基于延迟的动态比率负载均衡技术方案

项目信息

|  |  |  |  |
| --- | --- | --- | --- |
| 工程项目名称 |  | 项目实施地点 |  |
| 客户单位名称 |  | 实施单位名称 |  |

版本控制信息

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 文档标题 |  | | | |
| 项目名称 |  | | | |
| 类别 | 规范文档 □ 设计方案 □ 实施文档 ■  配置文档 □ 测试文档 □ 运维手册 □  项目调研 □ 其 他 □ | | | |
| 版本 | 发布时间 | 文档修改人 | 修改审核人 | 修改内容和原因 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

版权说明

本文件中出现的任何文字叙述、文档格式、插图、照片、方法、过程等内容，除另有特别注明，均为保密信息。任何个人、机构未经F5公司的书面授权许可，不得复制、引用或传播本文件的任何片断，无论通过电子形式或非电子形式。

目录

1 背景 4

2 部署方案 5

2.1 模块关系图 6

2.2 组件明细 7

2.2.1 延迟原始数据iRule 7

2.2.2 延迟展示iRule 10

2.2.3 长期数据保存脚本 11

2.2.4 长期数据统计脚本 15

2.2.5 EAV健康检查 23

2.3 部署步骤 33

3 现网测试 37

3.1 测试拓扑 37

3.2 测试过程 38

3.2.1 测试场景一：数据库切换 41

3.2.2 测试场景二：数据库回切 44

3.2.3 测试场景二：应用延迟单边上升 46

3.2.4 测试场景二：应用延迟单边下降 49

3.3 测试结果 51

4 系统运维 51

4.1 手工计算 51

4.2 一般运行日志 52

4.3 系统切换 53

4.4 系统重启 55

4.5 统计文件缺失 56

5 总结 56

# 背景

本方案主要描述在双活数据中心当中，在第二中心应用服务器访问第一中心数据库服务器存在固定延迟的情况，如何通过F5 iRule特性，实时计算服务器延迟，并实服务器流量分配比例跟随服务器延迟进行实时动态调整。

在正常场景下，主数据中心DB处于Active状态，此时主中心的APP访问DB速度比较快，备份数据中心连接主中心的DB由于存在广域传输延迟，访问速度比较慢，由于需要保证主备中心的应用都处于可用状态，在备中心APP也需要对外提供服务，为在访问体验和高可用性之间进行平衡，日常流量的1/10（暂定）通过F5发向备中心APP，其他流量发往主中心，该步骤可以通过BIGIP的比例负载均衡算法实现，如下图：

在数据库从主中心切到备中心的情况下，延迟情况出现逆转，备中心的APP延迟较低，主中心的延迟较高，此时BIGIP需要将更多的流量发送给备中心，如下图：



传统做法为找到受DB切换影响的所有业务进行手工一一调整，该做法缺点明显：

（1）配置业务量大，响应时间慢，业务情况紧急，需要短时间内调整完毕，否则严重影响业务

（2）容易出现配置缺失和错误，一个数据库涉及到的应用会非常多，在短时间内下进行大量的配置手工调整很可能造成配置错漏。

考虑到双中心出现数据库切换的场景，对于双活非等比例的APP，F5可以根据iRule和F5自定义健康检查EAV来实现自动化的、基于延迟的动态比率负载均衡算法，来实现该场景下流量的反转。

本方案将详细介绍这几个模块的协调工作如何实现场景需求。

# 部署方案

本次方案将通过iRule和F5自定义健康检查实现基于延迟的动态比率负载均衡算法，其中

* iRule将记录每个服务器在每秒的总连接数和所有连接的延迟累加，将每个服务器每秒的延迟累加

和总连接数保存在iRule Session Table中

* 自定义健康检查获取到Session Table中的数据，依据服务器的延迟值调整服务器的比率
* 转发平面保存延迟数据三分钟，通过外部脚本将延迟获取到BIGIP本身的内存磁盘挂点/dev/shm
* 每天对当前累计的所有数据，计算每三分钟的均值和标准差保存在/dev/shm挂点下
* iRule和健康检查将通过一个内部的VS进行通信，获取三分钟内的所有延迟，计算均值后和

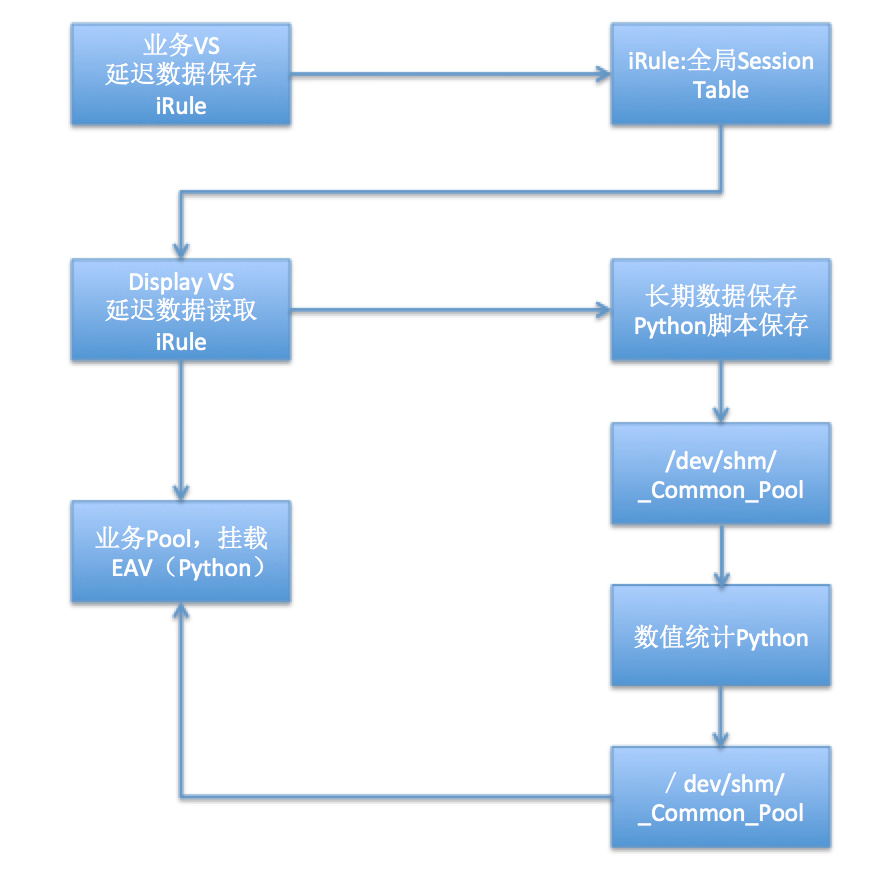
累计均值，标准差进行对比，在4个标准差范围内认为该服务器延迟处于正常范围，如超过

4个标准差，分为正向（增加）以及负向（减少），正向偏离将调低延迟为固定值，负向偏离

将调高为固定值

* 出现调整行为后，如某次健康检查获取到的延迟结果处于调整前均值一倍标准差范围内，将回复该服务器分配流量的比例

## 模块关系图



## 组件明细

### 延迟原始数据iRule

（1）获取延迟信息

Standard类型的VS，根据链接过程获取延迟， iRule为

|  |
| --- |
| when SERVER\_CONNECTED {  set tcp\_start\_time [clock clicks -milliseconds]  }  when LB\_SELECTED {  set fields [split [LB::server name] " "]  set pool [lindex $fields 0]  set server [lindex $fields 1]  set port [lindex $fields 2]  #set client "[IP::client\_addr]"  #set client\_port "[TCP::client\_port]”  }  when SERVER \_CLOSED {  set tcp\_end\_time [ clock clicks -milliseconds ]  set current\_time [ clock seconds]  set tcp\_delay [expr ($tcp\_end\_time - $tcp\_start\_time) ]  table incr -subtable $pool "delay:$server:$port:$current\_time" $tcp\_delay  table incr -subtable $pool "count:$server:$port:$current\_time" 1  } |

如为Performance L4类型的VS：

|  |
| --- |
| when SERVER\_CONNECTED {  set tcp\_start\_time [clock clicks -milliseconds]  }  when LB\_SELECTED {  set fields [split [LB::server name] " "]  set pool [lindex $fields 0]  set server [lindex $fields 1]  set port [lindex $fields 2]  #set client "[IP::client\_addr]"  #set client\_port "[TCP::client\_port]”  }  when CLIENT \_CLOSED {  set tcp\_end\_time [ clock clicks -milliseconds ]  set current\_time [ clock seconds]  set tcp\_delay [expr ($tcp\_end\_time - $tcp\_start\_time) ]  table incr -subtable $pool "delay:$server:$port:$current\_time" $tcp\_delay  table incr -subtable $pool "count:$server:$port:$current\_time" 1  } |

注：在standard模式下，VS工作在全代理模式，有SERVER\_CLOSE事件触发，而在Performance FastL4模式模式下，F5无后端状态机，SERVER\_CLOSE事件不会触发。

上述iRule会根据连接的发起时间以及客户端的连接结束时间来判断一个连接的延迟，延迟记录精度为毫秒

set tcp\_delay [expr ($tcp\_end\_time - $tcp\_start\_time) ]

同时会将该数据写入到内存session table 当中，table中的键值是根据时间戳进行区分，每秒每个member会记录一个延迟条目：

set current\_time [ clock seconds]

table incr -subtable $pool "delay:$server:$port:$current\_time" $tcp\_delay

同时会记录当前时间周期（一秒）内的连接总数

table incr -subtable $pool "count:$server:$port:$current\_time" 1

默认情况下的session table 中的条目time out时间为三分钟，三分钟后table中条目自动删除

### 延迟展示iRule

该组件将处理组件1中的table数据，并且通过一定的数据结构进行展示，用来给后面的组件进行延迟算法的判断依据。

该过程将需要另外建立一个用于监控的HTTP VS，通过该VS的response的结果获取session table的数据：

|  |
| --- |
| when HTTP\_REQUEST {  set poolname [HTTP::uri]  set response ""  foreach key [table keys -subtable $poolname] {  set value [table lookup -notouch -subtable $poolname $key]  set response "$response<br>$key=$value"  }  HTTP::respond 200 content $response  return  } |

该iRule中：

从http请求中的uri获取pool名称

set pool\_name [HTTP::uri]

获取所有的key

foreach key [table keys -subtable $pool\_name] {

set value [table lookup -notouch -subtable $pool\_name $key]

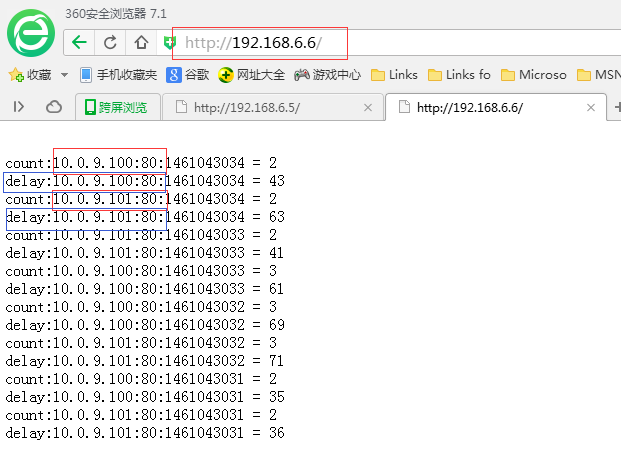
获取pool下所有member的

set response "$response<br>$key = $value"

}

将组件1当中的数据结构table进行处理并且赋值给response作为监控VS的请求结果，结果包括了3分钟内所有的连接在每秒时刻的毫秒延迟，以每秒连接数量统计。

测试192.168.6.5是业务VS，默认情况下10.0.9.100和10.0.9.101两业务网的pool member将正常轮询。可以通过监控VS来获得这两个pool member的延迟和连接情况，访问监控VS打印该结果：



可以获得所有连接的延迟和连接数情况。

当能够从内存中获取这些数据后，就能将该结果用于给后面的自定义健康检查程序去作为健康检查的依据。

### 长期数据保存脚本

/config/script/LatencyLongTermSave.py

下面的脚本设置在crontab里面，每间隔三分钟运行一次，将iRule Session Table中的数据通过业务展示VS读取出来，保存在/dev/shm/目录下以\_Common\_+Pool名称 为文件名的文件中

|  |
| --- |
| #!/usr/bin/env python  # -\*- coding: utf-8 -\*-  import httplib  import pickle  import os  import re  def GetDelayStatFromMonitorVs(ip,PoolName):  headers = {"Content-type": "application/x-www-form-urlencoded","Accept": "text/plain"}  conn = httplib.HTTPConnection(ip)  conn.request('GET', PoolName, None, headers)  httpres = conn.getresponse()  return httpres.read()  def \_\_isVirtualServer(line):  if re.match(r"^ltm\svirtual\s\S+\s{$", line):  return True  def \_\_GetLtmConfigAndFilterPoolHasRule(LtmFileNameV11, RuleFilter):  vsName = ''  vsRule =''  vsSet = {}  vsRulesList = []  FilteredPoolList = []  #with open(LtmFileNameV11, 'rb') as f:  try:  f = open(LtmFileNameV11, 'rb')  while 1:  row = f.readline()  if row == '':  break  if \_\_isVirtualServer(row):  vsName = re.search("^ltm\svirtual\s(\S+)\s{$", row).group(1)  vsPool = ''  while 1:  row = f.readline()  if re.match("^}.\*$", row):  break  row = row.rstrip()  if re.match("^\s+pool\s(.\*)$", row):  vsPool = re.search("^\s+pool\s(.\*)$", row).group(1)  vsSet[vsName + '-pool'] = vsPool  continue  if re.match("^\s+rules\s{", row):  while 1:  row = f.readline();  if re.match("^\s+}.\*$", row):  break  if re.match("^\s+\/Common\/(\S+).\*$", row):  vsRule = row.strip()  ####print vsRule  if vsRule == RuleFilter and vsPool != '':  FilteredPoolList.append(vsSet.get((vsName + '-pool'), vsPool))    else:  continue  else:  continue  except Exception,ex:  print 'open bigip.conf failed'  raise    return set(FilteredPoolList)  if \_\_name\_\_ == '\_\_main\_\_':  MonitorVsIp = '10.128.5.12'  Pools = \_\_GetLtmConfigAndFilterPoolHasRule('/config/bigip.conf', '/Common/calc')  for PoolName in Pools:  data\_array = GetDelayStatFromMonitorVs(MonitorVsIp,PoolName).split('<br>')  filename = '/dev/shm/'+PoolName.replace('/','\_')  file\_array={}  if os.path.isfile(filename):  inputfile = open(filename,'rb')  file\_array = pickle.load(inputfile)  inputfile.close()  for item in data\_array:  metrics = item.split('=')  if metrics[0]:  key= metrics[0]  value= metrics[1]  file\_array[key]= value  outputfile = open(filename,'wb')  pickle.dump(file\_array,outputfile,0)  print 'save file '+filename +' record :' +str(len(file\_array))  outputfile.close() |

MonitorVsIp = '10.128.5.12'

该参数指定本机session table数据提取VS的IP地址

Pools = \_\_GetLtmConfigAndFilterPoolHasRule('/config/bigip.conf', '/Common/calc')

/Common/calc 为数据采集iRule的名称，上述命令将自动提取包含该iRule的Pool名称列表

### 长期数据统计脚本

/config/script/CalcLatency.py

下面的脚本将2.2.3节保存的长期延迟数据文件进行统计计算，计算出特定Pool Member的延迟期望（数学均值）和标准差，计算粒度为3分钟

|  |
| --- |
| #!/usr/bin/python  # -\*- coding: utf-8 -\*-  from \_\_future\_\_ import division  import pickle  import math  import os  import re as re  def GetSample(rawfilename):  inputfile = open(rawfilename,'rb')  data\_array=pickle.load(inputfile)  inputfile.close()  count\_map={}  delay\_map={}  avg\_map={}  for item in data\_array:  if not item:  continue  metrics = item.split(':')  value = data\_array[item]  if value:  if metrics[0] == 'count':  count\_map[metrics[1]+':'+metrics[2]+':'+metrics[3]]=value  if metrics[0] == 'delay':  delay\_map[metrics[1]+':'+metrics[2]+':'+metrics[3]]=value  for key in count\_map.keys():  if delay\_map.has\_key(key):  delay=float(delay\_map[key])/float(count\_map[key])  temp=key.split(':')  member\_ip\_port=temp[0]+':'+temp[1]  timestamp = int(temp[2])  if avg\_map.has\_key(member\_ip\_port):  avg\_map[member\_ip\_port][timestamp]=delay  else:  avg\_map[member\_ip\_port]={}  avg\_map[member\_ip\_port][timestamp]=delay  return avg\_map  class StdDev(object):  def \_\_init\_\_(self):  self.\_\_Sample = []  self.\_\_AvgValue = 0  self.\_\_StdDev = 0  self.\_\_ActiveStatus='Enabled'  self.\_\_Ratio = 0  @property  def StdDev(self):  return self.\_\_StdDev  @property  def AvgValue(self):  return self.\_\_AvgValue  @property  def Ratio(self):  return self.\_\_Ratio  @property  def Sample(self):  return self.\_\_Sample  @property  def ActiveStatus(self):  return self.\_\_ActiveStatus  def SetSample(self,sample):  self.\_\_Sample = sample  def setStatus(self,status):  self.\_\_ActiveStatus = status  def setRatio(self,Ratio):  self.\_\_Ratio = Ratio  def AddSample(self,value):  self.\_\_Sample.append(value)  def GetSampleCount(self):  return len(self.\_\_Sample)  def CalcStdDev(self):  TotalCount = 0  TotalValue = 0  TotalValue\_sqrt = 0  print 'sample count is:'+str(len(self.\_\_Sample))  for value in self.\_\_Sample:  TotalValue += float(value)  TotalValue\_sqrt += float(value)\*\*2  TotalCount += 1  AvgValue= float(TotalValue/TotalCount)  StdDev = math.sqrt((TotalValue\_sqrt+TotalCount\*(AvgValue\*\*2)-2\*(TotalValue)\*AvgValue)/TotalCount)  self.\_\_AvgValue = AvgValue  self.\_\_StdDev = StdDev  def getSigma(self,value):  if value > self.\_\_AvgValue:  return [abs(value-self.\_\_AvgValue)/self.\_\_StdDev,1]  return [abs(value-self.\_\_AvgValue)/self.\_\_StdDev,-1]  def \_\_isVirtualServer(line):  if re.match(r"^ltm\svirtual\s\S+\s{$", line):  return True  def \_\_GetLtmConfigAndFilterPoolHasRule(LtmFileNameV11, RuleFilter):  vsName = ''  vsRule =''  vsSet = {}  vsRulesList = []  FilteredPoolList = []  #with open(LtmFileNameV11, 'rb') as f:  try:  f = open(LtmFileNameV11, 'rb')  while 1:  row = f.readline()  if row == '':  break  if \_\_isVirtualServer(row):  vsName = re.search("^ltm\svirtual\s(\S+)\s{$", row).group(1)  vsPool = ''  while 1:  row = f.readline()  if re.match("^}.\*$", row):  break  row = row.rstrip()  if re.match("^\s+pool\s(.\*)$", row):  vsPool = re.search("^\s+pool\s(.\*)$", row).group(1)  vsSet[vsName + '-pool'] = vsPool  continue  if re.match("^\s+rules\s{", row):  while 1:  row = f.readline();  if re.match("^\s+}.\*$", row):  break  if re.match("^\s+\/Common\/(\S+).\*$", row):  vsRule = row.strip()  ####print vsRule  if vsRule == RuleFilter and vsPool != '':  FilteredPoolList.append(vsSet.get((vsName + '-pool'), vsPool))    else:  continue  else:  continue  except Exception,ex:  print 'open bigip.conf failed'  raise  return set(FilteredPoolList)  if \_\_name\_\_ == '\_\_main\_\_':  pathname= '/dev/shm/'  ####Pools = ['/Common/Pool\_test']  Pools = \_\_GetLtmConfigAndFilterPoolHasRule('/config/bigip.conf', '/Common/calc')  for PoolName in Pools:  filename = '/dev/shm/'+PoolName.replace('/','\_')  sample\_set = GetSample(filename)  for member in sample\_set.keys():  inputfilename = pathname+'Calc'+PoolName.replace('/','\_')+'\_'+member.replace(':','\_')  if os.path.isfile(inputfilename):  input = open(inputfilename,'rb')  stddev=pickle.load(input)  input.close()  if stddev.ActiveStatus != 'Enabled':  print 'this is not in Enabled status for member: '+member  continue  stddev=StdDev()  avg\_map=sample\_set[member]  i=0  for key in avg\_map.keys():  i +=1  value = int(avg\_map[key])  if value >1000:  continue  count =1  for x in range(1,180):  if avg\_map.has\_key(key-x):  if avg\_map[key-x]<1000:  value +=int(avg\_map[key-x])  count += 1  stddev.AddSample(int(value/count))  stddev.CalcStdDev()  print 'save '+member+ ' recode '+ str(stddev.GetSampleCount())  print 'the avg value is '+str(stddev.AvgValue)  print 'the stddev value is '+str(stddev.StdDev)  outputfile = open(inputfilename,'wb')  pickle.dump(stddev,outputfile)  outputfile.close() |

该脚本通过crontab每天运行一次，运行后将在/dev/shm/路径生成以Calc开头的文件，该文件保存均值，标准差以及样本空间

Pools = \_\_GetLtmConfigAndFilterPoolHasRule('/config/bigip.conf', '/Common/calc')

/Common/calc 为数据采集iRule的名称，上述命令将自动提取包含该iRule的Pool名称列表

### EAV健康检查

经过上述的部署以及计算，系统将获取到特定member长期时间内的延迟均值以及标准差，我们将通过External Monitor来实现非built-in的负载均衡算法。可以根据具体的双数据中心延迟情况，服务器情况来调整算法的结果，在该monitor的运行过程当中，对延迟的情况进行判断，通过对node节点的ratio设置来实现按照流量比例的区分需求。

以下是按照现有的数据，进行3分钟内延迟计算和连接数平均值的python自定义monitor

|  |
| --- |
| #!/usr/bin/python  import httplib  import pickle  import math  import os  import traceback  import re  import logging  import logging.handlers  import commands  logger = logging.getLogger()  class StdDev(object):  def \_\_init\_\_(self):  self.\_\_Sample = []  self.\_\_AvgValue = 0  self.\_\_StdDev = 0  self.\_\_ActiveStatus='Enabled'  self.\_\_Ratio = 0  @property  def StdDev(self):  return self.\_\_StdDev  @property  def AvgValue(self):  return self.\_\_AvgValue  @property  def Ratio(self):  return self.\_\_Ratio  @property  def Sample(self):  return self.\_\_Sample  @property  def ActiveStatus(self):  return self.\_\_ActiveStatus  def SetSample(self,sample):  self.\_\_Sample = sample  def setStatus(self,status):  self.\_\_ActiveStatus = status  def setRatio(self,Ratio):  self.\_\_Ratio = Ratio  def AddSample(self,value):  self.\_\_Sample.append(value)  def GetSampleCount(self):  return len(self.\_\_Sample)  def CalcStdDev(self):  TotalCount = 0  TotalValue = 0  TotalValue\_sqrt = 0  print 'sample count is:'+str(len(self.\_\_Sample))  for value in self.\_\_Sample:  TotalValue += float(value)  TotalValue\_sqrt += float(value)\*\*2  TotalCount += 1  AvgValue= float(TotalValue/TotalCount)  StdDev = math.sqrt((TotalValue\_sqrt+TotalCount\*(AvgValue\*\*2)-2\*(TotalValue)\*AvgValue)/TotalCount)  self.\_\_AvgValue = AvgValue  self.\_\_StdDev = StdDev  def getSigma(self,value):  if value > self.\_\_AvgValue:  return [abs(value-self.\_\_AvgValue)/self.\_\_StdDev,1]  return [abs(value-self.\_\_AvgValue)/self.\_\_StdDev,-1]  class Latency(object):  ####form IP, make the instance, deal the data,count delay and their aaverage  def \_\_init\_\_(self, ip,poolname):  self.\_\_Count\_map = {}  self.\_\_Delay\_map = {}  self.\_\_DelayAvg\_map = {}  self.\_\_PoolName = poolname  data\_array = self.\_\_GetDelayStatFromMonitorVs(ip).split('<br>')  for item in data\_array:  if not item:  continue  metrics = item.split(':')  Ip\_InMap = metrics[1]+':'+metrics[2]  if len(metrics[3].split('=')) == 2:  value\_str = metrics[3].split('=')[1]  if value\_str:  value = int(value\_str)  if metrics[0] == 'count':  self.\_\_CountAndDelayPlus(self.\_\_Count\_map, Ip\_InMap, value)  elif metrics[0] == 'delay':  self.\_\_CountAndDelayPlus(self.\_\_Delay\_map, Ip\_InMap, value)  for item in self.\_\_Count\_map.keys():  self.\_\_DelayAvg\_map[item] = self.\_\_Delay\_map.get(item)/self.\_\_Count\_map.get(item)  @property  def Count\_map(self):  return self.\_\_Count\_map  @property  def Delay\_map(self):  return self.\_\_Delay\_map  @property  def DelayAvg\_map(self):  return self.\_\_DelayAvg\_map  ####the http function  def \_\_GetDelayStatFromMonitorVs(self, ip):  headers = {"Content-type": "application/x-www-form-urlencoded","Accept": "text/plain"}  conn = httplib.HTTPConnection(ip)  conn.request('GET', self.\_\_PoolName, None, headers)  httpres = conn.getresponse()  return httpres.read()  def \_\_CountAndDelayPlus(self, data\_map, ip, value):  old\_value = data\_map.get(ip, 0)  if old\_value:  value += old\_value  data\_map[ip] = value  else:  data\_map[ip] = value  def main():  node\_ips = os.getenv("NODE\_IP")  node\_ip = re.search(r"^\D+(\d+\.\d+\.\d+\.\d+).\*$", node\_ips).group(1)  node\_port = os.getenv("NODE\_PORT")  logging\_level = os.getenv('LogLevel')  Pid = str(os.getpid())  #hdlr = logging.handlers.TimedRotatingFileHandler("/var/log/LatencyLB.log", when='D', interval=1, backupCount=7)  hdlr = logging.handlers.RotatingFileHandler("/var/log/LatencyLB.log ", maxBytes=1000000, backupCount=7)  formatter = logging.Formatter('%(asctime)s %(levelname)s ' + Pid + ' ' + node\_ip + ':' + node\_port + ' %(message)s')  hdlr.setFormatter(formatter)  logger.addHandler(hdlr)  if 'DEBUG' == logging\_level:  logger.setLevel(logging.DEBUG)  else:  logger.setLevel(logging.INFO)  pathname= '/dev/shm/'  MemberIp = node\_ip+':'+node\_port  PoolName = os.getenv('PoolName')  ltsfilename = '/dev/shm/'+PoolName.replace('/','\_')  try:  logging.info('checking member '+node\_ip+':'+node\_port)  MonitorVsIp = '10.128.5.12'  inputfilename =pathname+'Calc'+PoolName.replace('/','\_')+'\_'+MemberIp.replace(':','\_')  stddev=StdDev()  if os.path.isfile(inputfilename):  latency=Latency('10.128.5.12',PoolName)  logging.info('MonitorVsIp :'+MonitorVsIp+' PoolName: '+PoolName)  logging.info('get Pool: '+PoolName +' latency record '+str(len(latency.DelayAvg\_map)))  input = open(inputfilename,'rb')  stddev = pickle.load(input)  input.close()  else:  logging.info(' The Calc File is not Exist ')  print 'up'  return  if not latency.DelayAvg\_map.has\_key(MemberIp):  print 'up'  return  sigma=stddev.getSigma(latency.DelayAvg\_map[MemberIp])  logging.info( 'static stddev:' +str(stddev.StdDev))  logging.info( 'static avg:'+str(stddev.AvgValue))  logging.info( 'current avg:'+str(latency.DelayAvg\_map[MemberIp]))  logging.info( 'sigma:'+str(sigma))  if stddev.ActiveStatus == 'Enabled':  logging.info(' member '+node\_ip+':'+node\_port+' in enable state')  if sigma[0] > 4:  ratio =commands.getstatusoutput('tmsh list ltm pool '+PoolName+' members {'+MemberIp+'} |grep ratio |grep -v dynamic')  ratio=ratio[1]  ratio = ratio.split()  ratio = ratio[1]  if sigma[1] == -1:  logging.info('offset value is too big,need switch ratio to 10')  os.system('tmsh modify ltm pool '+PoolName +' members modify {'+MemberIp+' {ratio 10 } }')  else:  logging.info(' offset value is too big,need switch ratio to 1')  os.system('tmsh modify ltm pool '+PoolName +' members modify {'+MemberIp+' {ratio 1 } }')  logging.info(' Disable this member static')  stddev.setStatus('Disabled')  stddev.setRatio(ratio)  output = open(inputfilename,'wb')  pickle.dump(stddev,output)  output.close()  else:  logging.info(' member '+node\_ip+':'+node\_port+' in disable state')  if sigma[0] < 2:  logging.info(' offset value is close to old value ,switch ratio to old value:'+stddev.Ratio)  os.system('tmsh modify ltm pool '+PoolName +' members modify {'+MemberIp+' {ratio '+stddev.Ratio+' } }')  logging.info(' Enable this member static')  stddev.setStatus('Enabled')  os.system('rm -f '+ltsfilename)  output = open(inputfilename,'wb')  pickle.dump(stddev,output)  output.close()  print 'up'  except Exception, e:  logging.critical(traceback.format\_exc())  print 'up'  if \_\_name\_\_ == '\_\_main\_\_':  main()  # print MemberStatFrag.GetMemberStatFrag(MonitorVsIp) |

当bigip调用该自定义monitor的时候，会根据所设定的环境变量，获取到该健康检查的需要的

两个参数，分别为

LogLevel 如需要进行debug模式日志记录，

PoolName 该健康检查关联的Pool名称

脚本中

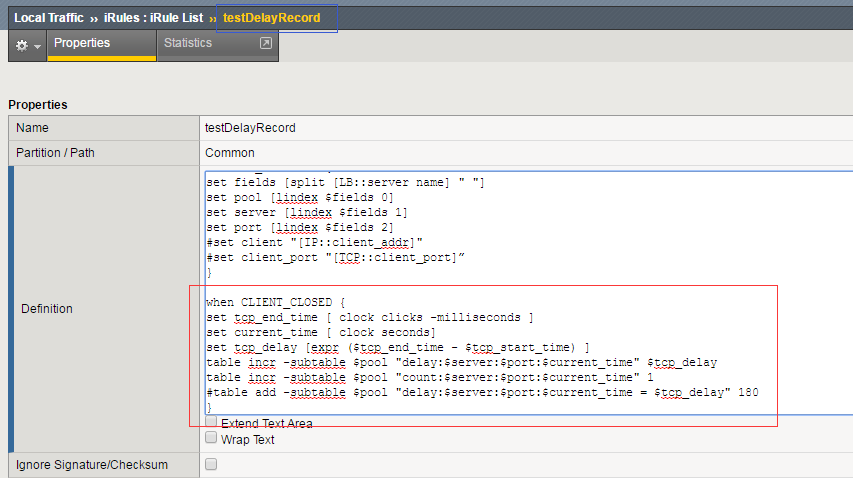
MonitorVsIp = '10.128.5.12'

该参数指定本机session table数据提取VS的IP地址

## 部署步骤

1. 创建延迟统计iRule和延迟展示iRule

延迟统计iRule



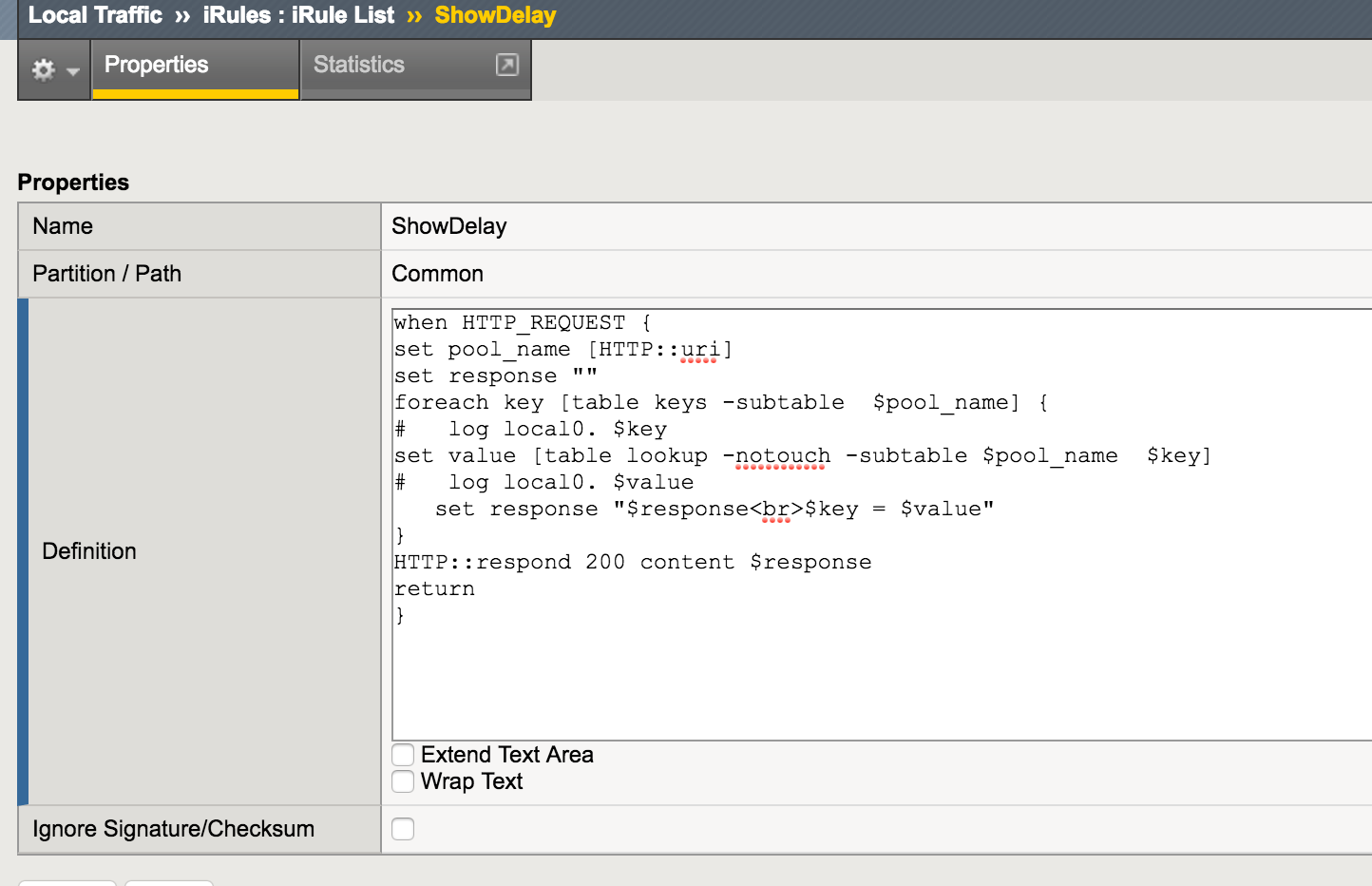
记录三分钟内延迟；

该iRule的命名和脚本中

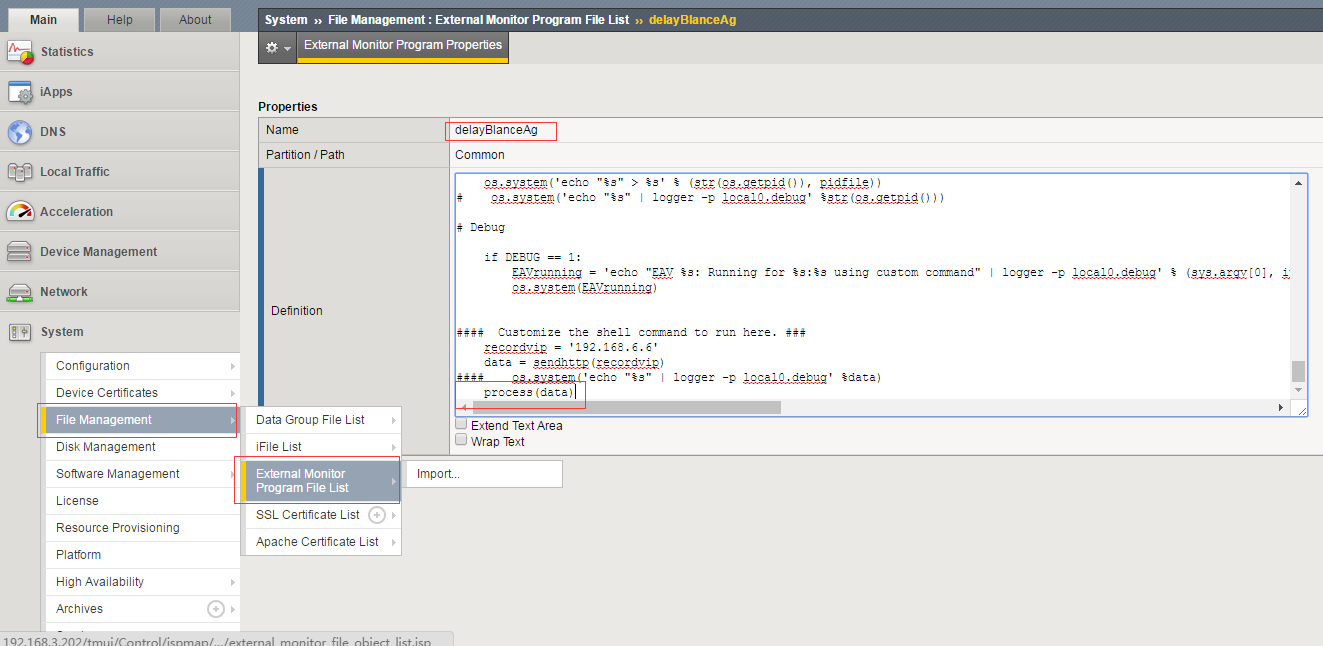
Pools = \_\_GetLtmConfigAndFilterPoolHasRule('/config/bigip.conf', '/Common/calc')

‘Common/calc’所指定的iRule名称一致

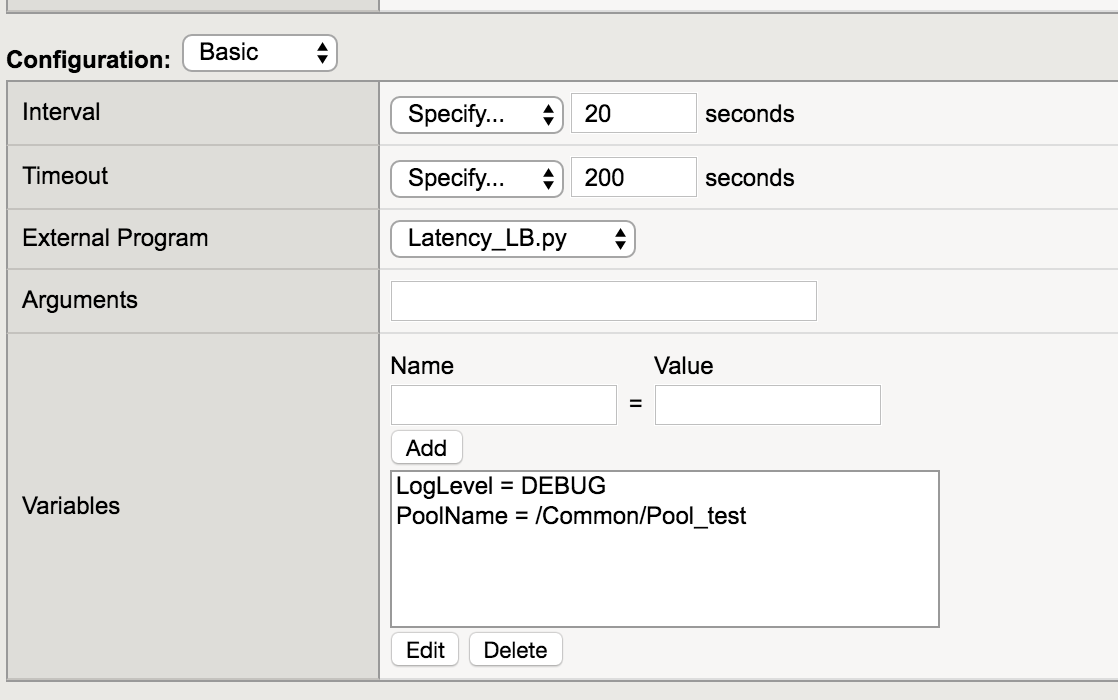
延迟展示iRule



1. 创建延迟展示VS，关联延迟性展示iRule，在需要延迟统计的VS上部署延迟统计iRule
2. 创建外部健康检查EAV

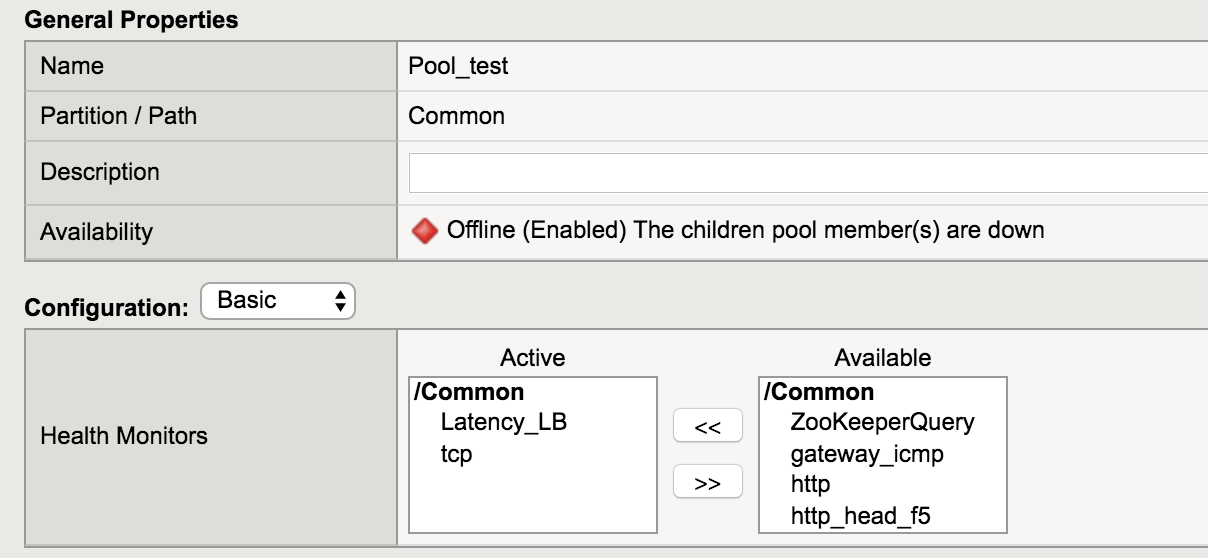


在该EAV中，需要进行下面的配置



请注意在环境变量中需要输入LogLevel和PoolName两个值

1. 在指定的Pool中部署该monitor



注：该自定义EAV它将不会决定一个pool member的状态，只会调整

5.上传长期数据保存脚本和统计脚本至/var/run目录，并且创建如下的crontab条目

\*/20 \* \* \* \* nice -n 19 /config/script/LatencyLongTermSave.py长期保存脚本

\* 2 \* \* \* nice -n 19 /config/script/CalcLatency.py数值计算脚本

开始运行后可以在/var/log/LatencyLB.log文件查看该健康检查运行日志

# 系统运维

本方案的主要思路是，通过iRule记录Pool member的延迟到session table，通过python脚本（/config/script/LatencyLongTermSave.py），将session table中的数据倒出到内存磁盘/dev/shm,通过Python脚本（/config/script/CalcLatency.py）每天定期对前24小时的历史数据计算标准差以及均值，保存在/dev/shm目录，最后外部健康检查脚本EAV，对当前的延迟进行计算，并和历史数据进行对比，决定是否进行比率的动态调整

## 手工计算

在正常工作状态下，可以检查/dev/shm目录下是否存在类似下面的文件

\_Common\_pool\_delay

Calc\_Common\_pool\_delay\_10.128.5.148\_8081

Calc\_Common\_pool\_delay\_10.128.5.148\_8080

其中\_Common\_pool\_delay 为Pool ： pool\_delay的长期延迟数据保存文件

Calc\_Common\_pool\_delay\_10.128.5.148\_8081

为计算后的数据文件，该文件保存Pool member的延迟均值，标准差等信息

上述文件将被EAV所调用，如果不存在，则无法实现本文档所描述的给予延迟的动态比率负载均衡功能

如无长期数据文件，请检查crontab是否有如下的语句

\*/20 \* \* \* \* nice -n 19 /config/script/LatencyLongTermSave.py

可以手工运行/config/script/LatencyLongTermSave.py 进行对前三分钟的session table数据进行保存

如缺少计算后的均值和标准差文件，可以检查crontab是否有如下语句

* 2 \* \* \* nice -n 19 /config/script/CalcLatency.py

可以手工运行/config/script/CalcLatency.py生成统计文件

注意：在EAV运行中，只要有统计文件即可。

## 一般运行日志

EVA所有运行日志保存在/var/log/LatencyLB.log文件中，该日志文件100M为上限，保持7个备份，上述特性通过EAV脚本中的

hdlr = logging.handlers.RotatingFileHandler("/var/log/LatencyLB.log ", maxBytes=1000000, backupCount=7)

指定.

一般日志文件输出内容为：

2016-10-05 04:34:47,919 INFO 3511 10.128.5.148:8081 MonitorVsIp :10.128.5.12 PoolName: /Common/pool\_delay

2016-10-05 04:34:47,919 INFO 3511 10.128.5.148:8081 get Pool: /Common/pool\_delay latency record 2

2016-10-05 04:34:47,920 INFO 3511 10.128.5.148:8081 static stddev:1.33809417124

2016-10-05 04:34:47,920 INFO 3511 10.128.5.148:8081 static avg:13.8064516129

2016-10-05 04:34:47,920 INFO 3511 10.128.5.148:8081 current avg:13

2016-10-05 04:34:47,920 INFO 3511 10.128.5.148:8081 sigma:[0.60268673927295602, -1]

2016-10-05 04:34:47,920 INFO 3511 10.128.5.148:8081 member 10.128.5.148:8081 in enable state

在业务没有流量的情况下会显示如下日志

2016-10-05 05:06:53,868 INFO 7604 10.128.5.148:8080 MonitorVsIp :10.128.5.12 PoolName: /Common/pool\_delay

2016-10-05 05:06:53,868 INFO 7604 10.128.5.148:8080 get Pool: /Common/pool\_delay latency record 0

该EAV在任何情况下都不会影响Pool member的可用性

## 系统切换

系统出现正常切换日志如下

2016-08-26 08:03:10,789 INFO 24093 99.3.0.207:8080 MonitorVsIp :99.1.2.108 PoolName: /Common/PL\_LB\_APP\_8080

2016-08-26 08:03:10,790 INFO 24093 99.3.0.207:8080 get Pool: /Common/PL\_LB\_APP\_8080 latency record 2

2016-08-26 08:03:11,257 INFO 24093 99.3.0.207:8080 static stddev:39.245750379

2016-08-26 08:03:11,258 INFO 24093 99.3.0.207:8080 static avg:528.20720518

2016-08-26 08:03:11,258 INFO 24093 99.3.0.207:8080 current avg:368

2016-08-26 08:03:11,258 INFO 24093 99.3.0.207:8080 sigma:[4.0821542111706508, -1]

2016-08-26 08:03:11,258 INFO 24093 99.3.0.207:8080 member 99.3.0.207:8080 in enable state

2016-08-26 08:03:11,898 INFO 24093 99.3.0.207:8080 offset value is too big,need switch ratio to 4

出现如上日志，系统将停止对已经切换的pool member进行统计计算

延迟值回复到正常范围内后：

］2016-08-26 08:16:50,688 INFO 28257 99.3.0.207:8080 checking member 99.3.0.207:8080

2016-08-26 08:16:50,690 INFO 28258 99.12.232.123:8080 checking member 99.12.232.123:8080

2016-08-26 08:16:51,771 INFO 28258 99.12.232.123:8080 MonitorVsIp :99.1.2.108 PoolName: /Common/PL\_LB\_APP\_8080

2016-08-26 08:16:51,772 INFO 28258 99.12.232.123:8080 get Pool: /Common/PL\_LB\_APP\_8080 latency record 2

2016-08-26 08:16:51,781 INFO 28257 99.3.0.207:8080 MonitorVsIp :99.1.2.108 PoolName: /Common/PL\_LB\_APP\_8080

2016-08-26 08:16:51,781 INFO 28257 99.3.0.207:8080 get Pool: /Common/PL\_LB\_APP\_8080 latency record 2

2016-08-26 08:16:52,109 INFO 28258 99.12.232.123:8080 static stddev:11.8357933072

2016-08-26 08:16:52,109 INFO 28258 99.12.232.123:8080 static avg:61.2109658224

2016-08-26 08:16:52,109 INFO 28258 99.12.232.123:8080 current avg:205

2016-08-26 08:16:52,109 INFO 28258 99.12.232.123:8080 sigma:[12.148660461186399, 1]

2016-08-26 08:16:52,109 INFO 28258 99.12.232.123:8080 member 99.12.232.123:8080 in disable state

2016-08-26 08:16:52,126 INFO 28257 99.3.0.207:8080 static stddev:39.245750379

2016-08-26 08:16:52,126 INFO 28257 99.3.0.207:8080 static avg:528.20720518

2016-08-26 08:16:52,126 INFO 28257 99.3.0.207:8080 current avg:509

2016-08-26 08:16:52,127 INFO 28257 99.3.0.207:8080 sigma:[0.48940853454579203, -1]

2016-08-26 08:16:52,127 INFO 28257 99.3.0.207:8080 member 99.3.0.207:8080 in disable state

2016-08-26 08:16:52,127 INFO 28257 99.3.0.207:8080 offset value is close to old value ,switch ratio to old value:1

2016-08-26 08:16:52,444 INFO 28257 99.3.0.207:8080 Enable this member static

恢复回原状态后，将重新将该member置为可计算状态

## 系统重启

如设备出现重启，/dev/shm下Calc开头的文件将自动消失，此时需要等待过凌晨2点才会自动生成文件，如需要提前生成文件，可以手工执行

/config/script/CalcLatency.py

运行结束后将在/dev/shm生成Calc开头的统计文件，此时可供EAV调用并正常运行

## 统计文件缺失

在/dev/shm下的Calc开头的文件对一个每一个Pool Member的延迟统计数据， 该文件直接关系到是否根据延迟进行调整，/config/script/CalcLatency.py可以自动生成统计文件，统计文件的数据来源为/dev/shm下的\_Common开头的长期数据保存文件，在下面两种情况下，无法正确生成统计信息文件

* 如果缺少\_Common文件，运行/config/script/CalcLatency.py不会生成任何统计信息文件
* 如果发生过首次切换，为避免切换后的数据污染之前的原始数据，运/config/script/CalcLatency.py将不会计算,在日志中存在disbale state字样，择表明该member已经发生过首次切换，正在等待回切，如此时需要重置，可以手工删除/dev/shm对应pool和pool member的文件，等待12小时以后，手工或者自动运行/config/script/CalcLatency.py

# 总结

通过上述方案，可以实现服务器的流量分配和服务器的延迟保持动态同步调整，可以应用下面两种场合

1 .双中心应用服务器到数据库服务器存在固定延迟的情况下，数据库服务器进行主备中心切换后，APP服务器无需手工调整，在极短的时间内实现流量比例的自动切换

2.在单台应用服务器由于工作异常出现较高延迟或者处理队列的时候，实时动态调整异常服务器的分发流量比例，消除单台服务器高延迟带来对用户体验的冲击