Journal of Computer Applications
计算机应用,2014,34(2):382-386
ISSN 1001-9081
CODEN JYIIDU
2014-02-10
http://www.joca.cn

文章编号:1001-9081(2014)02-0382-05
doi:10.11772/j.issn.1001-9081.2014.02.0382

面向计算流体力学应用开发框架的容错周期优化方法

张拥军*, 徐新海
(国防科学技术大学 计算机学院,长沙 410073)
(*通信作者电子邮箱 yjzhang@nudt.edu.cn)

摘　要:针对计算流体力学应用开发框架容错支持能力的不足,提出了一种新的容错周期优化方法。该方法基于系统故障的概率建模,计算得到理想最优容错周期;并结合计算流体力学应用场数据输出的特点,在线确定实际检查点备份时机。三个典型应用的实验结果表明,在不同平均无故障时间的系统上,与固定时间步进行容错的方法相比,该方法总能够得到最优的容错开销。用户可以基于该方法通过框架接口便捷地设置容错周期,并有效降低容错所引起的开销。

关键词:容错;周期优化;检查点;计算流体力学;开发框架
中图分类号:TP302.8　　**文献标志码**:A

Fault-tolerance period optimization method for computational fluid dynamics-oriented application development frameworks

ZHANG Yongjun*, XU Xinhai
(*School of Computer, National University of Defense Technology, Changsha Hunan* 410073, *China*)

Abstract: For the fault-tolerance shortage of CFD (Computational Fluid Dynamics)-oriented application development framework, a new fault-tolerance period optimization method was proposed. The method computed the ideal best fault-tolerance period based on the probability model of system's faults, and online determined the occasion of real check points with the consideration of CFD fields output characteristic. The experimental results of three applications show that on the systems with different mean time between faults, compared with the fault-tolerance method based on performing fault-tolerance between fixed steps, the proposed method can always get the best fault-tolerance overheads. Based on this method, user can set the fault-tolerance period with framework interfaces conveniently and reduce the fault-tolerance overheads.

Key words: fault-tolerance; period optimization; check point; Computational Fluid Dynamics (CFD); development framework

0　引言

计算流体力学(Computational Fluid Dynamics, CFD)是一门流体力学和数值方法相结合的交叉学科。由于具有成本低、能模拟较复杂或较理想过程等优点,利用并行计算机对CFD应用进行模拟计算已经成为与传统的理论分析和实验验证一样重要的流体力学研究方法[1]。近年来,随着高性能计算技术的突飞猛进,CFD方法得到飞速发展,在诸多工程技术领域都有广泛应用,其取得的丰硕成果已经受到了学术界和工业界的充分认可。

然而,高性能计算在不断提升CFD方法模拟性能的同时,也带来了两个问题:一方面,高效CFD问题涉及多个学科知识,高效的并行应用开发需要物理/化学/生物领域专家、数值计算专家和并行计算专家的相互合作;另一方面,由于芯片集成度的提高和系统并行度的增加,高性能计算机的可靠性问题日益突出,有研究结果表明当系统的并行度达到数十万核时,系统的平均无故障时间(Mean Time Between Faults, MTBF)仅为小时量级[2],而最新国际Top500排行榜前十名的机器均拥有数十万乃至数百万个处理器核[3]。

为了应对高效CFD并行程序开发困难的问题,研究人员设计了面向CFD应用领域的并行应用开发框架(后文简称为面向CFD应用开发框架),如商用框架软件Fluent[4]和开源框架软件OpenFOAM[5]。这类框架对应用采取了高层抽象,屏蔽了底层程序设计细节,用户无需了解太多计算机程序设计方面的细节就可以通过近似自然语言的方式方便地开发出适合自身需求的应用程序。但是,在面向CFD应用的开发框架中设计高效的容错方法仍处于起步阶段。当前大多数面向CFD应用的开发框架提供的是一种周期性的基于场数据的容错方法,即简单地将用于后期可视化显示的场数据作为检查点使用。这种方法虽然能够实现一定容错的功能,但可能存在检查点备份周期不满足系统容错需求的问题。

在此背景下,本文首先对面向CFD应用开发框架中现有的通过周期性场数据输出的容错技术进行了分析,指出其存在检查点时间间隔长度的不科学性;然后,提出了面向CFD应用开发框架的检查点周期优化方法;最后,在OpenFOAM中实现了该方法。实验表明,本文提出面向CFD应用开发框

收稿日期:2013-08-19;**修回日期**:2013-10-16。
基金项目:国家自然科学基金资助项目(61120106005,61303071);广州市科信局基金资助项目(134200026)。
作者简介:张拥军(1972-),男,湖南新邵人,副研究员,博士,主要研究方向:高性能计算、容错;　徐新海(1984-),男,江苏镇江人,助理研究员,博士,CCF会员,主要研究方向:高性能计算、容错。

架的容错周期优化方法可以在保证 CFD 应用在不可靠并行计算系统上高效运行的同时,准确获得较优的检查点备份间隔,有效降低容错引起的额外开销。

1　基于周期场数据的检查点容错

本章将首先对检查点容错方法的基本思想与相关概念展开介绍,然后说明现有 CFD 应用开发框架如何通过周期性的场数据保存实现基本的检查点容错,最后对这种基于简单周期场数据的检查点容错方法的不足展开分析。

1.1　检查点容错

检查点(check point)是一种在分布式和并行计算领域广泛使用的容错方法[6]。如图 1 所示,检查点方法在系统正常运行过程中周期性地将程序运行现场保存至稳定的存储介质,并称其为一个检查点,当系统失效时,程序回滚至最近的检查点,并待从稳定存储介质中恢复程序运行现场后,重新进行计算。

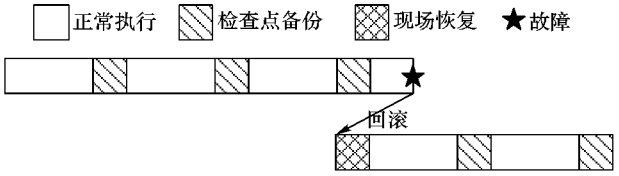

图 1　检查点容错方法

按负责保存检查点的抽象层次,检查点技术一般可以被分为系统级检查点(System-Level Checkpointing, SLC)和应用级检查点(Application-Level Checkpointing, ALC):其中 SLC 由系统负责定时保存全系统的运行现场,对用户透明,但是该方法保存的数据量较大,随着系统规模的增大,其检查点备份开销可能无法接受[7];而 ALC 却借助于程序员的参与,显式地指定检查点的时机和对象,大大降低检查点备份的开销[8]。

通常而言,容错方法均伴随着相应的时空代价,检查点方法也不例外。现有关于检查点方法的研究主要集中以下几个方面:1)检查点备份的周期,即在什么时机备份检查点才能使得整个程序的执行时间最小[9],一方面检查点周期过小将导致程序在正常计算过程中频繁备份检查点,从而加大程序在无故障执行过程中的容错开销;另一方面检查点周期过大将导致发生故障时的回滚代价增大,从而增加整个程序的执行时间。2)检查点的数据集,即备份检查点时具体应该保持哪些数据,数据越精简,保持时间越少[10]。3)检查点的写优化,即如何将检查点备份的写文件时间减少或者隐藏[11];4)对于并行程序,如何在程序运行过程中形成全局一致的检查点也是一个研究热点[12]。本文重点关注的就是第一方面的检查点备份周期问题。

1.2　CFD 应用中的周期性场数据

利用计算机对 CFD 应用进行模拟计算时,首先需要对待模拟的实际连续问题进行空间和时间两个维度的离散。所谓空间离散就是将待模拟的区域划分成若干网格,基于网格进行计算;而时间离散则是将原本连续的时间划分成若干时间步(也称为时间片),按时间步推进模拟。为了便于对流体在流动过程中的各种物理规律进行观察,CFD 应用往往需要在计算过程中将模拟区域的流场数据,即所有网格点上的物理属性值(也称为场数据),以文件形式周期性地保存到外存上,待模拟结束后,再通过后端处理软件对计算得到的场数据进行可视化演示。

这样,我们就得到一个如图 2 所示的 CFD 模拟及场数据保存的过程:CFD 模拟计算的过程按时间步推进,由若干时间步构成一个写周期,在每个写周期的末尾进行场数据的保存。为了支持周期性输出场数据的功能,现有 CFD 应用开发框架一般都提供一定配置方式,由用户指定一个写周期中所包含的时间步的个数。值得注意的是,上述方法配置得到的所有写周期中的时间步个数都是相同的。

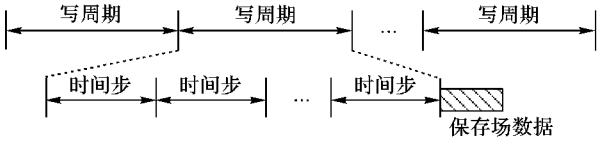

图 2　CFD 模拟及场数据保存过程

1.3　周期性场数据作检查点的问题分析

类比 1.1 节的检查点容错方法和 1.2 节 CFD 周期性场数据,我们不难发现二者的计算过程抽象存在一个相似之处:均在计算过程中周期性地将某些计算信息存储到稳定介质上去。同时,由于 CFD 场数据的信息已经完全包含了对应时间步末尾的所有有效现场,因此场数据可以被当作检查点数据使用,即当故障发生时,仅根据场数据就可以进行故障恢复。

基于场数据可以被当作检查点使用的事实,当前 CFD 应用开发框架为了应对越发严重的可靠性问题,往往提供类似于检查点容错的功能,即当 CFD 计算过程发生故障后,再次启动的 CFD 计算可以在上次计算最后得到的场数据基础上,无需对该场数据之前的写周期过程进行复算,而是直接对后续的时间步进行模拟计算。然而,通过分析发现这种基于简单周期场数据的检查点容错方法存在以下不足:

1)场数据所对应的写周期不一定满足检查点备份周期的要求:一方面,在使用检查点容错方法时,为了避免故障恢复时间过长从而导致多次回滚情况的出现,一般要求检查点之间的时间间隔小于系统平均无故障时间的一半;另一方面,场数据输出之间的写周期间隔由用户根据自身所期望看到的流程后期显示效果所决定,因此,可能出现场数据输出间隔大于检查点备份将要求的情况。而且当用户不关心 CFD 模拟中间过程的流场变化细节,仅关心最终稳态的流场状态时,可能出现没有中间场数据输出的极端情况。

2)场数据所对应的写周期和检查点的备份周期单位不一致:场数据所对应的写周期以时间步为单位,虽然每个时间步对应的流场实际流动时间是一样的,但不同时间步的模拟计算时间却可能是不一样的,一般而言流场变化剧烈的时间步所需要的模拟时间较长;而检查点备份的周期是以真实的模拟计算时间为单位的。

为了解决上述两个问题,在使用现有 CFD 应用开发框架时,必须强制用户缩短输出场数据的写周期,以保证最长的场数据写周期模拟执行时间满足检查点备份周期的要求。然而,由于不同场数据写周期执行时间的不同,上述方法会导致检查点之间的程序执行时间存在较大不同,流场变化不剧烈的时间段被插入了过多的检查点,引入了更多的容错开销。

2　CFD 检查点备份周期优化

解决第 1.3 节所分析得到的基于简单周期场数据的检查

点容错方法不足的关键在于解耦检查点备份周期与场数据输出周期之间的对应关系，允许检查点备份之间的时间步个数的动态变化，即对检查点备份的周期进行优化。本章首先假设检查点可以在 CFD 应用模拟计算过程中的任意时刻进行，从而对 CFD 应用的检查点容错过程进行理想建模；再引入检查点仅能插入在时间步模拟的末尾这一约束，从而建立更加符合 CFD 应用的离散模型，最终分析得到检查点插入的最优时机。

2.1 理想连续模型

为了描述方便，定义如表 1 所示的符号。

表 1 建模符号说明

符号	说明
T	不进行容错处理，也不发生故障时程序正常执行的总时间
T^{FT}	进行检查点容错，且故障发生后进行回滚的程序总执行时间
T_S	检查点保存数据的时间，即场数据的输出时间
T_R	现场恢复的时间
T_C	无故障发生时两个相邻检查点间的计算时间间隔
T_C^{FT}	考虑故障与回滚时，程序顺利通过一个检查点间隔的时间
λ	系统的平均无故障时间
ρ	系统发生故障的概率密度函数

不失一般性，假设故障的发生是随机的，可能发生在计算、检测点备份和现场恢复过程中，且系统发生瞬时故障的概率随程序执行时间 t 服从负指数分布，即系统在执行至 t 时刻发生瞬时故障的概率为 $\rho(t) = \frac{1}{\lambda}e^{-t/\lambda}$。另外，在本节的理想连续模型中，假设程序能够在执行过程中的任意时刻暂停，从而将此时刻作为检查点来进行程序现场备份。

在系统可能发生故障时，程序对某个检查点间隔进行计算可能出现以下两种情况：

1）在检查点间隔的计算过程和新检查点的备份过程中系统均未发生故障，此时程序通过该检查点间隔的执行时间为：$T_C + T_S$。

2）程序在首次进行检查点间隔的计算或新检查点备份的过程中发生故障，则其通过该检查点的间隔执行时间为：$t_{fault} + T_{recovery}$，其中 t_{fault} 表示以该检查点间隔开始计算为0时刻时故障发生的时刻，$T_{recovery}$ 表示进行故障恢复的时间。所谓故障恢复包括现场恢复、计算和新检查点备份等工作。

因此令 $T_{CS} = T_C + T_S$，则程序顺利通过一个检查点间隔的时间 T_C^{FT} 的数学期望如式(1) 所示：

$$E(T_C^{FT}) = \int_{T_{CS}}^{+\infty}[\rho(t)T_{CS}]dt + \int_0^{T_{CS}}[\rho(t)(t + E(T_{recovery}))]dt \quad (1)$$

而程序在故障恢复过程中，既可能一次执行顺利完成，也可能再次发生故障，因此令 $T_{RCS} = T_R + T_C + T_S$ 故障恢复时间的数学期望如式(2) 所示：

$$E(T_{recovery}) = \int_{T_{RCS}}^{+\infty}[\rho(t)T_{RCS}]dt + \int_0^{T_{RCS}}[\rho(t)(t + E(T_{recovery}))]dt \quad (2)$$

基于故障的概率分布函数 $\rho(t) = \frac{1}{\lambda}e^{-t/\lambda}$，对式(2) 求解得到 $E(T_{recovery}) = -\lambda + \lambda/e^{-(T_R+T_C+T_S)/\lambda}$，将其代入式(1)，最终得到 T_C^{FT} 的数学期望如式(3)所示：

$$E(T_C^{FT}) = \lambda e^{T_R/\lambda}(e^{(T_C+T_S)/\lambda} - 1) \quad (3)$$

则，在考虑容错情况下，程序总执行时间的数学期望近似于式(4)所示：

$$E(T^{FT}) = \frac{T}{T_C}E(T_C^{FT}) = \frac{\lambda e^{T_R/\lambda}T}{T_C}(e^{(T_C+T_S)/\lambda} - 1) \quad (4)$$

为了获得理想连续模型下最优的检查点备份间隔，令 $\frac{d(E(T^{FT}))}{d(T_C)} = 0$，求解得到如下方程：

$$e^x x - e^x + e^{-T_S/\lambda} = 0 \quad (5)$$

其中 $x = T_C/\lambda$。

至此，在已知系统备份一次检查点即进行一次场数据输出的时间 T_S 和系统平均无故障时间 λ 的情况下，就可以通过解方程(5) 得到最优的检查点备份间隔 T_C^{Ideal}。

2.2 实际离散模型

上述理想模型假设程序能够在执行过程中的任意时刻暂停，从而将此时刻作为检查点来进行程序现场备份。然而，基于 CFD 应用开发框架的模拟只能在时间步结束时输出场数据。也就是说，检查点只能设置在一些离散的时间点上，即无法保证总是能获取理想连续模型中提出的最优检查点时间间隔 T_C^{Ideal}。为了解决这一问题，基于 CFD 应用中场输出时机是离散的这一前提，建立了检查点备份时机选择的实际离散模型。

实际离散模型的核心思想是在每个模拟时间步结束时，对当前情况进行分析从而判断是否需要将此时的场数据作为检查点进行输出。记当前模拟时间步的计算时间为 t_c，模拟结束时刻为 t_s，与 t_s 最近的上一个检查点完成时刻为 t_b，$\Delta T = t_s - t_b$。实际离散模型确定检查点备份即场数据输出时机的流程如图 3 所示。

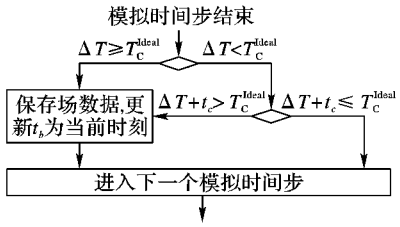

图 3 实际离散模型中检查点备份时机的选择流程

当某个模拟时间步结束时，将 $\Delta T = t_s - t_b$ 与最优检查点时间间隔 T_C 进行比较：如果二者刚好相等，则将当前的场数据作为检查点保存，并将 t_s 的值赋给 t_b；如果 ΔT 大于 T_C，即表明超过了最优时间间隔还没有保存过检查点，则将当前的场数据作为检查点保存，并将 t_s 的值赋给 t_b；如果 $\Delta T < T_C$，即表明还没有达到两个相邻检查点理论上的最优时间间隔。在上述最后一种情况下，由于无法预计下一个模拟时间步的计算时间，因此根据流场变化的连续性，假设下一个模拟时间步的计算时间近似等于本模拟时间步的计算时间 t_c。在此基础之上，将 $\Delta T + t_c$ 的值与理想最优检查点间隔 T_C^{Ideal} 比较：如果 $\Delta T + t_c > T_C^{Ideal}$，则说明当下一时间步模拟结束时，本次检查点备份周期很有可能已经超过了最优检查点备份间隔，因此直接将当前的场数据作为检查点保存；反之，则说明下一个模拟时间步结束时得到的检查点时间间隔很可能比本次更接近理想值，因此不进行场数据的输出，而直接进入下一个模拟时

间步的模拟计算。

3　基于 OpenFOAM 的实现

将第 2 章所设计得到的面向 CFD 应用开发框架的容错周期优化方法在 OpenFOAM 中实现。OpenFOAM 是一款基于 C++ 语言开发的,面向 CFD 应用的开源并行开发框架,其框架结构如图 4 所示。框架主要由前处理、后处理、平台接口、问题求解和基础支撑五部分组成:前处理主要负责对待模拟问题的几何建模、网格生成与为了并行计算的网格区域划分;后处理将模拟计算的结果进行格式转换后,组织成供第三方工具显示的数据;平台接口一方面通过配置文件的形式为领域用户提供已有模型的编程接口,另一方面也支持新的应用模型开发;问题求解对描述完全的连续问题进行数值离散和计算;基础支撑则提供以 MPI 为主的并行库和包括I/O管理在内的资源管理。

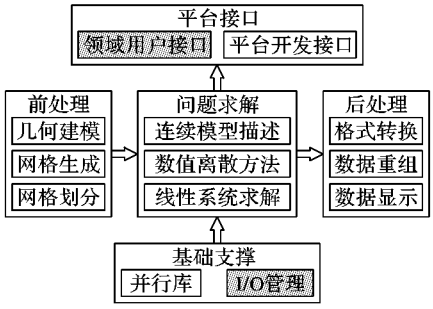

图 4　OpenFOAM 框架结构

OpenFOAM 受到学术界和产业界广泛认可的一大原因就是其层次分明的框架结构,基于这一优点,在 OpenFOAM 中实现容错周期优化功能时仅需要修改平台结构中的领域用户接口模块和基础支撑中的 I/O 管理模块:

1)在领域用户接口模块中,为用户通过文件配置 CFD 应用相关属性时增加了系统平均无故障时间 λ 和场数据输出时间 T_S 两个条目。其中,系统平均无故障时间是并行计算系统的固有属性,可以根据机器的历史维护日志得到;场数据的输出时间则和待模拟问题的场数据大小以及机器的 I/O 性能有关,这既可以通过计算得到,也可以通过 Profiling 方法得到。

2)在 I/O 管理模块,首先基于用户配置提供的系统平均无故障时间和场数据输出时间计算得到本 CFD 应用问题在系统上模拟的最优检查点间隔 T_C^{Ideal},然后在场数据输出控制逻辑中添加如图 3 所示的算法功能。

4　实验与分析

4.1　平台与测试用例

为了验证本文所提出的面向 CFD 应用开发框架容错周期优化方法的正确性与有效性,使用第 3 章介绍的方法在 OpenFOAM2.1.1 版本上实现了第 2 章所设计的容错周期优化方法,并将其安装在天河-1A[13] 的子系统上。该子系统拥有 20 个计算节点,每个节点包含 2 个 2.93 GHz 的 6 核 Intel Xeon X5670 处理器和 24 GB 的内存,系统运行在 Redhat5.5 操作系统上。需要指出的是,虽然全子系统一共拥有 240 个处理器核,但是在使用中仅适用了其中 200 个处理器核,即所有应用执行的并行度均为 200。测试用例方面,选择了 OpenFOAM 自带的 Cavity、PitzDaily 和 Dambreak 等 3 个三维测试用例,它们的问题规模和迭代时间步数的信息如表 2 所示。

表 2　测试用例基本执行信息

测试用例	网格规模	模拟步数	T/s	T_S/s
Cavity	150×150×150	100	214.43	0.60
PitzDaily	4 190 334	200	888.14	0.85
Dambreak	46×46×46	1 000	167.09	0.27

4.2　实验方法

由于实验平台仅由 20 个计算节点组成,所以其真实的平均无故障时间 λ 较大,因此在实验过程中假设系统的平均无故障时间 λ 为 25,50 和 100 s,并通过负指数分布的瞬时故障概率密度函数 $\rho(t) = \frac{1}{\lambda}e^{-t/\lambda}$ 模拟随机产生下一个故障到达的时间。需要说明的是,由于故障是随机注入的,因此后文所有容错效果的测试均采用了多次实验取平均的方法。

在具体实验过程中,首先在不注入故障的情况下对各个测试用例的正常执行性能进行测试,获得各自的正常执行时间 T,检查点备份时间即场数据输出时间 T_S。其次,以 Cavity 为分析案例对其容错周期优化情况展开详细分析:获得其模拟过程中各个模拟时间步的真实执行时间;测试不同 λ 下,以不同模拟时间步作为容错周期时的容错效果;计算不同 λ 下基于理想最优检查点备份周期 T_C^{Ideal},并使用图 3 所示优化方法进行检查点备份的容错效果。最后,我们分别选取合适的 λ,测试本文所提出的容错周期优化方法对另外两个测试用例即 PitzDaily 和 Dambreak 的效果。

4.3　结果与分析

表 2 中给出了三个测试用例各自的无容错正常执行时间 T 和检查点备份时间即场数据输出时间 T_S。进一步,测试用例 Cavity 在模拟过程中各个模拟时间步的真实执行时间如图 5 所示。不难发现,不同模拟时间步的执行时间是不尽相同的,初始时间步的执行时间较长的原因在于在模拟初期的流场变化较为剧烈。

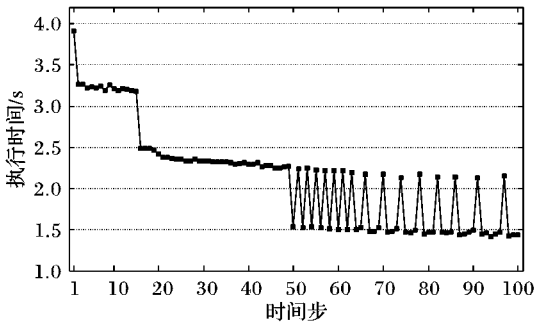

图 5　Cavity 各个模拟时间步的无故障执行时间

针对测试用例 Cavity,变换系统平均无故障时间 λ 为 25 s、50 s 和 100 s,并通过式(5)求解得到各自所对应的理想最优检查点备份周期 T_C^{Ideal} 分别为 5.085 s、7.35 s 和10.56 s。在三种 λ 下,使用不同容错周期的容错效果如图 6 所示,图中方点对应使用不同时间步为检查点间隔的程序总执行时间;而实线则表示基于各自 T_C^{Ideal},并使用图 3 所示方法进行容错周期优化的程序总执行时间。不难发现,对于 Cavity 用例,在不同系统平均无故障时间下,使用基于固定时间步的检查点

周期选择方法的最优周期是变化的;而本文所提出的容错周期优化方法总是可以得到较优的检查点备份间隔。

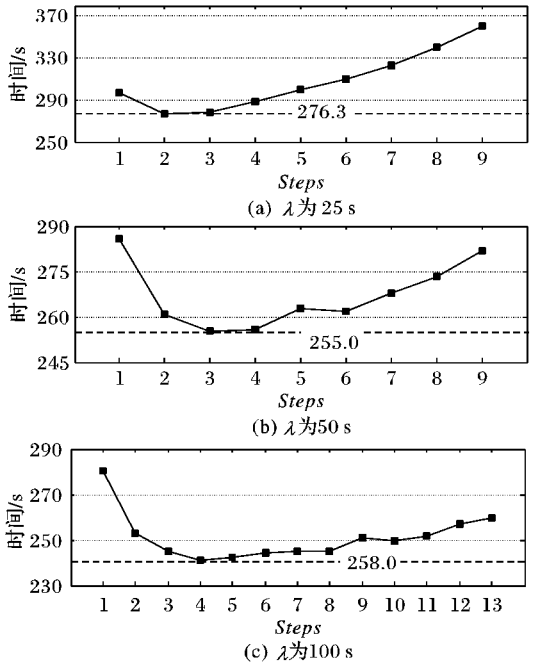

图6 Cavity 用例容错开销分析

类似地,测试 PitzDaily 和 Dambreak 用例分别在 $\lambda = 100$ s 和 $\lambda = 25$ s 情况下,使用不同容错周期方法的总执行时间。从图7中,不难发现,本文所提出的容错周期优化方法可以准确获得较优的检查点备份间隔,从而有效降低 CFD 应用的容错开销。

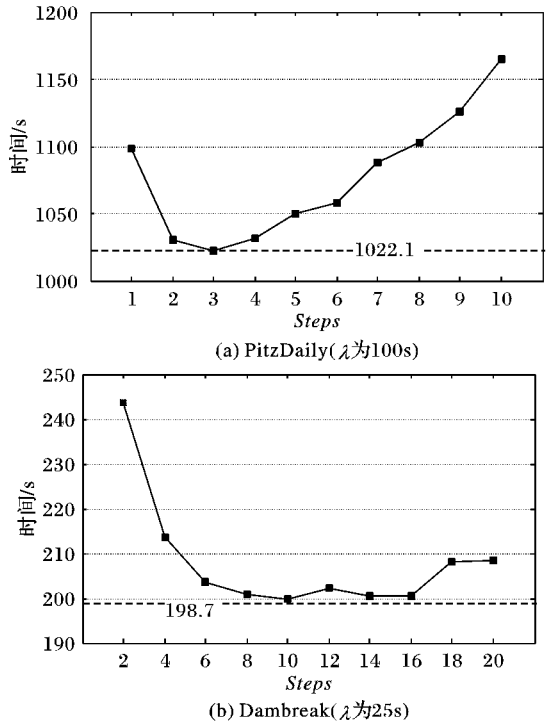

图7 PitzDaily 和 Dambreak 用例容错开销分析

5 结语

本文针对 CFD 应用在并行计算系统上的可靠性问题,分析了现有基于简单场数据输出以实现检查点容错方法在容错周期设置上的不足,通过理想连续与实际离散建模方法,设计得到了面向 CFD 应用开发框架的容错周期优化方法,并将其在 OpenFOAM 框架中实现。实验结果表明本文的容错周期优化方法可以有效降低 CFD 应用的总容错开销。

参考文献:

[1] YAN C, YU J, XU J, *et al.* On the achievements and prospects for the methods of computational fluid dynamics[J]. Advances in Mechanics, 2011, 41(5): 562 – 589. (阎超, 于剑, 徐晶磊, 等. CFD 模拟方法的发展成就与展望[J]. 力学进展, 2011, 41(5): 562 – 589.)

[2] LU C D. Scalable diskless checkpointing for large parallel systems [D]. Urbana-Champaign: University of Illinois at Urbana-Champaign, 2005.

[3] Top500 [OL]. [2013-10-15]. http://www.top500.org.

[4] COCHRAN C A. Analyzing FLUENT CFD models and data to develop fundamental codes to assess the effects of graphite oxidation in an HTGR air ingress accident [D]. Cambridge: Massachusetts Institute of Technology, 2010.

[5] FAVERO J L, SECCHI A R, CARDOZO N S M, *et al.* Viscoelastic fluid analysis in internal and in free surface flows using the software OpenFOAM[J]. Computers & Chemical Engineering, 2010, 34 (12): 1984 – 1993.

[6] IFEANYI P. EGWUTUOHA, DAVID L, *et al.* A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems[J]. Journal of Supercomputing, 2013, 65(3): 1 – 25.

[7] YANG X J, WANG Z Y, XUE J L, *et al.* The reliability wall for exascale supercomputing[J]. IEEE Transactions on Computers, 2012, 61(6): 767 – 779

[8] YU W K, HUANG W, PANDA D K. Application-transparent checkpoint/restart for MPI programs over InfiniBand[C]// Proceedings of the 2006 International Conference on Parallel Processing. Washington, DC: IEEE Computer Society, 2006: 471 – 478.

[9] DALY J T. A higher order estimate of the optimum checkpoint interval for restart dumps[J]. Future Generation Computer Systems, 2006, 22(3): 303 – 312.

[10] YANG X, WANG P, FU H, *et al.* Compiler-assisted application-level checkpointing for MPI programs[C]// Proceedings of the 28th International Conference on Distributed Computing Systems. Piscataway: IEEE, 2008: 251 – 259.

[11] LI K, NAUGHTON J, PLANK J. Low-latency, concurrent checkpointing for parallel programs[J]. IEEE Transactions on Parallel and Distributed Systems, 1994, 5(8): 874 – 879.

[12] AMINA G, THOMAS R, ELISABETH B, *et al.* Uncoordinated checkpointing without domino effect for send-deterministic MPI applications[C]// Proceedings of the 25th IEEE International Parallel and Distributed Processing Symposium. Piscataway: IEEE, 2011: 989 – 1000.

[13] XIE M, LU Y, LIU L, *et al.* Implementation and evaluation of network interface and message passing services for TianHe-1A supercomputer [C]// Proceedings of the 19th Annual Symposium on High-Performance Interconnects. Piscataway: IEEE, 2011: 78 – 86.