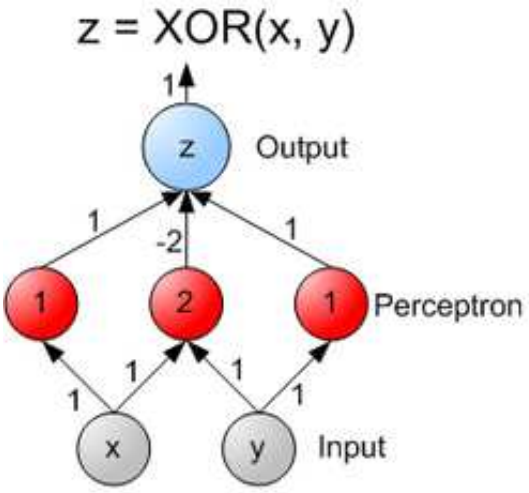
IoT as Brain ¶

Wei Lin

Abstract

Regarding concurrent/distributed computing, IoT, microservices, serverless-computing are all well known paradigms. With MQTT, IBM Bluemix is one of the leaders in this field, however, free of charge only for 30 days.

With Celery and Docker-Swarm, we can actually build a private IoT system in minutes, free of any charge. To provide an example, a distributed system using Celery and Docker-Swarm modeling a XOR neural-network will be explained below.



Steps:

1. Establish a Docker Swarm:

With reference to the article "Let Docker Swarm all over your Raspberry Pi Cluster (http://blog.hypriot.com/post/let-docker-swarm-all-over-your-raspberry-pi-cluster/)", a Docker-Swarm had been built upon two Raspberry Pi machines.

The swarm is composited fo two Docker machines:

- host rpi202(192.168.0.114) as Swarm Manager，the Docker machine name is "master01".
- host rpi201(192.168.0.109) as Swarm Node，the Docker machine name is "node01".

In [ ]:

```
HypriotOS: pi@rpi202 in ~
$ docker-machine ls
NAME       ACTIVE   DRIVER    STATE     URL                         SWARM
master01            hypriot   Running   tcp://192.168.0.114:2376   master01 (master)
node01              hypriot   Running   tcp://192.168.0.109:2376   master01
HypriotOS: pi@rpi202 in ~
$


# Nodes in the swarm:

HypriotOS: pi@rpi202 in ~
$ docker $(docker-machine config --swarm master01) info
Containers: 4
Images: 51
Role: primary
Strategy: spread
Filters: health, port, dependency, affinity, constraint
Nodes: 2
 master01: 192.168.0.114:2376
  └ Status: Healthy
  └ Containers: 3
  └ Reserved CPUs: 0 / 4
  └ Reserved Memory: 0 B / 972 MiB
  └ Labels: executiondriver=native-0.2, kernelversion=4.1.8-hypriotos-v7+, operatings
 node01: 192.168.0.109:2376
  └ Status: Healthy
  └ Containers: 1
  └ Reserved CPUs: 0 / 4
  └ Reserved Memory: 0 B / 972 MiB
  └ Labels: executiondriver=native-0.2, kernelversion=4.1.8-hypriotos-v7+, operatings
CPUs: 8
Total Memory: 1.899 GiB
Name: b7def5d9af98
HypriotOS: pi@rpi202 in ~
$
```

2. Copy celeryconfig.py、start_workers.sh、and the folder "IoT" to two hosts, under the folder of /data/celery_projects.

In [ ]:

```
# Swarm manager

HypriotOS: pi@rpi202 in /data/celery_projects
$ ll
total 20
drwxr-xr-x 3 999 root 4096 Jan 28 10:08 ./
drwxr-xr-x 3 999 root 4096 Jan 28 11:02 ../
-rw-r--r-- 1 999 root 1469 Jan 28 10:48 celeryconfig.py
drwxr-xr-x 3 999 root 4096 Jan 28 10:08 IoT/
-rwxr-xr-x 1 999 root  963 Jan 28 10:28 start_workers.sh*   <-- script to start up wo
HypriotOS: pi@rpi202 in /data/celery_projects
$


# Swarm node
HypriotOS: pi@rpi201 in /data/celery_projects
$ ll
total 16
drwxr-xr-x 3 root root 4096 Jan 28 12:54 ./
drwxr-xr-x 3  999 root 4096 Jan 25 22:55 ../
-rw-r--r-- 1 root root 1250 Jan 28 11:27 celeryconfig.py
drwxr-xr-x 3 root root 4096 Jan 28 12:54 IoT/
HypriotOS: pi@rpi201 in /data/celery_projects
$
```

Content of start_workers.sh

In [ ]:

```
# ./start_workers.sh

PROJECT='IoT'  # project name
CONCURRENCY=1  # number of subprocesses per worker.



echo "Starting Redis, Flower ____________________________________________________"
eval $(docker-machine env master01)

docker run -dit -p 6379:6379 --name=redis -v /data:/data hypriot/rpi-redis
docker run -d -p 5555:5555 --name=flower --volume=/data/celery_projects:/celery_proje



echo "Starting Celery cluster containers ____________________________________________
eval $(docker-machine env --swarm master01)

for id in 'x' 'y' 'h1' 'h2' 'h3' 'z'
do
  docker run -d --name=neuron_${id} --hostname=neuron_${id} --volume=/data/celery_pro
done
```

Execute start_workers.sh，will do followings:

- Establish broker for Celery, using Redis container.
- Establish Flower container for monitoring.
- Through Swarm Manager, establish Celery worker containers which stand for neurons.

In [ ]:

```
HypriotOS: pi@rpi202 in /data/celery_projects
$ ./start_workers.sh
Starting Redis, Flower ____________________________________________________
cb706da89689211601b931e88921df1564c939a2cdb3de7bda0f4fa878424553
e136f3f443a46b1a1082b26d367c6c146327d54a3eb9f16aa907dd48bce38a47
Starting Celery cluster containers ________________________________________________________
684e3d7b84bfa4713a972d434507473d33adcbaad092e32518a291f7e095c86f
8608740a5a86977f82dc2943feb315575a2ca9e38ebb1a2c73842567c87a865d
1b5180f0284c8c2aa723c63b4297e57fd35cc5a2c0b1ef5dded3bc3026202f61
a6679a9bb651dc735725ecd7b4793f27d598efdd4179c008afaae1e4322b0a42
3a59323ae8f61e76e80595a67e98293c0d42f96b1ac47205aff48d863b321aba
6c0d6a8bb590961e9947bf1d15587f1c0017b114a466fc7061d2fe888740026e
HypriotOS: pi@rpi202 in /data/celery_projects
$
```

There are 6 neurons:

neuron x, y, z, h1, h2 are deployed onto Swarm node "node01'，

In [ ]:

```
HypriotOS: pi@rpi201 in ~
$ docker ps
CONTAINER ID        IMAGE                     COMMAND                  CREATED
6c0d6a8bb590        wei1234c/celery_armv7   "/bin/sh -c 'cd /cele"   About a minute a
a6679a9bb651        wei1234c/celery_armv7   "/bin/sh -c 'cd /cele"   About a minute a
1b5180f0284c        wei1234c/celery_armv7   "/bin/sh -c 'cd /cele"   About a minute a
8608740a5a86        wei1234c/celery_armv7   "/bin/sh -c 'cd /cele"   About a minute a
684e3d7b84bf        wei1234c/celery_armv7   "/bin/sh -c 'cd /cele"   2 minutes ago
ef0c519ae7da        hypriot/rpi-swarm       "/swarm join --advert"   4 minutes ago
HypriotOS: pi@rpi201 in ~
$
```

```
neuron h3 is deployed onto Swarm node "master01'，
```

In [ ]:

```
HypriotOS: pi@rpi202 in /data/celery_projects
$ docker ps
CONTAINER ID        IMAGE                          COMMAND                  CREATED
3a59323ae8f6        wei1234c/celery_armv7          "/bin/sh -c 'cd /cele"   35 seconds ag
e136f3f443a4        wei1234c/celery_armv7          "/bin/sh -c 'cd /cele"   About a minut
cb706da89689        hypriot/rpi-redis              "/entrypoint.sh redis"   About a minut
966928d0a37c        hypriot/rpi-swarm              "/swarm join --advert"   3 minutes ago
b01b05cbe323        hypriot/rpi-swarm              "/swarm manage --tlsv"   4 minutes ago
ab78ab3e5476        nimblestratus/rpi-consul       "/bin/start -server -"   4 minutes ago
HypriotOS: pi@rpi202 in /data/celery_projects
$
```

The whole picture from the view point of Swarm Manager.

In [ ]:

```
HypriotOS: pi@rpi202 in /data/celery_projects
$ docker $(docker-machine config --swarm master01) ps
CONTAINER ID        IMAGE                          COMMAND                  CREATED
6c0d6a8bb590        wei1234c/celery_armv7          "/bin/sh -c 'cd /cele"   2 minutes ago
3a59323ae8f6        wei1234c/celery_armv7          "/bin/sh -c 'cd /cele"   2 minutes ago
a6679a9bb651        wei1234c/celery_armv7          "/bin/sh -c 'cd /cele"   2 minutes ago
1b5180f0284c        wei1234c/celery_armv7          "/bin/sh -c 'cd /cele"   2 minutes ago
8608740a5a86        wei1234c/celery_armv7          "/bin/sh -c 'cd /cele"   3 minutes ago
684e3d7b84bf        wei1234c/celery_armv7          "/bin/sh -c 'cd /cele"   3 minutes ago
e136f3f443a4        wei1234c/celery_armv7          "/bin/sh -c 'cd /cele"   3 minutes ago
cb706da89689        hypriot/rpi-redis              "/entrypoint.sh redis"   3 minutes ago
ef0c519ae7da        hypriot/rpi-swarm              "/swarm join --advert"   5 minutes ago
966928d0a37c        hypriot/rpi-swarm              "/swarm join --advert"   5 minutes ago
b01b05cbe323        hypriot/rpi-swarm              "/swarm manage --tlsv"   5 minutes ago
ab78ab3e5476        nimblestratus/rpi-consul       "/bin/start -server -"   6 minutes ago
HypriotOS: pi@rpi202 in /data/celery_projects
$
```

Begin to config the neural network.

In [38]:

```
from IoT.neuron import *
from time import sleep
import pandas as pd
from pandas import DataFrame

pd.options.display.max_colwidth = 400
REFRACTORY_PERIOD = 0.1   # 0.1 seconds
```

In [39]:

```
# There are 6 neurons, each one is represented by a Docker container, deployed random
neurons = ['neuron_x', 'neuron_y', 'neuron_h1', 'neuron_h2', 'neuron_h3', 'neuron_z']

# print log from a neuron.
def printConfig(neuron):
    print('{0:_^78}\n {1}\n'.format(neuron + " config:", getConfig.apply_async(routin

# clear logs in all neurons.
def emptyLogs():
    for neuron in neurons:
        emptyLog.apply_async(routing_key = neuron)

# merge all logs from all neurons, into a Pandas.DataFrame
def mergeLogs():
    logs = []

    for neuron in neurons:
        currentLog = getLog.apply_async(routing_key = neuron).get()
        logs += currentLog

    df = DataFrame(list(logs), columns = ['time', 'neuron', 'message'])
    df.set_index('time', inplace = True)
    df.sort_index(inplace = True)

    return df
```

Clear log files

In [40]:

```
# clear all logs in all neurons.
emptyLogs()
```

Setup connections

setup the connections between neurons, same as setting up the publisher / subscriber relationships in queues.

In [41]:

```
# input layer fan out
# neuron x
addConnection.apply_async(['neuron_h1'], routing_key = 'neuron_x')  # add neuron_x ->
addConnection.apply_async(['neuron_h2'], routing_key = 'neuron_x')
# neuron y
addConnection.apply_async(['neuron_h2'], routing_key = 'neuron_y')  # add neuron_y ->
addConnection.apply_async(['neuron_h3'], routing_key = 'neuron_y')

# hidden layer fan out
addConnection.apply_async(['neuron_z'], routing_key = 'neuron_h1')  # add neuron_h1 -
addConnection.apply_async(['neuron_z'], routing_key = 'neuron_h2')
addConnection.apply_async(['neuron_z'], routing_key = 'neuron_h3')
```

Out[41]:

```
<AsyncResult: 2c476123-b85e-4a50-8b31-56ab64cb0758>
```

Setup weights

In [42]:

```
# hidden layer
setWeight.apply_async(['neuron_x', 1], routing_key = 'neuron_h1')  # set neuron_x ->
setWeight.apply_async(['neuron_x', 1], routing_key = 'neuron_h2')
setWeight.apply_async(['neuron_y', 1], routing_key = 'neuron_h2')  # set neuron_y ->
setWeight.apply_async(['neuron_y', 1], routing_key = 'neuron_h3')

# output layer
setWeight.apply_async(['neuron_h1', 1], routing_key = 'neuron_z')
setWeight.apply_async(['neuron_h2', -2], routing_key = 'neuron_z')  # set neuron_h2 -
setWeight.apply_async(['neuron_h3', 1], routing_key = 'neuron_z')
```

Out[42]:

```
<AsyncResult: ca9824e0-05af-4b55-bf5e-3d0b097388ed>
```

Setup thresholds

In [43]:

```
# input layer
setThreshold.apply_async([0.9], routing_key = 'neuron_x')  # set neuron_x threshold =
setThreshold.apply_async([0.9], routing_key = 'neuron_y')

# hidden layer
setThreshold.apply_async([0.9], routing_key = 'neuron_h1')
setThreshold.apply_async([1.9], routing_key = 'neuron_h2')  # set neuron_h2 threshold
setThreshold.apply_async([0.9], routing_key = 'neuron_h3')

# output layer
setThreshold.apply_async([0.9], routing_key = 'neuron_z')  # set neuron_z threshold =
```

Out[43]:

```
<AsyncResult: 9f2e95ef-72a9-4d10-81a6-9ff41aed1b2d>
```

Simulating sensor input，then check the output status of each neurons.

note: the output of a neuron will cease after 5 seconds if no sustaining input received.

In [44]:

```
### simulating sensor input，force neuron x or neuron y to ouput 1(firing)
emptyLogs()  # clear logs
sleep(REFRACTORY_PERIOD)  # wait for output to cease.
mergeLogs()  # merge logs
```

Out[44]:

	neuron	message
time		

In [45]:

```
### simulating sensor input，force neuron x or neuron y to ouput 1(firing)
emptyLogs()  # clear logs
sleep(REFRACTORY_PERIOD)  # wait for output to cease.
fire.apply_async(routing_key = 'neuron_x') # force neuron x output 1 and fire.
mergeLogs()  # clear logs
```

Out[45]:

	neuron	message
time		
2016-03-10 20:51:18.489509	neuron_x	neuron_x fires.
2016-03-10 20:51:18.493309	neuron_x	Setting output of neuron_x to ACTION_POTENTIAL.
2016-03-10 20:51:18.559939	neuron_h1	neuron_x is kicking neuron_h1.
2016-03-10 20:51:18.581558	neuron_h2	neuron_x is kicking neuron_h2.
2016-03-10 20:51:18.587120	neuron_h1	neuron_h1 fires.
2016-03-10 20:51:18.595436	neuron_h1	Setting output of neuron_h1 to ACTION_POTENTIAL.
2016-03-10 20:51:18.654499	neuron_z	neuron_h1 is kicking neuron_z.
2016-03-10 20:51:18.689138	neuron_z	neuron_z fires.
2016-03-10 20:51:18.692448	neuron_z	Setting output of neuron_z to ACTION_POTENTIAL.

In [46]:

```
### simulating sensor input，force neuron x or neuron y to ouput 1(firing)
emptyLogs()  # clear logs
sleep(REFRACTORY_PERIOD)  # wait for output to cease.
fire.apply_async(routing_key = 'neuron_y') # force neuron y output 1 and fire.
mergeLogs()  # merge logs
```

Out[46]:

	neuron	message
time		
2016-03-10 20:51:21.721563	neuron_y	neuron_y fires.
2016-03-10 20:51:21.726665	neuron_y	Setting output of neuron_y to ACTION_POTENTIAL.
2016-03-10 20:51:21.796071	neuron_h3	neuron_y is kicking neuron_h3.
2016-03-10 20:51:21.818246	neuron_h3	neuron_h3 fires.
2016-03-10 20:51:21.822734	neuron_h3	Setting output of neuron_h3 to ACTION_POTENTIAL.
2016-03-10 20:51:21.858226	neuron_h2	neuron_y is kicking neuron_h2.
2016-03-10 20:51:21.899541	neuron_z	neuron_h3 is kicking neuron_z.
2016-03-10 20:51:21.922727	neuron_z	neuron_z fires.
2016-03-10 20:51:21.927111	neuron_z	Setting output of neuron_z to ACTION_POTENTIAL.

In [47]:

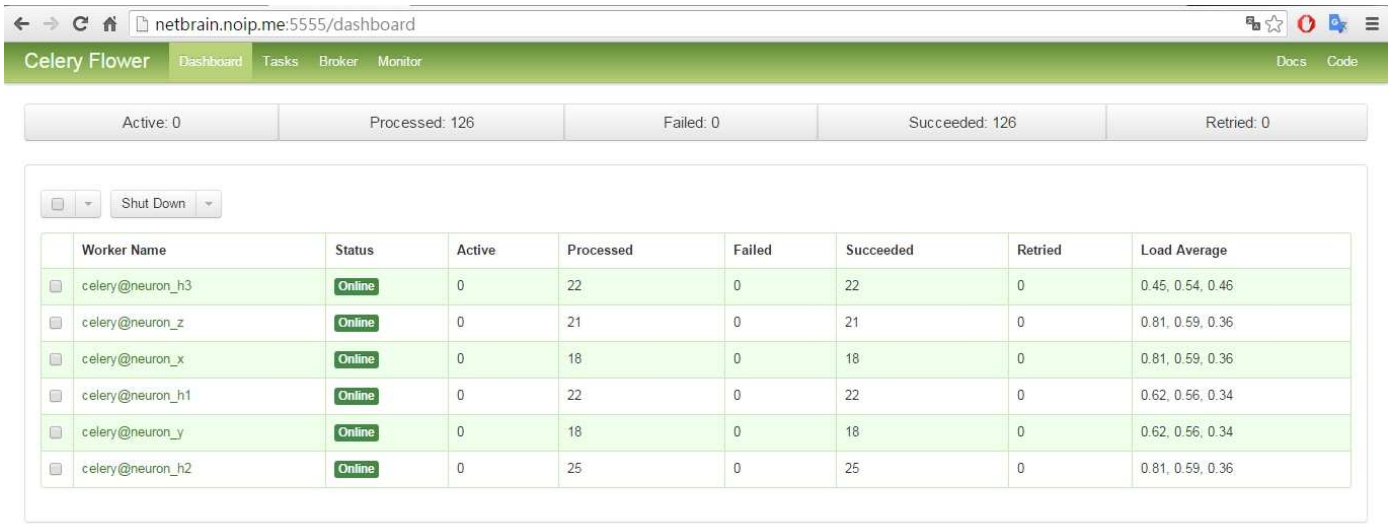
```
### simulating sensor input，force neuron x or neuron y to ouput 1(firing)
emptyLogs()  # clear logs
sleep(REFRACTORY_PERIOD)  # wait for output to cease.
fire.apply_async(routing_key = 'neuron_x') # force neuron x output 1 and fire.
fire.apply_async(routing_key = 'neuron_y') # force neuron y output 1 and fire.
mergeLogs()  # merge logs
```

Out[47]:

	neuron	message
time		
2016-03-10 20:51:25.524156	neuron_x	neuron_x fires.
2016-03-10 20:51:25.543150	neuron_x	Setting output of neuron_x to ACTION_POTENTIAL.
2016-03-10 20:51:25.557295	neuron_y	neuron_y fires.
2016-03-10 20:51:25.561307	neuron_y	Setting output of neuron_y to ACTION_POTENTIAL.
2016-03-10 20:51:25.620065	neuron_h3	neuron_y is kicking neuron_h3.
2016-03-10 20:51:25.632281	neuron_h1	neuron_x is kicking neuron_h1.
2016-03-10 20:51:25.649561	neuron_h1	neuron_h1 fires.
2016-03-10 20:51:25.649841	neuron_h3	neuron_h3 fires.
2016-03-10 20:51:25.653656	neuron_h1	Setting output of neuron_h1 to ACTION_POTENTIAL.
2016-03-10 20:51:25.657057	neuron_h3	Setting output of neuron_h3 to ACTION_POTENTIAL.
2016-03-10 20:51:25.689440	neuron_h2	neuron_x is kicking neuron_h2.
2016-03-10 20:51:25.759928	neuron_z	neuron_h1 is kicking neuron_z.
2016-03-10 20:51:25.787273	neuron_z	neuron_z fires.
2016-03-10 20:51:25.791270	neuron_z	Setting output of neuron_z to ACTION_POTENTIAL.
2016-03-10 20:51:25.813897	neuron_h2	neuron_y is kicking neuron_h2.

2016-03-10 20:51:25.835654	neuron_h2	neuron_h2 fires.
2016-03-10 20:51:25.842069	neuron_h2	Setting output of neuron_h2 to ACTION_POTENTIAL.
2016-03-10 20:51:25.869606	neuron_z	neuron_h3 is kicking neuron_z.
2016-03-10 20:51:25.892445	neuron_z	neuron_z is still in refractory-period.
2016-03-10 20:51:25.895733	neuron_z	neuron_z is still in refractory_period at action potential, then a neuron neuron_h3 kicks in, now sum_of_weighted_inputs >= threshold.
2016-03-10 20:51:25.943689	neuron_z	neuron_h2 is kicking neuron_z.

Flower (http://192.168.0.114:5555) shows worker and the messages count being processed:



Configuration and Status of each neuron:

In [48]:

```
for neuron in reversed(neurons): printConfig(neuron)
```

_____neuron_z config:_____
_____
{'inputs': {'neuron_h1': {'lasting': datetime.timedelta(0, 0, 500000),
'kick_time': datetime.datetime(2016, 3, 10, 20, 51, 25, 763889), 'valu
e': 1}, 'neuron_h2': {'lasting': datetime.timedelta(0, 0, 500000), 'kic
k_time': datetime.datetime(2016, 3, 10, 20, 51, 25, 946926), 'value':
1}, 'neuron_h3': {'lasting': datetime.timedelta(0, 0, 500000), 'kick_ti
me': datetime.datetime(2016, 3, 10, 20, 51, 25, 873372), 'value': 1}},
'output': {'lasting': datetime.timedelta(0, 0, 100000), 'polarized_tim
e': datetime.datetime(2016, 3, 10, 20, 51, 25, 803324), 'value': 1}, 't
hreshold': 0.9, 'weights': {'neuron_h1': 1, 'neuron_h3': 1, 'neuron_h
2': -2}}

_____neuron_h3 config:_____
_____
{'inputs': {'neuron_y': {'lasting': datetime.timedelta(0, 0, 500000),
'kick_time': datetime.datetime(2016, 3, 10, 20, 51, 25, 628021), 'valu
e': 1}}, 'threshold': 0.9, 'output': {'lasting': datetime.timedelta(0,
0, 100000), 'polarized_time': datetime.datetime(2016, 3, 10, 20, 51, 2
5, 660253), 'value': 1}, 'connections': {'neuron_z'}, 'weights': {'neur
on_y': 1}}

_____neuron_h2 config:_____
_____
{'inputs': {'neuron_x': {'lasting': datetime.timedelta(0, 0, 500000),
'kick_time': datetime.datetime(2016, 3, 10, 20, 51, 25, 693412), 'valu
e': 1}, 'neuron_y': {'lasting': datetime.timedelta(0, 0, 500000), 'kick
_time': datetime.datetime(2016, 3, 10, 20, 51, 25, 817076), 'value':
1}}, 'threshold': 1.9, 'output': {'lasting': datetime.timedelta(0, 0, 1
00000), 'polarized_time': datetime.datetime(2016, 3, 10, 20, 51, 25, 84
5780), 'value': 1}, 'connections': {'neuron_z'}, 'weights': {'neuron_
x': 1, 'neuron_y': 1}}

_____neuron_h1 config:_____
_____
{'inputs': {'neuron_x': {'lasting': datetime.timedelta(0, 0, 500000),
'kick_time': datetime.datetime(2016, 3, 10, 20, 51, 25, 635170), 'valu
e': 1}}, 'output': {'lasting': datetime.timedelta(0, 0, 100000), 'polar
ized_time': datetime.datetime(2016, 3, 10, 20, 51, 25, 657518), 'valu
e': 1}, 'weights': {'neuron_x': 1}, 'threshold': 0.9, 'connections':
{'neuron_z'}}

_____neuron_y config:_____
_____
{'inputs': {}, 'output': {'lasting': datetime.timedelta(0, 0, 100000),
'polarized_time': datetime.datetime(2016, 3, 10, 20, 51, 25, 565365),
'value': 1}, 'connections': {'neuron_h2', 'neuron_h3'}, 'threshold': 0.
9}

_____neuron_x config:_____
_____
{'inputs': {}, 'output': {'lasting': datetime.timedelta(0, 0, 100000),
'polarized_time': datetime.datetime(2016, 3, 10, 20, 51, 25, 547371),
'value': 1}, 'connections': {'neuron_h1', 'neuron_h2'}, 'threshold': 0.
9}

Summary

In this experiment, we deploy 6 Celery worker containers across Docker-Swarm on two Raspberry Pi machines. Each container represents a neuron and all 6 neurons composite a XOR neural-network. The neural-network functions with distributed neurons colaborate in a concurrent and distributed fashion.

Note (2016/01/31):

XOR networking will probably not exist in realy biological neural-network, for that it requires "waiting" or "sychronization" with other neurons. The waiting will induce loss in efficency and therefore seemingly not a good design pattern.

References:

Action potential (https://en.wikipedia.org/wiki/Action_potential)
Neural coding (https://en.wikipedia.org/wiki/Neural_coding)
Artificial neuron (https://en.wikipedia.org/wiki/Artificial_neuron)
"All-or-none" principle (https://en.wikipedia.org/wiki/Action_potential#.22All-or-none.22_principle)
Refractory period (https://en.wikipedia.org/wiki/Action_potential#Refractory_period)

- The absolute refractory period is largely responsible for the unidirectional propagation of action potentials along axons.[34] At any given moment, the patch of axon behind the actively spiking part is refractory, but the patch in front, not having been activated recently, is capable of being stimulated by the depolarization from the action potential.

In [ ]:

```
```