分布式存储

（第六部分）

李洪亮
lihongliang@jlu.edu.cn
云计算和网络安全实验室
吉林大学　计算机科学与技术学院

- 概论（4）
- 分布式系统基础（8）
  - 分布式系统发展
  - 高性能计算和数据中心（架构与管理）
- 分布式存储技术（9）
- 分布式存储系统（27）
  - 网络文件系统（NFS）
  - 分布式文件系统（HDFS，GFS）
  - 分布式键值对存储（Dynamo）
  - 分布式表格系统（BigTable）
  - 分布式对象存储（S3）

- 内容提要
  - Bigtable简介与数据模型
  - 系统架构
  - 数据分布
  - 存储实现
  - 容错和负载分配
  - 对比和应用

简介

- 面向高可用性的分布式存储系统
- 支持半结构化数据
  - URLs: content, metadata, links, anchors, page rank
  - User data: preferences, account info, recent queries
  - Geography: roads, satellite images, points of interest, annotations
- 大规模
  - PB级别数据，数以千计的服务器
  - 数十亿的URLs(多版本)
  - 数亿万用户
  - 每秒数千个查询
  - 100TB+遥感数据

简介

Bigtable: A Distributed Storage System for Structured Data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach
Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber
{fay,jeff,sanjay,wilsonh,kerr,m3b,tushar,fikes,gruber}@google.com
Google, Inc.

Abstract

Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers. Many projects at Google store data in Bigtable, including web indexing, Google Earth, and Google Finance. These applications place very different demands on Bigtable, both in terms of data size (from URLs to web pages to satellite imagery) and latency requirements (from backend bulk processing to real-time data serving). Despite these varied demands, Bigtable has successfully provided a flexible, high-performance solution for all of these Google products. In this paper we describe the simple data model provided by Bigtable, which gives clients dynamic control over data layout and format, and we describe the design and implementation of Bigtable.

1 Introduction

Over the last two and a half years we have designed, implemented, and deployed a distributed storage system for managing structured data at Google called Bigtable. Bigtable is designed to reliably scale to petabytes of data and thousands of machines. Bigtable has achieved several goals: wide applicability, scalability, high performance, and high availability. Bigtable is used by more than sixty Google products and projects, including Google Analytics, Google Finance, Orkut, Personalized Search, Writely, and Google Earth. These products use Bigtable for a variety of demanding workloads, which range from throughput-oriented batch-processing jobs to latency-sensitive serving of data to end users. The Bigtable clusters used by these products span a wide range of configurations, from a handful to thousands of servers, and store up to several hundred terabytes of data.

In many ways, Bigtable resembles a database: it shares many implementation strategies with databases. Parallel databases [14] and main-memory databases [13] have achieved scalability and high performance, but Bigtable provides a different interface than such systems. Bigtable does not support a full relational data model; instead, it provides clients with a simple data model that supports dynamic control over data layout and format, and allows clients to reason about the locality properties of the data represented in the underlying storage. Data is indexed using row and column names that can be arbitrary strings. Bigtable also treats data as uninterpreted strings, although clients often serialize various forms of structured and semi-structured data into these strings. Clients can control the locality of their data through careful choices in their schemas. Finally, Bigtable schema parameters let clients dynamically control whether to serve data out of memory or from disk.

Section 2 describes the data model in more detail, and Section 3 provides an overview of the client API. Section 4 briefly describes the underlying Google infrastructure on which Bigtable depends. Section 5 describes the fundamentals of the Bigtable implementation, and Section 6 describes some of the refinements that we made to improve Bigtable's performance. Section 7 provides measurements of Bigtable's performance. We describe several examples of how Bigtable is used at Google in Section 8, and discuss some lessons we learned in designing and supporting Bigtable in Section 9. Finally, Section 10 describes related work, and Section 11 presents our conclusions.

2 Data Model

A Bigtable is a sparse, distributed, persistent multi-dimensional sorted map. The map is indexed by a row key, column key, and a timestamp; each value in the map is an uninterpreted array of bytes.

(row:string, column:string, time:int64) → string

To appear in OSDI 2006

Bigtable: A Distributed Storage System for Structured Data, OSDI 2006

Google应用

- 支撑Google应用
  - Google Analytics
  - Google Finance
  - Personalized search – Blogger.com
  - Google Code hosting
  - YouTube
  - Gmail
  - Google Earth & Google Maps
  - Dozens of others... *over sixty products*

为什么不使用传统数据库

- 可扩展性不够
- 成本高
- 内部维护和更新更方便
- 强调性能优化的可行性

设计目标

- 支持异步数据访问(每秒百万级访问请求)
- 支持顺序访问数据(需要排序)
- 保证数据一致性(需要锁)

Bigtable简介

- 分布式数据多维映射表
- 支持容错和数据持久化
- 可扩展性
  - 数千台节点
  - TB级别内存存储
  - PB级别持久化存储
  - 每秒百万级读写，高效扫描数据
- 系统资源管理
  - 服务器动态加入/退出
  - 负载均衡

"A Bigtable is a sparse, distributed, persistent multidimensional sorted map"*

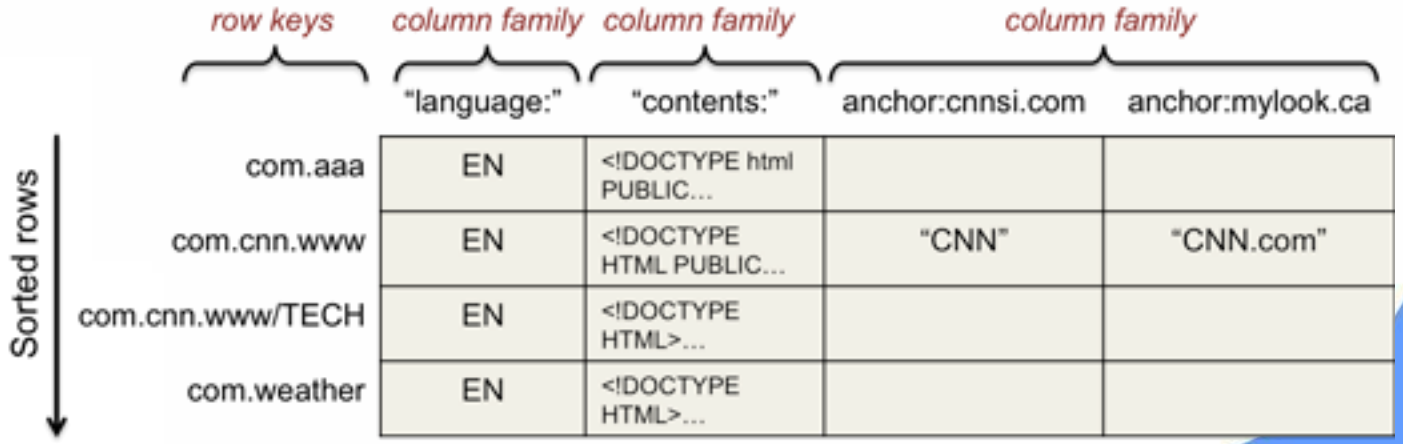
数据模型

- Bigtable系统由多个表格构成，每个表有多行
- 每行有一个主键(row key)和多个列(column)
- 每一行一列确定一个单元(cell)
- 每个单元包含多个版本的数据(时间戳)

(row:string, column:string, time:int64) → string

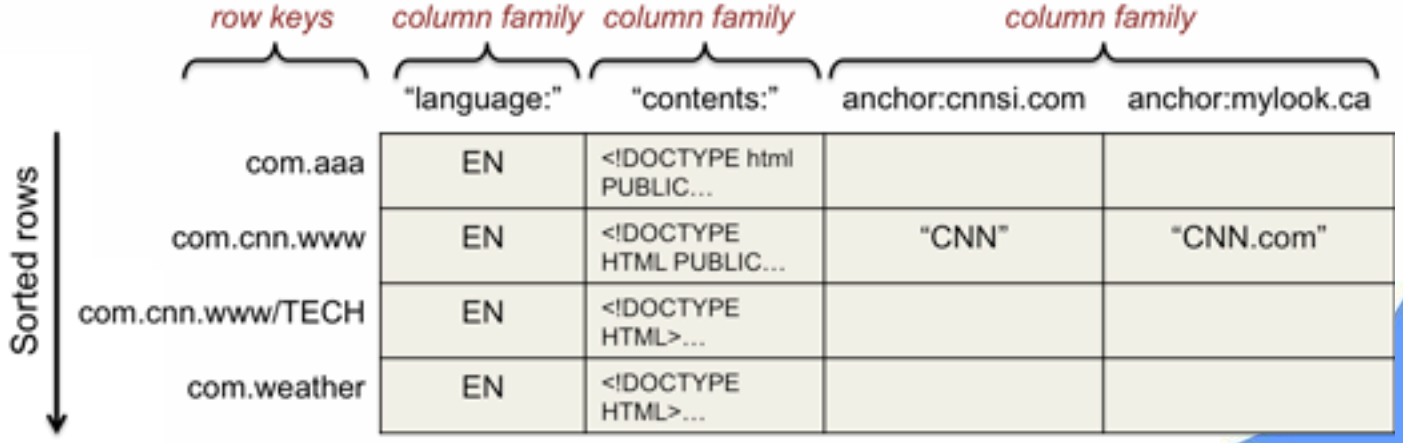
数据模型-行

- Row Key是一个自定义字符串
  - 对一行的数据访问是原子的(锁)
- Row按照Key值进行全局字典序排序(保持排序)
  - Key值相近的数据临近存储：AAAA，AAAB，AABA
- 不支持关系模型
- 不支持多行的事务(可多个单行连续操作)



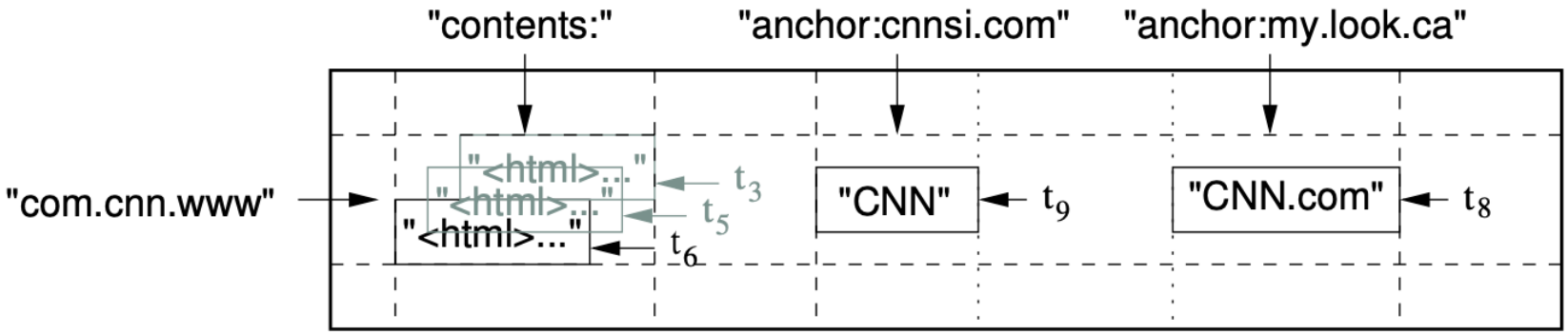
数据模型-列

- 多个列组织成列族(column family)
  - 每个列名为(column family: qualifier) 列族/限定词
  - 列族是基本访问单元，统一权限
  - 列族不允许过多，qualifier可以用户自定义多个
  - 结构化(列族)和非结构化(限定词)的平衡



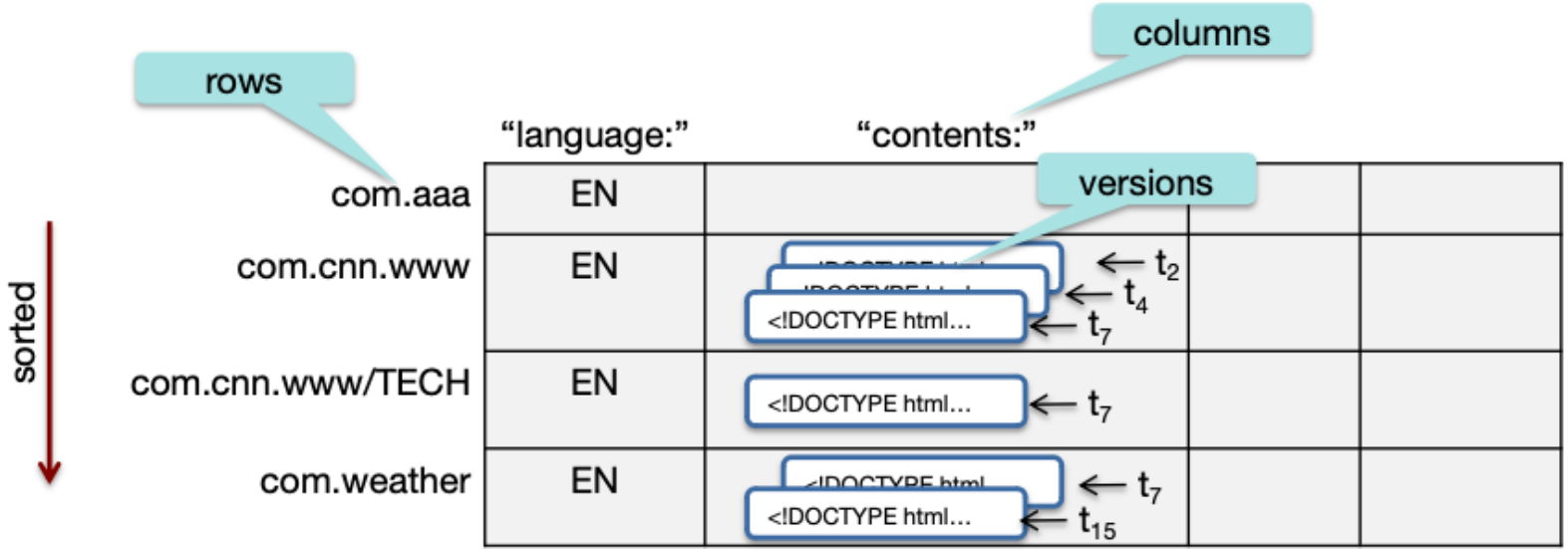
数据模型举例

- 列是任意字符串64KB
- 数据按照主键全局排序(字典序)
- com.cnn.www (为什么这样设计？)
- 列族：contents，anchor
- Qualifier可以用户自定义多个
- 时间戳代表多个版本(某个cell)



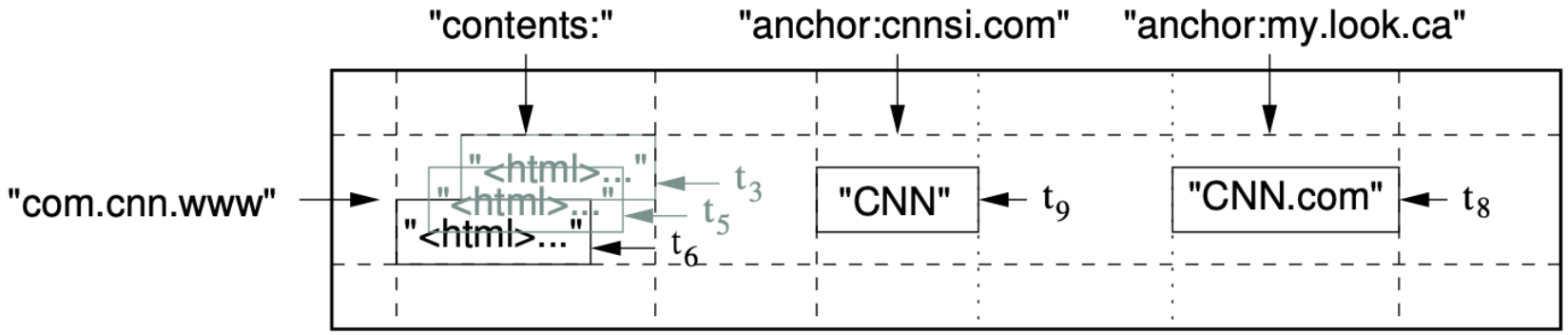
数据模型-时间戳

- 用于维护一个cell的多个数据版本
  - 每一次新的写入产生一个新的时间戳
  - 也可以由用户设置



数据模型-时间戳

- 用于维护一个cell的多个数据版本
  - 每一次新的写入产生一个新的时间戳
  - 也可以由用户设置
- 查找选项
  - 返回时间戳最新的K个值
  - 返回所有在时间戳范围内的数据
  - 存储时间戳数量可以设置



数据模型-列族

- 基本数据访问单元
- 一个列族中的数据通常是一种类型
- 列族数量通常比较小
- 列族中的qualifier数量可以很大（每行可以不同）
- Bigtable的表格一般比较大，而且稀疏

Column family *anchor*

	"language:"	"contents:"	anchor:cnnsi.com	anchor:mylook.ca
com.aaa	EN	<!DOCTYPE html PUBLIC…		
com.cnn.www	EN	<!DOCTYPE HTML PUBLIC…	"CNN"	"CNN.com"
com.cnn.www/TECH	EN	<!DOCTYPE HTML>…		
com.weather	EN	<!DOCTYPE		

sorted

"A Bigtable is a sparse, distributed, persistent multidimensional sorted map"*

数据模型-列族

- 基本数据访问单元
- 一个列族中的数据通常是一种类型
- 列族数量通常比较小
- 列族中的qualifier数量可以很大（每行可以不同）
- Bigtable的表格一般比较大，而且稀疏

行键	时间戳	ColumnFamily `contents`	ColumnFamily `anchor`	ColumnFamily `people`
“com.cnn.www”	T9		`anchor:cnnsi.com = "CNN"`	
"com.cnn.www"	T8		`anchor:my.look.ca = "CNN.com"`	
"com.cnn.www"	T6	`contents:html = "<html>…"`		
"com.cnn.www"	T5	`contents:html = "<html>…"`		
"com.cnn.www"	T3	`contents:html = "<html>…"`		
“com.example.www”	t5	`contents:html = "<html>…"`		`people:author = "John Doe"`

“A Bigtable is a sparse, distributed, persistent multidimensional sorted map”*


JILIN UNIVERSITY·CHINA
1946

- 内容提要
  - Bigtable简介与数据模型
  - 系统架构
  - 数据分布
  - 存储实现
  - 容错和负载分配
  - 对比和应用

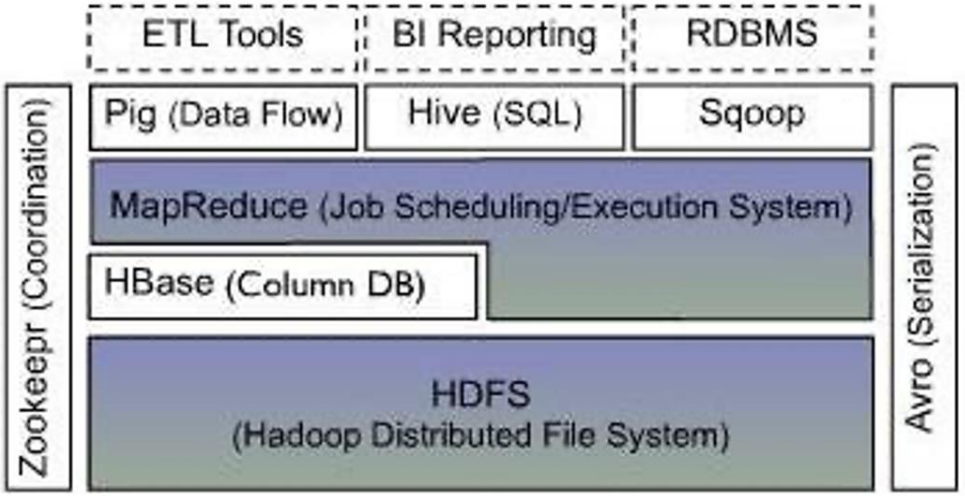
系统依赖

- Bigtable系统依赖于以下分布式系统：
  - GFS：可靠、并发、可扩展的持久化存储
  - 调度器：实现分布式系统中的任务调度
  - Chubby：分布式锁服务实现选举和全局信息维护
  - MapReduce：实现大规模数据处理(big table as input/output)

The Chubby lock service for loosely-coupled distributed systems, OSDI 2006

系统依赖

- Bigtable系统依赖于以下分布式系统：
  - GFS：可靠、并发、可扩展的持久化存储
  - 调度器：实现分布式系统中的任务调度
  - Chubby：分布式锁服务实现选举和全局信息维护
  - MapReduce：实现大规模数据处理(big table as input/output)



The Chubby lock service for loosely-coupled distributed systems, OSDI 2006

全局共享信息管理需求

- 保证数据一致性(需要锁)
- 分布式系统中共享信息的管理——Chubby
- Zookeeper的参考对象
- Bigtable的主要锁(全局信息)需求：
  - 命名空间一致性
  - 原子性的读写文件
  - 选举唯一主节点
  - 保存启动位置信息bootstrap location
  - 系统节点信息维护
  - 表格scheme信息维护
  - 访问控制

The Chubby lock service for loosely-coupled distributed systems, OSDI 2006

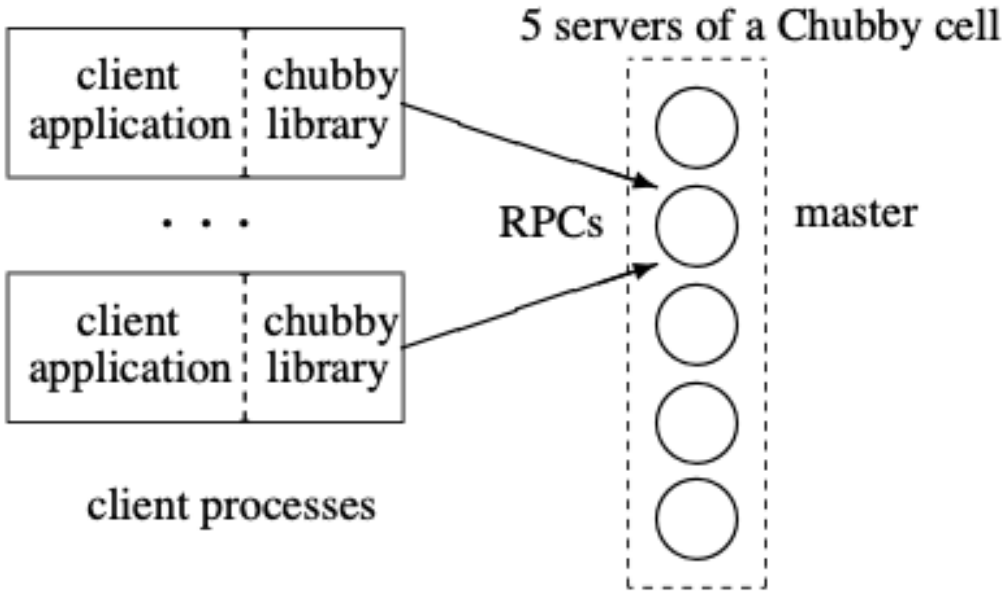
Chubby锁服务

- 系统架构：分布式系统
- 至少五个节点
  - 两地三数据中心五副本
  - 同城两个数据中心分别各部署两个副本
  - 异地数据中心布置一个副本
- Paxos算法的首次工业实现
  - 多数派一致
  - 保证一半以上节点工作

The Chubby lock service for loosely-coupled distributed systems, OSDI 2006

Chubby锁服务

- 由至少5个服务器构成一个Chubby Cell
- 服务于多个分布式客户端
- 须至少保证一半节点正常工作



The Chubby lock service for loosely-coupled distributed systems, OSDI 2006

Paxos协议(多个proposer)

- Prepare阶段：
  - Proposer发送prepare消息含序号 (proposal n*)
  - Acceptor promise (接收序号大的请求)
    - n* < n_current　　不回复
    - n* > n_current　　接受n*，同时回复之前已接受的最大请求号
- Accept阶段：
  - Proposer接收到多数派promise消息之后
  - 发送被承诺接受的最大请求号给Acceptor批准(accept消息)
  - 如未得到承诺接受的请求号，生成一个
  - Acceptor在不违背承诺的前提下接受请求
  - Proposer接收到多数Acceptor的接受消息
  - 发送acknowledge消息给所有人

基于GFS完成数据存储

- 存储引擎-支持顺序访问数据(需要排序)
  - 日志流派(仅支持扩展写入)
  - Sorted String Table
  - 通过数据合并优化存储空间和访问效率

- 分布式文件系统GFS
  - 保证数据存储的可靠性
- 存储表格数据(以文件形式存储于GFS)
  - 包括：操作日志和SSTable

LSM树存储引擎

- Log Structured Merge Tree
- 支持增、删、改、随机/顺序读
- 批量存储技术，可用于大规模存储系统
  - Google LevelDB
- 索引：加快读的速度/减慢写的速度
  - <key, offset>，数据无序存储
  - 日志结构流派Bitcast的局限性
    - 内存容量限制
    - 区间查询性能差（key值无序）

LSM树存储引擎

- 排序key值的索引：排序字符表SSTable
  - 稀疏索引/分布式索引/不在内存存放索引（解决内存容量限制）
  - Key值是有序的（解决区间查询性能差）
  - Sorted String Table （SSTable）

- 写性能如何保证？？？

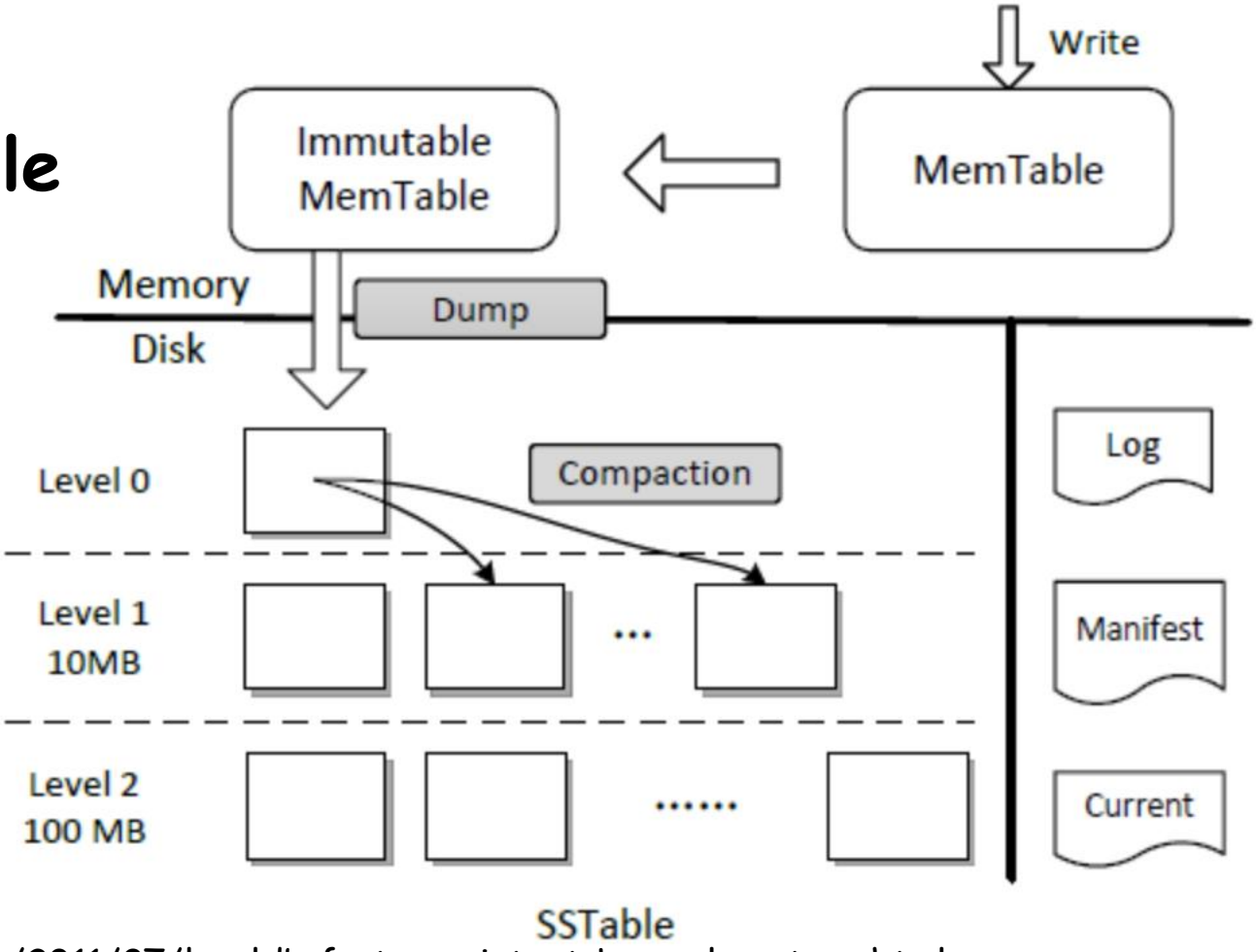
LSM树存储引擎

- 内存中维护有序的数据结构（平衡树）
  - 实现key值局部排序功能

- 内存中数据过大时，创建SSTable文件写入磁盘（持久化）

LSM树存储引擎

- 写入：MemTable（有序的结构）
  - 写入日志
  - 写入MemTable



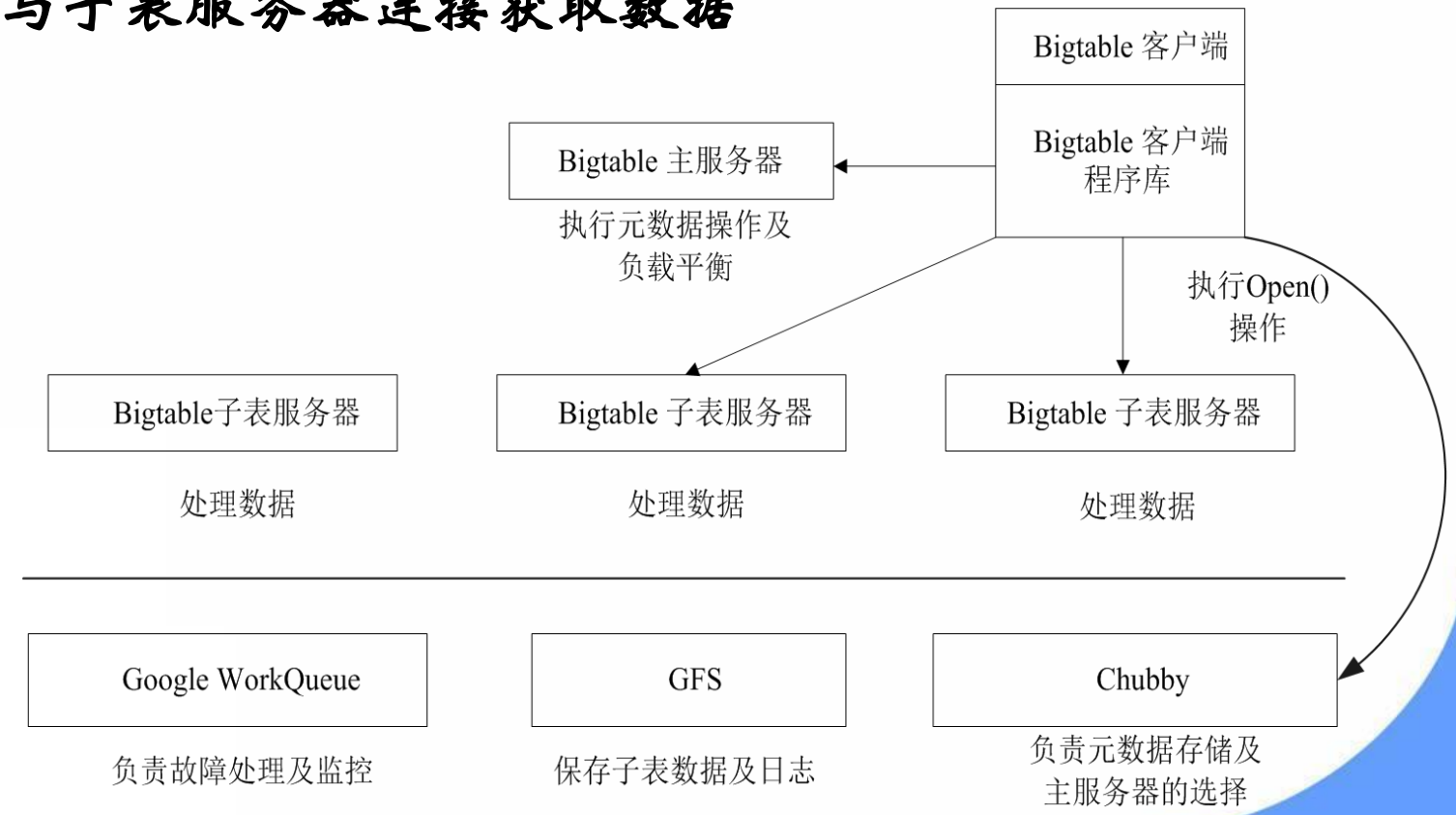
https://opensource.googleblog.com/2011/07/leveldb-fast-persistent-key-value-store.html

Bigtable系统结构(1)

- 主从结构(较灵活)
  - 客户端
    - 实现客户端接口，对数据单元进行增、删、改、查
  - 一个Master
    - 元数据操作、负载均衡
  - 多个TabletServer
    - 管理Tablet数据，提供读写服务

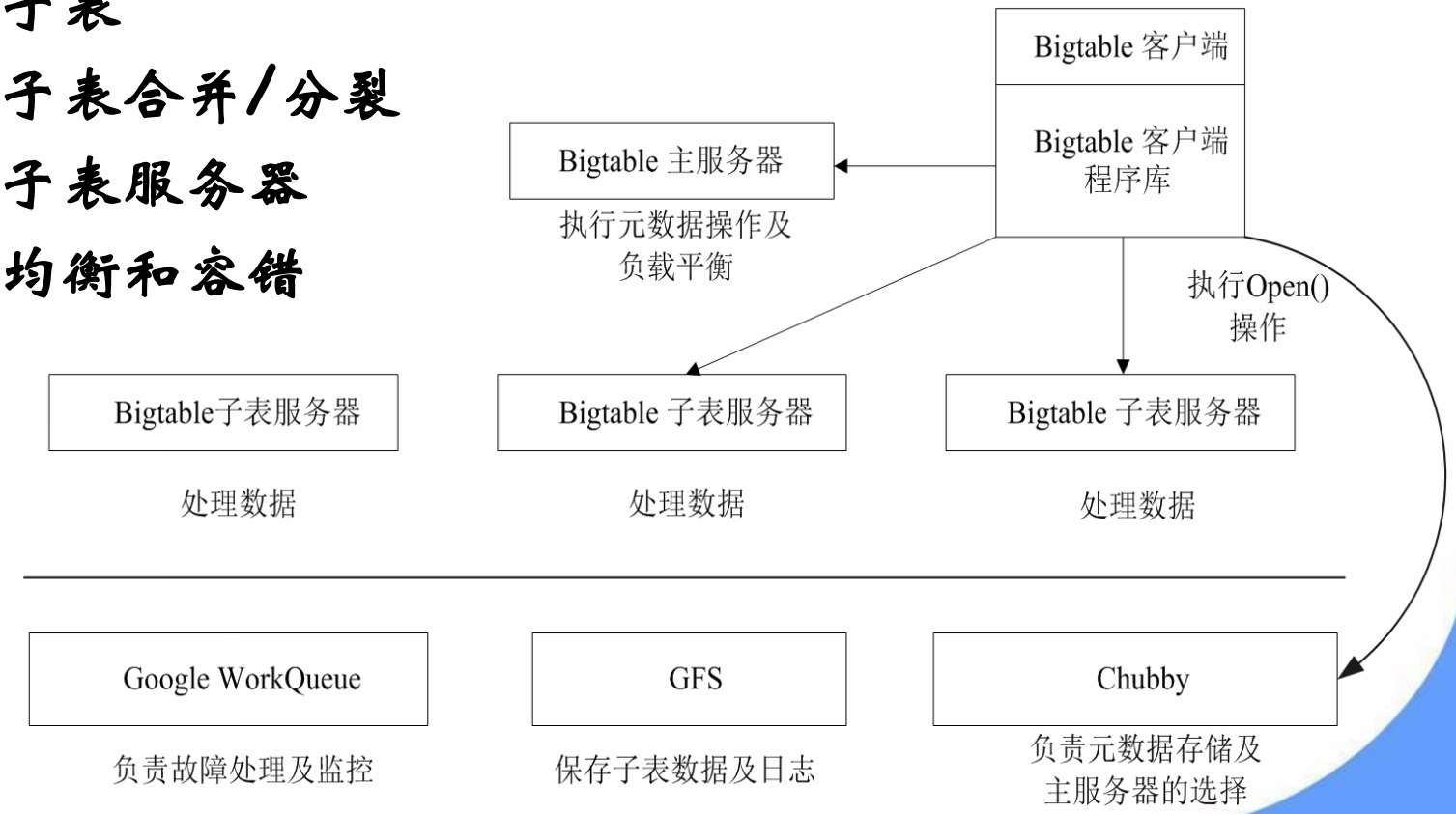
Bigtable系统结构(2)

- 客户端
  - 从Chubby获得锁服务
  - 直接与子表服务器连接获取数据



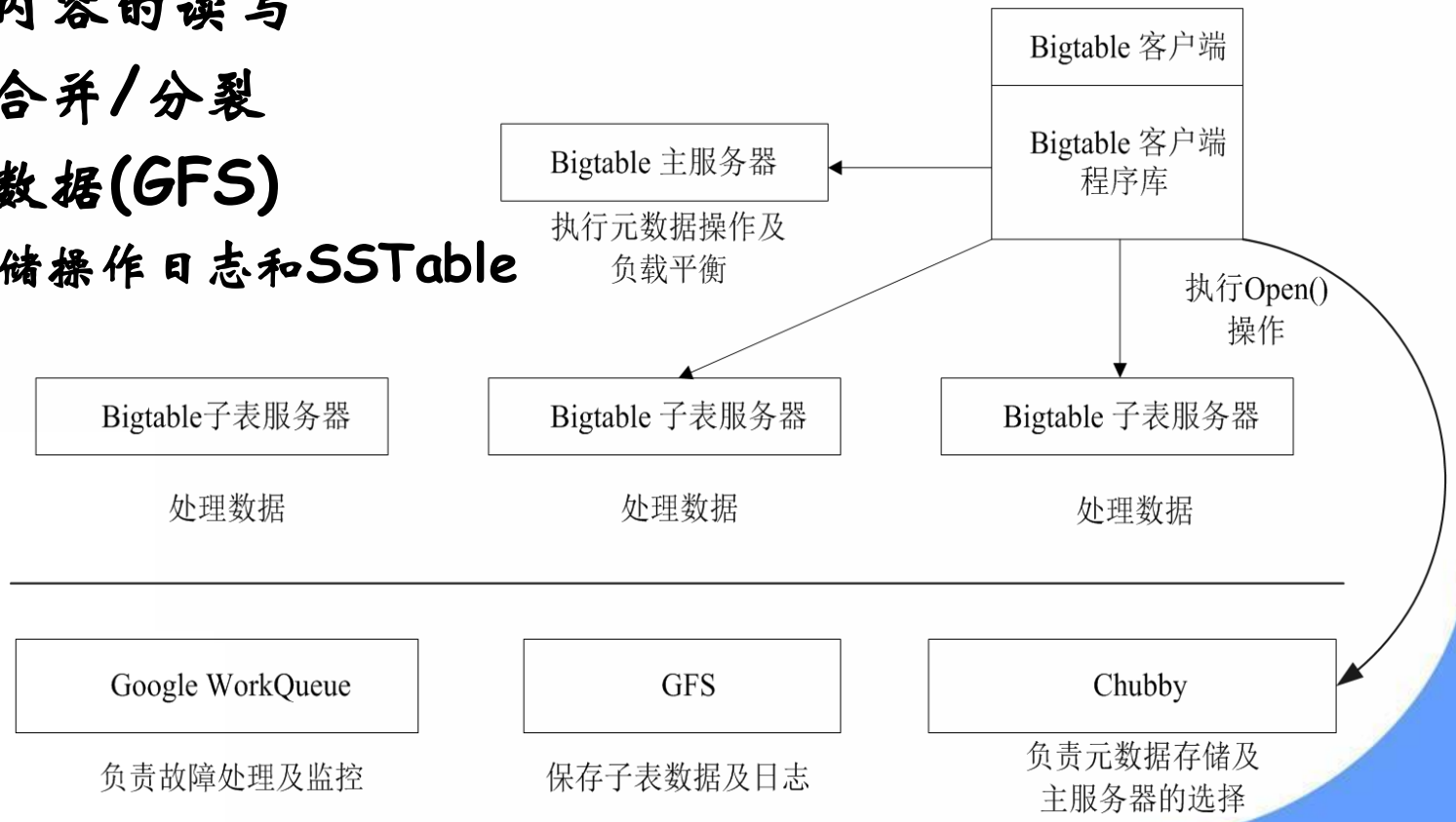
Bigtable系统结构(3)

- 主服务器Master
  - 管理子表服务器
  - 分配子表
  - 管理子表合并/分裂
  - 监控子表服务器
  - 负载均衡和容错



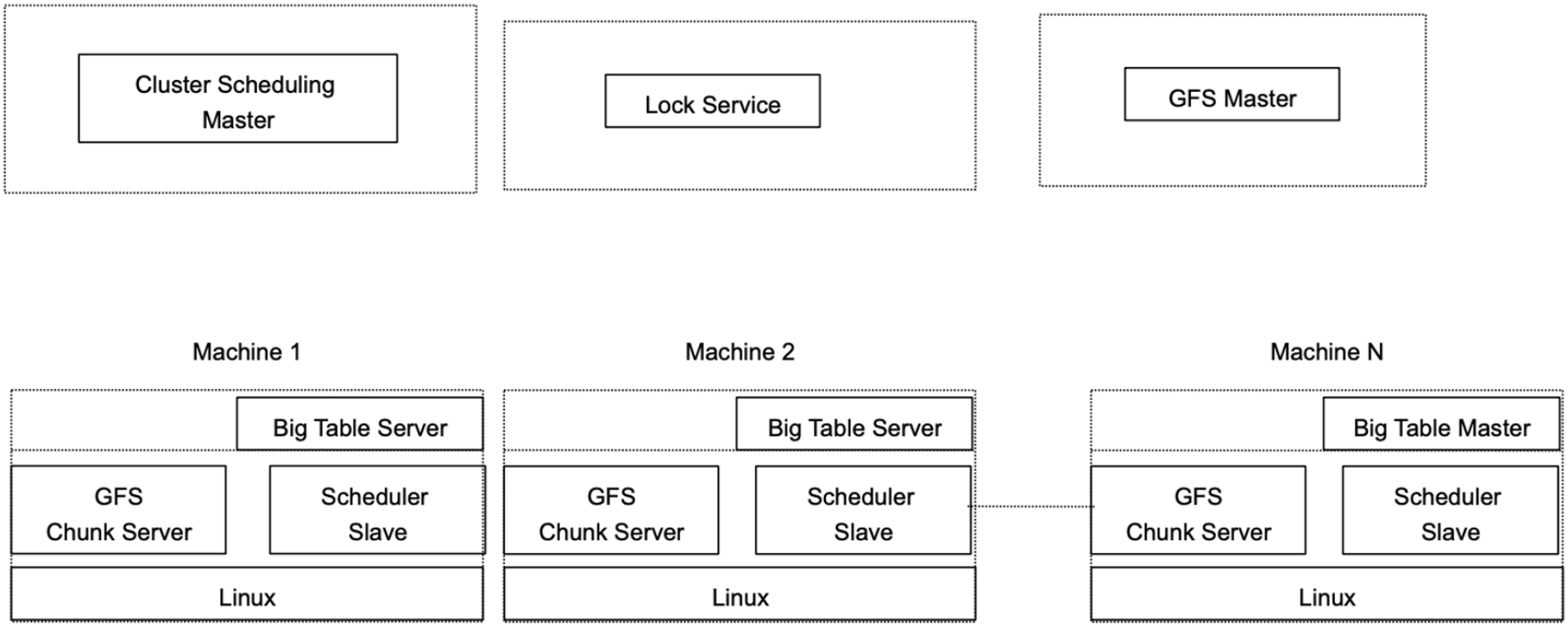
Bigtable系统结构(4)

- 子表服务器TabletServer
  - 子表的装载和卸出
  - 表格内容的读写
  - 子表合并/分裂
  - 管理数据(GFS)
    - 存储操作日志和SSTable



典型Bigtable系统

- 分布式文件存储系统 + 分布式索引层
  - 任务调度、分布式锁(共享信息管理)




JILIN UNIVERSITY·CHINA
1946

- 内容提要
  - Bigtable简介与数据结构
  - 系统架构
  - 数据分布
  - 存储实现
  - 容错和负载分配
  - 对比和应用

表格和子表

- 表格中行的操作是原子的
- 按照行数量将表格分为子表Tablet

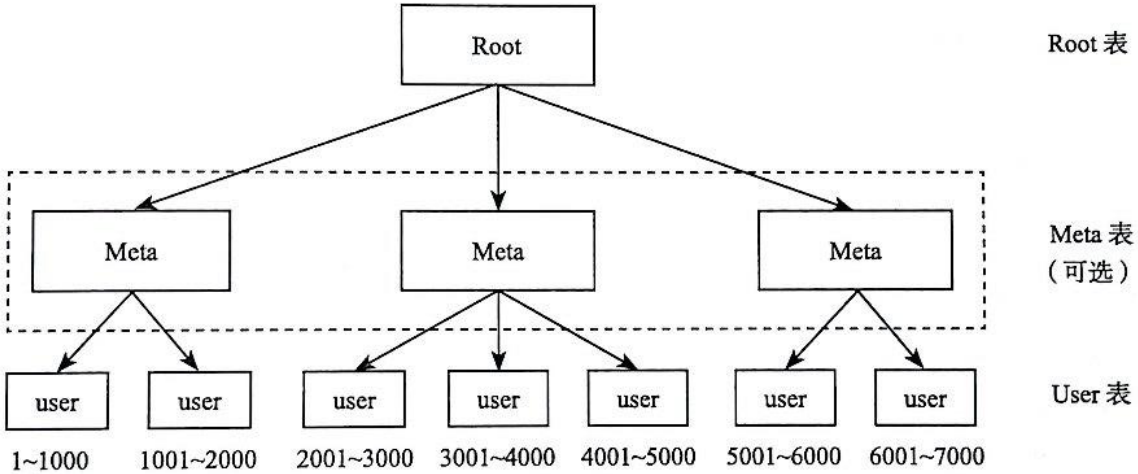
- 子表Tablet：
  - 一定范围内连续的行
  - 数据分布和负载均衡的基本单位
  - 行号相近的子表尽量存储于一台服务器上
  - 访问行号相近的子表，涉及到较少的服务器间通信
  - 需要合理设计row key来保证数据本地化

www.jlu.edu.cn　　ccst.jlu.edu.cn

cn.jlu.edu.www　　cn.jlu.edu.csw

顺序数据分布方式

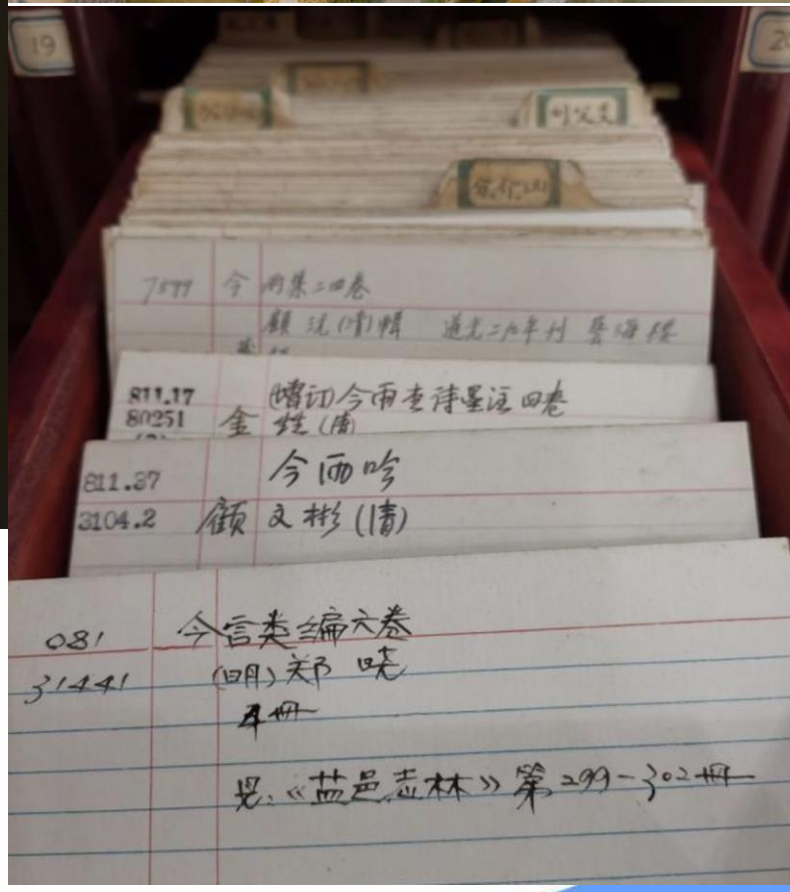
- 大的数据表格按照行的连续范围顺序分成子表
- 顺序分布
  - 数据按照key排序
  - 按照key值连续范围划分成子区域
  - 子区域单独管理
  - 子区域元数据和全局元数据



子表数据分布

- 子表存储连续的数据范围
- 用户可以根据row key的设计来实现数据本地化
- 每个tablet 100-200MB
- 每个server存储10-1000个tablet
  - 快速容错
    - 假设100台服务器，每台服务器100个tablet；每个服务器额外承担1个tablet数据的临时存储。
  - 负载均衡
    - 过载机器上的table被迁移到其他机器
    - 主节点负责制订负载均衡决策

表格数据和元数据



表格数据和元数据

- 用户表：用户数据
- 元数据表：关于用户表的元数据
  - 子表位置信息，SSTable号、操作日志文件标号、日志回放点
- 根(Root)表：关于元数据表的元数据

- 根表的元数据？Bootstrap引导信息(in Chubby)

定位子表(1)

- 如何定位某行数据在集群中的存储位置
  - 子表属性：起始和终止行startrowindex，endrowindex
  - 先查找tablet，再定位row
- 可以通过主节点查找
  - 可能引起主节点性能瓶颈问题

- 其他方案？
  - 子表位置信息存储为元数据表格
  - 元数据表格与其他表格一样存储于BigTable系统中

定位子表(2)

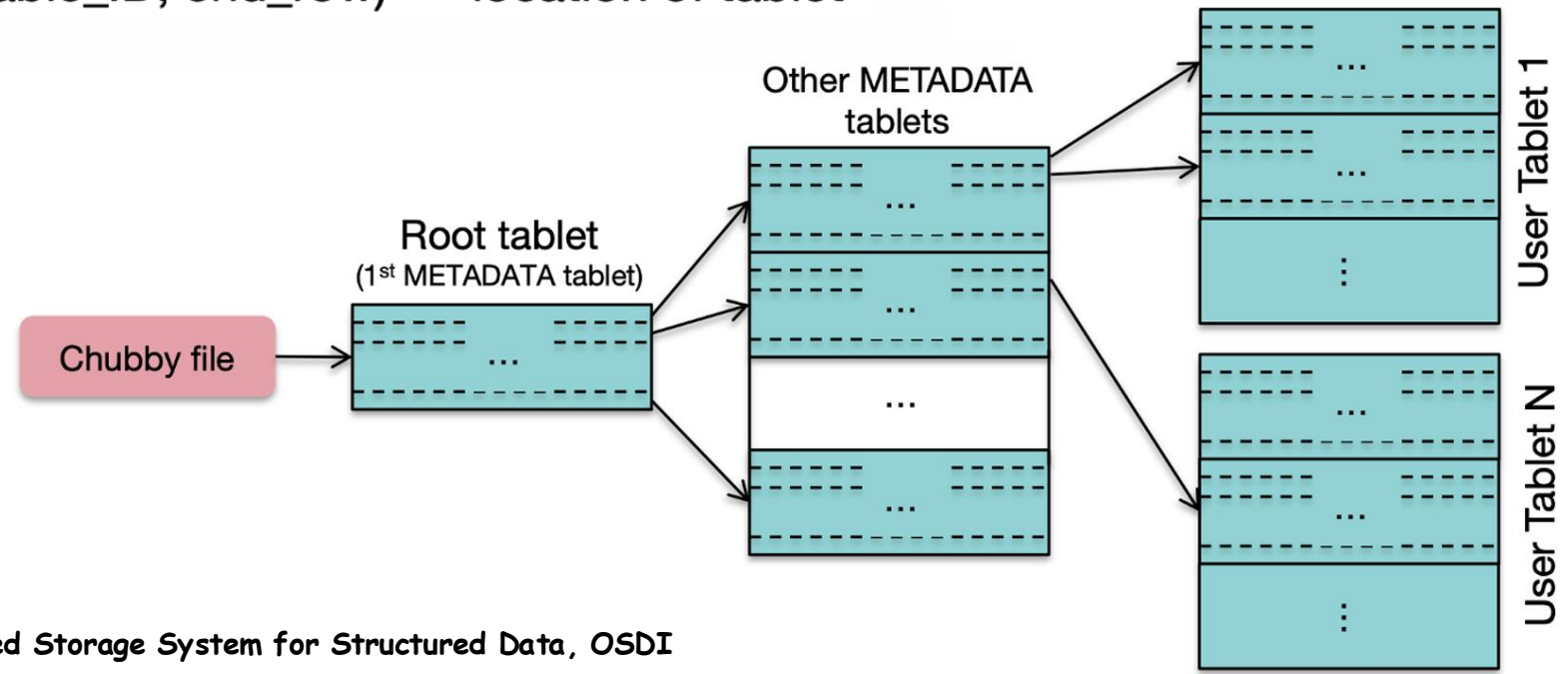
- 如何查找元数据表格信息？
- 将子表存储位置信息(bootstrap)共享在chubby中

- 三层分层查找机制：
  - 位置信息：服务器IP地址
  - 1st层：从chubby服务bootstrap，找到一级元数据位置(Root表)
  - 2nd层：利用一级元数据查找二级元数据表
  - 3rd层：使用二级元数据表查找某表格的子表tablet信息
    - 二级元数据表也可以被分裂成多个子表tablet

定位子表(3)

- 三层分层机制：O（1）
  - 类似B+树的平衡结构
  - Root表存储META表信息(元数据子表)
  - META表存储用户表信息(用户数据子表)

f(table_ID, end_row) ⇒ location of tablet



Bigtable: A Distributed Storage System for Structured Data, OSDI 2006

数据分布讨论

- 支持随机访问(索引结构)
- 支持顺序访问(row key排序)
- 易于实现负载均衡

- **多级元数据结构易于扩展**
  - 子表大小128MB，子表元数据1KB
  - 一级元数据支持数据量128MB*(128MB/1KB)=16TB
  - 二级元数据支持数据量16TB*(128MB/1KB)=2048PB


JILIN UNIVERSITY·CHINA
1946

- 内容提要
  - Bigtable简介与数据结构
  - 系统架构
  - 数据分布
  - 存储实现
  - 容错和负载分配
  - 对比和应用

子表的数据存储

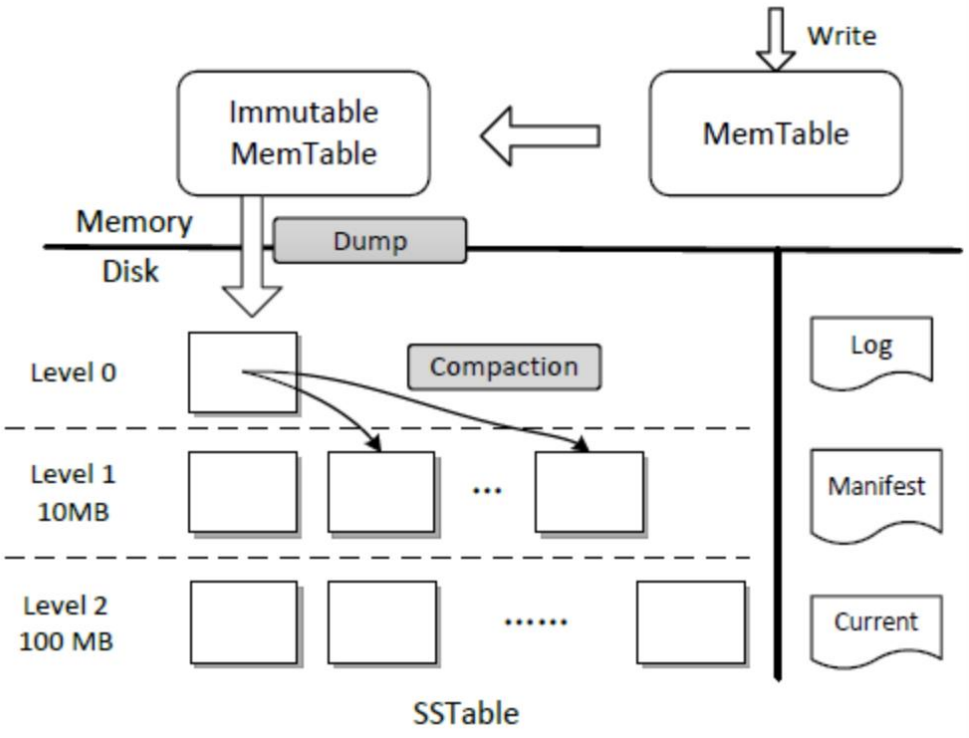
- 每个子表存储一定范围的行（有序）
- 基于SSTable存储

LSM树存储引擎(1)

- 索引：加快读的速度/减慢写的速度
  - 日志结构流派Bitcast
    - 内存容量限制
    - 区间查询性能差（key值无序）
    - <key, offset>，数据无序存储
- 排序key值的索引：排序字符表SSTable
  - 稀疏索引/不在内存存放索引（内存容量限制）
  - Key值是有序的（区间查询性能差）

LSM树存储引擎(2)

- 内存中维护有序的数据结构（平衡树）
  - 实现key值排序功能
- 内存中数据过大时，创建SSTable文件写入磁盘

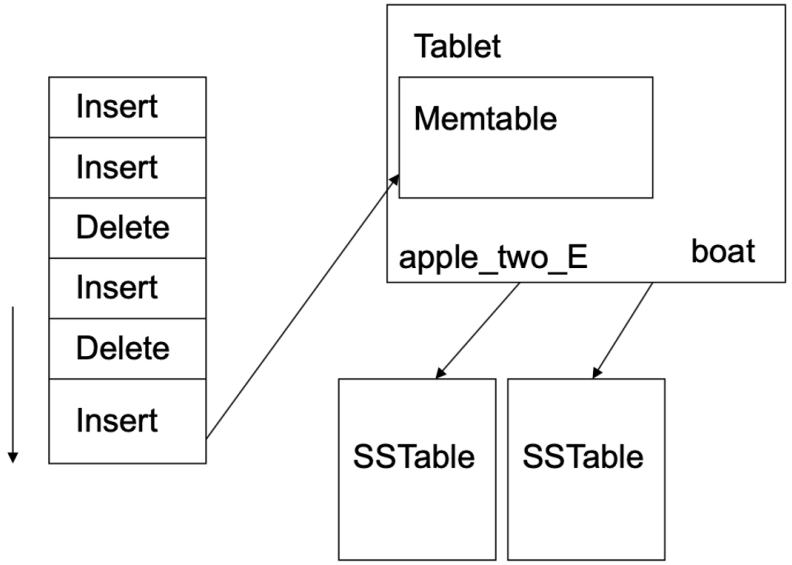


Google SStable

- SSTable(Sorted String Table)
  - 为流式IO和数据传类型<key,value>优化
  - 提供持久的排序索引表
    - 日志流派
  - 存储于内存或外存，日志缓存于内存
  - 对行的增/删/改
    - 日志流派追加
    - 定期合并操作

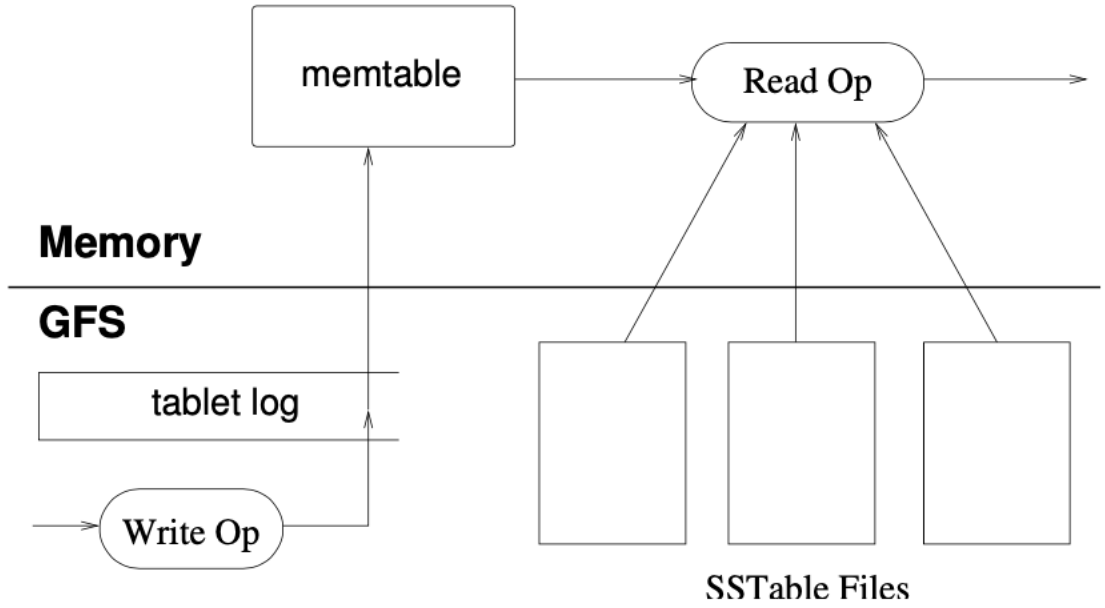
Merge-dump引擎(1)

- 数据写入：
  - 写操作日志
  - 成功后应用到内存Memtable
  - 内存超载dump到SStable文件
  - 从旧到新按照时间顺序合并
  - 定时合并SSTable数据



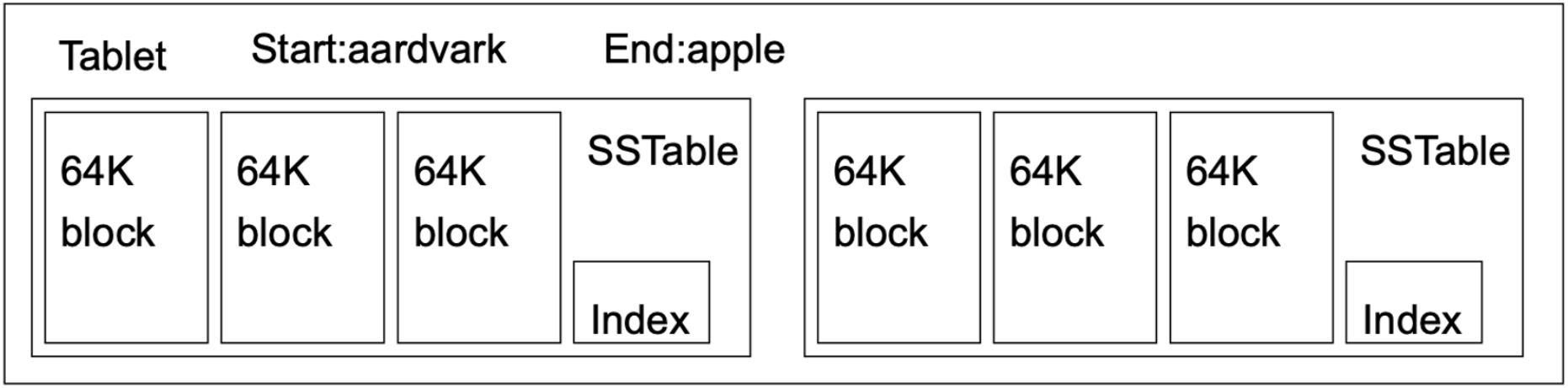
Merge-dump引擎(2)

- 数据读取：
  - 按照时间顺序合并多个相关SSTable记录
  - 返回最终数据结果



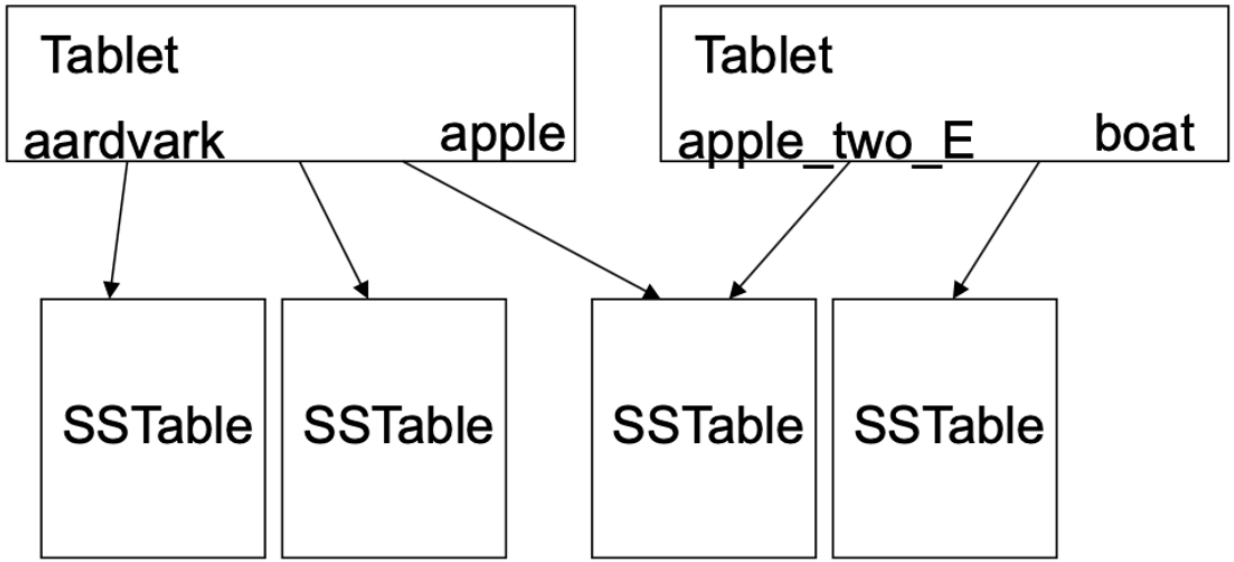
子表的数据存储

- 每个子表存储一定范围的行
- 基于SSTable存储



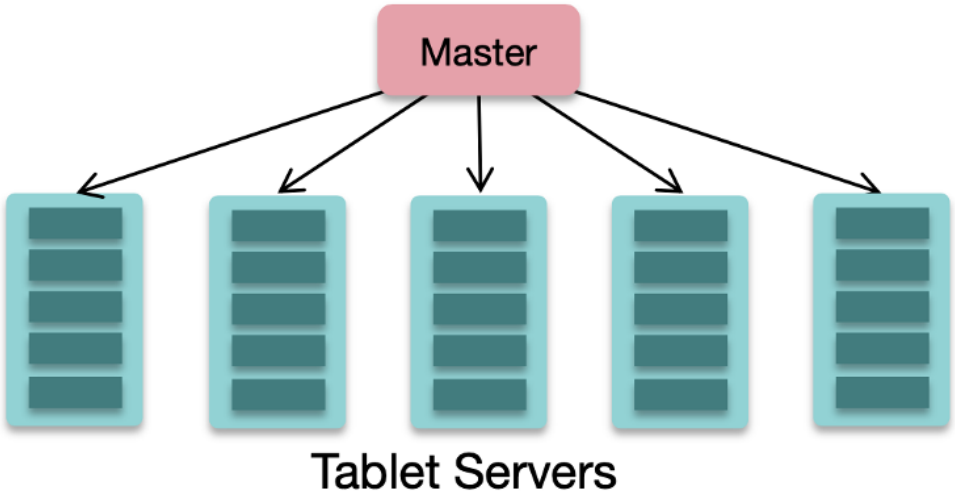
子表的数据存储（续）

- 多个子表格构成数据表
- Tablet不在多个表格之间共享
- Tablet不存在数据重叠，但是SSTable可以重叠



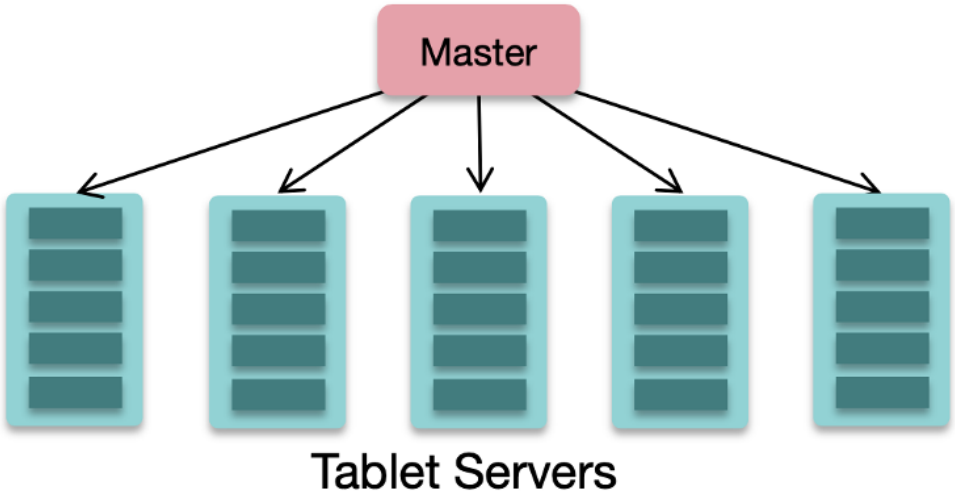
数据分布的实现

- 子表服务器
  - 可以在线进入/退出集群
  - 存储Tablet子表(1 - 1000个)
  - 处理Tablet读写
  - 处理子表分裂/合并



数据分布的实现

- 主节点
  - 分配Tablet到子表服务器
  - 负载均衡（基本单位是子表）
  - 垃圾回收
  - 维护表结构



子表分配

- 每个Tablet只存储在一个子表服务器上
- 主节点：
  - 向chubby申请锁(避免多个主节点)
  - 获取活跃子表服务器信息
  - 获取每个子表服务器存储的子表信息
  - 扫描META表，获取用户数据和子表信息
    - 找到未分配子表信息，建立子表列表，分配负载

GFS持久化存储

- 操作日志：
  - 日志文件存储于GFS中
  - 用于故障恢复过程中回放操作
- SSTable数据：
  - 当内存memtable数据量过大时，以SSTable存储于GFS
  - SSTable文件分块
  - 定期合并回收空间


JILIN UNIVERSITY·CHINA
1946

- 内容提要
  - Bigtable简介与数据结构
  - 系统架构
  - 数据分布
  - 存储实现
  - 容错和负载分配
  - 对比和应用

容错

- 通过GFS和Chubby实现
  - Chubby负责共享信息的一致性管理
  - GFS负责可靠的持久化存储

- 主要错误
  - 子表服务器错误
  - 主节点错误

子表服务器锁

- 子表服务器初始化时向chubby获取锁
- 子表服务器信息存储于chubby
  - Server directory目录
  - 活跃子表服务器，子表服务器分配的子表

子表服务器错误

- 主节点检测到子表服务器工作异常
- 询问状态(锁)
- 如果状态异常或丢失锁
  - 主节点尝试获取该子表服务器锁
  - 如成功，则确认子表服务器出错
  - 子表服务器将子表负载迁移到其他服务器(unassigned state)

子表服务器数据恢复

- 子表服务器的持久化数据依靠GFS恢复
  - 操作日志和SSTable数据

- 数据恢复：回放操作日志恢复内存状态
- 操作日志存储全局表信息
  - <表编号，行主键，日志序列号> 日志
- 按照操作日志，恢复数据状态(内存)

主服务器错误

- 主服务器在chubby租约到期后结束服务
- 集群管理系统检测到主节点无响应

- 选择一个新的主节点，向chubby申请锁
- 继续提供主节点服务

负载均衡

- 子表是Bigtable的负载均衡单位
- 子表服务器定时向主节点汇报状态
- 主节点检测到子表服务器负载过重
- 子表迁移
  - 请求原子表服务器卸载子表
  - 原子表服务器回复卸载成功，或锁过期
  - 选则一个轻负载服务器加载子表
  - 原子表服务器合并内存数据和SSTable数据
    - Minor Compaction

缩短负载均衡的服务暂停

- 迁移子表带来服务暂停（读和写都暂停）

- 如何缩短服务暂定(写操作)
  - 原子表服务器进行minor compaction
  - 过程中仍允许写操作
  - 写操作结束后，对子表再执行一次minor compaction
- 循环合并和“内存脏页”问题？

分裂和合并

- 数据不断写入和删除
  - 子表过大，需要分裂操作
  - 子表过小，需要多表合并操作

- 通过合并和分裂维持顺序分布，维护子表容量相当

分裂

- 每个子表的数据
  - 内存中memtable
  - GFS中多个SSTable

- 分裂：
  - 内存索引信息分成两份(1,10] -> (1,5] (5,10]
  - 形成两个memtable
  - 等Compaction操作时再生成SSTable文件
  - 由子表服务器发起修改，元数据增加一行

合并

- 由主节点发起合并
  - 迁移子表到同一个子表服务器
  - 子表服务器负责合并子表
    - Memtable 和 SStable
    - 更新元数据


JILIN UNIVERSITY·CHINA
1946

- 内容提要
  - Bigtable简介与数据结构
  - 系统架构
  - 数据分布
  - 存储实现
  - 容错和负载分配
  - 对比和应用

讨论

- 优点：
  - 兼顾一致性和可用性
    - Bigtable(一致性C、容错) + GFS(可用性A、可靠性P)
  - 分布式文件存储系统 + 分布式索引层
  - 线性可扩展(数千台)
  - 易于实现故障恢复(1分钟内)
- 缺点：
  - 单副本带来短暂性服务中断(故障恢复)，不适合实时性要求高的场景。
  - 架构复杂，子系统依赖较复杂。

系统对比

	Bigtable表格存储	Dynamo键值存储
数据分布	Key范围顺序	一致性哈希
架构	主从	P2P环
数据模型	表格	键值
接口	Get/Put/Scan/Delete	Get/Put
存储	SSTable GFS	索引
“一句话”	Distributed index (DB) built on GFS	Distributed hash table

应用(1)

Google Analytics

- Raw Click Table (~200 TB)
  - Row for each end-user session
  - Row name: {website name and time of session}
    - Sessions that visit the same web site are sorted & contiguous

- Summary Table (~20 TB)
  - Contains various summaries for each crawled website
  - Generated from the Raw Click table via periodic MapReduce jobs

应用(2)

Personalized Search

- One Bigtable row per user (unique user ID)
- Column family per type of action
  - E.g., column family for web queries (your entire search history!)
- Bigtable timestamp for each element identifies when the event occurred
- Uses MapReduce over Bigtable to personalize live search results

应用(3)

- Google Maps / Google Earth
  - Preprocessing
    - Table for raw imagery (~70 TB)
    - Each row corresponds to a single geographic segment
    - Rows are named to ensure that adjacent segments are near each other
    - Column family: keep track of sources of data per segment
      (this is a large # of columns – one for each raw data image – but sparse)
  - MapReduce used to preprocess data
  - Serving
    - Table to index data stored in GFS
    - Small (~500 GB) but serves tens of thousands of queries with low latency

应用(4)

Apache HBase

- Built on the Bigtable design
- Small differences (may disappear)
  - Access control not enforced per column family
  - Millisecond vs. microsecond timestamps
  - No client script execution to process stored data
  - Built to use HDFS or any other file system
  - No support for memory mapped tablets
  - Improved fault tolerance with multiple masters on standby


JILIN UNIVERSITY·CHINA
1946

- 概论（4）
- 分布式系统基础（8）
  - 分布式系统发展
  - 高性能计算和数据中心（架构与管理）
- 分布式存储技术（9）
- 分布式存储系统（27）
  - 网络文件系统（NFS）
  - 分布式文件系统（HDFS，GFS）
  - 分布式键值对存储（Dynamo）
  - 分布式表格系统（BigTable）
  - 分布式对象存储（S3）

- Thanks!

Hongliang Li

lihongliang@jlu.edu.cn

College of Computer Science and Technology
Jilin University