基于网络磁盘结构的 Global Address Manager 设计

赵延红[1],掛下哲郎[2]
(1. 西安交通大学,陕西 西安 710061;
2. 日本佐贺大学 理工学部,日本 佐贺 840-8502)

摘 要:为构成高性能的数据库,磁盘阵列的活用是很有必要的。其中尤为重要的是实现磁盘阵列之间负载的均衡。为此,设计了网络磁盘结构系统。该系统在磁盘阵列高负载的情况下也能够进行动态的负载均衡,并且具有优良的耐故障性。文中对网络磁盘结构构成要素中的 Global Address Manager (GAM)进行了设计。首先定义了必要的信息形式,信息的传送与 Bus 的宽度相符呈分割进行;然后,在 GAM 的设计方法中定义了输入/输出、数据的形式以及 GAM 内部数据的构造;在此基础上,制作了 GAM 的算法及测试用例设计。

关键词:磁盘阵列;动态负载均衡;结构设计;测试用例设计
中图分类号:TP31 **文献标识码**:A **文章编号**:1673-629X(2015)08-0013-07
doi:10.3969/j.issn.1673-629X.2015.08.003

Design of Global Address Manager Based on Net Disk Architecture

ZHAO Yan-hong[1], KAKESHITA Tetsuro[2]
(1. First Affiliated Hospital of Medical College, Xi'an Jiaotong University, Xi'an 710061, China;
2. Department of Information Science, Faculty of Science and Engineering, Saga University,
Saga 840-8502, Japan)

Abstract: For constructing high-performance database servers, it is essential to make the best use of parallel disks. It is important to execute load balancing among disks for parallel disks. Hence, design the net disk architecture for this purpose. This system can perform efficiently for dynamic load balancing even though the disk arrays are under heavy load conditions. The architecture also achieves high fault tolerance. In this paper, design the Global Address Manager (GAM) in the net disk architecture. At first, define the necessary message mode. Messages are splitted to transmit through the fixed size bus. Then, define the input/output, data pattern and internal GAM data structure in GAM design method. Based on these, the algorithms of GAM are developed and the test cases are designed.

Key words: disks arrays; dynamic load balancing; architecture design; test case design

0 引 言

随着中央处理器 CPU 的处理速度飞速增长,内存的存取速度也随之大幅增加,而数据存储器磁盘的存取速度相对发展较为缓慢,从而造成整个存储设备的运行速度不能和其他硬件系统相匹配,形成计算机系统数据传输速度的瓶颈,降低了计算机系统的整体性能。如果不能有效提高磁盘的存取速度,CPU、内存及磁盘间的不平衡将使 CPU 及内存的改进形成浪费。在大规模存储系统中,为了提高存储的可靠性并改善 I/O 性能,通常采用独立磁盘冗余阵列(Redundant Arrays of Independent Disks, RAID)[1-2]。RAID 把多个磁盘联合起来,形成统一的逻辑存储设备,常用技术有条带化、磁盘镜像和错误修正。如 RAID4、RAID5、RAID6,把数据条带化后,分散存储到阵列中不同的磁盘上以保证并行性,并采用冗余校验,在保证数据可靠性的同时,可获得大容量和高数据传输率[3],但是 RAID 不能保证磁盘间的动态负载均衡。因此,在此研究的基础上首先提出了动态数据再配置[4]。通过动态数据再配置,负载最大的磁盘存储的数据动态地向负载最小的磁盘移动。进而,设计了网络磁盘结构[5]。该结构是利用动态数据的再配置实现磁盘阵列间的负载均衡,并且基于硬件的多重化,实现系统的高信赖

收稿日期:2014-09-22 **修回日期**:2014-12-25 **网络出版时间**:2015-07-21
基金项目:日本文部科学省科学研究基金(2003)
作者简介:赵延红(1965-),女,硕士,西安交通大学医学院第一附属医院网络信息部高级工程师,研究方向为网络磁盘结构和医院信息化建设;掛下哲郎,博士,日本佐贺大学科学与工程学院信息科学系教授,研究方向为数据库和软件工学。
网络出版地址:http://www.cnki.net/kcms/detail/61.1450.TP.20150721.1448.042.html

性。网络磁盘结构是由复数的磁盘阵列,进行地址变换的组件和控制动态负载均衡的组件,基于网状网络连接起来。

图 1 是网络磁盘结构图。

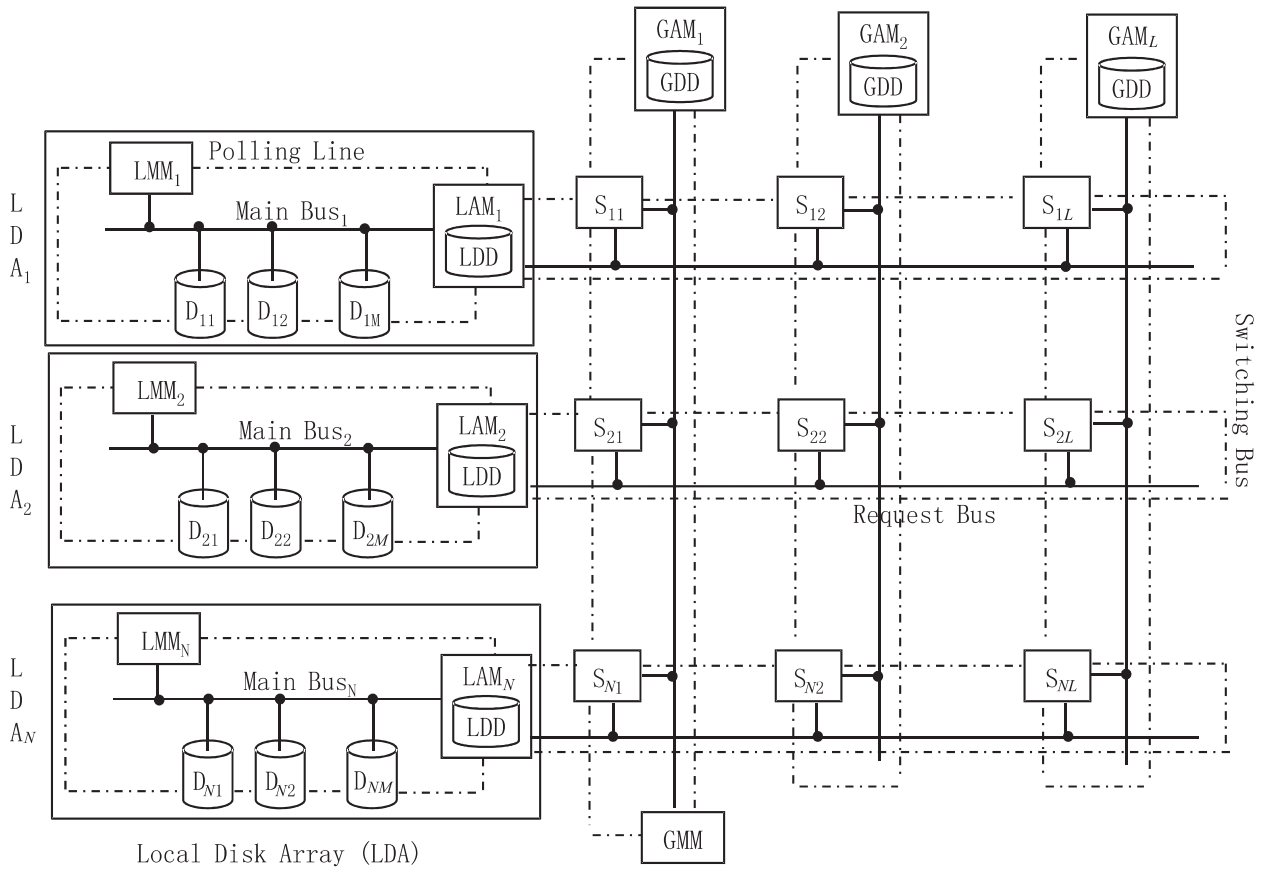

图 1 网络磁盘结构

图中的略称说明为:GAM(Global Address Manager),GDD(Global Data Dictionary),GMM(Global Migration Manager),S(Switch),LAM(Local Address Manager),LDD(Local Data Dictionary),LMM(Local Migration Manager),D(Disk),LDA(Local Disk Array)。

网络磁盘结构能够进行磁盘阵列内和磁盘阵列间的动态再配置,实现系统的负载均衡。

文中面向网络磁盘结构的实用化,对网络磁盘结构的构成要素中的 GAM 进行设计。在此之前曾进行了 Switch 的逻辑电路的设计[6],以及 LMM,GMM 的设计[7-8]。由于 Switch 是 2 个总线间信息传送的逻辑电路,而对 LMM,GMM,GAM 的处理比 Switch 复杂得多,用逻辑电路构成将无法实现。所以,LMM,GMM,GAM 计划使用微程序设计来实现[9-12]。文中对 GAM 的数据构造和算法进行了设计,并且为了发现设计的不完备,使用因果图[13-14]进行测试用例设计。

1 网络磁盘结构的构成

在网络磁盘结构中外部的访问要求由任意的 GAM 接受,数据由磁盘 D_{ij} 分散保存。

GAM:用 GDD 把访问要求的逻辑地址转换成一对 LDA 物理地址和 LDA 内部逻辑地址,然后把要求信息发送给 LDA,同时从 GMM 接受再配置开始命令,实行 LDA 间的动态再配置。

GMM:监视各个 LDA 的状态及负载,并把信息传送给 GAM,在 LDA 间发送再配置开始/停止命令。

LDA:由 LAM,LMM,D 构成,通过 Main Bus 连接。

LAM:把被传送要求信息的 LDA 内部逻辑地址转换成 LDA 内部物理地址,并把访问要求发送给相应的磁盘。这个组件内的地址变换,并不变更 LDA 外部数据的保存地址。同时,从 LMM 接受再配置开始命令,实行 LDA 内部的动态再配置。

LMM:监视 Main Bus 上的传送信息,自动计算各个磁盘的负载,并把这个信息传送给 LAM,在 LDA 内部发送再配置开始/停止命令。

网状网络在 Switching Bus 和 Request Bus 之间由 Switch 连接的控制线路,实行各个组件间的通信。

2 网络磁盘结构的动作

网络磁盘结构的负载均衡分为 LDA 内部本地动态再配置和 LDA 间全局动态再配置 2 个步骤进行。

GMM 控制全局范围的动态再配置。动态再配置和通常的数据访问处理同时实行。首先说明通常时的 Read(Write)流程,其次说明全局动态再配置时的动作。

2.1　通常时的数据访问处理

(1)来自客户的 Read(Write)要求由 GAM_i 接受,并把逻辑地址变换为 LDA 内部的逻辑地址和存放数据的 LDA 的物理地址。

(2) GAM_i 通过 $Switch_{ji}$,向 LAM_j 传送 ReadB(WriteB),LAM_j 把接收信息的逻辑地址转换为 LDA 内部的物理地址,并把 Read(Write)信息送往相应的磁盘 D。

(3) LAM_j 把从磁盘 D 接收来的信息,通过 $Switch_{ji}$ 送回 GAM_i。

(4) GAM_i 把回答信息送给客户。

2.2　全局动态再配置

各个 LAM 计算 LDA 的负载数,定期传送给 GMM。GMM 决定负载最大(负载最小)的 LDA 的编号 LDAMax(LDAMin),如果 LDAMax 的负载数/LDAMin 的负载数$>\delta$($\delta \geq 1$)成立,GMM 通过网状网络发送再配置开始命令,然后全部的 GAM 开始转入再配置模式。再配置模式下,GAM 实现全局的动态再配置。LDAMax 的负载数/LDAMin 的负载数$\leq \lambda$ ($1 \leq \lambda < \delta$)成立,GMM 给 GAM 发送再配置停止命令,然后各个 GAM 回归通常时的模式。

2.2.1　再配置模式时的 Read 处理

如果 GAM_i 发送的 Read 要求是给 LDAMax 的,GAM_i 把 ReadMig 信息传送给 LDAMax,LDAMax 把 ReadMigAck 信息在网状网络上播放后,这个信息由 GAM_i 和 LDAMin 接受,LDAMin 把数据保存以后,通过网状网络把 MigComplete 信息传送给 LDAMax 和 GAM_i,LDAMax 响应以后把旧的数据消去。GAM_i 接受 MigComplete 信息以后,使用 GDD 把地址转换为 LDA 的物理地址。

2.2.2　再配置模式时的 Write 处理

如果 GAM_i 发送的 Write 要求是给 LDAMax 的,GAM_i 就把 WriteMig 信息传送给 LDAMin,LDAMin 把接受的数据保存以后,再把 WriteMigAck 信息发送回网状网络,这个信息由 LDAMax 和 GAM_i 接受,LDAMax 把旧的数据消去,GAM_i 使用 GDD 把地址转换为 LDA 的物理地址。

因此,动态再配置的特点有以下三点:

(1)把负载最大的 LDA 的数据移动到负载最小的 LDA 上,实现了 LDA 间的动态负载均衡;

(2)数据访问和数据的再配置同时进行,减少了网状网络、LDAMax 等的系统开销;

(3)被访问数据再配置容易实现,使得最小限度的数据移动就能够实现负载均衡。

3　信息的数据构造

GAM 的设计方法中首先要确定的就是信息的数据构造。根据网络磁盘结构的动作,决定 User-GAM 间传送信息的数据构造、网状网络间传送信息的数据构造。

3.1　User-GAM 间的信息

网络磁盘结构中 User-GAM 间传送的信息有 4 种:User 来的读出要求 UserRead;User 来的写入要求 UserWrite;给 User 发的读出要求结果 UserReadAck;给 User 发的写入要求结果 UserWriteAck。信息的种类以 3 位定义。User-GAM 间交换信息的内容包含在信息的先头字段和信息的终端字段中,进行分割信息的传送。信息先头字段的信息内容部分,32 bit 总线宽度相符,信息先头字段分割为 3 个标志。标志 1 包括信息类型 MessageType(3 位)和 GAM 地址 GAMAddress(4 ~ 7 位);标志 2 包括数据的逻辑地址(8 ~ 39 位);标志 3 包括信息号码 MessageNumber(40 ~ 71 位)。数据部分是信息先头字段送出以后传送的,信息终端字段的信息内容全部为 1。

3.2　网状网络上的信息

网络磁盘结构中网状网络上传送的信息有 11 种:再配置开始命令 MigStart;再配置停止命令 MigStop;再配置完成命令 MigComplete;通常时读出要求结果 ReadAck;再配置时读出要求结果 ReadMigAck;通常时写入要求结果 WriteAck;再配置时写入要求结果 WriteMigAck;通常时读出要求 Read;再配置时读出要求 ReadMig;通常时写入要求 Write;再配置时写入要求 WriteMig。信息的种类以 5 位定义。和 User-GAM 间的信息同样,网状网络上传送的信息也是分割发送和接收的。信息的先头字段分割为 3 个标志,标志 1 包括信息类型 MessageType(1 ~ 5 位)、LDA 地址(通常时)/负载最大 LDA 地址(再配置时)LDAAddress(6 ~ 9 位)、负载最小 LDA 地址 LDAMinAddress(10 ~ 13 位)、GAM 地址 GAMAddress(14 ~ 17 位)、LDA 内逻辑地址 LDALAddress(18 ~ 21 位);标志 2 是数据的逻辑地址 Address(22 ~ 53 位);标志 3 是信息号码 MessageNumber(54 ~ 85 位)。

4　GAM 的设计方法

在网络磁盘结构和动作、User-GAM 间传送的信息及网状网络传送的信息的数据构造叙述的基础上,确定基于网络磁盘结构的 GAM 的设计方法,设计方法中明确了输入/输出线和数据形式以及内部数据。

GAM 的输入/输出如图 2 所示。

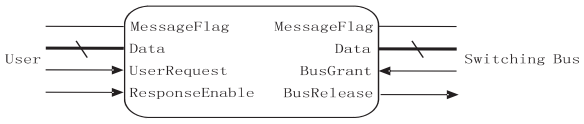

图 2 GAM 的输入/输出

MessageFlag 是为了识别信息先头字段/终端字段和数据部分的,User 侧的信号线中 UserRequest,ResponseEnable 是 GAM 从 User 收到要求信息以及给 User 返回结果信息时使用的。Switching Bus 侧连接的信号线中 BusGrant 和 BusRelease 是根据轮询实现总线调停使用的。Switching Bus 侧 GAM 外连接的 GMM 等组件是它们之间总线调停所必要的。各组件通过 BusGrant 取得总线利用权时,信息就可以通过 Switching Bus 播放,播放结束以后,通过 BusRelease 把总线利用权交给其他组件。GAM 的各个信号线的详细说明见表 1。GAM 的内部数据详见表 2。数据形式使用正则表达式记述。|是选择结构,* 表示各个重复结构。

5 GAM 的设计

本节进行 GAM 的设计。最初表示了 GAM 的全体构造。因为 GAM 的机能复杂,预定用微程序设计实现。包括 GAM 系统参数的初期设定,从 User、Switching Bus 收到信息的处理以及给 User、Switching Bus 发送信息的处理,在此基础上构建算法。

5.1 GAM 的全体构造

GAM 的全体构造如图 3 所示。

表 1 GAM 的输入/输出线和数据形式

信号线	Input/Output	数据形式	说明
MessageFlag	I/O	[0|1]	信息开始/终了标志/数据
Data	I/O		信息内容是记述先头字段、各种数据。终端字段时,全部为 1
UserRequest	I	[True|False]	User 给 GAM 发送信息命令
ResponseEnable	I	[True|False]	User 从 GAM 收到信息命令
BusGrant	I	[True|False]	取得总线利用权
BusRelease	O	[True|False]	总线利用权释放

表 2 GAM 的内部数据

名称	数据形式	说明
GAMTempQueue	(信息)*	临时保存从 User 接收的信息
DataDictionary	(数据的逻辑地址+LDA 号码+LDA 内逻辑地址)*	管理各个数据的逻辑地址、LDA 号码、LDA 内逻辑地址的对应表
SW_BusReceiveQueue	(信息)*	保存从 Switching Bus 接收的信息
MigrationStatus	(MigFlag+LDAMax+LDAMin)	保存再配置信息
SwitchingBusQueue	(信息)*	保存给 Switching Bus 播放的信息
ResultQueue	(信息)*	临时保存给 User 发送的要求结果

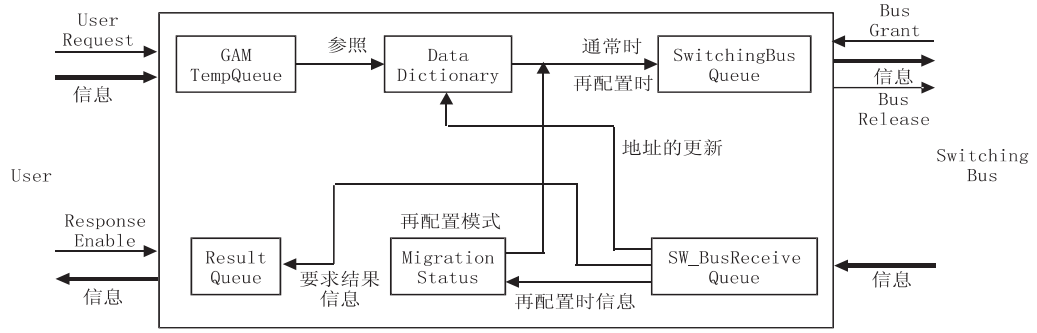

图 3 GAM 的全体构造

User 侧的 UserRequest 是 1 时,要求信息在 GAMTempQueue 中保存。GAM 用 GDD 把访问要求的逻辑地址转换成一对 LDA 物理地址和 LDA 内部逻辑地址。变换为通常时或再配置时信息,信息保存在 SwitchingBusQueue。BusGrant 输入后,SwitchingBusQueue 中保存的全部信息向 Switching Bus 播放。从 Switching Bus 侧传送的信息保存在 SW_BusReceiveQueue。如果信息是 MigStart,写入再配置信息;如果信息是 MigStop,返回通常时模式;如果信息是 MigComplete,更新 GDD;如果是要求结果信息,则转换为 User 侧的信息,信息保存在 ResultQueue。ResponseEnable 变为 1 时,ResultQueue 中保存的要求结果信息发送给用户。

5.2 系统参数的初期设定

(1)清空 GAMTempQueue,SW_BusReceiveQueue,

SwitchingBusQueue,ResultQueue。

(2)MigrationStatus 的 MigFlag 设置为 0。

(3)DataDictionary 从硬盘加载到 GAM 的主存储器。

5.3　信息的接收处理

GAM 接收 User、Switching Bus 来的信息,信息的处理程序如下:

1)User 侧。

User 把接收的 UserRead,UserWrite 信息一时保存在 GAMTempQueue,进行必要的转换后,发送给 SwitchingBusQueue。GAMTempQueue、SwitchingBusQueue 中数据的读写是以字段为单位进行。

(1)UserRequest 为 1 时,User 把接收的信息的各个字段依次写入 GAMTempQueue。

(2)GAMTempQueue 把该信息的种类,GAM 号码,数据的逻辑地址,信息的号码读入。

(3)使用 DataDictionary 把数据的逻辑地址转换为 LDA 号码和 LDA 内部的逻辑地址。

(4)通常的模式或者是再配置模式 LDA 的号码和负载最大的 LDA 的号码(LDAMax)不一致时,网状网络上的先头字段生成,并写入 SwitchingBusQueue(通常时的数据访问)。这时信息的种类是 Read 或 Write。同时把 LDA 号码写入 LDAAddress 中,并把 LDAMinAddress 设置为空。

(5)如果不是以上的情况,把网状网络生成的先头字段写入 SwitchingBusQueue(动态的数据再配置)。这时信息的种类是 ReadMig 或 WriteMig。同时把 LDA 号码(注:MigrationStatus 和 LDAMax 保持一致)写入 LDAAddress,LDAMinAddress 设置为负载最小的 LDA 号码(MigrationStatus 保持)。

(6)把先头字段以后的各个字段依次写入 SwitchingBusQueue。

(7)在 GAMTempQueue 中删除该信息。

2)Switching Bus 侧。

Switching Bus 上传送的信息(MigStart,MigStop,MigComplete,ReadAck,WriteAck,ReadMigAck,WriteMigAck)一时保存在 SW_BusReceiveQueue,并在 MigrationStatus,DataDictionary,ResultQueue 中写入必要的信息。SW_BusReceiveQueue、ResultQueue 中信息的读写以字段单位进行。

(1)在 Switching Bus 上传送信息的各个字段依次读入,并写入 SW_BusReceiveQueue。

(2)SW_BusReceiveQueue 保存该读入信息的种类。

(3)如果信息的种类是 MigStart,进行以下处理。

· 把 MigFlag 设置为 1。

· 更新 LDAMax,LDAMin。

(4)如果信息的种类是 MigStop,把 MigFlag 设置为 0。

(5)如果信息的种类是 MigComplete,更新 DataDictionary 中的主存储和二次存储保存的 LDA 号码。

(6)如果信息的种类是 ReadAck,WriteAck,ReadMigAck,WriteMigAck 中的任何一种,进行以下处理。

· 读入该信息的 GAM 号码、数据的逻辑地址、信息的号码。

· 信息的种类转换为 User-GAM 间的形式。

· 生成 User-GAM 间的信息,写入 ResultQueue。

(7)在 SW_BusReceiveQueue 中删除该信息。

5.4　信息的发送处理

GAM 给 User、Switching Bus 发送信息,信息的处理程序如下。

1)User 侧。

ResultQueue 中保存的信息为 UserReadAck,UserWriteAck。当 ResponseEnable 输入后,发送给 User 的数据以字段为单位进行读出。ResponseEnable 为 1 时,在 ResultQueue 中保存的全部信息发送给 User。

2)Switching Bus 侧。

SwitchingBusQueue 中保存的信息为 Read,Write,ReadMig,WrieMig。BusGrant 输入后,从 Switching Bus 读出的数据以字段单位进行。

(1)SwitchingBusQueue 的全部信息向 Switching Bus 播放。

(2)BusRelease 设置为 1。

6　测试用例设计

通过设计测试用例找出 GAM 设计的不足以及歧义。因此使用原因-结果图制作了有效测试用例。使用有效测试用例制作原因-结果图是通过以下 4 步进行:(1)列举原因和结果;(2)原因和结果间的关联识别;(3)确认原因的尽可能的组合;(4)测试用例的抽出。最后设计无效的测试用例。

6.1　原因和结果的列出

GAM 的影响原因为 GAM 输入的信息和 GAM 的内部状态,原因影响因素共有 13 种:编号 1 是输入 ResponseEnable;编号 2 是输入 UserRequest;编号 3 是输入 UserRead;编号 4 是输入 UserWrite;编号 5 是 GAM 是再配置模式且 LDAMax 和要求对象的 LDA 一致;编号 6 是输入 MigStart;编号 7 是输入 MigStop;编号 8 是输入 MigComplete;编号 9 是输入 ReadAck;编号 10 是输入 ReadMigAck;编号 11 是输入 WriteAck;编号 12 是输入 WriteMigAck;编号 13 是输入 BusGrant。输入的信息分为来自 User(编号 1 ~4)和 Switching Bus(编号

6 ~ 13)两个类型。编号 5 表示的是 GAM 的内部状态。

另一方面,GAM 的影响结果为 GAM 内部状态的变化和 GAM 给外部输入的结果,结果影响因素共有 11 种:编号 80 是 ResultQueue 的信息给 User 输出;编号 81 是 Read 信息在 SwitchingBusQueue 中追加;编号 82 是 ReadMig 信息在 SwitchingBusQueue 中追加;编号 83 是 Write 信息在 SwitchingBusQueue 中追加;编号 84 是 WriteMig 信息在 SwitchingBusQueue 中追加;编号 85 是转换为再配置模式;编号 86 是转换为通常模式;编号 87 是更新 GDD;编号 88 是 UserReadAck 在 ResultQueue 中追加;编号 89 是 UserWriteAck 在 ResultQueue 中追加;编号 90 是 SwitchingBusQueue 的信息输出到 Switching Bus,然后输出 BusRelease。编号 81 ~ 89 是 GAM 内部状态的变化,编号 80 和 90 分别是给 User 和 Switching Bus 的输出结果。

6.2 原因和结果间关联的识别

影响 GAM 的原因和结果并不是独立发生的。例如,原因编号 1 和 2 不存在同时发生的可能性。因此类似的排斥约束,同样适用于原因的 3 和 4,6 ~ 12,这个在图 4 中以 E(Exclusive)所示。此外,原因的 2 是 3 和 4 成立的一个前提条件。在图 4 中,这样的约束用 R(Require)表示。原因和结果之间有因果关系。在图 4 中原因(节点 1 ~ 13 编号)、中间节点(节点 30 ~ 40 编号)、结果节点(节点 80 ~ 90 编号)通过分支相连,通过必要的对应组合 AND,OR 和 NOT 的结合,确认涵盖了原因的所有可能的组合。

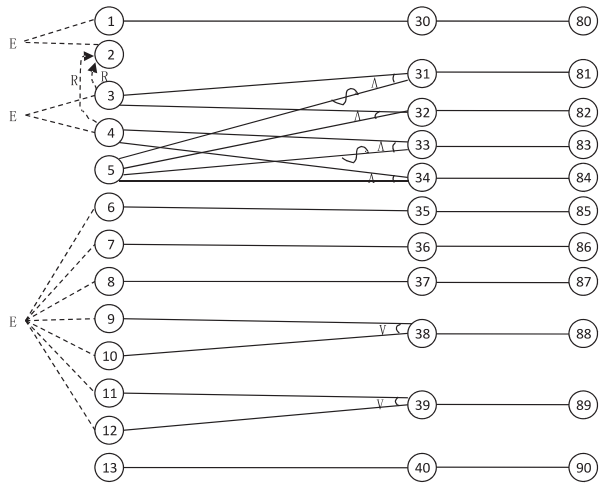

图 4 原因-结果图

为了确认图 4 的原因-结果图包括了所有的原因的组合,原因-结果图分割为以下 3 个部分。

(1)节点 1 ~ 5 编号,30 ~ 34 编号,80 ~ 84 编号;

(2)节点 6 ~ 12 编号,35 ~ 39 编号,85 ~ 89 编号;

(3)节点 13 编号,40 编号,90 编号。

这些连接成分是互相独立的原因,同时考虑到排斥约束和前提约束,这样每一个连接成分的所有可能的原因就容易确定了。

6.3 测试用例的提取

通过图 4 中连接的每一个成分来提取测试用例。从原因-结果图中相互独立的原因(的组合)和结果的组合提取后,可以被转换成一个测试用例。考虑以下有效测试用例的 13 种方式。

(1)输入 ResponseEnable 后,ResultQueue 的信息发送给 User(原因:编号 1,结果:编号 80)。

(2)输入 UserRead 后,Read 在 SwitchingBusQueue 中追加(原因:编号 2,3,5,结果:编号 81)。

(3)输入 UserRead 后,ReadMig 在 SwitchingBusQueue 中追加(原因:编号 2,3,5,结果:编号 82)。

(4)输入 UserWrite 后,Write 在 SwitchingBusQueue 中追加(原因:编号 2,4,5,结果:编号 83)。

(5)输入 UserWrite 后,WriteMig 在 SwitchingBusQueue 中追加(原因:编号 2,4,5,结果:编号 84)。

(6)输入 MigStart 后,转换为再配置时模式(原因:编号 6,结果:编号 85)。

(7)输入 MigStop 后,转换为通常时模式(原因:编号 7,结果:编号 86)。

(8)输入 MigComplete 后,更新 GDD(原因:编号 8,结果:编号 87)。

(9)输入 ReadAck 后,UserReadAck 在 ResultQueue 中追加(原因:编号 9,结果:编号 88)。

(10)输入 ReadMigAck 后,UserReadAck 在 ResultQueue 中追加(原因:编号 10,结果:编号 88)。

(11)输入 WriteAck 后,UserWriteAck 在 ResultQueue 中追加(原因:编号 11,结果:编号 89)。

(12)输入 WriteMigAck 后,UserWriteAck 在 ResultQueue 中追加(原因:编号 12,结果:编号 89)。

(13)输入 BusGrant 后,SwitchingBusQueue 中的信息在 Switching Bus 上播放,然后输出 BusRelease(原因:编号 13,结果:编号 90)。

6.4 无效测试用例的设计

以上使用原因-结果图设计了有效测试用例。为了使 GAM 的可靠性得到提高,需要增加一个无效测试用例,用于异常处理、错误处理。为了设计无效测试用例,首先要系统列出 GAM 的异常输入和内部状态。GAM 的输入信息已经定义过了,在此列出无效输入的信息。

(1)User-GAM 间传送的信息和网状网络上传送的信息中定义以外的指令输入时。

(2)不存在的地址被指定时。这个对应地址的 GAM 号码,数据的逻辑地址,LDA 号码,负载最大 LDA 号码,负载最小 LDA 号码,LDA 内部逻辑地址。

(3)不存在的信息号码被指定。

(4)指定序列的指令没有按照顺序到达。

(5)信息的终端字段没有到达。

另外,信号线 MessageFlag, UserRequest, ResponseEnable, BusGrant 是一位定义的,假设没有错误。而且数据部分是任意的序列,假设没有错误。GAM 输入的异常信息对应的测试用例,包括上面对应的各个场合。其中(1)~(4)中,GAM 没有输出和状态变化。

(1)定义以外的指令码输入的场合。

(2)GAM 号码是定义以外的地址的场合。

(3)UserReadAck 的信息号码和 UserRead 的信息号码不一致的场合。

(4)UserRead 命令和输入的 UserReadAck 不对应的场合。

(5)没有终端字段的 UserWrite 命令输入时,GAMTempQueue 依次保存已到达的数据直到存满为止。GAMTempQueue 中的数据存满以后忽略随后的数据。

GAM 的内部状态在图 3 和表 2 的各个数据结构中已经被定义,这些数据结构的异常做如下考虑。

(1)数据结构是满的。

(2)数据内容相互矛盾。

GAM 的内部状态异常对应的测试用例,包括上面的场合对应为:

(1)SwitchingBusQueue 的数据存满时,新的要求信息忽略。

(2)再配置模式并且 LDAMin 指定的 LDA 不存在的场合,LDAMax 对应的 UserRead 命令到达时,没有 ReadMig, Read 在 SwitchingBusQueue 中追加。

使用类似的技术可设计其他异常的测试用例。

7　结束语

网络磁盘结构能够实现磁盘阵列 LDA 内和 LDA 间的负载均衡,并且系统具有良好的耐故障性。文中从网络磁盘结构的实用化考虑,进行了 GAM 的设计。在定义了 User-GAM 间的信息和网状网络上传送的信息的数据结构、GAM 的输入/输出线和内部数据的基础上构建了 GAM 的算法。然后进行了有效测试用例和无效测试用例的设计。GAM 的功能是通过微程序设计实现的。微程序设计克服了组合逻辑控制单元线路庞杂的缺点,能够实现更加复杂的功能,同硬布线比较具有规整性、灵活性、可维护性等一系列优点[15-17]。今后,将考虑使用比微程序内容更容易理解的高级语言编写程序,在此基础上使用汇编语言编写程序[18-20]。汇编语言是一种面向机器的语言,具有高级语言无法比拟的优点,其特点是运行速度快、占用存储空间小、可直接对硬件进行控制。同时,进行网络磁盘结构的 LAM 的设计。

参考文献:

[1] Patterson P A, Gibson G, Katz R H. A case for redundant arrays of inexpensive disks (RAID)[C]//Proc of international conference on management of data. Chicago:[s. n.], 1988: 109-116.

[2] アンドリュ-S・タネンバウム. 構造化コンピュータ構成[M]. 第 4 版. 株式会社ピアソンエデュケーション, 2000.

[3] 李元章, 孙志卓, 马忠梅, 等. S-RAID5:一种适用于顺序数据访问的节能磁盘阵列[J]. 计算机学报, 2013, 36(6): 1290-1302.

[4] Kakeshita T, Kubo S. A transaction processing architecture for effective load balancing utilizing high speed bus[C]//Proc of international symposium on cooperative database systems for advanced applications. Kyoto:[s. n.], 1996:376-379.

[5] Kakeshita T, Zhang S. The net disk architecture for dynamic load balancing among disk arrays[C]//Processings of the seven international conference on parallel and distributed system. Iwate, Japan:[s. n.], 2000.

[6] 赵延红, 掛下哲郎. NetDiskアーキテクチャにおけるスイッチの論理回路設計[C]//電気関係学会九州支部第 56 回連合大会講演論文集:09-1P-01. 2003.

[7] 赵延红, 掛下哲郎. 基于网络磁盘结构的 Local Migration Manager 设计[J]. 计算机技术与发展, 2011, 21(1): 169-173.

[8] 赵延红, 掛下哲郎. Net Disk アーキテクチャにおける Global Migration Manager の設計[C]//電気関係学会九州支部第 57 回連合大会講演論文集:09. 1A. 03. 2004.

[9] 石田晴久. マイクロコンピュータのプログラミング[M]. 共立出版, 1978.

[10] 楠田喜宏. マイコン再入門[M]. 日刊工業新聞社, 1981.

[11] 相原隆文. 手作りマイコン[M]. 技術評論社, 1985.

[12] 伊藤誠. 基本ハードウェア技術[M]. CQ 出版社, 1978.

[13] Myers G J. ソフトウェア. テストの技法[M]. 近代科学社, 2001.

[14] 河村一樹. ソフトウェア工学入門[M]. 近代科学社, 2003.

[15] 车海康, 杨银堂, 周拥华, 等. 数值协处理器中微程序设计[J]. 微电子学与计算机, 2003, 20(6): 57-61.

[16] 郑文荣, 刘少伟, 王树宗. 基于微程序设计的内建自测试技术研究[J]. 国外电子测量技术, 2011, 30(5): 20-22.

[17] 钟　凡, 蒋存波, 刘　丽, 等. 基于微程序思想的粉状物料车卸料程序[J]. 微计算机信息, 2009, 25(6-1): 298-300.

[18] Hayes J P. Computer architecture and organization[M]. [s. l.]: McGARAW-HILL International Editions, 1988.

[19] 范　喆, 沙全友, 卓德保. 汇编语言在 32 位程序设计中的应用[J]. 计算机与数字工程, 2008, 36(4): 38-40.

[20] 王文东, 李竹林, 尚建人. 汇编语言与 C 语言的混合程序设计技术[J]. 计算机技术与发展, 2006, 16(8): 18-20.

基于网络磁盘结构的Global Address Manager设计

作者： 赵延红， 掛下哲郎， ZHAO Yan-hong， KAKESHITA Tetsuro
作者单位： 赵延红,ZHAO Yan-hong(西安交通大学,陕西 西安,710061)， 掛下哲郎,KAKESHITA Tetsuro(日本佐贺大学 理工学部，日本 佐贺 840-8502)

刊名： 计算机技术与发展 ISTIC
英文刊名： Computer Technology and Development
年，卷(期)： 2015(8)