软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2012,23(4):1022−1035 [doi: 10.3724/SP.J.1001.2012.04011]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

大规模 MPI 并行计算的可扩展三模冗余容错机制*

王之元+, 杨学军, 周 云

(国防科学技术大学 计算机学院,湖南 长沙 410073)

Scalable Triple Modular Redundancy Fault Tolerance Mechanism for MPI-Oriented Large Scale Parallel Computing

WANG Zhi-Yuan+, YANG Xue-Jun, ZHOU Yun

(College of Computer, National University of Defense Technology, Changsha 410073, China)

+ Corresponding author: E-mail: wang_zhiyuan17@163.com

Wang ZY, Yang XJ, Zhou Y. Scalable triple modular redundancy fault tolerance mechanism for MPI-oriented large scale parallel computing. *Journal of Software*, 2012,23(4):1022−1035. http://www.jos.org.cn/1000-9825/4011.htm

Abstract: The scale-up of system brings improvement in performance as well as reliability degradation, so there is a need to apply some fault tolerance mechanism to tolerate hardware failure or recover data. Currently, the popular fault tolerance mechanisms, such as Checkpoint/Restart and *N*-modular redundancy, all need additional overhead, which limits the scalability of parallel computing to some extent. Therefore, it is very important to develop scalable fault tolerance mechanisms for increasingly high performance supercomputing. This paper takes triple modular redundancy (TMR) as an example, describes the implementation of TMR on large-scale MPI parallel computing, and argues that traditional TMR fault-tolerant mechanism limits the scalability of parallel computing. To solve these practical problems, the paper proposes the scalable triple modular redundancy (STMR), and verifies the validity and scalability of it. STMR can not only handle the fail-stop failures that are traditionally handled by Checkpoint/Restart, but can also deal with most of data errors not perceived directly by the hardware. Finally, the study conducts the simulation using the system parameters of BlueGene/L, which shows the scalability change of parallel computing with the TMR and the STMR respectively when the system size increases. The results further validate STMR position as scalable fault-tolerant mechanism.
Key words: fault tolerance mechanism; scalability; triple modular redundancy; large scale parallel computing; MPI

摘 要: 随着系统规模的扩大,并行计算的性能不断提高,但可靠性却也在不断下降,因此需要采用某种容错机制来容忍或恢复硬件故障和数据错误.目前常用的容错机制 Checkpoint/Restart 和多模冗余均引入了额外的开销,这些开销均在某种程度上制约了并行计算的可扩展性.因此,在高性能计算需求不断增长的今天,可扩展容错机制的设计显得尤为迫切和重要.以三模冗余(triple modular redundancy,简称 TMR)为典型案例,描述了传统 TMR 在大规模 MPI 并行计算上的实现方法,分析了该机制所面临的实际问题,进而指出传统 TMR 制约了并行计算的扩展.根据该技术

* 基金项目: 国家自然科学基金(61003081, 61003087, 60921062)
收稿时间: 2010-10-08; 修改时间: 2011-01-20; 定稿时间: 2011-03-08

© 中国科学院软件研究所 http://www.jos.org.cn

所面临的问题,设计了可扩展三模冗余(scalable triple modular redundancy,简称 STMR),并进一步验证了其有效性和可扩展性.该机制不仅能够处理 Checkpoint/Restart 针对的 fail-stop 故障,还能够解决绝大部分硬件不能直接感知的数据错误.最后,借用 BlueGene/L 的系统参数进行模拟,预测当系统规模增大时,在分别采用 TMR 和 STMR 的情况下并行计算可扩展性的变化,结果进一步验证了 STMR 是可扩展的容错机制.
关键词: 容错机制;可扩展性;三模冗余;大规模并行计算;MPI
中图法分类号: TP302 **文献标识码**: A

随着现代高科技的蓬勃发展,实际应用中需要越来越高的计算性能.并行计算是实现高性能计算的主要技术手段.近十几年来,并行计算规模不断扩大,计算机的性能也不断提升.但在系统规模扩展的同时,系统出现硬件故障的可能性也随之增加,即系统可靠性越来越低,已经成为并行计算向更大规模扩展的主要制约因素之一.例如,根据 Los Alamos 国家实验室 1996 年~2005 年收集的 22 个高性能计算系统发生故障的数据,分析得到单结点故障率为 1/(512 小时)[1].对于拥有几万甚至几十万处理器或核的大规模并行计算系统,平均故障发生时间(mean time to failure,简称 MTTF)会降低到小时甚至分钟的量级[2].而目前许多大规模并行计算运行一次都要几个星期甚至几个月,在这种情况下,系统的平均故障间隔时间(mean time between failures,简称 MTBF)就会远低于大规模并行计算系统的运行时间,从而导致系统无法正常运行或得到正确结果.大规模并行计算系统需要采用必要的容错机制,容忍系统故障、提高系统可靠性以保证程序的正确运行.

Checkpoint/Restart(C&R)和多模冗余(*N*-modular redundancy,简称 NMR)[3−6]是目前常用的两种容错机制.C&R 需要周期性地将系统运行状态保存到系统的可靠存储器(硬盘)中,当系统出现故障时,程序被回滚到最近的一次检查点处重新计算.目前,该方法已广泛应用于大规模并行计算中,它能够有效地处理 fail-stop 故障,在实现过程中,除了消耗额外的存储设备外,其他额外的硬件资源消耗较小.三模冗余(triple modular redundancy,简称 TMR)[7]是 NMR 最常用的实现机制之一,它需要定期或不定期地比较冗余作业中的数据进行检错与容错.该机制不仅能够处理 Checkpoint/Restart 针对的 fail-stop 故障,还能够解决绝大部分硬件不能直接感知的数据错误[5],但由于其实现需要消耗大量的硬件资源,因而未在大规模并行计算中加以使用.

显然,上述两种容错机制在实现过程中均引入了额外的开销,但是随着系统规模的扩大,这部分开销是否也随之增大、是否限制了并行计算的可扩展性以及是否有新的容错机制能够避免对可扩展性的制约,这些问题都亟待加以研究和解决.

本文以 TMR 为典型案例进行讨论,首先分析了采用传统 TMR 的并行计算运行于 mesh 网络拓扑结构系统上的额外开销,从而得到传统 TMR 限制并行计算可扩展性的根本原因,并由此设计出了相应的解决办法,进而提出可扩展三模冗余(scalable triple modular redundancy,简称 STMR)容错机制.主要创新点表现在:

- 提出了采用 TMR 的并行计算可扩展性的度量模型和 TMR 可扩展性的定义,并揭示了 TMR 制约并行计算可扩展性的主要因素;
- 提出了可扩展性制约因素相应的解决技术,进而建立了 STMR 容错机制框架,并理论验证了其有效性和可扩展性;
- 实现了 STMR 容错机制框架中的各项关键技术;
- 实验模拟了在不同网络特征的系统上,应用于不同可扩展特性程序的 STMR 具有良好的可扩展性.

本文第 1 节定义 TMR 可扩展性的分类,通过对典型案例的分析判断传统 TMR 的类别,并揭示传统 TMR 制约可扩展性的主要因素.针对这些因素,第 2 节提出相应的解决办法,建立 STMR 容错机制实现框架,并验证其有效性和可扩展性.第 3 节针对 STMR 实现框架中的各项技术,给出具体的实现手段.第 4 节通过模拟实验进一步验证 STMR 具有良好的可扩展性.第 5 节介绍现有常用的容错机制以及 MPI 平台容错方面的相关工作.最后,第 6 节总结全文并做出展望.

© 中国科学院软件研究所 http://www.jos.org.cn

1 传统 TMR 的可扩展性

加速比是当前度量并行计算可扩展性的常用指标之一,本节首先基于采用 TMR 的并行计算的加速比模型,给出 TMR 的可扩展性的分类;其次,通过典型案例来分析传统 TMR 的可扩展性,最终揭示其制约可扩展性的主要因素并寻求解决方法.

1.1 TMR的可扩展性的分类

传统加速比 S 的公式为

$$S=\frac{T_S}{T_P}=\frac{T_S}{T_P'+T}=\frac{P}{1+T/T_P'} \tag{1}$$

其中,T_S 为程序 G 的串行版本在该系统单结点上的串行执行时间;P 为并行系统的结点数;T_P 为未引入 TMR 时,程序 G 的并行版本在 P 个结点的系统上的实际执行时间.显然,T_P 包含了并行计算开销 T_P' (某个常数)以及除 T_P' 以外的其他开销 T(如未被隐藏的通信开销等).当系统规模足够大时,并行程序 G 的串行部分执行开销可忽略不计,于是满足 $T_S=PT_P'$.

根据上述加速比模型,并行程序 G 采用 TMR 情况下的加速比模型(简称 R 加速比)S^R 定义为

$$S^R=\frac{T_S}{T_P^R}=\frac{T_S}{T_P+T_R}=\frac{T_S}{T_P'+T+T_R}=\frac{P}{1+T/T_P'+T_R/T_P'}=\frac{S}{1+T_R/T_P} \tag{2}$$

其中,T_P^R 为采用 TMR 的并行程序 G 的总执行时间,T_R 为 TMR 的开销.随着系统规模的增长,并行程序 G 表现出不同的可扩展性特征.首先对并行程序 G 的可扩展性进行分类.

定义 1. 并行程序 G 在具有 P 个结点的系统运行时,若 $S=\Theta(P)$,则称 G 为可扩展并行程序;若 $\Theta(1)\prec S\prec P$,则称 G 为弱可扩展并行程序;若 $S=\Theta(1)$,则称 G 为不可扩展并行程序.其中,$\Theta(\cdot)$**称为紧致界,符号 $\prec$***称为“弱于”.

由上述定义 1,本文给出 TMR 的可扩展性分类如下.

定义 2. 对于可扩展并行程序 G,当其采用 TMR 机制后,在具有 P 个结点的系统运行时,若 $\lim S^R\neq\infty$,则 TMR 为不可扩展的;若 $\lim S^R=\infty$ 且 $S^R\prec S$,则 TMR 为弱可扩展的;若 $S^R=\Theta(S)$,则 TMR 为可扩展的.记为可扩展三模冗余(scalable triple modular redundancy,简称 STMR)容错机制.

1.2 传统TMR可扩展性分析

Mesh 是当前常见的网络拓扑结构之一,本节以 P 个结点的 2D-mesh($\sqrt{P}\times\sqrt{P}$)系统为典型案例,分析传统 TMR 容错机制在该系统上实现的开销,从而判断传统 TMR 的可扩展性.假设该并行系统每个结点对应一个单核处理器,每个单核处理器运行作业中的一个进程.

传统 TMR 的容错方法是将给定程序 G 复制 3 份,用完全相同的 3 个作业(称为作业 I、作业 II 和作业 III)分别实现.根据现在典型的超级并行计算机作业管理的规则,3 个作业将系统分为 3 块区域,每块区域内相对位置相同的结点上运行相同的进程,称这样的结点为对应结点.例如图 1 中,N_i, N_i' 和 N_i'' 互为对应结点,且它们上面分别运行的进程 P_i, P_i' 和 P_i'' 互相冗余.

不失一般性,在运行过程中,假设作业 II 和作业 III 中的所有结点需定期或不定期地通过向作业 I 中的对应结点发送其内存中的数据,并在作业 I 中进行比较,以检测错误的发生.若检测出某个结点出错,则由编号较低的其他作业区域内的对应结点将内存中的数据发送到该出错结点的内存中并进行赋值.对应结点比较时,两个或 3 个结点的数据均不同或者两个或 3 个结点中的数据均出错且错的结果一样的情况为小概率事件,不予考虑.图 1 分别示意了 N_i, N_i' 和 N_i'' 对应结点间的数据比较通信以及若检测到 N_j' 结点出错,N_j 对其赋值通信的过程.

由此可见,传统 TMR 容错机制的开销 T_R 主要包括 4 部分:

** 若函数 $g(n)$记为$\Theta(g(n))$,则$\Theta(g(n))=\{f(n)$:存在正常数 c_1,c_2 和 n_0,使得$\forall n\geq n_0$,有 $0\leqslant c_1g(n)\leqslant f(n)\leqslant c_2g(n)\}$.

*** $f(n)\prec g(n)$表示 $\lim f(n)/g(n)=0$,$f(n)\preceq g(n)$表示 $\lim f(n)/g(n)=C$,C 为非负常数.类似地,$f(n)\succ g(n)$表示 $\lim g(n)/f(n)=0$,$f(n)\succeq g(n)$表示 $\lim g(n)/f(n)=Z$,Z 为非负常数.

© 中国科学院软件研究所 http://www.jos.org.cn

- 一是 3 个作业间进行多次数据比较时通信的开销,本文称其为检错通信开销;
- 二是检测出错误进行赋值处理时通信的开销,本文称其为容错通信开销;
- 三是在作业 I 中数据比较的开销;
- 四是在出错结点上的赋值操作开销.

由于进行比较的数据和赋值的数据量不随着系统规模的增长而变化,因此第 3 部分和第 4 部分的数据比较开销和赋值操作开销均不影响 TMR 的可扩展性.下面我们主要分析检错通信开销和容错通信开销这两部分.

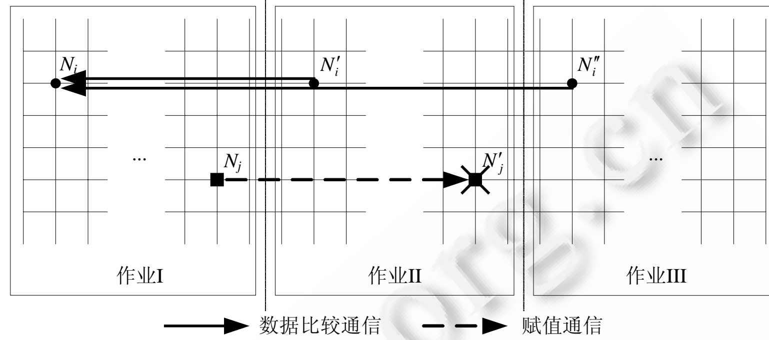


Fig.1 Implementation of TMR on 2D-mesh topology

图 1 TMR 在 2D-mesh 网络拓扑结构上的实现

本文考虑虫孔维序路由算法[8]指导的网络通信,则点对点通信开销 T'_C 为

$$T'_C = t_O + \frac{D}{B} + h \cdot t_R \tag{3}$$

其中,t_O 为初始化和结束传输的时间开销,t_R 表示单个路由器延迟,二者均为常量;D 为传输数据量;B 为传输带宽;h 为网络跳步数.

- 检错通信开销

传统 TMR 的检错通信可抽象为跨作业的点对点通信,因此由公式(3)可知,检错通信开销主要受到 3 个因素的制约:其一为跨越作业区域的带宽,如图 1 所示,跨作业区域的带宽为 2D-mesh 网络的二分带宽,此时,$B=b\cdot\sqrt{P}$,其中,b 为单链路带宽;其二为较远的通信距离,作业 II 和作业 III 向作业 I 传输数据的网络平均跳步数为 $h=\sqrt{P}/2$;第三为巨大的通信数据量,每次的通信数据量为两个作业区域内所有结点的内存容量,此时,$D=d\cdot(2P/3)$,其中,d 为作业 II 和作业 III 中单个结点的内存容量.

假设进行了 M 次数据比较,且每次的数据比较过程完全相同,因此,TMR 用于数据比较的总通信开销为

$$T_C = M \cdot T'_C = M \cdot t_O + M \cdot \frac{D}{B} + M \cdot h \cdot t_R = t_O M + \frac{2dM\sqrt{P}}{3b} + \frac{t_R M\sqrt{P}}{2} = \Theta(\sqrt{P}) \tag{4}$$

由上式可知,$\sqrt{P}$ 为 T_C 的紧致界.

- 容错通信开销

容错通信开销主要受到对应结点间的数据传输距离(网络跳步数)的影响.如图 1 所示,任意两对应结点间的网络跳步数为 $\sqrt{P}/3$ 或 $2\sqrt{P}/3$.由公式(3)可知,则两对应结点间用于赋值的通信开销 T'_G 满足:

$$t_O + \frac{d}{b} + \frac{t_R\sqrt{P}}{3} \leqslant T'_G \leqslant t_O + \frac{d}{b} + \frac{2t_R\sqrt{P}}{3} \tag{5}$$

假设系统在 M 次数据比较中有 F 次检测到错误,则 TMR 赋值处理的总通信开销 $T_G=F\cdot T'_G$,则有

$$t_O F + \frac{d}{b}F + \frac{t_R F\sqrt{P}}{3} \leqslant T_G \leqslant t_O F + \frac{d}{b}F + \frac{2t_R F\sqrt{P}}{3} \tag{6}$$

此时,$T_G=\Theta(\sqrt{P})$,$\sqrt{P}$ 也为 T_G 的紧致界.

因 $T_R \geqslant T_C+T_G, T_C=\Theta(\sqrt{P}), T_G=\Theta(\sqrt{P})$,则 $T_R \succeq \sqrt{P}$.我们考虑可扩展并行程序 G,由定义 1,则有 $S=\Theta(P)$,由公式(1)得到 $T/T_P'=\Theta(1)$.因此,公式(2)的 R 加速比为

$$S^R=\frac{P}{1+T/T_P'+T_R/T_P'} \preceq \frac{P}{1+\sqrt{P}}=\Theta(\sqrt{P}).$$

上式说明,传统 TMR 在 2D-mesh 系统上运行时为弱可扩展的.实际上,若考虑通信过程中存在资源冲突和网络拥塞等情况,甚至有可能出现 $\lim S^R \neq \infty$的情况.此时,TMR 为不可扩展的.

另外,对于当前大规模并行系统常用的其他网络拓扑结构,如 torus 和胖树等,根据上述 TMR 开销分析,采用传统 TMR 同样存在跨作业区域通信、较大的网络传输距离以及巨大的通信数据量等问题,因而传统 TMR 在这些网络拓扑结构下也不是可扩展的.由此可见,传统 TMR 不能满足大规模并行计算可扩展性的需要,必须进行重新设计.

1.3 TMR可扩展制约因素分析

对可扩展并行程序 G,若要实现 STMR,根据定义 2 以及公式(1)、公式(2),必须满足 $T_R=\Theta(1)$.然而传统 TMR 在 2D-mesh 上实现时,随着系统规模的增大,TMR 开销的主要部分,即检错通信开销和容错通信开销,均以 $\sqrt{P}$ 的函数形式增长.可见,二者均制约了 TMR 的可扩展性,因此需要同时针对检错过程和容错过程中面临的通信问题进行设计.

根据第 1.2 节的分析,传输两个作业区域内所有结点的内存数据,跨作业区域的通信带宽和数据传输距离造成的通信开销随着系统规模的增长,可通过以下途径解决:

- 通信局部化,以避免数据比较时受到跨作业区域通信带宽的影响,从而在一定程度上减少了数据的传输距离;
- 选择合适的数据比较点和比较数据,以减少比较通信的数据量;
- 采用标记技术处理单结点出错的情况,以避免赋值处理的通信开销.

前两项处理是为减少 TMR 检错中的数据比较开销而设计的,第 3 项是针对容错赋值的处理开销而设计的.

2 STMR 容错机制设计

2.1 数据比较的通信局部化

STMR 容错机制通过数据比较实现错误检测,与 TMR 不同,STMR 是将 3 个作业中对应结点上的进程作为一个整体,称它们为冗余进程簇,并以此为基础设计相应的实现技术.

定义 3. 并行 MPI 程序的任意一个进程 P_i 都存在两个冗余的副本 P_i' 和 P_i'',P_i 与其对应的冗余副本 P_i' 和 P_i'' 进行完全相同的运算,发送和接收完全相同的消息.称由这样 3 个进程构成的整体为冗余进程簇,记为 (P_i, P_i', P_i'').其中,P_i 称为元进程,P_i' 和 P_i'' 均称为影子进程.

根据定义 3,原先的 3 个作业之间数据比较的通信即为冗余进程簇内的通信.为了在数据比较时避免跨作业区域的通信,考虑在邻近结点上分配冗余进程簇.

定义 4. MPI 程序的多个进程进行结点分配时,将冗余进程簇分配到相互邻近的 3 个结点上运行,其中任意两结点间通信网络跳步数不超过 2,这种结点分配方式称为冗余进程簇结点分配技术.

例如,经过冗余进程簇结点分配后,图 1 中 3 个作业区域中的对应结点上运行的进程 P_i, P_i' 和 P_i'' 被划分到相互邻近的结点上运行,所在位置如图 2 所示.因此,冗余进程簇内(P_i, P_i', P_i'')数据比较时,结点 N_{i+1} 和 N_{i+2} 仅需将比较数据发送到 N_i 结点,使得数据比较不再受限于网络带宽的制约,且传输距离不超过 2 个跳步.需要说明的是,邻近的 3 个结点 N_i, N_{i+1} 和 N_{i+2} 中也可将 N_{i+1} 或 N_{i+2} 定义为元进程 P_i 运行的结点,若 N_{i+1} 上运行的是元进程,此时由结点 N_i 和 N_{i+2} 向 N_{i+1} 发送比较数据,传输距离不超过 1 个跳步.为清楚起见,下文仅以图 2 中示意的冗余进程簇分配到结点的位置为例进行说明和技术实现.

© 中国科学院软件研究所 http://www.jos.org.cn

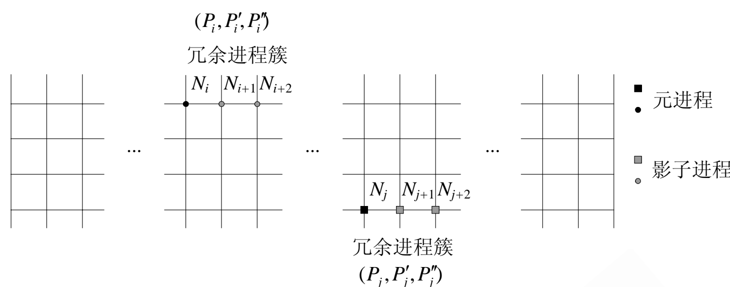


Fig.2 Node assignment technique for redundant process cluster

图 2 冗余进程簇结点分配技术

2.2 数据比较点及数据选择

MPI 中的各进程因故障引起的错误只能通过通信操作进行传播,因此选择在通信语句进行数据比较,检查流出或流入该进程数据的正确性,可实现各进程间的故障隔离.同时,选择在通信点进行数据比较,也有效地降低了数据比较的次数.

通信点包括发送数据语句和接收数据语句,若是在接收端进行数据比较,则 3 条通信链路分别传输冗余进程簇发送的消息,如图 3(a)所示.这种发送方式虽然能够检查接收到的数据的正确性,但却造成了通信的冗余,增加了通信网络的负担和网络拥塞.因此在图 3(b)中,在发送端进行数据比较后,仅在各元进程间进行通信,能够有效地解决这一问题.

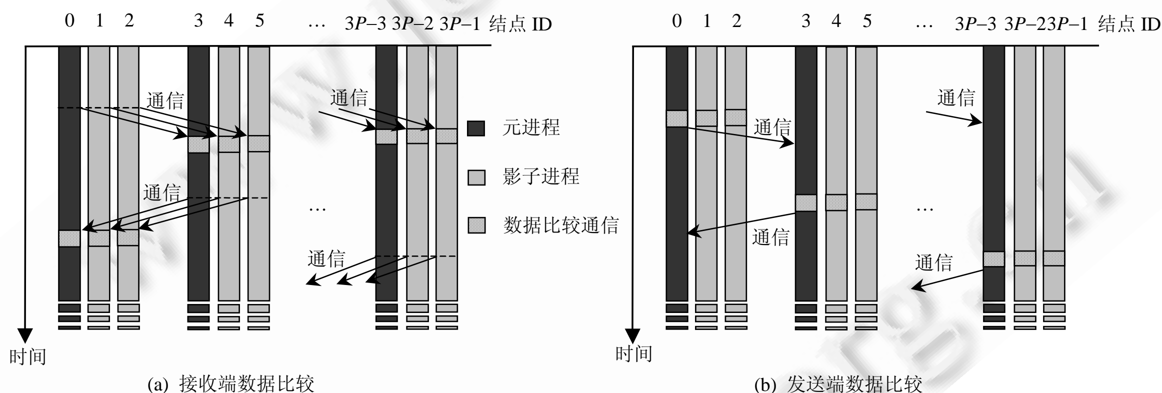


Fig.3 Data of communication comparison

图 3 通信数据比较

定义 5. MPI 并行程序在运行到通信发送点时,冗余进程簇内进行数据比较,称这种数据比较技术为发送端数据比较技术.

发送端数据比较技术不增加额外的通信,这使得冗余进程簇间的通信与原通信开销一致.注意到,虽然发送端数据比较技术检查了并行程序通信时冗余进程簇内数据的正确性,但是当通信语句发送的数据通过网络传递到其他结点时,通信网络故障也可能会影响到这部分数据的正确性,从而使另一冗余进程簇接收到的数据不一定正确.因此,发送端数据比较技术适用于具有容错能力的互联网中(如自适应路由、报文重传等).

另外,除了确定比较的时机,还需要确定比较的数据.为避免错误数据的流出,仅需比较通信数据.

定义 6. MPI 并行程序运行到某条通信语句时,冗余进程簇内仅比较该语句中通信的数据,称这种数据比较技术为通信数据比较技术.

通信数据比较技术在保证错误数据不外流的情况下,极大地减少了用于比较的数据量.

© 中国科学院软件研究所 http://www.jos.org.cn

2.3 容错处理技术

由于赋值处理开销制约了 TMR 的可扩展性,本节考虑采用标记处理的方式进行容错.当数据比较之后检测出某个进程出错,则标记该进程,系统继续执行,直到程序结束或 3 个进程均出错而重新运行.因此,这种标记处理方式几乎无需额外的时间开销.图 4 分别示意了标记和赋值处理技术的实现过程,从图中可以看出,赋值处理方式比标记处理需要额外的通信赋值开销,而这部分开销影响了计算的可扩展性.

定义 7. 冗余进程簇内数据比较后,若有一个进程出错,则标记该出错进程.若标记的进程为元进程,则选择冗余进程簇中结点编号较低的影子进程成为元进程继续执行,这种实现方式称为标记技术.

标记技术实现起来高效、易行,但是,如果遇到同一个冗余进程簇内两个结点均出错的情况,则该技术无法继续执行,而需要重新运行 MPI 程序.

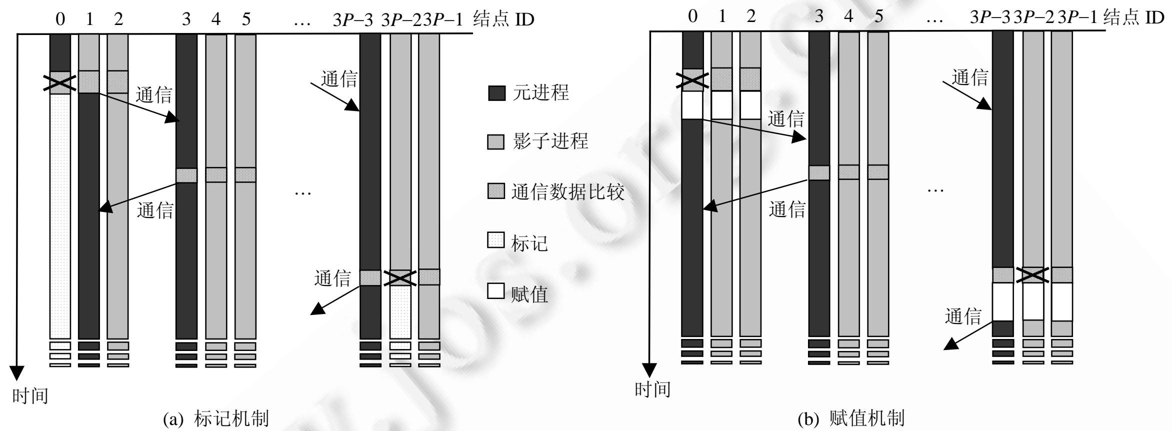


Fig.4 Marking technique and assignment technique

图 4 标记和赋值技术

2.4 STMR框架及可扩展性分析

根据上述的 TMR 实现可扩展性的技术设计,最终得到 STMR 容错机制的实现框架,如图 5 所示.

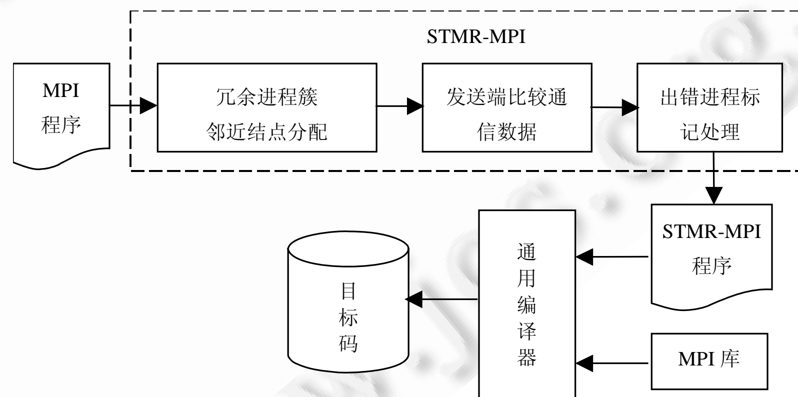


Fig.5 Framework of STMR

图 5 STMR 框架

于是,STMR 的实现方法定义如下:

定义 8. 首先对 MPI 并行程序 G 采用邻近拓扑结构感知划分冗余进程簇,运行时仅在通信的发送端比较通信数据,以检测发生的错误.若没有错误发生,则继续执行,直到程序 G 得到最终计算结果.最后,在冗余进程簇内

© 中国科学院软件研究所 http://www.jos.org.cn

进行最终计算结果的数据比较,以确保最终结果的正确;若在数据比较时检测出冗余进程簇内某个进程出错,则采用标记技术进行处理;若检测出超过一个进程出错,则程序 G 重新运行.称这样的容错实现方法为 STMR 容错机制.

接下来验证 STMR 容错机制的可扩展性,首先分析其实现过程中的时间开销.

图 6 分别反映了 P 个结点的大规模并行计算未引入 STMR 容错机制和引入 STMR 容错机制的运行情况,其中,并行程序 G 的原通信过程未发生改变,且采用标记技术处理出错进程几乎不引入额外开销.因此,STMR 容错开销仅为冗余进程簇内的数据比较时间,并且由于采用发送端通信数据比较技术,冗余进程簇内比较的数据仅为通信数据.

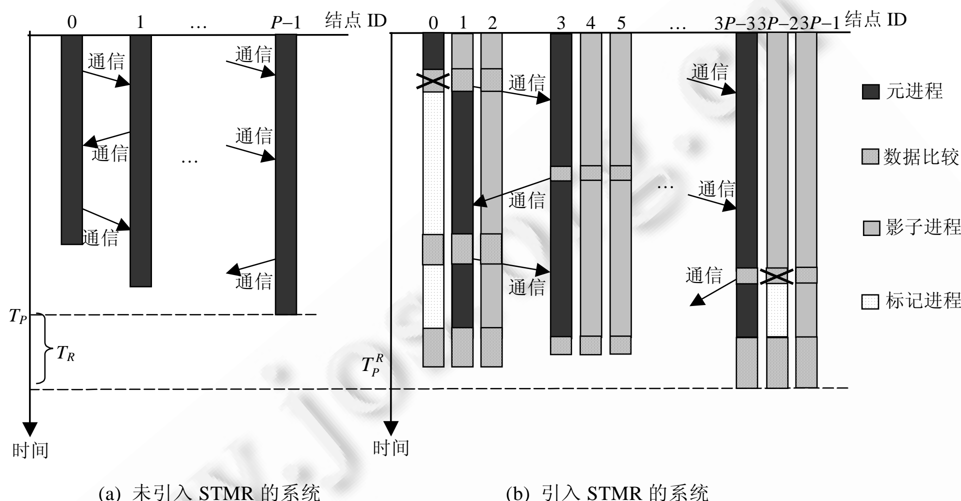


Fig.6 Running of large-scale parallel computing system

图 6 大规模并行计算系统的运行状态

不失一般性,假设由 N_j,N_{j+1} 和 N_{j+2} 上运行的冗余进程簇的运行时间 t_j^R ($j$mod3≡0)最长,即有 $T_P^R = t_j^R$,且该冗余进程簇进行数据比较的次数为 K_j+1(消息发送操作次数和一次最终结果比较),它与系统规模无关,因此有

$$T_R=(K_j+1)H_j \quad (7)$$

其中,H_j 为该组冗余进程簇进行数据比较时,两个运行影子进程的结点 N_{j+1} 和 N_{j+2} 向结点 N_j 传输比较数据的平均通信时间.

$$H_j = 2\frac{D_{size}}{b} \quad (8)$$

其中,D_{size} 为结点 N_{j+1} 和 N_{j+2} 向结点 N_j 单次传输的平均数据量;b 为冗余进程簇内传输数据的单链路带宽,则 b 为常数.由于可扩展并行程序 $S=\Theta(P)$,程序 G 的原通信开销不影响系统的可扩展性;而 D_{size} 由程序 G 本身的通信数据量以及最后一次数据比较两个影子进程运行结点的内存数据量计算得到,且在冗余进程簇内进行局部通信,因此由公式(7)、公式(8)得到:

$$T_R = 2(K_j+1)\frac{D_{size}}{b} = \Theta(1) \quad (9)$$

则得到 STMR 的 R 加速比满足 $S^R=\Theta(S)$,即 STMR 可扩展.

3 STMR 实现

根据 STMR 的设计框架,本节描述实现 STMR 的关键技术.

© 中国科学院软件研究所 http://www.jos.org.cn

3.1 冗余进程簇结点划分实现

对于一个需要进程数为 P 的程序,STMR 容错机制需要启动 $3P$ 个进程来运行.首先对进程从 0 到 $3P-1$ 进行编号,在初始化 MPI 环境之后,通过模三操作将通信域一分为三,由于冗余进程簇被分配在邻近的结点上,因此簇内进程的编号与簇内进程所在结点的 ID 号一致,记 $P_i(i\bmod 3\equiv 0)$为元进程,P_{i+1} 和 P_{i+2} 为其两个影子进程.图 7 给出了冗余进程簇创建的示例代码,其中,*rank* 为 *MPI_COMM_WORLD* 域中的进程号,*srank* 为划分后的域 *scomm* 中的进程号.

这段代码使用 *MPI_Comm_split* 调用将原有的通信域 *MPI_COMM_WORLD* 分割为 3 个通信域,程序的 3 个副本同时在这 3 个通信域上运行,其中每个通信域包含 P 个进程,进程号模数分别为 0,1 和 2,见表 1.在新的通信域中,每个进程都被重新编号,在 *MPI_COMM_WORLD* 中进程号为 $3h,3h+1$ 和 $3h+2(h=0,\ldots,P-1)$的进程在新的通信域中均被分配新的进程号 h.

```
…
MPI_Init(0,0);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
color=rank%3;
MPI_Comm_split(MPI_COMM_WORLD,color,rank,&scomm);
MPI_Comm_rank(scomm,&srank);
…
```

Fig.7 Example code of creating redundant process cluster

图 7 冗余进程簇创建示例代码

Table 1 Communicator's group partition

表 1 通信域的划分

通信域 0	通信域 1	通信域 2
color=0	*color*=1	*color*=2
rank=3*h*, *srank*=*h*	*rank*=3*h*+1, *srank*=*h*	*rank*=3*h*+2, *srank*=*h*

数据比较通过 P_i,P_{i+1} 和 P_{i+2} 之间发送比较的数据实现,因此需要在通信域 0,1 和 2 之间建立通信域,创建代码如图 8 所示.

```
…
if (color==0)
{MPI_Intercomm_create(scomm,0,MPI_COMM_WORLD,1,01,&FirstComm);
    MPI_Intercomm_create(scomm,0,MPI_COMM_WORLD,2,02,&SecondComm);}
else if (color==1)
{MPI_Intercomm_create(scomm,0,MPI_COMM_WORLD,0,01,&FirstComm);
    MPI_Intercomm_create(scomm,0,MPI_COMM_WORLD,2,12,&SecondComm);}
else if (color==2)
{MPI_Intercomm_create(scomm,0,MPI_COMM_WORLD,0,02,&FirstComm);
    MPI_Intercomm_create(scomm,0,MPI_COMM_WORLD,1,12,&SecondComm);}
…
```

Fig.8 Example code of creating inter-communicator in RPC

图 8 冗余进程簇内通信域创建示例代码

3.2 数据比较的MPI实现

根据 STMR 的定义,STMR 程序运行时或者运行结束后的数据比较包含两部分操作:首先,两个影子进程 P_{i+1} 和 P_{i+2} 分别将通信数据或者最终结果发送到元进程 P_i 上;然后,元进程 P_i 将接收到的数据与本地的数据进行比较.如果这 3 组数据至少有两组相同,则表明通信数据或者最终结果正确,于是执行 MPI 程序的通信语句或者程序运行结束;否则,发出错误信号,终止程序并重新运行.冗余进程簇内的数据比较技术实现如图 9 所示.

举例来说,假设影子进程 P_{i+1} 和 P_{i+2} 需要向元进程 P_i 发送变量 x,在元进程 P_i 中定义 y,z,分别用来接收影子进程发送过来的变量 x,示例代码如图 10 所示.

© 中国科学院软件研究所 http://www.jos.org.cn

由图 10 中元进程 P_i 的示例代码可知,当且仅当变量 x,y,z 中任意两个均不相等时程序重新启动.在仅有两个变量相等的情况下,进入出错进程标记处理模块.

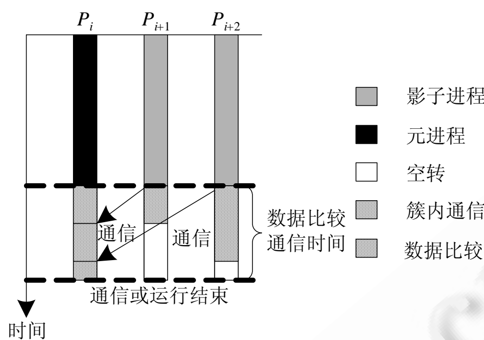


Fig.9 Data comparison in redundant process cluster

图 9 冗余进程簇内数据比较

P_i

```
…
int x[10], y[10], z[10];
…
MPI_Recv(y,10,MPI_INT,i/3,1,FirstComm);
MPI_Recv(z,10,MPI_INT,i/3,2,SecondComm);
if (x!=y){If (x!=z){If (y!=z){RESTRAT;}
          else {ERROR=0;}}
      else {ERROR=1;}}
else if (x!=z){
      ERROR=2;}
…
```

P_{i+1}

```
…
int x[10];
…
MPI_Send(x,10,MPI_INT,i/3,1,FirstComm);
…
```

P_{i+2}

```
…
int x[10];
…
MPI_Send(x,10,MPI_INT,i/3,2,FirstComm);
…
```

Fig.10 Example code of data comparison

图 10 数据比较示例代码

3.3 标记处理实现

根据标记技术的定义可知,标记处理有一个进程出错的情况.若出错进程为影子进程,则标记处理后不改变程序的运行;若出错进程为元进程 P_i,则编号较低的进程 P_{i+1} 转变为元进程继续执行.首先定义一个全局数组 *bool metaProcess*[*P*/3],用来标识每个冗余进程簇中的元进程是否出错(TRUE 表示出错,FALSE 表示未出错),示例代码如图 11 所示.

P_i

```
…
int MARK=0;
…
switch(ERROR){
0. MPI_Send(ERROR,…,i/3,1,FirstComm);
      MARK=1;
      metaProcess[i/3]=TRUE;
1. MPI_Send(ERROR,…,i/3,1,FirstComm);
2. MPI_Send(ERROR,…,i/3,2,SecondComm);}
…
```

P_{i+1}

```
…
int MARK=0;
…
MPI_Recv(ERROR,…,i/3,1,FirstComm);
if (ERROR==1){MARK=1;}
else if (ERROR==0){BE META PROCESS;}
else {…}
…
```

P_{i+2}

```
…
int MARK=0;
…
MPI_Recv(ERROR,…,i/3,2,FirstComm);
if (ERROR==2){MARK=1;}
…
```

Fig.11 Marking failed process

图 11 出错进程标记处理

© 中国科学院软件研究所 http://www.jos.org.cn

若从 P_i 到 P_j 点对点通信,元进程 P_i 被标记($MARK$=1),则由 P_{i+1} 与 P_j 进行通信代替原始的 P_i 到 P_j 点对点通信.因此,通信域 0 内的通信转化为通信域 1 到通信域 0 的域间通信,示例代码如图 12 所示.

```
Pi
…
COMPARE;
if (metaProcess[i/3]==FALSE){
if (metaProcess[j/3]==FALSE)
    MPI_Send(x,10,MPI_INT,j/3,1,scomm);
else
    MPI_Send(x,10,MPI_INT,j/3,1,FirstComm);}
…

Pi+1
…
COMPARE;
if (metaProcess[i/3]==TRUE){
if (metaProcess[j/3]==FALSE)
    MPI_Send(x,10,MPI_INT,j/3,1,FirstComm);
else
    MPI_Send(x,10,MPI_INT,j/3,1,scomm);}
…

Pj
…
COMPARE;
if (metaProcess[j/3]==FALSE){
if (metaProcess[i/3]==FALSE)
    MPI_Recv(x,10,MPI_INT,i/3,1,scomm);
else
    MPI_Recv(x,10,MPI_INT,i/3,1,FirstComm);}
…

Pj+1
…
COMPARE;
if (metaProcess[j/3]==TRUE){
if (metaProcess[i/3]==FALSE)
    MPI_Recv(x,10,MPI_INT,i/3,1,FirstComm);
else
    MPI_Recv(x,10,MPI_INT,i/3,1,scomm);}
…
```

Fig.12 Example code that the shadow process sends data instead of meta process

图 12 影子进程代替元进程发送数据示例代码

4 实 验

4.1 实验目的和方法

该实验旨在验证 STMR 容错机制的可扩展性,因此需要反映在系统规模不断扩大的情况下,STMR 的 R 加速比的变化趋势.然而,目前已有的大规模并行计算机系统的规模还不足以反映 STMR 容错机制的可扩展性.因此,本文将并行程序可扩展性和系统网络拓扑结构的特征抽象为紧致界函数,借助 Matlab 工具对这些函数进行模拟,比较当结点规模趋于无穷大时传统 TMR 容错机制与 STMR 容错机制的 R 加速比的变化情况,进而验证 STMR 容错机制的可扩展性.

由公式(1),参数 T 反映了原程序的可扩展性特征.由公式(2),TMR 和 STMR 的开销 T_R 是影响 R 加速比 S^R 的关键因素,从而决定了容错机制的可扩展性.因此,实验分别选择 T 和 T_R 合适的紧致界函数形式,以模拟不同扩展性类型的并行程序在不同网络拓扑结构类型的系统上运行时,传统 TMR 和 STMR 容错机制的可扩展性.

根据第 1.1 节的定义,程序的可扩展性有可扩展、弱可扩展和不可扩展 3 类.对于不可扩展程序,采用任何容错机制均是不可扩展的.因此,实验仅分别模拟可扩展和弱可扩展程序运行时的容错机制的扩展情况,即 T 的紧致界函数分别为常量(可扩展程序)和 $\sqrt{P}$ (弱可扩展程序)时的函数形式变化.

对于 T_R 的函数模拟,TMR 和 STMR 有所不同,需分别加以讨论.在 TMR 机制中,由公式(4)、公式(6),T_R 主要受结点数、二分带宽和跳步数的影响.假设二分带宽的紧致界函数为常量和 $\sqrt{P}$,网络跳步数的紧致界函数为常量和 $\lg P$,可计算得到 T_R 的紧致界函数为 P 和 $\sqrt{P}$.在 STMR 机制中,T_R 主要受到 D_{size} 的影响,而 D_{size} 受到程序中通信数据量的影响,根据上述对 T 的函数模拟假设及公式(3),则不妨令 T_R 的紧致界函数为常量、$\sqrt{P}$ 和 P.

根据上述假设,设计具体的实验方法如下:

- 当 MPI 程序为可扩展程序时,考察传统 TMR 和 STMR 的 R 加速比随结点数 P 的变化.以 T 取常量函数形式为例,由公式(1),则有 $S=\Theta(P)$,即 MPI 程序为可扩展程序.令通信数据量的紧致界函数分别为常量和 $\sqrt{P}$,对应的二分带宽的紧致界函数则分别为常量和 $\sqrt{P}$;
- 当 MPI 程序为弱可扩展程序时,考察传统 TMR 和 STMR 的 R 加速比变化随结点数 P 的变化情况.以当 T 取 $\sqrt{P}$ 函数形式为例,由公式(1),则有 $\Theta(1)\prec S\prec P$,于是可得 MPI 程序为弱可扩展程序.令通信数据量的紧致界函数分别为 $\sqrt{P}$ 和 P,对应的二分带宽的紧致界函数则分别为常量和 $\sqrt{P}$.

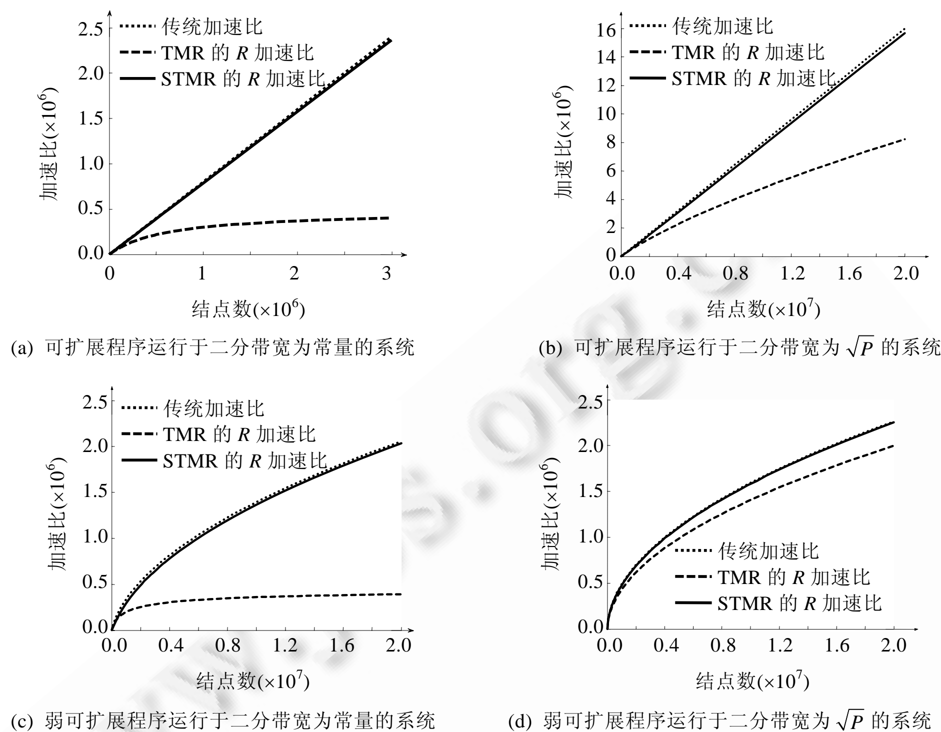
实验中需要用到的常量数据均为美国 Lawrence Livermore 国家实验室 BlueGene/L 的系统参数,其中,链路

© 中国科学院软件研究所 http://www.jos.org.cn

单方向的传输带宽为 1.4Gb/s,单结点内存容量为 4Gb[9].

4.2 模拟结果

根据上述实验方法进行函数模拟,系统规模 P 从 1 增长到 2×10^9,结果如图 13 所示.



(a) 可扩展程序运行于二分带宽为常量的系统　(b) 可扩展程序运行于二分带宽为 $\sqrt{P}$ 的系统

(c) 弱可扩展程序运行于二分带宽为常量的系统　(d) 弱可扩展程序运行于二分带宽为 $\sqrt{P}$ 的系统

Fig.13 *R* speedup for TMR and STMR

图 13 TMR 与 STMR 的 *R* 加速比

图 13 反映了具有不同可扩展性特性的程序,运行于具有不同二分带宽紧致界函数的系统上,随着结点数的增长,传统 TMR 和 STMR 的 *R* 加速比的变化情况.

图 13(a)和 13(b)反映的是可扩展程序 $S=\Theta(P)$的运行情况,其中,图 13(a)中通信数据量和二分带宽均是常量函数.此时,传统 TMR 的 S^R 随着系统规模的扩大趋近于某一常数,即 TMR 为不可扩展的;而 STMR 的 S^R 随着系统规模的扩大与传统加速比 S 的变化保持一致,即 STMR 为可扩展的.类似地,图 13(b)中通信数据量和二分带宽均为 $\sqrt{P}$ 函数.此时,传统 TMR 的 S^R 随着系统规模的扩大弱于 P 函数而增大,即 TMR 为弱可扩展的;而 STMR 的 S^R 仍然与传统加速比 S 的变化保持一致,即 STMR 为可扩展的.由此,验证了 STMR 的可扩展性.

图 13(c)和图 13(d)反映的是弱可扩展程序$\Theta(1)\prec S\prec P$ 的运行情况,其中,图 13(c)中通信数据量为 $\sqrt{P}$, 二分带宽为常量.此时,传统 TMR 的 S^R 随着系统规模的扩大趋近于某一常数,即 TMR 为不可扩展的.类似地,图 13(d)中通信数据量为 P,二分带宽为 $\sqrt{P}$ 函数.此时,传统 TMR 的 S^R 随着系统规模的扩大弱于 P 函数而增大,即 TMR 为弱可扩展的.而在图 13(c)和图 13(d)中,采用 STMR 的 *R* 加速比与传统加速比 S 增长曲线几乎重合,表明,对于弱可扩展程序,STMR 同样能够表现出较好的扩展性.

综上所述,STMR 具有良好的可扩展性,并且能够有效地解决传统 TMR 不可扩展或弱可扩展的问题.

5 相关工作

容错是通过冗余实现的[10],C&R[2]和 message logging[3]是两种主要的基于回滚恢复的容错机制.传统的

© 中国科学院软件研究所 http://www.jos.org.cn

system-level C&R 要求周期性地将所有进程的地址空间内容(堆、栈和全局变量)、寄存器信息和通信库状态存储到可靠的存储器上.恢复时,所有处理器读入最近的 checkpoint 并重新开始计算.当系统包含了数千甚至数万个处理器时,做一次 checkpoint 可能会导致传输 T(10^{12})量级的数据到存储器上,系统 I/O 成为性能瓶颈.另外,即使在无故障的情况下,这种方法的开销也与有故障时相当.Message logging 与 C&R 的差别在于,出现故障时,只有故障进程需要回滚到前一个 checkpoint,其他进程通过重发已经发送过的消息来帮助故障进程恢复计算.这就要求各进程记录全部已发送的消息,或在重启进程需要时重新产生这些消息[11].但是,并行程序的通信非常频繁,并且消息中可能包含大量的数据,在实际应用中,保存或重新产生这些消息的开销很大,这阻碍了 message logging 技术在高性能计算领域内的应用.

三模冗余[6]和 *N* 模冗余[5]容错方法由于需要消耗冗余的系统资源,因此目前还没有应用于大规模并行计算.文中以 TMR 为典型案例,分析讨论了这类容错机制在大规模并行系统上的实现方法,由此提出 STMR 容错机制.该容错机制基于空间冗余的思想,实现对大规模并行计算的容错,并且能够将容错开销控制在很小的范围内,从而使引入 STMR 容错机制的并行系统仍然具有可扩展性.

此外,消息传递并行编程接口 MPI[12]在大规模并行系统中被广泛使用,对 MPI 应用程序的容错是一个值得关注的问题.为了实现高性能而设计的一个开源的消息传递接口 OpenMPI[13]综合了早期所有 MPI 版本的特性.它为 C&R,message logging 等容错技术提供了高效的容错平台,保证网络或系统结点故障的恢复、用户或通信指导的容错以及数据的可靠性.因此,OpenMPI 主要支持 C&R 等容错方法的实现.

6 结论与下一步工作

本文提出了一种面向 MPI 大规模并行计算的可扩展三模冗余容错机制——STMR,该机制不仅能够处理 fail-stop 错误,还能够解决绝大部分硬件不能直接感知的数据错误.我们基于 TMR 容错机制影响并行计算可扩展性存在的问题,设计出支持STMR容错的实现技术,通过对这些技术的分析确定STMR的整体框架.最后,分析和模拟验证了 STMR 容错机制的有效性和可扩展性.

STMR 容错机制在 MPI 程序的基础上通过修改代码来实现.为了使该工作对用户透明,可设计一个自动化的工具来实现 MPI 程序到 STMR 程序的转化,这一点将在后续的工作中进一步加以研究.

References:

[1] Los Alamos National Laboratory. Operational data to support and enable computer science research. http://institute.lanl.gov/data/lanldata.shtml

[2] Lu CD. Scalable diskless checkpointing for large parallel systems [Ph.D. Thesis]. Urbana-Champaign: University of Illinois, 2005.

[3] Chakravorty S, Kale LV. A fault tolerance protocol with fast fault recovery. In: Proc. of the 21st IEEE Int’l Parallel and Distributed Processing Symp. Long Beach, 2007. 120−128. http://charm.cs.illinois.edu/newPapers/06-12/paper.pdf [doi: 10.1109/IPDPS.2007.370310]

[4] Elnozahy EN, Alvisi L, Wang YM, Johnson DB. A survey of rollback-recovery protocols in message-passing systems. ACM Computing Surveys, 2002,34(3):375−408. [doi: 10.1145/568522.568525]

[5] Mancini L, Koutny M. Formal specification of *N*-modular redundancy. In: Proc. of the ’86 ACM 14th Annual Conf. on Computer Science. New York, 1986. 199−204. http://www.cs.ncl.ac.uk/publications/trs/papers/213.pdf [doi: 10.1145/324634.325389]

[6] Neumann JV. Probabilistic logics and the synthesis of reliable organisms from unreliable components. In: Shannon CE, McCarthy J, eds. Proc. of the Automata Studies. Princeton: Princeton University Press, 1956. 43−98.

[7] Guo L, Tang ZS. Specification and verification of the triple-modular redundancy fault-tolerant system. Journal of Software, 2003, 14(1):54−61 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/14/54.htm

[8] Hwang K. Advanced Computer Architecture: Parallelism, Scalability, Programmability. Beijing: Tsinghua University Press, 2001. 301−303 (in Chinese).

[9] IBM Corp. Unfolding the IBM eserver blue gene solution. 2005. http://www.redbooks.ibm.com/redbooks/pdfs/sg246686.pdf

© 中国科学院软件研究所 http://www.jos.org.cn

[10] Heimerdinger WL, Weinstock CB. A conceptual framework for system fault tolerance. Technical Report, CMU/SEI-92-TR-33, Pittsburgh: Carnegie Mellon University, 1992.

[11] Bouteiller A, Herault T, Krawezik G, Lemarinier P, Cappello F. MPICH-V Project: A multiprotocol automatic fault-tolerant MPI. Int'l Journal of High Performance Computing and Applications, 2006,20(3):319−333.

[12] The message passing interface (MPI) standard. http://www.mcs.anl.gov/research/projects/mpi/

[13] Open MPI: Open source high performance computing. http://www.open-mpi.org/

附中文参考文献:

[7] 郭亮,唐稚松.三机冗余容错系统的描述和验证.软件学报,2003,14(1):54−61. http://www.jos.org.cn/1000-9825/14/54.htm

[8] 黄铠.高等计算机系统结构:并行性,可扩展性,可编程性.北京:清华大学出版社,2001.301−303.

王之元(1982－),女,河南新乡人,博士生,CCF会员,主要研究领域为计算机体系结构.

周云(1978－),男,博士生,CCF 学生会员,主要研究领域为人工智能.

杨学军(1963－),男,博士,教授,博士生导师,CCF高级会员,主要研究领域为计算机体系结构.

www.jos.org.cn
www.jos.org.cn