# PostgreSQL,Greenplum基准测试和最佳实践

digoal

# 自我介绍

PGer

乐于分享，撰写技术类文章2000余篇。

http://blog.163.com/digoal@126/

- PostgreSQL 中国社区发起人之一。
- PostgreSQL 中国社区PG大学发起人之一。
- DBA+社群联合发起人之一。
- 数十项数据库，网络相关专利。

- 曾就职于斯凯网络，负责数据库部门。主导了集团数据库系统、存储、主机、操作系统，多IDC的架构设计和部署。数据库的HA、容灾、备份、恢复、分布式、数据仓库架构设计。数据库管理和开发的标准化体系建立。在公司上市前使用PostgreSQL完成去O，并顺利通过SOX审计。
- 现就职于阿里巴巴，AliCloudDB RDS PG内核组。

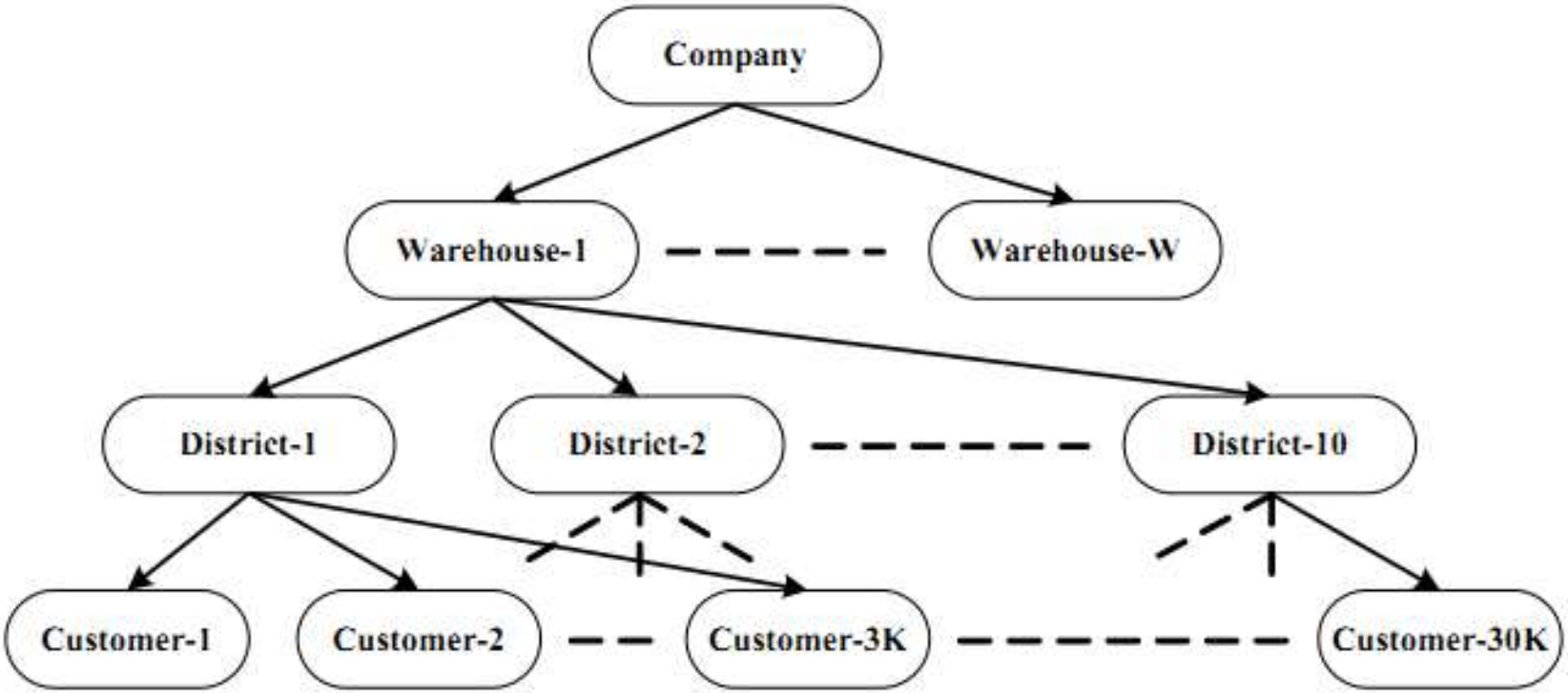
# 目录

- TPC-C 介绍
- TPC-H 介绍
- benchmarkSQL for TPC-C
- dbgen & pg-tpch for TPC-H
- PostgreSQL 最佳实践
- Greenplum 最佳实践
- tpc.org 参考数据

# TPC-C介绍

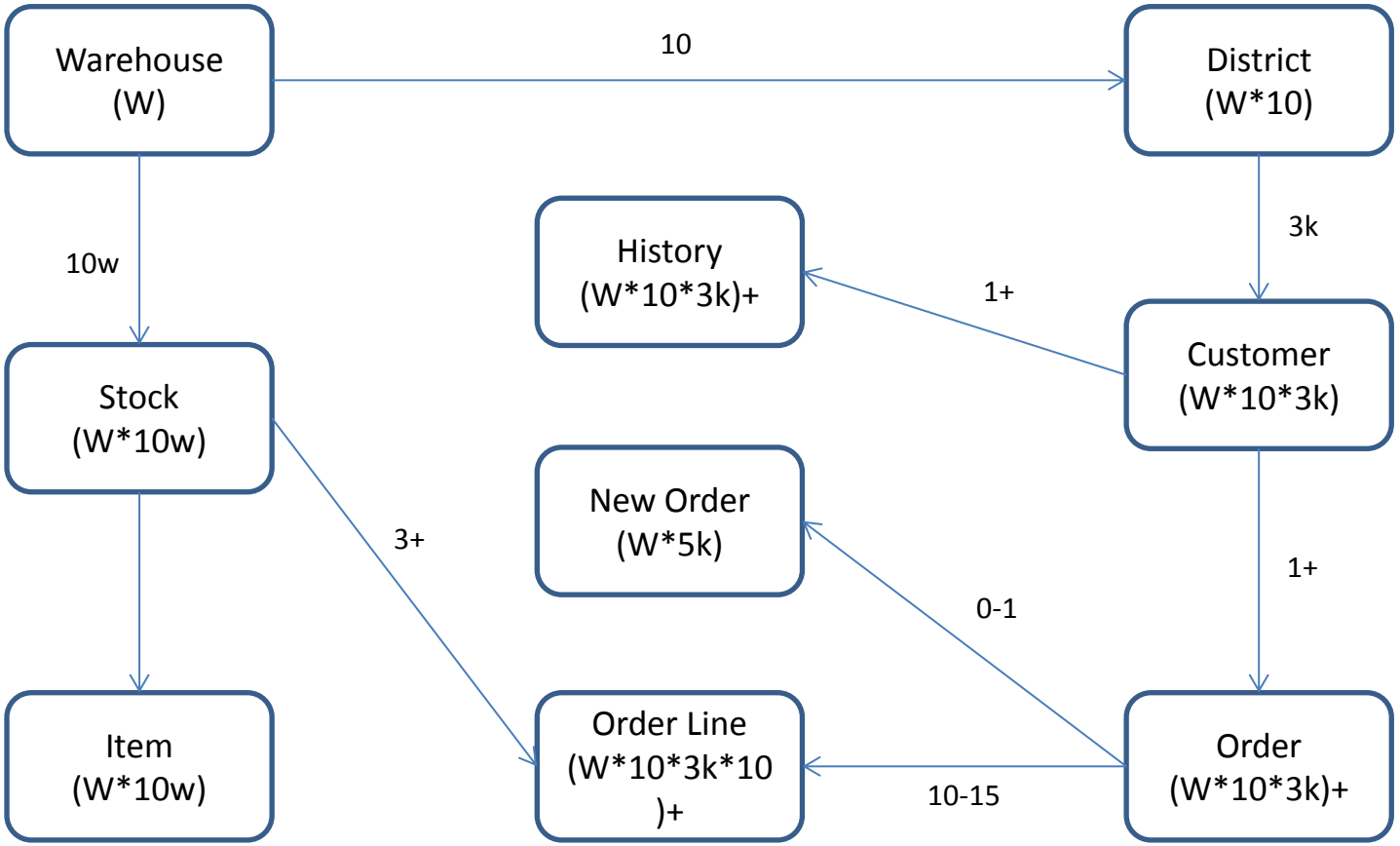
- 数据关系
- 公司 -> 仓库(W)
- 仓库 -> 销售点(10)
- 销售点 -> 客户(3000)

- W是生成测试数据时用户可以指定的



# TPC-C

- 数据关系


Warehouse
(W)
10
District
(W*10)
10w
3k
History
(W*10*3k)+
1+
Customer
(W*10*3k)
Stock
(W*10w)
New Order
(W*5k)
3+
1+
0-1
Item
(W*10w)
Order Line
(W*10*3k*10
)+
Order
(W*10*3k)+
10-15

# 新建订单

- **事务内容：对于任意一个客户端,从固定的仓库随机选取 5-15 件商品,创建新订单.其中1%的订单要由假想的用户操作失败而回滚。**
- **45%**

```
"SELECT c_discount, c_last, c_credit, w_tax" +
"  FROM benchmarksql.customer, benchmarksql.warehouse" +
" WHERE w_id = ? AND w_id = c_w_id" +
" AND c_d_id = ? AND c_id = ?");

"SELECT d_next_o_id, d_tax FROM benchmarksql.district" +
" WHERE d_id = ? AND d_w_id = ? FOR UPDATE");

"INSERT INTO benchmarksql.NEW_ORDER (no_o_id, no_d_id, no_w_id) " +
"VALUES ( ?, ?, ?)");

"UPDATE benchmarksql.district SET d_next_o_id = d_next_o_id + 1 " +
" WHERE d_id = ? AND d_w_id = ?");

"INSERT INTO benchmarksql.OORDER " +
" (o_id, o_d_id, o_w_id, o_c_id, o_entry_d, o_ol_cnt, o_all_local)" +
" VALUES (?, ?, ?, ?, ?, ?, ?)");

"SELECT i_price, i_name , i_data FROM benchmarksql.item WHERE i_id = ?");

"SELECT s_quantity, s_data, s_dist_01, s_dist_02, s_dist_03, s_dist_04, s_dist_05, " +
"       s_dist_06, s_dist_07, s_dist_08, s_dist_09, s_dist_10" +
" FROM benchmarksql.stock WHERE s_i_id = ? AND s_w_id = ? FOR UPDATE");

"UPDATE benchmarksql.stock SET s_quantity = ? , s_ytd = s_ytd + ?, s_remote_cnt = s_remote_cnt + ? " +
" WHERE s_i_id = ? AND s_w_id = ?");

"INSERT INTO benchmarksql.order_line (ol_o_id, ol_d_id, ol_w_id, ol_number, ol_i_id, ol_supply_w_id," +
"  ol_quantity, ol_amount, ol_dist_info) VALUES (?,?,?,?,?,?,?,?,?)");
```

# 支付

- 事务内容：对于任意一个客户端,从固定的仓库随机选取一个辖区及其内用户,采用随机的金额支付一笔订单,并作相应历史纪录.
- **43%**

```
"UPDATE benchmarksql.warehouse SET w_ytd = w_ytd + ?  WHERE w_id = ? ");

"SELECT w_street_1, w_street_2, w_city, w_state, w_zip, w_name" +
" FROM benchmarksql.warehouse WHERE w_id = ?");

"UPDATE benchmarksql.district SET d_ytd = d_ytd + ? WHERE d_w_id = ? AND d_id = ?");

"SELECT d_street_1, d_street_2, d_city, d_state, d_zip, d_name" +
" FROM benchmarksql.district WHERE d_w_id = ? AND d_id = ?");

"SELECT count(*) AS namecnt FROM benchmarksql.customer " +
" WHERE c_last = ?  AND c_d_id = ? AND c_w_id = ?");

"SELECT c_first, c_middle, c_id, c_street_1, c_street_2, c_city, c_state, c_zip," +
"      c_phone, c_credit, c_credit_lim, c_discount, c_balance, c_since " +
"  FROM benchmarksql.customer WHERE c_w_id = ? AND c_d_id = ? AND c_last = ? " +
"ORDER BY c_w_id, c_d_id, c_last, c_first ");

"SELECT c_first, c_middle, c_last, c_street_1, c_street_2, c_city, c_state, c_zip," +
"      c_phone, c_credit, c_credit_lim, c_discount, c_balance, c_since " +
"  FROM benchmarksql.customer WHERE c_w_id = ? AND c_d_id = ? AND c_id = ?");

"SELECT c_data FROM benchmarksql.customer WHERE c_w_id = ? AND c_d_id = ? AND c_id = ?");

"UPDATE benchmarksql.customer SET c_balance = ?, c_data = ? " +
" WHERE c_w_id = ? AND c_d_id = ? AND c_id = ?");

"UPDATE benchmarksql.customer SET c_balance = ? WHERE c_w_id = ? AND c_d_id = ? AND c_id = ?");

"INSERT INTO benchmarksql.history (h_c_d_id, h_c_w_id, h_c_id, h_d_id, h_w_id, h_date, h_amount, h_data) " +
" VALUES (?,?,?,?,?,?,?,?)");
```

# 订单状态查询

- 事务内容：对于任意一个客户端,从固定的仓库随机选取一个辖区及其内用户,读取其最后一条订单,显示订单内每件商品的状态.
- **4%**

```
"SELECT count(*) AS namecnt FROM benchmarksql.customer" +
 " WHERE c_last = ? AND c_d_id = ? AND c_w_id = ?");

 "SELECT c_balance, c_first, c_middle, c_id FROM benchmarksql.customer" +
 " WHERE c_last = ?" +
 " AND c_d_id = ?" +
 " AND c_w_id = ?" +
 " ORDER BY c_w_id, c_d_id, c_last, c_first");

 "SELECT c_balance, c_first, c_middle, c_last" +
 " FROM benchmarksql.customer" +
 " WHERE c_id = ?" +
 " AND c_d_id = ?" +
 " AND c_w_id = ?");

 "SELECT MAX(o_id) AS maxorderid FROM benchmarksql.oorder" +
 " WHERE o_w_id = ?" +
 " AND o_d_id = ?" +
 " AND o_c_id = ?");

 "SELECT o_carrier_id, o_entry_d" +
 " FROM benchmarksql.oorder" +
 " WHERE o_w_id = ?" +
 " AND o_d_id = ?" +
 " AND o_c_id = ?" +
 " AND o_id = ?");

 "SELECT ol_i_id, ol_supply_w_id, ol_quantity," +
 " ol_amount, ol_delivery_d" +
 " FROM benchmarksql.order_line" +
 " WHERE ol_o_id = ?" +
 " AND ol_d_id =?" +
 " AND ol_w_id = ?");
```

# 发货

- 事务内容：对于任意一个客户端,随机选取一个发货包,更新被处理订单的用户余额,并把该订单从新订单中删除.
- **4%**

```
"SELECT no_o_id FROM benchmarksql.new_order WHERE no_d_id = ?" +
" AND no_w_id = ?" +
" ORDER BY no_o_id ASC");

"DELETE FROM benchmarksql.new_order" +
" WHERE no_d_id = ?" +
" AND no_w_id = ?" +
" AND no_o_id = ?");

"SELECT o_c_id" +
" FROM benchmarksql.oorder" +
" WHERE o_id = ?" +
" AND o_d_id = ?" +
" AND o_w_id = ?");

"UPDATE benchmarksql.oorder SET o_carrier_id = ?" +
" WHERE o_id = ?" +
" AND o_d_id = ?" +
" AND o_w_id = ?");

"UPDATE benchmarksql.order_line SET ol_delivery_d = ?" +
" WHERE ol_o_id = ?" +
" AND ol_d_id = ?" +
" AND ol_w_id = ?");

"SELECT SUM(ol_amount) AS ol_total" +
" FROM benchmarksql.order_line" +
" WHERE ol_o_id = ?" +
" AND ol_d_id = ?" +
" AND ol_w_id = ?");

"UPDATE benchmarksql.customer SET c_balance = c_balance + ?" +
", c_delivery_cnt = c_delivery_cnt + 1" +
" WHERE c_id = ?" +
" AND c_d_id = ?" +
" AND c_w_id = ?");
```

# 库存状态查询

- 事物内容：对于任意一个客户端,从固定的仓库和辖区随机选取最后 20 条订单,查看订单中所有的货物的库存,计算并显示所有库存低于随机生成域值的商品数量.
- 4%
- "SELECT d_next_o_id" +
- " FROM benchmarksql.district" +
- " WHERE d_w_id = ?" +
- " AND d_id = ?");

- "SELECT COUNT(DISTINCT (s_i_id)) AS stock_count" +
- " FROM benchmarksql.order_line, benchmarksql.stock" +
- " WHERE ol_w_id = ?" +
- " AND ol_d_id = ?" +
- " AND ol_o_id < ?" +
- " AND ol_o_id >= ? - 20" +
- " AND s_w_id = ?" +
- " AND s_i_id = ol_i_id" +
- " AND s_quantity < ?");

# TPC-H介绍

- 模拟决策支持系统中的数据库操作，测试数据库系统复杂查询的响应时间，以每小时执行的查询数(TPC-H QphH) 作为度量指标

- 22 个查询语句
  - TPC- H 测试围绕22 个SELECT 语句展开，每个SELECT严格定义，遵守SQL-92语法，并且不允许用户修改。
  - 标准中从4 个方面定义每个SELECT 语句，即商业问题、SELECT 的语法、参数和查询确认。
  - 这些SELECT 语句的复杂程度超过大多数实际的OLTP 应用，一个SELECT 执行时间少则几十秒，多则达15 小时以上，22 个查询语句执行一遍需数个小时。

# TPC-H介绍

- 2 个更新操作
  - 为了逼真地模拟数据仓库的实际应用环境，在22个查询执行的同时，还有一对更新操作RF1 和RF2并发地执行。
  - RF1向Order 表和Lineitem 表中插入原行数的0.1%的新行，模拟新销售业务的数据加入到数据库中；
  - RF2 从Order 表和Lineitem表中删除等量与RF1 增加的数据，模拟旧的销售数据被淘汰。
  - RF1 和RF2 的执行必须保证数据库的ACID 约束，并保持测试前后的数据库中的数据量不变。
  - 更新操作除输出成功或失败信息外，不产生其它输出信息

# TPC-H介绍

- 3 个测试
  - TPC-H 测试分解为3 个子测试：数据装载测试、Power 测试和Throughput 测试。
  - 建立测试数据库的过程被称为装载数据，装载测试是为测试DBMS 装载数据的能力。装载测试是第一项测试，测试装载数据的时间，这项操作非常耗时。
  - Power 测试是在数据装载测试完成后，数据库处于初始状态，未进行其它任何操作，特别是缓冲区还没有被测试数据库的数据，被称为raw查询。Power 测试要求22 个查询顺序执行1 遍，同时执行一对RF1 和RF2 操作。
  - 最后进行Throughput 测试，也是最核心和最复杂的测试，它更接近于实际应用环境，与Power 测试比对SUT 系统的压力有非常大的增加，有多个查询语句组，同时有一对RF1 和RF2 更新流。

# TPC-H介绍

- TPC-H 标准的附录D，有两组ANSI C 语言源程序包，即DBGEN 和QGEN。
  - DBGEN 用于产生被测试数据，用户通过命令行参数控制执行结果。
  - QGEN 用于生产测试所需要的22 个SELECT、RF1 和RD2 两个更新操作。

# TPC-H介绍

- 数据量规定
  - 由于数据量的大小对查询速度有直接的影响，TPC- H 标准对数据库系统中的数据量有严格、明确的规定。
  - 用SF 描述数据量，1SF 对应1 GB 单位，SF 由低到高依次是1、10、30、100、300、1 000、3 000、10 000。需要强调，SF 规定的数据量只是8个基本表的数据量，不包括索引和临时表。
  - 从TPC- H 测试全程来看，需要的数据存储空较大，一般包括有基本表、索引、临时表、数据文件和备份文件，基本表的大小为x；索引和临时空间的经验值为3-5 位，取上限5x；
  - DBGEN产生的数据文件的大小为x；备份文件大小为x；总计需要的存储空间为8x。
  - 就是说SF=1，需要准备8 倍，即8 GB 存储空间，才能顺利地进行测试。

# TPC-H介绍

- 关系图

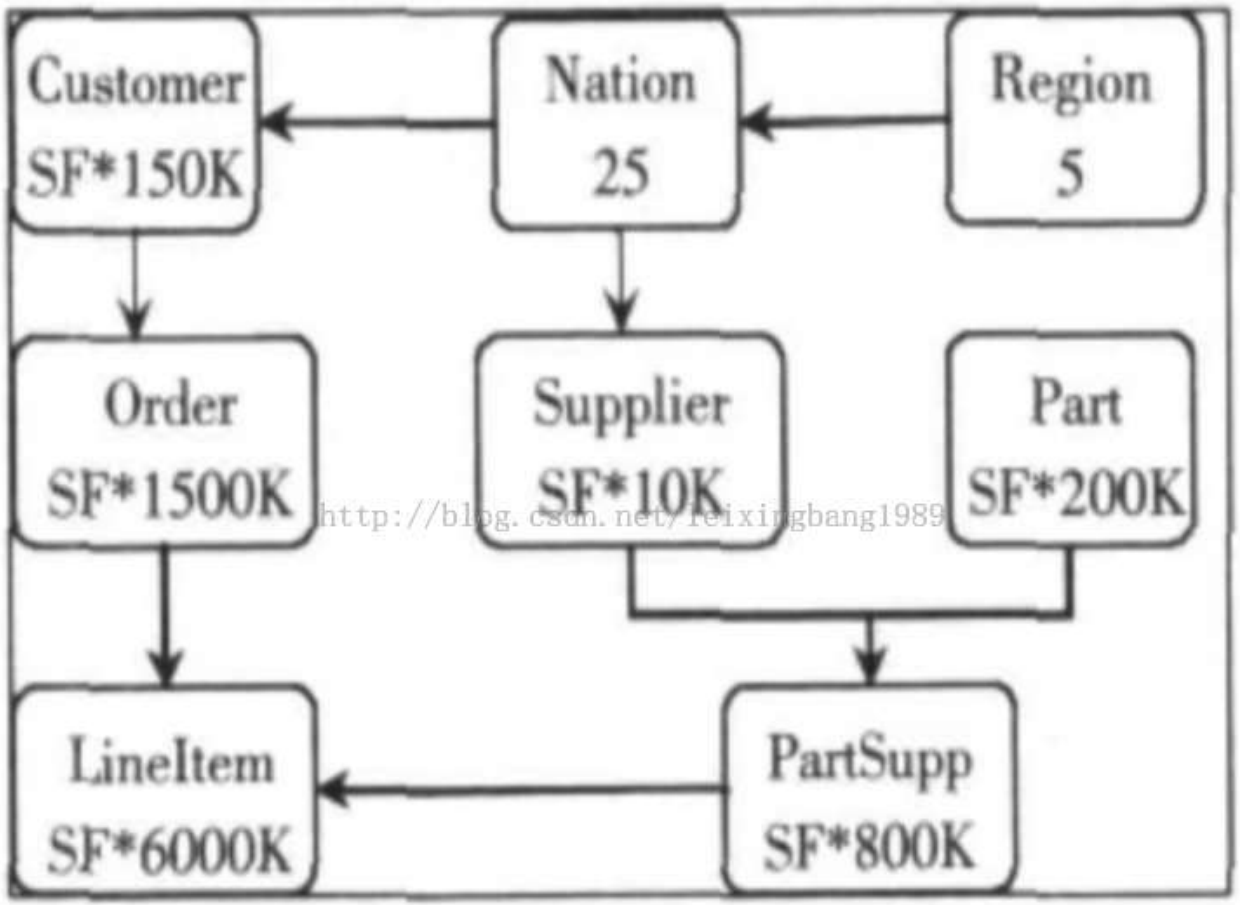
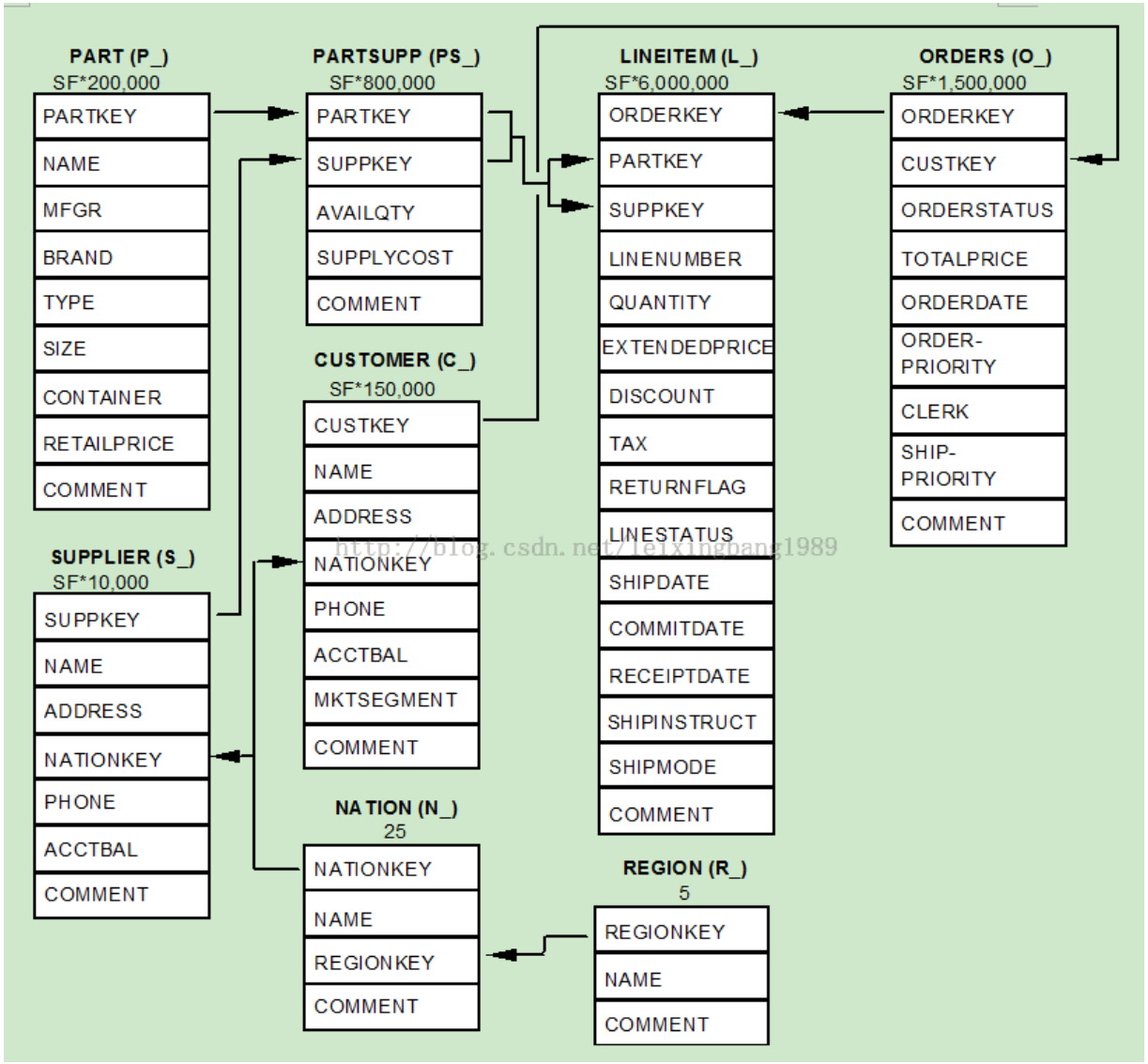



图 1　TPC-H 测试的数据库模式

# TPC-H介绍

- 关系图


PART (P_)
SF*200,000
PARTKEY
NAME
MFGR
BRAND
TYPE
SIZE
CONTAINER
RETAILPRICE
COMMENT
PARTSUPP (PS_)
SF*800,000
PARTKEY
SUPPKEY
AVAILQTY
SUPPLYCOST
COMMENT
LINEITEM (L_)
SF*6,000,000
ORDERKEY
PARTKEY
SUPPKEY
LINENUMBER
QUANTITY
EXTENDEDPRICE
DISCOUNT
TAX
RETURNFLAG
LINESTATUS
SHIPDATE
COMMITDATE
RECEIPTDATE
SHIPINSTRUCT
SHIPMODE
COMMENT
ORDERS (O_)
SF*1,500,000
ORDERKEY
CUSTKEY
ORDERSTATUS
TOTALPRICE
ORDERDATE
ORDER-PRIORITY
CLERK
SHIP-PRIORITY
COMMENT
CUSTOMER (C_)
SF*150,000
CUSTKEY
NAME
ADDRESS
NATIONKEY
PHONE
ACCTBAL
MKTSEGMENT
COMMENT
SUPPLIER (S_)
SF*10,000
SUPPKEY
NAME
ADDRESS
NATIONKEY
PHONE
ACCTBAL
COMMENT
NATION (N_)
25
NATIONKEY
NAME
REGIONKEY
COMMENT
REGION (R_)
5
REGIONKEY
NAME
COMMENT
http://blog.csdn.net/leixinghang1989

# TPC-C基准测试

- BenchmarkSQL support Oracle,PostgreSQL
- sysbench support MySQL,Oracle,PostgreSQL

- 下载benchmarksql
- http://sourceforge.net/projects/benchmarksql/

- 下载安装 JDK7
- http://www.oracle.com/technetwork/cn/java/javase/downloads/jdk7-downloads-1880260.html
- wget http://download.oracle.com/otn-pub/java/jdk/7u79-b15/jdk-7u79-linux-x64.rpm
- rpm -ivh jdk-7u79-linux-x64.rpm

- 检查包安装位置(使用rpm安装时也可以直接指定位置)
- rpm -ql jdk
- ...
- /usr/java/jdk1.7.0_79/bin/java
- ...

- $ export JAVA_HOME=/usr/java/jdk1.7.0_79
- $ export PATH=$JAVA_HOME/bin:$PATH
- $ export CLASSPATH=.:$CLASSPATH

- $ wget https://jdbc.postgresql.org/download/postgresql-9.4.1207.jre7.jar
- $ mv postgresql-9.4.1207.jre7.jar benchmarksql-4.1.0/lib/

# TPC-C基准测试

- 配置benchmarksql，使用新的postgresql java驱动
- $ vi runBenchmark.sh
- java -cp .:../lib/postgresql-9.4.1207.jre7.jar:../lib/log4j-1.2.17.jar:../lib/apache-log4j-extras-1.1.jar:../dist/BenchmarkSQL-4.1.jar -Dprop=$1 jTPCC

- $ vi runLoader.sh
- java -cp .:../lib/postgresql-9.4.1207.jre7.jar:../dist/BenchmarkSQL-4.1.jar -Dprop=$1 LoadData $2 $3 $4 $5

- $ vi runSQL.sh
- myCP="../lib/postgresql-9.4.1207.jre7.jar"
- myCP="$myCP:../dist/BenchmarkSQL-4.1.jar"

- myOPTS="-Dprop=$1"
- myOPTS="$myOPTS -DcommandFile=$2"

- java -cp .:$myCP $myOPTS ExecJDBC

# TPC-C基准测试

- 修改log4j，减少日志打印量。priority改成info，只输出最终结果，不输出产生订单的日志。

- 编辑连接配置和压测配置。
- 1000 个仓库，约5亿数据量。

- 修改配置比例
- $ vi props.pg
- driver=org.postgresql.Driver
- conn=jdbc:postgresql://localhost:1921/postgres
- user=postgres
- password=123
- warehouses=1000
- terminals=96
- //To run specified transactions per terminal- runMins must equal zero
- runTxnsPerTerminal=0
- //To run for specified minutes- runTxnsPerTerminal must equal zero
- runMins=1
- //Number of total transactions per minute
- limitTxnsPerMin=0

- newOrderWeight=45
- paymentWeight=43
- orderStatusWeight=4
- deliveryWeight=4
- stockLevelWeight=4

# TPC-C基准测试

- 配置postgres用户默认搜索路径
- $ psql
- psql (9.5.0)
- Type "help" for help.
- postgres=# alter role postgres set search_path='benchmarksql','public';

- 创建用于存放生成CSV的目录
- $ mkdir /u02/digoal/soft_bak/benchcsv
- 修改sqlTableCopies，指定目录
- $ vi sqlTableCopies

# TPC-C基准测试

- 建立表结构
- $ cd benchmarksql-4.1.0/run
- $ ./runSQL.sh props.pg sqlTableCreates

- 生成CSV
- $ ./runLoader.sh props.pg numWarehouses 1000 fileLocation /u02/digoal/soft_bak/benchcsv/

- 导入数据库
- $ ./runSQL.sh props.pg sqlTableCopies

- 创建约束和索引
- $ ./runSQL.sh props.pg sqlIndexCreates

# TPC-C基准测试

- 备份
- $ pg_dump -f /u02/digoal/soft_bak/benchmarksql.dmp -F c -n benchmarksql postgres

- 压测：
- nohup ./runBenchmark.sh props.pg >/dev/null 2>./errrun.log &

- 观察
- perf top
  - CPU时间占比
- iostat -x
  - 块设备使用率，平均IO响应时间，队列大小，平均IO大小
- top

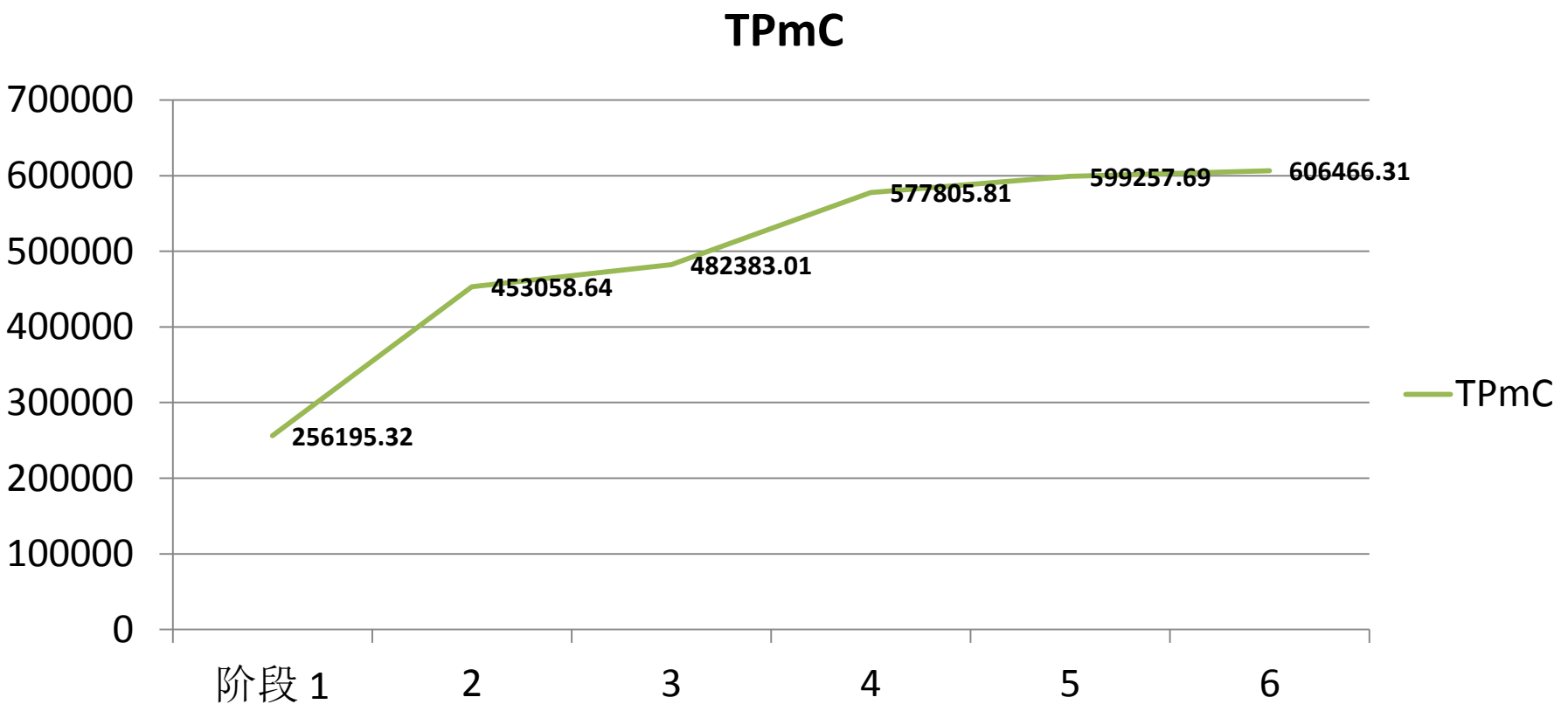
- dstat
  - CPU比例，IO，。。。
- pg_stat_statements
  - SQL请求耗时, IO调用, CPU耗时

# TPC-C基准测试

- benchmarksql 支持多个SCHEMA
- http://blog.163.com/digoal@126/blog/static/16387704020160102183822l/

# TPC-C基准测试

- 优化阶段数据



# PostgreSQL最佳实践

- 存储层
  - 条带大小，条带宽度
  - 对齐
- 块设备规划
  - pg_xlog、$PGDATA、user data tablespace、user idx tablespace
  - 分区位置对齐，卷DATA起始位置对齐，文件系统、卷条带对齐

# PostgreSQL最佳实践

- 操作系统
  - vm.zone_reclaim_mode=0 # 禁用 numa, 或者在vmlinux中禁止.
  - vm.swappiness = 0 # 关闭交换分区
  - net.core.rmem_max = 4194304 # The maximum receive socket buffer size in bytes
  - net.core.wmem_max = 4194304 # The maximum send socket buffer size in bytes.
  - net.core.rmem_default = 262144 # The default setting of the socket receive buffer in bytes.
  - net.core.wmem_default = 262144 # The default setting (in bytes) of the socket send buffer.
  - vm.dirty_background_bytes = 102400000 # 系统脏页到达这个值，系统后台刷脏页调度进程 pdflush（或其他） 自动将(dirty_expire_centisecs/100）秒前的脏页刷到磁盘
  - vm.dirty_expire_centisecs = 6000 # 比这个值老的脏页，将被刷到磁盘。6000表示60秒。
  - vm.dirty_writeback_centisecs = 50 # pdflush（或其他）后台刷脏页进程的唤醒间隔， 50表示0.5秒。
  - vm.dirty_ratio = 80 # 如果系统进程刷脏页太慢，使得系统脏页超过内存 80 % 时，则用户进程如果有写磁盘的操作（如fsync, fdatasync等调用），则需要主动把系统脏页刷出。
  - vm.nr_hugepages = 102352 # 大页数量，乘以/proc/meminfo Hugepagesize就是支持大页的内存数量。
  - vm.overcommit_memory = 0 # 在分配内存时，允许少量over malloc
  - vm.overcommit_ratio = 90 # 当overcommit_memory = 2 时，用于参与计算允许指派的内存大小。

- grub: numa=off elevator=deadline

# PostgreSQL最佳实践

- 数据库参数
  - max_connections = 300 # (change requires restart)
  - unix_socket_directories = '.' # comma-separated list of directories
  - shared_buffers = 194GB # 尽量用数据库管理内存，减少双重缓存，提高使用效率
  - huge_pages = on # on, off, or try ， 使用大页
  - work_mem = 256MB # min 64kB ， 减少外部文件排序的可能，提高效率
  - maintenance_work_mem = 2GB # min 1MB ， 加速建立索引
  - autovacuum_work_mem = 2GB # min 1MB, or -1 to use maintenance_work_mem ， 加速垃圾回收
  - dynamic_shared_memory_type = mmap # the default is the first option
  - vacuum_cost_delay = 0 # 0-100 milliseconds ， 垃圾回收不妥协，极限压力下，减少膨胀可能性
  - bgwriter_delay = 10ms # 10-10000ms between rounds ， 刷shared buffer脏页的进程调度间隔，尽量高频调度，减少用户进程申请不到内存而需要主动刷脏页的可能（导致RT升高）。
  - bgwriter_lru_maxpages = 1000 # 0-1000 max buffers written/round , 一次最多刷多少脏页
  - bgwriter_lru_multiplier = 10.0 # 0-10.0 multipler on buffers scanned/round 一次扫描多少个块，上次刷出脏页数量的倍数
  - effective_io_concurrency = 2 # 1-1000; 0 disables prefetching ， 执行节点为bitmap heap scan时，预读的块数。从而
  - wal_level = minimal # minimal, archive, hot_standby, or logical ， 如果现实环境，建议开启归档。
  - synchronous_commit = off # synchronization level; ， 异步提交
  - wal_sync_method = open_sync # the default is the first option ， 因为没有standby，所以写xlog选择一个支持O_DIRECT的fsync方法。

# PostgreSQL最佳实践

- 数据库参数
  - full_page_writes = off  # recover from partial page writes ， 生产中，如果有增量备份和归档，可以关闭，提高性能。
  - wal_buffers = 1GB  # min 32kB, -1 sets based on shared_buffers ， wal buffer大小，如果大量写wal buffer等待，则可以加大。
  - wal_writer_delay = 10ms  # 1-10000 milliseconds  wal buffer调度间隔，和bg writer delay类似。
  - commit_delay = 20  # range 0-100000, in microseconds ， 分组提交的等待时间
  - commit_siblings = 9  # range 1-1000 , 有多少个事务同时进入提交阶段时，就触发分组提交。
  - checkpoint_timeout = 55min  # range 30s-1h  时间控制的检查点间隔。
  - max_wal_size = 320GB  #  2个检查点之间最多允许产生多少个XLOG文件
  - checkpoint_completion_target = 0.99  # checkpoint target duration, 0.0 - 1.0 ， 平滑调度间隔，假设上一个检查点到现在这个检查点之间产生了100个XLOG，则这次检查点需要在产生100*checkpoint_completion_target个XLOG文件的过程中完成。PG会根据这些值来调度平滑检查点。
  - random_page_cost = 1.0  # same scale as above , 离散扫描的成本因子，本例使用的SSD IO能力足够好
  - effective_cache_size = 240GB  # 可用的OS CACHE
  - log_destination = 'csvlog'  # Valid values are combinations of
  - logging_collector = on  # Enable capturing of stderr and csvlog
  - log_truncate_on_rotation = on  # If on, an existing log file with the
  - update_process_title = off
  - track_activities = off
  - autovacuum = on  # Enable autovacuum subprocess?  'on'
  - autovacuum_max_workers = 4 # max number of autovacuum subprocesses  ， 允许同时有多少个垃圾回收工作进程。
  - autovacuum_naptime = 6s  # time between autovacuum runs  ， 自动垃圾回收探测进程的唤醒间隔
  - autovacuum_vacuum_cost_delay = 0  # default vacuum cost delay for  ， 垃圾回收不妥协

# PostgreSQL最佳实践

- 编译器, FLAG
  - newest clang
  - CC=/digoal/llvm/bin/clang CFLAGS="-O2 -fstrict-enums"
- 数据库编译参数
- ./configure --prefix=/u02/digoal/soft_bak/pgsql9.5 --with-pgport=1921 --with-perl --with-python --with-tcl --with-openssl --with-pam --with-ldap --with-libxml --with-libxslt --enable-thread-safety make world -j 32 make install-world -j 32

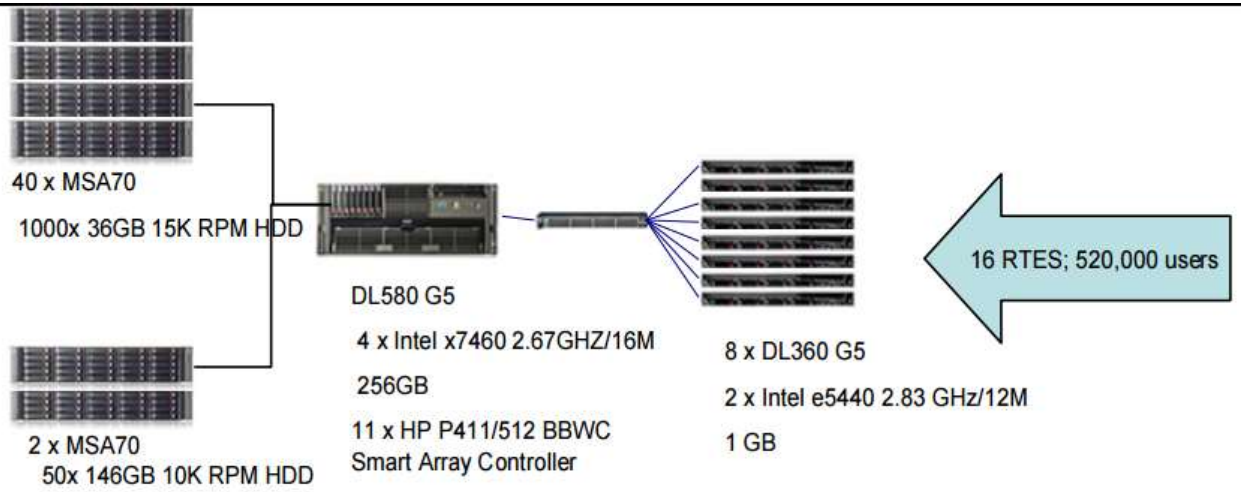
# PostgreSQL最佳实践

- 硬件采购前根据实际的业务逻辑进行基准测试，找出硬件木桶短板，均衡配置CPU,MEM,disk,netdev。
- 备份和恢复
- 审计
- 安全
- 日常维护
- 健康监控

# TPC-C参考值

- http://www.tpc.org/tpcc/results/tpcc_results.asp?orderby=dbms

| | | | | |
|---|---|---|---|---|
|  | **HP ProLiant DL580 G5 2.67 GHz 16MB L2**<br>**C/S with 8 ProLiant DL360G5** | | **TPC-C Version 5.9**<br>Report Date<br>**January 16, 2009** | |
| Total System Cost | TPC-C Throughput | Price/Performance | Availability Date | |
| **$615,914 USD** | **639,253 tpmC** | **$0.97 USD/ tpmC** | **January 26, 2009** | |
| Processors | Database Manager | Operating System | Other Software | Number of Users |
| 4/24/24 Intel Xeon 2.67 GHz 16MB L2 cache | Oracle Database 11g Standard Edition | Oracle Enterprise Linux | Microsoft COM+ | **520,000** |




| | Server | | Each Client | |
|---|---|---|---|---|
| **System Components** | Quantity | Description | Quantity | Description |
| Processor | 4/24/24 | 4/24/24 Intel Xeon 2.67 GHz 16MB L2 cache | 1/4/4 | 2.83 GHz Intel Xeon w/ 12MB L2 Cache |
| Memory | 256 GB | 32x8GB | 2.0 | 2x1024 MB |
| Disk Controllers | 11 | HP P411 BBWC Smart Array Controller | 1 | Integrated Smart Array 400i Controller |
| Disk Drives | 50 | 146 GB 10K SFF SAS drives (log) | 1 | 36 GB 15K SAS |
| | 1000 | 36 GB 15K SFF SAS drives (data) | | |
| | 2 | 36 GB 10K SFF SAS drives (OS) | | |
| Total Storage | 43372 GB | | | |

| Response Times (in seconds) | Average | 90% | Maximum |
|---|---|---|---|
| New-Order | 0.392 | 1.250 | 33.251 |
| Payment | 0.378 | 1.236 | 28.028 |
| Order-Status | 0.391 | 1.248 | 3.224 |
| Delivery (interactive portion) | 0.345 | 1.146 | 2.997 |
| Delivery (deferred portion) | 0.023 | 0.046 | 6.079 |
| Stock-Level | 0.432 | 1.313 | 3.431 |
| Menu | 0.346 | 1.149 | 2.999 |

# HP ProLiant DL370 G6 2.93 GHz 8 MB L2

**C/S with 8 ProLiant DL360G5**

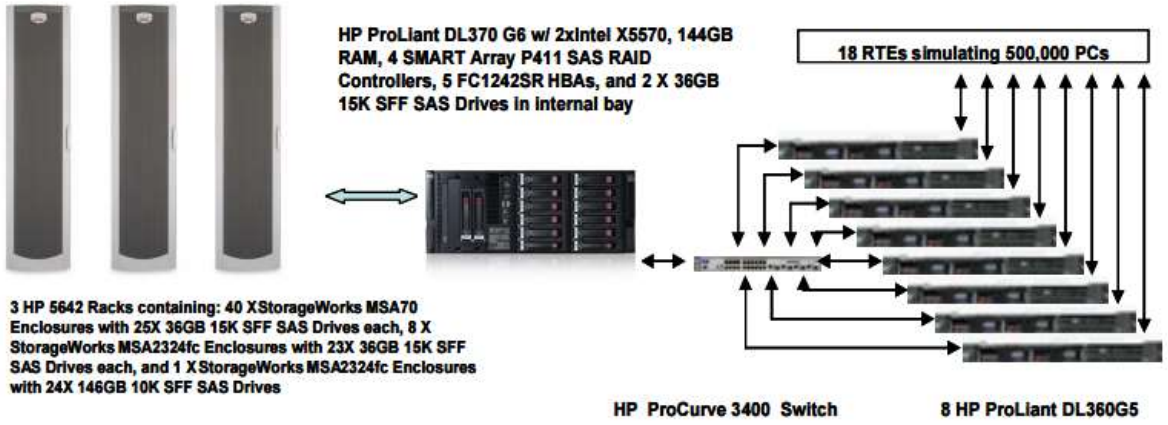
**TPC-C Version 5.10**

Report Date

**March 30, 2009**

| Total System Cost | TPC-C Throughput | Price/Performance | Availability Date |
|---|---|---|---|
| **$678,231 USD** | **631,766 tpmC** | **$1.08 USD/ tpmC** | **March 30, 2009** |

| Processors | Database Manager | Operating System | Other Software | Number of Users |
|---|---|---|---|---|
| 2/8/16 Intel Xeon X5570 2.93 GHz 8MB L2 cache | Oracle Database 11g Standard Edition One | Oracle Enterprise Linux | Microsoft COM+ | **500,000** |




| | Server | | Each Client | |
|---|---|---|---|---|
| **System Components** | Quantity | Description | Quantity | Description |
| Processor | 2/8/16 | Intel Xeon 2.93 GHz 8MB L2cache | 1/4/4 | 2.83 GHz Intel Xeon w/ 12MB L2 Cache |
| Memory | 144 GB | 18x8GB | 2.0 | 2x1024 MB |
| Disk Controllers | 4 | HP P411 BBWC Smart Array Controllers | 1 | Integrated Smart Array 400i Controller |
| | 9 | MSA2324fc | | |
| Disk Drives | 24 | 146 GB 10K SFF SAS drives (log) | 1 | 36 GB 15K SAS |
| | 1184 | 36 GB 15K SFF SAS drives (data) | | |
| | 2 | 36 GB 15 SFF SAS drives (OS) | | |
| Total Storage | 60,045 GB | | | |

| Response Times (in seconds) | Average | 90% | Maximum |
|---|---|---|---|
| New-Order | 0.177 | 0.255 | 4.209 |
| Payment | 0.156 | 0.230 | 4.170 |
| Order-Status | 0.181 | 0.258 | 1.866 |
| Delivery (interactive portion) | 0.119 | 0.170 | 0.534 |
| Delivery (deferred portion) | 0.043 | 0.077 | 3.985 |
| Stock-Level | 0.687 | 0.924 | 2.691 |
| Menu | 0.119 | 0.171 | 0.539 |

# TPC-H基准测试

- http://www.tpc.org/tpc_documents_current_versions/current_specifications.asp
- Download TPCH_Tools.zip

- $unzip TPCH_Tools.zip
- $cd tpch_2_17_0/
- $cd dbgen
- $cp makefile.suite Makefile
- $vi Makefile
- CC = gcc
- DATABASE = ORACLE
- MACHINE = LINUX
- WORKLOAD = TPCH

- $make

# TPC-H基准测试

- 使用dbgen产生一些测试数据, -s 表示scale（单位为GB, 不包括索引）:
- $./dbgen -s 100 -f

- 将测试数据转换为postgresql识别的格式，删除末尾的分隔符|
- $for i in `ls *.tbl`; do sed 's/|$//' $i > ${i/tbl/csv}; done

- 把包含csv文件的目录，软链接到/tmp/dss-data。tpch-pg脚本中一会要用到这个目录。

- $pwd
- /home/digoal/tpch/tpch_2_17_0/dbgen
- $ln -s /home/digoal/tpch/tpch_2_17_0/dbgen  /tmp/dss-data

# TPC-H基准测试

- 下载pg_tpch
- $ wget https://github.com/digoal/pg_tpch/archive/master.zip
- $ unzip master.zip
- $ cd pg_tpch-master/
- $ cd dss
- $ ls
- templates  tpch-alter.sql  tpch-create.sql  tpch-index.sql  tpch-load.sql  tpch-pkeys.sql

- 修改tpch-load.sql，对齐 JOIN 列数据类型，整型外的数字类型全部变更为float8

- 适配greenplum的语法，需要修改一下这个SQL文件。
- $cp tpch-load.sql tpch-load.sql.pg
- $vi tpch-load.sql
- COPY命令格式有问题，为了获得更好的效果，使用列存储，修改如下举例：
- ) with (APPENDONLY=true,BLOCKSIZE=2097152,ORIENTATION=COLUMN,COMPRESSTYPE=QUICKLZ,CHECKSUM=true,OIDS=false);
- COPY region FROM '/tmp/dss-data/region.csv' WITH csv DELIMITER '|';
- .....

# TPC-H基准测试

- 将pg_tpch的文件都拷贝到dbgen所在的目录：
- $cp -r pg_tpch-master/*  tpch/tpch_2_17_0/dbgen/
- $cd tpch/tpch_2_17_0/dbgen

- 创建一个queries目录，用于存放转换后的tpc-h 测试SQL。
- $mkdir dss/queries

- 使用qgen生成测试SQL。
- $for q in `seq 1 22`
- do
- DSS_QUERY=dss/templates ./qgen $q >> dss/queries/$q.sql
- sed 's/^select/explain select/' dss/queries/$q.sql > dss/queries/$q.explain.sql
- done

- 调整 tpch-alter.sql  tpch-index.sql  tpch-pkeys.sql, 不需要加FK.

- 第一次测试
- $./tpch.sh ./results postgres digoal

- 再次测试前需要修改tpch.sh, 不需要再次导入数据，创建索引等动作。

- 查看测试结果，关注results目录中 errors目录, explain目录, results目录中的数据是否正确.
- bench.log包含22条SQL的运行时间。

# Greenplum最佳实践

- 主机
  - segment个数 = cpu核心数 * 0.8
  - 块设备IO能力 和 单机segment个数的比例
  - 块设备读写带宽 和 单机segment个数的比例
  - 内存 和 单机segment个数的比例
  - 网卡 和 块设备读写带宽对齐
- 操作系统
  - numa=off , elevator=deadline
  - 块设备对齐
- 文件系统
  - xfs
  - AGcount要足够大, 条带对齐, journal盘要快
  - rw,noatime,nodiratime,allocsize=16M,inode64,nobarrier,largeio,logbsize=262144,swalloc
- 交换机
  - 所有节点在同一个交换机下
  - 采用多块网卡时，不同VLAN在不同交换机下

# Greenplum最佳实践

- 数据库
  - shared_buffers = 1024MB
  - max_fsm_pages = ?  # 主节点数据库大小 / 8K
  - max_fsm_relations = ?  # max_fsm_pages/16 - 1
  - gp_vmem_protect_limit = 7500  # MB 小于阈值
  - statement_mem = 2047000  # KB
  - gp_backup_directIO = on
  - gp_backup_directIO_read_chunk_mb = 20
  - checkpoint_segments=64
  - gp_set_read_only=off
  - gp_workfile_limit_per_segment=?GB  # 单segment大小 * 0.5
  - gp_workfile_compress_algorithm=ZLIB
  - gp_default_storage_options='appendonly=true, orientation=column'

# Greenplum最佳实践

- 备份和恢复
- 审计
- 安全
- 日常维护
- 健康监控

# TPC-H参考値

- http://www.tpc.org/tpch/results/tpch_results.asp?orderby=dbms

# 谢谢

- URL
  - https://github.com/digoal
  - http://blog.163.com/digoal@126
  - https://yq.aliyun.com/groups/29
  - https://yq.aliyun.com/groups/13

- PostgreSQL 社区沟通渠道
- 微信公众号: postgres用户会
- 微信群: PG圈
- 微博: PostgreSQL用户会
- Q群: 3336901 , 100910388 , 5276420 , 191516184
- 邮件列表: **pggeneral@groups.163.com** **pgadmin@groups.163.com** pglecturers@groups.163.com
- WEB: http://bbs.postgres.cn/ http://www.postgres.cn/