DOI：10.7652/xjtuxb201402001

一种具有容错机制的 MapReduce 模型研究与实现

史楠[1]，耿晨[2]，齐勇[1]
（1.西安交通大学电子与信息工程学院，710049，西安；2.中航工业西安飞行自动控制研究所，710065，西安）

摘要：针对传统 MapReduce 模型的容错机制对错误的处理效率低下等问题，提出了一种基于多核虚拟机的具有容错机制的 MapReduce 模型。该模型使用检查点机制进行错误恢复，并只对中间结果和必要的状态信息进行保存；利用虚拟机在隔离内存中保存中间结果；根据用户的需要及系统的负载情况动态调整系统中工作节点的个数。通过在 SUN 的 32 核、主频为 2.38 GHz、内存为 128 GB服务器上的测试，结果表明：与传统 MapReduce 模型相比，改进 MapReduce 模型降低了通信上的开销，提高了 MapReduce 运行过程的可靠性和错误恢复的性能，虚拟机监控器可以完全控制和管理多核平台的内存，使操作系统无法直接访问隔离的内存，数据恢复不会受到操作系统内部各种错误的影响，保证了恢复数据的安全性。

关键词：多核；虚拟机；容错；MapReduce
中图分类号：TP311 **文献标志码**：A **文章编号**：0253-987X(2014)02-0001-07

A MapReduce System with Fault-Tolerant Mechanism

SHI Yi[1], GENG Chen[2], QI Yong[1]
(1. School of Electronics and Information Engineering, Xi'an Jiaotong University, Xi'an 710049, China;
2. AVIC Xi'an Flight Automatic Control Research Institute, Xi'an 710065, China)

Abstract: A MapReduce with fault-tolerant mechanism based on multi-core virtual machine is proposed to solve the problem that the simple fault-tolerant mechanism in traditional MapReduce is prone for duplicate calculation in error processing. The system uses checkpoint mechanism to perform error recovery and only saves intermediate results and necessary state information. It stores the intermediate results in isolated memory through virtual machine. The number of worker nodes is dynamically adjusted according to the user's need and the system load. Experiments are conducted on 32-core, 2.38 GHz, 128 GB SUN server and the results show that the proposed MapReduce reduces communication cost, and improves system reliability and error recovery performance. Virtual Machine Monitor can entirely control and manage the memory of a multi-core system and disable OS to visit the isolated memory directly. The security of data recovery is guaranteed by avoiding affection of errors occurring inside the OS.

Keywords: multi-core; virtual machine; fault-tolerant; MapReduce

MapReduce[1]编程模型在任务处理的过程中，个别节点容易出现错误，出错节点对整个系统性能的影响较大。传统 MapReduce 模型的容错机制较为简单，对错误的处理容易产生重复计算，造成计算资源的浪费。在容错时如果使用传统的进程检查点机制，则开销较大并且难以保证恢复数据的安全性。

收稿日期：2013-09-17。 作者简介：史楠(1975—)，女，讲师。 基金项目：国家自然科学基金资助项目(60933003)；教育部高等学校博士学科点专项科研基金资助项目(20120201110010)。
网络出版时间：2014-01-15 网络出版地址：http://www.cnki.net/kcms/doi/10.7652/xjtuxb201402001.html

随着多核技术[2-3]的发展，多核资源越来越丰富，且呈现分布式的趋势。如何能够充分利用多核服务器的计算能力和硬件资源是一个随之而来的问题。现代操作系统通过不断增加系统的复杂性[4-5]来适应持续发展的硬件资源，虚拟化技术的出现在一定程度上解决了现有系统众多影响计算机性能的问题。虚拟化技术可以提高服务器资源利用率，同时减少服务器数量，以降低管理和维护的开销，降低能源消耗[6]。同时，虚拟化技术提供了很好的隔离性，使得虚拟机之间互不干扰，提高了系统的安全性和可靠性。

基于上述问题，本文通过分析现有 MapReduce 的实现方式，将多核平台与虚拟机技术相结合，提出了一种基于多核虚拟机的 MapReduce 模型(Virtual Machine Based Fault-Tolerant Mechanism，VMBFTM)。

1 任务分配模型

1.1 系统状态模型与分析

基于多核虚拟机的具有容错机制的 MapReduce，其系统状态模型如图 1 所示。

通过虚拟机监控器的虚拟化，在多核服务器上形成多个操作系统。其中一个操作系统叫做控制域，这个控制域把必要的接口暴露给用户，以便用户进行编程和数据处理。服务器管理员通过这个控制域对整个多核服务器进行维护，提交计算任务，调节整个系统负载，并对安全问题进行处理。其他操作系统受到控制域的监控，对网络或者外设的访问需要受到控制节点的限制。MapReduce 模型中调度节点 Master 进程运行在控制域 OS0。首先，用户在控制节点提交需要处理的数据，Master 进程对数据进行 Split 分割操作，划分成 N 个数据块。数据块的个数和整个多核服务器中 Worker 工作节点个数相同。然后 Master 和每个操作系统中的守护进程监控者进行通信，发出启动 Worker 的命令。监控者收到命令之后，启动 Worker，然后每个 Worker 创建指定个数的线程，对分配给它的数据进行 Map 映射操作。Map 操作处理待处理数据中每个记录，产生中间结果的键值对＜key，value＞，然后在相同 key 的结果上执行 Reduce 操作。在执行 Map 操作的过程中，根据用户指定或者系统默认的策略，进行检查点的设置。设置检查点时，需要陷入虚拟机，把 Map 当前的中间结果保存在虚拟机维护的 Map 结果存储区中，并记录相关控制信息，例如当前节点已经完成的任务量等。设置完检查点之后，返回操作系统，继续进行 Map 操作。完成 Map 任务之后，Map 产生的全部中间结果保存在共享内存中，接着关闭 Worker 进程，监控者通知 Master 本阶段完成，请求进行 Reduce 化简阶段的处理。Master 收到这个消息之后，重复上面的过程，进入 Reduce 阶段。Reduce 过程中，每个 Worker 需要读取 Map 的

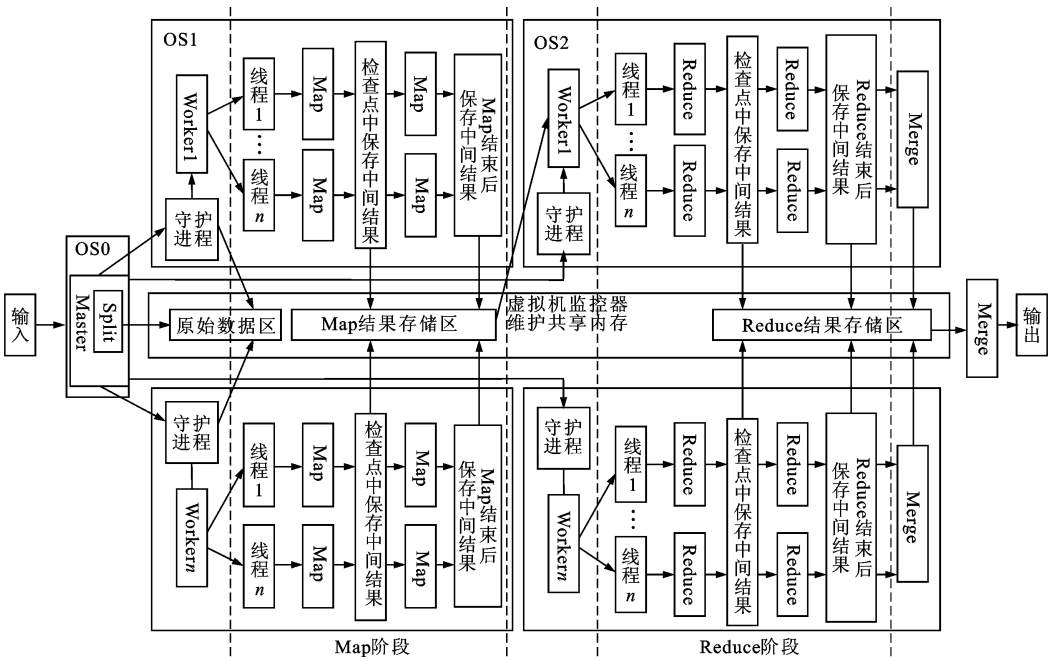

图 1 基于多核虚拟机的 MapReduce 模型

结果作为自己的输入。每个 Worker 通过访问共享内存中 Map 结果存储区，读取属于自己节点的数据进行 Reduce 操作，通过共享内存减少了传递大量中间结果带来的时间开销。在 Reduce 过程中，Reduce 的结果也被不断保存，以便在发生错误之后通过这些保存的数据进行恢复。

错误恢复模型如图 2 所示。接收到 Master 向监控者发出的命令之后，启动本节点内的 Worker，Worker 会创建多个线程进行 Map 或者 Reduce 操作。在此过程中，根据一定策略进行检查点的设置，通过增量方式把当前 Map 或者 Reduce 执行的任务保存在由虚拟机维护的内存区域内。如果 Worker 检测到错误发生则会通知监控者，监控者随即关闭本节点内的 Worker，并向 Master 发送消息，请求在本节点内重新启动一个 Worker。当 Master 发出启动命令之后，监控者重启 Worker，先从虚拟机维护的内存中读取已经做好的中间结果和原始分配的任务以及相应的控制信息等，然后对 Worker 进行初始化操作，完成错误恢复。系统就像没有发生错误一样，继续执行 Map 或者 Reduce 操作。如果本节点内分配的任务已经做完，监控者关闭 Worker。

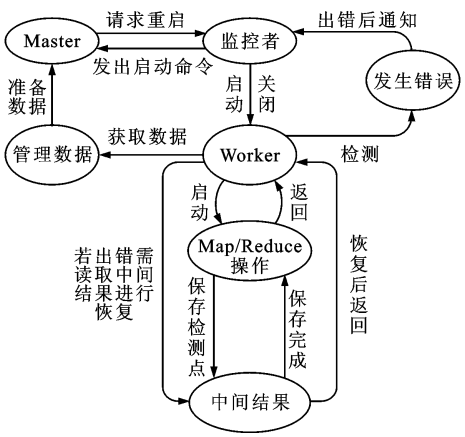

图 2　错误恢复过程模型

1.2　错误恢复开销研究

基于多核虚拟机的具有容错机制的 MapReduce 在 Worker 运行时通过对中间结果设置检查点达到错误恢复的效果。假设在 Worker 运行中，检查点是均匀分布的，设每个检查点间隔计算任务消耗的时间为 t_i，并且由于(key，val)的处理过程，每个检查点之间的计算任务消耗的时间大致相等，即 $t_i = T_{PA}$，T_{PA} 表示每两个检查点间隔计算任务消耗时间的平均值，T_{SA} 表示第 1 个检查点，即单位时间内保存中间结果的时间，且 $T_{PA} \approx T_{SA}$，T_{RA} 表示对单位时间内生成的中间结果进行恢复需要的时间。F_K 是第 K 次发生错误，n 表示程序运行完成过程中需要建立 n 个检查点。因此不设检查点的 Map 或者 Reduce 的执行时间为

$$T_N = t_1 + t_2 + \cdots + t_n = \sum_{i=1}^{n} t_i \tag{1}$$

设置检查点后，程序执行时间为

$$T_C = (t_1 + t_2 + \cdots + t_n) + nT_E + (T_{1S} + T_{2S} + \cdots + T_{(n-1)S}) = \sum_{i=1}^{n} t_i + nT_E + \sum_{i=1}^{n-1} T_{iS} \tag{2}$$

式中：T_E 表示每次进行检查点时陷入和返回虚拟机的时间，每次都相等，共有 n 次；T_{iS} 表示每次保存中间结果所需要的时间。

不设检查点出错之后，程序直接终止，所消耗的时间为

$$T_{NF} = t_1 + t_2 + \cdots + t_F = \sum_{i=1}^{F} t_i \tag{3}$$

不设检查点出错后进行恢复时，出错节点需要从头开始计算，由于可能在运行过程中多次产生错误，设每次错误之后运行到 t_{Fi}，因此整个过程消耗的时间为

$$T_N = t_1 + t_2 + \cdots + t_n = \sum_{i=1}^{n} t_i = (\sum_{i=1}^{F_1} t_i + \cdots + \sum_{i=1}^{F_K} t_i) + \sum_{i=1}^{n} t_i \tag{4}$$

式中：当 $i > j$ 时，$F_i > F_j$。进行检查后，在 F 处出错之后整个进程消耗时间为

$$T_{CF} = (t_1 + t_2 + \cdots + t_F) + FT_E + (T_{1S} + T_{2S} + \cdots + T_{FS}) = \sum_{i=1}^{F} t_i + FT_E + \sum_{i=1}^{F} T_{iS} \tag{5}$$

检查点出错后进行恢复时，发生错误次数为 K，错误发生点为 t_{Fi}，每次只要从最近的检查点继续执行即可，从最近检查点进行恢复的时间为 T_{FiR}，所消耗时间为

$$T_{CFR} = (t_1 + t_2 + \cdots + t_{F1}) + T_{F1R} + (t_{Fm} + t_{Fm+1} + \cdots + t_{F_K}) + T_{FKR} + nT_E + (T_{1S} + T_{2S} + \cdots + T_{(n-1)S}) + (t_{F_K+1} + \cdots + t_n) = (\sum_{i=1}^{n} t_i) + nT_E + \sum_{i=1}^{n-1} T_{iS} + \sum_{i=1}^{K} T_{FiR} \tag{6}$$

式中：T_{FiR} 表示从最近的检查点进行恢复数据所需要的时间，T_{FiR} 和保存最近检查点的数据所花费的时间大致相等。

为了比较不设置检查点和设置检查点之后恢复的效果，比较 T_{NFR} 和 T_{CFR}，即

$$T_{NFR}-T_{CFR}=$$
$$(\sum_{i=1}^{F_1}t_i+\cdots+\sum_{i=1}^{F_K}t_i)-(\sum_{i=1}^{n-1}T_{iS}+\sum_{i=1}^{K}T_{FiR}+nT_E) \tag{7}$$

如前所述，$t_i=T_{PA}$，且每次检查点保存和恢复的时间随着中间结果的增长而线性增加，因此 $T_{iS}=iT_{SA}$，所以可以进一步得出

$$T_{NFR}-T_{CFR}=$$
$$[F_1(1+F_1)/2+\cdots+F_K(1+F_K)/2]T_{PA}-$$
$$[F_1+\cdots+F_K+n(n-1)/2+nT_E/T_{SA}]T_{SA} \tag{8}$$

由于是轻量级虚拟机，虚拟机每次陷入和返回的时间和较短，因此 nT_E 可以忽略。而且对使用 MapReduce 模型进行处理的应用来说，计算产生中间结果花费的时间远远大于把这些中间结果保存在内存中花费的时间，即 $T_{PA}\gg T_{SA}$。进一步分析式(8)后，发现当 n 不太大的时候，$T_{PA}/T_{SA}<5$ 即可保证 $T_{NFR}>T_{CFR}$ 成立，即本文设计的容错机制相比传统的 MapReduce 容错机制更有优势。

2 系统的设计与实现

本文所设计的系统需要 3 个部分支持：具有容错机制的 MapReduce 引擎、用户定制程序以及虚拟化的多核服务器。三者关系如图 3 所示。前两者是软件支持，第 3 个是硬件平台支持。具有容错机制的 MapReduce 引擎是对 MapReduce 过程进行统一管理，对底层硬件平台进行抽象，提供良好的通信机制和数据管理能力，包括 Master 端、Worker 端以及监控者程序。用户定制程序是用户自己编写的对不同的应用案例进行特定处理的程序，主要是用户自定义的 Split，Map 和 Reduce 函数。虚拟化的多核服务器是运行系统的硬件平台，为具有容错功能的 MapReduce 系统提供硬件支持，这里是通过轻量级虚拟机监控器对多核服务器进行虚拟化，把多个隔离的操作系统作为隔离的节点，运行一个 Master 或若干 Worker。为完成系统功能要求，需要解决 3 个问题：一是如何进行数据处理；二是任务调度和通信效率；三是容错机制的建立。

2.1 数据处理过程

数据处理的前提是建立域间共享内存，它是整个系统的核心，也是其他模块运行的基础和前提条件。共享内存为多核系统 SMM[7] 提供简单编程模型，并且可以兼容大部分当前已有的应用程序和操作系统。共享内存系统[8]可使各个模块平等地访问系统中的物理内存。

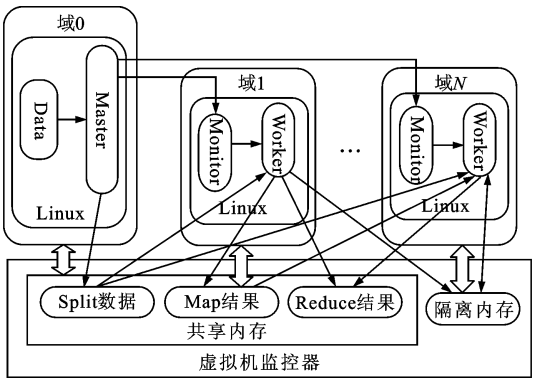

图 3 基于多核虚拟机的具有容错机制的 MapReduce

在 OS 内核中需要建立共享内存到内核空间的映射，然后建立用户地址空间和内核地址空间的映射，这样 OS 中的 Master 和 Worker 进程就可以自由地使用共享内存在不同 OS 之间通信。

数据处理模块把系统的控制信息等敏感数据保存在共享内存中；任务调度模块根据共享内存中的控制进行任务调度；节点通信模块使用共享内存进行节点之间的通信和交互；检查点模块把恢复数据保存在各个节点都能访问到的共享内存中，便于在出错后使用错误恢复模块进行恢复。

在数据输入阶段，对于 Master 节点，计算任务的数据一般以文件的形式提供，先存放在 Master 节点硬盘中。数据处理模块需要把计算任务文件读入内存，然后根据一定的规则进行分割，把这些数据存放在虚拟机维护的内存区域中。对于 Worker 节点，每个节点的输入数据是本节点需要处理的数据，由虚拟机调用。通过虚拟机维护的共享内存，Worker 进程可以对本节点的数据进行操作。由于每个节点只映射自己所要处理的数据的内存，不会对其他节点的数据进行影响和破坏。Map 阶段和 Reduce 阶段类似，从共享内存中读入本节点需要处理的数据，通过任务调度分发给每个线程，进行计算后输出到共享内存中，中途需要根据一定策略对中间结果进行必要的保存。

2.2 任务调度和通信

任务调度模块是在系统运行时，完成任务调度和负载均衡的功能。任务调度的过程包括监控者开启和关闭 Worker 进程，Worker 进程开创多个线程进行 Map 或者 Reduce 操作，操作完毕之后通知完

成工作。通过使用进程和线程结合的方式,在每个子域的操作系统上运行一个 Worker,然后每个 Worker 又创建多个线程,每个线程处理一块数据。这样能够很好的利用多进程和多线程的优势。

节点通信模块主要负责节点之间的通信,保证 Master 能够准确地控制 Worker 节点的执行,并在 Worker 节点的任务执行的过程中和完成任务之后,Master 节点能够及时了解此 Worker 节点的任务状态信息。

Master 进程直接和子域中的监控者进程进行通信。监控者是守护进程,一直运行在子域中,负责监听 Master 信号,在收到 Master 发过来的启动命令之后,启动本节点中 Worker 进程。在节点内部多线程执行任务的过程中,如果发生错误,Worker 进程崩溃,监控者进程发现 Worker 崩溃后,需要在全局任务控制表中属于本节点的表项中注明错误发生,同时 Master 进程会不断扫描全局任务控制表项。由于全局任务控制表是存放在各个域之间的共享内存中,因此 Master 通过定时扫描就可以及时发现状态的改变,并有效地对 Worker 进行监控和管理。

2.3　建立容错机制

本文给出的 MapReduce 的容错机制包括两方面:检查点设置和错误恢复。

在检查点时刻把当前线程的计算结果保存在虚拟机指定的隔离内存空间,出错之后在出错节点重新启动 Worker 进程,从最近的检查点读取中间结果数据和本节点任务的进度信息继续运行。由于(key,val)的特点,key 和产生它的进程以及所在的节点无关,因此可以由任意的线程继续对它进行处理。

这些恢复数据存放在由 VMM 指定内存中,OS 系统是无法感知和访问的,这样就形成了隔离屏障,保证当 Worker 进程甚至整个 OS 异常终止或者崩溃之后,这些恢复数据不会受到破坏和影响。VMM 接受虚拟机调用之后,在虚拟机里把线程已经处理好的数据保存在指定的专门存放中间结果的内存中,这块内存通过资源的隔离性保证了恢复数据的安全性。

在恢复的过程中,通过读取存放在全局任务控制表中的任务进度信息,Worker 进程创建线程并读取任务进度信息从而使每个线程恢复到设置检查点时的状态。由于(key,val)和线程状态无关,因此新的线程可以继续执行出错前任意一个线程的任务。由于检查点时刻的任务都保存在 VMM 维护的隔离内存中,因此需要陷入虚拟机,用保存的中间结果恢复 OS 中的所有线程共同操作的中间结果池。

3　实验结果与分析

测试的硬件平台是基于 SUN 的 32 核、主频为 2.38 GHz、内存为 128 GB 的服务器。硬件之上运行着本课题组设计实现的虚拟机监控器 OSV。相比于传统的虚拟机监控器,我们设计的 OSV 虚拟机监控器是轻量级的,易于维护,并且性能更高。虚拟机监控器对其上的操作系统分配资源,并且阻止其他操作系统未授权的访问,形成了资源的隔离。节点之间可以通过一般的 TCP/IP 进行通信,而对于大量的数据,通过虚拟机维护的共享内存进行数据交流在一定程度上提高了数据交互的性能。为了降低虚拟化对系统带来性能上的影响,虚拟机只是对资源进行访问控制,而没有对资源进行虚拟化。由于 VMM 在 OS 运行时处于挂起状态,并且 OS 对授权的硬件资源直接进行访问,因此相比于没有虚拟机的多核服务器,性能损耗很小。

3.1　MapReduce 应用性能测试

wordcount 应用是用来测试 MapReduce 性能的一个比较普遍的测试用例,其功能是对一个存放英文单词的大文件进行处理,统计其中的英文单词出现的频率,得出出现单词的种类和各个单词出现的次数。使用 wordcount 应用对基于多核虚拟机的具有容错机制的 MapReduce 在多核平台上进行测试,并和两种不同模式的多线程实现方式以及 Phoenix 系统进行对比。

斯坦福大学在多核服务器上实现了 Phoenix[9] 系统,直接运行在多核服务器上的一个操作系统,它是基于共享内存的 MapReduce,使用多个线程完成 Worker 的任务。

第一种多线程方式运行在多核服务器上唯一的操作系统中,这个操作系统完全掌控所有的硬件资源,即单操作系统多线程方式,通过使用 Pthread 多线程进行 wordcount 应用的处理。第二种多线程方式是在多核平台上使用虚拟机监控器进行资源隔离,运行多个操作系统,每个操作系统上运行一个 Worker 进程,每个 Worker 进程创建 4 个线程,即多操作系统多线程方式。每个操作系统相当于分布式环境下单独的一个计算节点,借助 MapReduce 思想进行 wordcount 处理。

对 4 线程、8 线程、12 线程、16 线程 4 种情况测

试 1 GB 大小的 wordcount 文件的计算时间，结果如图 4 所示。可以看出，Pheonix 性能略高，而单操作系统多线程方式和基于多核的具有容错机制的 MapReduce 性能类似，并且性能与 Pheonix 相差不大，而多操作系统多线程方式的性能最差。

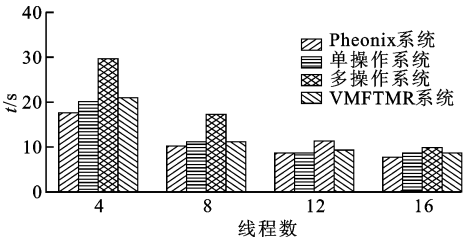

图 4 4 种方式进行 wordcount 计算的执行时间

基于多核虚拟机的具有容错机制的 MapReduce 和 Phoenix 性能差距较小。由于通过使用节点之间的共享内存进行数据传递比通过网络传输速度更快，而且作为 Reduce 输入数据的中间结果保存在节点之间的共享内存中，执行 Reduce 的 Worker 可以直接对这块内存进行操作，避免了对硬盘访问带来的开销。

3.2 错误恢复测试与评价

由图 5 可见，当错误发生之后，监控者进程迅速发现 Worker 进程出错，提示出错 Worker 进程的 PID，并重启 Worker 进程继续执行；如果 Worker 进程发生多次错误，每次出错之后监控者进程可以立即感知，启动新的 Worker 进程继续计算，直到完成整个任务。相比发生一次错误的情况，出现两次错误的执行过程所耗费的时间没有明显增加。

```
This is worker process with pid of
3612.worker start:
task runnning...
--------
the worker process 3612 crushed.
star new worker:

this is worker process with pid of
3615.worker start:
task runnning...
tatal time: 9.942 (s)
--------
worker end
the worker process3615 exi
normally.
```

```
this is worker process with pid of
3627 worker start:
task runnning...
--------
the worker process 3627 crushed.
start new worker:

This is worker process with pid of
3630. worker start:
task runnning...
--------
the worker process 3632 crushed.
start new worker:

This is worker process with pid of
3630 worker start:
task runnning...
total time: 11.012 (s)
---------
worker end.
the worker process 3632 exit
normally.
```

(a)发生一次错误 (b)发生多次错误

图 5 错误恢复过程

图 6 为不同出错次数情况下各种并行计算方式出错后重启并完成计算所耗费的时间。

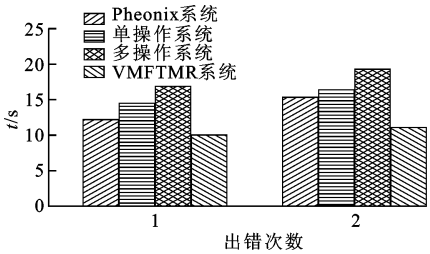

图 6 不同出错次数的恢复时间比较

基于多核虚拟机的具有容错机制的 MapReduce 在出错之后可以立即重启并继续执行，已完成的结果没有丢失，不会造成计算任务的浪费。由图 6 可见，多次恢复之后，本模型相对于其他 3 种方式，完成计算任务的时间最短，性能最优。

3.3 安全隔离功能测试与评价

通过对中间结果进行检查点设置，可以在错误恢复的过程中通过读取保存的数据进行恢复。传统使用检查点的保存方式是把数据保存在内存或者磁盘中，恢复的时候启动恢复策略将已经保存的数据恢复出来，然而由于操作系统受到攻击或者其他恶意程序，保存的数据受到污染，因此恢复之后的结果是错误的，如图 7a 所示，而基于多核虚拟机的具有容错机制的 MapReduce 系统可以进行正确的恢复，如图 7b 所示。

String:	occur time:
the	fdsajifowgew
add	gfghowxwfwrq
to	qpuzilwhfugz
a	orepgreiriew

String:	occur times:
the	4484434
add	3449565
to	1494911
a	574927

(a)传统检查点恢复 (b)基于多核虚拟机的恢复

图 7 恢复数据区对比图

虚拟机监控器可以完全控制和管理多核平台的内存，使操作系统无法直接访问隔离的内存，因此需要恢复的数据不会受到操作系统内部各种错误的影响，保证了恢复数据的安全性。

4 结 论

本文充分利用了多核服务器架构和虚拟化技术的特点，设计并实现了基于多核虚拟机的具有容错机制的 MapReduce，通过虚拟机监控器进行安全数据隔离和恢复，对中间结果进行保存以提高错误恢复的性能，降低了节点之间的数据传输开销。测试

了系统的性能、错误恢复的能力以及安全数据隔离的效果，并与其他并行程序的性能进行相应的对比。结果表明，本文所提出的改进的 MapReduce 模型提高了错误恢复的性能，保证了恢复数据的安全性。下一步将优化检查点策略，完善错误感知机制，减小保存和恢复过程中的性能损耗。

参考文献：

[1] DEAN J, GHEMAWAT S. MapReduce: a flexible data processing tool [J]. Communications of the ACM, 2010, 53(1): 72-77.

[2] MOHANTY R P, TURUK A K, SAHOO B. Analysing the performance of multi-core architecture [C]// Proceedings of the first International Conference on Computing, Communication and Sensor Networks. New York, USA: IJCA, 2013: 28-33.

[3] MERRITT R. CPU designers debate multi-core future [EB/OL]. (2008-02-06)[2012-10-02]. http://www.eetimes.com/document.asp? doc_id=1167932.

[4] DESNOYERS M, MCKENNEY P E, STEM A S, et al. User-level implementations of read-copy update [J]. IEEE Transactions on Parallel and Distributed Systems, 2012, 23(2): 375-382.

[5] Receive-side scaling enhancements in windows server [EB/OL]. (2008-11-05)[2012-10-15]. http://www.microsoft.com/whdc/device/network/ndis_rss.mspx.

[6] MATTHEWS J N, DOW E M, DESHANE T, et al. Running Xen: a hands-on guide to the art of virtualization [M]. New Jersey, USA: Prentice Hall, 2008: 56-59.

[7] CHAPMAN M. HEISER G. vNUMA: a virtual shared-memory multiprocessor [C]// Proceedings of the 2009 USENIX Annual Technical Conference. San Diego, USA: USENIX Association, 2009: 349-362.

[8] GULATI A, MERCHANT A, VARMAN P J. MClock: handling throughput variability for hypervisor I/O scheduling [C]// Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation. Berkeley, CA, USA: USENIX Association, 2010: 1-7.

[9] TALBOT J, YOO R M, KOZYRAKIS C. Phoenix++: modular MapReduce for shared-memory systems [C]// Proceedings of the Second International Workshop on MapReduce and Its Applications. New York, USA: ACM, 2011: 9-16.

[本刊相关文献链接]

赵银亮，朱常鹏，韩博，等. 以虚拟机为核心实现动态行为调整的方法. 2013，47(6)：6-10. [doi：10.7652/xjtuxb201306002]

苏莉，陈鹏飞，齐勇，等. 贝叶斯证据框架下最小二乘支持向量机的软件老化检测方法. 2013，47(3)：12-18. [doi：10.7652/xjtuxb201308003]

庄威，桂小林，林建材，等. 云环境下基于多属性层次分析的虚拟机部署与调度策略. 2013，47(2)：28-32. [doi：10.7652/xjtuxb201302005]

史槌，邱劲锋，侯迪，等. 一种敏捷服务组合方法模型的研究与设计. 2013，47(2)：1-6. [doi：10.7652/xjtuxb201302001]

李健，黄庆佳，刘一阳，等. 云计算环境下的大规模图状数据处理任务调度算法. 2012，46(12)：116-122. [doi：10.7652/xjtuxb201212020]

晁武杰，甘永梅，王兆安，等. 实时状态树结构模型的最优非阻塞模块化监督控制研究. 2012，47(4)：86-91. [doi：10.7652/xjtuxb201304015]

曹仰杰，杨海兵，钱德沛，等. 多核编程模型运行时环境的自适应性研究. 2011，45(6)：130-134. [doi：10.7652/xjtuxb201106023]

陈峰，李伟华，陈昊，等. 采用模型检测器的软件安全模型验证方法. 2011，45(2)：15-20. [doi：10.7652/xjtuxb201102004]

李小虎，杜海峰，张进华，等. 多层前向小世界神经网络的逼近与容错性能. 2010，43(12)：59-63. [doi：10.7652/xjtuxb201007014]

韦远科，赵银亮，宋少龙，等. 面向片上多核处理器的推测多线程机制下的独立栈模型. 2010，44(12)：10-15. [doi：10.7652/xjtuxb201012003]

崔筱宁，赵保华，李青，等. 传感器数据中事件样本与错误样本的系统化区分框架. 2010，44(10)：30-35. [doi：10.7652/xjtuxb201010006]

王得利，高德远. 片上多核中一种共享感知的数据主动推送 Cache 技术. 2010，44(10)：18-23. [doi：10.7652/xjtuxb201010004]

宋少龙，赵银亮，冯博琴，等. 支持推测多线程的展多核模拟器 Prophet^{+}. 2010，44(10)：13-17. [doi：10.7652/xjtuxb201010003]

赵敏，郑崇勋，赵春临，等. 利用 Fisher 判别式和事件相关电位的心理意识真实性识. 2010，44(8)：132. [doi：10.7652/xjtuxb201008026]

（编辑　武红江）