# 概述

参考：<https://blog.csdn.net/baichoufei90/article/details/84328666>

<https://baijiahao.baidu.com/s?id=1717147765473335948&wfr=spider&for=pc>

MPP，全称为Massively Parallel Processor，翻译过来就是大规模并行处理。MPP系统是由许多松耦合的处理单元组成的（要注意的是这里指的是处理单元而不是处理器）。每个处理单元内的CPU都有自己私有的资源，如总线，内存，硬盘等，且都有操作系统和管理数据库的实例复本。这种结构最大的特点在于不共享资源(share-nothing)。

MPP处理数据的思路：

面对海量数据和计算时，采用大事化小的思路，对数据进行分割，数据分割后单独存储，数据处理消耗的资源也是相互隔开的，对于MPP数据库来讲，整个数据库由多个完全独立的数据库构成，各个拥有完整的数据存储、数据管理、数据操作能力。基于网络实现节点互联，形成一个整体对外提供服务，节点间互不干扰，即Share Nothing，不共享磁盘和计算能力。

MPP架构有如下特点：

Share Nothing、节点之间数据不共享，只有通过网络连接实现的协同。

每个节点有独立的存储和内存。

数据根据某种规则(如Hash)散布到各个节点。

计算任务也是会发布到各个节点并行执行，最后再将结果聚合到整体返回。

用户使用时会看做整体。

MPP数据库（如GreePlum）往往优先考虑C一致性，然后是A可用性，最后考虑P分区容忍。

MPP架构目前被并行数据库广泛采用，一般通过scan、sort和merge等操作符实时返回查询结果。

## 并行查询 vs MPP

这里的并行查询主要侧重于单机的多线程查询，比如MySQL并发、PG并发等；MPP主要是多机器的并行查询，主要应用于大数据领域，比如Presto、ClickHouse等。

## 优点

任务并行执行；

数据分布式存储(本地化)；

分布式计算；

高可用、易维护：数据通过副本提供冗余保护，自动故障探测和管理，自动同步元数据和业务数据。提供图形化工具，以简化管理员对数据库的管理工作；

高并发：读写不互斥，支持数据的边加载边查询，单个节点并发能力大于300 用户；

高扩展、高可靠：支持集群节点的扩容和缩容，支持全量、增量的备份/恢复；

行列混合存储：提供行列混合存储方案，从而提高了列存数据库特殊查询场景的查询响应耗时；

标准化：支持SQL92标准，支持C API、ODBC、JDBC、ADO.NET等接口规范。

## 缺点

1、很难高可用 -> 影响可用性和可靠性

因为数据按某种规则如HASH已经散布到了各个节点上。

2、节点数=任务并行数 -> 影响扩展性

一个作业提交时，每个节点都要执行相同任务。而不像MapReduce那样做了根据实际开销进行任务拆分后散发到有资源的几个节点上。这一点大大影响了MPP架构应用的可扩展性。

3、每个客户端同时连接所有节点通信 -> 影响网络

MPP架构每个节点独立，所以客户端往往需要连接所有节点进行通信，这使得网络也成为瓶颈。

4、分区容错性差

前面提到过MPP主要考虑CA，最次才是P。那么一旦扩展节点太多后，元数据管理十分困难。

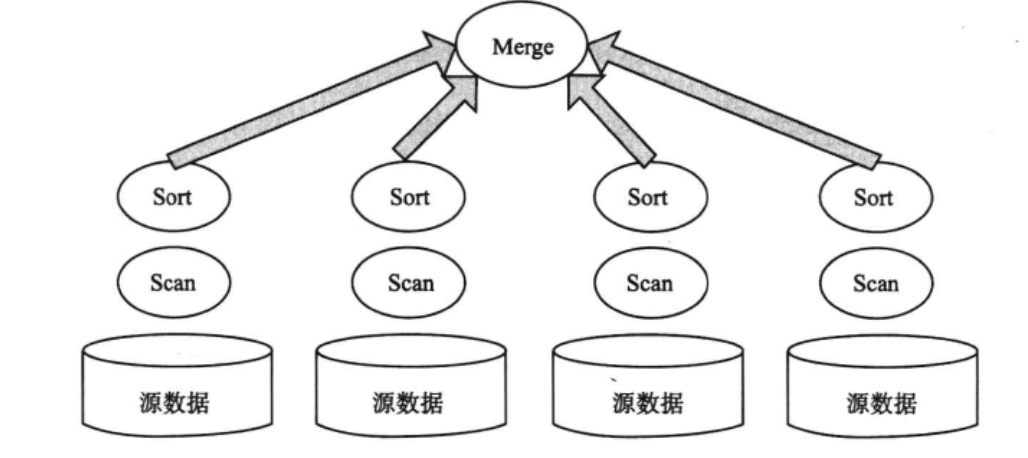
# 架构

## MPP架构

并行数据库往往采用MPP（Massively Parallel Processing，大规模并行处理）架构。MPP架构是一种**不共享**的结构，每个节点可以运行自己的操作系统、数据库等。每个节点内的CPU不能访问另一个节点的内存，节点之间的信息交互是通过节点互联网络实现的。

如图所示，将数据分布到多个节点，每个节点扫描本地数据，并由Merge操作符执行结果汇总。

**MPP Merge操作符：**



常见的数据分布算法有两种：

范围分区（Range Partitioning）：按照范围划分数据。

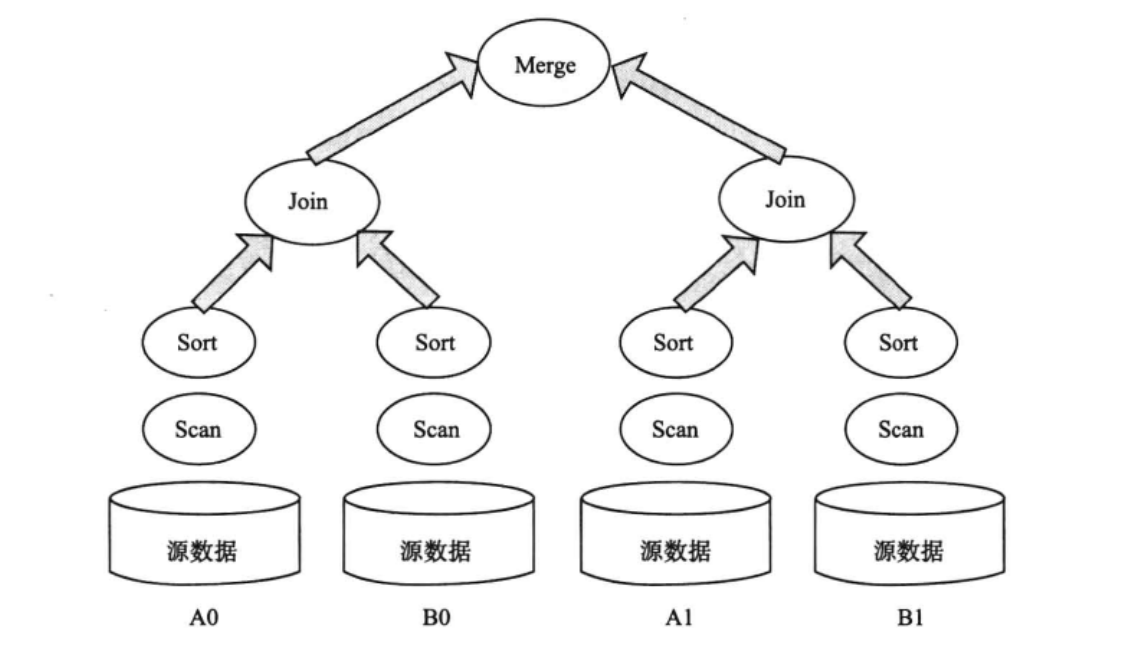
哈希分区（Hashing）：根据哈希函数计算结果将每个元祖分配给相应的节点。

Merge操作符：系统中存在一个或多个合并节点，它会发送命令给每个数据分片请求相应的数据，每个数据分片所在的节点扫描本地数据，排序后回复合并节点，由合并节点通过merge操作符执行数据汇总。**Merge操作符是一个统称，涉及的操作可能是limit、order by、group by、join等**。这个过程相当于执行一个Reduce任务个数为1的MapReduce作业，不同的是，这里不需要考虑执行过程中服务器出现故障的情况。

如果Merge节点处理的数据量特别大，可以通过Split操作符将数据划分到多个节点，每个节点对一部分数据执行group by、join等操作后再合并最终结果。

如图所示，假如需要执行“select \* from A,B where A.x=B.y”，可以分别根据A.x和B.x的哈希值将表A和B划分为A0、A1和B0、B1。由两个节点分别对A0、B0以及A1、B1执行join操作后再合并join结果。

**MPP Split操作符：**



并行数据库的SQL查询和MapReduce计算有些类似，可以认为MapReduce模型是一种更高层次的抽象。由于考虑问题的角度不同，并行数据库处理的SQL查询执行时间通常很短，出现异常时整个操作重做即可，不需要像MapReduce实现那样引入一个主控节点管理计算节点，监控计算节点故障，启动备份任务等。

## 其他架构

### SMP

### NUMA

### Hadoop

### 集群和MPP

# 产品

参考：<https://mp.weixin.qq.com/s/10YI8bXcjaxk48RQPGDShw>

## Impala

## Presto

## ClickHouse

参考：<https://developer.volcengine.com/articles/7176266450804408377>

### 背景

ClickHouse的执行模式与Druid、ES等大数据引擎类似，其基本的查询模式可分为两个阶段。第一阶段，Coordinator在收到查询后，将请求发送给对应的Worker节点。第二阶段，Worker节点完成计算，Coordinator在收到各Worker节点的数据后进行汇聚和处理，并将处理后的结果返回。

两阶段的执行模式能够较为高效地支持目前许多常见的业务场景，例如各类大宽表单的查询，这也是ClickHouse最擅长的场景。ClickHouse的优点是简单、高效，通常来说，简单就意味着高效。但随着企业业务的持续发展，愈加复杂的业务场景对ClickHouse提出了以下三类挑战。

第一类，当一阶段返回的数据较多，且二阶段计算较为复杂时，Coordinator会承受较大压力，容易成为Query的瓶颈。 例如一些重计算的Agg算子，如Count Distinct，若采用哈希表的方式进行去重，第二阶段需在Coordinator单机上去合并各个Worker的哈希表。这个计算量会很重且无法并行。

第二类，由于目前ClickHouse模式并不支持Shuffle，因此对于Join而言，右表必须为全量数据。 无论是普通Join还是Global Join，当右表的数据量较大时，若将数据都放到内存中，会比较容易OOM。若将数据spill到磁盘，虽然可以解决内存问题，但由于有磁盘 IO 和数据序列化、反序列化的代价，因此查询的性能会受到影响。特别是当Join采用Hash Join时，如果右表是一张大表，构建也会比较慢。针对构建问题，近期社区也进行了一些右表并行构建的优化，数据按照Join key进行Split来并行地构建多个Hash Table，但额外的代价是左右表都需要增加一次Split操作。

第三类，则是关于复杂查询（如多表 Join、嵌套多个子查询、window function 等），ClickHouse对这类需求场景的支持并不是特别友好， 由于ClickHouse并不能通过Shuffle来分散数据增加执行并行度，并且其生成的Pipeline在一些case下并不能充分并行。因此在某些场景下，难以发挥集群的全部资源。

随着企业业务复杂度的不断提升，复杂查询，特别是有多轮的分布式Join，且有很多agg的计算的需求会越来越强烈。在这种情况下，业务并不希望所有的Query都按照ClickHouse擅长的模式进行，即通过上游数据 ETL 来产生大宽表。这样做对ETL的成本较大，并且可能会有一些数据冗余。企业的集群资源是有限的，但整体的数据量会持续增长，因此在这种情况下，我们希望能够充分地去利用机器的资源，来应对这种越来越复杂的业务场景和SQL。所以我们的目标是基于ClickHouse能够高效支持复杂查询。

### 方案

对于ClickHouse复杂查询的实现，我们采用了分Stage的执行方式，来替换掉目前ClickHouse的两阶段执行方式。类似于其他的分布式数据库引擎，例如Presto等，会将一个复杂的Query按数据交换情况切分成多个 Stage，各Stage之间则通过Exchange完成数据交换。

Stage之间的数据交换主要有以下三种形式。

* 按照单个或者多个key进行Shuffle
* 将单个或者多个节点的数据汇聚到一个节点上，称为Gather
* 将同一份数据复制到多个节点上，称为Broadcast或广播

对于单个Stage执行，继续复用ClickHouse目前底层的执行方式。开发上按照不同功能切分不同模块。各个模块预定接口，减少彼此的依赖与耦合。即使模块发生变动或内部逻辑调整，也不会影响其他模块。其次，对模块采用插件架构，允许模块按照灵活配置支持不同的策略。这样便能够根据不同业务场景实现不同的策略。

### 优化与诊断

### 效果与展望

## Doris

## Druid

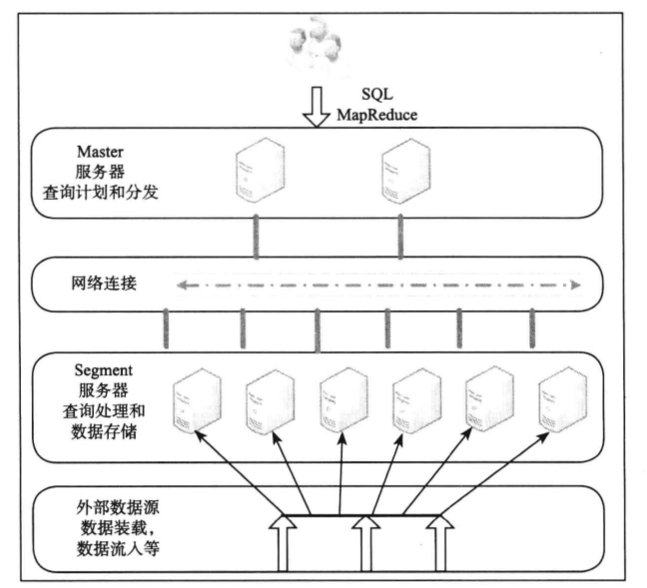
## TiDB

## EMC Greenplum

Greenplum是EMC公司研发的一款采用MPP架构的OLAP产品，底层基于开源的PostgreSQL数据库。

### 整体架构

Greenplum系统主要包含两种角色：Master服务器（Master Server）和Segment服务器（Segment Server）。在Greenplum中每个表都是分布在所有节点上的。Master服务器首先对表的某个或多个列进行哈希运算，然后根据哈希结果将表的数据分布到Segment服务器中。整个过程中Master服务器不存放任何用户数据，只是对客户端进行访问控制和存储表分布逻辑的元数据。



Greenplum支持两种访问方式：SQL和MapReduce。用户将SQL操作语句发送给Master服务器，由Master服务器执行词法解析、语法分析，生成执行计划树，并将查询请求分发给多台Segment服务器。每个Segment服务器返回部分结果后，Master服务器会进行聚合并将最终结果返回给用户。除了高效查询，Greenplum还支持通过数据的秉性装载，将外部数据秉性装载到所有的Segment服务器。

### 并行查询优化器

Greenplum的并行查询优化器负责将用户的SQL或者MapReduce请求转换为物理执行计划。Greenplum采用基于代价的查询优化算法（cost-based optimization），从各种可能的查询计划中选择一个代价最小的。Greenplum优化器会考虑集群全局统计信息，例如数据分布，另外，除了考虑单机执行的CPU、内存资源消耗，还需要考虑数据的网络传输开销。

Greenplum除了生成传统关系数据库的物理运算符，包括表格扫描（Scan）、过滤（Filter）、聚集（Aggregation）、排序（Sort）、联表（Join），还会生成一些并行运算符，用来描述查询执行过程中如何在节点之间传输数据。

广播（Brodcast，N:N）：每个极端节点将目标数据发送给所有其他节点。

重新分布（Redistribute，N:N）：类似MapReduce中的shuffle过程，每个计算节点将目标数据重新哈希后分散到所有其他节点。

汇总（Gather，N:1）：所有的计算节点将目标数据发送给某个节点（一般为Master服务器）。

## HP Vertica

Vertica是Michael Stonebraker的学术研究项目C-Store的商业版本，并最终被惠普公司收购。Vertica在架构上与OceanBase有相似之处。

### 混合存储模型

Vertica的数据包含两个部分：ROS（Read-Optimized Storage）以及WOS（Write-Optimized Storage），WOS的数据在内存中且不排序和加索引，ROS的数据在磁盘中有序且压缩存储。后台的“TUPLE MOVER”会不断地将数据从WOS读出并网ROS更新（同时完成排序和索引）。Vertica的折中设计和OceanBase很相似，ROS对应OceanBase中的ChunkServer，WOS对应OceanBase中的UpdateServer。由于后台采用“BULK”的方式批量更新，性能非常好。

### 多映射（Projections）存储

### 列式存储

Vertica中的每一列数据独立存储在磁盘的连续块上。查询数据时，Vertica只需要读取那些需要的列，而不是被选择的行的所有的列数据。

### 压缩技术

## Google Dremel

### 系统架构

### Dremel vs MapReduce

# 适用场景

集群规模100以内、并发小（50以下）。

MPP架构目前被并行数据库广泛采用，一般通过scan、sort和merge等操作符实时返回查询结果。