第35卷 第16期 计 算 机 工 程 2009年8月
Vol.35 No.16 Computer Engineering August 2009
·软件技术与数据库· 文章编号：1000—3428(2009)16—0056—03 文献标识码：A 中图分类号：TP311.52

空间机器人高可信软件检错技术

高 星，廖明宏，吴翔虎
(哈尔滨工业大学计算机科学与技术学院，哈尔滨 150001)

摘 要：提出一套适用于空间机器人的高可信软件设计模型和算法，将空间机器人软件错误检测分为单元级和系统级2个层级，针对单元级检测设计程序基本块模型以及基于该模型的数据流和控制流错误检测算法；针对分布式软件系统级的错误检测，设计多节点自适应冗余模型，在此基础上设计基于微检查点的错误检测算法，对空间机器人软件系统的错误检测形成一个完整覆盖，该方法已得到成功应用。
关键词：空间机器人；高可信；错误检测

Error Detection Technology of Space-robot High-dependable Software

GAO Xing, LIAO Ming-hong, WU Xiang-hu
(School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001)

【Abstract】This paper presents high-dependable software design model and algorithm used in space-robot software. The software errors in the space-robots are divided into two categories: the program block model and algorithms build on it are designed for the element level, and the distributed adaptive redundancy model and related algorithms are used to detect the system level errors. The cooperation leads to a complete error detection coverage. The model and algorithm is applied effectively.
【Key words】space robot; high-dependable; error detection

1 概述

空间机器人是一种低成本的轻型遥控机器人，可在行星的大气环境中导航及飞行。空间辐射常在电子器件中引起瞬时故障，而硬件瞬时故障往往引起软件运行错误。为实现这类系统的高可信，必须预防和应对这类错误[1]。

斯坦福大学的高级研究和全球观测卫星(Advanced Research and Global Observations Satellite, ARGOS)项目提出软件实现的硬件容错(Software Implemented Hardware Fault Tolerance, SIHFT)技术第一次尝试应用软件方法，使空间系统的软件对瞬时故障具有免疫能力[2]，该方案在低地球轨道上工作良好，但仍存在很多限制，比如它只能运行在具有超标量技术的处理器上，处理器不能使用cache技术，系统必须是单线程的，需要源代码，不支持分布式环境等。

为突破这些限制，文献[3]将SIHFT技术改造成软件实现的容错(Software Implemented Fault Tolerance, SWIFT)技术，使这项技术可以在现代商业处理器上运行，文献[4]提出一种iCOTS方法，该方法能帮助使用COTS组件开发可信软件系统，但这2种技术都不是基于组件的，且不适用于分布式计算环境。文献[5]提出一种变色龙的软件实现的容错框架(Software Implemented Fault Tolerance, SIFT)，使分布式软件有了容错的框架，该技术是基于组件的，但不能被应用于包含第三方组件软件系统中。

本文在单元级和系统级2个层级对空间机器人软件系统进行错误检测，在单元级采用一种基于程序流图的软件模型，在编译的同时添加一些用于错误检测的指令；而在系统级则是采用一种分布式的冗余模型，它利用系统中的空闲资源来实现错误检测。该套方案主要特点有：(1)不需要人工干预；(2)适用于多节点环境，不受单机单线程限制；(3)实现错误检测的完整覆盖。

2 单元级错误检测

2.1 基于程序基本块的控制流图

对于一个汇编语言源程序或者由高级语言源程序编译而得到的伪汇编程序，如果一系列指令是从第一条开始到最后一条结束顺序执行的，满足这一条件的指令的最大集合称为一个基本块；而一个无存储基本块是指除了最后一条指令不存在其它存储指令或分支指令的一系列指令的最大集合。

定义集合 $V=\{v_1, v_2,\cdots, v_n\}$表示程序中的所有基本块，另一个集合 $E=\{b_{ij}|b_{ij}$ 是从 v_i 到 v_j 的分支$\}$ 表示程序中所有基本块之间的跳转关系，则程序可表示为程序流图 $P=\{V, E\}$。若 $b_{ij}\in E$，v_j 是 v_i 的后继，则记作 $v_j\in suc(v_i)$；若 $b_{ij}\in E$，v_i 为 v_j 的前驱，记作 $v_i\in prev(v_j)$。如果 v_i 到 v_j 的分支是非法的，那么 $b_{ij}\notin E$，非法分支就表明了一个控制流错误。

同样，如果 $U=\{u_1, u_2,\cdots, u_n\}$表示程序中的所有无存储基本块的集合，对 U 中的每一块给出一个功能相同的影子块，称为 $U'=\{u_1', u_2', \cdots, u_n'\}$，如果 U 和 U'中程序的执行结果一致，则认为其间没有发生数据流错误，反之，则表明其间

基金项目：国家“863”计划基金资助项目(2005AA742013)
作者简介：高 星(1980－)，男，博士，主研方向：高可信系统，实时嵌入式计算；廖明宏，教授、博士、博士生导师；吴翔虎，教授、博士
收稿日期：2008-12-06 **E-mail：**hs_gaoxing@126.com

发生了数据流错误。

2.2 虚拟寄存器及其分配方法

现有的软件故障检测方法或是对源代码修改，或是对源代码编译的过程进行修改，它们都需要额外的寄存器支持，这限制了现有技术的使用范围。虚拟运行时签名寄存器(Virtual Run-time Signature Register, VRSR)是指利用内存或者一些极少使用的寄存器临时模拟寄存器，这使汇编语言源程序或高级语言源程序编译生成的伪汇编程序也能被处理。

2.3 控制流错误检测

基于虚拟寄存器的控制流检测(Control Flow Checking by Virtual Register, CFCVR)是基于 XOR 运算的，它通过虚拟寄存器的应用突破了在文献[1]中必须的 2 个保存动态签名的寄存器的限制，同时减少了错误检错盲点，原理如下:

对于程序流图中的每一个节点 v_i，分配唯一的静态签名 s_i，设程序运行时的动态签名为 G，G 在节点 v_i 的动态签名值为 G_i，用来保存 G 值的是一个已经分配好的 VRSR。

设 $G_{prev(i)}$是 v_i 的一个前驱节点的动态签名，v_i 节点用来辅助验证动态签名的静态值为 d_i，d_i 用来表示 $G_{prev(i)}$和 G_i 签名差异。在程序发生从 $prev(v_i)$到 v_i 跳转时，按一个功能函数 $f(G_{prev(i)}, d_i)$计算 v_i 节点的动态签名 G_i，比较 s_i 和 G_i 的值是否相等，如果 s_i 和 G_i 的值相等，则程序继续执行，反之则认为程序发生了控制流的错误，程序进入相应的错误处理。其中，$f(G_{prev(i)}, d_i)$是基于 XOR 运算的，定义为 $f(G_{prev(i)}, d_i)= G_{prev(i)} \oplus d_i$, $d_i=f^{-1}(s_{prev(i)}, s_i)=s_{prev(i)} \oplus s_i$。

假设有 M 个合法分支 $b_{(12\cdots M)j}$进入扇入节点 v_j，对 v_j 不同的前驱节点组成的集合 $V_{prev(j)}$中的每一个节点分别计算不同的 d_j 值组成集合 D_j，这些值并不相等。对此，尝试在进入节点 v_j 后对每一个可能的 d_j 值分别跟 G_j 进行匹配，遇到匹配的则停止匹配并跳转到 v_j 内的原有代码进行执行，反之如果在 D_j 中没有找到能匹配的 d_j 值，则认为发生了一个控制流错误。

2.4 数据流错误检测

基于虚拟寄存器的数据流检测(Data Flow Checking by Virtual Register, DFCVR)算法是一种基于 VRSR 的技术，它对每个无存储基本块进行重复调用(即影子块是其本身)，然后通过比较 2 次调用的结果来验证数据流的正确性，其间应用 VRSR 来存储块内需要保存的现场信息。使用重复调用策略可以避免应用影子指令从而可以避免在编译阶段保留一半寄存器。如果所操作的块是非可重入的，则需要在第 1 次执行块内代码前保存现场，执行后保存结果；第 2 次执行代码前恢复现场，执行后比较 2 次的计算结果。

3 系统级错误检测

3.1 多节点自适应冗余模型

分布式自适应冗余模型(Distributed Adaptive Redundancy Model, DARM)是一个基于 SIFT 技术的框架，它作为操作系统中的独立子系统存在，使应用程序可以可靠地运行在由多个非容错硬件节点组成的网络上。它定义了 3 类角色: (1)冗余单元(Element)，它是软件的基本组成部分，可能是用户开发的，也可能是第三方的，一个 Element 可以是由若干其他 Element 组合而成；(2)后台(Daemon)，它是指被安装在各个节点上的代理，充当 Element 之间通信的网关，对一个节点上的所有 element 提供错误检测服务；(3)管理者(Manager)负责监管整个冗余系统中的所有后台，并且负责安装后台、冗余单元，检测和处理错误，初始化或重置其从属的后台。

假定 *Soft* 是一个完整的软件系统，那么它可以看作是由一个冗余管理者和若干分布在各个节点上的其他部分组成的，假定 $Node_k$ 为分布在节点 k 上的那部分。$Node_k$ 又是由一个冗余管理后台和若干的冗余单元组成，定义节点 k 上的后台为 $Daemon_k$，它所有管理的该节点上的任一冗余单元为 $Element_{k,i}$，则定义为

$$Soft = \{Manager\} \cup \{Node_k \mid k \in N, K > 0\}$$

$$Node_k = \{Daemon_k\} \cup \{Element_{k,i} \mid i \in N,\ i > 0\}$$

$Element_{k,i}$ 被重复后的单元称为影子单元 $SElement_{k,i,j}$，它们被人工或自动地分布在系统内的各节点上。

$$Element_{k,i} = \{SElement_{k,i,j} \mid j \in N\}$$

每个后台管理所在节点的所有单元，它们使用本地冗余表(Local Redundancy Table, LRT)来记录该节点上所有影子单元、处理器和存储器等所有资源的分布；而管理者周期性地同步各节点上的 LRT 组成全局冗余表(Global Redundancy Table, GRT)，据此决定如何利用这些冗余来组成运行时冗余方案。

3.2 基于微检查点的错误检测

传统的检查点技术中为管理检查点的状态，须频繁挂起线程或进程，这是影响系统性能的主要原因。在 DARM 模型的基础上可在多个影子单元之间通过各自记录、异步检查的方式来减少系统挂起的频率，从而提高系统的性能。在影子单元的内部添加为检查点，用来记录其运行过程中的状态，并与其他同源的影子单元的状态数据进行比较，从而判断其是否正确执行。各单元的错误检测建立在各节点可以相互通讯的基础上，任意软件单元 $Element_{k,i}$ 的所有影子单元会收到发送给它的相同信息并由后台对输出进行比较，如发现异常，冗余管理者会决定重启模块或重新计算。

本地检查点状态表(Local Checking-point Status Table, LCST)和全局检查点状态表(Global Check-point Status Table, GCST)分别被 Manager 和 Daemon 用来记录检查点的状态，其中，GCST 是各 LCST 的动态合集。整个错误检测过程如下:

(1)Manager 端:

1)LCST 中的变化项被周期性地发送到 Manager 端，被更新到 GCST 中；

2)Manager 异步检查 GCST，对源于同一个 Element 的所有影子单元的相同位置、相同计算步骤的检查点状态进行检查，如果发现错误，则向错误处理单元发送一个错误报告；

3)当 Manager 中的错误处理单元收到一个错误报告，则记录并将错误处理决定通知 Daemon。

(2)Daemon 端:

1)判断和维护 LCST，定期将其更新发送到 Manager 端；

2)如果收到 Manager 的错处处理决定，则依照决定设置本节点，重新开始运行。

4 实验及分析

实验分为 2 个部分: (1)采用 LZW 压缩算法，矩阵乘法，快速排序等 3 个基准程序来测试该方法的可行性及性能；(2)选择哈工大某重大航天课题星务管理程序(简称 A 课题)来测试算法在实际项目中的表现。所有实验都是基于 GNU C 编译器编译生成的.s 文件进行。实验方案选用的故障注入工具是基于 GNU 调试器 gdb 的，它通过直接修改相应跳转指令的数据段来达到注入不同控制流故障的目的。检测结果如

表 1 所示。表 2 给出了程序源码的统计数据以及存储和性能代价的统计数据。

表 1　各级别检测比较

未测出的错误输出	LZW /(%)	矩阵乘法 /(%)	快速排序 /(%)	A 课题 /(%)
无措施	26.2	33.8	33.6	30.6
单元级	7.2	10.2	9.0	6.7
系统级	6.1	8.5	7.8	6.2
单元+系统	4.6	7.0	6.3	5.1

表 2　程序源码统计数据及存储和性能代价统计数据

	指令数	检查指令数	基本块数	存储代价 /(%)	性能代价 /(%)
LZW	452	252	105	55.7	44.3
矩阵乘法	763	232	102	30.4	27.2
快速排序	254	111	47	43.7	37.8
汉诺塔	179	60	27	33.5	30.1
A 课题	16 632	5 207	1 851	31.3	28.7

在统计的 1 851 个基本块中，总前驱数目为 2 280 个；1 423 个节点只有 1 个前驱节点；有 2 个前驱节点的节点数目为 427；有 3 个前驱节点的节点数目是 1 个；未发现超过 3 个前驱节点的节点。由此可见具有单个前驱的节点占节点总数的 77.88%；有 2 个前驱节点的节点数目为 23.07%；超过 2 个前驱节点的节点对算法性能的影响极小。系统级的冗余可以根据实际情况作调整，构建一个类似三模冗余的系统会引入约 200%的空间冗余，但是因为本方案充分利用了系统的冗余资源，选择资源使用率低的节点来动态组成冗余方案，所以任务的处理时间和系统的响应速度却不会成反比线性降低，在 A 课题的实测中系统时间开销约为原始状态的 185%。

5　结束语

本文结合哈工大某重大航天课题星务管理系统的设计，提出一种纯软件的控制流检测方法，它在引入平均 28.7%的性能代价(实际数值跟系统冗余状态相关，三模时的参考值为 85%)和平均 31.3%的存储代价(多个影子模块则乘以相应系数)的情况下，对于控制流错误检测率达到平均 94.9%，能够提高航天软件容错能力和可靠性。未来可以尝试将本方法引入其他领域的应用当中。

参考文献

[1] 包海超，杨根庆，李华旺．小卫星星载软件微内核的设计[J]. 计算机工程, 2008, 34(9): 81-82.

[2] Shirvani P P, McCluskey E J. Fault-tolerant Systems in a Space Environment: The CRC ARGOS Project[R]. Stanford, CA, USA: Stanford University, Tech. Rep.: CSL-TR-98-774, 1998.

[3] Reis G A, Chang J. SWIFT: Software Implemented Fault Tolerance[C]//Proc. of CGO'05. San Jose, California, USA: [s. n.], 2005.

[4] Asterio C G P R, Romanovsky C M F A. Integrating COTS Software Components into Dependable Software Architectures[C]//Proc. of the 6th IEEE International Symposium on Object-oriented Real-time Distributed Computing. Hakodate, Hokkaido, Japan: IEEE Press, 2003.

[5] Saurabh B K W, Kalbarczyk Z, Iyer R K. Hierarchical Error Detection in a Software Implemented Fault Tolerance(SIFT) Environment[J]. IEEE Trans. on Knowledge and Data Engineering, 2007, 12(2): 203-224.

编辑　金胡考

(上接第 55 页)

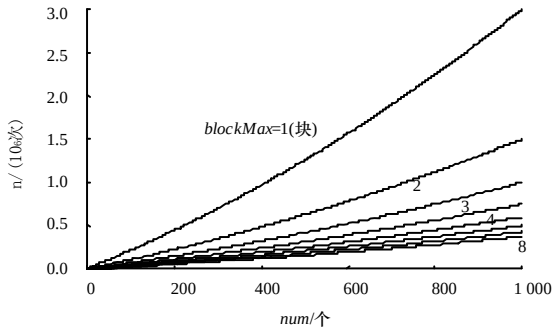

图 3　模型选取区间

2)当 $S>1j$ 时，由式(4)得：

$$(M\text{-}P\text{-}blockMax)\times n=num^2+20P\times num \quad (6)$$

显然，$M=P$ 时，式(5) ⇔ 式(6)。$M>P$ 时，相当于对参数 *blockMax* 的叠加。若 $S=2$，即使 *blockMax*=1，效果也相当于拥有 2 001 个文件信息块。如图 3 所示，此时的曲线几乎和 *num* 轴重合。所以当 $S>1$ 时，应该选择优化模型。

4　结束语

通过分析计算可以看出，FAT 离散度决定了 2 种模型的取舍，只有当平均离散度大于 1 时，优化模型才可以使文件的读写速度得到显著提升。优化模型避免了系统对 FAT 表冗余数据的频繁读取，加快了文件 FAT 信息入口地址的访问，具有较高的实用价值。

当然，优化模型还存在一些不足。若 FAT 离散度在 1 和大于 1 之间不断变化，需要动态地在一般模型和优化模型之间作出正确的选择。如何写出正确高效的动态选择算法将是今后这方面的研究重点。

参考文献

[1] Park S, Ohm Seong-young. New Techniques for Real-time FAT File System in Mobile Multimedia Devices[J]. IEEE Transactions on Consumer Electronics, 2006, 52(1): 1-9.

[2] Lim Seung-ho, Park Kyu-ho. An Efficient NAND Flash File System for Flash Memory Storage[J]. IEEE Transactions on Computers, 2006, 55(7): 906-912.

[3] Microsoft Corporation. Extended FAT File System[EB/OL]. (2008-03-12). http://msdn2.microsoft.com/en-us/library/aa 914353.aspx.

[4] Microsoft Corporation. FAT File System[EB/OL]. (2008-03-12). http://msdn2.microsoft.com/en-us/library/aa916708.aspx.

[5] Intel Corporation. Understanding the Flash Translation Layer (FTL) Specification[EB/OL]. (1998-12-10). www.embeddedfreebsd.org/Documents/Intel-FTL.pdf.

编辑　张正兴