一种基于 OpenStack 的云存储空间动态调整方案

郑 印[1],吴振宇[2],沈苏彬[1]
(1. 南京邮电大学 计算机学院,江苏 南京 210003;
2. 南京邮电大学 物联网学院,江苏 南京 210003)

摘 要:针对 OpenStack 云主机的资源监测和存储空间不能动态分配的问题,提出一种基于 OpenStack 云存储空间动态调整方案。在 OpenStack 云主机的内部安装 qemu-guest-agent(qga),实现宿主机与云主机之间的通信,然后扩展 OpenStack 云平台,实现监测云主机资源并且对数据进行分析,得到云主机资源的使用情况。当云主机的磁盘利用率很高时,可以利用 OpenStack 提供的 API,自动创建存储空间,分配给云主机使用,从而动态扩展了云主机的存储空间。实验结果表明,文中提出的方案能够根据云主机存储资源的使用情况,动态调整存储空间大小。

关键词:OpenStack;云存储;虚拟资源监测;qemu-guest-agent
中图分类号:TP301 **文献标识码**:A **文章编号**:1673-629X(2016)10-0045-05
doi:10.3969/j.issn.1673-629X.2016.10.010

A Dynamic Adjustment Solution of Cloud Storage Space Based on OpenStack

ZHENG Yin[1], WU Zhen-yu[2], SHEN Su-bin[1]
(1. School of Computer Science & Technology, Nanjing University of Posts and Telecommunications, Nanjing 210003, China;
2. School of Internet of Things, Nanjing University of Posts and Telecommunications, Nanjing 210003, China)

Abstract: Considering the problems that resource of OpenStack is monitored hardly and storage space cannot be allocated to the virtual machine dynamically, a dynamic adjustment solution of storage space based on OpenStack is proposed. The qemu-guest-agent is installed in the OpenStack virtual machine, which realizes the communication between the host and the virtual machine. Then functions of OpenStack are extended to monitor virtual machine resources and analyze the data. When the disk utilization of virtual machine is very high, OpenStack API is used automatically to create storage space which will be assigned to the virtual machine. Therefore, the storage space of the virtual machine is expanded dynamically. Experimental results show that the proposed solution can dynamically adjust the storage space on the basis of the usage of the virtual machine.

Key words: OpenStack; cloud storage; virtual resource monitoring; qemu-guest-agent

0 引 言

在当今互联网高速发展的时代,云计算[1-2]无疑是最热门的研究领域之一。在众多的云计算平台中,OpenStack 是一种开源的云计算平台[3],具有良好的架构和易扩展性,受到了许多 IT 公司的关注[4],因此获得了大量的人力、物力资源的支持,从而得到了快速发展。

利用 OpenStack 云平台完成创建云主机之后,用户就可以把自己的应用部署到云主机上,但是对一些用户来说,并不知道自己所用的云主机资源的使用情况,另外,当虚拟磁盘空间利用率很高时,也不能很好地扩展云主机的储存空间。针对上述提出的两个问题,文中提出在云主机内部安装 qemu-guest-agent(qga),实现宿主机与云主机之间的通信,进而扩展 OpenStack 云平台,监测云主机资源的使用情况,如内存、VCPU 和磁盘的利用率等,然后把监测结果反馈给

收稿日期:2016-01-10 **修回日期**:2016-04-14 **网络出版时间**:2016-09-19
基金项目:江苏省未来网络前瞻性研究(BY2013095-1-08)
作者简介:郑 印(1990-),男,硕士研究生,研究方向为计算机网络;沈苏彬,研究员,博士生导师,研究方向为计算机网络、下一代电信网及网络安全。

OpenStack 平台。当云主机的磁盘使用率很高时,就动态地为云主机扩展存储空间,满足不断增长的存储需求,达到动态扩展存储空间的目的。这种方案适用于公司内部,建立一个基于 OpenStack 的私有云,该私有云主要提供数据存储功能,满足公司内部不断增长的数据存储需求。

文中首先介绍 OpenStack 云平台和云主机的资源监测特点;其次,针对用户需求,设计出监测云主机的资源利用情况和动态分配 cinder 云硬盘给云主机的方案;然后,对该方案进行具体实现;最后,测试方案并分析结果。

1 相关技术分析

1.1 OpenStack 云平台与云存储

OpenStack 是一个开源的云计算平台,根据不同的需求,它可以构建公有云、私有云和混合云[5]。OpenStack 最初只是由 Nova 和 Swift 两个模块组成,随着 IT 公司的纷纷加入,OpenStack 开源社区不断壮大,功能也不断完善[6]。

目前,OpenStack 云存储[7]空间是在创建时分配得到的,这样就有一个缺点,如果刚开始存储空间分配得太大,就会造成资源浪费;如果分配得太小,则不能满足用户需求,还需要用户手动创建云硬盘来扩展云存储空间。因此,文中提出一种方案:在云主机创建时,存储空间分配得不要太大,然后对云主机的存储资源进行实时监测,当存储资源使用率很高时,再动态地为其分配 cinder 云硬盘,扩展其存储空间。

1.2 云主机资源监测

目前,OpenStack 云平台利用 Libvirt 提供的 API 接口[8-9]可以对云主机进行简单的监测,但是这些监测仅仅局限于云主机的一些基本资源,如 VCPU、内存和磁盘的大小等,并不能监测这些资源在云主机中实时的使用情况[10]。因此,基于 OpenStack 的云主机要想实现存储空间动态调整的目的,必须要实时监测云主机资源的使用情况,再据此决定是否需要动态扩展存储空间。

在 VMware 的虚拟机中[11-12],VMware-tools 软件安装在虚拟机内部,通过它可以实现云主机和宿主机之间的通信,提高虚拟机的功能和性能。同样在基于内核的虚拟机技术(KVM)中[13],它也有一款具有相同功能的软件叫 qemu-guest-agent。

2 方案设计

针对文中提出的基于 OpenStack 云存储空间的动态调整方案,首先设计出方案的功能视图,如图 1 所示。

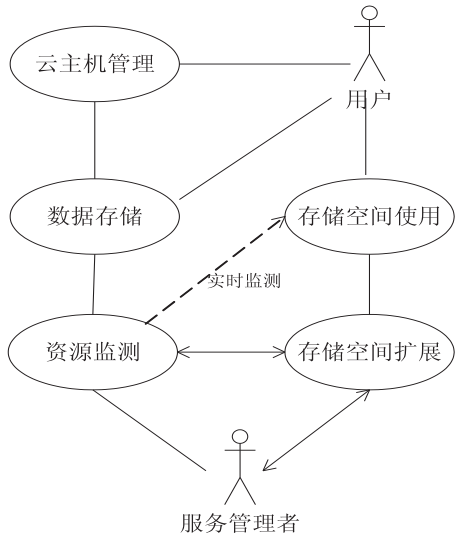

图 1 方案的功能视图

从图中可以看出,方案的功能视图中有五个用例,其中比较重要的是资源监测用例和存储空间扩展用例。所以采用模块化设计方法可以将该方案分为两个模块:监测云主机资源和将 cinder 云硬盘动态分配给云主机。

2.1 监测云主机模块设计

目前,OpenStack 云平台利用 Libvirt 提供的 API 接口可以对云主机进行简单的监测,但是它只能监测云主机的一些基本资源,并不能监测这些资源在云主机中实时的使用情况。文中提出一种方法,首先在云主机中安装 qga 软件,实现云主机与宿主机之间的通信。然后,扩展 OpenStack 云平台,利用 Libvirt API 查询云平台上的所有云主机,筛选出活跃的云主机,进而读取这些云主机/proc 目录下记录资源使用情况的一些文件,并且对其进行分析。当云平台得知云主机的存储空间使用率很高时,就为云主机分配云硬盘,扩大其储存空间。

2.2 动态扩大云主机的存储空间

当云主机的存储资源使用率已经很高时,如果想继续存储数据,就需要动态地分配一块 cinder 云硬盘给云主机,以扩大其存储空间。

文中利用 OpenStack 云平台提供的 RESTful API 接口,向 cinder 模块发送创建云硬盘请求,cinder 模块收到创建请求后,创建一块云硬盘,并返回云硬盘的 id,然后再把云硬盘挂载到云主机上,从而达到扩大云主机存储空间的目的。

云硬盘挂载到云主机后还不能直接使用,得到的云硬盘只能作为云主机的一个分区,必须要对这块云硬盘进行格式化,然后挂载到云主机的一个目录下,这样才可以使用。

动态扩大云主机存储空间的设计如图 2 所示。

3　方案实现

3.1　监测模块实现

监测模块主要实现云主机资源的监测,然后把数据传递给动态扩大存储空间模块,最后决定是否需要增大存储空间。

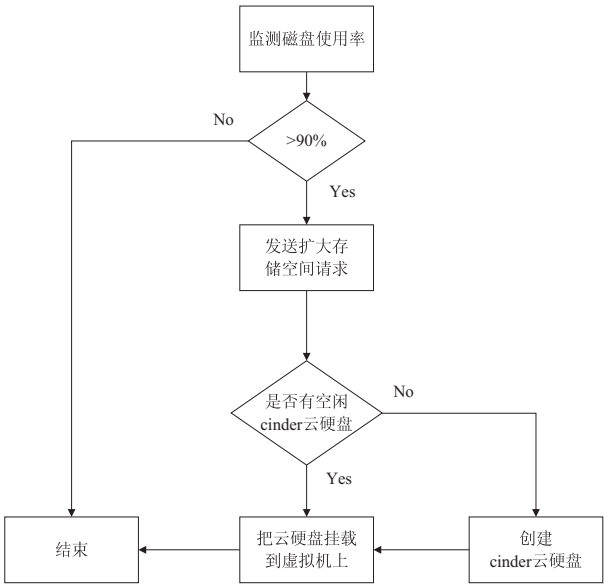

图 2　动态扩大云主机存储空间流程图

监测模块的实现步骤如下:

(1)制作 OpenStack 的镜像文件,在镜像中安装 qemu-guest-agent。

(2)虽然镜像中安装了 qemu-guest-agent,但还不能使用该功能,需要添加属性"hw_qemu_guest_agent = yes",这样通过镜像创建的云主机就可以使用 qemu-guest-agent 功能[14]。

(3)通过上述处理,云主机与宿主机之间可以进行通信,下面需要扩展 OpenStack 平台[14-15]。在 Nova 模块中添加一个 monitor 组件,读取云主机内部的记录文件并且对其进行分析,如果存储空间不足,就增大其存储空间。

(4)首先扫描 OpenStack 中所有的云主机,然后筛选出活跃的云主机,用_read_file_from_guest 函数打开云主机内部的文件。

(5)把云主机内部记录磁盘使用情况的文件路径传递给_read_file_from_guest 函数,通过该函数打开文件并且返回文件的内容。打开磁盘的相关文件操作如下:

```
#得到磁盘的分区信息
mounts_file = self._read_file_from_guest('/proc/mounts')
#得到磁盘的一些状态信息
diskstats = self._read_file_from_guest('/proc/diskstats')
```


(6)对读取的磁盘信息进行分析,得到磁盘总空间的大小,已使用和未使用的大小。

(7)通过上面的处理,把磁盘的使用情况,还有一些关于磁盘的其他信息存储在字典中,方便以后使用。

(8)对基于 OpenStack 的云主机而言,数据随时都可以存储在云主机上,那么就需要实时地对云主机进行监测。可考虑使用 OpenStack 提供的定时器,每隔一段时间获取磁盘的使用信息,从而达到实时监测的目的。定时器的使用代码如下:

```
fromnova.OpenStack.common import periodic_task
#在需要定时执行的函数上面添加定时任务
@periodic_task.periodic_task(spacing = CONF.txt_timer_interval)
```

3.2　动态扩大存储空间模块实现

通过监测模块的实验,实现了对云主机存储资源的实时监测,当云主机的磁盘使用率已经很高时,如果继续进行数据存储,原有的云主机将很难满足要求,这样就需要扩大云主机的存储空间。

实现的具体步骤如下:

(1)当云主机的磁盘使用率大于 90% 时,就发送扩大存储空间的请求。

(2)add_instance_volume(self,instance_uuid)函数收到云主机增大存储空间的请求后,需要得到一块空闲云硬盘,然后再把这块空闲云硬盘挂载到该云主机上。代码实现如下:

```
def add_instance_volume(self,instance_uuid):
#得到一块空闲云硬盘的 id
volume_id = self.check_and_create_volume.create()
#把云硬盘挂载到云主机上
self.attach_volume.attach_volume(instance_uuid,volume_id)
```

(3)check_and_create_volume.create()函数首先检查用户所在的租户下是否有空闲的 cinder 云硬盘,如果有,则直接返回云硬盘的 id,如果没有,则创建一块云硬盘并返回其 id。

(4)attach_volume(self,instance_uuid,volume_id)函数把获得的 cinder 云硬盘挂载到对应的云主机上,实现对云主机存储空间的扩展。它也是利用 RESTful API 发送挂载请求,实现过程与步骤(3)类似。

通过上述实验,可以成功地把 cinder 云硬盘挂载到云主机上,实现云主机存储资源的动态增长。

3.3　对新添加的云硬盘进行处理

文中提出一种方法,在云主机里写一个脚本文件,通过脚本文件的定时执行,完成云硬盘的格式化和挂载工作,使云硬盘真正能够为云主机所使用。

具体步骤如下:

(1)在 Linux 操作系统中,有两个非常有用的命令:一个是 df 命令,查看系统下可以直接利用的磁盘以及这些磁盘的利用率;第二个命令是 fdisk - l,查看系统下的分区,这个命令显示的数据必然包括 df 命令显示的可用磁盘,但是它还包括系统没有被格式化的分区,这些分区还不能直接使用,需要被格式化,然后挂载到一个目录下才可以使用。这样的话,通过比较两个命令的显示数据,就可以把还不能使用的磁盘(也就是新添加的云硬盘)分离出来。

(2)对新添加的云硬盘进行格式化,然后把云硬盘挂载到云主机的一个目录下。这个脚本文件实现了在云主机下检测新添加的云硬盘,然后对这个云硬盘进行处理,把它变成可以使用的磁盘。

(3)因为云主机不知道什么时候得到了一块云硬盘,所以它需要每隔一段时间进行检查,判定是否新添加了云硬盘,这样就需要编写执行文件 disk. sh,定时执行这个脚本文件。具体代码如下:

```
#! /bin/sh
PATH=/sbin:/bin:/usr/sbin:/usr/bin #添加环境变量
/usr/bin/python /home/centos/ mount_disk. py
>/dev/null 2>&1 #执行 python 脚本文件,并且输出结果不保存。
```

(4)把 disk. sh 文件添加到云主机的定时执行任务中,具体代码如下:

```
*/1 * * * * /bin/sh /home/centos/disk. sh > /dev/null 2>&1
#每隔一分钟执行一次脚本文件
```

4 测试结果及分析

文中的实验环境是在一台物理机上搭建 OpenStack,然后在 OpenStack 的 Nova 模块中添加一个自定义组件 monitor,用于监测云主机的资源使用情况和动态分配云硬盘。

物理机配置:Intel core i3,磁盘 500 GB,内存16 G。

操作系统:Centos6. 5。

云系统环境:OpenStack H 版。

第三方软件:RESTClient、MySQL、RabbitMQ。

为了验证文中设计方案的正确性以及发现方案中存在的不足之处,对设计方案进行测试。

首先打开云主机,运行 qemu-guest-agent,使宿主机和云主机之间能够通信,然后运行 monitor 的监测模块,周期性监测云主机资源的使用情况,选取其中几个时间点的数据,得到的结果如表 1 所示。

动态扩大云主机存储空间的设计目标是:当云主机的存储空间使用率达到 90% 时,OpenStack 云平台就为云主机分配一块云硬盘,从上面的监测结果可以看出随着数据的不断存储,云主机的存储空间不断变小,在 10:31 时刻,云主机的存储资源使用率已经达到 90. 1%。这时系统就会调用动态增加云主机模块自动创建一块云硬盘,挂载到云主机上,云主机监测到新添加的云硬盘,就调用脚本文件对云硬盘进行处理,使其真正能够被云主机使用,从而扩大其存储空间。

表 1 增加云硬盘之前的云主机监测信息

时间	memusage/M	cpuusage/%	diskusage/M	diskrate/%
10:02	93	16.93	1 638.4	80
10:18	101.3	18.2	1 771.52	86.5
10:31	105.7	19.8	1 845.29	90.1

再用 monitor 的监测模块监测此时的云主机磁盘使用情况,选取其中几个时间点,得到的结果如表 2 所示。

表 2 增加云硬盘之后的云主机监测信息

时间	memusage/M	cpuusage/%	diskusage/M	diskrate/%
10:32	97.4	17.3	1 850.7	45.1
10:47	99.5	18.9	1 908.41	46.6
10:59	102.8	20.6	1 998.2	48.7

云主机添加云硬盘和不添加云硬盘的存储空间变换折线图如图 3 所示。

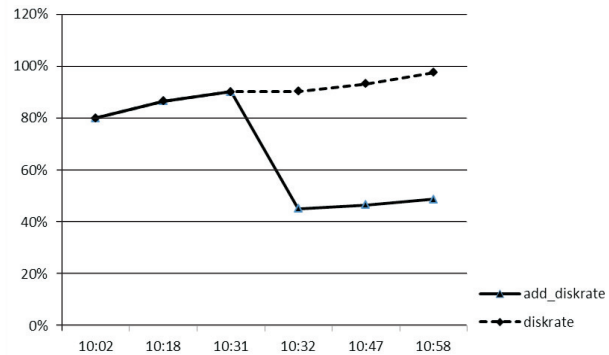

图 3 云主机存储空间使用率

图 3 中,在 10:31 时刻,云主机的存储空间使用率达到 90. 10%,增加云硬盘之后,它的存储空间使用率降了下来,达到 45. 10%(如实线所示)。如果这个时候没有增加云硬盘,它的存储空间使用率将继续增长(如虚线所示)。

实验结果表明,文中设计方案可以监测云主机的资源使用情况。当云主机的存储资源使用率很高时,可以动态地为其添加云硬盘,从而扩大其存储空间。

5 结束语

针对 OpenStack 云主机的资源监测和云硬盘不能动态分配的问题,文中提出一种基于 OpenStack 的云存储空间动态调整方案并对其进行测试,结果表明,该

方案达到了云主机资源监测和云硬盘动态分配的目的,具有可行性和有效性。

参考文献:

[1] Sahasrabudhe S S, Sonawani S S. Comparing OpenStack and VMware[C]//Proceedings of international conference on advances in electronics, computers and communications. [s. l.]:[s. n.],2014:1-4.

[2] Wang Endong, Wu Nan, Li Xu. QoS-oriented monitoring model of cloud computing resources availability[C]//Proceedings of fifth international conference on computational and information sciences. [s. l.]:[s. n.],2013:1537-1540.

[3] Zhang Zehua, Zhang Xuejie. Realization of open cloud computing federation based on mobile agent[C]//Proceedings of IEEE international conference on intelligent computing and intelligent systems. [s. l.]:IEEE,2009:642-646.

[4] Bist M, Wariya M, Agarwal A. Comparing delta, open stack and Xen cloud platforms:a survey on open source IaaS[C]//Proceedings of IEEE 3rd international conference onadvance computing conference. [s. l.]:IEEE,2013:96-100.

[5] Yang Chao-Tung, Liu Yu-Tso, Liu Jung-Chun, et al. Implementation of a cloud IaaS with dynamic resource allocation method using OpenStack[C]//Proceedings of international conference on parallel and distributed computing, applications and technologies. [s. l.]:[s. n.],2013:71-78.

[6] Bermudez I, Traverso S, Munafo M, et al. A distributed architecture for the monitoring of clouds and CDNs:applications to Amazon AWS[J]. IEEE Transactions on Network and Service Management,2014,11(4):516-529.

[7] Kuo Yen-Hung, Jeng Yu-Lin, Chen Juei-Nan. A hybrid cloud storage architecture for service operational high availability[C]//Proceedings of IEEE 37th annual computer software and applications conference workshops. [s. l.]:IEEE,2013:487-492.

[8] Lu X, Zhou W, Song J. Key issues of future network management[C]//Proceedings of IEEE international conference on computer application and system modeling. [s. l.]:IEEE,2010:649-653.

[9] 戢　友. OpenStack 开源云-王者归来[M]. 北京:清华大学出版社,2014:145-150.

[10] Zhu Zhenyu, Chen Hui, Wu Lianguo. Cloud computing model based on multiservice access dynamic detection[C]//Proceedings of IEEE 3rd international conference on cloud computing and intelligence systems. [s. l.]:IEEE,2014:461-464.

[11] Bing H. Research and implementation of future network computer based on cloud computing[C]//Proceedings of 2010 third international symposium on knowledge acquisition and modeling. [s. l.]:[s. n.],2010:406-408.

[12] Alabbadi M M. Cloud computing for education and learning:education and learning as a service (ELaaS)[C]//Proceedings of IEEE international conference on interactive collaborative learning. [s. l.]:IEEE,2011:589-594.

[13] Saibharath S, Geethakumari G. Design and implementation of a forensic framework for cloud in OpenStack cloud platform[C]//Proceedings of international conference on advances in computing, communications and informatics. [s. l.]:[s. n.],2014:645-650.

[14] Rossigneux F, Lefevre L, Gelas J P, et al. A generic and extensible framework for monitoring energy consumption of OpenStack clouds[C]//Proceedings of IEEE fourth international conference on big data and cloud computing. [s. l.]:IEEE,2014:696-702.

[15] Ristov S, Gusev M, Donevski A. Security vulnerability assessment of OpenStack cloud[C]//Proceedings of IEEE international conference on computational intelligence, communication systems and networks. [s. l.]:IEEE,2014:95-100.

(上接第 44 页)

[3] 吴果林,金　珍,邓小方. 稀疏网络的 Floyd 动态优化算法[J]. 江西师范大学学报:自然科学版,2013,37(1):28-32.

[4] 邓方安,雍龙泉,周　涛,等. 基于"矩阵乘法"的网络最短路径算法[J]. 电子学报,2009,37(7):1594-1598.

[5] 赵礼峰,梁　娟. 最短路问题的 Floyd 改进算法[J]. 计算机技术与发展,2014,24(8):31-34.

[6] 邹桂芳,张培爱. 网络优化中最短路问题的改进 Floyd 算法[J]. 科学技术与工程,2011,11(28):6875-6878.

[7] 谢　政. 网络算法与复杂性理论[M]. 长沙:国防科技大学出版社,2003.

[8] 刘焕淋,陈　勇. 通信网图论及应用[M]. 北京:人民邮电出版社,2010.

[9] Hougardy S. The Floyd-warshall algorithm on graphs with negative cycles[J]. Information Processing Letters,2010,110:279-281.

[10] 刘卫国. MATLAB 程序设计与应用[M]. 北京:高等教育出版社,2006.

[11] 李　科,袁　明. 小件快运中的最短运输时间问题[J]. 山东交通科技,2011(5):23-25.

[12] Feillet D, Dejax P, Gendreau M. Traveling salesman problems with profits[J]. Transportation Science, 2005, 39(2):188-205.

[13] Braess D, Nagurney A, Wakolbinger T. On a paradox of traffic planning[J]. Transportation Science, 2005, 39(4):446-450.

[14] Zhan F B. Three fastest shortest path algorithms on real road networks[J]. Journal of Geographic Information and Decision Analysis, 1997, 1(1):69-82.