DM7 & MPP架构

——同时满足OLAP与OLTP需求

韩朱忠

提纲

- DM7的发展历程
- DM7核心技术
- DM7大规模并行处理(MPP)
- 性能与应用场景

一、DM7的发展历程

达梦数据库发展历程

- 实验室原型
- 技术积累阶段
- 实现各类标准
- 1988年研制我国第一个自主版权数据库CRDS

1 DM1-DM3

1984-2003

- 持续的技术积累
- 5.6引入物理操作符,虚拟机
- 6.0实现了众多企业级高级特性和oracle 兼容特性

2 DM4

2004

3 DM5.6

2007

4 DM6

2009

5 DM7

2012

- 对DM4-DM6的技术总结
- 融合列存储与行存储
- MPP高可扩展架构
- 基于向量数据的执行内核
- RAC集群
- 原生的MVCC
- OLAP应用的支持

设计目标之一: Oracle 兼容

- PL/SQL几乎所有特性
  - package,
  - 自定义类型，
  - 记录、数组、嵌套表
  - 索引表、静态、动态、引用游标
  - 自治事务、异常处理
  - 大量的测例与验证
- 类似的设计架构
  - 用户，表空间，模式
  - 回滚段，事务、闪回
  - 重做日志
  - 多版本并发控制机制

设计目标之二: 高性能OLTP支持

- 传统的交易型应用
- 和数据分析相比，应用更广泛
- 现有达梦的应用多属OLTP
- DM7擅长高并发OLTP

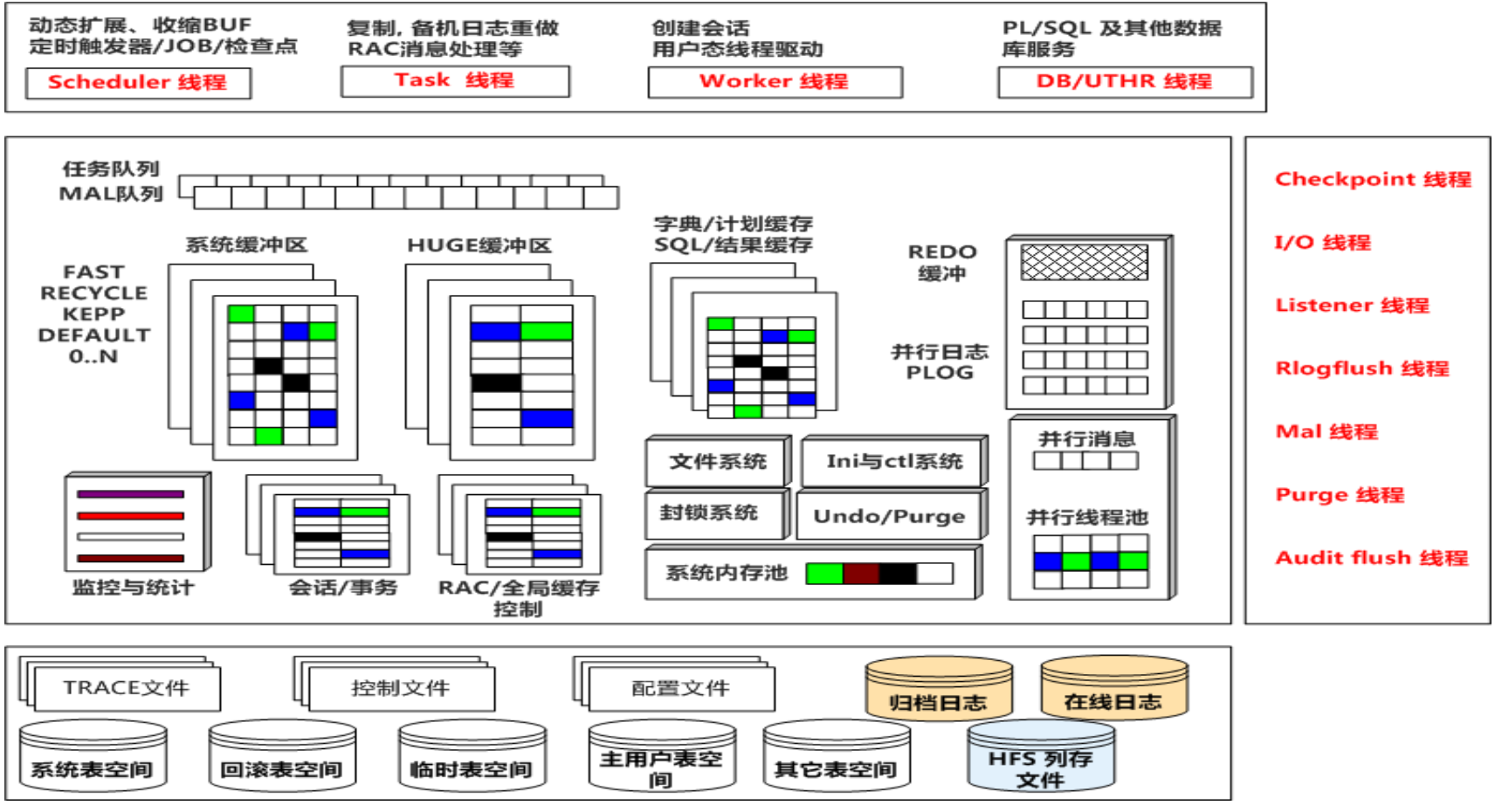
- 挑战Oracle在OLTP应用领域的霸主地位
- 逐步蚕食它的市场
- 目前所及的应用领域，性能不逊于Oracle
- 满足常规的OLTP应用

设计目标之三: 高性能数据分析OLAP

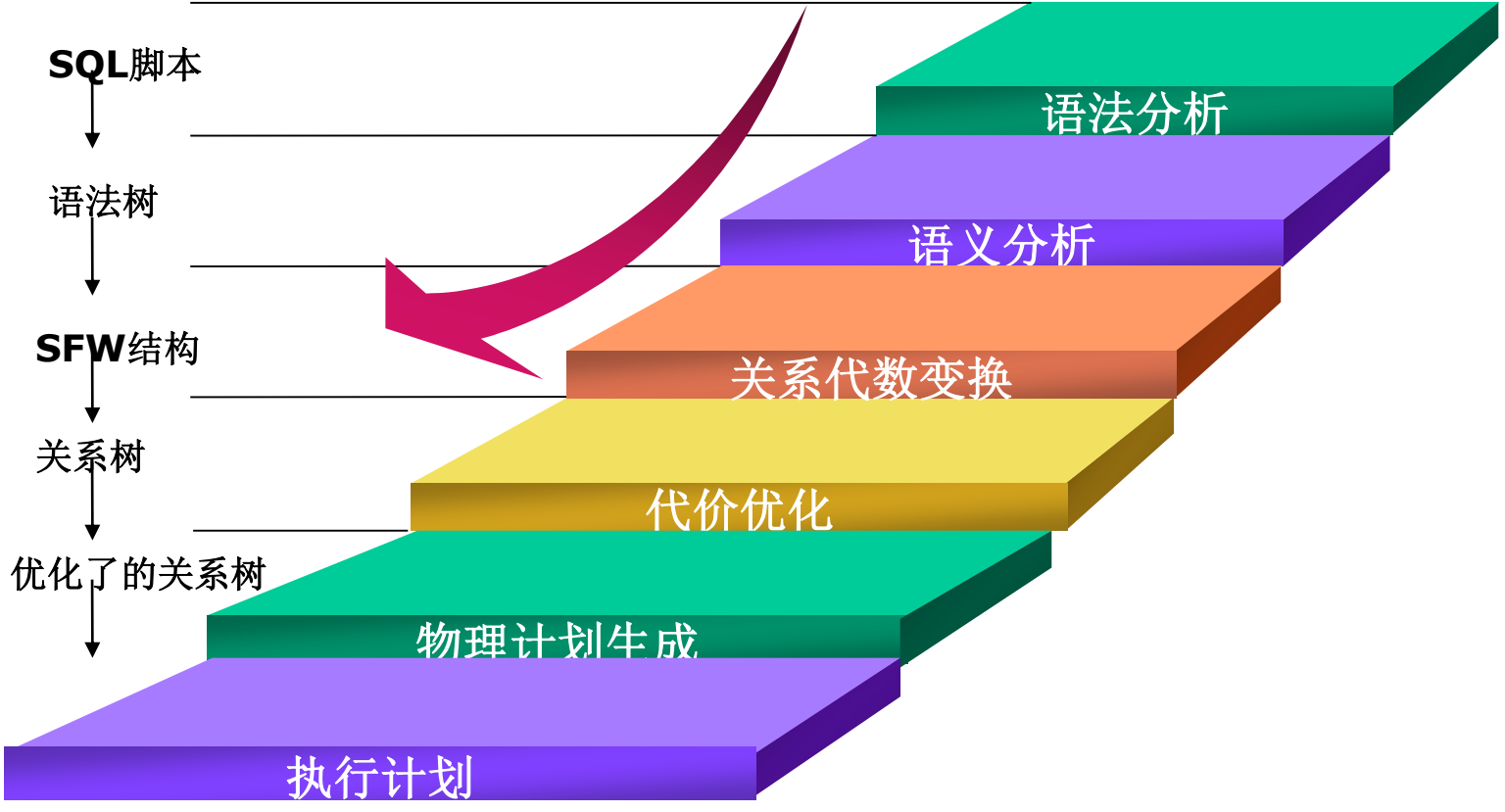
- 重新构思达梦七的源动力
- 为数据分析设计的
  - 批量处理
  - 列存储
  - 并行查询
  - 分区表
  - 分析函数
  - 压缩、物化视图等
- 大规模并行计算MPP
- 大数据
- 竞争产品
  - Geenplum
  - Teradata
  - MySQL Infobright
  - HP Vertica

二、DM7核心技术

达梦7 系统架构


动态扩展、收缩BUF
定时触发器/JOB/检查点
Scheduler 线程
复制，备机日志重做
RAC消息处理等
Task 线程
创建会话
用户态线程驱动
Worker 线程
PL/SQL 及其他数据
库服务
DB/UTHR 线程
任务队列
MAL队列
系统缓冲区
HUGE缓冲区
FAST
RECYCLE
KEPP
DEFAULT
0..N
字典/计划缓存
SQL/结果缓存
REDO
缓冲
并行日志
PLOG
并行消息
并行线程池
文件系统
Ini与ctl系统
封锁系统
Undo/Purge
系统内存池
监控与统计
会话/事务
RAC/全局缓存
控制
Checkpoint 线程
I/O 线程
Listener 线程
Rlogflush 线程
Mal 线程
Purge 线程
Audit flush 线程
TRACE文件
控制文件
配置文件
归档日志
在线日志
系统表空间
回滚表空间
临时表空间
主用户表空间
其它表空间
HFS 列存
文件

多趟智能优化器


SQL脚本
语法树
SFW结构
关系树
优化了的关系树
语法分析
语义分析
关系代数变换
代价优化
物理计划生成
执行计划

消除子查询

- 子查询完全分解
  - IN/EXISTS
  - 相关/不相关
  - 转换为半连接
  - 消除了代价不可控子计划

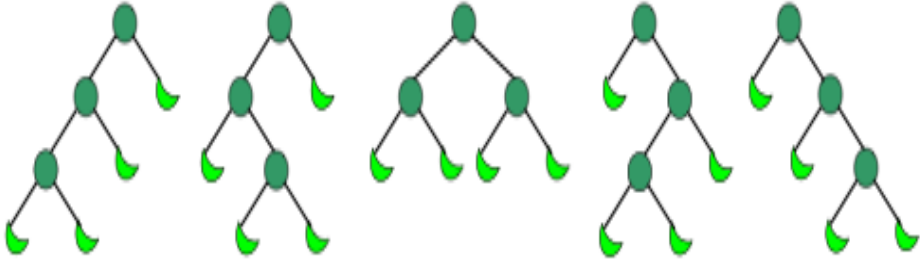
```
select *
from   t1
where  c1  + (select d1
              from   t2
              where  t1.c2 = t2.d2) < 10000
```

名称	附加信息	代价	结果集
NSET2		2	609
PRJT2	exp_num(3), is_atom(FALSE)	2	609
SLCT2	T1.C1+col_name < 10000	2	609
HASH2 INNER JOIN	KEY_NUM(1);	2	609
PRJT2	exp_num(2), is_atom(FALSE)	1	1
HAGR2	grp_num(1), sfun_num(2)	1	1
HASH2 INNER JOIN	KEY_NUM(1);	1	36
DISTINCT		0	6
CSCN2	INDEX33555437(T1)	0	609
CSCN2	INDEX33555438(T2)	0	609
CSCN2	INDEX33555437(T1)	0	609

代价优化

- 代价单位ms
- 影响代价因素
  - 表的数据行数
  - 数据块数
  - 可利用的索引
  - 内存、IO、CPU计算量
  - 内存紧张度
  - 基本操作的CPU周期数

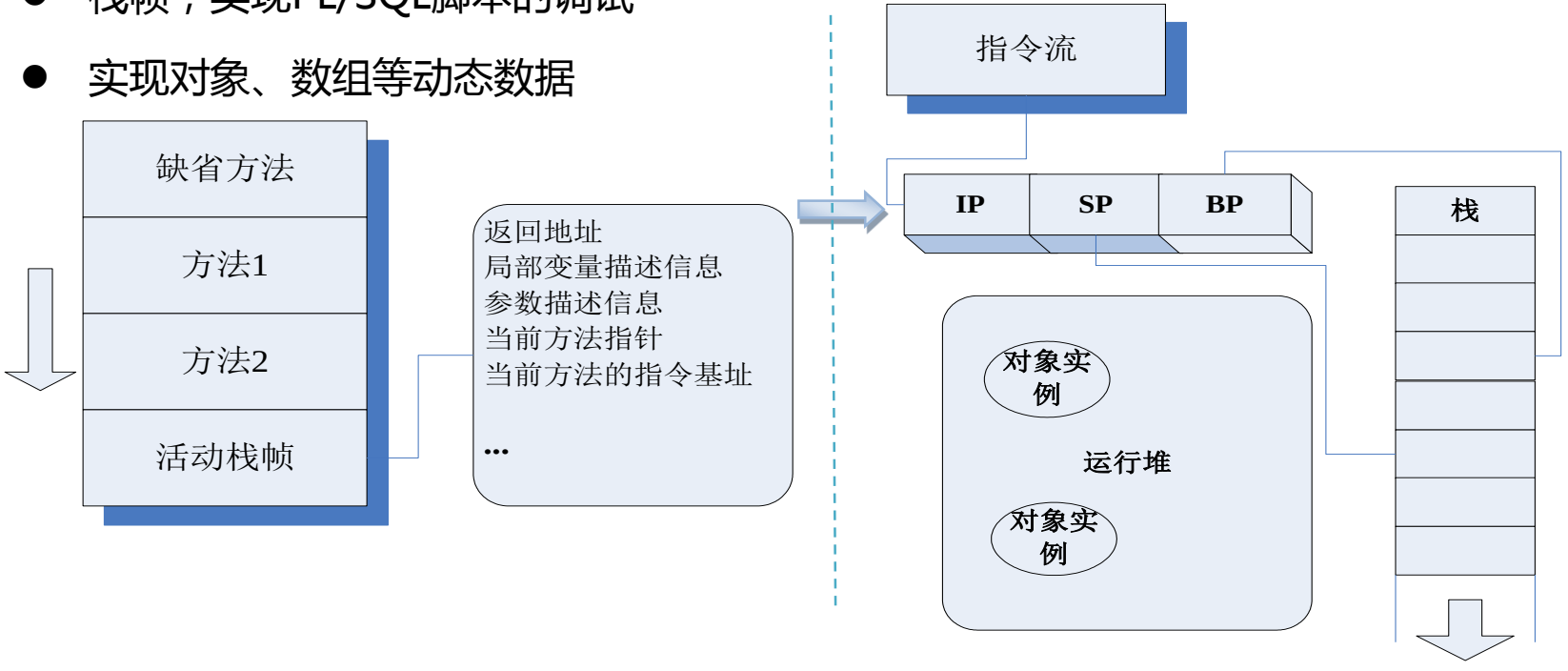
操作	符号	典型单行CPU周期
扫描	SCAN	400
定位	SEEK	4000
检索	LKUP	4000
嵌套索引连接	NLIJ	200
散列连接	HI	300
归并连接	MI	300
嵌套循环连接	NL	200
散列定位	HI_SEARCH	200
过滤	FLT	20



- 连接优化
  - 确定的连接次序
  - 基于卡特兰2叉树
  - 是否下放过滤条件

面向栈的PL/SQL指令虚拟机

- 字长为分配单位的标准堆栈
- 充分利用CPU的2级缓存
- 栈帧，实现PL/SQL脚本的调试
- 实现对象、数组等动态数据



指令系统


```
DECLARE
 PROCEDURE P( X int,
  Y OUT INT) IS
 BEGIN
   PRINT Y;
   SELECT COUNT(*) INTO
   Y  FROM T1, T2
   WHERE T1.ID = T2.ID
         AND T1.ID = X;
   PRINT Y;
 END;
 A  INT;
BEGIN
  A := 10;
  P(12, A);
END
```

```
0    savepoint $XSPT_ec99668
19   dop_try_begin 0
23   dop_try_begin 0
27   push        0
33   load 10
39   sloc  2
43   push        0
49   leal
51   load 12
57   invsubm   0
61   nop
63   jmp  116
69   nop
71   push        1
77   swap
79   sloc  1
83   err_set 1
87   rollback to $XSPT_ec99668
106   jmp  116
112  nop
114  throw
116  nop
118  savepoint $XSPT_ec99668
137  cop 'a'
141  hlt 0
```

- 装载/存储指令
- 跳转指令
- 运算与比较
- 对象访问指令
- SQL指令

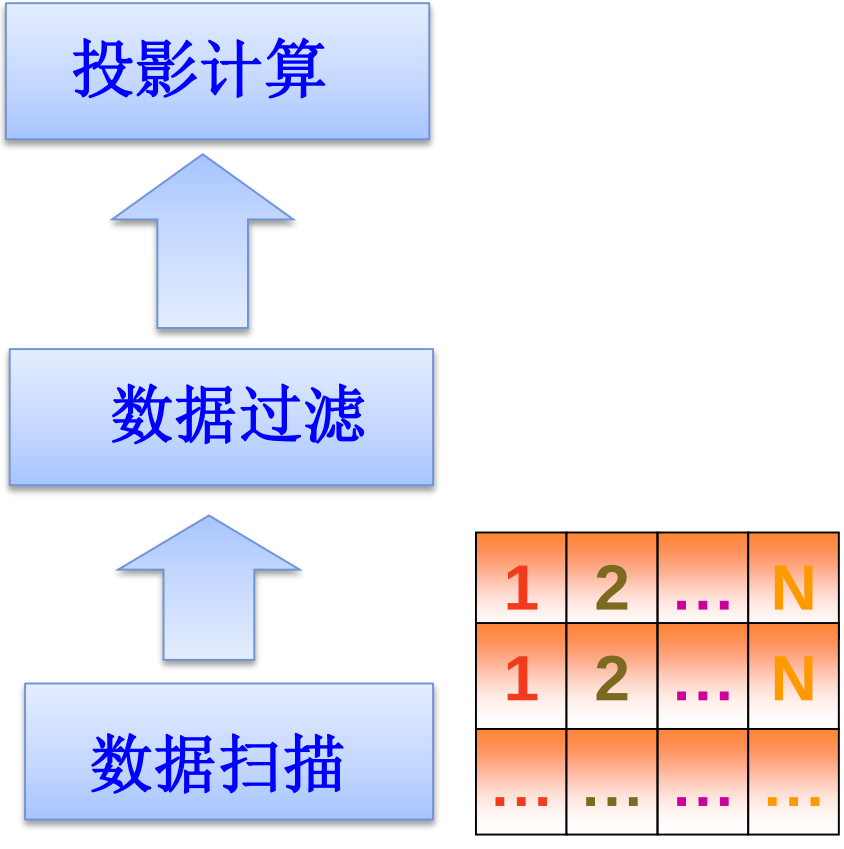
- PL/SQL的实现基础
- 一套指令
- 一个引擎

基于状态的SQL操作符执行


```
while (TRUE)
{
    switch(slct2_vm->state)
    {
    case SLCT2_STATE_START:
        code = slct2_exec_start(slct2_vm);
        if (!DM_SUCCESS(code))
            return vm_raise_runtime_error(vm, code, NULL);
        break;

    case SLCT2_STATE_FETCH:
        slct2_exec_fetch(slct2_vm);
        break;

    case SLCT2_STATE_AFTER_FETCH:
        code = slct2_exec_after_fetch(slct2_vm);
        if (!DM_SUCCESS(code))
            return vm_raise_runtime_error(vm, code, NULL);
        break;

    case SLCT2_STATE_COMPLETE:
        slct2_exec_complete(slct2_vm);
        vm_set_child_over(slct2_vm->common.vm);
        vm_set_absolute_ip(slct2_vm->common.vm, (byte*)slct2_vm->common.parent);
        break;
```

- 数据从叶子向根流动
- 完成数据加工
- 每个操作符内部对应一个自动机
- 统一的异常处理
- 统一的数据包传递格式

高性能批量处理



- 减少控制权限的反复传递
- 提升CPU的有效利用率
- 便于表达式批量计算

例：

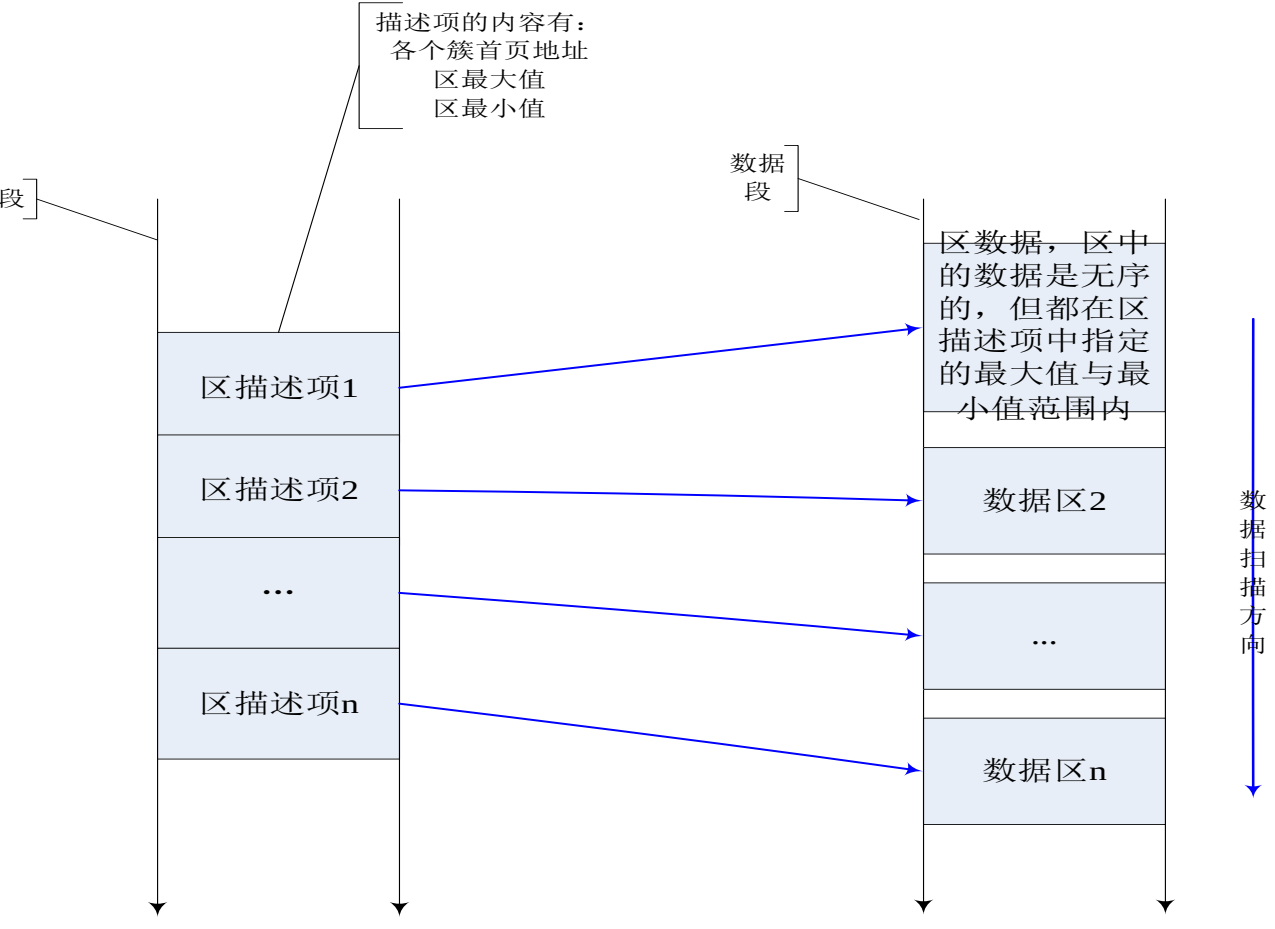
```
Select id + 10, name
From   T
Where  (id + 20) <> 100
```

高性能批量表达式计算

```
for (i = 0; i < n; i++)
{
    r = (int64)opr1[i] + opr2;

    if (r != (int)r)
        return EC_DATA_OVERFLOW;

    res[i]  = (int)r;
}
```

- 虚拟机支持批量计算指令
- 一次计算一批数据
- 利用CPU的CACHE
- 利用CPU的SIMD特性
- 避免传统DBMS的函数反复调用代价
- 接近于C的效率
- 比一次一行模式快10-100倍以上

列式存储



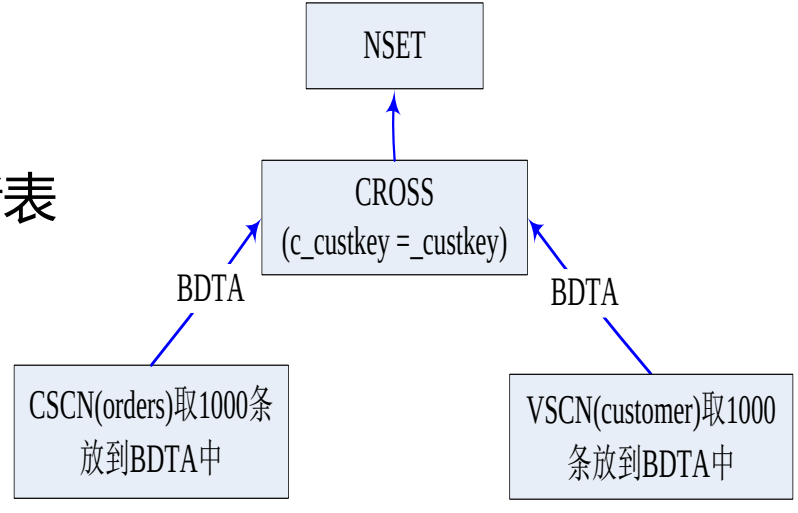
- 两个列式引擎
- Vertical表，支持 ACID
- HFS 表， 不支持ACID

- 提升OLAP应用性能
- 减少IO
- 便于压缩
- 消除物理记录解析代价

行列融合之查询

例： select count(*)
from orders, customer
where c_custkey = o_custkey;

- orders: 行存储表
- customer: 列存储表



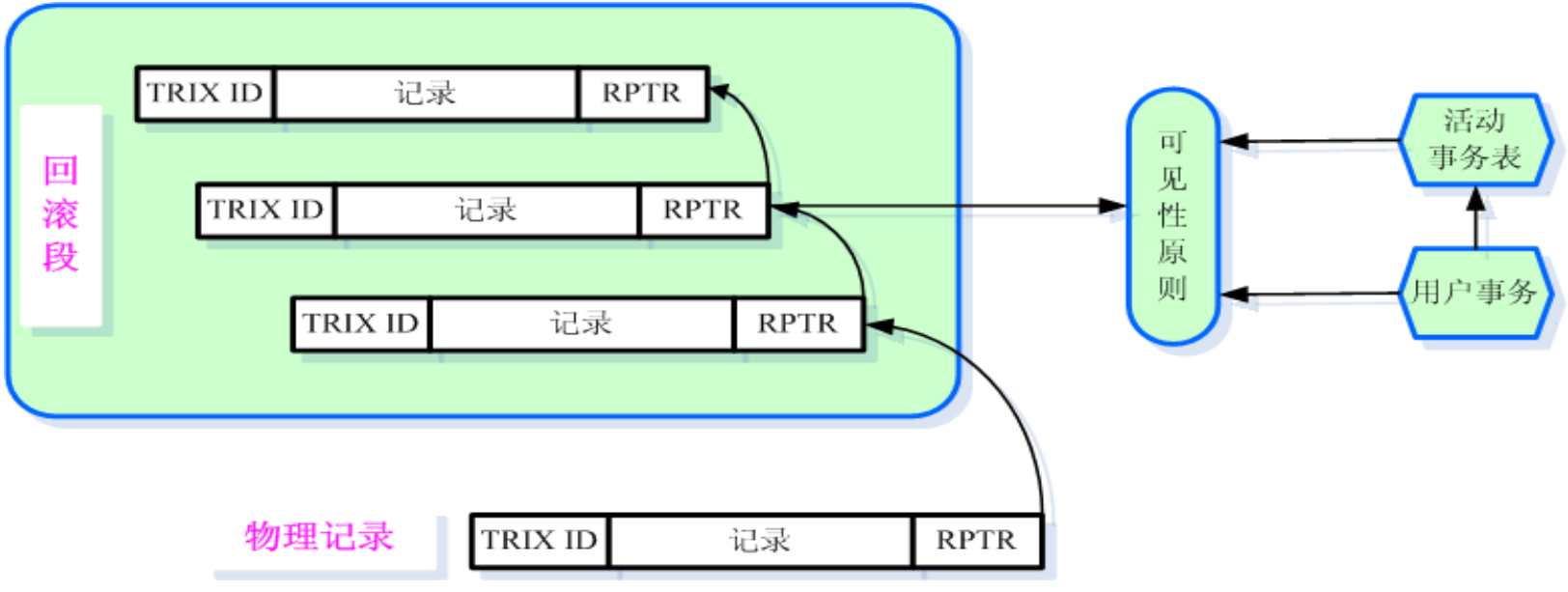
- 底层批量数据机制
- 行列混合查询与处理
- **同时支持OLAP和OLTP业务**

多版本并发控制(MVCC)

与ORACLE 完全一致的MVCC控制技术

基于回滚段的事务恢复机制

更新操作不影响读操作，查询永远不会阻塞



DDL高并发

- DDL转化为系统表的DML操作
- 不必封锁整个数据字典，提高并发度
- DDL的快速提交/回滚

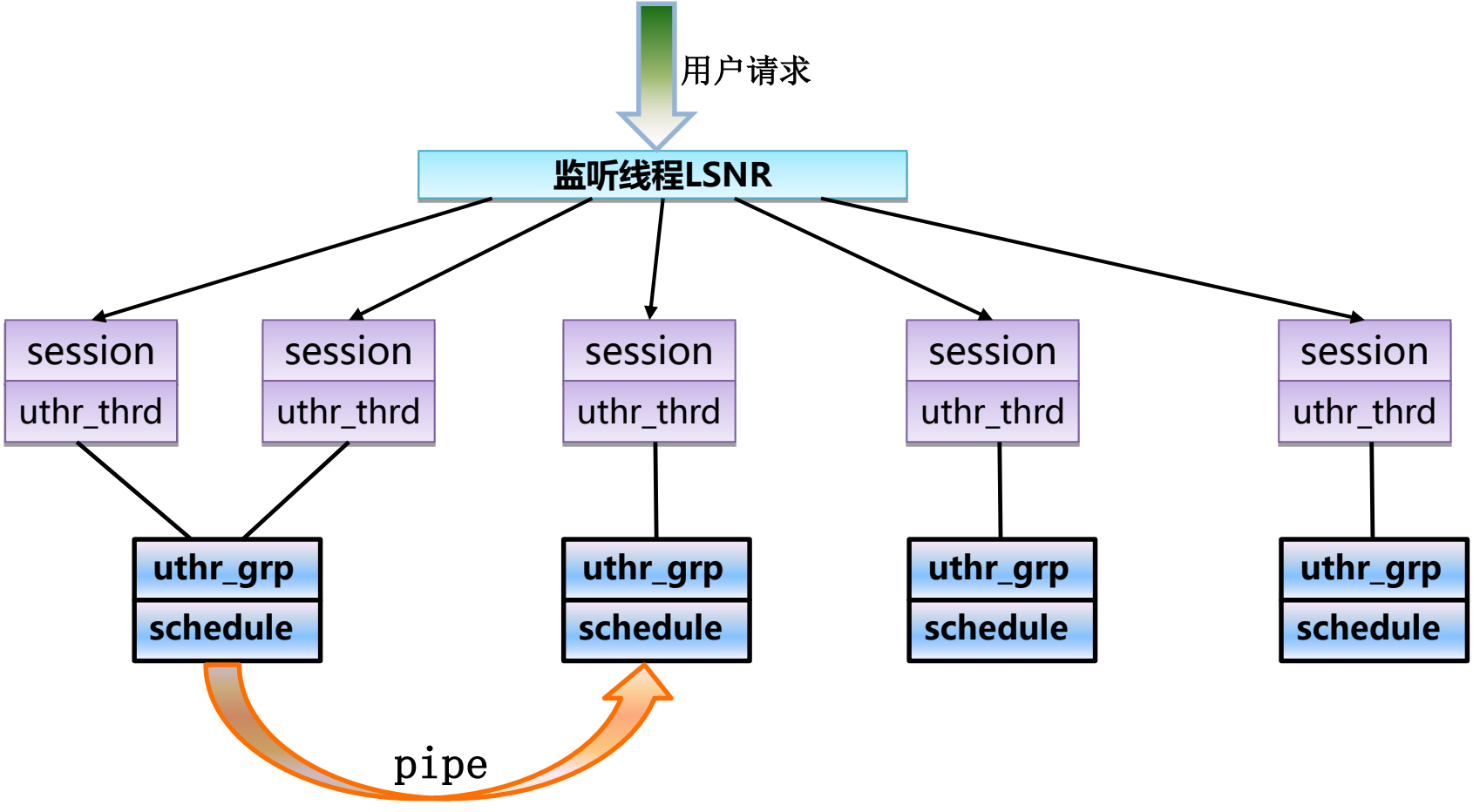
例: DROP TABLE T;

```
{
    DELETE FROM SYSINDEXES WHERE ID = 33555705;
    DELETE FROM SYSSTATS WHERE ID = 1250 AND COLID = 0 AND T_FLAG = 'C';
    DELETE FROM SYSOBJECTS WHERE NAME = 'T' AND TYPE$ = 'SCHOBJ' AND
        SCHID = 150994945;
    DELETE FROM SYSCOLUMNS WHERE ID = 1250;
}
```

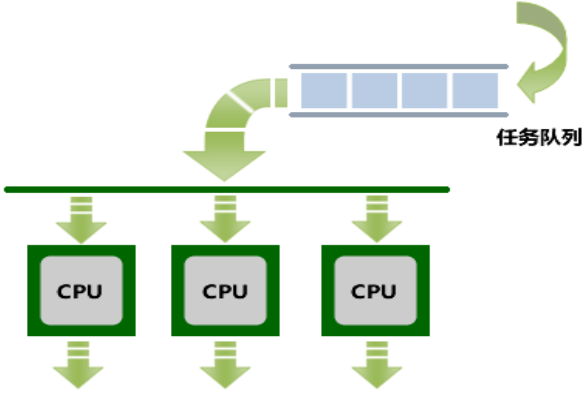
高性能的行级并发

- 没有行锁/表锁
- TID锁
  - 用于事务的等待与唤醒
  - 事务启动时创建TID X锁
  - 等待者LOCK(TID, S)
  - 仅在需要等待时才创建S型TID锁
- 对象锁
  - 字典锁和表锁的结合
- 技术前提
  - MVCC, X_LATCH
  - 一个数据块任意时刻最多仅有一个线程可改
- 不冲突就无需加锁

可配置的高性能用户态性程



并行、分区、压缩



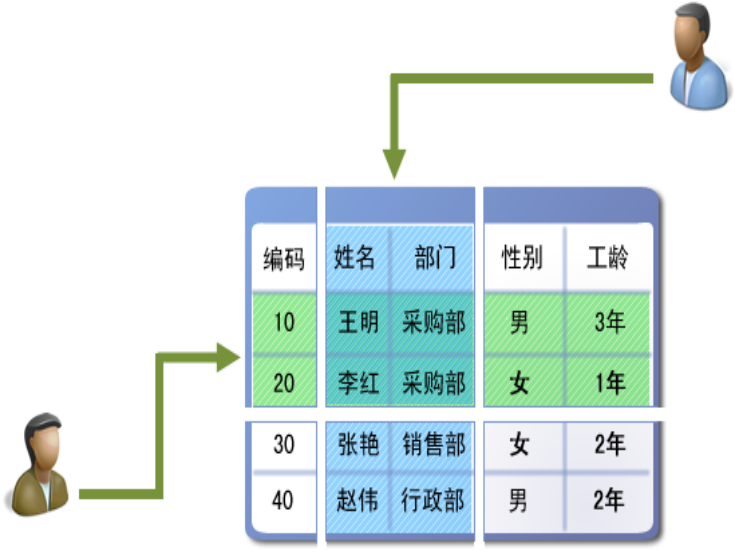
可配置的并行查询

最大并行度 / 并行策略

/*+ parallel(T 8) */

适合OLAP，多核 / 大内存 / 少并发

- 垂直分区（行表）
- 水平分区（行表 / 列表）
  - 列表、范围、HASH
  - 多级混合分区
- 数据压缩（行表 / 列表）



其他高级特性与技术

- P L/SQL调试
- 全文、位图、连接索引
- 物化视图

- 安全四级
- 全库、硬件加密
- 四权分立
- Trace 与监控事件

三、大规模并行处理(MPP)

对称无共享大规模并行处理

TB/PB级数据分析

并行装载

并行查询

高性价比

优良的伸缩性能

各节点功能完全对等

最大支持1024个节点

OLTP应用提供良好的支持

数据分布式存储

所有节点并行执行

无需特殊软、硬件

无单点故障

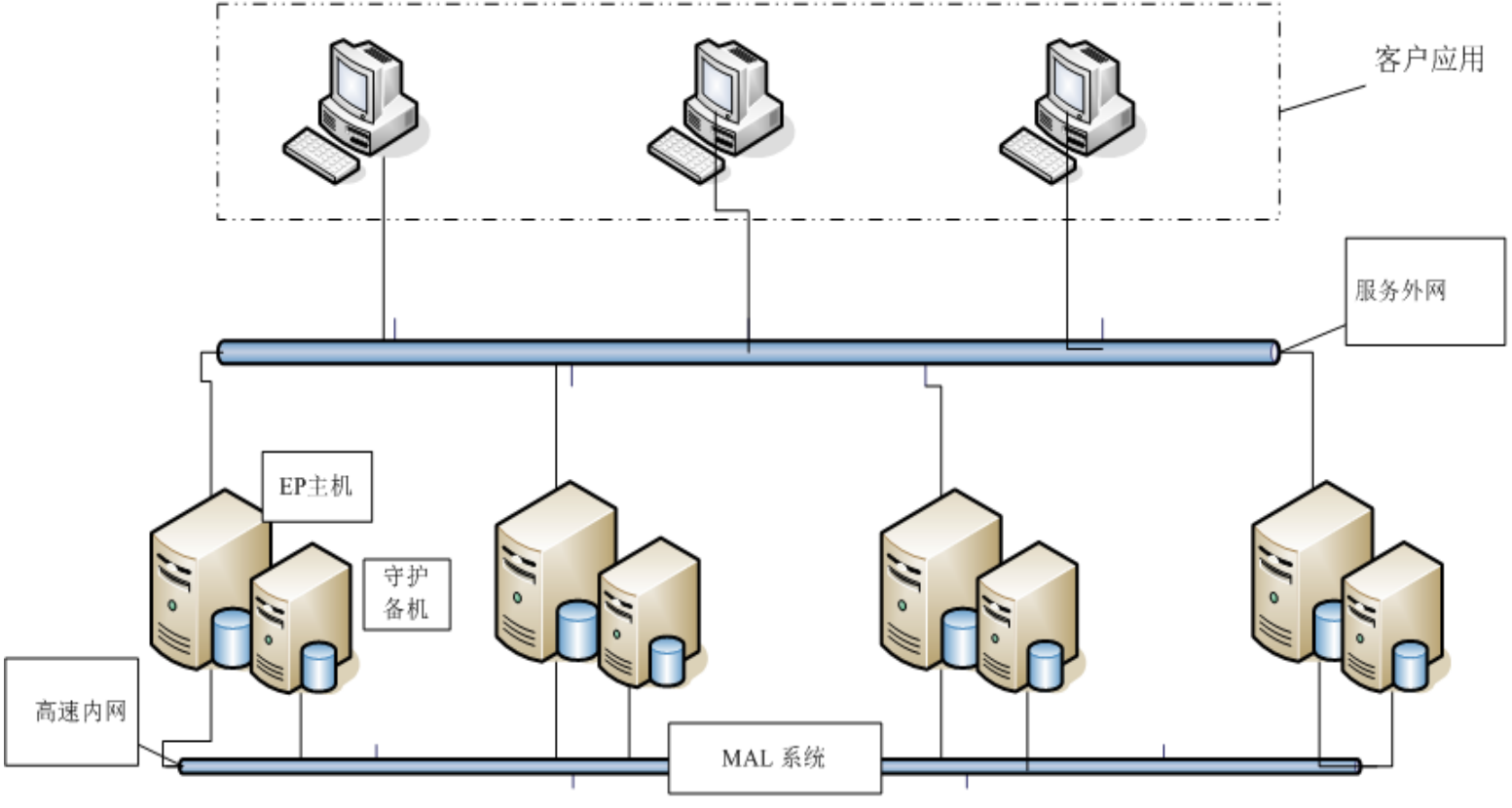
图形化监控和管理工具

支持绝大部分单机功能

同时支持行、列存储

支持存储过程、索引、分区表等

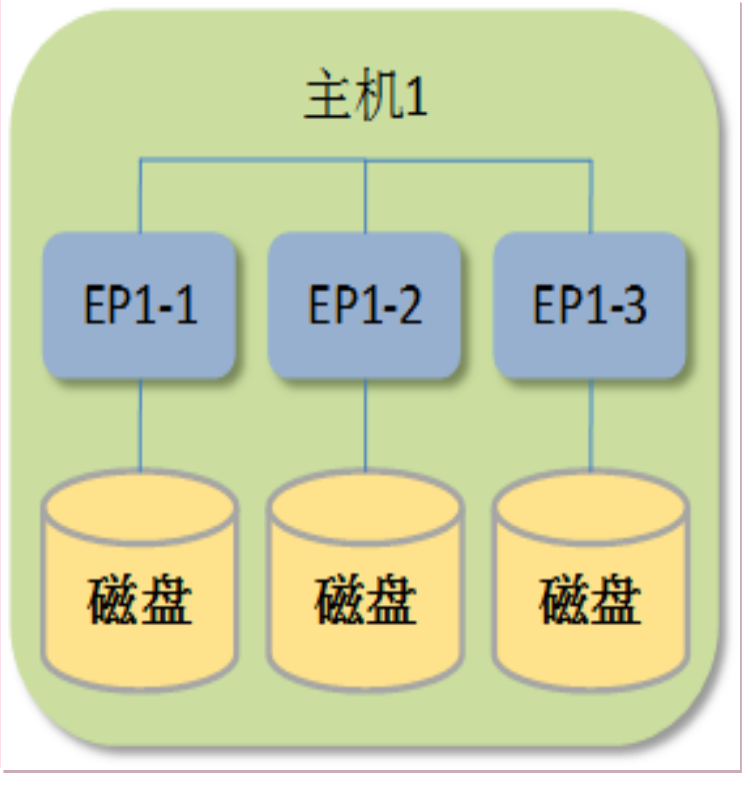
DM7 MPP 系统架构



高速邮件系统

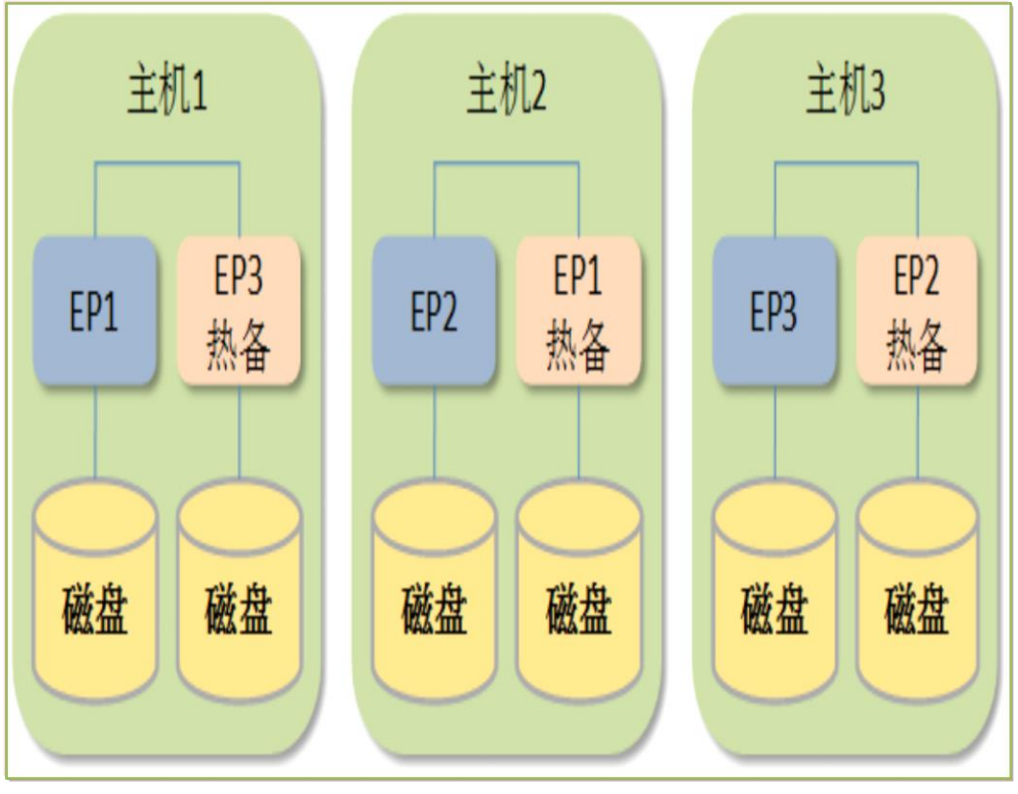
- 模拟真实世界的邮局功能
- 协调各节点通信和数据移动
- 流式处理，不堵塞
- 多条通信链路
- 发挥以太网交换机性能

执行节点(EP)



- 按照接入点区分节点主从类型
- 每个节点管理一份数据
- 一个物理主机可运行多个EP实例
- 各节点完全对等
- 都可接受用户接入
- 支持本地接入类型

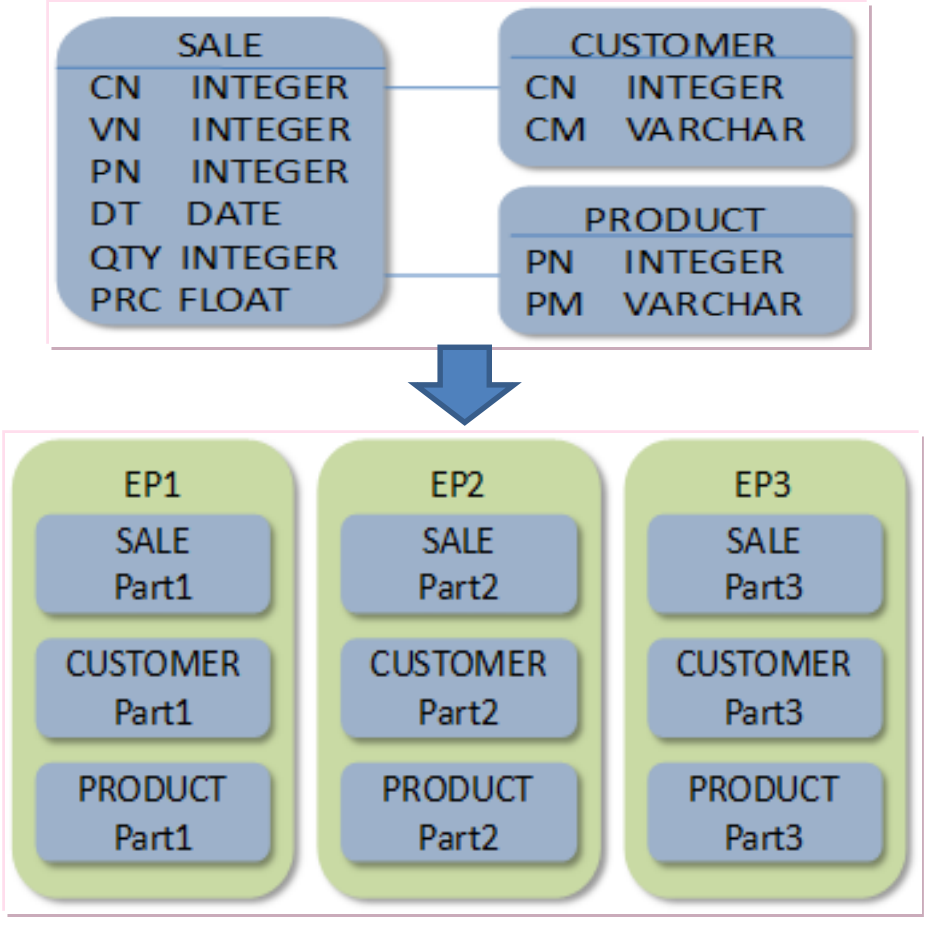
交叉数据守护



- 可交叉配置数据守护，提供数据镜像保护功能
- 通过日志传送来实现冗余
- 单机故障不影响系统功能
- 主机故障后，备机自动切换成主机并提供服务

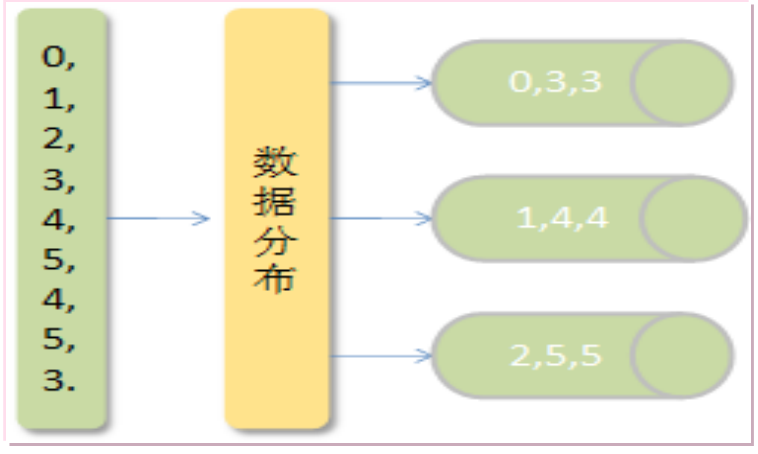
- 降级运行（开发中）

灵活的数据分布方式



- 多种数据分布
  - HASH分布
  - 范围分布
  - 随机分布
- 多种数据分区
  - 一级分区
  - 多级混合分区
- 数据分布与数据分区的组合
- 行存/列存
- 列存数据压缩
- 提供极高的灵活性

并行数据加载



- 本地加载模式
- 客户端分发模式
- 服务器端分发模式
- 数据均匀分布可实现最大并行效率

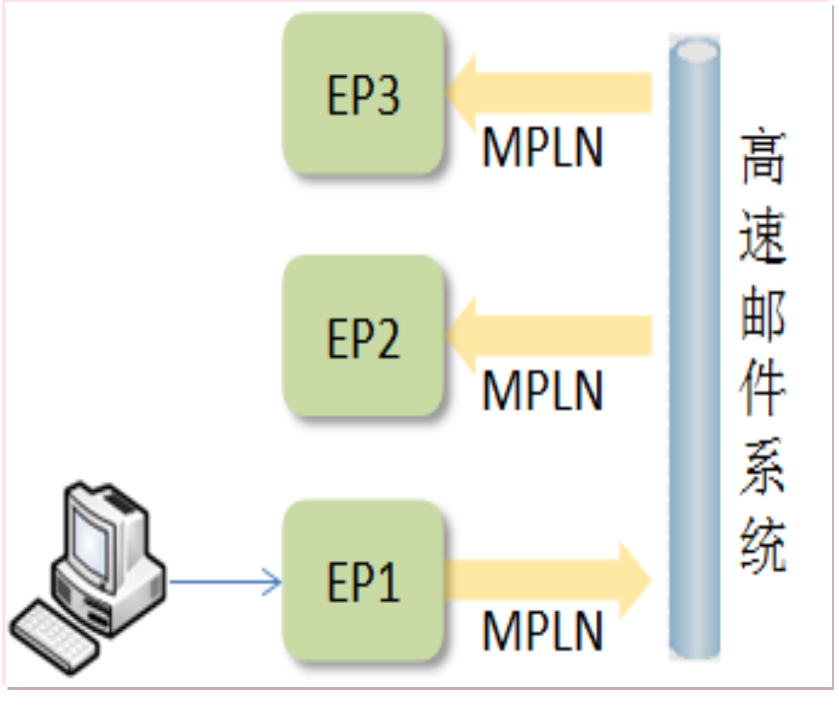
单节点平面数据本地加载性能

极限性能：400G/小时

TPC-H 100G 仅需15分钟

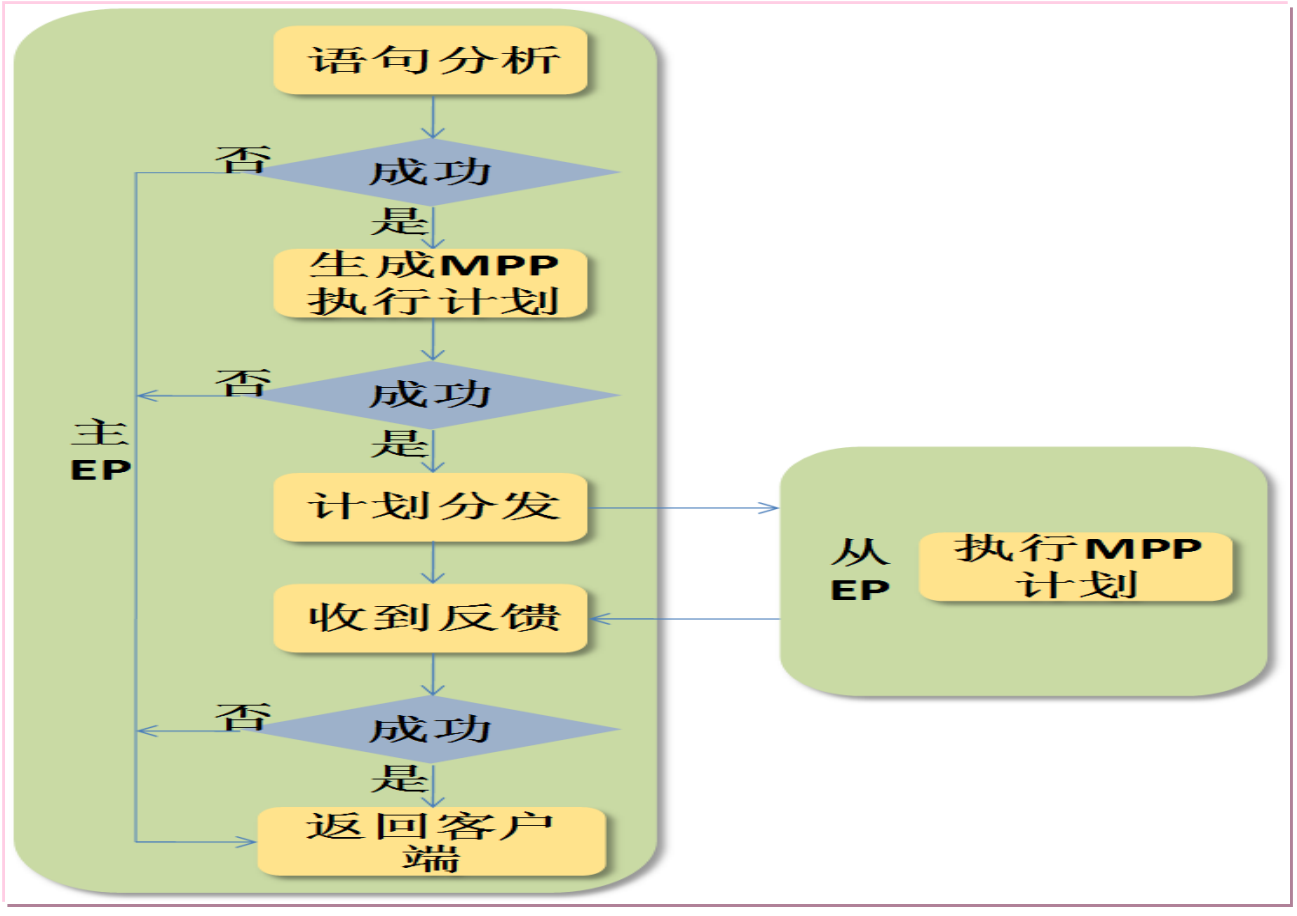
极限近 1,000,000行/秒

并行执行流程



1. 用户选择一个节点（如:EP1)接入
2. 主EP接受 SQL请求，并生成并行执行计划
3. 主EP把计划打包后分发给其他从EP
4. 各EP并行执行
5. 主EP收集各EP（包括自己）的执行结果
6. 汇总后返回给用户

DDL/DML的区别

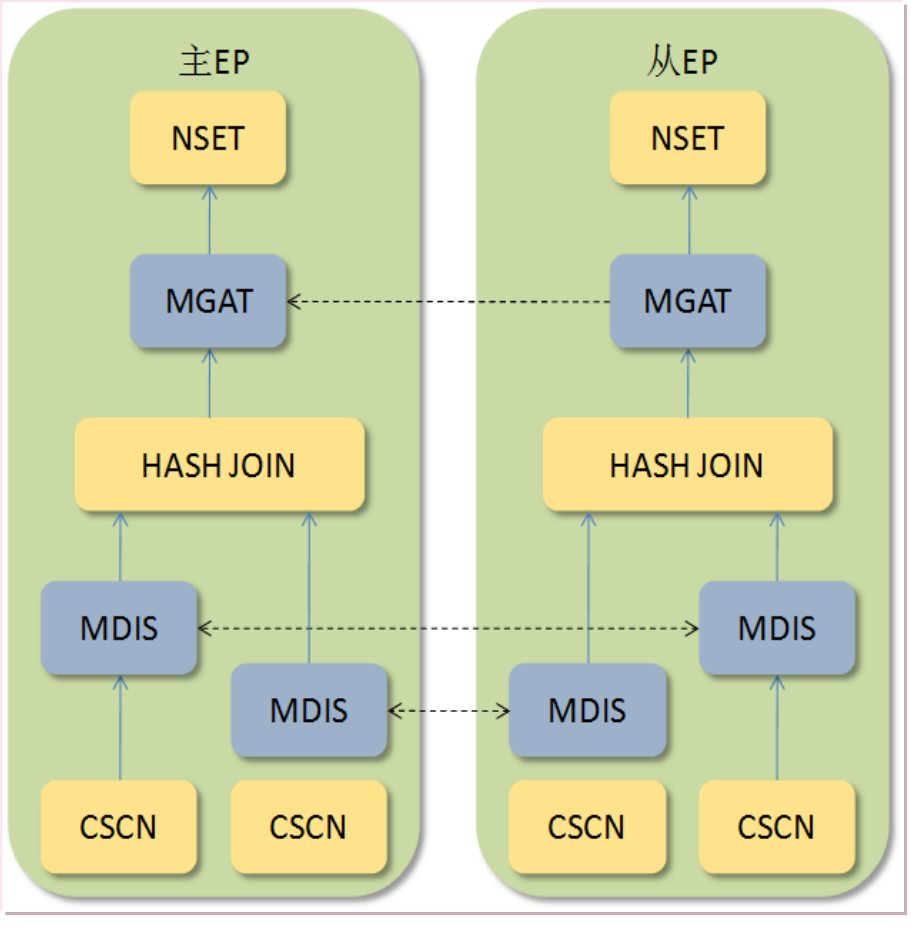


DDL是语句分发

DML是计划分发

EP间同一对象用名字关联

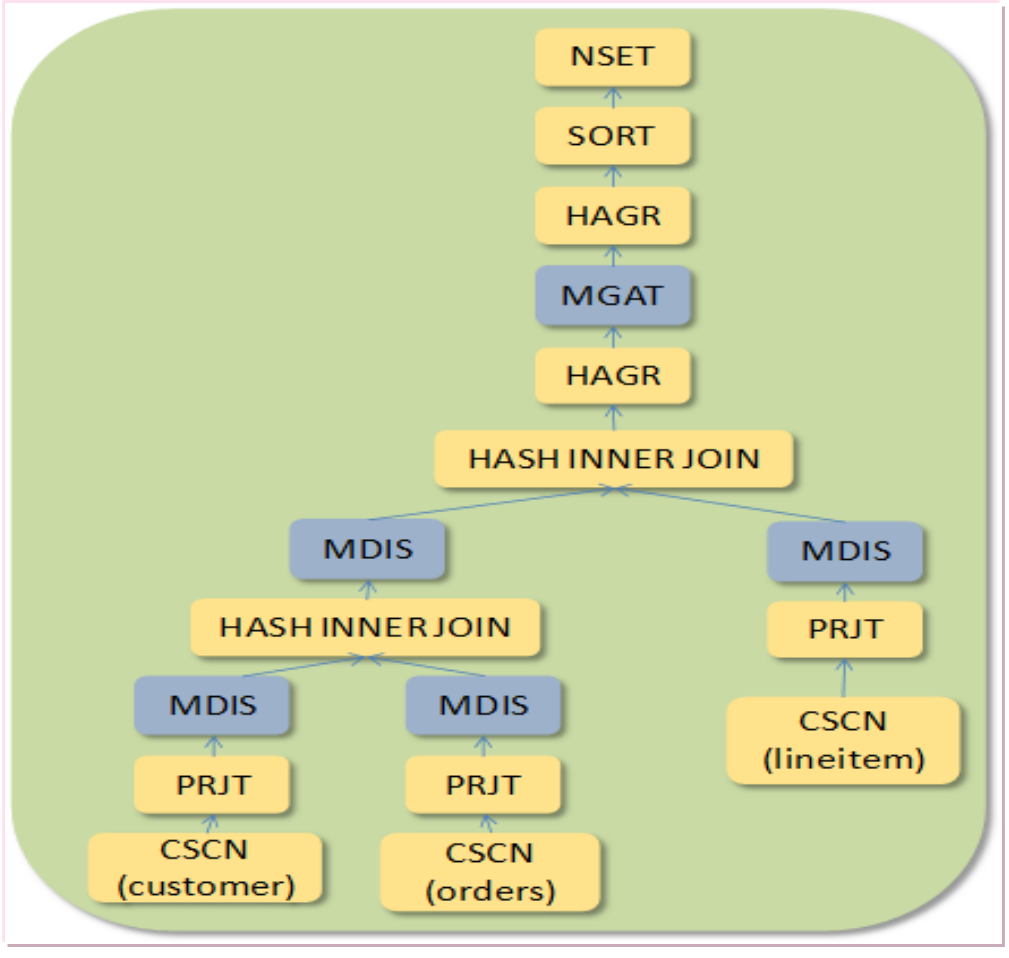
并行执行计划



- 扁平的执行计划（无子计划）
- 比单节点计划增加了通讯操作符号
- 优化器phf阶段，常规计划变换为并行计划
- 通讯操作符，通讯与同步
  - 数据收集 MGAT
  - 数据分发 MDIS
  - 数据广播 MSCT/MBRO

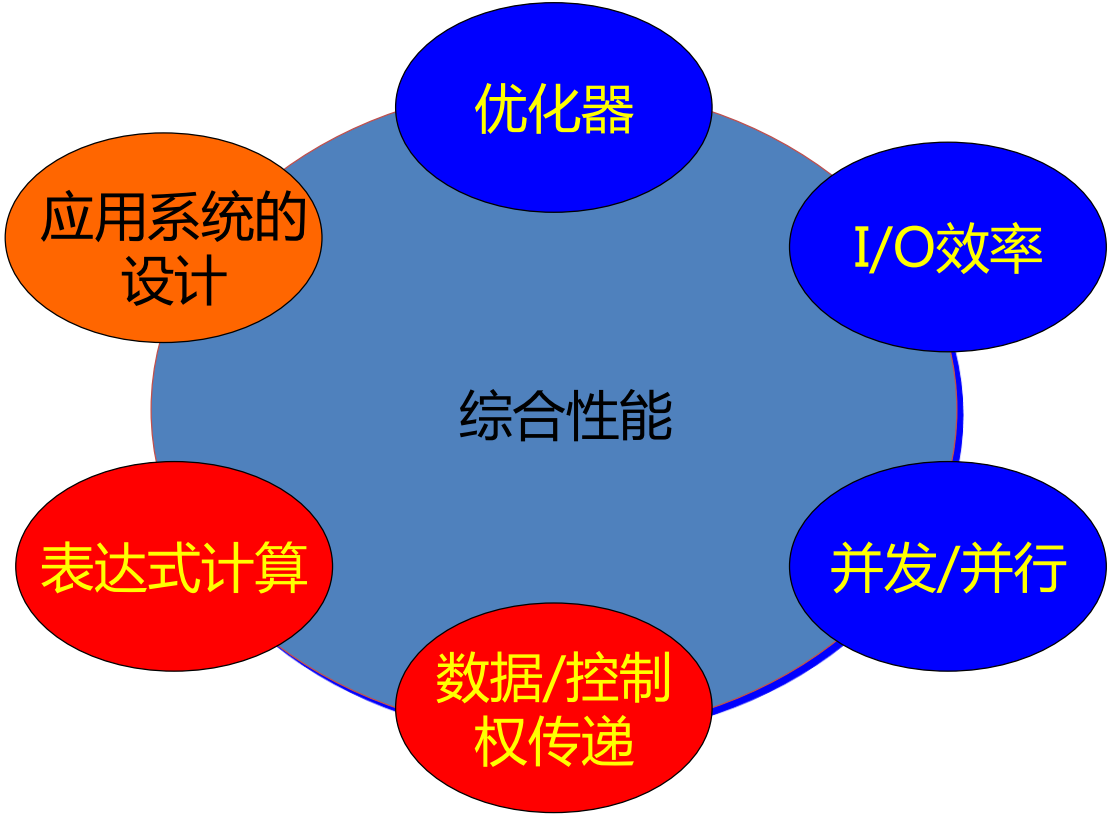
并行执行计划实例（三表连接）

```
select top 10
    l_orderkey,
    o_orderdate,
    o_shippriority
from customer, orders, lineitem
where c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and o_orderdate<'1995-03-15'
    and l_shipdate > '1995-03-15'
group by
    l_orderkey, o_orderdate
order by
    revenue desc, o_orderdate;
```



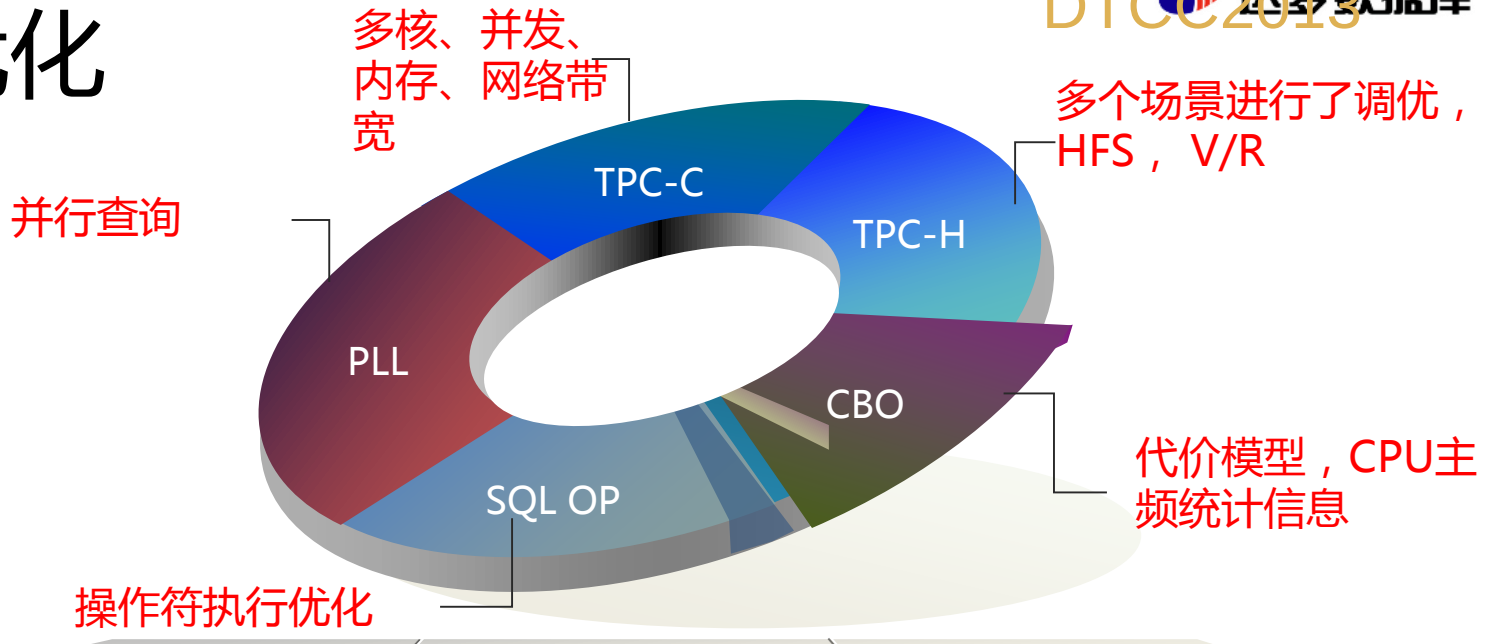
四、性能与应用场景

性能的理解



- OLTP
  - 应用设计
  - 优化器
  - Ｉ／Ｏ
  - 并发
- OLAP
  - 应用设计
  - 优化器
  - Ｉ／Ｏ
  - 并行
  - 数据／控制转移
  - 表达式计算

性能优化



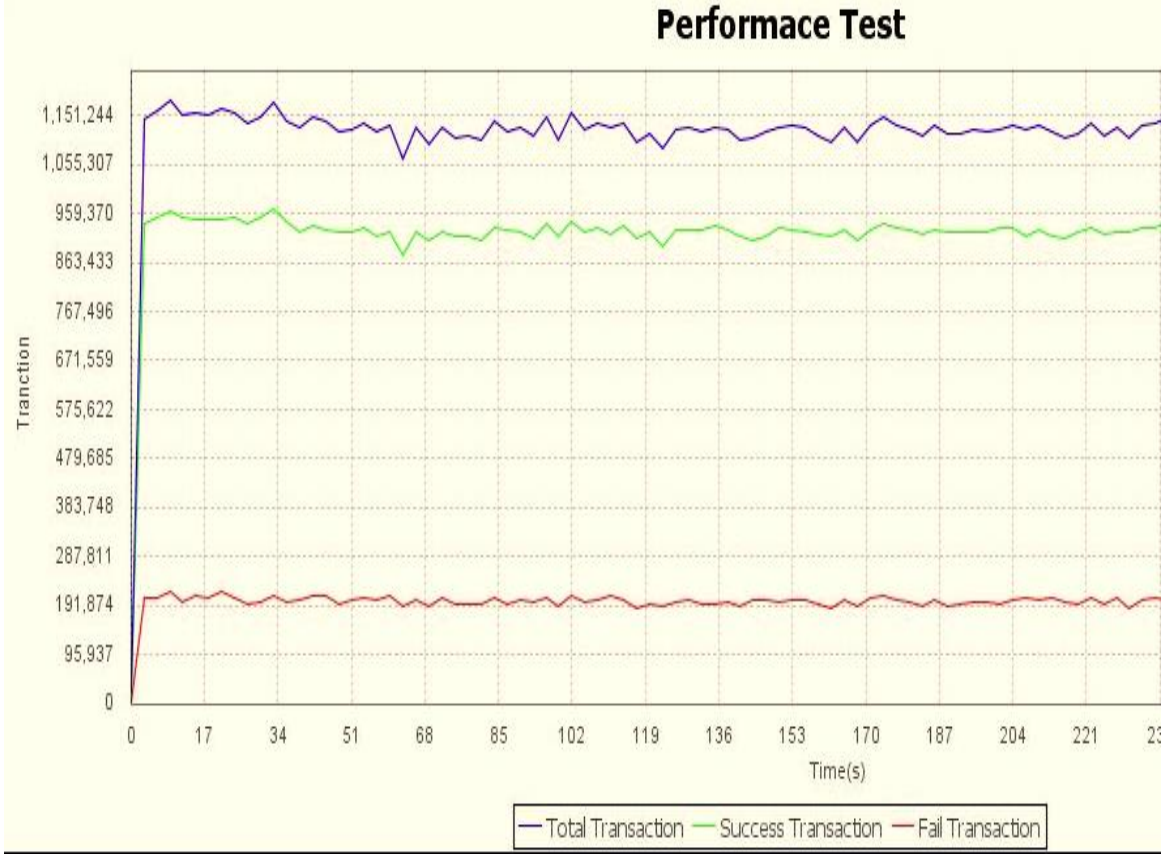
表达式与谓词	分区表优化	接口与工具
• 单表谓词下放 • OR表达式 • Exists (select * • 函数批量	• DDL调优 • 实现脚本完全改写 • 子表的获取方式优化 • DML调优	• DTS 分析效率 • SQL脚本调试性能 • OCI接口优化

单节点OLTP性能 (TPC-C)

- tpmC 158000
- 非存储过程测试模型
- 事务客户端用SQL驱动

- ORACLE 11g对比测试中，性能是ORACLE的**2**倍

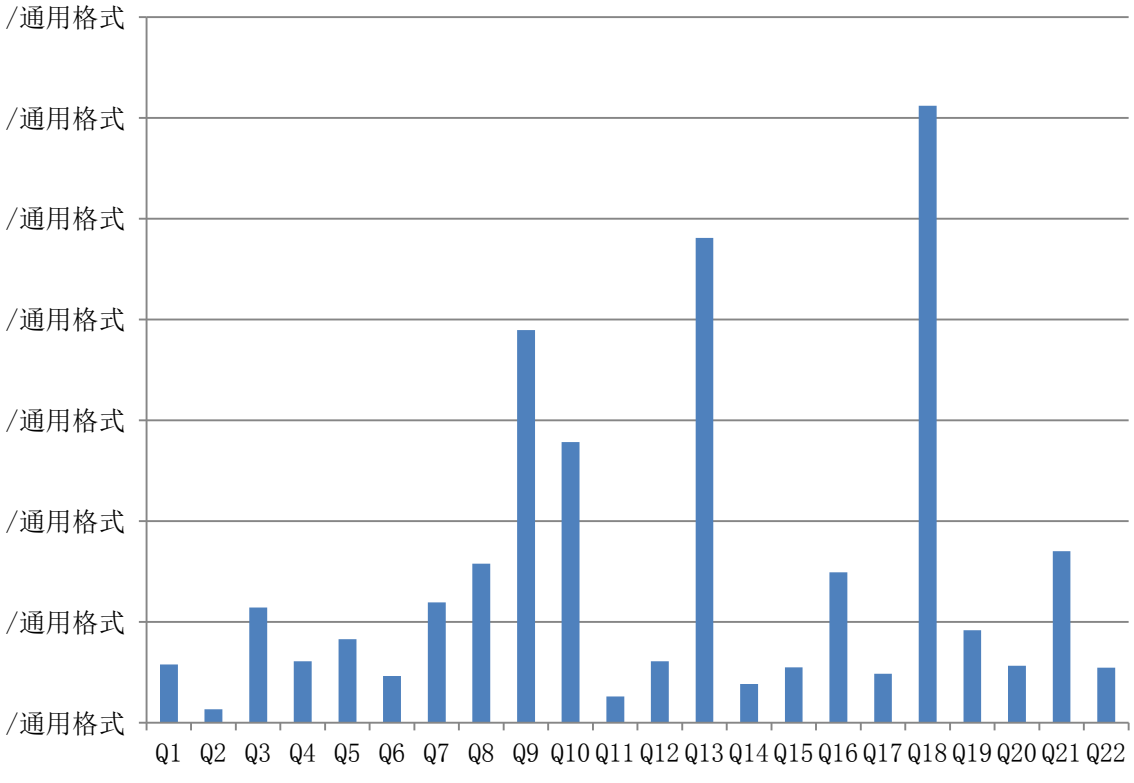
单节点OLTP性能 (TPC-C)



- 存储过程测试模型
- 每秒执行事务18000+

- ORACLE 11g对比测试中， 性能是ORACLE的**3**倍
- 存储过程的脚本与oracle兼容:
  - http://vdisk.weibo.com/u/1871115162

单节点OLAP性能 (TPC-H)



- 单位: 秒
- TPC-H 100G
- 6路并行
- Q1每秒处理约5000万记录

- Q9, Q13, Q18有优化余地

单节点OLAP性能 (电信应用测试)

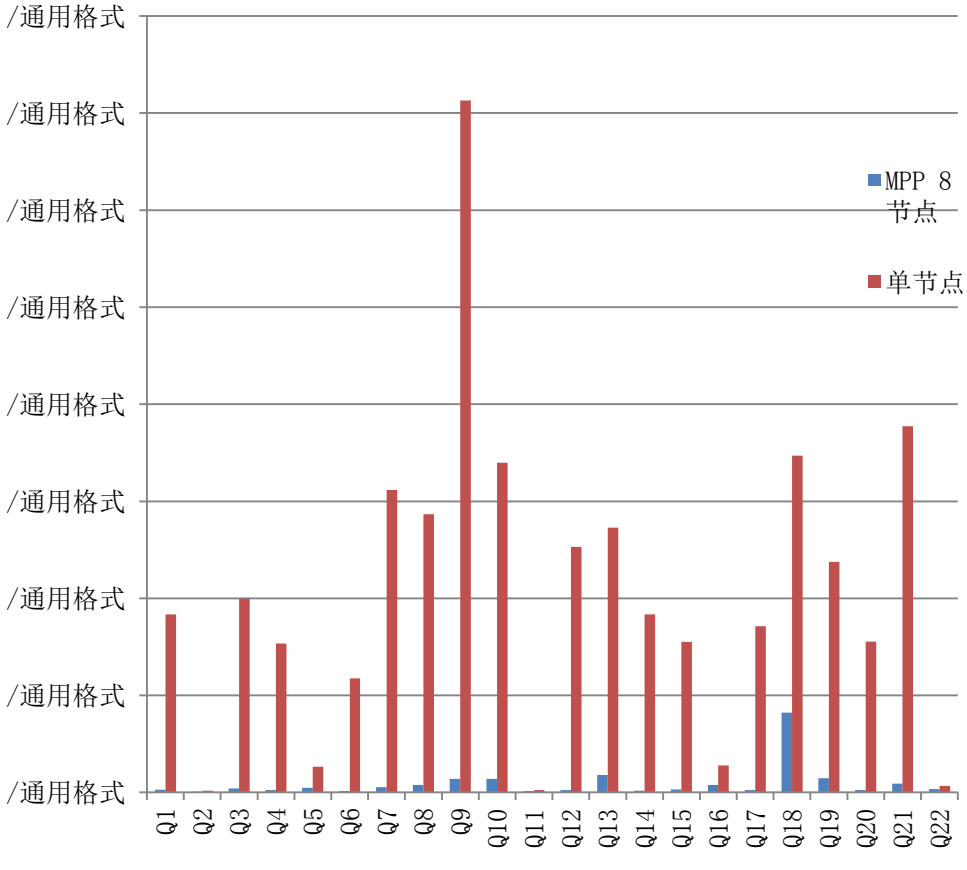
- 浪潮NF5280

- fact_revene_detail
  - 3.3亿记录, 1T
  - 列存压缩后235G
  - 装载时间: 2:46:12

	第一次	第二次
olap1 query	69	24.302
olap2 query	91.601	64.535
olap3 query	116.935	108.599
olap4 query	434.293	168.355

```
select
        partition_date          ,
        product_name            ,
        service_level1          ,
        service_level2          ,
        service_level3          ,
        payment_type            ,
        sum(hit) hit            ,
        sum(dur_second) durasi  ,
        sum(sms_length) sm_length  ,
        sum(main_balance) main_balance,
        sum(wallet) wallet      ,
        sum(free) free          ,
        sum(bonus) bonus
from
        fact_revenue_detail
group by
        partition_date,
        product_name  ,
        service_level1,
        service_level2,
        service_level3,
        payment_type;
```

MPP对复杂查询的高效处理



	4节点MPP	单实例	相对倍数
Q1	5.77	7.14	1.24
Q2	1.41	3.63	2.57
Q3	8.22	398.93	48.56
Q4	5.13	307.16	59.88
Q5	9.85	53.38	5.42
Q6	3.23	235.38	72.96
Q7	10.65	623.66	58.54
Q8	15.44	573.31	37.12
Q9	28.25	1426.27	50.49
Q10	28.17	679.81	24.13
Q11	2.76	4.97	1.80
Q12	5.11	746.41	146.04
Q13	35.93	545.80	15.19
Q14	3.26	366.90	112.41
Q15	6.43	310.35	48.30
Q16	15.53	55.41	3.57
Q17	5.17	342.56	66.21
Q18	164.17	773.83	4.71
Q19	29.64	615.23	20.76
Q20	5.10	310.64	60.91
Q21	18.42	754.66	40.98
Q22	6.97	13.69	1.96

- TPC-H 100G
- 4节点 vs 单节点
- MPP 消除IO
- MPP 高效处理
  - 多表连接
  - 子查询
  - 分组
  - 聚合

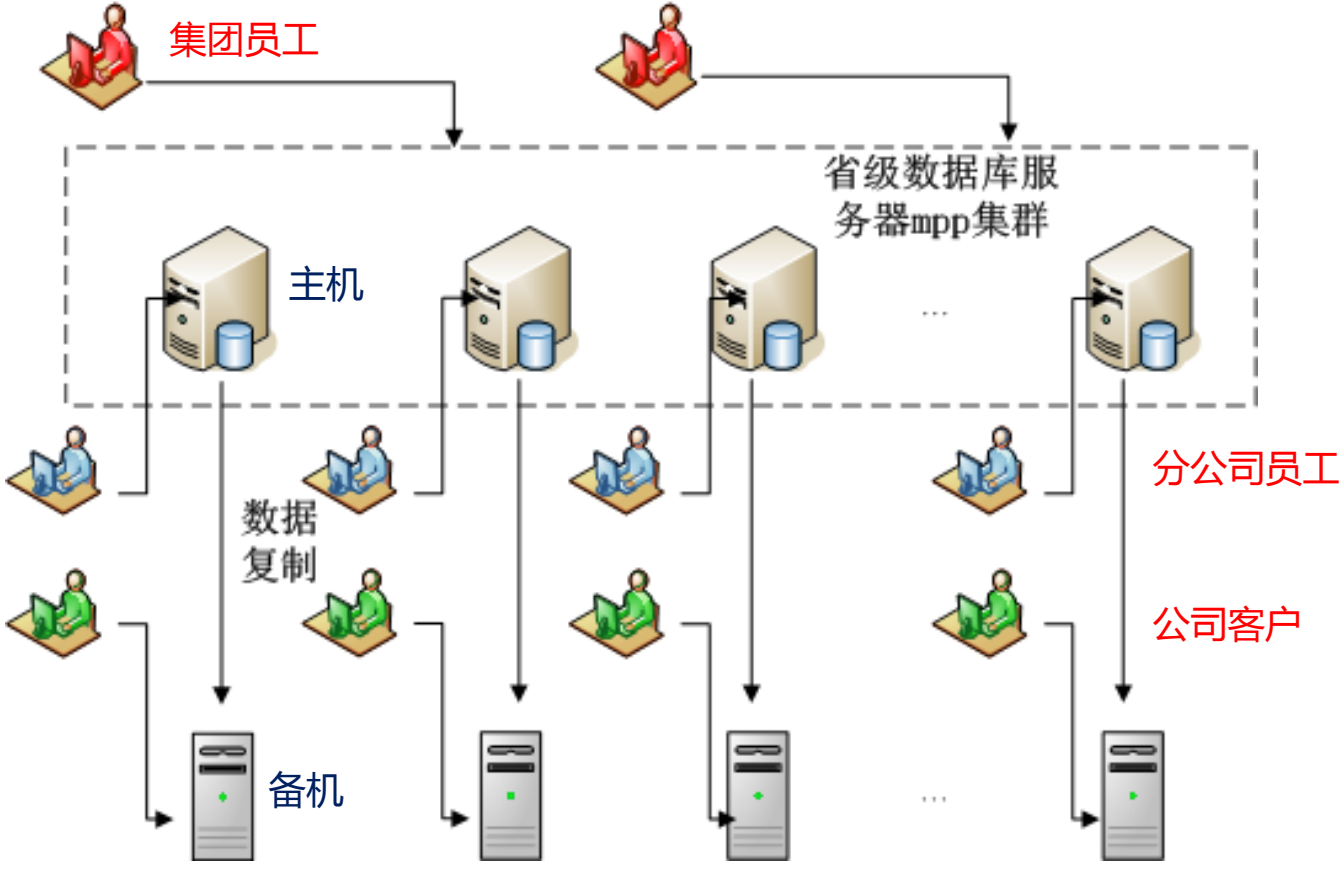
DM7 MPP与Greenplum 对比

题　目	V表二级分区	V表一级分区	HUGE表	GP 二级分区
精确查询一个值。例：column4 = '13476847881'	5.00	1.00	0.74	10.00
模糊查询包含某个值。例：column3 like '%889%'	171.00	48.00	13.00	68.00
模糊查询以某个值开头。例：column16 like '1397%'	17.00	50.00	166.00	54.00
模糊查询以某个值结尾。　例：column15 like '%999'	100.00	46.00	15.00	68.00
精确查询一个值与精确查询一个值，做'与'连接。例：column4 = ' 13476847881 ' and column16 = '	0.72	0.18	0.60	8.00
精确查询一个值与精确查询一个值，做'或'连接。例：column4 = '13476847881' or column16 = '13476847186'	19.00	8.00	0.60	9.00
精确查询一个值与模糊查询一个值，做'与'连接。例：column4 = '13476847881' and column16 like '%55%'	0.79	0.17	0.60	3.00
精确查询一个值与模糊查询一个值，做'或'连接。例：column4 = '13476847881' or column16 like '%55%'	194.00	13.00	26.00	99.00

- CPU
  - Intel(R) Xeon(R)
  - E5645 @ 2.40GHz 24核 TPC-H 100G
- RAM
  - 48GB
- DISK
  - SAS 10000RPM
- 5节点
- 数据量
  - 10亿记录
  - 150G

某即席查询应用场景

MPP 集群的OLTP业务应用



- 地区主机构成MPP
- 集团员工
  - 集群接入方式
  - 集群访问权
  - 集群读写
  - 集群级分类汇总
- 分公司员工
  - 本地接入方式
  - 地区主机访问
  - 读写本地区主机
- 公司客户
  - 本地接入方式
  - 地区备机查询
- 读写分离/负载均衡

谢谢！

hzz@dameng.com
http://weibo.com/u/1871115162
http://www.dameng.com/downloads-products/