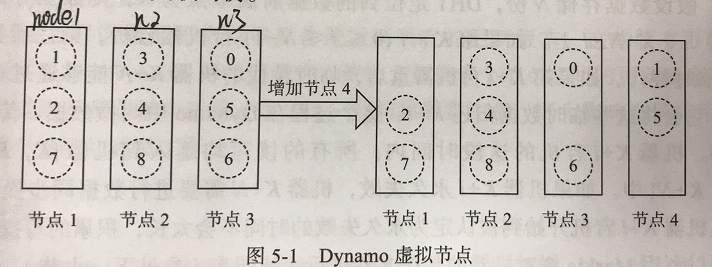
# Amazon Dynamo

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）

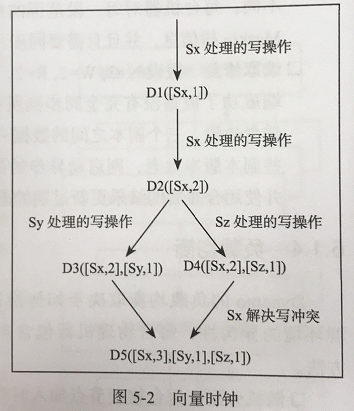
系统中经常出现节点加入和删除，为保证每个节点缓存的都是Dynamo集群中最新的成员信息，所有节点每隔固定的时间通过Gossip协议的方式从其他节点中选择任意的一个与之通信。如果连接成功，双方交换各自保存的集群信息。

## 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。

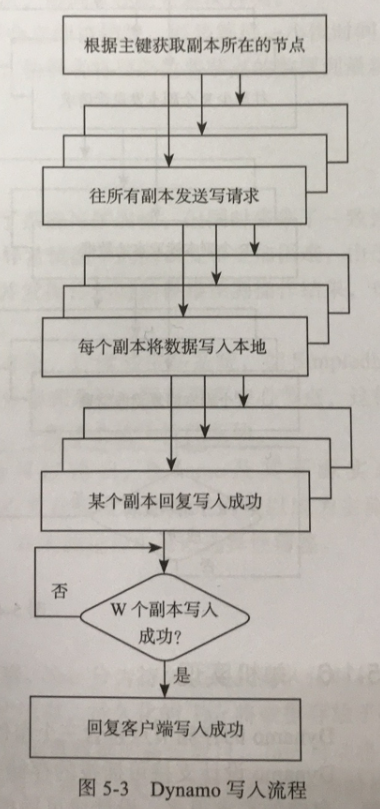
## 负载均衡

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。