第35卷 第14期 计 算 机 工 程 2009年7月
Vol.35 No.14 Computer Engineering July 2009
·软件技术与数据库· 文章编号：1000—3428(2009)14—0052—03 文献标识码：A 中图分类号：N945

面向恢复的集群计算技术

鲁晓佩，廖湘科，卢宇彤
(国防科技大学计算机学院，长沙 410073)

摘 要：针对面向恢复计算(ROC)技术致力于在故障发生后使系统尽快恢复，从而提高系统可用性，而非从根本上避免故障发生的特点，对面向恢复的相关技术进行研究，给出ROC技术在集群系统中的应用，提出基于节点组的递归重启方法和基于Checkpoint的Undo恢复模型，用以提高集群系统的可用性，并对2种方法的改善效果进行评估。

关键词：可用性；面向恢复计算；递归重启；Undo模型；集群

Recovery-Oriented Cluster Computing Technology

LU Xiao-pei, LIAO Xiang-ke, LU Yu-tong
(School of Computer, National University of Defense Technology, Changsha 410073)

【Abstract】 Recovery-Oriented Computing(ROC) improves system availability by repairing the system as soon as possible, instead of avoiding failure. This paper studies ROC techniques, gives its application in cluster system, proposes the method of Recursive Restartability(RR) based on node group and Undo recovery model based on Checkpoint to improve system availability. It evaluates the improvement effect of the methods.

【Key words】 availability; Recovery-Oriented Computing(ROC); recursive restartability; Undo model; cluster

1 概述

计算机应用得越来越广泛，特别是高性能集群计算系统，在科技、经济、军事等各个领域发挥着越来越重要的作用。但随着集群系统中硬件系统及应用规模的不断扩大，各种软硬件故障发生的几率呈指数增长，完成计算作业所需要的时间也越来越长。系统中单个节点发生故障可能对整个系统造成很大影响或者使长时间运行的作业执行失败，造成资源的极大浪费。因此，提高集群系统可用性具有十分重要的意义。通常用2个参数评价系统可用性：系统2次故障之间的平均时间间隔，即平均无故障时间 *MTTF*；系统从故障中恢复所需要的平均时间，即平均恢复时间 *MTTR*。

根据可用性的定义[1]：

$$Availability = \frac{MTTF}{MTTF + MTTR} \tag{1}$$

提高系统的可用性可以通过2种方式：(1)尽量减少故障的发生，提高系统的平均无故障时间。(2)允许故障的发生，但能够尽快实现恢复，降低系统的平均恢复时间。对式(1)做变换得到式(2)：

$$\frac{kMTTF}{kMTTF + MTTR} = \frac{MTTF}{MTTF + \frac{1}{k}MTTR} \tag{2}$$

可以看出，系统平均无故障时间提高为原来的 k 倍对系统可用性的改善与系统平均修复时间减小为原来的 $1/k$ 是等效的。另外，要从根本上避免错误的发生是不切实际的[2]。面向恢复计算(Recovery-Oriented Computing, ROC)技术致力于在故障发生后使系统尽快实现修复，降低系统 *MTTR*，从而有效提高系统可用性。本文通过研究ROC技术，提出了面向恢复的集群计算技术：基于节点组的递归重启方法以及基于Checkpoint的Undo模型，由此提高集群系统可用性。

2 递归重启技术

软件系统日趋复杂，检测和修复各种软件bug越来越困难，开销越来越大，而重启是解决这类故障快速且有效的一种方法。但较高级别的系统重启往往开销很大，而且可能造成一些硬件状态的丢失。文献[3]提出了在人造卫星地面站控制系统Mercury中应用递归重启技术实现系统部分重启，降低了系统 *MTTR*，提高了系统可用性。

2.1 重启树

递归重启技术的核心是重启树：将整个系统划分为一系列不同级别的组件，一个高级别组件的重启将导致其子树上的所有组件重启；同一级别的组件故障无关，即一个组件发生故障或重启不会对其他组件造成影响。从重启树中并不能看出组件之间功能和状态的相关性，而只能得出它们的“重启相关性”，即每个组件当它周围组件重启时所受到的影响。当然，功能和状态相关性是组件划分的重要依据，所以，重启树间接地表示了组件之间的功能和状态相关性。

重启树中的组件包括重启节点和叶节点(通常为系统中划分的软件组件)。图1为包含了3个软件组件(*A*,*B*,*C*)的重启树，它有5个重启节点：R_{ABC}，R_A，R_{BC}，R_B，R_C，每个重启节点称为一个重启单元。重启树中的每个子树构成一个重启组，图1的重启树包含了5个重启组，分别是 R_A，R_B，R_C，(R_{BC}，R_B，R_C)，(R_{ABC}，R_A，R_{BC}，R_B，R_C)。

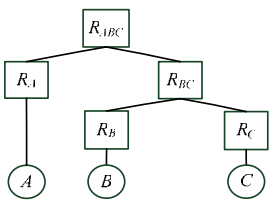

图1 重启树

作者简介：鲁晓佩(1985－)，男，硕士研究生，主研方向：高性能计算，容错技术；廖湘科，研究员、博士生导师；卢宇彤，研究员

收稿日期：2008-12-01 **E-mail：**luxp02@163.com

2.2 恢复器和决策器

在递归重启系统中，系统恢复器负责对重启树中需要重启的组件进行重启操作，而系统决策器负责制定重启规则，根据收集到的失效信息对引起故障的组件进行判断，并通知系统恢复器对需要重启的组件执行重启操作。重启一定数量的组件后，若故障排除，则系统恢复正常工作；若问题依然存在，则决策器通知恢复器对更高一级的重启组乃至整个系统进行重启操作，直至故障排除。

3 系统级 Undo 模型

研究发现，在 Internet 等计算机服务领域，人为操作失误造成的故障占很大一部分，建立系统级的 Undo 模型可以很好地解决这类失效，实现对用户数据的保护，提高系统可用性。

3.1 Undo 模型

Undo 模型的主要思想是：在系统正常运行时，通过一定的方式记录系统的历史状态，当故障发生时，将系统恢复到之前保存的某一时刻状态。评价 Undo 模型有 2 个关键方面：系统历史记录采用的方式，系统能实现的 Undo 操作类型。

系统记录操作历史可以通过逻辑或物理的方式：

(1)逻辑方式：系统通过记录操作指令表示历史状态的改变，并通过调用历史指令的反运算或改变初始指令使其具有反向作用实现 Undo 操作。逻辑方式的实现需要满足以下假设：每条历史指令都要有反向指令；每条指令都可以准确无误地执行。

(2)物理方式：系统将历史状态保存为一系列映像文件。映像文件的保存有 2 种方式：保存系统每个检查点状态的完整备份；保存最近一次检查点之后的增量状态改变。物理方式只能精确地恢复所保存的状态，不能够实现可改变的 Redo 操作，比如 Undo 操作之后需要对系统的状态进行修改或者插入一些新的指令时，物理方式实现的恢复没有意义。

由于逻辑或物理的方式都存在一定局限性，因此可以结合使用 2 种方式，并按照具体应用的需求选择合适的方式。

评价 Undo 模型的第 2 个方面即 Undo 模型能否实现选择性或非线性的 Undo 操作。选择性的 Undo 操作要求能够插入、删除、修改系统历史上的任何一项操作。非线性的 Undo 模型以类似树形结构记录系统操作历史的不同版本，可以根据需要选择不同的撤销路线，从而增强 Undo 操作的功能。

3.2 Three-R's Undo 模型

Undo 模型一个典型的应用是 Three-R's Undo 模型：回转(rewind)，修正(repair)，回放(replay)。在回转阶段，通过回滚操作把系统恢复到故障发生前保存的某时刻检查点状态；在修正阶段，由操作员对系统状态进行必要的改动，避免故障的再次发生；在回放阶段，修正后的系统重新执行系统日志中记录的用户操作。图 2 为 Three-R's Undo 模型结构。

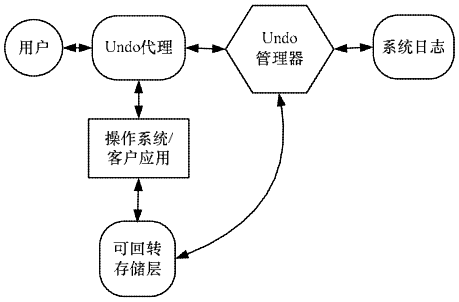

图 2 Three-R's Undo 模型结构

该模型保持原有服务应用及操作系统不变，增加了一些新的功能模块：其核心是 Undo 管理器，负责协调管理 Undo 代理层和可回转存储层，并维护用户与系统交互的历史记录及系统日志；Undo 代理层位于服务应用和用户之间，负责捕获并记录用户请求，并在需要时插入自己的指令实现回放等操作；可回转存储层负责存储相关的系统状态记录文件。文献[4]在 SPECmail 邮件系统中实现了 Three-R's Undo 模型，获得了良好的恢复性能。

4 集群系统中的 ROC 设计

以上 ROC 相关技术局限于单机系统中的应用，本文将 ROC 理论应用在集群系统中，致力于提高集群系统可用性。

4.1 集群系统中基于节点组的递归重启方法

集群系统由于长时间的运行且需要频繁地进行资源分配与调度，累积的系统误差容易造成系统异常甚至崩溃。而检测和修复这类故障通常十分困难，一般需要重启系统。为了缩短重启恢复时间，本文设计了基于节点组的递归重启方法。

为了实现细粒度的分级重启，对集群系统进行结构层次划分：先对作业的数据结构进行细化，将一个作业的所有进程划分为多个进程组；在进行作业资源分配时，根据作业的资源需求分配若干个计算节点给作业，作为一个节点组；在同一个节点组内，分别分配若干个节点给作业的每个进程组，作为一个节点域；在单个计算节点上，对操作系统和应用软件进行更细粒度软件组件划分。通过以上方法使各个节点组、节点域、软件组件之间尽量实现良好的故障隔离。节点组具有以下 2 个特性：(1)动态性：节点组的划分是根据作业的需求动态变化的，一旦作业执行结束，则释放它所在节点组内的节点，并根据新的作业重新进行分配。(2)独占性：一旦节点被分配给某作业作为一个节点组，则只有该作业能够使用该节点，而不能再分配给其他作业，直至该作业执行结束。

图 3 为应用了基于节点组的递归重启方法的集群系统结构。

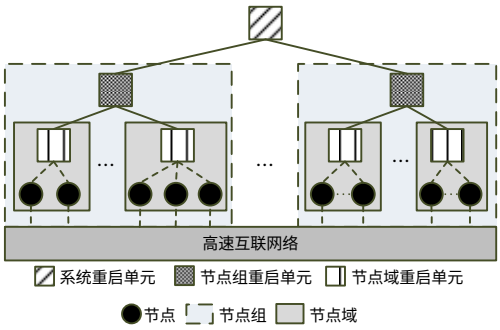

图 3 基于节点组的递归重启集群系统结构

当集群系统发生故障时，对故障的类型进行分析判断，选择重启的级别。主要的故障类型及处理方式有：

(1)单个节点故障：单个节点内发生故障后，对发生故障的软件组件进行重启，若故障无法排除，则重启整个节点。

(2)并行作业故障：并行作业运行中，运行在某节点作业进程发生故障，在不影响其他节点上的作业进程及并行作业全局一致性的前提下，单独对故障节点进行重启；否则，对故障节点所在的节点域乃至节点组进行重启。

(3)底层网络故障：高速互联网络负责节点间及与外界数据的通信。发生故障时，对故障的类型进行判断，若是通信软件故障，则对故障程序进行重启；若是网络接口硬件故障，则对故障部件进行重启。

(4)整个集群系统：出于系统维护目的、极少的灾难性故障或者其他原因，对整个系统进行重启。

4.2 集群系统中基于 Checkpoint 的 Undo 模型

目前，集群系统及应用的规模越来越大，作业执行时间较长，单个节点发生故障可能导致整个计算过程彻底失败，使已完成的计算白白浪费。本文建立基于 Checkpoint 的 Undo 模型，使系统从故障中恢复后能够从之前保存的某时刻检查点状态继续执行，有效地减少了重复计算时间。图 4 给出实现此机制的 4 个主要功能模块：

(1)中央控制模块：负责协调管理检查点设置与恢复、故障检测等工作。

(2)故障检测模块：周期性地检测各节点状态，发现故障后及时向中央控制模块报告，由中央控制模块进行分析并做出相应的决策和处理。

(3)检查点设置与恢复模块：定期进行检查点设置，保存系统当前一致性状态，并负责在系统发生故障时进行检查点恢复操作。

(4)检查点映像文件管理模块：将检查点映像文件保存在共享存储空间并进行管理。存储设备要有必要的冗余备份。

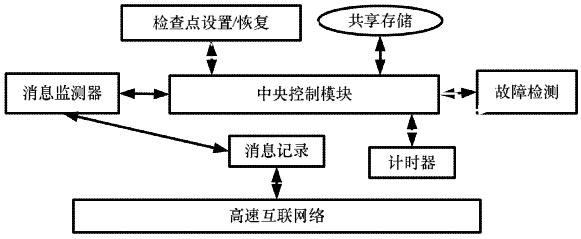

图 4 集群系统中基于 Checkpoint 的 Undo 模型逻辑结构

在系统正常运行时，由检查点设置与恢复模块定期进行检查点设置，将从中央控制模块获得的系统一致性状态保存为一系列检查点映像文件，并将各个作业所包含的进程、所在节点、检查点映像文件路径等信息保存为一个检查点控制文件。故障检测模块采用定时巡检的方式运行故障检测程序，周期性地对系统中节点的软硬件工作状态进行巡检，得到各节点的状态报告。巡检的结果作为本节点状态转换的判定条件报告给中央控制模块，并作为资源分配的依据。在以下 2 种情况下，由检查点设置与恢复模块进行检查点恢复操作：某节点通过重启或其他方式从故障中恢复，需要恢复之前保存的作业状态并继续执行；某计算节点发生永久故障，需要将运行在此节点上的作业迁移到其他节点继续执行。检查点恢复的方式有自动和手动 2 种：

(1)自动恢复：无需系统管理员干预，检查点设置与恢复模块读取检查点控制文件，获得检查点映像文件路径并在需要恢复的节点上自动加载，执行恢复操作。

(2)手动恢复：管理员通过命令行的方式手动执行检查点设置/恢复操作，以满足某些非故障条件下的检查点恢复操作需求。

在整个系统执行过程中，对通道中的消息进行监测及记录，保存系统检查点设置和恢复的历史记录。

5 ROC 性能分析

本文通过在集群系统中应用基于节点组的递归重启方法以及基于 Checkpoint 的 Undo 模型，降低集群系统的故障恢复时间，减少重复计算造成的资源浪费。下面分别就 2 种方法给系统带来的性能改善进行分析。

5.1 基于节点组的递归重启方法性能分析

假设集群中有一个 8 个节点的节点组，其中，节点 3、节点 4 之间、节点 6~节点 8 之间分别存在功能和状态相关性。图 5 中的左图为应用递归重启前的系统重启树：节点 3、节点 4 之间、节点 6~节点 8 之间由于无法准确判断发生故障的节点需要进行串行重启，恢复时间较长；右图为应用递归重启后的情况。

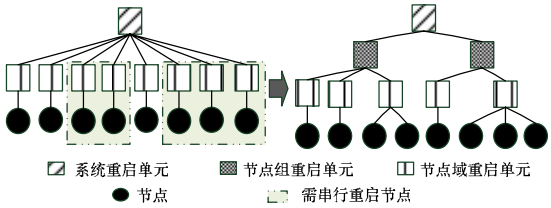

图 5 应用递归重启前后的重启树

式(3)用于计算系统 $MTTR$，其中，m_i 表示一段时间内节点 i 发生故障的次数；$MTTR_i$ 表示节点 i 的平均故障恢复时间：

$$MTTR = \sum_{n=1}^{n}(m_i \cdot MTTR_i) \Big/ \sum_{n=1}^{n} m_i \tag{3}$$

若系统为异构的，节点 1~节点 8 的平均无故障时间、单个节点重启所需要的时间 T_i 以及应用递归重启前后各个节点及整体的 $MTTR$ 如表 1 所示。

表 1 应用递归重启后异构系统性能比较

	节点 1	节点 2	节点 3	节点 4	节点 5	节点 6	节点 7	节点 8	整体
$MTTF$/天	30	5	10	15	2	30	20	15	30
T_i/s	60	15	20	30	20	30	35	40	60
$MTTR_{应用前}$/s	60	15	50	50	20	105	105	105	37.7
$MTTR_{应用后}$/s	60	15	30	30	20	40	40	40	24.8

若系统为同构的，节点 1~节点 8 的平均无故障时间、单个节点重启所需要的时间 T_i 分别为 10 天、30 s，应用递归重启前后各个节点及整体的 $MTTR$ 如表 2 所示。

表 2 应用递归重启后同构系统性能比较

	节点 1	节点 2	节点 3	节点 4	节点 5	节点 6	节点 7	节点 8	整体
$MTTF$/天	10	10	10	10	10	10	10	10	10
T_i/s	30	30	30	30	30	30	30	30	30
$MTTR_{应用前}$/s	30	30	60	60	30	90	90	90	60
$MTTR_{应用后}$/s	30	30	30	30	30	30	30	30	30

可以看出，无论是异构还是同构集群系统，应用递归重启技术后，整个节点组的 $MTTR$ 都明显降低，从而有效提高了系统的可用性。在节点数量更多、包含更多节点组的大型集群系统中，改善将更加显著。

5.2 集群系统中基于 Checkpoint 的 Undo 模型的性能分析

集群系统及其应用的规模日趋庞大，系统故障率越来越高，各种科学计算需要的执行时间越来越长。故障率提高到一定值后，作业发生 2 次故障的时间间隔将小于作业的有效执行时间，此时若不使用检查点，作业将无法执行完。本文在集群系统中应用基于 Checkpoint 的 Undo 模型，如图 6 所示：假设故障发生时刻 t_f 为作业 1~作业 5 分别所处的执行状态，t_{c1}, t_{c2} 为 2 次执行检查点设置的时间。故障发生后，处于不同执行阶段的作业需要的处理方式如下：

(1)作业 1 在第 1 次检查点设置之前已经执行完毕，无需备份和恢复。

(2)作业 2 需恢复至 t_{c1} 时刻的检查点状态，故障造成的重复计算时间为 t_f-t_{c1}。若没有进行检查点设置，则需要重新提交作业 2，重复计算时间为 t_f-t_2，通过设置检查点节约了重复计算时间 t_{c1}-t_2。 (下转第 57 页)