ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

OS Lab

Εαρινό Εξάμηνο 2015-2016

Αναφορά 1ης Εργαστηριακής Άσκησης Cgroups

Μάρτιος 2016

Καλλάς Κωσταντίνος 03112057 – Τζίνης Ευθύμιος 03112007

8ο Εξάμηνο, Σ.Η.Μ.Μ.Υ., Ε.Μ.Π.

Γενική Περιγραφή:

Σε αυτή την άσκηση, αφού καταφέραμε να στήσουμε το VM και να εγκαταστήσουμε τον δαίμονά, προσπαθήσαμε να καταλάβουμε από τα source αρχεία του κώδικα πώς λειτουργεί. Αυτό δεν απέδωσε και γι αυτό τον λόγο χρησιμοποιήσαμε τα log files που του monitor που βρίσκονται στο wd του δαίμονα. Με βάση τα errors αυτά καταλάβαμε που θα έπρεπε να είναι τα scripts που οφείλουμε να φτιάξουμε για να συμπληρώσουμε τον δαίμονα ώστε να λειτουργεί σωστά. Υλοποιήσαμε λοιπόν 2 scripts:

Το mod_policy_cpumin.py που παίρνει είσοδο από το stdin στο format που ορίζεται στην εκφώνηση της άσκησης , και επιστρέφει σε κατάλληλη μορφή το αν “χωράνε” οι εφαρμογές που μας ζητάει ο χρήστης, και το ποσοστό που πρέπει η καθεμία να πάρει.

Το δεύτερο script που υλοποιήσαμε mod_limit_cpu.py δέχεται εντολές στην μορφή που ορίζεται στην εκφώνηση και ύστερα τροποποιεί κατάλληλα αρχεία στο φάκελο /sys/fs/cgroup/cpu/monitor/ ώστε οι εφαρμογέ να λαμβάνουν όσο χρόνο απαιτείται στον επεξεργαστή.

Λεπτομέρειες Υλοποίησης:

Αποφασίσαμε να ακολουθήσουμε ένα απλό βασικό σχέδιο υλοποίησης για το script policy. Θεωρώντας ότι έχουμε σταθερά στο σύστημα μας 2000 χιλιοστά πόρων (2 πυρήνες), στο scrip policy απλώς αθροίζονται τα χιλιοστά που ζητά για κάθε εφαρμογή ο χρήστης και αν δεν ξεπερνάνε τα 2000 τότε θεωρείται ότι χωράνε και η policy απαντάει θετικά.

Πάνω σε αυτό το βασικό μοντέλο προσθέσαμε τις εξής βελτιώσεις:

- Θεωρώντας ότι οι ελαστικές εφαρμογές ζητούν πάντα 50 χιλιοστά κάναμε τα εξής. Επιθυμούμε ο επεξεργαστής μας να τρέχει συνέχεια στο 100% για αυτό το λόγο δεν θέλουμε να μονοπωλείται απο ανελαστικές εφαρμογές αφού αυτές έχουν peaks υψηλής ζήτησης και την υπόλοιπη ώρα είναι σχετικά αδρανείς. Για αυτό policy επιστρέφει αρνητική απάντηση αν ο χρήστης ζηταέι να “μονοπωλήσουν” στο σύστημα μας ανελαστικές εφαρμογές. Δηλαδή με βάση ένα ratio που ορίζουμε εμείς (στο παράδειγμα μας το θεωρήσαμε 90%) δεν επιτρέπει οι ανελαστικές να παίρνουν παραπάνω ποσοστό πόρων από αυτό το ratio. Για παράδειγμα έστω ότι ο χρήστης ζηταέι να τρέξουν δυο εφαρμογές με 1000 χιλιοστά η καθεμία. Το script μας δεν θα το επιτρέψει αυτό όχι γιατί δεν φτάνουν τα χιλιοστά του επεξεργαστή μας για να τους εξυπηρετήσουν αλλά επειδή ο επεξεργαστής θα μονοπωλείται από ανελαστικές εφαρμογές.

- Επίσης κάναμε άλλη μια βελτιστοποίηση στο policy. Όταν οι εφαρμογές του χρήστη δεν απαιτούν όλα τα χιλιοστά του επεξεργαστή αποφασίσαμε να παραχωρούμε όσα χιλιοστά απομένουν στις elastic εφαρμογές. Αυτό το κάναμε επειδή αν για παράδειγμα απαιτεί μια ανελαστική εφαρμογή 1000 χιλιοστά και μια ελαστική 50 χιλιοστά τότε αν στα εκάστοτε cpu.shares βάλουμε 1000 και 50 η πρώτη εφαρμογή θα παίρνει περίπου 1900 χιλιοστά και η δεύτερη περίπου 100 χιλιοστά (σε περιόδους που και οι 2 ζητούν τα μέγιστα) . Αυτό όμως δεν είναι επιθυμητή συμπεριφορά αφού η πρώτη εφαρμογή “πληρώνει” μόνο για 1000 χιλιοστά ενώ με οι ελαστικές μας ενδιαφέρει σε βάθος χρόνου να έχουν όσο περισσότερους πόρους γίνεται.

Η λειτουργία της mod_limit_cpu.py είναι αρκετά ξεκάθαρη για αυτό δεν θα μπούμε σε επεξηγηματικές λεπτομέρειες στην αναφορά μας για την υλοποίηση της.

Testing

Για να τεστάρουμε την ορθή χρήση των script που φτιάξαμε χρησιμοποιήσαμε την εντολή top που δείχνει το ποσοστό χρήσης κάθε διεργασίας στον επεξεργαστή. Ελέγξαμε τη λειτουργία εκτελώντας το script cgmon_demo.sh αλλά και με ένα δικό μας script (our_demo.sh) το ο οποίο “αναγκάζει” τις ανελαστικές εφαρμογές να εκτελούν ένα infinite loop από sleep και stress το οποίο υλοποιείται στο compute.sh που παρατίθεται στο τέλος της αναφοράς. Έτσι προσομοιώνουμε καλύτερα την λειτουργία των ανελαστικών εφαρμογών και παρατηρούμε ότι οι ελαστικές (που εκτελούν stress πάντα) παίρνουν παραπάνω χρόνο στον επεξεργαστή όταν μπορούν.

mod_policy.py

```
#!/usr/bin/env python
import sys
import os
from  math import floor
import traceback
from stat import *


mon_directory = "/sys/fs/cgroup/cpu/monitor/"
inputlines = sys.stdin.readlines()

ratio=0.9
total_cpu = 2000

elastic_apps = 0
total_shares = 0
non_elastic_total = 0
for line in inputlines:
 arguments = line[:-1].split(":")
 try:
    total_shares += int(arguments[3])
    if(int(arguments[3]) != 50):
        non_elastic_total += int(arguments[3])
    else:
        elastic_apps += 1
 except IndexError:
    print("IndexError!!")
    sys.exit(1)


#Check if the shares are ok for all
if(total_shares > total_cpu):
 print("score:-2.0")
elif(non_elastic_total > ratio * total_cpu):
 print("score:-1.0")
else:
 print("score:1.0")
```

```
    #Set the elastic cpu.shares as the remaining shares
    if(elastic_apps):
     elastic_share = int(floor(float(total_cpu - non_elastic_total)/elastic_apps))
    else:
     elastic_share = 0
    for line in inputlines:
     arguments = line[:-1].split(":")
     try:
         if( arguments[0] == "policy" and arguments[2] == "cpu") :
             app_name = arguments[1]
             value = arguments[3]
             if(int(arguments[3]) == 50 ):
                 value = str(elastic_share)
     except IndexError:
         print("IndexError!!")
         sys.exit(1)
     print("set_limit:" + app_name + ":cpu.shares:" + value)
```

mod_limit_cpu.py

```
#!/usr/bin/env python

import sys
import os
import re
import shutil

temp_log = open("/root/cgmon/demo/log_gia_pipes.txt", "a")
temp_log.write("==============================================\nNEW ENTRY\n")

inlines=sys.stdin.readlines()

for inline in inlines:
    temp_log.write("---------------------------------\nNEW LINE\n")
    #creation-rm-add-setlim re
    creation='create:'
    remove='remove:'
    add='add:'
    setlim='set_limit:'
    monitor=':[A-Za-z0-9_]+'
    app='monitor:[A-Za-z0-9_]+'
    app2='cpu:[A-Za-z0-9_]+'
    shares='cpu\.shares:[A-Za-z0-9_]+'
    usefull='[A-Za-z0-9_]+'
    pid='cpu:[A-Za-z0-9_]+:[A-Za-z0-9_]+'
    words=inline
    data={'monitor':'','appname':'','pid':'','shares':''}
    proc={'create':0,'remove':0,'add':0,'set_limit':0}
```

```
for i in proc:
    m=re.findall(i,words)
    proc[i]=len(m)
    if (len(m)>0):
        monitor=''.join(i)+''.join(monitor)
        mm=re.findall(monitor,words)
        mmstr=''.join(mm)
        ma=re.findall(usefull,mmstr)
        if(len(ma)>1):
            data['monitor']=ma[1]

        m1=re.findall(app,words)
        appstr=''.join(m1)
        #print(appstr)
        ma=re.findall(usefull,appstr)
        if(len(ma)>1):
            data['appname']=ma[1]
        m1=re.findall(app2,words)
        appstr=''.join(m1)
        #print(appstr)
        ma=re.findall(usefull,appstr)
        if(len(ma)>1):
            data['appname']=ma[1]
ms=re.findall(shares,words)
msstr=''.join(ms)
ms=re.findall(usefull,msstr)
if(len(ms)>2):
    data['shares']=ms[2]

mp=re.findall(pid,words)
mpstr=''.join(mp)
mp=re.findall(usefull,mpstr)
if(len(mp)>2):
    data['pid']=mp[2]

#Log
for i in data:
    temp_log.write(i+' '+data[i]+ '\n')
temp_log.write('\n')

command=""
#Choose the branch that we are going to do
for com in proc:
    temp_log.write(com+' '+ str(proc[com])+ '\n')
    if proc[com] == 1:
        command = com
        temp_log.write("Current command: " + command + "\n")
        break
temp_log.write('\n')
```

```
    cgroup_dir = "/sys/fs/cgroup/cpu/monitor/"

    #For command create
    app_dir = cgroup_dir + data["appname"] + "/"
    if command == "create":
        if not os.path.exists(app_dir):
            os.makedirs(app_dir)
            temp_log.write("Made the dir\n")


    #TODO: 8imios
    #Command: add
    if command== "add":
        exists = 0
        for line in open(app_dir + "tasks"):
            if data['pid'] in line:
                exists = 1
                break
        if not exists:
            os.system("echo " + data['pid'] + " > " + app_dir + "tasks")

    #Just add the task pid in the tasks folder
    #Command: remove
    app_dir = cgroup_dir + data["appname"] + "/"
    if command == "remove":
        if os.path.exists(app_dir):
            os.rmdir(app_dir)
            temp_log.write("Removed the dir\n")

    #Set_limit
    if command == "set_limit":
        if os.path.exists(app_dir + "cpu.shares"):
            cpu_shares_fp = open(app_dir + "cpu.shares", "w+")
            cpu_shares_fp.write(data["shares"] + "\n")
            cpu_shares_fp.close()

temp_log.close()
```

our_demo.sh

```
#!/bin/bash
#!/bin/sh

set -v

WD=${PWD}/demo

cgmon daemon stop -w ${WD}

sleep 1

cgmon daemon start -w ${WD} -p ${WD}/mod_policy_cpumin.py -l ${WD}/mod_limit_cpu.py

sleep 1

cgmon app list

sleep 1

# assuming total 2000 millicpus == 2 cpus
cgmon policy create -n platinum -p 1000
cgmon policy create -n silver -p 500
cgmon policy create -n bronze -p 100
cgmon policy create -n elastic -p 50
cgmon policy list

sleep 1

cgmon app spawn -p platinum -e "/root/cgmon/compute.sh" -n BANKDB
cgmon app spawn -p silver -e "/root/cgmon/compute.sh" -n WEBDB
cgmon app spawn -p silver -e "/root/cgmon/compute.sh" -n VIDEOPLAYER
cgmon app list

sleep 3

cgmon app spawn -p bronze -e "stress -c 2" -n DOTASTREAMING
cgmon app spawn -p bronze -e "stress -c 2" -n CHATROULETTE
cgmon app spawn -p elastic -e "stress -c 2" -n PHOTOSHOP
cgmon app spawn -p elastic -e "stress -c 2" -n VIDEOENC
cgmon app spawn -p elastic -e "stress -c 2" -n SPAMBOT
cgmon app list

sleep 3

cgmon policy apply -t CHATROULETTE -p platinum

python /root/cgmon/tasks_pid.py
```

```
cat ${WD}/tasks_pids

sleep 5

# This should fail: not enough cpu
cgmon app spawn -p platinum -e "stress -c 2" -n MEDICALDB

sleep 1

# forcing it violates policies
cgmon app spawn -p platinum -e "stress -c 2" -n MEDICALDB -f

sleep 10
pkill -f stress
```

compute.sh

```
#!bin/bash
echo 'Hi I am Anelastic its fantastic! Lets compute something'
while true; do
   stress -c 2 -t 4
   sleep 5
done
```