# 基于虚拟文件操作的文件检查点设置*

刘少锋, 汪东升, 朱 晶

(清华大学 计算机科学与技术系,北京 100084)
E-mail: {qingfengl,glasswort}@263.net; wds@tsinghua.edu.cn
http://www.tsinghua.edu.cn

摘要: 实现分布/并行系统容错的基础是单进程检查点设置和卷回恢复技术,而对活动文件信息进行保存和恢复则是这种技术的重要方面.提出一种虚拟文件操作策略,实现了对用户文件的检查点设置,有效地解决了发生故障时用户文件内容与进程全局状态的不一致的问题.该方法通过文件块式管理、检查点分布操作等技术,使得在空间开销、正常运行时间、恢复时间等性能指标上优于其他方法,并且具有对用户透明、可最大限度地保留已完成工作的特点.

关 键 词: 文件检查点;卷回恢复;虚拟文件操作;文件集;故障
中图法分类号: TP316 文献标识码: A

近年来,性能/价格比较高的工作站机群系统(network of workstations,简称 NOW)被广泛地用作分布/并行系统平台.但是,NOW 系统随着规模的扩大,故障率也呈指数增长,要使 NOW 系统在各领域中得以广泛应用,必须为其提供容错功能.检查点设置和卷回恢复技术是实现分布/并行系统容错的一种重要手段是指在一个程序运行时,周期性地将程序运行状态保存起来;当系统发生故障时,根据保存的程序状态恢复系统,并从该状态继续执行.

为了使进程在卷回恢复后能够正确地继续执行,必须保证所保存的进程状态信息的完备性以及这些信息之间的一致性.一般说来,要保存的进程状态应包括:进程数据段、用户栈内容;程序计数器、处理机状态字寄存器、栈指针;信号屏蔽码、信号栈指针、信号处理函数、被挂起的信号;当前活动的用户文件信息,包括文件描述符、访问方式、文件大小、读写指针、文件内容等.活动文件的内容是进程与外界联系的重要手段.但是在目前已经报道的各种系统中,由美国威斯康星大学开发的 Condor[1]不支持文件的保存与恢复;libckp[2]和 libfcp[3]虽然支持了对用户文件内容的保存与恢复,但是它们的开销都比较大;libcsm[4]系统通过采用延迟写策略,降低了开销,但是却没有考虑磁盘文件写入过程中的故障恢复问题.本文主要介绍我们开发的 LIBVFO 系统,它采用了一种低开销的虚拟文件操作(virtual file operation,简称 VFO)策略,支持对用户活动文件内容的检查点设置和卷回恢复,同时解决了磁盘写入过程中的故障恢复问题.

## 1 文件故障分析

用户文件内容与进程的其他状态有很大的不同:其他状态往往随着应用程序发生故障而丢失,在恢复时把保存在可靠存储上的状态写入进程空间就可以恢复这些状态;而用户文件本身就保存在硬盘上,不随应用程序发生故障而丢失.也就是说,如果不对文件内容进行特殊处理,当应用程序发生故障时,其他进程状态卷回恢复到上一个检查点所保存的状态,而用户文件内容仍然是发生故障时的状态,从而有可能造成卷回恢复后用户文

* 收稿日期: 2001-03-30; 修改日期: 2001-08-20
基金项目: 国家 863 高科技发展计划资助项目(863-306-ZD0102);国家教育部高等学校骨干教师资助项目
作者简介: 刘少锋(1977－),男,山东海阳人,硕士,主要研究领域为分布式系统;汪东升(1963－),男,黑龙江哈尔滨人,博士,副教授,主要研究领域为分布式系统;朱晶(1980－),女,江苏如皋人,博士,主要研究领域为 Web 集群,分布式系统.

件内容进程全局状态不一致的现象的发生,进而导致程序运行错误.

由于直接在检查点中保存用户文件全部内容的方法显然不可行,所以大多数系统只是在设置检查点时保存进程当前用户文件的活动信息;在卷回恢复时重新打开文件,并根据活动信息将相应文件截成原来的长度,并将文件指针指向原来的位置.但是,通过分析应用程序中由各种系统调用和读写方式构成的不同文件操作流程对文件内容的影响,可以发现,上述方法有时会导致卷回恢复后文件内容与进程全局状态不一致的现象的发生.

在图 1 中,在设置完检查点 $i$ 之后,文件 example 被以添加、只读方式打开,并写入一段内容.由于故障是在检查点 $i$+1 之前发生,而且文件的长度没有保存在检查点 $i$ 中,所以在卷回恢复时,文件 example 无法被截成正确的长度.因此,缓冲区 write_buf 中的内容实际上被添加了两次,引起执行错误.同样,如果先以文件尾作为起始地址调用系统调用 LSEEK,之后再进行写操作,也会产生类似的错误.这两种错误的原因都是写操作的内容和方式依赖于某些实时事件(在这里是读取文件长度),并且在该写操作之后尚未设置过检查点时发生故障.因此,此类故障被称为 FARE(failure after real-time event)故障.

在图 2 中,先以读写方式打开文件 example,在设置完检查点 $I$ 之后,对同一文件区域先后进行读和写操作.写操作完毕后,系统发生故障并卷回到检查点 $i$ 处重新执行,此时欲读的文件区域的内容已被修改,且未随卷回而得到恢复,从而导致文件内容与进程全局状态不一致.这次的重新执行就会因从该区域读出的内容是修改后的新内容而引起执行错误.这种故障被称为 FARW(failure after reading and writing the same area)故障.

```
checkpointing i
fd=open(“example”,O_WRONLY|O_APPEND);
write (fd, write_buf, BLOCKSIZE);
/*failure occurs, here*/
checkpointing i+1/*should be here*/
```

Fig.1 FARE fault

图 1 FARE 故障

```
fd=open(“example”, O_RDWR);
checkpoint i
read (fd, read_buf, BLOCKSIZE);
lseek (fd, 0, SEEK_SET);
write (fd, write_buf, BLOCKSIZE);
/*failure occurs, rollback*/
checkpoint i+1/*should be here*/
```

Fig.2 FARW fault

图 2 FARW 故障

作为更明显的情形,如果在设置完检查点 $i$ 之后,进程读文件后调用系统调用 UNLINK 将它删除,当系统发生故障并卷回到检查点 $i$ 处重新执行时,将会因读一个不存在的文件而产生更为直接的程序错误.这种故障被称为 FAD(failure after deleting)故障.

FARE,FARW 和 FAD 故障是仅保存活动信息所导致的 3 种典型故障.虽然通过限制用户所能进行的文件操作可以在一定程度上避免上述故障,但这显然会影响检查点设置工具的可用性和透明性.因此有必要对活动文件内容进行保存和恢复,以支持用户程序任意的文件操作,包括 Open;Close;Seek;Read;Write;Unlink;Dup;Dup2 等.

为此我们提出了虚拟文件操作(VFO)的方法.该方法能够解决活动文件内容的保存和恢复问题,并且考虑了其他一些系统遗漏的问题.VFO 保证了任何系统故障都可以正常恢复,而且在开销上要低于目前其他的方法.

## 2 虚拟文件操作(VFO)

### 2.1 基本思想

保存文件状态最简单的方法是保存整个文件内容,libfcp[3]采用的就是这种方法.而在实际情况中,用户打开的文件可能有几十兆甚至几百兆,那么文件内容的保存将带来很大的时间和空间开销,这种开销是不能被接受的.我们要研究一种可行的方案,既能够有效地记录文件的状态,又可以避免如此大的开销.

VFO 策略的基本思想是把相邻两个检查点之间的所有文件读、写操作作为一个整体来处理,这些操作或者全都正确执行完毕,或者由于发生了故障而全部放弃.具体而言,检查点 $i$ 到 $i$+1 之间的读、写操作并不真正与跟

磁盘打交道,而是将读、写操作的各项参数以及数据记录到一个虚拟文件操作管理表中(具体结构请参见下文),进程对文件的读、写实际上是对虚拟文件操作管理表表项的读、写,相当于在文件系统和用户进程之间增加了一层虚拟文件系统,所有的文件操作都利用函数包裹技术重定向到这个虚拟文件系统.如果两个检查点之间的文件操作全部正确完成,在检查点 $i+1$ 时,由虚拟文件管理器负责把与表项相对应的写操作执行到可靠存储上.VFO 策略使得用户文件内容在设置下一个检查点之前不会被实际改变,相当于通过对文件的修改内容做内存映像,从而达到对文件全部内容进行保存的效果.在两个检查点之间,硬盘上的检查点文件和用户文件的内容都是一致的.VFO 策略使得上述 3 种故障都不会再引起卷回后程序错误了.

在 VFO 策略中,最重要的两个操作是读、写操作.这两个操作的执行过程可以简单地描述如下:

对于写操作,首先看要写的部分是否在虚拟文件操作管理表的数据区中,如果在,则直接写入数据区;如果不在,则把要写的部分从文件读入虚拟文件管理器的数据区,然后修改.

对于读操作,首先察看在虚拟文件操作管理表中有无相关内容(通过察看虚拟文件操作管理表的几个指针域即可判断),如果有,则直接从虚拟文件操作管理表中的数据区获得,如果没有,则从硬盘中读取.

### 2.2 主要数据结构

我们把进程打开的所有文件用一个链表连接起来,链表的头指针存放在我们定义的 VirtualFileManager 数据结构中,每个文件指针指向一个自定义的 FileEntry 结构,该结构存放了有关文件的标准信息和一些自定义信息,FileEntry 中的一个重要域是 pFirstBlock,该指针指向一个文件块结构 FileBlock,该块是系统管理文件的数据单位,块的大小依不同的分块策略而有所不同.Block 记录了该块在文件中的始末位置、块的内容、块大小以及指向下一块的指针.具体数据结构定义如下:

FileBlock:文件块.当读文件数据缺失时,从文件载入内存的基本单位,通过固定大小的文件块对文件进行页式管理,当程序有频繁的小规模读写时,能够显著减少 I/O 访问次数,大大提高读写数据的速度,降低开销.包含的域如下:

```
typedef FileBlock{
    char *      szBlock;        //块的内容;
    Int         iStartPos;      //块在文件中的起始位置;
    Int         iEndPos;        //块在文件中的结束位置;
    Int         iBlockSize;     //块大小;
    Block*      pNextBlock;     //下一块指针;
}
```

FileEntry:文件项.一个加强定义的文件指针.包含的域如下:

```
typedef FileEntry{
    char*           szFileName;         //该文件的全路径名;
    FILE *          pFile;              //普通的文件指针,包含标准文件信息;
    Int             ifDeleted;          //文件是否已经删除;
    Unsigned Long   iBlockFlag;         //一个 64 位长整数,每一位 0/1 代表文件的一块是否被载入;
    Block*          pFirstBlock;        //文件块链表的头指针;
    FileEntry*      pNextEntry;         //指向下一个文件项目;
}
```

VirtualFileManager:虚拟文件管理器,以文件为对象,实际上是所有读写文件组成的链表的头指针,惟一的域是进程的第 1 个文件项目:

```
typedef VirtualFileManager{
    FileEntry*      pFirstEntry;    //文件链表的头指针;
}
```

### 2.3 算 法

要正确完成文件检查点的设置和恢复,我们必须保证文件的正常操作和恢复操作都是正确的,而且对用户

是透明的.

2.3.1 VFO 算法

正常操作的正确性主要通过 VirtualFileManager 实现.包裹后的函数把读、写重定向到虚拟文件操作管理器进行.虚拟文件操作(VFO)按如下思路处理文件操作:

Open:打开文件,向 VirtualFileManager 插入一个文件项,把数据结构的一些标志初始化;

Close:关闭文件;

Read:首先查找 VirtualFileManager,看读取内容是否在内存中,如果在,直接从中读取所需的内容,如果不在,则调用标准 read 函数从磁盘中读取;

Write:首先查找要写的位置所在文件块是否已经在内存中,如果不在,把该块载入,然后修改内容,如果已在,则直接进行修改(如图 3 所示).

Unlink:在 VirtualFileManager 中把文件项的各块释放,同时把删除标记置为 1.

```
Write File* f1,Char* string
{
//计算写入位置所在块;
    while( )//把大量数据按数据块循环写入;
    {
        if 当前写入位置所在块不在 VirtualFileManager 中
        //把该块内容载入 VirtualFileManger 中;
        if (strlen (string)<BlockSize){
            //把 string 内容写入 VFO 的相应块;
            break;//写完;
        }
        else{
            //把 string 中的内容写入 VFO 当前块,写满为止;
            string=string+BlockSize;//写入位置后移一块;
        }
    }//end of while;
}//End of Write Function;
```

Fig.3 Virtual file write operation

图 3 文件虚拟写操作

以上操作相当于在应用程序和标准接口之间增加了一个虚拟文件操作管理层.它完备地保存了文件被改动的信息,而不会影响正常的文件操作.我们将利用这个信息完成文件检查点的操作.

2.3.2 检查点设置算法

文件的恢复操作是文件检查点的核心部分.其关键技术是分布检查点技术.分布检查点技术的作用是确保无论系统在何时发生故障,都可以根据我们保存的有限信息加以恢复.其基本思路是,在使用虚拟文件管理器的信息对文件进行改写之前,首先在机群的其他结点上制作虚拟文件管理器数据结构的副本,确保无论何时发生故障都保存着一个文件改动信息的可靠拷贝,从而保证了文件的可恢复性.分布检查点设置算法如图 4 所示.

```
检查点 i 设置完毕;
    Fopen(…);
    [Fread (….)];
    [Fwrite(…)];          VFO 管理↔可靠存储(STABLE STORAGE);
    [Fseek (…)];
    Fclose(…);
Label:检查点 i+1;
{
    (1) 把内存中的 VirtualFileManager 导出到机群中的相邻结点,更为一般的情况,可以采用
        Harsh 表,导出到相应的结点;
    (2) 进程间同步,各个进程分别作检查点,不修改文件;
    (3) 根据 VirtualFileManager 将文件改动内容写入文件;
}
Continue;
…
```

Fig.4 Step checkpointing

图 4 分布检查点设置

在分布检查点技术流程中,如果程序在正确到达 Label 前出错,系统根据检查点 $i$ 进行恢复;如果系统正确运行到检查点 $i$+1,并进入分布检查点设置过程,在此过程中,按阶段分,故障只能发生在阶段(1)~(3)中:

阶段(1). 在导出过程中出错,文件改动信息还没有作用到文件上,此时文件集与检查点 $i$ 保持一致,可以共同恢复到检查点 $i$ 时的系统状态;

阶段(2). 在进程作检查点的时候出错,系统也可以恢复到检查点 $i$ 时的状态,理由同上;

阶段(3). 这是关键部分.在第(3)阶段,如果检查点已经设置完毕,而文件修改没有完毕,此时文件集与任何一个检查点都不一致.如果按照检查点 $i$ 恢复后,当时的文件已经被改过了;如果按照检查点 $j$ 恢复,文件还没有改完,文件内容不符合检查点 $j$ 的要求.所以直接恢复是不行的.但是由于我们已经把 VirtualFileManager 导出到相邻结点上(或者其 Harsh 结点上),所以可以根据 VirtualFileManager 重新修改文件.重新修改的文件内容会把未完成的部分覆盖掉,修改后的文件集与检查点 $i$+1 是一致的.这样做还有一个很大的优点,就是检查点 $i$ 和 $i$+1 之间的运算没有浪费.

### 2.4 算法特点

本文提出的主要算法是 VFO 策略,其中还包含有针对检查点的分布检查点策略.这两个算法结合起来共同完成了文件检查点的保存和恢复.

VFO 策略把内存作为硬盘的缓冲区加以利用,把对硬盘的修改虚拟到内存中进行,提高了普通文件的读写速度.在具体设计中,借鉴了内存页式管理的方法,对文件的存取也以块为基本单位,提高了文件读写在内存中的命中率,更好地改善了文件操作的速度.在检查点设置方面,第 1 次提出了分布检查点的思想,通过把整合的步骤合理地分离,巧妙地安排各个操作的顺序,使得任何一个阶段的错误都可以得到准确的恢复,同时最大限度地保存了程序运行的结果.

具体而言,VFO 及相关算法有如下特点:

♦ 虚拟文件操作策略提高了正常文件操作的速度.每当要写一个地方的数据时,则根据我们的分块策略,把该处所在的块加载到虚拟文件管理器该文件所对应的项中的相应块中,即某一 FileEntry 的对应 Block 中.

♦ 检查点分布策略将最大限度地保存已完成的工作.本算法能够保证:只要分布策略中的第 2 步——同步检查点设置完毕,则写磁盘过程中的故障不会使恢复时卷回到上一个检查点,这在最大限度上保存了已经完成的工作.

♦ 极大屏蔽.对于文件的操作有很多,打开、关闭、移动指针,读、写、删除.我们的策略能够保证每一个操作对用户都是透明的,而且正常操作和恢复操作都可以保证正确.

♦ 统一的文件管理.每个进程的所有文件统一管理,方便查找.

### 2.5 VFO算法总执行流程(如图5所示)

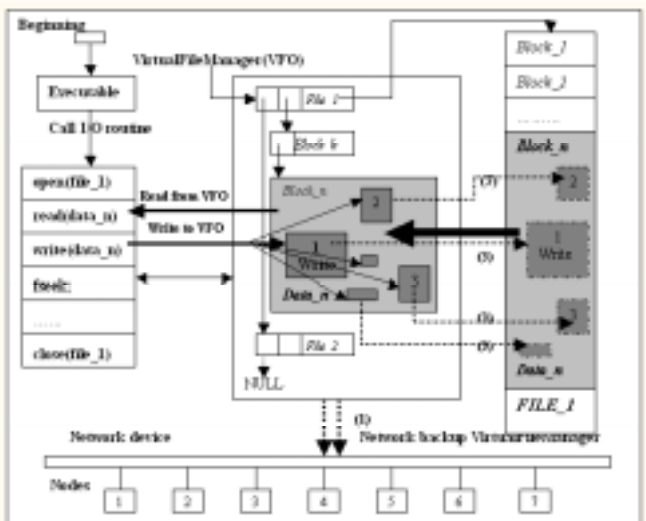


Fig.5 Virtual file operation

图 5 虚拟文件操作

## 3 相关工作

目前,已经有数个研究机构对文件检查点作了一定的研究工作,也提出了一些方案,典型的检查点设置库有libckp,libfcp,WOB 等.

在本文中提出的虚拟文件操作策略,借鉴了 WOB 的部分思想,针对写磁盘故障提出了检查点分布法、文件页管理方法,不仅保留并进一步优化了程序运行速度,而且很好地解决了写磁盘故障的恢复问题.

与 WOB,libckp 和 libfcp 的方法相比,本文提出的 VFO 策略具有如下特点:(1) 空间开销小.由于在一个检查点间隔内,用户文件中可能只有很小一部分内容发生改变,libckp 中对整个文件进行备份的方法空间开销很大.由于只对要写入用户文件的内容进行了缓存,VFO 引入的空间开销比 libckp,libfcp 的开销要小得多.(2) 正常运行时间开销小.VFO 策略由于采用了动态内存缓冲区分配和时延隐藏技术,给正常运行带来的额外时间开销较小,甚至有可能提高程序运行速度.(3) 可恢复程度最大.与其他方法相比,本文提出的分布检查点策略能够最大程度地恢复已有的工作,而不是轻易卷回上一个检查点,这在客观上降低了故障对程序运行造成的损害.

## 4 结论和进一步的工作

本文提出的 VFO 策略,实现了对用户活动文件的检查点设置,支持用户进程进行任何文件操作,有效地解决了在发生故障时用户文件的卷回恢复的问题.它对用户透明,并通过文件分布检查点设置、文件内容页式管理等手段,使得这种策略在空间开销、正常运行时间、恢复时间和卷回深度等性能指标上优于其他方法.

在今后的工作中,我们准备对 VFO 策略做进一步的性能优化.通过高效的文件动态分块策略,降低 I/O 开销;并且对 VFM 数据结构的联机备份方法做进一步的研究,以减少检查点设置开销. 网 网 网

**References:**

[1] Tannenbaum, T., Litzkow, M. The condor distributed processing system. Dr. Dobb's Journal, 1995,(2):40~48.

[2] Plank, J.S., Beck, M., Kinsley, G. Libckpt: transparent checkpointing under Unix. In: Usenix Winter 1995 Technology Conference. 1995. 213~223.

[3] Wang, Y.M., Huang, Y., *et al*. Checkpointing and its applications. In: Proceedings of the IEEE 25th Symposium on Fault-Tolerant Computing. 1995. 22~31.

[4] Pei, Dan, Wang, Dong-sheng. WOB: a new approach to checkpoint active file. Chinese Journal of Electronics, 2000,28(5):9~12 (in Chinese).

附中文参考文献:

[4] 裴丹,汪东升.WOB:一种新的文件检查点设置策略.电子学报,2000,28(5):9~12.

# A Files Checkpointing Approach Based on Virtual File Operations*

LIU Shao-feng, WANG Dong-sheng, ZHU Jing

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

E-mail: {qingfengl,glasswort}@263.net; wds@tsinghua.edu.cn

http://www.tsinghua.edu.cn

**Abstract:** Checkpointing and rollback recovery of Unix process are the underlying technique of fault tolerance for distributed system and parallel environment. To save and restore the state and the content of active file of the process is an important aspect of checkpointing and rollback recovery. A new file checkpointing approach called virtual file operation (VFO) is presented. VFO buffers all the write operations after a checkpoint until the next one, making all the operations between two checkpoints atomic as a whole. By step-to-step checkpointing, and managing file as blocks, this approach achieves lower overhead than the others.

**Key words:** file checkpointing; rollback recovery; VFO (virtual file operation); file set; fault

* Received March 30, 2001; accepted August 20, 2001

Supported by the National High Technology Development 863 Program of China under Grant No.863-306-ZD0102; the Foundation for University Key Teacher