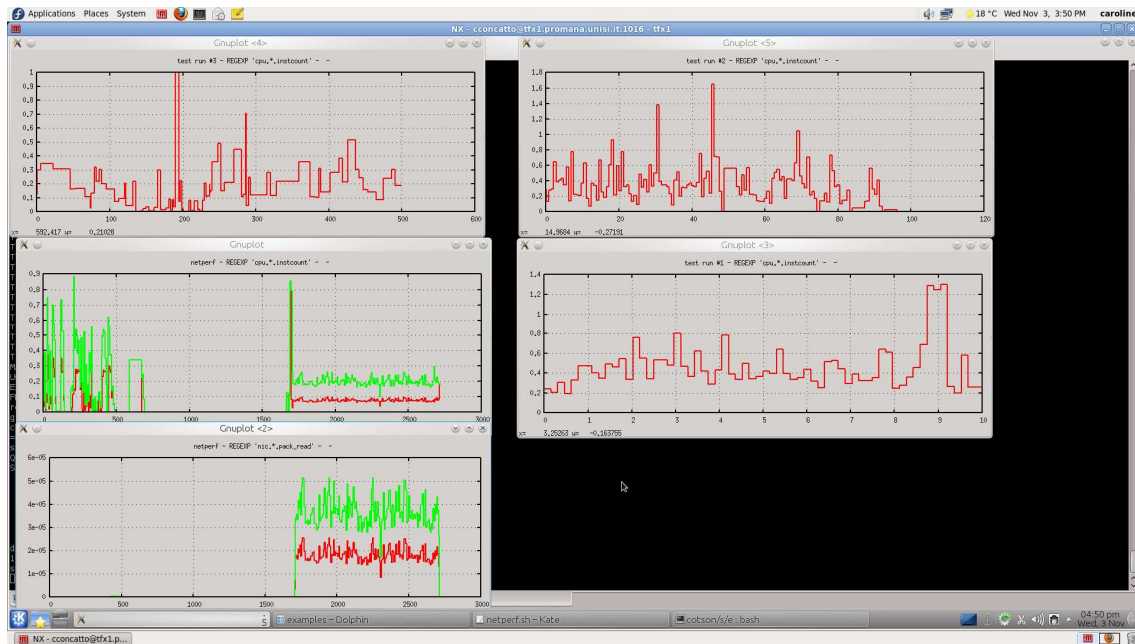


Figure 4: Result of Sqlite
Functional

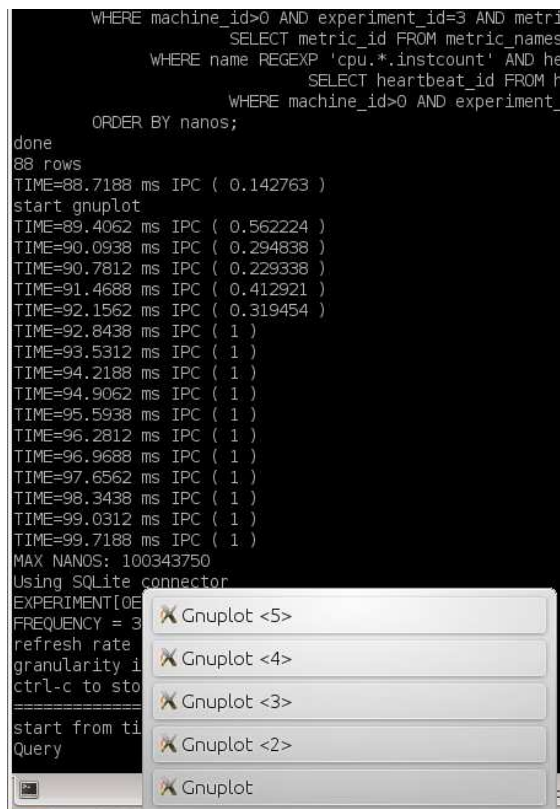


Figure 5: Report of Sqlite

Net_perf(net_perf.in)

It is also another program that generates graphics from data base. It shows the IPC and the number of packets of the application running on the VM.

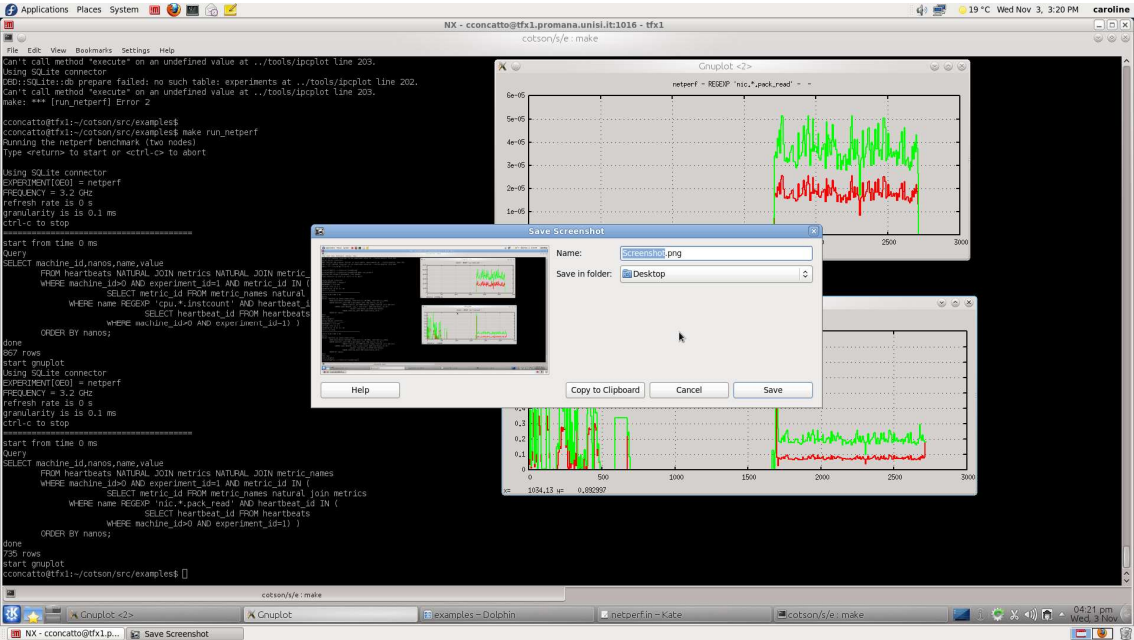


Figure 6: Report of net_perf example

Multifunctional_cores

To link the mediator in case of problems

In -s ../network/mediator