分布式系统设计模式

汪源

内容介绍

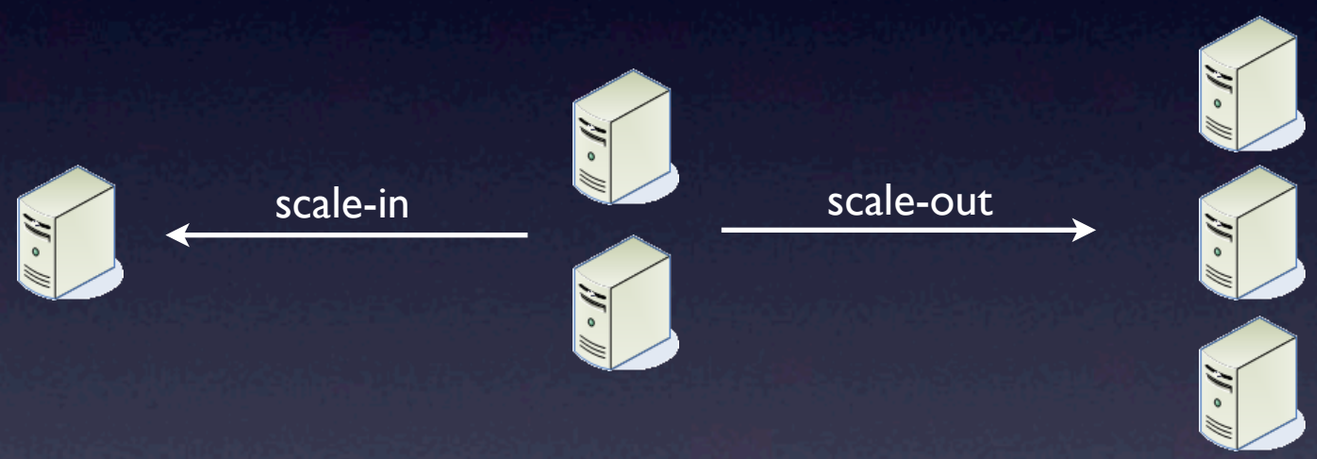
- 目标
  - 介绍一些分布式系统设计思路或经验，非严格意义的设计模式
  - 不求全面，但求最实用
- 提纲
  - 可伸缩设计模式
  - 高可靠可用设计模式
  - 低成本设计模式

可伸缩设计模式

伸缩模式(I)

垂直伸缩

水平伸缩



读写分离

伸缩模式(2)

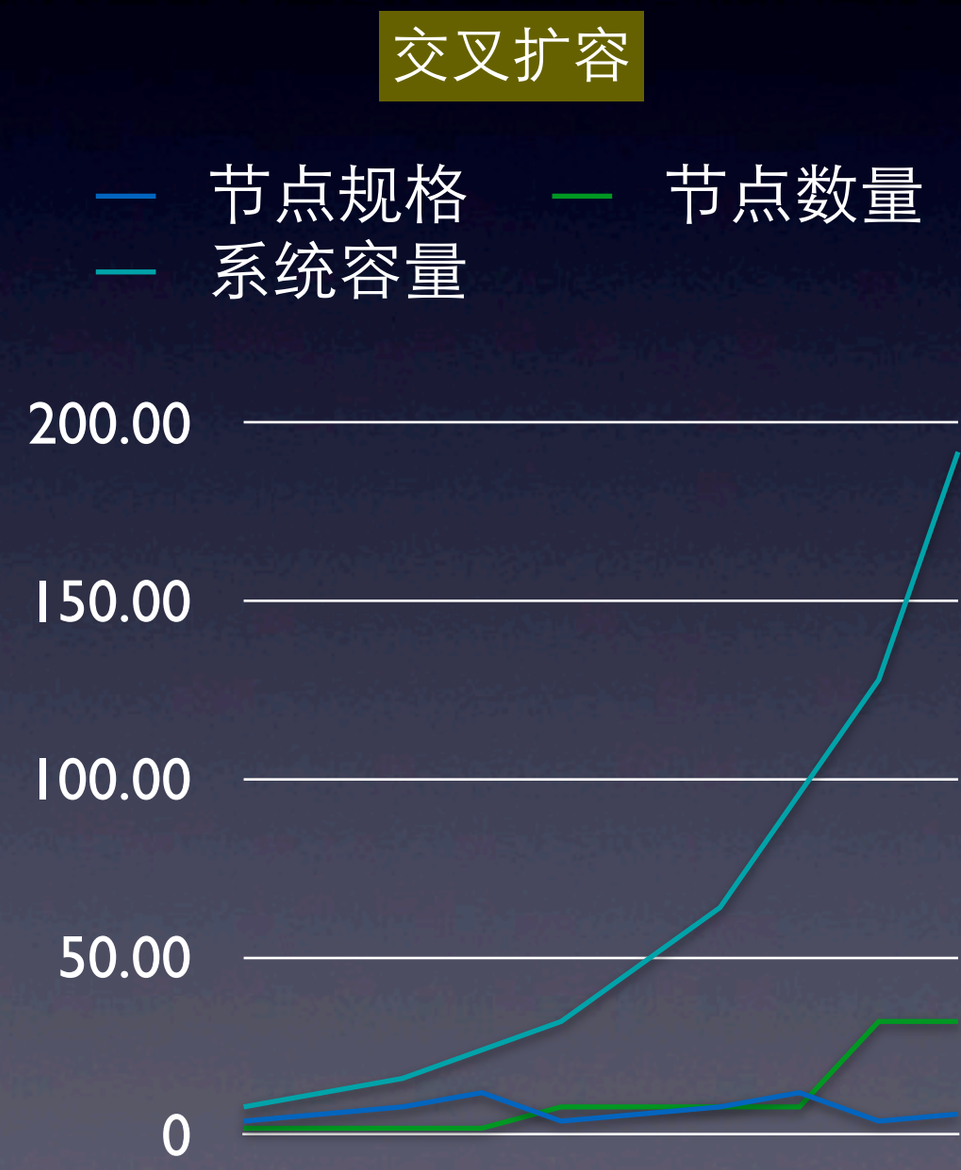
- 垂直伸缩
  - 易于实现
  - 传统观念：不好(规格有限，淘汰损失)
  - 基于IaaS云计算：不错

    丰富的硬件规格(1-64ECU/64x,1-64GMEM/64x,5G-1T存储/200x)

    资源共享无浪费

    优先进行垂直伸缩，简化系统实现与实施
- 水平伸缩
  - 伸缩性最佳
  - 难以实现与运维
- 读写分离
  - 实现难度中等
  - 只适合读多写少的应用

交叉扩容



优势：只需要进行1分为N的水平扩容

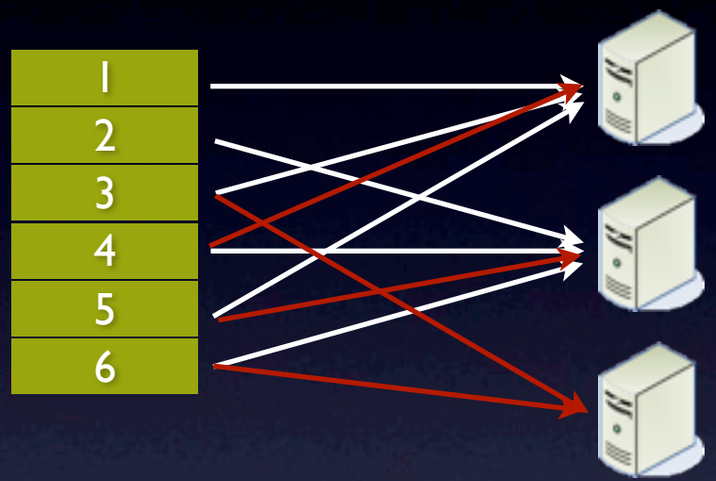
水平分区

- 哈希分区
  - 推荐：易于规划，易于实现，均衡性一般很好
  - 避免过于trivial的哈希函数（如取模），以免引起分布不均衡

    教训：DDB（按时间戳分配全局ID导致分布不均衡）
- 范围分区
  - 一般不推荐：难以规划，均衡性容易出问题
  - 时间分区比较常用
- 组合分区
- 自定义分区
  - 例：DDB通过用户自定义.jar包计算纪录的分区号

一致性哈希

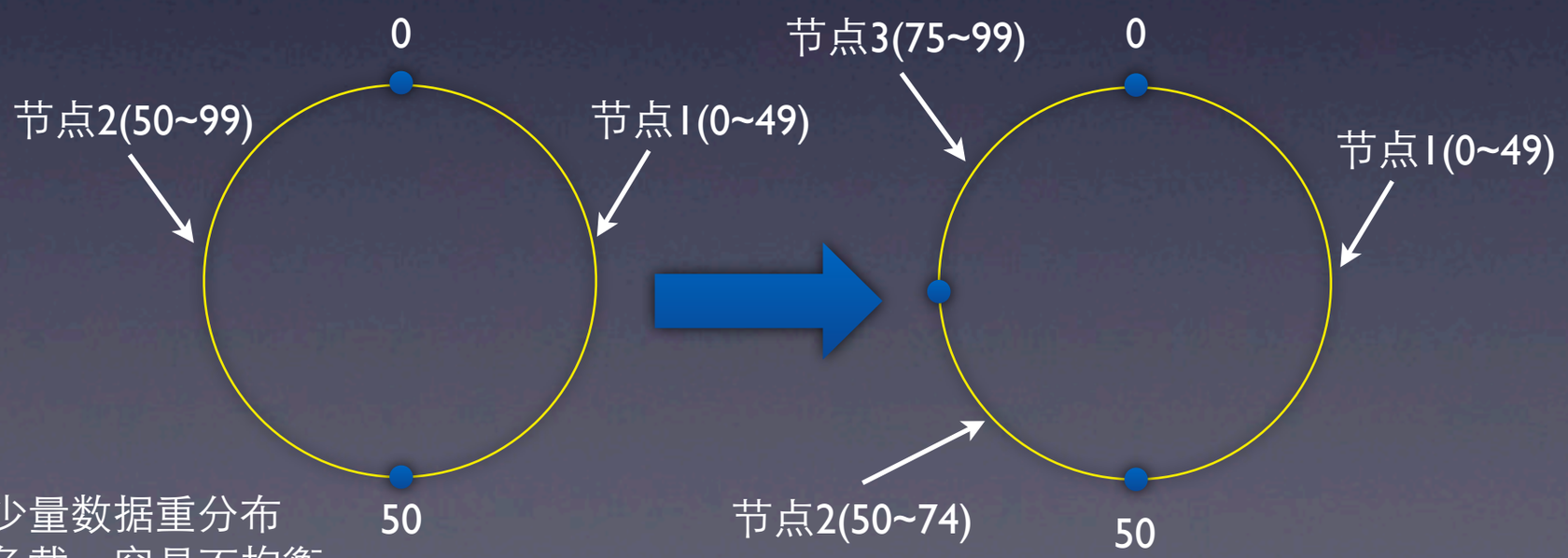
朴素哈希分区



分区策略: HASH%N
缺陷： 节点数变化导致数据完全重分布

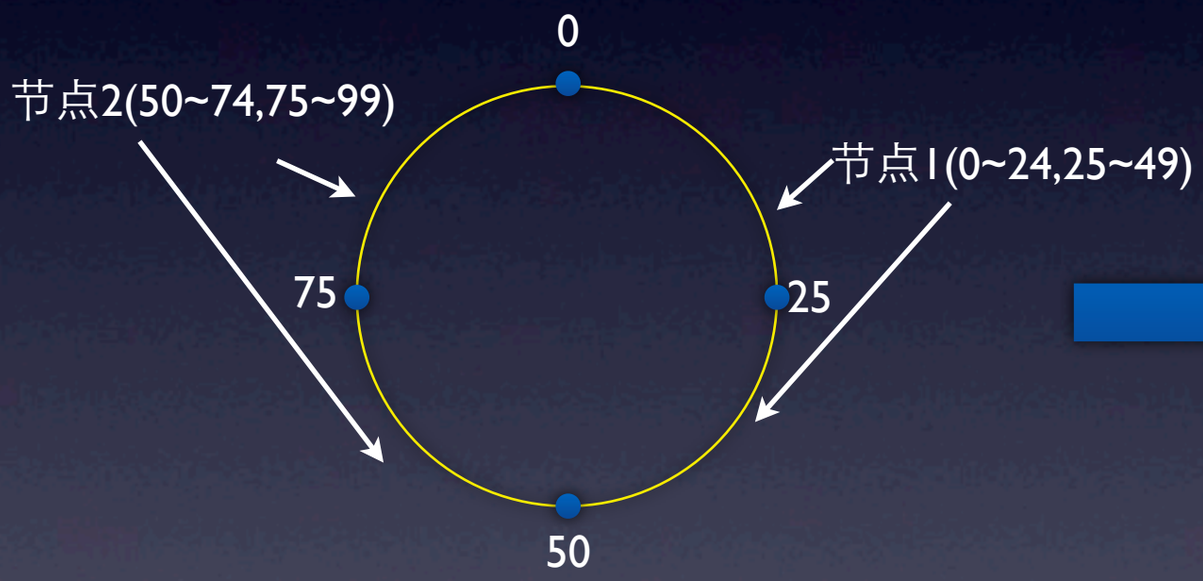
新增节点3

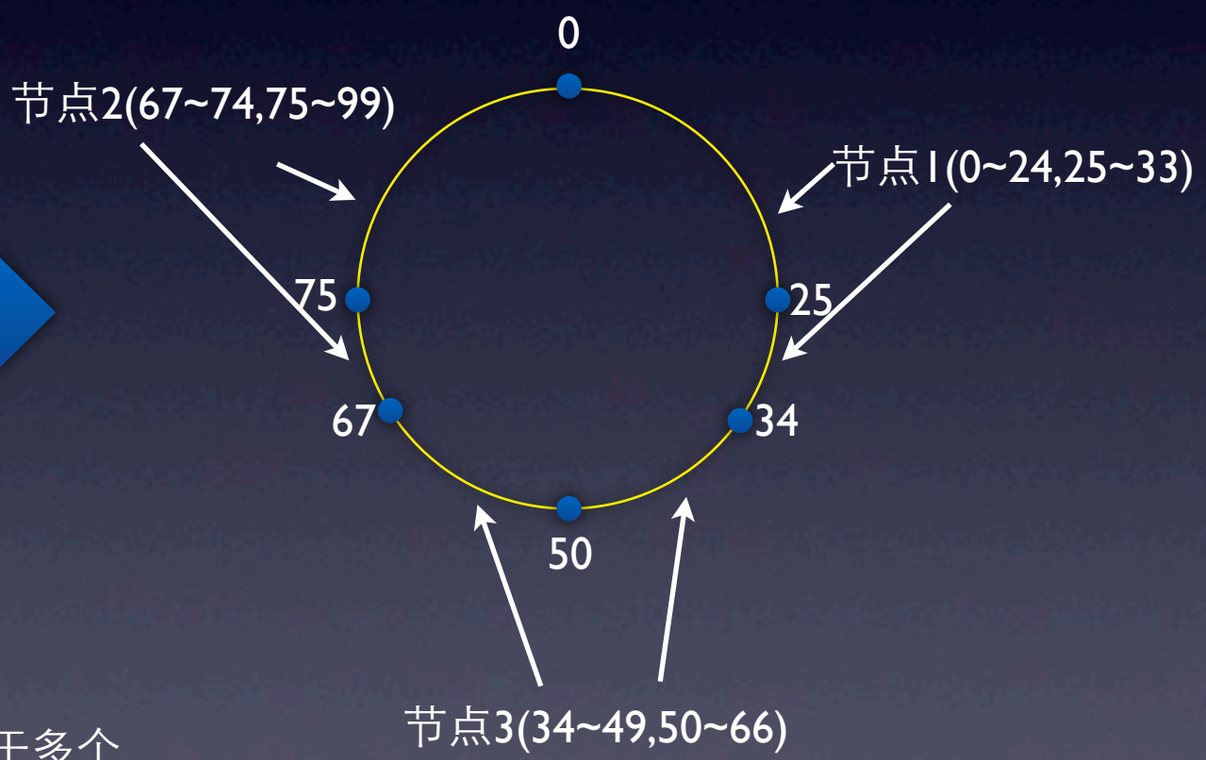
一致性哈希分区



优点： 节点数变化只导致少量数据重分布
缺点： 只分流一个节点的负载，容易不均衡

虚拟节点一致性哈希





一个物理节点负责多个虚拟节点，增加节点时从来自于多个物理节点的虚拟节点都分流一部分负载

固定分区路由表

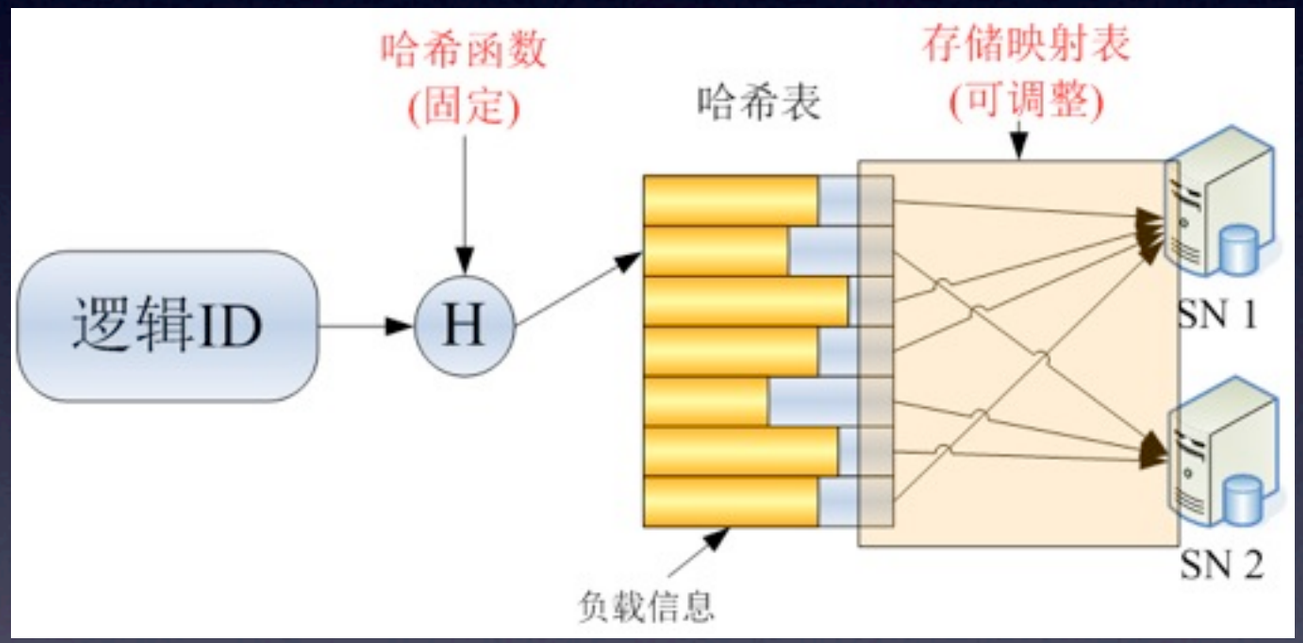
- 事先规划固定数量足够多的分区(相当于虚拟节点)，之后只调整分区到物理节点的映射关系(即只会迁移整个分区而不分裂分区)
- 简洁实用
- 分区数量规划原则

  > 10x 最大物理节点数

  因子数多(60, 3600)

  60 = 60 * 1
  = 30 * 2
  = 20 * 3
  = 15 * 4
  = 12 * 5
  = 10 * 6
  = 6 * 10
  = 5 * 12
  = 4 * 15

网易DDB



数据迁移(I)

- DDB数据迁移算法历程
  - 第一代：通过分布式事务进行细粒度迁移，迁移中的分区要合并源和目的结果

    性能极差，大量死锁，完全不可用
  - 第二代：禁止待迁移分区的访问

    性能不错，但影响可用性
  - 第三代：基于MySQL复制实现一分为二扩容

    在线迁移，不影响可用性，性能不错，但伸缩模式受限
  - 第四代：基于MySQL交叉复制实现多对多伸缩

    在线，性能不错，伸缩模式灵活，但操作复杂

数据迁移(2)

- 基于复制的高性能在线数据迁移
  - 分区数据隔离（推荐）

    在目的节点建立待迁移分区数据的复制，等待复制同步后，修改路由信息
  - 分区数据融合

    在目的节点建立源节点的复制， 等待复制同步后，修改路由信息，而后清理不需要的分区数据
- 路由表版本号
  - 同步修改缓存于所有客户端的路由表困难怎么办?

    为路由表设置递增的版本号，迁移时增加源节点的路由表版本号

    客户端请求源节点，发现路由表版本不匹配，同步路由表后正确路由至目的节点

无迁移扩容

- PB级以上存储需要采用无迁移扩容
- 元数据库
  - 对象级元数据记录存储位置，元数据与数据分离。数据扩容无需迁移，元数据扩容仍需迁移但数据量小
  - 案例：网易NOS，元数据使用网易DDB，数据使用网易SDFS
  - 凡是没有元数据的海量存储都是耍流氓
- 分区预留
  - 案例：网易DFS/SDFS

    文件访问键由系统分配而不是应用程序指定，因此访问键中可以包含分区号

    事先规划好65536个分区，但不立即启用

    增加一个物理节点时，启用一批分区分配给该物理节点

    系统生成文件的访问键时，使用新分配的分区号
  - 适用范围

    访问键由系统分配且可包含分区号

    负载在初始分配完成后基本不变

高可靠可用设计模式

可靠/可用性数据

- 软件/OS故障： >10%
- 硬盘故障： 8%(参考Google报告)
- 内存/网卡等故障： 1%
- 交换机故障： <1%

可靠性计算:复本

- 多复本系统综合可靠性(近似)：$P = p^r \times t^{r-1} / d^{r-1}$
  - p：单个设备的故障率
  - d：故障率周期
  - t：故障恢复时间
  - r：复本数
- 增加复本数对提高可靠性最有效
- 复本数一定时关键在于加快恢复速度

可靠性计算:组合

- 组合系统的综合可靠性：$P=1-(1-p)^n \approx np (np \ll 1)$
  - p：单个设备的故障率
  - n：组合包含的设备数，即组合规模
- 扩大组合规模伤害系统可靠性，故障率随组合规模线性增长
  - 不建议做很多块磁盘组合的RAID 0或RAID 1+0

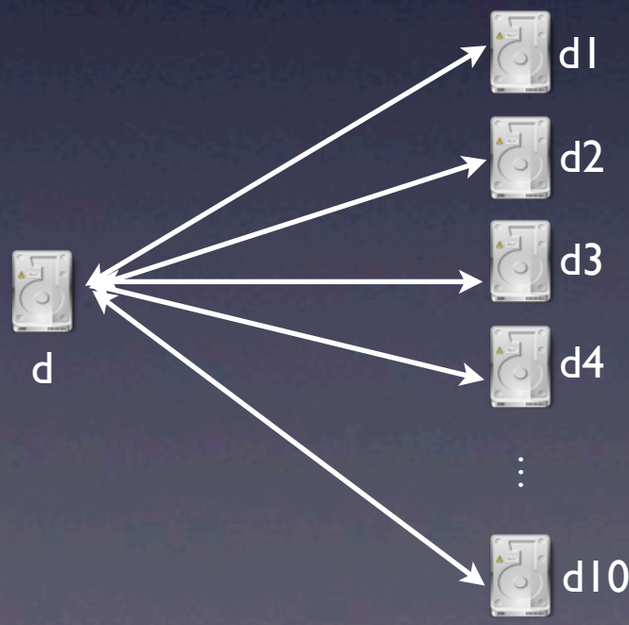
可靠性三原则

- 增加复本
- 加快恢复
- 避免组合

复本分布与规模效应

一对一复制

磁盘容量: 900GB
恢复带宽: 20MB/s
恢复时间: 45000s
可靠性: 99.999%



一对多复制

磁盘容量: 900GB
恢复带宽: 2000MB/s
恢复时间: 450s
可靠性: 99.99999%

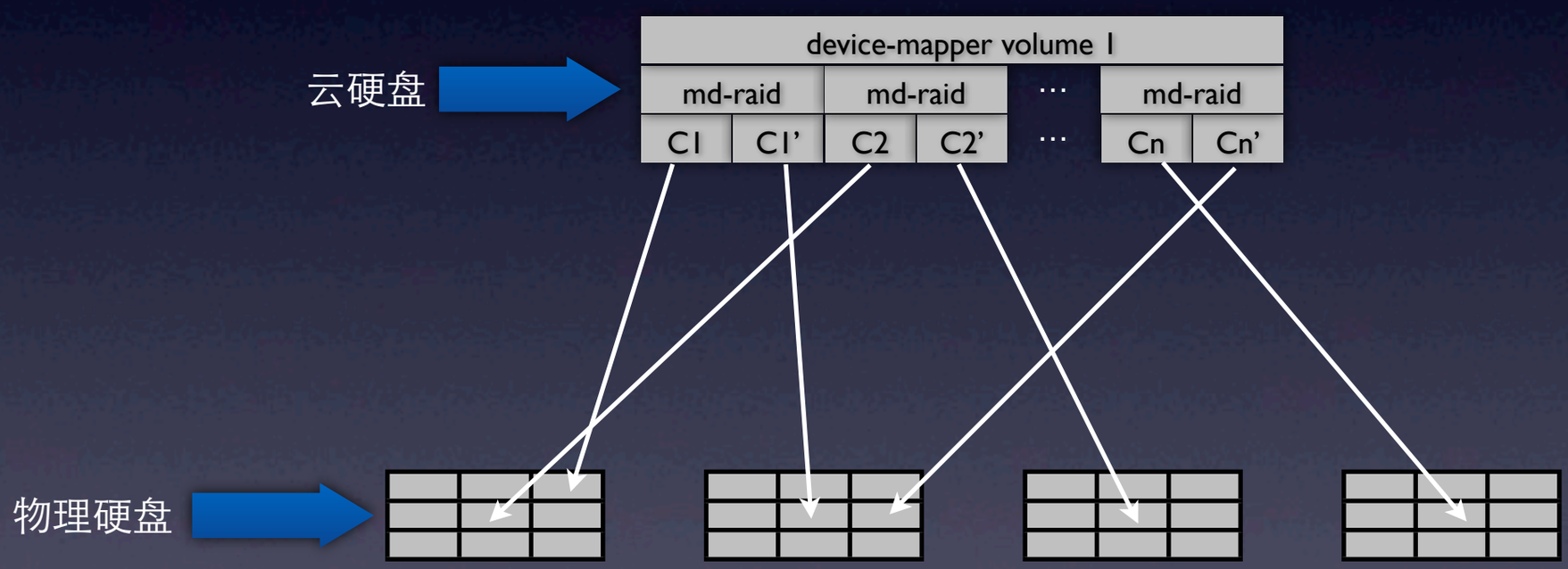
- 分布式一对多复制以进行并行快速恢复是提高数据可靠性的良策；
- 在一定规模范围内(如<200节点)，恢复并行度只取决于集群规模。恢复时间与集群规模成反比；
- 云计算的数据可靠性规模效应。

集群可靠性

- PB级本机RAID 1可靠性可以接受
- 10PB级分布式2复本可靠性可以接受

	有效容量	集群规模	磁盘数	数据恢复速度(MB/s)	年丢数据概率
本机 raid 1	1PB	50	500*2	20	1.01%
	10PB	500	5000*2	20	9.65%
分布式 2复本	1PB	50	500*2	20*20	0.05%
	10PB	500	5000*2	20*200	0.5%

案例: 云硬盘(I)

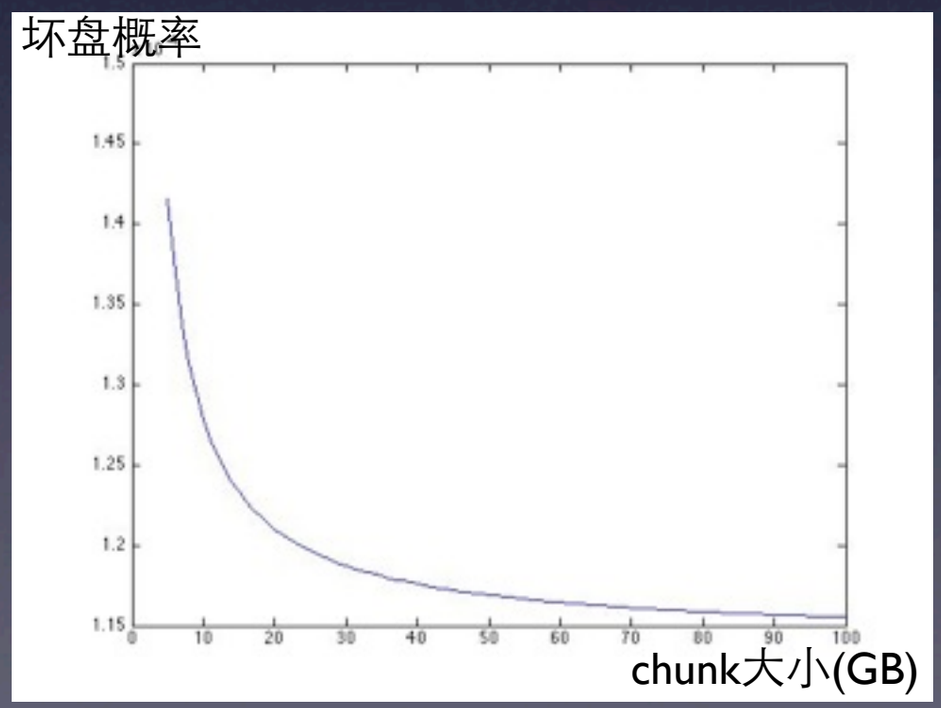


- 一个云硬盘划分为多个chunk
- 每个chunk含两个复本
- 每个物理硬盘存储来自于多个云硬盘的chunk

案例: 云硬盘(2)

```
disk_size = 900;     % disk size in GB
chunk_sizes = 5:1:100;    % chunk_size in GB, incremented in 5GB
check_time = 1;     % Time to affirm that a disk is demaged in munites
daily_fault_rate = 0.08 / 365;
recovery_bw = 20;    % Bandwith of recovery in MB/s
recovery_times = check_time + chunk_sizes * 1000 / recovery_bw / 60;
num_chunks = disk_size ./ chunk_sizes;
fault_one_probs = 1 - (1 - daily_fault_rate) .^ (recovery_times / 24 / 60);
fault_probs = 1 - (1 - fault_one_probs) .^ num_chunks;

plot(chunk_sizes, fault_probs);
```



- chunk大小多大最好?
  - 小chunk，恢复时间短
  - 大chunk，受影响的云硬盘数量少(组合规模小)
- 结论：增加chunk大小时，可靠性稍有下降，但幅度较小，设计时可忽略

多复本读写

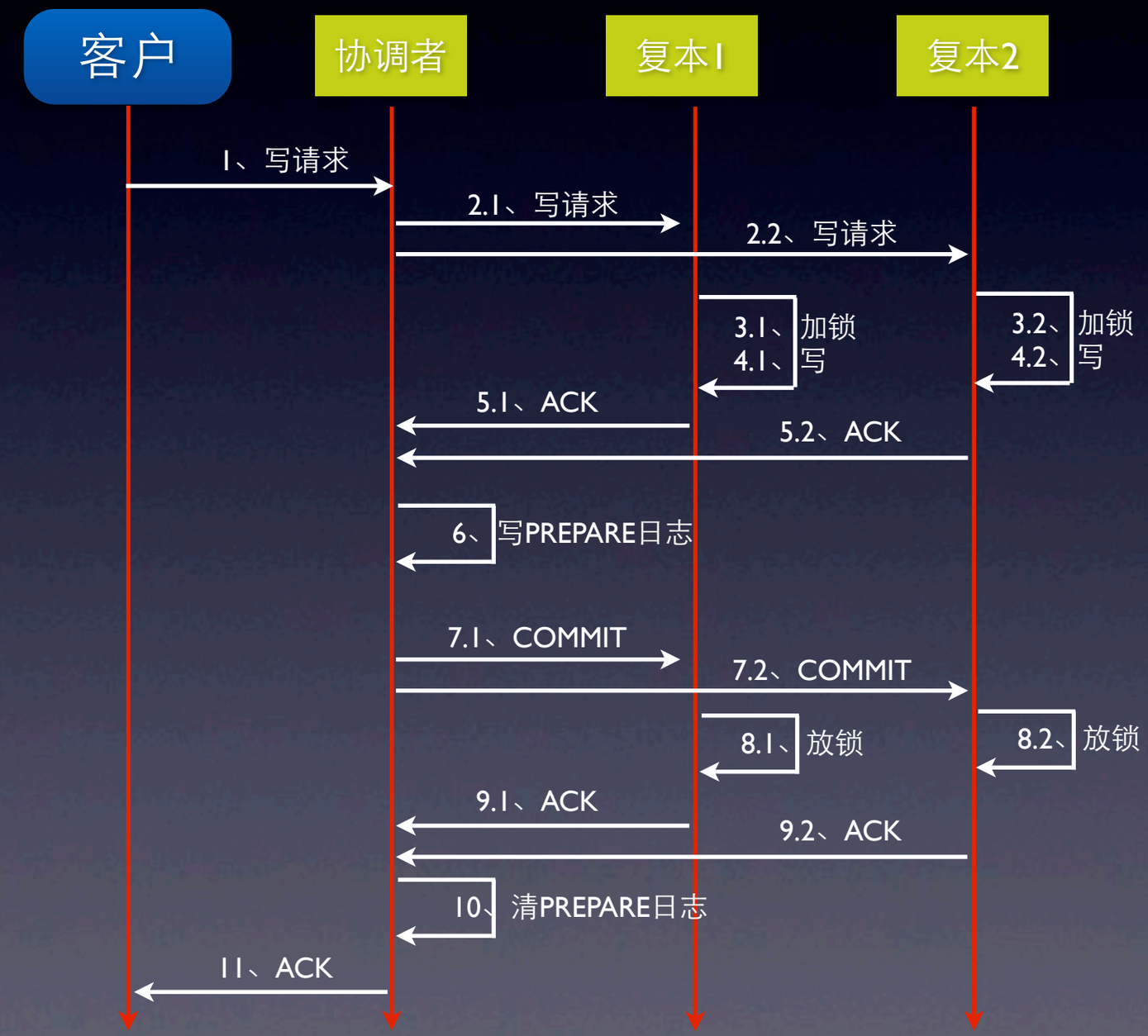
- 读一写全(WARO, write-all-read-one)
- Quorum读写
- 热备

读一写全

- 机制
  - 使用两阶段提交保证终态一致

    复本故障时用另一复本同步
  - 加锁以保证更新期间读取到一致数据

    若不用多版本将导致更新期间不可读
- 分析
  - 需要高可靠系统存储协调者日志
  - 可用性差



Quorum读写

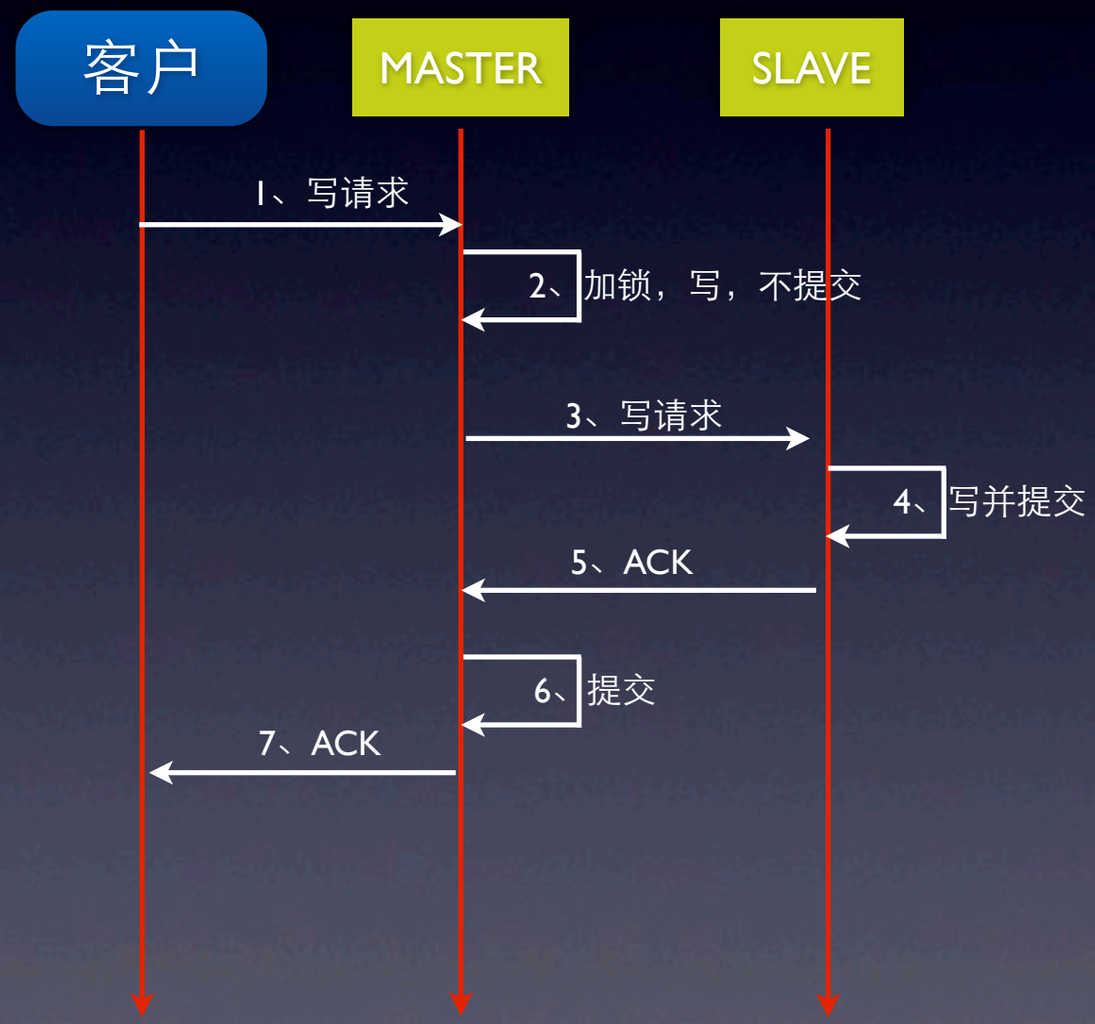
- 基本规则
  - 复本数N，写W个复本，读R个复本
  - 保证W+R>N
- 基本规则可保证最终一致，无法保证强一致
  - 无法区分更新成功还是失败
- 强一致规则
  - 前一个更新成功后才可以执行后一个更新（不易实现）
  - 一直读取到版本号最高复本的W个复本，若最终少于W个则判定更新失败
- 评价
  - 至少需要3复本，读效率低

热备

- 故障处理
  - SLAVE故障，中断复制，暂时退化为单复本，而后重建SLAVE

    SLAVE复活，从断点继续复制
  - MASTER故障，暂时退化为单复本，提升SLAVE为新MASTER，而后重建新SLAVE

    MASTER复活，成为新的SLAVE
- 评价
  - 牺牲暂时的可靠性，换取写效率与可用性
  - SLAVE不可承担一致性读



单主与租约(I)

- 故障检测不可能性
  - 无法判断服务故障还是检测者与服务器之间网络故障
- 案例：网络故障导致双主
  - 初始状态，所有客户端连接到MASTER
  - MASTER与控制中心间网络故障，控制中心认为MASTER已故障（其实还活着），启用新MASTER’
  - 部分客户端仍连接到MASTER，部分连接到MASTER’

单主与租约(2)

- 机制
  - 服务节点持续向控制中心申请短时间租约(Lease，一般10s)
  - 控制中心在已派发的租约过期之前，不启用新服务节点
  - 服务节点租约过期时若还无法从控制中心申请到新租约，自己中断客户连接(自杀)
- 评价
  - 易于实现
  - 依赖于高可用控制中心服务（通常用ZooKeeper）

    可用性受一定影响，但由于大面积网络故障罕见，可用性一般能满足需求

低成本模式

常见问题

- 怎么降低存储成本?
- 应该用什么存储介质?
- 要不要加缓存? 缓存要多大?
- 性能不足时，加内存还是磁盘?

IOPS/GB准则

- 最优单一存储介质只取决于
  - 应用IOPS/GB特征
  - 各类介质的每GB成本
  - 各类介质能提供的IOPS/GB性能
- 使用某存储介质时的每GB成本
  - 若应用IOPS/GB<介质IOPS/GB

    = 介质每GB成本
  - 若应用IOPS/GB>介质IOPS/GB

    = (应用IOPS/GB / 介质IOPS/GB) * 介质每GB成本

介质	$/GB	介质IOPS/GB	适用应用IOPS/GB
SATA	0.1	0.05	0~0.3
SAS	0.6	0.33	0.3~0.825
SSD	1.5	10	0.825~100
内存	15	>10000	>100

多级存储与缓存(1)

- 案例1： 设存储分10%热数据(IOPS/GB = 0.5)和90%冷数据(IOPS/GB=0.06)，综合IOPS/GB=0.104
  - 使用单一存储SATA：每GB成本=0.104/0.05*0.1=0.208
  - 使用二级存储，热数据用SAS，冷数据用SATA：每GB成本=0.1*(0.5/0.33)*0.6+0.9*(0.06/0.05)*0.1=0.199
  - 使用SATA存储，但用SSD缓存热数据：每GB成本=0.1*(0.5/0.3)*0.6+1*((0.9*0.06)/0.05)*0.1=0.199
- 案例2：设存储分10%热数据(IOPS/GB = 0.5)和90%冷数据(IOPS/GB=0.03)，综合IOPS/GB=0.077
  - 使用单一存储SATA：每GB成本=0.077/0.05*0.1=0.154
  - 使用二级存储，热数据用SAS，冷数据用SATA：每GB成本=0.1*(0.5/0.33)*0.6+0.9*0.1=0.181
  - 使用SATA存储，但用SSD缓存热数据：每GB成本=0.1*(0.5/0.33)*0.6+1*0.1=0.191

多级存储与缓存(2)

- 采用多级存储或热点数据缓存是否一定能降低系统成本?
  - 缓存过剩(或热点数据存储容量过大)时反而增加系统成本(案例2)
  - 缓存严重不足时，虽然能降低系统成本，但达不到最理想的效益
- 缓存容量经验法则(近似)
  - 冷数据访问IOPS/GB<介质IOPS/GB：缓存过剩
  - 冷数据访问IOPS/GB>介质IOPS/GB：缓存不足
  - 冷数据访问IOPS/GB=介质IOPS/GB：缓存合适
- 热点数据较少时，缓存与分级存储成本接近，而缓存更易于实施，工程实现往往实现缓存而非分级存储

常见问题解答

- 怎么降低存储成本?
  - 通用方法是选用合适的存储介质，且可区分热点与非热点数据，使用缓存或多级存储
- 应该用什么存储介质?
  - 统计应用访问IOPS/GB特征，结合各介质没GB成本和IOPS/GB能力计算
- 要不要加缓存？缓存要多大?
  - 比较访问IOPS/GB与冷数据介质IOPS/GB，调整缓存容量使二者接近
- 性能不足时，加内存还是磁盘?
  - 同上经验法则

其它成本优化方法

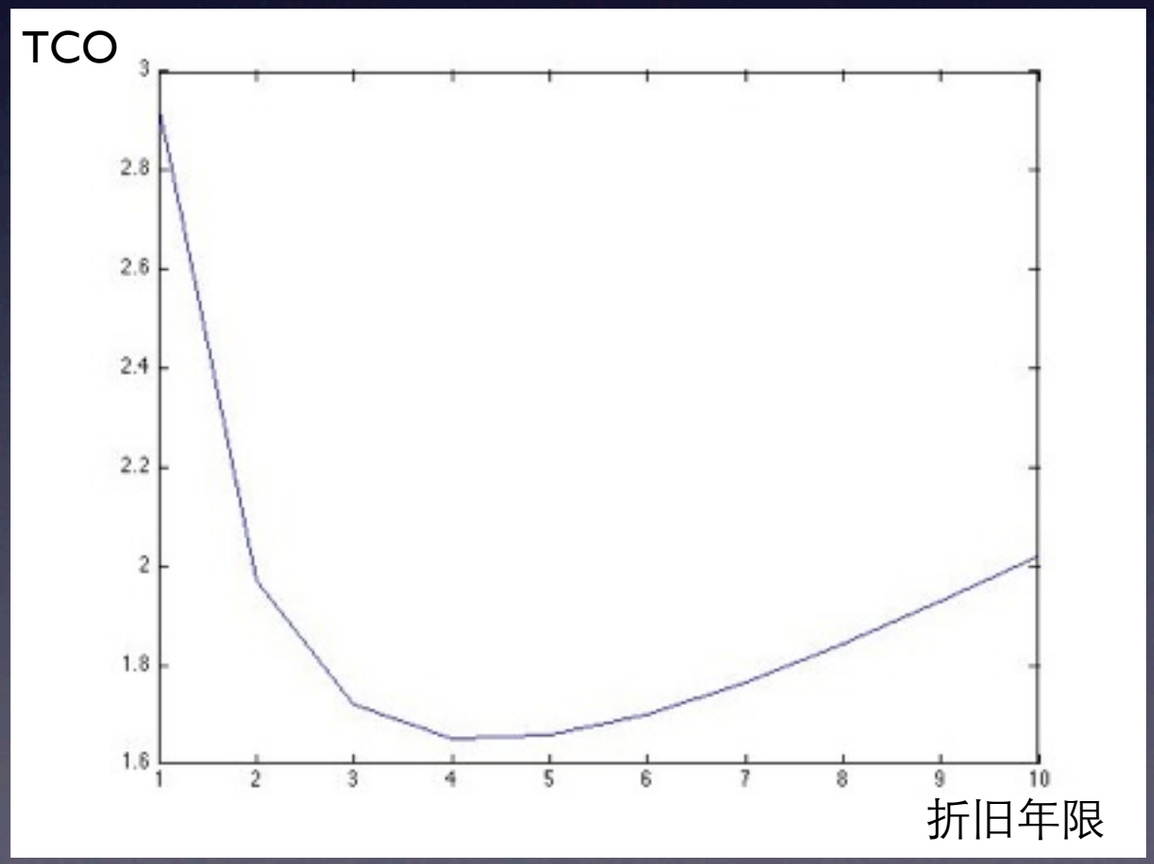
- 压缩
  - 除了zlib，可以尝试lzo、snappy等计算效率更高的压缩算法
- 去重
  - 对象级md5去重：相册/邮箱大附件去重率20-30%
  - 分块去重：最好基于内容（搜索哈希值符合某一特征的内容片段作为边界）而不是固定分块
    相册实验结果对象级去重与内容分块去重效果差别很小
- 纠删码
  - n块内容编码为n+k块，只要读取任意n块即可解码
  - 理论上可以用很小的存储代价实现非常高的数据可靠性（如10+2配置，20%开销，可靠性等效于3复本），但恢复时的带宽消耗非常高，分布式环境难实用（希望万兆网络普及）
  - 极度放大随机读写IOPS，从而影响性价比
- 升级硬件
  - 老旧低性能服务器机架托管成本很高
  - 一般推荐4-5年替换

升级周期计算

- 计算公式
  - 设： P=初始价格;T=托管价格;Y=折旧年限;g=年性能增长率

$$
\begin{aligned}
TCO &= (P+T\times Y)+\frac{1}{(1+g)^Y}(P+T\times Y)+(\frac{1}{(1+g)^Y})^2(P+T\times Y)+\dots \\
&= (P+T\times Y)(1+\frac{1}{(1+g)^Y}+(\frac{1}{(1+g)^Y})^2+\dots) \\
&= (P+T\times Y)\frac{1}{1-\frac{1}{(1+g)^Y}}
\end{aligned}
$$

- 典型数据
  - P=1
  - T=0.1
  - g=0.6



进阶阅读

《分布式基础》学习计划@网易云课堂

http://study.163.com/plan/planIntroduction.htm?id=272067#/planDetail

Q/A?