一种基于检查点的卷回恢复与进程迁移系统*

汪东升 沈美明 郑纬民 裴 丹

(清华大学计算机科学与技术系 北京 100084)

E-mail: wds@tsinghua.edu.cn

摘要 ChaRM是一种并行程序后向故障恢复与进程迁移系统.它不仅实现了对工作站机群系统瞬时故障的恢复,而且通过检查点设置时的Mirror存储技术和进程迁移技术,实现了对机群系统结点永久故障的恢复,并支持系统软硬件的在线维护、处理机资源的排他/限时使用和动态负载平衡等功能.文章主要介绍ChaRM系统的检查点设置与回卷恢复、进程迁移等实现技术,并给出了部分性能评测结果.

关键词 检查点,回卷恢复,进程迁移,工作站机群.

中图法分类号 TP393

现代航天、航空、邮电、石油等应用领域对计算机系统的性能和可靠性提出了越来越高的要求.并行处理是提高计算能力、满足不断增长的应用需求的有效途径.而容错技术是提高计算可靠性的重要保证.工作站机群NOW(networks of workstations)是利用现有的工作站资源,通过高速网络以某种结构连接起来而构成的并行系统.由于它具有用户投资风险小、结构可扩展性好、用户可继承原有的软件资源、编程方便并且构造简单等特点,目前已成为并行处理发展的主流方向之一.而NOW随着规模的不断扩大,它在计算过程中发生故障的几率也会呈指数增长,同时由于NOW通常为多用户使用,结点等资源此时可用,而另一时刻则可能不再可用,具有较大的可变特性.系统在发生下列异常事件时会导致本次并行计算的彻底失败,此前的大量计算不再可用,如:①运行不同应用程序的其他用户发生的异常事件;②某机器所有者需要独占CPU资源,禁止其他用户共享该结点资源;③异常关机;④结点(瞬时/间歇/永久)故障;⑤系统软件升级,结点更换等维护操作等等.

大规模科学工程计算任务执行时间都较长,一旦某计算结点发生上述异常事件,将导致系统运行失败,程序不得不从头开始执行.为了避免系统在发生上述事件后由于从头开始执行而引起计算上的大量浪费,充分提高机群系统的可用性,在系统正常运行的适当时刻设置检查点(Checkpoint),保存系统当时的规范状态,并对各进程进行相关性跟踪和记录.系统发生故障后,将相关进程回卷(Rollback)到故障前系统一致性状态(检查点),经过状态恢复后从该检查点处重新执行,而不是从程序开始执行,从而节省了大量重复计算时间,充分体现机群系统的并行性能.这种基于检查点的后向恢复技术不仅可以对系统瞬时/间歇故障进行自动恢复,也是恢复未知故障——在某一应用设计过程中未预料到的故障的唯一手段.[1]

ChaRM(checkpoint-based rollback recovery and process migration)是为了实现工作站机群系统的容错性能和实现负载平衡而研制的并行程序运行回卷恢复及进程迁移软件.ChaRM不仅实现了对机群系统瞬时故障的恢复,而且通过检查点设置时的Mirror存储技术和进程迁移技术,实现了对机群系统结点永久故障的恢复并支持系统在线维护、处理机资源的排他或限时使用和动态负载平衡.

---

* 本文研究得到国家863高科技项目基金资助.作者**汪东升**,1966年生,博士后,副教授,主要研究领域为并行处理,容错计算.**沈美明**,女,1938年生,教授,博士生导师,主要研究领域为并行/分布与智能计算机系统.**郑纬民**,1946年生,教授,博士生导师,主要研究领域为并行/分布系统,并行编译.**裴丹**,1973年生,硕士生,主要研究领域为进程迁移,后向故障恢复技术.

本文通讯联系人:汪东升,北京 100084,清华大学计算机科学与技术系

本文1997-10-06收到原稿,1998-01-19收到修改稿

1 基本设计思想与设计目标

实现并行/分布系统容错的方法有多种：使用特殊的容错硬件是有效的，但不易于加到现有的机群系统中去；与应用相关的方法和原子事物处理需要特殊程序设计模型（可能与消息传递模型不兼容）来支持；活动冗余的方法适用于实时系统，但它需要使用额外的处理器，因此这些方法都不能有效地应用于工作站机群系统. 检查点设置与回卷恢复 CRR（checkpointing and rollback recovery）技术作为一种后向恢复技术，通过在系统正常运行过程中设置检查点，保存系统运行时的一致性状态，并通过程序的回卷（到最近的检查点）来实现对系统故障的恢复. 同时，检查点机制也是实现进程迁移、作业交换、并行程序调试的有效手段.

现有的并行/分布系统 CRR 技术大致可以分为独立方式和协调方式两类[2~4]，在前一种方式中，各进程独立设置检查点而不需要相互间的协调，最大限度地满足各进程的自主特性，但易引起多米诺效应. 消息记录（Message-logging）是避免多米诺效应的有效手段. 它利用分段确定性模型 PDM（piecewise deterministic model），通过消息记录和确定性重放机制避免了多米诺效应的发生. 协调方式可以避免多米诺效应，但是系统为了建立检查点，需要许多控制消息，同步建立检查点的通信代价较大. 对于一般系统来说，通常认为在系统正常操作过程中基于记录的独立检查点设置方式比协调的检查点设置方式代价小，因为它避免了为建立系统一致性状态而导致的同步代价. 对于通信速度较慢的系统来说，情况确实如此，因为此时系统在正常操作时的消息记录代价相对较小. 而对于并行机群系统来说，由于网络通信速度的不断提高，用于消息记录的代价反而比协调建立检查点的代价要大. 对于计算密集型并行应用来说，许多研究和实验表明，协调检查点设置方法显示出较好的性能，并且具有回卷距离短、实现简单、不需要对进程执行的分段确定性假设，且同时能容忍多个进程故障的优点. 当系统发生故障而需要回卷恢复时，系统相关结点只需回卷到其上一个检查点，由于系统各结点只需保存一个检查点，因此，空间回收（Garbage Collection）算法简单.

由于协调的检查点设置方式具有回卷深度小、恢复算法简单、存储开销小、空间回收简单等优点，CoCheck[5]，MIST（migration and integrated scheduling tools）[6]等系统都选用了这种方式. ChaRM 也采用了协调的检查点设置技术. ChaRM 系统具有以下功能和特点：

- 自动恢复结点瞬时/永久故障；
- 采用并行和增益的检查点设置方式，减小系统正常运行代价；
- 同时支持容错与进程迁移；
- 在用户级实现，使用标准 UNIX 系统调用，不修改 UNIX 内核，不需要额外硬件支持，适用于不同 UNIX 版本；
- 对用户程序透明，系统以运行库形式提供给用户. 用户使用时只需将其应用程序目标文件与 ChaRM 提供的运行库链接，即可用命令方式或自动方式使用系统的容错及进程迁移等功能；
- 在并行计算环境通信库之上实现，具有一定可移植性.

2 ChaRM 系统总体结构

为了与并行机群系统其他子系统相吻合，程序运行回卷恢复系统选用 PVM 作为并行程序开发环境. 系统由 4 个模块构成：

- 总控模块（ChaRM-manager）：负责协调管理系统检查点设置、系统监测、进程迁移与程序回卷等工作；
- 检查点设置与回卷恢复模块（CRR）：定时地协调并行进程设置检查点，保存系统当时的全局一致性状态，在系统发生异常事件时，负责并行程序的回卷恢复；
- 系统监测模块（WatchD）：通过周期性协议，检测虚拟机各种异常以及各 daemon 进程状态，并及时向 ChaRM-manager 报告，以便由 ChaRM-manager 进行决策并进行相应处理；
- 进程迁移与文件镜像模块（Mig&Mir）：负责系统的进程迁移，检查点文件的存储镜像等工作.

系统总体结构如图 1 所示. 系统在正常操作时，通过计时（Timer）机制周期性地通知系统执行检查点设置操

作,由 ChaRM-manager 协调系统各进程同步设置检查点.当 WatchD 检测到故障时,由 error 事件触发程序回卷到故障前保存的检查点位置,通过信息的恢复(Restore)和程序的重放(Replay)以及必要时的进程迁移,达到故障恢复的目的.msg-filter 用于监测在建立检查点过程中传输通道中的消息(in-Transit),并对其进行消息记录(Message-logging).

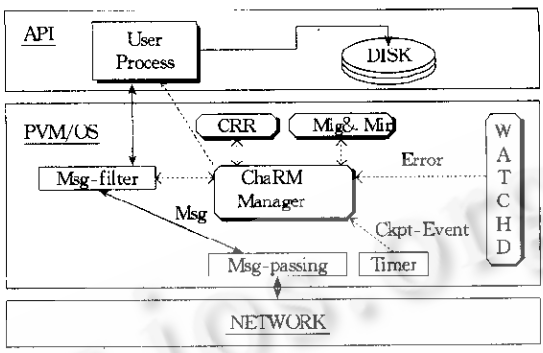

图1 ChaRM系统逻辑结构

3 ChaRM 系统 CRR 技术与进程迁移机制

3.1 ChaRM 系统检查点设置和回卷恢复

由于 ChaRM 系统在通信协议之上实现,满足通信通道可靠的假设,其检查点设置时必须考虑消除引起系统状态不一致的孤儿消息和中途消息.为此,一个可能的方案是选择适当的发起时机来避开包含中途消息的通信通道状态.但由于对通信通道状态的监测无法在通信库之上实现,而且我们采用定时的检查点插入方式,发起时机的选择无法确定.另一个可能的方案是在检查点设置之外辅以对中途消息的记录,回卷恢复时对这些消息进行重放.但在检查点发起时刻,这些中途消息可能停留在物理网络、操作系统缓冲、通信库缓冲器之中的任何一处,在不介入操作系统和通信库的前提下,直接访问这些地方以获取中途消息是不可能的.

ChaRM 系统的解决方案是,在检查点发起时刻,若通信通道状态包含中途消息,我们将利用通信机制本身"驱赶"中途消息,同时延迟真正的检查点设置,最终达到在无孤儿消息和丢失消息的全局状态下建立检查点的目的.在实现上就是利用 PVM 通信通道的 FIFO 性质,在通信库之上实现一种特殊的两步协议.系统以不依赖于任何并行计算环境的专用进程 cinit 作为协议的管理进程,更好地支持了在不同并行环境间的可移植性.协议如下.

第 1 步:同步

(1.1) cinit 进程 检查点设置的协调进程 cinit 向所有应用进程发出同步启动信号 CKPT_SIG,

(1.2) 应用进程 每个应用进程收到 CKPT_SIG 后,向其他所有应用进程发送"准备就绪"消息 RDY_ MSG,

(1.3) 应用进程 每个应用进程暂停计算,持续接收向它发送的消息,直至收到来自其他所有应用进程的 RDY_MSG.在此过程中,如果收到除 RDY_MSG 以外的其他消息(被"驱赶"而来的中途应用消息),则将它保存到预先指定的接收缓冲中,

(1.4) 应用进程 退出 PVM.

第 2 步:设置检查点.其组成的全局检查点状态具有一致性.

(2.1) 应用进程 设置检查点,

(2.2) 应用进程 重新加入 PVM,

(2.3) 应用进程 向 cinit 发送 RJN_MSG(old-tid,new-tid)消息,

(2.4) cinit 进程 根据各进程的 RJN_MSG,重新组织任务匹配表(mapping-table),

(2.5) cinit 进程 向各应用进程广播 mapping-table,

(2.6) 应用进程 接收到 mapping-table 后,调整其各自的任务匹配表,进入正常计算阶段.

由于协议利用协调进程 cinit 向所有应用进程发出同步启动信号 CKPT_SIG,以协调进行各进程检查点的同步设置,根据通信通道的 FIFO 语义,不会存在孤儿消息.同时,当收到来自某个应用进程的 RDY_MSG 时,可以推断从彼进程到本进程的单向通信通道已被清空,其上的所有中途消息(若存在),已被驱赶至本进程并已妥善保存.因此,当(1.3)步结束后,各应用进程可以肯定所有从其他应用进程至本进程的通信通道上的中途消息已被驱赶而提前接收,全局状态则为各个应用进程状态(检查点)组成的集合.因此,可以证明该两步协议设置的检查点是一致性全局检查点.ChaRM 也采用类似的两步协议实现了同步回卷,此处从略.

3.2 ChaRM 系统进程迁移机制

进程从源机迁移至目的机的过程,实质上是利用 CRR 机制实现的进程在源机的挂起和在目的机的恢复过程.迁移发起时,系统首先在目的机上为源机上迁移进程创建一个骨架进程,接着,迁移进程设置检查点,并将检查点内容直接通过 UNIX 中套接字(Socket)通道传给骨架进程.随后,迁移进程自动终止.骨架进程从通道中读出检查点内容,即刻将自身状态恢复为检查点保存的状态,并从此继续运行.这样,从总体效果来看,迁移进程从源机迁移到了目的机上执行.

为了保证系统在迁移过程中的状态一致性,应该避免迁移发起时各应用进程发往迁移进程的中途消息,同时避免迁移期间非迁移进程发往迁移进程的应用消息.迁移过程简单描述如下:

第 1 步:同步.迁移发起时,为避免发往迁移进程的中途消息,我们使用驱赶机制将这些消息驱赶至迁移进程并对其提前接收.由于只有各非迁移进程 NMP(non-migration process)至迁移进程 MP(migration process)的消息通道需要清空,方法与协调式检查点设置略有差异.

(1.1) 各 NMP 向所有 MP 进程发送 RDY_MSG 消息;

(1.2) 各 MP 等待来自所有应用进程的 RDY_MSG.在此期间,若接收到非 RDY_MSG 的消息,则表明它是被驱赶而来的应用消息,将它保存到特定缓冲中.各 NMP 进程则照常计算.

第 2 步:迁移.此时,发往 MP 的中途消息已全部被提前接收.各 MP 建立检查点,并将检查点信息通过 socket 直接传送到目的机,在目的机上恢复.各 NMP 则照常执行.迁移完毕后,恢复进程将任务号变化通知 cinit,cinit 将指导所有进程更新其任务号映射表.

在此两步协议中,各 NMP 除了在迁移发起时向所有 MP 发送 RDY_MSG 消息,以及在迁移结束时在 cinit 指导下更新映射表外,一直处于正常的计算过程中.

一般为了实现进程迁移,要引入十分复杂的消息转发和排序机制,并为此修改 PVM daemon 进程的实现来解决消息丢失问题,而 ChaRM 使用了"延迟发送"策略来支持异步迁移.可见,迁移期间发向迁移进程的应用消息,由于延迟到映射表更新后再实际发送,将被正确地发往目的机上的恢复进程.并且,延迟发送因消息延误给应用程序的运行带来的附加开销很小,这是由于:一方面,发送消息并不在执行的关键路径上,因此发送延迟对发送进程(NMP)的计算过程无直接影响;另一方面,由于接收进程(MP)直到迁移完毕映射表更新后,才继续计算过程,接收应用消息,因此,将发送延迟到迁移完毕后立即执行,不会因为等待消息而明显增加执行时间.

3.3 检查点镜像与结点永久故障恢复

通过改变检查点存放的位置,我们可以实现不同的容错模式.将检查点保存到本地磁盘上只能容忍瞬时故障或间歇故障.而借用冗余磁盘阵列 RAID(redundant arrays of inexpensive disks)技术的容错思想,我们可以灵活选择检查点存放位置而恢复单个或多个永久故障.我们在检查点设置时采用了镜像平均存储策略,即各结点除将其上进程的检查点保存到本地磁盘外,同时还启动后台进程,将这些检查点文件信息均衡地镜像到其他结点磁盘上.这样,如果某结点发生永久故障,其上正在执行的应用程序的检查点信息可在其他正常结点上获得,并从检查点处重加载该应用程序运行.若要容忍 $m(m>1)$ 个结点故障,需且仅需将每个结点上每个进程的检查点信息除存在本地磁盘外,还镜像到其他 m 结点的磁盘上.可见,容忍 m 个结点永久故障的检查点镜像方式比一般的检查点本地存放方式增加了 m 倍的空间开销.这种平均镜像存储技术不仅保证了系统对结点永久故障的恢复,而且避免了恢复后引起的负载不平衡现象.

4 性能测试及与相关工作比较

在这里,我们对4个有代表性的科学计算应用进行检查点设置和回卷恢复性能测试.我们使用的测试平台是由4台SUN SPARC20工作站(32M内存,1G硬盘,Solaris2.4操作系统)通过10M Ethernet连接而构成的机群系统.我们用来考察ChaRM系统的性能指标主要是检查点设置的时间开销和空间开销(平均每次检查点设置各进程建立的检查点文件大小的总和).同时,为了考察检查点保存的主要优化方式——写时拷贝对性能的改善,我们还测出了以上各程序在使用写时拷贝优化方式下的检查点设置时间开销.表1比较了在使用相同的检查点间隔和建立了相同个数的检查点的情况下,非优化方式与写时拷贝优化的时间开销.可见,写时拷贝优化由于使检查点写入本地磁盘与计算并行,不同程度地减少了检查点设置的时间开销.测试结果表明,在4个应用程序的不同运行情况下,回卷恢复到不同的检查点都能得到正确的计算结果;检查点设置带来的运行时间增加都小于5%,平均每次检查点设置的时间开销小于1%;平均每次检查点设置的空间开销随应用程序和其规模的不同而变化很大.

表1 非优化检查点设置与写时拷贝检查点设置开销比较

程序名称	无检查点时4台运行时间(s)	检查点间隔/空间开销(s/M字节)	非优化检查点设置				写时拷贝检查点设置			
			4台运行时间(s)	运行时间增加(%)	检查点个数	单位检查点设置时间开销(‰)	4台运行时间(s)	运行时间增加(%)	检查点个数	单位检查点设置时间开销(‰)
矩阵幂 180*180,300	718.43	100/8.9	744.91	3.69	7	5.27	726.58	1.13	7	1.62
矩阵幂 512*512,50	2 883.83	300/27.9	2 970.24	3.00	9	3.33	2 933.66	1.73	9	1.92
偏微分方程 $N=2^8,T=10000$	1 248.34	100/13.1	1 281.06	2.62	12	2.18	1 270.21	1.75	12	1.46
偏微分方程 $N=2^9,T=1000$	519.33	100/31.7	535.20	3.05	5	6.11	530.30	2.11	5	4.22
哈达玛变换 10000	8 774.11	600/9.5	8 939.60	1.89	14	1.35	8 848.05	0.84	14	0.60
EMBAR(16^7)	709.21	100/6.4	716.89	1.08	7	1.55	709.91	0.10	7	**0.14**

我们将基于检查点设置的并行程序回卷恢复和进程迁移系统ChaRM,CoCheck[5],MIST[6]进行了比较.如表2所示,ChaRM不仅继承了CoCheck和MIST对用户透明、在用户级实现、采用协调的检查点设置技术等有利的设计和功能特点,而且结合了CoCheck在通信库之上实现以及MIST支持异步迁移的优点.同时,ChaRM的CRR机制管理进程不依赖于并行计算环境实现,更好地支持了在不同并行环境之间的可移植性.并且ChaRM系统实现了对结点N-永久故障的容忍和容忍系统在检查点设置和故障恢复过程中出现的故障(简称CRR故障)的功能.

表2 基于检查点设置的并行程序回卷恢复和进程迁移系统比较

	ChaRM	CoCheck	MIST
对用户透明	√	√	√
在用户级实现	√	√	√
在通信库之上实现	√	√	×
协调式检查点设置	√	√	√
异步进程迁移	√	×	√
容忍CRR故障	√	×	×
结点N-永久故障的容忍	√	×	×
管理进程不依赖于并行计算环境	√	×	×

参考文献

1 Plank J S, Kim Y *et al*. Fault-tolerant matrix operations for networks of workstations using diskless checkpointing. Journal of Parallel and Distributed Computing, 1997,43(2):125～138

2 Plank J S, Beck M, Kinsley G. Libckpt: transparent checkpointing under Unix. http://www.cs.utk.edu/～plank/plank/papers/USEUIX-95w.html

3 Chandy K M, Lamport L. Distributed snapshots: determining global states of distributed systems. ACM Transactions on Computer Systems, 1985,3(1):63～75

4 Koo R, Toueg S. Checkpointing and rollback-recovery for distributed systems. IEEE Transactions on Software Engineering, 1987,13(1):23～31

5 Stellner G, Pruyne J. Resource management and checkpointing for PVM. In: Hermes P P eds. Proceedings of the 2nd Europe PVM User Group Meeting. 1995. 131～136

6 Casas J, Clark D *et al*. MIST: PVM with transparent migration and checkpointing. http://www.cse.ogi.edu/DISC/projects/mist/

A Checkpoint-based Rollback Recovery and Process Migration System

WANG Dong-sheng　SHEN Mei-ming　ZHENG Wei-min　PEI Dan

(*Department of Computer Science and Technology　Tsinghua University　Beijing*　100084)

Abstract　ChaRM is a checkpoint-based backward fault recovery and process migration system. It is designed to recovery the fault of NOWs (networks of workstations) by checkpointing and rollback recovery. It offers functions of on-line software and hardware maintenance, process migration and load balance, etc. ChaRM is able to run on NOWs that change over time due to failure, load or availability. As long as there is at least one node alive in the cluster, the computation will complete in an efficient manner. The checkpointing, rollback recovery and process migration techniques, and some performance evaluation results are discussed in this paper.

Key words　Checkpointing, rollback recovery, process migration, NOWs (networks of workstations)