面向大规模计算系统的Cache式并行检查点

刘勇燕[1] 刘勇鹏[2] 冯 华[2] 迟万庆[2]
(科学技术部信息中心 北京 100862)[1] (国防科学技术大学计算机学院 长沙 410073)[2]

摘 要 检查点机制是高性能并行计算系统中重要的容错手段,随着系统规模的增大,并行检查点的可扩展性受文件访问的制约。针对大规模并行计算系统的多级文件系统结构,提出了cache式并行检查点技术。它将全局同步并行检查点转化为局部文件操作,并利用多处理器结构进行乱序流水线式写回调度,将检查点的写回时机合理分布,从而有效地隐藏了检查点的写回开销,保证了并行检查点文件访问的高性能和高可扩展性。

关键词 Cache式检查点,并行计算,多级文件系统,多处理器,乱序流水线
中图法分类号 TP338.4 **文献标识码** A

Cache-style Parallel Checkpointing for Large-scale Computing System

LIU Yong-yan[1] LIU Yong-peng[2] FENG Hua[2] CHI Wan-qing[2]
(Information Center, Ministry of Science and Technology, Beijing 100862, China)[1]
(School of Computer Science, National University of Defense Technology, Changsha 410073, China)[2]

Abstract Checkpointing is a typical technique for fault tolerance, whereas its scalability is limited by the overhead of file access. According to the multi-level file system architecture, the cache-style parallel checkpointing was introduced, which translates global coordinated checkpointing into local file operation by out-of-order pipelining of checkpoint flushing opportunity. The overhead of write-back is hidden effectively to increase the performance and the scalability of parallel checkpointing.

Keywords Cache-style checkpointing, Parallel computing, Multi-level file system, Multi-processor, Out-of-order pipeline

1 引言

对大规模并行计算系统而言,单点故障的发生不可避免,系统固有的可靠性无法满足高性能应用的运行需求[1]。基于检查点的回卷恢复机制是并行计算系统中一种典型的软件容错手段。与单机检查点不同,并行检查点需要保存所有参与并行计算的计算线程的运行现场,以构成并行应用的全局一致映像。根据全局一致性协议的不同,并行检查点分为同步(coordinated)检查点和异步(uncoordinated)检查点。同步检查点在作检查点之前,所有计算线程同步,待进入全局一致状态后再保存各自的运行状态。对大规模系统而言,大量结点同时创建检查点并将检查点保存到文件系统中,形成突发式集中文件访问,文件访问成为瓶颈,不仅增加检查点本身的时间开销,而且对通讯和文件系统造成极大压力,容易引发网络和I/O故障,使系统可靠性雪上加霜。异步检查点在作检查点时,各个计算线程无需进行全局一致性判断,独立确定自己的检查点时机。回卷恢复时,根据各线程检查点之间的依赖关系,搜索全局一致的检查点集合。异步检查点回卷协议复杂,某些检查点可能不属于任何一个全局一致检查点集合,成为孤立的无效检查点。更为严重的是,异步检查点的回卷存在Domino效应,可能无法搜索到可用的全局一致检查点集合,从而失去检查点的作用[1]。

本文分析大规模并行计算系统体系结构和多级文件系统后,提出了基于多处理器系统的Cache式并行检查点模型。它通过在本地文件系统中缓存检查点,将全局同步并行检查点转换成本地文件操作,利用多处理器结构,乱序流水线式分布检查点的写回时机,克服了传统同步并行检查点突发式集中文件操作的缺点,具有良好的可扩展性。

2 相关工作

检查点技术已经有很多成功的实例。Berkeley实验室的BLCR[3]是一个典型的内核级检查点实现方案,支持MPI并行程序的同步检查点。Elnozah[2]等人对并行检查点技术进行了综述。为了降低检查点访问磁盘的时间开销,Plank等人[7]提出无盘(diskless)检查点的思想,将检查点保存在内存中,并利用冗余编码将内存中的检查点映像的编码保存在额外的检查点结点中。当某个计算结点发生故障时,可以利用其余正常计算结点内存中的检查点和检查点结点上的编码在新的结点上恢复故障结点的检查点,实现系统回卷恢复。Chen[8]等人则基于无盘检查点构造容错的并行应用。无盘

到稿日期:2010-06-14 返修日期:2010-11-11 本文受高效能服务器和存储技术国家重点实验室开放基金项目(2009HSSA04)资助。
刘勇燕(1978—),女,博士生,工程师,主要研究方向为电子政务、网络经济和系统容错;刘勇鹏(1977—),男,助理研究员,主要研究方向为高性能计算、系统容错和能耗管理,E-mail:liuyp@nudt.edu.cn;冯 华(1976—),男,助理研究员,主要研究方向为高性能计算和操作系统;迟万庆(1973—),男,副研究员,主要研究方向为高性能计算、操作系统和虚拟化技术。

检查点在内存中保存检查点映像，降低了检查点的创建开销，但是需要大量额外内存，可能大于正常使用内存量的1倍以上，来保存检查点映像，而且只能对少数结点的故障实现编码纠错，不适合大规模科学计算中对内存需要量大的并行应用。Vaidya等人[9]提出了两级检查点思想。他们的容错方案包括两个组件：第一个组件由每个处理器定期地做独立检查点到本地内存，无需与其他处理器同步，用于单结点容错；第二个组件定期地保存全局一致检查点集合到稳定存储介质。Hwang等人[10]进一步提出多级检查点，按照一定的策略将检查点保存在不同的存储介质中，通过保存在本地磁盘中的检查点实现单结点瞬时错误的容错。他们的检查点多级保存思想与本文类似。在并行科学计算中，计算结点间的存在与频繁的通讯、状态密切相关，需保持全局一致。与他们不同的是，我们从提高并行检查点文件访问的可扩展性出发，基于大规模计算系统的多级文件系统和多处理器结构，合理调度系统的空闲计算能力，提高不同层次检查点操作的并行性，二级检查点操作对应用程序在性能上透明，以确保检查点系统的高性能。

3 Cache式并行检查点

典型的大规模并行计算系统由服务结点、计算结点和存储结点3类结点组成，结点间通过通讯子系统彼此互联。

假设参与并行计算的处理器线程数目为 p，每个处理器的检查点映像尺寸为 M，文件系统的访问速率是 d，两个计算线程之间的同步开销是 t，则完成一次全局同步并行检查点到文件系统的时间 T 为：

$$T=\lambda(p,t)+\frac{p\times M}{d} \tag{1}$$

它包括全局同步时间与检查点创建的时间。全局同步时间 λ 与系统规模相关，至少随处理器数目线性增长，而检查点文件访问时间随系统规模线性增长。因此，全局同步检查点的时间开销至少随处理器的数目线性增长。在并行计算系统中，通讯子系统是整个系统的枢纽，是系统并行运行的关键。通讯子系统中发生的故障往往会波及其它结点，导致全系统性的致命故障。其故障发生的概率 δ 随通讯压力（单位时间的通讯量）的增加而增加，也随通讯持续的时间增加而增加，可简化表示为：

$$\delta=\kappa\times T=\kappa\times\left(\frac{p\times M}{d}+\lambda(p,t)\right) \tag{2}$$

由此可见，对大规模系统而言，特别是对检查点数据量巨大的并行应用，文件系统访问的可扩展性是制约检查点系统的重要因素。

为了提高文件访问的速率和文件操作的局部性，并降低通讯压力，许多高性能计算系统都采用多级文件系统结构[5]。计算结点配置访问速率快但容量受限的局部文件系统，并通过全局文件系统共享容量大但速度较慢的远程存储阵列。

本文利用大规模并行计算系统的多级文件存储结构，借鉴Cache高速缓存思想，设计了高效的Cache式多级并行检查点方案。Cache式检查点的基本思想是在做同步检查点时只将检查点映像保存在本地局部文件系统中，既提高检查点性能，又避免突发式大量文件访问造成的文件系统压力，以保证检查点系统的可扩展性。由于局部文件系统容量有限，不适合保存多次检查点映像。更重要的是为了避免单结点故障导致本地文件系统无法访问而造成没有全局一致的检查点映像集合，需要将本地文件系统中的映像写回到全局文件系统中。本文利用多处理器结构特点，进行乱序流水线式写回调度，合理分布检查点的写回时机，尽可能地隐藏检查点写回对系统正常运行的影响。基于检查点回卷恢复时，先尝试从本地文件系统读取检查点映像，如果本地磁盘没有对应的检查点映像，则将其从磁盘阵列读入本地磁盘，然后再基于本地磁盘中的检查点映像恢复原进程。通过快速的本地检查点保存和合理的文件远程flush相结合，实现高性能、高可靠的检查点服务。

4 实现

4.1 层次式结构

本文针对高性能并行计算系统常见的多级存储结构，利用本地文件系统的高速度和全局文件系统的大容量，基于BLCR[3]设计实现了两级cache式并行检查点原型系统CSCR（Cache-Style Checkpoint/Restart）。为了灵活支持不同的检查点策略，维护操作系统内核的简洁高效，避免系统调用超重[4]，CSCR结合了用户库级检查点和内核级检查点的优点，将检查点系统分为两层：用户库层（User-Lib Level）和内核层（Kernel Level）。层次关系如图1所示。

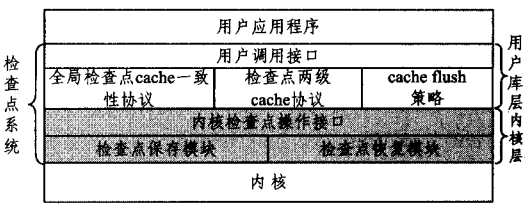

图1 CSCR层次式检查点系统

检查点库（CRLIB）实现检查点Cache协议，维护检查点的全局一致性，控制检查点的写回与读入。为应对不同的应用需求，CRLIB支持多种检查点写回策略，并向用户提供可编程策略定制接口。CRLIB最终调用内核提供的检查点系统服务完成检查点任务。

内核层基于BLCR，在内核中实现进程状态的保存与恢复，包括内核检查点接口模块、保存模块和恢复模块。

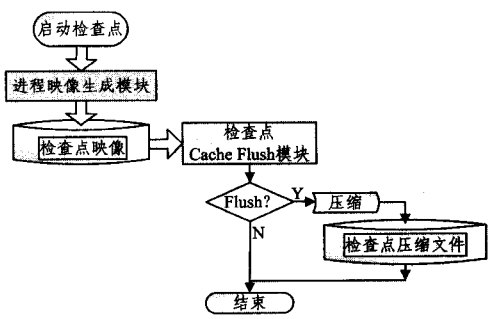

图2 Cache式检查点流程图

内核检查点接口模块在加载时创建服务接口/proc/cscr/ctrl，供CRLIB以ioctl方式调用。保存模块将进程的执行状态快速保存在本地磁盘中，并通知CRLIB中的Cache flush模块。Cache flush模块是一个用户级系统Daemon，负责根据相应的写回策略，选择合适的时机将本地检查点映像写回全局文件系统中。为了降低写回开销，并节省空间，写回时支持用户可选的压缩功能。写回过程采用边压缩边写回的方

式，如图 2 所示，调用 gzip 将本地检查点映像压缩后再写入全局文件系统。恢复子模块是保存子模块的逆过程，基于本地磁盘中的检查点映像恢复用户作业的回卷运行。如果本地磁盘没有相应的检查点映像(Cache 未命中)，则将对应检查点文件从全局文件系统解压到本地磁盘，然后基于本地检查点映像恢复原进程状态。

CSCR 支持多线程。进程内所有线程共享内存空间，只有 CPU 相关信息彼此独立。为此，进程映像生成模块采用“先到先服务”的原则挑选一个线程来保存完整的进程状态，其它线程只保存自己私有的 CPU 信息。一个进程的所有线程共用一个检查点文件。为保证进程内存空间和文件访问的互斥原则，各线程的检查点过程串行，由内核检查点模块负责线程之间的同步与互斥。

4.2 乱序流水线式写回

Cache 式检查点将进程的检查点映像缓存在本地文件系统中，将全局同步并行检查点转换成局部文件操作，文件访问开销不随系统规模的扩大而增加，提高了大规模并行检查点的可扩展性。但是，如何合理隐藏本地检查点映像写回全局文件系统的开销，避免写回操作对应用性能的影响，错开各个结点写回的时机，是 cache 式并行检查点需要重点考虑的问题。为此，针对大规模并行计算系统的多处理器结构，利用冗余的计算能力，设计了基于多处理器系统的乱序流水线式 Cache flush 模块，如图 3 所示。

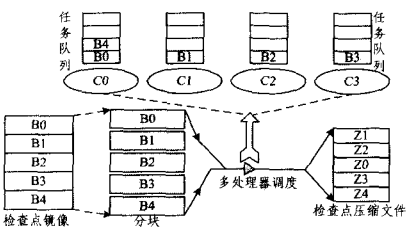

图 3 多处理器流水线 Cache Flush

Cache flush 模块按固定尺寸将本地文件系统中的检查点映像分块，以块为单位申请处理器将其写回。执行写回操作的处理器将该块压缩后，写入全局共享文件系统中对应的检查点压缩文件。多处理器调度模块根据处理器和文件系统的忙闲程度分配处理器，空闲处理器优先被调度。如果全局文件系统负荷较重，写回申请可能被驳回。为避免写回申请被饿死，每块申请被驳回的次数有一定的限制。写回一块相当于整个检查点映像写回流水线中的一段。

为了更好地在多处理器间并行，写回流水线允许乱序执行。为此引入 BlockID，用以标识每块在检查点映像中的位置。处理器压缩完一块后，竞争写检查点压缩文件锁。拿到锁后，首先将压缩块头 z_cscr_header 写入文件，然后写入压缩后的块数据。

```
struct z_cscr_header {
    UINT32 BlockId;
    UINT32 bSize;
    UINT32 zSize;
}
```

其中，bSize 是压缩前块的字节数，zSize 是压缩后块的字节数。由于写回流水线乱序执行，压缩文件中块的顺序可能与原检查点映像中不一致。图 3 中，映像中的第一块 B0，压缩后对应的 Z0 在压缩文件中为第 3 块。回卷解压时，根据 BlockID 恢复检查点块的顺序。

5 实验评测

为验证 Cache 式并行检查点的实际效果，本文搭建了一个由 16 个计算结点和 1 个存储结点构成的并行计算原型系统。每个计算结点配置 2 个 4 核处理器、1 个本地磁盘，构成一级文件系统，写回操作在结点内的 8 个核之间寻求并行。结点间通过 Gigabit Ethernet 互联，计算结点通过 lustre[11] 全局共享文件系统访问远程存储结点，构成二级文件系统。一级文件系统的实测带宽约为 53MB/s，二级全局文件系统在所有计算结点都满负荷文件操作时的结点实测带宽约为 9.8MB/s。

为了精确衡量检查点性能，本文编写了一个 MPI 测试用例，共生成 32 个进程。每个进程绑定在 1 个处理器上运行，每个进程申请指定尺寸的虚存空间，虚存空间写随机值遍历。待所有进程的虚空间写遍历完成之后，再全局同步创建内核检查点，保存进程的完整状态。测试程序分别使用标准的 BLCR 和 CSCR 创建检查点，以 BLCR 为基准标准化后的测试结果如图 4 所示。

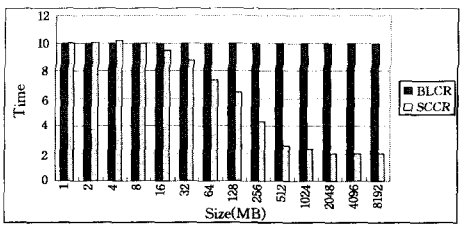

图 4 检查点创建时间对比

从实验结果看，当检查点对文件系统的访问达到一定程度时，文件系统成为并行检查点的瓶颈，CSCR 对时间开销的优化效果非常明显，与本地快速检查点相当。可见，CSCR 能有效提升大规模并行计算系统的同步并行检查点的延时性能。但是，当文件系统不是瓶颈时，CSCR 并不能降低检查点的时间开销。而且，由于 Cache 控制机制，时间开销还可能略有增加。无论如何，Cache 式检查点将全局同步并行检查点的文件操作本地化，可有效降低网络和存储结点的压力，提升整个系统的可靠性。

结束语 本文提出的 Cache 式多级检查点机制以及乱序流水线式写回策略，有效提高了并行检查点系统的延时性能和可扩展性。但是，对多级并行检查点模型的细化，对应用行为特征与 CSCR 优化效果的关系，对写回策略的最优化理论分析和动态自适应，还有待进一步研究。

参 考 文 献

[1] Gibson G, Schroeder B, Digney J. Failure Tolerance in Petascale Computers[J]. CTWatch Quarterly, 2007, 3(4): 4-10

[2] Elnozahy E N, Alvisi L, David B,, et al. A Survey of Rollback-recovery Protocols in Message-passing Systems[J]. ACM Computing Surveys, 2002, 34(3): 375-408

[3] Lawrence Berkeley National Laboratory. Berkeley Lab Checkpoint-Restart(BLCR)[EB/OL]. https://ftg.lbl.gov/CheckpointRestart/, Jun 2010

[4] 刘勇鹏，王小平，李根. 用户指导的多层混合检查点技术及性能优化[J]. 计算机应用研究，2008，25(7)：2097-2099

(下转封三)

[8] Lin Chi-sheng, Chang Jui-chuan, Liu Bin-da. A Low-Power Precomputation-Based Fully Parallel Content-Addressable Memory [J]. IEEE J. Solid-State Circuits, 2003, 38(4): 654-662

[9] Liu S C, Wu F A, Kuo J B. A novel low-voltage content-addressable-memory(CAM) cell with a fast tag-compare capability using partially depleted(PD) SOI CMOS dynamic-threshold(DTMOS) techniques [J]. IEEE J. Solid-State Circuits, 2001, 36: 712-716

[10] Lin P F, Kuo J B. A 1-V 128-kb four-way set-associative CMOS cache memory using wordline-oriented tag-compare (WOTC) structure with the content-addressable-memory(CAM) 10-transistor tag cell[J]. IEEE J. Solid-State Circuits, 2001, 36: 666-675

[11] Austin T, Larson E, Emst D. Simplescala. An infrastructure for computer system modeling[J]. IEEE Computer, 2002, 35(2): 59-67

[12] Burger D, Austin T M. The SimpleScalar Tool Set[R]. CS-TR-97-1342, 1997

[13] 张浩，林伟，周永彬，等. 通用处理器的高带宽访存流水线研究[J]. 计算机学报，2009，32(1)：142-150

[14] Clark L T, Choi B, Wilkerson M. Reducing Translation Lookaside Buffer Active Power[C]// ISLPED. 2003: 10-13

[15] 屈文新，樊晓桠. "龙腾"R2 微处理器存储管理单元的设计与实现[J]. 西北工业大学学报，2007，25(1)：137-141

(上接第 278 页)

区域检测和空间分割。从图 5 中可以看出，由于水纹和船上小旗对时域运动区域检测的影响，分割结果虽然能够完整提取出前景对象，得到对象的主要轮廓，但是边缘定位还不够准确，有待于进一步的提高。Coastguard 序列由于进行了全局运动补偿，处理速度约为 12 帧/s，基本满足实时性。

图 6 是 Akyio 序列的分割结果，Akyio 序列是背景静止的头肩序列，时域检测的运动对象区域准确，空间也得到了较好的分割，所以得到了较好的分割结果。Akyio 序列不用进行全局运动补偿，空间分割区域合并量小，处理速度约为 26 帧/s，满足实时性。

图 7 中，Silent 序列是背景复杂、局部运动快速的图像序列，而 Claire 序列是背景简单的头肩序列，运动较缓慢，从实验结果来看，本文的算法都得到了准确的视频运动对象分割结果。而且本文方法的准确性高于 Kim 提出的方法。

结束语 提出了一种在通用视频序列中联合时空信息分割运动对象的算法。对于运动背景，采用匹配加权的全局运动估计算法，消除了背景全局运动的影响；在时域检测中，使用基于直方图拟合的显著性检测及对称差分法来获得运动对象模板，克服了依据经验设定阈值的缺点，提高了运动对象模板的准确性；在空间区域分割中，对分水岭算法从形态重建滤波、多尺度形态梯度计算、粘性形态学修正及区域合并等方面进行了改进，克服了分水岭算法易受噪声影响及过分割问题，得到了有效的区域分割；最后使用双阈值比重法提取出视频运动对象。从实验结果来看，本文算法对运动背景、静止背景和复杂背景的图像序列都能够进行较好的分割，实时性较好，而且，通过对彩色图像每个像素的 R，G，B 取权值平均后，该算法也可用于彩色图像。但是，由于运动补偿的精度有限，在纹理丰富的背景区域和明显边缘处，会有大的运动误差，对运动背景视频图像序列的分割结果形成了一定影响，因此时域分割准确性还需要进一步改进。如何进一步改进全局运动估计算法的准确性，提高运动对象分割结果还应做进一步研究。

参 考 文 献

[1] 张泽旭，李金宗，李宁宁. 基于光流场分割和 Canny 边缘提取融合算法的运动目标检测[J]. 电子学报，2003，31(9)：1299-1302

[2] 彭小宁，杨明，邹北骥，等. 基于局部图金字塔的不规则块匹配视频分割方法[J]. 计算机科学，2008，35(4)：233-237

[3] Chien S Y, Huang Y W, Hsieh B Y, et al. Fast video segmentation algorithm with shadow cancellation[J]. IEEE Trans. on Circuits Systems for Video Technology, 2004, 6(5): 732-748

[4] 张晓波，刘文耀，吕大伟. 基于时空信息的自动视频对象分割算法[J]. 光电子激光，2008，19(3)：384-387

[5] 杨高波，张兆扬，余圣发. 一种基于小波分解和分水岭变换的视频对象自动分割算法[J]. 通信学报，2005，26(3)：7-13

[6] 李宏，李翔，胡可成. 基于时空融合的视频分割算法研究[J]. 信号处理，2009，25(1)：72-76

[7] Tsaig Y, Averbuch A. Automatic segmentation of moving object in video sequences: a region labeling approach[J]. IEEE Trans on Circuits and Systems for Video Technology, 2002, 12(7): 597-612

[8] Aach T, Kaup A, Mester R. Statistical model-based change detection in moving video[J]. Signal Processing, 1993, 31(2): 165-180

[9] Meryer F, Vachier C. Image segmentation based on viscous flooding simulation[C]// Proceedings of ISMM 2002. CSIRO, Sydney, 2002: 69-77

[10] Vincent L. Morphological gray scale reconstruction in image analysis: applications and efficient algorithms[J]. IEEE Trans. on Image Processing, 1993, 12(2): 176-201

[11] Kim C, Hwang J N. Fast and automatic video object segmentation and tracking for content-based application[J]. IEEE Trans on Circuits and Systems for Video Technology, 2002, 12(2): 122-129

(上接第 289 页)

[5] The 35th Top500 List [EB/OL]. http://www.top500.org, 2010

[6] Franck Cappello INRIA. Fault Tolerance & PetaScale Systems: Current Knowledge, Challenges and Opportunities [C] // EuroPVM/MPI 2008. LNCS 5205. Berlin Heidelberg: Springer-Verlag, 2008

[7] Plank J S, Li Kai, et al. Diskless Checkpointing. IEEE Transaction on Parallel and Distributed Systems[J]. 1998, 9(10): 972-986

[8] Chen Zi-zhong, Fagg G E, Gabril E, et al. Building Fault Survivable MPI Programs with FT-MPI Using Diskless Checkpointing [C]// Proceedings of the 10th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming(PPoPP). Chicago, Illinois, 2005: 213-223

[9] Vaidya N H. A Case for Two-Level Distributed Recovery Schemes[C]// ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems. Ottawa, Canada, 1995: 65-73

[10] Hwang K, Hai Jin, Chow E, et al. Designing SSI Clusters with Hierarchical Checkpointing and Single I/O Space[J]. IEEE Concurreycy. 1999, 7(1): 60-69

[11] Oracle. Lustre [EB/OL]. http://www.sun.com/software/products/lustre/index.xml, Jun. 2010