

ORACLE®

Disclaimer

"THE FOLLOWING IS INTENDED TO OUTLINE OUR GENERAL PRODUCT DIRECTION. IT IS INTENDED FOR INFORMATION PURPOSES ONLY, AND MAY NOT BE INCORPORATED INTO ANY CONTRACT. IT IS NOT A COMMITMENT TO DELIVER ANY MATERIAL, CODE, OR FUNCTIONALITY, AND SHOULD NOT BE RELIED UPON IN MAKING PURCHASING DECISION. THE DEVELOPMENT, RELEASE, AND TIMING OF ANY FEATURES OR FUNCTIONALITY DESCRIBED FOR ORACLE'S PRODUCTS REMAINS AT THE SOLE DISCRETION OF ORACLE."



ORACLE[®]

现实环境下的数据库性能优化

Andrew Holdsworth & Zhuo Qu
Real World Database Performance
Server Technologies

数据库性能现状

- 现在大多数数据库系统的在编码阶段已经优化，其性能可以满足客户的需求
- 为什么？
 - CPU的处理能力不断增长
 - 64-bit 计算和内存价格的降低
- 即使低效率的应用都可以在内存中运行

数据库性能现状

- 数据库竞赛的规则正在改变
 - 数据库不再适合放入缓存运行
 - 数据库大小以几何级数增长
 - 64位环境越来越流行
 - 可靠性的需求变得越来越严峻
 - 通常性能的提高是容易的
 - 保持系统的可靠性是困难的.
- 如果要取得成功，我们应该关注数据库的工程学基础而不是把赌注放在新特性和硬件的升级

当前数据库性能所面临的挑战

OLTP

- 事务变得更加复杂
 - 消耗更多的 CPU
 - 需要更多的 I/O
- 事务处理的频率越来越高
- 用户的期望变得越来越苛刻
 - 要求99%统计值是单一的逻辑处理而不是平均值
- 多数数据库不能再完全放入缓存中
 - I/O 规划是成功的数据库的一部分
- 数据库日益因中间件和体系结构的不良设计而备受诘难
- 可靠性的需求限制了数据库发挥出最高的性能

当前数据库性能所面临的挑战

Data Warehouse

- 当前的趋势是数据库的设计工作量和大量的廉价硬件来解决问题
- 这意味着要有良好的容量规划尤其是在I/O方面
- 在很多案例中许多**Oracle**先进的特性特意没有被使用和开发出来
- **Oracle**作为单一的套装软件供应商做了很多客户应该做的规划和打包工作而受到诘难
- **Oracle**很多客户天真地认为大表的扫描操作可以通过硬件的提高来满足要求

真实世界数据库性能的一般反馈

- 性能问题的根源分析
 - 根本原因通常是难以理解的
- 过多的强调平台的性能调整和初始化文件的调整
 - 这些调整带来的提高小于 10%. 一个好的性能调优工程师总是在追求获得最大的性能提高
- 没有足够地关注 SQL 语句的调整 (optimization and quality)
- 性能问题常常被认为是神秘和困难的.
- 你应该使用一个经过设计，建库和测试过的Oracle数据库
 - 这意味着使用缺省的数据库配置参数
 - 不存在一个 db_speed = FAST 这样的参数

数据库性能基础

- 总的来说目前对数据库性能而言有四个基本的核心
 - **Schema** 设计和**SQL**语句
 - 应用设计包括如下核心规则
 - Schema 设计(Tables, Indexes, Data Types, Partitions, Aggregates)
 - SQL 语句
 - 存储规范
 - 会话管理
 - 中间件体系结构
 - 光标管理
 - 应用代码在执行**SQL**语句时有一个可扩展的范围
 - 容量规划
 - 学习数据库物理设计

Schema 设计和SQL

- 经典的关系设计实践教材一直都能工作的最好和经得起时间考验
- 关注如下设计
 - 至少第三范式式设计
 - 使用不同的索引设计高效率的访问路径
 - 选择正确的数据类型(e.g. Char vs VARCHAR2 or NUMBER vs float)
 - 选择正确的分区 (考虑管理性和性能)
 - 选择正确的存储选项 (压缩, extents)
 - 编写简单高效的SQL

提高 SQL 语句的性能

- 确认你的典型的schema的统计值，例如：应用表和索引
- 确保SQL语句所使用的索引的有效性
- 通过检查SQL语句中每个行资源的cardinality评估的有效性来检查正确的优化方法
- 如果cardinality评估错误，那么访问路径和连接类型和顺序也会出现错误

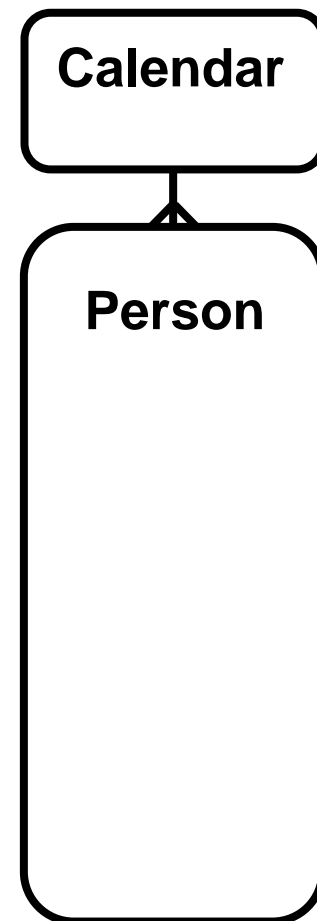
一些估算的简单方法

- 解释计划
 - `dbms_xplan.display`
- 最新的 **SQL** 语句
 - `dbms_xplan.display_cursor`
- 真正和估算的 **cardinality**
 - `GATHER_PLAN_STATISTICS` hint
 - `dbms_xplan.display_cursor(null, null, 'allstats last')`
- 其他光标
 - `Cursor cache`
 - `AWR`

列关联问题的实际例子

十二星座

- Calendar 表是个小表
 - Date
 - Month
 - Sign of the Zodiac
- Person 表是个大表
 - 几乎有 12M 行



关联列

十二星座

- 有多少人的生日在五月?
- $12M * (1 / 12) \sim 1M$

```
-----  
| Id | Operation | Name | Starts | E-Rows | A-Rows |  
-----  
| 1 | SORT AGGREGATE | | 1 | 1 | 1 |  
| 2 | NESTED LOOPS | | 1 | 1015K | 1015K |  
|* 3 | TABLE ACCESS FULL | CALENDARS | 1 | 31 | 31 |  
|* 4 | INDEX RANGE SCAN | PERSONS_N1 | 31 | 32768 | 1015K |  
-----
```

关联列

十二星座

- 多少人是金牛座的?
- $12M * (1 / 12) \sim 1M$

Id	Operation	Name	Starts	E-Rows	A-Rows
1	SORT AGGREGATE		1	1	1
2	NESTED LOOPS		1	1015K	1015K
* 3	TABLE ACCESS FULL	CALENDARS	1	31	31
* 4	INDEX RANGE SCAN	PERSONS_N1	31	32768	1015K

关联列

没有扩展统计值

- 多少人符合
 - 生日在五月?
 - 和
 - 是金牛座?
- $12M * (1 / 12) * (1 / 12) \sim 80K \dots$ 但是实际 **688K**

```
-----  
| Id | Operation | Name | Starts | E-Rows | A-Rows |  
-----  
| 1 | SORT AGGREGATE | | 1 | 1 | 1 |  
| 2 | NESTED LOOPS | | 1 | 86274 | 688K |  
|* 3 | TABLE ACCESS FULL | CALENDARS | 1 | 3 | 21 |  
|* 4 | INDEX RANGE SCAN | PERSONS_N1 | 21 | 32768 | 688K |  
-----
```

关联列

有扩展统计值

- 多少人符合
 - 生日在五月?
 - 和
 - 是金牛座?
- $12M * (1 / 24) \sim 500K$

```
-----  
| Id | Operation | Name | Starts | E-Rows | A-Rows |  
-----  
| 1 | SORT AGGREGATE | | 1 | 1 | 1 |  
| 2 | NESTED LOOPS | | 1 | 498K | 688K |  
|* 3 | TABLE ACCESS FULL | CALENDARS | 1 | 15 | 21 |  
|* 4 | INDEX RANGE SCAN | PERSONS_N1 | 21 | 32768 | 688K |  
-----
```

关联列

没有扩展统计值和柱状图

- 多少人符合
 - 生日在五月?
 - 和
 - 是金牛座?
- $12M * (21 / 365) \sim 688K!$

```
-----  
| Id | Operation | Name | Starts | E-Rows | A-Rows |  
-----  
| 1 | SORT AGGREGATE | | 1 | 1 | 1 |  
| 2 | NESTED LOOPS | | 1 | 688K | 688K |  
|* 3 | TABLE ACCESS FULL | CALENDARS | 1 | 21 | 21 |  
|* 4 | INDEX RANGE SCAN | PERSONS_N1 | 21 | 32768 | 688K |  
-----
```

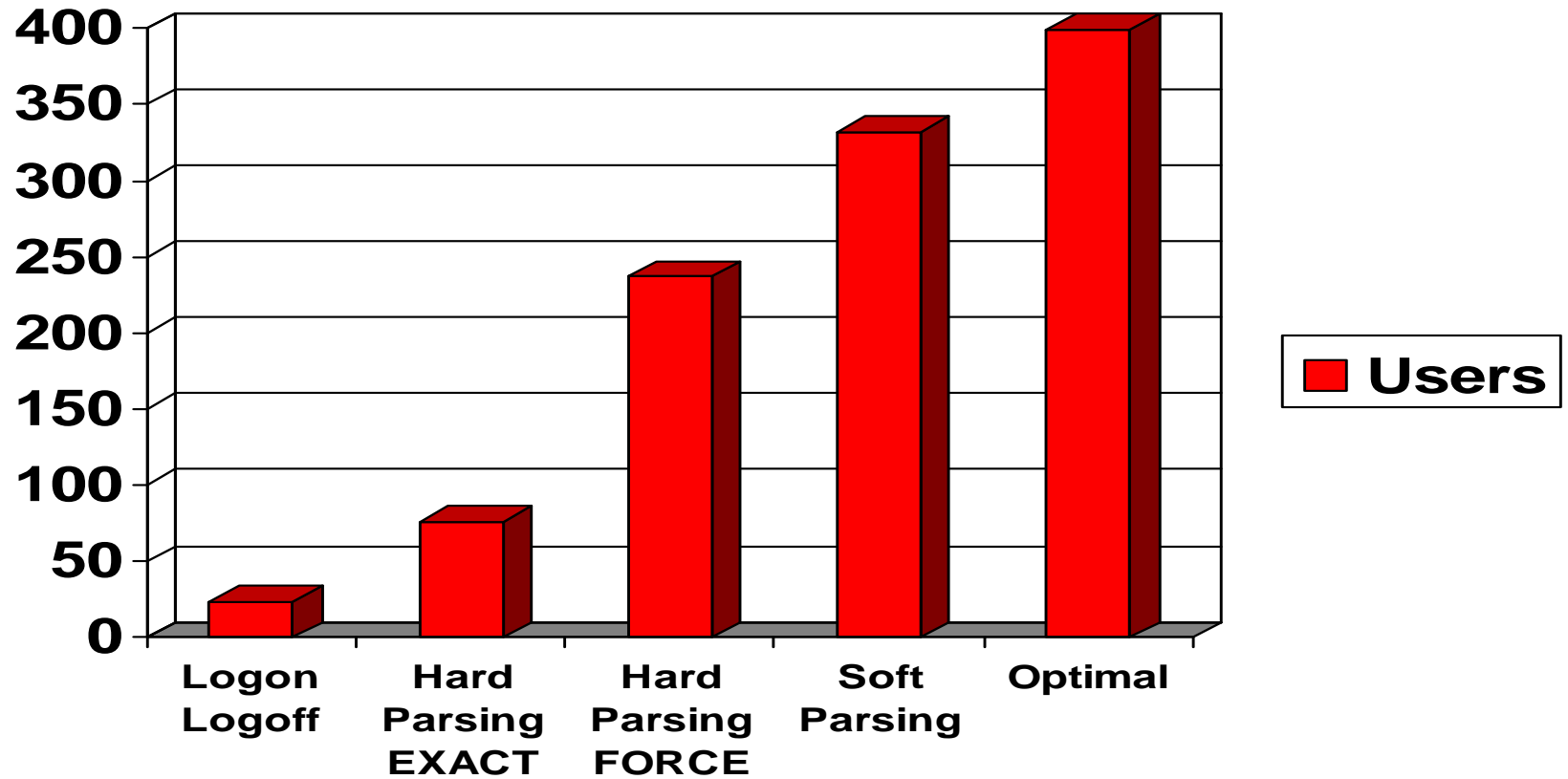
会话管理

- 大量的数据库连接的对性能的影响
 - 在一台机器上超过5000进程会使机器变的不稳定
 - 在实验室很容易模拟大量的数据库连接，但是在生产系统上要使用则非常困难
- logon/logoff对性能的影响
 - 进程的建立和撤销开销非常大
 - 连接和去连接到共享内存的开销也非常大
- 中间件的设计和配置是关键
 - 在设计上要避免在所以连接都忙的时候增加连接数，要设计连接获得的竞争条件

光标管理

- 考虑非共享SQL的影响
- 在OLTP应该中使用绑定变量

错误会话和关闭管理的开销



容量规划练习 (I/O)

- 容量规划
- IOPS规划
- 带宽规划

容量估算

- 以一个**300G**的数据库为例
 - 允许在线全备份
 - 允许使用归档日志
 - 允许使用闪回日志
 - 允许系统中期的增长
- **Let's say the accountant says 1 TByte**
- 系统设计的弹性
 - RAID1 8 @ 300 GByte
 - RAID5 5 @ 300 Gbyte

IOPS大小估算

- 同样的数据库
- OLTP和/或者批处理
- 考虑如下工作负荷
 - 20 TPS @ 每个交易50个读
 - 在一个批处理系统中每秒2000个物理读
- 采用比较高的配置值
 - 除以“Real World”安全比率
 - 2000 / 30 ~ 64 disks
- 增加系统的弹性
 - RAID1 128 disks
 - RAID5 80 disks

带宽估算

- 相同的数据库
- 完全都是数据仓库
- 目标是处理2000 MByte/s 的数据
- 除以“Real World” 安全比率
 - 2000 / 20 ~ 100 disks
- 使用弹性配置
 - RAID1 200 disks
 - RAID5 125 disks
- 你可以在SAN系统上重复练习故事
 - # Disk Arrays
 - # SAN Switches
 - # Host Bus Adapters (HBA)
 - 记住使用Bytes而不是bits计算

估算的结论

	容量	IOPS	带宽
RAID1	8	128	200
RAID5	5	80	125

所以数据库工程师应该知道和理解的数字

- 存储
 - 延迟
 - Milliseconds 0.001 s
 - 带宽
 - Gigabytes per second 1,000,000,000 bytes/s
- Network Components网络组件
 - 延迟
 - Milliseconds 0.001 s
 - 带宽
 - Gigabits per second 1,000,000,000 bits/s
- Memory内存
 - 延迟
 - Nanoseconds 0.000,000,001 s
 - Microseconds 0.000,001 s

提问时间