Tera 资料整理

目录

1 链接存储系统的现状

当前公司采用的链接存储系统有 hadoop 裸文件和 DDBS。链接库 linkbase 采用的 hadoop 裸文件，通过 hadoop hdfs 接口读写文件,并且到数据文件建立索引，可以实现数据的快速读，但是无法实现数据的随机写；链接属性的合并需要采用 hadoop mapreduce 任务进行处理，当前 dlb-saver 和 dlb-select 采用的 MAPRED 批量处理模式链接合并入库到链接选取需要天级处理时间，不能保证价值资源的快速发现和收录。

DDBS 最近两年才开始使用，但使用中遇到了各种问题和限制。DDBS 数据固定分区（分片），分区间无法进行负载均衡，随着业务的扩展收录了更多的新站点，分区负载出现很大的变化，重新调整分区代价很大。

Tera 实现的目标就是提供一个实时的，自动负载均衡的，可伸缩的高性能分布式存储系统，解决上面所描述的问题。

2 Tera 及其特性

Tera 是一个实时的，自动负载均衡的，可伸缩的高性能分布式存储系统，用来管理搜索引擎万亿量级的超链与网页信息。区别于现在 linkbase 链接库使用的 hadoop 裸文件和 DDBS，Tera 突出的特点是实时读写和自动负载均衡。

2.1 Tera 的主要特性

全局有序：整个表格按主键有序，可以高效访问一个小的区间（如获取某个站点的全部链接，只需要和少数节点通信）。

自动负载均衡：系统自动处理局部数据分布不均，对数据量大的热点区域自动分割，将负载到转移到更多机器。（数据分布不均、数据热点区域）

按列存储：表格按照行排列，但可以设置不同的列分别存储（可以在不同的物理介质上），比如调度相关属性列全内存存储（linkcache），链接短属性 flash 存储（linkbase 相关），而长属性可以存储在硬盘上（网页库），提高数据访问效率。

多版本存储：对于部分属性，可以指定存储一定时间内、一定数量的历史版本，用于更新对比和历史问题追查。（snapshot）

其他的特性还有：数据的强一致性，动态 schema，自动垃圾回收等.

3 Tera 的数据模型和架构

3.1 数据模型

首先来看一下传统的关系型数据库的模型，也就是 ER 模型。

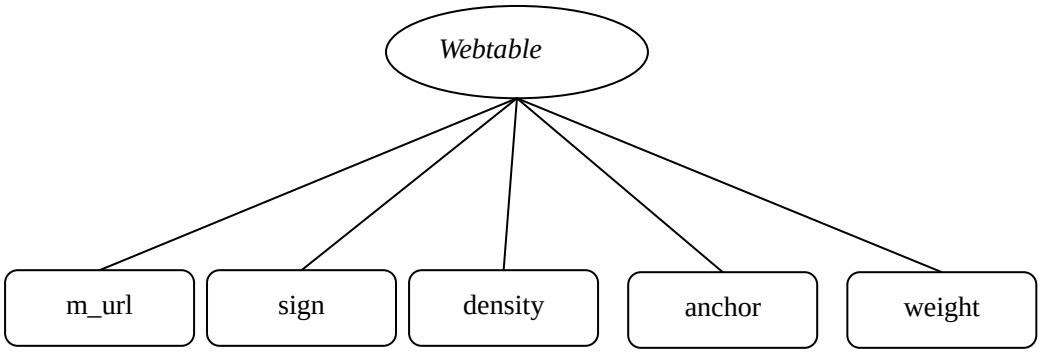

图 1：Webtable 实体模型

体现在数据库中就是如下的数据表以及建立在列上的索引：

表 1：Webtable 表

行　　列	m_url	sign	anchor	weight	density
Row1	com.sina.www	4677879969	新浪首页	30	19
Row2	com.jd.www	8736463483	中国最大的…	31	98

Tera 的数据模型和传统的关系型数据库有可比性，但是又有很大的不同；由于 Tera 底层采用 leveldb 作为数据操纵层，所以数据模型是一个 KV（key-value）模型，Tera 里面表是由 KV 对组成的。Tera 中的 KV 数据模型如下所示：

(row:string, column:string,time:uint64)->string

row 是用户的 key，相当于关系型数据库中的主键，如表 1 中的 m_url，com.sina.www 和 com.jd.www，可以是任意的字符串；

column 是列名称，在 tera 中由 column family 和 qualifier 构成（column

family:qualifier）。qualifier 就是具体的列，如表 1 中的 content 和 anchor。column family 是列簇，简称 CF，通常一个列簇是由一组业务相关、数据类型相近的列组成，一个列簇可包含任意多的列（qualifier），并且列是可以动态增加的；

time 是时间戳，tara 可以存储数据的多版本信息，可以设定数据的存活时间。

数据模型里面还有一个概念是 locality group，局部性群组（LG）。不同的列簇可以属于不同的局部性群组，局部性群组是数据物理存储介质类型的最小逻辑单位，不同的 LG 可以存储在不同的物理介质上。 LG 对 table 进行纵向的分割，使列能够按照不同的用途存储在不同介质上。

下面是一个具体的数据模型的例子。

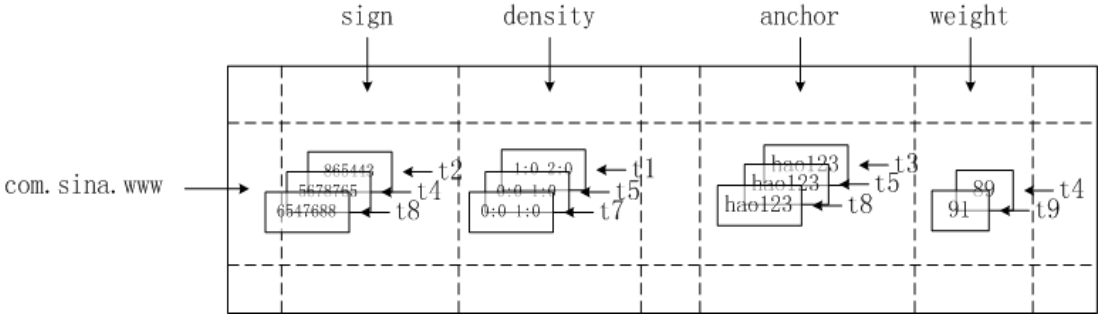

图 2：Webtable 的 Tera 数据模型

如上图中，如果 sign 和 density 属于列簇 cf1，anchor 和 weight 属于列簇 cf2，那么图中 com.sina.www 的 density 列 t7 时刻的 kv 模型实现就是：（com.sina.www cf1: density t7）-> “0:0 1:0”。

可以相比较前面的关系数据库，Tera 里面的 key 相当于主键，但不是单纯的一列作为主键，而是一个复合主键，包含了关系数据库里面讲的主键 row 之外，还有 column 列和时间戳也包含了，这就导致了进行查询的时候要进行 key 的解析的过程。

3.2 Tera 的总体架构

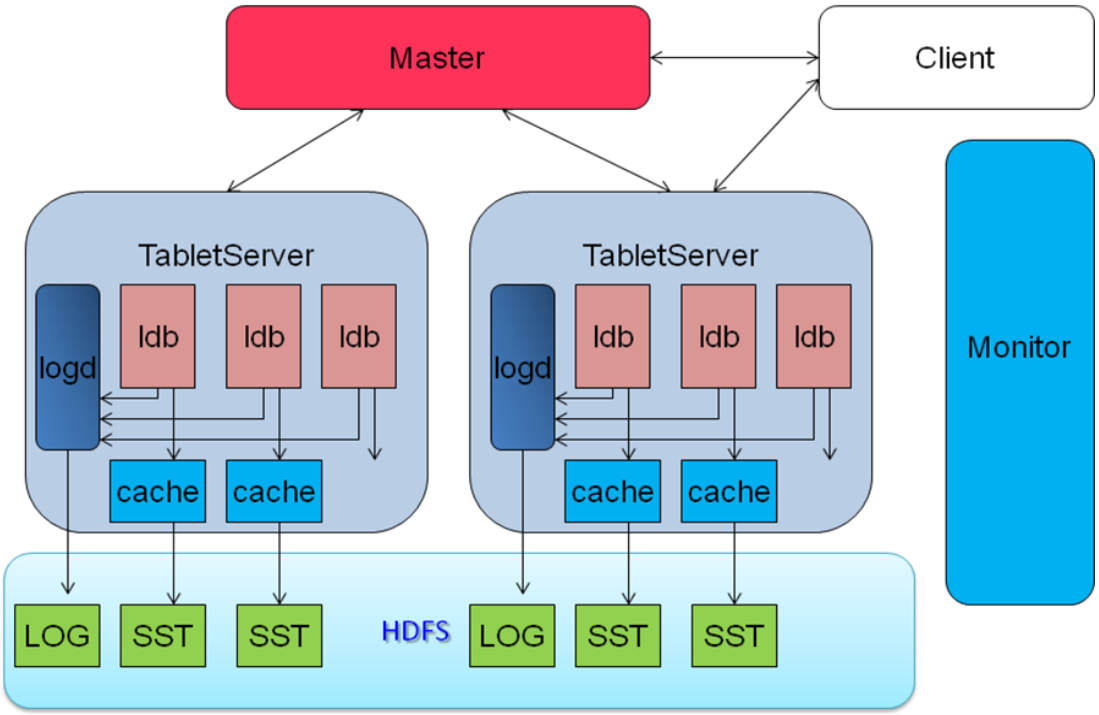

图 3：Tera 架构

Tera 包含三个功能模块和一个监控子系统：

Tabletserver，是核心服务器，负责 Tablets 管理和提供数据读写服务，是系统的数据节点，承载几乎所有客户端访问压力；

Master，是系统的仲裁者，负责表格的管理、schema 更新与负载均衡，用户只有在创建表格和修改表格属性时才访问 Master，所以负载较低，Master 还会监控所有 TabletServer 的状态，发起 Tablet 拆分合并操作和负载均衡操作；

Client，封装系统的各项操作，以 SDK 和命令行工具的形式提供给用户使用，管理员也可以通过 client 对集群行为进行人工干预，如强制负载均衡、垃圾收集和创建快照等；

监控子系统，负责整个表格系统的状态监控、状态展示，同时提供 Web 形式的管理接口，可以实现机器上下线、数据冷备等操作，方便运维。

4 Tera 功能模块

上面一节简要介绍了 Tera 包含的三个功能模块和监控子系统，这一节详细介绍三个功能模块的具体实现。

4.1 Master

Master 最重要的功能就是表格管理、tabletnode 管理和负载均衡管理这三个方面。

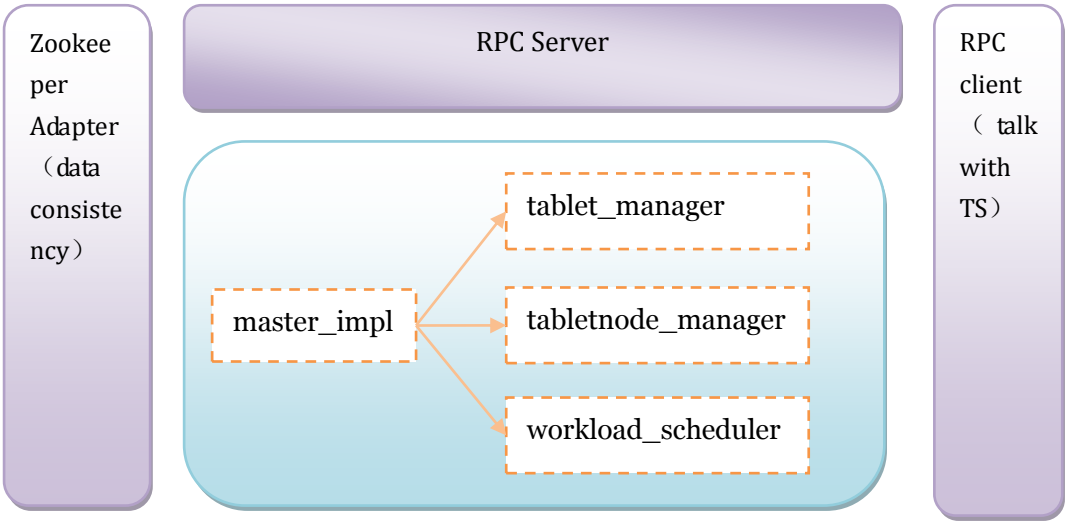

图 4：Tera Master 实现模型

4.1.1 表格管理

上面我们已经介绍了 Tera 的数据模型，也就是 tera 中 Table 的逻辑模型，也讲到 tera 保证数据按照主键（key）全局有序。下面介绍 tera 中 table 是如何管理的。

Tera 将数据表横向划分为若干个有序的区间，每一个区间就是一个 tablet，是数据分布和负载均衡的最小单位。Master 根据每个数据节点的负载状况，将 Tablet 安排到各个 TabletServer 上，每个 TabletServer 管理着若干个 Tablet，提供这些 Tablet 的读写服务，负责将数据持久化到 DFS 上。区间内的有序性由 tablet 自身保证，tablet 之间的有序性通过 master 来维护。

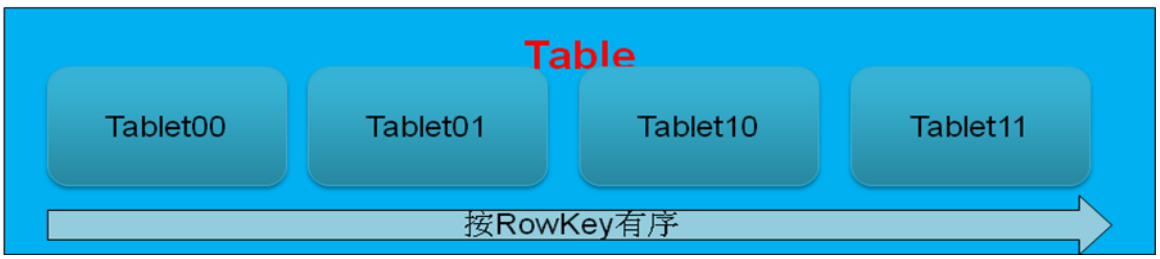

图 5：Tera 数据全局有序性

一个表格包含多少个 tablet，每个 tablet 被安排到哪个数据节点上，这些属于系统的 meta 信息，meta 信息存储在一个叫 meta_table 的表里，meta 表和普通表一样，也可以包含多个 tablet，所有数据的写入也都会落地到 hdfs 上。meta_table 的 tablet 存储在 root 表中，root 表地址保存在 zookeeper 上，在系统启动时 master 先从 zookeeper 上找到 root_table，并调度加载，root_table 内记录了 meta 表包含 tablet 的地址，master 读取这些信息后就可以完成整个 meta 表的加载，从而获得了系统中所有表格及其 tablets 的分布信息。

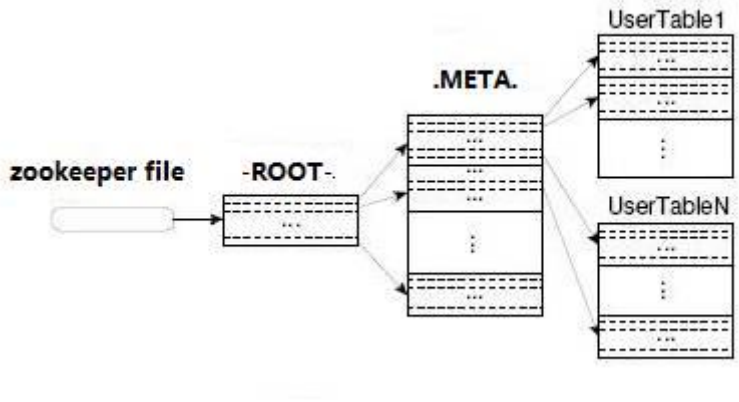

图 6：Meta 表

Root table 是一个不可分裂的 tablet，包含以下内容：

Key	列族 1：基本信息			列族 2：列属性				列族 3：快照			
表名#起始 Key	End-Key	Loca-tion	Path	列族 1	列族 2		列族 n	快照 1	快照 2		快照 n
	结束 Key	所在 TS	HDFS 路径	属性 1	属性 2		属性 n	TAG1	TAG2		TAGn

4.1.2 Master 的具体实现

Master 实现的代码在 master_impl 模块中，master_impl 通过 tablet_manager、tabletnode_manager 和 workload_scheduler 进行表格管理、tablet 节点管理和负载均衡管理等工作。

tablet_manager 实现表格信息的管理，拥有一个 all_table_list 成员，是一

个 table name 到 Table 指针的 hash 表；每一个 Table 拥有一个 m_tablets_list 数据成员，是一个 start_key 到 Tablet 类指针的 hash 表，而一个 Tablet 类包含了 Tablet 的 meta 信息，包括 tablet 的起始 key，结束 key，TS 地址，tablet 的状态、大小、是否压缩等详细信息。

通过这样的层次结构，tablet_manager 就拥有了从 table 名称到该 table 的所有 tablet 信息的映射关系。简单的说 all_table_list 就是 meta 表在 tablet_manager 内存中的数据结构，所有对内存中 all_table_list 的操作都会同步更新到 meta 表中。

tablet_manager 模块实现了如下功能：装载 meta table 信息，dump meta table 到本地，添加、删除、查找 table 和 tablet，合并 tablet，分裂 tablet 等。

tabletnode_manager 实现 tablet 节点的管理，拥有一个 m_tabletnode_list 的数据成员，m_tabletnode_list 是一个 server address 到 tabletnode 的 hash 表，TabletNode 类管理 tabletnode 的信息和相应的操作，TabletNode 包含一个 tabletnode 的状态，数据大小，地址，uuid 等。

tabletnode_manager 类实现了如下功能：添加、删除、更新 tabletnode，获得所有 tabletnode 的信息等。

Master_impl 通过 LoadBalanceTimer 定期（10s）对 tablet 信息进行监控，对满足条件（tablet 的 data_size 大于 512M）的 tablet 进行 split;

tablet_manager 通过 MergeTabletTimer 定期（180s）对 teblet 信息进行扫描，对满足条件的 tablets 进行合并。

3.2 Tablet Server

Tablet server 是 tera 的核心服务器，负责 tablets 管理和提供数据读写服务。Tablet Server 最重要的功能就是通过 TabletIO 提供数据读写服务。

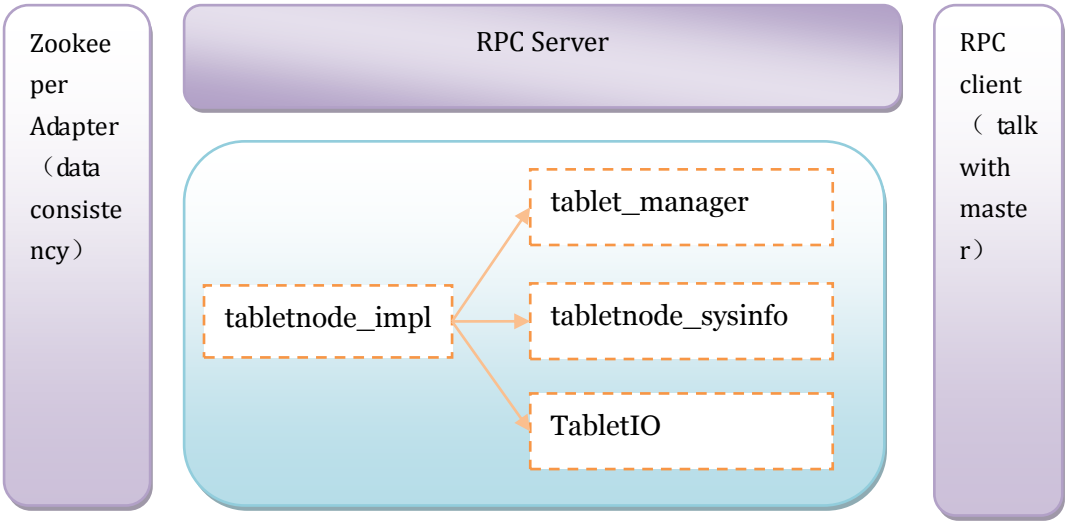

图 7：Tablet Server 实现模型

tablet server 在架构上和 master 是差不多的，他们都是 TeraEntry，但是角色不同，master 负责表格信息管理和负载均衡的角色，而 tabletnode 充当的是提供数据读写服务的功能，几乎所有的数据读写服务都是通过 tabletnode 提供的。

3.2.1 tablet_manager 实现

Tablet Server 通过 tablet_manager 管理 TabletIO，tablet_manager 拥有 m_tablet_list 成员变量，m_tablet_list 是 TabletRange（tablename，startkey，endkey 的三元组）对象到 TabletIO 的 hash 表，每一个 TabletIO 就是一个 leveldb 实例。

tablet_manager 提供如下的方法：

AddTablet 增加一个 tablet

RemoveTablet 删除一个 tablet

GetTable 通过 table name 和 key 获得一个 TabletIO 指针

GetAllTabletMeta 获得所有 Tablets 的元信息

GetAllTablets 获得 TS 中所有 TabletIO 指针

RemoveAllTablets 移除所有的 Tablets

Size 获得 TS 中 Tablet 的数量

3.2.2 tabletnode_impl 实现的主要方法

Init() 初始化 tablenode 实例，向 ZK 注册 TS 节点和端口

InitCacheSystem() 初始化 TS 的 leveldb cache 机制

Exit() 退出实例，关闭所有的 tablet

Register 向 master 注册 tabletnode 信息（好像没有找到实现）

Report 向 master 汇报 tabletnode 统计信息

LoadTablet 接收 master 的 loadtablet request，进行请求有效性的检查，

通过 tablet_manager 添加该 tablet，添加成功之后，通过 tabletIO load 指定的 tablet，tabletIO load 失败，tablet_manager 删除该 tablet，返回装载 tablet 成功。

UnloadTablet 卸载 tablet

CompactTablet 紧缩 tablet

ReadTablet 读 tablet 数据

WriteTablet 写入数据

GetSnapshot 获取快照

ReleaseSnapshot 释放快照

Query 响应 master 的 query 请求，返回 tabletnode 的相关信息（监控、meta 信息）

ScanTablet 对 Tablet 进行数据扫描，返回扫描结果

SplitTablet 对 Tablet 进行快速分裂

MergeTablet 合并连个 Tablet

Tabletnode 除了提供数据读写服务外，还定期向 master 汇报 tabletnode 本身的系统信息，包括 ts 的硬件信息（内存占用，网卡信息，cpu 等）和 tablet 的统计信息（所有 tablet 的数据大小，tablet 读写次数、读和 scan 任务 pending 的数量等等）

3.2.3 TabletIO

TabletIO 是 tera 中最接近 leveldb 的模块，是 tera 对 leveldb 的上层封装，提供 tablet 装载、卸载、split、compact、merge 等接口和数据读写 scan 服务，同时通过 StatCounter 记录 tablet 操作的统计信息。

Load 装载指定的 tablet

TabletIO 拥有一个 m_db 的 Leveldb::DB 指针，在 load tablet 的时候设置相关 ldn 参数后打开指定路径下的 leveldb，leveldb::DB::Open(m_ldb_options, m_tablet_path, &m_db)，如果打开失败，会尝试进行 repair 修复打开

Unload 卸载本 tablet

等待所有的写结束后，关闭 leveldb 实例，关闭 tablet 的写进程

Split 分裂 tablet

Tera 采用快速分类的方式，在 tabletIO 层面 split 只是通过 leveldb 找到一个分裂的 key，将状态设置为 kTableonSplit

Compact 执行 major compact

CompactMinor 执行 minor compact

Read 根据 key 读取数据

LowLevelScan 根据 start_key、end_key 和 filter 进行扫描

ReadCells 调用 LowLevelScan 获得多行数据

3.3 SDK/Client

Tera SDK/Client 封装系统的各项操作，以 SDK 和命令行工具的形式提供给用户使用，管理员也可以通过 client 对集群行为进行人工干预，如强制负载均衡、垃圾收集和创建快照等。

一般来说数据库系统有 DDL、DML 和 DCL 三种操作，目前 Tera 支持的有数据定义语言（DDL），包括创建表、删除表、修改表 schema、创建快照等和数据操作语言（DML），包括了读 GET、写（PUT、ADD、PutIfAbsent）、扫描（SCAN）等。

4 Tera 的分裂和合并过程

Master_impl 通过 LoadBalanceTimer 定期（10s）对 tablet 信息进行监控，对满足条件（tablet 的 data_size 大于 512M）的 tablet 进行 split;

tablet_manager 通过 MergeTabletTimer 定期（180s）对 teblet 信息进行扫描，对满足条件的 tablets 进行合并。

4.1 Tera 的分裂过程

Tera 在两种情况下进行 Tablet 分裂，一种是 Master 定期执行负载均衡过程中对符合条件的 Tablet 进行分裂，另外一种情况是管理员通过 Tera Client 强制执行分裂。下面从 Master 负载均衡的过程介绍 Tablet 的分裂过程。

1 通过 LoadBalanceTimer 定期（每 10s）执行 LoadBalance

2 Master 通过 tablet_manager 获得所有 table 和 tablet 的信息和通过 tablenode_manager 获得所有 tabletnode（ts）的信息

3 如果只是对特定 table 进行 loadbalance（默认是），如果对所有 table 进行均衡，调转到 7

4 获得 table 下面的所有 tablet，创建 tablet server 到 tablet 列表的 hash 表

5 根据 table 对每一个 tabletnode 按照负载降序排列，这样负载大的优先进行负载均衡

6 对每一个 tabletnode 和 table 对应的 tablet 调用 TabletNodeLoadBalance 进行负载均衡

7 对所有 tablet 不分 table 调用 TabletNodeLoadBalance 进行负载均衡

TabletNodeLoadBalance

1 对于 tabletlist 中的每一个 tablet 判断 tablet 的大小是否比设定的 splitsize 要大，如果要大并且 tablet 不在进行 compact，执行 TrySplitTablet

2 如果存在 split 的 tablet 或者设置 master 不能 move tablet 返回，否则

3 将 size 最小的 tablet 移动到负载最小的 tabletnode 上面

TrySplitTablet

1 获得 tablet 所在的 server address

2 通过 tablenode_manager 查看 server 是否在服务

3 tabletnode_manager TrySplit,将 tabletnode 的 datasize 减小 tablet 的 size，查看等待 split 的 tablet 队列大小是否小于设定的最大同步 split 限制，如果超过了放到等待 split 队列里面，

4 设置 tablet 的状态为 kTableOnSplit

5 执行 SplitTabletAsync(tablet)

MasterImpl::SplitTabletAsync

1 向 tabletnode server 发送 SplitTabletRequest 请求 SplitTablet

2 在回调函数中根据 tablet 的 startkey 和 endkey 查询 meta 表信息

3 如果返回记录数大于 2 或者等于 0 个表明 split 不成功，修复 meta 表信息，

4 如果返回是 1,表示 split 还没有完成，重新 load tablet

5 返回的结果是 2 个记录，说明 split 成功了，tablet_manager 添加第二个 tablet 信息，删除第原来的 tablet，添加第一个 tablet 信息，tabletnode 装载第一个分裂后的 tablet，装载第二个分裂后的 tablet。

TabletNodeImpl::SplitTablet

1 根据 request 中的 tablename 和 startkey、endkey，获得 TabletIO 指针

2 tablet_io 找到分裂的 splitkey

3 tablet_io unload

4 tablet_manager remove 相应的 tablet

5 更新 meta 表信息 UpdateMetaTableAsync

6 返回 MasterImpl::SplitTabletAsync

4.2 Tera 的合并过程

Master 的 Tablet_manager 通过 MergeTabletTimer 定期（180s）执行 TryMergeTablet，对符合条件的两个 Tablet 进行合并。

Master

1 tablet_manager 找到根据 table name 找到 table

2 调用 table 的 FindMergePair 找到两个符合合并条件的 tablet（两个 tablet 的区间相连并且文件大小小于 merge 设定的值最大值，大小之和小于某一个设定值）

3 设置 tablet1 和 tablet2 的状态为 kTableOnMerge

4 merge 第 1 步 try unload tablet1

5 merge 第 2 步 try unload tablet2

6 merge 第 3 步 在 tablet1 的 server 上面合并 tablet1 和 tablet2（如果 tablet1 和 tablet2 不在一个 tablet server 上则 mv tablet2 到 tablet 的 TS 上，如果失败，调度两个 tablet 到另外一个 tablet server 上执行合并）

Tablet Server

调用 MergeTablesWithLG 合并两个 tablets，返回 Master

Master

将 tablet1 的区间设置为 tablet1+tablet2 的区间，在 tablet1 上面 TryLoadTablet tablet1，将 tablet2 的 meta 信息删除更新 meta 表。

5 附录一

5.1 Master 是如何启动的

0 master 抢 master 锁，成功则将 master 地址写入 master

1 从 zk 上获取 meta_table_addr 和 tablet server 列表

2 从 tablet server 列表里面 query tablet server 的信息，创建 Tabletnode_manager 的 tabletnode 信息，包括所有的 tablet，以及 ts 管理的所有 tablet，其中包含 meta_table

3 从上一步返回的 tablet list 中加载 meta_table，创建 tablet_manager 需要的信息 从 table->tablet 的映射表

4 load 用户的 tablet,对 ts 返回的 tablet 信息和 meta 表信息进行校验，如果不对，比如 meta 表里面不存在 ts 回报上来的 table 或者区间不匹配，以 master 为准，卸载不合法的 tablet

5 尝试启动 offline 的 tablet，检查下线的 tablet 的比例，如果 offline 的比例超过阀值，进入 safemode 模式。

5.2 Leveldb 的读写和 Compaction

Leveldb 的写

Commitlog -> memtable(skiplist) 根据用户提供的 key comparator 保持有序，将 key 插入到相应的位置

当 memtable 的大小超过阀值，memtable 变成 immemtable，同时生成新的 memtable 和 log 文件，因为 leveldb 的写只是写 log 和一次 memtable 的内存写，所以写入非常快。

Leveldb 的读

按照文件的新鲜度逐层查找

Memtable->immemtable->level0（有重叠）-> 其他 level

Leveldb 的 compaction

Minor compact

Immemtable dump 成 sst，形成 level0 的一个文件

Major compaction

当各个 level 的文件数目（level0）或者文件大小（大于 level0）超过一定阀值时，触发 major compact。对 level0 来说，比较特殊，选文件的时候需要把重叠的文件都选上，和 level1 的文件进行多路合并。

6 附录二

什么是 Tera，Tera 的产生背景，需要解决的问题是什么，Tera 的特性是什么？

Tera 实现了哪些功能（自动负载均衡，自动切片，实时写入，数据强一致性，split，merge）

名词解释：

Tera、master、Tablet、tablet server（TS， aka tablet node）、LG、column family，qualifier、root table、meta table、zookeeper、RPC、protocol buff、leveldb、sst，dfs， memtable、immemtable，bigtable

Tera 的数据模型，root table， meta table， table 的数据模型，key-value 是怎么样组成的，Meta 表的内容是怎么构成的

Tera 的总体架构是怎么样，能够画出 Tera 的架构图；Tera 由几部分构成，各部分的架构又是怎么样的，包含哪些模块

Tera 各部分是如何工作的（启动、停止、split，merge 流程）

Tera 支持的操作

数据定义语言（DDL）：创建 table、删除 table、修改 table schema

数据操作语言（DML）：写入（put，受底层 leveldb，删除就是标记删除）、读取（GET）、扫描（SCAN）

数据控制语言（DCL）：目前没有支持

Tera 支持的操作都是如何实现的，应该各举一个例子说明具体的流程是怎么样

Tera 的监控功能，对系统内存、CPU、硬盘空间等信息的监控和对 ts 自身统计信息的监控

TabletIO 底层的具体是怎么实现的

Tera 的日志改造项目是什么样的情况

异常处理

Cache 机制