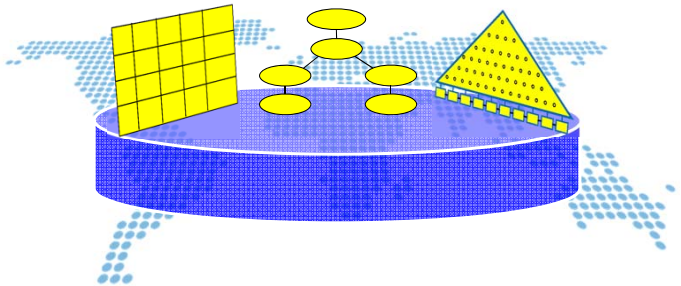
数据库系统

并行/分布式数据库

陈世敏

（中科院计算所）



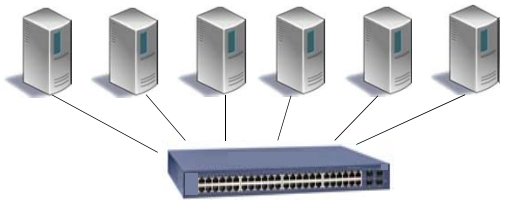
Outline

- 系统架构和关键技术
- 分布式查询处理
- 分布式事务处理

三种架构

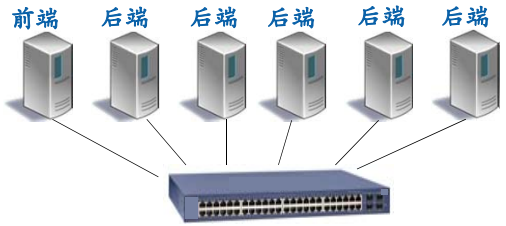
- Shared memory
  - 多芯片、多核
  - 或Distributed shared memory
- Shared disk
  - 多机连接相同的数据存储设备
  - 例如：分布式文件系统NFS，数据存储SAN设备等
- Shared nothing
  - 普通意义上的机群系统
  - 由以太网连接多台服务器

Shared Nothing



- 系统架构
- 关键技术

系统架构

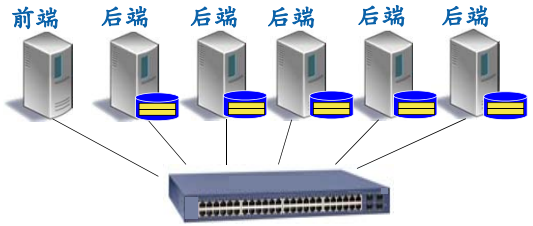


- 一个coordinator运行前端产生并行的query plan
- 每台worker服务器上都有后端
- Coordinator协调worker服务器执行

关键技术

- Partitioning（划分）
  - 把数据分布在多台服务器上
  - 通常采用Horizontal partitioning
    - 把不同的记录分布在不同的服务器上
    - 在大数据系统中又称为sharding
- Replication（备份）
  - 为了提高可靠性
  - 对性能的影响
    - 读？可能提高并行性
    - 写？额外代价（写多个副本，多个副本的一致性问题）

Horizontal Partitioning

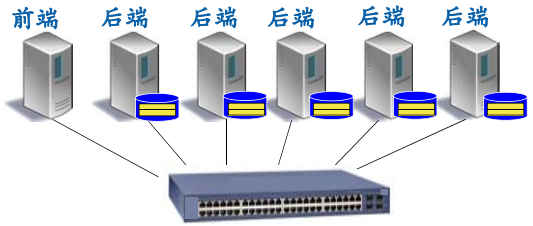


- Hash partitioning
  - 类似GRACE: machine ID = hash(key) % MachineNumber
- Range partitioning
  - 每台服务器负责一个key的区间，所有区间都不重叠
- 希望尽量负载均衡

Outline

- 系统架构和关键技术
- 分布式查询处理
- 分布式事务处理

并行执行

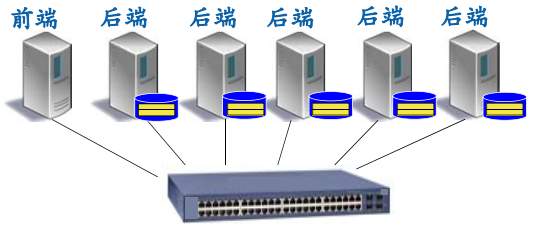


- Filter 和Project √
  - 在每条记录上完成，很容易并行计算

☞ Join 可以并行执行吗？

Join

$R \bowtie_{R.a = S.b} S$

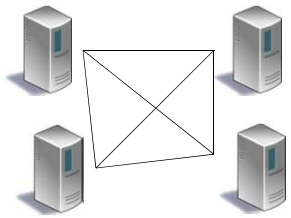


- 如果partition key就是join key，那么类似GRACE的思路，可以并行执行，每台机器单机join
- 其它情况？

partition key不是join key？

$R \bowtie_{R.a = S.b} S$

- 先在join key上进行partitioning
  - 这一步类似GRACE中的I/O partitioning
  - 如果有K台worker，那么每台机器都把本地数据划分为K个hash partitions，然后分别发送到K台worker上
  - 使同一个划分的所有数据都放在同一台机器上
  - 转化为前一种情形
- 然后在每台worker上join
- Partitioning步骤
  需要大量的数据传输

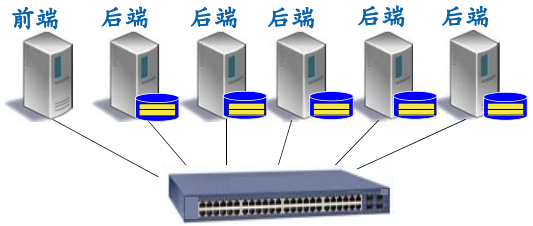


Semi Join

$R \bowtie_{R.a = S.b} S$

- 目标：优化分布式连接运算的数据传输代价
- 思路：最好把不产生匹配的记录先过滤掉
- 方法：用join key进行过滤
  - 提取S表的join key列S.b，把S.b发送到R所在的机器节点
  - R所在的机器节点，用S.b对R进行过滤
  - 然后，把过滤后的R分发到S机器上，完成join
  - 同样，可以用R.a对S进行过滤
- 这样的操作叫semi-join

并行执行



- Group-by+Aggregation
  - 每个Worker分别进行本机的Group by + Aggregation，获得中间结果
  - 分发中间结果，使同一个Group的发到同一台Worker上
  - 合并每个Group的所有Worker计算的Aggregation中间结果，得到每个Group的最终Aggregation结果

Outline

- 系统架构和关键技术
- 分布式查询处理
- 分布式事务处理

分布式事务

```
Begin Transaction;
X=X+1;
Y=Y+1;
Commit Transaction;
```

这里是简写
实际上是update …

- 如果一个事务读写的数据分布在同一台机器上
- 怎么支持？ **就是普通的Transaction**

分布式事务

```
Begin Transaction;
X=X+1;
Y=Y+1;
Commit Transaction;
```

这里是简写
实际上是update …

- 如果一个事务读写的数据分布在不同机器上
- 怎么支持？

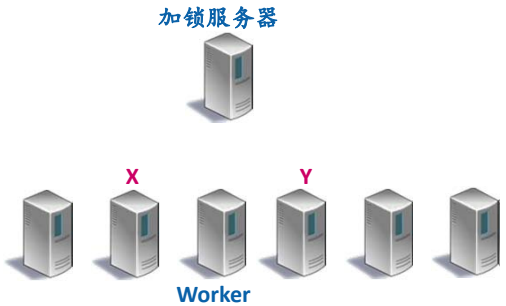
分布式事务

- 并发控制
- 崩溃恢复

分布式并发控制

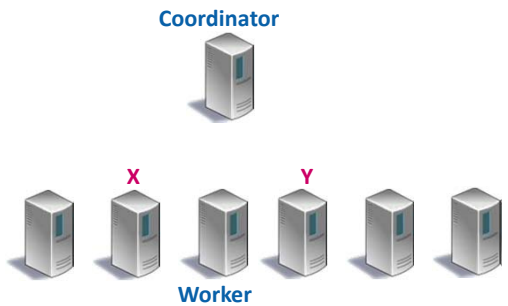
- 我们这里只考虑加锁的实现
- 集中式
  - 有一台机器提供集中式加锁服务
  - 所有加锁请求必须发送到这台机器集中处理
  - 问题：这台机器可能成为性能瓶颈
- 分布式
  - 每台机器分别对自己的数据维护锁
  - 复杂情况：有副本备份怎么办？
    - 一种方案：在主副本加锁，
      修改主副本时，必须同时修改其他副本

集中式



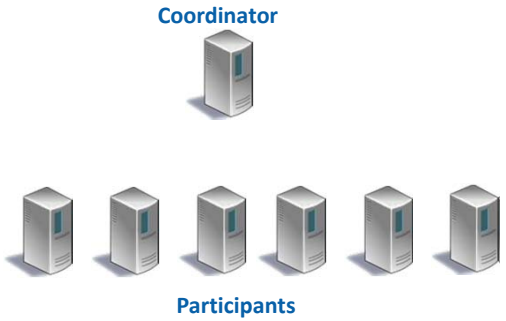
- 加锁服务器管理全局统一的锁空间
- 分布式事务采用2 phase locking进行并发控制
- 加锁服务器可能成为性能瓶颈

分布式



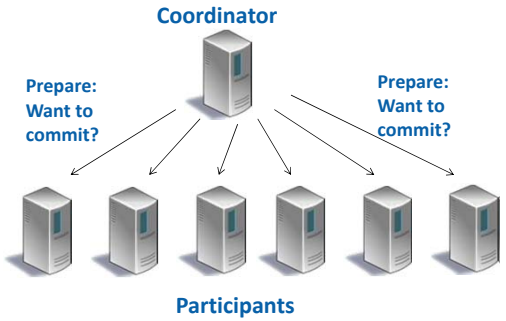
- 每个Worker分别处理事务相关内容，各自管理自己的锁
- 但是，是否能够commit需要协调

2 Phase Commit协议



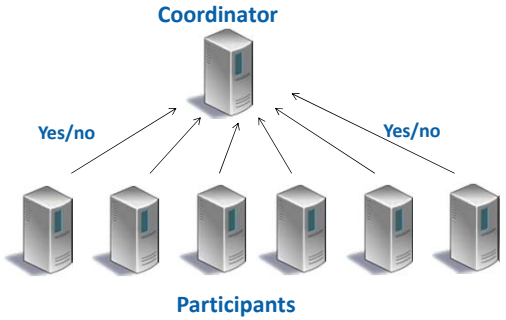
- Participant: 完成分布式事务的部分读写操作
- Coordinator: 协调分布式事务的进行

2 Phase Commit： phase 1 (voting)



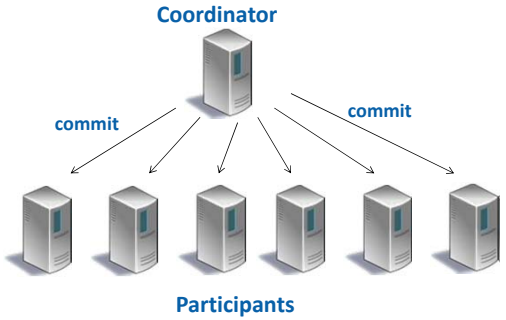
- Log记录prepare消息
- Coordinator向每个participant发送query to commit消息

2 Phase Commit： phase 1 (voting)



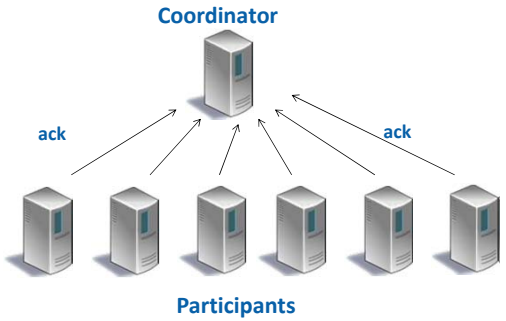
- Coordinator向每个participant发送query to commit消息
- 每个participant根据本地情况回答yes 或 no，并记录log preprare及回答

2 Phase Commit： phase 2 (completion)



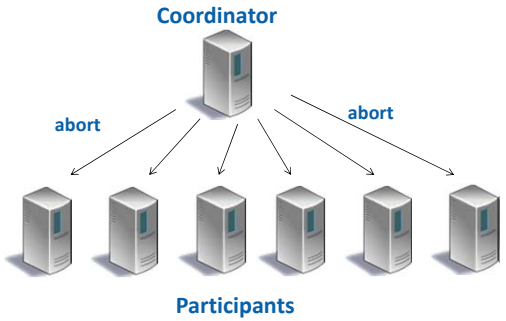
- 当所有的回答都是yes, transaction 将commit
- log记录commit消息
- Coordinator向每个participant发送commit消息

2 Phase Commit：phase 2 (completion)



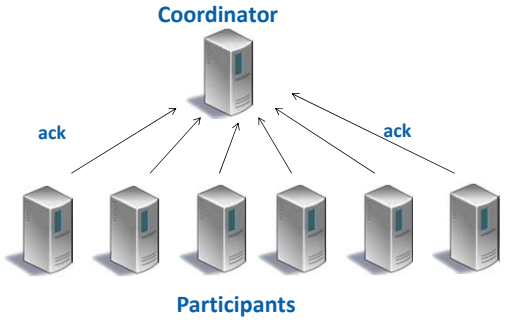
- Participant 回答acknowledgment，并记录log commit

2 Phase Commit：phase 2 (completion)



- 当至少一个的回答是no, transaction 将abort
- log记录abort消息
- Coordinator向每个participant发送abort消息

2 Phase Commit：phase 2 (completion)

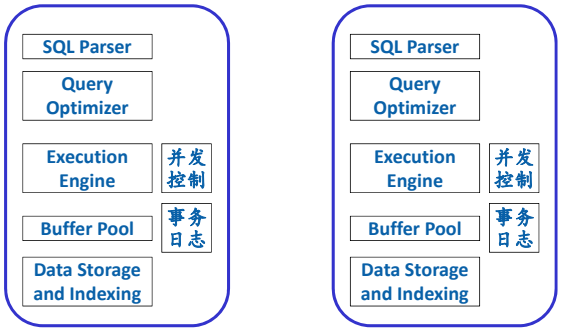


- Participant 回答acknowledgment，并记录log abort

崩溃恢复

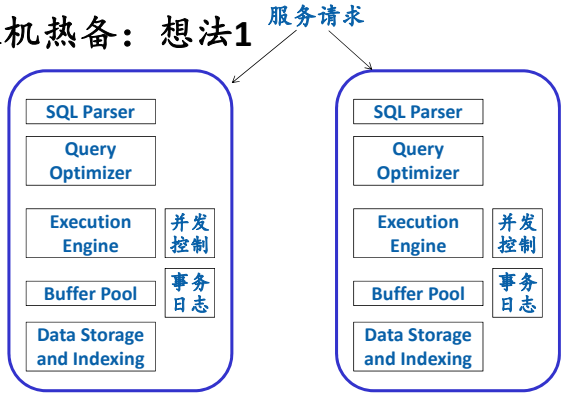
- 恢复时日志中可能有下述情况
  - 没有prepare/commit/abort：那么本地abort
    - 这是在voting前就宕机了
  - 有prepare，而没有commit/abort：那么分布式事务的处理结果未知，需要和prepare记录中的coordinator进行联系
    - 这是在voting后，completion前宕机
  - 有commit或abort记录：那么分布式事务处理结果已经收到，进行相应的本地commit或abort
    - 这是completion后宕机

双机热备



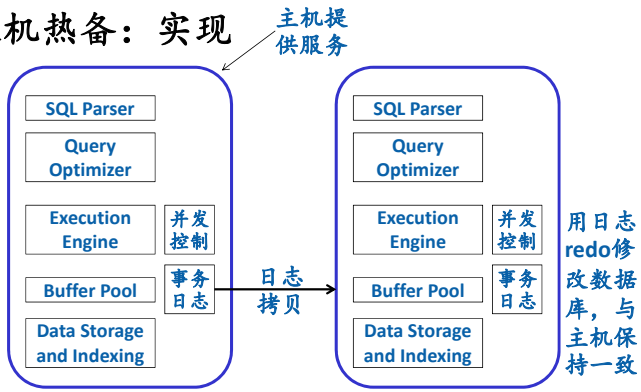
- 当主机(Primary)宕机，备机(Backup)就切换，继续提供服务
- 思路可以用于单机数据库和分布式数据库

双机热备：想法1



- 把服务请求同时发给两台机器，可以吗？
- 问题：每台机器上面多个事务并发控制的执行结果可能不同！

双机热备：实现



- 日志拷贝+redo，保证数据库内容一致

小结

- 系统架构和关键技术
- 分布式查询处理
- 分布式事务处理