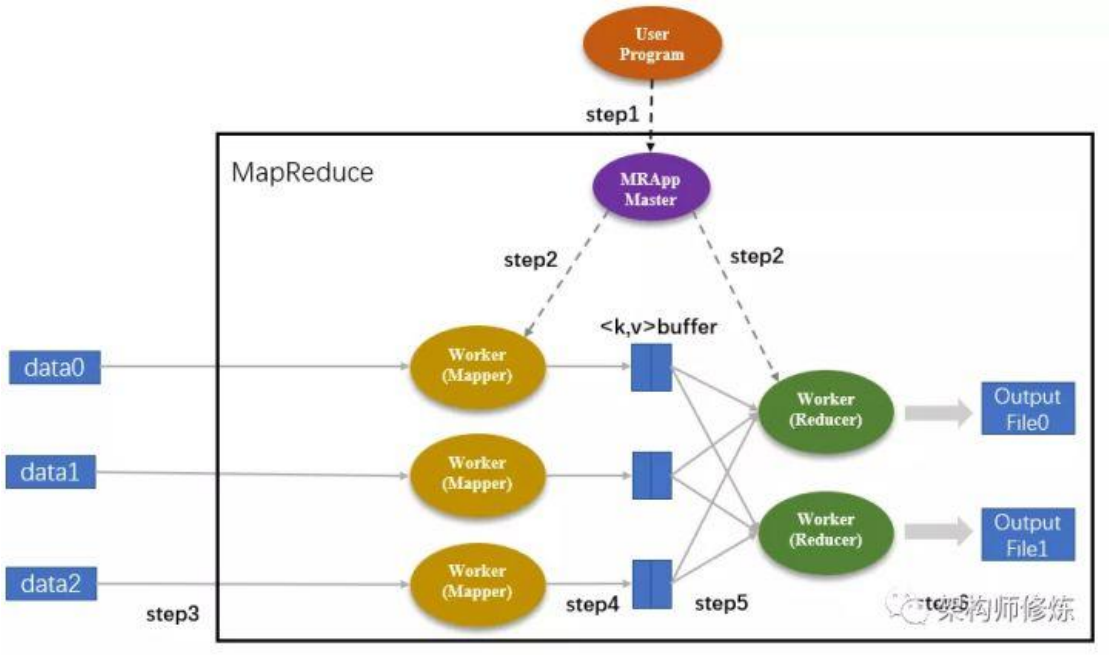
分布式计算

1. 分布式计算架构

-MapReduce

https://blog.csdn.net/Crazy_Clown/article/details/91978239



input

MapReduce 需要把要执行的大文件数据进行切割（split）

每一个输入分片（input split）对应一个 map 任务

输入分片（input split）存储的并非数据本身而是一个分片长度和一个记录数据位置的数组

输入分片（input split）和 HDFS（hadoop2.0 之后）的 block（块）关系很密切，HDFS（hadoop2.0 之后）的 block 块的大小默认是 128M，如果我们执行的大文件是 128x10M，MapReduce 会分为 10 个 map 任务，每个 map 任务都存在于它所要计算的 block(块)的 DataNode 上

map

程序员编写的 map 函数

因此 map 函数效率相对好控制

而且一般 map 操作都是本地化操作也就是在数据存储节点上进行

shuffle

负责将 map 生成的数据传递给 reduce

因此 shuffle 分为在 map 的执行过程和在 reduce 的执行过程

reduce

负责 shuffle 传递过来的数据进行合并

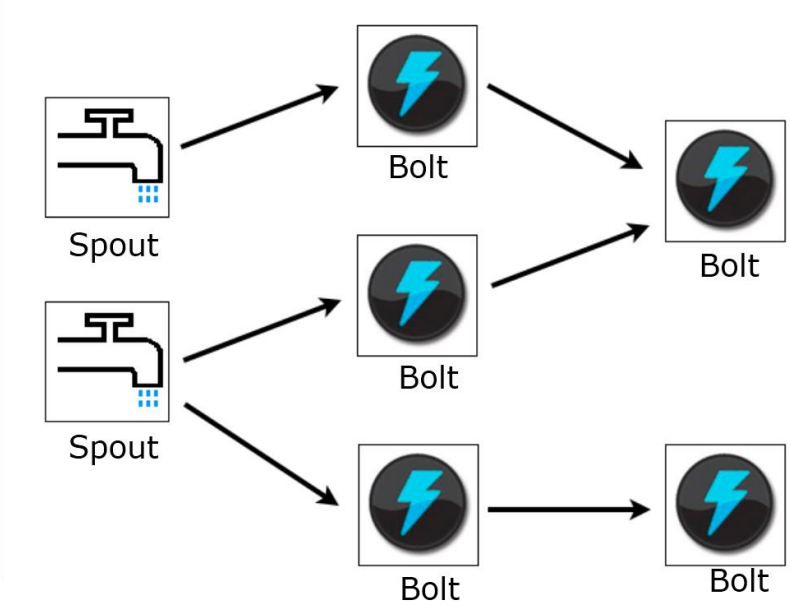
output

最后负责输出 reduce 中的数据信息

特点：可扩展性、可处理 PB 级别的数据，但为离线计算系统，从磁盘中读入数据进行计算，无法进行流式计算。

-Storm

https://blog.csdn.net/anzhsoft/article/details/38168025



Storm 将计算逻辑成为 Topology，其中 Spout 是 Topology 的数据源，这个数据源可能是文件系统的某个日志，也可能是 MessageQueue 的某个消息队列，也有可能是数据库的某个表等等；Bolt 负责数据的护理。Bolt 有可能由另外两个 Bolt 的 join 而来。而 Storm 最核心的抽象 Streaming 就是连接 Spout，Bolt 以及 Bolt 与 Bolt 之间的数据流。而数据流的组成单位就是 Tuple（元组），这个 Tuple 可能由多个 Fields 构成，每个 Field 的含义都在 Bolt 的定义的时候制定。也就是说，对于一个 Bolt 来说，Tuple 的格式是定义好的。

特点：有向无环、流处理，相比 MapReduce，对逻辑上稍微复杂一些的任务更有优势。

-参考流处理的方式搭建简单的分布式计算：

https://blog.csdn.net/weixin_34319640/article/details/85772695

利用消息队列，各机器一起从中读取数据，相比直接分发可以在一定程度上平衡负载，添加简单的监管机制就具备了分布式计算的基本特性。

-Spark

以 MapReduce 为基础，实现微批量数据处理，具有秒级的实时性。即通过将一秒内的数据收集起来，再针对这些数据进行批处理，因此和 Storm 相比实时性稍弱。通过将中间数据放在内存中而非磁盘中，提供了有向无环逻辑的处理可能。

2. 雾计算

雾计算通过在靠近终端的节点进行数据的处理计算，减少要传输到云端的数据量，

提高整体的运算能力。
雾计算与云计算的不同：
更低：雾节点在网络拓扑中位置更低，拥有更小的网络延迟（总延迟=网络延迟+计算延迟），反应性更强。
更多：相比较云平台的构成单位——数据中心，雾节点数量庞大。
更广：雾节点拥有广泛的地域分布。
更轻：雾节点更轻量，计算资源有限。

3. 总结

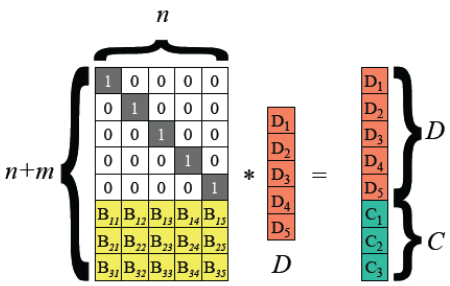
-考虑到局域网内设备数量，MapReduce 不一定适用于小规模的分布式计算
-关于怎么实现嵌入式设备上的分布计算没有找到有效的信息
-可以考虑和“雾计算“的概念进行结合
-可以考虑小规模分布式计算，怎样在少量设备中尝试提升计算效率？是否使用已有的开源架构？

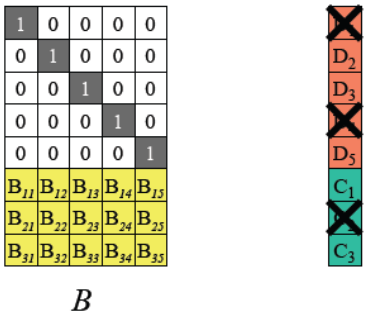
分布式存储

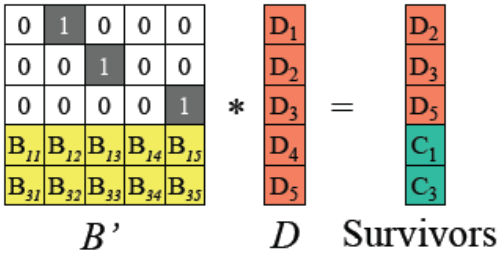
1. 纠删码

https://blog.csdn.net/shelldon/article/details/54144730

EC 的定义：Erasure Code 是一种编码技术，它可以将 n 份原始数据，增加 m 份数据，并能通过 n+m 份中的任意 n 份数据，还原为原始数据。即如果有任意小于等于 m 份的数据失效，仍然能通过剩下的数据还原出来。







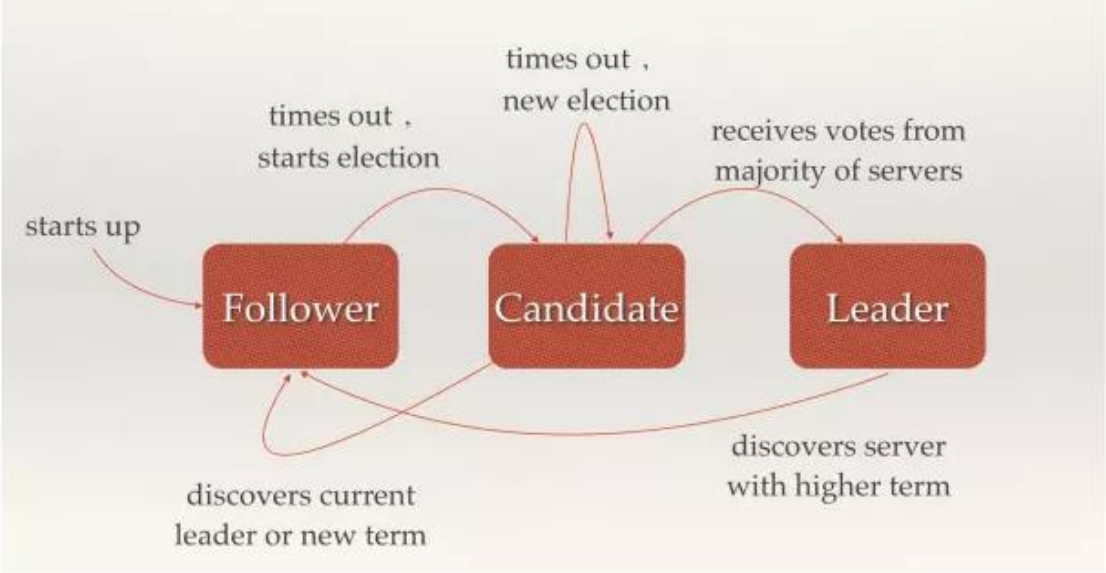
2. 一致性

https://www.jianshu.com/p/8e4bbe7e276c
分布式存储系统可分为强一致性/弱一致性/最终一致性。
Raft/Paxos　其中 Raft 在 2014 年被提出，相对更直观。

其中每个节点上都有一个倒计时器 (Election Timeout)，时间随机在 150ms 到 300ms 之间。有几种情况会重设 Timeout：

收到选举的请求

收到 Leader 的 Heartbeat (后面会讲到)



选主过程：

每个节点都有随机生成的计时器，计时器归零就开始发出询问，是否支持作为 leader。如果其他节点尚未支持过其他 leader 就会同意。要当选需要获得半数的同意。

故障处理：

如果 leader 节点故障，会重新进行选主。生成的 leader 会通过编号的方式记录，后选出的 leader 具有更高的优先级，当原先的 leader 重连会自动降级为 follower。

如果有多个同时竞选：

当各方都由于票数不过半导致无法继续时，计时器仍在运行。当某一个节点计时器结束时会发生优先级更高的新一轮选举。

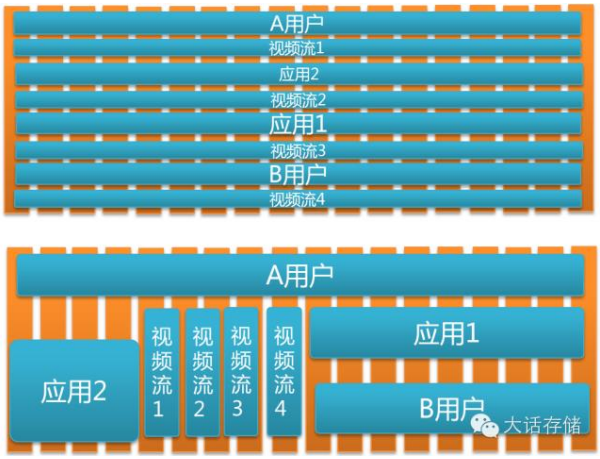
日志处理：

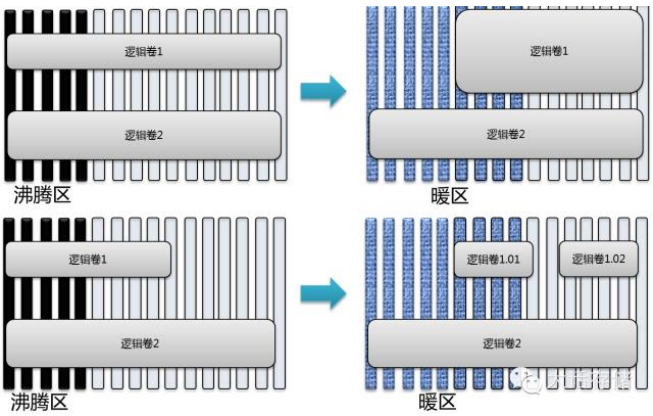
利用 committed 的状态进行确认。只有复制到超过半数的节点才会确认数据被保存。

3. 负载均衡 /应用感知

在分布式存储中，存储资源差异较大。处理逻辑和物理空间的矛盾。根据存储内容的使用频率/使用特征等，对其存储位置进行合理的分布。

在磁盘中有大量的运用：将高热度的内容存储在容易读取的部分，将不常访问的内容放在较难访问的部分。更智能的：根据读取时的速率要求/资源之间是否要求同时访问等，考虑将内容存储在一个磁盘上或多个磁盘上等。





在分布式存储中，如果有某些存储资源访问效率更高，可以使用负载均衡，合理分配资源存储位置，从而提高运算的效率。

相关工作：**《负载感知的基于对象存储的分布式混合存储系统》**

基于 Ceph 进行了相关的研究，编写算法衡量资源的热度，并对数据迁移等过程进行设计

4. 总结

-考虑使用共识算法 Raft 对某些文件系统进行改写？但或许有其他牵涉因素。

-针对负载均衡问题对文件系统进行一些优化