Google MapReduce基本架构

Google的三驾马车

- SOSP2003 The Google File System
- OSDI2004 MapReduce: simplified data processing on large clusters
- OSDI2006 Bigtable: a distributed storage system for structured data

GFS

MapReduce

Bigtable

摘要

- Google MapReduce的基本工作原理
- 分布式文件系统GFS的基本工作原理
- 分布式结构化数据表BigTable的基本工作原理

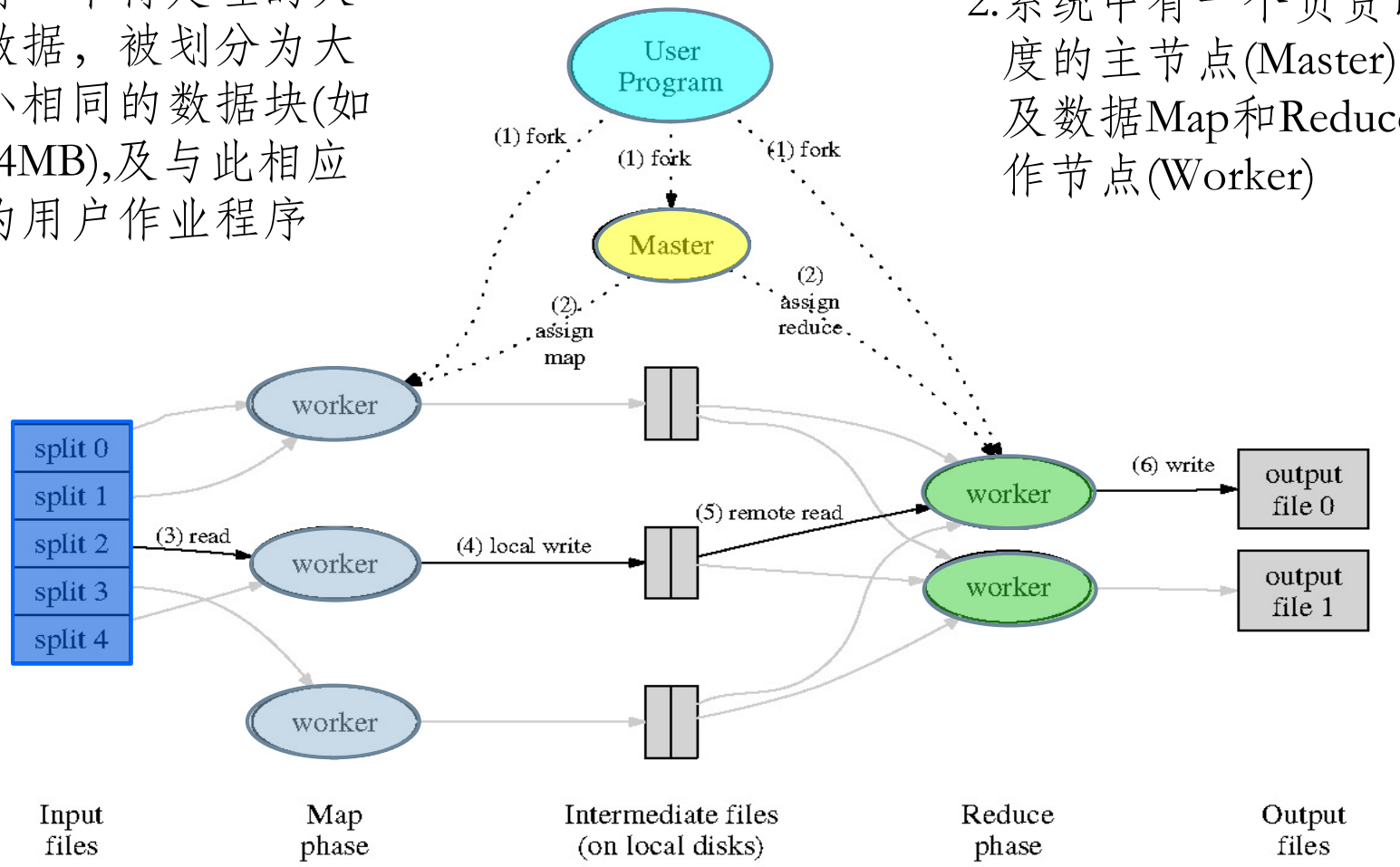
摘要

- Google MapReduce的基本工作原理
- 分布式文件系统GFS的基本工作原理
- 分布式结构化数据表BigTable的基本工作原理

Google MapReduce并行处理的基本过程

1.有一个待处理的大数据，被划分为大小相同的数据块(如64MB),及与此相应的用户作业程序

2.系统中有一个负责调度的主节点(Master),以及数据Map和Reduce工作节点(Worker)



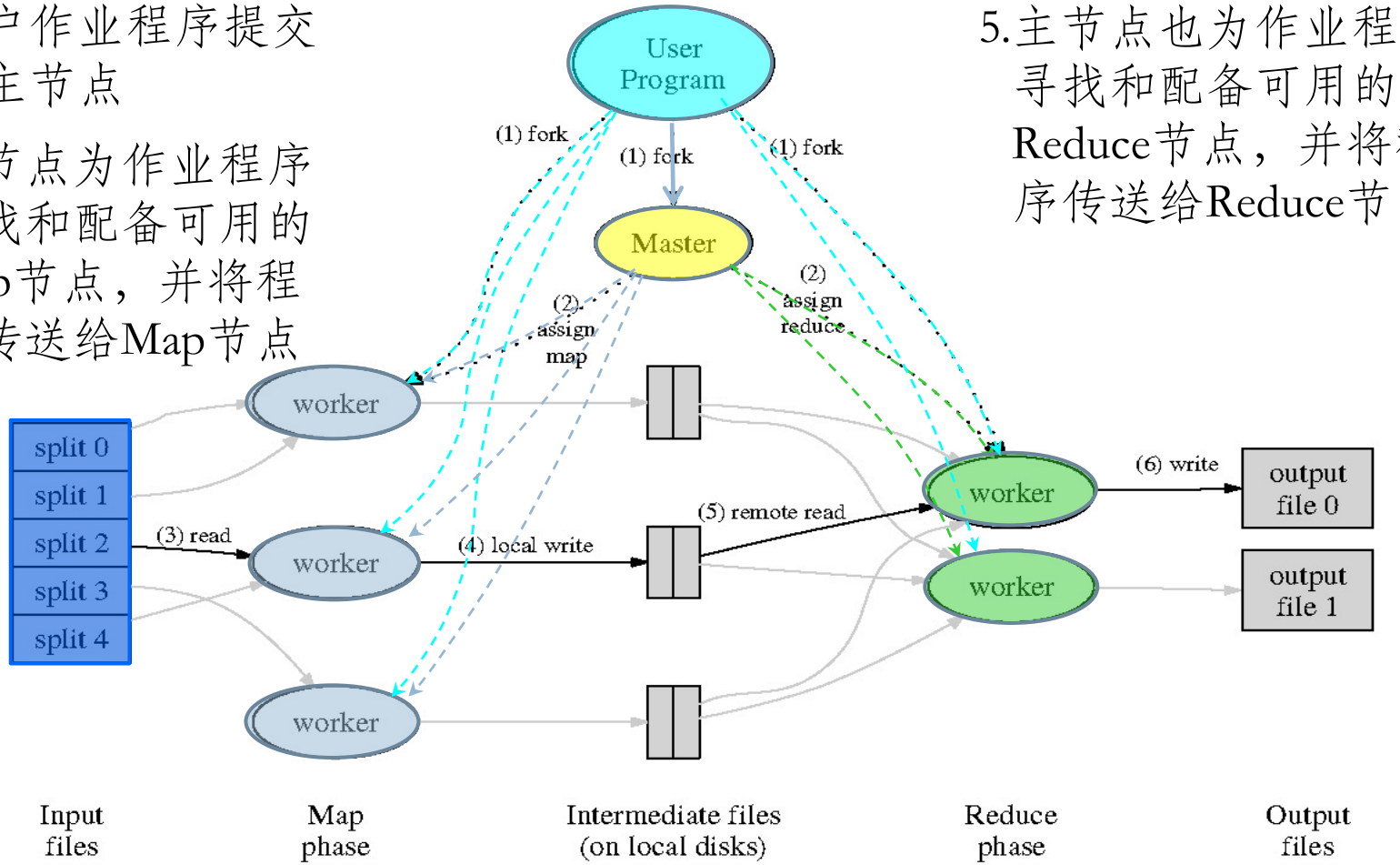
Cite from Dean and Ghemawat (OSDI 2004)

Google MapReduce并行处理的基本过程

3.用户作业程序提交给主节点

4.主节点为作业程序寻找和配备可用的Map节点，并将程序传送给Map节点

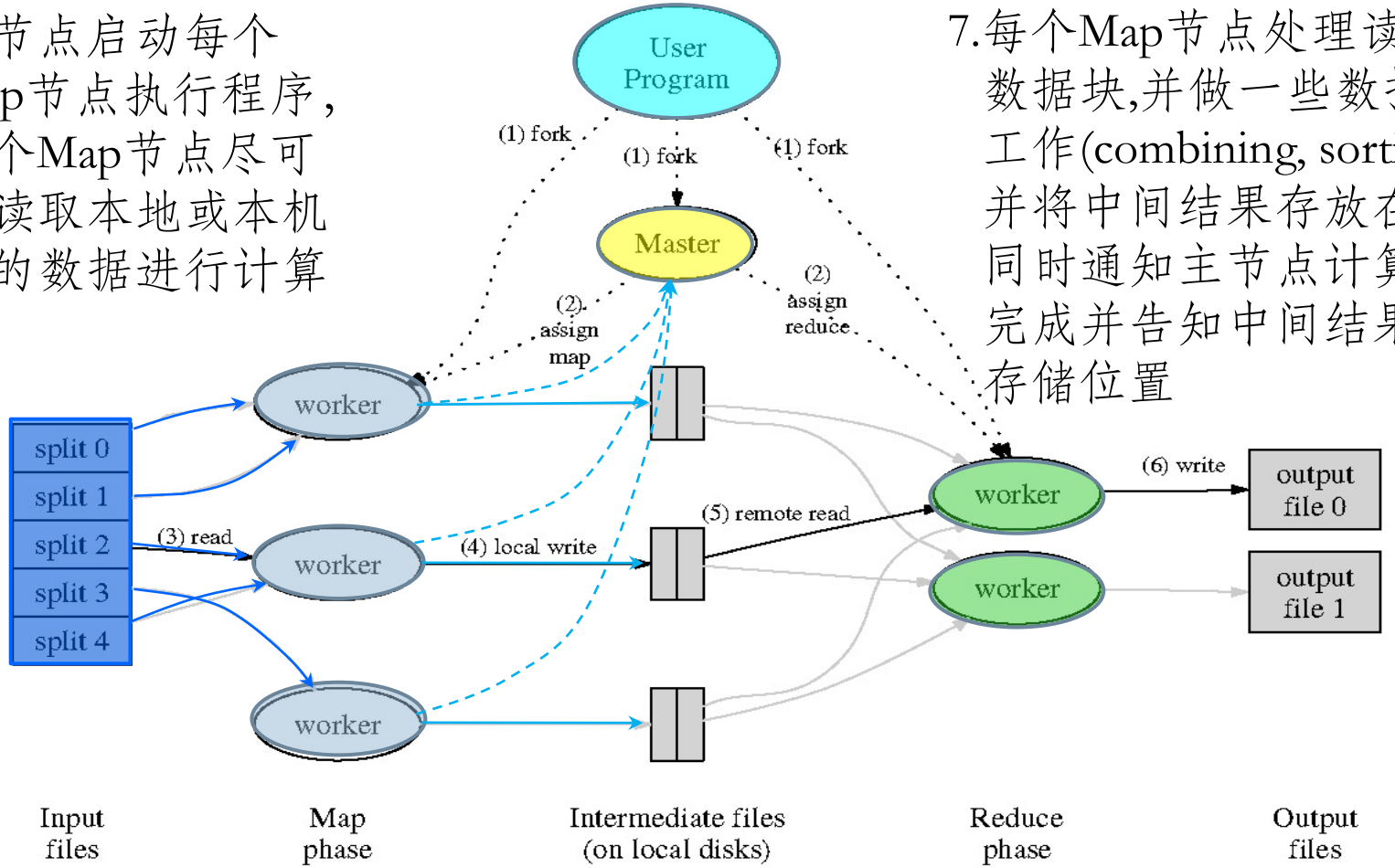
5.主节点也为作业程序寻找和配备可用的Reduce节点，并将程序传送给Reduce节点



Cite from Dean and Ghemawat (OSDI 2004)

Google MapReduce并行处理的基本过程

6.主节点启动每个Map节点执行程序，每个Map节点尽可能读取本地或本机架的数据进行计算



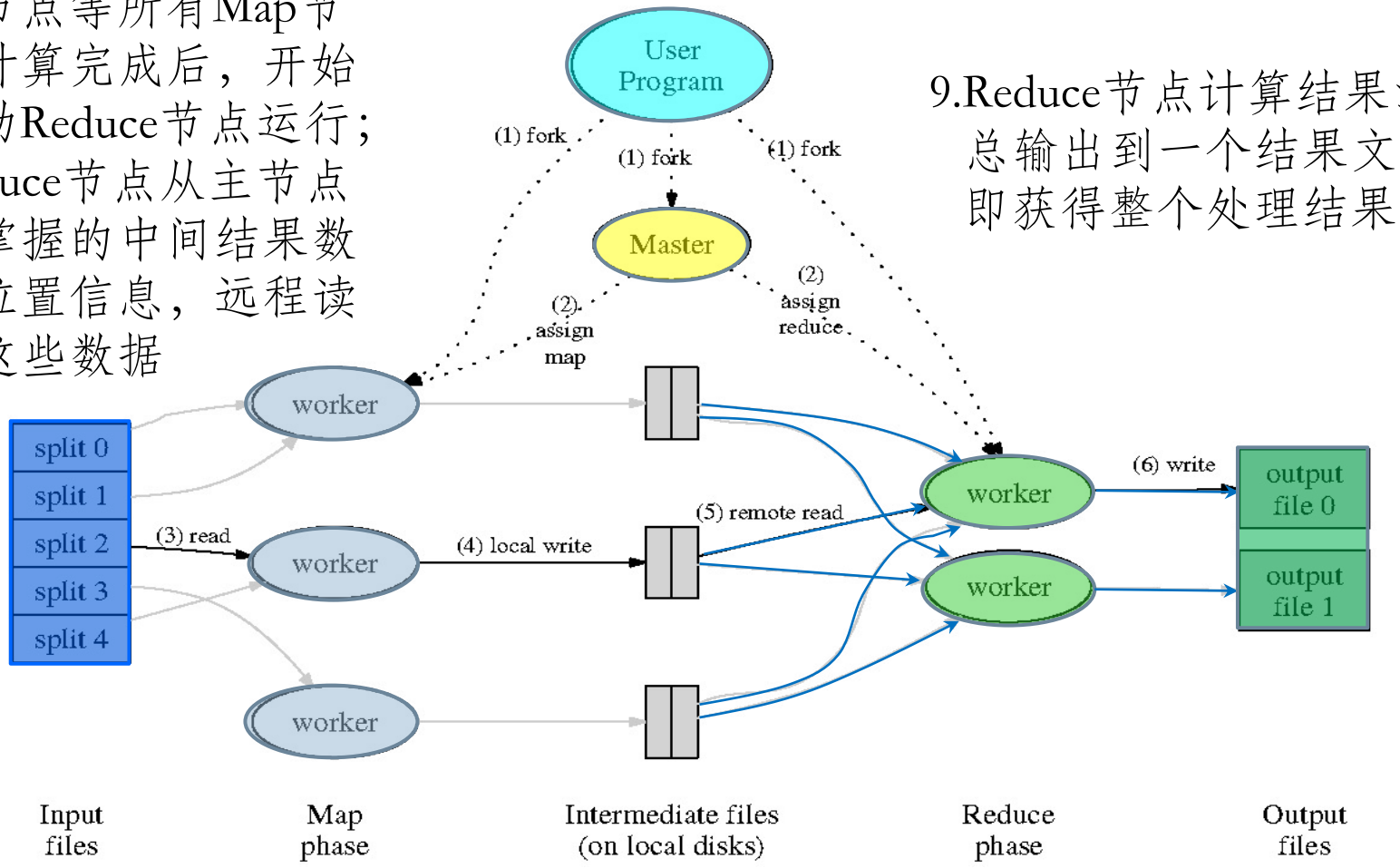
7.每个Map节点处理读取的数据块,并做一些数据整理工作(combining, sorting等)并将中间结果存放在本地;同时通知主节点计算任务完成并告知中间结果数据存储位置

Cite from Dean and Ghemawat (OSDI 2004)

Google MapReduce并行处理的基本过程

8.主节点等所有Map节点计算完成后，开始启动Reduce节点运行；Reduce节点从主节点所掌握的中间结果数据位置信息，远程读取这些数据

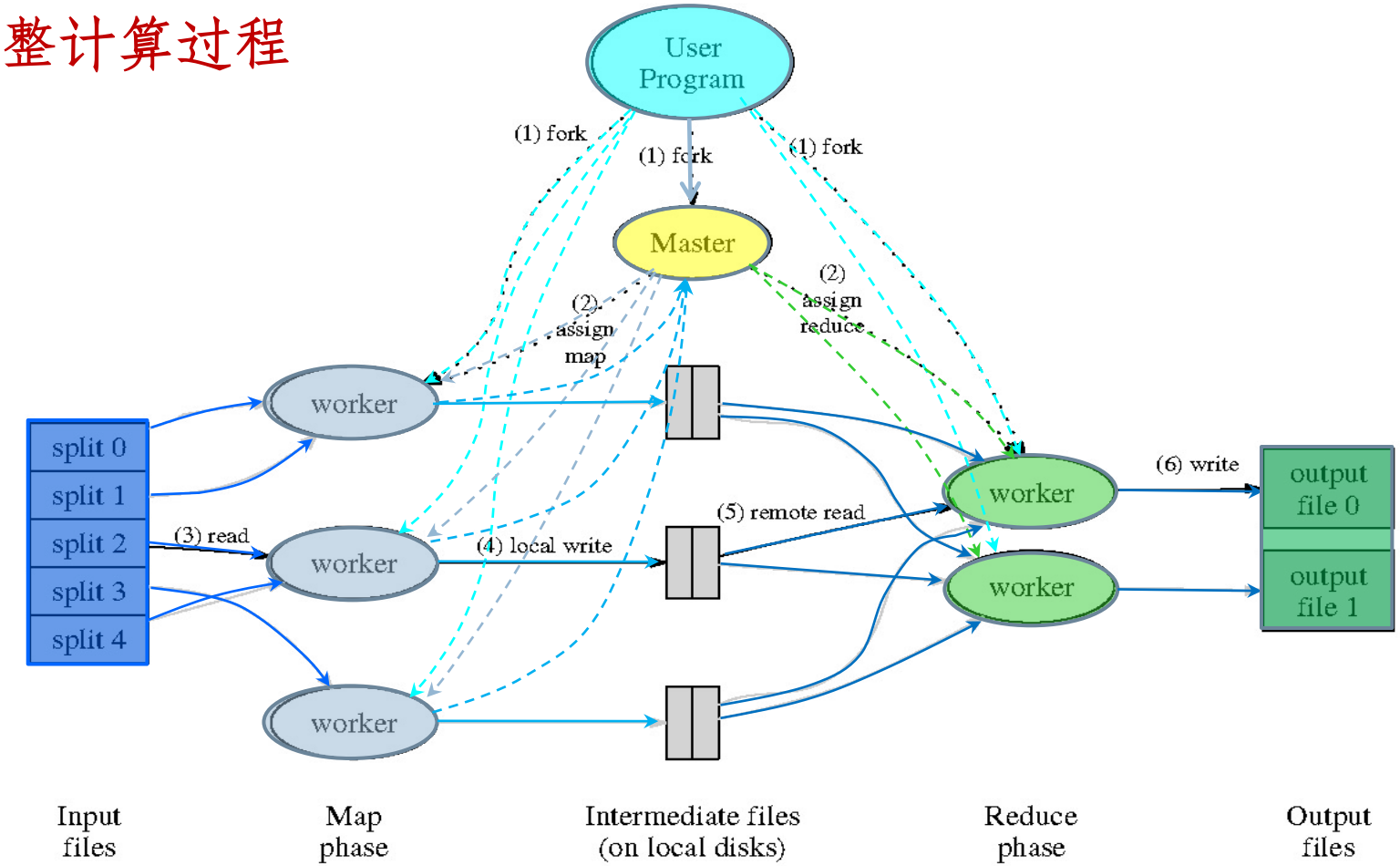
9.Reduce节点计算结果汇总输出到一个结果文件即获得整个处理结果



Cite from Dean and Ghemawat (OSDI 2004)

Google MapReduce并行处理的基本过程

完整计算过程



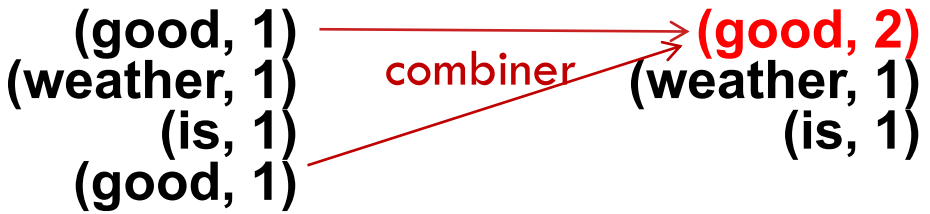
Cite from Dean and Ghemawat (OSDI 2004)

失效处理

- 主节点失效
  - 主节点中会周期性地设置检查点(checkpoint)，检查整个计算作业的执行情况，一旦某个任务失效，可以从最近有效的检查点开始重新执行，避免从头开始计算的时间浪费。
  - 如果只有一个Master，它不太可能失败；因此，如果Master失败，将中止MapReduce计算。
- 工作节点失效
  - 工作节点失效是很普遍发生的，主节点会周期性地给工作节点发送检测命令，如果工作节点没有回应，这认为该工作节点失效，主节点将终止该工作节点的任务并把失效的任务重新调度到其它工作节点上重新执行。

带宽优化

- 问题
  - 大量的键值对数据在传送给Reduce节点时会引起较大的通信带宽开销。
- 解决方案
  - 每个Map节点处理完成的中间键值对将由combiner做一个合并压缩，即把那些键名相同的键值对归并为一个键名下的一组数值。



计算优化

问题

- Reduce节点必须要等到所有Map节点计算结束才能开始执行，因此，如果有一个计算量大、或者由于某个问题导致很慢结束的Map节点，则会成为严重的“拖后腿者”。

解决方案

- 把一个Map计算任务让多个Map节点同时做，取最快完成者的计算结果。

根据Google的测试，使用了这个冗余Map节点计算方法以后，计算任务性能提高40%多！

用数据分区解决数据相关性问题

- 问题
  - 一个Reduce节点上的计算数据可能会来自多个Map节点，因此，为了在进入Reduce节点计算之前，需要把属于一个Reduce节点的数据归并到一起。
- 解决方案
  - 在Map阶段进行了Combining以后，可以根据一定的策略对Map输出的中间结果进行分区(partitioning)，这样即可解决以上数据相关性问题避免Reduce计算过程中的数据通信。
    - 例如：有一个巨大的数组，其最终结果需要排序，每个Map节点数据处理好后，为了避免在每个Reduce节点本地排序完成后还需要进行全局排序，我们可以使用一个分区策略如:(d%R)，d为数据大小，R为Reduce节点的个数，则可根据数据的大小将其划分到指定数据范围的Reduce节点上，每个Reduce将本地数据排好序后即为最终结果。

摘要

- Google MapReduce的基本工作原理
- 分布式文件系统GFS的基本工作原理
- 分布式结构化数据表BigTable的基本工作原理

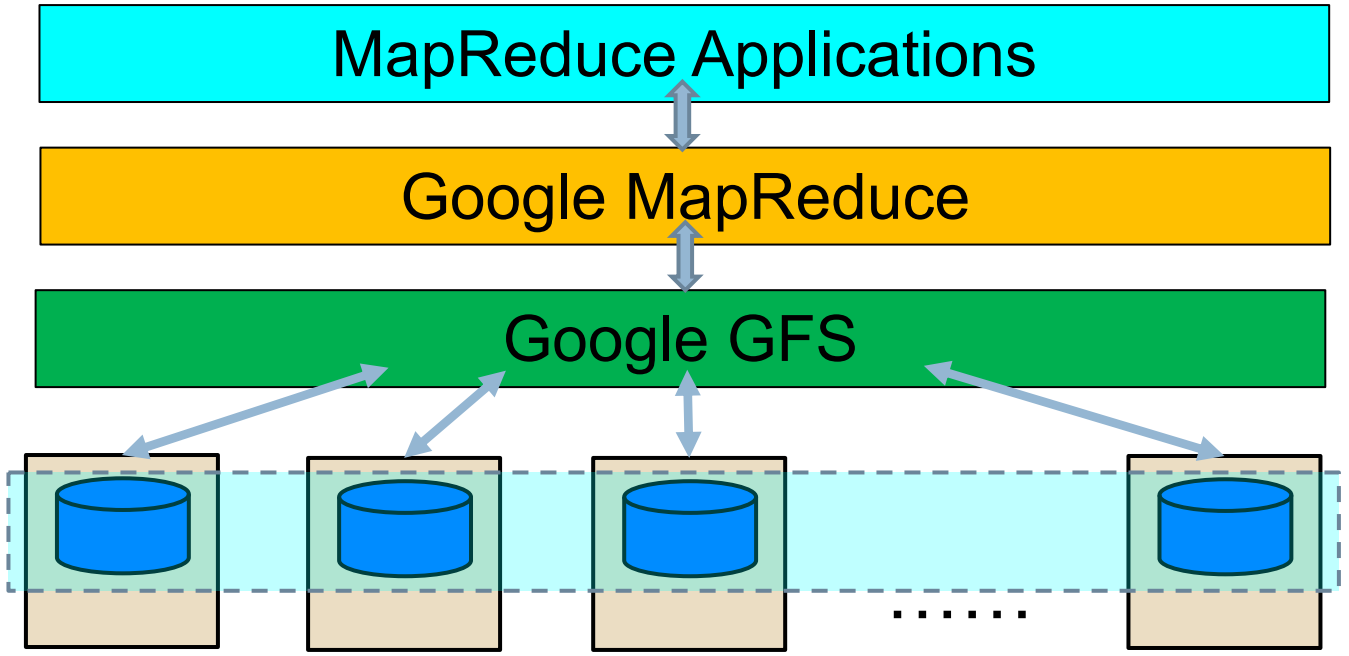
分布式文件系统GFS的工作原理

- 海量数据怎么存储？数据存储可靠性怎么解决？
- 主流的分布文件系统有：
  - RedHat的GFS
  - IBM的GPFS
  - Sun的Lustre等
- 主要用于对硬件设施要求很高的高性能计算或大型数据中心；
- 价格昂贵且缺少完整的数据存储容错解决方案
- 如Lustre只对元数据管理提供容错处理，但对于具体的分布存储节点，可靠性完全依赖于这些分布节点采用RAID或存储区域网(SAN)技术提供容错，一旦分布节点失效，数据就无法恢复。

Google GFS的基本设计原则

- Google GFS是一个基于分布式集群的大型分布式文件系统，为MapReduce计算框架提供数据存储和数据可靠性支撑；
- GFS是一个构建在分布节点本地文件系统之上的一个逻辑上文件系统，它将数据存储在物理上分布的每个节点上，但通过GFS将整个数据形成一个逻辑上整体的文件。

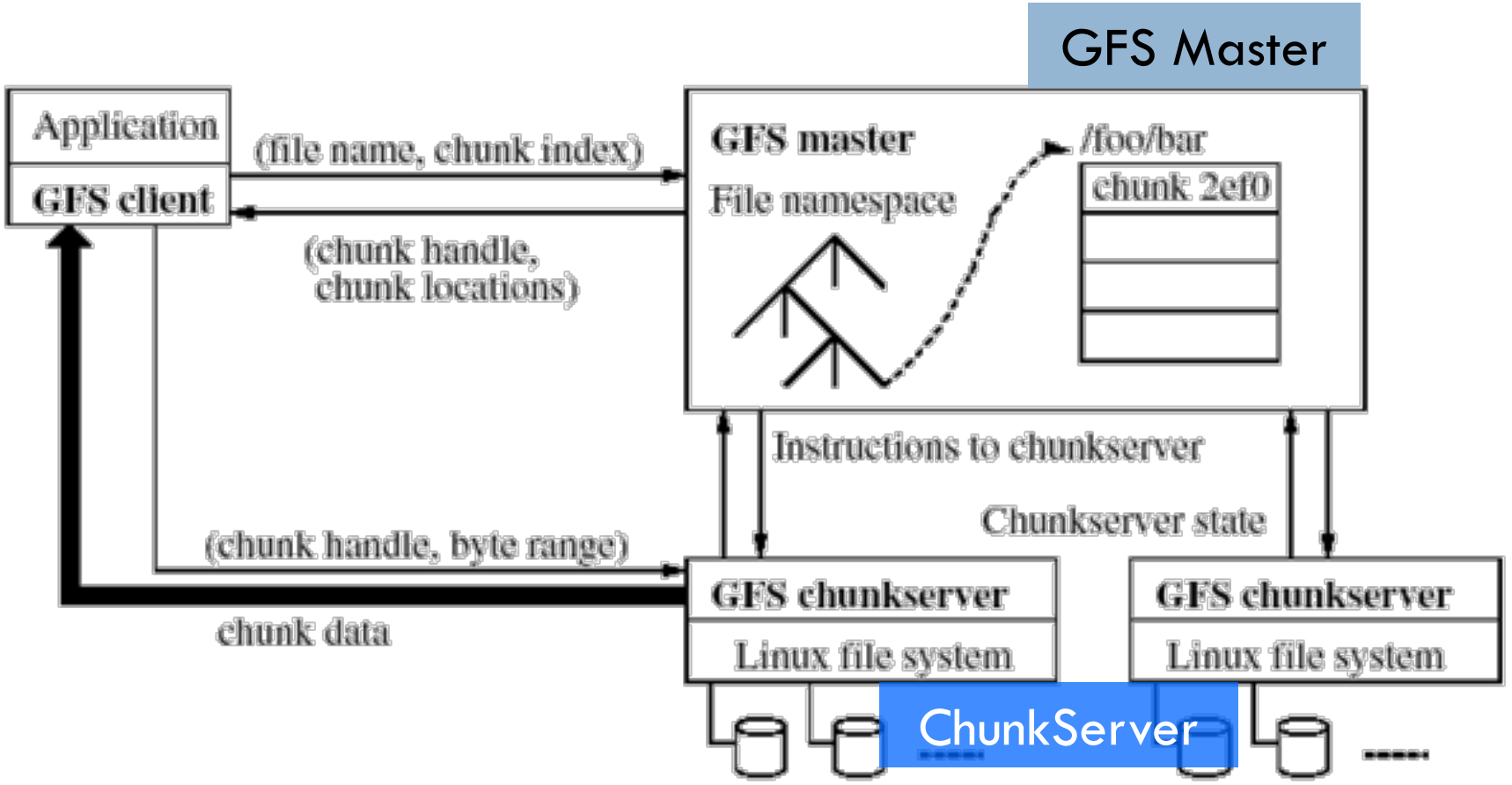
Google GFS的基本设计原则



Google GFS的基本设计原则

- 廉价本地磁盘分布存储
  - 各节点本地分布式存储数据，优点是不需要采用价格较贵的集中式磁盘阵列，容量可随节点数增加自动增加
- 多数据自动备份解决可靠性
  - 采用廉价的普通磁盘，把磁盘数据出错视为常态，用自动多数据备份存储解决数据存储可靠性问题
- 为上层的MapReduce计算框架提供支撑
  - GFS作为向上层MapReduce执行框架的底层数据存储支撑，负责处理所有的数据自动存储和容错处理，因而上层框架不需要考虑底层的数据存储和数据容错问题

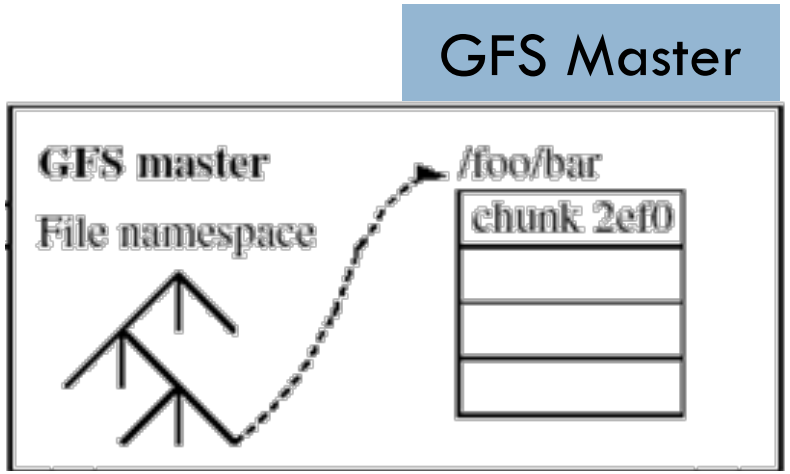
Google GFS的基本架构和工作原理



Cite from Ghemawat et al. (SOSP 2003)

Google GFS的基本架构和工作原理

GFS Master



Master上保存了GFS文件系统的三种元数据：

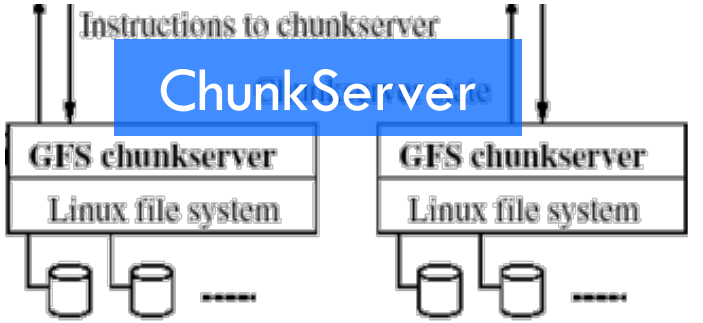
- 命名空间(Name Space)即整个分布式文件系统的目录结构
- Chunk与文件名的映射表
- Chunk副本的位置信息，每一个Chunk默认有3个副本

- 前两种元数据可通过操作日志提供容错处理能力；
- 第3个元数据直接保存在ChunkServer上，Master启动或Chunk Server注册时自动完成在Chunk Server上元数据的生成；
- 因此，当Master失效时，只要ChunkServer数据保存完好，可迅速恢复Master上的元数据。

Google GFS的基本架构和工作原理

GFS ChunkServer： 即用来保存大量实际数据的数据服务器。

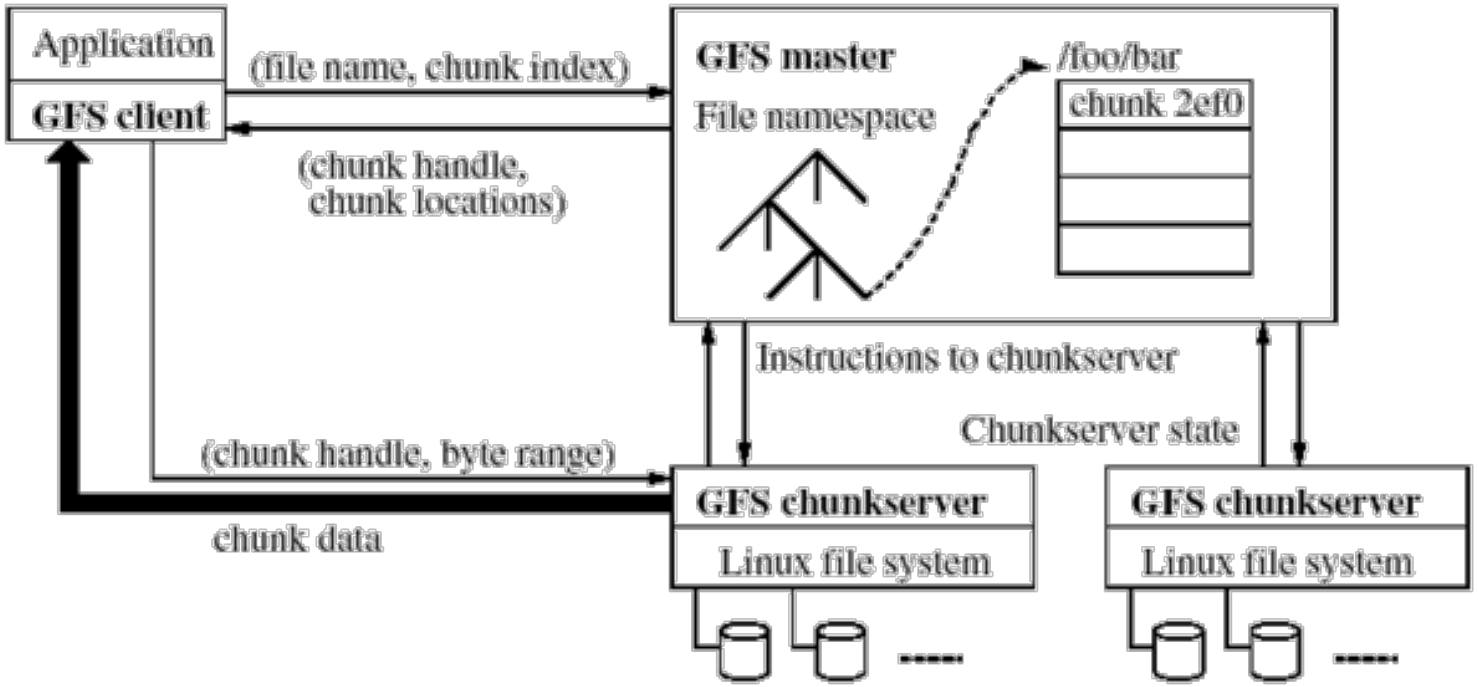
- GFS中每个数据块划分缺省为64MB
- 每个数据块会分别在3个(缺省情况下)不同的地方复制副本；
- 对每一个数据块，仅当3个副本都更新成功时，才认为数据保存成功。



- 当某个副本失效时，Master会自动将正确的副本数据进行复制以保证足够的副本数；
- GFS上存储的数据块副本，在物理上以一个本地的Linux操作系统的文件形式存储，每一个数据块再划分为64KB的子块，每个子块有一个32位的校验和，读数据时会检查校验和以保证使用未失效的数据。

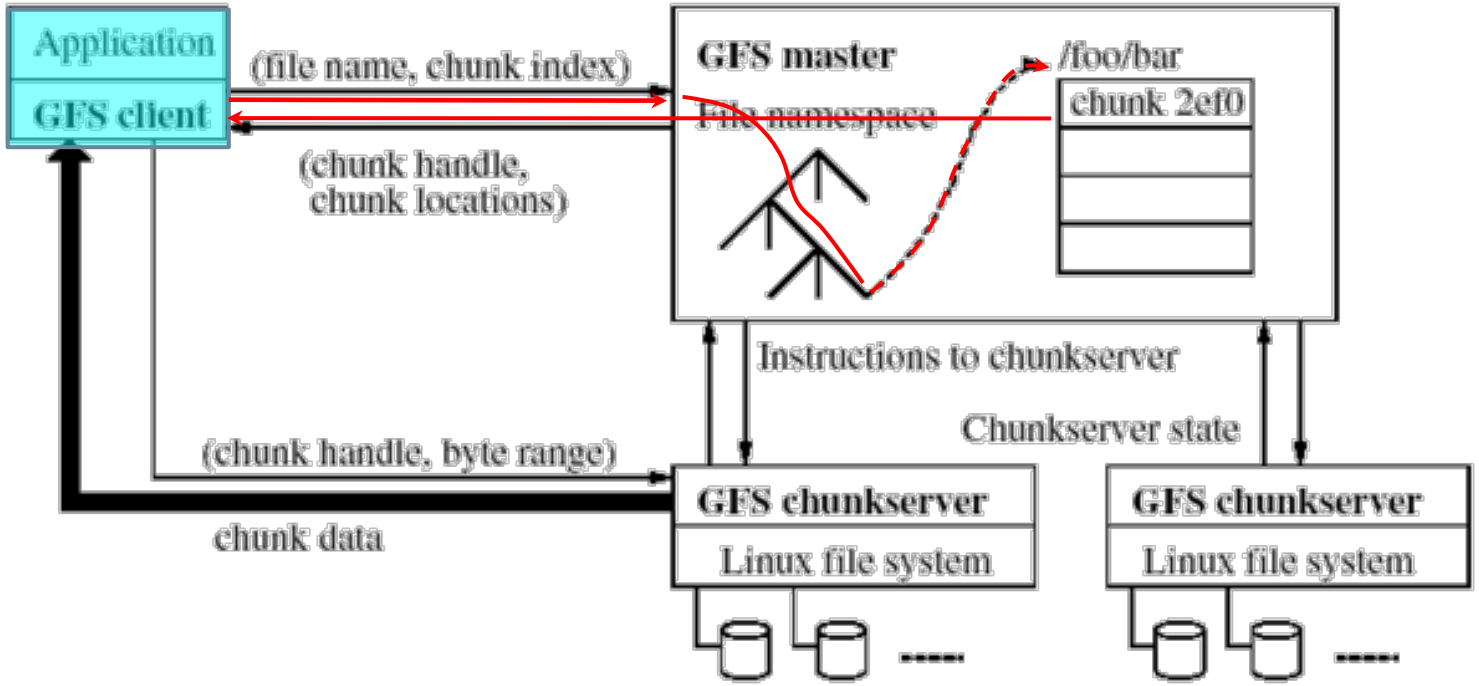
Google GFS的基本架构和工作原理

- 数据访问工作过程
  - 1.在程序运行前，数据已经存储在GFS文件系统中；程序运行时应用程序会告诉GFS Server所要访问的文件名或者数据块索引是什么



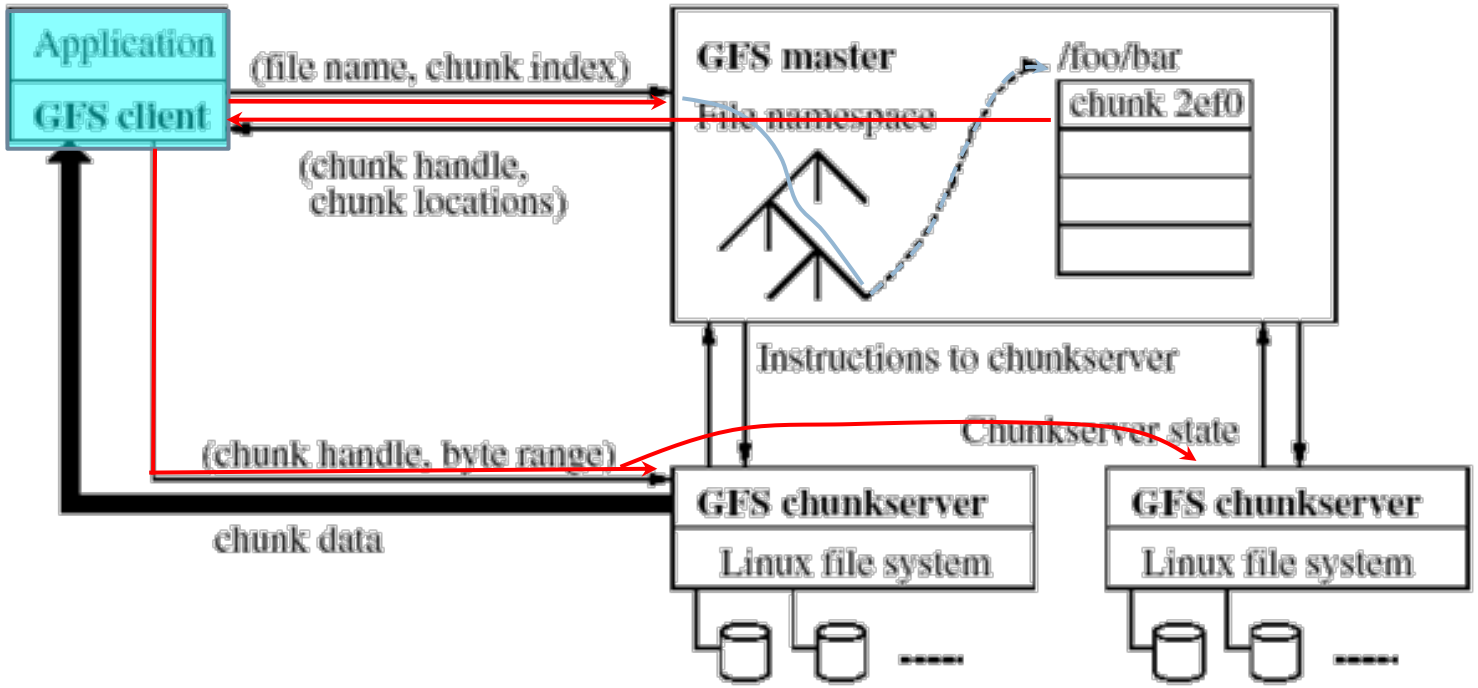
Google GFS的基本架构和工作原理

- 数据访问工作过程
  - 2. GFS Server根据文件名和数据块索引在其文件目录空间中查找和定位该文件或数据块，找出数据块具体在哪些ChunkServer上；将这些位置信息回送给应用程序



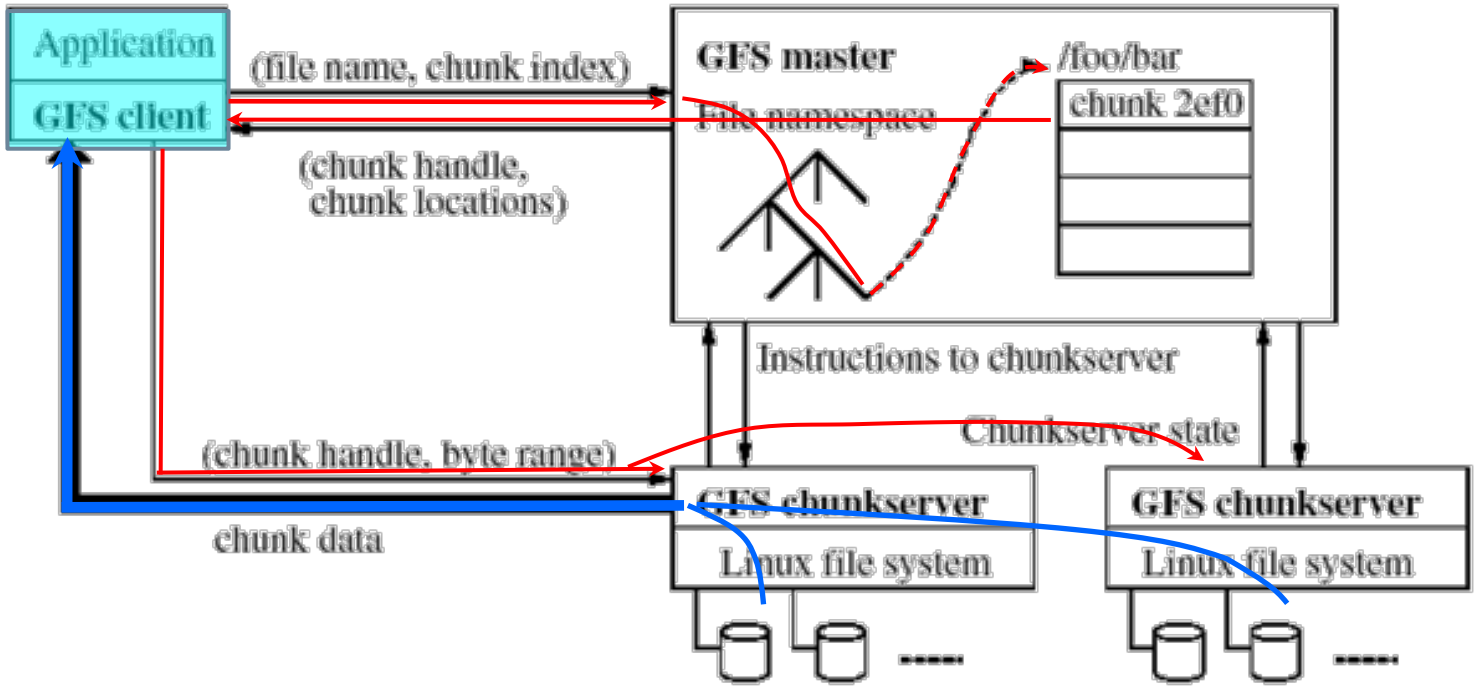
Google GFS的基本架构和工作原理

- 数据访问工作过程
  - 3.应用程序根据GFS Server返回的具体Chunk数据块位置信息，直接访问相应的ChunkServer



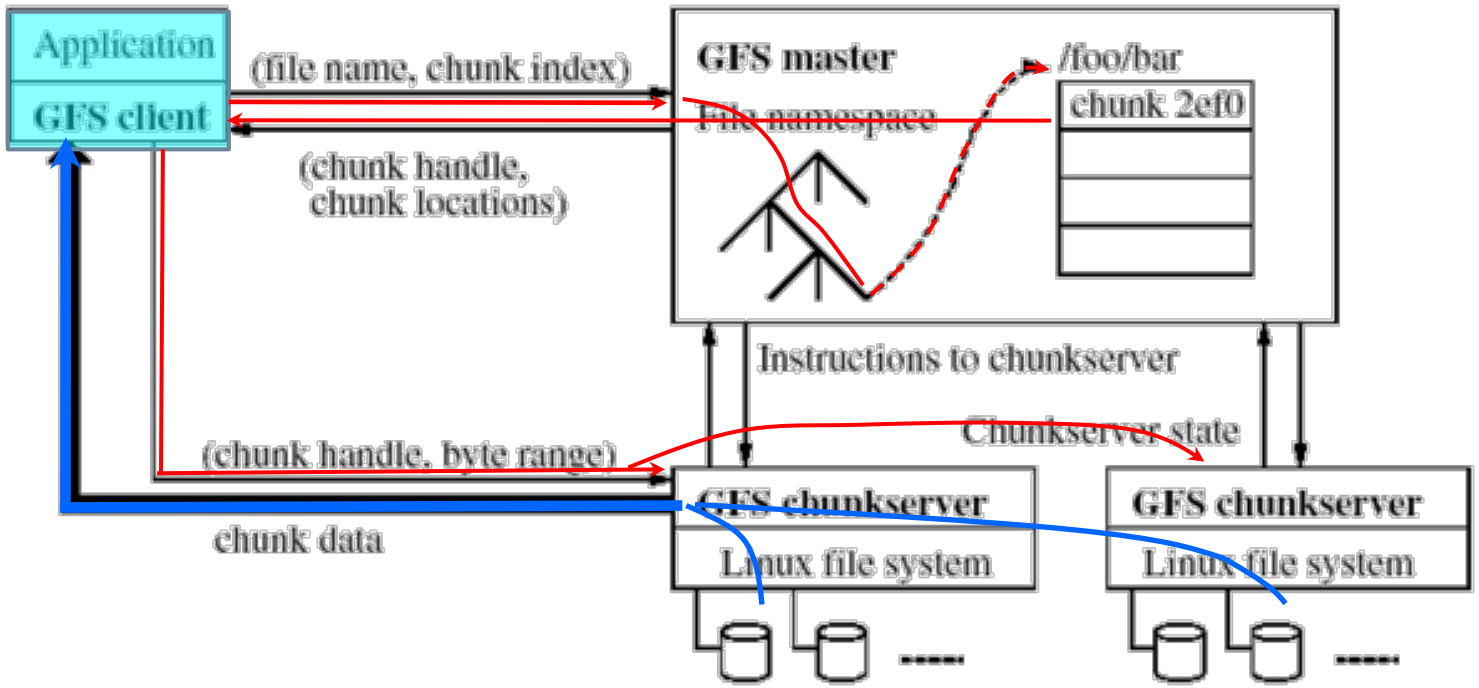
Google GFS的基本架构和工作原理

- 数据访问工作过程
  - 4.应用程序根据GFS Server返回的具体Chunk数据块位置信息直接读取指定位置的数据进行计算处理



Google GFS的基本架构和工作原理

- 数据访问工作过程
  - 特点：应用程序访问具体数据时不需要经过GFS Master，因此，避免了Master成为访问瓶颈
  - 并发访问：由于一个大数据会存储在不同的ChunkServer中，应用程序可实现并发访问



Google GFS的基本架构和工作原理

- GFS的系统管理技术
  - 大规模集群安装技术：如何在一个成千上万个节点的集群上迅速部署GFS，升级管理和维护等
  - 故障检测技术：GFS是构建在不可靠的廉价计算机之上的文件系统，节点数多，故障频繁，如何快速检测、定位、恢复或隔离故障节点
  - 节点动态加入技术：当新的节点加入时，需要能自动安装和部署GFS
  - 节能技术：服务器的耗电成本大于购买成本，Google为每个节点服务器配置了蓄电池替代UPS，大大节省了能耗。

摘要

- Google MapReduce的基本工作原理
- 分布式文件系统GFS的基本工作原理
- 分布式结构化数据表BigTable的基本工作原理

BigTable的基本作用和设计思想

- GFS是一个文件系统，难以提供对结构化数据的存储和访问管理。为此，Google在GFS之上又设计了一个结构化数据存储和访问管理系统—BigTable，为应用程序提供比单纯的文件系统更方便、更高层的数据操作能力。
- Google的很多数据，包括Web索引、卫星图像数据、地图数据等都以结构化形式存放在BigTable中。
- BigTable提供了一定粒度的结构化数据操作能力，主要解决一些大型媒体数据（Web文档、图片等）的结构化存储问题。但与传统的关系数据库相比，其结构化粒度没有那么高，也没有事务处理等能力，因此，它并不是真正意义上的数据库。

BigTable的设计动机和目标

- 需要存储多种数据
  - Google提供的服务很多，需要处理的数据类型也很多，如URL，网页，图片，地图数据，email，用户的个性化设置等
- 海量的服务请求
  - Google是目前世界上最繁忙的系统，因此，需要有高性能的请求和数据处理能力
- 商用数据库无法适用
  - 在如此庞大的分布集群上难以有效部署商用数据库系统，且其难以承受如此巨量的数据存储和操作需求

BigTable的设计动机和目标

- 广泛的适用性：为一系列服务和应用而设计的数据存储系统，可满足对不同类型数据的存储和操作需求
- 很强的可扩展性：根据需要可随时自动加入或撤销服务器节点
- 高吞吐量数据访问：提供P级数据存储能力，每秒数百万次的访问请求
- 高可用性和容错性：保证系统在各种情况下都能正常运转，服务不中断
- 自动管理能力：自动加入和撤销服务器，自动负载平衡
- 简单性：系统设计尽量简单以减少复杂性和出错率

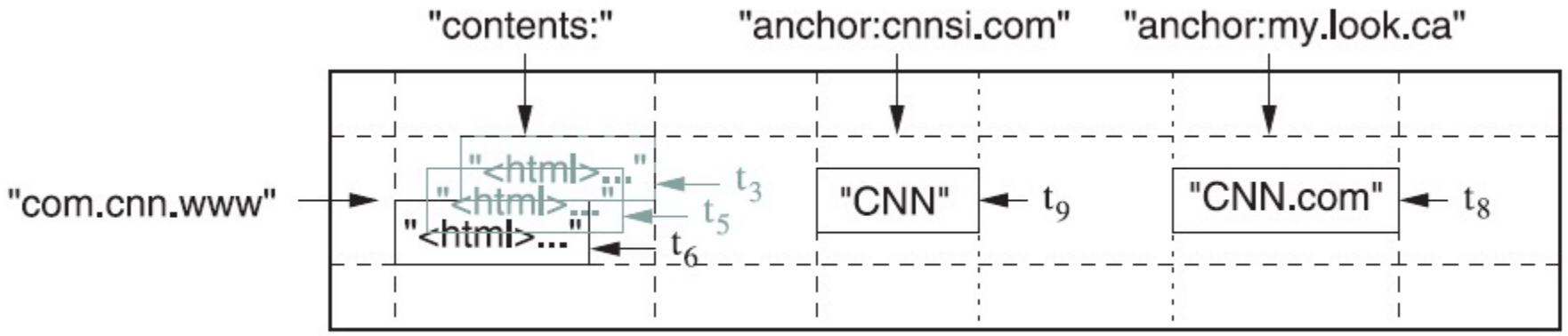
BigTable数据模型

- BigTable主要是一个分布式多维表，表中的数据通过：
  - 一个行关键字（row key）
  - 一个列关键字（column key）
  - 一个时间戳（timestamp）

  进行索引和查询定位的。
- BigTable对存储在表中的数据不做任何解释，一律视为字节串，具体数据结构的实现由用户自行定义。
- BigTable查询模型
  - (row:string, column:string,time:int64)→ 结果数据字节串
  - 支持查询、插入和删除操作

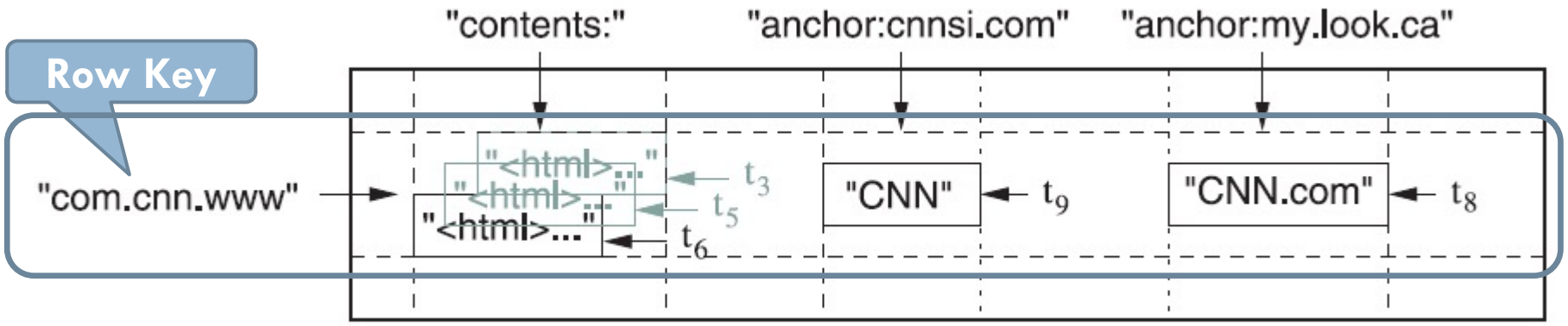
BigTable数据模型– WebTable Example

A Table in BigTable is a: Sparse, Distributed, Persistent, Multidimensional Sorted map.



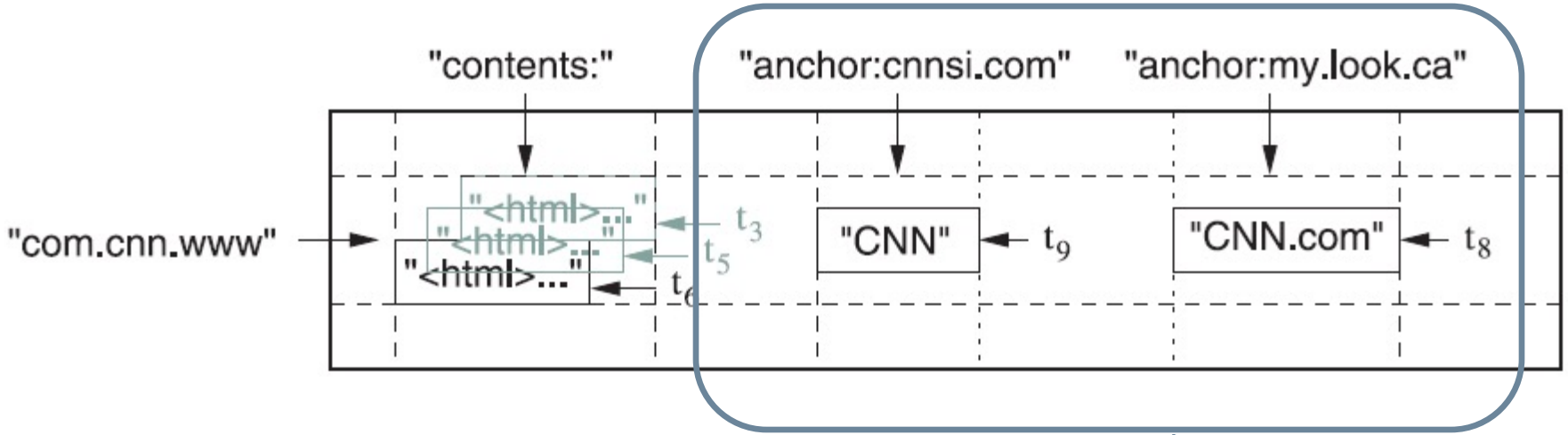
A large collection of web pages and related information

BigTable数据存储格式



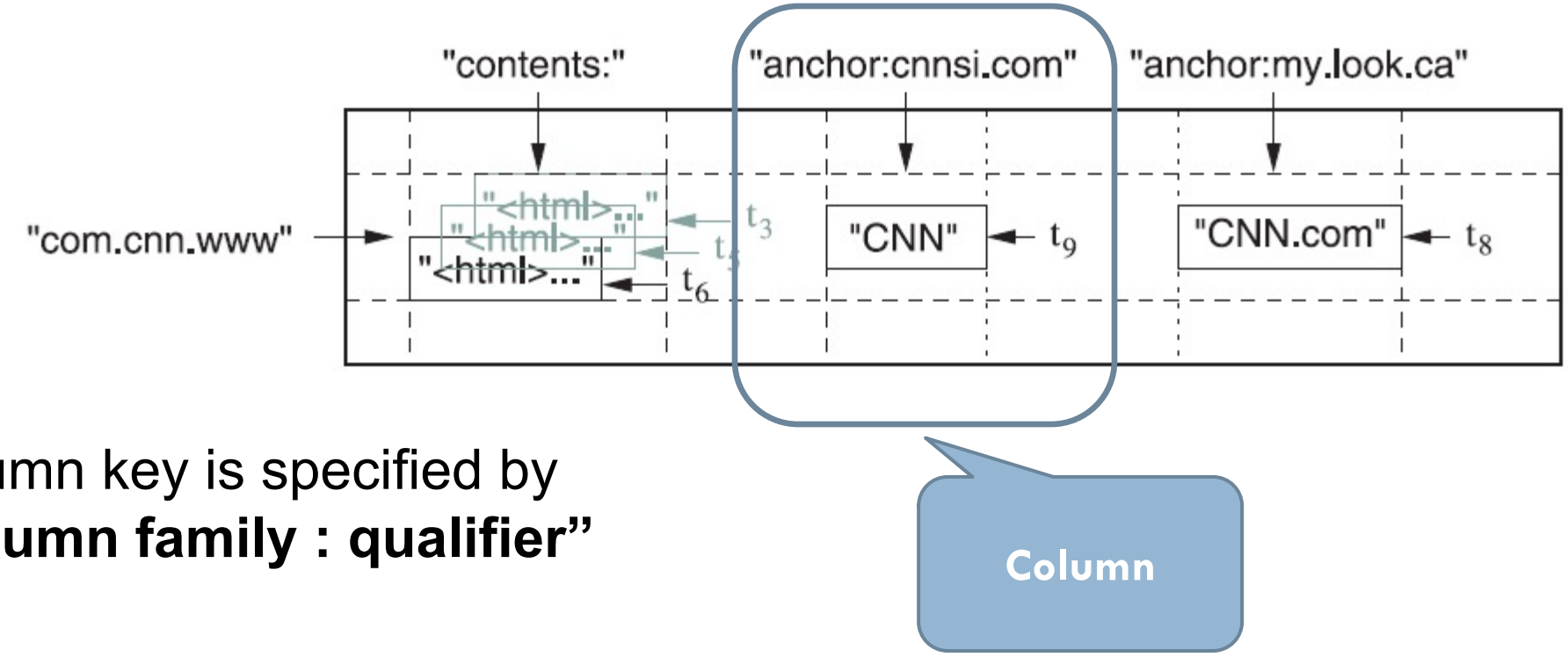
- 行(Row):大小不超过64KB的任意字符串。表中的数据都是根据行关键字进行排序的。
  - com.cnn.www就是一个行关键字，指明一行存储数据。URL地址倒排好处是：1)同一地址的网页将被存储在表中连续的位置，便于查找；2)倒排便于数据压缩，可大幅提高数据压缩率
- 子表(Tablet)：一个大表可能太大，不利于存储管理，将在水平方向上被分为多个子表

BigTable数据存储格式



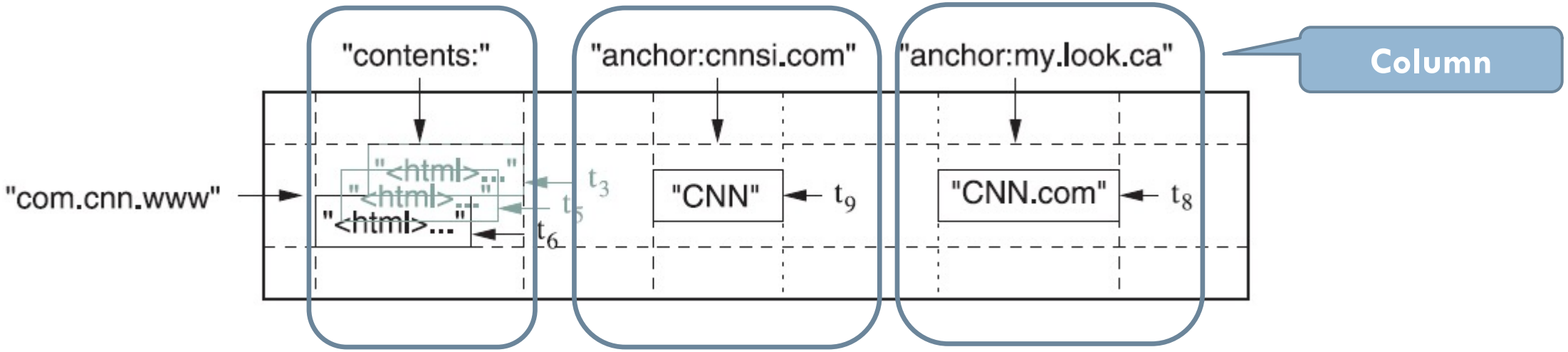
Column family is the unit of access control

BigTable数据存储格式



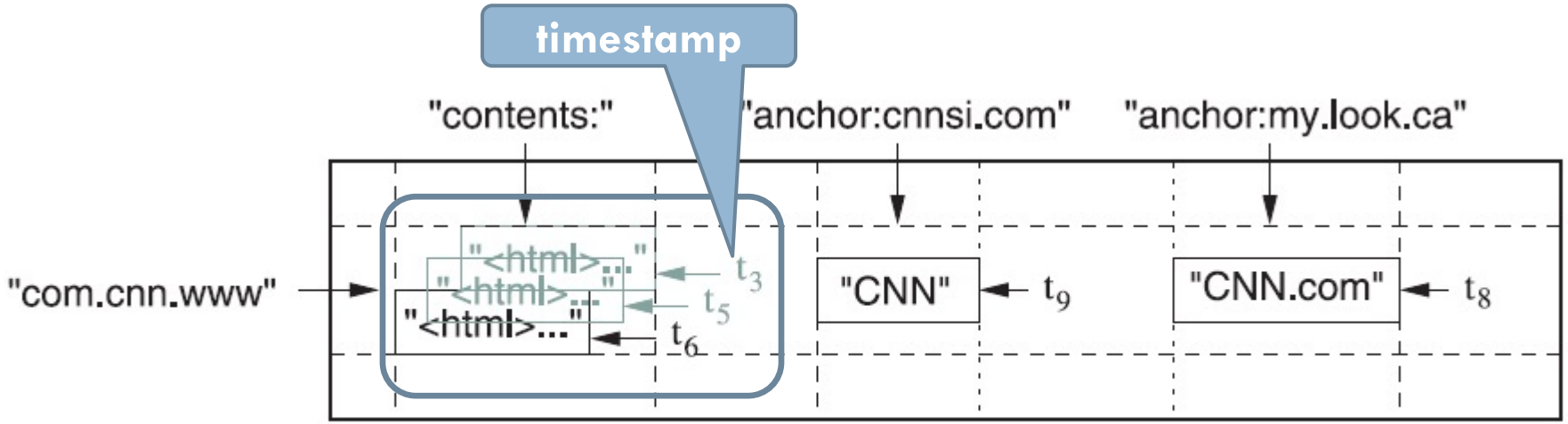
Column key is specified by
“**Column family : qualifier**”

BigTable数据存储格式



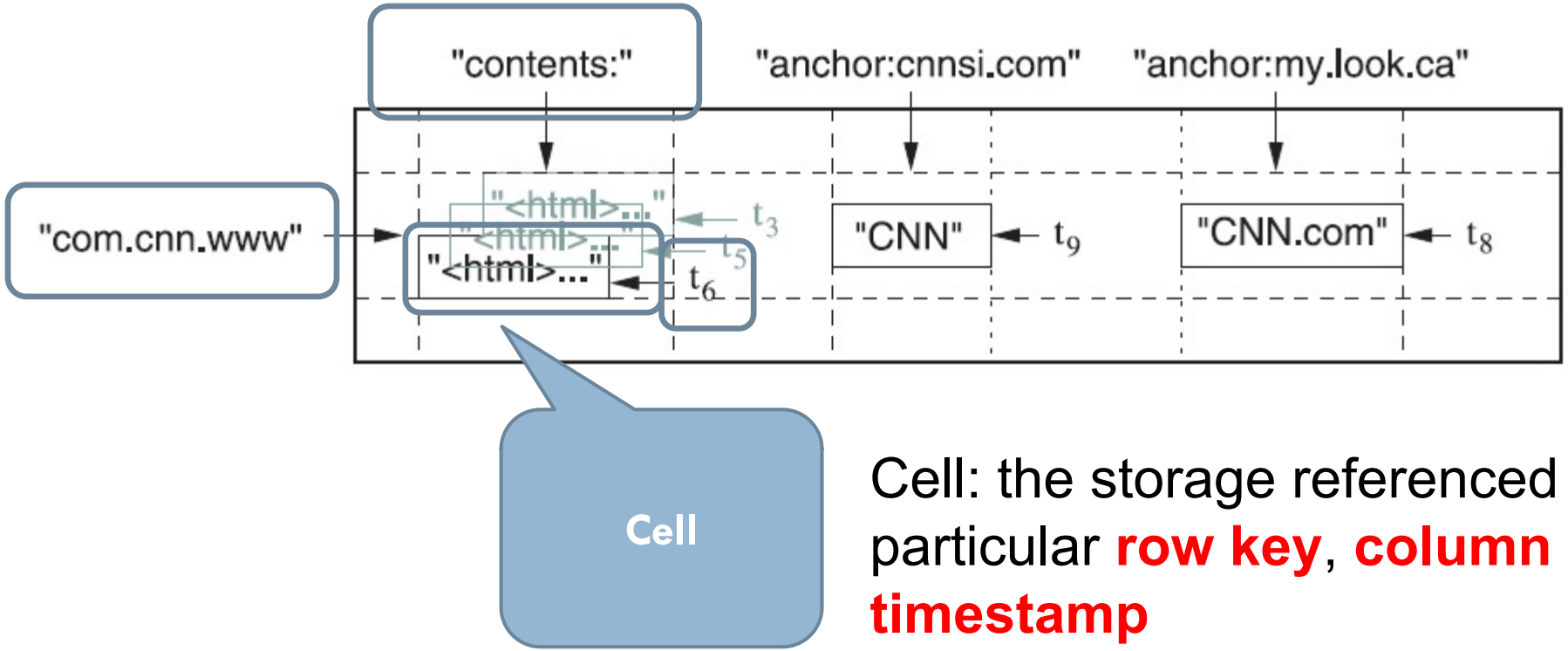
- 列(Column): BigTable将列关键字组织成为“列族”(column family)，每个族中的数据属于同一类别，如anchor是一个列族，其下可有不同的表示一个个超链的列关键字。一个列族下的数据会被压缩在一起存放（按列存放）。因此，一个列关键字可表示为： 族名：列名(family:qualifier)
  - content、anchor都是族名；而cnnsi.com和my.look.ca则是anchor族中的列名。

BigTable数据存储格式



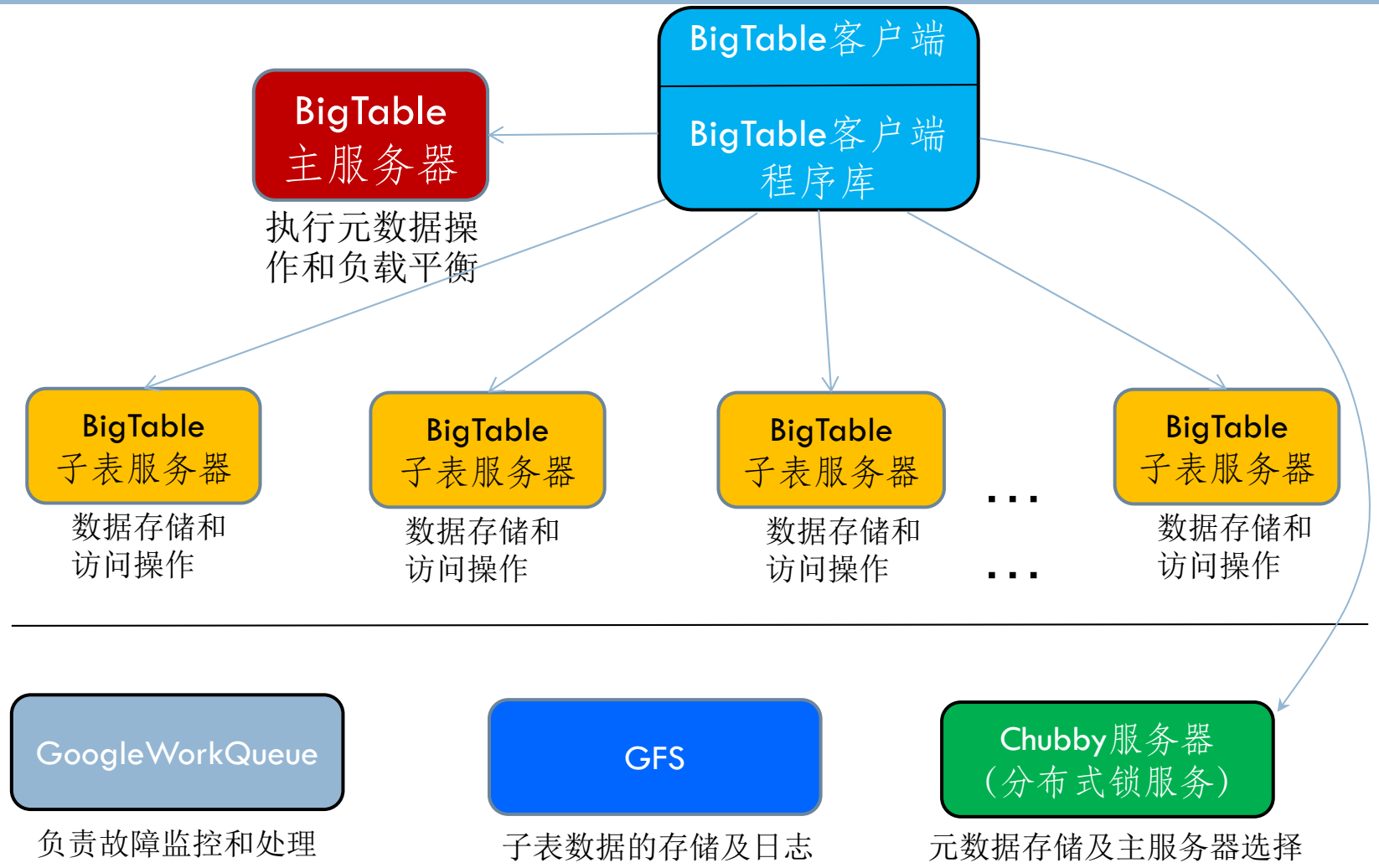
- 时间戳(time stamp):很多时候同一个URL的网页会不断更新，而Google需要保存不同时间的网页数据，因此需要使用时间戳来加以区分。
- 为了简化不同版本的数据管理，BigTable提供给了两种设置：
  - 保留最近的n个版本数据
  - 保留限定时间内的所有不同版本数据

BigTable数据存储格式



Cell: the storage referenced by a particular **row key**, **column key**, and **timestamp**

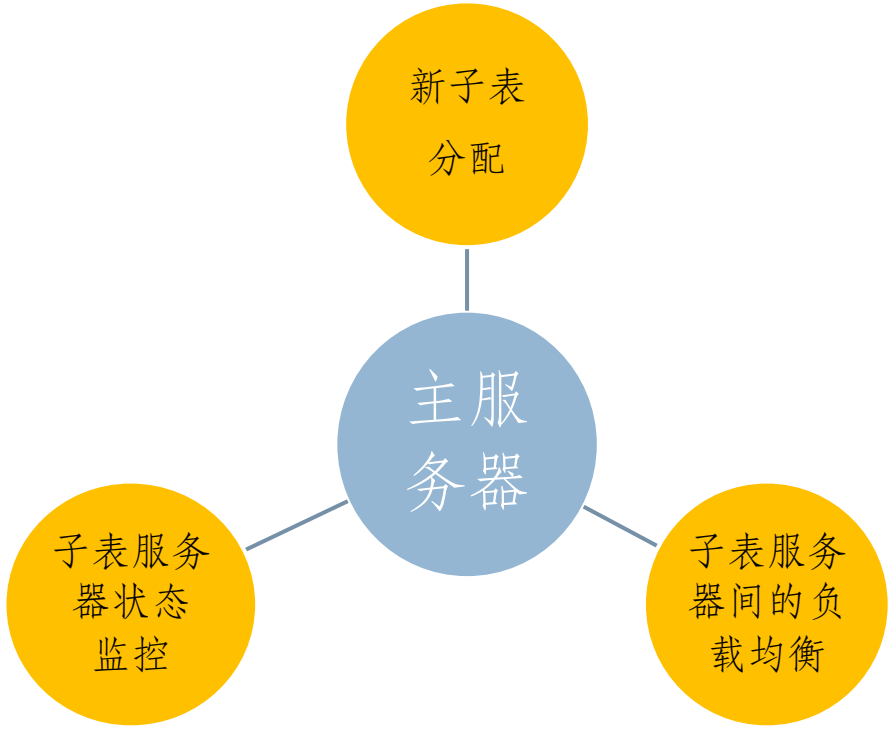
BigTable基本架构



BigTable基本架构

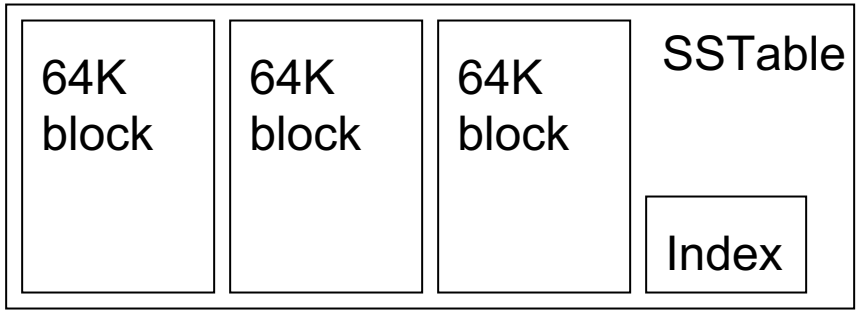
主服务器

- 新子表分配：当一个新子表产生时，主服务器通过加载命令将其分配给一个空间足够大的子表服务器；创建新表、表合并及较大子表的分裂都会产生新的子表。
- 子表监控：通过Chubby完成。所有子表服务器基本信息被保存在Chubby中的服务器目录中主服务器检测这个目录可获取最新子表服务器的状态信息。当子表服务器出现故障，主服务器将终止该子表服务器，并将其上的全部子表数据移动到其它子表服务器。
- 负载均衡：当主服务器发现某个子表服务器负载过重时，将自动对其进行负载均衡操作。



BigTable基本架构

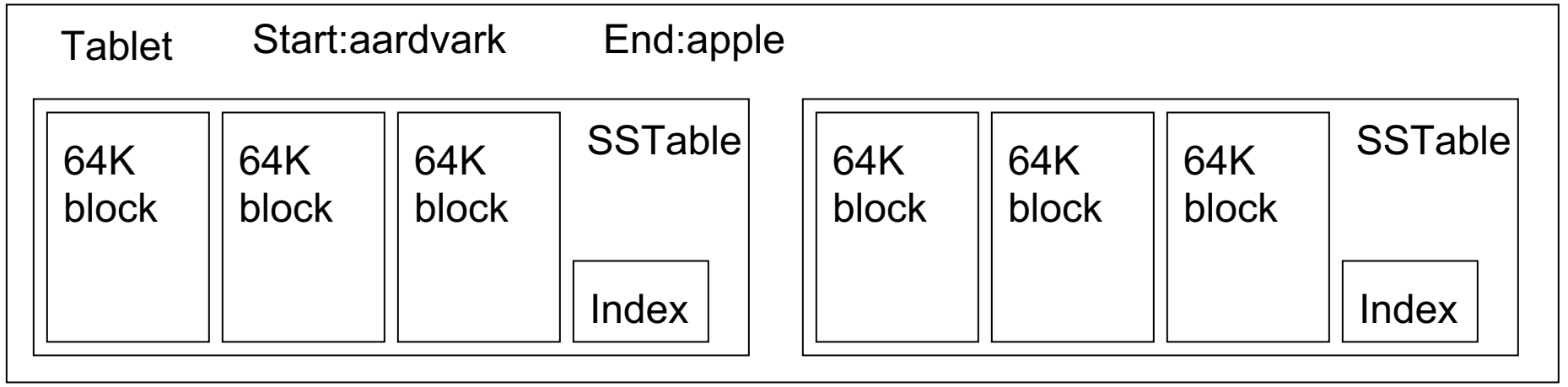
- 子表服务器
  - BigTable中的数据都以子表形式保存在子表服务器上，客户端程序也直接和子表服务器通信。
  - 分配：当一个新子表产生，子表服务器的主要问题包括子表的定位、分配、及子表数据的最终存储。



- 子表的基本存储结构SSTable
  - SSTable是BigTable内部的基本存储结构，以GFS文件形式存储在GFS文件系统中。一个SSTable实际上对应于GFS中的一个64MB的数据块(Chunk)
  - SSTable中的数据进一步划分为64KB的子块，因此一个SSTable可以有多达1千个这样的子块。为了维护这些子块的位置信息，需要使用一个Index索引。

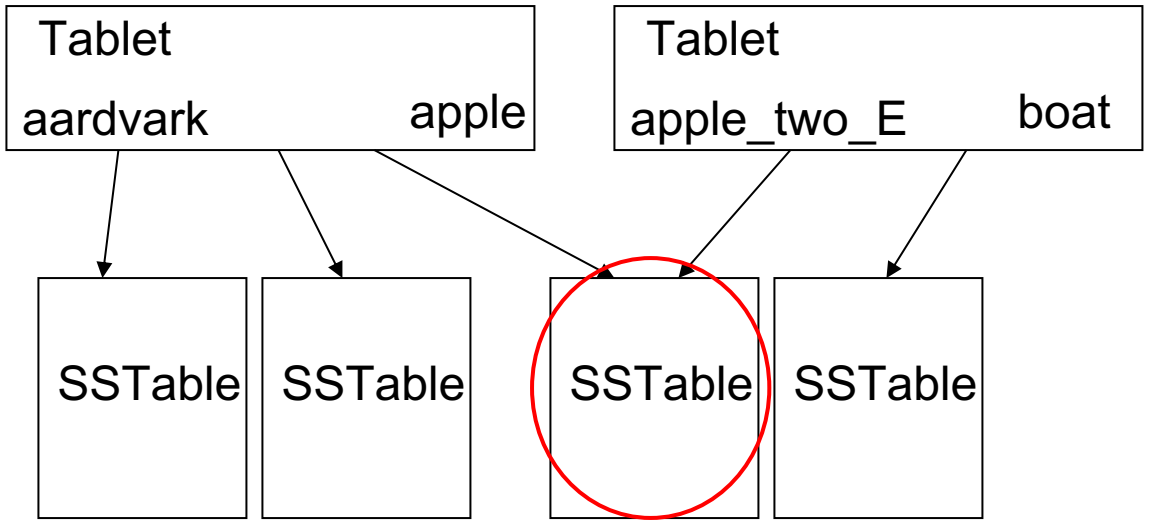
BigTable基本架构

- 子表数据格式
  - 概念上子表是整个大表的多行数据划分后构成。而一个子表服务器上的子表将进一步由很多个SSTable构成，每个SSTable构成最终的在底层GFS中的存储单位。



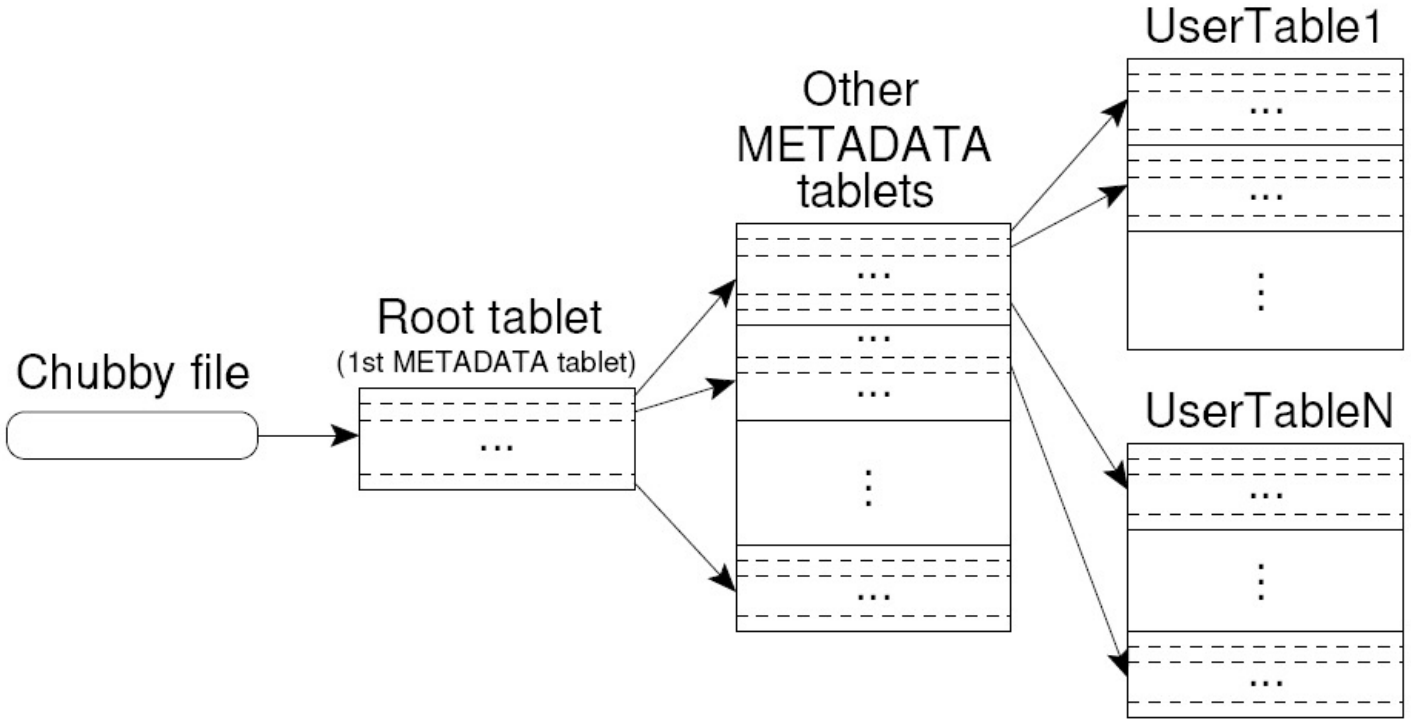
BigTable基本架构

- 子表数据格式
  - 一个SSTable还可以为不同的子表所共享，以避免同样数据的重复存储。



BigTable基本架构

- 子表寻址
  - 子表地址以3级B+树形式进行索引；首先从Chubby服务器中取得根子表，由根子表找到二级索引子表，最后获取最终的SSTable的位置



参考文献

1. Jeffrey Dean and Sanjay Ghemawat, **MapReduce: Simplied Data Processing on Large Clusters**, Proceedings of OSDI 2004.
2. Sanjay Ghemawat, et. al, **The Google File System**, Proceedings of the 19th ACM Symposium on Operating systems principles, SOSP'03, Oct. 2003.
3. Fay Chang, et. al, **Bigtable: a distributed storage system for structured data**, Proceedings of OSDI 2006.

THANK YOU

南京大学
NANJING UNIVERSITY

南京大学计算机软件研究所
Institute of Computer Software, Nanjing University