# 分布式键值系统

分布式键值模型可以看成是分布式表格模型的一种特例，由于它只支持针对单个key-value的增删改查操作，因此使用哈希分布算法。

Amazon Dynamo是分布式键值系统，最初用于购物车应用。Dynamo将很多分布式技术融合到一个系统里，它牺牲了一致性但是却没有换来什么好处，适合学习理解分布式系统的理论，不适合在工程中模仿。

Tair是Taobao借鉴Dynamo开发的键值系统，最大的变化是从P2P架构改为了带有中心节点的架构。

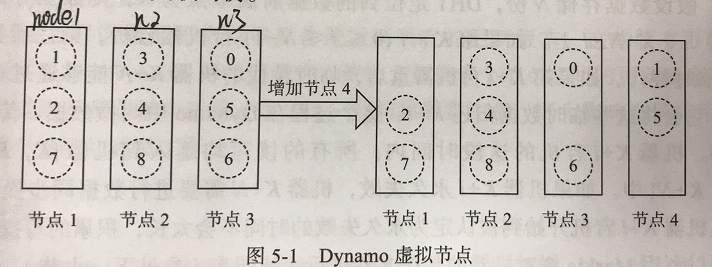
## Amazon Dynamo

Dynamo通过组合P2P的各种技术打造了线上可运行的分布式键值系统。

|  |  |
| --- | --- |
| **问题** | **采取的技术** |
| 数据分布 | 改进的一致性哈希（增加虚拟节点），因为考虑系统的异构性，机器性能不一样，好的机器多干点活 |
| 复制协议 | 复制写协议（Replicated-write protocol） |
| 数据冲突处理 | 向量时钟 |
| 临时故障处理 | 数据回传机制 |
| 永久故障后的恢复 | Merkle哈希树 |
| 成员资格及错误检测 | 基于Gossip的成员资格和错误检测协议 |

### 数据分布

一致性哈希的优点在于节点加入/删除时只会影响到在哈希环中相邻的节点，而对其他节点没影响。考虑到节点的异构性，不同节点的处理能力可能差别很大，Dynamo使用了改进的一致性哈希算法：每个物理节点根据其性能分配多个token，每个token对应一个“虚拟节点”。每个虚拟节点的处理能力基本相当，并随机分布在哈希空间中。存储时，数据按照哈希值落到某个虚拟节点负责的区域，然后被存储在该虚拟节点所对应的物理节点中。



如上图，Dynamo集群原来有3个节点，每个节点分配了3个token。在存放数据时，首先计算主键的哈希值，根据哈希值将数据存放到对应token所在的节点。假设增加了一个节点4，Dynamo集群可能会把token1和token5迁移到节点4，这样实现了自动负载均衡。

Dynamo系统中每一个节点都维护整个集群的信息，客户端也缓存整个集群的信息，所以绝大部分请求能够一次定位到目标节点。（这是一种P2P的方式，这种方式的问题在于有些节点对集群的认识是过时的或者错误的）

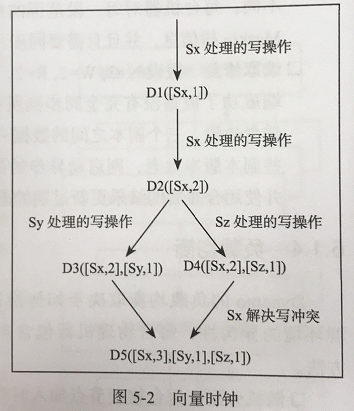
系统中经常出现节点加入和删除，为保证每个节点缓存的都是Dynamo集群中最新的成员信息，所有节点每隔固定的时间通过Gossip协议的方式从其他节点中选择任意的一个与之通信。如果连接成功，双方交换各自保存的集群信息。

### 一致性与复制

为处理节点失效的情况，需要对节点的数据进行复制。思路如下：假设数据存储N份，DHT(Distributed Hash Table，一致哈希表)定位到数据属于节点k，则数据存储在节点k,k+1,…,k+N-1上。如果k+i台机器宕机了，需要向后找k+N作为临时替代。如果k+i节点重启了，临时替代的机器k+N可以通过Gossip协议发现，它将会把这些临时数据归还给k+i。这个过程就叫做**数据回传**。在k+i宕机的这段时间，所有的读写均落在节点[k,k+i-1],[k+i+1,k+N]上。如果k+i永久失效了，这时k+N需要进行数据同步操作(是把k+i之前保存的数据同步到节点k+N上)。一般来说，从机器宕机到被认定为永久失效的时间不会太长，积累的写操作也不会太多，利用Merkle树对机器的数据文件进行快速同步。

NWR是Dynamo中的一个亮点。N表示复制的备份树，R指成功读取操作的最少节点树，W指成功写入操作的最少节点数。只要满足W+R>N，就可以保证当存在不超过一台机器故障的时候，至少能够读到一份有效的数据。

在Dynamo这样P2P的集群中，由于每个节点存储的集群信息有所不同，可能出现同一条记录被多个节点同时更新的情况，无法保证多个节点之间的更新顺序。为此，Dynamo引入了向量时钟（Vector Clock）的技术手段来尝试解决冲突。



在D2版本之后，出现了两个写操作，这是系统中有两个冲突版本。常见的冲突解决办法有两种：一种是通过客户端逻辑来解决；另外一种是“last write win”，即选择时间戳最新的副本，然而，这个策略依赖集群内节点之间的时钟同步算法，不能完全保证准确性，Dynamo只能保证每个读取操作能读到所有的更新版本，这些版本可能冲突，需要进行版本合并。Dynamo只保证最终一致性，如果多个节点之间更新顺序不一致，客户端可能读取不到期望的结果。

### 容错

Dynamo把异常分为两种类型：临时性的异常（机器假死）和永久性异常（硬盘故障或者机器报废）。Dynamo的容错机制分为两种：

**数据回传：**在Dynamo的设计中，一份数据会被写入到k，k+1…,k+n-1这n台机器上，假设k+i出现故障，本该写入这台机器的数据将被写入到k+n上，如果在指定的时间内这台机器恢复了重新提供服务，那么机器k+n将通过Gossip协议发现，并将启动传输任务将暂存的数据回传给机器k+i。

**Merkle树同步：**如果超过了时间机器k+i依然没有重新提供服务，那么这种是永久性的异常，需要借助Merkle树机制从其他副本进行数据同步。

**读取修复：**客户端在读取操作如果发现了某些副本版本太老，则启动异步的读取修复任务。该任务会合并多个副本的数据，并使合并后的结果更新过期的副本，从而使得副本之间保持一致。

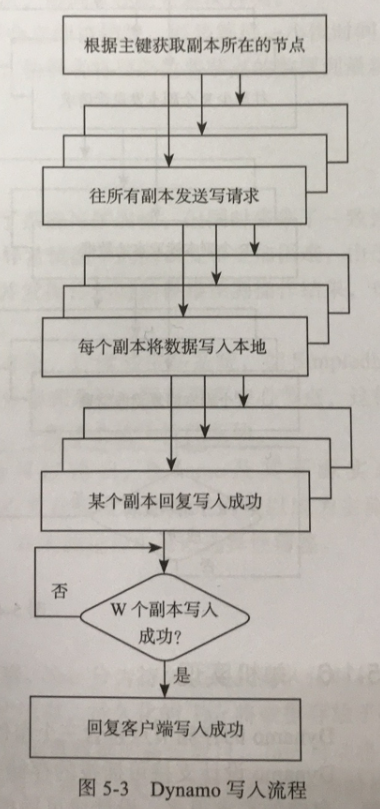
### 负载均衡

Dynamo的负载均衡取决于如何给每台机器分配虚拟节点号，即token。由于集群环境的异构，每台物理机器包含多个虚拟节点。分配token一般有两种方式：

**随机分配：**每台物理节点根据其配置情况随机分配一定个数的token。这种方法的负载均衡效果还是不错的，但是问题是可控性比较差，节点加入或者离开系统时，集群中原有的节点都需要扫描所有的数据从而找出属于新节点的数据，Merkle树也需要全部更新。

**数据范围等分+随机分配：**为了解决随机分配出现的问题，首先将数据的哈希空间等分为Q=N\*S份（N是机器的数量，S是每台机器的虚拟节点个数），然后每台机器随机选择S个分割点作为token。这种方法的负载也比较均衡，并且每台机器都可以对属于每个范围的数据维护一颗逻辑上的Merkle树，节点加入或者离开时只需要扫描部分数据进行同步，并更新这部分数据对应的逻辑Merkle树。

### 读写流程

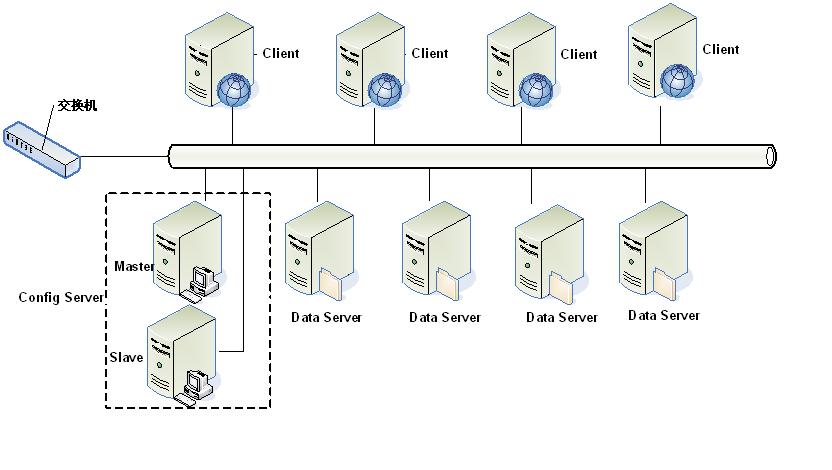


Dynamo写入数据时，首先根据一致性哈希算法计算出每个数据副本所在的存储节点，**其中一个副本作为本次写操作的协调者**。接着，协调者并发的往所有其他副本发送写请求，每个副本将接收到数据写入本地，协调者也将数据写入本地。当某个副本写入成功后，回复协调者。如果发送给某个副本的写请求失败，协调者会将它加入重试列表不断重试。等到w-1个副本（加上协调者一种w个副本，w在Dynamo里面的意思是写操作成功的最少节点数）回复写入成功后，协调者就回复客户端写入成功。协调者回复客户端成功后还会继续等待或者重试，直到所有的副本都写入成功。

Dynamo读取数据时，首先根据一致性哈希算法计算出每个副本所在的存储节点，其中一个副本作为本次都操作的协调者。接着，协调者根据负载策略选择R个副本，并发的向它们发送都请求。每个副本读取本地数据，协调者也读取本地数据。当每个副本读取成功后，回复协调者读取的结果。等待R-1个副本回复读取成功后（加上协调者一共R个），协调者可以回复客户端。这里分为两种情况：如果R个副本返回的数据完全一致，将某个副本的数据读取结果回复给客户端；否者，需要根据冲突处理规则合并多个副本的读取结果。Dynamo的默认策略是根据修改的时间戳选择最新的数据，用户也可以自定义冲突处理方法。读取的过程中如果发现某些副本上的数据版本太旧，Dynamo内部会异步发起一次读取修复操作，使用冲突解决后的结果修正错误的副本。

Dynamo采用无中心节点的P2P设计，增加了系统的可扩展性，但是同时带来一致性问题，影响上层应用。主流的分布式系统一般都会带有中心节点，这样能够简化设计，而且中心节点只维护少量元数据，一般不会成为系统性能的瓶颈。

## Taobao Tair



Tair是由一个中心控制节点（config server）和若干个服务节点（Data server）组成的。

* config server负责管理所有的data server，维护data server的状态信息，config server目前采用一主一备的形式来保证可靠性；
* data server对外提供各种数据服务，并以心跳的形式将自身状况汇报给config server。所有的data server地位都是相同的。

### 基本概念

configID，唯一标识一个tair集群，每个集群有一个对应的configID，在当前的大部分应用情况下configID是存放在diamond中的，对应了该集群的config server地址和groupname。业务在初始化tair client的时候需要配置此configID。

namespace，又称area， 是tair中分配给应用的一个内存或者持久化存储区域， 可以认为应用的数据存在自己的namespace中。 同一集群（同一个configID）中namespace是唯一的。**通过引入namespace，我们可以支持不同的应用在同集群中使用相同的key来存放数据，也就是key相同，但内容不会冲突**。一个namespace下是如果存放相同的key，那么内容会受到影响，在简单K/V形式下会被覆盖，rdb等带有数据结构的存储引擎内容会根据不同的接口发生不同的变化。

quota配额，对应了每个namespace储存区的大小限制，超过配额后数据将面临最近最少使用（LRU）的淘汰。持久化引擎（ldb）本身没有配额，ldb由于自带了mdb cache，所以也可以设置cache的配额。超过配额后，在内置的mdb内部进行淘汰。

expireTime，数据的过期时间。当超过过期时间之后，数据将对应用不可见，不同的存储引擎有不同的策略清理掉过期的数据。

### 存储引擎

tair分为持久化和非持久化两种使用方式：

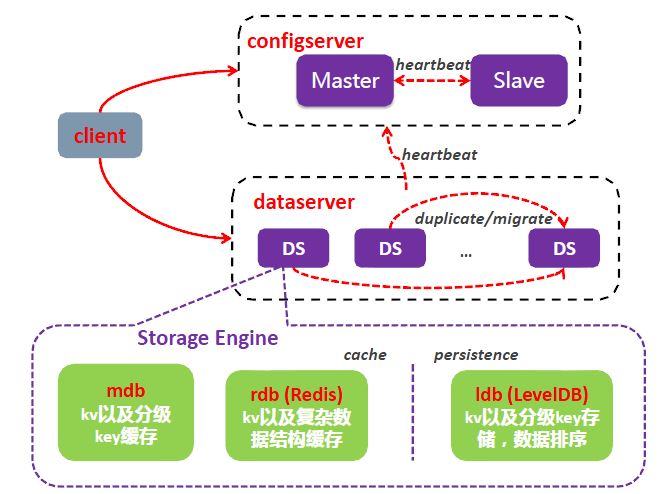
非持久化的tair可以看成是一个分布式缓存；

持久化的 tair 将数据存放于磁盘中，为了解决磁盘损坏导致数据丢失，tair 可以配置数据的备份数目。tair 自动将一份数据的不同备份放到不同的主机上，当有主机发生异常，无法正常提供服务的时候，其余的备份会继续提供服务。

tair的存储引擎有一个抽象层，只要满足存储引擎需要的接口，便可以很方便的替换Tair底层的存储引擎。比如你可以很方便的将bdb、tc、redis、leveldb甚至MySQL作为Tair的存储引擎，而同时使用Tair的分布方式、同步等特性。

tair主要有下面三种存储引擎：

* mdb，定位于cache缓存，类似于memcache。支持k/v存取和prefix操作；
* rdb，定位于cache缓存，采用了redis的内存存储结构。支持k/v，list，hash，set，sortedset等数据结构；
* ldb，定位于高性能存储，采用了levelDB作为引擎，并可选择内嵌mdb cache加速，这种情况下cache与持久化存储的数据一致性由tair进行维护。支持k/v，prefix等数据结构。今后将支持list，hash，set，sortedset等redis支持的数据结构。



### 关键问题

#### 分布式策略

Tair的分布采用的是一致性哈希算法，根据数据的主键计算哈希值后，分布到Q个桶中，桶是负载均衡和数据迁移的基本单位。Config server按照一定的策略把每个桶指派到不同的Data server上，因为数据按照主键哈希值，所以可以认为每个桶中的数据基本是平衡的，只要保证桶分布的均衡性，就能够保证数据分布的均衡性。Q的取值要远大于集群物理机器的数量。具体说，首先计算Hash(key)，得到key所对应的bucket，然后再去config server查找该bucket对应的data server，再与相应的data server进行通信。也就是说，config server维护了一张由bucket映射到data server的路由表。

|  |  |
| --- | --- |
| bucket | data server |
| 0 | 192.168.10.1 |
| 1 | 192.168.10.2 |
| 2 | 192.168.10.1 |
| 3 | 192.168.10.2 |
| 4 | 192.168.10.1 |
| 5 | 192.168.10.2 |

这里有6个bucket，有两个data server负责维护。客户端Hash(key)之后mod 6得到bucket的编号，并向config server请求bucket对应的data server，然后客户端直接和data server通信。表的行数通常远大于集群节点数，这个和一致性哈希中的虚拟节点很相似。

如果有新的机器假如到集群，tair会自动调整路由表，会将部分bucket交给新的机器负责。

|  |  |
| --- | --- |
| bucket | data server |
| 0 | 192.168.10.1 |
| 1 | 192.168.10.2 |
| 2 | 192.168.10.1 |
| 3 | 192.168.10.2 |
| 4 | 192.168.10.3 |
| 5 | 192.168.10.3 |

如果每个桶有多个备份，那么路由表就是有多个列存放data server的地址。

为了更进一步的提高数据的安全性，Tair的config server在build路由表的时候，可以配置考虑机房和机架信息。比如你配置备份数为3，集群的节点分布在两个不同的机房A和B，则Tair会确保每个机房至少有一份数据。当A机房包含两份数据时，Tair会确保这两份数据会分布在不同机架的节点上。这可以防止整个机房发生事故和某个机架发生故障的情况。这里提到的特性需要节点物理分布的支持，当前是通过可配置的IP掩码来区别不同机房和机架的节点。

Tair 提供了两种生成路由表的策略：

* 负载均衡优先，config server会尽量的把桶均匀的分布到各个data server上，所谓尽量是指在不违背下面的原则的条件下尽量负载均衡：每个桶必须有COPY\_COUNT份数据； 一个桶的各份数据不能在同一台主机上；
* 位置安全优先，一般我们通过控制 \_pos\_mask（Tair的一个配置项） 来使得不同的机房具有不同的位置信息，一个桶的各份数据不能都位于相同的一个位置（不在同一个机房）。

位置优先策略有一个问题，假如只有两个机房，机房1中有100台data server，机房2中只有1台data server。这个时候，机房2中data server的压力必然会非常大，于是这里产生了一个控制参数 \_build\_diff\_ratio（参见安装部署文档），当机房差异比率大于这个配置值时，config server也不再build新表，机房差异比率是如何计出来的呢？首先找到机器最多的机房，不妨设使RA，data server数量是SA，那么其余的data server的数量记做SB，则机房差异比率=|SA – SB|/SA，因为一般我们线上系统配置的COPY\_COUNT=3，在这个情况下，不妨设只有两个机房RA和RB，那么两个机房什么样的data server数量是均衡的范围呢? 当差异比率小于 0.5的时候是可以做到各台data server负载都完全均衡的。这里有一点要注意，假设RA机房有机器6台，RB有机器3台，那么差异比率 = 6 – 3 / 6 = 0.5，这个时候如果进行扩容，在机房A增加一台data server，扩容后的差异比率 = 7 – 3 / 7 = 0.57，也就是说，只在机器数多的机房增加data server会扩大差异比率。如果我们的\_build\_diff\_ratio配置值是0.5，那么进行这种扩容后，config server会拒绝再继续build新表。

#### 一致性和可靠性

分布式系统中的可靠性和一致性是无法同时保证的，因为我们必须允许网络错误的发生。Tair 采用复制技术来提高可靠性，并且为了提高效率做了一些优化。事实上在没有错误发生的时候，Tair提供的是一种强一致性，但是在有data server发生故障的时候，客户有可能在一定时间窗口内读不到最新的数据，甚至发生最新数据丢失的情况。

#### version

Tair中的每个数据都包含版本号，版本号在每次更新后都会递增。这个特性可以帮助防止数据的并发更新导致的问题。

get接口返回的是DataEntry对象，该对象中包含get到的数据的版本号，可以通过getVersion()接口获得该版本号。在put时，将该版本号作为put的参数即可。 如果不考虑版本问题，则可设置version参数为0，系统将强行覆盖数据，即使版本不一致。

很多情况下，更新数据是先get，然后修改get回来的数据，再put回系统。如果有多个客户端get到同一份数据，都对其修改并保存，那么先保存的修改就会被后到达的修改覆盖，从而导致数据一致性问题,在大部分情况下应用能够接受，但在少量特殊情况下，这个是我们不希望发生的。

比如系统中有一个值“1”, 现在A和B客户端同时都取到了这个值。之后A和B客户端都想改动这个值，假设A要改成12，B要改成13，如果不加控制的话，无论A和B谁先更新成功，它的更新都会被后到的更新覆盖。Tair引入的version机制避免了这样的问题。刚刚的例子中，假设A和B同时取到数据，当时版本号是10，A先更新，更新成功后，值为12，版本为11。当B更新的时候，由于其基于的版本号是10，此时服务器会拒绝更新，返回version error，从而避免A的更新被覆盖。B可以选择get新版本的value，然后在其基础上修改，也可以选择强行更新。

**version的改变逻辑：**

* 如果put新数据且没有设置版本号，会自动将版本设置成1；
* 如果put是更新老数据且没有版本号，或者put传来的参数版本与当前版本一致，版本号自增1；
* 如果put是更新老数据且传来的参数版本与当前版本不一致，更新失败，返回VersionError；
* put时传入的version参数为0，则强制更新成功，版本号自增1。

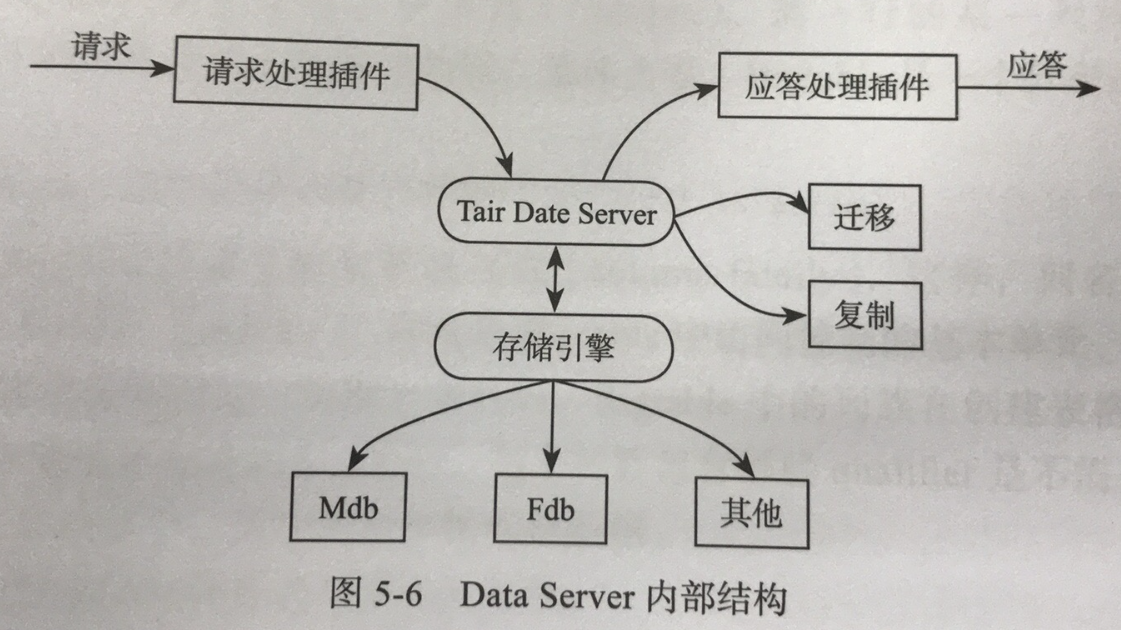
#### config server

client和config server的交互是为了获取数据分布的路由表，client第一次获得路由表后会缓存路由表，大多数情况下，客户端不需要访问config server，即使config server故障了也不影响客户端正常访问。这使得tair对外的服务不依赖config server，所以它不是传统意义上的中心节点，也不会成为系统的瓶颈。

config server维护的路由表有一个版本号，每次新生成表，该版本号都会增加。当有data server状态发生变化（新增节点或者节点不可用）时，config server会根据当前可用的节点重新生成路由表，并通过心跳把新表同步给data server。在client访问data server的时候，会发送client缓存的路由表的版本号，如果data server发现client的版本号过旧，则会通知client去config server取一次新的路由表。如果client访问某台data server 发生了不可达的情况(该 data server可能宕机了)，客户端会主动去config server取新的路由表。

这使得在正常的情况下，client不需要和config server通信，即使config server不可用了，也不会对整个集群的服务造成大的影响。有了config server，client不需要配置data server列表，也不需要处理节点的的状态变化，这使得Tair对最终用户来说使用和配置都很简单。

#### data server



data server具备抽象的存储引擎层，可以方便添加新的存储引擎。

#### 容错

当某台data server故障不可用时，config server会检测到，config server负责重新计算一张新的桶在data server上的分布表，将原来由故障机器服务的桶的访问重新指派到其它有备份的data server中。每个哈希桶在tair中存储多个备份，如果是备副本，那么config server会重新为其指定一台data server。如果是主副本，那么config server首先将某个正常的备副本提升为主副本，对外提供服务。接着选择另外一台data server增加一个备副本。

#### 扩容

当系统增加data server的时候，config server根据负载，协调data server将他们控制的部分桶迁移到新的data server上，**迁移完成后调整路由**。

不管是发生故障还是扩容，每次路由的变更，config server都会将新的配置信息推给data server。在client访问data server的时候，会发送client缓存的路由表的版本号，如果data server发现client的版本号过旧，则会通知client去config server取一次新的路由表。如果client访问某台data server 发生了不可达的情况(该 data server可能宕机了)，客户端会主动去config server取新的路由表。

#### 迁移

当发生迁移的时候，假设data server A 要把桶3，4，5 迁移给data server B。因为迁移完成前，client的路由表没有变化，因此对3, 4, 5 的访问请求都会路由到A。现在假设 3还没迁移，4正在迁移中，5已经迁移完成，那么：

* 如果是对3的访问，则没什么特别，跟以前一样；
* 如果是对5的访问，则A会把该请求转发给B，并且将B的返回结果返回给client；
* 如果是对4的访问，在A处理，同时如果是对4的修改操作，会记录修改log，桶4迁移完成的时候，还要把log发送到B，在B上应用这些log，最终A B上对于桶4来说，数据完全一致才是真正的迁移完成。