文章编号：2096-1618(2019)01-0044-05

基于 Openstack Swift 的高可用存储方案研究与实现

张晓莉， 文成玉， 原子桢
(成都信息工程大学通信工程学院，四川 成都 610225)

摘要：为提高 Openstack Swift 集群的高可用性，基于 Openstack 云计算平台，提出一种通用地址冗余协议 CARP (common access redundancy protocol) + HAProxy + Swift 对象存储相结合的高可用存储解决方案，并针对 TempAuth 不适合生产环境下集群部署的缺陷，提出将 Swift 集群与 Openstack Keystone 服务相整合的方法。方案将通用地址冗余协议和负载均衡技术有效地融合。经过实验测试可得，改进的集群框架，在保证 Swift 集群数据安全性的同时，减轻了 Swift 代理节点的负载压力，增强了整体处理请求能力，提高了集群的高可用性。实验证明，方案具有可行性。

关 键 词：Openstack；Swift；HAProxy；CARP；Keystone；负载均衡
中图分类号：TP302.1 **文献标志码**：A
doi：10.16836/j.cnki.jcuit.2019.01.010

0 引言

随着计算机技术的飞速发展，在社交网络、天文、购物等平台，每天都产生大量数据。根据 IDC 报告显示，2011－2020 年，全球数据总量预计增加 22 倍[1]。而传统的存储方法难以满足日益增长的数据存储需要，于是云存储技术应运而生。Openstack Swift 对象存储便是当代受追捧的云存储技术之一。

高可用性 HA (high availability) 指的是通过尽量缩短因日常维护操作和突发的系统崩溃所导致的停机时间，以提高系统和应用的可用性[2]。在实际的生产环境中，由于采用集群技术搭建的服务器通常在各服务器的系统资源利用率上存在很大差距，导致集群的 IO 响应时间增长，读写性能下降[3]。负载均衡技术可以很好地解决这类问题[4-5]。杨飞等[6]提出将 HAProxy + Keepalived 部署在 Ceph 云存储集群的方法，降低服务节点压力，解决单节点故障问题。但是当初始系统只有一台服务器时，Keepalived 不能正常工作。王利萍[7]利用 Nginx 实现对集群后端资源的充分利用，提高系统整体性能。但是不能实现集群的主/从故障转移。沈平等[8]提出用 LVS + Keepalived 分担网络压力、排查故障，但是 LVS 配置较复杂。将 HAProxy 的负载均衡算法和通用地址冗余协议 CARP (common access redundancy protocol) 结合的方案，既可以减少服务器的负载压力，又可以避免单个 HAProxy 节点失效，造成整个系统故障。其中 CARP 能够使多台主机共享同一虚拟 VIP 地址，同时每个主机又有自己的 IP 地址，不同于 Keepalived 配置的浮动 IP，故 HAProxy + CARP 的结合方法也适用于初始环境只有一个主机的情形。并且 CARP 原理和配置过程较为简单。此外，由于 Openstack Swift 中自带的 TempAuth 服务以明文方式存储密码的特点，导致每次变化用户名和密码之后都需要重启 Proxy，在操作中带来很多不便，故提出将 Openstack 的 Keystone 服务和 Swift 集群相整合的方法，在保证 Swift 集群中数据的安全性同时，提高了在 Swift 集群上的工作效率。

收稿日期：2018-04-20
基金项目：四川省科学技术厅应用基础研究资助项目(2018JY0290)

1 高可用性方案基本原理

1.1 HAProxy 负载均衡技术

负载均衡主要用来扩展系统的带宽，增强网络数据处理能力，提高系统的资源利用率，缩短系统的响应时间，避免系统出现单点故障的一种建立在网络结构之上的调度策略[9]。

HAProxy 是一个使用 C 语言编写的自由及开放源代码软件，可提供高可用性、负载均衡，以及基于 TCP 和 HTTP 的应用程序代理。支持虚拟主机，它是免费、快速并且可靠的一种解决方案。HAProxy 支持 8 种负载均衡算法，其常用的算法有轮询算法、基于请求源 IP 的负载均衡算法和最小连接算法。

轮询算法：算法最简单，是将进入集群的请求按着轮转的方法转发给集群里面的各个节点，适合集群中的各个节点具有相同的软硬件配置和相对均衡的网络压力的情形。轮询算法没有考虑各服务器的性能差异

和后端服务器的实时负载能力。

最小连接算法:算法把当前请求分配给连接数最少的服务器。HAProxy 中最小连接算法是动态的,可以在运行时调整各个节点的权重。该方法中的权重应该合理地体现各个服务器存储性能的差异,否则会对系统性能产生很大的影响[10]。假定服务器 $N_1, N_2, N_3, \cdots, N_n$ 的初始权重分别为 $W_1, W_2, W_3, \cdots, W_n$,可用公式(1)计算权值:

$$W_i = M(N_i) \cdot P_1 + D(N_i) \cdot P_2 \quad 1 \leqslant i \leqslant n \tag{1}$$

其中,M 为内存剩余百分比,D 为磁盘剩余百分比,P_1 和 P_2 分别为各自权重。$P_1 + P_2 = 1, 0 \leqslant P_1, P_2 \leqslant 1$。$P_1, P_2$可以在实验中不断地修正,找到较适于集群的值。

基于请求源 IP 的负载均衡算法:算法将请求的源地址进行哈希运算,使相同客户端 IP 的请求被转发至特定的服务器,但当有服务器添加或者删除时,部分请求可能会被转发至与之前请求不同的服务器。

通过以上分析可知,最小连接算法综合考虑了实时负载情况和服务器的性能差异,更适合用于 Swift 集群。所以在利用 HAProxy 进行负载均衡时,可采用最小连接算法。

1.2 CARP

CARP 能够使多台使不同 IP 地址的主机共享同一 VIP(虚拟 IP)地址[11]。当某个主机发生故障时,其他主机会自动接管服务[12]。在集群配置中,这样做可以将“主备模式”引入集群,提高整体系统的可用性。CARP 协议的特点在于其非常低的开销,主机间使用加密数据传递信息,并且在冗余主机之间不需要任何额外的网络链接。

1.3 集群可靠性的计算方法

系统的可靠性是指系统在规定条件下和规定时间内完成规定功能的能力。对于如何创建 Swift 对象存储的可靠性模型,就要清楚会造成数据丢失(并且无法恢复)的情况。可以假定一个 3 个存储节点的 Swift 集群,每个存储节点有一块磁盘,副本数为 3(3 个副本分布在 3 个存储节点上)。只有当 3 个存有副本的存储节点的磁盘全部出现故障,数据才无法恢复。根据分析,认为 Swift 集群的可靠性是与存储节点的数量 N、副本数 R、每一个存储节点的磁盘数量 S 和磁盘的年失败率 AFR 有关。故 Swift 集群的丢失所有副本概率 P 可用公式(2)表示为

$$P = \text{func}(N, R, S, AFR) \tag{2}$$

磁盘的年失败率 AFR(annualized failure rate)指的是设备在使用一整年内会发生故障的概率[13],计算公式为

$$AFR = (24 \times 365) / MTBF \tag{3}$$

式(3)中,MTBF(mean time between failures)是磁盘两次故障的时间间隔小时。而单块的硬盘损坏的期望值 FIT(failures in time)是指每 10 亿小时硬盘的失败率(failure rate),计算公式为

$$FIT = \frac{1}{MTF} \approx \frac{1}{MTBF} = \frac{AFR}{24 \times 365} \tag{4}$$

式(4)中,MTTF(mean time to failure)表示磁盘连续工作的时间。现在假定任意 N 个服务器(Node),每个服务器只有一块磁盘,R 个副本分布在不同的服务器上。

(1)任意一个 Node 出现损坏的概率 $P_1(\text{any})$,因为任意一个 Node 出现损坏的概率 $P_1(\text{any})$ 与无节点出现损坏的概率之和为 1,故

$$\alpha = FIT \cdot N \tag{5}$$

$$P_1(\text{any}) = 1 - P_0(\alpha, t) = 1 - \frac{(NFT \cdot N \cdot t) e^{-FIT \cdot N \cdot t}}{n!} = 1 - e^{-FIT \cdot N \cdot t} \tag{6}$$

其中 α 表示硬盘失败率,P_0 表示泊松分布函数,N 表示存储节点数。

(2)任意第 2 个 Node 出现故障的概率 $P_2(\text{any})$,则

$$P_2(\text{any}) = 1 - P_0(\alpha, t) = 1 - \frac{(NFT \cdot (N-1) \cdot t) e^{-FIT \cdot (N-1) \cdot t}}{n!} = 1 - e^{-FIT \cdot (N-1) \cdot t} \tag{7}$$

任意第 3 个节点出现故障的概率 $P_3(\text{any})$ 计算同上。则 1 年内任意 R 个 Node 出现故障的概率 P_r 为

$$P_r = P_1(\text{any}) \cdot P_2(\text{any}) \cdot P_3(\text{any}) \tag{8}$$

因为损坏的 R 个存储节点未必保存着一个数据的全部副本信息,故不一定代表数据全部丢失。丢失全部保存副本信息的节点的概率 $P = 1/C_N^R$。

故 Swift 集群可靠性 W 的算法为

$$W = 1 - P = 1 - P_r / C_N^R \tag{9}$$

式(9)中,C_N^R 为损坏 R 个存储节点的可能集合,R 为副本数,N 为节点数,P 为丢失所有副本的概率。

2 高可用性方案的实现

2.1 Keystone 和 Swift 的整合

在 Openstack 平台框架中 Keystone 组件的主要职责是用户身份的认证和对用户信息的管理等[14]。在方案中,当客户端访问 Swift 服务时,要先向 Keystone

发出认证请求,Keystone 接收请求进行权限认证,若客户端有权限访问,则返回 Swift 服务的 endpoint,客户端才能成功访问 Swift。

方案计划将 Openstack 的 Keystone 服务部署在 Openstack 的控制节点,由于需要在 Swift 客户端和负载均衡设备 HAProxy 之间加入 CARP 服务,故将存储服务的 endpoint 指向共享的虚拟 VIP 地址和相应端口号。

2.2 高可用方案设计

在 Keystone 和 Swift 整合的框架之上,还将 CARP 和 HAProxy 引入集群,构建高可用的 Swift 集群。高可用的整体框架如图 1 所示。

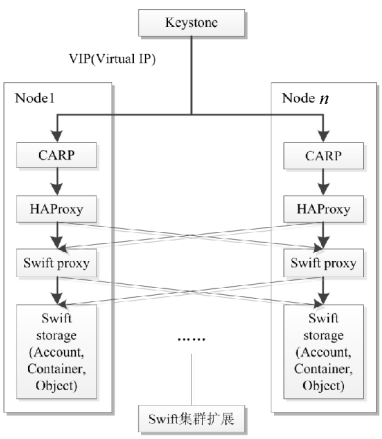

图 1 Openstack Swift 高可用整体框架

为保证基于 Openstack Swift 对象存储集群的高可用性、平衡性和对称性,在 n 个节点上分别部署 Swift 存储服务和代理服务,并使用开源的 HAProxy 和最小连接算法,分担单个 Swift 代理节点上的负载压力,实现 Swift 集群的 AA 模式(active-active);将 CARP 和 HAProxy 相结合,通过多个节点共享同一虚拟 IP,实现 Swift 集群的 AS 模式(active-standby);将 Keystone 和 Swift 集群整合,利用 Keystone 保证 Swift 集群内部信息的安全性与隔离性。设计方案对请求的处理过程(以 3 节点为例)如图 2 所示。

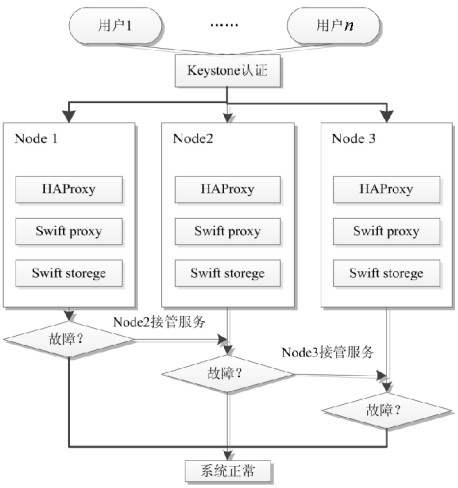

图 2 改进框架请求处理流程

图 2 中,HAProxy 负责接收外部对 Swift 集群发来的大量请求,并利用最小连接算法将请求分发给 3 个 Swift 代理节点。同时可对 HAProxy 节点的请求转发情况进行监控。

CARP 允许 Node1,Node2,Node3,…,Node$n$ 共享一个虚拟的 VIP 地址,当其中某个节点宕机时,剩余的节点会自动接管服务。

3 实验环境和测试结果

3.1 实验环境

为测试框架的性能,在虚拟机上搭建了由一个认证节点和 3 个存储节点构成的小型 Swift 集群,各个服务的部署方式如表 1 所示。

表 1 实验环境

节点名称	IP	操作系统	硬盘大小/GB	内存大小/GB	主要服务
controller	192.168.0.155	Centos7	20	1	Swiftclient,keystone
Node1	192.168.0.156	Centos7	20	1	CARP,HAProxy,Swift proxy,Swift storage
Node2	192.168.0.157	Centos7	20	1	CARP,HAProxy,Swift proxy,Swift storage
Node3	192.168.0.158	Centos7	20	1	CARP,HAProxy,Swift proxy,Swift storage

其中,Node1,Node2,Node3 共享的虚拟 IP(VIP)是 192.168.0.154。

3.2 安全测试和高可用测试

高可用集群框架,完成了 Keystone 和 Swift 的结合,合法用户能够实现对 Swift 集群的数据操作和管理。

当断开任意一个 HAProxy 节点,其他主机会自动接管功能,保证了系统的整体稳定性。

3.3 性能测试

(1)通过对 HAProxy1、HAProxy2 和 HAProxy3 的

DNS Server 的端口、IP 进行实时的状态监控,HAProxy1 请求转发情况如图 3 所示。

由图 3 可以得知,进入 Swift 集群的请求被最小连接算法分配到了 3 个代理节点,有效缓解了单个服务器的压力。

(2)对该高可用框架进行了大量的实验,用 webbench 对集群进行压力测试,分别测试在 50clients、100clients、150clients 并发的情况下,分析加入 HAProxy 负载均衡前后,系统的处理请求的能力。统计、记录如表 2。选取每分钟处理请求数量为指标,得到的综合对比如图 4 所示。

swift-cluster

		Queue			Session rate			Sessions						Bytes		Denied		Errors			Warnings	
		Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	Last	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status
	proxy1	0	0	-	0	424		0	34	-	13 303	13 303	58m33s	54 624 345	4 275 923		0		0	0	0	0	1d16h UP
	proxy2	0	0	-	0	576		0	34	-	18 296	18 296	58m29s	125 100 765	5 867 663		0		0	0	0	0	1d13h UP
	proxy3	0	0	-	0	238		0	34	-	6 889	6 889	10h55m	11 625 204	2 220 682		0		0	0	0	0	49s UP
	Backend	0	0		0	1 213		0	100	10	38 488	38 488	58m29s	191 350 314	12 364 268	0	0		0	0	0	0	1d16h UP

图 3　Swift proxy 请求转发监控状态图

表 2　系统优化前后测试

编号	并发数	优化前处理请求数量/(pages/min)	优化后处理请求数量/(pages/min)
1	50	40818	77858
2	150	40666	74758
3	250	40230	77694
4	350	40824	64920
5	450	42444	67830
6	550	40234	65812
7	650	40070	77448

通过测试实验得知,该高可用框架,由于 HAProxy 负载均衡算法将请求较为合理的分发到后端服务器,使得系统在处理请求速度上面有明显的改善。

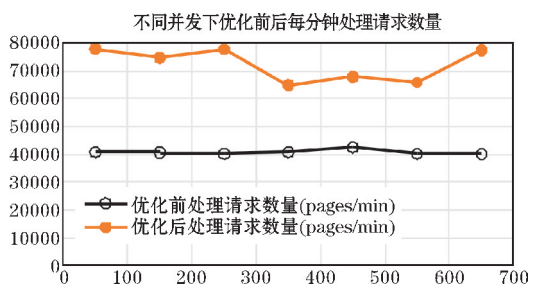

图 4　优化前后处理请求能力对比图

3.4　高可用框架的可靠性计算

在高可用方案中,假设集群中存储节点的数量为 9、副本数为 3、每一个存储节点的磁盘数量为 1、磁盘的年失败率 $AFR=1.2\%$。首先,根据公式(4)可以求得单块硬盘损坏的期望值 $FIT=1.369863e-6$,根据公式(6)、(7)可分别求得 1 年内任意节点出现损坏的概率 $P_1(\text{any})=1.102372e-1$,1 年内任意第 2 节点出现损坏的概率 $P_2(\text{any})=1.095884e-5$,1 年内任意第 3 个节点出现损坏的概率 $P_3(\text{any})=8.05687e-2$。然后,根据公式(8)可以求得,1 年内任意 3 个节点损坏的概率 $P_3(\text{any})=9.733283e-8$。最后,由于 $C_9^3=84$,根据公式(9)求得可靠性 $W=99999999884$,因此计算得到该高可用框架的可靠性大约为 8 个 9。

4　结束语

对于改进后的 Openstack Swift 集群高可用云存储框架,首先,用户可以通过统一的 Keystone 认证系统访问集群资源,保证了内部资源的安全性。其次,该框架充分利用了 CARP 的故障处理技术和 HAProxy 的负载均衡技术,有效地减轻了 Swift proxy 的 IO 负载压力,避免了单节点 HAProxy 失效时造成的故障问题。最后,通过对系统的整体能力测试,可得该高可用系统,在处理请求和数据量的能力明显增强,达到了高可用性的目的。

在实验中,调度策略选取的是最小连接算法。在算法中,只考虑了服务器连接量,因而还不能非常有效地衡量服务器的负载情况。可以综合考虑后端各个存储节点的 IO 状况、CUP 利用率、内存资源利用率等对服务器负载进行衡量,从而进一步改善集群性能,这有待后续进一步研究。

参考文献:

[1] Turner V, Gantz J F, Reinsel D, et al. The digital universe of opportunities: Rich data and the increasing value of the internet of things[J]. IDC Analyze the Future, 2014, 2(3): 12-14.

[2] 彼达. 高可用性:设计、技术和运作过程[M]. 北京:社会科学文献出版社, 2003: 2-3.

[3] Zhao Y, Huang W. Adaptive Distributed Load Balancing Algorithm Based on Live Migration of Virtual Machines in Cloud[C]. International Joint Conference on Inc, Ims and IDC. New York: IEEE, 2009: 170-175.

[4] Godfrey B, Lakshminarayanan K, Surana S, et al. Load balancing in dynamic structured P2P systems[J]. Proc of IEEE Infocom, 2004, 2735(4): 2253-2262.

[5] 徐敏, 李明, 郑建忠, 等. 基于 OpenStack 的 Swift 负载均衡算法[J]. 计算机系统应用, 2018, 27(1): 127-131.

[6] 杨飞, 朱志祥, 梁小江. 基于 Ceph 对象存储集群的高可用设计与实现[J]. 微电子学与计算机, 2016, 33(1): 60-64.

[7] 王利萍. 基于 Nginx 服务器集群负载均衡技术的研究与改进[D]. 济南:山东大学, 2015.

[8] 沈平, 潘志安, 袁瑛. 一例基于 LVS+Keepalived 架构的服务器访问故障分析[J]. 电脑知识与技术, 2013(8): 1762-1763.

[9] Cardellini V, Colajanni M, Yu P S. Dynamic Load Balancing on Web-Server Systems[M]. New York: IEEE Educational Activities Department, 1999: 1-3.

[10] Zinke J, Schnor B. The impact of weights on the performance of Server Load Balancing systems[C]. International Symposium on PERFORMANCE Evaluation of Computer and Telecommunication Systems. New York: IEEE, 2013: 30-37.

[11] Attebury G, Ramamurthy B. Router and Firewall Redundancy with OpenBSD and CARP[C]. IEEE International Conference on Communications. New York: IEEE, 2006: 146-151.

[12] Antunes C, Vardasca R. Building Low Cost Cloud Computing Systems[J]. International Journal of Advanced Computer Science & Applications, 2013, 4(5): 47-52.

[13] 赵畅. 云存储系统可靠性分析[D]. 天津:南开大学, 2014.

[14] 陈辉, 李陶深, 岑霄. Openstack 核心存储件 Swift 与 Keystone 的集群整合方法[J]. 广西科学院学报, 2015, 31(1): 73-76.

Research and Implementation of High Availability Storage Scheme based on Openstack Swift

ZHANG Xiaoli, WEN Chengyu, YUAN Zizhen

(College of Communication Engineering, Chengdu University of Information Technology, Chengdu 610225, China)

Abstract: In order to improve the high availability of Openstack Swift clusters, based on the Openstack platform, a highly available storage solution combining CARP (common access redundancy protocol) + HAProxy + Swift clusters is proposed. For the defect of TempAuth not suitable for cluster deployment in production environment, the method of integrating Swift cluster with Openstack Keystone is proposed. The solution effectively integrates the Common Access Redundancy Protocol and load balancing technology. After experimental tests, the improved cluster framework not only guarantees the data security of the Swift cluster, but also relieves the load pressure of the Swift proxy node, enhances the overall ability to process requests, and improves reliability and high availability. Therefore, the scheme is available.

Keywords: Openstack; Swift; HAProxy; CARP; Keystone; load balancing