容错并行算法的性能分析

杜云飞 唐玉华 杨学军

(国防科技大学计算机学院并行与分布处理国家重点实验室 长沙 410073)

摘 要 容错并行算法是一种应用级容错方法,它通过并行复算的方法实现快速的故障恢复。容错并行算法是在并行算法设计的基础上增加了容错设计部分,因此其性能评估必须考虑故障对程序性能的影响。研究了评估故障情况下容错并行算法性能的各种度量,建立了性能模型预测容错并行算法的期望执行时间,以此为基础评估了程序段的运行时间、数据保存开销、故障率以及并行复算加速比等系统参数对容错并行算法性能的影响。

关键词 容错并行算法,执行时间,加速比,效率

Performance Evaluation for Fault-tolerant Parallel Algorithm

DU Yun-fei TANG Yu-hua YANG Xue-jun

(National Laboratory for Paralleling and Distributed Processing, College of Computer, National University of Defense Technology, Changsha 410073, China)

Abstract The fault-tolerant parallel algorithm (FTPA) is an application-level technique for tolerating hardware failures. FTPA achieves fast failure recovery making use of parallel recomputing. How to deal with system failures is a concern in the design of FTPA. Thus, evaluating the performance of FTPA under system failures is necessary. In this study, we presented the performance metrics to evaluate the performance of FTPA and a model to predict the application completion time under system failures. Then, the influence of program section executing time, checkpointing cost, failure rate, and speedup of parallel recomputing on the performance of FTPA were evaluated.

Keywords Fault-tolerant parallel algorithm, Application completion time, Speedup, Efficiency

1 前言

传统并行算法的性能度量主要分为5类:执行时间、总并行开销、加速比、效率和成本,这些性能度量都是以并行程序的执行时间为基础[1]。随着大规模并行计算机系统规模扩大、系统可靠性降低,为保证运行在系统上的大规模并行应用能够正常完成,必须针对这些并行应用采用必要的容错技术。因此对容错技术的度量成为当前高性能领域的关键问题。

容错并行算法是一种面向大规模并行系统的容错技术。更好地理解故障对容错并行算法性能的影响,对于设计优化的容错并行算法是很重要的,因此,对容错并行算法的性能评估必须考虑系统故障对程序性能的影响。然而,现有的并行程序的性能度量只能用于评估无故障时的程序性能,不能评估有故障运行时大规模并行应用的性能。本文建立了性能模型来预测有故障运行时容错并行算法的执行时间,以此为基础完成了对容错并行算法的性能评估。

本文第2节分析了容错并行算法的开销来源;第3节给出了容错并行算法的3种性能度量:执行时间、加速比和效率;第4节对不同的系统参数对容错并行算法性能的影响进行了评估;最后给出总结。

2 容错并行算法概述

容错并行算法是能够在线检测故障并自容错的并行算法[2]。容错并行算法执行时,在数据保存点保存计算的中间状态,以保证故障时正确地复算;发生故障时,未发生故障的处理器通过在线的方式感知故障处理机的故障,并自动通过并行复算恢复故障处理器上的负载。并行复算是多个进程重新执行故障进程上的负载。

逻辑上,一个传统并行算法由若干个程序段构成,程序段是传统并行算法中的一段代码。不失一般性,假设一个并行算法由程序段 $S_0S_1\cdots S_n$ 构成。容错并行算法设计的基本思想是以程序段为基础,添加数据保存段、故障检测段和复算段构成相应的容错程序段。容错程序段结构如图1所示。

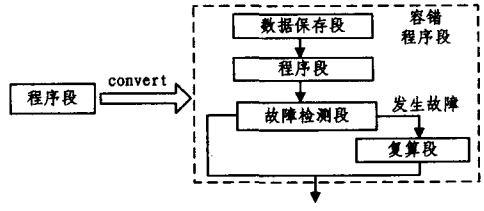

图1 容错程序段

到稿日期:2008-10-15 返修日期:2009-01-04 本文受国家自然科学基金项目(60621003,60633050和60873014)和国家863项目(2008AA01Z110)资助。

杜云飞 博士生,主要研究方向为高性能计算和容错,E-mail:forest80@163.com;唐玉华 教授,主要研究方向为高性能计算;杨学军 教授,博士生导师,主要研究方向为高性能计算、系统结构等。

为了支持正确的复算，需要在数据保存段中保存并行算法计算的中间状态，实际上是并行算法执行过程中的一组变量的值。

故障检测段通过查询系统故障状态向量 FV，在线感知哪个处理器发生故障。假设并行计算中有 N 个处理器参与计算，故障状态向量 FV 是一个 N 元组 $\langle F_0, F_1, \cdots, F_N \rangle$，$F_i$ 是第 i 个处理器上的故障序列，它记录了故障类型。

复算段的代码利用 SPMD 的编程方式实现。通过对原程序段的改造产生复算段的代码。复算段和原程序段在同一程序中实现，以实现自动的故障恢复。假设程序段 S_k 在故障进程 P_j 上的负载是 W_{Skj}，对应的复算段 R_{Sk} 在每个存活进程 P_i 上的负载是 W_{RSki}，则 $W_{*j} = \sum_{i \neq j} W_{RSki}$。

3 容错并行算法的开销来源

由于涉及故障的处理，容错并行算法的实际运行时间不但包括执行有效计算的时间，还包括执行容错方法的额外开销。对这些开销的分析是评估容错并行算法性能的关键。

除原有并行算法的运行时间外，容错并行算法还需要在数据保存段、故障检测段以及故障恢复段中花费时间。

数据保存时间：每个数据保存段中保存两类变量所花费的开销：活跃变量以及进程间引用-定值链中被定值的变量。这类开销与数据保存量、系统的 I/O 带宽以及处理故障的数目密切相关。数据保存量越小，系统 I/O 带宽越大时，存储数据的开销越小。处理单故障时，第二类变量可以存储在本地内存中，存储数据的速度取决于系统的访存带宽；处理多故障时，第二类变量必须保存在磁盘中，存储数据的速度取决于系统的 IO 带宽。由于 IO 带宽往往小于访存带宽，多故障时存储第二类变量的开销大于单故障时的开销。

故障检测时间：程序段结束前和程序段中通信语句前的故障检测语句所花费的开销。这类开销与并行应用的负载均衡性以及通信语句的类型相关。由于故障检测包含同步语义，在并行应用的负载不均衡时，这种开销可能比较高。另外，由于一部分通信语句本身已经包含了同步语句，在这些通信语句前的故障检测语句开销可能比较低。

故障恢复时间：出现故障后，从程序段的入口处至故障点的恢复开销。这类开销与恢复数据保存段所保存的数据的开销、恢复计算的工作量、并行复算的效率以及复算进程的数目相关。容错并行算法的故障恢复过程中采用了并行复算的方法。在恢复计算的工作量固定的情况下，如果并行复算效率随着进程数目的增加而增大，那么复算进程数目越多，并行复算时间越短；如果并行复算效率不能随着进程数目的增加而增大，那么复算进程数目和复算效率乘积最大时，并行复算时间是最短的。

对于同一个并行算法，用不同的方法设计出的容错并行算法会导致不同的开销。对每个容错并行算法建立一个性能特征来量化这些开销是很重要的。

4 容错并行算法的性能度量

系统故障对容错并行算法的性能有很大的影响。本节考虑故障情况下容错并行算法的性能度量。

4.1 执行时间

一般来说，系统故障是非确定性事件，难以使用确定性模型来描述。根据文献[3,4]系统故障发生模型服从参数为 λ_f 的泊松分布，系统的平均故障时间间隔为 $1/\lambda_f$。T_p 表示并行算法的运行时间，并行算法执行过程中发生 $K = T_p\lambda_f$ 次故障。容错并行算法中每个程序段的运行时间是 γ，程序段的数目为 $m = T_p/\gamma$，T_{si} 表示第 i 个程序段的数据保存，T_{di} 和 T_{ri} 分别表示故障检测时间以及并行复算的时间。复算段的数据恢复时间等于 T_{si}。容错并行算法的执行时间可以表示为

$$T = T_p + T_{s1} + T_{d1} + \cdots + T_{sm} + T_{dm} + T_{s1} + T_{r1} + \cdots + T_{sk} + T_{rk} \quad (1)$$

科学计算类的应用具有负载均衡、通信模式规整的特点[5-7]。同时，此类应用中广泛采用组通信进行进程间的交互，因此故障检测开销 T_{di} 很低。在后面的讨论中，认为每次故障检测的开销为常数 D。假设每个程序段中的数据保存开销为 T_w，则式(1)可以表示为

$$T = T_p + mT_w + mD + KT_w + T_{r1} + \cdots + T_{rk}$$

假设程序段入口和故障点之间的时间服从指数分布，并行复算的加速比为 S_{pr}，则并行复算的平均时间为

$$T_{pr} = \frac{1 - e^{-\lambda_f\gamma} - \lambda_f\gamma e^{-\lambda_f\gamma}}{\lambda_f(1 - e^{-\lambda_f\gamma})S_{pr}}$$

根据上述分析，容错并行算法的期望执行时间为

$$T_{ftpa} = E(T) = T_p + mT_w + mD + KT_w + KT_{pr} = T_p + \frac{T_p}{\gamma}T_w + \frac{T_p}{\gamma}D + T_p\lambda_f T_w + T_p\frac{1 - e^{-\lambda_f\gamma} - \lambda_f\gamma e^{-\lambda_f\gamma}}{(1 - e^{-\lambda_f\gamma})S_{pr}}$$

4.2 加速比

加速比能够直观地反映并行系统的性能，一直是并行计算机系统设计和应用的重要问题，如传统的 Amdahl[8]，Gustafson[9] 等加速比公式。然而，已有的加速比公式是用于评估无故障时并行程序的性能，因此给出了面向容错并行算法的加速比公式。

传统的加速比公式是单个进程上求解问题所花的时间 T_s 与用 p 个相同处理器并行计算机求解同一问题所花时间 T_q 之比。将容错并行算法的加速比定义为串行计算时间 T_s 与容错并行算法的期望执行时间 T_{ftpa} 之比，用符号 S_{ftpa} 表示。

$$S_{ftpa} = \frac{T_s}{T_{ftpa}} = \frac{T_s}{T_s/(T_p + \frac{T_p}{\gamma}T_w + \frac{T_p}{\gamma}D + T_p\lambda_f T_w + T_p\frac{1 - e^{-\lambda_f\gamma} - \lambda_f\gamma e^{-\lambda_f\gamma}}{(1 - e^{-\lambda_f\gamma})S_{pr}})} = \frac{S}{1 + \frac{1}{\gamma}T_w + \frac{1}{\gamma}D + \lambda_f T_w + \frac{1 - e^{-\lambda_f\gamma} - \lambda_f\gamma e^{-\lambda_f\gamma}}{(1 - e^{-\lambda_f\gamma})S_{pr}}}$$

4.3 效率

效率是处理器在并行程序运行期间利用率的度量。传统的效率定义是加速比与进程数目的比值。将容错并行算法的效率定义为容错并行算法的加速比与进程数目的比值，即

$$E_{ftpa} = \frac{S_{ftpa}}{p}$$

5 系统参数对容错并行算法性能的影响

使用本文第 2 节中的度量公式可以评估系统参数对容错

并行算法性能的影响。考虑下面7个系统参数:故障率λ_f、数据保存开销T_w、进程数目p、并行程序运行时间T_p、程序段的运行时间γ、并行程序加速比S以及并行复算加速比S_{pr}。由于故障检测的开销D较小,不考虑故障检测的开销对性能的影响。每次只考虑两个参数的变化对性能的影响,其它的参数保持默认值。每个参数的默认值为

- 故障率$\lambda_f=(1/64)$h
- 数据保存开销$T_w=0.1$h
- 进程数目$=128$
- 并行程序运行时间$T_p=128$h
- 程序段的运行时间$\gamma=4$h
- 并行程序的加速比$S=60$
- 并行复算加速比$S_{pr}=8$

5.1 程序段的运行时间对性能的影响

图2给出了程序段运行时间在并行复算加速比不同的情况下对容错并行算法性能的影响。从图中可以看出,对于一个给定的并行复算加速比,容错并行算法的期望执行时间并不能随着程序段运行时间的增大而减小,存在一个最优的程序段运行时间。例如,在并行复算加速比为1时,程序段运行时间为4h,容错并行算法的期望执行时间最低,加速比和效率达到最大值。在并行复算加速比为4时,程序段运行时间为8h,容错并行算法的期望执行时间最低,加速比和效率达到最大值。程序段运行时间较长时,增大并行复算的加速比可以有效减少容错并行算法的期望运行时间,原因是此时并行复算的时间比较长。

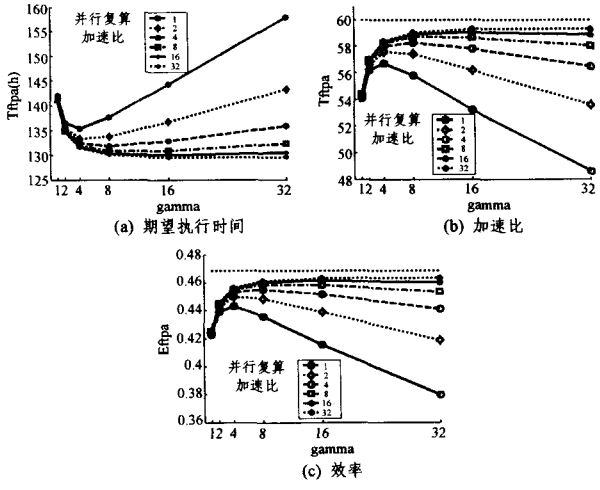

图2 程序段运行时间和并行复算加速比不同情况下的性能度量

5.2 数据保存开销对性能的影响

图3给出了数据保存开销在并行复算加速比不同的情况下对容错并行算法性能的影响。容错并行算法的期望执行时间随着数据保存开销的降低而减小,加速比和效率随之而增大。从图3可以看出,数据保存开销比并行复算加速比对于容错并行算法的性能影响更大。例如,在数据保存开销为0.2h和并行复算加速比为16时,如果将数据保存开销降至0.1h,容错并行算法的期望执行时间从135.05h降至131.65h。如果将并行复算加速比降至32时,容错并行算法的期望执行时间仅降至134.92h。

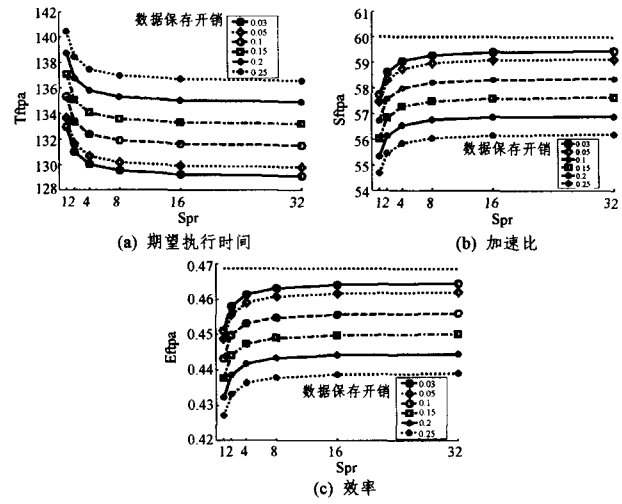

图3 数据保存开销和并行复算加速比不同情况下的性能度量

5.3 故障率对性能的影响

图4给出了故障率在程序段运行时间不同的情况下对容错并行算法性能的影响。从图中可以看出,对于一个给定的程序段运行时间,容错并行算法的期望执行时间随着故障率的增大而增大,加速比和效率随之而减小。

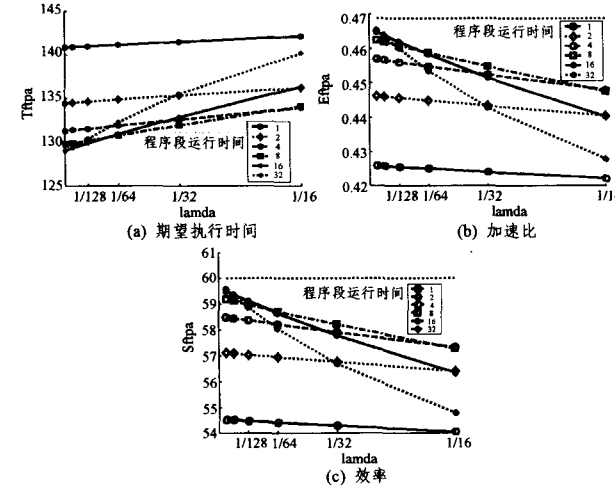

图4 故障率和程序段运行时间不同情况下的性能度量

5.4 并行复算加速比对性能的影响

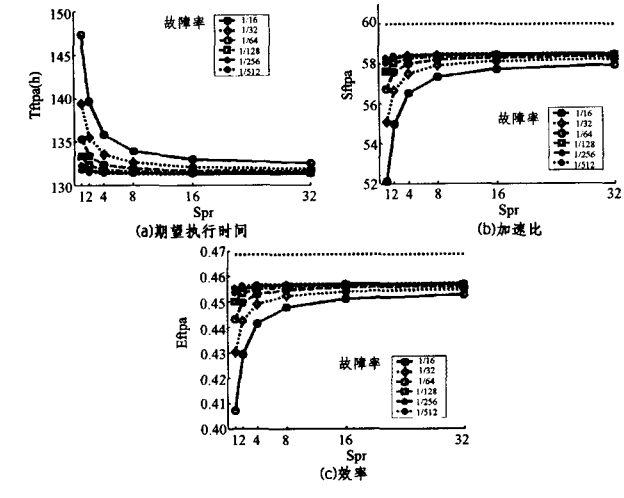

图5 故障率和并行复算加速比不同情况下的性能度量

图 5 给出了并行复算加速比在故障率不同的情况下对容错并行算法性能的影响。容错并行算法的期望执行时间随着并行复算加速比的增大而减小，加速比和效率随之而增大。故障率较高时，增大并行复算的加速比可以有效提高容错并行算法的性能。然而，故障率较低时，并行复算的加速比对容错并行算法性能的影响也较小。

结束语 传统的并行算法性能度量用于评估无故障时并行程序的性能。本文建立了考虑系统故障情况下的性能模型来预测容错并行算法的完成时间，以此为基础评估了程序段的运行时间、数据保存开销、故障率以及并行复算加速比等系统参数对容错并行算法性能的影响。程序段的选择是个优化设计问题，存在最优程序段运行时间；数据保存开销对容错并行算法的影响较大，它比并行复算加速比对容错并行算法性能的影响更大，数据保存开销越低，容错并行算法的性能越好；故障率越低，系统发生故障的概率越低，容错并行算法的性能越好；并行复算加速比的增大可以提高容错并行算法的性能，尤其是对于故障率较高以及程序段运行时间较长的容错并行算法，增大并行复算加速比可以有效提高容错并行算法的性能。

参 考 文 献

[1] Grama A，Gupta A，Karypis G，et al. Introduction to Parallel Computing，Second Edition[M]. Addison Wesley，January 2003

[2] Yang Xuejun，Du Yunfei，Wang Panfeng，et al. FTPA：Supporting Fault Tolerant Parallel Computing Through Parallel Recomputing[J]. IEEE Transactions on Parallel and Distributed Systems (Accepted)

[3] Duda A. The Effects of Checkpointing on Program Execution Time[J]. Information Processing Letters，1983，16：221-229

[4] Young J W. A First Order Approximation to the Optimal Checkpoint Interval[J]. Comm. ACM，1974，17(9)：530-531

[5] Kim JunSeong，Lilja D J. Characterization of Communication Patterns in Message-Passing Parallel Scientific Application Programs[C]// Proceedings of the Second International Workshop on Network-based Parallel Computing：Communication，Architecture，and Applications. January 1998：202-216

[6] Vetter J S. Communication Characteristics of Large-scale Scientific Applications for Contemporary Cluster Architectures[C]// International Parallel and Distributed Processing Symposium. 2002

[7] Zamani R，Afsahi A. Communication Characteristics of Message-Passing Scientific and Engineering Applications[C]// Proceedings of the 17th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS '2005). Phoenix，AZ，USA，November 2005：644-649

[8] Amdahl G M. Validity of the single-processor approach to achieving large scale computing capabilities[J]. AFIPS Conference Proceedings，1967，4(30)：483-485

[9] Gustafson J. Reevaluating Amdahl's Law[J]. Communication of ACM，1988，31(5)：532-533

(上接第 233 页)

经过 *ComPatterns* 函数处理的矩阵 D_n 包含了日志中所有的活动间直接依赖关系，可以直接通过 D_n 生成频繁模式，由有向图 $G=(V,F)$ 表示，其中 $V=AS(C_n)$ 代表有向图的节点集合 $(a_1,a_2,\cdots,a_n)$；F 是有向图的边集。对于矩阵 D_n 元素 $d_{ij}=1$，则工作流模型中存在一条从有向边 a_i 指向 a_j；反之，$d_{ij}=-1$ 存在一条从有向边 a_j 指向 a_i。

最后根据直接依赖矩阵 D_3'，D'_4 得到最终的工作流频繁模式，如图 2 所示。

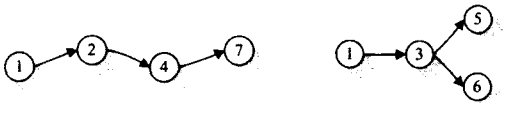

图 2 最终工作流频繁模式

结束语 工作流频繁模式包含了丰富的信息，可以为流程动态变化问题的解决提供基础。同时，可以为工作流管理系统中重要的商业决策、风险预测等提供支持，也为工作流模型优化提供依据。利用日志信息挖掘工作流频繁模式，是近年来新出现的研究领域。本文提出的算法与其他工作流模式挖掘方法相比，能够处理具有串、并行关系的工作流模式，更具优越性，在工作流模型中，活动间除串、并行关系外还存在循环关系。本算法在处理活动间并行循环关系时存在不足，不能完全处理循环关系，这是今后工作的主要方向。

参 考 文 献

[1] 范玉顺. 工作流管理技术基础[M]. 北京：清华大学出版社，2001：20-35

[2] Agrawal R，Imielinski T，Swami A. Mining association rules between sets of items in large databases [C]// Proceedings of the ACM SIGMOD International Conference Management of Date. Washington，1993：207-216

[3] Agrawal R，Srikant R. Mining sequential patterns[C]// Proceedings of International Conference on Data Engineering. Taipei，Taiwan，1995：135-139

[4] Srikant R. Agrawal R. Mining sequential patterns：generalizations and performance improvements[C]// Proceedings of the 5th International Conference on Extending Database Technology (EDBT). Avignon，France，1996：117-133

[5] Mannila H，Toivonen H，Verkamo A I. Discovering frequent episodes in sequences[C]// Proceedings of the First International Conference on Knowledge Discovery and Data Mining. Avignon，France，1995：194-204

[6] Sadiq W，Orlowska M E. Analyzing Process Models Using Graph Reduction Techniques[J]. Information Systems，2000，25(2)：117-134

[7] Sadiq S，Orlowska M. On correctness issues in conceptual modeling of workflows[C]// Proceedings of the 5th Euopean Conference on Information Systems. Cork，Ireland，1997：19-21