一种适用于实时处理的专用内存数据处理系统

雷晓全 陈东 冯晞 (郑州商品交易所 450003)

摘要:本文介绍一种在 UNIX 平台上实现的适用于实时数据处理的专用内存数据库系统的设计思想、实现方法及其在期货交易中应用。

关键词:服务引擎 共享内存 消息队列 通信服务

引言

通用的数据库系统有很多,仅在 UNIX 系统上常用的就有 Oracle、Sybase、Informix、Unify 等,这些数据库系统面市多年,经过众多用户的实际使用,版本几经升级和完善,具有强大的数据处理和管理功能,在信息处理和信息管理领域中取得良好的应用效果。但是对某些实时数据处理系统(如证券交易系统和期货交易系统),用这些通用的数据库系统进行开发,通常不能达到预期的效果。

一、专用内存数据处理系统的设计思想

这里介绍的专用数据处理系统基于 Client/Server 结构。整个系统由 Server 端服务程序和 Client 端应用程序构成。Server 服务程序包括服务引擎和通信服务程序,服务引擎负责后台数据的初始和管理,应答 Client 端的访问请求,维护数据的完整性和安全性,提供共享访问,对事务性操作记录操作日志;通信服务程序提供网络通信。Client 端应用程序是根据 Server 端与 Client 端约定好的应用编程接口和任务需求开发的可视化前台程序。为了实现这个专用的内存数据库系统,有几个问题需要解决:1.选择合适的网络通信方式;2.制定有关的通信协议;3.制定数据储存和管理的方法;4.选择合适的内部通信方式。

为简化设计和提高系统性能,用 UNIX 系统作为 Server 端服务程序(包括服务引擎和通信服务程序)的开发和运行平台,使用共享内存、消息队列、信号灯技术,采用多进程方式;网络通信使用基于 TCP/IP 的网络协议,通过 BSD SOCKET 提供可靠的和容错的通信服务;Client 端应用程序的开发和运行平台可以为 DOS、WINDOWS 或 UNIX。系统不限制网络的协议或结构,只要 TCP/IP 支持的网络结构均可运行本系统(如 FDDI, Token - Ring, PPP 等)。

如图 1 所示，整个内存数据处理系统采用分层结构，第一层为服务引擎(ServerEngine)，第二层为通信服务程序(CommServer)，第三层为应用程序(Client)；第一层和第二层为 Server 端服务程序，第三层为 Client 端程序；第一层与第二层之间采用 UNIX 内部通信机制 - - 消息队列进行通信，第二层与第三层之间采用 TCP 协议进行通信；第一层由服务引擎和组成数据库表的共享内存组成，第二层为用 BSD SOCKET 编写的一个独立的通信服务程序(CommServer)，第三层为用 SOCKET API 编程接口或 WinsockAPI 编程接口开发的应用程序。

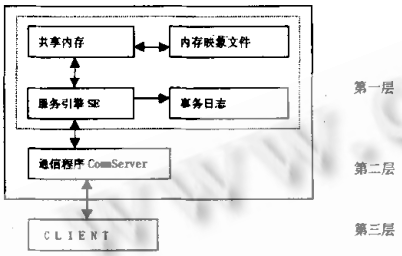

图 1　内存数据处理系统模块结构

二、Server 端服务程序

Server 端服务程序由服务引擎和通信服务程序两部分组成，服务引擎使用共享内存存放数据库表。

1. 共享内存的管理

共享内存的分配和管理影响服务引擎的使用，影响整个系统的效率，因此在内存的分配和使用上，放弃了传统的用链表的方法来管理和使用内存，而采用共享内存和静态数组存放数据库的表。即根据应用系统的要求创建 n 个数据库表，或者称 n 个数据结构，对于每个数据库表的最大记录个数用 m1，m2，...，mn 表示，每个数据库表的大小为固定数，则可计算出 n 个数据库表的大小为 size，可以一次申请 size 大小的共享内存，静态将 n 个数据库表分配在 size 大小的共享内存中。如在一个期货交易系统中需 20 个表，则设 21 个表，其中第一个表是用来存放管理信息，管理其余 20 个表。共享内存表的定义和分配如下所示：

```
typedef struct STATE-TAG{
    ...
    }STATE;
typedef struct FIRST-TAG{
    ...
    }FIRST;
    ...
typedef struct 20TH-TAG{
    ...
    }20TH;
state * STATE;
first * FIRST;
...
20th * 20TH;
size = sizeof(STATE);
size = size + sizeof(FIEST) * max-first;
...
size = size + sizeof(20TH) * max-20th;
分配 size 大小的共享内存 shm;
state = (STATE * )shm;
state - >offset-first = sizeof(STATE);
state - >offset-second = state - >offset-first + sizeof
(FIEST) * max-first;
....
state - >offset-20th = state - >offset-19th + sizeof
(19TH) * max-19th;
first = (FIRST * )state + state - >offset-first;
second = (SECOND * )state + state - >offset-second;
...
20th = (20TH * )state + state - >offset-20th;
```

除 state 表只有一个记录外，其余每一个表中记录的个数可以用一个参数文件来控制，在系统启动时将参数调入系统。在共享内存中 state 表放在最前面，其余数据库表依次排列。在 state 表中记录了其余每个表的最大记录个数、当前记录位置和每个表表头相对于 state 的偏移量。在每个表中可以设置指向别的表的前后指针。对数据库的访问如下所述，设 xxx 是内存库表中的一个域，则

对 state 表的访问，state - >xxx；

对 first 表第 1 个记录的访问，(first + 0) - >xxx；对 first 表第 2 个记录的访问，(first + 1) - >xxx；对 first 表第 n 个记录的访问，(first + n - 1) - >xxx；

对 second 表第 1 个记录的访问，(second + 0) - >xxx。

根据需要，表的数量和每个表的结构可以随意变化，以满足不同任务的需要。

共享内存中存放的是数据库，可能有多个用户或进

程在某个时刻都要访问数据库，为保证数据库中数据的完整性和一致性，防止两个用户同时修改同一个数据，用一个信号灯来管理对共享内存的访问。当对数据库进行写操作时，在操作前调用信号灯对共享内存进行加锁，当操作完成后，调用信号灯对共享内存进行解锁。

2. 服务引擎

服务引擎是整个系统的管理核心，它创建共享内存，初始化有关数据，创建消息队列和信号灯，从公共的接收消息队列读取信息和处理信息，反馈处理结果。服务引擎是一个事件驱动系统，它总是在接收消息队列上等待新的事件。

服务引擎在完成一次性的初始操作之后，进入循环等待操作状态。从公共的接收消息队列读一条信息，如果没有取得信息，则在队列上等待。如果取得信息，则分析这条信息，检查这条信息是查询信息还是事务处理信息。如果是查询信息，则从共享内存数据库中整理并发送查询结果。如果是事务处理信息，则首先记录事务日志，记录事务处理开始标记，并根据请求内容对事务进行相应处理，当完成事务处理后，记录事务处理完成标记，并向事务请求者发回处理结果。发送结果是通过发送消息队列完成的。发送消息队列有若干个，每个前台用户进程有一个发送消息队列；接收队列只有一个，为所有前台用户进程共用。

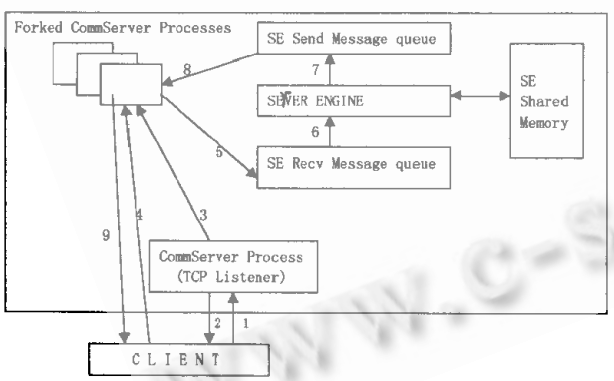

图 2　服务引擎和通信服务程序的结构

如图 2 所示，一个完整的信息传递过程如下所述：

(1)CLIENT 向通信服务程序 CommServer 发送联接请求；

(2)CommServer 向 CLIENT 应答联接请求；

(3)CommServer 创建子进程 CommServer；

(4)CLIENT 向子进程 CommServer 发操作请求(如委托请求)；

(5)子进程 CommServer 将此操作请求发送到公共接收消息队列；

(6)服务引擎从接收消息队列读此操作请求并进行相应处理；

(7)服务引擎将处理结果发送到发送消息队列；

(8)子进程 CommServer 从发送消息队列取处理结果；

(9)子进程 CommServer 将处理结果发送到 CLIENT。

CLIENT 程序、通信服务程序和服务引擎之间的消息传递结构如下：

```
typedef MESSAGE-TAG{
    int type; /* 消息的类型 */
    int length; /* 消息的长度 */
    char reserved[10]; /* 保留 */
    char *data; /* 消息的内容 */
    }MESSAGE;
```

无论是接收消息还是发送消息，都使用这个结构。对每一种请求消息类型，规定相应的应答消息类型；当消息类型很多时，为便于管理，可以用 type 作主类型，在 data 中再分子类型。

在服务引擎中用 switch...case... 语句实现消息的分类处理，每增加一种新的消息类型，增加一个 case 语句。

3. 通信服务程序 CommServer

通信服务程序(CommServer)在整个系统中起至关重要的作用。它好象一个透明的管道，连接着前台应用和后台管理系统。它使用了消息队列技术和 socket 编程技术。由于通信服务程序与服务引擎对消息队列的接收和发送的概念正好相反，一个队列对通信服务程序来讲是发送队列，但对服务引擎来讲是接收队列；反之依然。通信服务程序与服务引擎必须在同一台主机上。

图 3 表示了通信服务程序的处理流程。父进程是一个守护进程，它首先创建公共的发送消息队列 msq，之后在某个端口上等待远端的联接。远端程序向主机的这个端口发送 socket 联接请求，父进程接到联接请求后，便产生联接应答，生成子 socket 联接，并 fork 子进程，由子进程完成与该远端程序的通信任务。父进程返回，继续在端口上等待下一个联接请求。每来一个联接请求，父进

程创建一个子进程。

子进程首先创建与本进程号有关的接收消息队列 mrecv，然后在子 socket 上等待接收应用程序发来的请求，如果没有请求，则一直等待。当接到请求时，子进程将其发送到发送队列 msq；之后子进程在自己的接收队列 mrecv 上等待接收服务引擎发来的应答，将应答发送到子 socket，检查本次应答是否全部接收完成，如果没有接收完成，则转到接收队列 mrecv 继续接收。如果本次应答全部接收完成，则转到子 socket 上等待接收应用程序发来的请求。

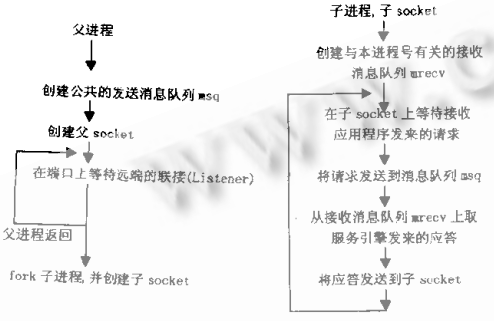

图 3　通信服务程序的处理流程

4. 数据恢复

由于数据库是存放在共享内存中的，在数据库的操作过程中服务器意外掉电就会导致数据库数据的丢失，因此需要有相应的数据恢复机制。

服务引擎在完成创建共享内存和数据初始化之后，生成内存映象文件，保存全部内存数据。服务引擎对所有影响内存数据库数据的事务操作记录事务日志。当需要进行数据恢复时，按内存映象文件创建共享内存并载入相应数据，按事务日志文件重新处理所有事务即可将内存数据库数据恢复到破坏前的状态。

三、Client 端应用程序

为开发应用程序，设计了一个应用开发编程接口 API。使用这个编程接口，应用程序可以在 dos、windows 及 unix 环境下开发。应用程序可以与数据库系统在同一台主机上，也可在不同的主机上，应用程序只要与数据库系统通过 tcp 协议能够联通即可。为此开发了 4 个 API 接口程序：

ClientConnect()

ClientSend()

ClientReceive()

ClientDisconnect()

应用程序在与内存数据处理系统进行数据交换之前，必须调用 ClientConnect()与数据处理系统建立 tcp 联接，在完成数据交换之后，用 ClientDisconnect()关闭与数据库系统的 tcp 联接。用 Clientsend()向数据库系统发送请求，用 ClientReceive()接收数据库系统返回的应答。

在调用 ClientSend()之前，将请求类型添入 message－＞type，请求内容的长度添入 message－＞length，请求内容添入 message－＞data。在调用 ClientReceive()取得应答结果之后，根据 message－＞type 判断应答消息的类型，message－＞length 判断应答信息的长度，并从 message－＞data 取出应答信息的内容。

四、应用

使用专用内存数据处理系统开发的期货交易系统与使用通用数据库开发的期货交易系统相比在系统的处理速度和扩展能力上都有极大的提高。在服务器配置和网络设备配置相同的条件下，用 Sybase 数据库系统开发的期货交易系统，最多联接了 250 个客户机，在 250 个客户机同时进行委托或查询时，平均响应时间在 2－3 秒，峰值时每秒处理 50 个委托事务。使用专用内存数据库系统开发的期货交易系统，最多可以支持数千个客户机同时操作；在联接 500 个客户机同时进行委托或查询时，客户机的平均响应时间小于 1 秒，峰值时每秒处理 2000 个委托事务。这个专用内存数据处理系统使用 UNIX 系统的基本功能，能极大的减小系统的开销，提高系统的性能。在同样的服务器和网络配置情况下，能将系统的性能提高一个数量级，因此适合实时数据处理系统。

（来稿时间：1998 年 7 月）