计算机研究与发展
Journal of Computer Research and Development
ISSN 1000-1239/CN 11-1777/TP
42(2):315~321, 2005

基于软件容错的动态实时调度算法

韩建军[1] 李庆华[1] Abbas A.Essa[2]
[1](华中科技大学计算机科学与技术学院 武汉 430074)
[2](南京大学电子工程与科学系 南京 210093)
(han_j_j@163.com)

A Dynamic Real-Time Scheduling Algorithm with Software Fault-Tolerance

Han Jianjun[1], Li Qinghua[1], and Abbas A.Essa[2]
[1](*School of Computer Science and Technology, Huazhong University of Science & Technology, Wuhan* 430074)
[2](*Department of Electronic Engineering and Science, Nanjing University, Nanjing* 210093)

Abstract A hard real-time system is usually subject to stringent reliability and timing constraints due to the fact that failure to produce correct results in a timely manner may lead to a disaster. Almost all fault-tolerant scheduling algorithms at present are designed to deal with hardware faults, while less of those take possible software faults into account. Presented in this paper is a new software fault-tolerant real-time scheduling algorithm that is similar to EDF, called EBPA (expectation-based probing algorithm). The important contributions of the algorithm are probing a certain steps during the executions of primaries, which leads to improving the predictive quality of canceling ineffective primaries when heavy workload occurs and preventing early failures in execution from triggering failures in the subsequent primary executions as soon as possible. Therefore, the algorithm increases the successful percentage of tasks' completion, and meanwhile decreases the wasted CPU time slots. The simulation experiments show that the algorithm has a better trade-offs between scheduling costs and scheduling performance than the well-known algorithms so far. Moreover, some experimental parameters, such as the number of probing steps and failure probability, are also taken into account.

Key words real-time system; software fault-tolerance; dynamic scheduling algorithm

摘 要 在硬实时系统中,由于任务超时完成将会导致灾难性后果,因而硬实时系统具有严格的时间及可靠性限制条件.目前实时容错调度算法大部分针对硬件的容错,很少考虑软件运行的故障.提出了一种类似 EDF 基于软件容错的动态实时调度算法 EBPA(expectation-based probing algorithm),该算法在任务执行过程中通过基于期望值的若干试探性检测步骤,提高了任务可执行性的预测,尽可能避免了任务早期的错误对后续任务的影响,因此提高了任务的完成率并同时有效地减少了浪费的 CPU 时间片.通过实验测试,同目前所知的同类算法相比,具有更佳的调度性能-调度成本比.

关键词 实时系统;软件容错;动态调度算法

中图法分类号 TP316

1 引 言

目前实时系统在经济、军事、科学等多个领域中起着重要的作用.对及时性及可靠性的要求不同,实时系统可以分为硬实时系统和软实时系统.若任务未能及时产生正确结果将导致灾难性后果,则该实时系统称为硬实时系统,反之,则称为软实时系

收稿日期:2003-09-01;修回日期:2004-07-19
基金项目:国家自然科学基金项目(60273075)

统.因此容错调度算法对于硬实时系统的有效性及可靠性至关重要. 目前,硬实时系统中容错调度算法的计算模型一般有以下3种:①硬件容错模型.该模型中任务包含基版本和副版本. 首先调度基版本,当基版本出错时,再在不同的处理机上调度副版本,一般采用静态调度方法[1~3]. ②非精确计算(imprecise computation)模型. 每个任务分为强制性子任务(mandatory subtask)和选择性子任务(optional subtask),该模型调度算法在保证强制性子任务完成的同时,尽可能多地完成选择性子任务以提高计算精确性[4~6]. ③软件容错模型. 该模型中任务分为主部分(primary)及替代部分(alternate). 其中主部分计算时间较长且计算结果较为精确,但不保证程序完全正确地运行,而替代部分计算时间较短且计算结果仅保证可接受的精度要求,但能保证正确地运行. 主部分与替代部分并不要求调度在不同处理机上,且两者只要完成一个即可. 该类模型主要针对软件错误(如程序bug),提供软件容错的可靠性[7]. 在目前各个应用领域中,不仅要求实时系统对硬件具有良好的容错性,同时也要具备软件容错的功能,以充分满足硬实时系统对及时性及可靠性的需求. 在当前硬实时系统中,容错调度算法大多基于前两种模型.

本文提出了一种新的类似EDF(earliest deadline first)的软件容错的动态实时调度算法EBPA(expectation-based probing algorithm). 该算法以容错截止期作为任务优先权值,来分配替代部分占用的时间间隔,在任务主部分执行过程中,通过基于期望值的若干试探性检测步骤,提高了该任务可执行性的预测,尽可能避免了任务早期的错误对后续任务的影响,因此提高了任务的完成率,并同时有效地减少了浪费的CPU时间片.

由于目前实时容错调度算法大部分针对硬件的容错,很少考虑软件运行的故障,因此针对软件容错的动态实时调度算法极少有人研究. 而文献[7]中的BCE算法作为作者目前所知的最新、最权威且已公开发表的同类算法,率先提出基于周期性任务的软件容错计算模型,并针对以前发表的类似算法[8]的缺陷,给出了较好的启发式策略(在第2节中详细叙述),以降低算法的运行成本并同时提高算法的调度性能,因此本文选用BCE算法作为参考算法,以验证本算法的有效性. 实验测试数据表明,性能指标不论是任务完成率还是浪费的CPU时间片,EBPA都明显地优于BCE,证明了本文提出的EBPA算法的先进性.

本文第2节介绍计算模型;第3节描述文献[7]中的BCE算法;在第4节详细介绍EBPA算法;第5节给出模拟测试结果及结果分析,最后一节总结全文.

2 计算模型

硬实时系统包含一个周期性任务集合:$T=\{T_i\}$,$1\leqslant i\leqslant N$,其中N为任务总数. 每个任务T_i的周期为PE_i,软件错误的概率为FP_i,且包含主部分P_i及替代部分A_i. P_i的执行时间为p_i,A_i的执行时间为a_i,且$a_i<p_i$. 任务i的每个实例j($T_{i,j}$)包含两个部分:主部分$P_{i,j}$及替代部分$A_{i,j}$. $T_{i,j}$的到达时间$r_{i,j}$为$(j-1)\cdot PE_i$,截止期$d_{i,j}$为$j\cdot PE_i$. 每个$A_{i,j}$有一个通知时间$NT_{i,j}$(不同的调度算法计算出不同的通知时间)表示$A_{i,j}$开始执行时间,在其之前若$P_{i,j}$已完成则撤消$A_{i,j}$,否则若$P_{i,j}$未及时完成或因错误而中止,则在$NT_{i,j}$开始执行$A_{i,j}$. 计划周期PC(planning cycle)为所有N个任务周期的最小公倍数,即$LCM(PE_1,PE_2,\cdots,PE_N)$. 在一个计划周期内$T_i$执行的次数($N_i$)为$PC/PE_i$.

计算模型中任务具有可抢占性,且仅考虑单机系统. 假定所有任务的替代部分均可调度(即均可在其截止期内完成). 不失一般性,本计算模型仅考虑一个计划周期内的任务调度,每个任务的每个实例在其截止期内必须完成其主部分或替代部分. 由于主部分提供了更好的计算质量,因此该模型调度算法设法执行更多的主部分,但是一旦主部分失败,必须保证替代部分在截止期内完成以获得可以接受的计算结果. 因此,调度算法力求在任务每个实例主部分或替代部分及时完成的前提下,尽可能多地完成主部分,以获得更高的计算质量.

3 BCE算法

在文献[7]中提出了一种基于速率单调(rate-monotonic)[9]的算法. 该算法包括3个部分:基本算法(basic algorithm)、可用时间检测算法(CAT)、消除空闲时间算法(EIT).

基本算法按照任务周期值的非降顺序,确定每个任务的优先权,周期短的任务具有较高的优先权值. 按照优先权顺序,对每个任务实例的替代部分,采用向后速率单调算法(对任务T_i首先调度A_{i,N_i},

最后调度 $A_{i,1}$)分配替代部分占用的时间间隔 $[ST_p(i,j),ED_p(i,j)]$(其中 i 为任务编号,j 为实例编号,p 为第 p 个时间间隔),使得替代部分在其截止期内尽可能地推迟执行,即具有最大的 $NT_{i,j}$. 由于计算模型中任务的可抢占性,因此每个替代部分所占用的时间片并不保证连续性. 每次执行主部分时,选择已到达但未完成的主部分中优先权值最大的主部分,即周期短的任务具有较高的执行优先权.任务实例的替代部分在任务执行时具有最高优先权,即在任一时间 t,若 $\exists i,j,p$ 使得 $t\in[ST_p(i,j),ED_p(i,j)]$,则中断当前执行的主部分,转而执行 $A_{i,j}$. 若 $P_{i,j}$在时间 t 完成,则释放 $A_{i,j}$所占用时间间隔,并调整相应受影响的其他任务替代部分所占用的时间间隔,计算新的通知时间.

设当前时间为 t,准备执行的主部分为 $P_{i,j}$,t 到 $NT_{i,j}$之间分配给替代部分的时间间隔所占时间的总和为 $I=\sum_m\sum_n\sum_p(ED_p(m,n)-ST_p(m,n))$,其中 $t\leqslant ST_p(m,n)<ED_p(m,n)\leqslant NT_{i,j}$,则 $P_{i,j}$可用时间 $AT_{i,j}=NT_{i,j}-t-I$. 可用时间检测算法在任一时刻,若当前执行的是主部分时,检测从该时刻至该实例通知时间之间的空闲时间是否足以完成主部分剩余执行时间,若满足则执行该主部分,否则取消该主部分的执行.

若在当前时间没有可运行的主部分或替代部分,则消除空闲时间算法执行主任务已失败而替代部分未完成且具有最小通知时间的替代部分,并修改该替代部分的通知时间,以充分利用系统资源,并为后续主部分留出更多时间.

4 EBPA 算法

本文提出一种新的类似 EDF[7] 的容错算法. 本算法中截止期与传统算法的截止期不同,指的是 PE_i 与 a_i 之间的差值,称为容错截止期.

算法首先按照容错截止期的非降顺序确定每个任务的优先权,容错截止期较小的任务具有较高的优先权. 按照优先权顺序,并按类似 BCE 方法向后分配替代部分占用的时间间隔,并确定通知时间. 每次执行主部分时,选择具有最小通知时间的主部分作为待执行主部分.

引理 1. 若在时间 t'待执行主部分为 $P_{i,j}$,在下一时间 t 没有新的主部分到达且 $P_{i,j}$未完成,$t\notin$ 任一$[ST_p(m,n),ED_p(m,n)]$,则在时间 t 仍然执行 $P_{i,j}$.

证明. 按照算法从未完成主部分中选择通知时间最近的任务,由于时刻 t 没有新的主部分到达且不需执行任一替代部分,在上一时刻 t',$P_{i,j}$具有最近的通知时间且未完成,因此在时刻 t,$P_{i,j}$具有最近的通知时间,按照算法仍然选择 $P_{i,j}$执行. 证毕.

从引理 1 看出每次选择主部分执行时,并不需要从未完成的主部分中重新选择通知时间最近的任务,因而可以节省调度成本.

引理 2. 若在时间 t 选择执行主部分 $P_{i,j}$,$\exists m$,n 使得 $t<NT_{m,n}<NT_{i,j}$,且 $P_{m,n}$在时间 t 未完成,则 $r_{m,n}>r_{i,j}$.

证明. 假设 $t<NT_{m,n}<NT_{i,j}$,且 $P_{m,n}$在时间 t 未完成,若 $r_{m,n}\leqslant r_{i,j}$,则按照算法在时间 t 时,由于 $P_{m,n}$的通知时间小于 $P_{i,j}$的通知时间,因此在时间 t 应选择 $P_{m,n}$执行,与假设矛盾. 证毕.

引理 2 指出了在时间 t 对于任一待执行主部分 $P_{i,j}$在其执行时,由于其他任务主部分具有更高优先权而被抢占,但是抢占的主部分的到达时间都大于 t,因此对 $P_{i,j}$在执行期间产生影响的主部分执行时间均在 t 到 $NT_{i,j}$之间. 而 BCE 由于选择以任务周期为优先权值的方法,在主部分执行期间,对其产生影响的主部分的执行时间区间不具备这个特点.

定理 1. 若对于任一任务 i 有 $a_i<p_i$,则在时间 t 对于任一待执行主部分 $P_{i,j}$,在 $NT_{i,j}$之前可以执行的时间不大于可用时间 $NT_{i,j}$(同第 2 节定义).

证明. 由引理 2,对 $P_{i,j}$执行产生影响的主部分的执行时间区间在 t 到 $NT_{i,j}$之间,因此根据计算模型的要求,这些任务 m 执行时间至少为 $\sum_m\min(a_m,p_m)=\sum_m a_m$,即至少为 t 到 $NT_{i,j}$之间这些任务替代部分时间间隔所占时间的总和;对于区间中主部分已失败的替代部分执行时间为区间内替代部分时间间隔的总和. 根据第 2 节中 $AT_{i,j}$的定义,定理得证.

根据定理 1,可以计算待执行主部分的可用时间,但此上界较为宽松. 而由引理 2,在时间 t 对于任一待执行主部分 $P_{i,j}$的执行产生影响的主部分的执行时间区间在 t 到 $NT_{i,j}$之间,因此可以对这些主部分试探地性计算其占用时间,提高 $P_{i,j}$可执行性的预测精度. 在处理机负载较重的情况下,提前撤消待执行主部分,可以留出更多时间以供其他任务执行,同时降低了浪费的 CPU 时间片数量. 因此本

算法对于待执行主部分可以执行 K 次基于期望值的试探性计算. 对任一 $P_{i,j}$ 其执行时间的期望值为 $p_i\cdot(1-FP_i)$,$A_{i,j}$ 执行时间的期望值为 $a_i\cdot FP_i$. 算法过程如下:

步骤 1. 设置试探步数 K 及一临时时间间隔链表. 初始时令 $k=0$. 在时间 t,对任一待执行主部分 $P_{i,j}$,根据定理 1 判断临时的时间间隔链表是否有足够的空闲时间,若不满足,则撤消该主部分,否则转向步骤 2,从时间 t 到 $NT_{i,j}$,依次判断占有该时间间隔的任务.

步骤 2.

步骤 2.1. 若占有该间隔的任务的主部分已失败,则将其时间间隔添加至临时时间间隔链表.

步骤 2.2. 若该任务未开始,$k=k+1$,则根据定理 1 判断临时时间间隔链表是否有足够的空闲时间,

步骤 2.2.1. 若满足,则将该任务主部分按其执行时间期望值的时间间隔,及相应替代部分执行时间期望值的时间间隔分别添加到临时时间间隔链表.

步骤 2.2.2. 若不满足,则同步骤 2.1 处理.

同时在上述两种情况下计算目前已占用的时间总和 CT.

步骤 3. 若还有时间间隔未考虑且 $k<K$,则执行步骤 2.

步骤 4. 若仍有时间间隔未考虑,则按步骤 2 的第 1 种情况处理.

步骤 5. 若 $P_{i,j}$ 剩余计算时间大于 $NT_{i,j}-t-CT$,则撤消 $P_{i,j}$,否则执行 $P_{i,j}$.

EBPA 算法描述如下:

步骤 1. 按照任务优先权值,分配替代任务占用的时间间隔.

步骤 2. 在时间 t,①将到达时间为 t 的主部分添加到候选执行队列.②若有替代部分到达,则执行替代部分并转向步骤 4.③若前一时刻待执行的主部分 $P_{i,j}$ 在时间 t 完成任务,则从候选执行队列中删除 $P_{i,j}$,撤消相应的替代部分 $A_{i,j}$ 占用的时间间隔,并调整受影响的替代部分的时间间隔(令 p 为 $A_{i,j}$ 最后的时间间隔,从 $ED_p(i,j)$ 至 t 按步骤 1 方法重新调整该区间内的时间间隔).④若前一时刻待执行的主部分 $P_{i,j}$ 在时间 t 未完成任务,且未有新的主部分到达,则转向步骤 2.1. ③若有新的主部分到达,则执行步骤 2.1.

步骤 2.1. 若待执行主部分为空,则根据通知时间的大小,从非空候选执行队列中选择新的待执行主部分,对待执行主部分进行错误概率测试,若失败,则从候选队列中撤消该主部分并重复步骤 2.1.

步骤 2.2. 若待执行主部分为未开始执行的主部分,则执行 K 次基于期望值的试探性算法. 若失败,则从候选队列中撤消该主部分并重复步骤 2.1.

步骤 2.3. 若待执行主部分为已开始执行的主部分,则根据定理 1 判断其可执行性. 若失败,则从候选队列中撤消该主部分并重复步骤 2.1.

步骤 3. 若待执行主部分为空,则按照第 2 节 EIT 算法处理.

步骤 4. 若仍有任务未完成,则设置下一时间 t,转向步骤 2.

从算法中可以看到本算法与传统的 EDF 的区别,传统的 EDF 算法在上述步骤 2 的情况④时,要求重新选择任务,而本算法仅在有新主部分到达或者容错测试失败时,才重新选择,因此可以降低调度成本. 同时,对已开始执行的主部分没有使用 K 次基于期望值的试探性计算,同样也是为了降低调度成本.

5 模拟测试及结果分析

测试中随机选择 20~80 个任务,由于一般情况下求得的计划周期数值很大,因此,测试中令计划周期为 100000. 为使测试更加全面,考虑了不同任务粒度,任务周期分别为 [100,300],[500,800],[1000,12000],主部分在每个周期区间的计算时间分别为 [20,100],[80,200],[100,600],替代部分计算时间分别为主部分计算时间的 0.4~0.8 倍. 随机选择周期区间及主部分计算时间区间以构成不同的任务粒度. 任务软件错误概率分别为 [0,0.4]. 处理机利用率($\sum_{i=1}^{N}\frac{p_i}{PE_i}$)选择 0.8~1.8 的区间. 测试中的随机数均采用均匀分布. 该模拟程序采用 GNU C,并在 Linux OS2.4.18,256MB RAM,1.7GHz Intel CPU 的单机上模拟测试.

5.1 EBPA 和 BCE 算法的性能测试

在测试中,性能指标采用任务丢失率(%)及浪费的 CPU 时间片的数量. 任务丢失率是未完成主部分(不包含因错误而失败的主部分)的计算时间总和与所有有效主部分(没有错误而正常执行的主部分)计算时间的比例,反映了调度的质量. 浪费的 CPU 时间片的数量是因到达通知时间而主部分被中断所浪费的已执行的 CPU 时间片数量,反映调度算法中撤消主部分预测的准确性. 本次测试的数据为不同错误概率情况下的平均值. 本次测试中 EBPA 算法在

利用率为 0.8～1.1 时,K 取值为 0;1.2～1.5 时取值为 4;1.6～1.8 时取值为 8(其原因在下一测试中阐述). 为显公平性,算法的调度成本(时间)已包含在调度性能中. 图 1 显示 TLP,以百分比表示,横轴代表利用率 0.8～1.8. 图 2 显示浪费的 CPU 时间片的数量. 从实验结果表明,EBPA 无论在任务丢失率还是在浪费的 CPU 时间片的数量上,较 BCE 均取得了明显的改善. 在利用率为1.1～1.8 时,任务丢失率平均降低了 5.59%,即平均多执行了 5600 个任务单位,浪费的 CPU 时间片数量平均降低了 5937 个任务单位. 因而 EBPA 比 BCE 具有更好的性能-成本的均衡性.

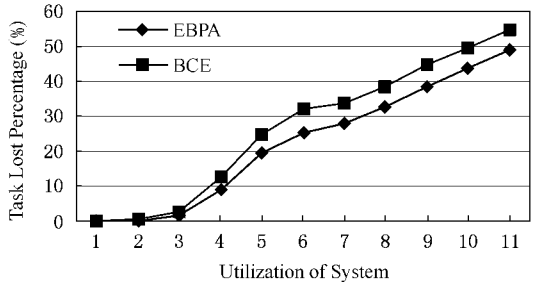

Fig. 1 Comparison of TLP between EBPA and BCE.

图 1 任务丢失率的比较

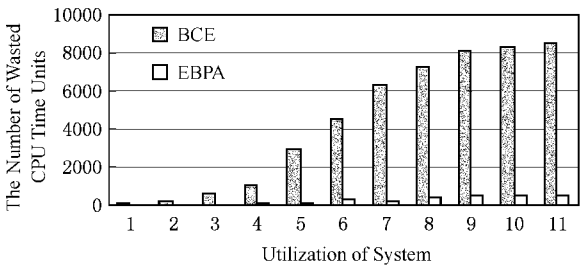

Fig. 2 Comparison of the wasted CPU time units.

图 2 浪费的 CPU 时间片数量的比较

5.2 K 值对 EBPA 的影响

本次测试是在错误概率为 0～0.4 时,测试 K 值对 EBPA 算法性能的影响. 本次测试的数据为不同错误概率情况下的平均值. 从图 3 及图 4 中可以看出,对于利用率在 0.8～1.1 时,任务间拥挤度较低,K 值对算法影响较小,因此 K 取 0 即可,且可以降低计算成本. 随着利用率的提高,负载增大,任务主部分执行期间对其产生影响的主部分的数量逐渐增加,因而 K 值增加有利于提高调度性能,但是 K 值到了一定限度以后,性能指标反而出现略有下降趋势,其原因在于:①试探步骤并不是对执行情况的精确反映,因为试探步骤中当主部分完成并撤消替代部分时,并未调整受影响的替代部分时间间隔,这是出于降低调度成本的考虑. 另外期望值不能完全精确地反映任务的执行时间. 随着利用率的提高,由于影响待执行主部分的主部分数量的增加,及错误概率的存在,预测的不精确度就越高. ②对于负载较重的处理机,适当的 K 值足以判断待执行主部分的可执行性. 因此 K 值过大不仅提高了调度成本,同时也加大了预测步骤的不精确性.

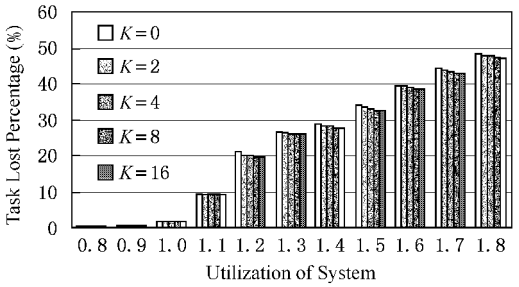

Fig. 3 Influence of different K on TLP.

图 3 不同 K 值对 TLP 的影响

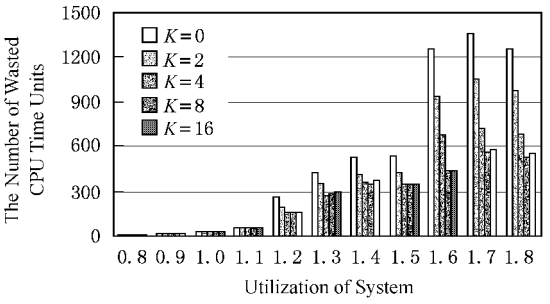

Fig. 4 Influence of different K on NWTS.

图 4 不同 K 值对 NWTS 的影响

5.3 错误概率对 EBPA 及 BCE 的影响

本次测试中,K 值按测试 1 选择. 图 5 及图 6 显示利用率在 0.8～1.8 的情况. 一般情况下,随着错误概率的提高,任务失败率会随之下降,原因在于有可能因到达通知时间而撤消的主部分因错误而撤消. 另一方面,某些主部分的撤消会给其他主部分留出更多执行时间. 在利用率大于 1.6 时,任务失败率的下降幅度明显减小,其原因在于过重的机器负载情况下,机器利用率的影响起着主导作用. 同时随着错误概率的提高,浪费的 CPU 时间片的数量反而有所增加. 这是由于随着错误概率的提高,试探性算法的不完全精确性的误差也会提高. 令 $TLP_{FP=x}$ 为算法 A 在错误概率为 x 时的 TLP. 图 7 显示 EBPA 与 BCE 以错误概率为 0 为基准,不同错误概率下任务失败率的降低幅度(相对幅度)的差值,即 $\frac{TLP_{FP=0}(EBPA)-TLP_{FP=x}(EBPA)}{TLP_{FP=0}(EBPA)}$ –

$\frac{TLP_{FP=0}(BCE)-TLP_{FP=x}(BCE)}{TLP_{FP=0}(BCE)}$,其中,$x=0.1$,0.2,0.3,0.4. 图7仅显示利用率在1.2～1.8的情况. 实验数据表明,利用率保持不变时,随着错误概率的提高,EBPA任务失败率降低的幅度高于BCE,显示出EBPA对错误概率更好的适应性,同时,降低的幅度的差值逐渐减小,表明在较大的错误概率下,任务调度的随机性增强,不同算法中,正常运行的任务的可调度性都逐渐地提高. 当错误概率保持不变时,随着处理机利用率提高,EBPA任务失败率的降低幅度比BCE更大,显示出在较重的机器负载下,EBPA仍然显示较佳的调度性能.

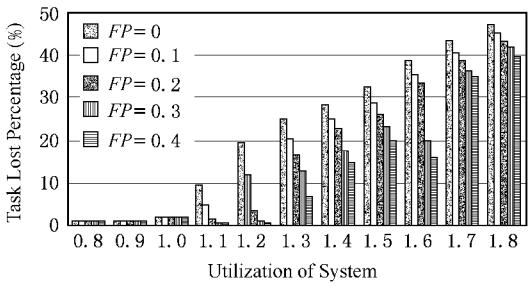

Fig. 5 Influence of fault probability on TLP of EBPA.

图5 不同错误概率对EBPA任务失败率的影响

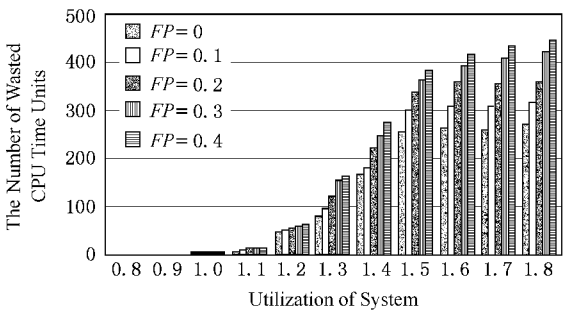

Fig. 6 Influence of fault probability on NWTS of EBPA.

图6 不同错误概率对EBPA浪费的CPU时间片的数量的影响

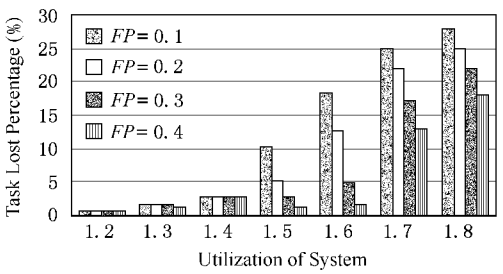

Fig. 7 Comparison of adaptability to fault probability between EBPA and BCE.

图7 两种算法对错误概率的适应性

6 总 结

本文针对实时系统中软件容错模型提出了一种新的类似EDF的算法,并在任务执行过程中通过若干基于期望值的试探性检测步骤,提高了任务可执行性的预测,尽可能避免了任务早期的错误对后续任务的影响,因此提高了任务的完成率并同时有效地减少了浪费的CPU时间片. 通过实验测试,同目前所知的同类算法相比,具有更佳的调度性能-调度均衡性. 同时也阐述了K值的选择及错误概率对调度性能的影响.

参 考 文 献

1 Qin Xiao, Han Zongfen, Pang Liping. Real-time scheduling with fault-tolerance in heterogeneous distributed systems. Chinese Journal of Computers, 2002, 25(1): 49～56 (in Chinese)
(秦啸,韩宗芬,庞丽萍. 基于异构分布式系统的实时容错调度算法. 计算机学报, 2002, 25(1): 49～56)

2 Sunondo Ghosh, *et al*.. Fault-tolerant rate-monotonic scheduling. Journal of Real-Time Systems, 1998, 15(2): 48～60

3 H. Aydin, *et al*.. Tolerating faults while maximizing reward. In: Proc. of the 12th Euromicro Conf. on Real-Time Systems, Stockholm, Sweden, 2000

4 A. Garvey, V. Lesser. Representing and scheduling satisfactory tasks. In: Imprecise and Approximate Computation. Dordrecht: Kluwer Academic Publishers, 1995

5 J. Yen, S. Natarajan. A decision-theoretic treatment of imprecise computation. In: Imprecise and Approximate Computation. Dordrecht: Kluwer Academic Publishers, 1995

6 Charlie McElhone, Alan Burns. Scheduling optional computations for adaptive real-time systems. Journal of Systems Architechture, 2000, 46: 49～77

7 Ching-Chih Han, Kang G.Shin, Jian Wu. A fault-tolerant scheduling algorithm for real-time periodic tasks with possible software faults. IEEE Trans. on Computers, 2003, 52(3): 362～372

8 H. Chetto, M. Chetto. Some results of the earliest deadline scheduling algorithm. IEEE Trans. on Software Eng., 1989, 10(15): 1261～1269

9 C. L. Liu, J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. J. ACM, 1973, 1(20): 46～61

Han Jianjun, born in 1972. Now he is a doctor candidate in the School of Computer Science and Technology, Huazhong University of Science and Technology. His current research interests lie in the area of parallel and distributed computing, scheduling algorithms, etc.

韩建军,1972年生,博士研究生,主要研究方向为实时系统调度算法、并行分布式计算.

Li Qinghua, born in 1940. Now he is professor and doctor supervisor in the School of Computer Science and Technology, Huazhong University of Science and Technology. His current research interests lie in the area of parallel algorithms, scheduling algorithms, etc.

李庆华,1940 年生,教授,博士生导师,主要研究方向为实时系统、并行算法、入侵检测.

Abbas A. Essa, born in 1966. Now he has completed his post-doctor training in the Department of Electronic Engineering and Science, Nanjing University. His current research interests lie in the area of network computing, etc.

Abbas A. Essa,1966 年生,博士后,主要研究方向为网络计算、实时系统.

Research Background

At present real-time systems are playing an important role in the domain of economy, military, science, etc. Since a hard real-time system is usually subject to stringent reliability and timing constraints, the feasible ways to avoid missing deadline in hard real-time systems are to trade-off the quality of computation and provide diverse versions. Nowadays, the computing models dealing with fault-tolerant schedule in hard real-time systems are classified into three classes: (1) Hardware fault-tolerant model. (2) Imprecise computation model. (3) Software fault-tolerant model. This kind of computing model is used mainly to cope with the running faults of software, such as bug, in order to provide both reliability and timeliness for real-time systems. However, most fault-tolerant scheduling algorithms presented so far in real-time systems are based on the first two models. In this paper, priority-driven policy is adopted to balance the trade-off between scheduling costs and scheduling performance. This paper presents a software fault-tolerant real-time scheduling algorithm, called EBPA. The important contributions of our algorithm include: (1) Two kinds of priority are mixed in our algorithm, the first one is static priority based on static slack that is used to schedule alternates of tasks offline, and the second one is dynamic priority that is used to schedule primaries online to improve scheduling quality; (2) We probe a certain steps during the executions of primaries in order to prevent early failures in execution from triggering failures in the subsequent primary executions as soon as possible. Therefore, ineffective primaries will be cancelled as soon as possible to make more room for subsequent primaries to improve scheduling quality.

This work is supported by the National Natural Science Foundation of China under Grant No.60273075.