14

内存计算模型

14 内存计算模型

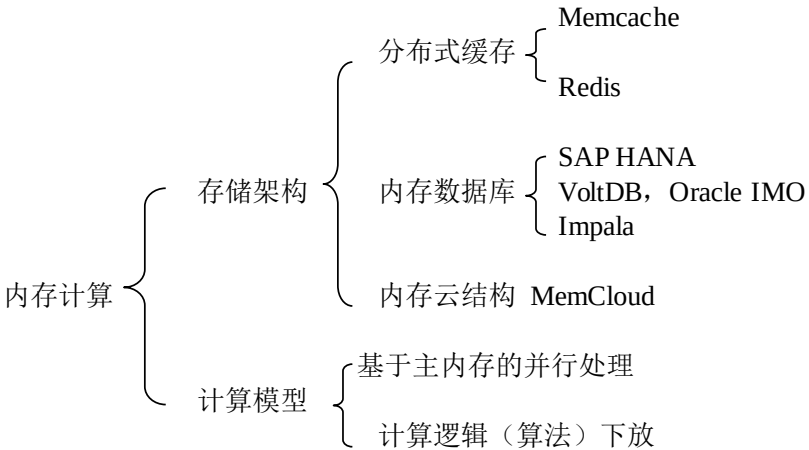
- 内存计算模型
- 分布式缓存体系
- 数据存储一致性
- 内存数据库
- MemCloud计算架构

14.1 内存计算模型

- 内存计算概念

内存计算（In-memory Computing）指采用了各种内存技术在计算过程中让CPU从主内存（main memory）而不是从磁盘（disk）读写数据的计算模型。这里的内存技术包括列存储格式、数据分区与压缩、增量写入、无汇总表等方法。

目前内存计算主要是从存储架构（分布式缓存、内存数据库、内存云体系）和计算模型（基于主内存的并行处理、算法下放到数据层）两个方面提出解决方案



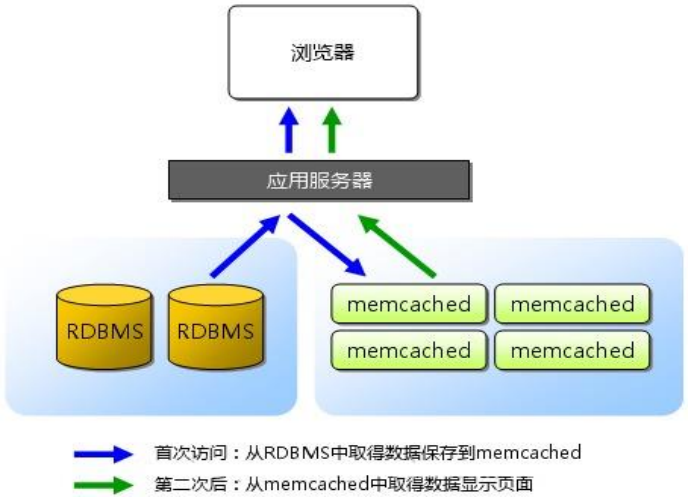
14.2 分布式缓存体系

- 分布式缓存体系

分布式缓存系统（Distributed Cache System）包含两层含义：

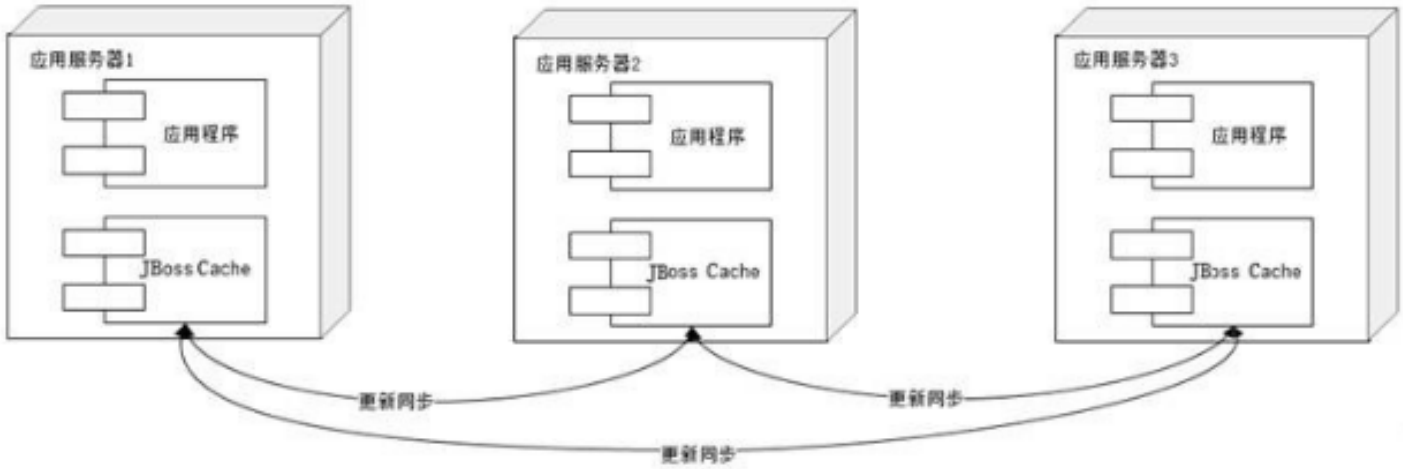
1）由多台服务器组成一个缓存服务器集群，以多节点集群方式提供缓存服务，即物理架构上是分布式；

2）缓存数据（可看作一个大数据表）被分布式存储在多台缓存服务器上，即逻辑架构上也是分布式的。



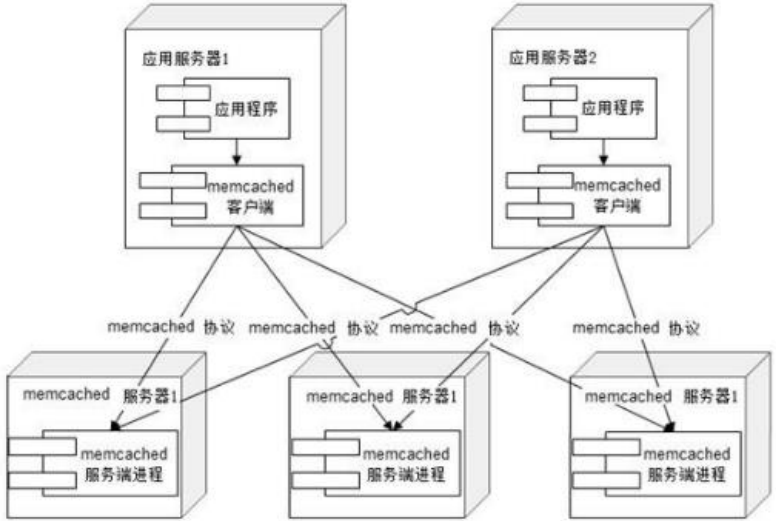
14.2 分布式缓存体系

数据同步缓存系统以JBoss Cache为代表[9]，如下图所示。JBoss Cache的缓存服务器集群中所有节点均保存一份相同的缓存数据，当某个节点有缓存数据更新的时候，会通知集群中其他机器更新内存或清除缓存数据。



14.2 分布式缓存体系

Memcached则采用了数据不同步的架构，如下图所示。Memchache采用一组专用缓存服务器，缓存与应用分离部署。在存放和访问缓存数据时，应用程序通过一致性Hash算法选择缓存节点，集群缓存服务器之间不通信，也不需要数据同步，因此集群规模可以很容易地实现扩容，具有良好的可伸缩性。



14.2 分布式缓存体系

● 内存技术

在系统实现方面，分布式缓存系统主要通过如下的内存关键技术来实现数据的快速访问

◆ 数据压缩存储

包括字典编码算法、高效压缩存储、数据操作等。以下图为例介绍字典编码基本原理

Row ID	Date/ Time	Material	Customer Name	Quantity
1	14:05	Radio	Dubois	1
2	14:11	Laptop	Di Dio	2
3	14:32	Stove	Miller	1
4	14:38	MP3 Player	Newman	2
5	14:48	Radio	Dubois	3
6	14:55	Refrigerator	Miller	1
7	15:01	Stove	Chevrier	1

#	Customers
1	Chevrier
2	Di Dio
3	Dubois
4	Miller
5	Newman

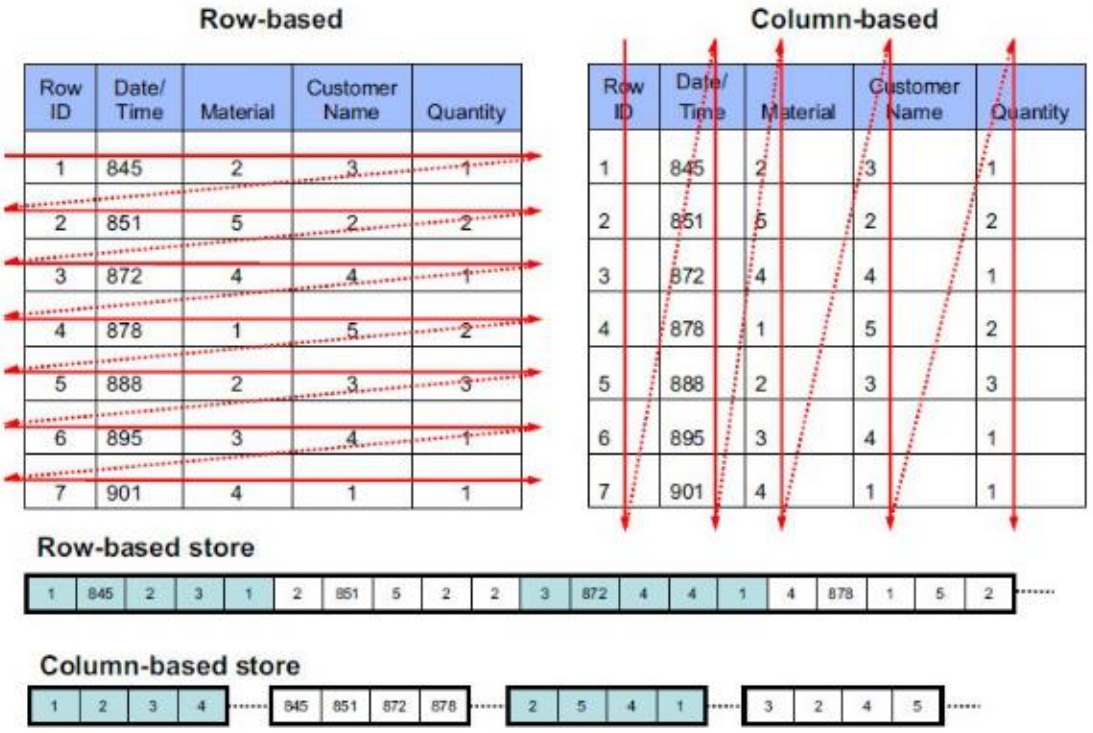
#	Material
1	MP3 Player
2	Radio
3	Refrigerator
4	Stove
5	Laptop

Row ID	Date/ Time	Material	Customer Name	Quantity
1	845	2	3	1
2	851	5	2	2
3	872	4	4	1
4	878	1	5	2
5	888	2	3	3
6	895	3	4	1
7	901	4	1	1

14.2 分布式缓存体系

◆列存储结构

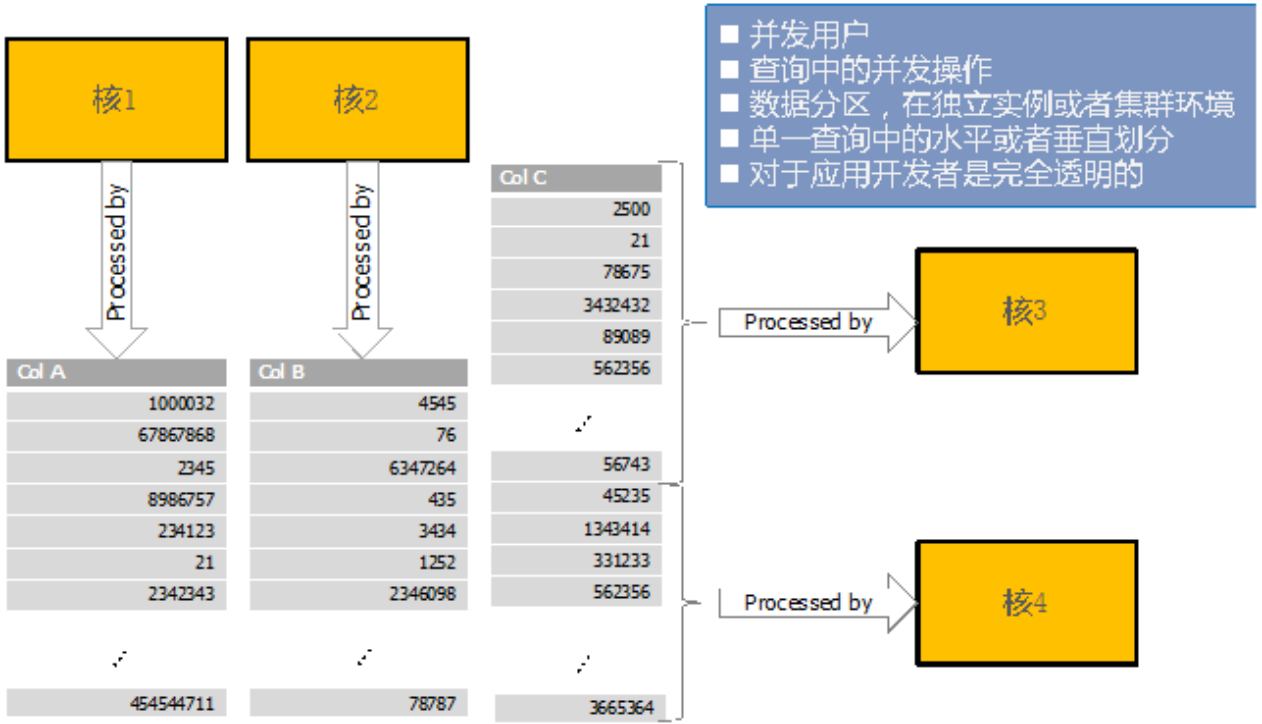
包含内存数据格式、内存索引等技术。在经过压缩后，在内存空间内的存储方式有行存储（Row-based store）和列存储（Column-based store）两种方式。



14.2 分布式缓存体系

◆分区

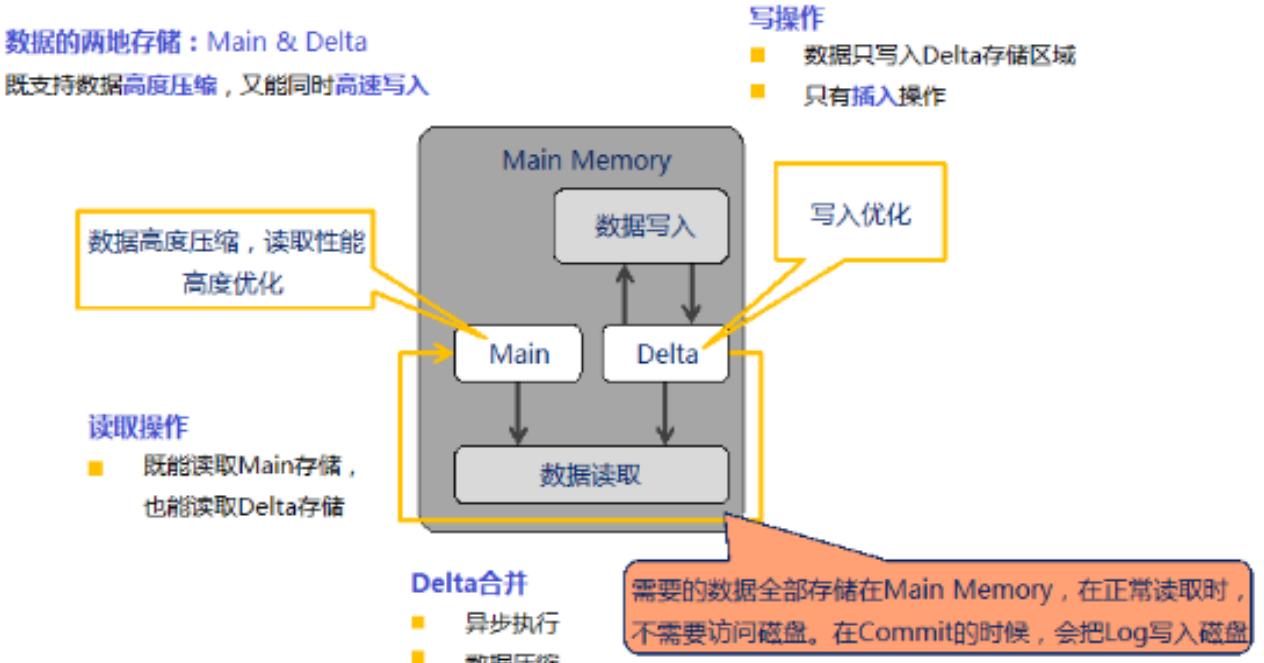
对数据表的划分及多节点并行处理。分布式缓存系统主要采用水平划分和垂直划分两种方式。



14.2 分布式缓存体系

◆ 只插入差异数据

分布式缓存系统还常常采用一种只写入差异数据技术来提高访问效率。如图所示，在内存中划分两个区域：主表（Main）和差异表（Delta）。主表包含完整的数据，采用高度压缩的列存储方式，支持高效率的读数据操作；差异表只包含少量的新增数据，支持写数据操作。



14.2 分布式缓存体系

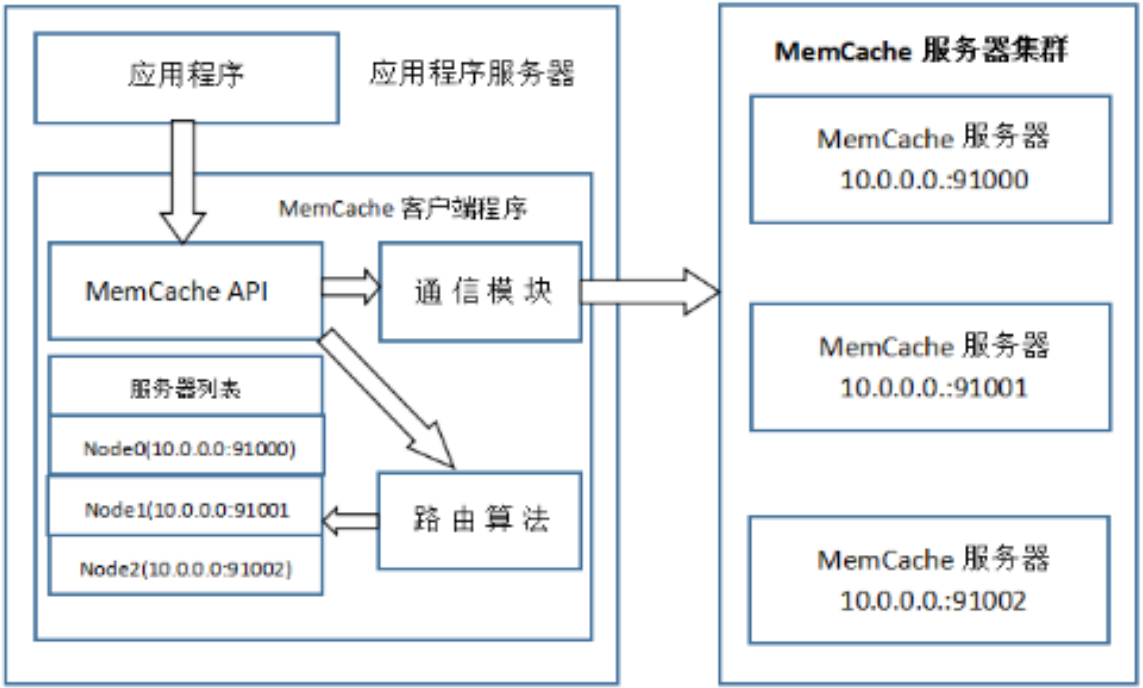
Memchache工作机制

Memcache 是一个有代表性的高性能分布式内存对象缓存系统，它通过缓存数据和对象来减少读取数据库次数，从而提高数据库驱动网站的访问速度。Memchache采用一组专用缓存服务器，缓存与应用分离部署。在存放和访问缓存数据时，应用程序通过一致性Hash算法选择缓存节点，集群缓存服务器之间不通信，也不需要数据同步。

Memchache的计算架构中，其计算系统由应用服务器和缓存服务器集群组成，应用服务器上部署应用程序和Memchache客户端，缓存服务器上部署memchached服务器程序。应用程序通过Memchache API向Memchache客户端提交访问任务，客户端通过通信模块与缓存服务器集群连接，并基于路由算法选择一个Memchache服务器节点执行访问任务。

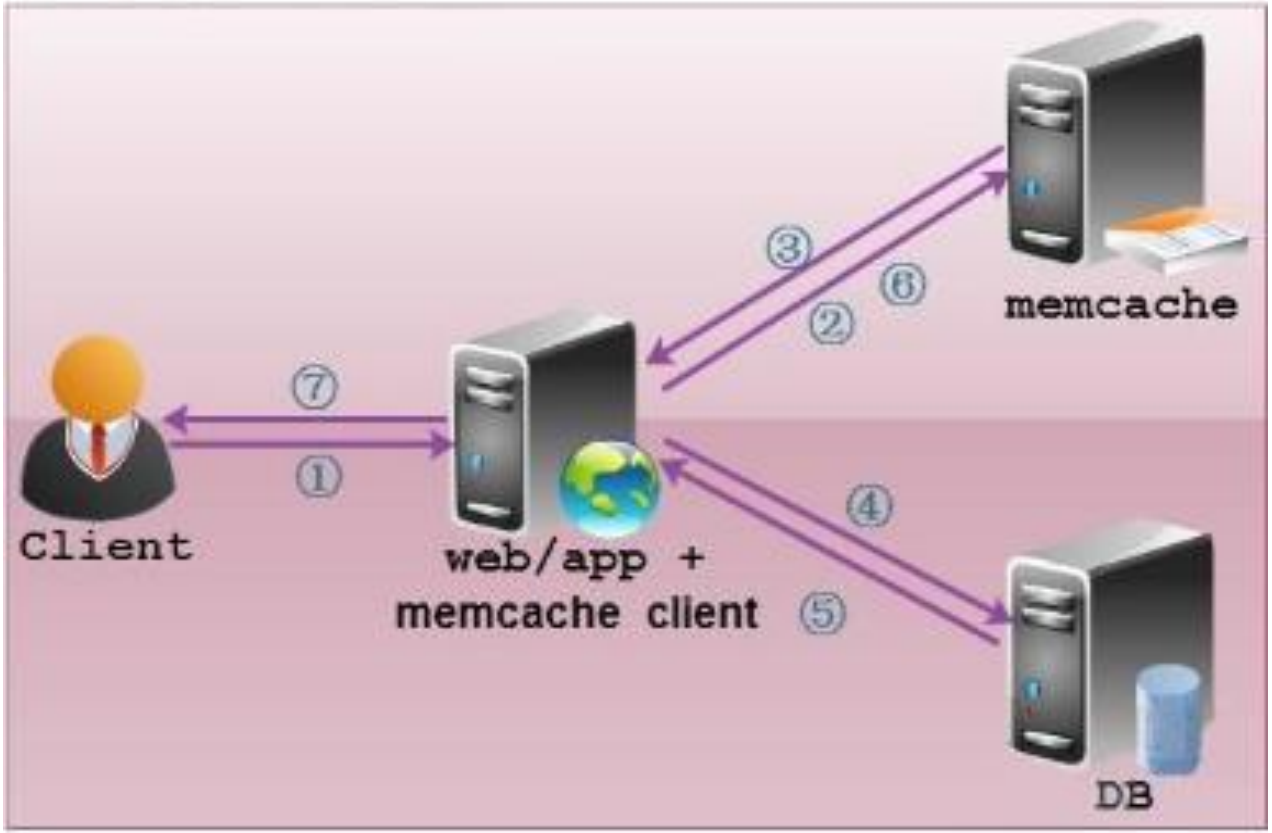
14.2 分布式缓存体系

Memcached的分布式计算架构可以将在一台机器上的多个Memcached 服务端程序、或者分散部署在多个机器上的Memcached 服务端程序组成一个虚拟的服务端Server，对于应用程序来说完全屏蔽和透明，提高了单机内存利用率，并且提供了优良的系统可扩展性。



14.2 分布式缓存体系

- **工作流程**



14.2 分布式缓存体系

● 内存管理

Memcached采用了一种简便易行的内存管理方式，所有缓存数据都保存在内存中，高速读取数据，当内存空间满后，通过LRU算法自动删除不使用的缓存。但这套方法没有考虑数据的容灾恢复问题，如果重启服务，所有缓存数据都会丢失。

◆ Slab与Page

这是Memcached划分内存空间的基本单位，默认大小为1MB。Memcached把整个内存空间按照Slab为单位进行管理的，用一个数据结构struct slab_list来定义一个Slab。

如果Slab是内存管理模型的一个抽象概念，那Page就是系统开辟内存时与Slab对应的物理单位，可以理解为内存中实际开辟的空间，大小与Slab一样。

◆ Chunk与Item

在一个Slab内部划分的更小的存储单位。比如，一个1MB大小的Slab可以划分为2个Chunks，每个Chunk大小为0.5MB；也可以划分为1024个Chunks，每个Chunk大小则为1KB。

14.2 分布式缓存体系

● 内存管理

◆ Slab Class

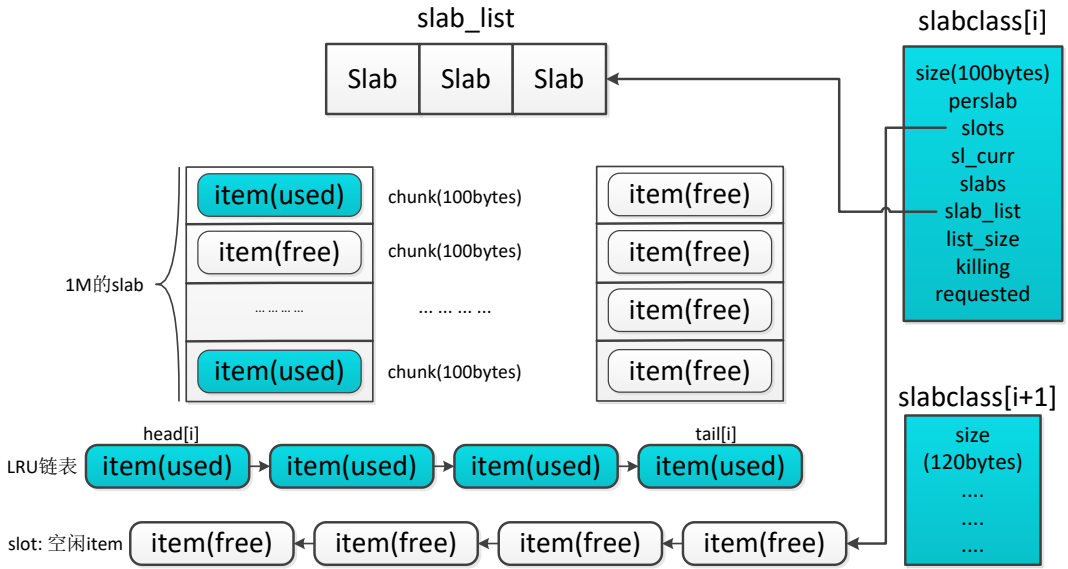
可以看出，Memchache管理的一个内存空间内可以划分多个slabs，而这些slabs又可以按照它们内部设置的chunk大小分为不同的组（比如一组slabs的chunk size都是512 bytes，另一组slabs的chunk size都是640 bytes等等），这样的组就称为Slab Class。

◆ 内存回收的LRU算法

内存空间总是有限的，当负载达到一定状况，所有的chunks都被占用了，这时就需要设计一个算法来淘汰旧的Item，腾出空间接纳新的Item。Memchache采用的是LRU（Least Recently Used）算法，即从当前Item中找出最少使用的那个淘汰。一个内存空间内各个slab class的即时状态：每个slab的红色chunks表示已被占用空间，绿色chunk表示尚未使用或已释放（free）的空间。Memchache使用两个链表来支持一个slab class的LRU算法，由于内存内有多个slab class，因此Memchache需使用多组链表。

14.2 分布式缓存体系

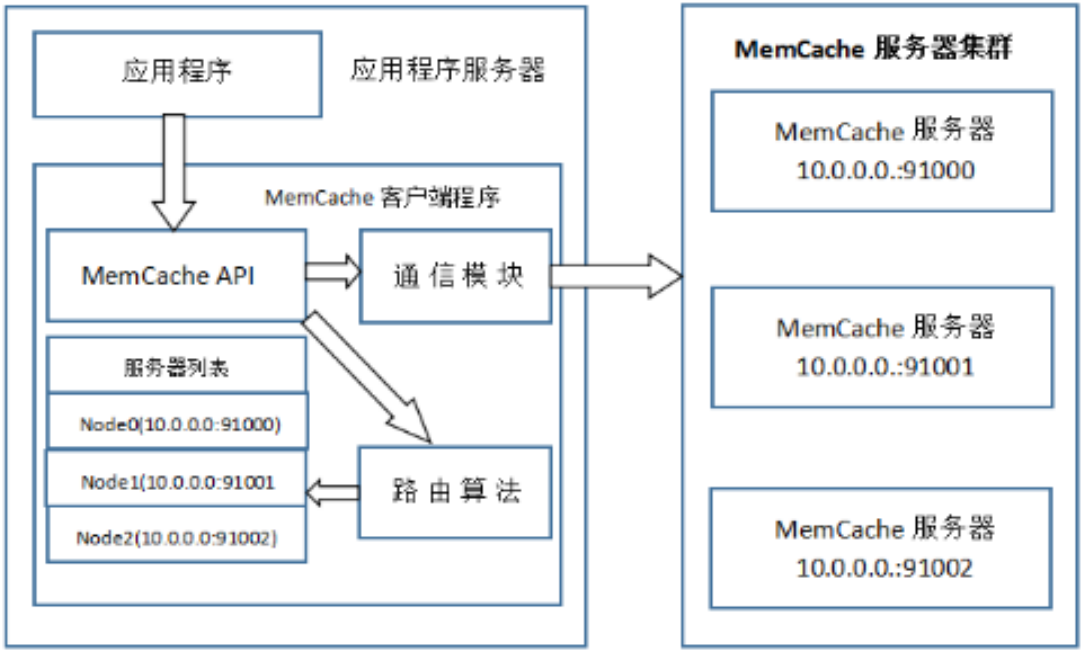
下图表示的是一个内存空间内各个slab class的即时状态。每个slab的红色chunks表示已被占用空间，绿色chunk表示尚未使用或已释放（free）的空间。Memchache使用两个链表来支持一个slab class的LRU算法，由于内存内有多个slab class，因此Memchache需使用多组链表。



14.3 数据存储的一致性

- **数据存储的一致性哈希算法**

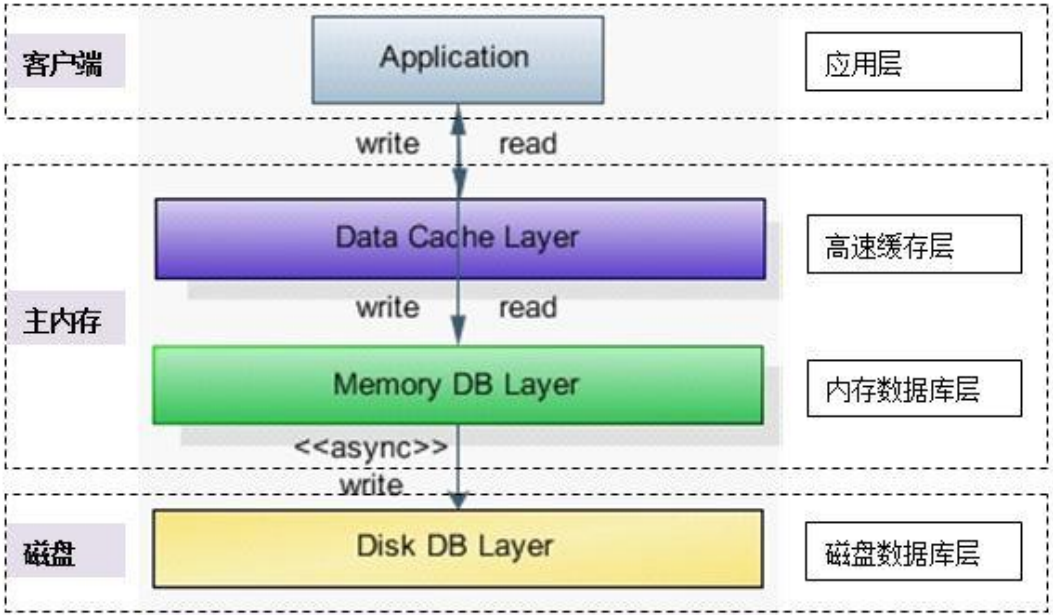
Memcached支持的缓存数据格式为键值对（key-value pair），当一个键值对数据项被Item提交给Memcached客户端，如图所示，假设我们有3个节点在Memcached集群中，需要有一个数据存储分配的路由算法帮助我们把写入的数据均衡地分配到这3台机器上。



14.4 内存数据库

● 内存数据库计算架构

一个完整的数据库应用系统计算架构见下图，它包含应用层、高速缓存层、内存数据库层、磁盘数据库层（持久性存储）四个层次。



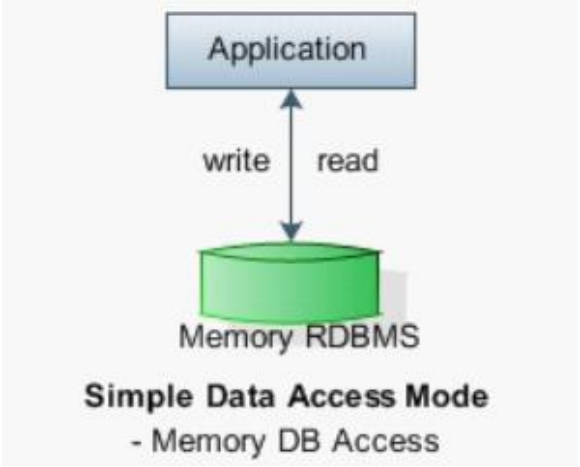
14.4 内存数据库

● 内存数据库计算架构

全内存架构

为了提高数据访问速度，一种理想的模式是把全部数据存储在内存中，所有的数据计算和事务性操作均在内存中完成，如图所示。但这种结构立即会带来如下问题：

- ◆ 机器主内存空间有限，难以一次性装载全部的数据库数据；
- ◆ 内存并非持久化存储介质，一旦断电或系统重启，内存中的数据就会丢失；
- ◆ 系统扩展性差，如果加入新的机器，无法立即对新机器的内存空间进行寻址，需要修改程序代码。

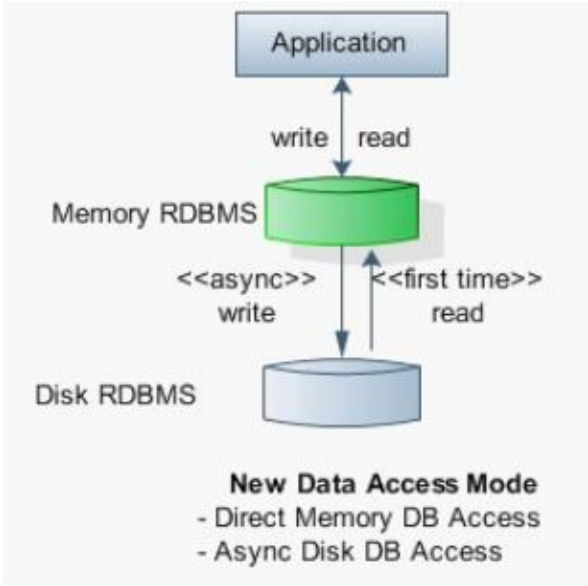


14.4 内存数据库

读写分离架构

为了克服全内存数据库无法提供及时的持久性存储的缺点，我们在系统中增加了磁盘存储，但为了提高数据访问速度，又在内存中另外实现了一套存储结构或内存数据库。通过两层结构的读/写分离模式，

读数据由内存数据库承担，内存中找不到才去访问磁盘数据库；写数据则是写入磁盘数据库，不影响内存数据库访问速度；内存数据库定期与磁盘数据库同步，从磁盘数据库导入新写入数据、或是把内存计算结果持久化到磁盘上，达到既能保证高速访问速度、又能持久化存储数据的目的。

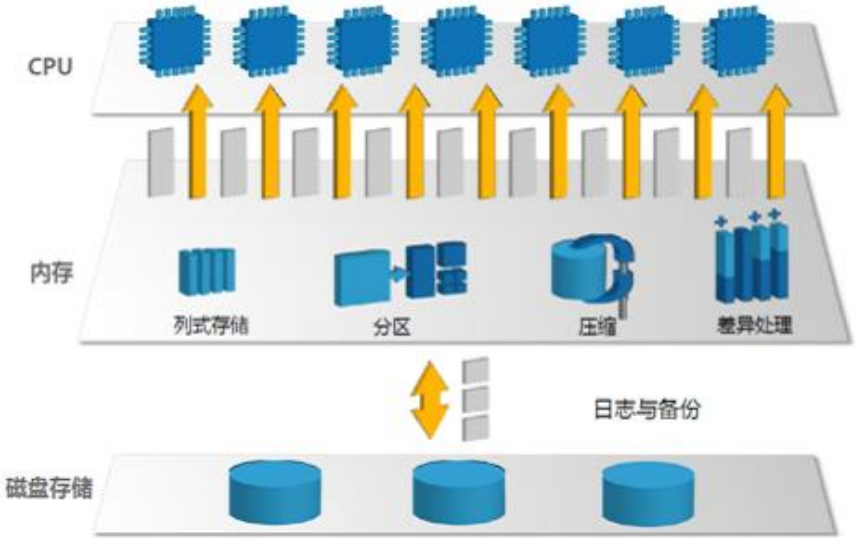


14.4 内存数据库

SAP HANA内存计算系统

德国SAP公司的HANA（High-performance Analytic Appliance）是一个软硬件结合支持内存计算模式的高性能计算分析平台。HANA的分层计算模式如图所示。

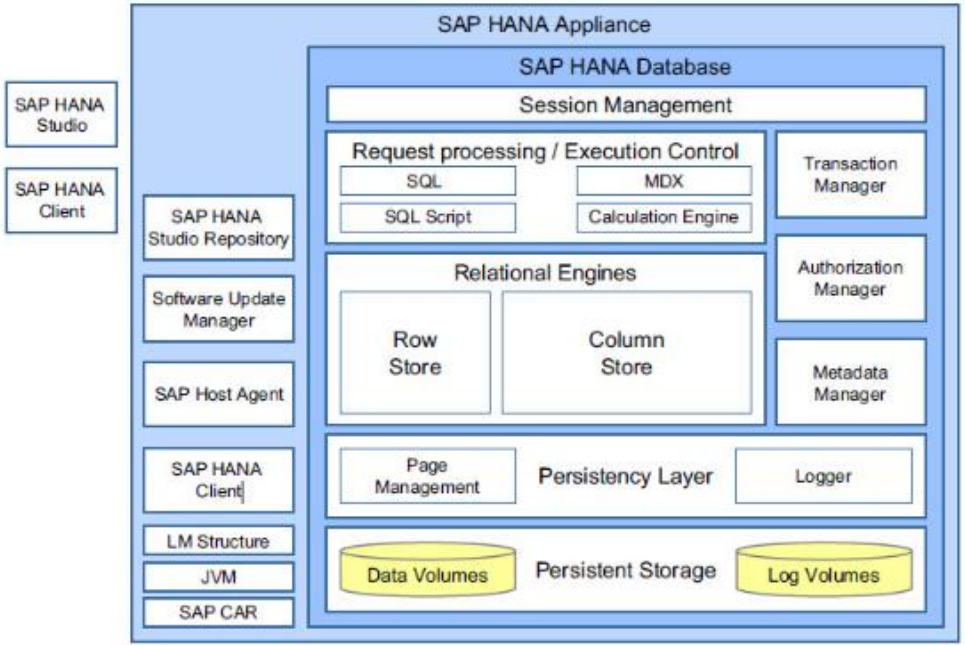
这种混合分区数据库架构将形成水平方向的多分区、垂直方向的二级数据库（2-Level DB）。



14.4 内存数据库

对应于上述分层计算模式，HANA设计了下图所示的计算架构。可以看出，图中SAP HANA Database子系统主要提供数据存储、预处理、计算分析。

支持行存储（Row Store）和列存储（Column Store）两种结果、事务处理、数据访问、持久化存储等功能和服务。除了数据持久化存储（Persistent Storage）这一功能，其他功能和操作都在内存中完成。

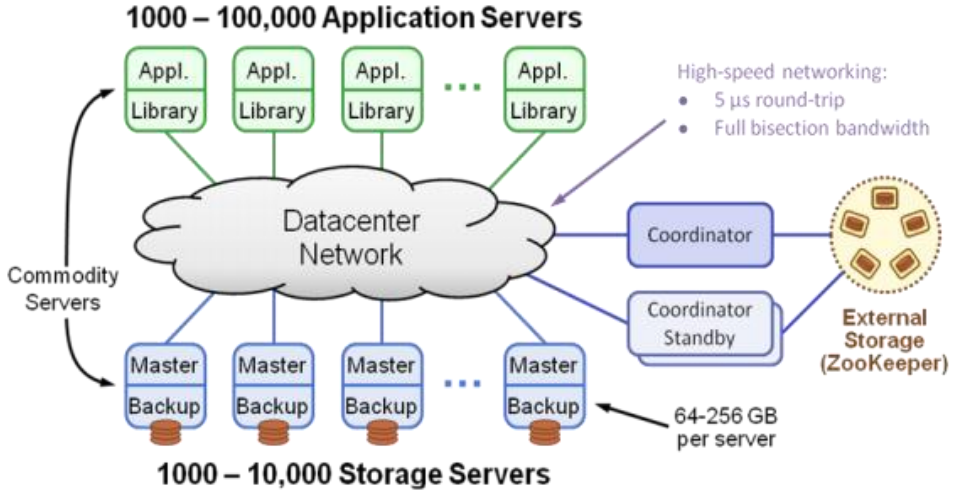


14.5 MemCloud计算架构

● 计算体系架构

每个RAMCloud 节点的计算架构如图所示，它包含Master和Slave两个模块：

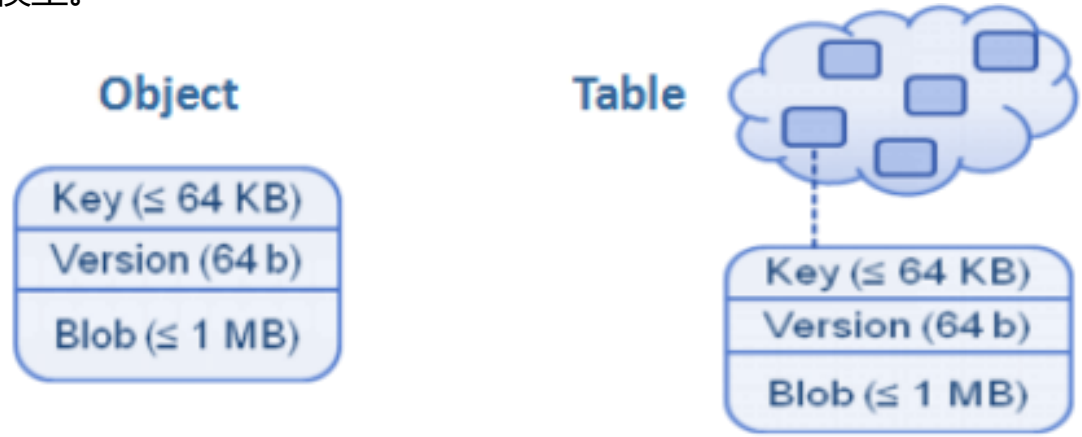
- ◆ Master模块管理节点主内存和存储在内的RAMCloud Objects，并负责处理客户端程序的读写数据要求；
- ◆ Backup模块负责管理节点本地磁盘和闪存，以及存储在磁盘上的其他节点数据文件的副本。



14.5 MemCloud计算架构

- **数据存储架构**

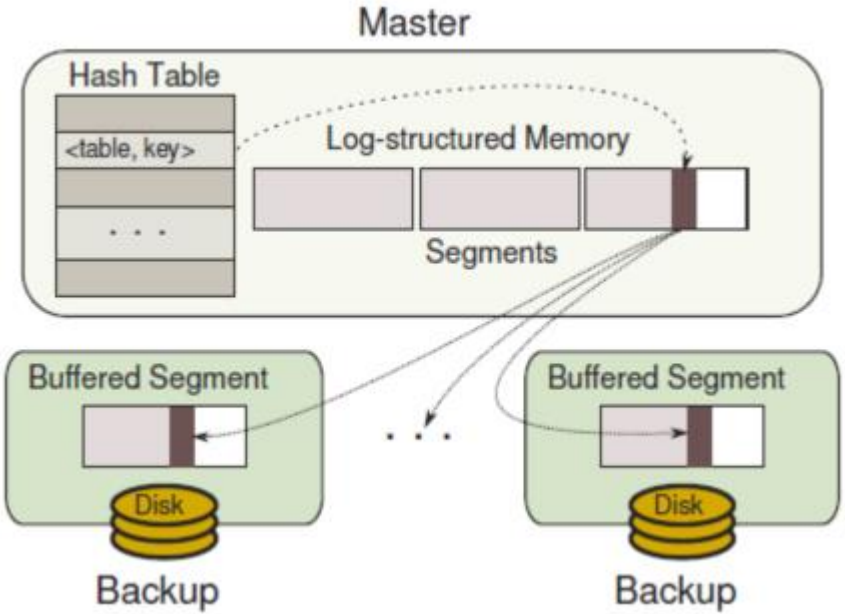
RAMCloud使用简单的键值对（key-value pair）数据结构，数据被封装为Object，每个Object都被长度不一的唯一的Key标记，是RAMCloud处理的基本数据单位（即围绕Object的操作都是原子化操作atomic operation）。Object的大小介于几十bytes到1MB之间，一般使用较小单位。多个Objects组成一个Table，一个Table可以有多个副本存放在集群不同节点上。下图描绘了RAMCloud的数据模型。



14.5 MemCloud计算架构

- **数据存储架构**

RAMCloud在每个集群节点上的Master程序管理着存放在内存里的一组Objects和一个哈希表（下图），表里面每一条entry都对应着内存里存放的一个Object。



14.5 MemCloud计算架构

每个Object是以一条**日志条目（Log Entry）**形式存储在内存里，Object日志条目值域包括所属表序号（Table ID），键值（Key），版本（Version），时间戳（Timestamp），以及数据区（Value）。在数据恢复时，将根据最新版本的日志条目数据来重建Hash Table。

内存里还存放着以下日志数据：

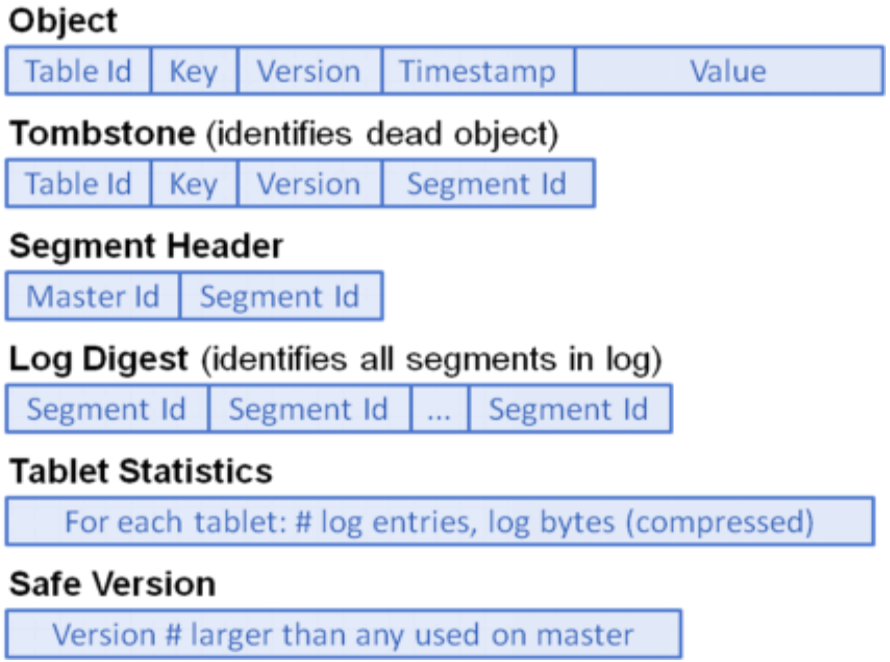
失效标志（Tombstone）：包含表序号（Table ID）、键值（Key）、版本（Version）、分区序号（Segment ID）。它代表了被删除的object。日志一旦写入就不可修改，如果一个object被删除，就在日志中添加一条tombstone的记录，这样将来在数据恢复时，被删除的object就不会被重建。

分区头（Segment Header）：包含主程序号（Master ID）、分区序号（Segment ID）。

日志摘要（Log Digest）：包含日志中所有的Segment ID。在数据恢复时，将根据最新的日志摘要来加载所有的日志数据。每个节点的日志摘要使得节点恢复可以自我完成，而不需要有一个保留所有节点元数据的集群中心节点的存在。

14.5 MemCloud计算架构

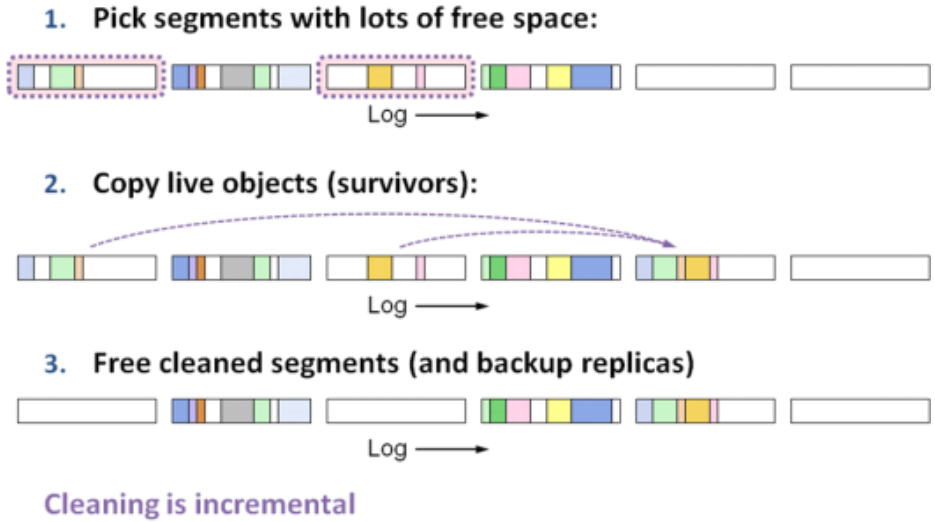
表统计（Tablet Statistics）：包含每个Tablet的统计数据。
安全版本（Safe Version）：比日志中所有已使用的版本号都大的一个版本号。



14.5 MemCloud计算架构

- **内存清除机制**

内存空间在使用一段时间后，不可避免地有一些Log会失效，一些Log的空间没有使用完，即所谓的内存碎片化（memory fragmentation），由于内存空间非常宝贵，RAMCloud设计了一套内存清除（Cleaning）机制来有效地使用和管理内存。内存清除流程分三步。



14.5 MemCloud计算架构

RAMCloud设计了一个如下的Two-level Cleaning机制：

First-level Cleaning：Segment Compaction（分区压缩）。在此阶段清除线程只对内存内Segment内部的Logs进行清理和压缩，释放清除后的内存空间供再次使用。

Second-level Cleaning：Combined Cleaning（综和清除）。同时清除多个Segments,并进行磁盘清除、同步。

End of Session
Thank you!