

文章编号:1001-9081(2006)03-0703-02

## 服务器系统管理的自律计算模型

樊 星,李战怀,刘全中

(西北工业大学 计算机学院,陕西 西安 710072)

(galaxy\_fxstar@yahoo.com)

**摘 要:**利用时序模型算法和事件分类匹配算法对服务器系统多个节点的事件和系统参数进行分析与预测,结合策略库技术,构建了一个服务器系统管理的自律计算模型。该模型能够根据已知的操作环境自我配置,在各种灾难发生前自我保护,发生后自我修复,并且可以自我优化。

**关键词:**自律计算;事件;预测;策略

**中图分类号:** TP311.131 **文献标识码:** A

## Autonomic computing model of server system management

FAN Xing, LI Zhan-huai, LIU Quan-zhong

(College of Computer, Northwestern University of Technology, Xi'an Shaanxi 710072, China)

**Abstract:** Based on the analysis and prediction of events and system parameters of various server nodes with time-series model algorithm and event classification matching algorithm, combined with policy database technology, an autonomic computing model of server system management was established. This model was used to configure and reconfigure the computer system in accordance with the operating environment, protect it from disasters and heal it after disasters, and optimize it.

**Key words:** autonomic computing; event; prediction; policy

### 0 引言

随着企业的发展,日益庞大的服务器系统使得传统的以计算机管理员管理为主的管理方法已经不再适用。目前的应用对服务器系统提出了高可靠性(Reliability),高可用性(Availability),高可服务性(Serviceability),即高RAS的需求。因此,IBM提出了自律计算(Autonomic Computing)的概念,“一种能够以最少的人工干预实现系统的自我管理(Self Managing)的技术”<sup>[1]</sup>。自律计算的核心是系统的自我管理,系统利用一台高性能的服务器,控制和管理多个节点,逻辑上可以将整个系统作为一个服务器进行管理,从而实现系统自我配置、自我恢复、自我优化和自我保护。本文以事件驱动为基础,利用时序模型算法,事件集模式匹配算法,对节点的事件和系统参数进行分析和预测,然后根据策略库做出决策的自律计算模型。通过原型系统的检验,该模型实现了服务器系统的自律计算特征,达到了减轻管理负担,提高RAS的目的,具有很高的实用价值。

### 1 模型描述

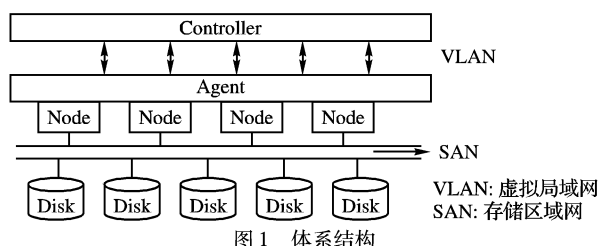


图1 体系结构

本模型的体系架构如图1所示,共有 Agent 和 Controller

两层。Agent 层作为代理负责节点和 Controller 的交互,而 Controller 则负责了整个系统。两者之间通过虚拟局域网 VLAN 连接。

#### 1.1 Agent 层

Agent 层架设在每一个节点上,由软件和硬件两部分组成。见图2。

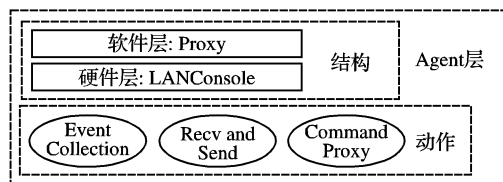


图2 Agent层说明

软件部分由一个 Proxy Service(代理服务)程序完成,主要完成以下功能:

- 1) 收集节点事件和系统参数,通过过滤得到 Controller 关心的节点信息;
- 2) 定期发送节点信息,接收 Controller 的命令;
- 3) 代理执行 Controller 的命令,包括日常维护、任务委托等。

硬件部分由一个称为 LANConsole 的固件组成,主要完成一些在软件层无法完成的远程控制任务,如开/关电源,安装 OS,启动 OS,系统完全崩溃的数据和任务恢复。

#### 1.2 Controller 层

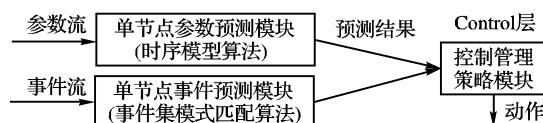


图3 Controller层说明

收稿日期:2005-09-12 修订日期:2005-11-21

**作者简介:**樊星(1981-),男,陕西西安人,硕士研究生,主要研究方向:软件理论、网络软件; 李战怀(1961-),男,陕西西安人,教授、博士生导师,主要研究方向:软件工程、数据库、数据仓库、Web与数据库、多媒体实时数据库技术; 刘全中(1979-),男,陕西西安人,硕士研究生,主要研究方向:软件理论、网络软件。

Controller 层是整个系统的核心部分。它通过对 Agent 层传来的节点信息进行分析,预测出节点在未来时间的系统参数和危险事件。然后,根据预测结果实施相应的策略,从而完成了整个系统的自律管理,如图 3。

Controller 层主要由单节点参数预测模块、单节点事件预测模块和控制管理策略模块三部分组成,分别应用时序模型算法和事件集模式匹配算法完成各部分的功能。

### 1.2.1 单节点系统参数预测

单节点系统参数预测利用时序模型算法实现。在本文中我们使用 LAST 线性时序模型<sup>[2]</sup>对一些系统参数进行预测,如系统利用率(% sys),系统空闲时间(% idle),网络 I/O(% IO)等。我们在记录的系统日志中选取一部分的系统参数信息为样本,以此作为时序模型的初始输入参数,建立时序预测模型。

在单节点分析的情况下, LAST 时序模型有以下特点<sup>[3]</sup>:

一般而言,由于 LAST 模型对系统参数类型的改变不敏感,在更换需要预测的参数时 LAST 模型并不需要进行太大的改动,因此,在这种意义上,可以认为 LAST 模型优于其他的线性时序模型。

整体上,在训练数据集比较大的时候, LAST 模型的预测结果优于其他的线性时序模型。特别的,对于一些连续数据, LAST 模型的预测平均错误率随训练数据集的增大而单调减小。

通过以上的模型,我们可以得到系统参数的预测结果。

### 1.2.2 单节点事件预测

我们设计了一个事件集模式匹配算法框架来实现单节点事件预测。

#### 1) 预测目标集的定义

基于单节点维护的经验,可以首先标识出一些预测优先级较高的事件,比如一些会对系统的运行产生不利影响的事件,我们把这些事件称为危险事件集<sup>[4,5]</sup>,定义为:

$Events_{danger} = \{event_{danger1}, event_{danger2}, \dots, event_{danger_n}\}$  这里的事件  $event_{danger_i} (i=1, \dots, n)$  正是我们的预测目标。

#### 2) 预测映射关系集的建立

系统的事件并不是孤立的,尤其是一些危险事件。实践表明,80% 以上的危险事件是由它之前若干事件导致的。基于这个事实,我们可以认为:危险事件  $event_{danger_i}$  与它之前发生的离散的事件集  $Events_{reason\_danger}$  之间存在映射关系如下:

$$f: Events_{reason\_danger} \rightarrow event_{danger} \quad (1)$$

事件集  $Events_{reason\_danger}$  定义为:

$$Events_{reason\_danger} = \{event_{danger\_r1}, event_{danger\_r2}, \dots, event_{danger\_rk}\}$$

其中  $event_{danger\_rj} (j=1, \dots, k)$  为离散事件,即  $event_{danger\_rj}$  和  $event_{danger\_r(j+1)}$  不需要在系统事件流中连续。

显然,理论上,在事件记录充足的情况下,我们可以对危险事件集  $Events_{danger}$  中的所有事件建立起如式(1)的映射关系。

在上述逻辑推导的基础上,我们对一个拥有 200 个节点的服务器集群系统在一年的运行中产生的系统事件日志进行分析学习,从而得到了针对每一个危险事件的映射关系:

$$f_{ij}: Events_{reason\_danger_{ij}} \rightarrow event_{danger_i}, \\ (i=1, \dots, n; j=1, \dots, m)$$

对一个危险事件  $event_{danger_i}$  可能存在多个事件集导致它的发生,因此对于  $event_{danger_i}$  可以存在多个映射关系  $f_{i1}, f_{i2}, \dots, f_{im}$ 。

如上,我们得到了一个映射集:

$$F = \{f_{ij} | i=1, \dots, n; j=1, \dots, m\},$$

可以一般化为  $F: Events \rightarrow event$ 。

### 3) 运行时事件预测

系统运行之后, Controller 从 Agent 层获得节点事件流,并将其存储在 Controller 层的事件流数据库中。每经过一个特定的时间间隔, Controller 便在节点事件流中查找是否存在与映射关系集  $F$  中的一个映射  $f_a$  的左边相匹配的事件集,如果存在,则根据  $f_a: Events_a \rightarrow event_a$ , 可以预测出节点有可能发生危险事件  $event_a$ 。这样,我们便完成了单节点事件预测的任务。

### 1.2.3 控制管理策略

控制管理策略模块是 Controller 层的控制核心。

控制管理策略模块以单节点系统参数和事件的预测结果为基础,选择相应的策略,并委托实施<sup>[6]</sup>。

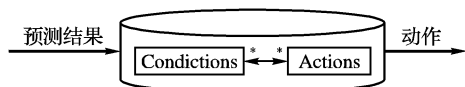


图4 策略库

如图4,系统维护着一个策略数据库,它由两部分组成,条件部分是若干条事件与参数的组合,而动作部分则是由若干个动作集组成,这两部分之间存在着多对多的对应关系。系统将预测结果与策略库中的条件部分进行匹配,如果匹配成功,则执行对应的若干个动作集(动作集中的动作不只针对一个节点,可能是整个系统的协调,如负载均衡工作等),如果匹配不成功,则执行默认动作集。在默认动作集中追加一些通用的配置动作即可实现自我配置功能。

同时,该策略数据库是可以扩充的,可以随时对条件部分和动作部分进行追加和重定义,以此满足新情况的需求。

综上, Controller 层通过分析,预测,策略选择,以及最终的动作实施,完成系统的自我配置、自我恢复、自我优化和自我保护,达到建立一个自律系统模型的目标。该模型具有较大的扩展性,用户可以通过扩充预测模块的初始训练数据提高预测的准确度,通过扩充策略增加解决方案的灵活性,增强处理能力。

## 2 模型的应用

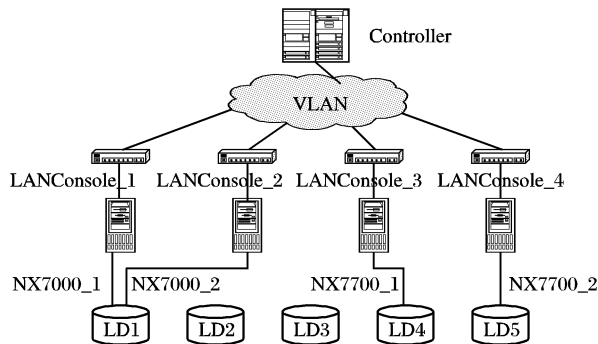


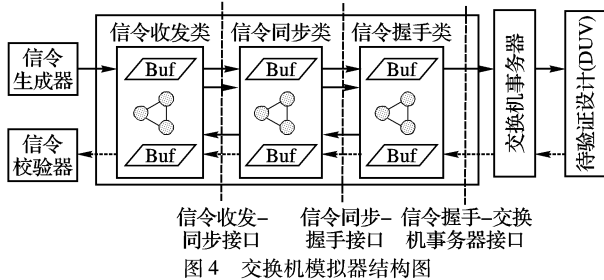
图5 DPM 说明

上述自律计算模型已经应用在一个实际的自律管理系统 DeploymentManager 中,以下简称 DPM。该系统的设计目的是为了减轻某公司 HP NX7000, NX7700i 服务器群的管理负担,并提高其 RAS。如图5。

该系统能够根据各个服务器的负载自动进行任务调度,

(下转第 710 页)

图4中垂直虚线所示,我们在类的接口方法内进行事务记录操作,可以记录每个事务发生的情况,并且可以人为地引入差错,记录待验证设计的响应来分析其设计是否符合协议规范。例如为了验证ARQ机制的算法是否有漏洞,我们人为地引入差错,通过记录在信令同步模块与信令收发模块间的信令交互过程,来分析待验证设计是否正确检测到错误,并请求交换机进行差错重传操作。我们还可以人为地请求重传,并检验待验证设计是否正确执行了重传操作等。



其次,介绍一下信令校验器是如何设计的。由于我们的系统只是在主机与交换机间传递信令,而不做变换,故我们在设计校验器时采用了 feed-forwarding 校验方法<sup>[6]</sup>。所谓 feed-forwarding 校验方法是指由校验器自动进行校验,而只将错误信息记录下来以供事后分析。信令生成器在将信令传给交换机模拟器的同时还将其作为参考信令交给信令校验器,信令校验器从主机模拟器收到信令后将其与存储的参考信令进行比对,如果出错就记录到文件中以供分析。

最后,介绍一下信令生成器是如何设计的。在交换机与我们的待验证设计之间是通过串行线路通信的,为了在比特流中定位出一个帧的开始,采用了伪随机数生成与鉴别机制。我们需要验证设计的伪随机数鉴别机制不会错误地认可其他的非特定伪随机比特流。我们采用了随机数生成机制<sup>[4]</sup>来产生激励以检验伪随机数鉴别模块的响应。

### 2.3 对比试验

为了说明使用 SystemC 的基于事务的验证方法的有效性,我们进行了对比试验,分别由两个验证小组采用不同的验证方法独立地对同一个待验证设计进行了验证,并且对比了这两种验证方法的验证效果。结果证明,使用 SystemC 的基于事务的验证方法比传统验证方法具有更高的验证效率。

例1 我们在设计ARQ机制时产生了一个很隐蔽的缺

陷,即有时整条信令接收正确而缺少部分码组,通过对事务记录的分析发现是在处理每条信令的第0xF个码组时出现了丢失,但不同信令长度不同,所以造成了好像出错是随机发生的情况。在普通的验证方法时,由于将注意力关注在信号层,我们只能将注意力集中到某个部分在某一时刻的静态状况。此时,我们的关注点就容易被限制在验证一个码组是否被正确地接收,面对这种码组接受“随机”出错的情况,很难着手分析,结果用了将近4天来定位这个缺陷。而在事务层,我们对整个芯片各个部分的状态、状态变化、各个部分间的通信及由此引发的各部分状态变化的因果关系有一个高层、全面而详尽的观察。此时,我们将关注点提高到验证码组流是否被正确地接收,通过对码组流的分析,只用了几个小时就定位了问题。

例2 我们在设计伪随机数鉴别机制时产生了另外一个漏洞,对一种特殊序列会进行误判。当用简单的验证方法时,由于只是手工产生激励数据,测试覆盖率很低,很难发现这个引起误判的序列,事实上用简单验证法并未发现此缺陷,而是在实际测试中发现的。而应用本文所述方法的随机数生成技术后,由激励生成模块自动生成大量测试数据,进行自动、长期、高覆盖率的验证,在进行前端验证时就找出了这个漏洞。

通过具体的对比试验,可见两种验证方法的效果明显不同。从第一个实例可看出,在事务层进行验证,让我们把精力集中到事务的因果分析上,忽略了大量无关的细节。从第二个实例可看出,使用 SCV 提供的机制提高了验证的效率。

### 参考文献:

- [1] IP CN, SWAN S. Using Transaction-Based verification in SystemC [Z]. Candence Design System, 2002.
- [2] PERANANDAM PM, WEISS RJ, RUF J, et al. Transactional Level Verification and Coverage Metrics by Means of Symbolic Simulation [Z]. Department of Computer Engineering University of Tübingen, 2003.
- [3] WANG R. Reuse Issues in SoC Verification Platform [A]. Proceedings of the 8th International Conference on Computer Supported Cooperative Work in Design [C], 2004. 2685 - 2688.
- [4] SystemC Verification Standard Specification [Z] SystemC Verification Working Group, 2003.
- [5] XU S, POLLITT - SMITH H. Creating a reusable testbench using cadence's testbuilder and AMBA TVM [A]. Canadian Conference on Electrical and Computer Engineering [C], 2003, 1. 97 - 102.
- [6] TestBuilder Reference Manual [Z]. Candence Design System, 2003.

(上接第704页)

根据服务器的具体作业需求进行资源分配,在服务器出现故障后根据故障种类进行任务回滚或者任务重新分配并修复服务器。这些特点满足减轻管理负担,提高RAS的要求。

### 3 结语

本文提出了一种自律计算模型,用以实现对服务器系统的自律管理。该模型以预测为基础,以策略管理为核心,通过策略的实施对系统进行管理。应用该模型,可以实现服务器系统的自我配置、自我恢复、自我优化和自我保护,解决当前服务器系统所面临的主要问题。

### 参考文献:

- [1] KEPHART JO, CHESS DM. The Vision of Autonomic Computing [J]. IEEE Computer, 2003, 36(1): 41 - 50.
- [2] BROCKWELL PJ, DAVIS R. Introduction to Time-Series and Fore-

casting [M]. Springer-Verlag, 2002.

- [3] SAHOO RK, RISH I, OLINER AJ, et al. Critical Event Prediction for Proactive Management in Large-scale Computer Clusters [A]. KDD-2003 [C], 2003.
- [4] VILALTA R, MA S. Predicting Rare Events in Temporal Domains [A]. Proceedings IEEE Conference on Data Mining (ICDM'02) [C], 2002.
- [5] SAHOO RK, BAE M, VILALTA R, et al. Providing Persistent and Consistent Resources through Event Log Analysis and Predictions for Large-scale Computing Systems [A]. SHAMAN Workshop, ICS'2002 [C], 2002.
- [6] CALO SB, VERMA D. A Toolkit for Policy Enablement in Autonomic Computing [A]. ICAC-04, International Conference on Autonomic Computing [C]. IEEE Computer Society, National Science Foundation, IBM Corporation, SUN Microsystems, 2004.