~~>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；~~

~~发起节点在搜索完成后，发起改写操作。~~

~~发送announce peer给目标节点；~~

~~设置监督节点ip,设置序列数为0；~~

~~目标节点收到后写入数据，更改序列加一；~~

~~给监督节点和发送节点返回设置成功；~~

~~发送节点没有相关任务忽略返回；~~

~~监督节点记录数据；~~

~~错误处理延后；~~

~~>数据的查询查流程连续查询多个节点；~~

~~发送给共识的c节点直接返回结果。~~

~~>storage的map化~~

~~>引入gooisp广播解决DHT网络树形化分区化的问题。~~

~~>ping临近节点大于等于3的判定节点删除~~

>节点的增加，

在什么时候通过gossip发送全局广播通知nodeup新节点添加。

临近节点收到广播后，给受到影响的key发送syn同步数据。

临近节点要记录已经同步的数据,这里有问题效率太低。

枚举所有key要消耗时间

对于新添加的cp节点，要发送 sync将符合规定的key进行同步 。

>节点的删除，

发送全局nodedown节点删除的广播。

触发所有相关头节点发送syn同步数据指令。

~~>cmake的安装脚本;以及跨平台~~

>引入sqlite;

>通过控制命令访问的双向通道引入json----线程分离延后

>debug信息封包化。

>脚本化写入请求和查询请求

分布式第一解决的是稳定性的系统

分布式的稳定性只与修复能力成正比。

系统的稳定性为每个模块稳定性的乘积，其中x为每个模块稳定概率n为系统模块的个数。如果每个模块的稳定性都相同那么可以把系统整体稳定性简化为，如果每个模块分为两个主备方式那么其整体稳定性为。

网络传输时间统一为t,接收方处理时间为r,发送方处理时间为s,一次请求所要处理数据量为n，发送消息的时间为ss

在单一系统下所需要的总时间si=>ss+nr+t+s=>；

假设集群数量与所需要处理数据量相同都是n,所需要处理数据量被均匀分布在集群内。

那么一次请求所需要的总时间d=>ss+r+t+sn=>;

上述公式可以看出si/d的比值与集群数量n无关只与r/s有关。所需要处理的总时间并没有因此减少。 那些因素导致分布式会拖慢每次传输的速度。