Rabbitmq集群高可用

Rabbitmq集群高可用

RabbitMQ是用erlang开发的，集群非常方便，因为erlang天生就是一门分布式语言,但其本身并不支持负载均衡。

Rabbit模式大概分为以下三种：单一模式、普通模式、镜像模式

单一模式：最简单的情况，非集群模式。

没什么好说的。

普通模式：默认的集群模式。

对于Queue来说，消息实体只存在于其中一个节点，A、B两个节点仅有相同的元数据，即队列结构。

当消息进入A节点的Queue中后，consumer从B节点拉取时，RabbitMQ会临时在A、B间进行消息传输，把A中的消息实体取出并经过B发送给consumer。

所以consumer应尽量连接每一个节点，从中取消息。即对于同一个逻辑队列，要在多个节点建立物理Queue。否则无论consumer连A或B，出口总在A，会产生瓶颈。

该模式存在一个问题就是当A节点故障后，B节点无法取到A节点中还未消费的消息实体。

如果做了消息持久化，那么得等A节点恢复，然后才可被消费；如果没有持久化的话，然后就没有然后了……

镜像模式：把需要的队列做成镜像队列，存在于多个节点，属于RabbitMQ的HA方案。

该模式解决了上述问题，其实质和普通模式不同之处在于，消息实体会主动在镜像节点间同步，而不是在consumer取数据时临时拉取。

该模式带来的副作用也很明显，除了降低系统性能外，如果镜像队列数量过多，加之大量的消息进入，集群内部的网络带宽将会被这种同步通讯大大消耗掉。

所以在对可靠性要求较高的场合中适用(后面会详细介绍这种模式，目前我们搭建的环境属于该模式)

了解集群中的基本概念：

RabbitMQ的集群节点包括内存节点、磁盘节点。顾名思义内存节点就是将所有数据放在内存，磁盘节点将数据放在磁盘。不过，如前文所述，如果在投递消息时，打开了消息的持久化，那么即使是内存节点，数据还是安全的放在磁盘。

一个rabbitmq集 群中可以共享
user，vhost，queue，exchange等，所有的数据和状态都是必须在所有节点上复制的，一个例外是，那些当前只属于创建它的节点的消息队列，尽管它们可见且可被所有节点读取。rabbitmq节点可以动态的加入到集群中，一个节点它可以加入到集群中，也可以从集群环集群会进行一个基本的负载均衡。
集群中有两种节点：
1 内存节点：只保存状态到内存（一个例外的情况是：持久的queue的持久内容将被保存到disk）
2 磁盘节点：保存状态到内存和磁盘。
内存节点虽然不写入磁盘，但是它执行比磁盘节点要好。集群中，只需要一个磁盘节点来保存状态 就足够了
如果集群中只有内存节点，那么不能停止它们，否则所有的状态，消息等都会丢失。

思路：
那么具体如何实现RabbitMQ高可用，我们先搭建一个普通集群模式，在这个模式基础上再配置镜像模式实现高可用，Rabbit集群前增加一个反向代理，生产者、消费者通过反向代理访问RabbitMQ集群。

架构图如下：图片来自http://www.nsbeta.info

上述图里是3个RabbitMQ运行在同一主机上，分别用不同的服务端口。当然我们的生产实际里，多个RabbitMQ肯定是运行在不同的物理服务器上，否则就失去了高可用的意义。

- 集群模式配置

> 设计架构可以如下：在一个集群里，有4台机器，其中1台使用磁盘模式，另2台使用内存模式。2台内存模式的节点，无疑速度更快，因此客户端（consumer、producer）连接访问它们。而磁盘模式的节点，由于磁盘IO相对较慢，因此仅作数据备份使用，另外一台作为反向代理。
>
> 四台服务器hostname分别为：queue 、panyuntao1、panyuntao2、panyuntao3（ip:172.16.3.110）
>
> 配置RabbitMQ集群非常简单，只需要几个命令，配置步骤如下：

step1：queue、panyuntao1、panyuntao2做为RabbitMQ集群节点，分别安装RabbitMq-Server ，安装后分别启动RabbitMq-server

启动命令 # Rabbit-Server start ，安装过程及启动命令参见：http://www.cnblogs.com/flat_peach/archive/2013/03/04/2943574.html

step2：在安装好的三台节点服务器中，分别修改/etc/hosts文件，指定queue、panyuntao1、panyuntao2的hosts，如：

> 172.16.3.32 queue
>
> 172.16.3.107 panyuntao1
>
> 172.16.3.108 panyuntao2

> 还有hostname文件也要正确，分别是queue、panyuntao1、panyuntao2，如果修改hostname建议安装rabbitmq前修改。
>
> 请注意RabbitMQ集群节点必须在同一个网段里，如果是跨广域网效果就差。

step3：设置每个节点Cookie

> Rabbitmq的集群是依赖于erlang的集群来工作的，所以必须先构建起erlang的集群环境。Erlang的集群中各节点是通过一个magic cookie来实现的，这个cookie存放在/var/lib/rabbitmq/.erlang.cookie中，文件是400的权限。所以必须保证各节点cookie保持一致，否则节点之间就无法通信。
> -r--------. 1 rabbitmq rabbitmq 20 3月 5 00:00 /var/lib/rabbitmq/.erlang.cookie
> 将其中一台节点上的.erlang.cookie值复制下来保存到其他节点上。或者使用scp的方法也可，但是要注意文件的权限和属主属组。
> 我们这里将queue中的cookie 复制到 panyuntao1、panyuntao2中，先修改下panyuntao1、panyuntao2中的.erlang.cookie权限
> #chmod 777/var/lib/rabbitmq/.erlang.cookie
> 将queue的/var/lib/rabbitmq/.erlang.cookie这个文件，拷贝到panyuntao1、panyuntao2的同一位置（反过来亦可），该文件是集群节点进行通信的验证密钥，所有节点必须一致。拷完后重启下RabbitMQ。
> 复制好后别忘记还原.erlang.cookie的权限，否则可能会遇到错误
> #chmod 400 /var/lib/rabbitmq/.erlang.cookie
> 设置好cookie后先将三个节点的rabbitmq重启
> #rabbitmqctl stop
> #rabbitmq-server start

step4：停止所有节点RabbitMq服务，然后使用detached参数独立运行，这步很关键，尤其增加节点停止节点后再次启动遇到无法启动都可以参照这个顺序

```
queue#rabbitmqctl stop
panyuntao1#rabbitmqctl stop
panyuntao2#rabbitmqctl stop

queue# rabbitmq-server -detached
panyuntao1# rabbitmq-server -detached
panyuntao2# rabbitmq-server -detached
```

分别查看下每个节点

```
queue# rabbitmqctl cluster_status
Cluster status of node rabbit@queue ...
```

> [{nodes,[{disc,[rabbit@queue]}]},
> {running_nodes,[rabbit@queue]},
> {partitions,[]}]
> ...done.

```
panyuntao1#rabbitmqctl cluster_status
Cluster status of node rabbit@panyuntao1...
```

> [{nodes,[{disc,[rabbit@panyuntao1]}]},
>
> {running_nodes,[rabbit@panyuntao1]},
>
> {partitions,[]}]
> ...done.

```
panyuntao2#rabbitmqctl cluster_status
   Cluster status of node rabbit@panyuntao2...
```

> [{nodes,[{disc,[rabbit@panyuntao2]}]},
>
> {running_nodes,[rabbit@panyuntao2]},
>
> {partitions,[]}]
> ...done.

step4：将panyuntao1、panyuntao2作为内存节点与queue连接起来，在panyuntao1上，执行如下命令：

```
panyuntao1# rabbitmqctl stop_app
```

> panyuntao1# rabbitmqctl join_cluster --ram rabbit@queue
>
> panyuntao1# rabbitmqctl start_app
>
> panyuntao2# rabbitmqctl stop_app
> panyuntao2# rabbitmqctl join_cluster --ram rabbit@queue（上方已经将panyuntao1与queue连接，也可以直接将panyuntao2

> 与panyuntao1连接，同样而已加入集群中）
>
> panyuntao2# rabbitmqctl start_app

上述命令先停掉rabbitmq应用，然后调用cluster命令，将panyuntao1连接到，使两者成为一个集群，最后重启rabbitmq应用。在这个cluster命令下，panyuntao1、panyuntao2是内存节点，queue是磁盘节点（RabbitMQ启动后，默认是磁盘节点）。
queue如果要使panyuntao1或panyuntao2在集群里也是磁盘节点，join_cluster命令去掉--ram参数即可
#rabbitmqctl join_cluster rabbit@queue

只要在节点列表里包含了自己，它就成为一个磁盘节点。在RabbitMQ集群里，必须至少有一个磁盘节点存在。

step5：在queue、panyuntao1、panyuntao2上，运行cluster_status命令查看集群状态：

```
[root@queue ~]# rabbitmqctl cluster_status
Cluster status of node rabbit@queue ...
[{nodes,[{disc,[rabbit@queue]},{ram,[rabbit@panyuntao2,rabbit@panyuntao1]}]},
{running_nodes,[rabbit@panyuntao2,rabbit@panyuntao1,rabbit@queue]},
{partitions,[]}]
...done.

[root@panyuntao1 rabbitmq]# rabbitmqctl cluster_status
Cluster status of node rabbit@panyuntao1 ...
[{nodes,[{disc,[rabbit@queue]},{ram,[rabbit@panyuntao2,rabbit@panyuntao1]}]},
{running_nodes,[rabbit@panyuntao2,rabbit@queue,rabbit@panyuntao1]},
{partitions,[]}]
...done.

[root@panyuntao2 rabbitmq]# rabbitmqctl cluster_status
Cluster status of node rabbit@panyuntao2 ...
[{nodes,[{disc,[rabbit@queue]},{ram,[rabbit@panyuntao2,rabbit@panyuntao1]}]},
{running_nodes,[rabbit@panyuntao1,rabbit@queue,rabbit@panyuntao2]},
{partitions,[]}]
...done.
```

这时我们可以看到每个节点的集群信息，分别有两个内存节点一个磁盘节点

step6:往任意一台集群节点里写入消息队列，会复制到另一个节点上，我们看到两个节点的消息队列数一致：（如何发送消息参见：http://www.cnblogs.com/flat_peach/archive/2013/03/04/2943574.html）

```
root@panyuntao2:~# rabbitmqctl list_queues -p hrsystem

Listing queues …
test_queue 10000
…done.
root@panyuntao1:~# rabbitmqctl list_queues -p hrsystem
Listing queues …
test_queue 10000
…done.

root@queue:~# rabbitmqctl list_queues -p hrsystem
Listing queues …
test_queue10000
…done.
```

-p参数为vhost名称

这样RabbitMQ集群就正常工作了，

这种模式更适合非持久化队列，只有该队列是非持久的，客户端才能重新连接到集群里的其他节点，并重新创建队列。假如该队列是持久化的，那么唯一办法是将故障节点恢复起来。

为什么RabbitMQ不将队列复制到集群里每个节点呢？这与它的集群的设计本意相冲突，集群的设计目的就是增加更多节点时，能线性的增加性能（CPU、内存）和容量（内存、磁盘）。理由如下：

> 1. storage space: If every cluster node had a full copy of every queue, adding nodes wouldn't give you more storage capacity. For example, if one node could store 1GB of messages, adding two more nodes would simply give you two more copies of the same 1GB of messages.

> 2. performance: Publishing messages would require replicating those messages to every cluster node. For durable messages that would require triggering disk activity on all nodes for every message. Your network and disk load would increase every time you added a node, keeping the performance of the cluster the same (or possibly worse).

当然RabbitMQ新版本集群也支持队列复制（有个选项可以配置）。比如在有五个节点的集群里，可以指定某个队列的内容在2个节

> 点上进行存储，从而在性能与高可用性之间取得一个平衡。

- 镜像模式配置

> 上面配置RabbitMQ默认集群模式，但并不保证队列的高可用性，尽管交换机、绑定这些可以复制到集群里的任何一个节点，但是队列内容不会复制，虽然该模式解决一部分节点压力，但队列节点宕机直接导致该队列无法使用，只能等待重启，所以要想在队列节点宕机或故障也能正常使用，就要复制队列内容到集群里的每个节点，需要创建镜像队列。
> 我们看看如何镜像模式来解决复制的问题，从而提高可用性

step1:增加负载均衡器

> 关于负载均衡器，商业的比如F5的BIG-IP，Radware的AppDirector，是硬件架构的产品，可以实现很高的处理能力。但这些产品昂贵的价格会让人止步，所以我们还有软件负载均衡方案。互联网公司常用的软件LB一般有LVS、HAProxy、Nginx等。LVS是一个内核层的产品，主要在第四层负责数据包转发，使用较复杂。HAProxy和Nginx是应用层的产品，但Nginx主要用于处理HTTP，所以这里选择HAProxy作为RabbitMQ前端的LB。
> HAProxy的安装使用非常简单，在Centos下直接yum install haproxy，然后更改/etc/haproxy/haproxy.cfg
> 文件即可，文件内容大概如下：

```
#---------------------------------------------------------------------
 defaults
mode http
log global
option httplog
option dontlognull
option http-server-close
option forwardfor except 127.0.0.0/8
option redispatch
retries 3
timeout http-request 10s
timeout queue 1m
timeout connect 10s
timeout client 1m
timeout server 1m
timeout http-keep-alive 10s
timeout check 10s
maxconn 3000

 listen rabbitmq_cluster 0.0.0.0:5672
mode tcp
balance roundrobin
server rqslave1 172.16.3.107:5672 check inter 2000 rise 2 fall 3
server rqslave2 172.16.3.108:5672 check inter 2000 rise 2 fall 3
# server rqmaster 172.16.3.32:5672 check inter 2000 rise 2 fall 3
#---------------------------------------------------------------------
```

负载均衡器会监听5672端口，轮询我们的两个内存节点172.16.3.107、172.16.3.108的5672端口，172.16.3.32为磁盘节点，只做备份不提供给生产者、消费者使用，当然如果我们服务器资源充足情况也可以配置多个磁盘节点
，这样磁盘节点除了故障也不会影响，除非同时出故障。

step2:配置策略

使用Rabbit镜像功能，需要基于rabbitmq策略来实现，政策是用来控制和修改群集范围的某个vhost队列行为和Exchange行为

cluster

 # rabbitmqctl set_policy -p hrsystem ha-allqueue"^" '{"ha-mode":"all"}'

vhosthrsystemha-allqueue, all

“^”

rabbitmqctl set_policy -p hrsystem ha-allqueue "^message" '{"ha-mode":"all"}'

"^message" 这个规则要根据自己修改，这个是指同步"message"开头的队列名称，我们配置时使用的应用于所有队列，所以表达式为"^"
set_policy

set_policy[-pvhostpath] {name} {pattern} {definition} [priority]
（http://www.rabbitmq.com/man/rabbitmqctl.1.man.html）

ha-mode:

ha-mode	ha-params	Result
all	(absent)	Queue is mirrored across all nodes in the cluster. When a new node is added to the cluster, the queue will be mirrored to that node.
exactly	count	Queue is mirrored tocountnodes in the cluster. If there are less thancountnodes in the cluster, the queue is mirrored to all nodes. If there are more thancountnodes in the cluster, and a node containing a mirror goes down, then a new mirror willnotbe created on another node. (This is to prevent queues migrating across a cluster as it is brought down.)
nodes	node names	Queue is mirrored to the nodes listed innode names. If any of those node names are not a part of the cluster, this does not constitute an error. If none of the nodes in the list are online at the time when the queue is declared then the queue will be created on the node that the declaring client is connected to.

step3：
创建队列时需要指定ha 参数，如果不指定x-ha-prolicy 的话将无法复制

下面C#代码片段

```
using(varbus =RabbitHutch.CreateBus(ConfigurationManager.ConnectionStrings["RabbitMQ"].ToString()))
 {
 bus.Subscribe<TestMessage>("word_subscriber", message => RunTable(message),x=>x.WithArgument("x-ha-policy","all"));
Console.WriteLine("Subscription Started. Hit any key quit");
Console.ReadKey();
 }
```

step4：
客户端使用负载服务器172.16.3.110
（panyuntao3）发送消息，队列会被复制到所有节点，当然策略也可以配置制定某几个节点，这时任何节点故障
、或者重启将不会影响我们正常使用某个队列
到这里我们完成了高可用配置（所有节点都宕机那没有办法了）。

使用rabbitmq管理端可以看到集群镜像模式中对列状态

参考：

http://www.rabbitmq.com/clustering.html

http://www.rabbitmq.com/ha.html

http://www.rabbitmq.com/parameters.html#policies

http://www.nsbeta.info/archives/555

http://blog.csdn.net/linvo/article/details/7793706