~~>改存储为key buf加写入版本；新加搜索任务化~~

~~>k桶的map化~~

>数据的改写；

采用节点自动执行控制节点监督的模式，当节点执行失败或停止的情况下，控制节点可以重启流程；

这里要将原来广播写入改为按顺序写入；

串行广播最大的问题就是速度较慢；

如果某个节点不可到达那么将严重影响性能；

>数据的查询查流程连续查询多个节点；

发送给第一个节点，第一个节点转发给临近的节点；

依次返回给原节点；

>节点的增加，如果是头节点要删除后一个节点的头节点标志。其他节点忽略

>节点的删除，检测到节点删除要通知头节点发起数据的延后。注意延后的数据以最后的节点为准而不是头节点，防止较旧的数据覆盖新数据，这种触发是一个数据同步指令，而不是数据改写指令。

>添加临近节点搜索的优化；新加在原来的基础上包装接口---目前有暂时先不做

K桶的搜索算法要改

>引入gooisp广播解决DHT网络树形化分区化的问题。

>cmake的安装脚本;

>引入leveldb;

>storage的map化

>新节点的数据同步以后一个节点为依据，

只有数据为空白的情况下才能写入；

后节点有的ap数据要同步；

后节点的是首节点数据要比较是否上节点是否有所有权，如果有转移首节点所有权。

>通过控制命令访问的双向通道----线程分离延后

在DHT内确定C与AP转换问题。

如果某个节点按各自需求确定一个C

数据写入AP的过程中是否可以确定唯一性；

这个问题可以归为写入流程。

分布式系统不稳定率，

新加入或离开的服务器之间的平均时间间隔

与每台服务器数据同步的时间比。

得出集群是否处于崩溃状态。

暂时先不考虑这个问题

cp本身就有事务方面的需求。

例如数据ＡＢ需要事务，ＢＣ需要事务，

需要使用流程的概念来保证CP的事务。

增改查和服务器加入，及同步数据的流程。

增改，控制服务器找到离目标最近的Ｃ

尝试在当前Ｃ写入数据，并获得计数器返回和临近节点。

并尝试写入临近节点，如果临近节点返回计数高于Ｃ返回计数

就失败并重新开始在Ｃ写入。

这里使用控制模式可以加快写入的速度。

查，控制服务器找到离目标最近的Ｃ

获得Ｃ的返回结果和写入计数器和临近节点。

尝试从临近节点索要数据并比较计数器。

连续获得几个计数器相同的结果表示查询结束。

服务器加入后要尝试获得临近节点，

并从左右临近节点复制数据。

针对某个ＫＥＹ要重新规范ＫＥＹ的位置。

服务器离开，定时检查临近节点是否发生了改变。

根据每个数据ＡＰ位子判断是否要同步数据。

在服务器添加或离开都会导致数据的ap排序发生变化。

假设在１前面添加一个新节点同步数据后后面的排序号是否都要改变。

如果在1后面添加一个那么前面那个是否要删除。

这样可能会导致大量数据的同步工作。

如果没有排序号那么可能导致的原因是不知道是否要同步数据。

同步多少数据。极大的可能就是重复同步数据。

因为不知道ap的末尾在哪里。

临近节点的搜索算法可以简化为在一堆数据中的查找算法。

在一个不平衡树中的查找算法。

目前没有很好的和确定的临近节点的查找算法只能用goosip广播的方式。

新节点发布之前应充分同步数据之后。

这样当写入数据时版本号才会正确。

如果只是C节点是空白的那么传递到第二个A节点

就会出现后面版本大于等于前面版本的错误情况。

但是如果前面几个节点都是空白的到第N个节点才发现有更高版本的数据。

那么这个错误将无法会滚导致写入数据严重错误。

这种情况是在连续出现多个空白服务器作为C的情况。

当新服务器出现时必然要有一个同步数据的过程。

这个过程中的数据的更改都会导致数据错误。

没有数据和数据较旧的问题。。。

因为没有明显的创建动作，

新机器会有明显的同步的问题。

增改的操作要明显的分开。

写入流程改为二阶段流程。

写入流程改为依次写入N各服务器。

再写入第N个服务器后。

发起广播给前面的N个服务器确认写入。

写入版本以最后广播的为准。

取消版本号，以写入为准。

如果左侧判断节点不可到达，临近的右侧也判断节点不可到达。

那么就会判定节点消失，对这个线段上临近左右共8个节点发送广播。

促使消息的尾端节点开始延长。但每个节点并不知道自己是不是末端节点。

这里要标记末端节点。如果刚刚好消失的是末端节点。哪第4个节点就要判断

后面自己是不是第4个节点。这样就要把非末端节点的所有数据与下一个节点比较

看下一个节点是否已经存在。这不可接受………

只能给第1个节点发送数据，由第一个节点触发产生末端节点。

所以当检测到一个节点短线之后，应该给前面头节点发送广播，头节点删除断线的节点。

给第4个节点发送延长数据。第4个节点把数据向第5个节点同步。