北京交通大學

《非关系型数据库技术》
论文阅读报告

学校：北京交通大学

学院：软件学院

班级：软件 1905 班

姓名：张驰

学号：19301130

2021 年 12 月 31 日

Google File System读后感

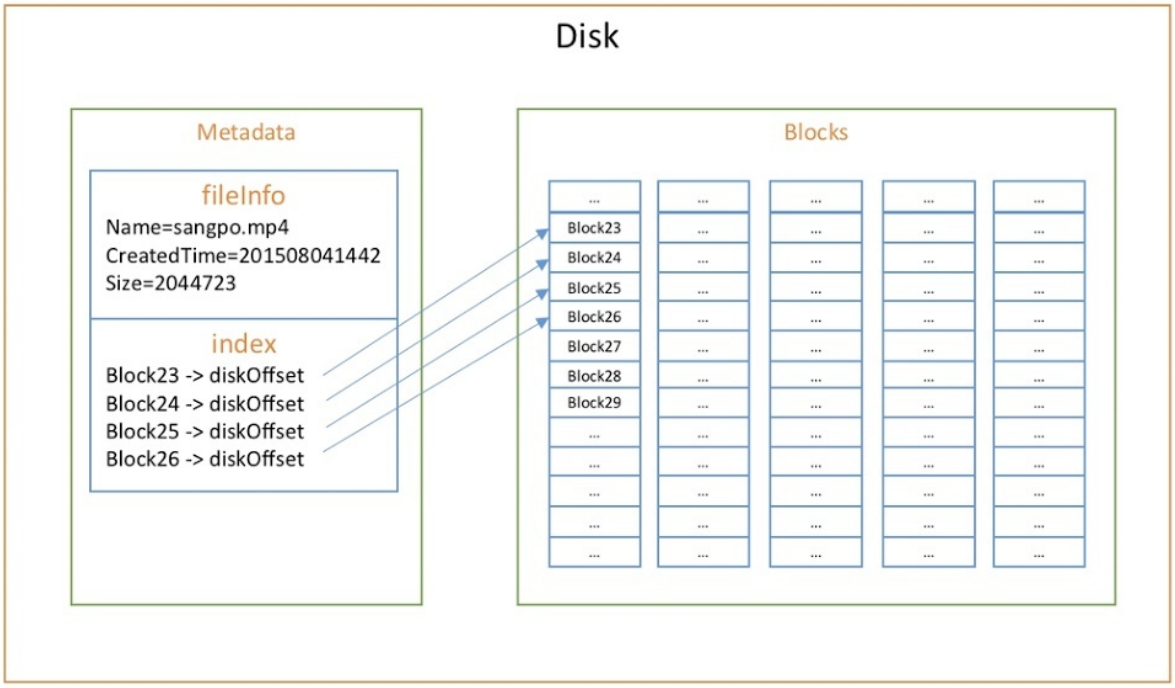
一、Google File System是什么?

GFS是一个可扩展的分布式文件系统，用于大型的、分布式的、对大量数据进行访问的应用。它运行于廉价的普通硬件上，提供容错功能。从根本上说：文件被分割成很多块，使用冗余的方式储存于商用机器集群上。

二、GFS设计概括

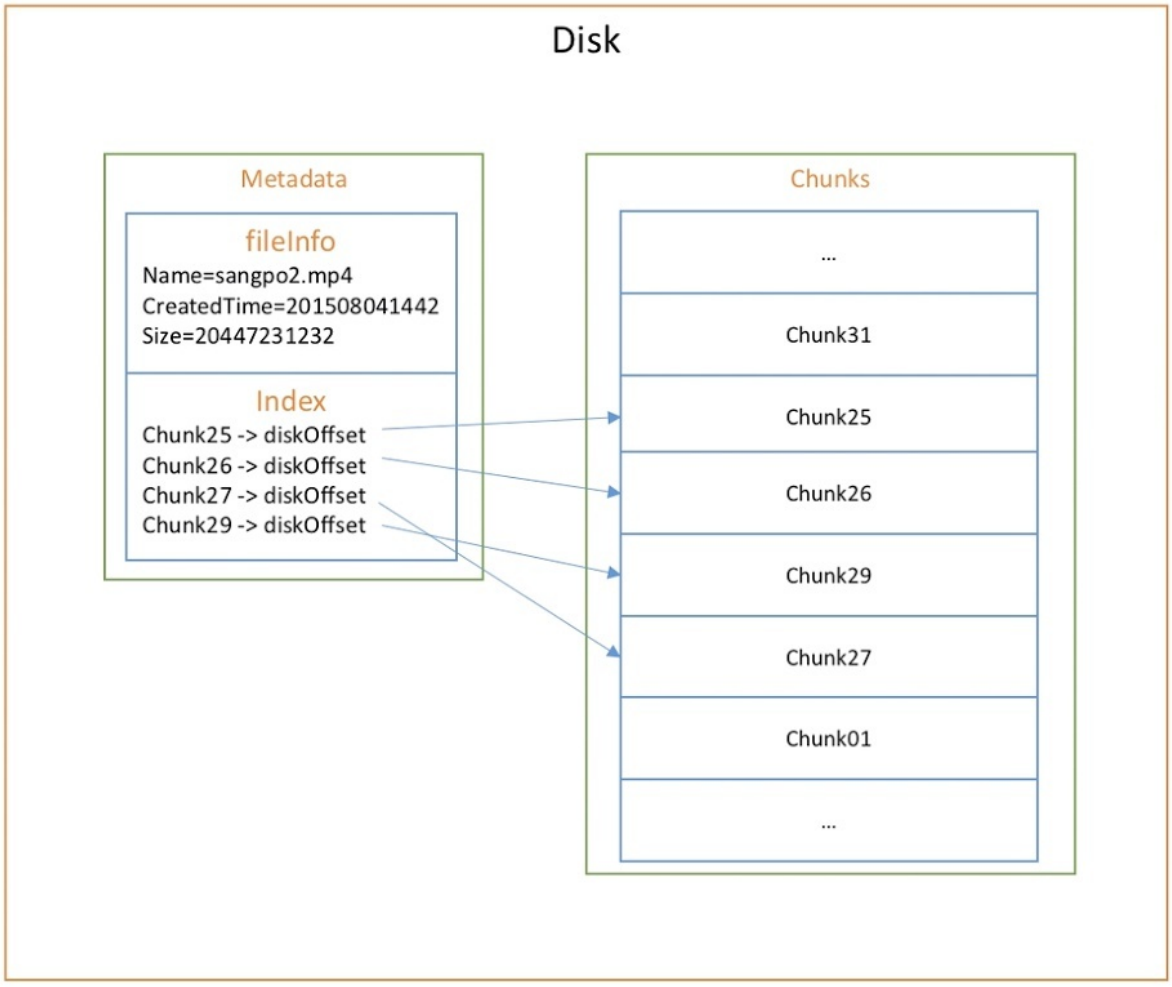
如何保存小文件

我们要保存一个文件，那么我们得有一个硬盘，硬盘中肯定得有一些元数据，比如说这个文件叫什么名字、创建时间和大小。同时，为了寻找数据在硬盘上的位置，我们就需要有索引（比如Block23，24，25），通过知道数据在硬盘上的哪一块以及对应的偏移量，我们就能找到数据在哪。



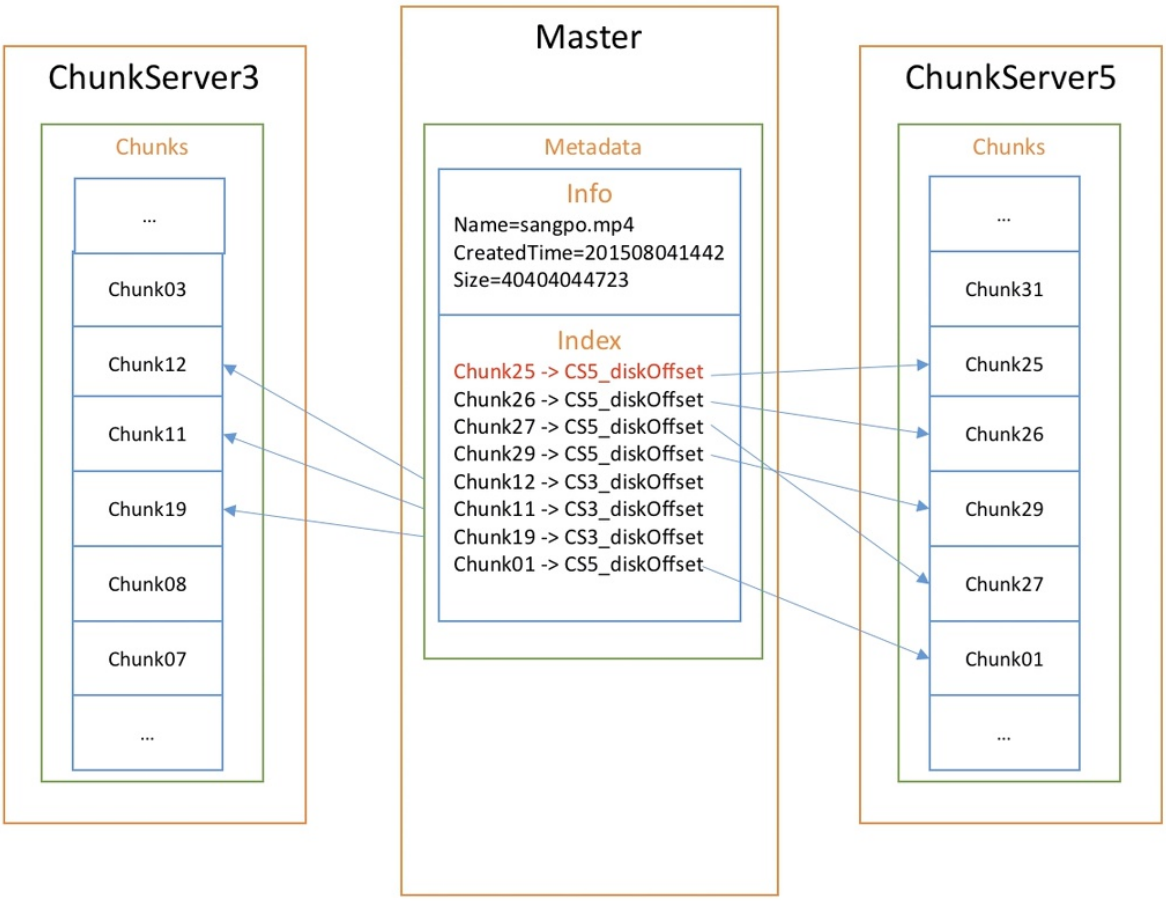
如何保存大文件

对于大文件来说，我们不能保存小块，因为块数太多了，所以我们只要将每一块变成大块，就是64MB，就变成大文件了。这里的大块叫Chunk，相当于6万多个小块（64MB = 64 * 1024 = 65536 blocks），这样可以极大地节省空间和原始信息大小。而缺点是如果存储小文件就会极大地浪费存储空间。



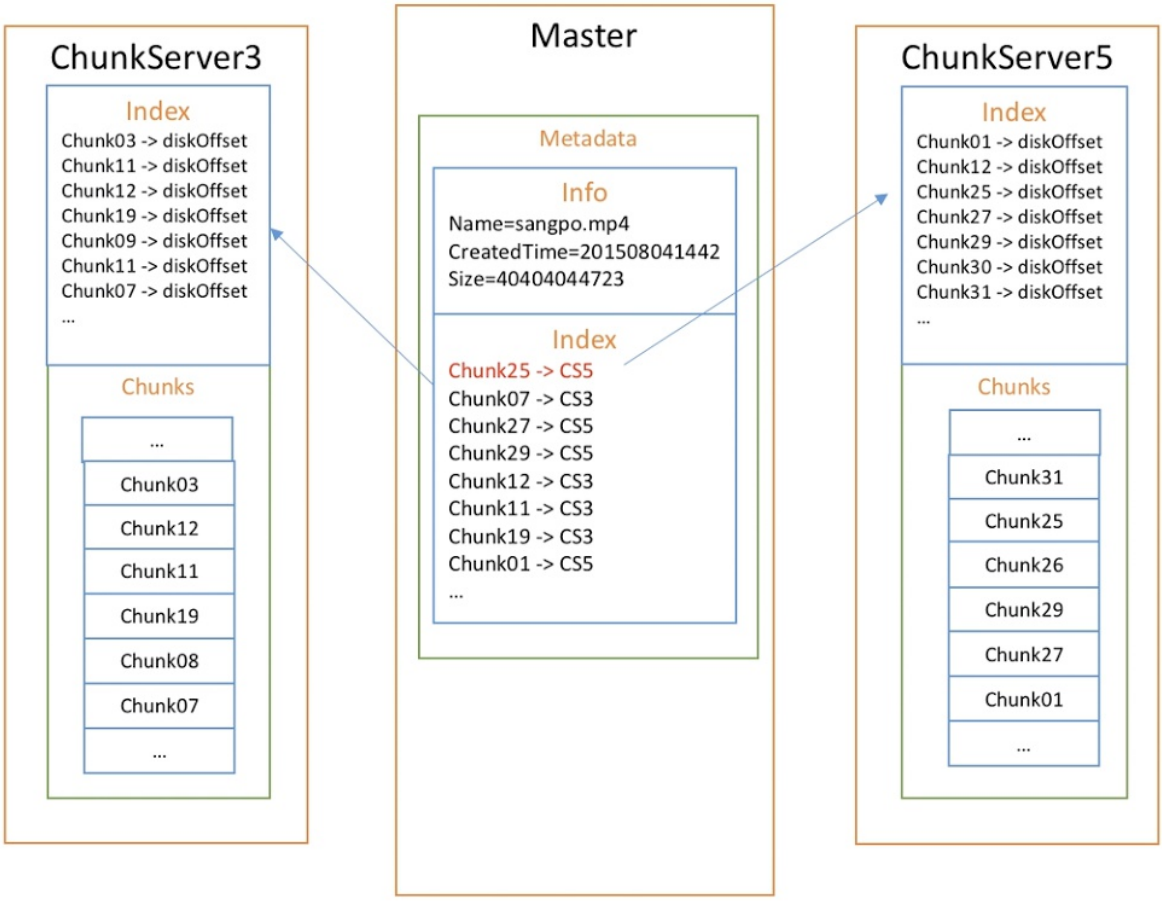
如何保存超大文件

保存超大文件的话，我们就不能像之前那样了。因为超大文件往往单机是存不下的，这时候我们就需要分布式地来保存文件，相当于是一个Master + 许多Chunk Servers结构。Master用于保存超大文件的原始信息，这时候的索引建立便可以是Chunk保存在哪个ChunkServer上，以及对应的偏移量，这样便将文件不同部分分布到不同的机器上。 但这缺点也很明显：ChunkServer数据的任何改变都需要通知Master，这样就会极大影响Master存储的数据，增大其流量。



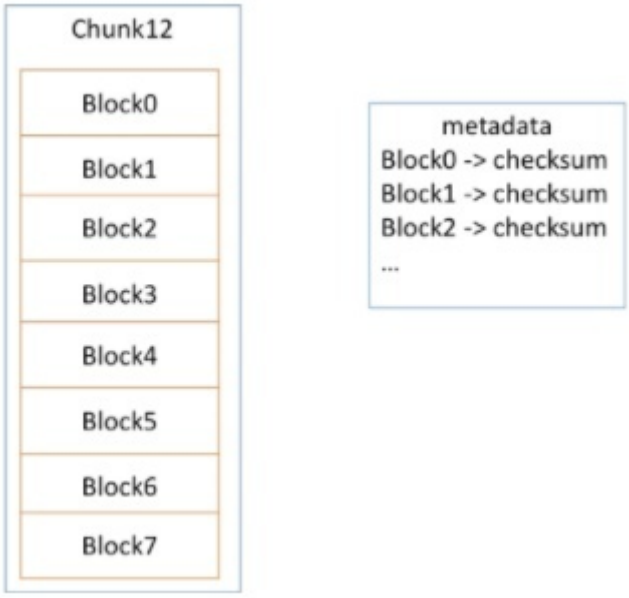
如何减少Master存储的数据和流量

这里我们可以优化成在Master服务器上，我们只保存每一个Chunk保存在哪个ChunkServer上，然后在对应的ChunkServer上来保存Chunk的偏移量。这样当Chunk发生变化时，本地的数据只用去改本地的索引就行，而不需要通知Master，除非其从一个ChunkServer去到另一个ChunkServer才需要通知。这样子的好处便是减少了Master的Metadata信息，同时也减少了Master和Chunkserver之间的通信。



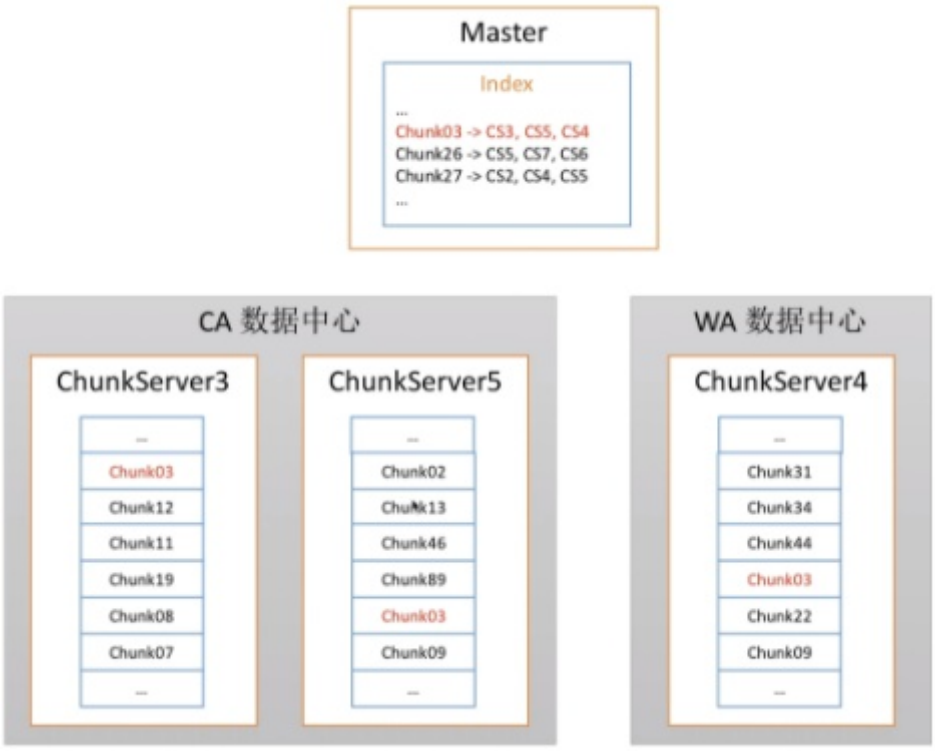
如何发现数据损坏

一个Chunk由很多小Block组成，每一个小Block（64KB）可以再保存一个checksum（32bit）。每次读取的时候通过checksum进行校验。假设说我们对1T的文件进行校验，那么其checksum的大小为：1T / 64KB * 32bit = 64MB。



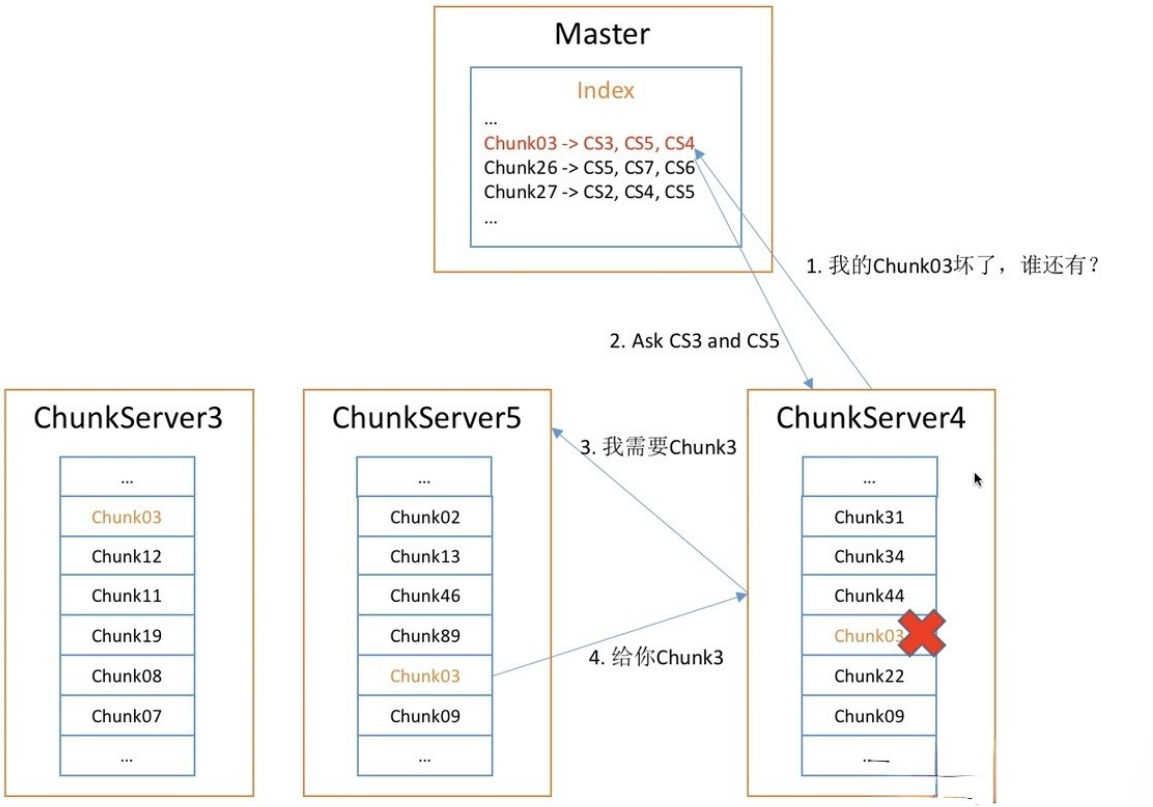
如何减少ChunkServer挂掉带来的损失

最直接的思路就是创建副本，即同一份Chunk可以存在多个ChunkServer中。当其中一个ChunkServer挂掉，都可以通过其他几个ChunkServer来恢复数据。 通常会选择硬盘利用率低的ChunkServer，同时，为了避免所有最新的数据块都写入同一个利用率低的ChunkServer带来的热点聚集问题，同时还会限制最新数据块的写入数量。



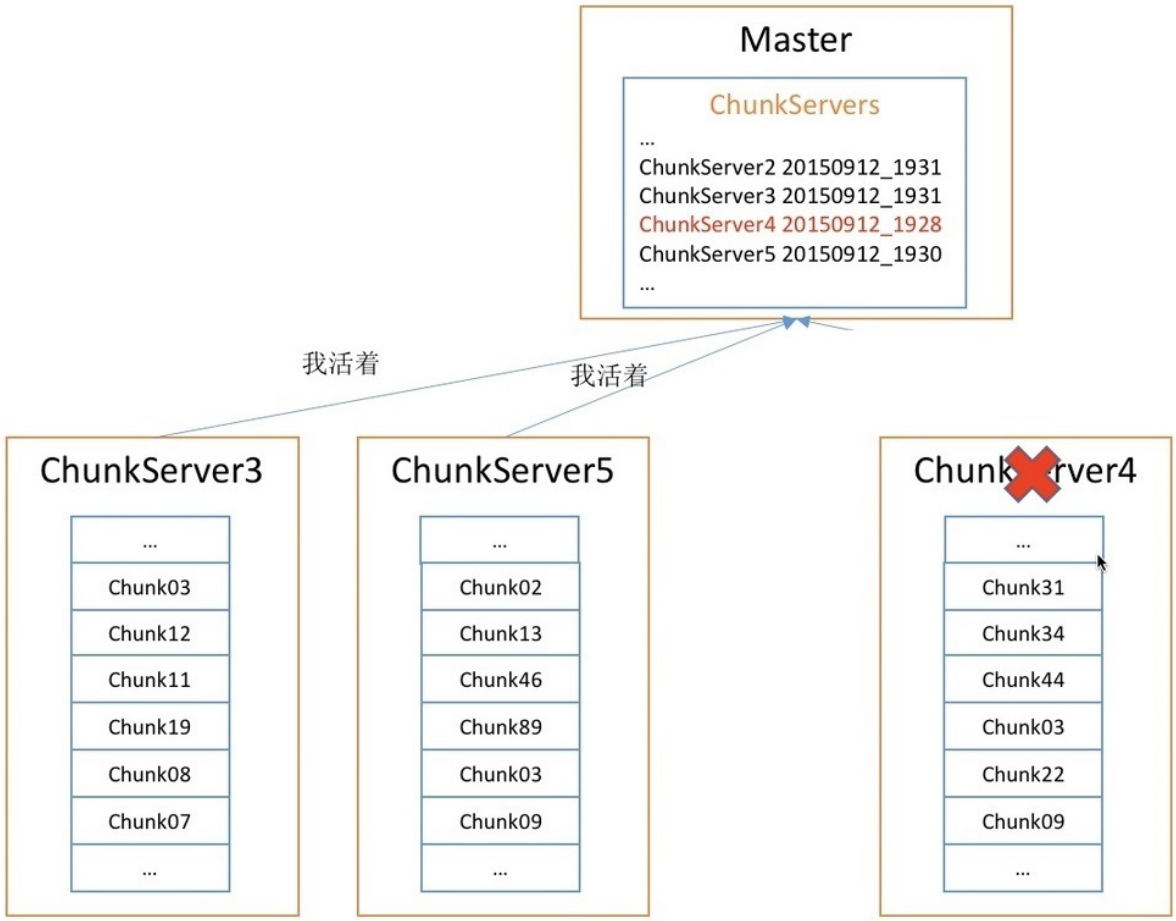
如何恢复损坏的Chunk

之前提到了Chunk副本的建立，那么当Chunk损坏时便可以利用副本来恢复。当ChunkServer发现其有Chunk损坏，会寻找Master请求帮助，Master会让该ChunkServer去找有这个Chunk的ChunkServer要副本。



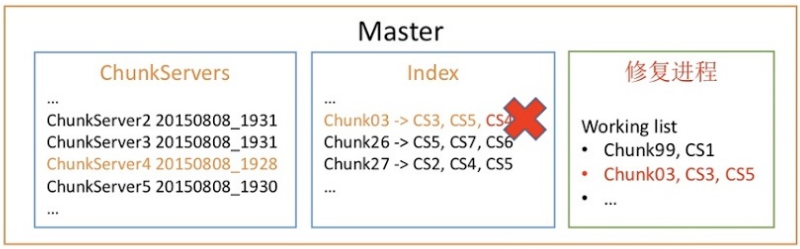
如何判断ChunkServer是否挂掉

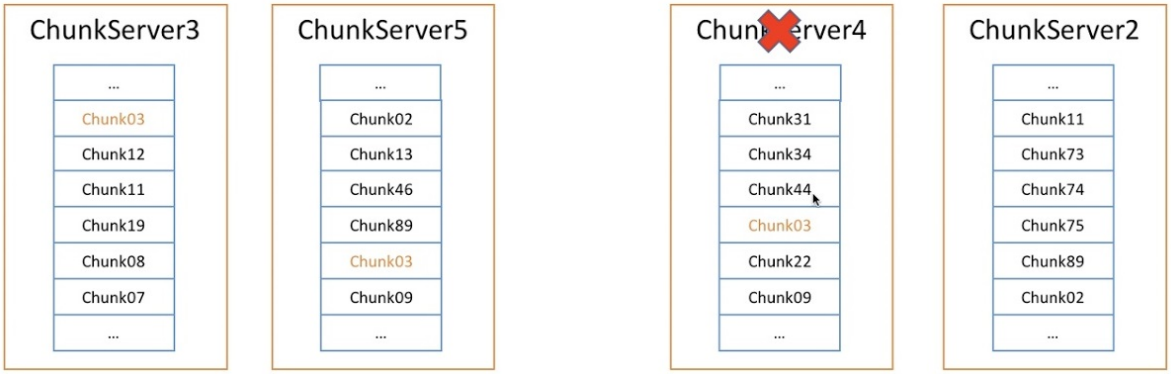
通过“心跳”的方式进行判断，如果ChunkServer很久都不回复，那么可能就是挂掉了。为了确保这个判断不是因为Master与其网络连接出现异常所造成的，其他的ChunkServer也会去呼叫这个可能挂了的ChunkServer。



ChunkServer挂掉后如何恢复数据

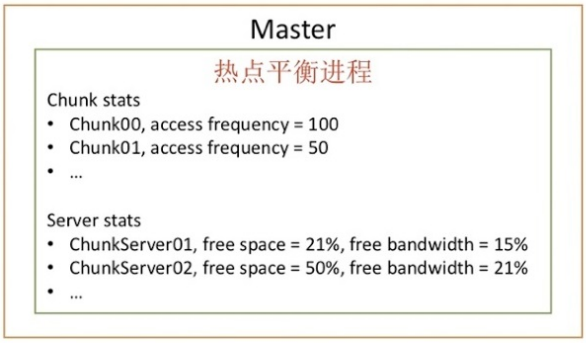
当有ChubnkServer挂掉后，Master便会启动一个修复进程，这个进程会记录需要恢复的Chunk，并且按照Chunk存活的数量递增进行排序，也就是说Chunk存活最少的会先被修复。

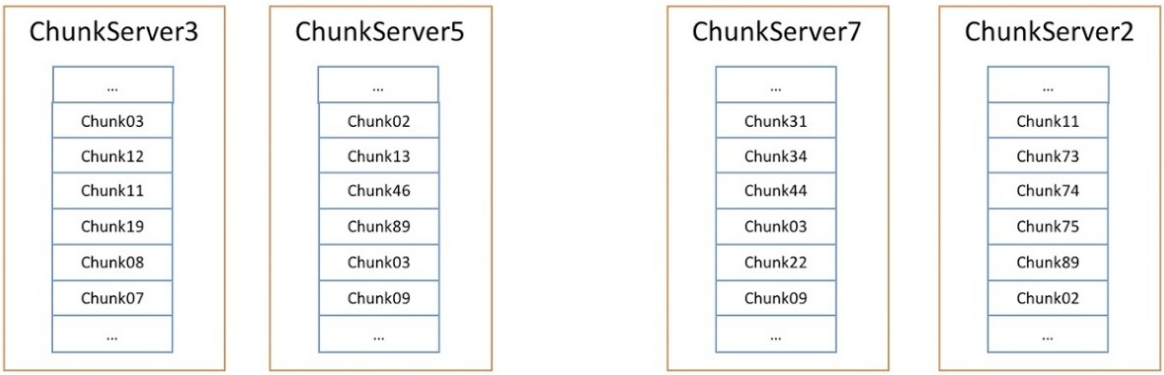




如何应对热点

在Master中有一个热点平衡进程，这个进程会记录每个数据块访问的热度，也可以记录下ChunkServer对应的剩余空间和带宽。当一个Chunk过度繁忙时，我们便可以把他复制到更多的ChunkServer中，这里ChunkServer的选择便可以基于记录下来的对应剩余空间和带宽。

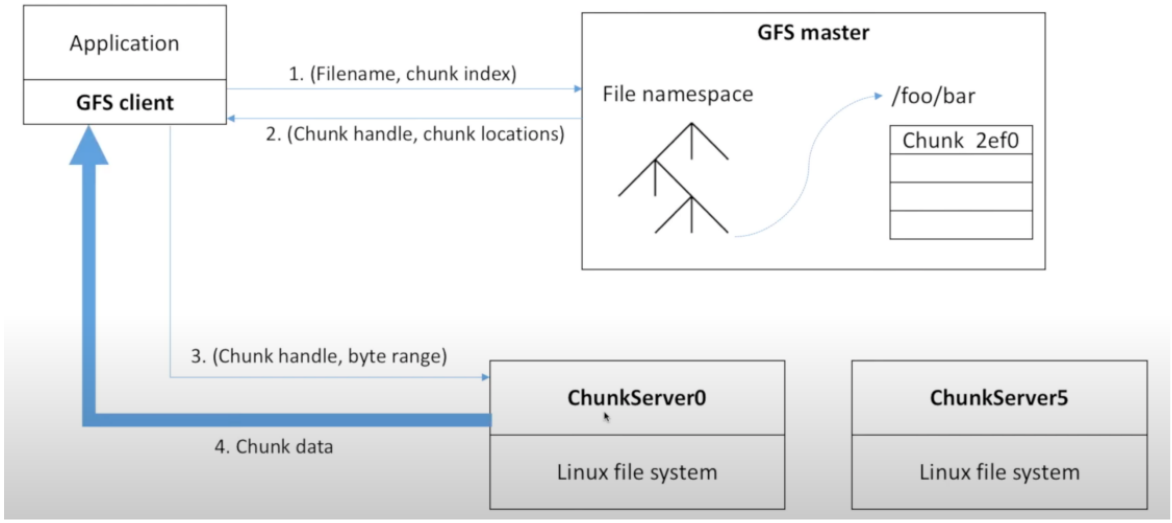




三、读文件与写文件

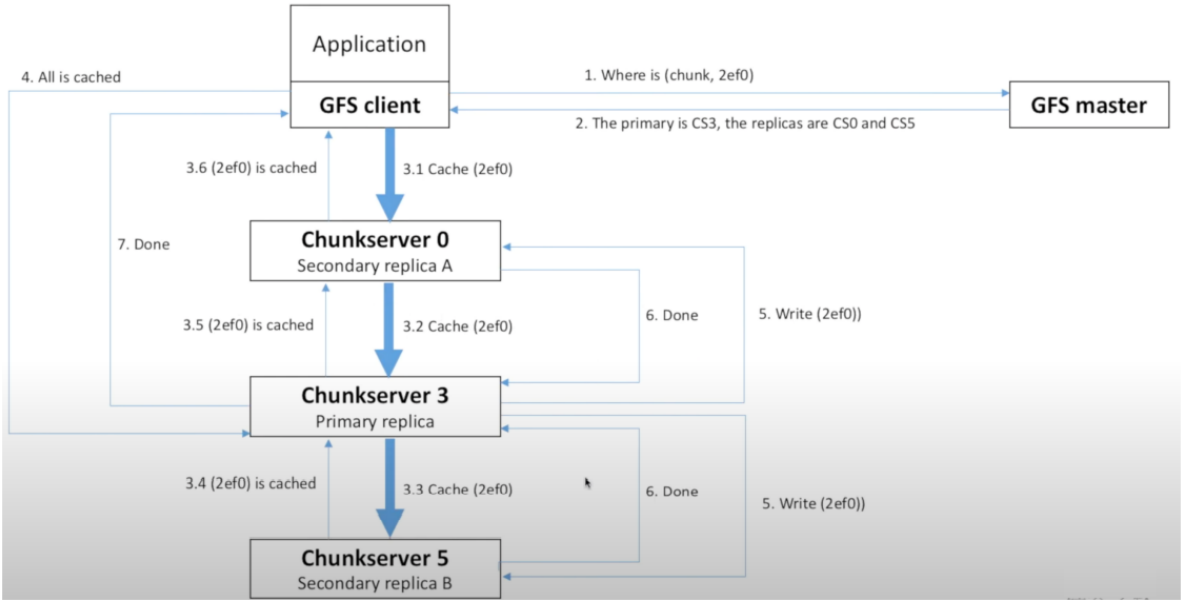
读文件

我们的实际应用在使用GFS时，需要挂载一个GFS client，这个client会告诉GFS master说它需要什么文件，以及对应的chunk index。GFS master收到后会在File namespace中寻找到对应的Metadata，进而寻找到对应的ChunkServer，并返回给client。再由client去对应的ChunkServer去读数据。



写文件

当client要写文件时，同样需要找master询问对应ChunkServer在哪，master会返回对应的几个ChunkServer，同时也会指定一个作为primary Server，而这个primary Server就是用来协调这次写操作的服务器。但Client并不一定会先去连接这个primary Server，而是会去找这几个ChunkServer中离他最近的Server，让服务器之间再串行传输数据。当然，一开始的数据传输只是放在了服务器缓存之中，并没有真正写入。当所有ChunkServer都缓存好了之后，client便会给primary Server发出开始写入的命令，primary Server也会通知其他server开始写入。最后当所有都写完了后，由primary Server通知client。



MapReduce读后感

一、MapReduce的层次

Mapreduce是针对分布式并行计算的一套编程模型。其主要思想是将任务分解然后在多台处理能力较弱的计算节点中同时处理，然后将结果合并从而完成大数据处理。它通过map函数把基于行的输入转化成不同的键值对，再通过reduce函数把这些键值对针对相同的键进行聚合，并在聚合的过程中进行相应的计算。

二、什么是Map和Reduce

什么是Map

map函数：对输入数据进行处理，输入一对key/value，通过map函数生成输出中间值key/value对，MapReduce将有着相同key和value的中间对组合，输给reduce函数

什么是Reduce

reduce函数：接受一个键以及相关的一组值，将这组值进行合并产生一组规模更小的值（通常为一个或零个）

三、MapReduce六大过程

- input
- Split
- Map
- Shuffle
- Reduce
- Finalize

MapReduce会先考虑将数据拆成几份，然后再去分配对应的worker。这里会有一个Master worker，它会作为用户的代理来协调其他worker。读数据的worker会在本地把数据Map完后，写到本地硬盘上。后面的worker就能远程地Shuffle并在本地Reduce，最后将结果写到final file里就是Finalize。

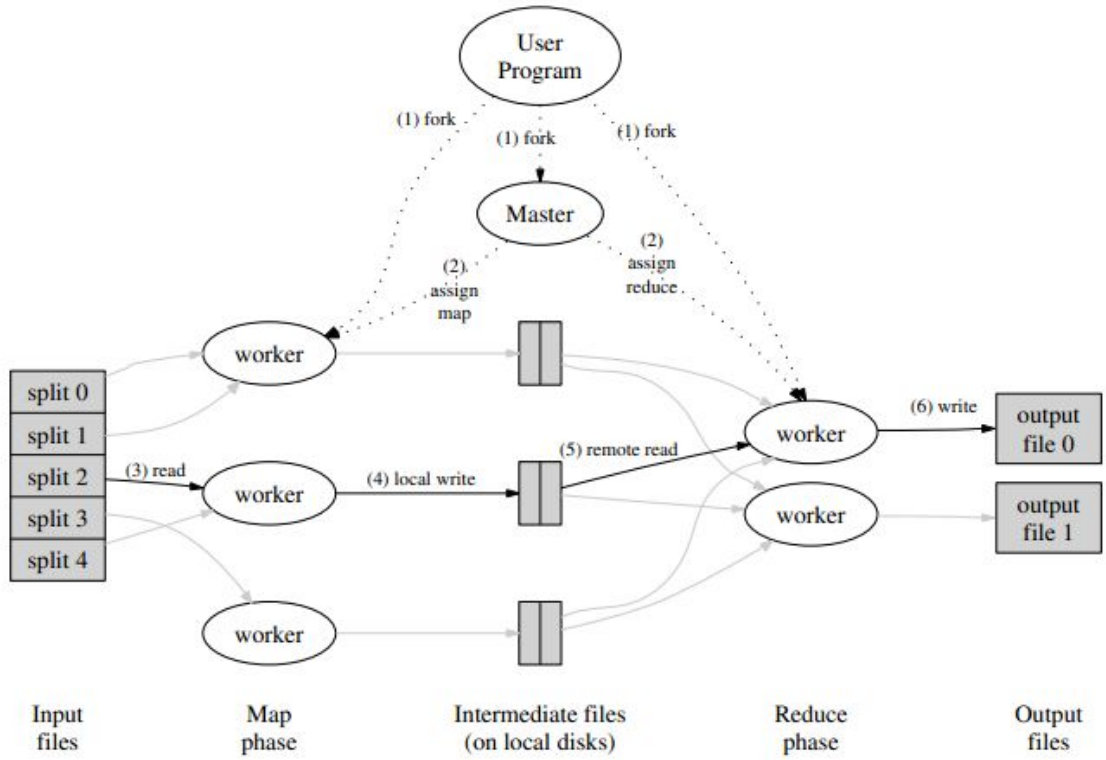


Figure 1: Execution overview

四、示例：统计单词出现数

比如说现在有文档需要统计，文档的每一行是不同的单词组合起来的，这个文档就是Input。

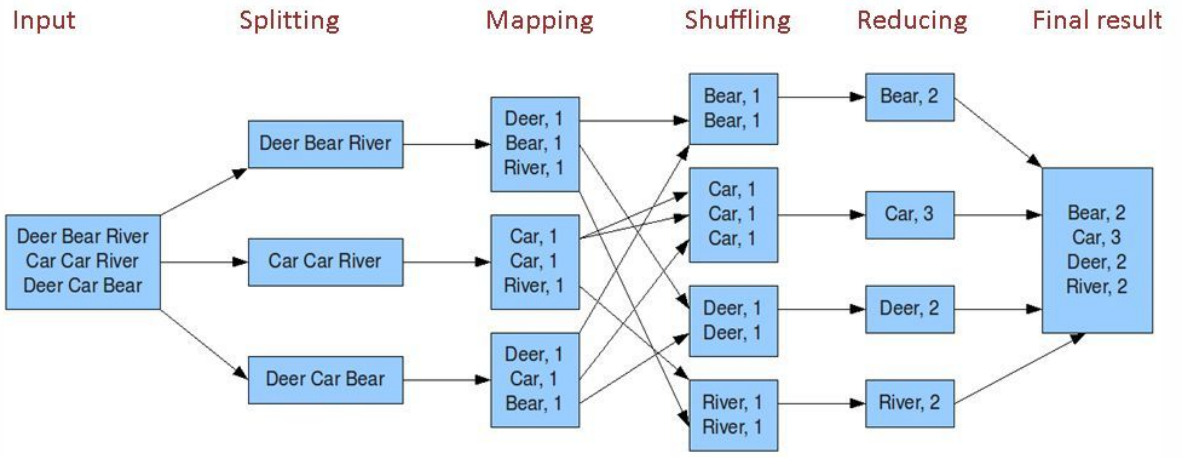
这里就可以有多个worker来去把文档切分成一行一行并拿到每个work手上，这个过程就是Split。

将每个work手上的一行文档拆解成单词和对应的单词数，这个过程就是Map。

再将相同的单词放入不同的“盒子”里，这个过程就是Shuffle。

而Reduce就是把前一步放在一个“盒子”里的单词做一个总和。

最后的Finalize就是将这些总和放到一块。



BigTable读后感

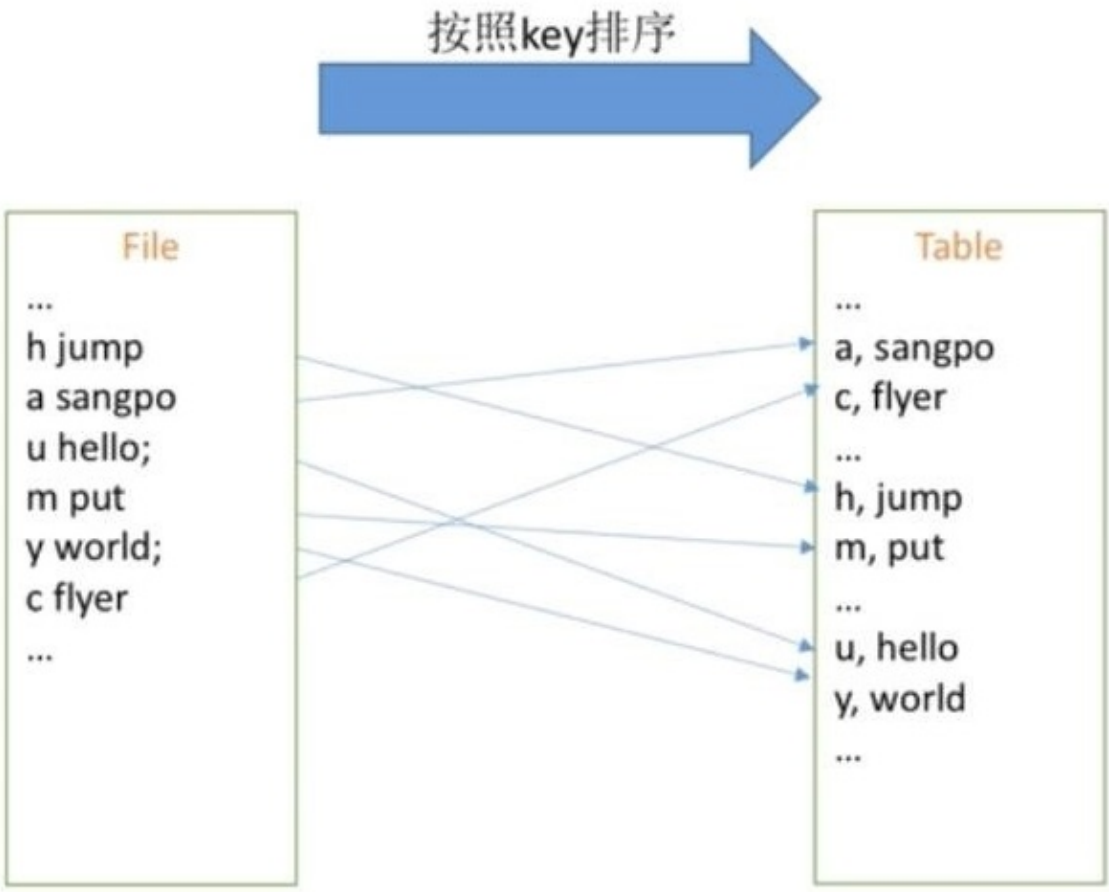
一、什么是BigTable

BigTable 是建立在 GFS 和 MapReduce 之上、为管理大规模结构化数据而设计的分布式存储系统，它可以扩展到PB级数据和上千台服务器。

二、设计概括

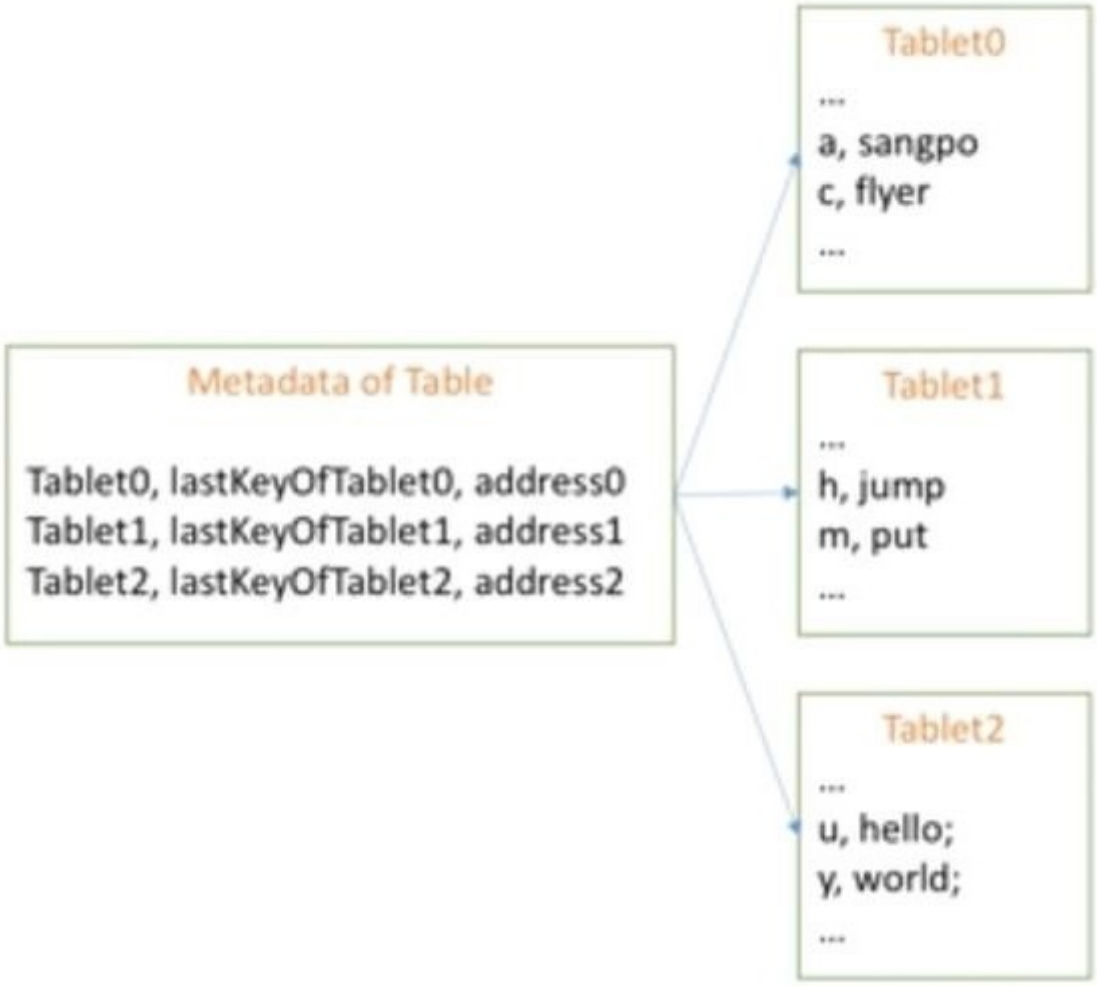
如何保存一个文件表

对文件内的很多数据，我们需要找到他的一个Key，然后按照这个Key排序。之后我们就可 以利用诸如二分查找来在表中查找文件了。



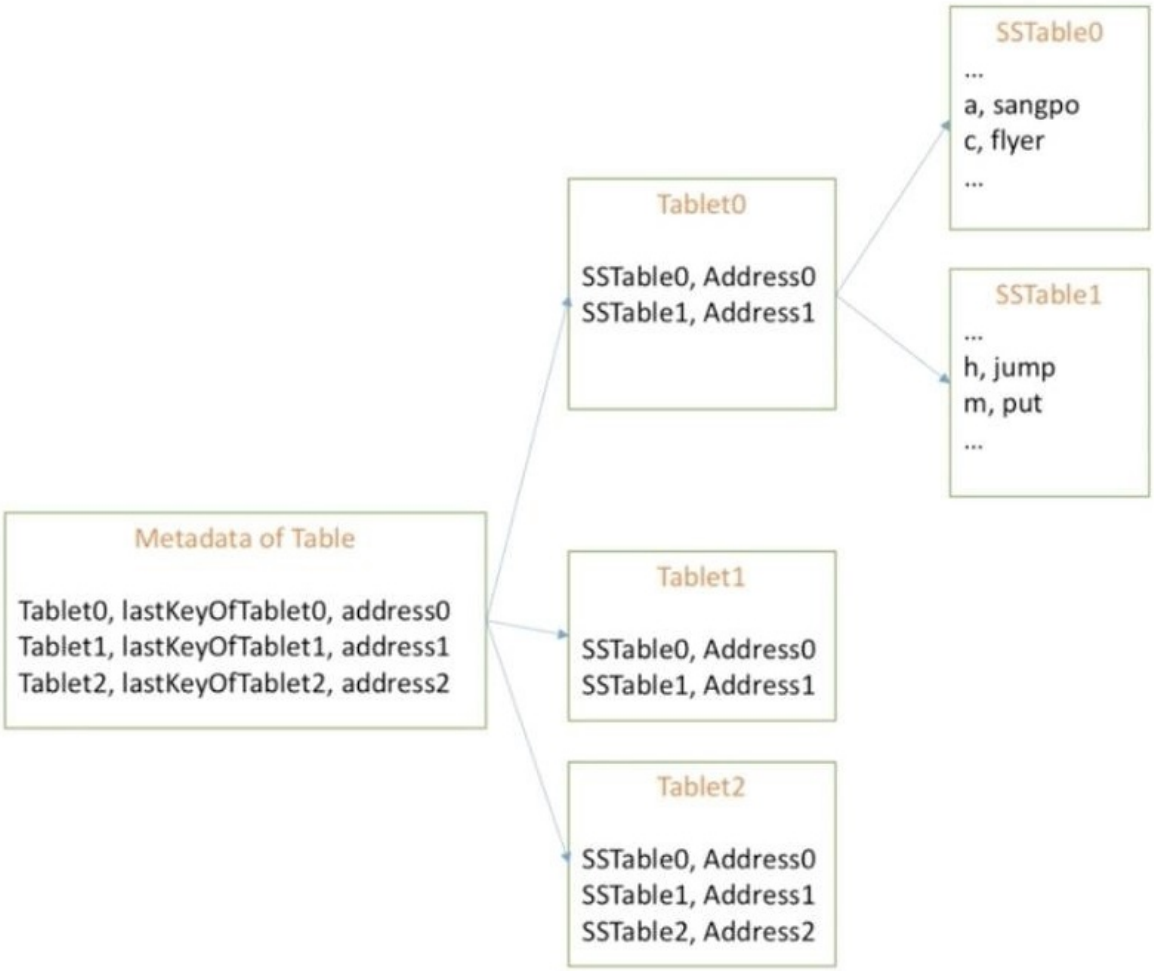
如何保存一个很大的表

把一个大表拆成很多小表，这样每个小表的内部是一些排好序的串。然后我们可以用Metadata，也就是元数据的方法保存每个小表的存储位置。这样就相当于大表拆成了很多小表，小表是排序的关键词。



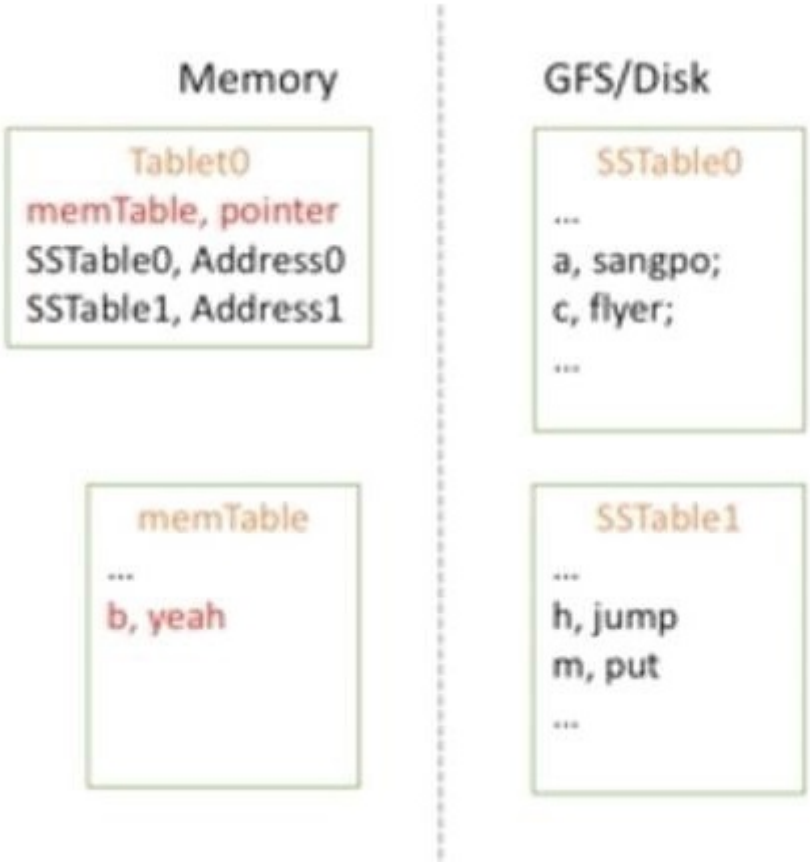
如何保存一个超大表

思路就是把表再拆一层，大表变成小表，小表变成小小表。这样就能保证一定能存储下这些很大的表。



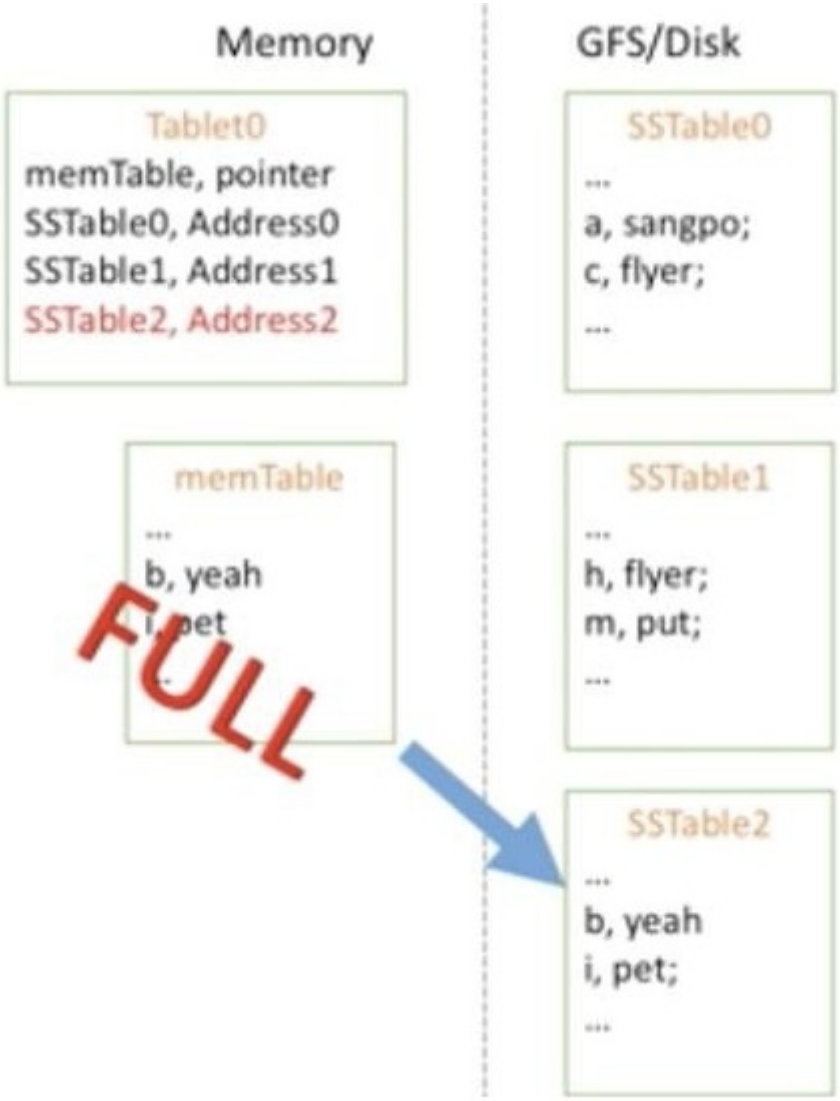
如何写数据

假设说我们需要添加一个K-V对，在整个系统架构中，我们会有内存和硬盘两部分，这里的硬盘也可以认为是底层的GFS（Google file system），在内存中我们可以存放小小表的位置。当我们写数据的时候，并不是直接就写入硬盘中，为了加快速度，我们会在内存中建一个表，将这个K-V对插入到这个表中。这里的好处就是如果我们是直接插入硬盘中的小小表，那么每一次插入的时候都需要对硬盘中的小小表进行排序，而内存中因为是随机存储所以排序是较容易的。所以此时一个tablet就等于内存表+小小表的集合



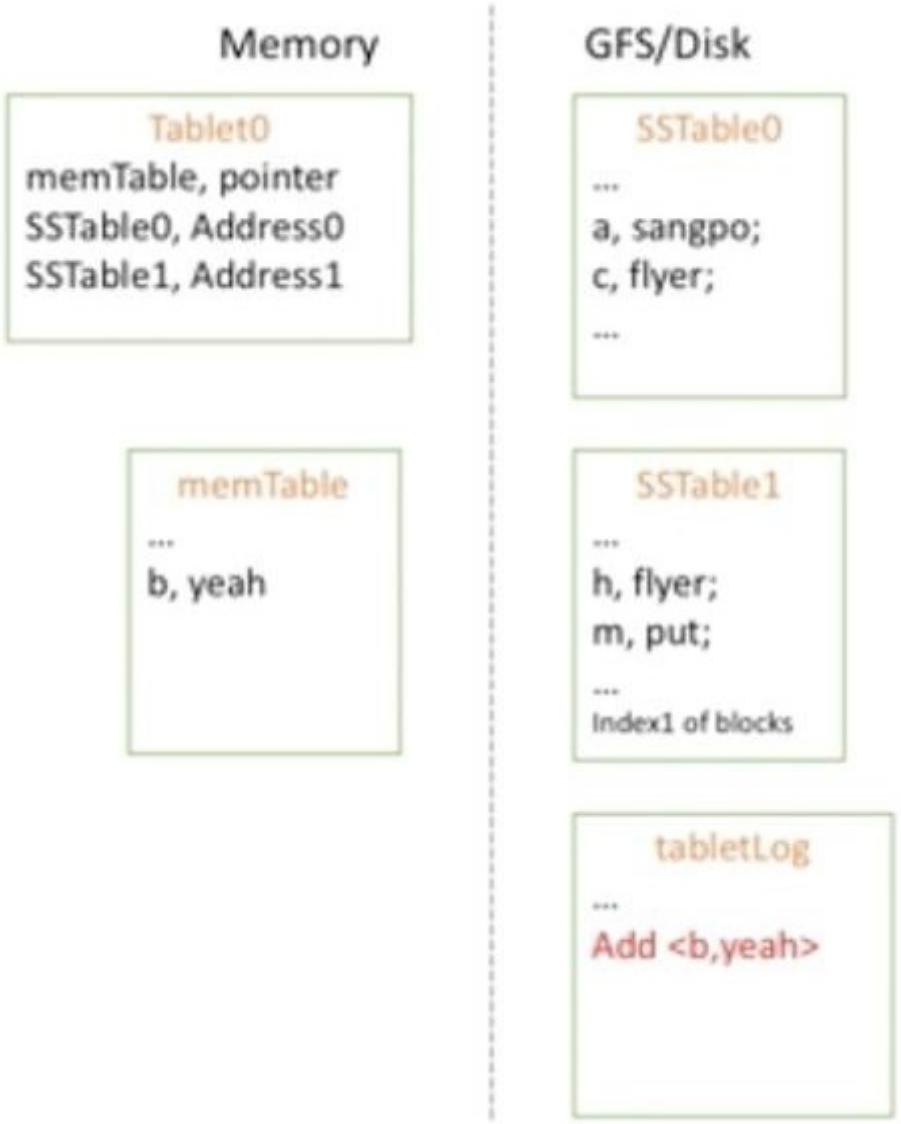
内存表过大该如何

这里会给内存表设置一个大小上限，一旦内存表满了就会把内存表，重新写成一个新的小小表，进行持久性保存，也就是将内存表导入硬盘。



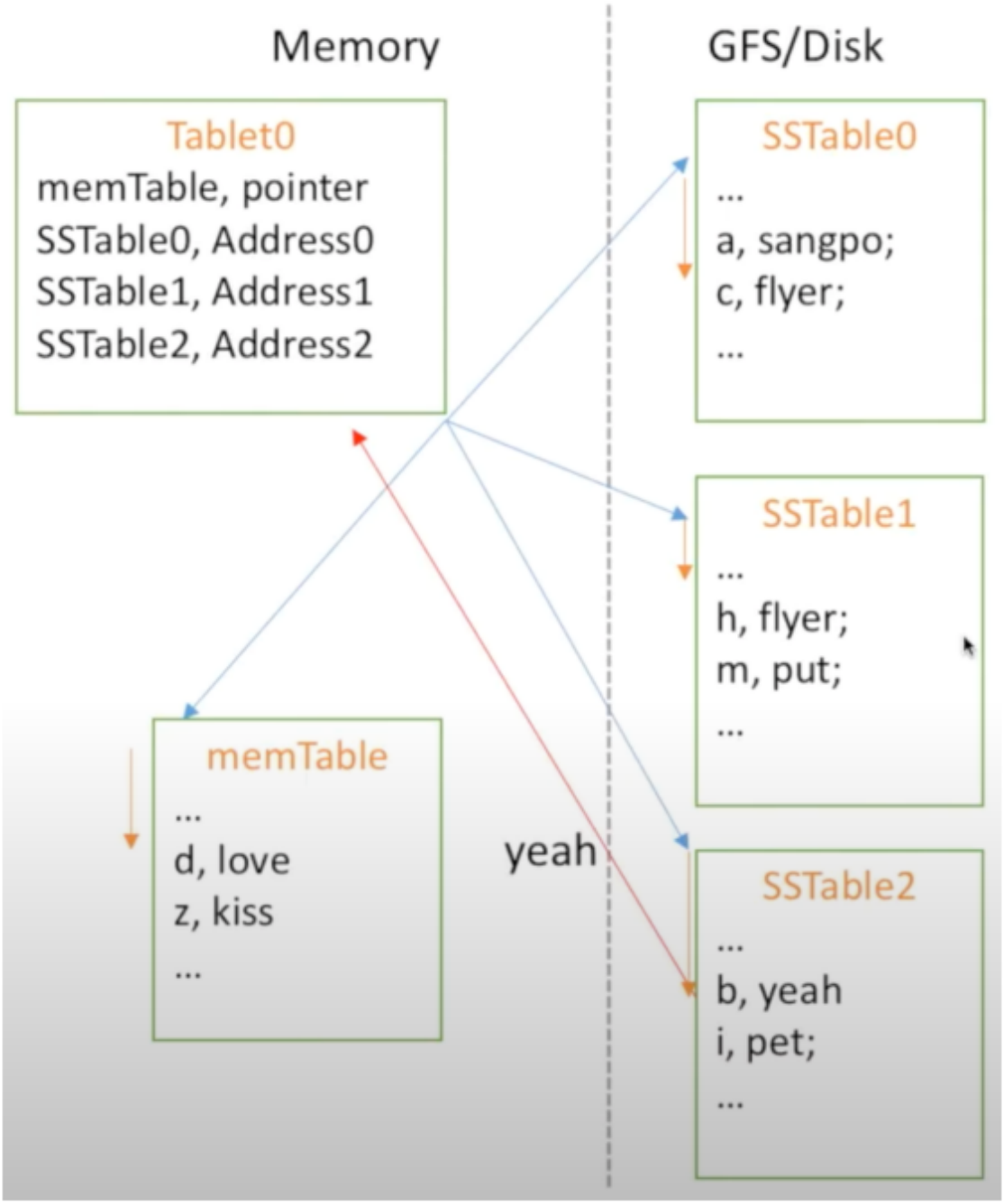
如何避免内存表数据丢失

当我们添加的时候，我们首先在硬盘上写下一个添加的Log。这样再在内存中写就不会发生丢失数据的情况。



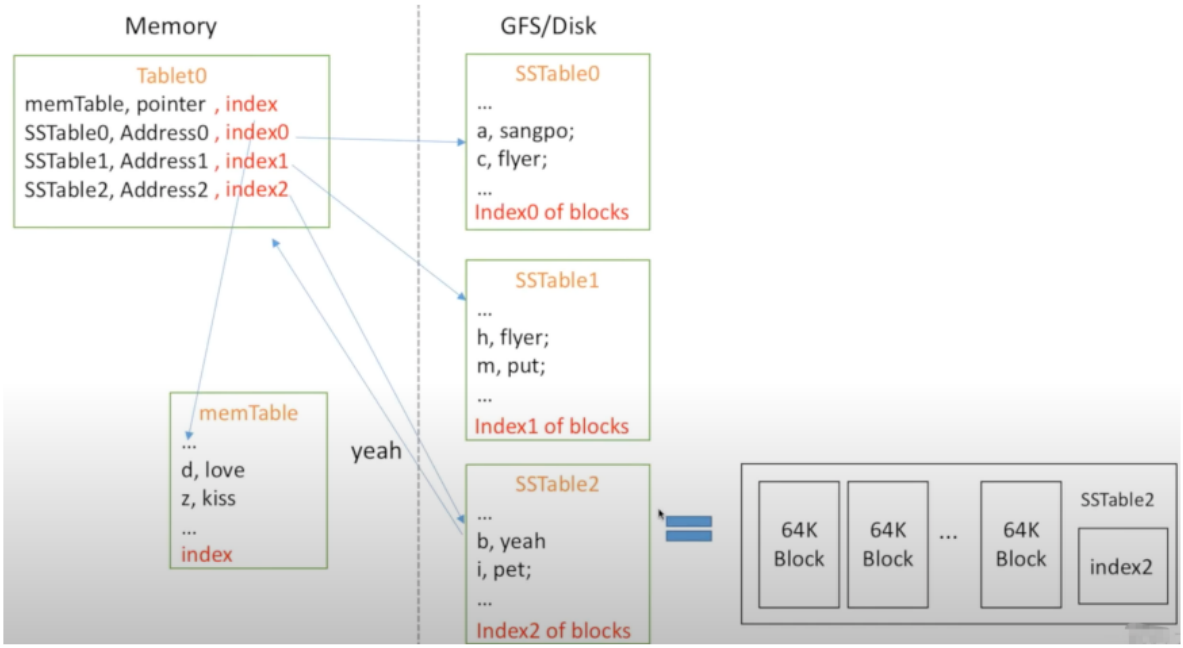
如何读数据

由于之前的写策略导致小小表的内部是有序的，但小小表之间是无序的，所以每次查找需要查找所有的小小表和内存表，性能是很差的。



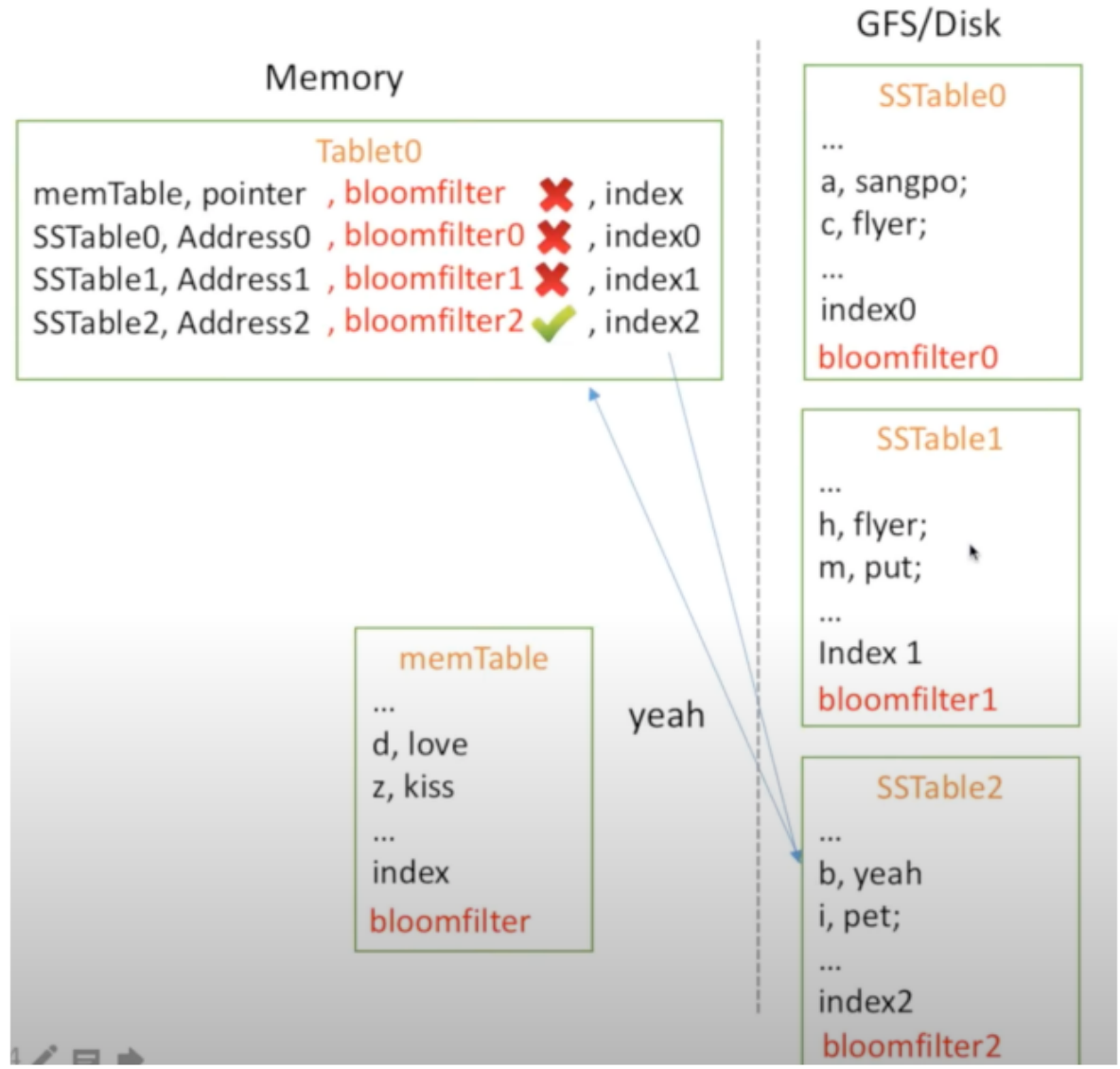
如何加速读数据

最好的方法就是建立索引。实际上，我们在写入每个小小表的时候，它往往是固化在那里的。所以我们可以固化一个索引在这里，然后这些索引可以预先加载到内存里，这样可以减少我们遍历的次数和时间，所以性能更优。



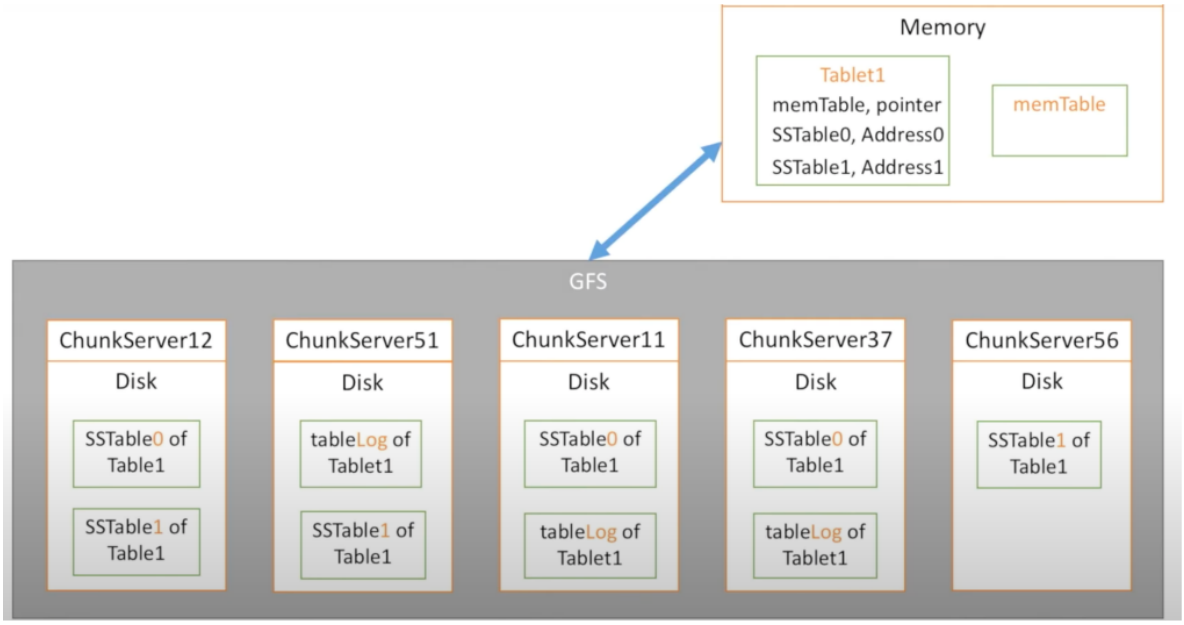
如何继续加速读数据

之前的方法依旧进行了大量的硬盘遍历，这里的话我们可以使用布隆过滤器（bloom filter），它能以很少的花费以一个很大的正确率告诉我们查找的东西在不在一个集合里面。所以我们每次查找之前先用bloom filter先过滤一下就行了，只有当有可能在的时候才去遍历。



如何将表存入GFS

根据我们之前的概括，这里其实就是将小小表和日志放入到GFS的ChunkServer中，并且也会对应有副本。



架构

Client上会有对应的Client Library来提供对应的操作库函数等，同时也会有Chubby来提供表的metadata以及获取锁服务。之后，Client就能到特定的Tablet Server进行读写。Tablet Server之上也有一个Master负责处理metadata以及处理负载均衡，例如去协调放多少Tablet到某个Server上。更底层来说，GFS就提供了存放小小表和日志的能力，另外还有诸如Cluster Scheduling System来监控整个系统