OceanBase——海量结构化数据的分布式存储解决方案

OceanBase是为满足海量结构化数据存储需求而设计的。她具有良好的可扩展性，同时提供结构化数据的支持，较好地支持了业务相对复杂的系统对海量数据存储的需求。

# 介绍

随着互联网的进一步普及，互联网应用的用户进一步增长，对现有的很多架构提出了挑战。这在存储系统方面尤为突出。

首先，互联网应用需要存储的数据直接由用户生成，而不再是由应用提供者提供。随着用户量的增长，用户生成的数据量也随之增长，对系统能支撑的数据量提出了更高的要求，进而对底层的存储系统的可扩展性要求也越来越高。

随着移动互联网用户的增多，用户在线时间大大增长，这对服务的可用性提出了更高的要求，任何时刻服务不可用影响的用户数可能都很大。

用户对服务的用户体验要求也越来越高，用户体验的一个重要因素便是响应时间，底层存储系统的响应时间非常关键。

很多应用开始阶段为了开发效率，往往不会太多的考虑系统的可扩展性，尤其是底层的存储系统，比如数据存储大多使用类似MySQL的关系型数据库。这在开始阶段工作的很好，随着用户数的增长，扩展性的问题越来越突出。由于使用传统的关系型数据库，现有的数据模型通常较为复杂，使得其向很多NoSQL产品（比如key/value系统）迁移比较困难。

OceanBase在设计的时候便考虑了以上的问题，她兼顾了NoSQL存储系统的可扩展性和传统关系型数据库在数据结构表达上的便利性，为解决以上问题提供了一种新的思路。

本文介绍了OceanBase的设计与实现，并简单介绍了其在淘宝收藏夹应用中的使用。第二部分介绍OceanBase的数据模型。第三部分介绍客户端的接口。第四部分介绍了其具体的实现。第五部分展示OceanBase的性能数据。最后一部分我们介绍OceanBase在收藏夹中的应用。

# 数据模型

OceanBase支持应用的概念（这类似与数据库中的schema），一个应用可以创建多张table，table中包含一系列的列。table之间支持基于列的关联（join）关系。应用在使用前需要先创建数据模型，并支持在运行过程中动态修改数据模型。

## Rowkey

每张表都需要指定一个rowkey，rowkey的最大长度可以在创建时指定，其内容为二进制字符串（Binary String）。Rowkey在一张表内需要确保唯一（这和关系型数据库中的主键类似），OceanBase在内部存储时，数据按照rowkey排序。一张表的数据会根据rowkey动态的切分，切分后的单位为tablet，tablet由startKey和endKey指定其负责的数据的范围。

OceanBase的rowkey还支持split属性。该属性指定了rowkey内容的一个前缀，当数据在被动态切分时，系统会确保split后前缀内容相同的rowkey所对应的数据不会被切分至多个tablet中。

比如在淘宝的收藏夹应用中，用户可以收藏某个店铺，也可以收藏某件商品。用户的一条收藏记录可以由{userId, object type , object id}唯一确定，所以将其作为rowkey，分别表示收藏者的用户Id，被收藏对象的类型（店铺或者商品）和被收藏对象的Id，其中userId为一个8字节的整型。我们为了确保同一个用户的收藏在动态切分时分布在同一个tablet中，则可以指定rowkey在8字节的位置split。

## Column

一张表可以创建多个列，OceanBase中列支持的数据类型包括：

|  |  |
| --- | --- |
| 类型 | 说明 |
| Int | 整数，范围为[-2^63, 2^63] |
| Varchar | 字节流，可以指定其最大长度，比如varchar(1024) |
| Datetime | 表示自1970-1-1 00:00:00到现在的秒数 |
| Precise\_datetime | 表示自1970-1-1 00:00:00到现在的微秒数 |
| Create\_time | 和Precise\_datetime类似，该列的值由数据新增时系统自动生成，不支持修改。 |
| Modify\_time | 和Precise\_datetime类似，该列的值由数据更新时系统自动生成，不支持修改。 |

## 关联

为了简化对有关联的数据的查询操作，OceanBase支持在创建表时指定表之间的关联关系。当客户端查询时，服务器端根据表之间的关联关系定义，自动将相应的数据合并，然后将合并后的数据结果返回给客户端。

比如下面的关联关系定义：

|  |
| --- |
| Join=rowkey[8,16]%collect\_item\_info:item\_name$item\_name,item\_price$new\_price |

该关联关系定义了当前表和collect\_item\_info表之间的关联关系，关联关系由该表rowkey的8-16字节与collect\_item\_info的rowkey指定。当查询该表时，将使用collect\_item\_info中的item\_name和new\_price分别和当前表的item\_name和item\_price合并，合并后的结果返回给用户。合并操作不修改OceanBase中存储的数据，只影响返回给用户的结果。

# API

OceanBase的API提供了查询单条数据，查询范围，新增，修改，删除数据的功能。

## 查询

客户端可以获取rowkey指定的单条数据，也可以获取由startKey和endKey指定范围的数据集合。获取数据时可以指定需要获取的列。如果是获取数据集合，还可以指定这些数据的排序规则，数据的起始位置（类似MySQL中offset）和单次查询返回的数据量（类似MySQL中的limit），排序的字段需要包含在指定获取的字段列表中。

下面的代码便是使用Java客户端获取一个范围数据的例子：

|  |
| --- |
| QueryInfo qinfo = new QueryInfo();  qinfo.setStartKey(startKey);  qinfo.setEndKey(endKey);  qinfo.addColumn("title").addColumn("price");  qinfo.addOrderBy("price", true).addOrderBy("title", false);  results = client.query(table, qinfo);  for (RowData row : results.getResult())  {  System.out.print(row.get("title") + ", ");  System.out.println(row.get("price"));  } |

## 更新

OceanBase的更新接口支持新增，修改和删除操作。一次可以提交对多张表的多个rowkey操作，这些操作都是在同一个事务中，即要么全部失败，要么全部成功。

|  |
| --- |
| ObMutator mutator;  // 在table1中更新rowkey1对应的column1和column2  mutator.addCell(new ObCell(table1, rowkey1, column1, value1), OB\_UPDATE);  mutator.addCell(new ObCell(table1, rowkey1, column2, value2), OB\_UPDATE);  // 在table1中更新rowkey2对应的column1  mutator.addCell(new ObCell(table1, rowkey2, column1, value1), OB\_UPDATE);  // 在table2中新增rowkey3, 包含column1和column2两列  mutator.addCell(new ObCell(table2, rowkey3, column1, value1), OB\_INSERT);  mutator.addCell(new ObCell(table2, rowkey3, column2, value2), OB\_INSERT);  // 删除table3中的rowkey1对应的整行数据  mutator.addCell(new ObCell(table3, rowkey1), OB\_DEL\_ROW);  // 执行以上操作  client.apply(mutator); |

OceanBase的更新操作和传统关系型数据库的语义稍有不同。如果在新增时，对应的数据已经存在，系统不会返回失败，而是使用新的内容更新老的内容。同理，如果更新的时候rowkey指定的数据不存在，则update的语义类似insert。

# 实现

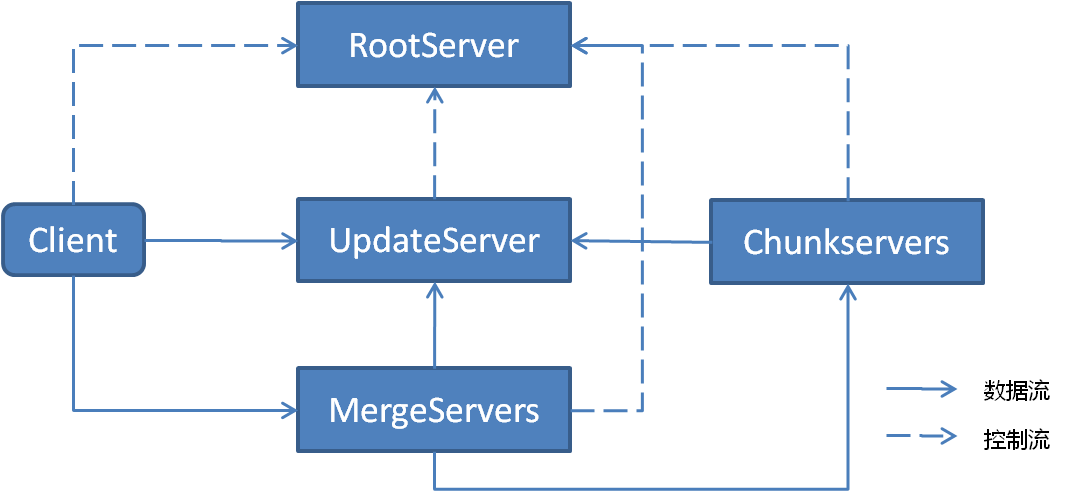
OceanBase系统主要包括5个组件：维护系统元数据的rootserver，服务更新操作的updateserver，存储静态数据的chunkserver，服务查询请求的mergeserver和为应用提供服务接口的客户端。

## 整体架构

OceanBase将表的数据动态切分为tablet，tablet的数据分为动态和静态两部分。静态的数据存放在chunkserver上，所有对数据的修改都存储在updateserver中。updateserver的修改定期同步到chunkserver，chunkserver将updateserver的更新和本地的静态数据合并，生成合并后的新数据。

tablet的信息由rootserver维护，客户端在初始化时会请求rootserver，获取updateserver的地址信息。客户端的更新请求（包括新增，修改和删除）都直接访问updateserver。查询请求时客户端根据相应的rowkey向rootserver查询其对应的tablet信息，rootserver返回相应的mergeserver地址，客户端根据返回的信息请求相应的mergeserver获取数据。

Mergeserver收到请求时，根据rowkey从rootserver获取相应的tablet信息，该信息中包括负责该tablet的chunkserver列表，mergeserver请求相应的chunkserver，获取静态数据（如果有的话），然后根据返回的数据，请求updateserver获取相应的更新数据，将更新数据和静态的数据合并，将合并后的结果返回给客户端。OceanBase的整体架构如图1。



图表 1 OceanBase整体架构

## tablet

tablet主要包括元信息和地址信息，都由rootserver维护。

元信息主要包括该tablet负责的数据的范围（由startKey和endKey指定），tablet的数据条数、大小以及tablet的checksum等。

tablet地址信息主要包括负责存储该tablet的chunkserver列表，以及负责对外提供该tablet查询服务的mergeserver列表。客户端在查询时需要从rootserver获取tablet对应的mergeserver信息，mergeserver在查询时需要从rootserver获取负责存储该tablet的chunkserver信息。为了减少对rootserver的请求，客户端和mergeserver都会缓存tablet的地址信息。

## rootserver

rootserver主要负责管理和维护OceanBase集群的元数据信息，包括tablet信息，updateserver地址，chunkserer列表及其状态，表的schema信息等。

rootserver需要维护chunkserver的状态，chunkserver启动时会自动向rootserver注册，并和rootserver维持一个租约。rootserver定期通知chunkserver更新租约，如果chunkserver在规定时间内没有更新租约，则认为chunkserver不可用。

当集群中的chunkserver发生变化时，比如节点不可用或者新加入节点，rootserver会发起迁移命令，并负责管理迁移的状态。

## updateserver

大部分存储系统最后的瓶颈都在IO上，传统磁盘上随机访问性能通常较差，所以提高存储系统整体性能很重要的一个方法便是避免随机读写，尽量采用顺序读写。

OceanBase采用了一种更为彻底的方法，将所有的更新，包括新增、修改和删除都统一交由updateserver负责，物理的将数据分割为动态和静态两部分。静态的数据存储在chunkserver上，动态的更新全部存在updateserver中。

updateserver将更新数据全部存储在内存中，使用自己开发的支持COW（Copy On Write）的B+树作为存储结构，得益于B+树的COW特性，读写之间不互相干扰，updateserver能达到较高的性能。

updateserver定期的将内存中的数据冻结，同时切换至新的内存表提供服务，冻结的内存表将不再提供更新服务，但提供查询服务。当chunkserver合并成功后，rootserver会通知updateserver卸载冻结内存表，此时updateserver将冻结的内存表释放。

由于使用单个updateserver提供更新服务，OceanBase可以比较容易的做到多表之间更新的事物。

## chunkserver

chunkserver主要负责tablet静态数据的存储，tablet的数据以SSTable的形式存储在磁盘上，一个tablet可以对应多个SSTable文件。chunkserver还负责接收updateserver的更新数据，将更新数据合并到本地。tablet支持备份，当tablet需要发生迁移时，chunkserver负责将tablet数据迁移至目标节点。

chunkserver主要的工作便是服务对数据的读取请求。当接收到读取请求时，chunkserver根据请求的rowkey找到对应的Tablet，然后访问该tablet对应的SSTable文件，将相应的数据返回给请求者。

为了提高读取的性能，chunkserver对部分数据结构进行了缓存。一个SSTable由多个block组成，为了加快定位需要请求的数据位于SSTable的哪个block中，chunkserver包含一个block index的功能，block index由该block负责的数据的最后一个key和该block在SSTable文件中的位置组成。为了提高block的读取性能，chunkserver还将block缓存在内存中。Block index和block的cache都采用LRU的策略淘汰。

为了减小SSTable在磁盘上所占用的空间，数据在存储时会经过压缩。可以配置使用不通的压缩算法，比如LZO。

updateserver会将数据的修改以patch的方式发送给chunkserver，chunkserver负责将patch合并到本地原有的SSTable中。在合并完成前，chunkserver在服务读取请求时，会将本地原有SSTable中的数据和patch中的数据merge，返回请求方merge后的结果数据，直至patch merge完成为止。

由于ChunkServer只有在合并顺序的时候有写入，并且是顺序写入，这是的ChunkServer和使用采用SSD作为存储介质。

## mergeserver

当数据被物理的分割为动态和静态两部分后，带来的一个问题便是用户怎么获取最新的数据。OceanBase引入MergeServer用于解决该问题。

MergeServer向OceanBase客户端提供查询服务，当收到查询请求时，MergeServer先向ChunkServer发送请求以获得相应的静态数据，然后根据ChunkServer返回的静态数据，向updateserver获取这些数据的更新，将更新数据和静态数据合并，并按照请求设置的排序规则执行排序，然后将结果数据返回给请求者。

## 容灾

在OceanBase中，数据通常存储多个备份，这些备份会分布在不通的chunkserver上，所以单台chunkserver不可用不会影响可用性。

为了增加rootserver和updateserver的容错性，我们对rootserver和updateserver做了Master/Slave的容灾。

我们使用HA的方式，即rootserver和updateserver对外提供服务器时使用一个虚拟IP（VIP），由HA检测主备节点的状态，如果主节点不可用，则虚拟IP会自动漂移到备节点，备节点接着提供服务。

主节点和备节点之间使用update log同步，主节点将修改以update log的形式发送给备节点，备节点收到日志后马上返回主节点成功，然后将日志写入本地的文件，由一个日志回放线程将更新日志应用到本地。当发生切换时，备节点会确保本地日志应用完成再对外提供服务。

为了减少节点恢复的时间，我们也添加了checkpoint的支持，节点恢复时，从当前的主节点上获取checkpoint文件，从checkpoint文件恢复，然后应用该checkpoint后的更新日志即可。

# 性能

//TODO 等性能结果出来以后再补充

# 应用

OceanBase在淘宝已经投入生产环境使用，这里我们以收藏夹作为例子说明OceanBase的使用情况。

## 收藏夹介绍

淘宝收藏夹是深受淘宝用户喜爱的一个服务，用户可以在收藏夹中添加自己感兴趣的商品，店铺等信息。访问收藏夹时，用户可以看到自己收藏的所有商品和店铺，收藏夹会显示被收藏的商品的价格，卖家，收藏时间等信息，为用户关注、比较、访问相关信息提供了一个有效、便捷的途径。

## 面临的问题

收藏夹的数据原本存储在MySQL中，主要包括收藏信息和被收藏的商品或店铺信息。随着用户的增加，数据量增长很快，收藏信息接近百亿。MySQL进行了分库分表操作，由于单张表的数据量仍然很大，对于一些复杂的查询，比如按照价格排序，系统开销很大。

为了确保主体功能，收藏夹进行了一些优化，有些优化甚至牺牲了用户体验，比如按照价格排序不再是针对所有的收藏商品，而改为在当前显示的页面上使用客户端的Javascript排序。

收藏夹的现有数据存储方案，已经成为功能进一步发展的瓶颈，限制了很多功能的改进和新增。

## OceanBase的解决方案

收藏夹主要的数据结构便是用户的收藏信息，以及所收藏的对象（商品或者店铺）信息。在之前的MySQL存储方案中，这两者分别有自己的表。在了解了收藏夹的特性后，在OceanBase中，我们的方案如下：

1. 在chunkserver上新建一张表，该表包含收藏信息和被收藏的对象信息，我们称之为宽表。
2. 在updateserver上新建两张表，分别存储收藏信息和被收藏的对象信息。

在chunkserver的宽表中，我们冗余了被收藏的对象信息，这带来了一定的存储空间开销，但是在查询的时候，我们只需要访问这一张表即可。减少了IO的次数，带来的性能上的提升。

updateserver将收藏信息和被收藏的对象信息分开，这样当被收藏的对象发生修改时，通过和宽表中的数据合并，可以将最新的对象信息展示在收藏信息结果中。

收藏信息的rowkey使用用户id、类型（商品/店铺）和对象Id（商品id/店铺id）组合而成，由于用户对同一件商品/店铺只能收藏一次，所以这个组合可以保证收藏信息的唯一性。rowkey的结构为：

|  |  |  |
| --- | --- | --- |
| userId（8 byte） | Object type （1 byte） | Object Id （8 byte） |

我们设置rowkey在userId处split，这样同一个用户的数据会被存放在同一个tablet中，数据在存放时会按照rowkey排序，所以同一个用户内的数据，相同类型的数据会被连续存放，这可以加快用户指定收藏对象类型查询时的速度。

用户可以对所收藏的商品执行排序，比如按照商品价格，收藏时间等。使用MySQL时，为了减少数据库服务器的压力，排序是在页面上针对当前页排序的。在使用OceanBase后，MergeServer会获取该用户相应的完整数据，在完整的数据中排序，返回用户最终结果，这解决了困挠收藏夹用户很久的一个问题。

借助OceanBase的高可扩展性，收藏夹在功能上的改进有了保障，相信以后会有更多更好的功能呈献给用户。

# 总结

本文简单介绍了OceanBase设计时期望解决的问题，并介绍了其具体的实现。将数据物理切分为静态和动态的两部分的思路，不仅简化了我们的设计和实现，还拓宽了针对更新和查询的优化空间。在淘宝收藏夹中的使用，验证了我们的设计和实现。OceanBase为海量结构化数据的存储提供了一种新的思路。