

一个专注做 Data的朋友

胡怡文

经历

公司：支付宝，平安付

职业：DW->DBA->架构师

主要工作：支付宝数据仓库运维主管
负责支付宝RAC->Greenplum->Hadoop的平台转型

支付宝线上DBA，主要贡献是LDC架构DB部分实施
13年双11大促前完成架构部署，支持100%可用率

平安付高级架构师

技术圈：贡献过2本书
《Oracle性能诊断艺术》
《Oracle性能优化求生指南》

Postgres 全国用户会组委会成员，上海分会负责人

今天我想聊的是我经历过的那些：oltp和olap

OLTP：联机事务处理OLTP (on-line transaction processing)

OLAP：联机分析处理OLAP (On-Line Analytical Processing)

哪些是较为可靠的方案：

一、Oracle + RAC + 一体机 (可惜没用过)

二、Postgres (or mysql) + Greenplum (基于postgres)

重点聊聊postgres在2个领域的技术

OLTP

高可用
易开发
高并发

OLAP

高吞吐量
易扩展
易开发



高可用概念

百科：“高可用性”（High Availability）通常来描述一个系统经过专门的设计，从而减少停工时间，而保持其服务的高度可用性。

支持可用性的计算公式：

$$\%availability = (Total\ Elapsed\ Time - Sum\ of\ Inoperative\ Times) / Total\ Elapsed\ Time$$

elapsed time为operating time+downtime。

高可用衡量

业界常用 N 个9 来量化可用性，最常说的就是类似 “4个9(也就是99.99%)” 的可用性。

描述	通俗叫法	可用性级别	年度停机时间
基本可用性	2个9	0.99	87.6小时
较高可用性	3个9	0.999	8.8小时
具有故障自动恢复能力的可用性	4个9	0.9999	53分钟
极高可用性	5个9	0.99999	5分钟

PostgreSQL的高可用方案

- 一：老百姓
- 二：土豪金
- 三：高富帅
- 四：白富美
- 五：三体星

方案一：老百姓

*PG*异步流复制

数据，日志都放本地盘

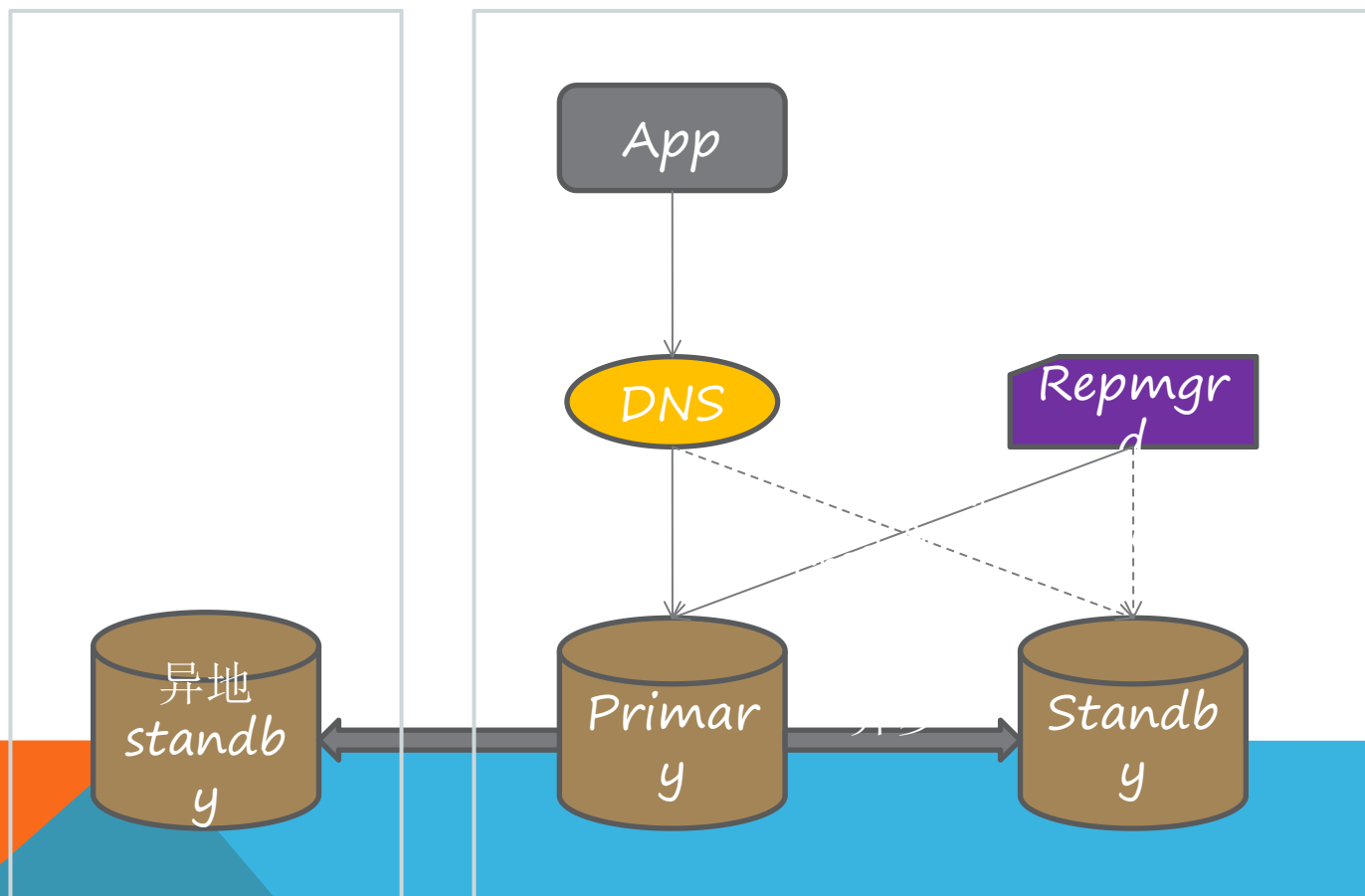
成本低，使用简单

有一致性的风险，可能要重搭备库

常用软件：*repmgr*



方案一架构



方案一：要点

配置文件`repmgr.conf`，主备分别配

建备份非常方便 `standby clone`

注册实例到`repmgr`的`schema`中，`schema`在`master register`时自动创建

检测进程常驻后台`repmgrd`，可手动或自动`failover`

主库正常关闭，将来可以直接作为备库

主库异常关闭，原备库和主库存在`lag`，则需要根据应用可接受数据丢失的等级来确定恢复方案：

A:应用可用优先，可接受少部分数据丢失，备库直接切

B:对数据完整性要求高，则要先恢复主库

方案二：土豪金

*PG*同步流复制

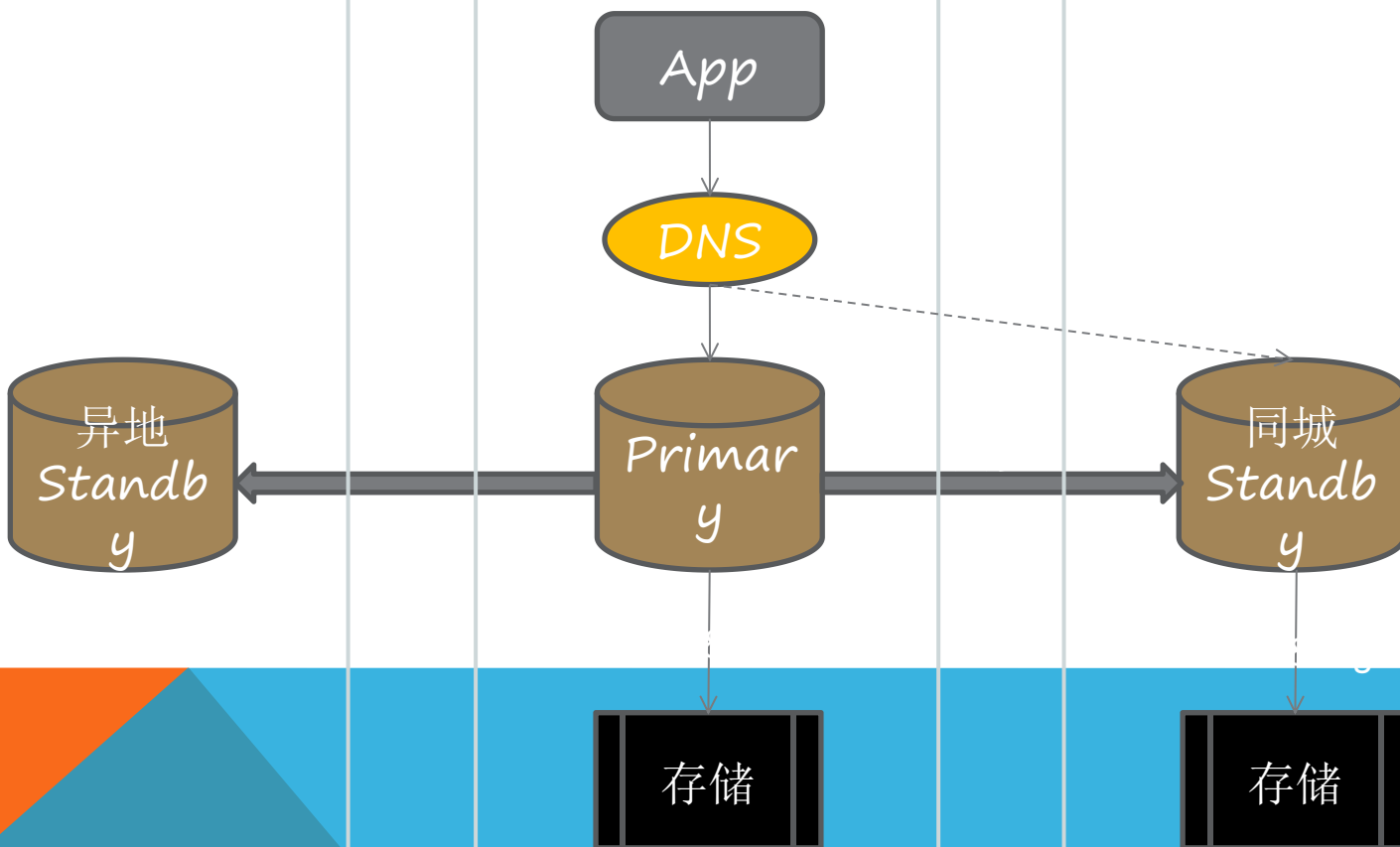
数据，日志都放在存储上

满足强一致性和高可靠性

性能有所下降，价格高



方案二架构



方案二：要点

要有设备：SAN交换机和存储

Postgresql同步复制

a、设置 `postgresql.conf` (on primary)

`synchronous_standby_names = 'pg_sync_1'`

b、配置 `recovery.conf` (On standby)

`primary_conninfo = 'host=192.168.1.1 port=5432
user=repuser application_name=pg_sync_1'`

方案三：高富帅

*PG*异步流复制

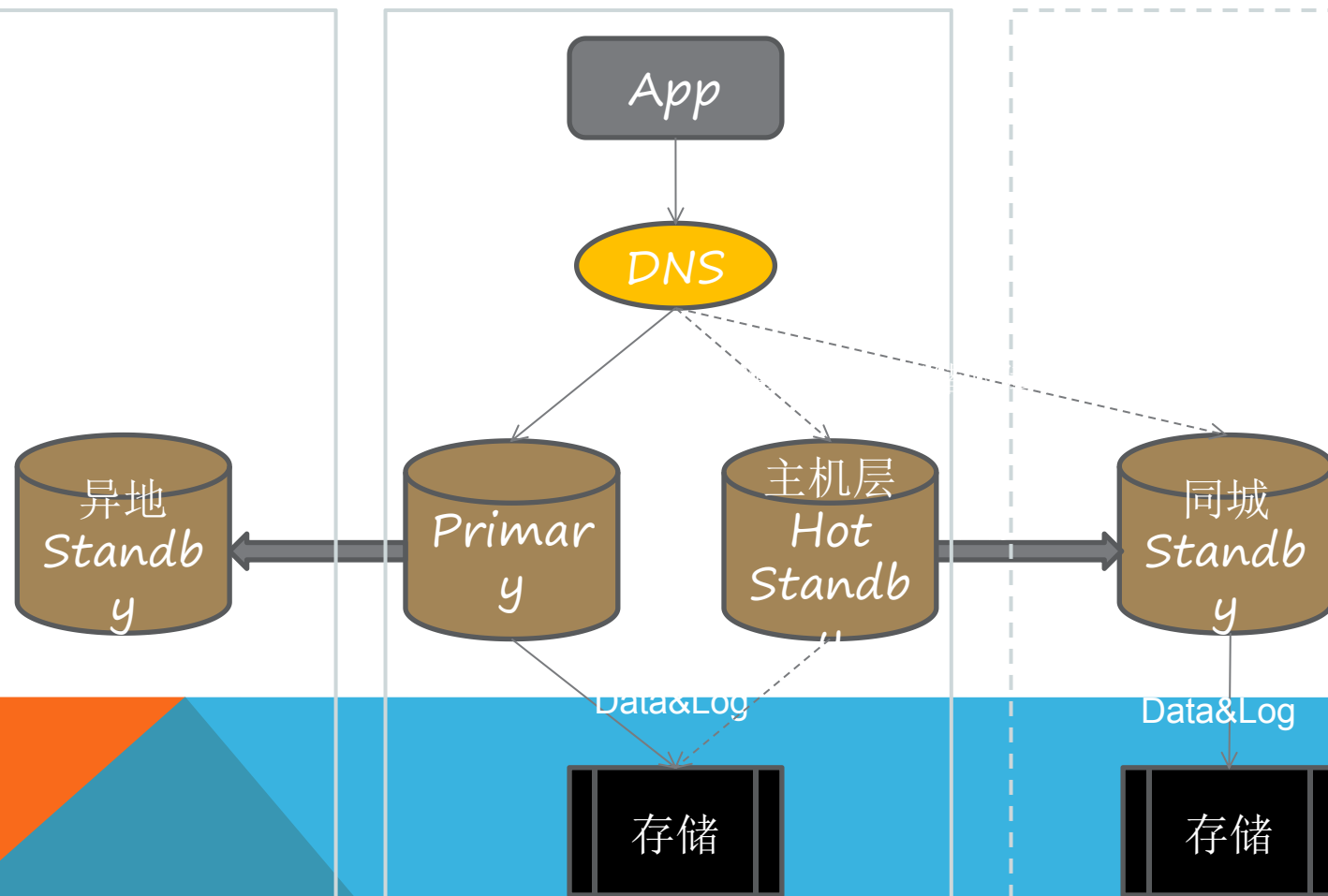
数据，日志都放在存储上

*VCS*集群，冗余一台主机

本地机房冗余，异地可视需求而定



方案三架构



方案三：要点

无他，VCS尔



方案四：白富美

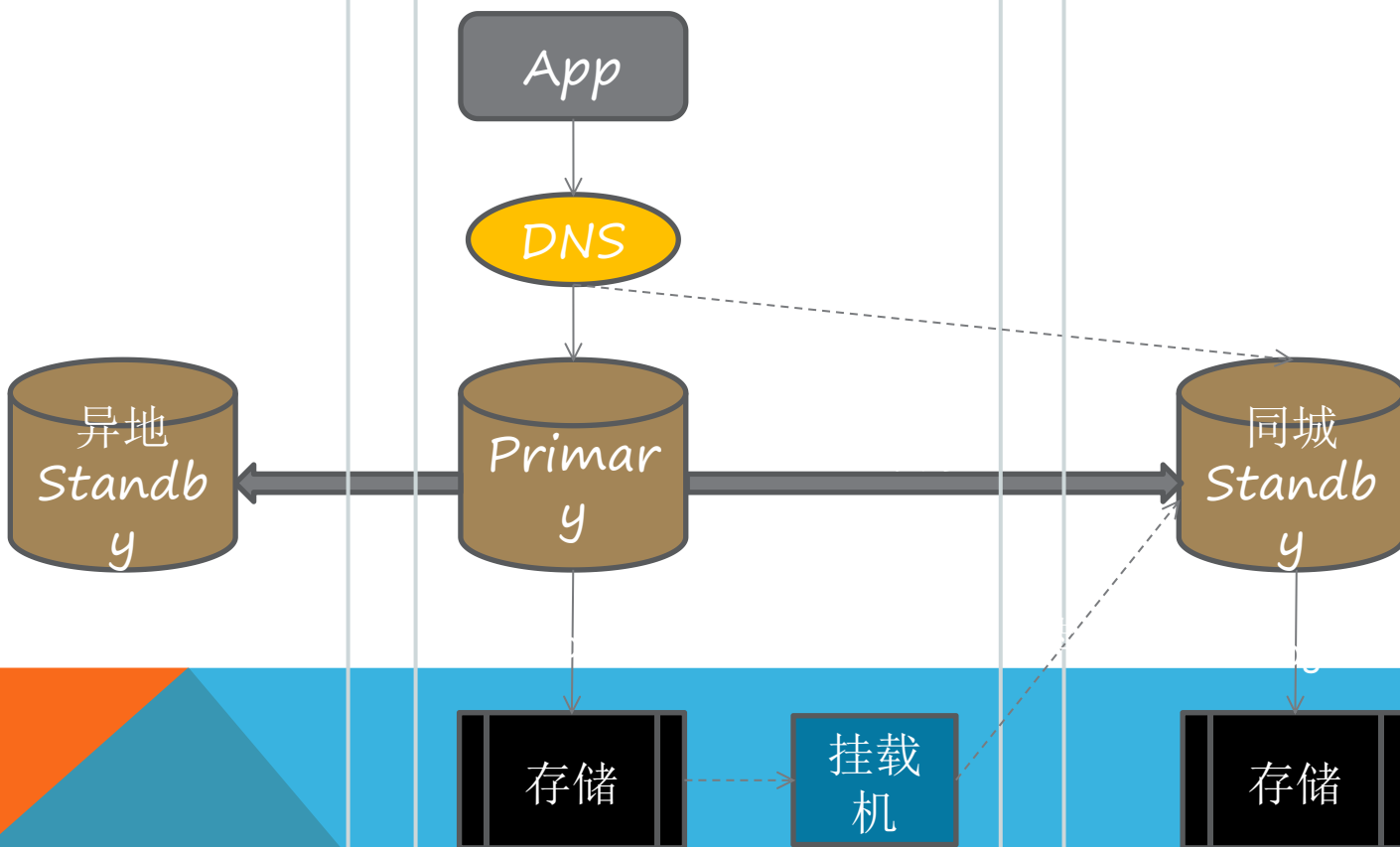
*PG*异步流复制

数据放在本地盘

日志放在存储，同时把存储挂载到另一台备机



方案四架构



方案四：要点

挂载机：只能读哦

数据可以放本地盘，但最好是高速*ssd*

可用于本机房或异地机房主备环境

恢复花时间稍长，但保障一致性，适应性强，性价比高



方案五：三体星

*Pg-X2*是未来科技进步的方向

集群化

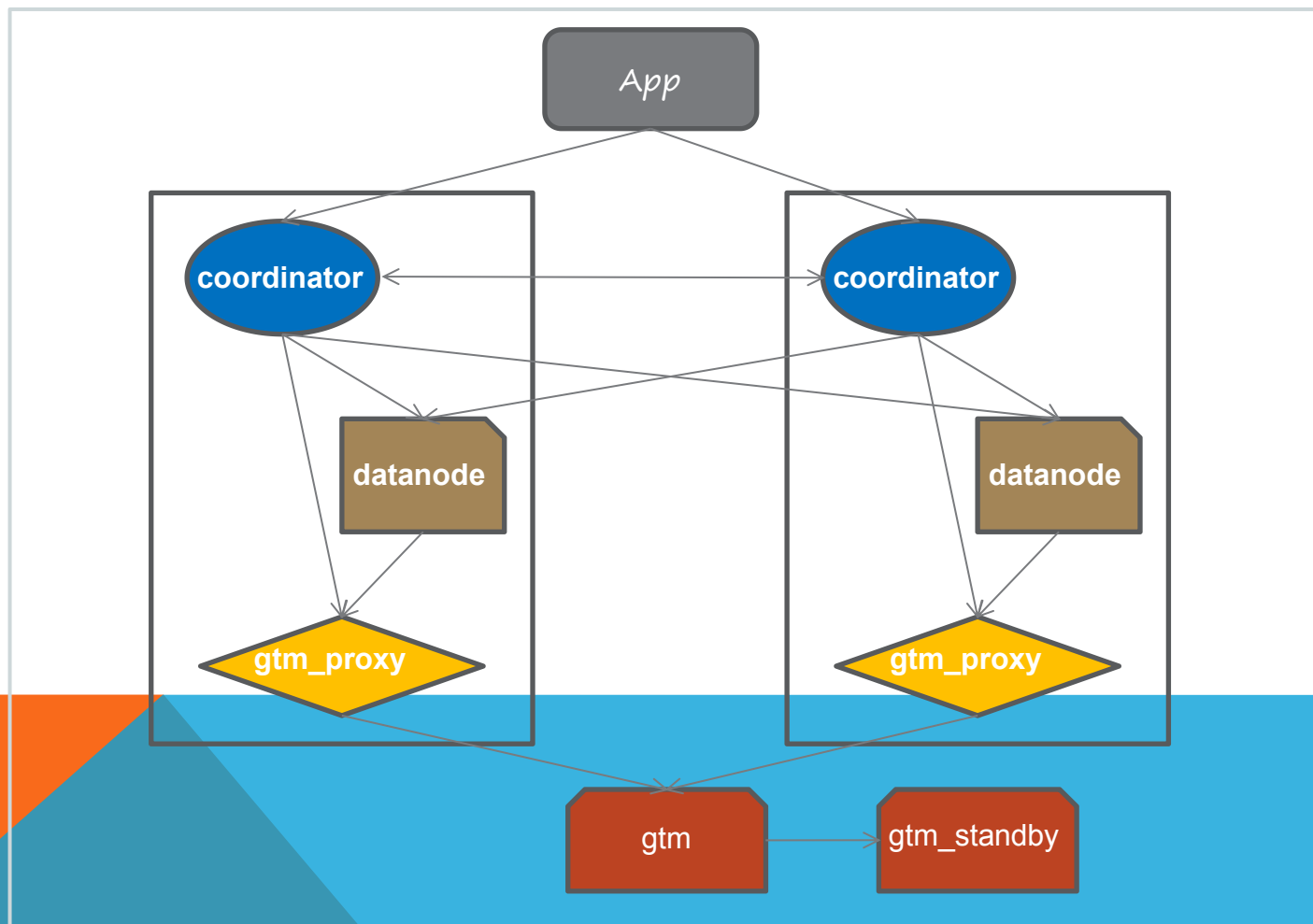
保障强一致性，可横向扩展

高可用，无单点故障

目前还未完全成熟，集群备份方案待完善



方案五架构



方案五：要点

Replication or Distribution

Replication: 每个数据节点都有完整的表数据

Distribution: 即每个数据节点仅保留表的部分数据

Write - scalable PostgreSQL cluster

水平的扩展写能力

Synchronous multi - master configuration

在任意master的操作立即为其它节点可见

Table location transparent

应用无需更改，在外界看来事务处理没有变化

本身还没有大型成熟的应用

在高可用上还有很多事情要做

备库难做

易开发

sql支持标准语法，对Oracle兼容性强

全面支持集合操作

全面支持windows函数操作

DDL变更不长时间锁表，不移动数据

查询解析器功能强大，经典基于cost的优化器

索引类型全面

支持多种过程化语言

支持trigger，函数等，自定义开发便捷

社区贡献丰富

高并发

postgresql和Oracle是进程模式，mysql是线程模式

进程模式对CPU利用率比较高

进程模式共享数据需要通过共享内存，线程模式本身在进程空间中就是共享的，只需要控制好线程间的同步

postgresql在连接创建上吃点亏。但postgresql有出色的连接池软件：

比如pgbouncer

通过pgbouncer可以很好得管理连接的创建和使用



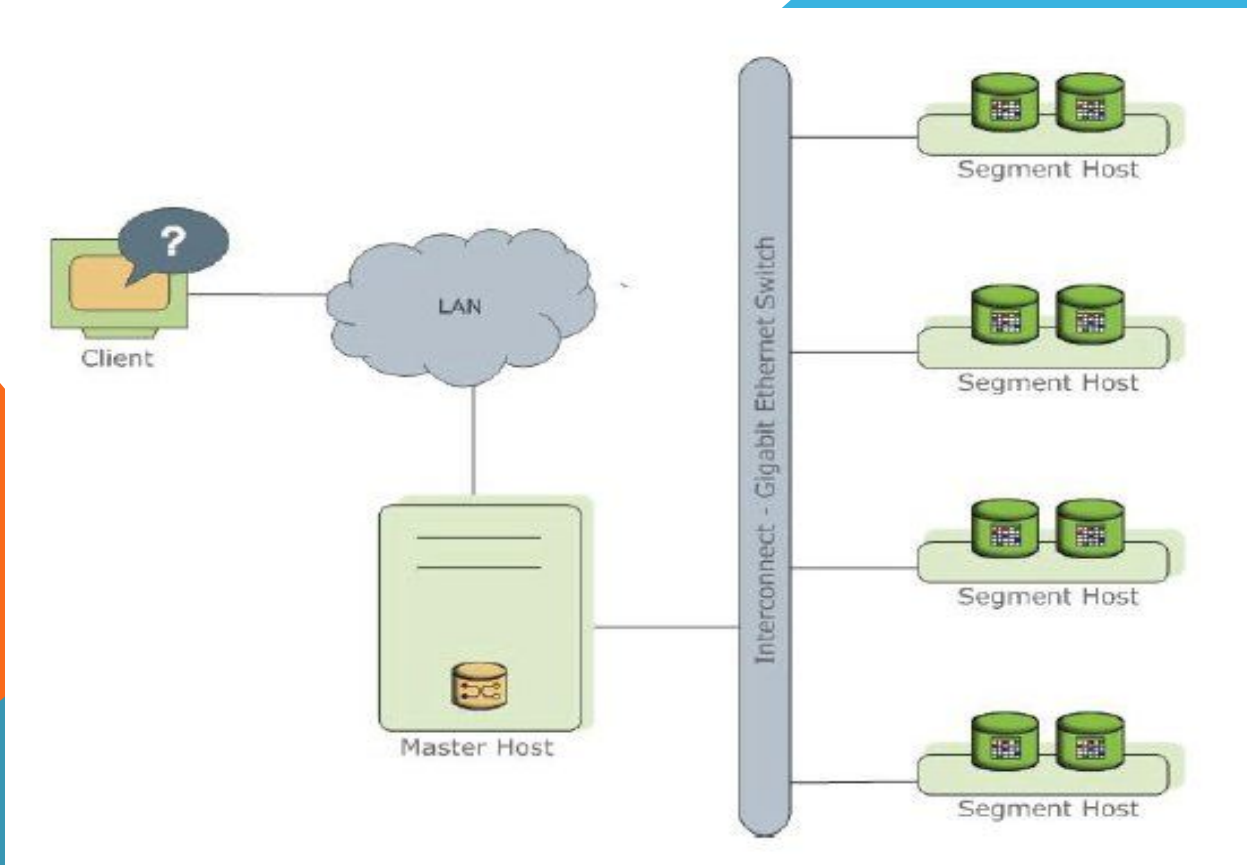
分布式数据库Greenplum的简介

*Greenplum*是基于开源数据库*PostgreSQL*的并行处理结构的数据库系统，
采用*shared nothing*的架构，每个实例有自己的内存和进程。

*Greenplum*针对数据仓库技术对*PostgreSQL*的内核进行了优化，比如系统数据字典，查询计划，优化器，查询执行器和事务管理组件等。
使多个*PostgreSQL*进程能够并发处理，并逻辑上成为一个实例。

Greenplum的体系结构

Greenplum由一个master和多个segment实例构成，Master是Greenplum系统入口，负责和client交互。客户端只和master建立连接并提交SQL。Master负责在各个segment间协调工作。而segment则具体存储并处理数据操作。以下为Greenplum结构图



Master , Segment及Interconnect

Master是Greenplum的入口，也是一个典型的PostgreSQL数据库实例，所以用户可以把它当作一个PostgreSQL实例来连接，可以使用命令行工具psql，或JDBC，ODBC。Master主要存储了Greenplum的全局数据字典（包含Greenplum自身元数据的系统表），它不存储任何用户数据，此外Master的主要工作还有，验证到各segment的连接，接受client的SQL命令，给各segment安排工作，协调各点返回的信息，并给客户展现最终结果。

Master , Segment及Interconnect

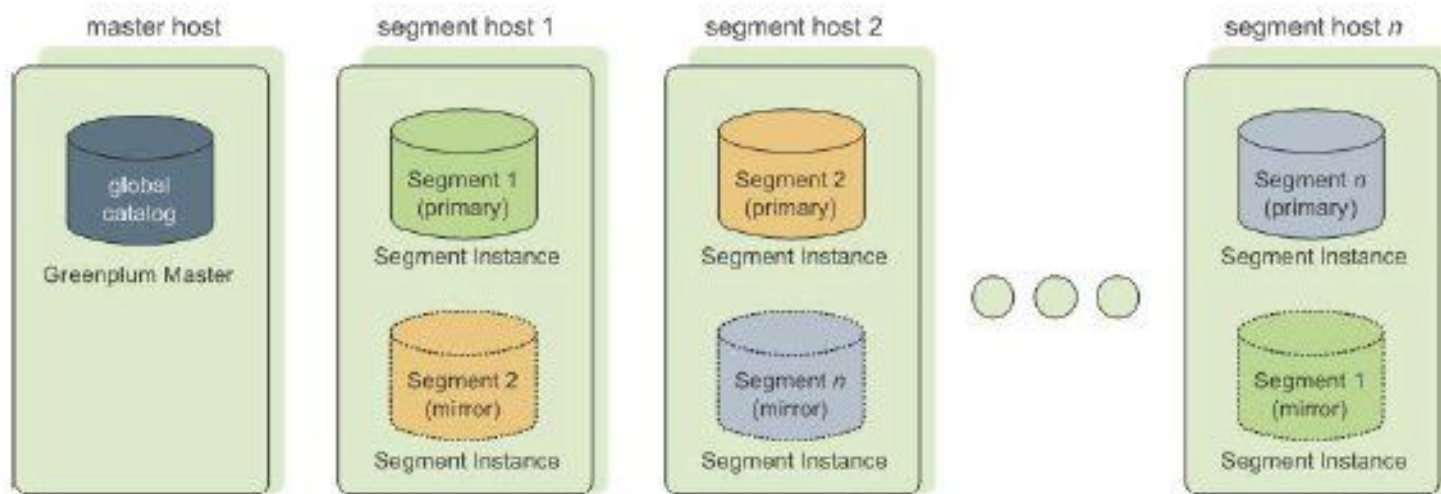
在Greenplum系统中，segment负责存储数据，而且是大多数SQL执行的地方。用户定义的表和索引等对象都散列在各个segment结点。用户不直接和segment交互。这里注意不要混淆Greenplum segment和segment host。segment host是指一台机器，上面可以运行多个Greenplum segment，即PostgreSQL数据库实例。

Master , Segment及Interconnect

当用户连接并提交一个查询后，进程在各个Greenplum segment创建，Interconnect是指各segment间的内部进程通讯。这还要依赖网络基础设施，一般是标准的千兆以太网交换机。默认，内部通讯使用UDP协议，由Greenplum软件而不是UDP负责额外的包校验，这样可以提供和TCP一样的可靠性，并且性能和容量还将超过TCP。因为如果使用TCP协议传输，Greenplum限制最大1000个segment实例

Greenplum的冗余和失败切换

Greenplum可以通过配置部署来避免单点失败。我们可以配置**mirror segments**。当某个**segment**出现问题时可以立即切换到备份上（也就是**mirror**上）。当然配置**mirror**，需要花费一定的磁盘空间。同时**segment**和它的**mirror**总是在不同的**host**上。下图显示了数据是如何在集群间分布的。

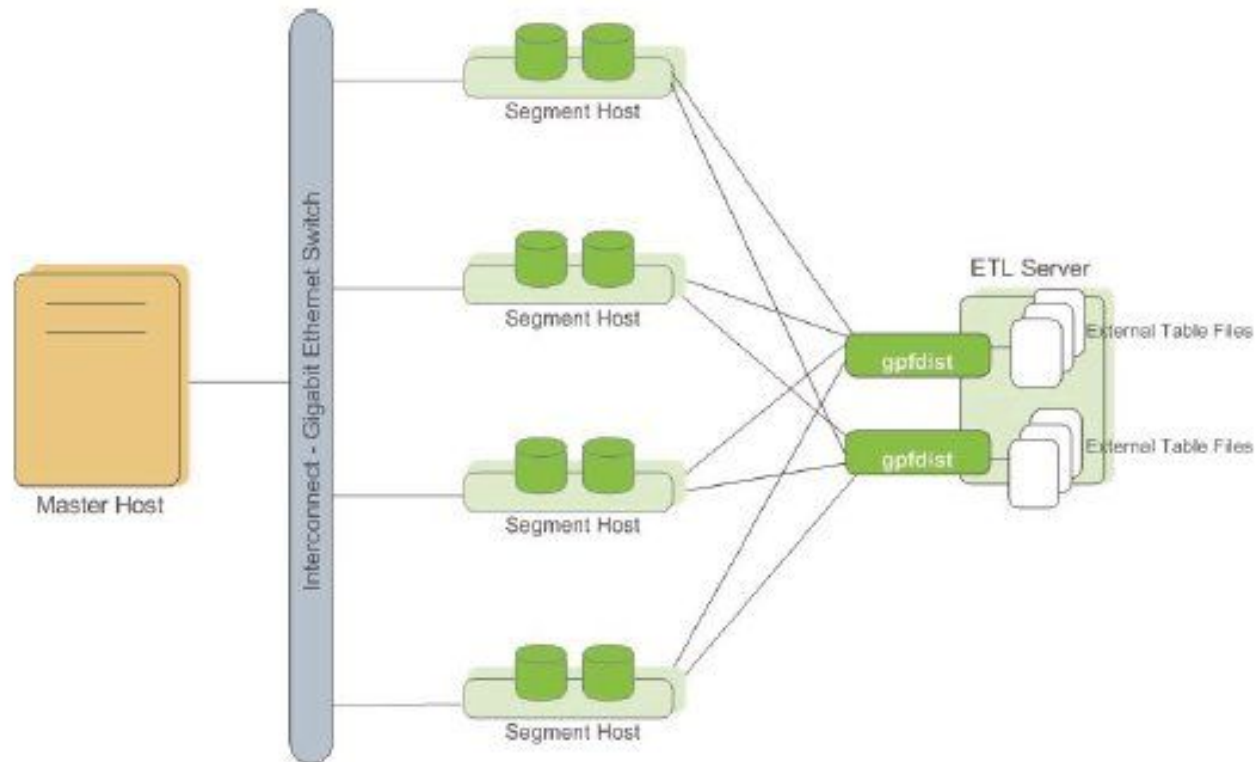


Greenplum的冗余和失败切换

除了segment, Master也可以选择配置一个backup, 或称为standby。Master和standby间通过事务日志复制进程完成同步。具体来说, 在standby上有一个事务日志复制进程, 如果master失败, 此进程关闭, 同时激活standby。因为master上不存储任何用户数据, 只有系统数据字典需要同步, 这些字典表不经常更新, 但一旦发生变化, 就会立即同步到standby

此外, 为了高可用性的Greenplum, 也可以选择配置一台冗余的千兆以太网交换机, 在各结点间形成一个冗余的千兆连接。

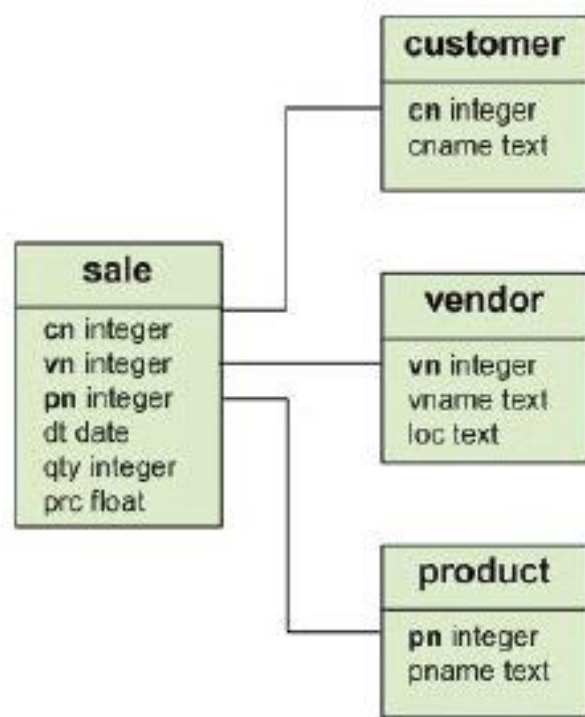
性能关键之并行数据导入



在大型数据仓库系统实施中会遇到的一个挑战是，在有限的时间窗口内，高效地导入海量的数据。**Greenplum**可以使用它的外部表特性支持快速，并行的数据导入。外部表可以以“单行失败隔离”模式工作，即允许管理员过滤部分格式错误的记录到一张单独的错误表，同时继续导入正确格式化的数据。管理员可以配置允许错误的记录数目。**Greenplum**还提供了并行文件服务工具（**gpfdist**）和外部表配合使用，以达到最佳的并行性能和load带宽。此图显示了**gpfdist**和**external table**的作用方式。

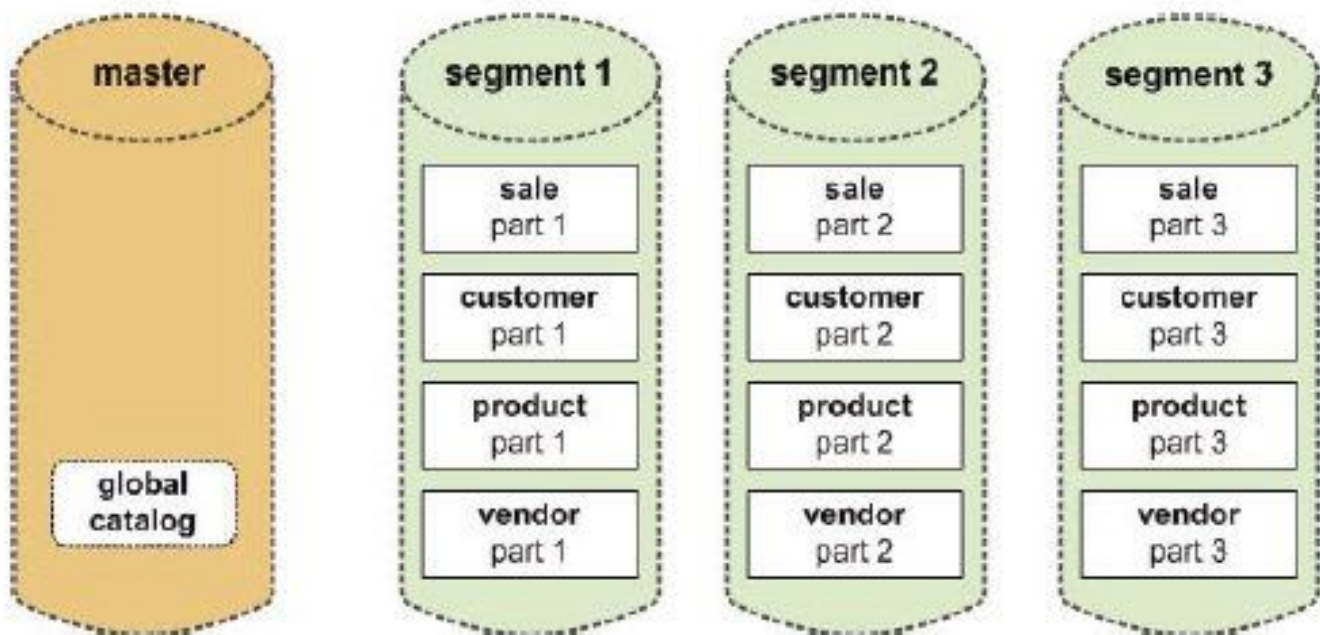
性能关键之并行数据存储

Greenplum是一个分布式数据库系统，这意味着数据物理的存储在多台数据库服务器上。为了理解Greenplum是如何在各结点间组织数据的，用下面这个典型的表结构来说明：



性能关键之并行数据存储

每张表都是不重复不遗漏的分布进多个segment。分布是通过一个精密的hash算法来实现的。数据库管理员在定义表的时候选择hash key（一个或多个字段）。Master只存全局数据字典，segment各存数据的一部分。如下图所示



性能关键之并行数据存储

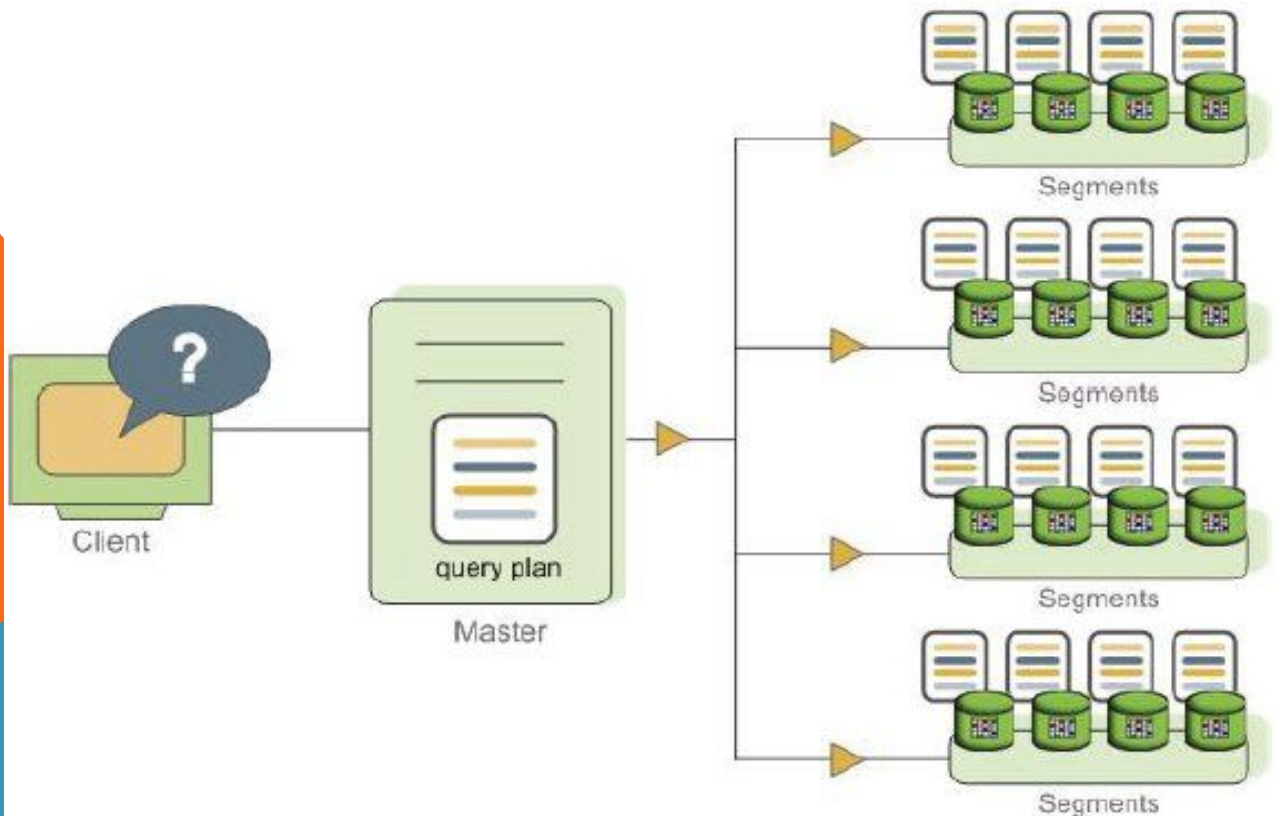
下面具体介绍下Greenplum的分布策略，这对性能有很大的影响：

Hash分布：选择一个或多个字段作为distribution key。distribution key被hash算法用来分配每一条记录到一个特定的segment，有相同key的记录被散列到相同的segment。选择一个唯一键，比如主键，将保证数据被尽可能打散。Hash分布是Greenplum的默认分布策略，如果没有应用DISTRIBUTED子句（Greenplum特有的语法，用来指定distribution key），则用表的主键（如果存在）或表的第一个字段作为distribution key。

Random分布：随机分布，采用循环的方式分散数据到各个结点，有相同字段值的记录不一定被散到同一个segment。虽然随机分布能让数据完全分散，但hash分布在性能上的优势是无可争议的。

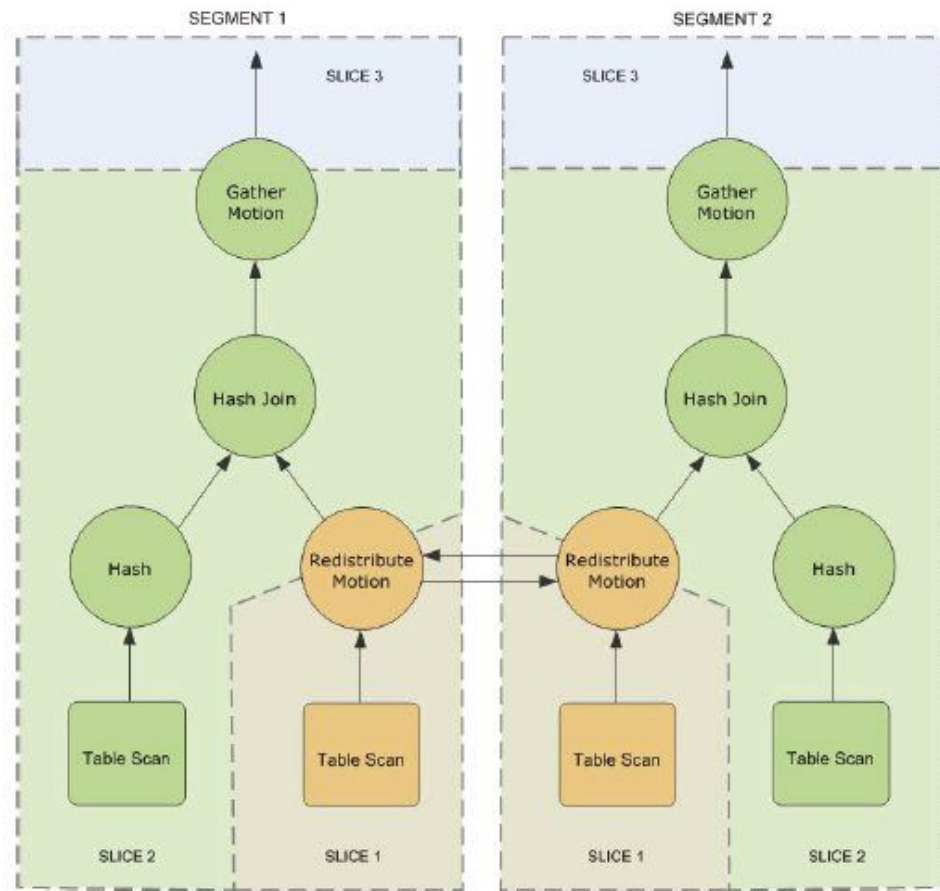
性能关键之并行操作

虽然是集群系统，但对最终用户来说，我们只需对一个DBMS操作。连上master，提交SQL。Master经过解析，优化，创建并行执行计划，然后把执行计划分配到所有segment。每个segment只负责本地数据操作。所有数据库操作，比如，table scans, joins, aggregations, and sorts。在各segment同时运行，下图显示了此过程



性能关键之并行操作

执行计划是数据库为了完成任务所要进行的一系列操作的集合，任何一个step都代表一个数据库操作。对于典型的数据库操作（如table scans, joins等），Greenplum有一个额外的操作叫motion。Motion包含在segment间移动元组记录。为了达到最大的并行，Greenplum把操作切片（slices）。一个切片是执行计划的一部分，可以在segment-level独立执行。如下图所示：



执行计划

greenplum的查询优化器

查询计划包括了一些传统的操作，比如：扫描、Join、排序、聚合等等。

greenplum中有三种数据的移动操作：


Broadcast Motion (N:N) ,即广播数据，每个节点向其他节点广播需要发送的数据。

Redistribute Motion (N:N) ，重新分布数据，利用join的列值hash不同，将筛选后的数据在其他segment重新分布。

Gather Motion (N:1)，聚合汇总数据，每个节点将join后的数据发到一个单节点上，通常是发到主节点master。

执行计划

简单示例

Sql代码 

```
1.  explain select d.*,j.customer_id from data d join  jd1 j on d.partner_id=j.partner_id wher
2.                                     QUERY PLAN
3. -----
4.  Gather Motion 88:1 (slice2) (cost=3.01..939.49 rows=2717 width=59)
5.    -> Hash Join (cost=3.01..939.49 rows=2717 width=59)
6.        Hash Cond: d.partner_id::text = j.partner_id::text
7.        -> Seq Scan on data d (cost=0.00..260.74 rows=20374 width=50)
8.        -> Hash (cost=1.91..1.91 rows=88 width=26)
9.            -> Broadcast Motion 88:88 (slice1) (cost=0.00..1.91 rows=88 width=26)
10.                -> Seq Scan on jd1 j (cost=0.00..1.02 rows=1 width=26)
11.                    Filter: gmt_modified > ('now'::text::date - 80)
```

执行计划

它做了什么？

执行计划执行从下至上：

- a, 在各个节点扫描自己的 jd1 表数据，按照条件过滤生成数据 rs
- b, 各节点将自己生成的 rs 依次发送到其他节点。（ Broadcast Motion (N:N), 即广播数据）
- c, 每个节点上的 data 表的数据，和各自节点上收到的 rs 进行 join。这样就保证本机上的数据只和本机的数据 join。
- d, 各节点将 join 后的结果发送给 master（ Gather Motion (N:1) ）

由上面的执行过程可以看出，Greenplum 是将 rs 给每个含有 data 表数据的节点都发了一份的。

要是 RS 很大或者压根就没有过滤条件怎么办呢：

执行计划

原来如此

Sql代码 

```
1. => select count(*) from jdl;  
2.   count  
3.   -----  
4.         20  
5.   (1 row)
```

Sql代码 

```
1. => select count(*) from data;  
2.   count  
3.   -----  
4.  113367
```

要是 rs 很大的话，广播数据 网络就会成为瓶颈。可以看出 greenplum 很聪明：

它是将小表广播到各个 segment 上。可以看出统计信息对于生成好的查询计划是何等重要。

执行计划

可以优化吗？

```
dw=# explain select e.* from emp e where e.deptno in( select deptno from dept2);  
QUERY PLAN
```

```
Gather Motion 12:1 (slice2; segments: 12) (cost=234.97..817.62 rows=4779 width=51)  
-> Hash Join (cost=234.97..817.62 rows=4779 width=51)  
    Hash Cond: e.deptno = dept2.deptno  
        -> Seq Scan on emp e (cost=0.00..9.14 rows=2 width=51)  
        -> Hash (cost=234.92..234.92 rows=1 width=7)  
            -> Broadcast Motion 12:12 (slice1; segments: 12) (cost=234.80..234.92 rows=1 width=7)  
                -> HashAggregate (cost=234.80..234.84 rows=1 width=7)  
                    Group By: dept2.deptno  
                    -> Seq Scan on dept2 (cost=0.00..193.84 rows=1366 width=7)
```

--注意这一步

(9 rows)

执行计划

改个写法试试！

```
dw=# explain select e.* from emp e join dept2 d on d.deptno=e.deptno;  
QUERY PLAN
```

Gather Motion 12:1 (slice2; segments: 12) (cost=9.60..961.20 rows=4779 width=51)

-> Hash Join (cost=9.60..961.20 rows=4779 width=51)

Hash Cond: d.deptno = e.deptno

-> Seq Scan on dept2 d (cost=0.00..193.84 rows=1366 width=7)

-> Hash (cost=9.42..9.42 rows=2 width=51)

-> Redistribute Motion 12:12 (slice1; segments: 12) (cost=0.00..9.42 rows=2 w

Hash Key: e.deptno

-> Seq Scan on emp e (cost=0.00..9.14 rows=2 width=51)

(8 rows)

执行计划

再看一个例子

```
dw=# explain select distinct (deptno) from dept2;  
QUERY PLAN
```

```
Gather Motion 12:1 (slice1; segments: 12) (cost=1340.72..1422.64 rows=1 width=7)  
Merge Key: deptno  
-> Unique (cost=1340.72..1422.64 rows=1 width=7)  
Group By: deptno  
-> Sort (cost=1340.72..1381.68 rows=1366 width=7)  
Sort Key (Distinct): deptno  
-> Seq Scan on dept2 (cost=0.00..193.84 rows=1366 width=7)  
(7 rows)
```

```
dw=# explain select deptno from dept2 group by deptno;  
QUERY PLAN
```

```
Gather Motion 12:1 (slice1; segments: 12) (cost=234.80..234.84 rows=1 width=7)  
-> HashAggregate (cost=234.80..234.84 rows=1 width=7)  
Group By: deptno  
-> Seq Scan on dept2 (cost=0.00..193.84 rows=1366 width=7)  
(4 rows)
```

测试环境

机器: Master 1台 (172.17.208.190)

Segment hosts 8台 (172.17.208.191-198)

CPU: Master为2颗物理cpu, 每个CPU 2core

Segment host为2颗物理cpu, 每个CPU 4core

内存: Master 4G RAM

Segment host 16G RAM

硬盘: Master 1块145G

Segment host 172.17.208.191-195 4块145G盘

172.17.208.196-198 3块 145G盘

OS: Red Hat Enterprise Linux AS release 4 (Nahant Update 6)

File system: ext3

安装时需要的一些软件, 在装机时已经安装。**Greenplum**要求**segment host**机器和配置要相同, 因为当数据基本均匀散列到各台机器后, 最差的那台机器的完成时间即是任务的最终完成时间, 不能让个别机器的问题影响了集群的性能。**Master**的配置稍差一点是可以的, 因为**master**不存储和处理用户数据, 它主要存储**global**数据字典信息, 以上的配置能够满足测试需求。

测试应用

本次主要测试了日志里的一块应用。每天日志大约70G-80G。由perl或hadoop格式化后，约有35G左右。

用Greenplum提供的工具gpfdist提供文件服务，让集群能够从文件所在的机器读数据，同时并行导入数据库中。以下分别用1-4个gpfdist进程测试外部表导入效率。

文件大小 记录条数		Gpfdist进程数	文件服务器读	导入时间
8.4G	34794391	1个	103 (MB/s)	98s
8.3G	35381777	2个	114 (MB/s)	88s
8.3G	34263222	3个	107 (MB/s)	94s
9.2G	38500757	4个	99 (MB/s)	112s

结论：根据上面的测试，增加gpfdist进程的数目，并不能加快数据的导入，因为瓶颈在网卡的出口速度。

建议：准备多台大内存文件服务器，每台机器配备多块网卡，将文件分散成多份导入文件服务器，每台机器起2-3个gpfdist服务进程。以供集群用外部表的方式从文件服务器导入数据。

测试应用

在仓库日志的具体应用中，有一项步骤如下：

- 1.从日志服务器导入数据到文件服务器并格式化。
- 2.回补没有userid的log记录。
- 3.导入到数据库
- 4.查找访问特定页面的userid。
- 5.查找这批userid在一天内访问过的所有页面。

现在将从第二步起全部放到Greenplum中处理，和原来做比较。

操作	Hadoop完成	Oracle完成	Greenplum完成
导入数据并格式化	40分钟		
回补没有userid的log记录。	1小时15分钟		7分钟
导入到数据库		8分钟	6.5分钟
查找访问特定页面的userid		4小时15分钟	2.5分钟
查找这批userid在一天内访问过的所有页面		45分钟	0.5分钟

测试应用

原先用hadoop+oracle的方式得到一天的数据要7个小时，现在只需要1个小时，其中用hadoop导入数据并格式化花了40分钟，Greenplum只用了不到20分钟就处理了原来6个小时的工作。可见Greenplum的处理效率远远高于原先。虽然过程中有一点cache的影响，但这本来就是流程中所应该有的，上一步处理完，一定会有部分数据被cache。而且就算把数据块重新读一遍，Greenplum每台机也只需要读 (M/N) 的数据（ M 为总数据量， N 为segment host的数目），读写速度仍然会大大提高。

不仅在I/O上Greenplum有优势，在做大数据量的表join时，Greenplum的性能会格外出众，主要因为其按hash key散列表的原理，可将有相同join key的记录放在一台机器上，这样本来大小为 $M1$ 和 $M2$ 的大表做join，就变成 S 个大小为 $M1/S$ 和 $M2/S$ 的小表同时做join（ S 为Greenplum segment的数目），因为是并行处理，所以只相当于其中一对小表join的时间。效率提高是理所当然的。

交流

微信: 18958088285



Thank
You