**内核程序测试报告**

原雅茹 2019-08-12

一、测试环境

|  |  |  |  |
| --- | --- | --- | --- |
| **节点信息** | | | |
| **序号** | **IP** | **OS** | **内存** | |
| 1 | 192.168.100.61 | Centos7 | 3G | |
| 2 | 192.168.100.62 | Centos7 | 3G | |
| 3 | 192.168.100.63 | Centos7 | 3G | |
| 4 | 192.168.100.64 | Centos7 | 3G | |

1. 重做系统

这一步的目的是确保4个节点是裸机状态。

166：/home/sherry/vagrant-manifest/YYR-v

./vagrant-restart.sh

1. 编译

167：/home/young/workspace-git/core

make

1. 部署

将第2步编译后得到的二进制文件，分别拷贝至”61-64”四个节点，并运行二进制文件。

* 方式一：

167：/home/young/workspace-git/core-deploy-yuan

make

说明：使用ansible分发机制完成二进制文件的拷贝并运行。

* 方式二：

在167上使用scp完成二进制文件的拷贝，并分别在4个节点下运行二进制文件。

测试过程中，可以根据需要，分别在4个节点上管理内核服务:

systemctl status dynas #查询内核服务的状态

systemctl start dynas #启动内核服务

systemctl restart dynas #重启内核服务

systemctl stop dynas #终止内核服务

1. 测试过程
2. 组网情况

（1）组网结果测试

部署完成之后，分别访问四个节点查询组网结果的接口，如下图所示，4个节点已经组网成功。由于访问结果相同，这里仅列出61和62节点的查询结果。

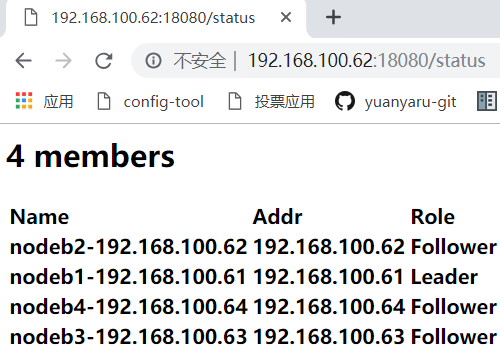
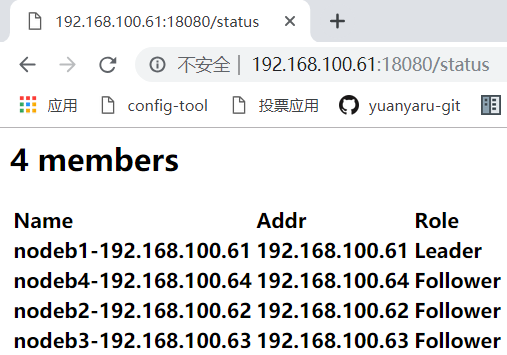


图1.1 节点61、62自组网结果

1. 选主性能测试

在当前leader节点61上手动重启内核服务，观察新leader节点的选取。经过测试，当前leader节点失效，重新选取新的leader节点大概需要2s，这里新选出leader为节点62.





1. Log日志
2. 日志的写入

首先查询当前组网情况：

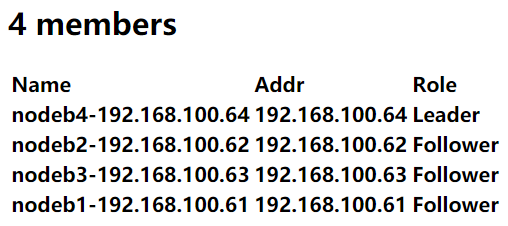


图2.1 集群状态

* Leader节点写入

将级别为debug，内容为debug log test的日志写入leader（192.168.100.64），返回写入的时间戳（1565244528）和格式化的日志内容。



图2.2 日志通过Leader节点写入

* Follower节点写入

选取一个Follower节点（192.168.100.63）写入级别为info，内容为info log test的日志。写入节点返回转发的写入请求，实际写入数据的Leader节点并返回写入的结果。

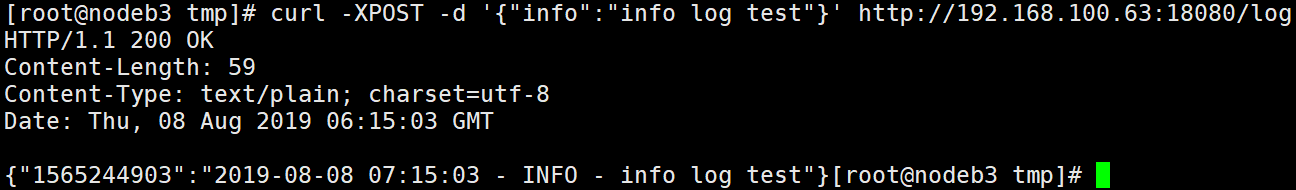
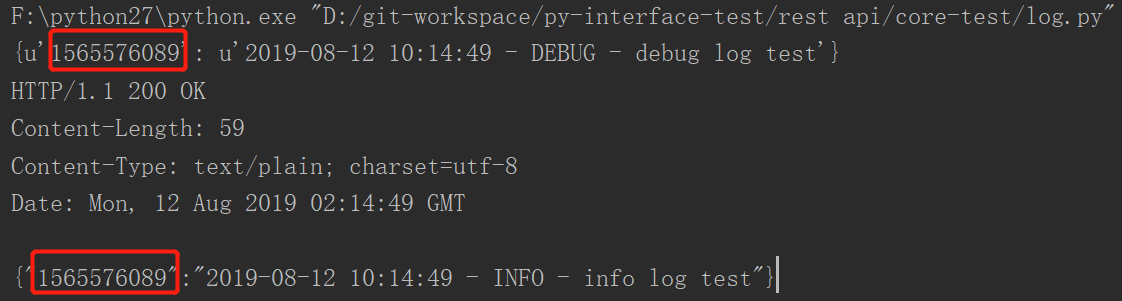


图2.3 日志通过Follower节点写入

存在的问题：

1. 两个节点不能同时写入日志，如果同时写入，后一个会覆盖之前的，如下图可以看出，两次写入日志的时间戳一样，证明日志被覆盖。

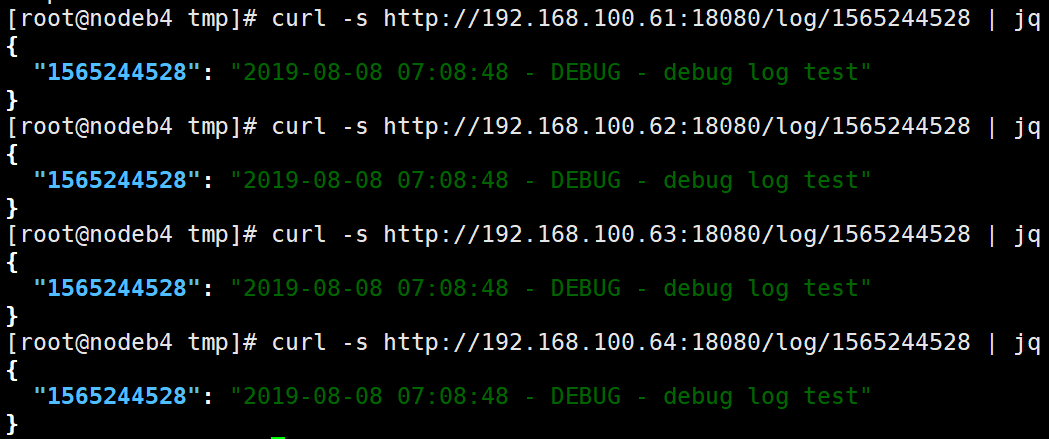


1. 日志的读取

查询操作需要借助处理JSON的命令行工具：jq，因此首先安装jq。

1. 使用时间戳进行查询

首先在所有节点上查询写入的debug级别的日志，时间戳为1565244528：



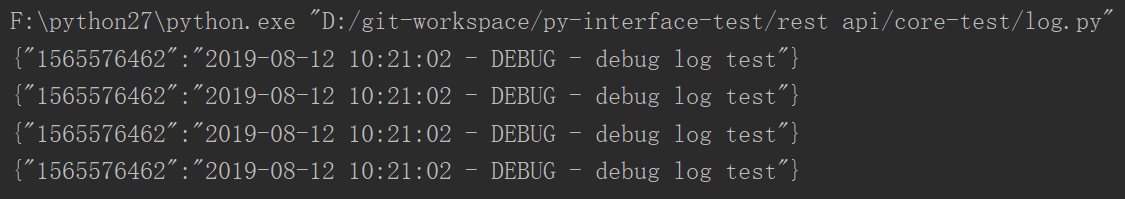


图2.4 查询1565244528条目

由查询结果可知，通过Leader节点写入的日志数据在任何节点都可以查询到，且查询结果相同。

继续在所有节点上查询写入的info级别的日志，时间戳为1565244903：



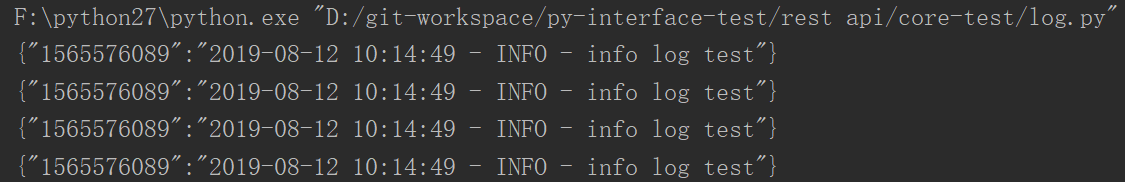


图2.5 查询1565244903条目

由查询结果可知，通过Follower节点写入的日志数据在任何节点都可以查询到，且查询结果相同。

1. 范围查询

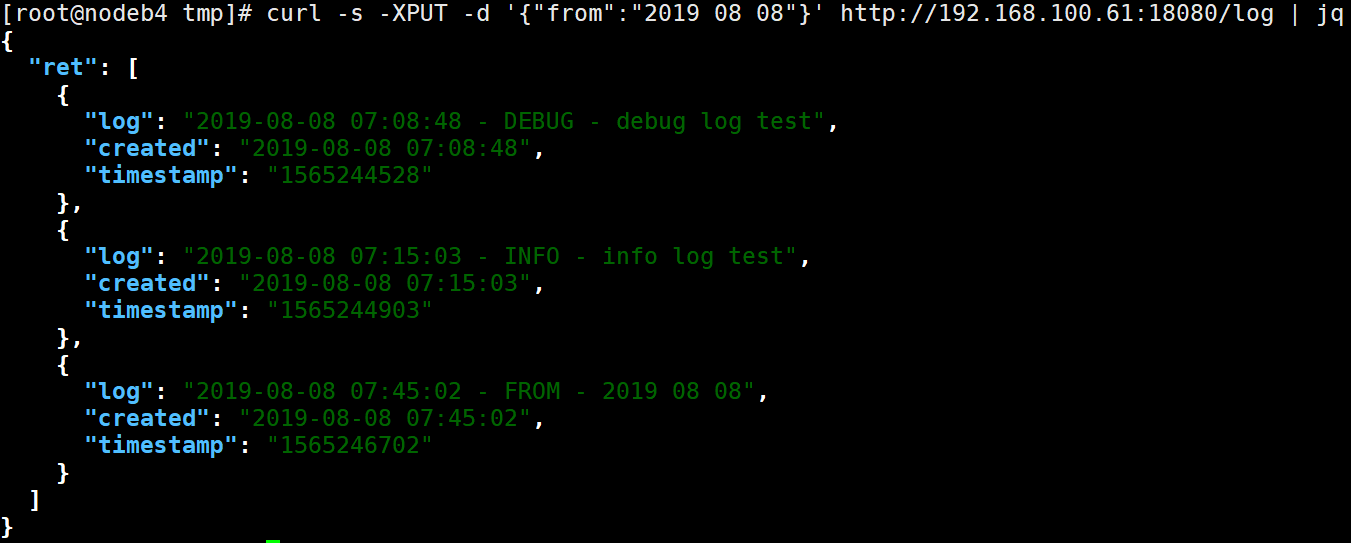


图2.6 在节点192.168.100.61进行范围查询



图2.7 在节点 192.168.100.62 进行范围查询

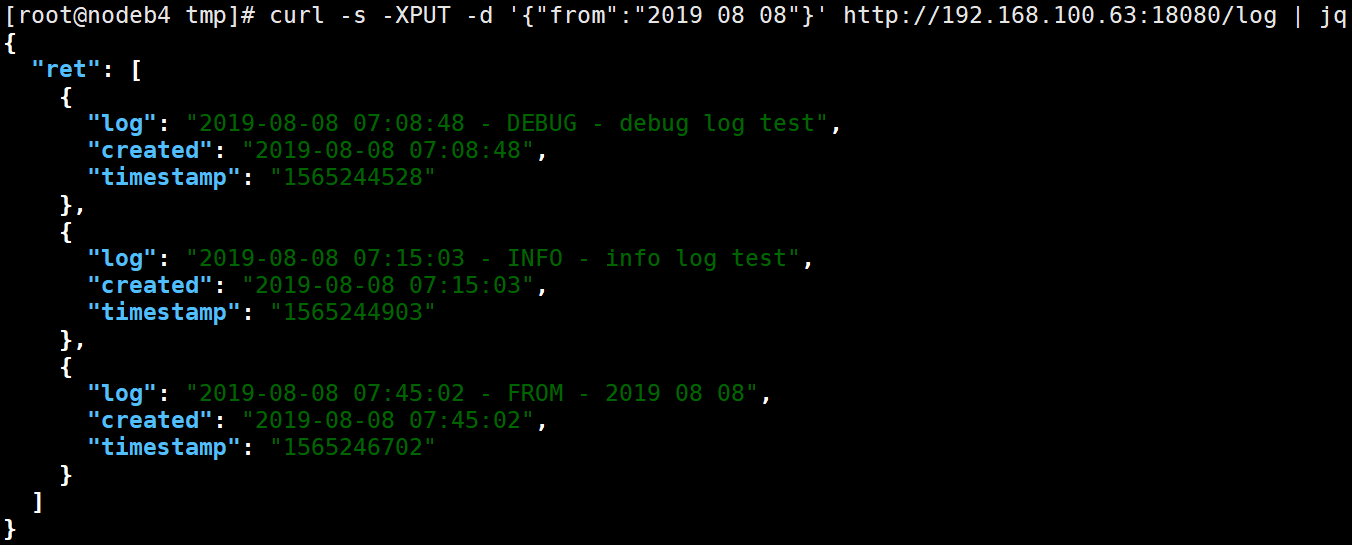


图2.8 在节点 192.168.100.63 进行范围查询

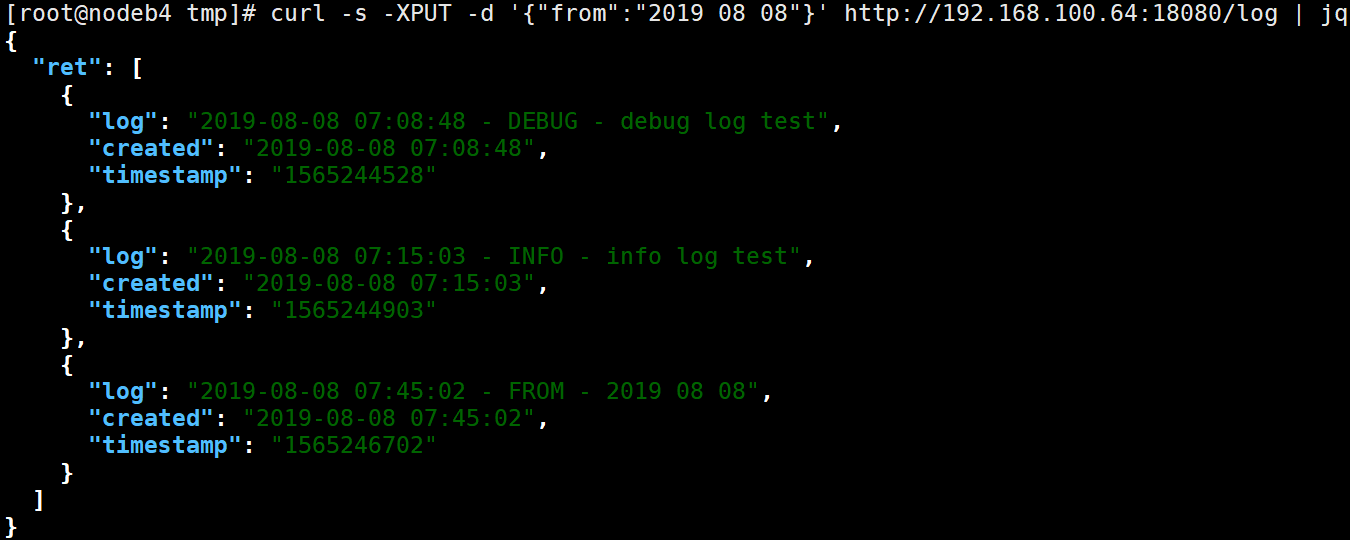
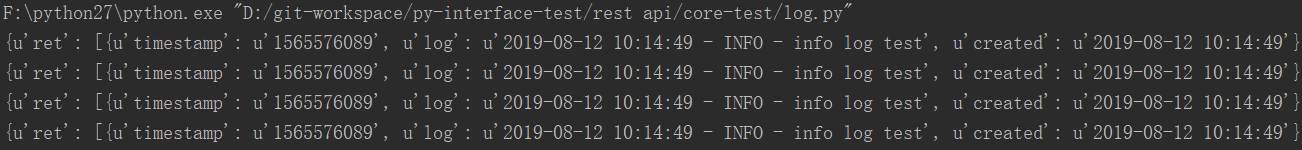
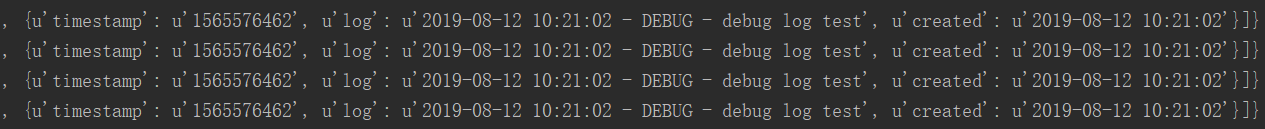


图2.9 在节点 192.168.100.64 进行范围查询

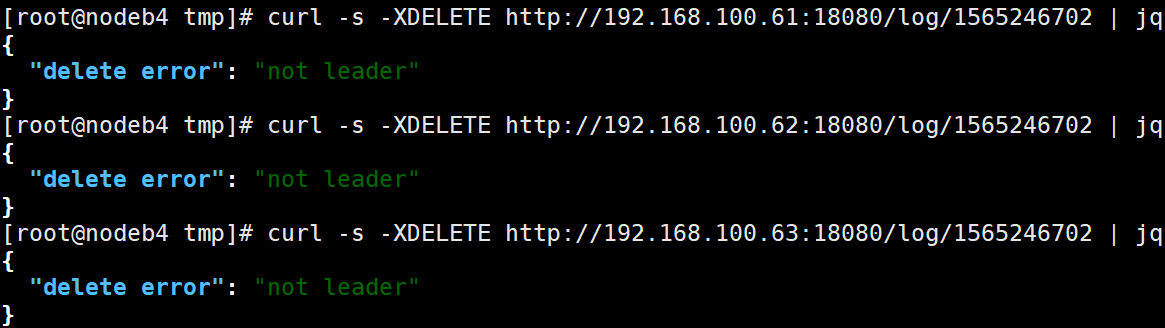




由查询结果可知，写入的日志数据在任何节点中都可以进行范围查询，且查询结果相同。

1. 日志的删除

日志删除只能通过访问 Leader 节点进行操作。



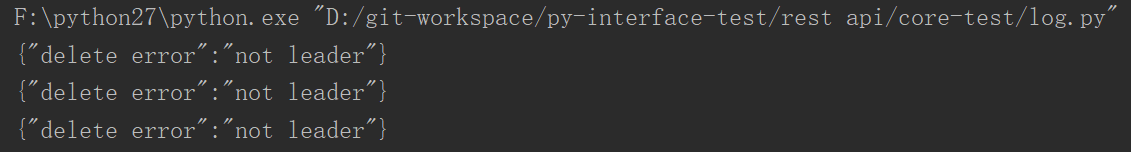
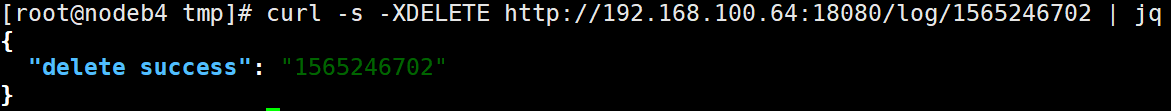


图2.10 在Follower节点删除数据，返回失败信息

三个节点都返回not leader的删除错误，证明无法在Follower节点删除数据。



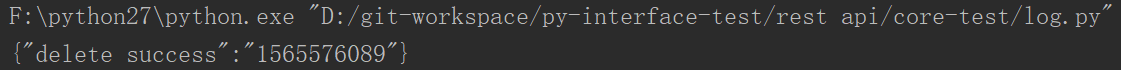
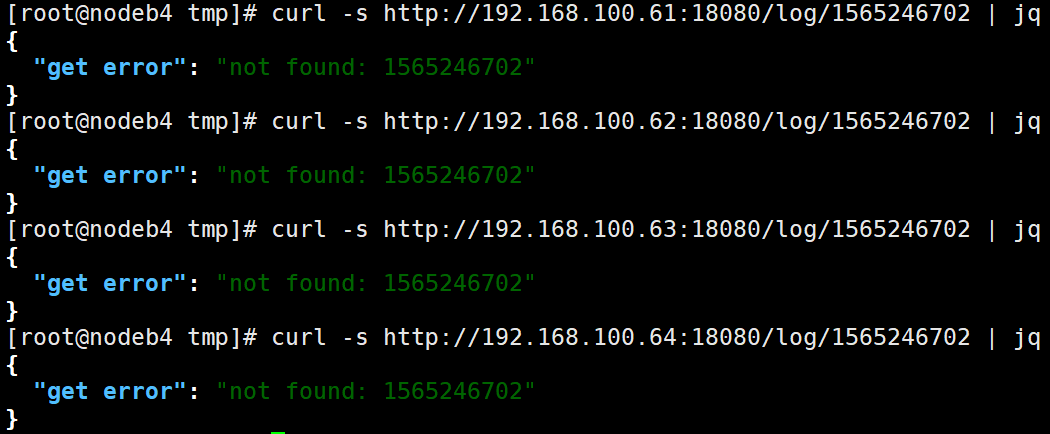


图2.11 在Leader删除数据，返回成功信息

在集群中查询删除的日志：



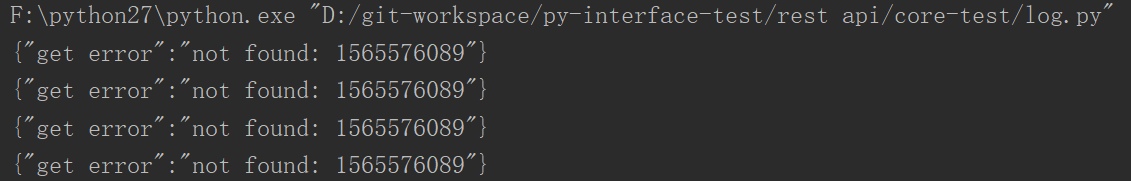
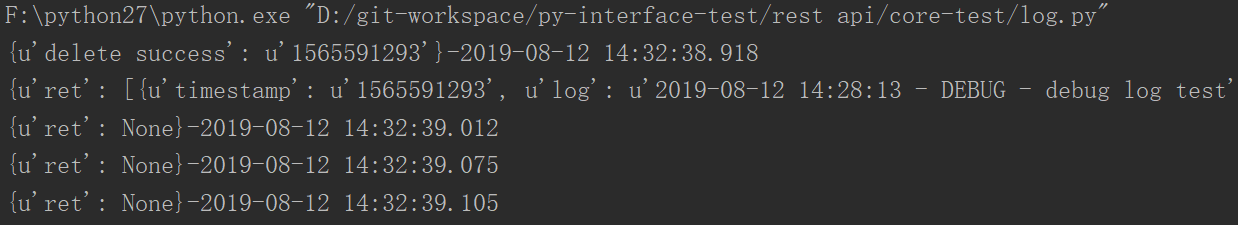


图2.12 查询被删除的条目，返回错误

各节点均返回not found的错误。上述实验验证了删除操作的有效性和一致性。

存在的问题：

1. 删除不同步，显示时间戳为“1565591293”的数据已经删除成功，但是当再次查询集群中各节点的数据时，有一个节点的数据没有立即删除成功。延迟大约0.1s.



1. 分布式共享内存
2. 内存的写入和读取

首先查询当前组网情况：

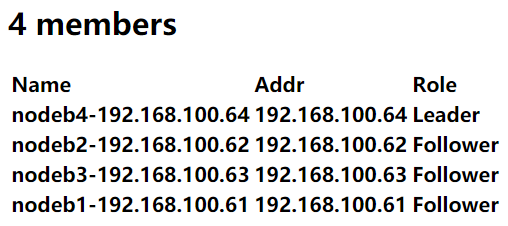
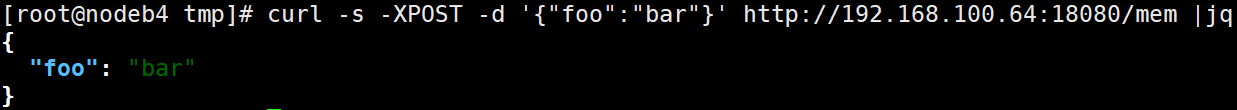


图3.1 集群状态

* Leader节点写入

首先向Leader 节点（192.168.100.64）写入数据（“foo”，“bar”） ：



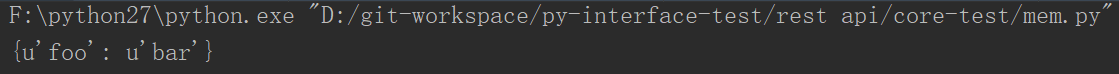
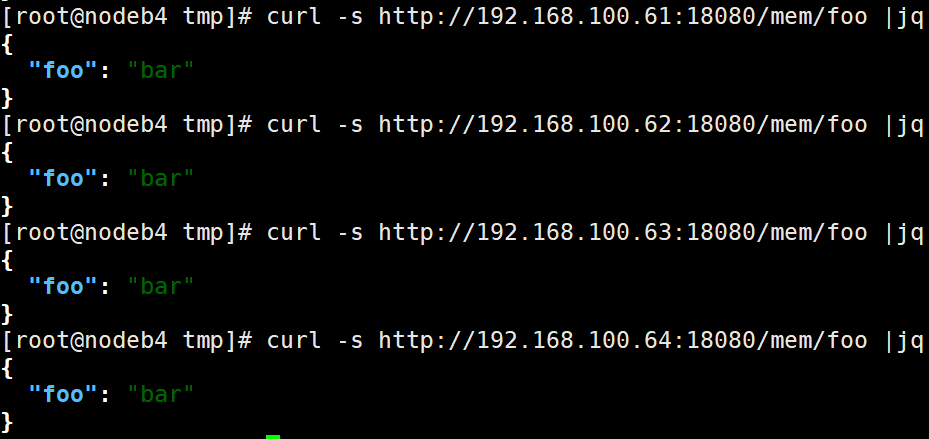


图3.2 向 Leader 节点写入数据，返回被写入的数据

读取：



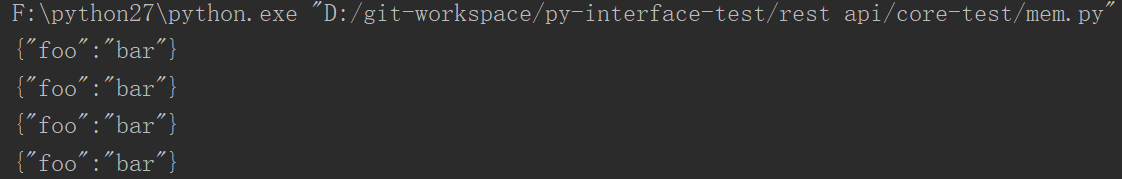
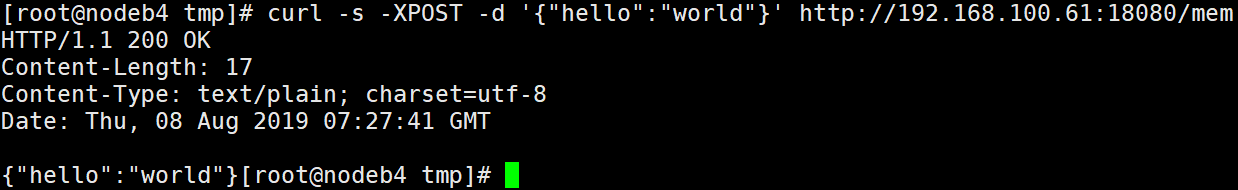


图3.3 在集群所有节点上查询写入的数据

结果显示：通过leader节点写入的数据已被正确地写入到DSM 中，且各节点的数据一致。

* Follower 节点写入

向非 Leader 节点（选取 192.168.100.61）写入数据（“hello”，“world”） ：



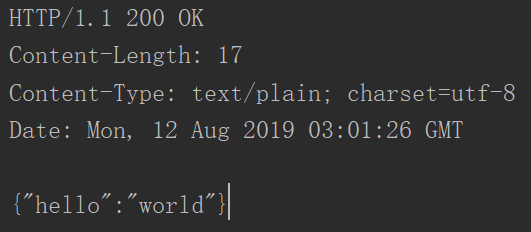
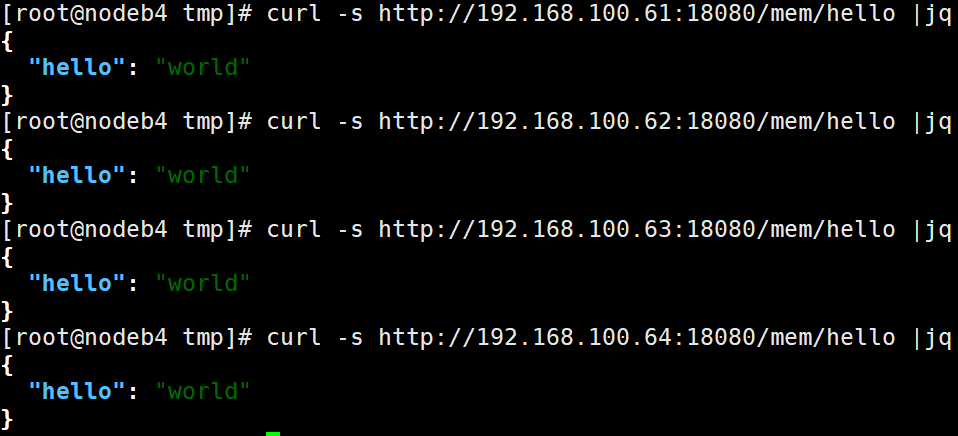


图3.4 向非 Leader 节点写入数据

如图所示， 向非Leader节点发出的写入数据请求，被转发至Leader节点进行处理，并返回被写入的数据。

读取：



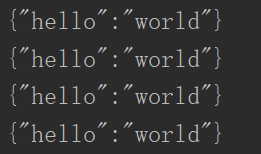


图 3.5 在集群所有节点上查询写入的数据

结果显示：通过follower节点写入的数据数据已被确地写入到 DSM 中，且各节点的数据一致。

1. 内存的删除

删除操作与写入操作类似，都是仅在Leader节点进行，如果其余节点接收到操作请求，则将请求转发到Leader节点，由Leader节点进行处理。以下分别在 Leader节点与非Leader节点进行删除操作。

* 首先在Leader节点（192.168.100.64）上删除之前写入的数据（“hello”，  
  “world”）：



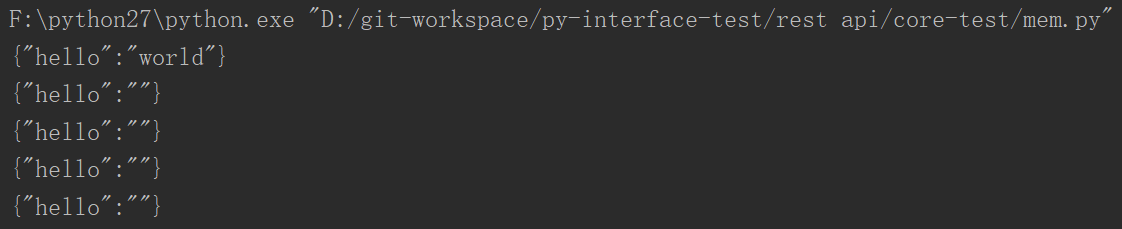


图 3.6 在 Leader 节点删除数据

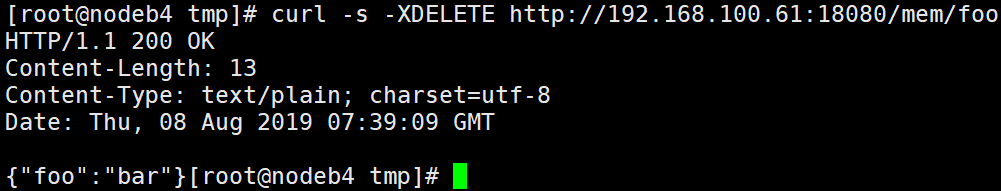
如图所示，在Leader节点删除数据返回被删除的数据。之后，在集群所有节点  
上查询该数据：



图 3.7 在集群中所有节点查询被删除的数据，返回空值

如图所示，在集群中全部节点均查询不到被删除的数据，说明删除操作已经被正  
确执行，且各节点的数据一致。

* 在非 Leader 节点（选取 192.168.100.61）上删除之前写入的数据  
  （“foo”，“bar”）：



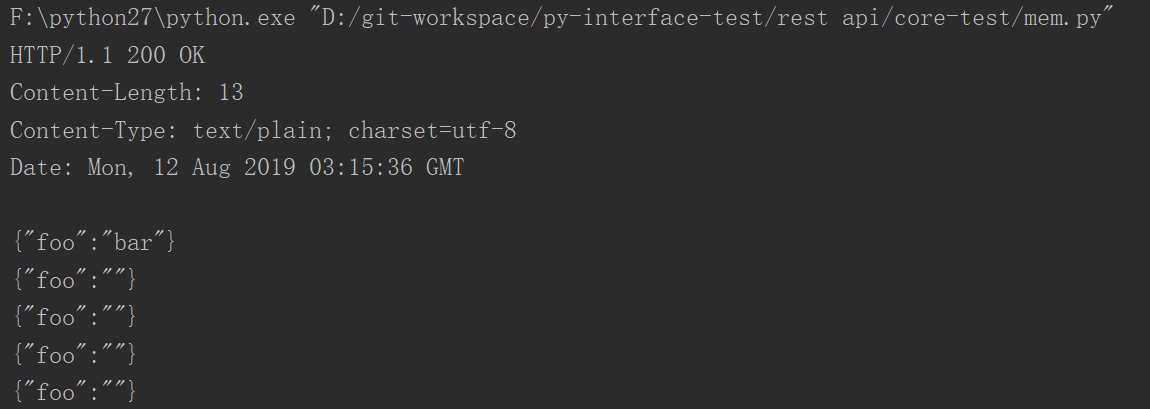


图 3.8 在非Leader节点删除数据

如图所示，向非Leader节点发送删除数据的请求，该请求被转发至Leader节点，  
并返回被删除的数据。之后，在集群所有节点上查询该数据：

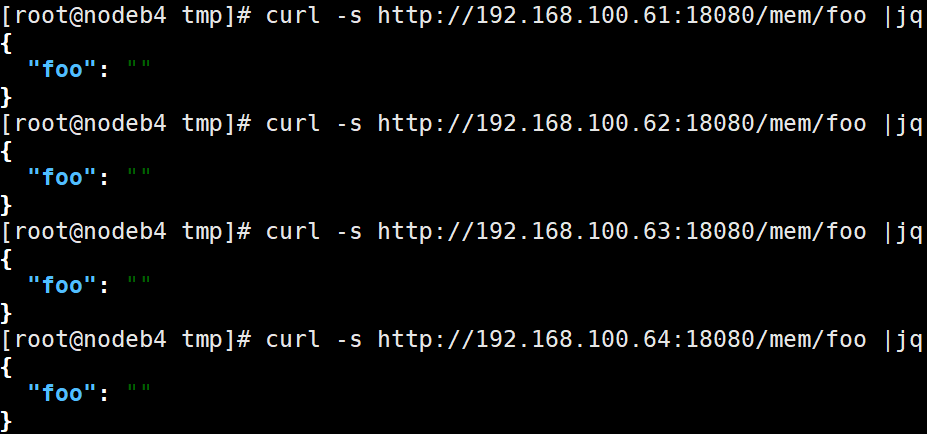


图 3.9 在集群中所有节点查询被删除的数据，返回空值

如图所示，在集群中全部节点均查询不到被删除的数据，说明删除操作已经被正  
确执行，且各节点的数据一致。

1. 内存中的数据同步

事先向DSM中写入（“foo”，“bar”）、（“hello”，“world”）2组数据。



图 3.10 通过统计接口查询，集群中4个节点都已写入

如上图所示，写入的两组数据在全部节点都存在。之后进行同步的实验：

* 在1个节点删除数据，此时数据仍满足Quorum一致性，观察数据在缺失节点的恢复；
* 在2个节点删除数据，此时数据不满足项目设置的Quorum一致性，观察数据在DSM中的删除。

本测试使用重启某个节点的核心进程删除单个节点的数据。

1. 删除1个节点的数据

首先删除64节点的数据，查询各节点的数据情况：

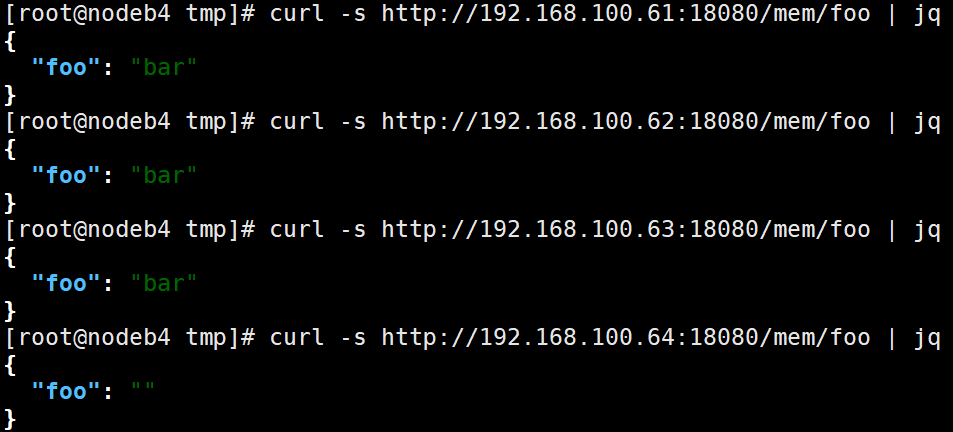


图3.12 查询集群节点中的数据，仅64节点无数据



图3.13 在一个节点重启服务后，使用统计接口查询数据

如上图所示，与图3.10对比，数据驻留节点的数量从4变为3，此时满足项目设定的Quorum一致性，按照设定数据将在缺失节点（192.168.100.64）恢复。

测试结果：观察一段时间后，缺失节点数据恢复成功。

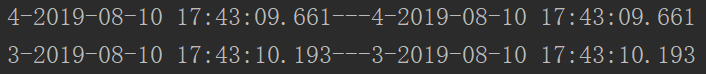
此功能不稳定，存在以下问题：

删除一个节点，驻留节点的数量从4变成3，但数据恢复不成功

删除一个节点，驻留节点的数量从4直接变成2，过段时间后，数据清除；

删除一个节点，驻留节点无变化

性能测试：数据恢复大概需要20s





1. 删除2个节点的数据

在上一步的基础上，再删除62节点的数据。

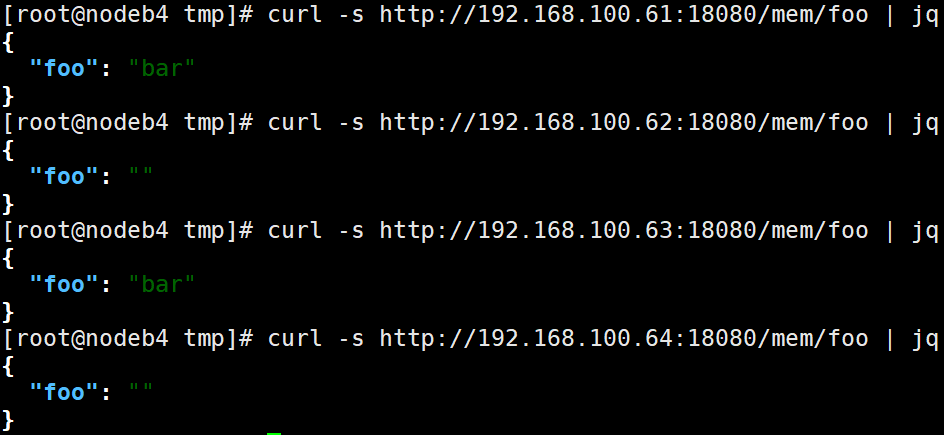


图3.14 查询集群节点中的数据，64、62 节点无数据



图 3.15 在两个节点重启服务后，使用统计接口查询数据

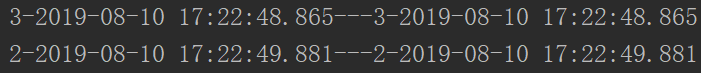
数据驻留的节点从图3.10中的4个变为2，此时数据驻留节点数量不满足

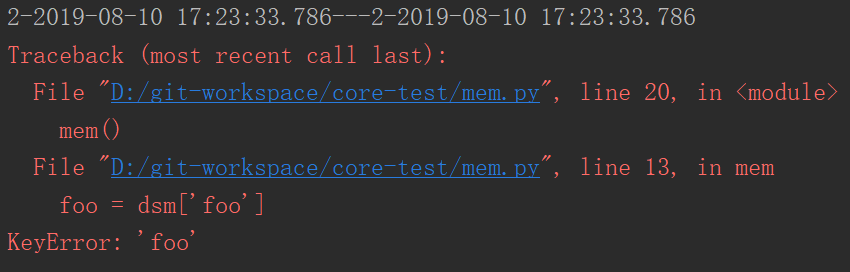
项目设定Quorum一致性，按照项目设定数据将被清除。

测试结果：观察一段时间后，数据在集群中清除。

此功能不稳定，有时候数据清除不成功

性能测试：数据清除大概需要1min





1. K8s安装

节点组网成功之后，访问leader节点的kube-install接口，会开始自动安装k8s平台。由于k8s安装包下载比较慢，提前将下载好的安装包拷贝至任意节点。

* 访问Leader节点



图4.1 [http://192.168.100.61:18080/kube](http://192.168.100.61:18080/status)-install

开始安装k8s,通过实时查看61节点下面的日志文件kube.log，查看安装过程。

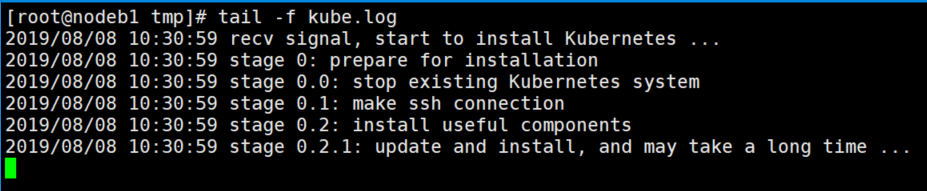


图4.2 k8s安装日志

* 访问非Leader节点



图4.3 [http://192.168.100.62:18080/kube](http://192.168.100.61:18080/status)-install

访问非Leader节点，会提示当前节点不是leader，前往leader节点安装。

1. 测试结果
2. log日志不能同步写入，在两个节点同时写入日志会覆盖

log日志删除，集群内不同节点有误差

1. 分布式共享内存数据同步

Quorum一致性，不稳定

1. K8s安装

在k8s还没有安装完成的情况下，如果leader节点发生变化，会导致k8s安装失败。但是通过查看leader节点下的安装日志，安装过程还是会继续进行下去，但不会安装成功。

