第 35 卷 第 8 期 计 算 机 工 程 2009 年 4 月
Vol.35 No.8 Computer Engineering April 2009
·网络与通信· 文章编号：1000—3428(2009)08—0137—03 文献标识码：A 中图分类号：TP301.6

分布式故障诊断在 SNMP 中的模拟与应用

申 彦[1]，杜建国[2]
(1. 江苏大学计算机科学与通信工程学院，镇江 212013；2. 江苏大学信息管理与信息系统系，镇江 212013)

摘 要：分析传统网络管理集中式及分层次体系结构的缺陷，提出分布式网络管理的体系结构，把用于大规模集成电路故障诊断的系统级故障诊断算法 ADSD 应用到计算机网络故障诊断中。为了验证计算机网络分布式故障诊断的可行性，在 NS 中嵌入 ADSD 算法，模拟整个算法的执行，对算法的执行效率如诊断延时、报文流量进行分析。在此基础上对原算法进行改进，以减小算法对正常网络应用的影响。通过对 SNMP 的模拟实验，验证在网络管理中引入分布式故障诊断的可行性。

关键词：计算机网络管理；分布式网络管理；网络模拟；分布式算法

Simulation and Application of Distributed Fault Diagnosis in SNMP

SHEN Yan[1], DU Jian-guo[2]
(1. School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang 212013;
2. Department of Information Management and Information System, Jiangsu University, Zhenjiang 212013)

【Abstract】This paper analyzes the innate disadvantages of centralized architecture and hierarchical architecture in traditional network management, and presents a new distributed architecture of network management. ADSD algorithm, which is originally used in digital large scale integrated circuit, is applied into management of computer network. To validate the feasibility of distributed fault diagnosis in network, ADSD is compiled into NS. The performance of this algorithm is analyzed, such as the delay of fault diagnoses and the traffic of message. The visual experimental results are presented. The algorithm is improved to reduce the effect in normal network applications. According to the simulation system, it can be concluded that the distributed architecture for network management is feasible.

【Key words】computer network management; distributed network management; network simulation; distributed algorithm

1 概述

近年来，计算机网络迅速发展，网络管理变得越来越重要。通过分析传统计算机网络管理软件产品故障诊断的实现，可以根据其体系结构分为 2 种类型：集中式的体系结构和分层次的体系结构，但这 2 种网络管理的体系结构都由于一些内在因素而有着不可克服的弊端。例如，随着网络规模的扩大、节点数量的不断增多，如果网络管理还是按照集中式体系结构工作，管理站将会花费很长的时间用于探测设备，这对于网络管理者将是不可忍受的。另外，整个网络中的诊断流量都集中在管理站处，对承担管理站任务的计算机来说是一种很大的负担，如果管理站不能正常工作，那么整个网络的诊断就不能正常地进行，因此，这种体系结构还存在着可靠性不强的问题。再比如，分层次的体系结构可能会给数据采集造成困难。每一个域管理者的设备列表需要在逻辑上预先定义并配置好，否则会使多个管理者轮询管理相同的设备，造成资源浪费。另外，当一个域管理站不能正常工作时，就会缺失掉这个域管理站所管理域内设备的诊断信息。

本文吸收了大规模集成电路系统级故障诊断方法的思想，在计算机网络管理中引入分布式的管理体系结构[1-3]以克服上述缺陷，利用网络模拟环境 NS 实现了 ADSD 系统级诊断算法，并且给出性能分析和原型系统。

2 分布式系统级故障诊断算法 ADSD

本文在计算机网络中引入大规模集成电路故障诊断算法 ADSD[4]，该算法假设存在一个逻辑上完全连接的网络，且算法是分布式的，测试节点的集合是自适应的，对发生故障节点的数量也没有限制。ADSD 算法规定：在诊断进行测试时，一个节点在 2 次测试之间不能发生故障或者从故障中恢复。该算法是投入实际使用的第 1 个分布式系统级故障诊断算法。

在 ADSD 算法中，一个节点 i 使用一个二维矩阵 $TestedUp_i$ 更新测试的结果以及使用这个二维数组的信息响应测试这个节点的测试者。现在假设有一个 8 个节点的系统，这个 8 节点的系统也是之后要在 NS 中进行算法模拟的原型，以节点 2 中的 *TestedUp* 加以说明。

二维数组 $TestedUp_i$ 包括了 N 项实体内容，N 为系统中的节点数量。数组中的序号以及值代表了系统中节点的标识。比如，实体 $TestedUp_i[u]=v$ 表明节点 i 已经从邻居节点接收到诊断信息，这个诊断信息隐含说明了节点 u 已经测试了节点 v，而且发现节点 v 是正常工作的。再比如，实体 $TestedUp_i[i]=u$ 表明节点 i 通过自身的测试发现节点 u 是正常工作的。如果实体的值是“x”，那么表明测试的结果是不确定的、任意的。下文给出一个实际的由 8 个节点组成的系统

基金项目：中国博士后科学基金资助项目(20060400918)；江苏大学校高级人才基金资助项目(06JDG025)
作者简介：申 彦(1982－)，男，硕士，主研方向：网络技术，网络管理，信息技术；杜建国，副教授、博士
收稿日期：2008-09-28 **E-mail：**104186179@qq.com

中节点 2 中 $TestedUp_2$ 的内容，这时节点 1、节点 3、节点 4 是发生了故障的。

$Tested\_Up_2$ [0] =2
$Tested\_Up_2$ [1] =x
$Tested\_Up_2$ [2] =5
$Tested\_Up_2$ [3] =x
$Tested\_Up_2$ [4] =x
$Tested\_Up_2$ [5] =6
$Tested\_Up_2$ [6] =7
$Tested\_Up_2$ [7] =0

在这个二维数组里有一个特别的属性，就是在一个故障事件被正确诊断出后，只要从一个正常工作的节点 i 出发，按照一个正常工作节点的路径，连接所有正常工作的节点，这个二维数组就会形成一个“测试环”。以节点 2 中的 *TestedUp* 为例进行说明。从正常工作的节点 2 开始，按照正常工作的节点路径 2->5, 5->6, 6->7, 7->0, 0->2，把这个路径上所有的节点连接起来，就形成了一个系统中正常工作的节点的环 2->5->6->7->0->2。这个特性在 ADSD 的诊断算法中起着非常重要的作用。

本文在网络管理中引入 ADSD 分布式管理，在此基础上逐步完善，引入诊断更加高效、诊断延时更小的算法，如基于事件驱动的 ADSD 算法、层次性的 ADSD 算法。

3 NS 模拟系统的实现

为了验证在计算机网络管理中引入 ADSD 算法的可行性以及算法实现的正确性，受限于开发分布式程序的环境问题，本文采用 NS 软件模拟平台[5]，并对 NS 源代码做相应的修改，使之达到模拟要求。

(1)修改 NS 的源代码，使得可以在 NS 中传送实际数据。

在 NS 的模拟环境中一般不涉及真实的数据传输，所有的数据传输都只是依据其数据大小以及网络的连接情况计算出数据的传输情况，而并非将相应的数据传送过去。本文分析了 NS 源代码的结构，修改了相应的代码，通过应用类中的一个指向传输层对象的指针调用传输层的 sendmsg 函数，通过地址传递的方式让其传输数据。

在从传输层向应用层传递信息的过程中运用了一个小技巧：UDP 类中有一个指向所连接的应用的指针 app_，通过这个指针调用应用对象的 process_data 函数处理传递过来的数据。

```
void UdpAgent::recv(Packet* pkt, Handler*){
if (app_ ) {
// If an application is attached, pass the data to the app
hdr_cmn* h = hdr_cmn::access(pkt);
app_->process_data(h->size(), pkt->userdata());
}
else if(pkt->userdata() && pkt->userdata()->type() == PACKET_DATA){
…
}
```

(2)实现 ADSD 算法以及模拟简单网络管理协议 SNMP 所需的相关类。

如图 1 所示，AdsdData 类用于实现 ADSD 算法中的 *TestedUp* 数据结构，SnmpOPData 类用于模拟实现 SNMP 的操作，这个类用 OPType 区别 snmpget, snmptrap, snmpwalk 等 SNMP 操作类型，用 OID 属性模拟 SNMP 操作的对象标识符。

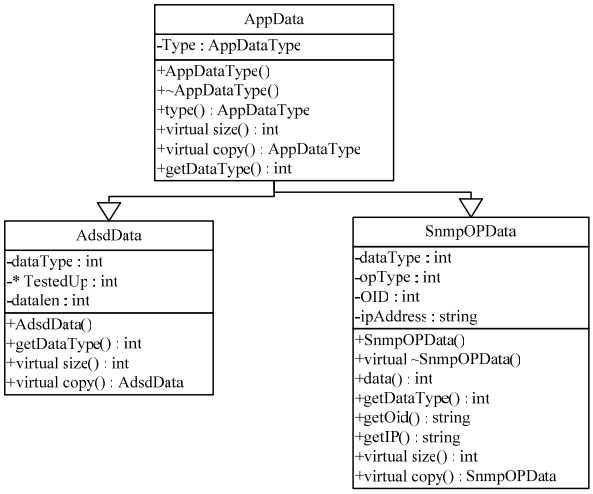

图 1　数据类型的实现

(3)在 NS 中创建实现 ADSD 算法的应用类 SnmpApp，算法相关操作都在这个类中实现，如图 2 所示，本文用 SnmpApp 在 NS 中模拟一个 SNMP 应用，在 SnmpApp 中使用 *TestedUp*[*MAX*][2]表示 ADSD 算法所需的 *TestedUp* 数据结构，并且依靠 send_CsnmpOP, send_ServerDA，process_data 模拟 ADSD 算法的整个交互的过程。

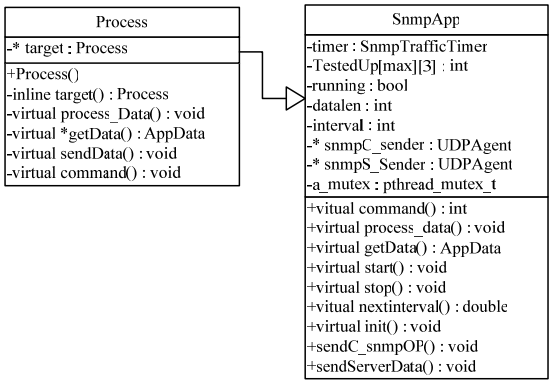

图 2　SnmpApp 应用类的实现

4 模拟试验的结果分析

4.1 NS 中 ADSD 算法模拟

在 NS 中，网络场景的模拟主要是通过编写 TCL 脚本语言来描述。试验模拟了一个由 8 个节点组成的局域网，每个节点都绑定了一个 SnmpApp 应用，用于模拟 ADSD 算法。由于 SnmpOPData 以及 SnmpStatusData 包含的信息量比较少，而 AdsdData 的信息量比较多，包含 *TestedUp* 整个系统级故障诊断信息，同时为了定性地分析算法在进行诊断时产生的报文流量对网络带宽的占用情况，因此在模拟环境中，定义 SNMP 的普通探测 SnmpOPData 报文以及 SNMP 响应 OID 为 status 的 SnmpStatusData 报文，其大小都小于 SNMP 响应 OID 为 ADSD 的 AdsdData 的报文。

场景 1 在模拟时间 35.0，节点 5、节点 7 同时发生故障，在模拟时间 40.0，节点 7 恢复正常，分析算法产生的报文流量。TCL 描述如下：

```
$ns at 35.0 “$snmp5 seterror 1”
$ns at 35.0 “$snmp7 seterror 1”
$ns at 40.0 “$snmp7 seterror 0”
```

模拟结果如下：在上述场景下运行 ADSD 算法进行诊断，

得到整个算法运行过程中的报文流量记录文件，截取该记录文件中部分数据，如图 3 所示，并利用 Xgraph 解析该文件得到如图 4 所示的波形图。

```
0 0.0
  0.10000000000000001 0.0
0. 0.  20000000000000001 0.0
...
    79.799999999999741 0.0023999999999999998
    79.899999999999736 0.0008000000000000004
  79.99999999999973 0.0
...
```

图 3　ADSD 的报文流量记录

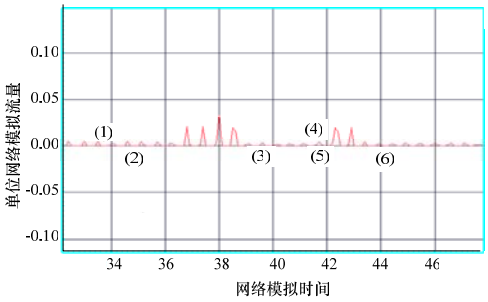

图 4　报文流量分析

在图 4 中：(1)在 35.0 处节点 5 和节点 7 发生故障。(2)系统测试出发生故障，进行故障诊断，流量增大。(3)诊断结束，系统的每个节点都有正确的诊断信息。(4)在 40.0 处节点 7 恢复正常工作。(5)系统检测到恢复事件，重新进行诊断。(6)诊断结束，系统的每个节点都有正确的诊断信息。

根据模拟场景的定义，系统在模拟时间 35.0 发生故障事件，随后进入诊断阶段，节点之间相互传递 *TestedUp* 数组中的信息。从波形图也可以看出，随后系统的报文流量有个突增，符合预期的设想。算法很快趋于稳定，报文流量随之减少，完成此次诊断。后面的一次报文突增同样是对节点 7 的恢复事件进行诊断的结果。输出节点 0 中的诊断信息，这是因为在利用 ADSD 算法诊断时，整个诊断信息是逆向传递的，所以最晚得到正确诊断信息的是节点 0。如图 5(a)所示，得到一个正常工作节点所形成的环：0->1->2->3->4->6->0，图 5(b)同理。

```
01
12
23
34
46
56
60
70
39.0487584 //time
```

```
01
12
23
34
46
56
67
70
42.8298144//time
```

(a)节点 7 故障时的诊断信息　(b)节点 7 恢复正常时的诊断信息

图 5　正常工作节点的诊断信息

从节点输出的诊断信息可见，在模拟时间 35.0，节点 5 及节点 7 发生故障，在模拟时间 39.048 758 4，节点 0 正确诊断出这个故障事件，而在这之前，系统中的其他节点都已正确诊断出这个故障事件，在 40.0 时，节点 7 恢复正常工作，在 42.829 814 4，节点诊断出此次恢复事件。从试验的结果来看，算法的诊断延时比较小，可以满足实际故障诊断的需求。

场景 2 只有一个节点正常，其余节点全部发生故障，分析算法产生的报文流量。之所以考察只有一个节点正常工作的情况，是因为某些系统级故障诊断算法要求系统中正常工作的节点数目 $t>a$ 且 $a>1$。TCL 描述如下：

```
$ns at 30.0 “$snmp1 seterror 1”
$ns at 30.0 “$snmp7 seterror 1”
```

由图 6、图 7 的试验结果可以看到，即使只有一个节点正常工作，ADSD 算法仍然能够正确地诊断出系统全部节点的状态。

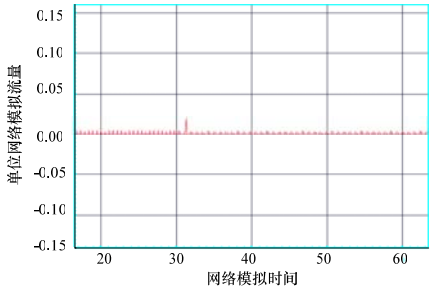

图 6　唯一节点正常工作时的报文流量分析

```
00 // 只有一个节点 0 正常工作
12
23
34
45
56
67
70
 30.9871608 //time
```

图 7　唯一正常工作节点的诊断信息

从上面的结果可以看出，ADSD 算法完全可以用于计算机网络的故障诊断，正确地诊断出网络中的故障事件，并且诊断时延比较小，适应实际的诊断需要。

4.2　ADSD 算法的改进

在原始的 ADSD 算法描述中，节点每次探测到一个正常工作的节点都会要求那个节点传递的诊断信息，即 *TestedUp* 中的所有信息。经研究认为：(1)每次探测时，可能整个系统中没有发生任何的故障事件，或者那个被探测的节点本身没有探测到该故障事件，在这种情况下，该被探测节点的 *TestedUp* 中的状态没有发生变化，没必要传递 *TestedUp* 中的数据，可以节省网络带宽。(2)即使被探测的节点中的 *TestedUp* 内容发生了变化，也不需要传递整个 *TestedUp*，只需传递其中发生变化的部分。本文改进了 ADSD 算法的实现，重新将算法融入到 NS 中进行模拟分析，再次选用场景 1，以便做比较分析。重新模拟算法的执行情况如图 8 所示。

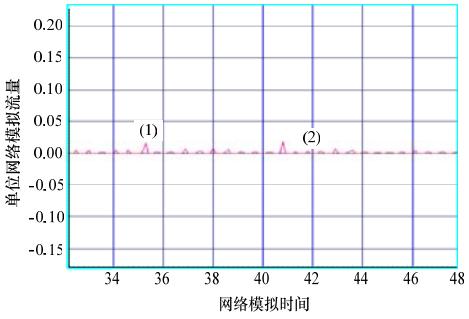

图 8　改进后算法的报文流量分析

(下转第 142 页)