自适应一致表决算法

欧阳城添[1,2] 王 曦[1] 郑 剑[1,2]
(同济大学计算机科学与技术系 上海 201804)[1] (江西理工大学信息工程学院 赣州 341000)[2]

摘 要 容错技术已经在许多领域的高可靠控制中得到应用,N版本程序技术是实现容错的基本手段之一。在软件系统中,表决算法可以屏蔽错误的输出结果。冗余技术可以防止错误的结果传递到系统的下一个子模块中,并且提高系统的安全性。许多表决算法在容错技术中得到广泛的应用,其中一致性表决算法同样得到了广泛的应用。但一致表决算法适合输出结果空间基数小的情况,因此更容易产生相同并错误的结果(IAW)。针对这个问题,提出一种自适应的一致性表决算法,它将版本历史记录信息应用到一致性表决中,降低了不正确结果通过表决的概率,提高了系统安全性和可靠性。实验证明了所提算法的有效性。

关键词 一致表决算法,N版本程序,容错技术

Adaptive Consensus Voting Algorithm

OUYANG Cheng-tian[1,2] WANG Xi[1] ZHENG Jian[1,2]
(Department of Computer Science and Technology, Tongji University, Shanghai 201804, China)[1]
(Jiangxi University of Science and Technology, Ganzhou 341000, China)[2]

Abstract Fault-tolerant technologies have been applied in many high reliability fields. N-version programming technology is one of those Fault-tolerant technologies. In the software system, voting algorithm can mask the error outputs, redundant technology can prevent false results from propagating to the next sub-module in the software system, and enhance the reliability of the system. Many voting algorithms are widely used in fault-tolerant technology, in which Consensus Voting Algorithm are also widely applied, but the consensus voting strategy is particularly effective in small output spaces because it automatically adjusts the voting to the changes in the effective output space cardinality. But the consensus voting algorithm is more prone to cause identical-and-wrong(IAW) results in this case. Therefore, to resolve this problem, this paper proposed an adaptive consensus voting algorithm. The history records information of modules are used to improve the power of consensus voting algorithm, the proposed algorithm reduces the probability of the incorrect results passing the voter and improves the system safety and reliability. Experiments demonstrate the power of our proposed algorithm.

Keywords Consensus voting algorithm, N-version programming, Fault-tolerant technology

1 引言

冗余技术在数字电子系统和软件系统中得到了广泛的应用,极大地提高了系统容错能力,从而提高了系统的可靠性。在一个系统中,使用冗余模块可以屏蔽有故障的模块;在软件系统中,表决算法也可以屏蔽错误的输出结果。冗余技术可以防止错误的结果传递到系统的下一个子模块中,并且提升系统的安全性。文献中提出了许多表决算法,例如多数表决算法[1]、n 中取 2 表决算法[2]、一致性表决算法[3]等。其中多数表决及其改进版在系统中得到了广泛的应用。多数表决器有 n 个输入,如果至少有$[(n+1)/2]$个模块一致,则一致的结果被选为表决器的输出;如果结果不一致,多数表决器产生一个异常代码,使系统转入一个失效安全状态和失效停止状态。因此,多数表决器在 TMR 冗余系统中可以屏蔽模块在表决中的失效,但不能识别出是哪个模块失效。

然而,在很多应用中,识别错误的模块是有必要的。为了提高系统的可用性等级,依据应用的性质,通过识别出错通道中相应的模块,就可以用备份的模块在线替换出错的模块,或者从系统中断开出错的模块,或用安全的方式关闭这个出错的模块。这些策略称为系统重置或称为安全降级和失效停止。在容错系统的环境下,有一种方法可以识别这些出错的模块,就是使用它们的历史记录信息。在这种方法中,历史记录最差的模块被认为是在系统运行中最容易出错的模块。应用模块历史信息也可以提高表决算法的性能。表决算法中使用模块的历史信息在其他文献中也有应用。

在文献[4]中,使用 TMR 冗余系统中软件模块的累积次数作为版本的历史信息。软件版本结果不一致的情况下,有最高历史次数的选为表决器的输出结果。文献[4]中提出的

到稿日期:2010-08-01 返修日期:2010-11-09

欧阳城添 男,讲师,主要研究方向为计算机系统结构、容错计算等,E-mail:oyct@163.com;王 曦 女,博士生,主要研究方向为容错计算、软件可靠性工程;郑 剑 男,副教授,主要研究方向为计算机软件与理论。

表决器比一般的多数表决器出错的概率更低。

上述研究工作使用了模块的近期历史信息，在不一致的情况下，根据历史信息产生输出结果。本文论述一致表决算法在使用历史信息时的表决机制。实验表明，它比普通的一致表决算法出错的概率更低。

本文第2节给出相关的一些定义、假设和概念；第3节介绍一致性表决算法及其存在的问题，并提出一种新的一致表决算法，即自适应一致性表决算法；第4节在C++环境下，分别实现了一致性表决算法和自适应一致表决算法，并进行了比较分析；最后总结本文的工作。

2 定义、假设和概念

2.1 定义

n：功能上等价的软件版本数。

m：赞同数，同意输出结果的软件版本数。

r：输出结果空间基数。

2.2 假设

(1)所有软件版本功能上是等价的，即满足相同的规格说明。各个版本的输出结果在统计学上是独立的。

(2)对于给定的输入，只有一个唯一正确的输出结果。如果输出结果标识为1…r，那么输出结果1表示唯一正确的输出。输出2…r是不正确的输出。

(3)对于给定的输入，所有版本程序都能够输出结果，不会出现没有结果输出的可能。输出结果的空间为小输出空间。

2.3 区别表决通过和正确结果的概念

实验表明，N版本程序发生相同错误的可能性比预期的要高得多[5-8]。例如，如果输出空间的基数为2，那么输出结果只有两种可能，假设为0和1，所有不正确的结果就自动一致，并且这种可能性很高。这就使N版本程序在表决过程中失效的可能性更高。因为表决器不可能确保正确地识别正确与不正确的结果，就只能把表决通过的结果当成正确的结果输出。

在基于软件多样性和表决策略的容错软件系统中，表决通过的结果和正确的结果是不同的，要加以区别[9]。例如，一个多数表决器如果多数模块输出结果一致，即通过表决，可以分为两种情况。第一种情况，多数模块产生正确的输出结果并且表决通过，见图1中CAA(Correctness And Agreement)；第二种情况是多数模块产生不正确的结果但也表决通过，见图中IAA(Incorrectness and Agreement)，这些不正确的结果也称为IAW。同样的表决没有通过的这些结果，也分为两种情况：第一种情况是正确的结果因为不是由多数模块输出的结果，在表决中被否决，即是图1中的CAF(Correctness And Failure)；第二种情况为不正确的结果没有通过表决，即是IAF(Incorrectness and Failure)。我们尽可能使正确的结果能通过表决，而不正确的结果在表决中被否决；尽可能避免正确的结果没能通过表决，而不正确的结果通过表决。本文就是为了这个目的而提出自适应一致表决算法。

导致正确的结果没能通过表决，而不正确的结果通过表决的原因有两点：第一，N版本程序设计虽然采用软件的相异性(diversity)策略进行设计和开发，但是由于软件功能需求是相同的，人员获取知识的来源也可能是相同的，手段和工具又可能是相近的，版本之间就存在不可克服的相关性，很难保证容错软件的版本独立性。第二，在极端的情况下模块输出结果空间的基数很小时，容易出现IAW错误，使正确的结果没能通过表决，而不正确的结果通过表决的可能性就更高。

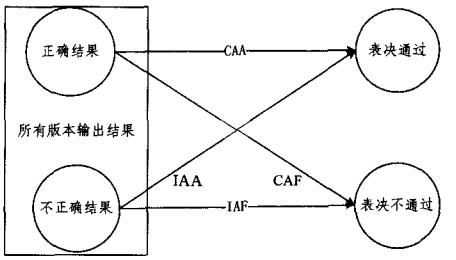


图1 多数表决器表决过程

例如图2中表示的是P1，P2和P3这3个程序失效空间的覆盖情况。如果3个版本程序是完全独立相关的，那么它们的失效空间就应该没有重叠情况。但这是很难做到的，多数情况下它们的失效空间是相互重叠的，如图2所示。从P1，P2和P3的失效空间的覆盖情况可以知道，程序的失效情况可以分为单一失效，即每个程序失效的空间没有重叠的部分，这时不可能发生IAW的情况；双重失效是两个程序的失效空间重叠的部分；三重失效是3个程序失效空间重叠的部分。当发生双重失效或三重失效时，就可能出现IAW的情况。

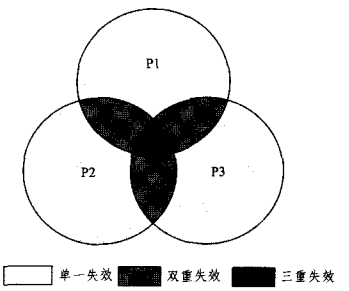


图2 3个程序P1、P2和P3的失效空间的覆盖情况

3 自适应一致表决算法

3.1 一致表决算法

一致表决算法[9]是多数表决算法的推广，是1990年由David F. McAllister等人提出的一种算法。一致表决算法描述如下：

1)如果输出结果为多数赞同，即$m>(n+1)/2, n>1$，那么选择这个结果作为输出。

2)如果有唯一的最大赞同数，并且赞同数小于$(n+1)/2$，那么选择这个结果作为输出。

3)如果几个输出有相同最大赞同数，那么随机地从中选择一组结果作为程序的输出。

4)其他情况，表决失效，停止，转入失效安全状态。

一致表决算法对于输出结果空间小的表决十分有效，因为它可以使表决自动地适应有效输出空间基数的变化。如果输出结果空间基数为2，一致表决算法和多数表决算法是相同的。若输出结果空间的基数趋于无穷，赞同数$m=2$就可以了，一致表决算法就和"2-out-of-N"表决算法是相同的。

3.2 一致表决算法中存在的问题

可能会把错误的结果当成正确的结果输出。例如上例

中，如果出现三重失效和双重失效，由于它们的失效空间重叠在一起，因此可能出现两个结果或3个结果相同并且是不正确的结果。

在算法第3步中，如果几个输出有相同最大赞同数，那么随机地从这些组中选择一组作为程序输出结果。这样就有50%的可能输出不正确的结果，从而降低容错软件的可靠性。

3.3 构建版本的历史记录信息

为了解决一致性表决算法存在的上述问题，本文提出一种新的解决方案，一种基于历史信息的一致表决算法，称之为自适应一致表决算法。本算法的核心步骤就是要构建版本的近期历史记录信息。

构建版本历史记录的常用方法是累计每个版本程序通过表决的次数[10]。在这种方法中，如果版本程序的计算结果通过表决，则记录这个版本程序的次数。那么，在某一次表决周期中，具有最大累积次数的模块可以认为是最可靠的模块，累积次数最小的模块认为是可靠性最低的模块。对于NMR系统，构建版本历史记录的策略定义如下：

1)构建一个 N 个元素的集合$\{m_1,m_2,m_3,\cdots,m_n\}$。如果在表决中某个输出结果通过了多数表决，则这个输出对应的模块 $m_i=1$，否则设置参数 $m_i=0$。因此，$m_i=1$ 表示在这次表决中模块 i 通过表决。

2)在每次表决周期累计 m_i 的值，在 n 次系统运行中获得 $H(i)=\mathrm{sum}(m_{ij})$，$j=1\cdots n$。这个值表示在 n 次系统运行中模块 i 通过表决的次数。

3)对历史记录 $H(i)$ 进行规格化处理，即 $P_i=H(i)/n$。那么，可以认为这个值是模块 i 的可靠性等级的参考值。一个 P_i 最高的模块 i 是最有可能产生正确结果的模块。另外，在任意一个表决周期，$P_i(j)$ 表示为模块 i 从第1次表决到这个表决周期 j 为止模块 i 的一种状态，所以这个值可以认为是在周期 j 的模块 i 的一个“状态指示器”。而 $H_i(j)$ 表示从第1次表决到这个表决周期 j 为止模块 i 的“历史记录”。

版本程序的历史信息可能有不同的用途。①离线使用历史记录可以识别在系统运行中最有可能出错的模块，识别出错模块可以在系统运行结束后进行。当系统终止运行后，通常可以把最后 P_i 值最大的模块认为是最有可能出错的模块。这个模块可以进行改进，或在下一次任务时用一个没有错误的备份模块替换。②在线使用历史记录信息。在任意一个表决周期，以这个周期 j 的模块 i 的“状态指示器”$P_i(j)$ 作为参考值，可以重置系统结构，降级运行、失效停止或安全失效，在运行期间提高表决算法的性能。本文主要工作就是利用历史记录信息在系统产生异常数据时对系统进行降级运行、失效停止或安全失效，以提高系统的安全性和可靠性。

3.4 自适应一致表决算法

假定有版本数 n 的程序，版本程序的输出空间基数为 r，用矢量$(V_1,V_2,V_3,\cdots,V_n)$表示各种可能输出状态的频次($V_1+V_2+V_3+\cdots+V_n=n$)，假定 V_1 表示正确输出的频次，V_2，V_3，$\cdots$，V_n 表示不正确输出的频次。算法描述如下：

1)赞同数为 $m>(n+2)/2$，选择频次大的输出作为结果，并记录每个版本历史信息。例如矢量$(V_1,V_2,V_3,\cdots,V_n)$中的 $V_1>(n+1)/2$，则选择 V_1 作为输出。

2)如果有唯一的最大赞同数，赞同数小于$(n+2)/2$，并且对应模块的“状态指示器”$P_i(j)$大于阈值 T，那么选择这个结果作为输出，阈值 T 可以根据用户的实际情况而设定。例如矢量$(V_1,V_2,V_3,\cdots,V_n)$中的 V_2 是最大的赞同数，$V_2<(n+1)/2$，且对应的模块“状态指示器”$P_i(j)>0.90$，则选择 V_2 作为输出。

3)如果有相同最大赞同数，赞同数小于$(n+2)/2$，则选择对应模块的“状态指示器”$P_i(j)>T$ 的结果作为输出。例如，假设矢量$(V_1,V_2,V_3,\cdots,V_n)$中的 V_1 和 V_2 具有最大的赞同数，且 $V_1=V_2$，$V_1<(n+1)/2$。但 V_1 对应的模块“状态指示器”$P_i(j)>0.90$，则选择 V_1 作为输出。

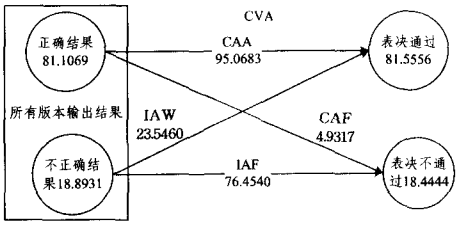
4)其他情况，表决失效，停止，转入失效安全状态。

4 实验与分析

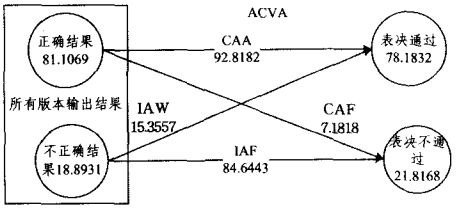
我们实现了一致性表决算法和自适应一致表决算法，实验环境为DELL PC(3.0GHz Intel(R) Core(TM)2 Duo E8400处理器、1.98GB内存)，程序采用C++开发。实验版本程序数为 $n=11$，结果空间基数 $r=3$，阈值 $T=0.9$。并采用仿真方法模拟11个版本程序的输出结果，这些输出结果属于结果空间(V_1,V_2,V_3)，再分别由一致表决算法和自适应一致表决算法进行表决，选择输出结果。

在实验中，假定 V_1 为正确的输出结果，11个版本程序的模拟产生的结果大都落在 V_1 空间中。每次实验时，进行10万次表决。在每次试验时会注入一些异常数据，用来调节正确输出结果和不正确输出结果的比例。例如实验1中正确结果占81.1069%，不正确结果占18.8931%。

实验1如图3(a)所示。用一致表决算法进行表决时(CVA)，版本程序输出的正确结果占81.1069%，不正确结果占18.8931%；正确的结果表决通过的有95.0683%(CAA)，正确结果没有表决通过的有4.9317%(CAF)；不正确结果表决通过的有23.5460%(IAW)，不正确结果没有通过表决的有76.4540%(IAF)；输出结果表决通过的占81.5556%，表决没有通过的占18.4444%。



(a)一致表决算法实验



(b)自适应一致表决算法实验

图3 一致表决算法和自适应一致表决算法实验图示

实验2如图3(b)所示。用自适应一致表决算法进行表决时(ACVA)，版本程序输出的正确结果占81.1069%，不正确结果占18.8931%；正确的结果表决通过的有92.8182%(CAA)，正确结果没有表决通过的有7.1818%(CAF)；不正

确结果表决通过有15.3557%(IAW),不正确结果没有通过表决的有84.6443%(IAF);输出结果表决通过的占78.1832%,表决没有通过的占21.8168%。

分析实验数据可知,在输出结果空间基数 r 较小时,通过表决而不正确的结果占比比较大(18.8931%),误报率(false positive rate)会比较高(23.5460%)。通过算法的改进,误报率降低,即产生 IAW 的可能性降低了 8.1903%。通过实验表明改进算法是有效的,提高了系统的安全性和可靠性。

表 1 是实验 1 和实验 2 的一致表决算法和自适应一致表决算法实验数据。两个实验都表明算法改进后误报率下降,说明通过改进算法提高了系统的安全性和可靠性。而 CAA 在实验 1 中有所下降,在实验 2 中则有所上升,说明改进后的算法更适应正确结果占比更高的情况。

表 1 一致表决算法和自适应一致表决算法实验数据

试验	实验 1			实验 2		
	CVA	ACVA	Incr.	CVA	ACVA	Incr.
Correct %	81.1069	81.1069	0.0000	93.0615	93.0615	0.0000
Incorrect %	18.8931	18.8931	0.0000	6.9385	6.9385	0.0000
CAA %	95.0683	92.8182	−2.2501	99.7875	99.8086	0.0211
IAW %	23.5460	15.3557	−8.1903	2.4410	1.2940	−1.1470
CAF %	4.9317	7.1818	2.2501	0.2125	0.1914	−0.0211
IAF %	76.4540	84.6443	8.1903	97.5590	98.7060	1.1470
Agreement %	81.5556	78.1832	−3.3724	93.0331	92.9731	−0.0599
Disagreement %	18.4444	21.8168	3.3724	6.9669	7.0269	0.0599

结束语 一致性表决算法在 N 版本程序设计中得到了广泛的应用。一致表决算法适合输出结果空间基数小的情况,因此产生 IAW 错误的可能性会更高。针对这个问题,本文提出了自适应一致性表决算法。实验表明,改进后的算法降低了不正确结果通过表决的概率,提高了系统的安全性和可靠性。

参 考 文 献

[1] Lorczak P R, Caglayan A K, Eckhardt D E. A Theoretical Investigation of Generalised Voters[C]// IEEE 19th Int. Ann. Symp. on Fault-Tolerant Computing Systems. Chicago, June 1989: 444-451

[2] Kanekawa K, Maejima H, Kato H, et al. Dependable On-board Computer Systems with a New Method: Stepwise Negotiated Voting[C] // IEEE 19th Int. Ann. Symp. on Fault-Tolerant Computing Systems. Chicago, June 1989: 13-19

[3] Bass J M. Voting in Real-tTime Distributed Computer Control Systems[D]. Sheffield, UK: Department of Automatic Control and System Engineering, The University of Sheffield

[4] Gersting J L, Nist R L, Roberts D B, et al. A Comparison of Voting Algorithms for N-version Programming[C]// IEEE 24th Ann. Hawaii Int. Conf. on Systems Sciences. Kauai, HI, USA, Jan. 1991, 2: 253-62

[5] Scott R K, Gault J W, McAllister D F, et al. Investigating version dependence in fault-tolerant software[R]. AGARD 361. 1984: 201-210

[6] Knight J C, Leveson N G. An experimental evaluation of the assumption of independence in multisession programming [J]. IEEE Trans. Software Engineering, 1986, SE-12: 96-109

[7] Bishop E, Humphreys B, Lahti D. PODS-A project on diverse software[J]. IEEE Trans. Software Engineering, 1986, SE-12: 929-940

[8] Eckhardt K, Knight C, McAllister V. A large scale second generation experiment in multi-version software: Description and early results[C]// Proc. FTCS 18. June 1988: 9-14

[9] McAllister D F, Sun C-E, Vouk M A. Reliability of Voting in Fault-tolerant Software Systems for Small Output-spaces[J]. IEEE Log, 1990, 39(5): 524-534

[10] Latif-Shabgahi C, Bennett S. Adaptive Majority Voter: A Novel Voting Algorithm for Real-time Fault-tolerant Control Systems [C]// Proceedings of 25th EUROMICRO Conference. Milan, Italy, September 1999, 2: 113-120

(上接第 116 页)

[3] Gousios G, Spinellis D. A Platform for Software Engineering Research[C] // 6th IEEE International Working Conference on Mining Software Repositories. 2009: 31-40

[4] Gousios G, Spinellis D. Alitheia Core: An extensible software quality monitoring platform [C]// Proceedings of the 31st International Conference on Software Engineering(ICSE '09)—Formal Research Demonstrations Track. 2009: 579-582

[5] Tsunoda M, Monden A, Yadohisa H. Productivity Analysis of Japanese Enterprise Software Development Projects [C]// Proceedings of the International Workshop on Mining Software Repositories. 2006: 14-17

[6] Van Antwerp M, Madey G. Advances in the SourceForge Research Data Archive [C]// The 4th International Conference on Open Source Systems(WoPDaSD 2008). Milan, Italy, September 2008

[7] Walker R J, Holmes R, Hedgeland I, et al. A Lightweight Approach to Technical Risk Estimation via Probabilistic Impact Analysis [C]// Proceedings of International Workshop on Mining Software Repositories. 2006: 98-104

[8] Zimmermann T, Zeller A, Weissgerber P, et al. Mining version histories to guide software changes [J]. IEEE Transactions on Software Engineering, 2005, 31(6): 429-445

[9] Shang Wei-yi, Jiang Zhen-ming, Adams B, et al. MapReduce as a general framework to support research in mining software repositories(MSR) [C]// 6th IEEE International Working Conference on Mining Software Repositories. 2009: 21-30

[10] Mockus A, Fielding R F, Herbsleb J. A case study of open source development: The apache server [C]// 22nd International Conference on Software Engineering. Limerick, Ireland, June 2000: 263-272

[11] Lim S L, Quercia D, Finkelstein A. StakeNet: Using Social Networks to Analyse the Stakeholders of Large-scale Software Projects[C]// 32nd International Conference on Software Engineering. Cape Town, South Africa, 2010: 285-294

[12] Ossher J, Bajracharya S, Lopes C. Automated Dependency Resolution for Open Source Software[C]// 7th IEEE International Working Conference on Mining Software Repositories. Cape Town, South Africa, 2010: 130-140