硬实时系统中基于软件容错的动态调度算法*

李庆华[1], 韩建军[1+], Abbas A Essa[2], 张 薇[1]

[1](华中科技大学 计算机科学与技术学院,湖北 武汉 430074)

[2](南京大学 电子工程与科学系,江苏 南京 210093)

Dynamic Scheduling Algorithms with Software Fault-Tolerance in Hard Real-Time Systems

LI Qing-Hua[1], HAN Jian-Jun[1+], Abbas A Essa[2], ZHANG Wei[1]

[1](School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China)

[2](Department of Electronic Science and Engineering, Nanjing University, Nanjing 210093, China)

+ Corresponding author: Phn: +86-27-82417143, E-mail: han_j_j@163.com, http://www.hust.edu.cn

Received 2003-09-11; Accepted 2004-06-10

Li QH, Han JJ, Essa AA, Zhang W. Dynamic scheduling algorithms with software fault-tolerance in hard real-time systems. *Journal of Software*, 2005,16(1):101−107. http://www.jos.org.cn/1000-9825/16/101.htm

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 so far are designed to deal with hardware faults, less of those take possible software faults into account. This paper presents two software fault-tolerant real-time scheduling algorithms that are similar to EDF (earliest deadline first), called PKSA (probing *K*-step algorithm) and CUBA (changing utilization-based algorithm). The most important contribution of the algorithms is the probing of certain steps during the execution of tasks in order to prevent early failures in execution from triggering failures in the subsequent job executions. Therefore, the algorithms increase the successful percentage of task's completion, and meanwhile decrease the wasted CPU time slots. The simulation experiments show that the algorithms have much better trade-offs between scheduling costs and performance than the well-known algorithms so far.

Key words: real-time system; software fault-tolerance; 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.提出了两种类似EDF(earliest deadline first)的软件容错的动态实时调度算法:PKSA(probing *K*-step algorithm)和CUBA(changing utilization-based algorithm).两种算法在任务执行过程中,通过若干试探性检测步骤,提高了任务可执行性的预测,尽可能地避免了任务早期的失败对后续任务的影响,因此提高了任务的完成率,同时也有效地减少了浪费的

* Supported by the National Natural Science Foundation of China under Grant No.60273075 (国家自然科学基金)

作者简介: **李庆华**(1940−),男,教授,博士生导师,主要研究领域为并行算法,入侵检测;**韩建军**(1972−),男,博士生,主要研究领域为并行分布式计算,调度算法;**Abbas A Essa**(1966−),男,博士后,主要研究领域为网格计算;**张薇**(1972−),女,博士生,主要研究领域为并行分布式计算,网格计算.

CPU 时间片数量.通过实验测试,与目前所知的同类算法相比,具有更佳的调度性能——调度成本平衡性.

关键词: 实时系统;软件容错;调度算法

中图法分类号: TP316 **文献标识码**: A

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、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,在硬实时系统中,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(imprecise computation)计算模型.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、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.

本文提出了两种新的类似 EDF(earliest deadline first)的软件容错动态实时调度算法 PKSA(probing K-step algorithm)及 CUBA(changing utilization-based algorithm),算法以容错截止期作为任务静态优先权,以通知时间作为任务的动态优先权.在任务主部分执行过程中通过若干试探性检测步骤,提高了该任务可执行性的预测,尽可能地避免了任务早期的失败对后续任务的影响,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.

本文第 1 节介绍计算模型.第 2 节描述文献[7]中的 BCE 算法.第 3 节、第 4 节详细介绍 PKSA 及 CUBA 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.

1 计算模型

硬实时系统包含一个周期性任务集合 $T=\{T_i\}, 1\le i\le N$,其中 N 为任务总数.每个任务 T_i 的周期为 PE_i,软件的错误概率为 FP_i.每个任务 T_i 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 $a_i < p_i$.令 $APR=\sum\frac{a_i}{p_i\cdot N}$ 为系统中任务替代部分计算时间与主部分计算时间之比.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($NT_{i,j}$)表示 $A_{i,j}$ 的开始执行时间,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 $A_{i,j}$. $ED_{i,j}$ 为 $A_{i,j}$ 的结束时间.计划周期(planning cycle)PC 为所有 N 个任务周期的最小公倍数,即 $LCM(PE_1, PE_2, \ldots, PE_N)$.在一个计划周期内, T_i 执行的次数为 N_i.处理机利用率 U 为 $\sum_{i=1}^{N}\frac{p_i}{PE_i}$.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.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的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.

2 BCE 算法

在文献[7]中提出了一种基于速率单调(rate monotonic)[8]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.按照优先权非升顺序,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间隔并不保证连续性.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.任务实例的替代部分在任务执行时具有最高优先权,在任意时间 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,调整相应受影响的其他任务替代部分所占用的时间间隔,并计算新的通知时间.

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总和为 $I=\sum_m\sum_n\sum_p(ED_p(m,n)-ST_p(m,n))$,其中 $t \le ST_p(m,n) < ED_p(m,n) \le NT_{i,j}$,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意时刻,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可用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.

3 PKSA 算法

本文提出两种新的类似 EDF[8]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确定每个任务的优先权(静态优先权),容错截止期较小的任务具有较高的优先权.令 $P(T_i)$ 为 T_i 的静态优先权.然后按照优先权非升顺序,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,选择已到达但未完成的主部分中具有最小通知时间(动态优先权)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小的任务,由于时刻 t 没有新的主部分到达且不需要执行任意替代部分,在上一时刻 t',$P_{i,j}$ 具有最小的通知时间且未完成,因此在时刻 t,$P_{i,j}$ 仍然具有最小的通知时间,因此选择 $P_{i,j}$ 执行. □

从引理 1 可以看出,每次选择待执行主部分时,并不需要从未完成的主部分中重新选择通知时间最小的任务,因而可以节省调度成本.

引理 2. 若在时间 t,待执行主部分为 $P_{i,j}$,$\exists m,n$ 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 $r_{m,n} \le r_{i,j}$,则按照算法,在时间 t 时,由于 $P_{m,n}$ 及 $P_{i,j}$ 均已到达,且 $P_{m,n}$ 的通知时间小于 $P_{i,j}$ 的通知时间,因此在时间 t,应选择 $P_{m,n}$ 为待执行主部分,与假设矛盾. □

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小的通知时间而被抢占,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的执行时间均在 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. 若对于任务 i 有 $a_i < p_i$,则在时间 t,对任一待执行主部分 $P_{i,j}$,在 $NT_{i,j}$ 之前可以利用的执行时间不大于可用时间 $AT_{i,j}$ (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 T_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占用的时间间隔的总和.根据第 2 节中

$AT_{i,j}$的定义,定理得证. □

定理 2. 在任意时刻 t,若 $P_{i,j}$完成并释放 $A_{i,j}$,对任意替代部分 $A_{m,n}$占用的时间间隔[$ST_p(m,n), ED_p(m,n)$],若 $ST_p(m,n) \geq ED_{i,j}$或 $P(T_m) > P(T_i)$,重新执行向后分配替代部分时间间隔的方法之后,则该时间间隔保持不变.

证明:对任意未完成的 $A_{m,n}$,(1) 若 $P(T_m) > P(T_i)$,按照向后分配替代部分时间间隔的算法, $A_{m,n}$ ($1 \leq n \leq N_m$) 先于 $A_{i,j}$ 分配时间间隔,因此 $A_{i,j}$ 的释放并不影响 $A_{m,n}$.(2) 若 $P(T_m) < P(T_i)$,1) 如果 $NT_{m,n} \geq ED_{i,j}$,则按照向后分配算法, $A_{m,n}$ 在 $ED_{i,j}$"之前"已经分配到了具有最大的 $NT_{m,n}$,因此 $A_{i,j}$ 的释放并不影响 $A_{m,n}$.2) 如果 $NT_{m,n} < ED_{i,j}$ 且 $ED_{m,n} > ED_{i,j}$,则 $A_{m,n}$ 在[$ED_{i,j}, ED_{m,n}$]之间的时间间隔在 $ED_{i,j}$"之前"被分配,因此 $A_{i,j}$ 的释放并不影响 $A_{m,n}$ 在[$ED_{i,j}, ED_{m,n}$]之间的时间间隔.综合上述情况,定理得证. □

从定理 2 可以看出,在任意时刻 t,若 $P_{i,j}$ 完成并释放 $A_{i,j}$,不需要重新执行向后分配替代部分时间间隔的算法,仅对区间[$t, ED_{i,j}$]之内,优先权小于 T_i 的替代部分的时间间隔重新分配即可,这样可以降低调度成本.而 BCE 并未阐述如何调整时间间隔.

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,对于任意待执行主部分 $P_{i,j}$ 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性地计算其占用时间,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,提前撤消无效的待执行主部分,可以留出更多的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,执行 K 次试探性计算.其算法过程如下:

Step 1. 设置试探步数 K 及一个临时的时间间隔链表.初始时令 k=0.在时间 t,对任意待执行主部分 $P_{i,j}$,根据定理 1 判断是否有足够的可用时间,若不满足,则撤消该主部分,否则从时间 t 到 $NT_{i,j}$,依次判断占有该时间间隔的任务.

Step 2. (1) 若占有该时间间隔的任务的主部分已失败,则将其时间间隔添加至临时的时间间隔链表.(2) 若该任务未开始执行,k=k+1,则根据定理 1 判断临时的时间间隔链表是否有足够的空闲时间,1) 若满足,则将该任务主部分实际执行的时间间隔添加到临时时间间隔链表.2) 若不满足,则同(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.

Step 3. 若还有时间间隔未考虑且 k<K,则执行 Step 2.

Step 4. 若仍有时间间隔未考虑,则按 Step 2 的第 1 种情况处理.

Step 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 $P_{i,j}$.

PKSA 算法描述如下:

第 1 步. 按照任务优先权非升顺序,向后分配替代任务占用的时间间隔.

第 2 步. 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,并从候选队列中撤消该替代部分的主部分(若未被撤消),转向第 4 步.(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 $A_{i,j}$ 占用的时间间隔,并调整受影响的替代部分的时间间隔(根据定理 2,从 $ED_{i,j}$ 至 t 按第 1 步方法,重新调整该区间内的时间间隔),并转向第 2.1 步.(4) 若上次选择的待执行主部分运行错误,则从候选队列中撤消该主部分,并转向第 2.1 步.(5) 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,则根据引理 1 处理,并转向第 4 步.

第 2.1 步. 若待执行主部分为空,则从非空候选执行队列中,选择新的待执行主部分.

第 2.2 步. 若待执行主部分为未开始执行的主部分,则执行 K 次试探性算法.若失败,则从候选队列中撤消该主部分,并重复第 2.1 步.

第 2.3 步. 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,并重复第 2.1 步.

第 3 步. 若待执行主部分为空,则按照第 2 节 EIT 算法处理.

第 4 步. 若仍有任务未完成,则设置下一时间 t,转向第 2 步.

从算法中可以看到本算法与传统的 EDF 的区别.传统的 EDF 算法在上述第 2 步情况(5)时,要求重新选择任务,而本算法仅在有新主部分到达,或者运行失败时,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 次试探性计算,同样也是为了降低调度成本.

4 CUBA 算法

CUBA 算法与 PKSA 算法类似,仅在第 2.2 步当 $U \geq 1.1$ 时(利用率较小时,使用 CUBA 性能变化不大,且提高了调度成本),执行基于变动利用率的试探性步骤.令主任务 $P_{i,j}$ 的变动利用率为 $CU_{i,j} = \frac{p_i}{AT_{i,j}}$ ($AT_{i,j}$ 起始时间为 $r_{i,j}$).该步骤显示如下:

Step 1. 初始设 u=0.在时间 t,对任意待执行主部分 $P_{i,j}$,根据定理 1,判断是否有足够的可用时间,若不满足,则从候选队列中撤消该主部分.

Step 2. 否则,从时间 t 到 $NT_{i,j}$,依次判断占有该时间间隔的任务.若占有该间隔的任务的主部分 $P_{m,n}$ 未开始执行,则计算 $CU_{m,n}$,若 $CU_{m,n} \leq 1$,则 $u = u + CU_{m,n}$.求出 $r_{p,q} = \min(r_{m,n})$,$ED = \max(ED_p(m,n))$,其中 $t < ST_p(m,n) < ED_p(m,n) \leq NT_{i,j}$.

Step 3. 若 u>1,则 u=1. $AT'_{i,j} = NT_{i,j} - t - (ED - r_{p,q}) \cdot u$,若 $AT'_{i,j} < p_i$,则从候选队列中撤消该主部分,否则 $P_{i,j}$ 为待执行主部分.

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 100 000.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],[200,800],[800,12000].主部分在每个周期区间的计算时间分别为[20,50],[30,90],[50,300],替代部分计算时间分别为主部分计算时间的 0.3~0.7 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 0~0.2.处理机利用率选择 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 OS2.4.18,256M RAM,1.7GHz Intel CPU 的单机上模拟测试.

5.1 PKSA,CUBA和BCE算法的性能测试

在测试中,性能指标采用任务丢失率(TLP)及浪费的 CPU 时间片的数量(NWTS).TLP 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主部分计算时间($\sum_i p_i \cdot N_i$)的比例,反映了调度的质量.NWTS 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,反映了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 PKSA 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).图 1 显示出 TLP,以百分比表示,横轴代表利用率为 0.8~1.8.图 2 显示 NWTS.BCE 平均计算时间为 1.68s,CUBA 平均计算时间为 1.88s,PKSA 平均计算时间为 1.94s,PKSA 及 CUBA 调度成本均高于 BCE.但是实验结果表明,PKSA,CUBA 无论在 TLP,还是在 NWTS 上,较 BCE 均取得了明显的改善.PKSA 在利用率为 1.1~1.8 时,与 BCE 相比,TLP 平均降低了 5.96%,即平均多执行了 5 960 个任务时间单位,NWTS 平均降低了 6 184 个 CPU 时间单位.尤其在 U>1.5 时,CUBA 及 PKSA 的 NWST 停滞增长.总体而言,随着处理机利用率的提高,我们的算法在调度性能上比 BCE 的优势就越明显.一般情况下,CUBA 调度成本低于 PKSA,但调度性能均略低于 PKSA.而当处理机利用率大于 1.6 时,CUBA 与 PKSA 的 TLP 相当,NWTS 略高于 PKSA.因而 PKSA,CUBA 比 BCE 具有更好的性能——成本的均衡性.

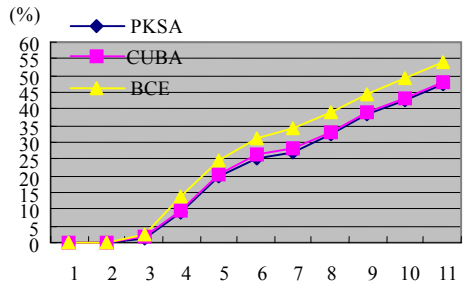

Fig.1 Comparison of TLP between three algorithms

图 1 3 种算法 TLP 的比较

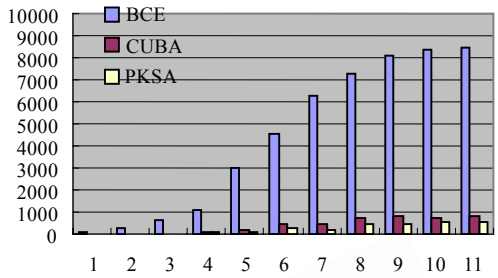

Fig.2 Comparison of NWTS between three algorithms

图 2 3 种算法 NWTS 的比较

5.2 *K*值对PKSA的影响

本次实验测试 *K* 值对 PKSA 算法性能的影响.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,当利用率在 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,负载的增大,在任务主部分执行期间,对其产生影响的主部分的数量增加,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.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.

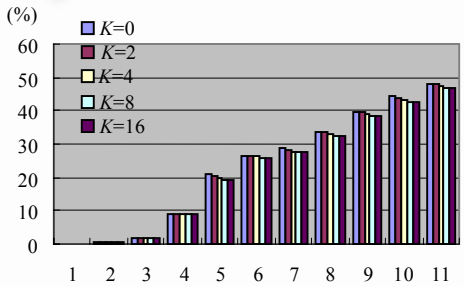

Fig.3 The influence of different *K* on TLP

图 3 不同 *K* 值对 TLP 的影响

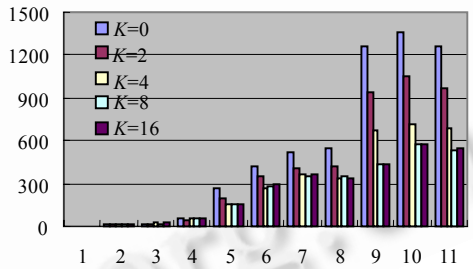

Fig.4 The influence of different *K* on NWTS

图 4 不同 *K* 值对 NWTS 的影响

5.3 *APR*对调度性能的影响

本次测试分析 *APR* 对调度性能的影响.当 *APR*>0.6,且处理机利用率 *U*>1.5 时,替代部分的可调度性较差,因此这里只考虑 *APR* 为 0.3~0.6,*U* 为 1.0~1.5,错误概率为 0~0.2 的情况.本次测试的数据为不同错误概率情况下的平均值.

一般而言,随着 *APR* 的提高,TLP 及 NWTS 都会提高,其原因在于替代部分计算时间的增加造成主部分可利用时间的相对减少,降低了调度性能.图 5 显示在 *U* 为[1.0~1.5]的情况下,BCE 与 PKSA 以 *APR* 为 0.3 为基准,不同 *APR* 下 TLP 的提高幅度的差值.实验数据表明,随着 *APR* 的提高,BCE 的 TLP 提高幅度高于 PKSA,且提高的幅度也加大.当 *U*>1.1 时,随着处理机利用率的提高,两种算法 TLP 提高幅度的差值逐渐减少.当 *U*>1.4 时,两种算法 TLP 的提高幅度大致相当,其原因在于过重的处理机负载下利用率成为主导因素,*ARP* 对调度性能影响不大.图 6 显示在 *U* 为[1.0~1.5]时,BCE 与 PKSA 算法在不同处理机利用率下,*APR* 为 0.6 与 *APR* 为 0.3 时 NWTS 的差值.当 *U*>1.4 时,两种算法 NWTS 的差值趋于 0,原因同上,在过重的处理机负载下,利用率成为主导因素.这里未列出 *APR* 对 CUBA 的影响,数据表明,性能指标上 CUBA 略低于 PKSA,也具有上述的现象.因此 CUBA,PKSA 与 BCE 相比,对 *APR* 具有更好的适应性.

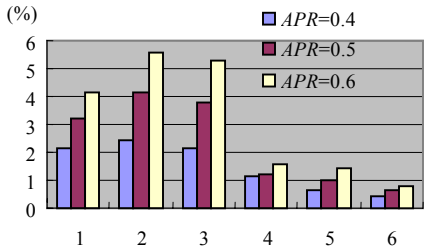

Fig.5 The adaptability of TLP of PKSA and BCE to *APR*

图 5 PKSA 及 BCE 的 TLP 对 *APR* 的适应性

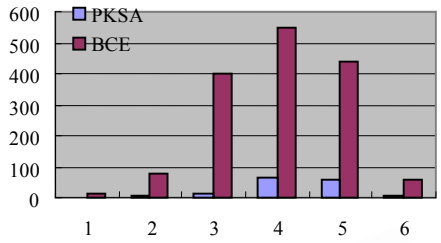

Fig.6 The adaptability of NWTS of PKSA and BCE to *APR*

图 6 PKSA 及 BCE 的 NWTS 对 *APR* 的适应性

6 结束语

本文针对硬实时系统中软件容错模型,提出了两种新的类似 EDF 的算法,并在任务执行过程中通过若干试探性检测步骤,提高了该任务可执行性的预测,尽可能地避免了任务早期的失败对后续任务的影响,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,并与目前所知的同类算法相比,具有更佳的调度性能——调度成本平衡性.同时也阐述了 *K* 值的选择及 *APR* 对调度性能的影响.

References:

[1] Xu LH, Bruck J. Deterministic voting in distributed systems using error-correcting codes. IEEE Trans. on Parallel and Distributed Systems, 1998,9(8):813−824.

[2] Liu H, Wei SM. A fault-tolerant scheduling algorithm based on EDF for distributed control systems. Journal of Software, 2003,14(8):1371−1378 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/14/1371.htm

[3] Qin X, Han ZF, Pang LP. Real-Time scheduling with fault-tolerance in heterogeneous distributed systems. Chinese Journal of Computers, 2002,25(1):49−56 (in Chinese with English abstract).

[4] Castorino A, Ciccarella G. Algorithm for real-time scheduling of error-cumulative tasks based on the imprecision computation approach. Journal of Systems and Architecture, 2000,46:587−600. http://www.elsevier.com/locate/sysarc

[5] Manimaran G, Manikutty A, Murthy CSR. DHARMA: A tool for evaluating dynamic scheduling algorithms for real-time multiprocessor systems. Journal of Systems and Software, 2000,50:131−149. http://www.elsevier.com/locate/jss

[6] McElhone C, Burns A. Scheduling optional computations for adaptive real-time systems. Journal of Systems and Architecture, 2000,46:49−77. http://www.elsevier.com/locate/sysarc

[7] Han CC, Shin KG, Wu J. A fault-tolerant scheduling algorithm for real-time periodic tasks with possible software faults. IEEE Trans. on Computers, 2003,52(3):362−372.

[8] Liu CL, Layland JW. Scheduling algorithms for multi-programming in a hard real-time environment. Journal of ACM, 1973,20(1): 46−61.

[9] Chetto H, Chetto M. Some results of the earliest deadline scheduling algorithm. IEEE Trans. on Software Engineering, 1989,15(10):1261−1269.

附中文参考文献:

[2] 刘怀,费树岷.基于 EDF 的分布式控制系统容错调度算法.软件学报,2003,14(8):1371−1378. http://www.jos.org.cn/1000-9825/14/1371.htm

[3] 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.