← 返回博客列表

原 PHP: 深入了解一致性哈希

陈亦

发布时间: 2014/02/27 12:46 阅读: 4338 收藏: 32 点赞: 4 评论: 13

摘要

随着memcache、redis以及其它一些内存K/V数据库的流行，一致性哈希也越来越被开发者所了解。因为这些内存K/V数据库大多不提供分布式支持(本文以redis为例)，所以如果要提供多台redis server来提供服务的话，就需要解决如何将数据分散到redis server，并且在增减redis server时如何最大化的不令数据重新分布，这将是本文讨论的范畴。

随着memcache、redis以及其它一些内存K/V数据库的流行，一致性哈希也越来越被开发者所了解。因为这些内存K/V数据库大多不提供分布式支持(本文以redis为例)，所以如果要提供多台redis server来提供服务的话，就需要解决如何将数据分散到redis server，并且在增减redis server时如何最大化的不令数据重新分布，这将是本文讨论的范畴。

取模算法

取模运算通常用于得到某个半开区间内的值：m % n = v，其中n不为0，值v的半开区间为：[0, n)。取模运算的算法很简单：有正整数k，并令k使得k和n的乘积最大但不超过m，则v的值为：m - kn。比如1 % 5，令k = 0，则k * 5的乘积最大并不超过1，故结果v = 1 - 0 * 5 = 1。

我们在分表时也会用到取模运算。如一个表要划分三个表，则可对3进行取模，因为结果总是在[0, 3)之内，也就是取值为：0、1、2。

但是对于应用到redis上，这种方式就不行了，因为太容易冲突了。

哈希(Hash)

Hash，一般翻译做“散列”，也有直接音译为"哈希"的，就是把任意长度的输入（又叫做预映射， pre-image），通过散列算法，变换成固定长度的输出，该输出就是散列值。这种转换是一种压缩映射，也就是散列值的空间通常远小于输入的空间，不同的输入可能会散列成相同的输出，而不可能从散列值来唯一的确定输入值。

简单的说就是一种将任意长度的消息压缩到某一固定长度的消息摘要的函数。

目前普遍采用的哈希算法是time33，又称DJBX33A (Daniel J. Bernstein, Times 33 with Addition)。这个算法被广泛运用于多个软件项目，Apache、Perl和Berkeley DB等。对于字符串而言这是目前所知道的最好的哈希算法，原因在于该算法的速度非常快，而且分类非常好(冲突小，分布均匀)。

PHP内核就采用了time33算法来实现HashTable，来看下time33的定义：

```
hash(i) = hash(i-1) * 33 + str[i]
```

有了定义就容易实现了：

```
<?php
function myHash($str) {
        // hash(i) = hash(i-1) * 33 + str[i]
        $hash = 0;
        $s = md5($str);
        $seed = 5;
        $len = 32;
        for ($i = 0; $i < $len; $i++) {
                // (hash << 5) + hash 相当于 hash * 33
                //$hash = sprintf("%u", $hash * 33) + ord($s{$i});
                //$hash = ($hash * 33 + ord($s{$i})) & 0x7FFFFFFF;
                $hash = ($hash << $seed) + $hash + ord($s{$i});
        }
```

```
        }

        return $hash & 0x7FFFFFFF;
}

echo myHash("却道天凉好个秋~");
```

```
$ php -f test.php
530413806
```

利用取模实现

现在有2台redis server，所以需要计算键的hash并跟2取模。比如有键key1和key2，代码如下：

```
<?php
function myHash($str) {
        // hash(i) = hash(i-1) * 33 + str[i]
        $hash = 0;
        $s = md5($str);
        $seed = 5;
        $len = 32;
        for ($i = 0; $i < $len; $i++) {
                // (hash << 5) + hash 相当于 hash * 33
                //$hash = sprintf("%u", $hash * 33) + ord($s{$i});
                //$hash = ($hash * 33 + ord($s{$i})) & 0x7FFFFFFF;
                $hash = ($hash << $seed) + $hash + ord($s{$i});
        }

        return $hash & 0x7FFFFFFF;
}

echo "key1: " . (myHash("key1") % 2) . "\n";
echo "key2: " . (myHash("key2") % 2) . "\n";
```

```
$ php -f test.php
key1: 0
key2: 0
```

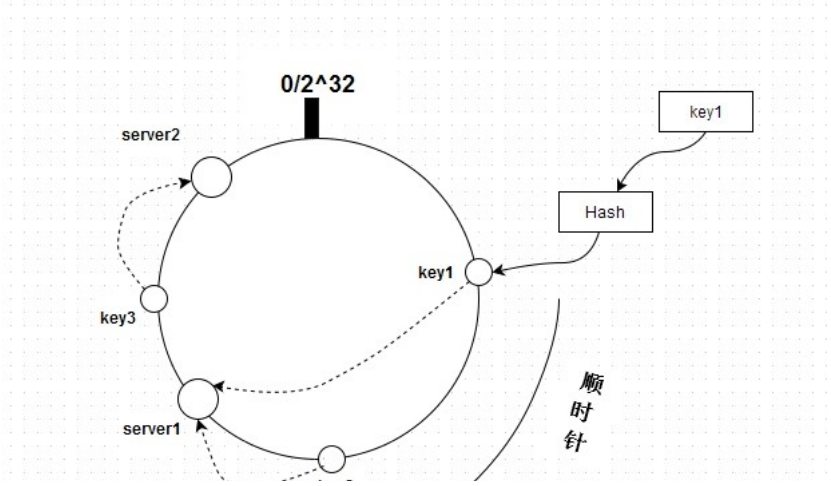
对于key1和key2来说，同时存储到一台服务器上，这似乎没什么问题，但正因为key1和key2是始终存储到这台服务器上，一旦这台服务器下线了，则这台服务器上的数据全部要重新定位到另一台服务器。对于增加服务器也是类似的情况。而且重新hash(之前跟2进行hash，现在是跟3进行hash)之后，结果就变掉了，导致大多数数据需要重新定位到redis server。

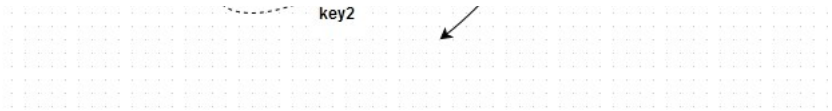
在服务器数量不变的时候，这种方式也是能很好的工作的。

一致性哈希

由于hash算法结果一般为unsigned int型，因此对于hash函数的结果应该均匀分布在[0,2^32-1]区间，如果我们把一个圆环用2^32 个点来进行均匀切割，首先按照hash(key)函数算出服务器(节点)的哈希值， 并将其分布到0～2^32的圆环上。

用同样的hash(key)函数求出需要存储数据的键的哈希值，并映射到圆环上。然后从数据映射到的位置开始顺时针查找，将数据保存到找到的第一个服务器(节点)上。如图所示：





key1、key2、key3和server1、server2通过hash都能在这个圆环上找到自己的位置，并且通过顺时针的方式来将key定位到server。按上图来说，key1和key2存储到server1，而key3存储到server2。如果新增一台server，hash后在key1和key2之间，则只会影响key1(key1将会存储在新增的server上)，其它不变。

上图这个圆环相当于是一个排好序的数组，我们先通过代码来看下key1、key2、key3、server1、server2的hash值，然后再作分析：

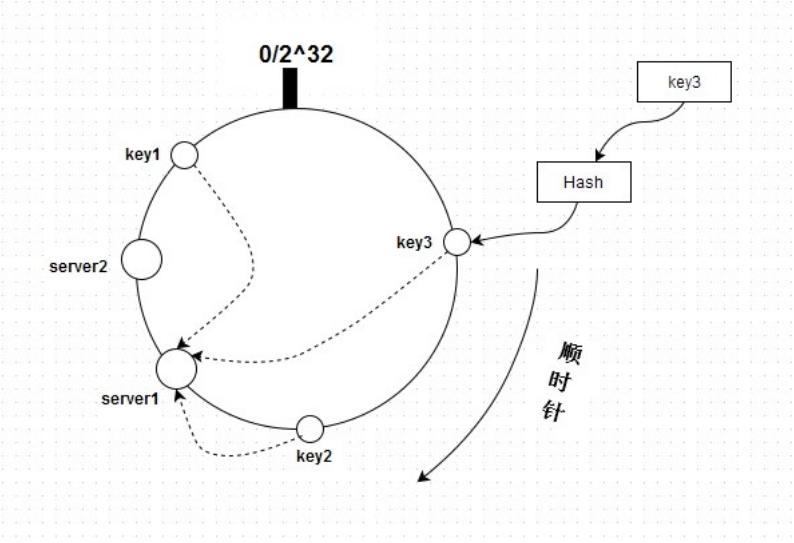
```
<?php
function myHash($str) {
        // hash(i) = hash(i-1) * 33 + str[i]
        $hash = 0;
        $s = md5($str);
        $seed = 5;
        $len = 32;
        for ($i = 0; $i < $len; $i++) {
                // (hash << 5) + hash 相当于 hash * 33
                //$hash = sprintf("%u", $hash * 33) + ord($s{$i});
                //$hash = ($hash * 33 + ord($s{$i})) & 0x7FFFFFFF;
                $hash = ($hash << $seed) + $hash + ord($s{$i});
        }

        return $hash & 0x7FFFFFFF;
}

//echo myHash("却道天凉好个秋~");
echo "key1: " . myHash("key1") . "\n";
echo "key2: " . myHash("key2") . "\n";
echo "key3: " . myHash("key3") . "\n";
echo "serv1: " . myHash("server1") . "\n";
echo "serv2: " . myHash("server2") . "\n";
```

```
$ php -f test.php
key1: 351111878
key2: 1305159920
key3: 1688027782
serv1: 1003059623
serv2: 429427407
```

现在我们根据hash值重新画一张在圆环上的分布图，如下所示：



key1、key2和key3都存储到了server1上，这是正确的，因为是按顺时针来定位。我们想像一下，所有的server其实就是一个排好序的数组(降序)：[server2, server1]，然后通过计算key的hash值来得到处于哪个server上。来分析下定位过程：如果只有一台server，即[server]，则直接定位，取数组的第一个元素。如果有多台server，则要先看通过key计算的hash值是否落在[server2, server1, ...]这个区间上，这个直接跟数组的第一个元素和最后一个元素比较就知道了。然后就可以通过查找来定位了。

利用一致性哈希实现

下面是一个实现一致性哈希的例子，仅仅实现了addServer和find。其实对于remove的实现跟addServer是类似的。代码如下：

```php
<?php
function myHash($str) {
        // hash(i) = hash(i-1) * 33 + str[i]
        $hash = 0;
        $s = md5($str);
        $seed = 5;
        $len = 32;
        for ($i = 0; $i < $len; $i++) {
                // (hash << 5) + hash 相当于 hash * 33
                //$hash = sprintf("%u", $hash * 33) + ord($s{$i});
                //$hash = ($hash * 33 + ord($s{$i})) & 0x7FFFFFFF;
                $hash = ($hash << $seed) + $hash + ord($s{$i});
        }

        return $hash & 0x7FFFFFFF;
}

class ConsistentHash {
        // server列表
        private $_server_list = array();
        // 延迟排序，因为可能会执行多次addServer
        private $_layze_sorted = FALSE;

        public function addServer($server) {
                $hash = myHash($server);
                $this->_layze_sorted = FALSE;

                if (!isset($this->_server_list[$hash])) {
                        $this->_server_list[$hash] = $server;
                }

                return $this;
        }

        public function find($key) {
                // 排序
                if (!$this->_layze_sorted) {
                        asort($this->_server_list);
                        $this->_layze_sorted = TRUE;
                }

                $hash = myHash($key);
                $len = sizeof($this->_server_list);
                if ($len == 0) {
                        return FALSE;
                }

                $keys = array_keys($this->_server_list);
                $values = array_values($this->_server_list);

                // 如果不在区间内，则返回最后一个server
                if ($hash <= $keys[0] || $hash >= $keys[$len - 1]) {
                        return $values[$len - 1];
                }

                foreach ($keys as $key=>$pos) {
                        $next_pos = NULL;
                        if (isset($keys[$key + 1]))
                        {
                                $next_pos = $keys[$key + 1];
                        }

                        if (is_null($next_pos)) {
                                return $values[$key];
                        }

                        // 区间判断
                        if ($hash >= $pos && $hash <= $next_pos) {
                                return $values[$key];
                        }
                }
        }
}

$consisHash = new ConsistentHash();
$consisHash->addServer("serv1")->addServer("serv2")->addServer("server3");
echo "key1 at " . $consisHash->find("key1") . ".\n";
echo "key2 at " . $consisHash->find("key2") . ".\n";
echo "key3 at " . $consisHash->find("key3") . ".\n";
```

```
$ php -f test.php
key1 at server3.
key2 at server3.
key3 at serv2.
```

即使新增或下线服务器，也不会影响全部，只要根据hash顺时针定位就可以了。

结束语

经常有人问在有多台redis server时，新增或删除节点如何通知其它节点。之所以会这么问，是因为不了解redis的部署方式。这些都是依赖一致性哈希来实现分布式的，这种实现都是由各种语言的driver去完成。所以了解一致性哈希算法的原理以及应用场合是很有必要的。

分类：PHP 字数：2131 标签： PHP hash 一致性哈希 哈希


陈亦 关注此人

粉丝: 170 博客数: 23 共码了 53187 字

评论（13）

winerQin

1楼 2014/02/27 18:59

原理很细。受益。

Adam1

2楼 2014/03/21 17:32

新增和删除节点后，原节点上的数据怎么办？

陈亦

3楼 2014/03/21 17:40

> 引用来自“Adam1”的评论
> 新增和删除节点后，原节点上的数据怎么办？

LRU

Adam1

4楼 2014/03/26 12:44

> 引用来自“陈一回”的评论
> > 引用来自“Adam1”的评论
> > 新增和删除节点后，原节点上的数据怎么办？
>
> LRU

我想说的是，假设你新增了节点，那根据一致性hash的算法，原来某个用户对应的节点就会从A转向B，那么就会出现原来的数据取不到的问题，同理，删除一个节点时也是如此，你对此有什么办法？

陈亦

5楼 2014/03/26 12:51

> 引用来自“Adam1”的评论
> > 引用来自“陈一回”的评论
> > > 引用来自“Adam1”的评论
> > > 新增和删除节点后，原节点上的数据怎么办？
> >
> > LRU
>
> 我想说的是，假设你新增了节点，那根据一致性hash的算法，原来某个用户对应的节点就会从A转向B，那么就会出现原来的数据取不到的问题，同理，删除一个节点时也是如此，你对此有什么办法？

这个没有什么办法，但也只是一部分数据会这样。总比简单的哈希要好得多。

AladdinJoin

6楼 2015/04/30 19:56

强哥就是屌

澶飞扬

7楼 2015/08/27 19:39

这里的 asort($this->_server_list);排序应该是以键值倒序排序的吧krsort($this->_server_list);并且服务列表是倒序排列的，楼主是按正序处理的吧？下面的比较就不对了。个人见解。

战线

8楼 2015/09/22 14:04

为什么要MD5啊? 是防止近似值吗

此用户已关机

9楼 2016/04/21 13:32

没看懂啊，实际开发中 server1代表什么？key1又代表什么？

miyae

10楼 2016/06/19 16:42

文章开头说redis不支持分布式？是这样的吗？
不知道博主会看我评论么

陈亦

11楼 2016/06/24 23:25

> 引用来自“miyae”的评论
> 文章开头说redis不支持分布式？是这样的吗？
> 不知道博主会看我评论么

现在redis已经支持主从和集群了

miyae

12楼 2016/07/10 18:54

> 引用来自“miyae”的评论
> 文章开头说redis不支持分布式？是这样的吗？
> 不知道博主会看我评论么
>
> 引用来自“陈亦”的评论
> 现在redis已经支持主从和集群了

谢谢！

miyae

13楼 2016/07/12 00:31

博主你好，阅读了文章，收益，谢谢！
有个疑问：$this->_server_list 这个变量，我猜博主是想得到一个按照hash值顺序排序的数组吧，而asort($this->_server_list);得到的是原server字符串的顺序排序数组，应该是ksort($this->_server_list)吧？

登录后评论