

文章编号:1001-9081(2005)07-1695-03

## 基于 Linux 的高可用系统中 IP 接管的设计与实现

王国豪<sup>1</sup>, 陈文智<sup>2</sup>, 石教英<sup>2</sup>

(1. 丽水学院 计算机系, 浙江 丽水 323000; 2. 浙江大学 计算机科学与技术学院, 浙江 杭州 310027)  
(ghwang@ls0578.net)

**摘 要:**介绍了一个基于 Linux 的双机热备份高可用性系统, 这个系统采用的 IP 地址接管保证了网络失效时主备系统间的平滑切换, 并给出了基于 ARP 欺骗的 IP 地址接管的实现。

**关键词:**双机热备份; IP 地址接管; HeartBeat; CheckPoint; 高可用性

**中图分类号:** TP302.8 **文献标识码:** A

## Design and realization of IP address takeover of high available system based on Linux

WANG Guo-hao<sup>1</sup>, CHEN Wen-zhi<sup>2</sup>, SHI Jiao-ying<sup>2</sup>

(1. Computer Science Department, Lishui College, Lishui Zhejiang 323000, China;  
2. College of Computer Science and Technology, Zhejiang University, Hangzhou Zhejiang 310027, China)

**Abstract:** A hot standby high available system based on Linux was introduced. It used IP address takeover to guarantee the smooth swift between the host system and the backup system when the network was failed.

**Key words:** hot standby; IP address takeover; HeartBeat; CheckPoint; high availability

当今计算机技术已进入以网络为中心的时期,大量的应用都围绕着网络进行,对服务器的性能和可靠性提出了越来越高的要求。为了满足如电信和银行等这些可用性要求极高的关键性应用,人们使用专用结构和广泛冗余的容错系统,通过比常规系统昂贵若干倍的代价将系统的可用性提高到 99.999% 以上,也就是平均每年的停机时间降到 5 分钟!虽然它的可用性的确很高,但代价也很高。为了以较小的代价获得较高的可用性,人们提出了各种解决方案。当网络上的某台机器失效时,通过 IP 地址接管可以将网络服务迁移到其他机器上继续执行,这样就保证了系统在遇到灾难时的可用性。

简单的来说,高可用性指的是通过尽量缩短因日常维护操作和突发性的系统崩溃所导致的停机时间,来提高系统和应用的可用性。高可用性在目前的要求是一年中的故障时间在 5 分钟以内,也称为 5 个 9 的可用性。

高可用性系统<sup>[1]</sup>是在冗余的一般可用性系统基础之上,运行高可靠性软件而构成。高可靠性软件用于自动检测系统的运行状态,在一台服务器出现故障的情况下,自动地把设定的服务转到另一台服务器上,由另一台服务器继续提供服务。高可用系统的主要功能包括:软件故障监测与排除;备份和数据保护;监视各个服务器的运行情况,能随时或定时报告系统运行状况,发生故障能及时报告和告警,并有必要的控制手段;实现错误隔离以及主、备份服务器间的服务切换,保证提供不间断服务。

### 1 高可用性系统的解决方案

#### 1.1 基于集群技术的高可用性系统<sup>[2]</sup>

集群技术是实现系统高可用性的重要手段,集群是两台

或更多台计算机(节点)在一个群组内共同工作。服务器集群是作为单一系统进行管理的一组独立的服务器,用于实现更高的可用性、可管理性和更优异的可伸缩性。本文侧重于介绍基于 Linux Virtual Server(虚拟服务器)的 HA 解决方案。基于集群技术的高可用性系统目前主要有两种解决方案:

##### 1) mon + heartbeat + fake + coda

利用现有的“mon”,“heartbeat”,“fake”和“coda”四个软件来构筑具有高可用性的 Virtual Server(虚拟服务器)。“mon”是一个多用途的资源管理系统,用来监控网络上的服务器节点和网络服务。“heartbeat”实现在两台计算机间通过串行线或者 UDP 协议传送“心跳信息”。“fake”是一个使用 ARP 欺骗的方法来实现 IP 接管。“coda”是一个容错的分布式文件系统,服务器上的目录能够存储在“coda”上,所以文件能够实现高可用性,并且易于管理。

客户端通过虚拟 IP 地址访问他们需要的服务,客户端的访问请求被运行了“mon”的 Linux Director 定向到对应的服务器上。为了防止因负载均衡器 Linux Director 的失败而造成整个系统的失败,必须要有备用的负载均衡器,当主负载均衡器失效时能够及时承担起主负载均衡器的工作。

Linux Virtual Server 不是完美无缺的,它的最大问题是,当主负载均衡器失效或被接管时,建立在主负载均衡器上的连接 hash 表就会丢失,客户端不得不重发请求。所以说这种解决方案还只是解决了一部分的高可用性问题。

##### 2) ldirectord + heartbeat

“ldirectord”(Linux Director Daemon)是 Jacob Rief 编程实现的一个独立进程,用来实现对服务和物理服务器的监测,广泛地用于 http 和 https 服务。与上面的“mon”相比,“ldirectord”有以下的优势:

收稿日期:2004-12-13;修订日期:2005-03-07

**作者简介:**王国豪(1971-),男,浙江丽水人,讲师,硕士研究生,主要研究方向:基于 Linux 的嵌入式系统; 陈文智(1969-),男,副教授,博士,主要研究方向:嵌入式系统、分布式系统; 石教英(1937-),男,浙江宁波人,教授,博士生导师,主要研究方向:计算机系统结构、CAD/CG、多媒体技术。

1) “ldirectord”是专门编写的 LVS 监控程序。

2) 它从 /etc/ha.d/xxx.cf 文件中读取所有关于 IPVS 路由表的配置信息。当“ldirectord”运行起来后,IPVS 路由表将会被适当地配置。可以将 Virtual service 配置放在多个配置文件中,单独修改某一种服务的参数,而不影响其他的服

3) “ldirectord”能被“heartbeat”轻松地管理——启动或关闭。“ldirectord”也能够手动开启和关闭。可以在无备份负载均衡器的 LVS 集群中使用它。

mon + heartbeat + fake + coda 方案中存在的问题,在 ldirectord + heartbeat 方案中也存在。

## 1.2 双机热备份方案<sup>[3]</sup>

双机热备份技术是一种软硬件结合的有较高容错性的应用方案。该方案由两台服务器系统和一个外接共享磁盘阵列柜(也可没有,而是在各自的服务器中采用 RAID 卡)及相应的双机热备份软件组成。双机热备份系统采用“心跳”方法保证主系统与备用系统的联系。所谓“心跳”,指的是主从系统之间相互按照一定的时间间隔发送通讯信号,表明各自系统当前的运行状态。一旦“心跳”信号表明主机系统发生故障,或者备用系统无法收到主机系统的“心跳”信号,则系统的高可用性管理软件认为主机系统发生故障,主机停止工作,并将系统资源转移到备用系统上,备用系统将替代主机发挥作用,以保证网络服务运行不间断。双机热备份方案中,根据两台服务器的工作方式可以有三种不同的工作模式,即:双机热备模式、双机互备模式和双机双工模式。

双机热备模式即目前通常所说的 active/standby 方式,active 服务器处于工作状态;而 standby 服务器处于监视准备状态。服务器数据包括数据库数据同时往两台或多台服务器写入(通常各服务器采用 RAID 磁盘阵列卡),保证数据的即时同步。当 active 服务器出现故障的时候,通过软件检测或手工方式将 standby 机器激活,保证应用在短时间内完全恢复正常使用。其典型应用是证券资金服务器或行情服务器。这是目前采用较多的一种模式,但由于另外一台服务器长期处于后备的状态,从计算资源方面考量,就存在一定的浪费。双机互备模式,是两个相对独立的应用在两台机器同时运行,但彼此均设为备机,当某一台服务器出现故障时,另一台服务器可以在短时间内将故障服务器的应用接管过来,从而保证了应用的持续性,但对服务器的性能要求比较高。配置相对要好。双机双工模式是目前 cluster(群集)的一种形式,两台服务器均为活动,同时运行相同的应用,既保证了整体的性能,也实现了负载均衡和互为备份。

在很多时候,人们容易将双机热备份技术与集群技术混为一谈,其实它们具有本质上的区别:即能否实现并行处理和节点机失效后的任务平滑接管。双机或多机热备份当主机发生故障时,备份机不能实现平滑接管,不具备负载均衡、并行处理的能力。而集群技术能实现并行处理和节点机失效后的任务平滑接管。

## 2 基于双机热备份的 Linux HA 开发方案

### 2.1 系统的硬件环境

基于双机热备份的 Linux HA 系统的硬件平台基于 CompactPCI<sup>[4]</sup>工业标准,如图 1 所示。

CompactPCI 是基于标准 PCI 电气规范的高性能工业总线,主要连接 3U 或 6U 两种欧式插卡模式,它集合了 PCI 总线和基于 PC 的即插即用软件的全部特点。不同于台式 PC

机的是,CompactPCI 板卡采用了符合 IEC 和 Bellcore 标准的高质量 2-mm 引脚和插座连接器以及无源背板技术。

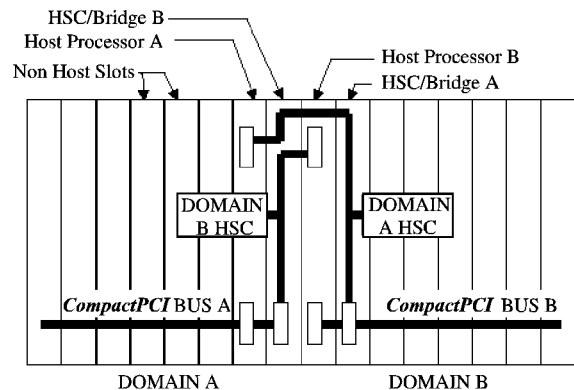


图1 系统硬件平台

CompactPCI 系统有 2 段 CPCI 总线,每段总线上有 6 块 I/O 板,这 2 段总线可以分别由系统板 A 和 B 控制,也可以统一由系统板 A 或 B 之一进行控制。这 2 段总线通过 P2P 桥在热插拔控制器的控制下连接到系统板上。一个 P2P 桥在系统板上,另外一个在 HSC 桥板上。

这 2 段总线可以有 2 种工作方式:1) 两块系统板平时独立工作,在其中一块系统板出现故障的情况下,另一块板可以通过与之配对的桥板接替故障板管理的 CPCI 总线,但前提条件是两块系统板的负荷不能太大,避免在一块系统板出现故障,另一块板管理两条总线时出现负荷过重而系统崩溃。2) 两块系统板平时是主备工作,一块系统板除了管理自己一侧的总线,还通过桥板管理另一侧的总线。当该对系统板和桥板出现故障的时候,由另一对系统板和桥板接管整个系统。因此基于双机热备份的 Linux HA 系统也有两种工作模式:双机热备模式和双机互备模式。以太网和串口连接是两个系统的唯一的交流途径,也就是 HeartBeat 协议和 CheckPoint 协议执行的通道。

所有的接口板在硬件上支持热插拔,可以通过系统板/桥板对来控制该接口板的插入和拔出。桥板上没有处理器,完全由与之相对应的系统板来控制。

### 2.2 系统框架

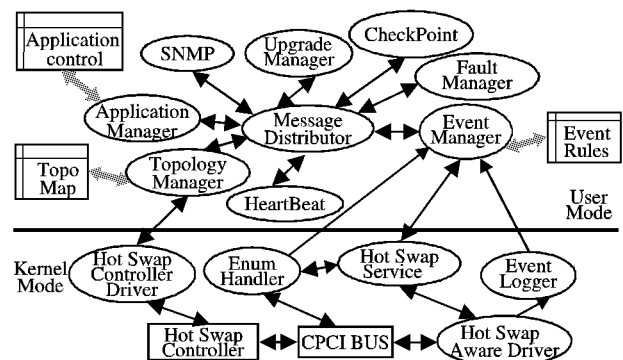


图2 系统整体结构

基于双机热备份的 Linux HA 系统的整体结构如图 2 所示。系统由 9 个模块构成,这 9 个模块分别为事件管理模块、Heartbeat 模块、CheckPoint 模块、拓扑管理模块、故障管理模块、应用程序管理模块、SNMP 远程管理模块、在线升级模块和 Windows 端管理界面。整个系统以消息驱动,一方面事件管理模块作为中心模块,负责各个模块间的信息交换和系统整体的调控。另外一方面,事件管理模块还必须和底层的热插拔设备交换信息。这 9 个模块各自形成一个完整的体系,

模块间的通讯都是通过事件管理模块来进行的。这样一方面系统模块间耦合性小,一些非关键模块损坏不会影响到整个系统的稳定性;另外一个方面是模块间关联小,容易被升级、替换,简化了在线升级模块的设计和实现。

### 2.3 关键技术——HeartBeat 协议和 CheckPoint 协议的实现

HeartBeat 是 HA 软件的基础和重要部分,是实现整个 HA 系统的关键。HA 系统中 HeartBeat 这一块主要负责的任务是让两个正在运行的系统知道对方是否 keep alive。HeartBeat 不仅要不断检测对方发过来的心跳,来确认对方是 keep alive,而且还要定时向对方发送心跳,让对方知道自己也是 keep alive。一旦在规定的 warning time 里接收不到对方的心跳,就向系统报告对方机器已经进入 warning 状态;在规定的 dead time 内接收不到对方的心跳,就向系统报告对方机器已经进入 dead 状态。系统根据 HeartBeat 模块的报告,用不同的处理策略来处理。

Checkpoint 是用来保持两个系统的数据一致性的协议。checkpoint 是 HA 系统的重要组成部分,是实现 HA 的关键。在主系统中,Checkpoint 模块每隔一段时间对系统中的应用程序进行扫描,如果进程状态正常则进行数据备份,备份的内容包括进程相关数据,如 PCB 内容、CPU 寄存器的数据、内存数据等等,这些数据都是为了在进程出现问题时,恢复进程所必需的。如果进程出现异常或者已经死亡,那么根据 Checkpoint 模块在上一次的备份恢复进程;在备系统中,Checkpoint 的数据会得到备份,这样两个系统之间的数据保持一致,在一个主系统 out of work 的时候,备系统才能接着主系统的状态继续运行下去。Checkpoint 是两个独立系统的备份和检测动作通过一个模块进行,所有的备份和检测动作上报到这个模块后转发给另一个系统,再由另一个系统的对应模块接收并转发给其他功能模块。

## 3 网络失效接管策略

一个服务器上的所有部分:中央处理器、电源、主板、内存、适配卡以及其他的部分都有可能会出现故障,也就是说所有这些部分中任何一个出现故障都会引起整个系统的不可用。但最普遍的情况是网络失效,如网络不可访问、网速极慢或受到 Dos 攻击等。网络失效会引起整个系统的不可用,这在 HA 系统中是不允许的,此时必须要由备份服务器接管主服务器的工作。

网络失效的接管策略<sup>[5]</sup>依赖与本地网络的拓扑结构,主要有以下三种:IP 地址接管(IP address takeover)、MAC 地址接管(MAC address takeover)和动态 DNS 重配置(Dynamic DNS reconfiguration),它们都有一个固有的接管时间。IP 地址接管<sup>[5]</sup>(IPAT):当主服务器网卡故障时,它的 IP 地址应该由备份服务器上正常的网卡来接管。IP 地址接管比 MAC 地址接管稍慢,可靠性也较差。MAC 地址接管:当 IPAT 发生时,客户端机器面对相同的 IP 地址却有不同 MAC 地址。根据 RFC 826,如果发生一个 ARP 请求时已有一个 IP 地址在 ARP Cache 中,应该更新 ARP Cache。这样就要求我们在 IPAT 之后清空接管接点的 ARP Cache 并 ping 客户端。Ping 导致一个 ARP 请求进而更新客户端的 ARP Cache。MAC 地址接管可以在瞬间完成,但易引起混乱。动态 DNS 重配置最慢,但负载均衡能力较好。

## 4 基于 ARP 欺骗的 IPAT 的设计与实现

### 4.1 ARP 欺骗原理<sup>[6]</sup>

ARP(Address Resolution Protocol)是地址解析协议,是一种将 IP 地址转化成物理地址的协议。从 IP 地址到物理地址的映射有两种方式:表格方式和非表格方式。ARP 具体说来就是将网络层(IP 层,也就是相当于 OSI 的第 3 层)地址解析为数据连接层(MAC 层,也就是相当于 OSI 的第 2 层)的 MAC 地址。

ARP 原理:某机器 A 要向主机 B 发送报文,会查询本地的 ARP 缓存表,找到 B 的 IP 地址对应的 MAC 地址后,就会进行数据传输。如果未找到,则 A 广播一个 ARP 请求报文(携带主机 A 的 IP 地址 Ia——物理地址 Pa),请求 IP 地址为 Ib 的主机 B 回答物理地址 Pb。网上所有主机包括 B 都收到 ARP 请求,但只有主机 B 识别自己的 IP 地址,于是向 A 主机发回一个 ARP 响应报文,其中就包含有 B 的 MAC 地址。A 接收到 B 的应答后,就会更新本地的 ARP 缓存,接着使用这个 MAC 地址发送数据(由网卡附加 MAC 地址)。因此,本地高速缓存的这个 ARP 表是本地网络流通的基础,而且这个缓存是动态的。

ARP 协议并不只在发送了 ARP 请求才接收 ARP 应答。当计算机接收到 ARP 应答数据包的时候,就会对本地的 ARP 缓存进行更新,将应答中的 IP 和 MAC 地址存储在 ARP 缓存中。因此,当局域网中的某台机器 B 向 A 发送一个自己伪造的 ARP 应答,而如果这个应答是 B 冒充 C 伪造来的,即 IP 地址为 C 的 IP,而 MAC 地址是伪造的,则当 A 接收到 B 伪造的 ARP 应答后,就会更新本地的 ARP 缓存,这样在 A 看来 C 的 IP 地址没有变,而它的 MAC 地址已经不是原来那个了。由于局域网的网络流通不是根据 IP 地址进行,而是按照 MAC 地址进行传输。所以,那个伪造出来的 MAC 地址在 A 上被改变成一个不存在的 MAC 地址,这样就会造成网络不通,导致 A 不能 Ping 通 C! 这就是一个简单的 ARP 欺骗。

### 4.2 基于 ARP 欺骗的 IPAT 的实现

如果备份服务器在规定的 dead time 内接收不到来自备份服务器的心跳("I'm alive")信息时,将自动激活 ARP 欺骗进程接管主服务器的 IP 地址,并开始提供服务;而当再次收到来自主服务器的心跳("I'm alive")消息时,备份服务器将自动将 ARP 欺骗进程关闭,释放出它接管的服务器,主服务器重新开始工作,提供服务。

为了实现基于 ARP 欺骗的 IPAT,要构造一个和 ARP 分组格式一样的结构体。该结构体如下:

```
struct arp_packet
{
    u_char targ_hw_addr[ETH_HW_ADDR_LEN];           //目标以太网地址
    u_char src_hw_addr[ETH_HW_ADDR_LEN];             //源以太网地址
    u_short frame_type;                               //帧类型
    u_short hw_type;                                  //硬件类型
    u_short prot_type;                                //协议类型
    u_char hw_addr_size;                             //硬件地址长度
    u_char prot_addr_size;                           //协议地址长度
    u_short op;                                       //操作类型,1 表示请求包,2 表示应答包
    u_char sndr_hw_addr[ETH_HW_ADDR_LEN];           //源以太网地址
    u_char sndr_ip_addr[IP_ADDR_LEN];                //源 IP 地址
    u_char rcpt_hw_addr[ETH_HW_ADDR_LEN];           //目标以太网地址
}
```

(下转第 1710 页)

```

    } else if ( msg.sameKind( "AgletMessage" ) )
        doAgletMessage();    //处理 Agent 传递的 Message 对象
    }
}
//定义 SlaveAgent 的超类
public abstract class SlaveAgent extends Slave{
    protected void initializeJob() {
        RESULT = null;
    }
    protected void doJob() {
        Arguments args = ( Arguments ) ARGUMENT;
        String SlaveAgentName = new String( ( String )
            ( args.getArg( "SlaveAgentName" ) ) );
        if ( SlaveAgentName == "InitAgent" ) {
            doInit;    //发送初始化数据消息至 SManagerAgent 并返回
        } else
            //执行控制、巡视或协作监控任务,并调用 setResult() 方法
            //返回执行结果或调用 sendAgletMessage() 方法
            //发送 Message 至 HManagerAgent,而代理继续在本机执行
            if ( SlaveAgentName == "MonitoringAgent" ) {
                doMonitoring( args );
            } else if ( SlaveAgentName == "ItinerantAgent" ) {
                doItin( args );
            } else if ( SlaveAgentName == "CoopAgent" ) {
                doCooperating( args );
            } else setResult( "" );
        }
    }
}
//定义 UpdatingAgent 的超类
public abstract class UpdatingAgent extends Agent{
    Arguments HManagerAgentURL_ID_InitARG = new Arguments();
    public void onCreate( Object init ) {
        HManagerAgentURL_ID_InitARG = ( Arguments ) init;
    }
    public void run() {
        getUpdatingDATA( ( String )
            HManagerAgentURL_ID_InitARG.getArg( "InitARG" ) );
        //定期检索本地信息数据库获取刷新数据
    }
}

```

(上接第 1697 页)

```

    u_char rept_ip_addr[ IP_ADDR_LEN ];    //目标 IP 地址
    u_char padding[ 16 ];
    //因为分组格式只有 42 个字节,而一个 IP 数据包至少要
    //60 个字节,所以 padding 在这里起填充作用
};

```

实现基于 ARP 欺骗的 IP 地址接管,只需广播一个 ARP 包,并在包中指定备份服务器的 MAC 地址对应于主服务器的 IP 地址即可。具体实现如下:

```

sock = socket( AF_INET, SOCK_PACKET, htons( ETH_P_RARP ) );
...
//指定 ARP 包中的源 IP、目标 IP、源 MAC 地址、目标 MAC 地址.
get_hw_addr( pkt.targ_hw_addr, targ_hw_addr );
get_hw_addr( pkt.rept_hw_addr, targ_hw_addr );
get_hw_addr( pkt.src_hw_addr, sndr_hw_addr );
get_hw_addr( pkt.sndr_hw_addr, sndr_hw_addr );
get_ip_addr( &src_in_addr, sndr_ip_addr );
get_ip_addr( &targ_in_addr, targ_ip_addr );
...

```

sendto( sock, &pkt, sizeof( pkt ), 0, &sa, sizeof( sa ) ); //发送 ARP 包  
备份服务器必须频繁地发送 ARP 包,以使本地网络上其他节点的 ARP CACHE 不断更新。如果 ARP CACHE 不及时更新,而已失效主服务器处于能够响应 ARP 请求的状态,将导致主服务器与备份服务器 ARP 响应的竞争,从而可能导致将 IP 包发送到主服务器。

```

//并调用 sendData() 方法传送给 HManagerAgent
}
}

```

## 4 结语

作为现代制造技术的一个重要组成要素和支持技术,本文在分析了虚拟企业协同生产过程远程监控特点的基础上,提出了一种基于移动 Agent 计算模式的分布式多层远程监控模型。该模型充分利用了移动 Agent 的特点,不仅较好地克服了目前基于 P2P、C/S 和 B/S 计算模式的远程监控系统对 Internet 带宽和可靠性依赖较高,系统的远程实时交互性及系统运行的可靠性和稳定性难以保证以及系统的流程和功能固定实现,缺少可扩展性、灵活性和适应性等缺点,而且能较好地适应虚拟企业制造环境下远程监控的分布异构性、可重构性、协同性和多目标性等特点,具有 Agent 的配置、控制及运行灵活性好,代码和版本易于控制和维护等优点。该模型在某模具制造动态联盟企业远程监控中的应用表明,在实时性和刷新率均要求很高的情况下,系统具有良好的控制效果和运行的稳定性,对虚拟企业协同生产过程的远程监控具有广泛的应用前景。

### 参考文献:

- [1] 严新民. 计算机集成制造系统[M]. 西安: 西北工业大学出版社, 1999.
- [2] 任伟, 王坚, 张浩, 等. 制造企业工业现场远程监控系统的设计与开发组合[J]. 机床与自动化加工技术, 2000, (5): 29-31, 40.
- [3] FUGGETTA A, PICCO G, VIGNA G. Understanding code mobility [J]. IEEE Transactions on Software Engineering, 1998, 24(5): 342-361.
- [4] LANGE DB, OSHIMA M. Mobile Agents with Java: The Aglet API [J]. World Wide Web Journal, 1998, 1(3): 111-121.
- [5] <http://www.trl.ibm.com/aglets/index.html> [EB/OL], 2003-12.

## 5 结语

基于 ARP 欺骗的 IP 地址接管要求主服务器和备份服务器在同一网段,如果主服务器和备份服务器处于不同网段,上面的方法是不起作用的。把 ARP 欺骗和 ICMP 重定向结合在一起,就可以基本实现跨网段欺骗的目的。

### 参考文献:

- [1] ROSE 高可用软件[EB/OL]. <http://article.itebook.net/article.php?articleid=532>, 2004-05-18.
- [2] High Availability[EB/OL]. <http://www.linuxvirtualserver.org/HighAvailability.html>, 1998-12-05.
- [3] 两种容错方案的比较一[EB/OL]. <http://article.itebook.net/article.php/548.html>, 2004-05-18.
- [4] FORCE COMPUTERS Inc. Answering the Call: High Availability CompactPCI for Telecommunications [EB/OL]. [http://www.ganymed.com/data/force/white\\_papers/ha\\_cpici.pdf](http://www.ganymed.com/data/force/white_papers/ha_cpici.pdf), 1999-10.
- [5] Network failover strategies [EB/OL]. <http://linux-ha.org/failover/>, 2004-06-02.
- [6] An Introduction to ARP Spoofing. Sean Whalen[EB/OL]. <http://node99.org/projects/arp spoof>, 2001-04.
- [7] Setting Up Redundant Networking[EB/OL]. <http://linux.math.tifr.res.in/sysadmin/High-Availability-HOWTO-8.html>, 2004-05-23.