参赛队员姓名：周益帆

中学：无锡市大桥实验学校

省份：江苏省

国家/地区：中国

指导教师姓名：吴咏

论文题目：一种基于行为特征的文件检查点优化策略

本参赛团队声明所提交的论文是在指导老师指导下进行的研究工作和取得的研究成果。尽本团队所知，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果。若有不实之处，本人愿意承担一切相关责任。

参赛队员：周益帆　　指导老师：吴咏

2017 年 9 月 10 日

# 一种基于行为特征的文件检查点优化策略

周益帆
（无锡市大桥实验学校，江苏省无锡市 214000）

**摘要：**检查点机制是系统容错的一项重要手段。它能够在程序运行的某一时刻保存程序的运行状态，并在系统故障后恢复程序状态继续执行。由于文件操作在应用程序中的普遍性，支持文件回卷对于检查点技术来说是十分必要的。文件数据完全备份可以使文件在回卷后恢复到正常状态，但是开销太大。本文提出了一种基于行为特征的文件检查点优化策略（BBFC），能够提供文件数据的正确恢复，有效保证了程序回卷恢复到上一个检查点时文件状态与进程其它状态保持一致。BBFC 对文件行为特征进行分类，并根据这些行为特征采取相应的保存恢复策略，从而在很大概率上减少了检查点间隔需要保存的文件内容，降低了文件检查点的时间、空间开销。它对用户透明，简单易用。

**关键词：**计算机软件；文件检查点；行为特征；回卷恢复；一致性；检查点间隔
**中图分类号：**TP316　　**文献标识码：**　A

# A Behavior Based File Checkpointing Strategy

ZHOU Yifan
(*Wuxi Big Bridge Academy, Wuxi 214000, China*)

【**Abstract** 】 Checkpoint/restart is an important method of system fault tolerance. It saves the state of an executing program periodically and recovers it after a failure. As many applications involve file operations, supporting file rollbacks is essential for checkpoint/restart. Complete backup can restore files to the correct state, but its cost is too high. In this paper we propose a behavior based file checkpointing strategy (BBFC), which provides a correct recovery of file data and ensures consistency between file state and other states of a process when a rollback is done by restarting the program from the last checkpoint. BBFC classifies details of the file operation behaviors and provides a guidance on what to be saved during file checkpointing according to those behaviors. It dramatically minimizes the overhead of file checkpointing due to the reduction of file data which need to be saved. And it is transparent and easy to use.

【Key words】computer software; file checkpointing; behavior based; rollback recovery; consistency; checkpoint interval

# 目录

# 0 引言

随着信息技术的发展，人们对计算机的依赖性日益增强。科学计算、数据分析、信息处理等各种问题的解决都离不开计算机，计算机系统的可靠性也越来越受到广泛关注。检查点机制作为系统容错的一项重要手段，能够在进程运行的某一时刻保存当时进程的运行状态到磁盘文件，并在需要的时候从保存的磁盘文件中恢复进程状态继续执行下去。检查点机制能够避免在系统故障后程序从头开始执行带来的计算损失，也给无法满足长时间占用计算资源的应用场景提供了便利的解决方案。

文件读写是应用程序的一个普遍行为。很多程序在正常运行时离不开对数据的处理和对文件的访问，因此在检查点中实现文件的可恢复性是至关重要的。文件检查点能够提供文件数据内容的正确恢复，使得程序回卷到上一个检查点时，文件内容与进程的其它状态保持一致。文件数据全备份能够解决这一问题，但是由于实际应用中大文件越来越多，保存文件内容所带来的开销不容忽视。

本文提出了一种基于行为特征的文件检查点优化策略，可以在很大概率上减少检查点间隔需要保存的文件内容，从而大大减小了因文件检查点给系统正常运行带来的额外开销，让检查点技术更为实用。

# 1 文件检查点技术现状

国际上主要的检查点系统有 Condor[1]、libckpt[2]、libckp[3]、CRAK[4]、BLCR[5]等，其中 Condor、libckpt、libckp 是在用户层实现的，优点是不用修改操作系统核心代码，可移植性较好，缺点是需要修改应用程序源代码，进程的一些系统状态不能被完全恢复；CRAK、BLCR 提供了操作系统内核级支持，可以较充分地恢复进程状态，对用户透明。Berkeley 实验室的 Linux Checkpoint/Restart 项目（BLCR）实现比较完善，包含了两个核心模块和一些用户库，支持单线程、多线程、进程树、进程组的检查点。

活动文件状态的正常保存与恢复是检查点技术的重要组成部分。文件状态包括文件属性信息和文件数据内容。文件的某些属性信息如文件名、文件属主、文件权限等不依赖于进程的运行状态，另外一些属性信息如文件描述符、访问方式、文件长度、读写指针等则依赖于进程运行状态。文件数据内容属于持续性状态，在进程终止后不会丢失。如果进程在检查点间隔中对文件内容做过修改，而上一个检查点中又没有保存被修改前的原数据，那么在进程回卷恢复后可能会因为文件状态和进程其他状态不一致而发生错误。

在检查点中保存文件属性信息并备份整个文件的数据内容可以保证文件状态的正确恢复，但是在涉及大文件的应用场景中，这样会导致很大的时间空间开销，让用户不能接受。因此大部分检查点系统例如 Condor[1]、BLCR[5]等仅支持文件名、读写指针等活动文件属性信息的保存与恢复，而没有保存文件内容。目前仅有少数检查点系统支持文件内容的回卷恢复，如 libckp[3]、libfcp[6]、LIBVFO[7]等。AT&T 实验室开发的 libckp 把活动文件内容作为进程状态的一部分，在文件被修改或被删除前做文件的影子拷贝。libckp 的改进版 libfcp 采用日志文件的策略，在文件被修改前将 undo 信息记录在日志文件里，在程序回卷恢复时从日志文件中倒序执行 undo 操作即可恢复文件状态。这两种策略给程序正常运行带来的开销都

比较大。清华大学开发的 LIBVFO 系统采用了一种延迟写的策略，将检查点间隔中的文件操作暂存在虚拟文件系统的数据区，等下一个检查点时才真正写入文件。LIBVFO 新增了一层虚拟文件系统，对操作系统改动比较大，实现上较为复杂。

# 2 文件行为特征及其保存恢复策略

根据文件的访问方式，可以大致预测出文件操作的行为特征。例如对于以只读方式打开的文件只能进行读操作，对于以只写方式打开的文件只能进行写操作，而以读写方式打开的文件则可能进行读操作，也可能进行写操作。假设应用程序在检查点 n 之后的 t 时刻中断执行，并回卷恢复到上一个检查点 n 的状态继续向下执行。根据行为特征的不同，文件检查点可以采取以下不同的保存恢复策略。

## 2.1 只读文件的保存恢复策略

只读文件内容不会被修改，因此检查点仅需保存文件属性信息，如文件名、打开方式、读写指针等，不需要保存文件内容；当程序回卷恢复时，重新打开文件，并重置这些文件属性信息，就可以继续正常运行[8]。

## 2.2 只写文件的保存恢复策略

只写文件内容在检查点间隔可能会发生变化。一种情况是部分文件内容被修改，如图 1 所示，图 1（a）是 n 时刻的文件状态，图 1（b）是 t 时刻文件被修改后的状态。这种情况仅需保存文件名、打开方式、读写指针等文件属性信息。当应用程序回卷到上一个检查点，恢复 n 时刻的文件属性信息后，由于后续还会在相同位置重新执行写操作，覆盖掉脏数据，因此不需要保存文件内容。

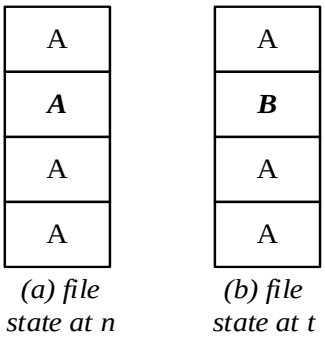


图 1 被修改前后的文件状态

Fig.1 File state before and after being modified

另一种情况是文件被追加了，如图 2 所示，图 2（a）是检查点 n 时刻文件状态，图 2（b）是检查点 n 后的 t 时刻文件被追加后的状态。由于追加操作是将读写指针定位到文件末尾后进行的写操作，因此在程序在回卷到检查点 n 时，必须在恢复其他文件属性信息的基础上，将文件长度恢复到 n 时刻的状态，否则可能会发生对同一内容追加两次的情况，从而造成文件内容与进程状态不一致。在这种情况下同样的，程序在回卷恢复后，重新执行的写操作会覆盖脏数据，因此不需要保存文件内容。

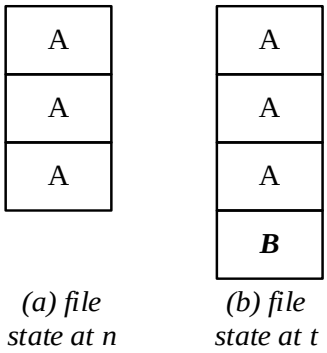


图 2 被追加前后的文件状态

Fig.2 File state before and after being appended

### 2.3 读写文件的保存恢复策略

以读写方式打开的文件，行为特征比较复杂，可以分为以下几种类型。

1）如果从 n 时刻到 t 时刻之间没有发生过写操作，文件内容未发生变化，那么显然仅需保存文件属性信息，不需要保存文件内容。

2）如果在 n 时刻和 t 时刻之间发生过写操作，该写操作发生在 w 时刻，并且从 n 时刻到 w 时刻之间该写入区域没有发生过读操作，那么文件状态的变化仍可归结为图 1、图 2 这两种情况。这时仅需在检查点 n 中保存文件名、文件长度、打开方式、读写指针等文件属性信息，当程序回卷恢复时，恢复文件属性信息并将文件长度截取到 n 时刻的文件长度。由于后续还会在相同位置重新执行写操作，覆盖掉脏数据，因此不需要保存文件内容。

3）如果在 n 时刻和 t 时刻之间发生过写操作，该写操作发生在 w 时刻，并且在 n 时刻到 w 时刻之间该写入区域发生过读操作，该读操作发生在 r 时刻，如图 3 所示，那么当程序回卷到检查点 n 继续往下执行时，在 r’ 时刻会读取到与 r 时刻状态不一致的数据，造成应用程序运行错误。因此如果在检查点 n 之后对文件同一区域发生了先读后写的操作，则需要保存检查点 n 时刻的文件内容。

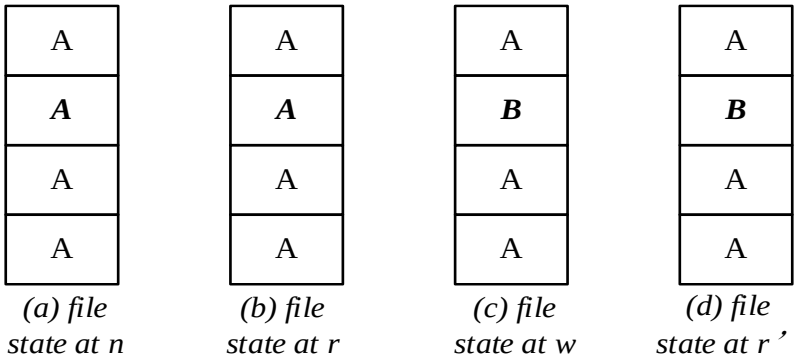


图 3 文件被读取后又被修改

Fig.3 File being read before modified

## 3 文件检查点算法及实现

### 3.1 基本思想

为了保证检查点系统中文件状态与进程状态的一致性，备份整个文件内容是最简单的方

法，但这也带来了巨大的时间和空间开销，严重影响了检查点系统的可用性。本文基于文件操作的行为特征，提出一种基于行为特征的文件检查点优化策略 BBFC（Behavior Based File Checkpointing），能够在很大概率上减少检查点间隔需要保存的文件内容，从而降低文件检查点的时空开销。

BBFC 的基本思想是根据文件的行为特征，尽可能少地保存文件内容来减小开销。根据文件的打开方式，对于只读和只写的文件，在每次执行检查点时只需保存文件的属性信息，而不必保存文件内容。具体做法是在检查点中保存文件名、文件长度、打开方式、读写指针等，当程序回卷恢复时重新打开文件，恢复读写指针等文件属性信息，并将文件截取到原来的文件长度。

对于读写文件，也仅有在检查点间隔中对文件同一区域发生了先读后写操作的情况下，必须保存相应区域的文件内容，以避免在程序恢复执行后读取到被修改过的错误数据，其它情况只需保存文件的属性信息。具体做法是对于要做文件检查点的读写文件，在检查点间隔中通过读写状态表来标记文件的读或写操作，对于同一文件块发生读操作后第一次写操作的，将该文件块的偏移以及文件块原数据记录到 log 文件中。程序回卷恢复时除了恢复文件属性信息及文件长度，还需要将 log 文件中保存的文件块原数据写回。保存文件内容的时点选择在一个检查点间隔中对同一文件块发生了读操作后的第一个写操作处，这样即使后续又发生了多次写操作，每个文件块在一个检查点间隔中还是最多只被保存一次，避免了文件内容的多次重复保存。这样做既降低了时间空间开销，又能使文件回卷恢复后的状态与 log 文件中的文件块数据被写回的顺序无关。

### 3.2 主要数据结构

#### 3.2.1 读写状态表

BBFC 通过文件块来对文件操作进行管理，将文件划分成相同长度的文件块，当发生多次涉及相同文件块的少量数据读写时，可以自动合并文件块内容的保存操作，显著减少开销。在设置检查点和回卷恢复时生成读状态表和写状态表，状态表的一位对应一个文件块，如图 4 所示。如果文件长度为 L，文件块大小为 B，则状态表大小为 L/(8B)字节，不足补齐。读状态表中的比特位置 1 表示该文件块在该检查点间隔中发生了读操作，比特位为 0 表示该文件块在该检查点间隔中尚未发生读操作。写状态表中的比特位置 1 表示该文件块在该检查点间隔中发生了写操作，比特位为 0 表示该文件块在该检查点间隔中尚未发生写操作。如果在检查点间隔中发生了原文件长度 L 范围外的读写操作，则一定有追加操作在先，不符合先读后写的顺序，因此不用在读写状态表中维护。

维护文件操作状态表的数据结构如下：

```
#define BBFC_BLOCK_SIZE 4096        //文件块大小，暂定为 4096 字节；
struct fileop_state_map{
    char BBFC_flag;                 //BBFC 标志，是否启动 BBFC，1 为是，0 为否；
    unsigned long file_length;      //上一次检查点时的文件长度 L；
```

```
    char * r_state_map;          //读状态表指针；
    char * w_state_map;          //写状态表指针；
}
```

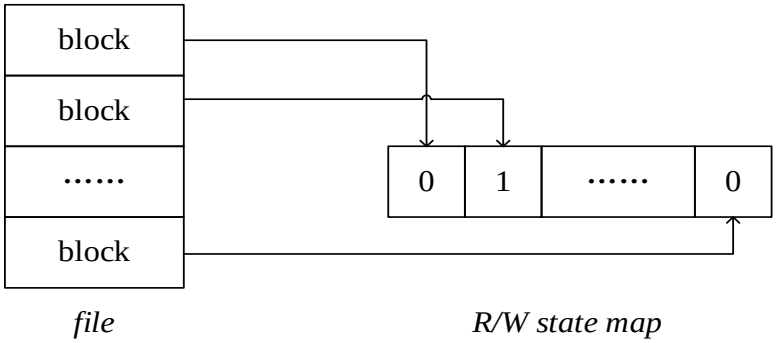


图 4 读写状态表

Fig.4 Read/write state map

### 3.2.2 log 文件

BBFC 中每个以读写方式打开的文件都对应一个 log 文件。log 文件在对应文件打开后的第一次检查点时被创建，在每次执行完检查点后被清空。log 文件记录的是为维持文件状态与进程状态的一致性必须保存的文件块信息，包括文件块偏移和文件块数据。每个被记录的文件块对应 log 文件中的一个 offset 字段和一个固定长度的 data 段。当程序回卷恢复时，从 log 文件中顺序读取文件块信息，并按照文件块偏移将文件块数据写回原文件的相应区域。log 文件的格式如图 5 所示。

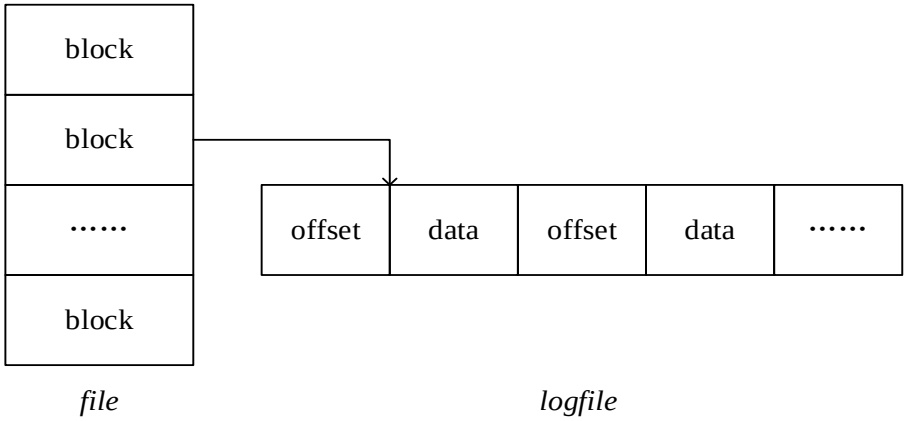


图 5 log 文件格式

Fig.5 logfile format

## 3.3 算法

### 3.3.1 文件操作的系统调用封装

为实现对文件行为特征的监控，BBFC 对 read、write、truncate、unlink 等系统调用进行

封装，在这些函数里截获文件操作的具体信息，进行读写状态表的维护和文件块数据备份等操作。系统调用封装对应用程序是透明的，用户不必对应用程序源程序做修改。具体算法如下：

```
read(){
    if(BBFC_flag==1){
        遍历该读操作在文件长度 L 范围内的相关文件块在读状态表中对应的比特位{
            读状态表对应比特位置 1；
        }
    }
    original_read();
}
write(){
    if (BBFC_flag==1){
        遍历该写操作在文件长度 L 范围内相关文件块在读写状态表中对应的比特位{
            if(读状态表对应比特位==1 &&写状态表对应比特位==0){
                记录该文件块信息到 file.log；
            }
            写状态表对应比特位置 1；
        }
    }
    original_write();
}
```

truncate 可以看作是对截断范围内相关文件块的清空写操作， unlink 可以看作对整个文件的清空写操作，实现流程都与 write 类似。

### 3.3.2 文件检查点的设置和恢复

文件检查点的设置和恢复具体算法如下：

```
file_checkpoint(){
    遍历每一个已打开的活动文件{
        保存文件名、文件长度、打开方式、读写指针等文件属性信息;
        if(以读写方式打开){
            置 BBFC_flag 为 1；
            置 file_length 为当前文件长度 L；
            如非空则释放原读、写状态表；
            生成读状态表并清 0;
            生成写状态表并清 0;
```

```
                if(file.log 不存在)
                    创建 file.log;
                }else  清空 file.log;
            }
        }
    }
    file_restore(){
        遍历每一个检查点保存的文件{
            打开文件，恢复文件属性信息;
            恢复文件到原来长度;
            if(以读写方式打开){
                从 file.log 中依次读取所有文件块信息写入原文件，恢复原文件内容;
                置 BBFC_flag 为 1;
                置 file_length 为当前文件长度 L;
                生成读状态表并清 0;
                生成写状态表并清 0;
                清空 file.log;
            }
        }
    }
```

### 3.4 算法性能及特点

BBFC 是一种基于行为特征的文件检查点优化策略。BBFC 算法在时间上的开销主要包含设置检查点时的开销和程序运行期间读写操作的额外开销。我们选取两种极端情况，以只读/只写方式打开文件以及以读写方式打开文件并对每个文件块都先读后写，其它情况的开销都应介于两者之间。原型系统基于 linux2.6.32 和 blcr-0.8.5[9]改造实现，实现进程状态检查点的基本功能由 BLCR 完成。由于 BLCR 仅支持文件名等属性信息的保存与恢复，BBFC 在它的基础上增加了对于文件内容回卷恢复的支持。测试环境是 CentOS 6.5，1G 内存，40G 硬盘的系统，文件块大小 4096 字节。测试结果如图 6 所示，横坐标是文件长度(G)，纵坐标是系统时间开销（S）。

如果文件的打开方式是只读或者只写，那么由于只需保存文件的属性信息作为检查点的一部分，时间和空间开销都很小，程序运行期间的文件正常读写不受影响，几乎可忽略。如果文件以读写方式打开，那么需要在文件正常读写操作前增加维护读写状态表的操作，由于读写状态表在内存里，这部分时间开销是很小的；如果进一步还满足在检查点间隔中对相同文件区域先读后写的条件，那么就需要备份相关文件块内容，这部分开销相对较大，在图 6 中可以看到这种情况下的系统时间开销几乎与需要备份的文件块数量成正比。BBFC 剔除了

只读、只写、追加、先写后读等其它很多情况，只对先读后写的文件块进行保存，并且相关的文件块最多只会备份一次。因此相比 libckp 在每次设置检查点时都要做整个文件的影子备份，libfcp 保存检查点间隔的所有写操作相关的文件数据，BBFC 可以在很大概率上减少检查点间隔需要保存的文件内容，时间、空间开销明显低于 libckp、libfcp 等检查点系统。BBFC 策略对用户透明，它对操作系统的改动较小，相比 LIBVFO 在实现上更为简单。

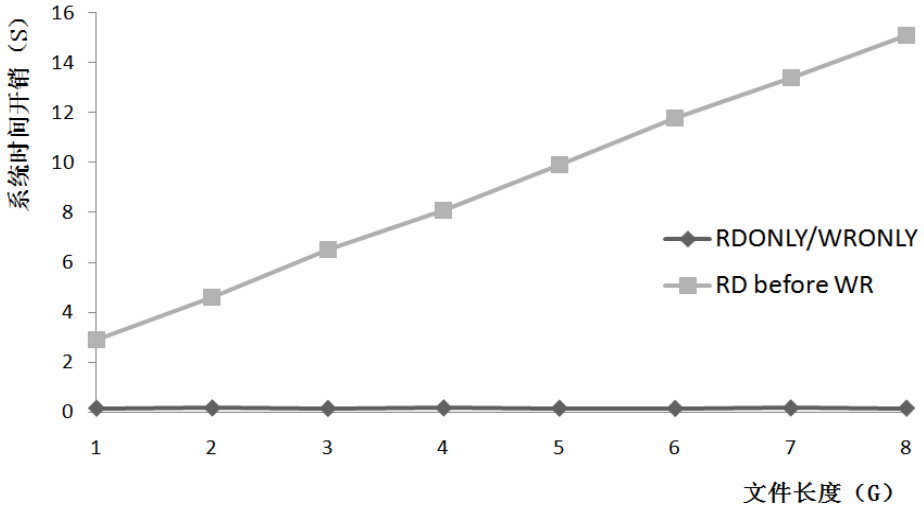


图 6 两种极端情况下 BBFC 的系统时间开销

Fig.6 time overhead of BBFC in two extreme cases

## 4 结束语

本文提出的 BBFC 策略，实现了对进程相关的活动文件设置文件检查点的功能，能够有效保证程序回卷恢复时文件状态与进程全局状态保持一致。BBFC 策略在操作系统核心层实现，对用户透明，不需要修改应用程序的源代码。通过对文件操作行为特征的分类和分析，BBFC 策略大大减少了检查点间隔需要保存的文件内容，降低了文件检查点的时间、空间开销，在性能上优于 libckp、libfcp 等检查点系统。

BBFC 策略中文件块大小的划分相当重要，文件块太大可能会保存一些无用数据，文件块太小又会引起频繁保存操作。未来计划通过分析具体应用的文件行为特征，设计合理的动态文件分块策略，使得 BBFC 策略的性能得到进一步优化。

# 参考文献

[1] Michael Litzkow, Todd Tannenbaum, Jim Basney, et al. Checkpoint and Migration of UNIX Processes in the Condor Distributed System [OL]. [2017-08-29]. http://www.cs.wisc.edu/condor/doc/ckpt97.ps.

[2] J S Plank, M Beck, G Kingsley, et al. Libckpt: Transparent Checkpointing under UNIX [A]. Usenix Winter 1995 Technology Conference[C]. New Orleans, Louisiana: Usenix, 1995.213-223.

[3] Yi-Min Wang, Yennun Huang, Kiem-Phong Vo, et al. Checkpointing and Its Applications[A].International Symposium on Fault-Tolerant Computing[C]. Pasadena, California: IEEE, 1995. 22-31.

[4] Hua Zhong, Jason Nieh. CRAK: Linux Checkpoint / Restart as a Kernel Module [OL]. [2017-08-29]. http://systems.cs.columbia.edu/files/wpid-cucs-014-01.pdf.

[5] Jason Duell. The Design and Implementation of Berkeley Lab's Linux Checkpoint/Restart [OL]. [2017-08-29]. http://crd-legacy.lbl.gov/~jcduell/papers/blcr.pdf.

[6] P E Chung, Y Huang, S Yajnik, et al. Checkpointing in CosMic: a User-level Process Migration Environment [A]. Proceedings of the 1997 Pacific Rim International Symposium on Fault-Tolerant Systems[C]. Washington DC, USA: IEEE, 1997.187-193.

[7] Liu Shaofeng, Wang Dongsheng, ZhuJing. A File Checkpointing Approach Based on Virtual File Operations [J]. Journal of Software, 2002, 13(8):1528-1533 (in Chinese).

(刘少锋, 汪东升, 朱晶. 基于虚拟文件操作的文件检查点设置[J]. 软件学报,2002,13(8):1528-1533).

[8] Eric Roman. A Survey of Checkpoint/Restart Implementations [OL]. [2017-08-29]. http://crd.lbl.gov/assets/pubs_presos/CDS/FTG/Papers/2002/checkpointSurvey-020724b.pdf.

[9] Berkeley Lab Checkpoint/Restart for Linux (BLCR) Downloads [OL]. [2017-08-29]. http://crd.lbl.gov/assets/Uploads/FTG/Projects/CheckpointRestart/downloads/blcr-0.8.5.tar.gz.

# 致谢

在这个项目中，我承担了研究报告撰写相关的所有工作。在科研老师的指导下，我查阅了很多检查点相关资料，了解检查点技术作为系统容错的一项重要手段，它的现状、关键技术和尚待解决的问题。在确定了项目的选题以后，我比较了几种主流文件检查点的解决策略，在此基础上通过分析文件操作的行为特征进行算法改进，使算法尽量能提高效率并且实现简便。在原型系统设计过程中，我选取了一个实现较为完善的开源项目 BLCR，用它来实现保存、恢复进程状态的基本功能，并在 linux2.6.32 和 blcr-0.8.5 的基础上增加了对于文件内容回卷恢复策略的支持。经测试评估，该算法的性能明显优于几种主流检查点系统。

在此我要感谢我的指导老师，她曾无数次不厌其烦地解答我的问题，并在整个项目过程中给予我耐心细致的指导和帮助。我还要感谢我的舍友们，他们创造了一个和谐的学习环境，在我遇到困难时总能及时伸出援助之手。最后感谢所有培养过我的老师以及和我朝夕相处的同学们，谢谢大家！