自组织负载均衡算法

# 1、背景

现在主流支持数据分片的数据库（mysql）中间件如TDDL，Mycat等，虽然其功能十分强大，但都有一个通病，就是配置复杂。而且更重要的是，不支持动态加入结点。这对于一些小公司或者小项目，不是十分方便。因为一般的小公司，在项目调研的前期，是不可能准确预测到将来上线后的业务规模，因此，较常见的做法，“看菜吃饭”先部署小规模的集群（通常是1~2台）服务器。当性能达到瓶颈时，才通过加机器的办法，增加系统的吞吐量。

因此，如果有一种算法，除了能达到数据切分，均衡负载的功能外，还能简化配置，同时提供动态加入\退出结点功能。这对于小项目来说，会是一个十分吸引人的选择。

下文将介绍一种自组织算法，可以达到动态加入结点的目的。

在介绍算法之前，先做如下假设：

1. 所有数据的访问频率一致。
2. 数据的访问都是随机的。
3. 不管数据规模达到多大，都希望数据的分布是均匀的
4. 不考虑单点失效的问题

假设1表明，程序访问每个数据的概率P(A) = P(B)。

假设2表明，数据与数据之间没有必然的依赖关系。也就是访问了A数据之后，再访问B数据的概率，与访问除B之外的任一数据的概率相等。也就是P(A|B) = P(A|C)。

假设4可以通过其它例如双机热备份等成熟机制来解决。

# 2、数据水平分片

现今大部分声称支持集群的数据库（不管是sql的或是nosql）都支持数据水平分片(sharding)。数据水平分片的目的就是为了均衡负载，将读写访问压力平均到每台机器上面。这是一种典型的“削峰填谷”的做法。因此，为了达到削峰填谷的目的，都会希望程序对数据的访问都是分散的。最简单的做法是将唯一键值取哈希值，然后再取余。如果哈希值在整个值域里面是均匀分布的，那么取余可以在一定程度上达到分散访问的效果。

举例，某大型企业的CRM系统内部有一个客户表，客户表有亿级的数据，现在希望把这几亿的数据分散到10台服务器的10个库的100个表中，理论上每个表应当只需要承载数万条数据访问压力。这对单台mysql服务器来说，不构成什么性能压力。问题上，如何将这些数据均匀地划分到每台服务器每个库每张表上面。

按照前面提到的简单做法-取余。利用客户表的id值(长整型，唯一)以服务器的个数取余。这样数据就可以均匀散落到0-9号服务器。然后再以mysql实例的个数取余，这样又可以把数据分散在10个实例中，最后再以每个实例中的100 个分表进行取余，把这些数据再分散到100个表中。

上面的方法看起来好像是可行的。但事实上，却隐藏着一个巨大的问题，有一些库的一些表出现空置现象了。举例，2号服务器只有2号库是生效的，其它库都是不可能有数据插入。因为任何一个整数模10只有一个余数。

发现了这个问题后，修正一下刚才的算法。在计算库号的时候，先拿id除10，再拿余下的值取余。这样就可以解决空库问题。

但事实上，还是会有空表的问题。举例，3号库中只有13号表，23号表，。。。。93号表有数据，其它都不可能存在数据。道理很简单，一个数模10余3，也就是说个位数只能是3，因此只有个位数是3的表编号才有可能被命中。

幸好，前面的办法还是可以继续用的，也就是先用id值余10得到服务器号，商值余10得到库号，再拿商值的商值余100得到表号。例如id = 1234 ，可知这个键值的数据落入4号机3号库12号表中。从上面的算法，可以轻易得到结论，算法其实拿数字值的个位数作服务器编号，十位数作库表编号，百千位作表编号。

现在没有空库空表问题了，而且我们平时设计的库表中，很多时候插入数据都是使用自增id的。使用自增id的好处是，可以使上述分片算法的散分效果更好，问题似乎得到了很好的解决。这也是很多人爱用的方案，简单易懂，容易实现。

# 3、痛苦的扩容

事实上，上述方案（或其其它变种）有一个通病。就是数据库服务器很难进行动态扩容。

举个例子，假设某企业突然业务量大增，导致即使进行了这样的分库分表设计。数据库还是挺不住。于是老板决定加机器。这样服务器的数量由10台变成11台。按照前面的数据分片原则，所有的数据不得不重新地划分。举上面的例子，1234本来是放在4号机3号库12号表上的面。现在可能变成要放在在2号机2号库11号表上面，这是因为1234≡2(mod 11 ) , 1234 / 11 = 112 ≡2(mod 10),11≡11(mod 100)。

那到底做了以上改动之后，有多少数据受到牵连？或者换个说法，做了以上改动，有多少数据可以幸免于难。幸运的是，这个数字可以用数学的办法统计。

首先，一个整数值模10与模11同余，是可能存在的。假设一个数n≡a(mod 11,10)通过中国剩余定理可以轻易求得n = 111a + 110t。其中0<=a<=10。也就是说id值等于111,222,333，或者111 + 110 = 221。这些都不需要跨服务器迁移数据。因为这些数据都模11,10同余。而且从通式知道这种数占总数量的大约1 / 110。

但不需要跨服务器迁移数据，不代表不需要跨库迁移数据。例如111/10=11,111 /11 = 10 ,11≡1(mod 10) , 10 ≡0(mod 10)。这说明111虽然逃过了迁服的命运，但是仍少不了迁库的命运。

那么有没有数据可以既不用迁服也不用迁库。有！接下来要计算n’ = n / 10 模10值。已经知道当n=111a + 110t时，数据同服，当服务器数量为10时，n’ = 11a + a / 10 + 11t。而当服务器数量为11时，则n’‘ = 10a + a / 11 + 10t。当n’’≡n’(mod 10)，数据同库。于是有以下等式11a + a / 10 + 11t = 10a + a / 11 + 10t + 10k => 111a + 1100t = 1100k，由于等式右边能被10整除，a必须也只能被10整除（111与10互质）。由于0<=a<=10，所以a的取值只能为0或者10。当a = 0时，t = 10k，也就是当a=0，只有10/1数据符合要求。当a=10，可得到矛盾的不定方程110k-11t = 111。综上所述，要使数据在服务器增加的情况，同服同库，n = 1100t。例如1100 ≡0(mod 11,10)，1100 /10 = 110≡0(mod 10), 1100 / 11 = 100 ≡0(mod 10)。也就是说，总共只有1 / 1100的数据同库同服。

最后，还需要它同表，使用前面的办法，我们可以要求 n’ = n / 100与n’ = n /110模10同余。 由前面推导的通式，容易得到11t = 10t + 10k ，t=10k, n = 11000k。也就是说，在服务器增加的情况下，数据不需要迁移的id值必须为 11000的整数倍。换句话说，只有1/11000的数据不需要进行移动！

在得到具体的数据迁移量后，DBA势必要抓狂。这意味着要把上亿的数据进行一次重新划分。这个过程是痛苦而漫长的。DBA这时要骂娘了，哪个傻X架构师设计出这样的方案？但理论证明，不管案例中的表数T，库数R，服务器数S如何变化,那么在这三者中哪怕只有其中一个发生轻微的变化(例如加1)时，都极有可能造成余数发生剧烈变化。

下面的定理阐明了这一点。

**定理1：n是正整数,a≠b≠c,a,b,c>0， n≡r1(mod a) , [n/a]≡r2(mod b)([n/a]表示比n/a舍弃小数部分的整数)，[[n/a]/b]≡r3(mod c)，若有整数d>0,d与a,b,c均互素，使n≡r1(mod a + d),[n / (a + d)]≡r2(mod b),[[n / (a + d)] / b]≡r3(mod c),当且仅当r1 = 0时成立，且：**

1. **当k1=b,k2=c时，有r1=r2=r3=0，且同时满足所有约束条件的n数目只占原来数目的比例为R = 1/a \* b \* c \* (a +d)**
2. **若k1≠b，则R = 1 / a \* (a + d)**

证明：

首先由题设和中国剩余定理有n = r1 + a \* (a + d)t。

再由[n / (a + d)]≡r2(mod b)可导出r1/a + (a + d)t≡r1/(a+d) = at(mod b)。

由此进一步得到不定方程a \* d \* (a + d) \* t + a \* b \* (a + d) \* k = d \* r1

因为d 与a,b,c均互素，所以d 与a + d均素。因此由不等方程有解的充要条件知 a \* (a + d) | r1。因为r1 < a ，所以r1 = 0。

1)若k1 = b ，因r2 < b，所以r2 = 0。同理因为k2 = c ，r3 < c，所以r3=0。所以n = a \* b \* c \* (a + d) \* t ，所以R = 1 / a \* b \* c \* (a + d)。

2)由r1 = 0，n = a \* (a + d)t， 所以R = 1 / a \* (a + d)

从上面的定理可以清楚看到，这种设计方案哪怕是微小的变动，都会造成大量的数据失效。而且随着a值的增长，R急剧变小。

# 4、能否不动数据？

有没有一个分库分表方案，既简单直观，容易实现。又可以在服务器扩容的时候，不会导致数据出现大量的迁移。从第3节的定理1，我们可以清楚地了解到上述分库分表方案在服务器扩容时候所面临的难题。

事实上，可以直接利用余数来决定服务器号，库号和表号。而且理论证明，只要T,R,S互素，取模就不会出现空库空表的情况。

下面的定理说明了这个问题：

**定理2：n是正整数,a≠b≠c,a,b,c>0， n≡r1(mod a) , n≡r2(mod b)，n≡r3(mod c)，若a,b,c两两互素，则r1,r2,r3可以遍历所有余数的组合。**

证明：这个定理的证明在介绍初等数论的书中都可以看到。但证明这个定理也并不困难，使用反证法可以轻易证之。

反证法：假设存在一个r1，使r2不能取尽模b的余数。

那么必然存在模b的一个余数r’使不定方程 ax + by = r’ – r1不能成立。不定方程无解的充要条件是(a,b)=c, c≠1。这显然与题设矛盾，命题得证。

同理，也可以证明不存在一个r1，使r3不能取尽模a的余数。

既然知道，出现空库空表情况的元凶是T,R,S不互素，就没必要使用上面那个复杂的方案。

举例，以质数1001为T，11为R, 7为S，可以保证不会出现空库空表的情况，而且同样可以达到数据分散的效果。根据中国剩余定理，同一个服务器同一个库同一张表数据的插入周期为R \* T \* S。这跟期望一致。

但即便采用直接取模的方案，可以使问题简化，但数据迁移的问题还是没有得到解决。

这里将引入一个特殊的机制来避免数据迁移的问题。

假设我们现在已经有T个表，R个库，S台服务器，现在想再增加一台服务器。

如果S < MIN(R,T) – 1，那么由欧拉定理可推知，T,R,S+1也必定互素。

如果S > MIN(R,T)，那么当S = n \* MIN(R,T) – 1时，T,R,S + 1不互素。例如7,13都是素数，但14 = 13 + 1 = 2 \* 7。为了让他们互素，可以将s+2，这是因为s+1 与s + 2不可能同时被同一质数整除。下面给出上面结论的数学证明:

**引理1: p是大于1的素数，p | n，则p与 n + 1互素。**

证明：(反证法)假设p可同时被n 和n + 1整除。

则必有n ≡0 (mod p) , n + 1 ≡ 0 (mod p)

这等价于不定方程 n = pt1 ,n + 1 = pt2。令这两式相减得 p(t1 + t2) = 1，由此可知p | 1。这与p > 1矛盾。因此p 不可能同时整除 n 和n + 1。

因为p是素数，所以p与n+1互素。

既然不能被同一个素数整除，那么有没有可能分别被不同的质数整除呢？答案是可能的。例如2 \* 7 =14，3 \* 5 = 15。当出现这样的情况时，可以再令步长+1。例如16就是与7和5互素的。

一般地有比引理1更强力的结论。

**引理2:p是大于1的素数，p |n ，则p 与n + k互素 , k < p。**

证明：(反证法)假设p可同时被n 和n + k整除。

则必有n ≡0 (mod p) , n + k ≡ 0 (mod p)

这等价于不定方程 n = pt1 ,n + k= pt2。令这两式相减得 p(t1 + t2) = k，由此可知p | k。这与k < p 矛盾。因此p 不可能同时整除 n 和n + k。由p是素数，p 不整除n+k，知p与n+k互素。

由引理2可推导出下面结论

**定理2：假设 a, b是素数，a,b,c互素，且a,b,c > 3，则 c + 1,c + 2,c + 3中，必有一个与a，b互素。**

证明：（反证法）设不存在与a,b互素的数。

若a | c + 1，a > 3,则根据引理2，C+2与C + 3必然与a 互素。假设b | c + 2，则根据引理3，必有C + 3与b互素。由此，我们得到C+3与a，b互素。这就与假设矛盾。

若b | c + 3，那么c + 2与b互素，这样 C + 2与a,b互素，与假设矛盾。

若a | c + 2, b | c + 3，那么c由a,b>3可知，C + 1与a,b互素，与假设矛盾。

若a \* b | c + 1，则 C+2 与C +3均与ab互素。

若a \* b | c + 2，则 C+1 与C +3均与ab互素。

若a \* b | c + 3，则 C+1 与C +2均与ab互素。

至此，所有的可能讨论完毕。所有可能都表明至少存在一个数与a,b互素，定理证毕。

上面的结果表明，只要T,R > 3，S最多只需要向前搜索3步，就可以得到一个与T,R互素的新数S’。

但因为增长的步长的可能取值为1,2,3，而事实上只需要增加一台服务器。为此，与很多缓存算法一样，需要引入虚拟结点的概念。举例，现在有5个库，7个表，13个服务器，前面已经得知13 + 1 = 2 \* 7 =14，13 + 2 = 3 \* 5 = 15。但16与5和7均互素。

于是下一个服务器数应为16。但事实上只有14 台服务器。这时，可让14号服务器同时承担14,15,16号机的角色。因为这时14号机是最空的（在不进行数据迁移的情况下），负载最低。

解决了服务器取模问题，接下来就要解决数据迁移的问题。事实上，在不考虑容灾和冗余备份的情况下，我们并不希望进行任何的数据的复制和转移。

有没有一种办法，可以在不迁移数据的情况下，还能让数据的访问能有序进行。为此，这里引入分片周期（sharding age）的概念。现假设数据都是按id值进行自增长的（当然也可以去掉这个假设，只是实现机制会稍复杂），那么从数据插入的时间顺序来说，id值小的应该比id值大的更早插入到数据库中。如果在某个id值之后，发生了服务器数量变化，那么系统将记录当前服务器集群的状态，下一个id值与上一次发生服务器数量变化时的id值。

下面介绍引入这个分片周期概念的好处。回到上面的例子，假设现在要增加一台服务器，每台服务器有5 \* 7 = 35张表。每张表的每行数据的id值都满足

N ≡ r (mod 5)

N ≡ t (mod 7)

N ≡ s (mod 13)

其中r是库号,t是表号,s是服务号。

现在服务器数量发生了变化。那么按照之前的设计。服务器数量变化了之后，每台服务器每个库每张表每行数据的id值应该需要满足:

N ≡ r (mod 5)

N ≡ t (mod 7)

N ≡ s’ (mod 16)

由定理1知道，做了这样的变动后，数据会发生剧烈的变化。很多数据会在不同的服务器之间移来移去。这是我们极不愿意看到。但引入分片周期概念之后，情况就不同了。假设在没增加服务器之前，id 值来到了10000000。也就是说当n < 10000000时，数据的分布还是按13台机的时候来分配的。现在保存这些信息。

这样当id >= 10000000时，为了使新加入的机器能够发生作用。新加入的数据按新规则来进行分配。当Id < 10000000时，则按老规则进行访问。这样，就可以做到新旧规则并存。新老数据互不干扰。

这样，随着服务器数量的增加，我们只需要维护一个分片周期列表，就可以既可以按新规则插入数据，又可以用旧规则访问数据。

# 5、数据分配

以上办法确实可行，但是又会引入一个新的问题：数据倾斜。如前面的例子，1-13号机实际已经承担了以前的所有旧数据。如果有新数据到来的情况下，还要按照公平分配的办法来承担任务，那些旧机器实际上会不堪重负。于是，我们希望新来的机器多扛一些。前面引入虚拟结点的概念，可以继续在这上面做文章。

现在的问题是怎样才算是公平呢？关于这个问题，可能会有很多个不同的答案。这里提供一个可参考方案。

假设目前的数据量为n，服务器数量为s，假设增加服务器Δs（含虚拟节点），假设插入了n’条数据后，原s台服务器中的每一台的机器的数据量都与新加入的服务器数据量相等。即有以下等式:

N / s + n’ \* Po = n’ \* Pn

其中Po,Pn为新，旧服务器在新的规则下插入新数据的概率。令s’ = s + Δs，那么

Po = 1 / s +Δs , Pn =Δs / s + Δs。

代入原式并整理得

N’ = n \* (s + Δs) / s \* (Δs - 1) （7）

我们希望N’尽可能小，起码要比n小，那么会有

S + Δs – s \* (Δs - 1) < 0

移项得: Δs > 2s / (s – 1)

也就是说当Δs > 2s / (s - 1), n’ 会比n 小。

**方案1：**因为Δs 是一个整数且s 与 s – 1互素。

所以当s 是偶数时，可取Δs = (s - 1) \* 2s / (s - 1) = 2s。

当s是奇数时，可取Δs = (s – 1 / 2) \* 2s / (s - 1) = s

把Δs = 2s，Δs = s代入式(7)得到 n’ = 3n / 2s – 1或 n’ = 2n / s – 1

可见n’的大小跟s有关。S越大，n’越小，服务器越容易达到负载平衡。

另外，不管s’=s +Δs =2s还是3s , s’均一定与R,T互素。

问题似乎得到了很好的解决。但事实上，如果进一步深挖还会发现一个不容忽视的问题。就是当服务器达到均衡状态时，继续让新来的机器承担更重的任务，肯定是不合适的，这时候，又需要对新加入的机器进行减负。如果按照s’ = t \* s来设计，当达到均衡状态时，s + 1 必须要整除 s’。否则就会有服务器受到“不公平对待”。

于是又有s’ ≡0(mod s + 1)，但因为s与s+1互素，所以当t与s+ 1互素时，s \* t 与 s + 1互素。Δs取s的整数倍似乎不是一个好选择。

**方案2：**能不能使s’=(s + 1)t。若s’ = (s + 1)t，那么就有Δs=(t-1)s + t，由Δs> 2s / (s – 1)

得t > (s^2 + 2s – 1) / (s^2 – 1)。但是这样的设计方案还是有问题，原因在于s + 1不一定与T,R互素。

上述方案均存在着毛病，方案1不能做到完全的均衡，方案2有可能导致空库空表的情况发生。所以在实际应用的时候，应先判断s+1是否与T,R互素，如果条件成立，就应当选择方案2。如果方案2的判定条件不成立，则只能采用方案1，牺牲一定的公平性。幸运的是当R,T取得比较大时，方案2的判定条件总是能够成立。

例子 R = 11 ,T =101，S=3，能得 S + 1 =4 与R,T互素，令t = 2 > (3^2 + 2 \* 3 - 1) / (3^2 - 1) = 7 / 4，则s’ = 8。假设目前的数据量n，由n’= n \* (s + Δs) / s \* (Δs - 1) = ( 2 / 3) \* n，可算得当数据量达到n + n’< 时，所有机器达到了均衡。

事实上，当集群服务器数量达到的s =n \* R - 1 或者 s = n \* T -1 时 ,如果只增加一台物理机，那么后面实际服务器数量与数据库实例数或表数不互素。因此并不建议采用这样的扩容方案，更好的扩容方案，应该是增加2台，根据定理2，如果S+ 1与R（或T）不互素，则S+2也有很大概率同时与R和T互素。这样既解决了互素的问题，也可以保证在数据量达到一定程度后，服务器之间的负载依然是均等的。

再者，对于一些中小型项目，服务器的加入一般都是一台台的。下面列出了一些在服务器逐台增加的情况下，虚拟结点数目和真实物理机数目之间的变化规律。

1(1) 🡪 2(2) 🡪 6(3) 🡪 24(4) 🡪 120(5) 🡪 720 (6) 🡪 5040(7)……

这里虚拟结点数实际是阶乘函数。虚拟结点数将会膨胀得很快，带来的好处是，服务器很快就可以达到均衡。

至此，关于自动加入的问题都得到了解决。

# 6、自动退出

自动退出的办法与自动进入类似，与自动加入不同的是，自动退出需要迁移数据。现假设服务器也是一台台地进行退出，那么加入时间最短的服务器，应当是优先选择退出的对象。

关于服务器集群的状态信息，包括当前有多少台机器（或集群），机器的ip地址列表，所负责的模数等。因为服务器的信息经常变动，为此，这里为服务器引入虚拟身份的概念。这样即便实际提供服务的物理机发生了变化，整个系统还可以正常跑起来。引入了虚拟身份概念后，集群信息变成了保存这些虚拟身份信息。