可恢复的软件 DSM 系统 JIACKPT*

张福新+, 章隆兵, 胡伟武, 唐志敏

(中国科学院 计算技术研究所,北京 100080)

JIACKPT: A Recoverable Software Distributed Shared Memory System

ZHANG Fu-Xin+, ZHANG Long-Bing, HU Wei-Wu, TANG Zhi-Min

(Institute of Computing Technology, The Chinese Academy of Sciences, Beijing 100080, China)

+ Corresponding author: Phn: +86-10-62565533 ext 9320, E-mail: fxzhang@ict.ac.cn, http://www.ict.ac.cn

Received 2004-07-07; Accepted 2004-10-10

Zhang FX, Zhang LB, Hu WW, Tang ZM. JIACKPT: A recoverable software distributed shared memory system. *Journal of Software*, 2005,16(2):165−173. http://www.jos.org.cn/1000-9825/16/165.htm

Abstract: Software distributed shared memory (DSM) system has constructed a virtual shared memory abstract on cluster, which combines the programmability of shared memory and fine scalability of cluster. So it is widely studied. Software DSM system is easy to fail because it is a distributed system, some kinds of fault tolerance are necessary for it to be more practical. A recoverable and portable software DSM system, JIACKPT (JIAjia with ChecKPoinTing), has been designed and implemented to tolerate the fault of system. JIACKPT, based on JIAJIA, has adopted the checkpointing technology. By maintaining the strict global consistent state and using some optimization techniques, JIACKPT has gotten high performance. The experimental results on an 8-node PC cluster show that the checkpoint overhead is less than 10% of the whole execution time when checkpoint is done once per minute. JIACKPT also has good portability and can run on several operating systems, such as Linux, Solaris, etc. JIACKPT is a practical recoverable software DSM system.

Key words: software distributed shared memory system; checkpoint; global consistent state; JIAJIA

摘 要: 软件 DSM(distributed shared memory)系统在机群上构造了共享存储编程环境,结合了共享存储的易编程性和机群的可扩展性,引起了广泛的研究.由于软件 DSM 系统是一个分布式系统,系统失败风险大,需要实现容错技术以促进其实用化.利用用户级检查点技术,在支持域存储一致模型的软件 DSM 系统 JIAJIA 的基础上,设计并实现了一个可恢复的高可移植的软件 DSM 系统 JIACKPT(JIAjia with ChecKPoinTing).由于采用适合软件 DSM 系统的强全局一致状态以及多种优化措施,JIACKPT 易于实现且获得很好的性能.在一个 8 节点的 PC 机群上的应用测试表明,即使每分钟做一次检查点,大部分应用的检查点开销也小于 10%.此外,JIACKPT 还具有高可移植性.这些都表明 JIACKPT 已经成为一个比较实用的系统.

* Supported by the National Natural Science Foundation of China under Grant No.60303016(国家自然科学基金)

作者简介: **张福新**(1976−),男,福建永定人,博士生,主要研究领域为机群计算,微处理器系统结构设计和性能评估;**章隆兵**(1974−),男,博士,主要研究领域为机群计算,微处理器设计;**胡伟武**(1968−),男,博士,研究员,博士生导师,主要研究领域为高性能计算机体系结构,并行处理,VLSI 设计;**唐志敏**(1966−),男,博士,研究员,博士生导师,主要研究领域为高性能计算机体系结构,CPU 芯片,网络并行计算.

关键词: 软件 DSM 系统;检查点;全局一致状态;JIAJIA
中图法分类号: TP311 文献标识码: A

随着微处理器和高速网络技术的发展,以工作站网络为代表的机群正逐渐成为主流的并行计算平台.软件 DSM(distributed shared memory)系统在机群上提供了共享存储的编程模型,它组合了共享存储的易编程性和机群的可扩展性,引起了广泛的研究.典型的软件 DSM 系统包括 Midway[1],TreadMarks[2],CVM[3],JIAJIA[4]等.然而,机群上的软件 DSM 系统是分布式系统,这不可避免地增大了系统失败的风险.对于分布式系统而言,其平均失败时间反比于系统中的节点数.软件 DSM 程序对通信系统的密切依赖更增加了它们失败的风险.大型的 DSM 程序需要容错能力来保证其顺利执行.在软件 DSM 系统中实现容错能力,成为该领域研究的一个热点[5−9].

研究人员提出了多种容错 DSM 技术,大部分是针对某种存储一致性模型的特定技术.早期技术多数是针对严格的顺序一致性(sequential consistency)[5−7],近期的一些技术则针对放松的一致性模型,如懒惰的释放一致性(lazy release consistency)[8]、因果一致性(causal consistency)[9]等.这些技术常常假定一个进程的检查点随时可以获得,不考虑开销,并在此基础上针对 DSM 协议做理论讨论或模拟分析.而实际上,用户级的检查点并不很容易获得,而且开销往往比较大,在要求很好地适应比较复杂的程序时尤其如此.软件 DSM 系统本身往往比较复杂.首先,它们是一个分布式系统,大量使用进程间通信;其次,它们常常直接操作页面状态来构造虚拟共享内存,对操作系统底层细节依赖较多;此外,很多系统还使用线程.一般的检查点函数库,例如,著名的 Libckpt[10]不足以应付这些情况.到目前为止,实用的具有容错能力的软件 DSM 系统并不多.

本文的贡献是针对支持域一致性(scope consistency)的软件 DSM 系统 JIAJIA,利用用户级并行检查点技术设计并实现一个高可移植的、可恢复的软件 DSM 系统 JIACKPT.这使得软件 DSM 程序因故障中止后可以恢复到最近的检查点继续执行,从而促进软件 DSM 系统真正走向实用.JIACKPT 具有高可移植性,目前可以运行在 Linux,Solaris 和 AIX 操作系统上.

本文第 1 节简要介绍软件 DSM 系统 JIAJIA.第 2 节讨论适合软件 DSM 系统的强全局一致状态.第 3 节是 JIACKPT 的设计与实现.第 4 节是性能评价.最后是相关工作讨论和未来工作.

1 JIAJIA 简介

软件 DSM 系统通过软件在机群上构筑了一个虚拟共享存储空间.JIAJIA 是本实验室开发的一个基于 Home 的软件 DSM 系统[4].JIAJIA 系统有两个显著特征:① 大内存特点.由于采用类似于 NUMA(nonuniform memory access)的存储器组织方式,能够把多个机器的存储器组织起来形成一个更大的存储空间.例如,在 AMD64 的机群上,JIAJIA 可以提供几近无限的 64 位共享空间.② 采用一种基于锁的新型一致性协议.该协议支持域存储一致性模型和写无效的传播策略,并采用多写协议来避免假共享.此外,JIAJIA 还实现了写向量、预取、Home 迁移等性能优化技术.JIAJIA 获得了与消息传递系统相当的性能[11,12].

由于要结合 JIAJIA 特点来实现检查点,本文比较关注 JIAJIA 的通信子系统.考虑到性能因素,JIAJIA 的通信子系统建立在 UDP/IP 基础上,具有以下特点:① 可靠性.利用超时重发和序列号机制,保证消息传递的可靠性.② 保序性.先调用发送过程的消息一定先发送,先收到的消息一定先服务.JIAJIA 的通信模式是请求/答复式,消息分为请求和答复两类,一个请求消息对应一个(或者多个)答复消息.

2 适合软件 DSM 系统的强全局一致状态

在分布式系统中实现检查点的关键问题是如何决定一个分布式系统的全局一致状态.因此,在软件 DSM 系统中实现并行检查点功能,关键是确定一个适合的全局一致状态.本节主要讨论适合软件 DSM 系统的强全局一致状态.

2.1 分布式系统的强全局一致状态

一个分布式系统的全局一致状态是通过分布式系统的某种正确执行可以到达的状态.Chandy 和

Lamport[13]正式地定义了分布式系统一致状态的概念,给出了一个用于决定全局一致状态的算法.该算法指出,如果一个进程状态的集合满足:若进程间传递的消息已记录在接收方的状态中,则也一定已记录在发送方的状态中,那么这个集合构成一个全局一致的分布式系统状态.如果分布式系统中每个进程的检查点记录的状态构成一个全局一致的分布式系统状态,我们一定可以从中恢复出可继续正确运行的状态来.但是,必须有某种方法记录已发送的消息,因为全局一致的状态可能包含了消息已发出但未接收的情况.

下面,我们扩展分布式系统的全局一致状态的定义来描述强全局一致状态.如果一个分布式系统的进程状态的集合满足:若进程间传递的消息已记录在接收方,则也一定已记录在发送方;若已经记录在发送方,则也一定已记录在接收方,那么这个进程状态集合构成强全局一致状态.从以上定义可见,强全局一致状态是全局一致状态的一个子集.强全局一致状态的好处在于没有"正在传送中"的消息.显然,如果一个检查点记录了强全局一致的状态,可以直接恢复出能继续运行的正确状态,则无须记录消息.

例如,图 1 中用时间图来表示一个并行系统的执行.横线表示各进程执行的时间轴,箭头表示消息,P1,P2,P3,P4 是程序的 4 个进程,a,b,c 分别表示分布式系统的一个全局状态(由每个进程在交点处的状态集合构成).根据上述定义,a 是不一致的状态,b 和 c 是全局一致状态,且 b 不是强一致状态,c 是强一致状态.因为在 a 中进程 P1 处于收到消息 m1 后的状态,而发送该消息的 P2 处于发送消息之前的状态.如果系统从 a 状态重新启动,那么进程 P1 将会两次收到 m1 导致运行不正确.这种不正确执行就是由于不一致状态造成的.另外,虽然 b 是一致状态,但必须以某种方式记录消息 m3,m4,m5,否则从 b 重新执行将导致消息丢失.最理想的情况就是 c 状态,c 是强一致状态,从 c 重新执行,无须记录任何消息.可见,记录强全局一致状态的检查点容易实现.

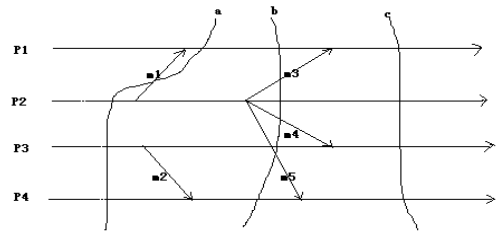

Fig.1 Global status of a distributed system
图 1 分布式系统的全局状态

2.2 两类并行检查点系统

对使用检查点(或加消息记录)实现分布式系统的容错这一问题,研究人员已经进行了大量的研究,重点是如何记录检查点以建立一个在出错后可以恢复的全局一致状态.根据什么时候和怎样建立一个一致状态,现有的系统可以分成以下两类:① 独立的检查点+消息记录.主要思想是分布式系统的各进程在做检查点和记录消息时不互相同步,独立进行.在错误恢复时,每个进程通过记录的消息决定与其他进程的依赖关系,通过回退和消息重放最终决定一个可以继续运行的一致状态.优点是减少了检查点和记录消息时的通信开销,缺点是需要消息记录的开销,恢复复杂,而且可能引起多米诺现象(通信依赖引起各进程回退过多甚至返回起始点而失去恢复的意义).这类系统通常假定错误很少发生,对各进程所在机器物理上非常分散的情况比较有利.② 协调记录一致检查点.这类系统在检查点阶段各进程之间进行协商,试图建立一个全局一致状态.这样,进程的恢复变得比较简单,检查点的次数和空间开销也比较容易控制.缺点是建立一致状态需要同步开销,也同样可能需要记录消息以备恢复.

2.3 适合软件DSM系统的检查点算法

软件 DSM 系统中存在大量的消息传递,其性能对通信速度的依赖性很强,常使用快速网络.这样,需要大量消息记录的独立检查点不利于系统性能,显然不适宜用在这种环境中.而由于通信速度快,进程间协调记录一致检查点时的同步开销小,与记录进程状态时间相比完全可以忽略.因此,协调记录一致检查点的系统比较适合软件 DSM 环境.协调记录一致检查点的系统又有两种,其中一种仍然需要一些消息记录(如 Chandy-Lamport 算法)而允许一些消息"穿过"一致状态点的连线(图 1 中 b),即记录全局一致状态;另一种则更"同步":它通过更进一步的同步协调,保证在任何发出消息都已经到达目的时再记录检查点,从而彻底摒弃了消息记录的需要,即记录强全局一致状态.对于软件 DSM 系统来说,两者的性能差别不大,具体的实现和环境足以影响比较结果.例如,一般认为后者需要同步等待,可能会比前者稍慢,但在软件 DSM 环境中,一个全局同步通常只是极短的一个过程,这

时 Chandy-Lamport 算法中在检查点 $O(n^2)$的额外消息的开销可能还更大.本文中采取同步等待的做法,额外消息只有 3n 个(n 为分布式系统中的进程数).可见在软件 DSM 系统中,记录下强全局一致状态来实现检查点是比较合适的,既易于实现且性能也不差.软件 DSM 系统中实现检查点的主要算法步骤是:先利用全局同步方法保证系统到达强全局一致状态,记录下强全局一致状态,出错恢复时从强全局一致状态开始执行.本文在 JIACKPT 使用了该检查点算法,实现了一种取得强一致状态的检查点并证明了其正确性.

3 JIACKPT 的设计与实现

JIACKPT 的主要设计目标包括:① 透明性.尽量对用户透明,提供接口使得 JIAJIA 程序可以不做修改地使用检查点;② 高可移植性.使得可以在多种操作系统上运行,例如 Linux,Solaris 等.这就要求在设计和实现时尽量避免使用某种操作系统的特性,而使用通用的方案;③ 高性能.一方面,紧密结合 JIAJIA,充分利用 JIAJIA 的特性,实现高效检查点;另一方面,采用先进性能优化措施来提高检查点的效率,包括增量技术、采用专门的进程完成检查点功能等.采用增量技术,使得只需记录自上一次检查点来进程写过的数据,有效减少需要写到检查点文件中的数据,而代价是检测被写过页和合并多个检查点文件以取得完整的进程状态的时间开销和多个检查点文件的空间开销.采用专门的进程完成检查点功能,可以使得主进程的计算过程与检查点记录过程并行执行,提高程序性能.

3.1 JIACKPT的界面

考虑到用户透明性和兼顾用户使用的灵活性,我们开发了函数调用接口和配置文件两种界面来方便用户使用检查点功能.函数调用接口的好处是,用户可以通过在程序中调用函数以灵活设置做检查点的时机.用户也可以直接通过配置文件来自动启动检查点功能,无须修改程序.

3.1.1 函数调用接口

JIACKPT 提供了下列函数调用:① int jia_ckpt(int flag):显式激活检查点例程.Flag 取值为 CKPT_PERIOD,指示检查点做完之后继续原任务,取值为 CKPT_QUIT,则做完后程序终止.如果成功,返回 0,否则返回−1.失败可能由检查点例程出现异常或检查点没有使能或不满足时间间隔要求导致.② int ckpt_setinterval(uint *interval,uint *mininterval):设置检查点的间隔时间参数.如果用户选择了自动做检查点,*interval 用于指定从某个检查点被提交到下个检查点的间隔;*mininterval 用于指定两个检查点的最小时间间隔.如果 *interval<=*mininterval,检查点功能被禁止.原值在两个参数中返回.

3.1.2 配置文件

JIACKPT 给用户提供了一个配置文件:.ckptrc.每个用户程序可以有不同的配置.该文件必须放在用户程序运行目录下.配置文件格式见表 1.从表中可以看出,用户可以通过设置 Enabled==yes 和 Autocheck==yes 以及参数 Interval 来使得程序每隔一段时间自动做检查点.

Table 1 Format of configuration file

表 1 配置文件格式

Name	Meaning	Value	Default
Enabled	Enable?	Yes/No	Yes
Autocheck	Automatically checkpoint?	Yes/No	Yes
Interval	Time interval between checkpoints	Int (s)	100
Mininterval	Minimal interval	Int (s)	10
Use_fork	Use fork to do check point?	Yes/No	Yes
Incremental	Enable Incremental checkpoint?	Yes/No	Yes
Maxfile	Maximum files	Int	3
Ckpt_filename	Checkpoint file name	String	Ckptlog
Manager	Manager processor	[0,process number−1]	0

表 1 中 Use_fork 表示是否创建(fork)专门的进程来完成检查点功能.Incremental 表示是否采用增量检查点技术.Maxfile 仅当使用增量技术时有效,它表明当增量检查点文件个数达到这个值时,应把这些文件合并成一个完整的检查点文件.Manger 表示检查点管理进程号.

3.2 处理流程

JIACKPT 的处理流程主要包括检查点功能初始化、做检查点和恢复.

3.2.1 初始化

用户程序调用 jia_init(argc,argv)初始化 JIAJIA,我们扩展了 jia_init 以调用 ckpt_init 来完成检查点所需的数据结构初始化工作.初始化的主要步骤如下:① ckpt_init 设置各选项的缺省值,如果.ckptrc 文件在当前目录下,则从文件读出并处理选项值;安装时钟中断 SIGALRM 的处理函数用于定时自动做检查点;初始化检查点状态和一些统计量.② 接着分析命令行参数,取下自身使用的参数,如果发现恢复标记(:recover),则记录这一信息.③ 然后,根据配置文件启动各进程,进程检查恢复标记,如果已置上,开始恢复过程,否则,由检查点管理进程根据配置文件设置全局状态,决定是否设置第 1 个定时器,然后进程继续完成 JIAJIA 的内部数据初始化.

3.2.2 检查点过程

当用户调用 jia_ckpt 或定时时钟到时,指定的检查点管理进程得到一个时钟中断信号.在(可能的)当前正在进行的通信操作完成之后,管理进程响应该信号,由其信号处理例程开始启动一个检查点过程.一个检查点进程又分为 3 步:初始化同步、状态保存、检查点状态决定(同步).

1) 初始化同步

初始化同步的主要任务是等齐所有进程,保证将要记录的进程状态是强一致的.如图 2 所示,主要包括:① 管理进程首先检查当前的全局状态,如果状态不为 OK,则本次检查点失败返回.否则,管理进程向其他进程发出检查点请求,然后等待应答;② 各进程收到检查点请求消息后,向自己发送时钟信号以启动检查点过程(管理进程除外),同样要等到(可能的)当前正在进行的通信操作完成之后才响应该信号.在开始检查点过程后,每个进程把全局状态设置成待决定(pending),再向管理者发送同步消息;③ 管理进程收到所有进程的同步消息后,发出授权消息;④ 各进程收到授权后进入状态保存阶段.

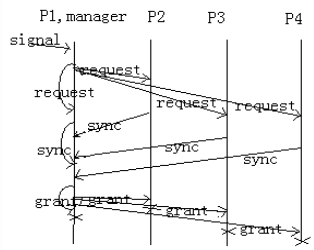

Fig.2 Synchronization process
图 2 初始化同步过程

下面我们来证明这时记录的状态集合将是强一致的,即各进程在记录状态之前发送的消息均已被接收.分两种情况讨论:

① 管理进程在开始保存状态之前发出的消息已被接收,可分两种情况讨论.A:此前发出的请求消息.根据 JIAJIA 通信子系统的消息保序性,在响应中断之前发出的消息一定在管理进程发出 request 消息前就已经发送完毕.又根据可靠性,这些消息已经被接收.B:此前发出的答复消息.即在 request 消息之后发送的对其他进程的请求消息的应答.因为其他进程只可能在响应 request 之前发出请求消息,因此,这些请求消息一定在它们发出同步消息之前发送,且一定在它们的同步消息被服务之前被服务了,因此管理进程在服务这些请求时发出的应答消息早于发出授权消息,故这些应答消息也已经被接收.

② 非管理进程在开始保存状态之前的消息已被接收.A:发出的请求消息.它们一定在响应 request 之前发出,因此早于同步消息的发出,根据消息保序和可靠性,已经被接收.B:发出的答复消息.同前面证明,所有的请求消息都在发出者发出同步消息之前已被接收,而授权消息在收到所有进程的同步消息之后才发出,这个时间比任意一个进程发出同步更晚,因而任一进程收到授权的时间比任一进程发出同步时间要晚,因此所有的请求消息都比授权消息早收到,根据保序性,在开始保存状态(即服务授权消息)之前,请求消息已经被服务,这就是说,它们的答复消息已经发出并被接收.

综上可知,完成同步过程后记录的状态一定是强全局一致的.

2) 状态保存

现代操作系统中进程的状态包括用户地址空间的进程映像、文件、信号、套接字、处理机状态等.进程映像包括正文段、数据段、栈段.一旦得到这些段的起始地址、结束地址、保护状态、类型等信息后就可以写入检查点文件进行保存了.我们对各段的处理是:不保存正文段,程序启动时装入程序会自动恢复;对于其他段,

如果该段没有读许可,则用 mprotect 改为可读.然后,把段信息(起止地址,保护状态等)和内容写入检查点文件.对于文件、信号、套接字、处理机状态也需要采用相应的步骤进行保存,以用于恢复.

状态保存的主要任务是把当前进程的状态保存到文件中,另外还要记录一些统计信息,做一些状态保存的预处理和善后处理.在此阶段,进程首先增加检查点总数,如果使用了增量技术而且文件总数达到指定的 maxfile,则先要把已有的文件合并成一个文件.接着,如果要创建(fork)一个进程来专门完成检查点功能,则 fork 一个进程,该进程调用过程 take_ckpt(具体描述见下面)完成写检查点后退出,否则直接调用该过程.然后,如果使用增量技术,则要把共享内存空间保护成只读(除去原来保护为禁止存取的页)以检测写过的页,统计花费的时间,根据写检查点过程的返回值返回(对 fork 的情况,返回 1).

Take_ckpt 的主要流程如下:① 调用 setjmp 保存恢复返回点,后面步骤在返回值为 0 的分支中;② 取得文件信息、signal 信息;③ 找出地址空间的各个段依次写入检查点临时文件中.在使用增量技术时,临时文件名为 ckpt_filename.jiapid.count.temp,其中 count 根据检查点的总次数和 Maxfile 计算出,取值从 1~Maxfile−1.不使用增量时,名为 ckpt_filename.jiapid.temp.在写各段时,根据用户指定的排除或包含区域、系统可排除的数据(如没有使用的缓冲区等)以及增量技术得到的可排除数据区域,从段中剔除.剩下的片段,按一定的格式存入文件,栈写在最后;④ 最后同步文件.如果过程没有异常,返回 1,否则返回−1(fork 的情况是以该值调用 exit()退出).

3) 检查点状态决定

如果没有使用 fork,那么 2)返回时,进程状态保存已经实际完成,所以立即进行检查点状态报告和全局状态决定.使用 fork 时,则延迟到 fork 的进程完成,退出进程得到 SIGCHLD 信号,在 SIGCHLD 的处理函数中进行状态决定.

状态决定的主要流程是:① 各进程把本进程本次检查点的成功或失败报告给管理进程(实际上我们通过一个外部管理程序帮助收集所有信息后告诉管理进程.否则,我们无法保证检查点管理进程比别人先完成检查点记录,其他进程的状态报告可能干扰其检查点).② 管理进程收集各进程的报告,如果全部成功,则把全局状态设置为 OK,提交本次检查点(把临时文件更名为正式文件),否则是 FAIL 并禁止下一次的检查点.管理进程决定状态后将状态广播给各进程.③ 如果状态为 OK,准备下一次的检查点(设置定时中断).

3.2.3 恢 复

由于我们记录的是强全局一致状态的检查点,恢复过程相当简单,各进程分别按自己的检查点恢复,再进行适当同步即可.主要流程是:① ckpt_init 检测到:recover 参数,进入恢复过程;② 依次恢复保存在文件中的段(用 mmap 分配空间,用 mprotect 改变段保护状态,用 read 从文件读数据); ③ 用递归长栈法恢复栈;④ longjmp 到保存的 setjmp 点,进入返回非 0 的分支,根据恢复的数据段中的文件表,信号处理表等恢复文件、信号等;⑤ JIAJIA 通信的 socket 由 JIAJIA 原过程重新运行恢复;⑥ 进行恢复同步,任务是等齐所有的进程.与检查点结束状态收集一样,这里也通过外部的管理程序协助完成;⑦ 从 sigalrm_handler 返回,cpu 状态恢复,继续进程.

4 性能评价

由于设计和实现时考虑到了可移植性,目前 JIACKPT 可以运行在 Linux,Solaris 和 AIX 操作系统上,本文测试时仅以 Linux 操作系统为例.测试平台为 8 个节点的 PC 机群,每个节点的 CPU 为 PII-400,内存为 256M,IDE 硬盘,操作系统为 Linux(核心版本 2.2.10).节点间用 100Mbps 交换以太网连接.在测试中,各节点在本地硬盘记录检查点.本文采用了国际上常用的软件 DSM 测试程序和几个实用程序来测试系统性能,包括来自 SPLASH2[14] 中的 Water,Barnes 和 Lu,Rice 大学的 Sor,中科院航空物理所的 IAP18 和中科院高能物理所的 SU2.其中 Water 是水分子模拟程序;Barnes 是天体运动模拟程序;Lu 是矩阵分解;Sor 采用逐次超松驰法解偏微分方程;IAP18 是 18 层的大气模拟程序;SU2 进行胶体球质量计算.除 IAP18 和 SU2 是 FORTRAN 程序以外,其余都是 C 程序.对于 Sor 程序,我们测试了两种规模.

4.1 评价函数

我们使用下面函数来评价检查点系统的性能:$\eta(T_{interval},size)=(T_{checkpoint}-T_{original})/T_{original}\times 100\%$.其中,$T_{interval}$

是检查点平均间隔,size 为应用程序每个进程使用的地址空间平均大小,$T_{checkpoint}$ 为使用检查点功能后程序运行时间,$T_{original}$ 为不使用检查点功能时程序运行时间.η是 $T_{interval}$ 和 size 的函数,实际上反映了系统的开销,该值越小,则系统性能越好.最理想的情况为η=0 时,这表明做检查点不影响原来程序的执行.

同样地,$T_{interval}$ 和 size 在不同程序上可能有不同的η(因为使用优化措施后,有些程序记录的检查点尺寸可能远小于 size.另外,如果使用增量方式或写时拷贝的并行记录方式,检查点对程序的影响也有所不同),但这两个因素通常起决定作用.对于使用消息记录的系统,通信量也是一个关键的因素.评价时应该对所在环境的各种典型程序取平均.

$T_{interval}$和η的关系一般是离散的,对某个程序,在一定范围内的T都导致记录同样多的检查点,从而导致大致相同的开销.我们可以取一些实际使用时比较有意义的值来测试η和 size 的关系.

4.2 测试结果和分析

表 2 中列出了程序的规模、单机运行时间(T_{single})以及未使用检查点功能时的 8 机运行时间($T_{orginal}$).表 3 为系统的测试结果.其中,规模代表了进程需要的平均地址空间;Iter 为程序的循环次数;$T_{interval}$ 为检查点间隔时间;$S_{checkpoint}$ 为平均每个进程记录的每个检查点文件大小;Num 为记录的检查点总数;$T_{checkpoint}$ 为记录检查点时 8 机运行时间,$T_{overhead}$ 为每个检查点的平均时间开销.

Table 2 Program input size and run time of 1/8 nodes

表 2 程序规模及单机和 8 机运行时间

Program	Scale	T_{single} (s)	$T_{orginal}$ (s)
Sor1	2048*2048, 1000 Iteration	352.2	58.8
Sor2	4096*4096, 1000 Iteration	1 411.8	192.8
Lu	4096*4096	492.8	85.2
Water	1728 water molecule,20 steps	457.4	65.5
Barnes	16384	168.5	37.3
IAP18	72*46*18, 1 day	1 046.1	306.0
SU2	8^3 *24, 10 Iteration	214.4	70.3

Table 3 Test results

表 3 测试结果

Program	$T_{interval}$ (s)	$T_{checkpoint}$ (s)	Num	$T_{overhead}$ (s)	$S_{checkpoint}$ (MB)	η
Sor1	10	72.2	6	2.2	7.6	22.8
	30	62.3	2	1.8	7.6	5.9
	60	61.0	1	2.2	7.6	3.7
Sor2	30	218.9	7	3.7	11.8	13.5
	60	204.0	3	3.7	11.8	5.8
LU	30	111.6	2	13.2	52.3	30.9
	60	98.3	1	13.1	52.3	15.4
Water	10	72.0	6	1.1	4.4	9.9
	30	67.4	2	1.0	4.4	2.9
Barnes	10	44.6	3	2.4	8.9	19.5
	30	40.2	1	2.9	8.9	7.7
IAP18	60	355.8	4	10.5	58.8	16.0
	120	331.9	2	12.9	58.8	8.5
SU2	30	76.9	2	3.3	13.3	9.4
	60	73.4	1	3.1	13.3	4.4

从表 3 可以看出,如果每分钟做一次检查点,η的值一般小于 10%,在每次检查点文件(小于进程使用的进程空间)小于 15MB 时,一般小于 5%,在文件更小时,开销也随之减小.在每个检查点文件达到 58.8MB(IAP18)时,η的值也只有 16%.间隔增大时,记录的检查点个数少了,效率还会进一步提高.

为了分析每个检查点的时间开销,我们测试了在机群上写文件的速度(见表 4).测试方法是使用一个程序,先在内存中用 mmap 分配要写的文件大小的空间,初始化后写入文件,再用 fsync 同步,过程重复多次取平均.我们将平均每次检查点开销和写相应大小文件的开销相比,可知:① 同步时间开销很小.当检查点文件只有 4.4MB(water)时,平均每次检查点需 1.1s,即同步和其他保存文件等开销加起来只有约 1.1–0.7=0.4s.其中主要的又是非同步开销(对每个打开文件的 fsync,读取虚存段信息等都比较耗费时间).② 检查点时间与写磁盘时间可比,随规模的变化趋势基本一致.(除写文件外的)额外开销随进程映像大小的增大而缓慢增大.主要原因应该是

进程映像大了之后被换出的页增加了,写检查点要遇到更多的缺页.

Table 4 File write speed of the disk

表 4 磁盘写文件速度

S (MB)	1	2	3	4	5	6	7	8	9
T (s)	0.17	0.30	0.51	0.65	0.81	0.93	1.05	1.20	1.34
S (MB)	10	15	20	25	30	40	60	80	
T (s)	1.50	2.24	2.87	3.72	4.45	5.82	8.78	11.73	

测试表明,即使是 1 分钟 1 次的比较密集的检查点,它带来的系统开销已可以完全容忍.而且对于长时间运行的程序,1 分钟 1 次并不必要,继续增大间隔将使得检查点带来的时间开销占的比例趋于 0.此外,还可以利用减少检查点文件大小的优化技术来进一步提高系统性能.

5 相关工作讨论和未来工作

单进程 Checkpointing 技术已经比较成熟,本文使用的技术基于 Plank 等人的 libckpt[10],并在其基础上解决了 mmap,共享库和通信恢复问题.与本文工作最相似的工作是 Angkul[8]等人在 TreadMarks[2]系统上实现的几种 checkpointing 技术.它所提出的几种技术在每 2 分钟做 1 次 checkpoint 时各应用程序的执行时间开销分别为(CCC)0.5%~4%,SCC(4%~100%)以及 ICC(3%~64%).JIACKPT 在 1 分钟 1 次 checkpoint 时对上述应用程序时间开销大约是 1%~16%,由于两者使用的应用程序和间隔不完全相同,不能直接比较.不过大致可以看出,JIACKPT 和 ICC/CCC 的性能比较接近.JIACKPT 目前的实现和 ICC 最接近,通过增量技术来减少 checkpoint 的数据量.此外,利用 JIAJIA 协议的特性做了一定的优化,进一步减少了一些数据量.CCC 必须在 barrier 点做 checkpoint,而通过实现本文的强一致状态获得方法,JIACKPT 的 checkpoint 不依赖于 barrier,使得它的灵活性大大提高(对 barrier 很少的程序一样适用).

JIACKPT 目前只是实现了对 JIAJIA 系统的错误恢复,在此基础上进一步实现对 JIAJIA 系统单节点失效的透明恢复、动态负载平衡以及容忍运行时节点数变化等,对推进 JIAJIA 的实用化具有重要意义.

References:

[1] Bershad BN, Zekauskas MJ, Sawdon WA. The midway distributed shared memory system. In: Proc. of the 38th IEEE Computer Society Int'l Conf. 1993. 528−537. http://www.cs.cmu.edu/afs/cs/project/midway/WWW/CompCon93.ps

[2] Keleher P, Cox AL, Dwarkadas S, Zwaenepoel W. TreadMarks: Distributed shared memory on standard workstations and operating systems. In: Proc. of the 1994 Winter Usenix Conf. 1994. 115−131. http://www.cs.rice.edu/~willy/papers/wusenix94.ps.gz

[3] Keleher PJ. The relative importance of concurrent writers and weak consistency models. In: Proc. of the 16th Int'l Conf. on Distributed Computing Systems. 1996. 91−98. http://x1.cs.umd.edu/papers/writers.pdf

[4] Hu WW, Shi WS, Tang ZM. JIAJIA: A software DSM system based on a new cache coherence protocol. In: Proc. of the HPCN Europe'99. LNCS 1593, Springer-Verlag, 1999. 463−472. http://citeseer.ist.psu.edu/240766.html

[5] Richard GG, Singhal M. Using logging and asynchronous checkpointing to implement recoverable distributed shared memory. In: Proc. of the 12th Symp. on Reliable Distributed Systems. 1993. 58−67. http://citeseer.ist.psu.edu/richard93using.html

[6] Suri G, Janssens B, Fuchs WK. Reduced overhead logging for rollback recovery in distributed shared memory. In: Proc. of the 25th Int'l Symp. on Fault-Tolerant Computing. Washington DC: IEEE computer Society, 1995. 279−288.

[7] Kermarrec AM, Cabillic G, Gefflaut A, Morin C, Puaut I. A recoverable distributed shared memory system integrating coherence and recoverability. In: Proc. of the 25th Int'l Symp. on Fault-Tolerant Computing. Washington DC: IEEE computer Society, 1995. 289−298.

[8] Angkul K, Santipong T, Tzeng NF. Coherence-Based coordinated checkpointing for software distributed shared memory systems. In: Proc. of the 20th Int'l Conf. on Distributed Computing Systems. Washington DC: IEEE computer Society, 2000. 556−563.

[9] Jerzy B, Michal S. An extended coherence protocol for recoverable DSM systems with causal consistency. In: Proc. of the Int'l Conf. on Computational Science. 2004. 475−482. http://www.springerlink.com/index/YVA5RPUQDQQW8QT0.pdf

[10] Plank JS, Beck M, Kingsley G, Li K. Libckpt: Transparent checkpointing under Unix. In: Proc. of the USENIX 1995 Technical Conference. 1995. 213−223. http://www.cs.utk.edu/~plank/plank/papers/usenix-95w.html

[11] Tang ZM, Shi WS, Hu WW. Message-Passing versus shared-memory on dawning 1000A. Chinese Journal of Computers, 2000, 23(2):134−140 (in Chinese with English abstract).

[12] Hu MC, Shi G, Hu WW, Tang ZM, Zhang FX. Comparing JIAJIA with MPI on PC cluster. Journal of Software, 2003,14(7): 1187−1194 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/14/1187.htm

[13] Chandy KM, Lamport L. Distributed snapshots: Determining global states of distributed systems. ACM Trans. on Computer Systems, 1985,3(1):63−75.

[14] Woo SC, M. Ohara M, Torrie E, Singh JP, Gupta A. The SPLASH-2 programs: Characterization and methodological considerations. In: Proc. of the 22th Annual Symp. on Computer Architecture. New York : ACM Press, 1995. 24−36.

附中文参考文献:

[11] 唐志敏,施巍松,胡伟武.曙光 1000A 上消息传递与共享存储的比较.计算机学报,2000,23(2):134−140.

[12] 胡明昌,史岗,胡伟武,唐志敏,张福新.PC 机群上 JIAJIA 与 MPI 的比较.软件学报,2003,14(7):1187−1194. http://www.jos.org.cn/1000-9825/14/1187.htm

第 10 届全国青年通信学术会议

征 文 通 知

由中国通信学会青年工作委员会主办，北京邮电大学信息安全中心承办，《通信学报》、《北京邮电大学学报》、《电信科学》协办的“第 10 届全国青年通信学术会议”将于 2005 年下半年在四川绵阳(暂定)召开。

一、 征文范围(但不限于这些领域)

1. 宽带交换技术 BX
2. 计算机通信与 WLAN CC
3. 数字城市与数字地球 DC
4. 电路与系统 CI
5. 计算机安全 CS
6. 计算机技术与应用 CT
7. 数字信号处理 DSP
8. 电子商务与电子政务 EC
9. 电磁场与微波技术 EM
10. 电子技术及应用 ET
11. 地理信息系统 GIS
12. 智能卡技术与应用 IC
13. 三网融合 INT
14. 宽带 IP 与 IP 电话 IP
15. 信息安全 IS
16. 信息技术与数据挖掘 IT
17. 智能通信与智能计算 ITS
18. 移动通信 MC
19. 现代密码学 MC
20. 微电子技术与应用 ME
21. 下一代网络技术 NGN
22. 网络安全与网络管理 NS
23. 网格理论与技术 NT
24. 个人通信 PC
25. 光互联网结构及交换技术 OI
26. 光纤通信 OC
27. 光电子技术与电子材料 OE
28. 雷达、通信、电子对抗 RC
29. 通信安全 SC
30. 系统工程与应用 SE

二、 来稿要求

具体要求详见会议网址

三、 重要日期

1．征文截止日期: 2005 年 5 月 6 日(其中邮寄稿件以收稿日期为准)。

2．录用通知日期: 2005 年 5 月 30 日(以电子邮件方式通知)。

四、投稿及联络方式

邮局投稿请寄：100876 北京邮电大学信息安全中心 126 信箱

电子投稿请寄：cuibj@bupt.edu.cn(请注明“第 10 届青通会投稿”主题)

联系人：崔宝江博士 电话：010-62283086 传真：010-62283366

网址：http://www.bupt.edu.cn/isc http://www.buptisc.org