Versant 数据库大规模数据条件下的高性能查询试验报告

版本：1.0
发布时间：2011-12
编制：刘隆国

1、背景

随着国内各种业务系统数据量的快速提升，传统关系型数据库变得越来越慢。尤其是在处理业务逻辑关联的时候，传统关系型数据库几乎已经无法完全满足业务系统的需求，只能通过修改库表结构的方式进行优化。

在这种情况下，根据国外同类系统的相关经验，使用下一代的 Versant 数据库可以大大加快整体系统的运行效率，

为了验证 Versant 数据库在金融行业的使用情况，作为一个初步的系统，我们设计了一个模拟的业务逻辑：客户（Customer）和信用卡（CreditCard）以及他们之间的关联，来作为性能验证的基础。在此基础上，我们完成了两种规模和复杂结构的条件下的性能测试。

本文是对本阶段测试的一个总结，并作为下一阶段测试的起始点。

对本文有任何意见与建议，您可以随时联系：longguo_liu@versant.com.cn 。

本文依赖配套文件包：**QuickEcifDemo.rar**，其中包含本次测试的所有关键源代码和封装命令。

2、测试环境定义

本次测试的测试环境随意选择了一台 PC 机来完成。在该 PC 机上同时搭建了客户端（自行编制的 Java 代码）和服务器端（Versant 数据库）。测试机按照一般方式配置，已经安装有杀毒软件和软件防火墙，并且没有经过任何特殊优化。

- 硬件环境：

- CPU：Intel Core i7 CPU 920(4 core) 2.67Ghz；
- 内存：3GB；
- 硬盘：1.0TB 硬盘/7200RPM。

- 软件环境：
  - 操作系统：Windows 7 专业版 32 位；
  - Versant 数据库：V/OD 8.0.2
  - JDK：Sun JDK 1.6。

3、数据生成方案

根据大致的硬件要求，测试数据结构由两个类构成 Customer 和 CreditCard，其业务逻辑构成如下：

```
/**
 * Demo Business Class for customer in ECIF system.
 *
 * @author LIU LONGGUO
 *
 */
public class Customer {
    // 姓名
    private String name;
    // 证件类型
    private String IDType;
    // 证件编号
    private String ID;
    private LinkedList<CreditCard> cardList = new
LinkedList<CreditCard>();
}

/**
 * Demo Class for Business Card
 *
 * @author LIU LONGGUO
 *
 */
public class CreditCard {
    // 卡号
```

```
    private String cardNo;

    // 客户对象引用
    private Customer customer;

    // 失效日期
    private long expireDate = 0;
}
```

为完成性能测试，在业务逻辑的基础上，我们编制了一个测试类：TestDataGenerator，并在此基础上实现了以下两个大规模数据的生成方案：

测试方案 1：一亿条客户信息，五亿条信用卡记录。

描述：生成一亿条客户（Customer）数据，每条客户数据有五条信用（CreditCard）记录，在此情况下直接根据用户的姓名，查找用户对象，并打印出对应的所有信用卡记录。

其中：

- 客户记录包含客户信息，身份证号，身份证类型信息，以及客户所拥有的信用卡信息；
- 信用卡记录包含卡号，失效日期，到客户对象的引用信息。

实现方法：generateTestData()

命令行调用的命令：gen_data.bat

实施结果：

成功生成数据，数据规模大约在 120GB 左右。

后续处理：为查询需要，在 name 属性上生成一个 BtreeIndex。生成命令如下：

dbtool -index -create –btree com.versant.quickecif.domain.Customer name uam

测试方案 2：十亿条客户信息

描述：在方案 1 相同的数据结构条件下，生成十亿条客户（Customer）数据，但不生成信用卡信息，在此数据集直接根据用户的姓名（Customer.name），查找用户对象，并打印出该用户的详细信息。

其中：

- 客户记录包含客户信息，身份证号，身份证类型信息，以及客户所拥有的信用卡信息。

*实现方法：*generateTestData2()

*命令行调用的命令：*gen_data2.bat

实施结果：

成功生成数据，数据规模大约在 155GB 左右。

后续处理：为查询需要，在 name 属性上生成一个 BtreeIndex。生成命令如下：

dbtool -index -create –btree com.versant.quickecif.domain.Customer name uam

测试数据的清理：可以利用：***clear_data.bat*** 来清理在库数据。

4、数据查询测试以及查询测试结果

在上述两个测试数据集上，分别执行以下查询和动作：

1）随机执行 SQL：`select * from com.versant.quickecif.domain.Customer where name = '测 30000'`

2）打印该客户的客户信息以及其所属的所有信用卡信息。

*命令行调用的命令：*query_data.bat

5、测试结果

测试方案 1 的查询效率：

在测试方案 1 的数据集条件下，执行数据查询（即执行 query_data.bat）：

总共耗时：**63 毫秒**

其中，第一次查询之后的查询效率可以控制在 **20 毫秒**以内。

（参考：数据创建耗时：5 小时左右，平均每万个客户的创建成本在 1350

毫秒左右。索引生成耗时：3 小时左右，可以进行优化。）

性能原理分析：

Versant 能够实现这样的性能，其大致的原因可以总结如下：

1）在 Versant 数据库里，数据关联无需额外的联立查询，查询条件可以简化为单表查询。当查到用户记录时，打印该用户所属的信用卡记录几乎没有额外成本，查询性能与数据规模成线性关系。

2）在 Versant 数据库里，通过在要查询的字段上建立 Btree 索引，可以实现高效的数据索引检索机制。同时，Btree 索引在第一次查询之后，可以自动置入内存，使得数据的查询效率可以得到进一步提高。（Btree 索引可以通过手动方式让数据库装载该索引，从而在系统启动时直接实现较高的性能。）

测试方案 2 的查询效率：

在测试方案 2 的数据集条件下，执行数据查询（即执行 query_data.bat）：

总共耗时：**68 毫秒**

（参考：数据创建耗时：35522 秒，10 小时左右，平均每万个客户的创建成本在 280 毫秒左右。索引生成耗时：3 小时左右，可以进行优化。）

测试方案 2 中使用的查询条件与测试方案 1 的查询条件相同，即以 Customer.name 为目标查询条件。

在 Versant 数据库中，三参数的联立查询的性能与单参数查询的性能呈线性关系。

6、性能提升方向

本次测试仅仅完成了在大规模数据查询条件下的简单业务，实现了基础的业务查询处理。但是，考虑到本次测试所使用的主机也只是一般的 PC 机，其性能相对一般。因此，在正式的生产环境中，我们的测试性能还可以进一步进行提升。

1）本次测试使用的是标准的单 CPU 4 核 PC 机，使用的是标准内置磁盘。

如果使用高性能，多 CPU 的专用服务器，配合以高转磁盘，则系统查询的性能还有可能可以进一步提高。

2）本次测试使用的是简单业务模型，在处理复杂业务模型，例如查找客户的其它信息以及客户之间的关联关系时，通过将原来关系型数据库不得不做的复杂查询简化为简单查询，并通过对象导航方式来获得原来通过查询联立方式获得的数据，可以在一些复杂业务环境中，获得极高的业务处理效率。这一点可以作为下一阶段测试的基础。