利用 POMDP模型来增强分布式系统的生存性

莉娜[1] 郑 宁[1] 孔 霆[2] 徐 明[1]
[1](杭州电子科技大学计算机学院 浙江 杭州 310018)
[2](浙江警察学院实验中心 浙江 杭州 310053)

摘 要 针对分布式系统存在的状态信息不完全问题,引入部分可观察的马尔可夫决策过程(POMDP)模型到生存控制系统中。在该控制系统的构造过程中,结合前瞻的思想,提出一种简易、有效的搜索算法(NSL算法)来作出决策,从而在一定程度上弥补了现有生存控制系统的不足,提高了分布式系统的可生存性。

关键词 POMDP 生存控制系统 NSL算法 可生存性

USING POMDP MODEL TO ENHANCE SURVIVABILITY OF DISTRIBUTE SYSTEM

Ding Lina Zheng Ning Kong Ting Xu Ming
[1] (Computer of Science, Hangzhou Dianzi University, Hangzhou 310018, Zhejiang, China)
[2] (Zhejiang Police College, Hangzhou 310053, Zhejiang, China)

Abstract In this paper it introduces partially observable Markov decision process (POMDP) model into survival control system in order to solve the problem of incomplete state information of the distributed system. Based on strategy of look-ahead, this paper put forward a facile and effective searching algorithm named NSL algorithm to make decision in the process of the control system construction, thus fetches up the insufficiency of the existing survival control system and enhances survivability of the distributed system to a certain extent.

Keywords POMDP Survival control system NSL algorithm Survivability

0 引 言

生存性是指即使系统或运行环境受到破坏,但系统仍然具有在这种运行环境下提供持续服务(或许是降级服务)的能力[1]。目前,生存性的研究可大致归为两种方法:一种是基于生存性设计的,即入侵掩盖。利用容错技术,使得即使系统的一部分受到攻击,新的生存性设计使系统仍能正常工作;另一种方法是研究入侵响应。不去重新设计系统,相反,给系统配备一组入侵容忍措施,在入侵检测器的帮助下能够对入侵有所反应。生存控制系统延用了第二种方法的思想,在现有分布式系统之上再施加一个控制系统,当分布式系统受到攻击时,生存控制系统能够执行恢复行动,使系统进入无错状态,保证用户服务的持续进行。

当前,大多数生存控制系统一般通过生成"if-then"规则库来处理恢复问题。Knight等学者提出的控制系统模型通过提取分布式系统状态信息来对分布式系统进行重新配置,并通过产生配置文件的不同决定不同的运行模式[2]。基于该方法的生存控制系统虽然在一定程度上提高了分布式系统的可生存性,但仍存在着不足之处,主要表现在:1)人工监控器提供的系统状态信息不精确[3]。2)规则库的复杂性以及规则之间不可预测的关联性影响着决策的质量。

针对上述问题,本文在现有生存控制系统的基础上,引入了POMDP模型[4,5],结合前瞻的思想,提出了基于有限阶段POMDP的NSL算法,保障了生存控制系统作出的决策质量,甚至当监控器提供的状态信息不精确的时候。

1 POMDP模型简介

POMDP首先由运筹学领域提出,后来被广泛应用到人工智能的机器人和增强学习领域。在POMDP模型中,用来决策的状态信息是不完全的,出于这一点,考虑将这个模型应用到生存控制系统,进行状态估计,来解决不明确的状态信息问题。这种不完全的状态信息被称为信念状态 $\pi = [\pi(1), \pi(2), \cdots, \pi(S)]$[6],表示系统处在状态集合中各状态的概率分布。POMDP可以看成是MDP,是因为它是基于信念状态的MDP;POMDP不可以看成是MDP,是因为信念状态是通过贝叶斯公式计算出来的,它是根据先验概率以及监视器观察结果的一种估计。

一个POMDP被定义为一个$(S, A, O, P(\cdot|\cdot,a), q(\cdot|\cdot, r(\cdot,a))$的六元组,其中:

S:有限的状态的集合;

A:行动的集合;

O:一个有限的观察 o的集合;

P代表转移概率函数,例如 $P(s'|s,a)$,其中 $s', s \in S, a \in A$,代表在状态 s的时选择行动 a转移到状态 s'的概率;

$q(o|s,a)$:表示任何情况下,系统通过行动 a到达状态 s,观察到 o的概率。论文假设与所选择的行动无关,也就是,$q(o|s$

收稿日期:2007-07-20。浙江省自然科学基金(Y106176);浙江省科技厅科技计划项目(2007C33058)。丁莉娜,硕士生,主研领域:信息安全。

$a) = q(o|s)$, $q(o|s)$代表系统在状态 s下观察到 o的概率;

$r(s,a)$: 报酬函数, 它代表在状态 s时, 采取行动 a获得的报酬(负的为代价, 论文考虑的都是代价)。

POMDP的值(最小累积代价)可以用函数 $V(\pi)$表示:

$$V(\pi) = \max_{a \in A}\{\pi \cdot r(a) + V(\pi^{\pi,a})\} \quad (1)$$

这里, $r(a) = [r(s,a), \forall s \in S]^T$, 代表选择行动 a的收益列向量; $\pi^{\pi,a}$代表下一个信念状态。

接收到一个观察将引起信念状态的更新。公式(2)是贝叶斯原理的直接应用, 当 o正确观察, 目标系统处于 s状态的概率等于在状态 s下观察到 o的概率占所有状态下观察到 o的概率的比例:

$$\pi^{o}(s) = \frac{\pi(s)q(o|s)}{\sum_{s' \in S}\pi(s')q(o|s')} \quad (2)$$

执行一个行动也将引起信念状态的更新:

$$\pi^{a}(s) = \sum_{s' \in S}\pi(s')p(s|s',a) \quad (3)$$

因此, 给定当前的信念状态π, 选择行动 a, 和观察结果 o, 下一个信念状态可以用公式(4)来计算:

$$\pi^{\pi,a,o}(s) = \frac{q(o|s)\sum_{s' \in S}p(s|s',a)\pi(s')}{\sum_{s' \in S}q(o|s')\sum_{s'' \in S}p(s'|s'',a)\pi(s'')} \quad (4)$$

假设当前信念状态为π, 采取行动 a, 则观察到 o的概率:

$$\gamma^{\pi,a}(o) = \sum_{s \in S}q(o|s)\sum_{s \in S}p(s|s',a)\pi(s') \quad (5)$$

综合公式(4)和(5), 可得:

$$\pi^{\pi,a}(s) = \frac{\sum_{o \in O}\gamma^{\pi,a}(o)\pi^{\pi,a,o}(s)}{\sum_{o \in O}\gamma^{\pi,a}(o)} \quad (6)$$

2 基于 POMDP的生存控制系统

2.1 生存控制系统的组成结构

一个基于 POMDP的生存控制系统(其结构如图 1所示)由以下部分组成:

- 监视器

这里的监视器实质上是一个监视系统, 它由分布在分布式系统中的许许多多的监视控件 M_1, M_2, …, M_N组成, 这些监视部件不断地侦查其控制范围内的异常信息。一旦发现异常, 马上触发监视器输出相应的观察 $\{o_M\}$。

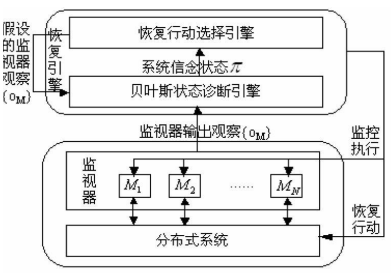

图 1 基于 POMDP的生存控制系统的结构

- 恢复引擎

1)贝叶斯状态诊断引擎: 把监视器输出的观察信息或者恢复行动选择引擎假设的观察信息映射成为系统的信念状态。

2)恢复行动选择引擎: 利用贝叶斯状态诊断引擎提供的系统信念状态, 来选择累积(N步)代价最小的行动。另外恢复行动选择引擎还向贝叶斯状态诊断引擎提供假设的监视器观察 $\{o_M\}$, 并请求其给出相应的系统信念状态。

恢复引擎主要由贝叶斯状态诊断引擎和恢复行动选择引擎构成。这两个引擎相互协调工作不可分离。此外恢复引擎还有其它的一些功能, 如初始化系统信念状态, 请求额外的监控执行, 执行恢复行动选择引擎选择的行动。

2.2 生存控制系统的工作过程

在生存控制系统运行过程中, 当监视器发现异常的时候, 向恢复引擎报告观察结果 o_M, 并触发其工作。恢复引擎构造初始的π, 将π和步数 N作为参数开始调用 NSL算法, 通过一系列的"状态-行动"循环步骤来选择合适的恢复行动。最后, 恢复引擎根据 NSL算法中 Flag的返回值来确定下一步的行动: 1)返回值为 success恢复完成; 2)返回值为 over恢复没有完成, 恢复引擎将执行额外的监控请求, 进一步触发恢复引擎进行工作。

3 基于有限阶段 POMDP的 NSL算法

3.1 算法思想

在现实应用中, 考虑无限阶段的最小代价是复杂并且是不必要的, 因此论文提出一种基于有限阶段 POMDP的 NSL算法。N是需要恢复的最大深度, NSL算法的目标是选择一个行动序列 a_1, a_2, …, a_N使得这 N步的代价最小。该算法并不要求执行完一次 NSL算法就一定能使系统达到π_ϕ(正常状态), 即使系统没有达到π_ϕ, 也可以通过恢复引擎向监视器发出监视请求, 根据监视器给出新的观察重新调用 NSL算法, 直到完成恢复为止。

理解 NSL算法的一个最简单的方法是图 2阐述的"N步向前看"中的一步的例子。对于每一个可能的恢复行动, 算法利用公式(6)计算系统下一个信念状态。

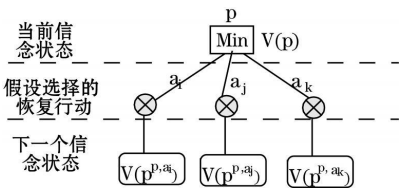

图 2 N-Step Look-ahead中的一步

3.2 算法描述

```
NSL(π, N, Input Flag, Output)
1 if P[π = πφ] ≥ 1 − ε then Flag ← success; return 0;
2 if N = 0 then Flag ← over; return 0;
3 foreach a ∈ A do
5 V_next(a) ← NSL(π^{π,a}, N − 1, Flag);
7 Action ← Push(argmin_{a∈PosAction}(V_next(a) + π · r(a)));
8 Flag ← continue; return min(V_next(a) + π · r(a));
9 end foreach
```

NSL算法首先将当前的信念状态(通过观察结果和先验概率计算得到)作为起始信念状态。

1)如果信念状态接近π_ϕ的概率($P[\pi = \pi_\phi]$)接近 1, 算法认为恢复成功;

2)如果算法利用公式(1)扩展递归已经到达了深度 N但仍没有到达π_ϕ, 结束本次算法的执行。接下来通过恢复引擎向监视器发出监视请求(根据 Flag的值), 重新调用该算法;

3)如果不是以上情况, 则继续利用公式(1)进行递归。

(下转第 99页)

异常操作序列 4	异常	异常	异常
异常操作序列 5	正常	正常	异常

4 结　论

本文设计并实现了一个基于隐 Markov模型的数据库异常检测系统。该系统检测方式灵活,可以实现离线异常检测与在线异常检测,其中在线异常检测采用了 VC++ 6.0的多线程技术。离线异常检测也非常灵活,可以任意选择某一个数据库用户,对其进行异常检测,还可以灵活地选择滑动窗口的长度。

通过使用滑动窗口的宽度 K分别取值为 3、5和 8时进行实验,从试验的结果中可以看出,本文实现的系统能够检测出数据库用户的异常操作,但如何选取滑动窗口的宽度还有待进一步研究。

参 考 文 献

[1] Dorothye E Denning, Peter J Denning. Data Security[J]. Computing Surveys, 1979, 11(3).

[2] Castano S, Fugini M G, et al. Database Security[M]. Addison Wesley, 1994.

[3] Anderson J P. Computer security threat monitoring[R]. Technical Report TR80904. Washington: Anderson Co, 1980.

[4] Denning D E. An intrusion detection model[J]. IEEE Transactions on Software Engineering, 1987, 13(2): 222-232.

[5] Axelsson S. Intrusion Detection Systems: A Survey and Taxonomy. Dept of Computer Engineering, Chalmers University of Technology, Goteborg, Sweden[Technical Report 99-15], 2000.

[6] Rabiner L R. A tutorial on hidden markov models and selected applications in speech recognition[J]. Proceedings of the IEEE, 1989, 77(2): 257-286.

[7] Chung C Y, Gertz M, Levitt K. DEMIDS: A Misuse Detection System for Database Systems. In: The Thirteen Annual IFIP TC-11 WG 11.5 Working Conf. On Integrity and Internal Control in Information Systems, 1999.

[8] 阳国贵,满家巨. ORACLE数据库管理与使用教程[M]. 长沙:国防科技大学出版社, 1998.

[9] 郑阿奇. Oracle实用教程[M]. 北京:电子工业出版社. 2006-12.

[10] 吕亮. 数据库入侵检测[D]. 武汉大学, 2003.

[11] 张超. SQL Server数据库入侵检测系统的研究[D]. 西安电子科技大学, 2004.

[12] 詹伟. 关系数据库入侵检测系统的设计与实现[D]. 华中科技大学, 2004.

[13] 王丽娜,董晓梅,郭晓淳,等. 基于数据挖掘的网络数据库入侵检测系统. 东北大学学报, 2003, 24(3): 225-228.

[14] 邝祝芳,阳国贵,李清,等. 基于隐马尔可夫模型的数据库异常检测技术[J]. 计算机研究与发展, 2006, 43(11).

(上接第 92页)

4 实验结果

用 10台微机组成小型网络系统进行模拟测试。对外提供 5种服务:FTP、TELNET、RPC、SOCKET、DNS。实验通过向该网络系统发起 10000次攻击来评估生存控制系统做出的决策的质量。表 1给出了算法的详细测试结果。代价指从一次攻击恢复过来所消耗资源的平均值;算法时间指执行一次算法的平均时间;行动指恢复一次攻击所选择的行动的平均个数;监视请求指恢复一次攻击而执行的监视的平均次数。

表 1　发起攻击的测试结果(每次攻击的平均值)

算法	N	代价	算法时间(s)	行动	监视请求
一般	—	199.79	0.09	15.42	15.42
NSL	1	199.61	0.11	15.1	15.1
NSL	2	130.5	0.29	12.24	6.12
NSL	3	108.56	0.73	11.09	4.01

从表 1可观察到,随着 N值的增大,平均从一次攻击恢复过来有以下规律:1)所需要的代价、行动的个数、监视请求次数的值均呈现递减的趋势;2)算法时间呈现递增的趋势。总之,相比一般的匹配规则库来处理恢复问题,NSL算法在一定程度上提高了分布式系统恢复的准确性和有效性。给定一个较小的 N值,NSL算法对于一个现实的分布式系统的恢复已经是足够快的。

图 3显示当向该网络系统发起攻击时,平均无故障时间的异常概率函数。可以看出随着 N值的增大,系统的异常概率呈现递减的趋势。

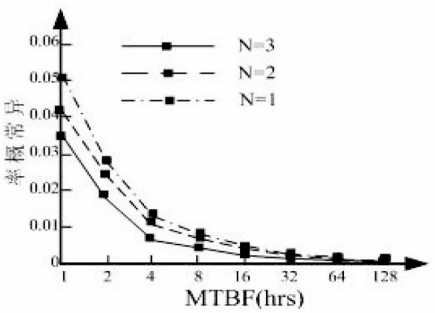

图 3　异常概率函数图像

5 结　论

分布式的可生存性是一个非常复杂并且值得研究的课题。本文在借鉴现有生存控制系统的基础上,引入了 POMDP技术,保障了生存控制系统作出决策的质量,在一定程度上提高了分布式系的可生存性。本文在以下方面还存在着改进的空间:当给定一个较大的 N值时,引入高效的搜索技术,加快 NSL算法的速度,提高该控制系统的响应速度与处理能力。

参 考 文 献

[1] Knight J C, Sullivan K J. On the definition of survivability[EB/OL]. http://www.cs.virginia.edu/~jck/recentpapers.htm, 2000.

[2] Knight J C, et al. Survivability Architectures: Issues and Approaches[C]. Hilton Head, South Carolina, 2000.

[3] Fischer M, Lynch, Paterson M. Impossibility of distributed consensus with one faulty process[J]. Journal of ACM, 1985, 32(2): 374-382.

[4] Anthony R C, Leslie P, Michael L. Acting optimally in partially observable stochastic domains[C]. Seattle, WA, 1994.

[5] Cassandra A, Littman M L, Zhang G N L, et al. Incremental pruning: A simple, fast, exact method for partially observable markov decision processes[C]. Rhode Island, 1997.

[6] Nair R, Tambe M, Yokoo M, et al. Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings[C]. Mexico: Morgan Kaufmann Press, 2003.