# 实时交互进程的并发检查点技术

廖剑伟，李 莉，陈善雄，余建桥

(西南大学计算机与信息科学学院 重庆 北碚区 400715)

**【摘要】**提出了一种并发检查点技术，允许进程在设置检查点的同时尽可能保持继续执行。在拷贝进程地址空间(设置检查点的步骤之一)的同时，阻塞页面写操作并拷贝该原始页面到指定缓冲区，达到不需要在拷贝进程空间的同时停止该被设置检查点进程，最后结合缓冲区中拷贝页面，得到具有一致性的进程状态的映像文件。实验结果表明，可以减少20%～70%被检查进程的停止时间，使得检查点的设置与进程的执行具有一定的并发性。减少被检查进程在设置检查点时的停止时间，适合实时性和交互性要求较高的进程设置检查点。

**关 键 词** 设置检查点； 并发性； 实时交互进程

**中图分类号** TP393 **文献标识码** A doi:10.3969/j.issn.1001-0548.2011.04.020

# Concurrent Checkpoint/Restart Mechanism for Real-Time Interactive Processes

LIAO Jian-wei, LI Li, CHEN Shan-xiong, and YU Jian-qiao

(College of Computer and Information Science, Southwest University Beibei Chongqing 400715)

**Abstract** This paper presents a new concurrent checkpoint/restart mechanism, it allows the checkpointed process (checkpointee) without stopping while doing checkpointing to some extent. The checkpointee can continue to run until a write request is captured by tracing TLB during dumping memory pages (which is a critical step of doing checkpoint). At that time, the checkpoint module will copy the requested page to the designated memory buffer, then enable the write operation. Then, it can form an image file with consistent process state by using original pages which copied in the designate memory buffer. The experimental results show that this mechanism can reduce 20% ~70% stopping time of checkpointee to ensure the concurrency between doing checkpoint and execution of checkpointee. This mechanism is a quite good choice for hign real-time and interactive priority applications.

**Key words** checkpoint/restart; concurrency; real-time interaction process

源代码中简单的编程错误可能导致整个操作系统的崩溃，文献[1]的研究表明，1 000行可执行代码中包括6～16个错误。另外，不可信任的第3方驱动程序导致系统崩溃的可能性是普通代码的3～7倍[2]。因此操作系统需要提供一种从故障中自动恢复的容错机制[3-4]。系统检查点技术恢复应用程序运行是一种有效的容错方法，该技术将正在运行的进程状态以进程映像的形式保存到外部磁盘，当该进程发生故障时，系统可以根据保存在磁盘的进程映像文件从检查点处恢复执行。同样，通过使用检查点技术可实现进程的迁移，达到负载均衡的目的[5-6]。

传统的检查点技术需要在设置检查点时停止设置检查点目标进程，即该目标进程在检查点设置时必须停止执行。对于实时性、交互性要求比较高的进程，停止时间过长是不可接受的。本文提出了一种实时交互性进程的并发检查点技术，从程序上允许进程在被设置检查点时保持继续运行，从而满足实时性、交互性较高的应用需求。

## 1 相关工作

通常，检查点技术分为用户级检查点技术和系统级检查点技术。因为系统级检查点具有较好的透明性，本文研究仅关注系统级的检查点技术。BLCR(Berkeley lab checkpoint/restart)[7]是美国伯克利国家实验室开发，以模块方式加载的基于Linux的检查点系统，是目前比较具有代表性的Linux检查点系统。该系统除了支持常规应用程序，也支持对MPI并行应用程序设置检查点。唯一的不透明性是被设

收稿日期：2009－12－01； 修回日期：2010－05－07
基金项目：国家863计划(2006AA10Z1E6)； 中央高校基本科研业务费专项资金(XDJK2009C025)； 重庆市自然科学基金(CSTC2010BB2006)
作者简介：廖剑伟(1981－)，男，博士生，主要从事可靠性操作系统方面的研究.

置检查点的应用程序必须包含部分BLCR库文件代码。Cryopid[8]是一款较有特色的检查点系统，它不需要在内核态运行，普通用户可以使用该系统保存并恢复用户进程，进程状态被保存于一个压缩的自解压可执行程序。CRAK[9]是美国哥伦比亚大学2001年开发的一款基于Linux的检查点系统。同BLCR一样，该系统以模块的方式进行加载，但由于是多年前设计和开发的，不支持Linux内核2.6，而且在设置检查点时所需要的时间比较长。Linux CKPT[3]是Oren Laadan实现的Linux检查点模块，支持目前最新的Linux内核2.6.31，最新版本号为V18。

为了确保设置检查点目标进程状态的一致性，前述的所有检查点系统都是在设置检查点之前暂停进程运行直到完成整个进程状态的保存。由于设置检查点可能会导致较长的进程停止时间，使得这些检查点系统并不适用于实时性和交互性要求较高的场合。

## 2 并发检查点系统的设计与实现

### 2.1 系统设计

考虑到实时性和交互性进程对停止时间的限制，本文提出一种实时交互性进程的检查点技术。如前所述，为了确保进程状态的一致性，在设置检查点之前必须停止被设置检查点进程，然后再进行检查点的设置。一般来说，检查点设置过程包括将线程信息、CPU寄存器信息、进程的内存地址空间，以及所打开的文件、管道和socket等信息保存到磁盘。直到所有进程状态信息被保存到磁盘文件(或者是将进程状态信息保存到某内存缓冲区)后再恢复进程的继续执行。因此，进程在检查点设置的整个过程中无法继续执行。

在Linux CKPT v14的Linux内核2.6.28的qemu[10]模拟器下进行实验，结果如表1所示。

**表1 设置检查点时保存内存页面所需时间**

| Benchmark | 设置检查点/ms | 拷贝内存空间/ms | 比例/(%) |
|---|---|---|---|
| grep | 266 | 177 | 67 |
| bzip2 | 342 | 190 | 56 |
| 矩阵相乘(128×128) | 433 | 271 | 79 |
| 矩阵相乘(256×256) | 1 139 | 1 008 | 88 |
| 矩阵相乘(512×512) | 8 233 | 8 024 | 97 |

从表1可以看出，保存内存地址空间所需要的时间占据了检查点设置的60%以上，特别对于大内存应用程序，如不同大小的矩阵相乘，所需要的时间甚至会超过80%。因此，本文提出了并发检查点技术，即在拷贝进程的内存址空间的同时，允许进程继续执行，从而提高进程的并发性。

图1为该并发检查点算法的基本流程图。

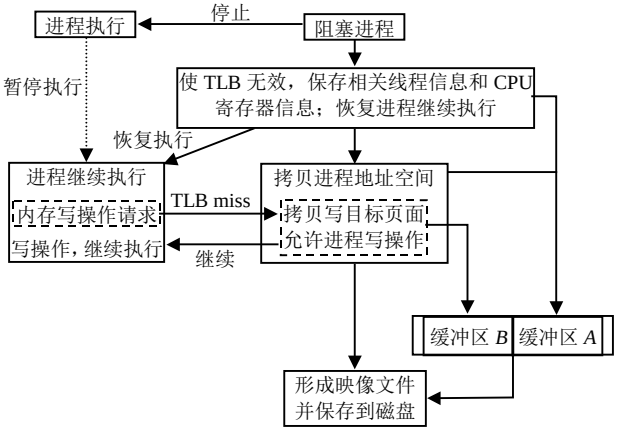


图1 并发检查点技术的基本流程算法

并发检查点技术的操作流程如下：

1) 停止被设置检查点的进程，保存该进程的线程信息、CPU寄存器信息和相关共享库信息到内存缓冲区*A*；

2) 使进程TLB记录无效，恢复被设置检查点的进程执行；

3) 拷贝进程的内存地址空间页面到内存缓冲区*A*；

4) 在设置检查点时内存页面写操作会导致进程状态的不一致性，通过追踪TLB缺页截获所有内存的读写操作，如果是内存写操作，延迟该写操作请求，将目标页面拷贝到内存一固定缓冲区*B*后，允许针对该页面的写操作；

5) 综合缓冲区*A*的内存页面信息和缓冲区*B*的内存页面信息，将这些信息以一定的格式写入进程映像文件并保存到外部磁盘。

综上所述，在设置检查点时通过阻塞并延迟内存写操作，将原始页面拷贝到固定内存缓冲区后并允许写操作，在综合缓冲区信息形成进程映像文件时不需要写入最新修改的页面，使用内存缓冲区*B*中的拷贝页面，将其写入映像文件并保存到磁盘。

### 2.2 系统实现

目前的并发检查点系统的实现版本仅支持SH4体系结构，作为Linux的一个可选择模块编译进入内核。其包括17个C源代码程序，约10 500行源代码。因为线程信息、CPU寄存器信息的保存与传统的检查点技术并没有实质的区别，在此只讨论如何通过TLB页面失效捕捉到页面写操作。

在设置检查点之前，TLB缓存无效，使在设置检查点时所有的页面读写操作都会发生TLB失效。通过查找页表得到页面地址，将该写操作目标页面(原始页面，设置检查点开始页面内容没尚未有任何改变)拷贝到缓冲区，然后将包含该页面的TLB条目载入TLB，继续写操作。

进程地址空间中的代码段和共享库并不会随着进程的运行而发生变化，并且这些信息都可以通过读取进程的可执行文件获取。因此，该并发检查点系统并不保存进程的代码段和共享库达到加快设置检查点和减小映像文件大小的目的。

进程的恢复就是重新创建一个新进程，将保存在磁盘的进程映像文件中的相关进程状态信息写入到该新进程，该新进程即可从检查点处开始执行。

## 3　实验与比较

为检验系统正确性，本文进行了一系列的模拟实验。通过将实验结果与传统的检查点系统比较，验证了所提方法的正确性和更好的并发性。所有实验均在qemu 10.5[10]模拟器下完成，实验使用矩阵相乘作为测试基准程序，矩阵元素为Double类型。

### 3.1　设置检查点开销

3种检查点系统设置检查点后得到的进程映像文件的大小如图2所示。从图中可以看出，并发检查点系统形成的映像文件最小，因为它没有保存进程内存地址空间中的代码段和共享库文件，所以设置检查点后的映像文件小于利用BLCR和Linux CKPT V14设置检查点后得到的进程映像文件。

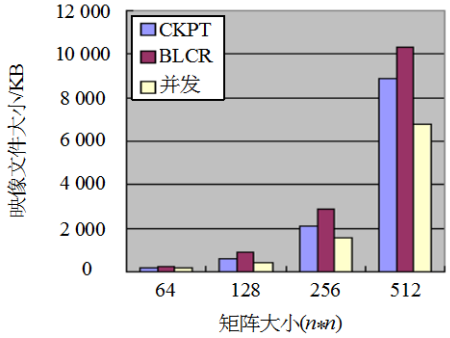


图2　磁盘进程映像文件大小

BLCR目前不支持SH4体系结构，为了使数据具有可比性，实验数据是根据式(1)进行同等变化：

$$TC_{sh4} = TC_{x86} * (TE_{sh4} / TE_{x86}) \quad (1)$$

式中，TC(time with checkpoint)表示对进程(测试基准程序)设置检查点后的总共执行时间；TE (time just execution)表示不设置检查点的进程执行时间。

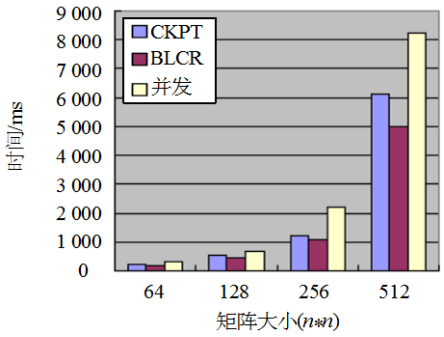


图3　设置检查点时间

设置检查点所需时间如图3所示。并发设置检查点系统需要综合两个缓冲区的数据形成进程映像文件并保存到磁盘，因此该设置检查点的时间较长。设置检查点所需的时间与并发性没有直接的关系，因为并发检查点系统设置检查点时，进程很有可能是在运行过程中。

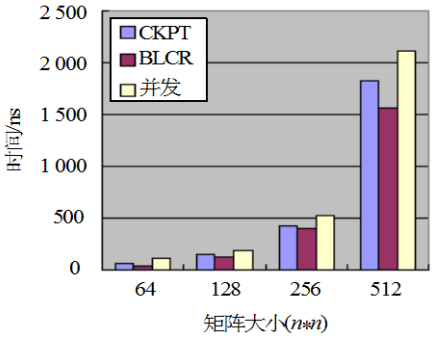


图4　进程恢复时间

图4为完成进程恢复所需的时间。因为并发检查点系统在设置检查点时，并没有在映像文件中保存代码段，在恢复执行时需要从可执行文件中获取代码段信息，因此恢复时间相对于图4中所示的其他检查点系统稍长些。

本文研究的重点是减少设置检查点时的进程停止时间，提高系统的并发性，因此较长的恢复时间是可以接受的。另外，并非每次设置检查点以后都会执行进程恢复，因为进程恢复仅仅发生在当前执行进程发生故障时。

### 3.2　并发性

定义减少的进程停止时间百分率(percentage of reduced downtime，PRDT)作为反映并发设置检查点技术带来的并发性，具体为：

$$PRDT = (DT_{non} - D_{con}) / DT_{non} \quad (2)$$

式中，$DT_{non}$表示传统检查点技术，即非并发检查点技术设置一个检查点带来的进程停止时间；$DT_{con}$表示并发检查点技术带来的进程停止时间。

因为PRDT的定义以非并发检查点技术的进程停止时间作为参照，因此非并发检查点技术的PRDT为0。

图5显示了当基准测试程序是矩阵相乘时，并发检查点技术带来的所减少的进程停止时间的百分率。从图中可见，相对于使用传统的非并发检查点技术，该并发检查点技术可以减少21.4%～71.8%的进程停止时间。

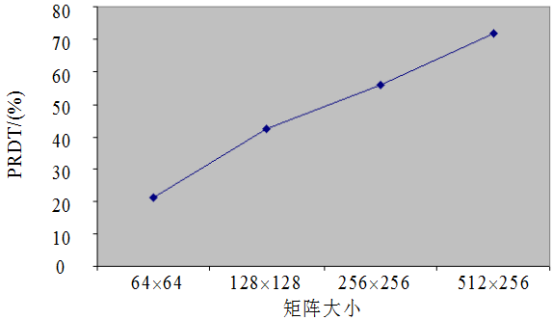


图 5 矩阵相乘进程 PRDT

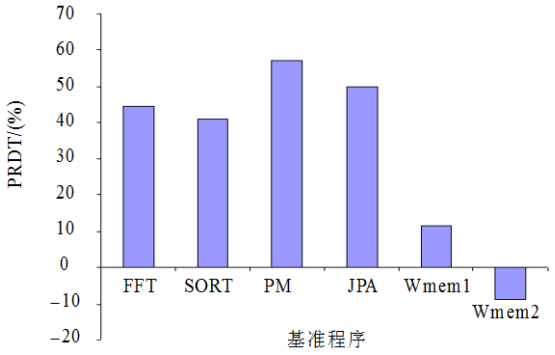


图 6 计算集中型进程与随机写操作测试进程 PRDT

此外，还选择部分计算密集型的基准测试程序评估该并发设置检查点技术的性能，如快速傅里叶变换(FFT)、元素类型是int类型的冒泡排序(SORT)、从输入文本文件中搜索指定字符串的模式匹配(PM)和约瑟夫算法(JPA)，结果如图6所示。并发设置检查点技术可以减少41.18%以上的进程停止时间。

图6还显示了两个随机内存写操作基准测试程序所减少的进程停止时间的百分率。Wmem1是间隔2 KB写一次内存，Wmem2是间隔4 KB(在Linux中设置的默认页面大小为4 KB)写一次内存。从图6中可以看出，使用并发设置检查点技术为Wmem2进程设置检查带来的进程停止时间甚至大于使用传统检查点技术带来的进程停止时间。

### 3.3 算法时间复杂性分析

该算法的复杂度主要由设置检查点的内核进程与被设置检查点的用户进程之间的消息数量决定。从图1可以看出，在设置检查点的过程中，写操作引起的TLB失效的页面数为$N$，控制消息为$2N+2$，即复杂度为$O(N)$。

## 4 总 结

本文提出了并发检查点技术，以减少被设置检查点的进程停止时间，尽可能提高进程执行与设置检查点的并发性。该并发检查点技术允许在拷贝进程内存地址空间的同时，通过阻塞页面写操作并在拷贝该原始页面后允许写操作，达到在拷贝进程内存地址空间的同时不必一直停止进程。实验结果表明，对于写操作不是非常频繁的进程，可以减少20%～70%被检查进程的停止时间。该并发检查点技术适用于实时性和交互性要求较高的进程，构建容错系统。

目前并发检查点实现版本仅支持SH4体系结构，不支持socket网络连接信息的保存，需要完善该检查点系统。另外，该并发检查点系统对于写操作频繁的进程进行检查点设置时所需要的进程停止并不优于传统的检查点技术，如何优化该并发检查点技术达到减少写操作频繁的进程设置检查点所需的停止时间，是以后进一步的工作。

## 参 考 文 献

[1] BASILI V R, PERRICONE B T. Software errors and Complexity: an empirical investigation[J]. Commun of the ACM, 1984, (27): 42-52.

[2] KALOGERAKI V, Zeinalipour-Yazti, GUNOPULOS D, et al. Distributed middleware architectures for scalable media services[J]. Journal of Network and Computer Applications, 2007, 30(1): 209-243.

[3] DAVID F M, CARLYLE J C, CHAN E M, et al. Improving dependability by revisiting operating system design[C]//Workshop on Hot Topics in Dependability. Edinburgh, UK: [s.n.], 2007.

[4] DAVID F M, CAMPBELL R H. Building a self-healing operating System[C]//Proceedings of the Third IEEE International Symposium on Dependable, Autonomic and Secure Computing. Columbia, Maryland: IEEE, 2007: 3-10.

[5] LAADAN O, NIEH J. Transparent checkpoint-restart of multiple processes on commodity operating systems[C]// USENIX Annual Technical Conference. [S.l.]: [s.n.], 2007: 323-336.

[6] SANKARAN S, J SQUYRES M, BARRETT B, et al. The LAM/MPI checkpoint/restart framework: System-initiated checkpointing[C]//Proceedings, LACSI Symposium. Sante Fe, New Mexico, USA:[s.n.], 2003: 1-12.

[7] HARGROVE P H, DUELL J C. Berkeley lab checkpoint/restart (blcr) for linux clusters[C]//Proceedings of SciDAC 2006. [S.l.]: [s.n], 2006.

[8] CRYOPID. A Process Freezer for Linux[CP/OL]. [2009-12-01]. http://cryopid. berlios.de/.

[9] ZHONG H, NIEH J. CRAK: Linux checkpoint/restart as a kernel module, technical report CUCS-014-01[R]. New York, USA: Department of Computer Science, Columbia University, 2001.

[10] BELLARD F. Qemu: a fast and portable dynamic translator[C]//Proceedings of the USENIX 2005 Annual Technical Conference. [S.l.]: [s.n.], 2005: 41-46

编 辑 税 红