# 背景

# 概述

# 架构

# 原理

# 事务与并发

# 数据分布

# 复制/一致性

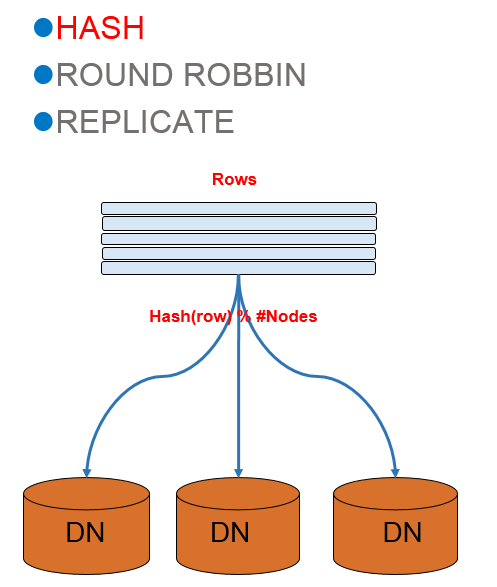
# 导入导出

# 备份恢复

# 兼容性

# 扩展性

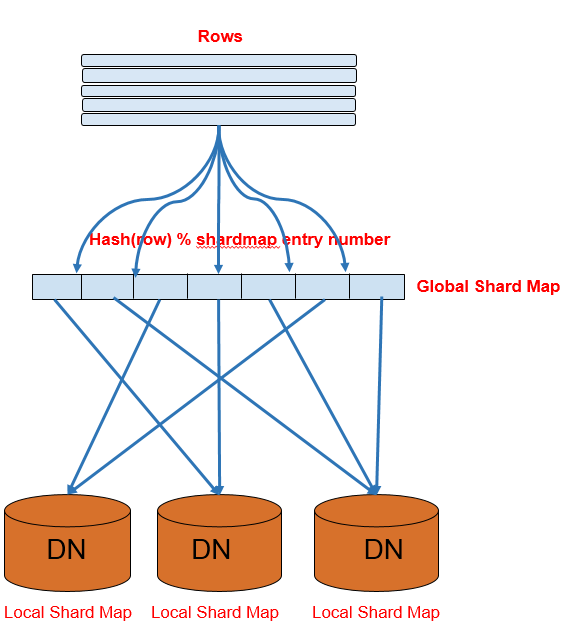
为了迎接业务的快速增长，系统不可避免的需要进行扩容，传统的分布式数据库所采用分库分表模式的实现使得扩容成本高昂，需要对业务进行长时间的中断。一般情况下，数据分布方式有以下几种，HASH最常用。



在HASH分布的情况下，通过如下方式决定一条记录的存储节点：

**DN = Hash(row) % nofdn**

也就是说先对分布列计算hash值，然后使用这个值对集群中的节点个数取模来决定记录去哪个节点。这里有一个问题，扩容后节点数会变多，数据分布的计算逻辑就会导致已经存在的数据无法正常访问。为了解决这个问题，就必须把业务停掉，把所有数据导出，扩容后重新导入，在数据量较多时，这个过程会持续几天，这对一个每天几千万笔交易的系统来说显然是不能接受的。为了解决这个问题，TBase引入了一种新的分表方法：sharded table。Sharded table的数据分布采用如下的方式：



1. 引入一个抽象的中间层-shard map，Shard map中每一项存储shardid和DN的映射关系。
2. Sharded table中的每条记录通过 **Hash(row) % #shardmap entry**来决定记录存储到哪个shardid，通过查询shardmap的存储的DN。
3. 每个DN上存储分配到本节点shardid信息，进而进行可见性的判断

有了上面的支撑，在我们新加节点时，我们只需要把一些shardmap中的shardid映射到新加的节点，并把对应的数据搬迁过去就可以了。扩容对业务的中断也就仅仅是在切换shardmap中映射关系的时刻，时间大大缩短。

## 列存表在线扩容能力

在超大数据量情况下，用户可以选择用列存压缩表来降低用户存储成本。同时我们也针对列存表设计了高效的扩容方案。

在存储结构上，列存表为了支持多并发变更问题，引入了Slice的概念。简单讲，数据插入时系统会为每个并发会分配一个活跃Slice进行写入操作。而上述行存表shardmap逻辑被保留了下来，但是针对列存表需要进一步改进。为了避免搬迁时复杂内核逻辑带来的性能下降，我们引入了文件直接搬迁方式。这就需要存储结构做一些调整，系统初始状态时我们把shard分为8个shardgroup，相当于对shardmap进行了二次路由。每一个shardgroup的数据写在相同物理文件中，这样在搬迁的时候我们就不需要对文件内部结构进行解析，直接把其中部分shardgroup通过物理文件copy的方式迁移到新的节点中。极大的提高了搬迁效率。

文件二级shardmap路由如下图所示：

图示

描述已自动生成

而列存表设计同样支持在线扩容的方式，那么我们就需要一套完整的扩容流程来保证增量数据也被同步过来。这里主要是通过物理文件搬迁 + 增量XLOG追赶的方式完成整个扩容流程。具体流程如下图所示：图示

描述已自动生成

# 高并发

## Forward Node

## 分布式执行框架

## 向量化引擎

## 延迟物化技术

# 高可用

# 数据安全

# 数据压缩

# 数据迁移

# 数据校对

# 运维/监控告警