golang哈希一致性算法实践

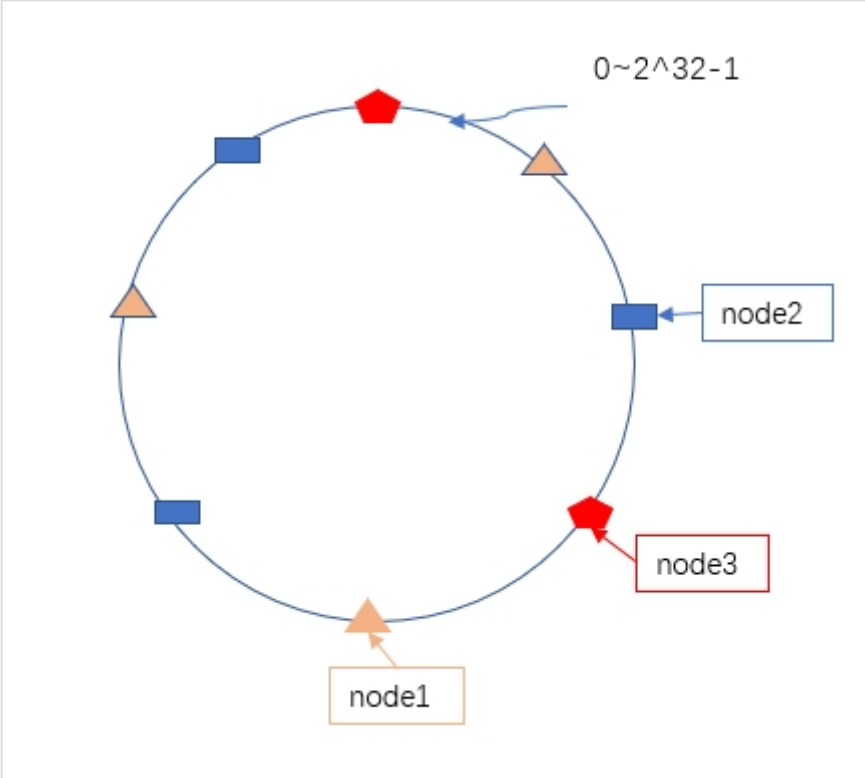
2019-05-18 | Golang | 21

原理介绍

最近在项目中用到哈希一致性算法，它的需求是将入库的视频根据id均匀的分配到不同的容器中，当增加或者减少容器时，使得任务状态更改尽可能的少，于是想到了哈希一致性。

在做负载均衡时，简单的做法是将请求按照某个规则对服务器数量取模。取模的问题是当服务器数量增加或者减少时，会对原来的取模关系有非常大的影响。这在需要数据迁移或者更改服务状态的情况很难接受，hash一致性能在满足负载均衡的同时，尽可能少的更改服务状态或者数据迁移的工作量。

- 哈希环：用一个环表示0~2^32-1取值范围
- 节点映射： 根据节点标识信息计算出0~2^32-1的值，然后映射到哈希环上
- **虚拟节点**： 当节点数量很少时，映射关系较不确定，会导致节点在哈希环上分布不均匀，无法实现复杂均衡的效果，因此通常会引入虚拟节点。例如假设有3个节点对外提供服务，将3个节点映射到哈希环上很难保证分布均匀，如果将3个节点虚拟成1000个节点甚至更多节点，它们在哈希环上就会相对均匀。有些情况我们还会为每个节点设置权重例如node1、node2、node3的权重分别为1、2、3，假设虚拟节点总数为1200个，那么哈希环上将会有200个node1、400个node2、600个node3节点
- 将key值映射到节点： 以同样的映射关系将key映射到哈希环上，以顺时针的方式找到第一个值比key的哈希大的节点。
- **增加或者删除节点**：关于增加或者删除节点有多种不同的做法，常见的做法是剩余节点的权重值，重新安排虚拟的数量。例如上述的node1，node2和node3中，假设node3节点被下线，新的哈希环上会映射有有400个node1和800个node2。要注意的是原有的200个node1和400个node2会在相同的位置，但是会在之前的空闲区间增加了node1或者node2节点，因为权重的关系有些情况也会导致原有虚拟的节点的减少。
- **任务(数据更新)**：由于哈希环上节点映射更改，需要更新任务的状态。具体的做法是对每个任务映射状态进行检查，可以发现大多数任务的映射关系都保持不变，只有少量任务映射关系发生改变。总体来说就是**全状态检查，少量更改**。



实践

目前，Golang关于hash一致性有多种开源实现，因此实践起来也不是很难。这里参考 https://github.com/g4zhuj/hashring, 根据自己的理解做了一些修改，并在项目中使用。

核心代码：hash_ring.go

```
package hashring

import (
        "crypto/sha1"
        "sync"
        "fmt"
        "math"
        "sort"
        "strconv"
)

/*
        https://github.com/g4zhuj/hashring
        https://segmentfault.com/a/1190000013533592
*/

const (
        //DefaultVirualSpots default virual spots
        DefaultTotalVirualSpots = 1000
)

type virtualNode struct {
```

```
        nodeKey   string
        nodeValue uint32
}
type nodesArray []virtualNode

func (p nodesArray) Len() int           { return len(p) }
func (p nodesArray) Less(i, j int) bool { return p[i].nodeValue < p[j].nodeValue }
func (p nodesArray) Swap(i, j int)      { p[i], p[j] = p[j], p[i] }
func (p nodesArray) Sort()              { sort.Sort(p) }

//HashRing store nodes and weigths
type HashRing struct {
        total           int            //total number of virtual node
        virtualNodes    nodesArray     //array of virtual nodes sorted by value
        realNodeWeights map[string]int //Node:weight
        mu              sync.RWMutex
}

//NewHashRing create a hash ring with virual spots
func NewHashRing(total int) *HashRing {
        if total == 0 {
                total = DefaultTotalVirualSpots
        }

        h := &HashRing{
                total:           total,
                virtualNodes:    nodesArray{},
                realNodeWeights: make(map[string]int),
        }
        h.buildHashRing()
        return h
}

//AddNodes add nodes to hash ring
func (h *HashRing) AddNodes(nodeWeight map[string]int) {
        h.mu.Lock()
        defer h.mu.Unlock()
        for nodeKey, weight := range nodeWeight {
                h.realNodeWeights[nodeKey] = weight
        }
        h.buildHashRing()
}

//AddNode add node to hash ring
func (h *HashRing) AddNode(nodeKey string, weight int) {
        h.mu.Lock()
        defer h.mu.Unlock()
        h.realNodeWeights[nodeKey] = weight
        h.buildHashRing()
}

//RemoveNode remove node
```

```
75  func (h *HashRing) RemoveNode(nodeKey string) {
76          h.mu.Lock()
77          defer h.mu.Unlock()
78          delete(h.realNodeWeights, nodeKey)
79          h.buildHashRing()
80  }
81
82  //UpdateNode update node with weight
83  func (h *HashRing) UpdateNode(nodeKey string, weight int) {
84          h.mu.Lock()
85          defer h.mu.Unlock()
86          h.realNodeWeights[nodeKey] = weight
87          h.buildHashRing()
88  }
89
90  func (h *HashRing) buildHashRing() {
91          var totalW int
92          for _, w := range h.realNodeWeights {
93                  totalW += w
94          }
95          h.virtualNodes = nodesArray{}
96          for nodeKey, w := range h.realNodeWeights {
97                  spots := int(math.Floor(float64(w) / float64(totalW) * float64(h.total))
98                  for i := 1; i <= spots; i++ {
99                          hash := sha1.New()
100                         hash.Write([]byte(nodeKey + ":" + strconv.Itoa(i)))
101                         hashBytes := hash.Sum(nil)
102
103                         oneVirtualNode := virtualNode{
104                                 nodeKey:   nodeKey,
105                                 nodeValue: genValue(hashBytes[6:10]),
106                         }
107                         h.virtualNodes = append(h.virtualNodes, oneVirtualNode)
108
109                         hash.Reset()
110                 }
111         }
112         // sort virtual nodes for quick searching
113         h.virtualNodes.Sort()
114 }
115
116 func genValue(bs []byte) uint32 {
117         if len(bs) < 4 {
118                 return 0
119         }
120         v := (uint32(bs[3]) << 24) | (uint32(bs[2]) << 16) | (uint32(bs[1]) << 8) | (uir
121         return v
122 }
123
124 //GetNode get node with key
125 func (h *HashRing) GetNode(s string) string {
126         h.mu.RLock()
```

```
        defer h.mu.RUnlock()
        if len(h.virtualNodes) == 0 {
                fmt.Println("no valid node in the hashring")
                return ""
        }
        hash := sha1.New()
        hash.Write([]byte(s))
        hashBytes := hash.Sum(nil)
        v := genValue(hashBytes[6:10])
        i := sort.Search(len(h.virtualNodes), func(i int) bool { return h.virtualNodes[i
        //ring
        if i == len(h.virtualNodes) {
                i = 0
        }
        return h.virtualNodes[i].nodeKey
}
```

测试：hashring_test.go

```
package hashring

import (
        "fmt"
        "testing"
)

func TestHashRing(t *testing.T) {
        realNodeWeights := make(map[string]int)
        realNodeWeights["node1"] = 1
        realNodeWeights["node2"] = 2
        realNodeWeights["node3"] = 3

        totalVirualSpots := 100

        ring := NewHashRing(totalVirualSpots)
        ring.AddNodes(realNodeWeights)
        fmt.Println(ring.virtualNodes, len(ring.virtualNodes))
        fmt.Println(ring.GetNode("1845"))  //node3
        fmt.Println(ring.GetNode("994"))   //node1
        fmt.Println(ring.GetNode("hello")) //node3


        //remove node
        ring.RemoveNode("node3")
        fmt.Println(ring.GetNode("1845"))  //node2
        fmt.Println(ring.GetNode("994"))   //node1
        fmt.Println(ring.GetNode("hello")) //node2

        //add node
        ring.AddNode("node4", 2)
```

```
        fmt.Println(ring.GetNode("1845"))  //node4
        fmt.Println(ring.GetNode("994"))   //node1
        fmt.Println(ring.GetNode("hello")) //node4

        //update the weight of node
        ring.UpdateNode("node1", 3)
        fmt.Println(ring.GetNode("1845"))  //node4
        fmt.Println(ring.GetNode("994"))   //node1
        fmt.Println(ring.GetNode("hello")) //node1
        fmt.Println(ring.realNodeWeights)
}
```

‹ golang mysql基本使用