# 基于事件的连续数据保护系统研究*

王德军[1]，何　征[2]，王丽娜[2]
（1. 中南民族大学 计算机科学学院，武汉 430074；2. 武汉大学 计算机学院，武汉 430074）

**摘　要**：针对现有连续数据保护系统存在恢复点选择困难、数据一致性差和无法支持事务级数据恢复粒度的问题，提出了基于事件驱动的连续数据保护恢复系统（EBRS）解决方案，给出了 EBRS 的三个核心组件应用感知模型、应用协同模型和事件恢复视图的设计与实现方案。在 Oracle 环境下实现了 EBRS 的关键功能，实验结果表明，基于事件驱动的连续数据保护恢复系统具有恢复粒度细、恢复效率高、数据一致性好的特点，具有广阔的应用前景。

**关键词**：连续数据保护；应用感知；应用一致性；基于事件恢复
**中图分类号**：TP309.2　　**文献标志码**：A　　**文章编号**：1001-3695(2014)02-0465-07
doi:10.3969/j.issn.1001-3695.2014.02.034

## Research of event-based continuous data protection system

WANG De-jun[1], HE Zheng[2], WANG Li-na[2]
(1. *School of Computer, South-Center University for Nationalities, Wuhan* 430074, *China*; 2. *School of Computer, Wuhan University, Wuhan* 430074, *China*)

**Abstract**: Traditional continuous data protection (CDP) had difficult in selecting the right one from many restore points, could not guarantee consistence of data recovery and could not support transaction-level recovery for the application. This paper suggested an event-based continuous data protection recovery system, and gave the detail design of the three core components including the application-aware model, the application-collaborative model and the event-based recovery view. It implemented the key functions of EBRS in Oracle database environment. Experiment results show that event-based continuous data protection recovery system has the features of supporting fine-grained data recovery, efficient recovery and good data consistence, and has broad application prospects.

**Key words**: continuous data protection; application-aware; application-consistent; event-based restore

## 0　引言

当今社会对计算机信息系统的依赖越来越明显。企业的运营无时无刻都离不开信息系统所依赖的数据，任何规模的企业都难以承受由于数据丢失而造成关键业务或应用无法访问所带来的损失[1]。随着竞争形势的日益严峻，企业对业务的持续性提出了更高的要求，要求在信息系统发生故障时，能实时恢复业务并保证业务的持续性。在这种形势下，连续数据保护技术以其在 RPO（recovery point objective）[1] 和 RTO（recovery time objective）方面的优势脱颖而出，并逐渐成为备份容灾领域研究的热点。连续数据保护技术（continuous data protection，CDP）[2] 通过持续跟踪捕获数据的更新变化，生成具有时间标签的日志记录。在数据被破坏或出现错误时，通过重做日志的方式将数据恢复到过去任意时间点，为应用提供了连续性的数据副本。数据的持续性保护为应用和业务的连续性运行提供了保障。

根据实现模式的不同，CDP 可以分为应用级 CDP[3]、文件级 CDP[4] 和块级 CDP[5~9] 三类。应用级 CDP 是针对特定应用定制的 CDP 系统，这种 CDP 的方式通过与应用进行深度集成，确保应用数据在连续保护中的一致性；文件级 CDP 工作在文件系统层，通过捕获文件系统级的更新操作（如创建、写、删除、重命名等）并将更新记录发送到服务器端，以便将来实现任意时间点的文件恢复；块级 CDP 直接运行在物理的存储设备或逻辑的卷管理器上，独立于上层应用，具有很强的通用性[10]。图 1 显示了一种基于文件和块级的 CDP 系统模型[11~13]。

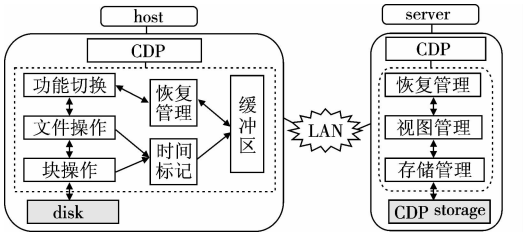


图 1　CDP 系统模型

CDP 系统模型分为 CDP agent 和 CDP server 两个部分。CDP 代理端运行在被保护数据所在的主机上，其主要功能是对数据进行监控，捕获更新操作（文件级和块级）、标记数据并发送到服务器端。在数据恢复阶段，CDP 代理从服务器端读取操作数据完成数据恢复过程。CDP 服务器主要进行操作日志的存储、组织、数据版本生成等工作。

**收稿日期**：2013-03-03；**修回日期**：2013-06-13　**基金项目**：国家民委科研基金资助项目（12ZNZ009，12ZNZ008）；国家自然科学基金资助项目（60603008）

**作者简介**：王德军（1974-），男，湖北钟祥人，博士，主要研究方向为容灾备份、面向恢复计算、信息安全（wangdj@whu.edu.cn）；何征，男，工程师，硕士，主要研究方向为信息安全；王丽娜（1964-），女，教授，博士，主要研究方向为信息安全、数字水印、可信计算.

CDP 系统的执行流程如下:CDP agent 通过文件过滤驱动和磁盘过滤驱动,分别在 file 级别和 block 级别捕获文件和数据块更新操作,通过 time marker 为操作数据加入时间标签,然后通过网络发送到 CDP server。CDP server 接收操作数据,并对数据进行组织、存储和管理。在需要恢复数据时,为用户提供基于时间点的恢复点视图,并根据用户选择的时间点进行基于时间点(time-based restore)的数据恢复。

图 1 的 CDP 系统模型采用了文件级/块级的实现方式,它并不关心数据所对应的上层应用。这样实现的好处是数据保护独立于应用,简化实现过程并达到通用的目的。但同时也存在以下三方面问题:

a)恢复点选取困难。由于不关心上层应用, CDP 捕获的数据流缺少语义解释,所提供的数据恢复视图只能是一系列散列的时间点,无法为用户恢复数据提供详细的数据版本信息。

b)无法保证恢复点应用数据的一致性。CDP 提供了众多的时间点给用户进行恢复,但这众多的时间点其所对应的数据版本不一定是应用一致的。某个时间点的数据版本需要应用检查和事务回滚机制才能恢复到一致性状态,并不是每个时间点的数据版本都是可用的。

c)恢复粒度问题。对于一些特定的事务性应用,如数据库,其操作的粒度为事务。所以真正的连续数据保护应该提供针对事务级别的恢复服务。

针对原 CDP 系统模型存在的这三方面的问题,本文提出了一种基于事件的连续数据保护恢复技术。该技术通过以下三个方法解决以上三方面的问题:

a)基于传统的 CDP 解决方案,通过应用感知技术,感知应用产生的关键事件,并为 CDP 数据流加入事件标签,为恢复点数据提供语义解释。

b)通过协同技术与应用集成,在应用端通过代理和脚本的形式,调整应用到一致性状态,保证恢复点数据的一致性。

c)提出了基于事件的 CDP 恢复解决方案。在 CDP 原有的基于时间恢复机制的基础上,通过捕获应用事务级别的更新操作生成日志记录。在恢复时,通过重载事务操作的方式实现基于事务的恢复,数据恢复粒度更精细,且每个恢复点的数据都是应用一致的。

本文的目的是通过基于事件的技术,在原 CDP 系统模型的基础上进行扩展,实现一个基于事件的 CDP 恢复系统。其意义在于使 CDP 系统在使用时具有更好的体验:详细的事件标签为用户选择恢复点提供向导;协同技术为用户提供协同应用产生一致性副本和加入自定义标签的接口;基于事件的恢复为用户提供更加细粒度的恢复服务。

## 1 设计目标及思想

基于事件的 CDP 恢复系统(event-based CDP restore system,EBRS)是以原 CDP 系统模型为基础,通过应用感知和协同技术,以保证恢复点数据的应用一致性和接近于零的 RPO 为目标,实现的基于事件的 CDP 恢复系统。EBRS 主要提供以下几个方面的功能:

a)配置管理功能。向用户提供了对系统进行策略化管理的配置接口,用户可以根据需要通知系统产生一致性数据版本,并插入事件标签来描述恢复数据映像。

b)事件捕获。针对特定的应用,需要捕获应用产生的关键事件,为基于事件的恢复提供数据源。

c)应用协同。它主要用来完成数据的一致性控制,通过脚本或代理的形式调整应用到一致性状态,保证恢复点数据的一致性。

d)事件视图管理。EBRS 在恢复时需要为用户提供具有详细语义标签的事件恢复视图指导恢复过程,因此需要在服务器端维护一个事件视图结构,从而在恢复时快速地生成恢复视图。

e)基于事务的恢复。EBRS 通过两级数据恢复机制来支持事务恢复。在数据恢复的第一阶段,通过基于时间点的恢复方法,生成距离恢复点最近的一致性数据版本;第二阶段,提取一致性数据版本到恢复点之间的事务操作序列,在一致性数据版本的基础上,通过事件装载技术重做这些事务操作序列,从而完成基于事务恢复。

## 2 基于应用感知的 CDP 模型

### 2.1 EBRS 在 CDP 中的层次

EBRS 并不是一个独立的系统,而是在原 CDP 系统模型的基础上扩展而来。图 2 显示了基于事件的 CDP 系统模型。从图中可以看出,该模型在原 CDP 系统模型的基础上添加了 EBRS agent 和 EBRS server 两个部分。

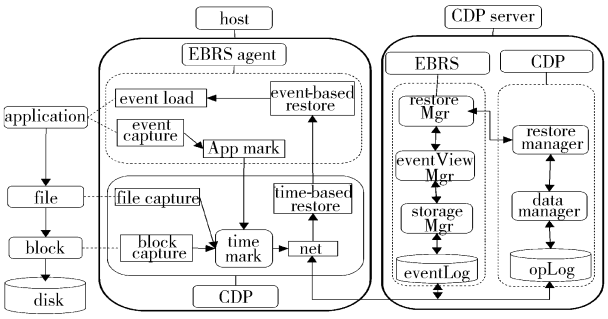


图 2 基于事件的 CDP 系统模型

EBRS agent 和 EBRS server 构成了 EBRS。该系统扩展了 CDP 系统的功能,其工作在应用层,通过事件捕获技术捕获应用产生的事件操作,并对事件进行应用标记,产生基于应用的事件日志,通过网络发送到 EBRS server。EBRS server 接收事件日志,根据事件日志生成事件视图,并对日志进行存储和管理。在需要恢复时,为用户提供基于事件的恢复视图,并根据用户选择的事件点进行基于事件(event-based restore)的数据恢复。

图 3 显示了加入 EBRS 模块以后,整个 CDP 系统的工作过程分为事件监控和数据恢复两个阶段。

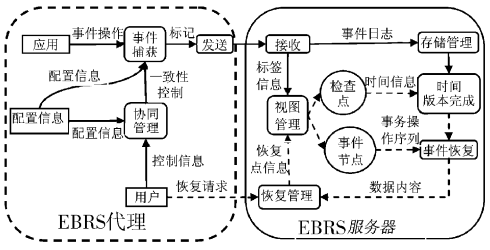


图 3 基于事件的恢复流程

a)事件监控。事件监控阶段是产生原始数据版本和事务日志的过程。在事件监控阶段,CDP agent 继续在文件级和块级对数据进行监控,EBRS agent 则在应用层面进行监控。这一

阶段主要完成事件捕获、应用标记、日志存储任务。具体流程如下:

(a)EBRS agent 在应用层监控应用的执行流程,并根据用户的配置信息捕获应用产生的事件,同时如果需要协同应用,则事件捕获模块,通过一致性控制技术使应用达到一致性点,并记录协同事件的信息;

(b)EBRS agent 对捕获到的事件进行分类标示,区分检查点事件和一般事务事件,同时给事件加上应用标签,标注事件所属的应用;

(c)通过 time marker 给事件操作加上时间戳,为 EBRS server 进行事件与数据更新操作的语义映射提供支持;

(d)通过网络将事件日志发送到 EBRS server;

(e)EBRS server 存储事件日志数据,并根据标签信息生成和更新事件视图。

b)恢复阶段。具体过程如下:

(a)EBRS agent 获取要恢复的应用信息,并发送到 EBRS server;

(b)EBRS server 按应用类别提供两级恢复视图,即检查点视图和事件视图,这些视图的每个节点对应一个完整的数据版本映像,并具有不同的语义标示;

(c)用户选择恢复点,并将恢复点的信息发送到 EBRS server;

(d)判断用户选择的恢复点类别,如果选择的是检查点,则 EBRS server 将该检查点对应的时间标记和恢复信息发送给 CDP server 的恢复管理模块,由 CDP server 提供该时间点的数据版本,并发送到 CDP agent 完成恢复,否则转步骤(e);

(e)事件节点的恢复过程采用两级恢复机制:第一阶段,EBRS 恢复管理模块,通过基于时间的恢复方法,生成距离恢复点最近的一致性数据版本;第二阶段,提取检查点节点和事件节点之间的所有事务操作序列,发送到 EBRS agent,EBRS agent 在检查点数据版本的基础上装载事务操作,完成恢复过程。

### 2.2　EBRS 的模块结构

图 4 显示了 EBRS 的核心模块,EBRS 分为 EBRS agent 和 EBRS server 两部分。其中 EBRS agent 又分为应用代理层和代理管理层两层。

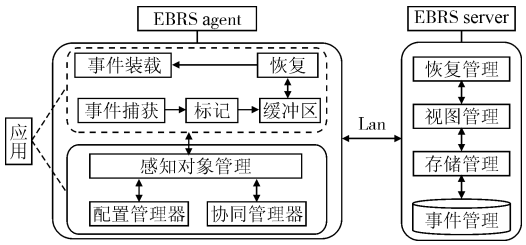


图 4　EBRS 的核心模块

每个应用对应一个应用代理,代理完成对单个应用的应用感知功能。应用代理分为事件捕获、标记、事件装载和恢复模块。事件捕获负责监控应用的执行流程,截获应用产生的事件;标记模块为事件加上应用和时间标签,并提取出事件的描述信息;事件装载模块主要在恢复时,通过重做事务操作完成基于事务的恢复过程,它是恢复模块的基础;恢复模块负责从 EBRS server 获取事件视图并向用户展示,同时根据用户选择的恢复点向 server 端索取恢复数据,完成恢复过程。

代理管理层负责对所有的应用代理进行集中的管理。感知对象管理器负责应用代理的注册、注销、删除,它记录了所有应用代理的详细信息。协同管理模块负责多个应用的协同管理,它通过感知对象管理器获取应用代理信息,并协同多个代理在同一时间点产生一致性的数据副本。配置管理器是面向用户的管理接口,为用户提供自定义恢复点标签的功能。

EBRS server 分为存储、视图及恢复管理模块。存储管理模块负责事件日志的存储和归档管理;视图管理模块通过一个事件树结构,提供一个虚拟的基于事件的恢复视图,视图中的每个节点表示了一个潜在的数据恢复点,并提供了语义标签和恢复所需要数据的索引;恢复管理模块负责恢复过程的管理,在恢复阶段,首先通过视图管理模块向 EBRS agent 发送事件视图,然后根据用户的选择生成初始数据版本,最后提取事务操作序列发送到客户端完成恢复过程。

## 3　协同管理设计

### 3.1　协同管理技术原理

EBRS 协调多个关联应用产生一致性数据副本,通过多个模块的协作来完成,即前端驱动、调度管理器、EBRS 后端驱动、控制台,图 5 显示了各个模块的关系。

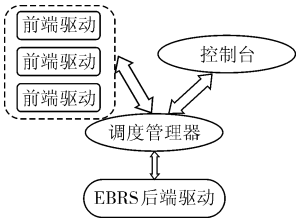


图 5　协同管理模块关系图

a)控制台,为 EBRS 提供给用户的配置管理接口。用户可以通过控制台,在需要时手工发送协同请求,也可以定制协同策略,如可以设置按 1 h 为间隔执行协同过程。

b)前端驱动,主要任务是保证应用数据的一致性。具体地,前端驱动执行以下两个操作:锁定应用,冻结应用的写请求;刷新应用缓存。很多应用都维护有自己特有的缓存结构,如数据库系统。只有在应用缓存中进行刷新操作,CDP 数据捕获模块才能截获这部分写操作。

c)EBRS 后端驱动,主要任务是刷新文件系统缓存,该模块只在块级 CDP 系统中起作用。在 Windows 系统平台下,除了应用自己的缓存结构以外,文件系统为了提高读写效率,一般都提供了文件系统缓存。从应用端对文件的写入操作,首先会被缓存在文件系统的缓存中,并执行延迟写操作,而块级磁盘过滤驱动程序工作在磁盘卷级。后端驱动程序进行以下操作:冻结文件系统的写请求,刷新文件系统缓存到磁盘。

d)调度管理器,负责整个协同过程的调度管理任务。它接收控制台端的协同请求,通知各个应用对应的前端驱动程序刷新应用缓存,然后通知后端驱动程序,刷新文件系统缓存,随后标记该时刻的数据为协同一致性数据版本,完成整个协同管理过程。

### 3.2　协同处理流程

EBRS 的调度管理模块由感知对象管理器和协同管理器两部分组成。由于不同应用特性不同,针对不同的应用需要不同的前端驱动来完成应用缓存的刷新工作。感知对象管理器负责管理这些前端驱动模块;协同管理模块负责整个协同过程

的调度工作。图 6 显示了协同管理的流程。

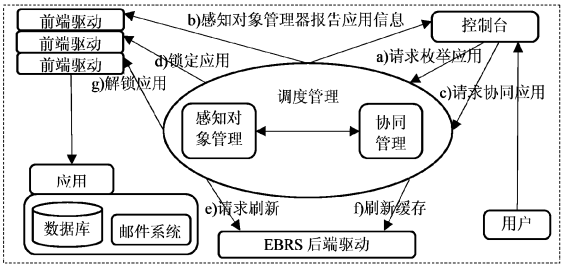


图 6 协同管理处理流程

a)用户通过控制台界面,请求枚举系统中运行的应用信息(被保护的应用)。

b)感知对象管理器将此刻正在运行的受保护应用信息发送到控制台。

c)用户通过控制台建立协同组,将需要协同一致的应用加入同一组中,并设置组内协同策略,如以小时为间隔进行协同处理,同时提供协同数据版本的语义标签,随后发出协同请求。

d)协同管理器通知协同组应用的前端驱动程序完成准备工作。

e)每个前端驱动根据应用的特点刷新应用缓存,同时冻结应用的写入请求。这些完成后,通知协同管理器应用层的刷新工作已完成。

f)协同管理器通知 EBRS 后端驱动程序完成文件系统级数据刷新工作。

g)后端驱动程序冻结文件系统的 I/O 写请求,同时刷新文件系统缓存,保证应用元数据的正确性及磁盘数据写入顺序的一致性。

h)协同管理器标记 CDP 系统当前时刻的数据版本为协同一致数据版本,并根据用户输入的标签信息对数据进行标记;随后解锁应用和文件系统缓存。

## 4 事件视图

事件视图是对于应用产生的事件的形式化组织和描述,提供基于事件的恢复基础。如图 7 所示,事件树是一个扩展简化的二叉树结构,每个应用对应一棵二叉树,同时所有的二叉树串联起来。

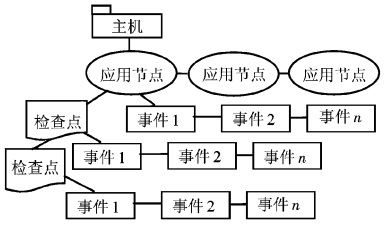


图 7 事件视图的结构

事件树的节点分为以下三类:

a)应用节点。它保存与应用相关的信息,包括应用名、应用涉及的数据文件的 ID 号、初始化时间等信息。根据该节点,可以生成应用数据的初始化版本。每个被保护的应用对应于一个应用节点,所有的应用节点串联起来。

b)检查点节点。这里将该节点命名为检查点节点,是因为该节点对应时刻的数据版本都处于应用一致状态。无论是由应用自身执行的检查点事件生成,还是由用户自定义产生的数据一致性事件,都以检查点的结构来标示。检查点节点保存一致性数据产生的事件描述信息、发生时间、指向下一个检查点结构的指针,以及指向该检查点之后应用执行的第一个事务操作结构的指针。

c)事件节点。该节点描述自上次数据一致性副本产生之后,所执行的一系列事务操作信息。该节点并不存储实质的事务操作数据,而是保存事务操作数据的摘要信息、发生时间、索引。摘要信息和发生时间主要用来在用户查询时生成事件标签,提示用户该点可以进行事务级别的恢复。

在恢复阶段,EBRS agent 从服务器端获取事件视图,然后在客户端生成恢复视图,并向用户展示应用对应的潜在恢复点。

## 5 EBRS 在 Oracle 环境下的实现

EBRS 是一个面向多应用的基于事件的恢复系统。系统的实现依赖于应用事件的捕获,而不同的应用其产生的事件千差万别,事件捕获及装载也不尽相同,因此针对每个应用都需要开发专有的事件捕获和事件装载模块。本文在 Oracle 环境下实现 EBRS 基于事件的恢复功能。

### 5.1 Oracle 环境下的事件捕获

EBRS 捕获 Oracle 产生的关键事件,为基于事件的恢复提供基础,事件包括检查点事件和事务操作事件(如 insert 操作、delete 操作、update 操作等)两类。本文主要阐述事务操作事件的捕获。Oracle 的重做日志机制记录了 Oralce 对数据库的所有更改操作,本文通过解析 Oracle 的日志文件的结构提取 Oralce 的事务操作序列。

### 5.2 Oracle 日志文件结构

Oralce 的日志文件由多个固定大小的数据块组成,日志文件的大小在日志文件创建时设置,通常块的大小为 512 Byte[14,15]。日志文件的逻辑结构如图 8 所示。

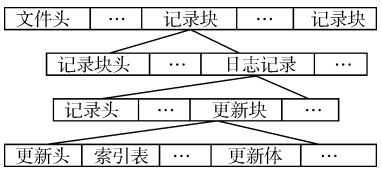


图 8 日志文件的逻辑结构

Oralce 的日志文件以 512 Byte 划分为块,其中开始的两块为文件头,之后的块为记录块。记录块是物理结构,可能包括多个日志记录,记录块分为记录块头和日志记录。日志记录是逻辑结构,一个日志记录可能跨越多个数据块,代表数据库的一个原子操作。日志记录由记录头和多个更新项组成。每个更新项又分为更新头、索引表和更新体,代表了对数据块的一次更新操作。

a)文件头结构。Oracle 的日志文件包括一个固定大小的文件头,通常占据两个数据块为 1 024 Byte。文件头会记录数据库名、数据库 ID 号、数据块大小、数据块数等信息。日志解析只涉及到文件头部分的数据块大小和数据块数两部分,其结构定义如下:

```
typedef struct _fileheader
{
    unsigned int un_used;
    unsigned int block_size;
    unsigned int block_counts;
```

```
}fileheader;
```

b)记录块头。Oracle 从第三块起,每一块代表了一个记录块。每个记录块开始的 16 Byte 为记录块头,其内容分别为日志文件序号、记录块号、时间、第一条日志记录的偏移量。其结构定义如下:

```
typedef struct _log_ header
{
  unsigned int file_seq;
  unsigned int block_num;
  unsigned int time;
  unsigned short offset;
  unsigned short un1;//对齐位
}log_header; //16 Byte
```

通过字段 offset 可找到该块第一条日志记录的起始地址。

c)日志记录头

记录块头之后是 Oracle 的日志记录项。日志记录可能位于一个记录块中,也可能跨越多个块,长度并不是固定的,Oracle 通过日志记录头描述日志记录的索引信息。日志记录由日志记录头和多个更新向量(change vector)组成。日志记录头的结构定义如下:

```
typedef struct _record_header
{
  unsigned int record_len;
  unsigned char vld;
  unsigned char unused; //用来字节对齐
  unsigned short scn1;
  unsigned int scn2;
}record_header; //12 Byte
```

根据 record_len 字段可以确定该日志记录的长度。

d)更新头 & 索引表

在 record_header 之后是更新头,更新头主要记录更新项的操作类型(操作码)、数据块地址(DBA)、数据对象 ID(oid)等。

```
typedef struct _cv_header
{
  unsigned short opcode; //操作码
  unsigned short bcls; //block class
  unsigned short afn;
  unsigned short un1; //字节对齐
  unsigned int dba; //data block address
  unsigned int scn2;
  unsigned short scn1;
  unsigned short un2; //字节对齐
  unsigned char seq;
  unsigned char type;
  unsigned short oid;
}cv_header;//24 Byte
```

其中,opcode 为操作码,表示更新项的类型;oid 为表对象的 id,可以根据 oid 在数据字典查找到表对象的信息,包括表名、行数、列名等。

更新项索引表紧接 cv_header 之后,由索引表长度和更新项长度组成。索引表长度为 2 Byte,表示整个表的长度;更新项长度有多个,每个为 2 Byte,代表更新项的具体长度。索引表的结构定义如下:

```
typedef struct _index_table
{
  unsigned short len;
  unsigned short *index;
}index_table;
```

### 5.3 SQL 语句提取

5.2 节分析了日志文件的结构,主要包括文件头、记录块头、日志记录头、更新头和索引表结构。日志解析是要提取出 Oracle 执行的 SQL 语句,这些语句存在于日志记录的更新体中。在每个更新项的索引表之后就是更新体,更新体的结构根据更新头 cv_header 的操作码 opcode 而定。本文日志解析主要分析 Oracle 的事务操作,即 insert、delete 和 update,这三种事务对应的操作码分别为 0x0B02、0x0B03、0x0B05。根据这三种操作码对应的更新体即可以解析出一个原子的 SQL 语句。

1)提取 insert 操作

操作码为 0x0B02 的操作为 insert 操作,该操作包含了插入操作的内容,包括插入的 ROWID 以及数据信息。插入操作的更新体主要有事务的信息(vector_tran)、控制信息(vector_control)和数据信息(vector_data_insert)。

表 1 显示了事务信息结构 vector_tran 的关键字段及其含义。XID 是事务的唯一标志,由回滚段号(xidusn)、事务槽号(xidslot)、事务序列号(xidsqn)三部分组成。可以根据 XID 定位本次插入操作所属的事务过程。

表 1　vector_tran 关键字段及含义

| 字段 | 类型 | 偏移/Byte | 意义 |
|---|---|---|---|
| xidusn | unsigned short | 8 | 回滚段号 |
| xidslot | unsigned short | 10 | 事务槽号 |
| xidsqn | unsigned int | 12 | 事务序号 |

Vector_control 结构主要包含 insert 的控制信息,解析时需要用到的字段为 col_num(插入的列数)、bdba(表的数据块地址)、slot(插入的记录在数据块中的行号)。通过 cv_header 中的表对象 id、bdba、slot 可以计算出插入记录在行 ROWID。ROWID 由四部分组成:数据对象编号(oid)、文件编号、块编号、行号(slot)。其中文件编号和块编号由 bdba 生成。表 2 显示了 vector_control 的关键字段及其含义。

表 2　vector_control 关键字段及含义

| 字段 | 类型 | 偏移/Byte | 意义 |
|---|---|---|---|
| bdba | unsigned int | 0 | block dba |
| col_num | unsigned char | 19 | 列数 |
| slot | unsigned short | 42 | 行号 |

在 vector_tran 和 vector_control 之后就是插入的数据部分信息 vector_data_insert。Vector_data_insert 由 col_num(插入的列数)个数据项组成,每个数据项表示了插入该列的数据,每个数据项的长度由更新项索引表指定。解析时,依次根据数据项的长度就可以取出所有插入的数据。

根据 insert 操作的更新体结构就可以提取出 SQL 语句,其提取过程如下:

a)从更新体起始位置获取 vector_tran,从中提取出事务的 XID 号。

b)获取 vector_control 结构,提取出插入的列数 col_num,插入记录所在的数据块地址 bdba,插入记录所在的行号 slot,并从 cv_header 中提取出 oid,根据 bdba、slot、oid 计算出插入记录的 ROWID。

c)从索引表 index_table 中获取插入记录每列数据的宽度。Index_table 以 2 Byte 为一个单位,其中第一项为表长度,第二项

为 vector_tran 的字节数,第三项为 vector_control 的字节数,从第四项开始记录了数据部分的信息。对于 insert 操作,从第四项开始的每两个字节表示了插入的列数据的宽度,即列数据的字节数。

d)根据每列的宽度从数据项部分解析插入数据每一列的值。数据项的起始地址在 vector_control 之后。

e)构造插入操作的 SQL 语句,其格式为

INSERT INTO table_name (col_1,col_2,…) VALUES(value_1,value_2,…) WHERE rowid = ROWID

2)提取 update 操作

Update 操作的操作码为 0x0b05,其更新体结构的前两部分与 insert 相同,而索引表存储的信息和数据项部分的信息结构不同。同样可以根据 insert 中的步骤 a)b)解析出事务的 XID 和记录的 ROWID,然后进行数据部分的抽取过程。Update 索引表 index_table 从第四项开始是更新操作的列号列表,每个列号占据 2 Btye,由此可以抽取出更新操作涉及到了哪些列及其列号。数据项部分由多个二元组(column_len,column_data)组成。其中 column_len 占据 1 Byte,列数据部分的字节数根据 column_len 而定。列数据的抽取过程为:定位到数据项部分的起始位置,循环读取 1 Btye 的 column_len 和 column_len 字节的列数据部分,从而获得所有列的数据信息解析出的更新操作的 SQL 语句,其格式为

update set table_name col_name1 = value1,col_name2 = value_2 where rowid = ROWID

3)提取 delete 操作

Delete 操作对应的操作码为 0x0b03。对于 delete 操作,只需要解析出其所属事务 XID 和删除记录的 ROWID 即可。其更新体结构只包括 vector_tran 和 vector_control 两部分。XID 和 ROWID 的解析步骤与 insert 操作中的解析过程相同,解析出的 SQL 语句格式为

DELETE FROM table_name WHERE col_name = value and rowid = ROWID

### 5.4 Oracle 基于事件的恢复

Oracle 基于事件的恢复分为事件视图获取和基于事件恢复两个过程。其中基于事件的恢复又分为数据库初始化装载(对应于检查点)和事件装载(对应于事件节点)两个阶段。数据库的初始化装载是一个物理恢复过程,EBRS 通过调用文件级或块级 CDP 的恢复接口进行恢复;事件装载是逻辑恢复过程,通过调用 Oracle 的数据库操作接口,重做 SQL 语句的方式来完成。

#### 5.4.1 视图获取

EBRS 在服务器端维护了一个事件视图结构,针对每个应用都保存了在被保护期间所产生的关键事件,包括检查点事件和数据更新事件。用户在提出恢复请求时,EBRS agent 首先请求 EBRS server 发送事件视图结构,并在客户端生成事件恢复视图。恢复视图为用户呈现在被保护时间内数据库产生的一致性事件(checkpoint event)和更新事件(sql_op)。

图 9 显示了某个时间段内 Oracle 执行的操作序列。图中显示 Oracle 在该时间段内产生了三个检查点事件以及六个 sql_op 事件,为描述方便,这里每个 sql_op 分别代表一次事务操作,事务编号为 $T_i$($i$ = 1,2,3,4,5,6)。如 $T_1$ 代表第一条事务,并且只包含一条 SQL 语句 insert table a,表示事务 $T_1$ 向表 $a$ 插入一条记录。

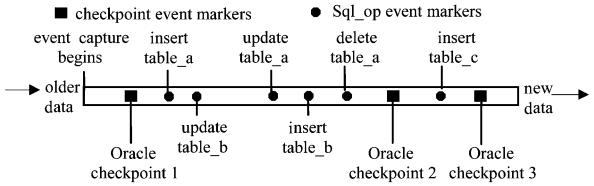


图 9 Oracle 操作序列

Oracle 的事件捕获模块在捕获到这些事件以后,会发送到 EBRS server;EBRS server 根据这些操作序列生成事件视图。图 10 显示了服务器端所生成的事件视图。从图 10 中可以清楚地看到,在检查点节点 1 和检查点节点 2 之间存在 5 个事务节点 $T_1 \sim T_5$;在检查点 2 和 3 之间,存在 1 个事务节点 $T_6$。这些事务节点与上面的操作序列是一一对应的。

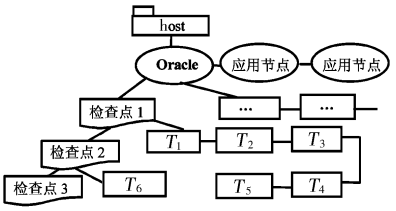


图 10 服务器端事件视图

在恢复阶段,客户端首选请求服务器发送事件视图数据,构建客户端恢复视图,并向用户展示。图 11 显示了客户端向用户展示的恢复视图结构。客户端的恢复视图向用户指明了那些潜在的恢复点,每个恢复点都标记的信息有时间、数据版本书签 bookmark 以及其他描述信息。客户端的恢复视图是根据服务器端的事件视图产生的,与之一一对应,反映了 Oracle 在该段时间内所执行的所有操作。

| time | bookmark | description |
|---|---|---|
| 04/18 12:31:44 | checkpoint1:2538 | |
| 04/18 12:31:58 | $T_1$ | insert table_a |
| 04/18 12:32:02 | $T_2$ | update table_b |
| 04/18 12:32:35 | $T_3$ | update table_a |
| 04/18 12:34:25 | $T_4$ | insert table_b |
| 04/18 12:34:34 | $T_5$ | delete table_a |
| 04/18 12:36:59 | checkpoint2:2539 | |
| 04/18 12:38:11 | $T_6$ | insert table_c |
| 04/18 12:42:10 | checkpoint3:2540 | |
| … | … | |

图 11 客户端的恢复视图

#### 5.4.2 基于事件的恢复

EBRS 根据恢复视图指导用户的恢复过程。用户可以根据自己的需要选择恢复点,如用户可以直接选择一致性数据恢复点(Oracle checkpoint),也可以选择事务恢复点 $T_1 \sim T_6$。针对不同的选择,EBRS 将会采取不同的恢复策略,具体如下:

a)检查点恢复。Oracle 检查点对应的数据版本为一致性数据版本,EBRS 恢复系统通过调用文件级和块级 CDP 的恢复接口,通过物理方式进行恢复。具体地,EBRS agent 获取用户选择的检查点标签,将恢复请求发送到 EBRS server;EBRS server 判断恢复的类型为检查点数据版本,则提取该检查点执行的时间信息,通知 CDP server 构建该时刻的数据版本,然后发送到 EBRS agent,从而完成恢复过程。

b)事务节点恢复。在图 11 的恢复视图中,$T_1 \sim T_6$ 是事务恢复点的标签,标明了该时刻 Oracle 执行的事务操作的信息。对于这些时刻的数据恢复,称之为事务节点恢复。事务节点的恢复分为两个阶段,即数据库初始化和事务装载阶段。在进行恢复之前,首先要获取距离该事务节点最近的检查点节点,然后

提取检查点节点到事件节点之间的事务操作序列。假设用户要恢复到时间点 12:34:34 执行 $T_5$ 操作的时刻,其具体恢复流程如下:

(a)EBRS server 在事件视图中,从 $T_5$ 节点开始向上查找第一个检查点节点(这里为检查点 1),获取该检查点执行的时间 ckp_time:04/18 12:34:44;

(b)以 ckp_time 为参数调用 CDP server 构建该时刻 Oracle 对应的数据版本,并将数据发送到客户端;

(c)EBRS agent 获取到检查点对应的数据,通过数据库初始化装载过程,完成 Oracle 的物理恢复过程,然后启动 Oracle;

(d)EBRS server 以该检查点 1 为起点向下遍历其右孩子节点,提取从检查点 1 到事务 $T_5$ 之间的所有事务操作序列,即事务 $T_1 \sim T_5$ 所包含的所有 SQL 语句序列,并将这些 SQL 语句序列发送到 EBRS agent;

(e)EBRS agent 通过调用 Oracle 的应用接口,顺序执行这些 SQL 语句,即依次执行 insert table a($T_1$)、update table b($T_2$)、update table a($T_3$)、insert table b($T_4$)和 delete table a($T_5$);

(f)事务恢复结束。

## 6　系统测试

系统测试主要测试系统的功能性,EBRS 提供基于事件的恢复机制,在恢复时为用户提供具有应用层语义标签的恢复视图来指导用户的恢复过程。本章主要测试在 Oracle 环境下,事件视图的生成及视图展示。

首先建立两个数据库表 test1 和 test2,然后对数据库进行以下 10 项操作。表 3 显示了测试用例的内容。

表 3　测试用例

| | 操作 | 时间 |
|---|---|---|
| 1 | alter system checkpoint:1 | 10:23:05 |
| 2 | insert into test1 values(1, 'a. txt',sysdate) | 10:23:32 |
| 3 | insert into test1 values(2, 'b. txt',sysdate) | 10:24:14 |
| 4 | insert into test2 values(1,0,1024,sysdate) | 10:25:03 |
| 5 | alter system checkpoint:2 | 10:25:48 |
| 6 | insert into test1 values(3, 'c. txt',sysdate) | 10:26:02 |
| 7 | update test1 set filename = 'd. txt' where id = 2 | 10:30:11 |
| 8 | delete from test1 where id = 'c. txt' | 10:31:26 |
| 9 | insert into test1 values(4, 'f. txt',sysdate) | 10:32:18 |
| 10 | alter system checkpoint:3 | 10:32:30 |

从表 3 中可以看出,测试用例包括三项检查点操作和七项数据库 SQL 语句操作。EBRS 的事件捕获模块截获到的 SQL 语句操作序列如下所示:

```
sqlredo_ test. log
1 XID = 0x0100.1900.AB000000 ROWID = AAADVjAABAAAKUyAAA
2 INSERT INTO test1(ID,FILENAME,CREATE_TIME) VALUES
  (1,'a,txt','2011-04-22 10:23:32');
3 XID = 0x0200.2400.A7000000 ROWID = AAADVjAABAAAKUyAAB
4 INSERT INTO test1( ID,FILENAME,CREATE_TIME) VALUES
  (2,'b. txt','2011-04-22 10:24:14');
5 XID = 0x0800.1900.B9000000 ROWID = AAADVkAABAAAKU6AAA
6 INSERT INTO test2(ID,OFFSET,LENGTH,WRITE_TIME) VALUES
  (1,0,1024,'2011-04-22 10:25:03')
7 XID = 0x0400,2700,AC000000 ROWID = AAADVjAABAAAKUyAAC
8 INSERT INTO test1( ID,FILENAME,CREATE_TIME) VALUES
  (3,'C. txt','2011-04-22 10:26:02');
9 XID = 0x0500.2500.AE000000 ROWID = AAADVjAABAAAKUyAAB
10 UPDATE test1 SET FILENAME = 'd, txt' WHERE ID = 1 AND
FILENAME = 'b,txt' AND CREATE_TIME = '2011-04-22 10:24:14');
11 XID = 0x0600.2100.AE000000 ROWID = AAADVjAABAAAKUyAAC
12 DELETE FROM test1 WHERE ID = 3 AND FILENAME = 'C. txt' AND
  CREATE TIME = '2011-04-22 10:26:02';
13 XID = 0x0800.1C00.B8000000 ROWID = AAADVjAABAAAKUyAAD
14 INSERT INTO test1( ID,FILENAME,CREATE_TIME) VALUES
  (2,'f. txt','2011-04-22 10:32:18');
```

从以上序列可以看出,每一项 SQL 语句都以“XID =”和“ROWID =”开始,其中 XID 表示事务编号,ROWID 为更新操作对应的行号。根据提取出的 SQL 语句序列,服务器端生成事件视图,在恢复时,为用户提供恢复视图。图 12 显示了恢复时客户端获取的恢复视图。

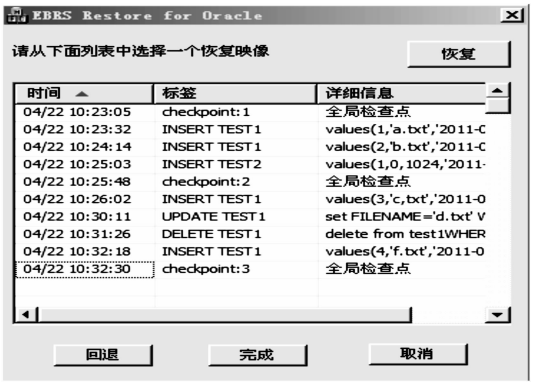


图 12　恢复视图

从图 12 可以看出,恢复视图提供了一个恢复映像列表,其中每一项都描述了一个潜在的数据库重建点。恢复视图包括恢复映像的时间、恢复映像的语义标签及标签的详细描述信息。测试结果表明,系统在 Oracle 环境下进行事务日志解析和基于事务的恢复过程,功能良好,实现了预定的目标。

## 7　结束语

本文针对现有 CDP 系统在恢复点选择、数据一致性和数据恢复粒度三个方面存在的问题,提出了基于事件驱动的 CDP 恢复解决方案,并在 Oracle 环境下实现了 EBRS 的关键功能。

对整个系统的综合分析,仍然有很多工作还需继续进行:

a)对 Oracle 事务操作的捕获需要进一步完善,日志分析需要能够处理更复杂的数据库操作,如行链接、簇等,需要提供对更多 Oracle 版本的支持。

b)需要实现对更多主流应用的支持。EBRS 是面向多应用的系统,而对不同的应用,事件捕获和装载模块需要不同的技术,因此需要研究每个应用的特点,并提供相应的基于事件恢复的功能。

**参考文献:**

[1] 王德军. 容灾技术研究[D]. 武汉:武汉大学,2004.

[2] SNIA Data Management Forum. An overview of today's continuous data protection(CDP) solutions[EB/OL]. 2012. http://www. snseurope. com/supplements/snia-1-4. pdf.

[3] WANG Yan-long,LI Zhan-huai,XU Juan. Protecting and recovering database systems continuously[C]//Proc of the 9th Joint Asia-Pacific Web and the 8th International Conference on Web-age Information Management Conference on Advances in Data and Web Management. Berlin:Springer-Verlag,2007:765-776.

[4] 王丽娜,熊琦,王德军,等. 基于文件过滤驱动的连续数据捕获技术及其容灾应用研究[J]. 小型微型计算机系统,2011, 32(3): 477-482.

(下转第 475 页)

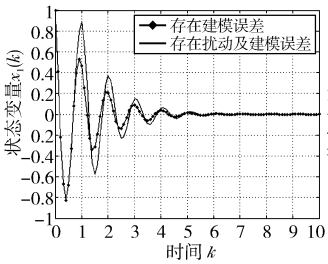


图 4　时延大于一个采样周期时延状态变量 $x_1$ 的响应曲线

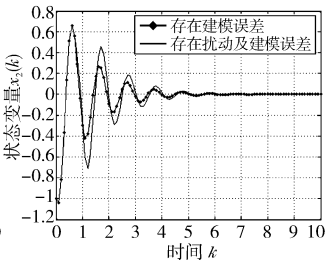


图 5　时延大于一个采样周期时延状态变量 $x_2$ 的响应曲线

通过图 2 ~5 可知,考虑存在建模误差情况下的网络控制系统是渐近稳定的,说明所设计的控制器对于建模误差所引起的不确定性具有一定的抑制能力。分别对比时延小于一个采样周期及大于一个采样周期的状态响应可知,考虑同时存在建模误差及未知扰动时,虽然系统的超调量、上升时间、调节时间有所变化,但系统在短时间内仍然可以恢复到稳定的平衡点(0,0),说明所设计的控制器可以处理同时存在的建模误差及未知扰动,对建模误差和未知扰动所引起的不确定性具有较强的抑制能力。对比文献[11,13]可知,该控制器设计方法不仅可以解决文献[11]中控制系统中不确定时延小于一个采样周期的情况,而且可以处理文献[13]中不确定时延大于一个采样周期的情形;同时由于考虑了建模存在的模型误差,该方法达到了更高的控制精度。

## 4　结束语

本文针对存在不确定时延的网络控制系统进行建模和控制器设计。由于网络控制系统的构成,使得系统中不可避免地存在时延,在此基础上,综合考虑工程实际中所面临的未知扰动、建模误差、系统控制输入直接作用于输出等未被考虑的问题。在已知闭环系统控制律的前提下,构造 Lyapunov 函数,运用线性矩阵不等式(LMI)理论,得出了满足给定 $H_\infty$ 性能指标网络控制系统渐近稳定的充分条件,完成控制器的设计。对时延小于一个周期和大于一个周期的网络控制系统进行了仿真,仿真结果证明本文所提出的 $H_\infty$ 鲁棒控制器设计方法,对时延是否大于一个采样周期没有限制,不需要满足已知的某种随机分布,而且对提高网络控制系统的稳定性并进一步改善控制精度具有一定的作用。

**参考文献:**


[1] LIU An-dong, YU Li, ZHANG Wen-an. One-step receding horizon $H_\infty$ control for networked control systems with random delay and packet disordering[J]. ISA Transactions, 2011, 50(1): 44-52.

[2] HESPANHA J P, NAGHSHTABRIZI P, Xu Yong-gang. A survey of recent results in networked control systems[J]. Proceedings of the IEEE, 2007, 95(1): 138-162.

[3] SHI Yang, YU Bo. Robust mixed $H_2/H_\infty$ control of networked control systems with random time delays in both forward and backward communication links[J]. Automatica, 2011,47(4): 754-760.

[4] ZHANG Wei, BRANICKY M S, PHILLIPS S M. Stability of networked control systems[J]. Control Systems, 2001, 21(1): 84-99.

[5] LUCK R, RAY A. An observer-based compensator for distributed delays[J]. Automatica,1990,26(5):903-908.

[6] NILSSON J. Real-time control systems with delays[D]. Lund: Department of Automatic Control, Lund Institute of Technology, 1998.

[7] LIAN F L, MOYNE J, TILBURY D. Network design consideration for distributed control systems[J]. IEEE Trans on Control Systems Technology, 2002,10(2): 297-307.

[8] HU Shou-song, ZHU Qi-xin. Stochastic optimal control and analysis of stability of networked control systems with long delay[J]. Automatica, 2003, 39(11): 1877-1884.

[9] CHAE S, HUANG Dan, NGUANG S K. Robust partially mode delay dependent $H_\infty$ control of discrete-time networked control systems[J]. International Journal of Systems Science, 2012, 43(9): 1764-1773.

[10] 樊卫华,蔡骅,吴晓蓓,等. 一类时延网络控制系统的 $H_\infty$ 鲁棒控制[J]. 兵工学报,2006,27(1):188-192.

[11] 于飞,于洁,魏克泰,等. 基于网络控制系统的 $H_\infty$ 鲁棒控制[J]. 系统工程与电子技术,2009,31(4):927-929.

[12] ZHANG Wen-an, YU Li, YIN Shu. A switched system approach to $H_\infty$ control of networked control systems with time-varying delays[J]. Journal of the Franklin Institute, 2011, 348(2):165-178.

[13] 王岩,刘涛,孙增圻,等. 不确定长时延网络控制系统的 $H_\infty$ 鲁棒控制[J]. 控制工程,2011,18(2):210-214.

[14] 赵虹,吴敏,刘国平. 带时变时延的网络化控制系统控制器设计方法[J]. 信息与控制,2006,35(3): 325-328.

[15] XIE Li-hua. Output Feedback $H_\infty$ control of systems with parameter uncertainly [J]. International Journal of Control, 1996, 63(4): 741-750.


(上接第 471 页)


[5] 李旭,谢长生,杨靖,等. 一种改进的块级连续数据保护机制[J]. 计算机研究与发展,2009,46(5):762-769.

[6] 王超,李战怀,胡娜,等. 一种低恢复时间低存储空间的块级连续数据保护机制[J]. 西北工业大学学报,2011,29(3):429-434.

[7] 生拥宏,汪东升,鞠大鹏,等. 一种卷级连续数据保护一致点插入方法[J]. 高技术通讯,2010,20(11):1101-1107.

[8] SHENG Yong-hong, XU Dan, WANG Dong-sheng. A high effective indexing and retrieval method providing block-level timely recovery to any point-in-time[C]//Proc of the 5th IEEE International Conference on Networking, Architecture and Storage. Washington DC: IEEE Computer Society, 2010:41-50.

[9] 李琥,谭毓安,李元章,等. 一种块级连续数据保护系统的快速恢复方法[J]. 北京理工大学学报,2011,31(6):679-684.

[10] 李琥,谭毓安,李元章,等. 一种连续数据保护系统的快照方法[J]. 软件学报,2011,22(10):2523-2537.

[11] 王福波. 连续数据保护存储模型设计与实现[D]. 武汉:武汉大学, 2008.

[12] 王丽娜,武开智,王德军,等. 一种面向连续数据保护的分布式存储模型研究[J]. 小型微型计算机系统,2011,32(8):1587-1592.

[13] 王德军. 面向业务连续性的容灾关键技术研究[D]. 武汉:武汉大学,2009.

[14] LITCHFIELD D. Oracle forensics, part 1: dissecting the redo logs[EB/OL]. (2007-03-21). http://www.dcs.co.jp/security/NGS_freedownloads/OracleForensicsPart1_Dissecting_theRedologs.pdf.

[15] 杨泽平,龚正良,万锋,等. Oracle 8i 数据库日志文件的分析与恢复[J]. 计算机应用与研究,2005,22(6):108-110.