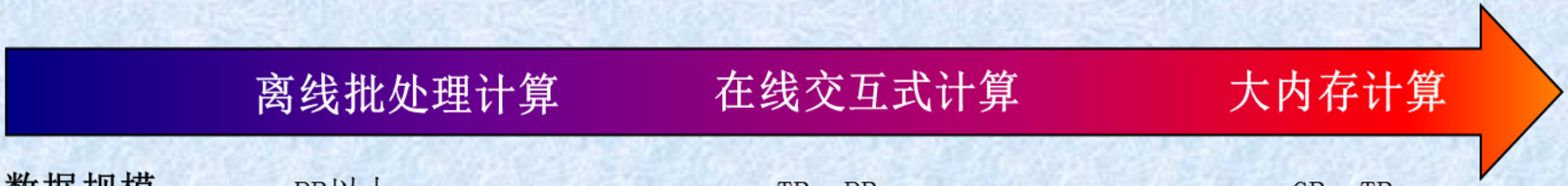
Lecture 18 内存计算模型

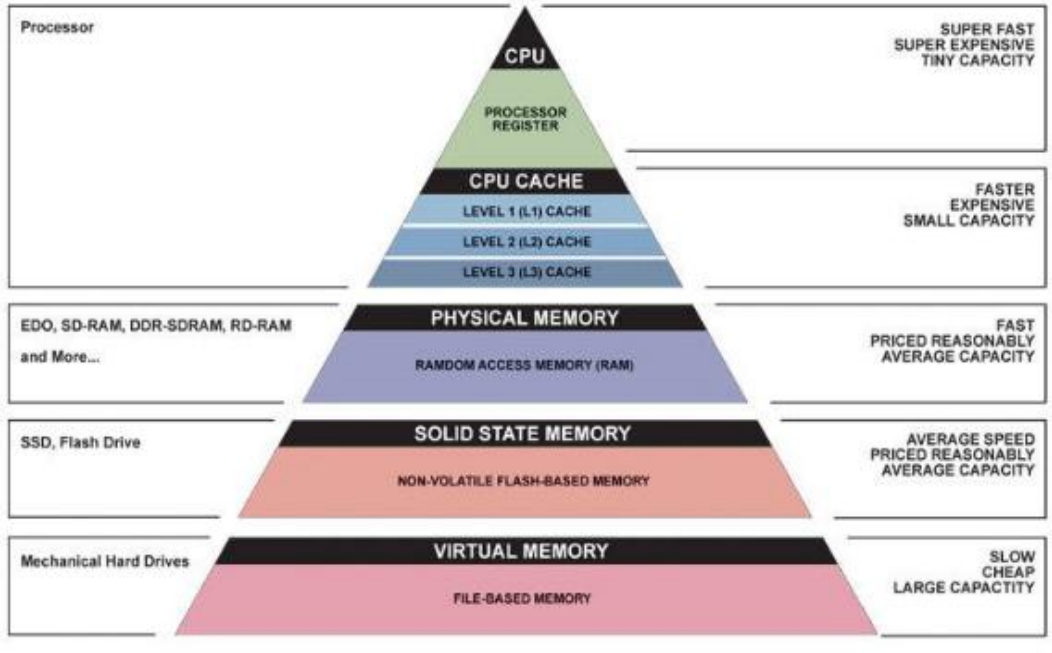
- 内存计算概念
- 分布式缓存系统
- 内存计算技术

大数据计算光谱



	离线批处理计算	在线交互式计算	大内存计算
数据规模	PB以上	TB～PB	GB～TB
时延性	离线计算(分钟～小时)	在线分析(秒～分钟)	实时计算(秒级)
计算模型	MapReduce Pregel HAMA	Dremel Drill PowerDrill	MemCloud HANA
系统结构	分布式体系	分布式体系	集中式结构
采用技术	大数据迭代循环 硬盘读写次数多	提高数据内存存驻率 data locality columnar data structure	内存一次加载 硬件成本高

存储结构访问速度比较



操作	时间
主内存访问	100 ns
内存顺序读取 1 MB数据	250 000 ns
磁盘寻道	5 000 000 ns
磁盘顺序读取 1 MB数据	30 000 000 ns

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。

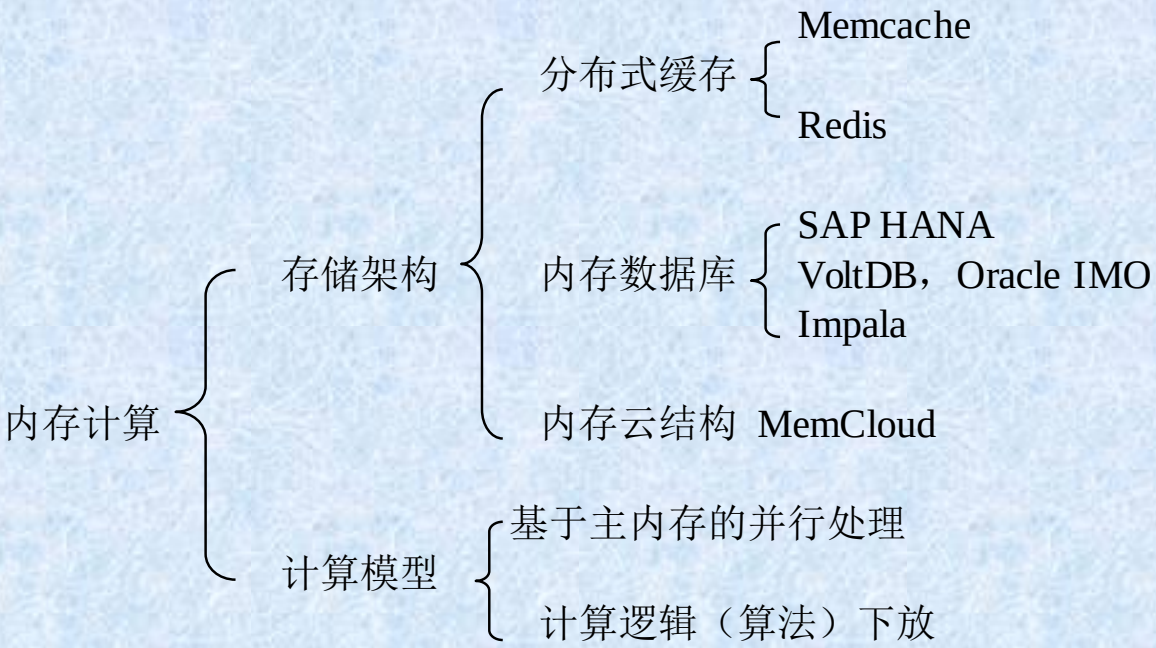
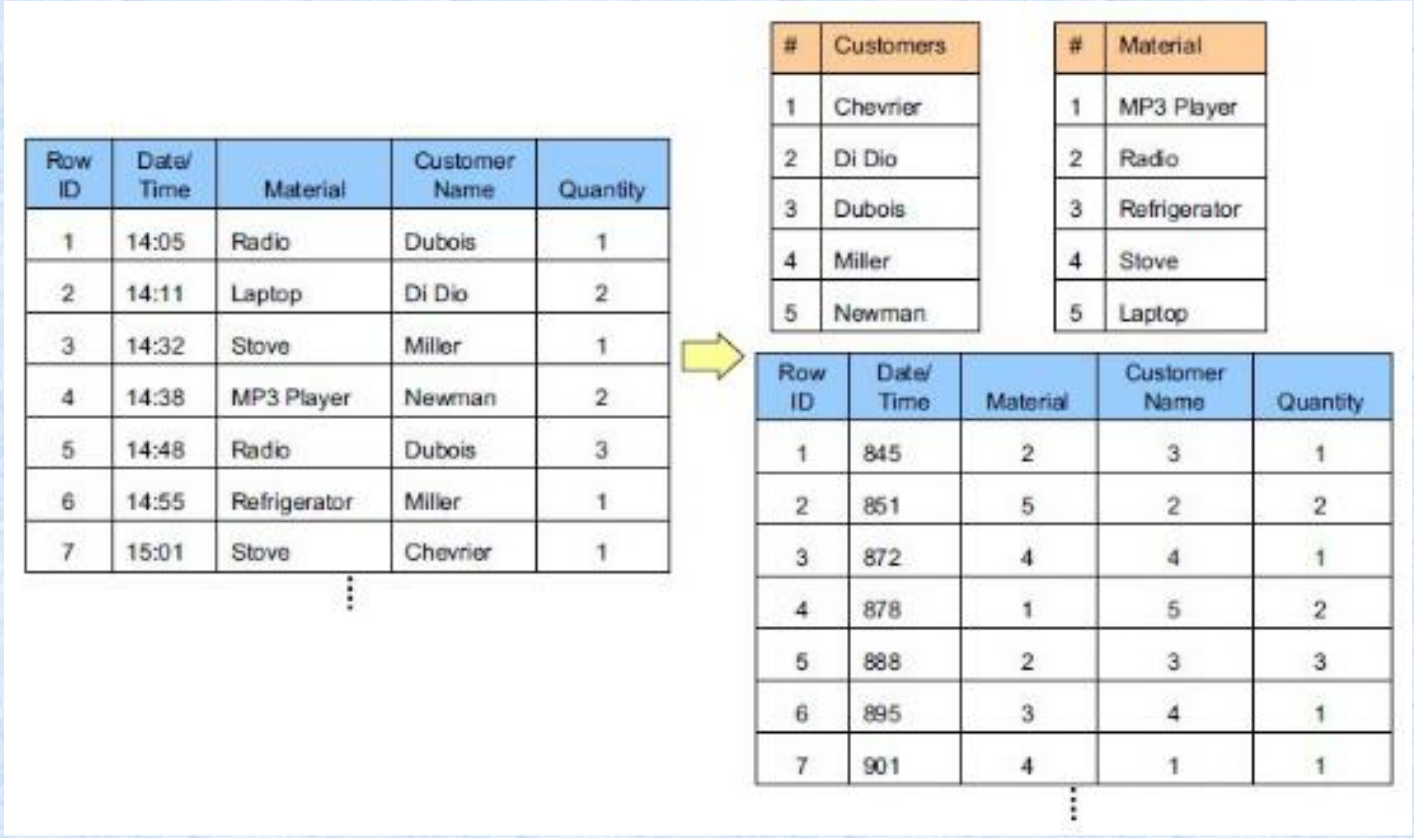


图 16-1 内存计算架构

内存技术 – 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。下图为字典编码压缩基本原理。



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

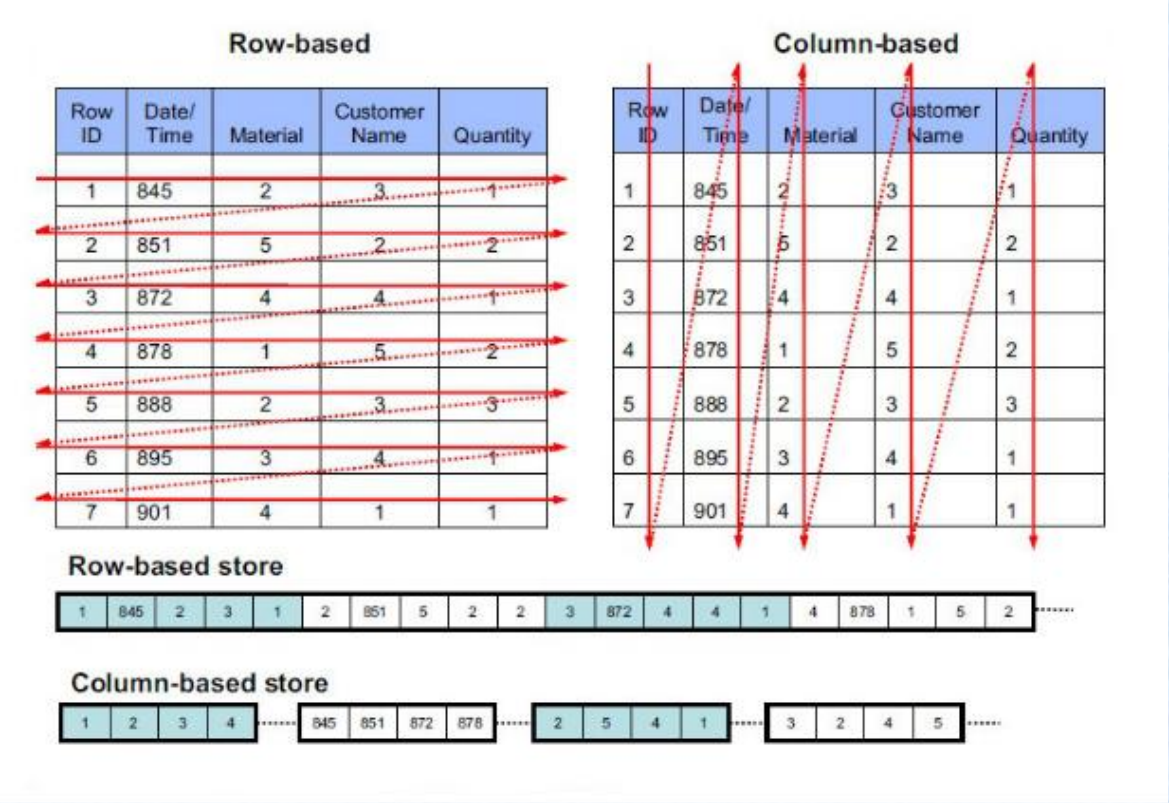
#	Customers
1	Chevrier
2	Di Dio
3	Dubois
4	Miller
5	Newman

#	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

内存技术 – 列存储结构（columnar storage structure）

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



列存储结构算例

问题：1000万条数据的购买记录表，100万顾客（Customers检索表有100万条），200万商品（Materials检索表有200万条）。如何在2000万条数据（假设分别包含在2个大表中，每个大表存1000万条数据）的购买记录表中找出“Miller”购买“refrigerator”的次数？

行存储方法： 首先搜索“Miller” *Select Rows by Customer Name=“Miller”*
最坏情况 1000万次，得到一个子表；
再搜索“refrigetor” *Select Rows by Material =“refrigerator”*
最坏情况 1000万次，得到一个子表；
Join两个子表，得到同时包含“Miller”和“refrigetraor”的项，
最坏情况 1000万 x 1000万次操作

总操作次数 = 1000万+1000万+1000000万 = 100.2 亿次

列存储结构算例（续）

列存储方法：1）首先搜索Customer Name检索表，找到“Miller” 对应的局部ID （4），最坏情况 100万次；

2） 再搜索Material检索表，找到“refrigerator”对应的局部ID（3），最坏情况 200万次；

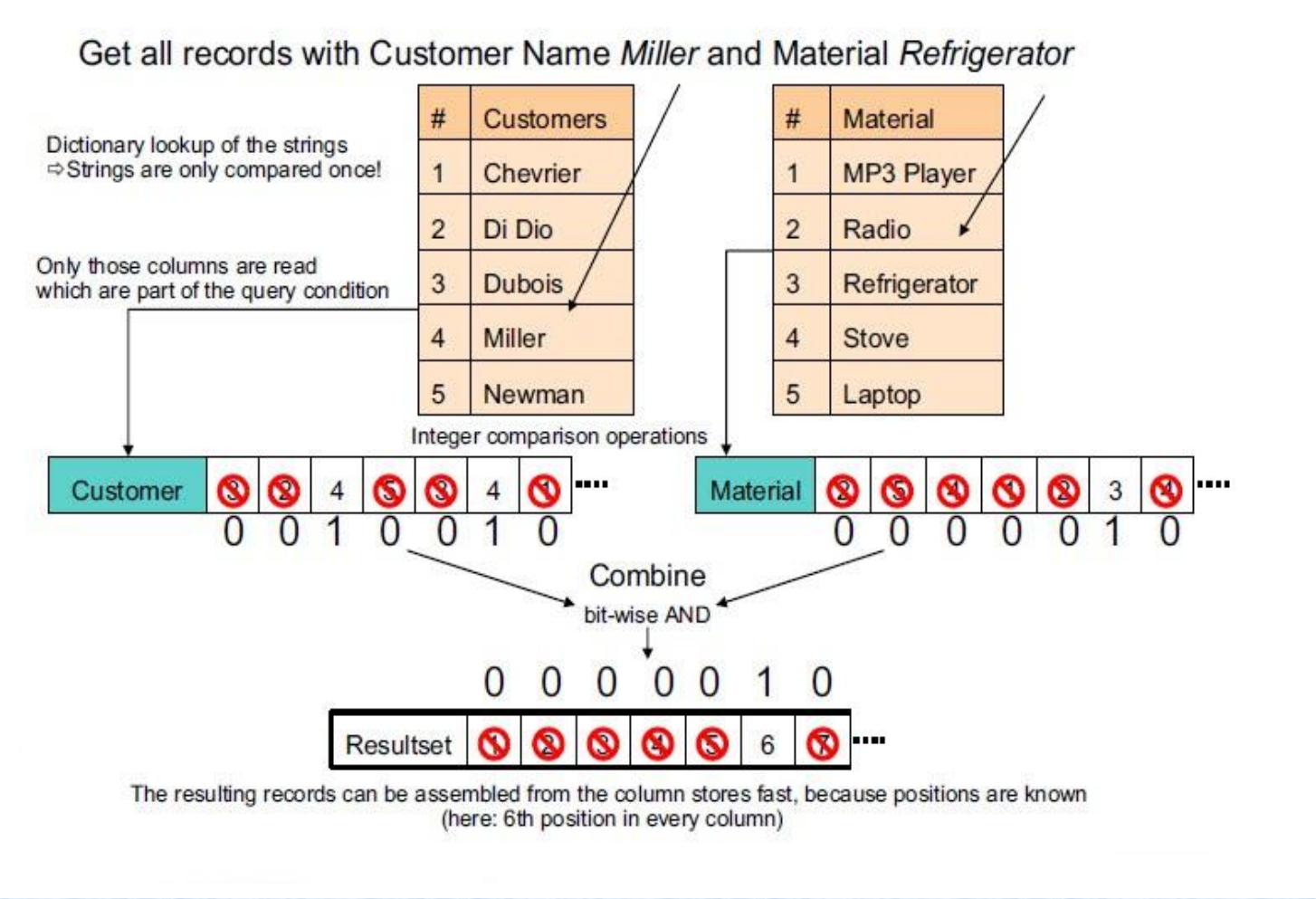
3）用查得的Miller对应的局部ID值4去遍历内存中顺序存储的“Customer Name”一列的各个元素（已转换为局部ID），若不等于4就设为0，等于4就设为1，遍历完一个表需要1000万次操作，得到一个“0010010 ...”这样的二进制数组；

4）用查得的Refrigerator对应的局部ID值3去遍历内存中顺序存储的“Material”一列的各个元素（已转换为局部ID），若不等于3就设为0，等于3就设为1，遍历完另一个表需要1000万次操作，得到一个“0000010 ...”这样的二进制数组；

5）将步骤3和4得到的两个二进制数组进行bit-wise AND操作，得到的结果也是一个二进制数组“0000010 ...”，其中的“1”位就是满足我们查询条件的数据项。

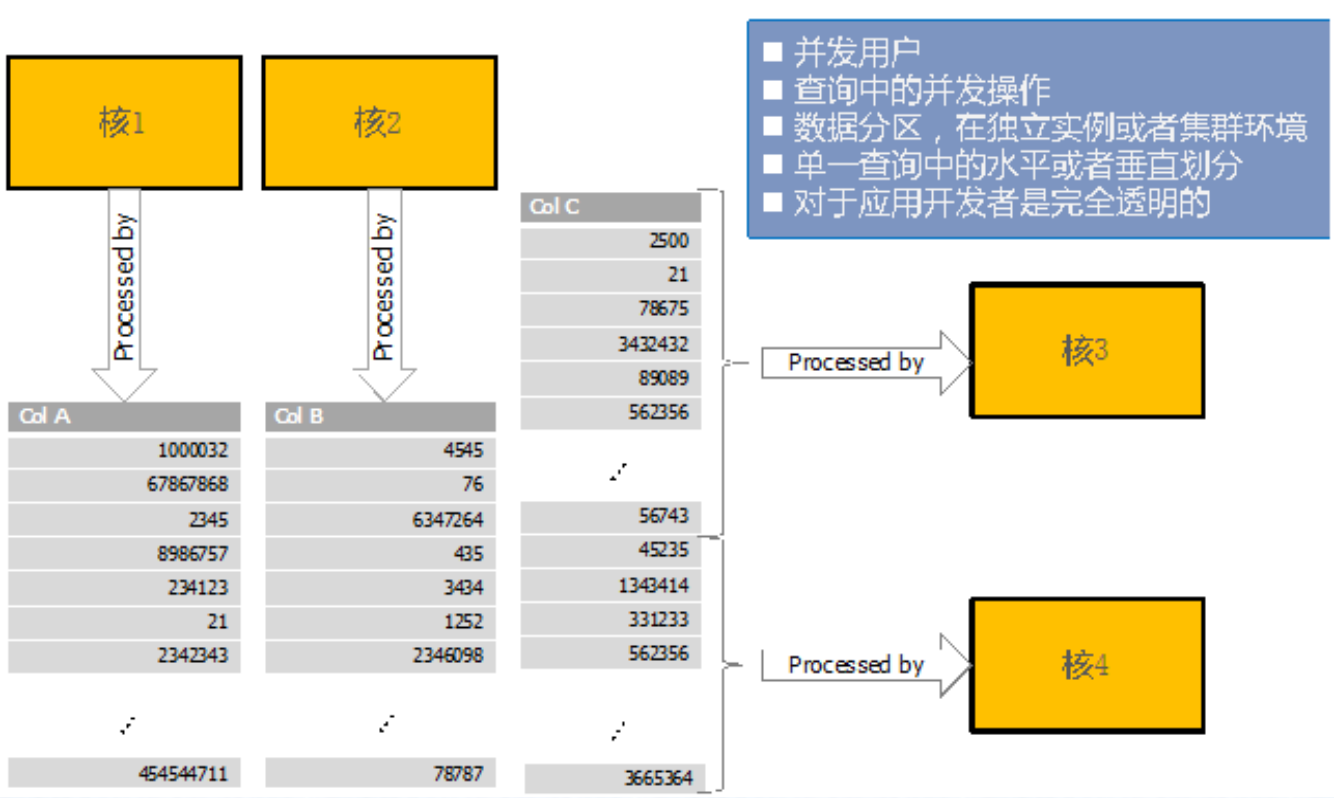
总操作次数 = 100万+200万+2*1000万+1 = 0.23 亿次

列存储结构算例（续）


Get all records with Customer Name Miller and Material Refrigerator
Dictionary lookup of the strings
⇨Strings are only compared once!
Only those columns are read
which are part of the query condition
Customers
1 Chevrier
2 Di Dio
3 Dubois
4 Miller
5 Newman
Material
1 MP3 Player
2 Radio
3 Refrigerator
4 Stove
5 Laptop
Integer comparison operations
Customer
Material
0 0 1 0 0 1 0
0 0 0 0 0 1 0
Combine
bit-wise AND
0 0 0 0 0 1 0
Resultset
The resulting records can be assembled from the column stores fast, because positions are known
(here: 6th position in every column)

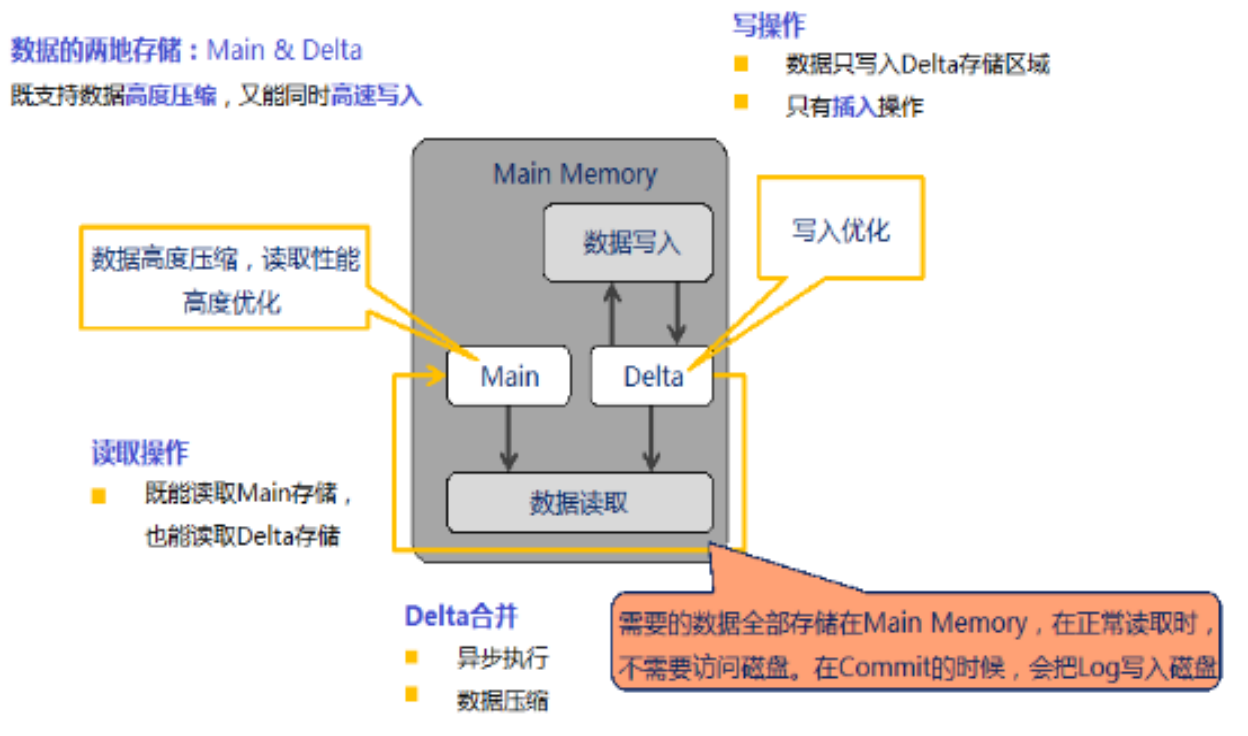
内存技术 – 数据表分区

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，如图所示。



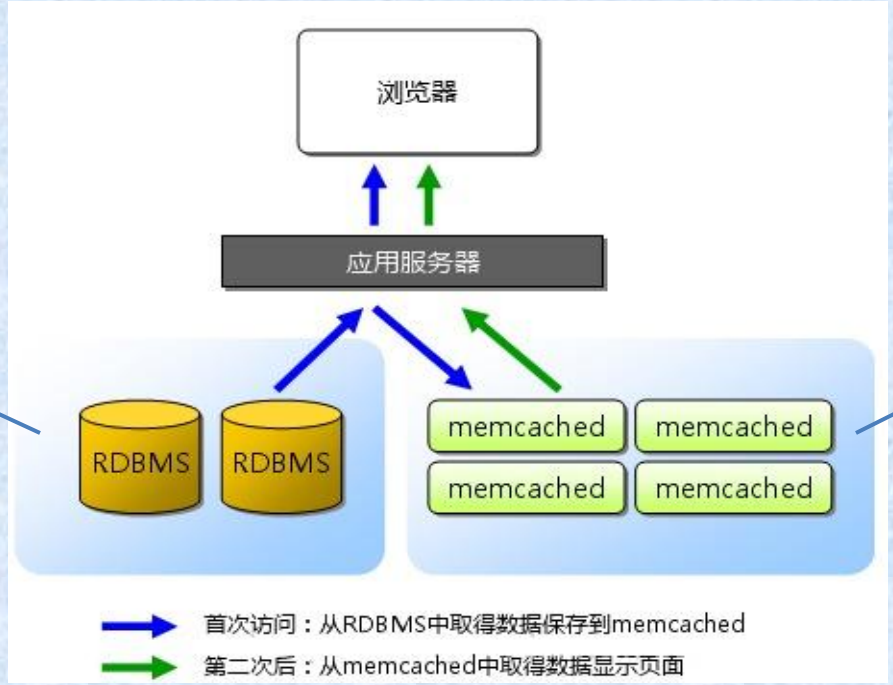
内存技术 – 只插入差异数据

只写入差异数据来提高访问效率。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。



1. 分布式缓存体系

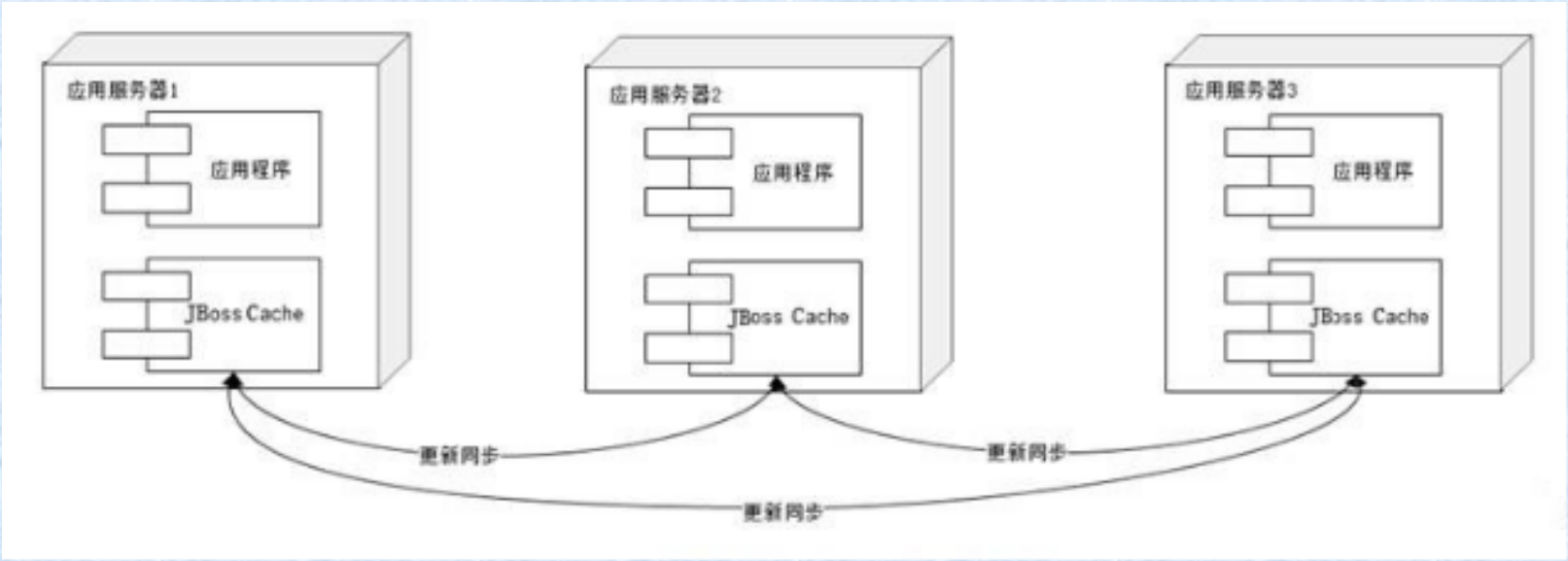
分布式缓存系统（Distributed Cache System）包含两层含义：
1）由多台服务器组成一个缓存服务器集群，以多节点集群方式提供缓存服务，即物理架构上是分布式；
2）缓存数据（可看作一个大数据表）被分布式存储在多台缓存服务器上，即逻辑架构上也是分布式的。



分布式缓存架构

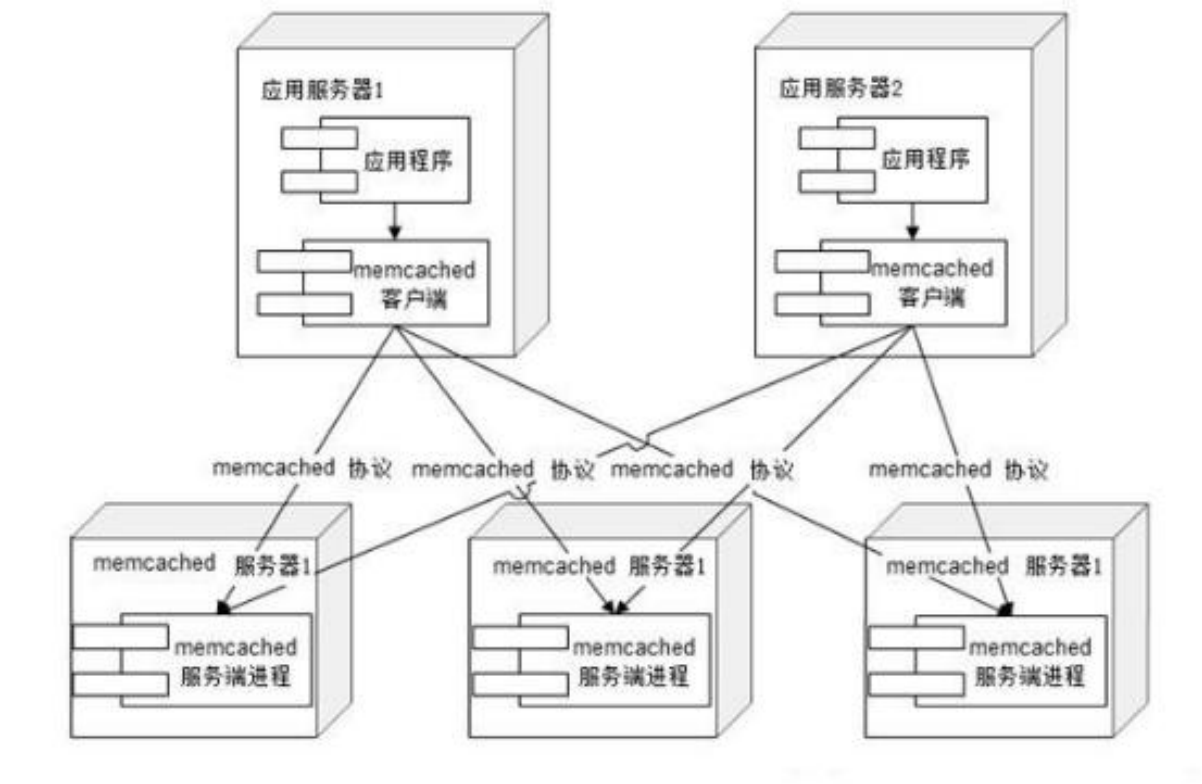
缓存服务器不是把数据存储在磁盘上而是存放在内存中，多个缓存服务器按分布式共同存放一个数据表，涉及到数据同步的问题。

数据同步缓存系统以JBoss Cache为代表。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。



数据同步的JBoss Cache缓存系统

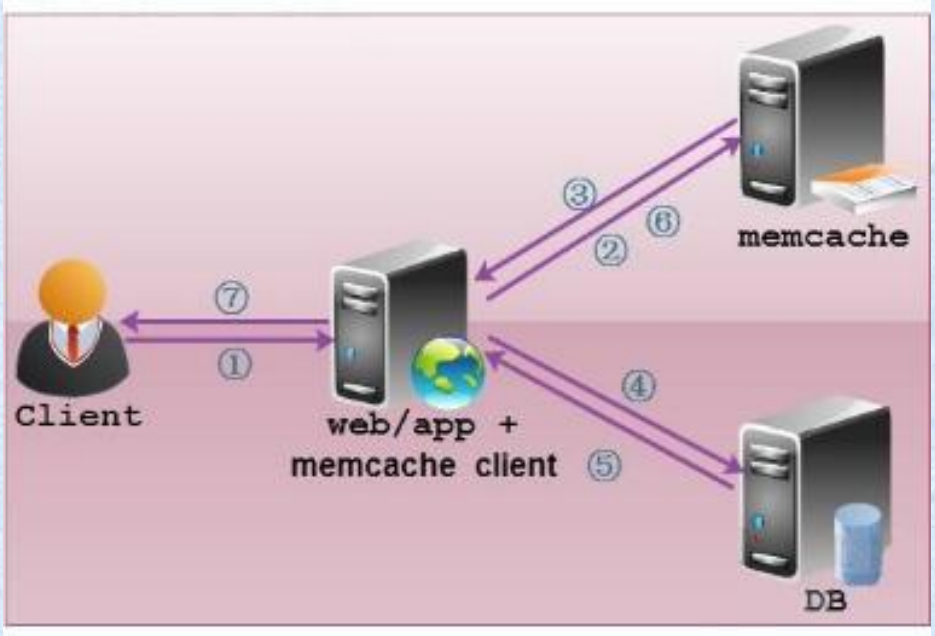
Memchache则采用了数据不同步的架构，采用一组专用缓存服务器，缓存与应用分离部署。在存放和访问缓存数据时，通过一致性Hash算法选择缓存节点，集群缓存服务器之间不需要数据同步，因此集群规模可以很容易地实现扩容，具有良好的可伸缩性。



数据不同步缓存系统

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的系统架构如图。

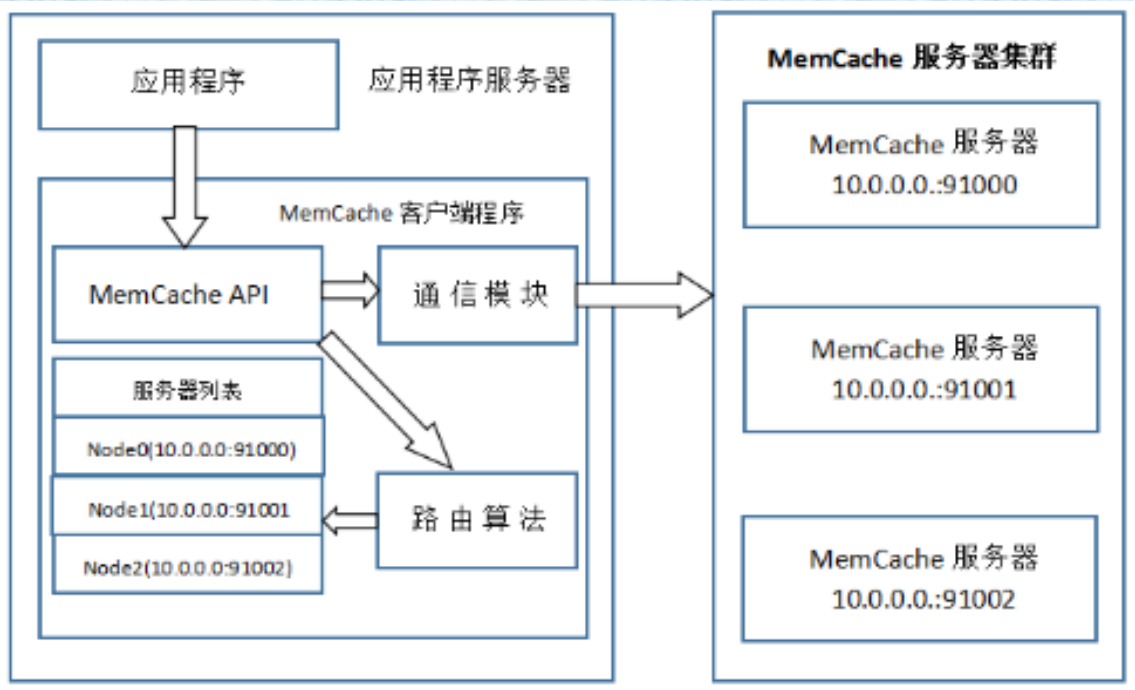


Memchache工作流程

- 当客户端提交了读数据请求，首先扫描memcache看数据是否存在，如是，直接读取数据返回客户端，不对数据库作任何操作，操作路径为：①→②→③→⑦；
- 如果请求的数据不在缓存中，则访问数据库DB，把从数据库中获取的数据返回给客户端，同时把数据缓存一份到memcache中（memcache客户端不对此负责，需要应用程序明确实现），操作路径为：①→②→④→⑤→⑦→⑥；
- 当客户端提交了写数据或更新数据库请求，更新数据库的同时同步memcache，保证数据一致性，操作路径为：①→④→⑥→③→⑦；
- memcache的内存空间满溢时，使用LRU（Least Recently Used）算法及其他失效策略对缓存数据进行更换。

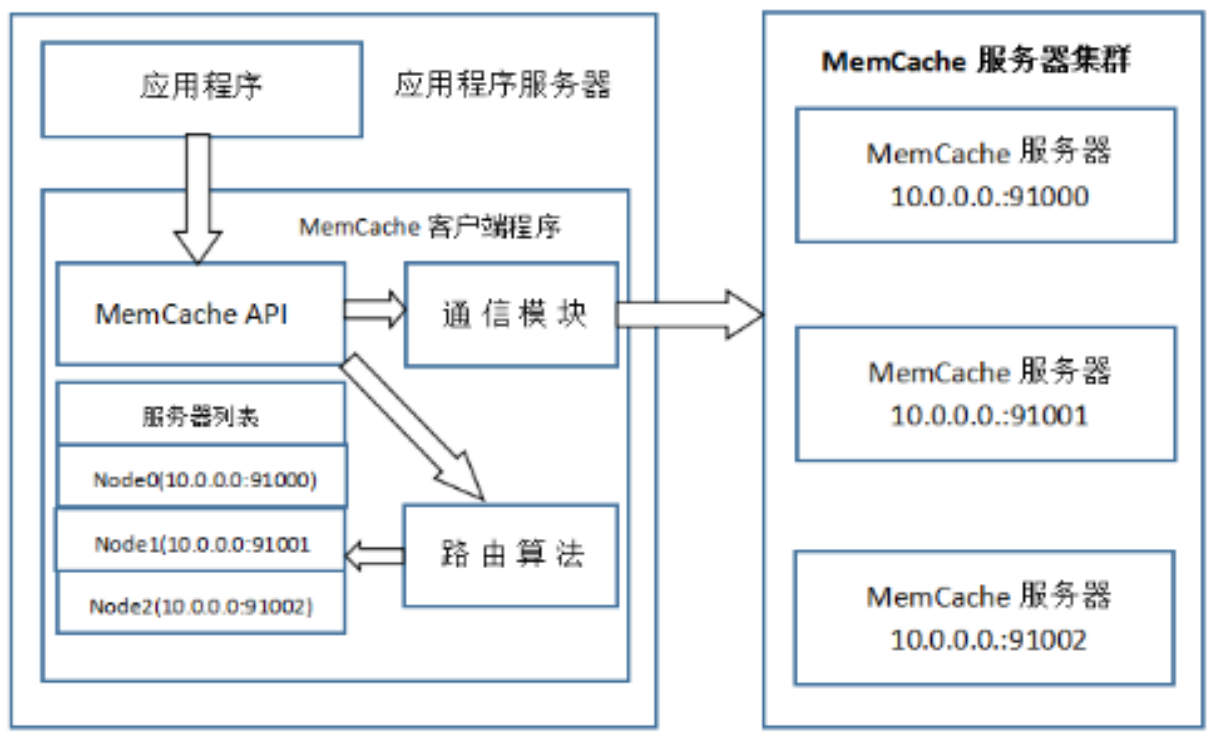
Memchache的计算架构

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。



数据存储“数据项 -> 服务器”的映射

Memchache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提交给memchached客户端，如图所示，假设有3个节点在memchached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。



服务器扩容带来的数据缓存命中率降低问题

计算选择服务器ID公式（共有3台服务器）：

选择服务器ID = HashCode Mod 3

HashCode	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
选择服务器ID	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1

扩容后（有4台服务器，选择服务器ID = HashCode Mod 4）：

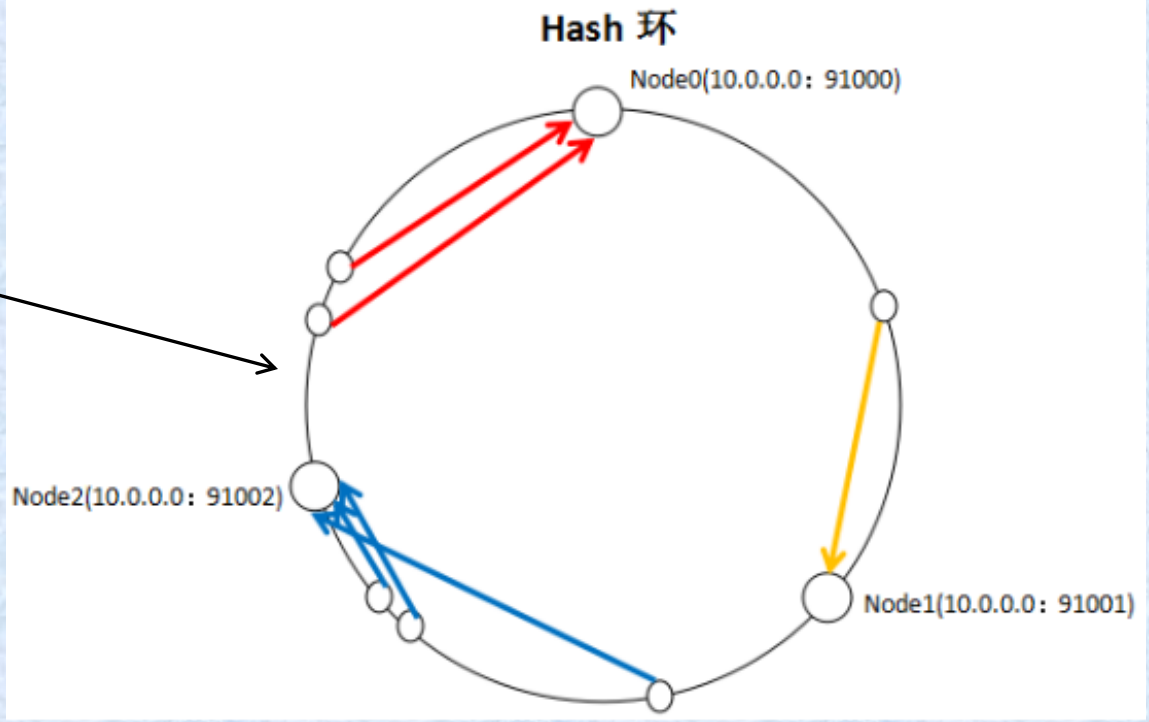
HashCode	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
选择服务器ID	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3

扩容后仅有30%的数据项保留原有映射关系！

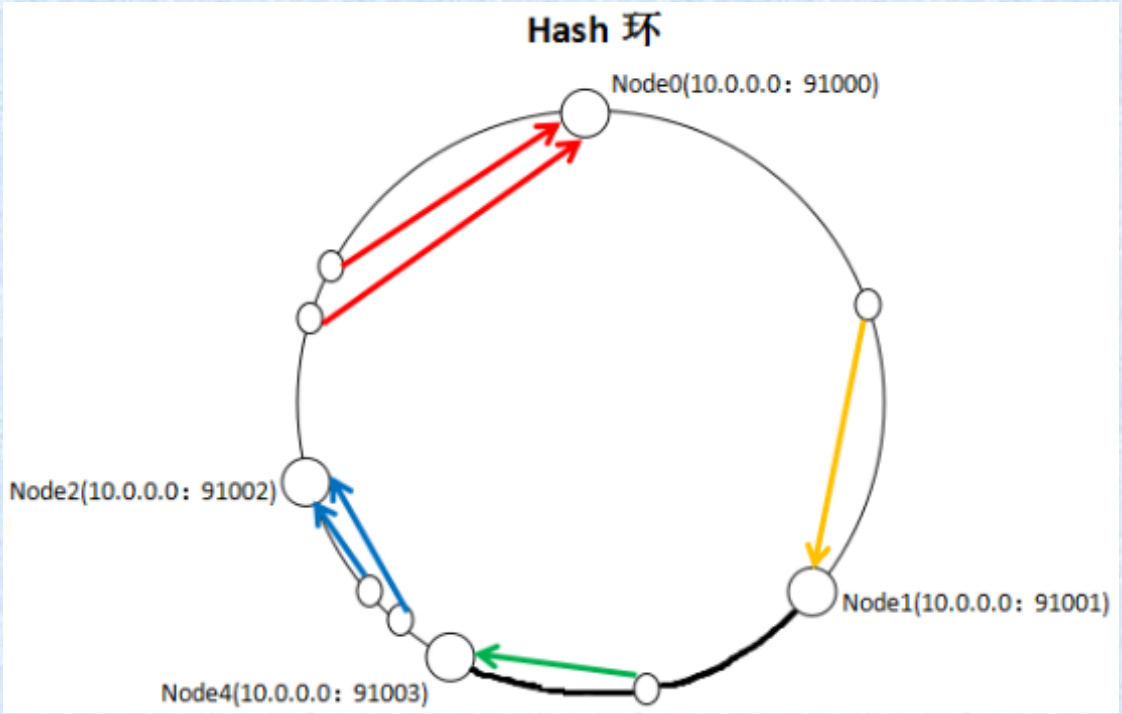
存储分配的一致性哈希算法

目前Memchache采用了一致性哈希算法（Consistent Hash）。其原理为通过一个称做一致性Hash环的数据结构实现数据项Key到缓存服务器ID的Hash映射，这个一致性Hash环如图所示，它跨越了长度区间[0, 2^{32} - 1]。

一致性Hash环



如果memchached服务器集群扩容，比如上例中增加了服务器节点Node4（第4台服务器），它在一致性Hash环上的位置比如说落在Node1与Node2之间的左下角位置（如图所示），这就影响到了原来映射到Node2的一个数据项（绿色箭头），现在就改成映射到Node4。

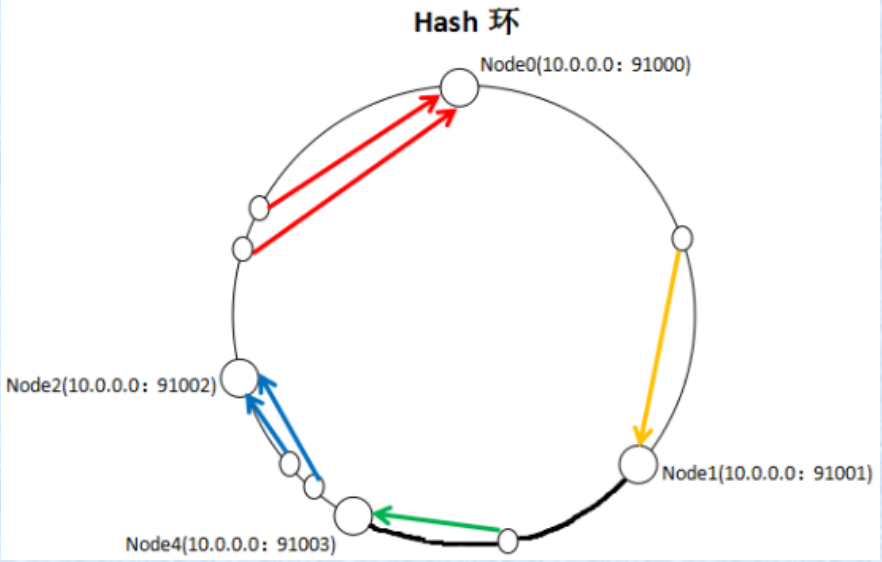


扩容后的一致性Hash环

一致性哈希算法的优势

1）扩容时一致性Hash算法只影响Hash环上新加入服务器节点与顺时针方向它身后节点这个区间的数据项，也即是影响是局部的而非全局性的；

2）随着节点数增加，环上服务器节点排列越来越密，上述受影响区间会变得越来越小，原有映射关系保持正确性的概率越来越大，这就意味着服务器规模扩大反而使得一致性Hash算法的结果倾向稳定，这是算法的优势。



内存数据库技术

计算架构

一个完整的数据库应用系统计算架构见图，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。

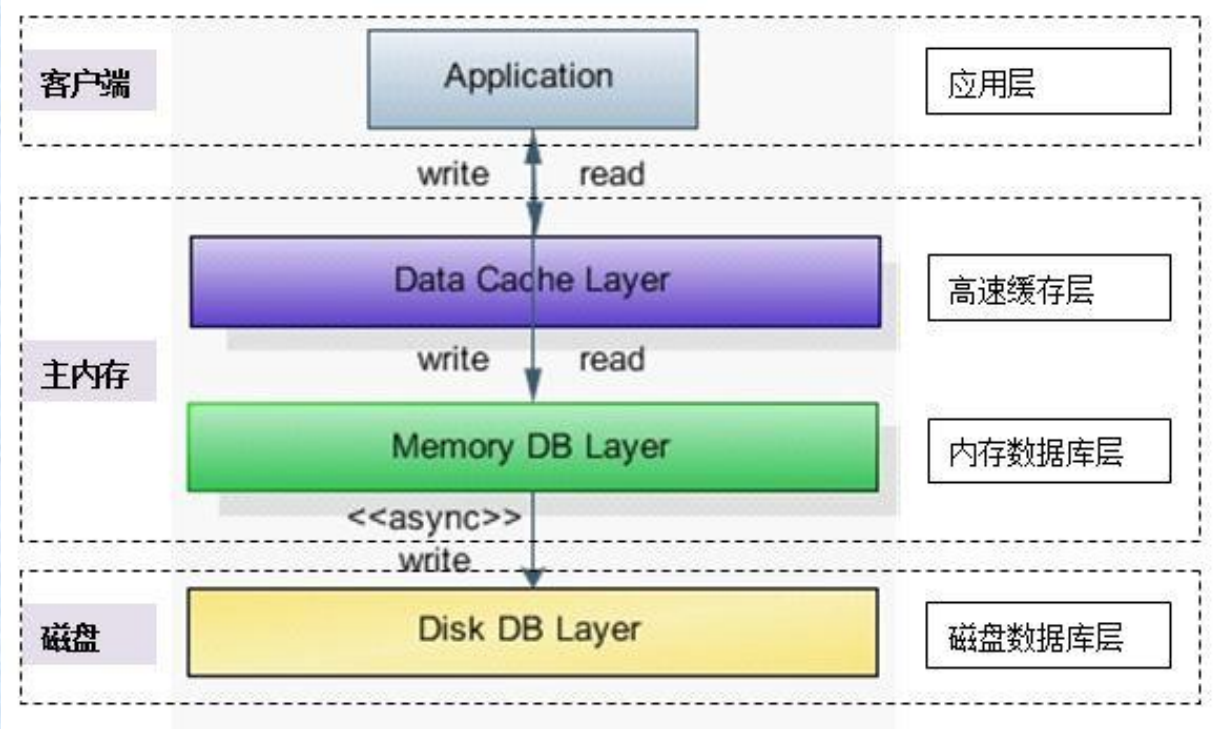


图 16-20　完整的数据库计算架构

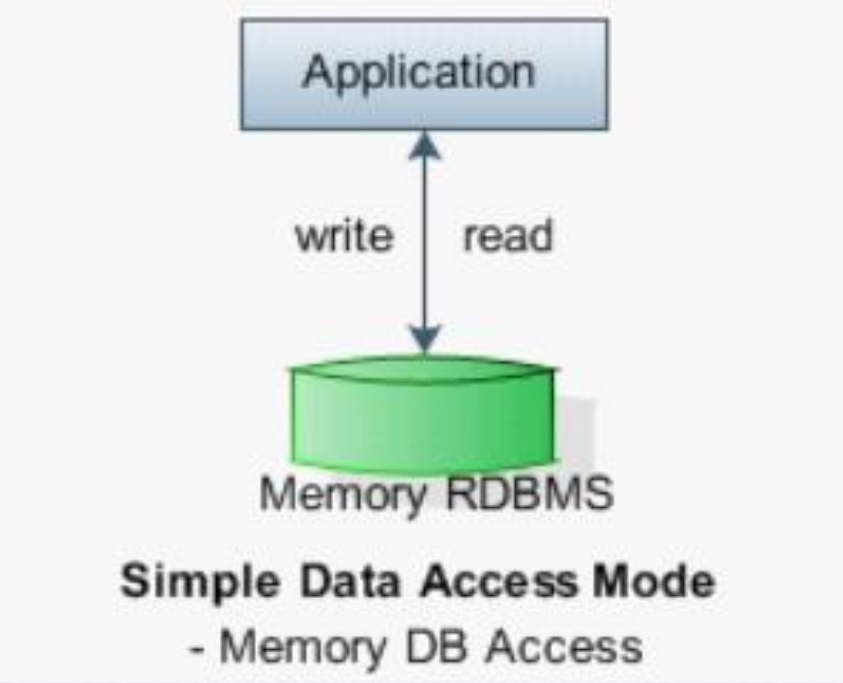
■ 全内存架构

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，数据就会丢失；

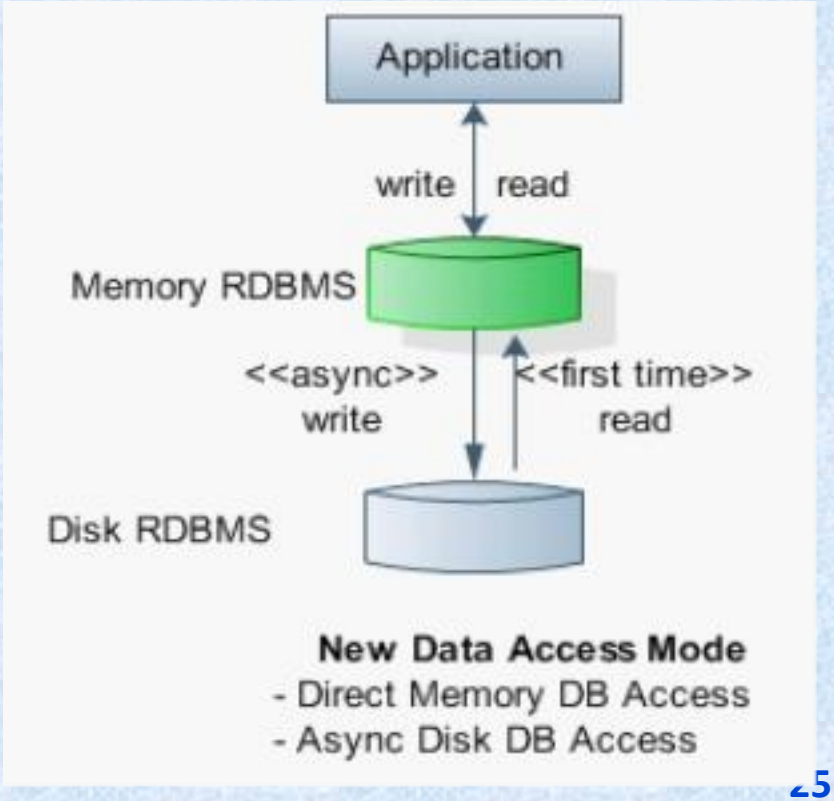
✓ 系统扩展性差，如果加入新的机器，无法立即对新机器的内存空间进行寻址，需要修改程序代码。



■ 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，如图所示。

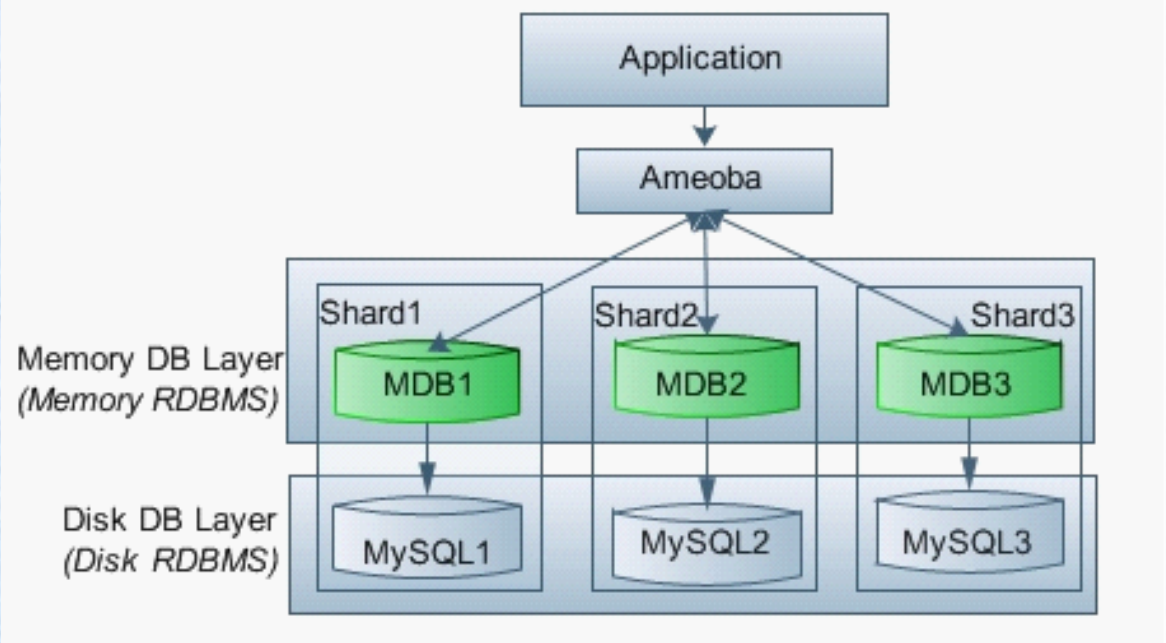
读数据由内存数据库承担，内存中找不到才去访问磁盘数据库；写数据则是写入磁盘数据库，不影响内存数据库访问速度；内存数据库定期与磁盘数据库同步，从磁盘数据库导入新写入数据、或是把内存计算结果持久化到磁盘上），达到既能保证高速访问速度、又能持久化存储数据的目的。



■ 混合分区架构（Hybrid Shard）

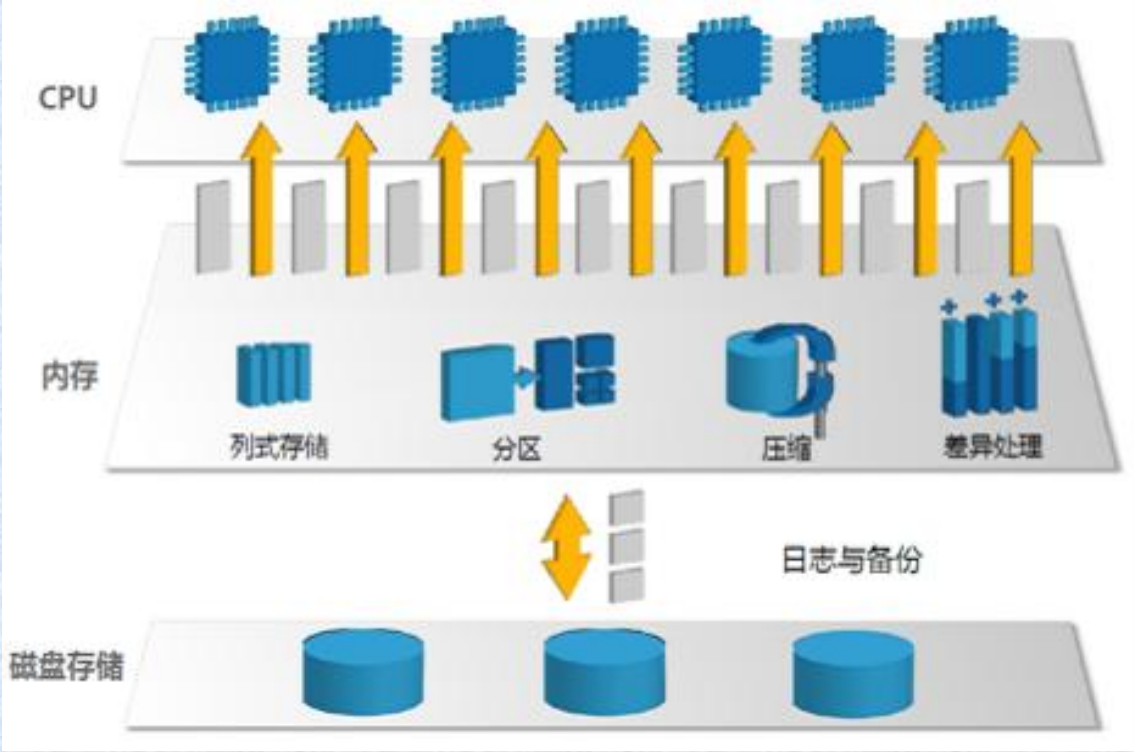
即使采用可扩容的集群架构仍需解决内存数据持久化问题，因此在集群分区方中采用混合模式（Hybrid Shard），即每个分区由一个内存数据库节点和一个MySQL节点共同组成。原来一个MySQL节点承担的一个水平分区现在变成　H-Shard = MMDB + MySQL.

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），如图所示。



内存数据库产品

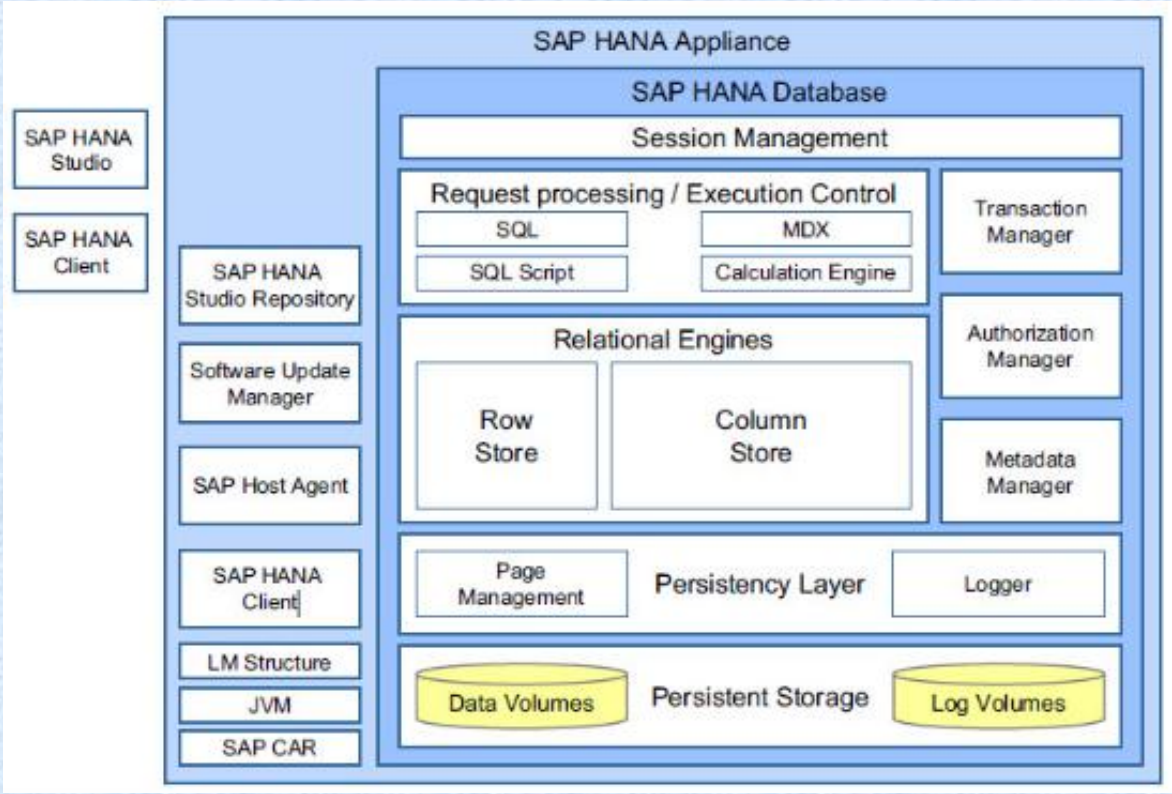
德国SAP公司的HANA（High-performance Analytic Appliance）是一个软硬件结合支持内存计算模式的高性能计算分析平台。HANA的分层计算模式如图所示。



HANA的分层计算模式

HANA内存数据库

对应于上述分层计算模式，HANA设计了图所示的计算架构。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。

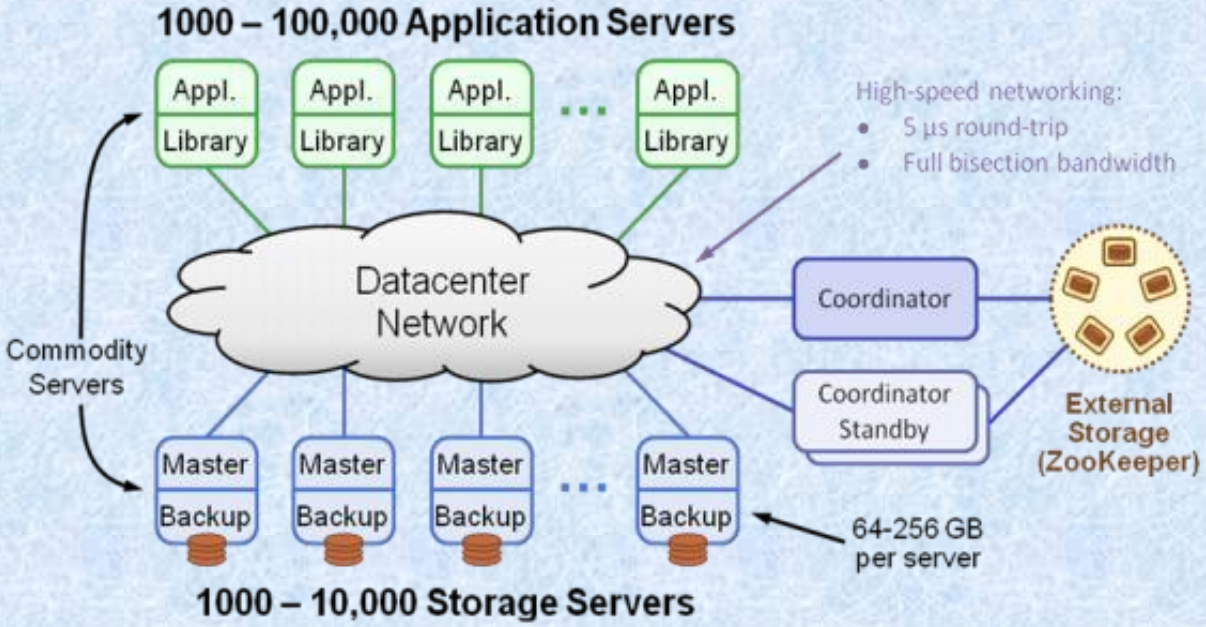


HANA的分层计算模式

2. MemCloud计算架构

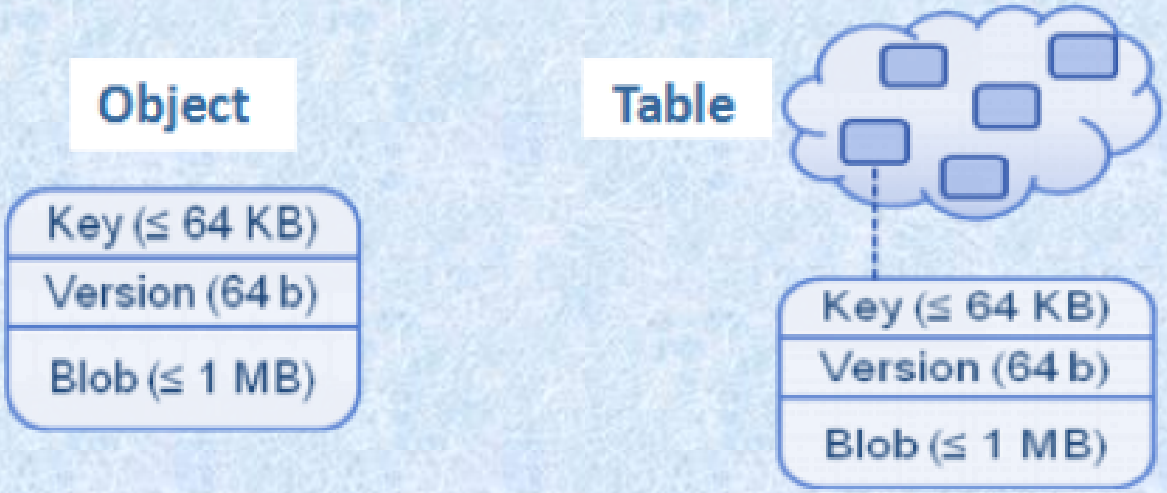
系统架构

每个RAMCloud 节点的计算架构如图所示，它包含Master和Backup两个模块：Master模块管理节点主内存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。



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，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。

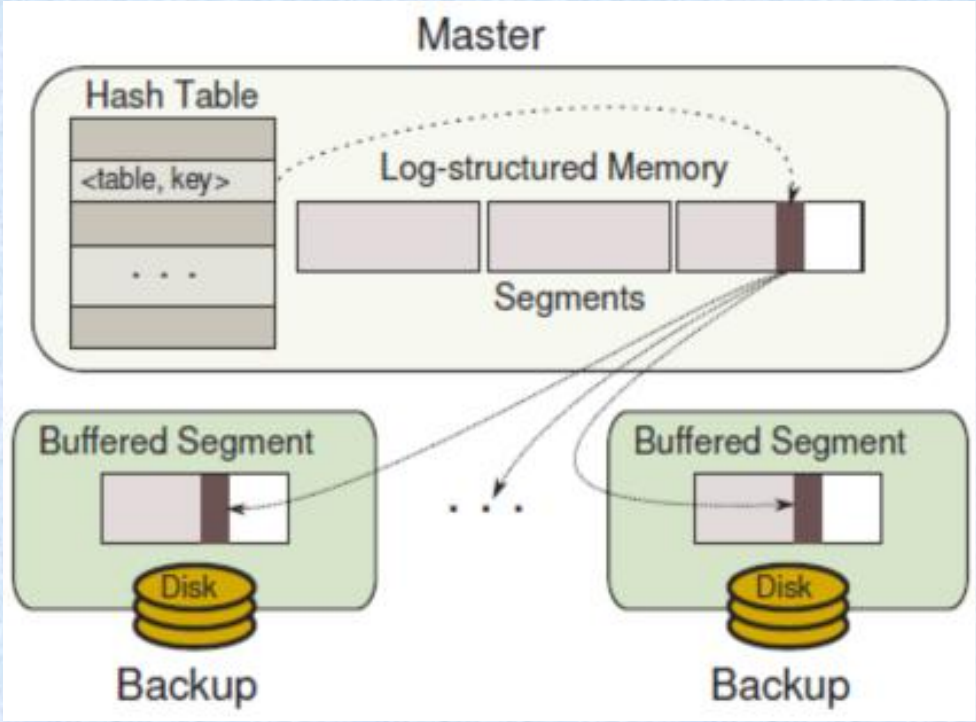


节点存储结构

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，表里面每一条entry都对应着内存里存放的一个Object。多个Objects组成内存里一个Segment (64MB)。

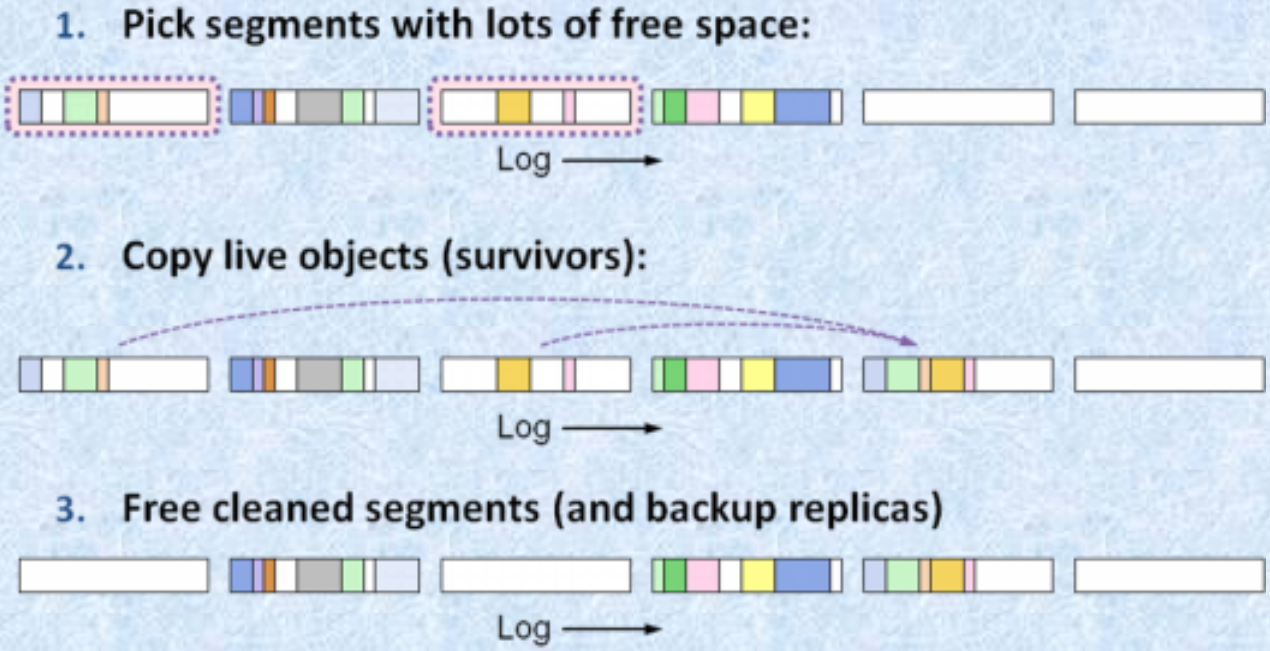
RAMCloud采用了日志形式结构（Log-structured Memory）来划分内存，即Object（数据）是以日志队列形式存储在内存里，而且这个队列是append-only类型，即新Object只能添加在队尾，而不是插入。

每个Segment生成2~3个副本，分散在集群中其他节点持久化存储在当地磁盘上，由Backup程序管理。



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空间没有使用完，即内存碎片化（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：



内存清除效率

当内存使用率低时（有较多空余内存空间），进行内存清除工作有较高收益；内存使用率高时（空余内存空间已很小），内存清除的收益就很低。

Segment中仍有效的Log的百分比 U	50%	90%	99%
需要迁移的百分比U	50%	90%	99%
释放的百分比 (1 - U)	50%	10%	1%
清除效率 = (1 - U) / U	100%	11%	1%

另外注意到RAMCloud在Master内存和Backup磁盘上保留了两套Segment体系，因此，清除工作也需要对两套体系完成。早期的对内存和磁盘的清除工作是结合在一起进行，但发现的一个问题是，当内存使用率高时（80~90%），与内存清除同时进行的磁盘清除占用了大量的网络带宽（这时磁盘使用率也高，迁移数据需要很大的开销），影响了集群写入数据（Write Data）的效率（Throughput）。

但RAMCloud的设计目标是达到内存高使用率的同时，也要有较高的写数据效率（Write Throughput）。对比内存与磁盘的性能特点可看出，内存空间昂贵，但读写带宽足够；磁盘存储空间廉价富余，但带宽有限。如何把性能特点相差悬殊的两种存储结构放在一个清除流程中达到较理想效果？--- Two-level Cleaning

	Space	Bandwidth
Memory	expensive	cheap
Disk	cheap	expensive

RAMCloud 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，同步进行。