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 3

2 部署方案 4

2.1 模块关系图 5

2.2 组件明细 6

2.2.1 延迟原始数据iRule 6

2.2.2 延迟展示iRule 9

2.2.3 长期数据保存脚本 10

2.2.4 长期数据统计脚本 12

2.2.5 EAV健康检查 19

2.3 部署步骤 29

3 现网测试 32

3.1 测试拓扑 32

3.2 测试过程 33

3.3 测试结果 41

4 总结 41

# 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

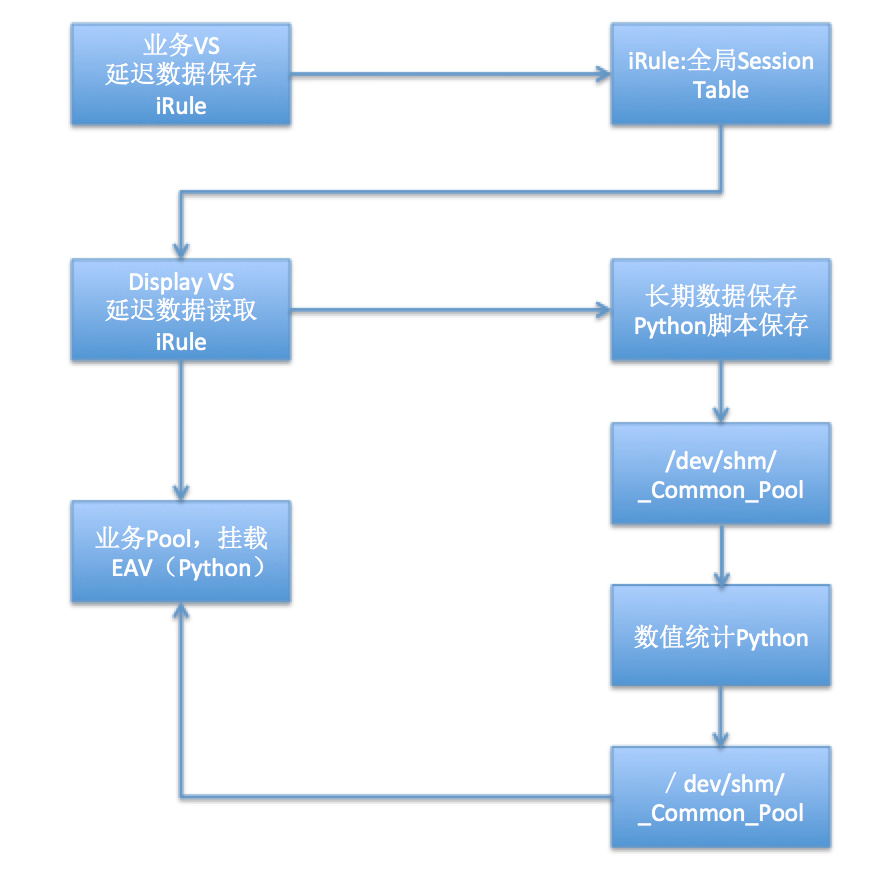
累计均值，标准差进行对比，在4个标准差范围内认为该服务器延迟处于正常范围，如超过

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 pool\_name [HTTP::uri]  set response ""  foreach key [table keys -subtable $pool\_name] {  # log local0. $key  set value [table lookup -notouch -subtable $pool\_name $key]  # log local0. $value  set response "$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]

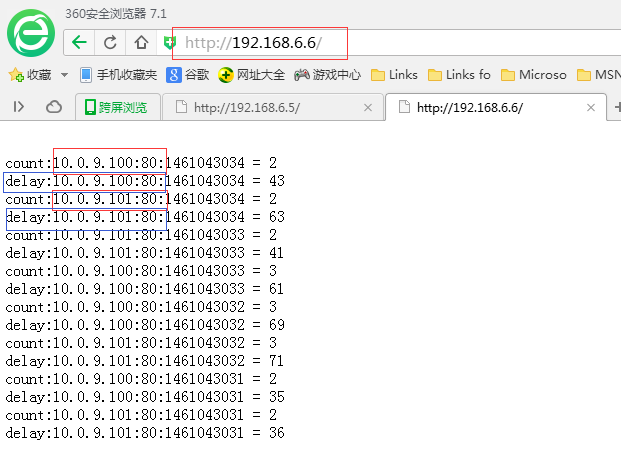
获取pool下所有member的

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

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  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()  if \_\_name\_\_ == '\_\_main\_\_':  MonitorVsIp = '10.128.5.12'  Pools = ['/Common/Pool\_test']  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() |

目前该脚本计算的pool需要在脚本中写，修改

Pools = ['/Common/Pool\_test']

这一行即可，Pools为数组对象，可以通过append方法添加每次需要保存长期数据的Pool名称即可

### 长期数据统计脚本

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  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]  if \_\_name\_\_ == '\_\_main\_\_':  pathname= '/dev/shm/'  Pools = ['/Common/Pool\_test']  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(pathname+'Calc\_Common\_Pool\_test'+'\_'+member.replace(':','\_'),'wb')  pickle.dump(stddev,outputfile)  outputfile.close() |

同样需要在

Pools = ['/Common/Pool\_test']

这一行中添加其他Pool，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

### 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)  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  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() |

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

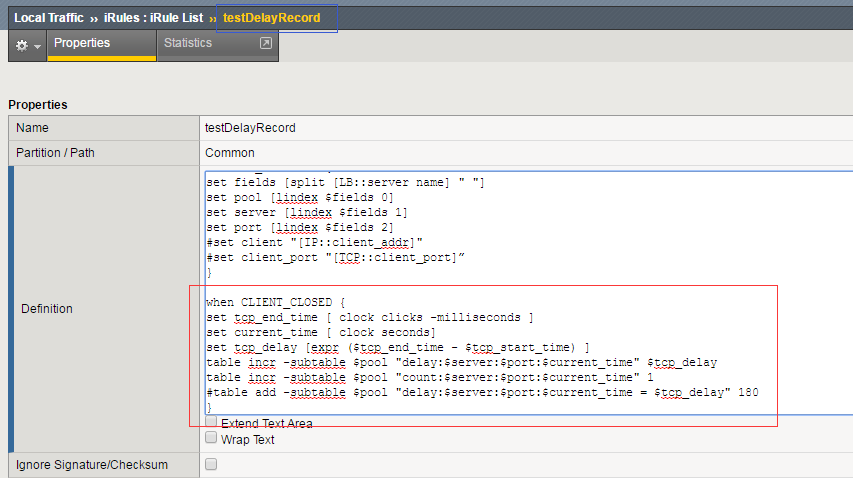
LogLevel 如需要进行debug模式日志记录，

PoolName 该健康检查关联的Pool名称

## 部署步骤

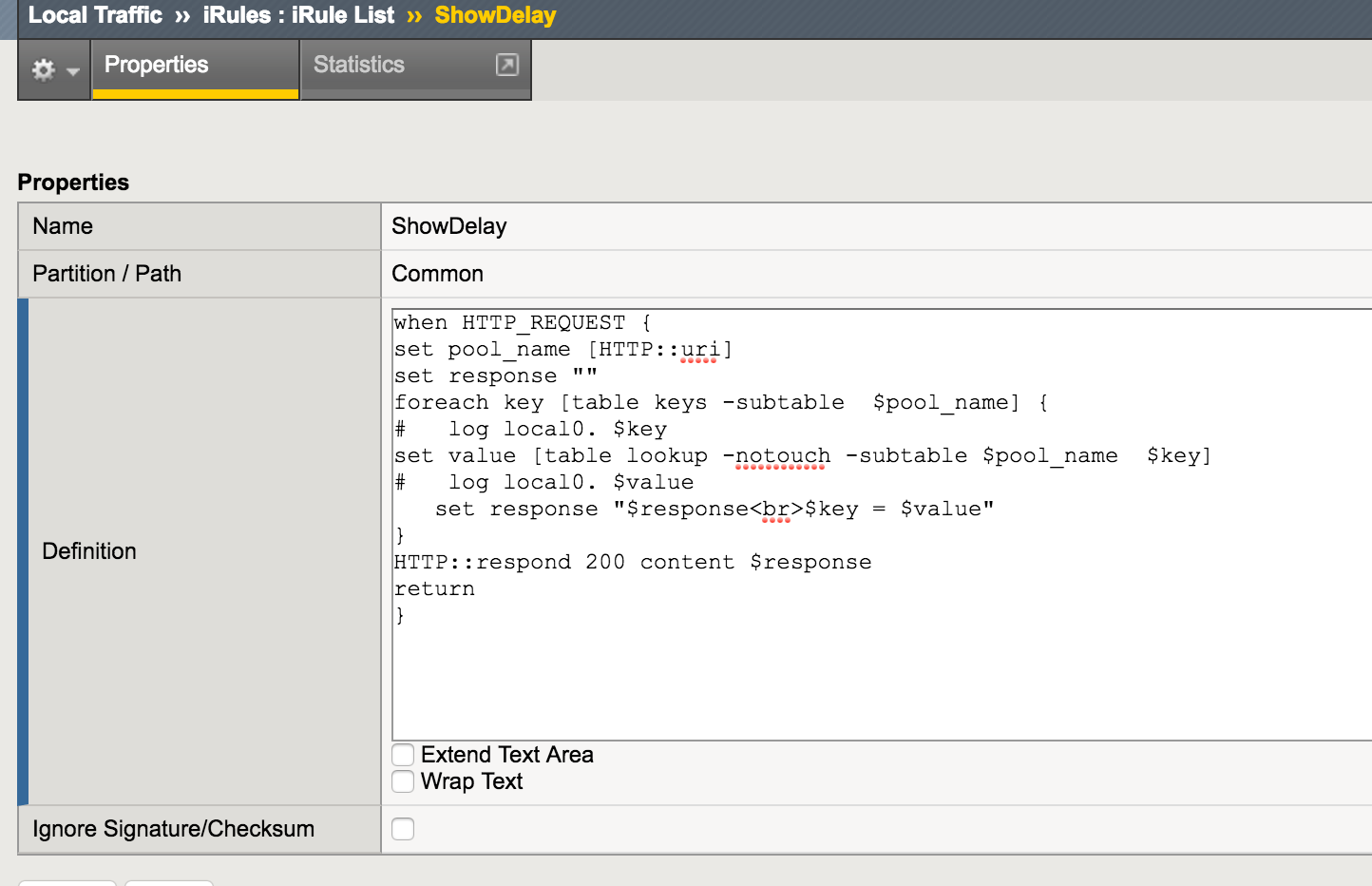
1. 创建延迟统计iRule和延迟展示iRule

延迟统计iRule

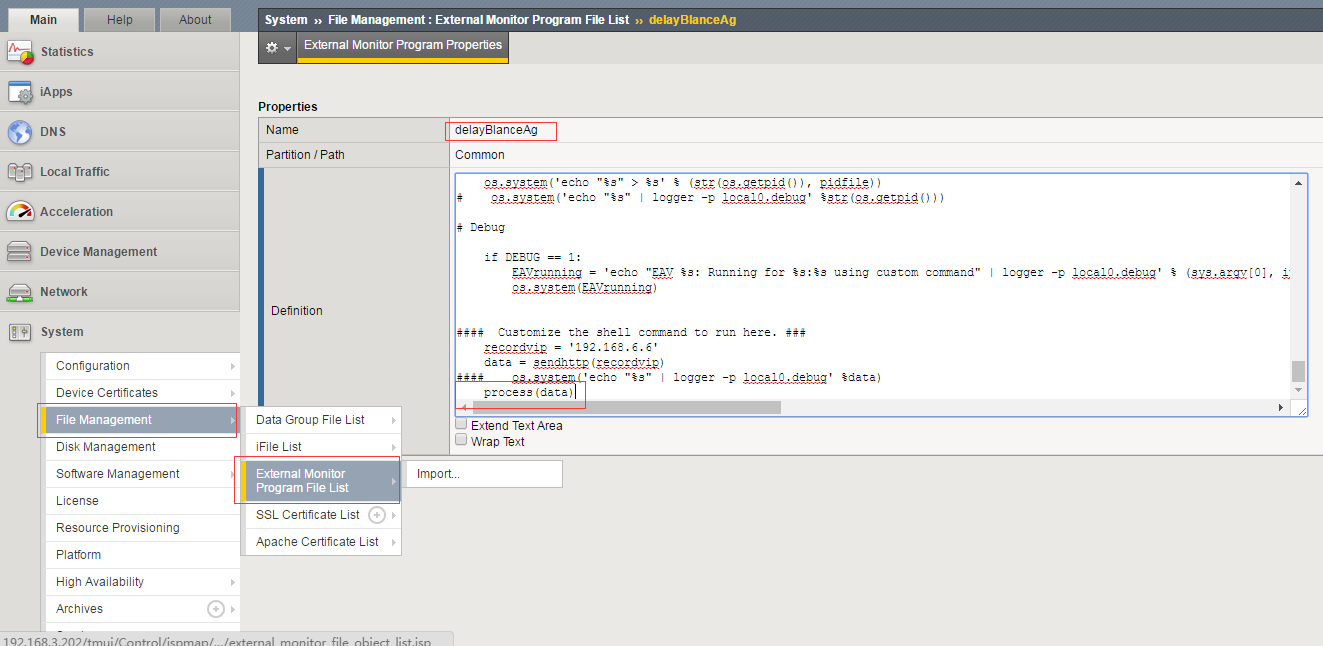


记录三分钟内延迟；

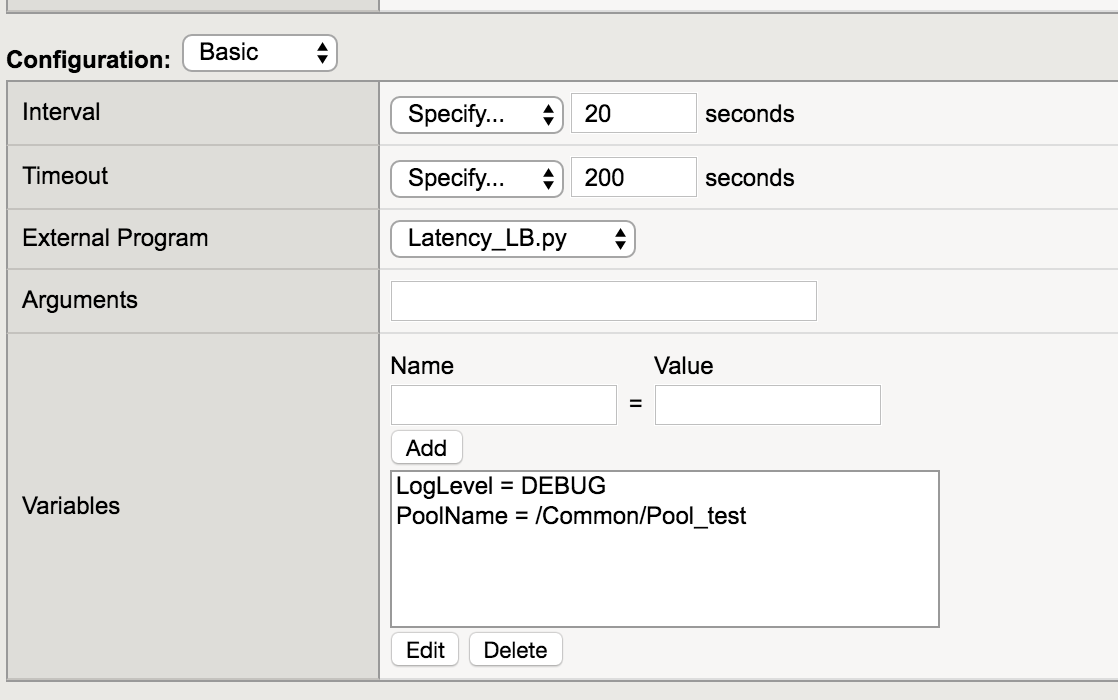
延迟展示iRule



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

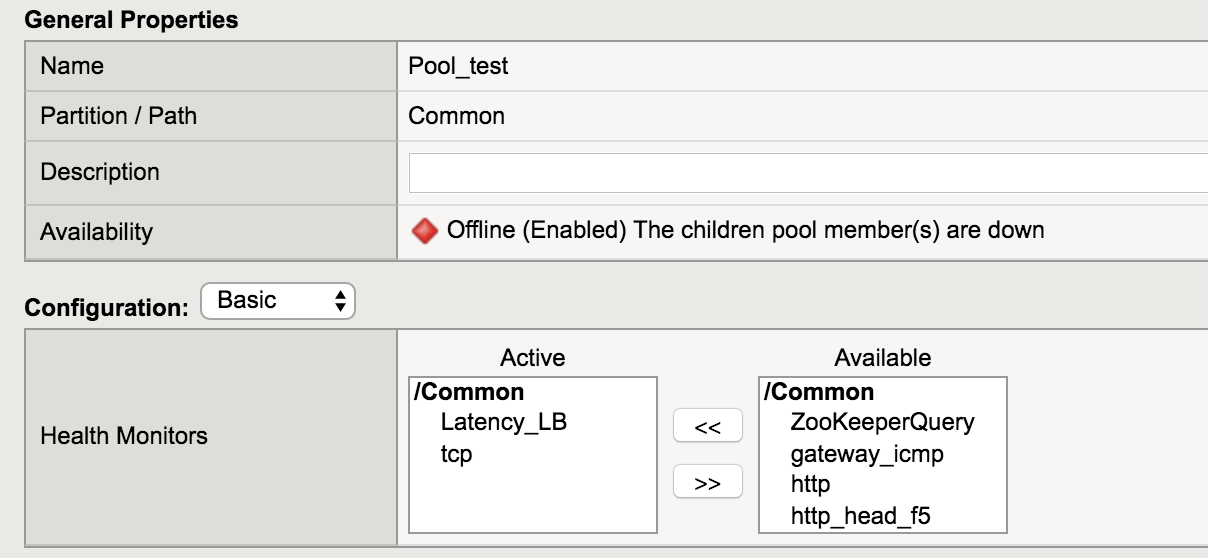


在该EAV中，需要进行下面的配置



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/save.py 长期保存脚本

\* 2 \* \* \* nice -n 19 /config/script/calc.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

# 现网测试

## 测试条件

### 测试设备

（1）在深圳数据中心部署F5负载均衡，用于探测两地app交易业务的延迟，用于控制延迟负载均衡的计算和对负载比例的自动调控，部署业务VS以及探测VS，部署延迟性负载均衡自动化工具。

（2）在深圳、上海两个数据中心分别部署app前端web（部署tomcat服务器），分别部署app后端数据库（mysql5.1.7）。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **设备名称** | **设备数量** | **设备配置** | **软件版本** | **备注** |
| F5负载均衡 | 1 | bigip-2000 | 11.4.1 |  |
| 部署app前端 | 2 | Memory 8G | Centos6.5 kernel2.6 tomcat6 |  |
| 部署mysql数据库 | 2 | Memory 8G | Centos6.5 kernel2.6 mysql5.1.7 |  |

### 测试组网拓扑



### 测试地址表

部署清单：

深圳数据中心业务入口VS（VS\_LB\_APP\_8080）：99.1.2.45：8080

深圳数据中心延迟探测VS：99.1.2.108：80

深圳web：99.12.232.123：8080

深圳db：99.3.232.129：3306

上海web：99.3.0.207：8080

上海db：99.3.0.135：3306

|  |  |  |
| --- | --- | --- |
| **业务名称** | **业务类型** | **ip地址和端口** |
| VS\_LB\_APP\_8080 | 深圳数据中心业务入口VS | 99.1.2.45：8080 |
| 延迟探测VS | 深圳数据中心延迟探测VS | 99.1.2.108：80 |
| 深圳web | 深圳数据中心app前端 | 99.12.232.123：8080 |
| 深圳db | 深圳数据中心db | 99.3.232.129：3306 |
| 上海web | 上海数据中心app前端 | 99.3.0.207：8080 |
| 上海db | 上海数据中心db | 99.3.0.135：3306 |

## 测试过程

### 基础网络架构和延迟分析

|  |  |
| --- | --- |
| 测试项目：基础环境下的网络架构以及延迟情况 | |
| 测试目的：分析正常情况下数据库未切换，APP前端位于深圳，DB位于深圳的正常延迟情况分析 | |
| 预测结果：正常状态下配置，深圳APP和上海APP都会处理不同比率的流量，数据库在深圳的情况下比率为F5配置中所示4:1，即大部分流量由深圳APP处理，访问效果较好，上海只处理小部分流量，确认APP端可用。 | |
| （1）通过流量测试工具loadrunner对BIG-IP业务VS进行流量测试。    （2）BIG-IP会将APP层pool member的延迟数据（每秒钟延迟）记录在BIGIP系统的内存中，系统中部署自动化脚本  \*/3 \* \* \* \* python /var/tmp/latency/LatencyLongTermSave.py >> /var/tmp/err\_save.log 2>&1  将延迟数据以大数据形式记录在/dev/shm内存目录下。    通过该探测脚本，BIG-IP获取并计算了所有APP前端的延迟情况，保存在  /var/tmp/latency目录下的二进制文件：  Calc\_Common\_PL\_LB\_APP\_8080\_99.3.0.207\_8080  Calc\_Common\_PL\_LB\_APP\_8080\_99.12.232.123\_8080  其中Calc\_Common\_PL\_LB\_APP\_8080\_99.12.232.123\_8080记录了深圳APP前端完成交易所需要用到的各个连接需要用到的延迟的概况。  如图：    Calc\_Common\_PL\_LB\_APP\_8080\_99.12.232.123\_8080记录了上海APP前端完成交易所需要用到的各个连接需要用到的延迟的概况。  如图：    （3）根据获取到的大量数据计算各个pool member在正常状态下（app在深圳，db在深圳）的延迟数据，包括各个时间段内延迟的变化状态，变化量的多少。  python /var/tmp/latency/CalcLatency.py  显示数据如下：   |  | | --- | | 2016-07-27 10:48:53,967 INFO 30513 99.3.0.207:8080 checking member 99.3.0.207:8080 | | 2016-07-27 10:48:53,968 INFO 30514 99.12.232.123:8080 checking member 99.12.232.123:8080 | | 2016-07-27 10:48:54,088 INFO 30514 99.12.232.123:8080 MonitorVsIp :99.1.2.108 PoolName: /Common/PL\_LB\_APP\_8080 | | 2016-07-27 10:48:54,089 INFO 30514 99.12.232.123:8080 get Pool: /Common/PL\_LB\_APP\_8080 latency record 2 | | 2016-07-27 10:48:54,561 INFO 30514 99.12.232.123:8080 static stddev:11.3727617685 | | 2016-07-27 10:48:54,561 INFO 30514 99.12.232.123:8080 static avg:63.4631691417 | | 2016-07-27 10:48:54,561 INFO 30514 99.12.232.123:8080 current avg:64 | | 2016-07-27 10:48:54,561 INFO 30514 99.12.232.123:8080 sigma:[0.04720320967374033, 1] | | 2016-07-27 10:48:54,562 INFO 30514 99.12.232.123:8080 member 99.12.232.123:8080 in enable state | | 2016-07-27 10:48:55,233 INFO 30513 99.3.0.207:8080 MonitorVsIp :99.1.2.108 PoolName: /Common/PL\_LB\_APP\_8080 | | 2016-07-27 10:48:55,233 INFO 30513 99.3.0.207:8080 get Pool: /Common/PL\_LB\_APP\_8080 latency record 2 | | 2016-07-27 10:48:55,705 INFO 30513 99.3.0.207:8080 static stddev:39.8760374715 | | 2016-07-27 10:48:55,705 INFO 30513 99.3.0.207:8080 static avg:532.321732669 | | 2016-07-27 10:48:55,705 INFO 30513 99.3.0.207:8080 current avg:527 | | 2016-07-27 10:48:55,731 INFO 30513 99.3.0.207:8080 sigma:[0.13345690811124317, -1] | | 2016-07-27 10:48:55,731 INFO 30513 99.3.0.207:8080 member 99.3.0.207:8080 in enable state |   根据BIG-IP上的CalcLatency.py计算模块经过大量延迟数据的计算，记录的日志：  能够发现  Client->业务VS->深圳APP->深圳DB的访问延迟状况如下   |  | | --- | | 长期记录延迟的平均值为static avg:63.4631691417 | | 长期记录延迟的标准差为static stddev:11.3727617685 | | 现时探测链接的平均值为current avg:64 | | 根据记录大数据计算正态分布参数sigma:[0.04720320967374033, 1] |   Clent->业务VS->上海APP->深圳DB的访问延迟状况如下   |  | | --- | | 长期记录延迟的平均值为static avg:532.321732669 | | 长期记录延迟的标准差为static stddev:39.8760374715 | | 现时探测链接的平均值为current avg:527 | | 根据记录大数据计算正态分布参数sigma:[0.13345690811124317, -1] | | |
| 测试结果：正常状态下配置，深圳APP和上海APP都会处理不同比率的流量，数据库在深圳的情况下比率为F5配置中所示4:1，即大部分流量由深圳APP处理，访问效果较好，上海只处理小部分流量，确认APP端可用。  符合预期。 | |
| 客户工程师签字： | F5工程师签字： |

### 测试场景一：数据库切换

|  |  |
| --- | --- |
| 测试项目：测试数据库由深圳切换至上海过程中的延迟负载均衡情况 | |
| 测试目的：测试数据库由深圳切换至上海，客户端延迟发生变化，负载均衡算法进行流量分配的结果 | |
| 预期预期：  当数据库发生切换，根据延迟值的情况，BIG-IP调整广州和深圳APP端处理流量的比例RATIO。如数据库切换至上海，后台数据显示上海APP端能够获得更好的数据体验，BIG-IP系统将调整较大的RATIO至上海的APP，提高用户交易体验。 | |
| 测试过程：  当数据库发生切换，测试过程中模拟数据库连接池的切换。  通过修改配置文件：  url="jdbc:mysql://99.12.232.129:3306/javatest"  将深圳的db地址切换至上海99.3.0.135：3306  如图：    此时延迟状态数据如下：   |  | | --- | | 2016-07-27 10:52:34,009 INFO 31818 99.12.232.123:8080 MonitorVsIp :99.1.2.108 PoolName: /Common/PL\_LB\_APP\_8080 | | 2016-07-27 10:52:34,010 INFO 31818 99.12.232.123:8080 get Pool: /Common/PL\_LB\_APP\_8080 latency record 2 | | 2016-07-27 10:52:34,640 INFO 31818 99.12.232.123:8080 static stddev:11.3727617685 | | 2016-07-27 10:52:34,640 INFO 31818 99.12.232.123:8080 static avg:63.4631691417 | | 2016-07-27 10:52:34,640 INFO 31818 99.12.232.123:8080 current avg:121 | | 2016-07-27 10:52:34,641 INFO 31818 99.12.232.123:8080 sigma:[5.0591784150120302, 1] | | 2016-07-27 10:52:34,641 INFO 31818 99.12.232.123:8080 member 99.12.232.123:8080 in enable state | | 2016-07-27 10:52:35,403 INFO 31818 99.12.232.123:8080 offset value is too big,need switch ratio to 1 | | 2016-07-27 10:53:35,246 INFO 32174 99.3.0.207:8080 MonitorVsIp :99.1.2.108 PoolName: /Common/PL\_LB\_APP\_8080 | | 2016-07-27 10:53:35,246 INFO 32174 99.3.0.207:8080 get Pool: /Common/PL\_LB\_APP\_8080 latency record 2 | | 2016-07-27 10:53:35,736 INFO 32174 99.3.0.207:8080 static stddev:39.8760374715 | | 2016-07-27 10:53:35,736 INFO 32174 99.3.0.207:8080 static avg:532.321732669 | | 2016-07-27 10:53:35,736 INFO 32174 99.3.0.207:8080 current avg:343 | | 2016-07-27 10:53:35,737 INFO 32174 99.3.0.207:8080 sigma:[4.7477569155204415, -1] | | 2016-07-27 10:53:35,737 INFO 32174 99.3.0.207:8080 member 99.3.0.207:8080 in enable state | | 2016-07-27 10:53:36,098 INFO 32174 99.3.0.207:8080 offset value is too big,need switch ratio to 4 | | 2016-07-27 10:53:36,446 INFO 32174 99.3.0.207:8080 Disable this member static |   根据BIG-IP上的CalcLatency.py计算模块经过大量延迟数据的计算，记录的日志：  能够发现  Client->业务VS->深圳APP->上海DB的访问延迟状况如下   |  | | --- | | 长期记录延迟的平均值为static avg:63.4631691417f | | 长期记录延迟的标准差为static stddev:11.3727617685 | | 现时探测链接的平均值为current avg:121 | | 根据记录大数据计算正态分布参数sigma:[5.0591784150120302, 1] |   从日志可以看出，自动化的根据延迟负载均衡调整了往深圳APP的比率  2016-07-27 10:52:35,403 INFO 31818 99.12.232.123:8080 offset value is too big,need switch ratio to 1  Client->业务VS->上海APP->上海DB的访问延迟状况如下   |  | | --- | | 长期记录延迟的平均值为static avg:532.321732669 | | 长期记录延迟的标准差为static stddev:39.8760374715 | | 现时探测链接的平均值为current avg:343 | | 根据记录大数据计算正态分布参数sigma:[4.7477569155204415, -1] |   从日志可以看出，自动化的根据延迟负载均衡调整了往上海APP的比率  2016-07-27 10:53:36,098 INFO 32174 99.3.0.207:8080 offset value is too big,need switch ratio to 4  此时虽然数据库切换至上海数据中心，延迟负载均衡算法根据现在的延迟状态将流量比率进行了切换，避免了从深圳app访问上海数据库的延迟过大，自动化的减少了管理员的切换动作。  此时负责健康检查探测的自动化的monitor将根据规则对pool当中的member流量状态进行重新分配。 | |
| 测试结果：  当数据库发生切换，根据延迟值的情况，BIG-IP调整广州和深圳APP端处理流量的比例RATIO。如数据库切换至上海，后台数据显示上海APP端能够获得更好的数据体验，BIG-IP系统将调整较大的RATIO至上海的APP，提高用户交易体验。  符合预期 | |
| 客户工程师签字： | F5工程师签字： |

### 测试场景二：数据库回切

|  |  |
| --- | --- |
| 测试项目：在场景1的情况下，数据库由上海db回切至深圳db | |
| 测试目的：验证在回切情况下，根据延迟负载均衡的调度算法能否重新分配流量 | |
| 预期结果：  重新调整了流量比率，让大部分流量从深圳APP访问。 | |
| 测试过程：  当数据库从上海切换回深圳，根据目前的方案逻辑，如本地App的延迟恢复到和原来统计数据一个标准差范围内，就将该服务器的分配比例值调整为原比例值    延迟数据如下：   |  | | --- | | 2016-07-27 11:03:35,284 INFO 3456 99.3.0.207:8080 MonitorVsIp :99.1.2.108 PoolName: /Common/PL\_LB\_APP\_8080 | | 2016-07-27 11:03:35,285 INFO 3456 99.3.0.207:8080 get Pool: /Common/PL\_LB\_APP\_8080 latency record 2 | | 2016-07-27 11:03:35,800 INFO 3456 99.3.0.207:8080 static stddev:39.8760374715 | | 2016-07-27 11:03:35,800 INFO 3456 99.3.0.207:8080 static avg:532.321732669 | | 2016-07-27 11:03:35,800 INFO 3456 99.3.0.207:8080 current avg:475 | | 2016-07-27 11:03:35,800 INFO 3456 99.3.0.207:8080 sigma:[1.4374982145529731, -1] | | 2016-07-27 11:03:35,800 INFO 3456 99.3.0.207:8080 member 99.3.0.207:8080 in disable state | | 2016-07-27 11:03:35,801 INFO 3456 99.3.0.207:8080 offset value is close to old value ,switch ratio to old value:1 | | 2016-07-27 11:03:36,487 INFO 3456 99.3.0.207:8080 Enable this member static | | 2016-07-27 11:03:55,773 INFO 3580 99.12.232.123:8080 static stddev:11.3727617685 | | 2016-07-27 11:03:55,773 INFO 3580 99.12.232.123:8080 static avg:63.4631691417 | | 2016-07-27 11:03:55,773 INFO 3580 99.12.232.123:8080 current avg:76 | | 2016-07-27 11:03:55,774 INFO 3580 99.12.232.123:8080 sigma:[1.1023558844818013, 1] | | 2016-07-27 11:03:55,774 INFO 3580 99.12.232.123:8080 member 99.12.232.123:8080 in disable state | | 2016-07-27 11:03:55,774 INFO 3580 99.12.232.123:8080 offset value is close to old value ,switch ratio to old value:4 | | 2016-07-27 11:03:56,082 INFO 3580 99.12.232.123:8080 Enable this member static |   Client->业务VS->深圳APP->深圳DB的访问延迟状况如下   |  | | --- | | 长期记录延迟的平均值为static avg:63.4631691417 | | 长期记录延迟的标准差为static stddev:11.3727617685 | | 现时探测链接的平均值为current avg:76 | | 根据记录大数据计算正态分布参数sigma:[1.1023558844818013, 1] |   延迟负载均衡判断，业务访问的延迟大大降低，需要将更大的流量比率切换至深圳APP；  Client->业务VS->上海APP->深圳DB的访问延迟状况如下   |  | | --- | | 长期记录延迟的平均值为static avg:532.321732669 | | 长期记录延迟的标准差为static stddev:39.8760374715 | | 现时探测链接的平均值为current avg:475 | | 根据记录大数据计算正态分布参数sigma:[1.4374982145529731, -1] |   延迟负载均衡判断，从上海APP访问的延迟大大提升，需要将更少的流量比率切换至上海APP。  BIG-IP自动化探测将再次自动调整Pool的比率，使大部分流量通过深圳进行处理，提高用户体验。  如日志所示   |  | | --- | | 2016-07-27 11:03:35,801 INFO 3456 99.3.0.207:8080 offset value is close to old value ,switch ratio to old value:1 | | 2016-07-27 11:03:55,774 INFO 3580 99.12.232.123:8080 offset value is close to old value ,switch ratio to old value:4 | | |
| 测试结果：  延迟负载均衡算法根据延迟重新调整了流量比率，让大部分流量从深圳APP访问。  使得大部分比例的用户流量走延迟较低的一个数据中心。 | |
| 客户工程师签字： | F5工程师签字： |

### 测试场景二：应用延迟单边上升

#### 本地服务器单边上升

|  |  |
| --- | --- |
| 测试项目：当本地服务延迟单边上升，延迟负载均衡的流量调整 | |
| 测试目的：测试当本地服务延迟单边上升，延迟负载均衡算法能否根据延迟进行流量调整 | |
| 预期结果：  深圳服务器延迟大量上升，负载均衡算法将均衡两地的访问量，待服务器延迟回复后，回复至正常状态 | |
| 测试过程：  方差基值如下：   |  | | --- | | sample count is:912 | | save 99.12.232.123:8080 recode 912 | | the avg value is 63.125 | | the stddev value is 10.0230955885 | | sample count is:913 | | save 99.3.0.207:8080 recode 913 | | the avg value is 529.743702081 | | the stddev value is 60.2340635897 |   修改服务器99.12.232.123上的页面代码，代码，将延迟修改从50修改为350，修改代码如下  java.lang.Thread.sleep(50);-》java.lang.Thread.sleep(350);  在半小时后可以在EAV Log上可以看到如下日志   |  | | --- | | 2016-08-03 16:08:42,553 INFO 16461 99.12.232.123:8080 static avg:63.125 | | 2016-08-03 16:08:42,553 INFO 16461 99.12.232.123:8080 current avg:108 | | 2016-08-03 16:08:42,553 INFO 16461 99.12.232.123:8080 sigma:[4.4771597361257545, 1] | | 2016-08-03 16:08:42,553 INFO 16461 99.12.232.123:8080 member 99.12.232.123:8080 in enable state | | 2016-08-03 16:08:42,896 INFO 16461 99.12.232.123:8080 offset value is too big,need switch ratio to 1 | | 2016-08-03 16:08:43,190 INFO 16461 99.12.232.123:8080 Disable this member static |   显示出现延迟异常升高，本地服务器的权重值调整为1    此后，将本地服务器代码进行调整  java.lang.Thread.sleep(350);-》java.lang.Thread.sleep(350);  恢复成原延迟范围  BIGIP后台日志显示   |  | | --- | | 2016-08-03 16:18:41,538 INFO 20284 99.12.232.123:8080 checking member 99.12.232.123:8080 | | 2016-08-03 16:18:41,653 INFO 20284 99.12.232.123:8080 MonitorVsIp :99.1.2.108 PoolName: /Common/PL\_LB\_APP\_8080 | | 2016-08-03 16:18:41,653 INFO 20284 99.12.232.123:8080 get Pool: /Common/PL\_LB\_APP\_8080 latency record 2 | | 2016-08-03 16:18:41,672 INFO 20284 99.12.232.123:8080 static stddev:10.0230955885 | | 2016-08-03 16:18:41,673 INFO 20284 99.12.232.123:8080 static avg:63.125 | | 2016-08-03 16:18:41,673 INFO 20284 99.12.232.123:8080 current avg:68 | | 2016-08-03 16:18:41,673 INFO 20284 99.12.232.123:8080 sigma:[0.48637668442591758, 1] | | 2016-08-03 16:18:41,674 INFO 20284 99.12.232.123:8080 member 99.12.232.123:8080 in disable state | | 2016-08-03 16:18:41,675 INFO 20284 99.12.232.123:8080 offset value is close to old value ,switch ratio to old value:4 | | 2016-08-03 16:18:42,132 INFO 20284 99.12.232.123:8080 Enable this member static |   显示根据延迟调整回原来的比率值  此时F5的配置调整为 | |
| 测试结果：  深圳服务器延迟大量上升，负载均衡算法将均衡两地的访问量，待服务器延迟回复后，回复至正常状态.  符合预期结果。 | |
| 客户工程师签字： | F5工程师签字： |

#### 异地服务器单边上升

|  |  |
| --- | --- |
| 测试项目：当异地服务延迟单边上升，延迟负载均衡的流量调整 | |
| 测试目的：测试当异地服务延迟单边上升，延迟负载均衡算法能否根据延迟进行流量调整 | |
| 预期结果：  异地服务器的延迟上升，不会影响正常负载均衡的比率，用户依然获得正常的用户体验。 | |
| 测试过程：  同样通过调整服务器代码，将99.3.0.207的延迟代码从50ms调整为350ms  EAV日志：   |  | | --- | | 2016-08-03 16:27:42,596 INFO 23555 99.3.0.207:8080 static stddev:60.2340635897 | | 2016-08-03 16:27:42,596 INFO 23555 99.3.0.207:8080 static avg:529.743702081 | | 2016-08-03 16:27:42,596 INFO 23555 99.3.0.207:8080 current avg:778 | | 2016-08-03 16:27:42,596 INFO 23555 99.3.0.207:8080 sigma:[4.1215266432982229, 1] | | 2016-08-03 16:27:42,597 INFO 23555 99.3.0.207:8080 member 99.3.0.207:8080 in enable state | | 2016-08-03 16:27:43,118 INFO 23555 99.3.0.207:8080 offset value is too big,need switch ratio to 1 | | 2016-08-03 16:27:43,490 INFO 23555 99.3.0.207:8080 Disable this member static |   此时远端服务器被调整成低挡位，但由于远端初始化就是在低挡位运行，所以ratio依旧为1  在异地服务器延迟降低到正常水平的时候   |  | | --- | | 2016-08-03 16:31:22,699 INFO 24886 99.3.0.207:8080 static stddev:60.2340635897 | | 2016-08-03 16:31:22,699 INFO 24886 99.3.0.207:8080 static avg:529.743702081 | | 2016-08-03 16:31:22,699 INFO 24886 99.3.0.207:8080 current avg:641 | | 2016-08-03 16:31:22,699 INFO 24886 99.3.0.207:8080 sigma:[1.8470661165557964, 1] | | 2016-08-03 16:31:22,699 INFO 24886 99.3.0.207:8080 member 99.3.0.207:8080 in disable state | | 2016-08-03 16:31:22,700 INFO 24886 99.3.0.207:8080 offset value is close to old value ,switch ratio to old value:1 | | 2016-08-03 16:31:23,131 INFO 24886 99.3.0.207:8080 Enable this member static |   执行结果ratio依旧为1. | |
| 测试结果：  异地服务器的延迟上升，不会影响正常负载均衡的比率，用户依然获得正常的用户体验。  符合预期。 | |
| 客户工程师签字： | F5工程师签字： |

### 测试场景二：应用延迟单边下降

#### 本地服务器单边下降

|  |  |
| --- | --- |
| 测试项目：测试本地服务器（深圳）的延迟发生单边下降情况下，延迟负载均衡的流量调整 | |
| 测试目的：测试当本地服务延迟单边下降，延迟负载均衡算法能否根据延迟进行流量调整 | |
| 预期结果：  本地服务器的延迟下降，不会影响正常负载均衡的比率，用户依然获得正常的用户体验。 | |
| 测试过程：  通过代码修改，将本地服务器原来的固定50毫秒延迟修改为0  EVA日志为：   |  | | --- | | 2016-08-03 16:35:42,574 INFO 26436 99.12.232.123:8080 static stddev:10.0230955885 | | 2016-08-03 16:35:42,575 INFO 26436 99.12.232.123:8080 static avg:63.125 | | 2016-08-03 16:35:42,575 INFO 26436 99.12.232.123:8080 current avg:21 | | 2016-08-03 16:35:42,576 INFO 26436 99.12.232.123:8080 sigma:[4.2027934013213901, -1] | | 2016-08-03 16:35:42,576 INFO 26436 99.12.232.123:8080 member 99.12.232.123:8080 in enable state | | 2016-08-03 16:35:42,910 INFO 26436 99.12.232.123:8080 offset value is too big,need switch ratio to 4 | | 2016-08-03 16:35:43,217 INFO 26436 99.12.232.123:8080 Disable this member static |   BIGIP管理界面显示此时服务器ratio依旧为4  恢复延迟为50  EAV日志显示：   |  | | --- | | 2016-08-03 16:40:22,994 INFO 28089 99.12.232.123:8080 static stddev:10.0230955885 | | 2016-08-03 16:40:22,994 INFO 28089 99.12.232.123:8080 static avg:63.125 | | 2016-08-03 16:40:22,995 INFO 28089 99.12.232.123:8080 current avg:49 | | 2016-08-03 16:40:22,995 INFO 28089 99.12.232.123:8080 sigma:[1.4092452651315048, -1] | | 2016-08-03 16:40:22,995 INFO 28089 99.12.232.123:8080 member 99.12.232.123:8080 in disable state | | 2016-08-03 16:40:22,995 INFO 28089 99.12.232.123:8080 offset value is close to old value ,switch ratio to old value:4 |   调整结果为服务器ratio值不变依旧为4 | |
| 测试结果：  本地服务器的延迟下降，不会影响正常负载均衡的比率，用户依然获得正常的用户体验。  符合预期。 | |
| 客户工程师签字： | F5工程师签字： |

#### 异地服务器单边下降

|  |  |
| --- | --- |
| 测试项目：测试异地服务器（上海）的延迟发生单边下降情况下，延迟负载均衡的流量调整 | |
| 测试目的：测试当本地服务延迟单边下降，延迟负载均衡算法能否根据延迟进行流量调整 | |
| 测试预期：  异地的服务器延迟下降，延迟负载均衡算法将提升异地的访问量，如果异地服务器延迟重新恢复到原来的级别，延迟负载均衡算法重新调整流量比率至原始状态。 | |
| 测试过程：  通过修改异地服务器代码，将SQL语句从10句查询修改为1句，此时代码执行为110ms毫秒左右  EAV的日志为   |  | | --- | | 2016-08-03 17:01:41,716 INFO 3378 99.3.0.207:8080 static stddev:60.2340635897 | | 2016-08-03 17:01:41,717 INFO 3378 99.3.0.207:8080 static avg:529.743702081 | | 2016-08-03 17:01:41,717 INFO 3378 99.3.0.207:8080 current avg:285 | | 2016-08-03 17:01:41,717 INFO 3378 99.3.0.207:8080 sigma:[4.0632108726435012, -1] | | 2016-08-03 17:01:41,717 INFO 3378 99.3.0.207:8080 member 99.3.0.207:8080 in enable state | | 2016-08-03 17:01:42,172 INFO 3378 99.3.0.207:8080 offset value is too big,need switch ratio to 4 | | 2016-08-03 17:01:42,478 INFO 3378 99.3.0.207:8080 Disable this member static |   显示已经把异地服务器的分配比率调整为和本地服务器一致，此时BIGIP显示为    注意此时：主中心延迟为60，备中心延迟110  将代码恢复后，异地服务器延迟恢复到500ms左右，此时EAV日志显示为   |  | | --- | | 2016-08-03 17:07:41,741 INFO 5528 99.3.0.207:8080 static stddev:60.2340635897 | | 2016-08-03 17:07:41,741 INFO 5528 99.3.0.207:8080 static avg:529.743702081 | | 2016-08-03 17:07:41,742 INFO 5528 99.3.0.207:8080 current avg:434 | | 2016-08-03 17:07:41,742 INFO 5528 99.3.0.207:8080 sigma:[1.5895275260404247, -1] | | 2016-08-03 17:07:41,743 INFO 5528 99.3.0.207:8080 member 99.3.0.207:8080 in disable state | | 2016-08-03 17:07:41,743 INFO 5528 99.3.0.207:8080 offset value is close to old value ,switch ratio to old value:1 |   显示恢复到原延迟后，分配比例恢复为原延迟比例 | |
| 测试结果：  异地的服务器延迟下降，延迟负载均衡算法将提升异地的访问量，如果异地服务器延迟重新恢复到原来的级别，延迟负载均衡算法重新调整流量比率至原始状态，用户获得较好的延迟体验。  符合预期。 | |
| 客户工程师签字： | F5工程师签字： |

## 测试结果

（1）通过BIG-IP系统上对延迟的采集，能够判断出深圳上海两地数据中心的数据库是否发生了切换，并根据延迟状态对APP前端进行切换，提高了访问效率。

（2）提高了切换效率和管理员对F5设备的操作准确性，同时提高业务交易体验。

（3）人工干预的接口能较好的防止自动化过程中出现的异常情况。

（4）根据大数据采集和算法的延迟负载均衡算法经过测试，基本符合预期设计。

# 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