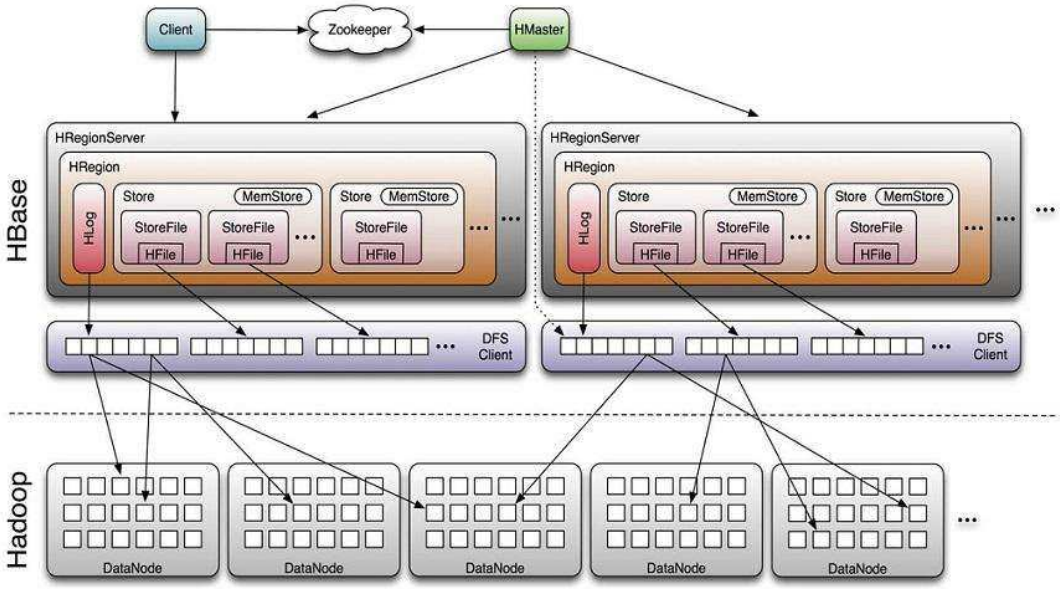
HBase 的优点：

列可以动态增加，并且列为空就不存储数据，节省存储空间。

Hbase 自动切分数据，使得数据存储自动具有水平 scalability。

Hbase 可以提供高并发读写操作的支持。



Client

包含访问 HBase 的接口并维护 cache 来加快对 HBase 的访问

Zookeeper

保证任何时候，集群中只有一个 Master
存储所有 Region 的寻址入口
实时监控 RegionServer 的上线和下线信息，并实时通知 Master
储存 Hbase 的 schema 和 table 的元数据

Master

为 RegionServer 分配 region，发现失效的 RegionServer 并重新分配其上的 Region
负责 RegionServer 的负载均衡（避免所有请求到一个 RegionServer 中去）
管理用户对 table 的增删改操作（table 存在于 region 里面）

RegionServer

RegionServer 维护 Region，处理对 Region 的 io 请求
RegionServer 负责切分在运行过程中变得过大的 Region

Region（一个表由多个或单个 region 组成）

HBase 自动把表水平划分成多个区域（region），每个 region 会保存一个表里面某段**连续**（关系到 rowkey 的设计，rowkey 的设计关系到查询效率的高低）的数据
每个表一开始只有一个 region，随着数据不断插入表，region 会不断增大，当增大到一个阀值后，region 就会等分为两个新的 region（裂变）
当 table 中的行不断增多，就会有越来越多的 region，这样一张完整的表就被保存在多个 Region 甚至多个 RegionServer 上

MemStore 与 storefile(StoreFile 对应 HFile，StoreFile 以 HFile 格式保存在 HDFS 上)（一个 store 对应一个列族 CF）

一个 region 由多个 store 组成
store 包括位于内存中的 memstore 和位于磁盘的 storefile，写操作先写入 memstore，当 memstore 中的数据量达到某个阀值，hregionserver 会启动 flashcache 进程写入 storefile（可以手动启动 flashcache 进程写入 storefile），每次写入形成单独的一个 storefile
当 storefile 文件数量增长到一个阀值后，系统会进行合并（minor（少个文件合并，不会对系统进程影响太大，因此设置自动开启），major（多个文件合并，文件数量过多时，合并会影响系统进程，导致其他进程阻塞，因此设置定时开启，读写数据少的时候开启）），在合并过程中会进行版本合并和删除工作（major）（在 hbase 表中设置版本数，如果为一，则表示只能存一条版本最新的，老版本并不会直接删除，而是会加上失效的标签，再进行 major 的时候，才会合并和删除），形成更大的 storefile
当一个 region 中所有 storefile 的大小和数量超过一定阀值后，会把当前的 region 分割为两个，并由 hmaster 分配到相应的 regionserver 服务器，实现负载均衡
客户端检索数据，先找 memstore(主要作用是写数据，但是也可以分担读取压力)，找不到再找 storefile

热点问题

热点的危害：

大量访问会使热点 region 所在的单个主机负载过大，引起性能下降甚至 region 不可用。

热点产生原因：

有大量连续编号的 row key ==> 大量 row key 相近的记录集中在个别 region ==> client 检索记录时,对个别 region 访问过多 ==> 此 region 所在的主机过载 ==> 热点

避免方法（随机散列与预分区）和优缺点

加盐

这里所说的加盐不是密码学中的加盐，而是在 rowkey 的前面增加随机数，具体就是给 rowkey 分配一个随机前缀以使得它和之前的 rowkey 的开头不同。给多少个前缀？这个数量应该和我们想要分散数据到不同的 region 的数量一致（类似 hive 里面的分桶）。加盐之后的 rowkey 就会根据随机生成的前缀分散到各个 region 上，以避免热点。

哈希

哈希会使同一行永远用一个前缀加盐。哈希也可以使负载分散到整个集群，但是读却是可以预测的。使用确定的哈希可以让客户端重构完整的 rowkey，可以使用 get 操作准确获取某一个行数据。

反转

第三种防止热点的方法是反转固定长度或者数字格式的 rowkey。这样可以使得 rowkey 中经常改变的部分（最没有意义的部分）放在前面。这样可以有效的随机 rowkey，但是牺牲了 rowkey 的有序性。反转 rowkey 的例子：以手机号为 rowkey，可以将手机号反转后的字符串作为 rowkey，从而避免诸如 139、158 之类的固定号码开头导致的热点问题。

时间戳反转

一个常见的数据处理问题是快速获取数据的最近版本，使用反转的时间戳作为 rowkey 的一部分对这个问题十分有用，可以用 Long.Max_Value - timestamp 追加到 key 的末尾，例如 [key][reverse_timestamp] ,[key] 的最新值可以通过 scan [key]获得[key]的第一条记录，因为 HBase 中 rowkey 是有序的，第一条记录是最后录入的数据。

其他办法

列族名的长度尽可能小，最好是只有一个字符。冗长的属性名虽然可读性好，但是更短的属性名存储在 HBase 中会更好。也可以在建表时预估数据规模，预留 region 数量，例如 create 'myspace:mytable'，SPLITS => [01,02,03,,...99]