计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2023,32(1):109−118 [doi: 10.15888/j.cnki.csa.008894]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

实时异构系统中的积极复制容错调度算法①

毛灯锋[1,2], 胡　威[1,2], 刘　静[1,2], 甘　雨[1,2]

[1](武汉科技大学 计算机科学与技术学院, 武汉 430065)
[2](智能信息处理与实时工业系统湖北省重点实验室, 武汉 430065)
通信作者: 胡　威, E-mail: huwei@wust.edu.cn

摘　要: 在设计实时异构系统中的容错调度算法时, 既要考虑到实时性的约束, 又要最大化系统的可靠性. 此外, 异构系统中的并行应用调度问题已经被证明了是 NP 完全问题. 现有的容错调度算法大多采用复制技术来提升系统的可靠性, 但是任务的多次执行会导致应用执行时间变长, 系统实时性下降. 为此, 提出了一个基于积极复制技术的容错调度算法, 该算法连续的复制任务集中对当前系统实时性影响最小的任务, 然后将任务集中的所有任务调度至最早完成的处理器, 用以在满足实时性约束的同时, 提升系统的可靠性. 实验表明, 相比于同样着眼于实时异构系统的 DB-FTSA 算法, 该算法在实时性约束严格的情况下, 可靠性有较大提升.

关键词: 异构系统; 容错调度; 积极复制; 可靠性; 实时性

引用格式: 毛灯锋,胡威,刘静,甘雨.实时异构系统中的积极复制容错调度算法.计算机系统应用,2023,32(1):109–118. http://www.c-s-a.org.cn/1003-3254/8894.html

Active Replication Fault-tolerant Scheduling Algorithm in Real-time Heterogeneous Systems

MAO Deng-Feng[1,2], HU Wei[1,2], LIU Jing[1,2], GAN Yu[1,2]

[1](School of Computer Science and Technology, Wuhan University of Science and Technology, Wuhan 430065, China)
[2](Hubei Province Key Laboratory of Intelligent Information Processing and Real-time Industrial System, Wuhan 430065, China)

Abstract: When designing a fault-tolerant scheduling algorithm in a real-time heterogeneous system, it is necessary to consider the constraints of real-time and maximize the reliability of the system. Furthermore, parallel application scheduling problems in heterogeneous systems have been shown to be NP-complete. Most of the existing fault-tolerant scheduling algorithms use replication technology to improve the reliability of the system, but the multiple execution of tasks will lead to longer application execution time and reduced system real-time performance. Therefore, a fault-tolerant scheduling algorithm based on active replication technology is proposed. The algorithm continuously replicates the tasks that have the least impact on the real-time performance of the current system in the task set and then schedules all tasks in the task set to the earliest completed processor. While meeting the real-time constraints, the algorithm improves the reliability of the system. Experiments show that the reliability of the proposed algorithm has been improved under strict time constraints compared with that of the DB-FTSA algorithm which also focuses on real-time heterogeneous systems.

Key words: heterogeneous systems; fault-tolerant scheduling; active replication; reliability; real-time

实时异构系统的应用非常广泛, 例如无人驾驶系统, 无人机系统, 工业控制系统等. 一个异构系统通常由一组处理器组成, 处理器之间通过总线相连, 能够进行数据通信. 随着处理器的增多, 当今的应用程序并行

① 基金项目: 国家自然科学基金面上项目 (62176191)
收稿时间: 2022-05-19; 修改时间: 2022-06-20; 采用时间: 2022-07-01; csa 在线出版时间: 2022-08-26
CNKI 网络首发时间: 2022-11-15

程度也变得越来越高, 并且这些应用中的任务拥有明显的数据依赖[1−3]. 比较典型的应用有快速傅里叶变换[4]和高斯消元[5]. 异构系统中的处理器拥有不同的计算性能和能量消耗, 在执行并行应用时相较于同构系统拥有明显的优势[6,7].

但由于严峻的运行环境, 处理器在运行过程中偶尔也会发生错误. 错误一般分为两种, 暂时性错误和永久性错误. 暂时性错误会造成任务执行结果的错误, 进而造成整个应用执行出错, 通常因为辐射、高温和电磁干扰等环境因素导致. 永久性错误产生的原因是处理器设计方面的缺陷, 一旦发生永久性错误, 除非更换处理器, 否则该处理器不能继续执行任务[8]. 两种错误都会导致系统可靠性的下降. 可靠性指的是一个调度成功执行的概率, 现在已经逐渐成为评判一个系统性能好坏的标准之一, 并且已在一些相关标准中定义, 例如 IEC 61508 和 ISO 9000[9]. 复制技术是一个有效的提升系统可靠性的解决方案. 在使用了复制技术的系统中, 只要任务的任意一个副本执行成功, 那么该任务被认为执行成功[3]. 复制技术会通过增加任务的执行次数来提升任务的执行成功率, 进而提升系统的可靠性. 然而, 系统中需要执行的任务增加了, 应用的执行时间会变长, 系统的实时性会下降.

为了解决实时异构系统中的并行应用调度问题, 最早 Wang 等人采用了遗传算法[9]. 但是遗传算法的时间复杂度较高, 不适用于大规模的任务调度. HEFT 算法是一个经典的启发式算法[6], 它的调度策略是, 先为任务分配优先级, 构造任务优先队列, 然后不断出队, 为出队的任务选择完成时间最早的处理器, 最终可以得到一个较短的应用执行时间. 此外, 还有基于复制的调度算法. 该类算法的核心在于减少任务之间数据传递产生的时间成本, 从而达到减小整个应用执行时间的目的[10]. 上述的 3 种算法都没有考虑到处理器中可能出现错误导致最终应用执行结果出错.

为了提升系统的可靠性, 前人提出了很多容错调度算法. 容错调度算法大多使用了复制技术来提升系统的可靠性. 复制技术分为两种, 一种为积极复制, 另一种为被动复制. 积极复制技术是在任务调度前, 就对任务进行复制. 而被动复制需要系统额外配备一个错误检测器, 每当一个任务执行完成后, 错误检测器都会对任务的结果进行检测, 假如任务结果错误, 再对执行失败的任务进行复制[11]. 被动复制技术大多被用于节能调度算法, 其在任务执行出错后才会复制任务, 并且重新执行任务[12,13]. 使用被动复制技术可以有效地减少系统执行任务的数量, 从而达到节能的效果. 两种复制技术都对应用的执行时间有不同程度的增加, 但相比于被动复制, 积极复制对于硬件的要求不高, 对实时性的影响较小, 因此很多同时考虑可靠性和实时性的容错调度算法都采用了积极复制技术进行容错, 例如 FTSA 算法[14]、DB-FTSA 算法[15]、DBSA 算法[16]. 其中 FTSA 算法同时考虑系统中出现的暂时性错误和永久性错误, DB-FTSA 只考虑暂时性错误, DBSA 只永久性错误. 这 3 个算法都面向实时异构系统, 在提升系统可靠性的同时, 兼顾了系统的实时性. 其中考虑了永久性错误的容错调度算法, 一定要将同一任务的两个副本任务分配至不同的处理器, 这样才能保证其中一个处理器出现永久性错误后, 任务在另一个处理器上依然能执行成功. 而 HRRM[8] 面向异构云计算系统, 不考虑实时性的约束, 更加侧重于节省系统的算力资源, 在保证系统可靠性的同时, 尽量减少系统中的任务冗余. 这些采用了积极复制技术的容错调度算法的执行过程包含为两个部分, 调度策略和复制策略. 调度策略是指为每个任务分配执行优先级, 然后调度至合适的处理器. 复制策略指的是为每个任务制定合理的复制次数.

本研究提出了一个面向实时异构系统的, 基于积极复制技术的容错调度算法 (fault-tolerant scheduling algorithm with replication strategy of minimum impact on real-time performance, FTSA-RSMI). 由于处理器执行任务时, 出现的错误大多为暂时性错误[7,8], 因此该算法针对实时异构系统中出现的暂时性错误, 在满足实时性约束的条件下, 最大化系统的可靠性. 本研究和上述研究的不同点及主要贡献如下.

(1) 本文提出了一个全新的复制策略, 该复制策略会优先复制任务集中对实时性影响最小的任务. 基于这个复制策略, 本文提出了一个全新的基于积极复制技术的面向实时异构系统的容错调度算法 FTSA-RSMI.

(2) 实验证明 FTSA-RSMI 能够在截止期限的约束下高效地提升系统的可靠性, 例如, 在调度随机生成

DAG 应用时, FTSA-RSMI 能够得到 90.68% 的系统可靠性, 而 DB-FTSA 只能得到 40.17% 的系统可靠性. 在调度真实应用分子动力学代码时, FTSA-RSMI 得到的系统可靠性比 DB-FTSA 高 15%.

1 模型及问题阐述

1.1 系统模型

定义一个由 m 个全连通的处理器组成的异构多处理器系统, 使用 $U=\{u_1, u_2, u_3, \cdots, u_m\}$来表示这个处理器集, 其中不同的处理器拥有不同的运行速度. 图 1 给出了一个由 3 个处理器 u_1, u_2, u_3 组成的系统架构模型.

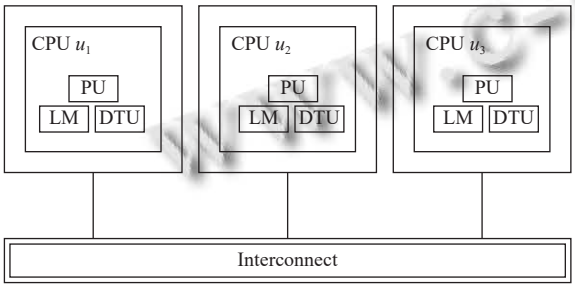


图 1 3 个异构处理器组成的系统架构模型

每个处理器模块分别由 3 个模块组成, 处理器单元 (processing unit, PU)、本地内存 (local memory, LM) 以及数据传输单元 (data transfer unit, DTU). PU 负责一系列的计算操作. LM 是一个小型的低延时的存储并且负责在 PU 的执行过程中供 PU 访问. DTU 负责实现处理器和存储之间的数据传输操作. 处理器之间的信息交流通过互连进行, 例如总线. 假设所有处理器间通信都在没有争用的情况下进行.

1.2 应用模型

一个并行应用通常使用一个有向无环图 (DAG) 来表示, 其中的节点表示任务, 边表示任务间的数据依赖[17–19]. 使用一个四元组来表示一个 DAG 应用 $G=(V, E, C, W)$. $V=\{v_1, v_2, v_3, \cdots, v_n\}$用来表示 DAG 中的节点任务集, 其中 n 表示任务节点的数量. E 用来表示任务之间的交流边集, e_{ij} 表示 DAG 中的边, 表示任务 v_i 和任务 v_j 之间存在数据依赖. C 表示的是数据传递消耗的时间集, c_{ij} 表示任务 v_i 和任务 v_j 在分配至不同处理器时数据传递消耗的时间. W 是一个 $m\times n$ 的矩阵, 其中 m 表示处理器数量, n 表示应用中的任务数量, w_{ik} 表示任务 v_i 在处理器 u_k 上的执行时间. 定义 $pre(v_i)$ 表示任务 v_i 的前驱任务集合, $suc(v_i)$ 表示任务 v_i 的后继任务集合. 一个没有前驱任务的任务被称为入口任务, 被定义为 v_{entry}, 一个没有后继任务的任务被称为出口任务, 被定义为 v_{exit}. 如果一个 DAG 拥有多个出口任务或者多个入口任务, 那么一个零权值依赖的虚拟出口任务或入口任务将会生成并被添加至图中. 图 2 展示了一个包含 5 个任务的并行应用.

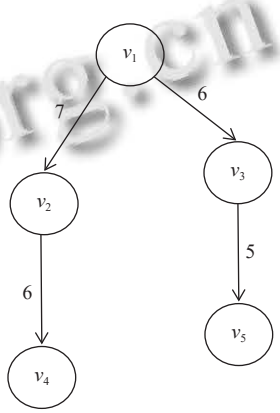


图 2 由 5 个任务组成的并行应用

图 2 中的应用将会被执行在图 1 中的系统中. 表 1 展示了一个 3×5 的表格, 表示的是图 2 中的所有任务在 3 个不同处理器上的执行时间. 例如, 任务 v_1 在处理器 u_2 上的执行时间为 8 个单位时间, 用公式表示为 $w_{12}=8$.

表 1 任务在不同处理器上的执行时间

任务	u_1	u_2	u_3
v_1	7	8	5
v_2	6	10	9
v_3	5	7	10
v_4	6	4	9
v_5	6	7	5

1.3 错误模型

假定处理器的故障率是恒定的, 处理器在连续执行一段时间时, 出现暂时性错误的概率服从泊松分布[1,2,8,14,20,21]. 那么任务 v_i 在被分配至处理器 u_k 上执行一次的成功率计算方式如式 (1)[1,2,8,22]:

$$R(v_i, u_k) = \mathrm{e}^{-\lambda_k w_{ik}} \tag{1}$$

其中, λ 表示处理器的故障率. 在使用了复制技术的系统中, 只要该任务的其中一个副本 (包括原始任务) 执行成功, 该任务被认为执行成功, 那么当系统中执行了

p 个任务 v_i 的副本时, 任务 v_i 的执行成功率的计算方式如式 (2):

$$R(v_i)=1-\prod_{\beta=1}^{p}(1-R(v_i^{b\beta},pro(v_i^{b\beta}))) \tag{2}$$

其中, $v_i^{b\beta}$ 表示任务 v_i 的副本, $pro(v_i^{b\beta})$ 表示执行任务 $v_i^{b\beta}$ 的处理器. 整个应用的可靠性等于应用中所有任务执行成功率的乘积, 计算方式如式 (3):

$$R(G)=\prod_{i=1}^{n}R(v_i) \tag{3}$$

1.4 问题阐述

FTSA-RSMI 用来解决实时异构系统中出现的暂时性错误导致的可靠性下降的问题, 问题的描述如下: 输入一个包含 n 个任务节点的 DAG 应用 $G=(V, E, C, W)$, 一个由 m 个处理器 $U=\{u_1, u_2, u_3, \cdots, u_m\}$ 组成的实时异构系统, 以及应用的截止期限 D, 目标是设计一个高效的容错调度算法, 在截止期限的约束下, 使用复制技术复制任务, 然后将所有任务调度至处理器执行, 并且最大化系统的可靠性. 可以表示为: 在满足式 (4) 的条件下, 最小化式 (5).

$$makespan \Leftarrow D \tag{4}$$

$$R(G)=\prod_{i=1}^{n}R(v_i) \tag{5}$$

2 容错调度算法 (FTSA-RSMI)

一个使用了积极复制技术的容错调度算法包含两个部分, 调度策略和复制策略. 调度策略决定了任务的执行次序和处理器选择规则. 复制策略决定了任务集中的所有任务的复制次数. FTSA-RSMI 会动态的根据截止期限来调整任务复制的数量, 在截止期限的约束下, 最大化系统的可靠性. 下文将首先给出调度策略, 然后给出复制策略, 之后会详细地描述 FTSA-RSMI 的执行过程, 最后会给出一个例子用来比较 DB-FTSA 和 FTSA-RSMI 的区别.

2.1 调度策略

FTSA-RSMI 使用经典的向上等级法 (upward rank) 来确定任务的执行优先级. 定义 $rank_u(v_i)$ 作为任务 v_i 的优先级, 计算方式如式 (6):

$$rank_u(v_i)=\overline{w_i}+\max_{v_j\in suc(v_i)}(\overline{c_{ij}}+rank_u(v_j)) \tag{6}$$

其中, 出口任务 v_{exit} 的优先级为 $\overline{w_{exit}}$, $\overline{w_i}$ 表示任务 v_i 在所有处理器上的平均执行时间, $\overline{c_{ij}}$ 表示任务 v_i 和任务 v_j 的平均交流时间. 任务的 $rank_u$ 越高, 任务的执行次序就越前. 表 2 展示了图 2 中所有的任务优先级计算结果和执行次序.

表 2 任务的优先级

优先级/执行次序	v_1	v_2	v_3	v_4	v_5
$rank_u$	34	21	18	6	6
执行次序	1	2	3	4	5

FTSA-RSMI 在为任务选择处理器时, 都会遍历所有处理器, 并且选择其中完成时间最早的处理器. 定义 $est(v_i, u_k)$ 为任务 v_j 在处理器 u_k 上的开始时间, $f(v_i)$ 为任务的 v_i 的完成时间, 那么处理器选择策略可以表示为:

$$f(v_i)=\min_{u_k\in U}\{est(v_i,u_k)+w_{ik}\} \tag{7}$$

任务 v_i 在处理器 u_k 上开始执行前, 处理器必须要是空闲的, 并且任务 v_i 需要的输入数据都已经被传递至处理器 u_k. 所以 $est(v_i, u_k)$ 的计算方式如下:

$$est(v_i,u_k)=\max\{\max_{v_p\in pre(v_i)}\{f(v_p)+c_{pi}\},idle(u_k)\} \tag{8}$$

任务 v_p 是任务 v_i 的前驱集合中的任意一个任务, 需要注意的是, 假如任务 v_p 和任务 v_i 被分配至同一个处理器执行, 那么两个任务之间的依赖数据不需要在处理器之间进行传递, c_{pi}=0. $idle(u_k)$ 表示处理器的空闲时刻, 即当前处理器上当前正在执行的最后一个任务的结束时刻. 当所有的任务都被分配至处理器之后, 整个应用的执行时间通过式 (9) 计算出来:

$$makespan(G)=\max_{v_i\in V}\{f(v_i)\} \tag{9}$$

其中, $makespan$ 即为整个应用的执行时间, 在本文中也称之为调度长度.

调度策略的算法如算法 1.

算法 1. 调度策略算法

1) 遍历任务集, 使用式 (6) 计算任务优先级.
2) 按照任务的优先级与任务之间的依赖关系构造任务执行队列.
3) while 队列不为空 do
4) 出队一个任务, 遍历处理器集, 依照式 (7) 选择任务完成时间最早的处理器.
5) 依照式 (1) 和式 (2) 可以计算出任务的执行成功率.
6) end while
7) 依照式 (7) 计算整个应用的执行时间 *makespan*.

2.2 复制策略

对于一个任务集 S 而言, 定义 S_{com} 表示任务集中那些复制次数达到 α 次的任务子集, 定义 S_{inc} 表示任务集中那些复制次数未达到 α 次的任务子集. 将任务集 S_{inc} 中的任务 v_i 复制 1 次后, 新的任务集 S_i 将会生成, 用公式表示为$S_i - S = \{v_i^x\}$, 其中的 v_i^x 为任务 v_i 的新副本. 使用上文给出的调度策略, 调度任务集 S_i, 就能够得到新的应用执行时间 $makespan(S_i)$. 这样一来, 复制任务 v_i 对系统当前的实时性造成的影响 $IMP(v_i)$ 可以通过式 (10) 计算:

$$IMP(v_i) = makespan(S_i) - makespan(S) \quad (10)$$

为了尽可能地减小任务复制对系统实时性造成的影响, FTSA-RSMI 优先复制当前任务集 S 中对系统实时性影响最小的任务 $v_{rep(S)}$, $v_{rep(S)}$ 的查找方式如式 (11):

$$IMP(v_{rep(s)}) = \min_{v_i \in S_{inc}} \{IMP(v_i)\} \quad (11)$$

在实际的实验过程中, 每增加一个任务的复制次数, 调度长度都会增加. 当任务的复制次数大于 2 次后, 系统的可靠性增长速度变得缓慢. 为了避免超过截止期限, 任务的最大复制次数被设置为 2 次. 表 3 展示了查找任务集 S={v_1, v_2, v_3, v_4, v_5}中对实时性影响最小的任务的过程.

表 3 复制不同的任务对系统实时性的影响

v_1	v_2	v_3	v_4	v_5	*makespan*	*IMP*
0	0	0	0	0	23	0
1	0	0	0	0	27	4
0	1	0	0	0	26	3
0	0	1	0	0	28	5
0	0	0	1	0	24	1
0	0	0	0	1	28	5

表 3 中的第 1 行表示所有的任务复制次数为 0, 任务集 S={v_1, v_2, v_3, v_4, v_5}, 应用的执行时间 *makespan*=23. 第 2 行表示将任务 v_i 复制一次后, 新的任务集 S_1={v_1, v_1^{b0}, v_2, v_3, v_4, v_5}生成, 使用上文的调度策略调度这个任务集, 可以得到应用的执行时间 *makespan*=27, 那么复制任务 v_1 对于系统实时性的影响 $IMP(v_1)$=4. 根据式 (9) 可知, 对于任务集 S={v_1, v_2, v_3, v_4, v_5}, 应该被复制的任务为 v_4, 该任务对当前系统实时性的影响最小, $IMP(v_4)$=1.

2.3 FTSA-RSMI 执行过程

FTSA-RSMI 的执行过程分为以下 4 步. 第 1 步, 将任务的最大复制次数 α 设置为 0, 构造原始任务集; 第 2 步, 使用复制策略复制任务后调度任务集, 记录调度结果; 第 3 步, 重复执行第 1 步和第 2 步, 直至所有任务的复制次数都达到 α 次, 然后将任务的最大复制次数 α 设置为 2, 重新执行第 1 步和第 2 步; 第 4 步, 一旦调度结果中的应用执行时间大于截止期限, 那么算法停止, 从记录中选择出, 应用执行时间小于截止期限的, 可靠性最大的调度结果.

FTSA-RSMI 的算法如算法 2.

算法 2. FTSA-RSMI

输入: DAG 应用 G={V, E, C, W}, 截止期限 D, 处理器集 U={u_1, u_2, u_3, ⋯, u_m}

输出: 一个实现了容错的调度结果, 应用执行时间 *makespan*, 系统可靠性 $R(G)$

1) for α = 0; α<=2; α++ do
2) while *makespan*<=D 且任务集中仍有部分任务的复制次数小于 α do
3) 使用复制策略复制任务集中的任务.
4) 使用调度策略调度任务集, 并得到 *makespan*, $R(G)$.
5) end while
6) end for
7) 从记录中选择出 *makespan*<=D 且 $R(G)$ 最大的调度结果.

FTSA-RSMI 的时间复杂度的计算步骤如下.

第 1 步. 计算调度策略的时间复杂度. FTSA-RSMI 会遍历整个任务集并计算任务的优先级, 这一步操作的时间复杂度为 O(n). 在计算完任务优先级后, FTSA-RSMI 会不断地从任务集中取出优先级最高的任务, 遍历处理器集, 为该任务选择处理器, 这一步的时间复杂度为 O(m×n).

第 2 步. 计算复制策略的时间复杂度. FTSA-RSMI 会遍历整个任务集, 不断地增加任务的复制次数, 然后调度整个任务集, 遍历完成后, 选择出对实时性影响最小的任务. 这一步的时间复杂度为 O(m×n×n).

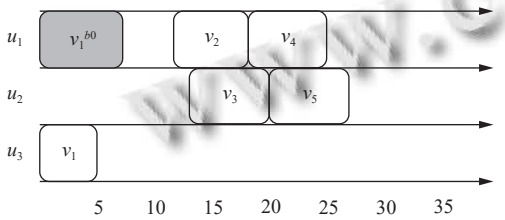
第 3 步. 计算整个算法的时间复杂度. FTSA-RSMI 会不断地复制任务, 在最坏的情况下, 每个任务均被复制两次. 这一步的时间复杂度即为 O(m×n×n×n). 因此整个算法的时间复杂度为 O(m×n+m×n^3).

2.4 实例分析

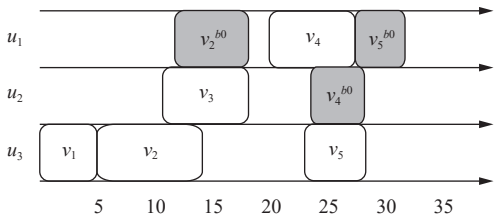
图 3 给出了在截止期限为 32 时, 分别使用 DB-FTSA 和 FTSA-RSMI 调度图 2 中的 DAG 应用的结果. 其中处理器 u_1, u_2, u_3 的故障率分别为 0.008, 0.006, 0.007, 任务在不同处理器上的执行时间在表 1 中给出, 任务

之间的交流时间在图 2 中给出. 图中的矩形表示任务的副本集, 由白色填充的圆角矩形表示原始任务, 由灰色填充的圆角矩形表示复制任务, 任务 v_1^{b0} 即是任务 v_1 的复制任务. 图 3(a) 表示使用 DB-FTSA 调度图 2 中的应用至图 1 中的系统, 最终得到的调度结果. 表 4 中给出了每个任务的开始时刻和结束时刻, 以及每个原始任务的执行成功率.

DB-FTSA 的复制策略是按照任务执行优先级的顺序依次复制任务, 但是当它将任务 v_2 的复制次数增加一次后重新调度任务集, 得到的应用执行时间超过了截止期限, 因此 DB-FTSA 最终只复制了任务 v_1. 最终系统可靠性计算过程为 99.81%×95.31%×95.89%×95.31%×95.89%=83.37%.



(a) DB-FTSA 调度结果, 应用执行时间为 27, 系统可靠性为 83.37%



(b) FTSA-RSMI 调度结果, 应用执行时间为 32, 系统可靠性为 92.07%

图 3 分别使用两种算法在截止期限为 32 的情况下, 调度图 2 中的 DAG 应用

FTSA-RSMI 的复制策略是优先复制任务集中对实时性影响最小的任务. 按照这个复制策略, 任务 v_2, v_4, v_5 被依次复制. 最终系统可靠性的计算过程为 96.56%×99.71%×95.89%×99.89%×99.84%=92.07%.

3 实验结果和讨论

在本节中, FTSA-RSMI 将会和 DB-FTSA 比较从而评估其性能. DB-FTSA 解决的问题和 FTSA-RSMI 相同, 同样只考虑实时异构系统中出现的暂时性错误, 采用了积极复制技术提高系统的可靠性, 但是两个算法的调度策略的实现不同, 复制策略也完全不同. 为了测试算法性能, 实验进行了两组仿真测试. 很多研究都采用了 TGFF 来随机生成 DAG 应用[1,2,6,8,15,16], 因此实验数据第一组为使用 TGFF 随机生成的 DAG 应用, 另一组为真实世界的应用. 仿真平台是一个配备了 2.8 GHz Intel CPU 和 8 GB 内存的电脑, 应用模型和处理器的模型通过 Java 语言模拟.

表 4 调度完成后得到的任务开始时刻和结束时刻, 以及原始任务的执行成功率

调度	任务	开始时间	结束时间	执行成功率 (%)
DB-FTSA	v_1	0	5	99.81
	v_1^{b0}	0	7	—
	v_2	12	18	95.31
	v_3	13	20	95.89
	v_4	18	24	95.31
	v_5	20	27	95.89
FTSA-RSMI	v_1	0	5	96.56
	v_2	5	14	99.71
	v_2^{b0}	12	18	—
	v_3	11	18	95.89
	v_4	20	26	99.89
	v_4^{b0}	24	28	—
	v_5	23	28	99.84
	v_5^{b0}	26	32	—

3.1 实验参数和指标

随机生成的应用是通过 TGFF 生成的, 使用这个软件需要输入以下的参数: (1) n, 任务的数量; (2) CCR, 任务之间信息传递的平均时间与平均执行时间的比值, 将会从{0.1, 0.5, 1, 5}中选取; (3) 任务的入度, 即任务的前驱数量; (4) 任务的出度, 即任务的后继数量; (5) $\overline{w}$, 任务的平均执行时间. 在实验中, 任务的数量会从 10 到 100 之间选取. 为了避免其他因素对实验结果造成影响, 所有随机生成的应用的最大入度和最大出度都设置为 4.

系统设置为由 4 个全连通的处理器组成的异构系统. 为了模拟处理器的故障, 处理器的故障率将会随机地从区间 $[6\times10^{-4}, 14\times10^{-4}]$ 中选取. 实验将会保证单个任务在不同处理器上的执行时间不同, 以保证处理器的异构性.

实验使用了两个评估指标如下: (1) 系统可靠性, 一个调度成功执行的概率; (2) PSS, 应用中的所有任务在截止期限前完成的概率, 可以通过计算调度成功的次数和总调度次数的比值得到. 在本文中, 只要应用调度后执行时间小于截止期限, 那么认为该次调度成功.

总调度次数在实验中被设置为 11, *PSS* 值越大, 证明算法越能保证系统的实时性.

3.2 随机生成应用

本节将会展示 4 组实验结果. 因为 DB-FTSA 和 FTSA-RSMI 都能够针对截止期限不断地调整任务复制数量来保证实时性, 所以截止期限的选取非常重要. 为了确保实验的合理性, 在使用两个算法调度前, 我们都会先使用 HEFT 算法调度一次. HEFT 算法已经被证明在异构系统中的实时性任务调度问题中表现优秀, 而且经常用来作为实时性调度算法的对比算法, 但是 HEFT 算法不具有容错能力, 因此在设置实验中的截止期限D时, 我们将会先用 HEFT 将所有任务调度一次, 得到应用的执行时间 *makespan*, 最后通过$D = makespan \times 1.5$计算得到.

图 4(a) 展示了在 *CCR*=5, 任务平均执行时间$\overline{w} = 15$以及任务数量从 10 增加到 100 的条件下, 分别使用 DB-FTSA 和 FTSA-RSMI 调度应用后得到的系统可靠性结果. 可以明显地看到无论任务数量如何变化, FTSA-RSMI 的系统可靠性都是大于 DB-FTSA 的. 当任务数量等于 100 时, FTSA-RSMI 的可靠性优于 DB-FTSA 50%.

图 4(b) 展示了在 *CCR*=1, 任务数量 *n* 为 50 的情况下, 不断增加处理器的故障率, 分别使用 DB-FTSA 和 FTSA-RSMI 调度应用后得到的系统可靠性结果. 随着故障率的变化, 两个算法得到的系统可靠性都下降了, 但是 FTSA-RSMI 始终高于 DB-FTSA.

图 4(c) 展示了在截止期限$D = 300$, 任务数量等于 50, 以及任务平均执行时间不断增加的条件下, 分别使用 DB-FTSA 和 FTSA-RSMI 调度应用后得到的系统可靠性结果. 随着任务平均执行时间不断增加, 系统的实时性约束不断严格, 两个算法得到的系统可靠性总体上都呈下降的趋势.

图 4(d) 展示了在任务数量等于 50, 任务平均执行时间等于 15, *CCR* 从 0.1 增加到 5 的条件下, 分别使用 DB-FTSA 和 FTSA-RSMI 调度应用得到的系统可靠性. 当 *CCR* 等于 5 时, 两个算法的系统可靠性差距达到最大为 50%.

3.3 真实应用

在这个实验中, 我们将会用 3 个真实世界的应用来测试算法的性能, 分别是高斯消元 (Gaussian elimination)、快速傅里叶变换 (fast Fourier transform) 和分子动力学代码 (molecular dynamics code). 这些应用的结构都是已知的, 所以我们只需要设置 *CCR* 和任务的平均执行时间$\overline{w}$.

在高斯消元中, *MS* 用来定义高斯消元问题中的系数矩阵的大小, 同时也被用于计算高斯消元应用中的任务数量, 用公式表示为式 (12):

$$n = \frac{MS^2 + MS - 2}{2} \tag{12}$$

在调度成功率方面, DB-FTSA 和 FTSA-RSMI 调度应用得到的调度成功率 *PSS* 均为 1. 图 5 展示了在任务平均执行时间$\overline{w} = 15$, *CCR*=1, 以及截止期限$D = MS \times (\overline{w} + \overline{w} \times CCR)$的情况下, 不断增加系数矩阵大小, 分别使用 DB-FTSA 和 FTSA-RSMI 调度高斯消元应用得到的可靠性结果. 从图中可以看出, 随着系数矩阵 *MS* 不断增加, FTSA-RSMI 的可靠性依然是高于 DB-FTSA 的.

在快速傅里叶变换实验中, 我们使用 *VS* 来表示快速傅里叶变换应用需要输入的参数. 应用的截止期限计算方式如式 (13):

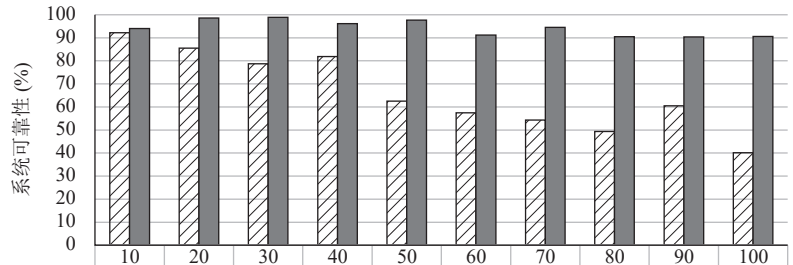
$$D = (2 \times \log_2 VS + 1) \times (\overline{w} + \overline{w} \times CCR) \tag{13}$$

实验结果表明 3 种算法的调度成功率 *PSS* 均为 1. 图 6 展示了在任务平均执行时间$\overline{w}$=15, *CCR*=1 的情况下, 不断增加 *VS*, 分别使用 DB-FTSA 和 FTSA-RSMI 调度快速傅里叶变换应用得到的可靠性对比. 从图 6 中可以看到当 *VS* 大于 32 时, 所有算法得到的可靠性都下降了. 但是当 *VS* 等于 32 时, FTSA-RSMI 得到的可靠性依然是最高的.

分子动力学代码拥有一个不规则的任务结构, 需要考虑两个参数 *CCR* 和任务平均执行时间$\overline{w}$.

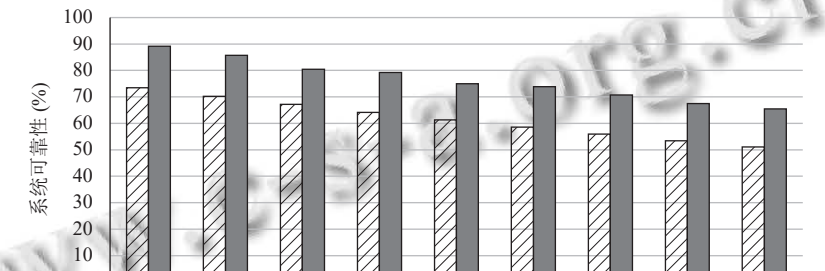
两种算法在这个实验中得到的 *PSS* 都是 1. 图 7 展示了分别使用 DB-FTSA 和 FTSA-RSMI 在任务平均执行时间$\overline{w} = 15$, 任务数量 *n*=40, 截止期限 *D*=500 的情况下, *CCR* 从 0.1 到 5, 调度分子动力学代码的系统可靠性结果. 当 *CCR*=5 时, DB-FTSA 得到的可靠性为 84.57%, 但是 FTSA-RSMI 得到的可靠性依然能够达到 98.21%.

对于调度 3 个真实世界的应用而言, DB-FTSA 和 FTSA-RSMI 都能有效的提升系统的可靠性, 但是当截止期限约束较为严格时, FTSA-RSMI 有明显的优势.



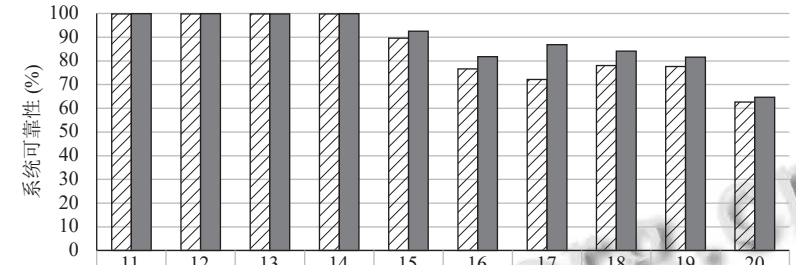
	10	20	30	40	50	60	70	80	90	100
DB-FTSA	92%	85.46%	78.71%	81.90%	62.49%	57.43%	54.34%	49.36%	60.51%	40.16%
FTSA-RSMI	94.00%	98.57%	98.90%	96.11%	97.65%	91.15%	94.48%	90.47%	90.35%	90.58%

(a) 任务数量



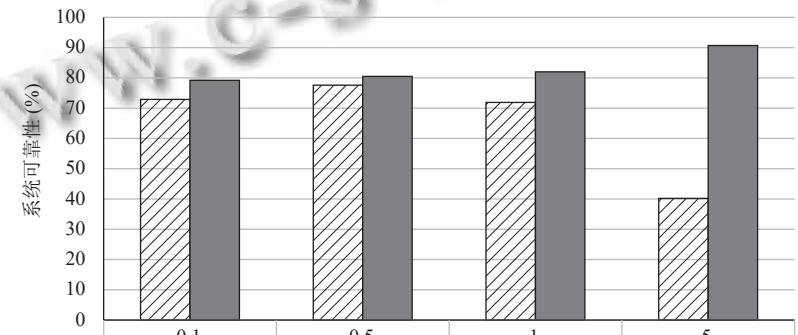
	6	7	8	9	10	11	12	13	14
DB-FTSA	73%	70.25%	67.14%	64.16%	61.30%	58.57%	55.95%	53.44%	51.03%
FTSA-RSMI	89.11%	85.76%	80.45%	79.21%	75.00%	73.86%	70.77%	67.45%	65.45%

(b) $\lambda\times10^{-4}$



	11	12	13	14	15	16	17	18	19	20
DB-FTSA	100%	99.83%	99.79%	99.85%	89.54%	76.60%	72.12%	78.07%	77.67%	62.66%
FTSA-RSMI	99.92%	99.89%	99.85%	99.90%	92.51%	81.77%	86.85%	84.14%	81.55%	64.70%

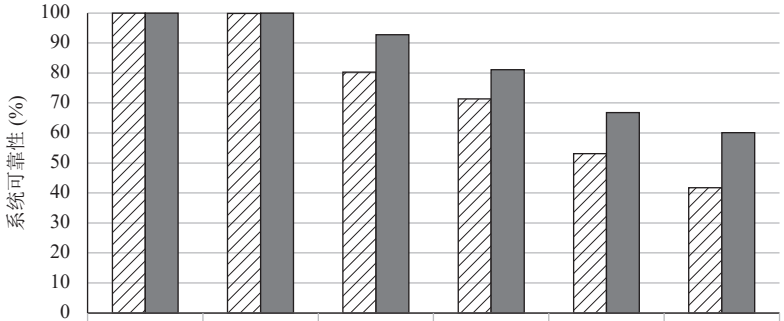
(c) 任务平均执行时间



	0.1	0.5	1	5
DB-FTSA	73%	77.59%	71.85%	40.17%
FTSA-RSMI	79.21%	80.49%	81.97%	90.68%

(d) *CCR*

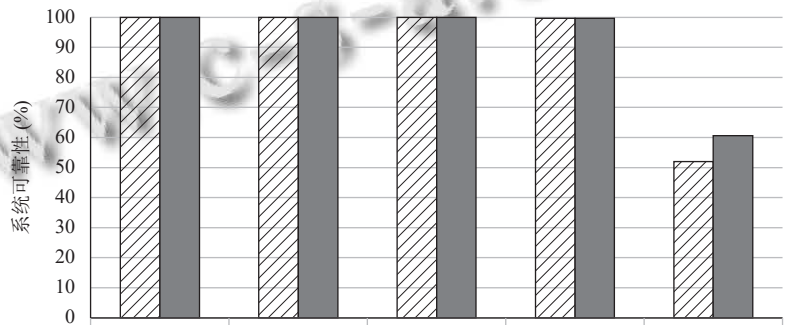
图 4　在任务数量等于 50, 任务平均执行时间等于 15, 不断增加 *CCR*, 2 种算法得到的可靠性对比



	5	8	11	14	17	20
DB-FTSA	99.99%	99.90%	80.31%	71.33%	53.09%	41.71%
FTSA-RSMI	99.99%	99.97%	92.80%	81.05%	66.82%	60.15%

MS

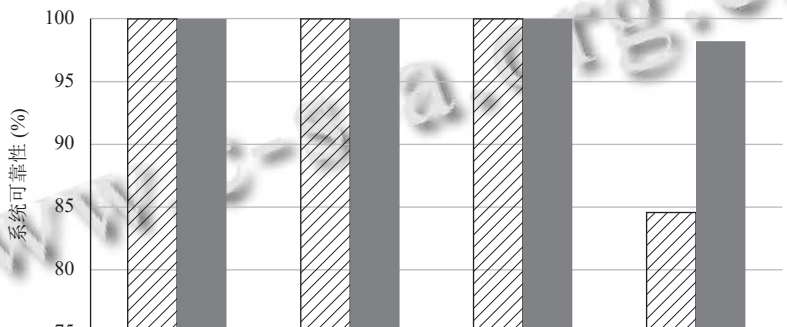
图 5 调度高斯消元应用的系统可靠性对比



	2	4	8	16	32
DB-FTSA	99.99%	99.99%	99.99%	99.62%	51.99%
FTSA-RSMI	99.99%	99.99%	99.99%	99.68%	60.62%

MS

图 6 调度快速傅里叶变换应用的系统可靠性对比



	0.1	0.5	1	5
DB-FTSA	99.99%	99.99%	99.99%	84.57%
FTSA-RSMI	99.99%	99.99%	99.99%	98.21%

CCR

图 7 调度分子动力学代码应用的系统可靠性对比

4 结论与展望

在本文中, 一个新的基于复制策略的容错调度算法 FTSA-RSMI 被提出. FTSA-RSMI 针对异构实时系统中的出现的暂时性错误, 确保实时并行应用能在截止期限前完成并且得到一个较高的系统可靠性. FTSA-RSMI 不断地增加对系统实时性影响最小的任务的复制次数, 从而提高系统的可靠性. 实验结果证明 FTSA-RSMI 在严格的截止期限的约束下依然能够保证比

DB-FTSA 更高的系统可靠性. 在未来研究实时性调度问题时, 我们会尽量减少系统中的任务冗余, 并且将系统的节能也纳入研究范畴.

参考文献

1 邓建波, 张立臣, 邓惠敏. 异构分布式系统混合型实时容错调度算法. 计算机科学, 2011, 38(3): 87–92, 102. [doi: 10.3969/j.issn.1002-137X.2011.03.019]

2 董崇杰, 陈俞强. 异构分布式系统动态实时容错调度启发式算法. 系统仿真学报, 2017, 29(5): 1132–1140. [doi: 10.16182/j.issn1004731x.joss.201705027]

3 刘云生, 张童, 张传富, 等. 异构分布式实时仿真系统的容错调度算法. 软件学报, 2006, 17(10): 2040–2047.

4 Chung YC, Ranka S. Applications and performance analysis of a compile-time optimization approach for list scheduling algorithms on distributed memory multiprocessors. Proceedings of the 1992 ACM/IEEE Conference on Supercomputing. Minneapolis: IEEE, 1992. 512–521.

5 Wu MY, Gajski DD. Hypertool: A programming aid for message-passing systems. IEEE Transactions on Parallel and Distributed Systems, 1990, 1(3): 330–343. [doi: 10.1109/71.80160]

6 Topcuoglu H, Hariri S, Wu MY. Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems, 2002, 13(3): 260–274. [doi: 10.1109/71.993206]

7 Kumar N, Mayank J, Mondal A. Reliability aware energy optimized scheduling of non-preemptive periodic real-time tasks on heterogeneous multiprocessor system. IEEE Transactions on Parallel and Distributed Systems, 2020, 31(4): 871–885. [doi: 10.1109/TPDS.2019.2950251]

8 Xie GQ, Zeng G, Chen YK, *et al*. Minimizing redundancy to satisfy reliability requirement for a parallel application on heterogeneous service-oriented systems. IEEE Transactions on Services Computing, 2020, 13(5): 871–886. [doi: 10.1109/TSC.2017.2665552]

9 Wang L, Siegel HJ, Roychowdhury VP. A genetic-algorithm-based approach for task matching and scheduling in heterogeneous computing environments. Proceedings of the Heterogeneous Computing Workshop, 1996. 72–85.

10 Ahmad W, Alam B. An efficient list scheduling algorithm with task duplication for scientific big data workflow in heterogeneous computing environments. Concurrency and Computation: Practice and Experience, 2021, 33(5): e5987.

11 韩莉. 实时系统工作流的能量感知容错算法 [博士学位论文]. 上海: 华东师范大学, 2020.

12 Roy A, Aydin H, Zhu DK. Energy-efficient fault tolerance for real-time tasks with precedence constraints on heterogeneous multicore systems. Proceedings of the 2019 10th International Green and Sustainable Computing Conference (IGSC). Alexandria: IEEE, 2019. 1–8.

13 Roy A, Aydin H, Zhu DK. Energy-aware standby-sparing on heterogeneous multicore systems. Proceedings of the 54th Annual Design Automation Conference 2017. Austin: ACM, 2017. 21.

14 Benoit A, Hakem M, Robert Y. Fault tolerant scheduling of precedence task graphs on heterogeneous platforms. Proceedings of the 2008 IEEE International Symposium on Parallel and Distributed Processing. Miami: IEEE, 2008. 1–8.

15 Wei MX, Liu J, Li T, *et al*. Fault-tolerant scheduling of real-time tasks on heterogeneous systems. Proceedings of the 2017 12th IEEE Conference on Industrial Electronics and Applications (ICIEA). Siem Reap: IEEE, 2017. 1006–1011.

16 Liu J, Wei MX, Hu W, *et al*. Task scheduling with fault-tolerance in real-time heterogeneous systems. Journal of Systems Architecture, 2018, 90: 23–33. [doi: 10.1016/j.sysarc.2018.08.007]

17 Gupta P, Sahoo PK, Veeravalli B. Dynamic fault tolerant scheduling with response time minimization for multiple failures in cloud. Journal of Parallel and Distributed Computing, 2021, 158: 80–93. [doi: 10.1016/j.jpdc.2021.07.019]

18 Shi L, Xu J, Wang LF, *et al*. Multijob associated task scheduling for cloud computing based on task duplication and insertion. Wireless Communications and Mobile Computing, 2021, 2021: 6631752.

19 Huang J, Li RF, An JY, *et al*. A DVFS-weakly dependent energy-efficient scheduling approach for deadline-constrained parallel applications on heterogeneous systems. IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems, 2021, 40(12): 2481–2494. [doi: 10.1109/TCAD.2021.3049688]

20 邓建波, 张立臣, 符利华. 一种基于负载均衡异构分布式系统的改进容错调度算法. 计算机应用研究, 2010, 27(7): 2479–2482.

21 Chen G, Guan N, Huang K, *et al*. Fault-tolerant real-time tasks scheduling with dynamic fault handling. Journal of Systems Architecture, 2020, 102: 101688. [doi: 10.1016/j.sysarc.2019.101688]

22 朱萍. 硬实时容错调度算法研究 [博士学位论文]. 武汉: 华中科技大学, 2011.

(校对责编: 牛欣悦)