第三章作业

1.

- 早期Memcached采用标准的分布式方法（对键的存储根据服务器台数的余数进行分散）
  1. 求得键的整数哈希值
  2. 除以服务器台数，根据其余数来选择服务器
  3. 当选择的服务器无法连接时，rehash——将连接次数添加到键之后再次计算哈希值并尝试连接。
- 后期Memcached采用一致性hash
  1. 求出服务器节点的哈希值， 将其配置到$[0, 2^{32}]$的圆上
  2. 用同样的方法求出存储数据的键的哈希值并映射到圆上
  3. 从数据映射到的位置开始顺时针查找，将数据保存到找到的第一台服务器上
  4. 如果超过2^{32}仍然找不到服务器，就保存到第一台服务器上
  5. 后续改进：**虚拟节点**：为每个物理节点（服务器）在圆环上分配100～200个点，从而抑制分布不均匀，进一步优化服务器增减时的缓存重新分布。
- 采用第二种策略的原因：
  - 标准分布式方法当添加或移除服务器时，会涉及缓存重组，其代价较大。
- DynamoDB的数据分布策略的异同：
  - 同：都使用一致性hash算法，都有虚拟节点
  - 异：
    - DynamoDB虚拟节点的位置和大小不再随机分布，而是固定所有虚拟节点的大小和位置，只改变虚拟节点和节点的对应关系。
      - 将整个地址空间平均分成Q个虚拟节点，每个物理节点（假设有S个）分配Q/S个虚拟节点
      - 如果有节点加入：从现有节点每个拿出等量的虚拟节点分给新节点
      - 如果有节点离开：该节点的所有虚拟节点平均分配给余下节点
    - 此外DynamoDB对数据采用了多副本策略，每个数据有一个副本的preference list节点列表

2.

- Dynamo系统里，主要包括表（Table）、数据项（Item）、属性（Attribute）等组件
  - 表类似于关系数据库，表是所有数据的集合，每个表包含多个数据项（项目），每个项目包含一组属性，具有不同于所有其他项目的唯一标识，类似于关系数据库中的一行
  - 属性是基础的数据元素，可以不再进一步分解，类似于关系数据库中的列。除主键外，表的属性及其数据类型不需要预先定义，每个数据项都能拥有自己独特的属性。

    - 除主键外，表的属性及其数据类型不需要预先定义，每个数据项都能拥有自己独特的属性
    - 大多数属性是标量类型的，只能具有一个值，如字符串、数字等。也有一些属性是复杂类型的，具有嵌套属性如Address。Dynamo系统支持最高 32级深度的嵌套属性
- 读写数据过程：
    - PUT：
        1. coodinator生成新的数据版本，及vector clock分量
        2. 本地保存新数据
        3. 向preference list中的所有节点发送写入请求
        4. 收到W-1个确认后向用户返回成功
    - GET：
        1. coodinator向preference list中所有节点请求数据版本
        2. 等到R-1个答复
        3. coodinator通过vector clock处理有因果关系的数据版本
        4. 将不相容的所有数据版本返回用户

3.

- (1)
    1. 节点 i 本地发生一个事件时，$VC_i[i]$加1
    2. 节点 i 给节点 j 发送消息m时，将整个VC_i存在消息内
    3. 节点 j 收到消息m时，$VC_j[k] = max(VC_j[K], VC_i[k])$，同时$VC_j[j]$加1
- (2)
    - 会因为网络传输延时而丢失数据,信息可能会后发而先至，导致祖先消息被丢失
    - 向量剪枝只丢掉一些向量时钟的信息，而不是丢掉实实在在的数据，偶尔会有“false merge”，产生兄弟数据

4.

- (1)
    - 每个DB中的所有数据都是Key-Value对，其中的Value支持5种类型，分别是
        1. 字符串类型（String）
        2. 哈希表类型（Hash）
        3. 链表类型（List）
        4. 集合类型(Set)
        5. 有序集合类型（Sorted set）
- (2)
    - 查找主要基于Skip List的数据结构

1. 从最高层开始，从左到右遍历每一层，直到找到第一个小于等于目标值的节点
2. 如果当前节点的下一个节点不为空且小于等于目标值，则继续在当前层向右遍历；否则，将当前层数降低一层
3. 重复步骤2，直到遍历到底层为止
4. 如果在底层找到了目标值，则查找成功；否则，查找失败

- 查找过程：
  1. 从TOP出发，找到节点21，21<117，继续
  2. 找到节点37，37<117,继续
  3. 找到INT_MAX，层数降为2
  4. 找到节点71，71<117，继续
  5. 找到节点INT_MAX,层数降为1
  6. 找到节点85，85<117，继续
  7. 找到节点117，查找完毕

5.

- 核心数据结构为哈希表和日志段
- 内存访问基于hash表进行，内存数据采取LSM树结构，是由若干个日志段构成的，hash表记载记录在内存中的存储位置信息

6.

- SSTable提供一个可持久化的、有序的、不可变的从键到值的映射关系，其中键和值都是任意字节长度的字符串。SSTable由多个块组成，SSTable末尾存储了每个块的块索引
- SSTable包括块索引和行索引两级索引，根据块索引定位到块上，将块加载到内存后通过行索引查找到具体某一行