ISSN1000-9825　　Journal of Software　软件学报　　2000,11(2):235～239

基于故障敏感图的回卷算法和故障恢复*

柳颖[1]　陈道蓄[1]　谢立[1]　曹建农[2]

[1](南京大学计算机软件新技术国家重点实验室　南京　210093)

[2](香港理工大学计算系　香港)

E-mail: yliu@dislab.nju.edu.cn

摘要　扩充的面向图结构的分布式程序设计模型(extended graph-oriented model,简称ExGOM)提供了一个支持动态配置的系统框架.系统的动态配置包括系统运行时的伸缩、运行时的升级以及出现故障后的重配置.故障后的重配置所涉及的问题之一是如何恢复系统原状态,该文着重就此问题进行了讨论,给出了基于故障敏感图的异步检查点回卷算法和故障恢复策略.该算法和策略考虑了在暂时性主机故障中单个主机上有多个故障进程的情况.与其他异步回卷及故障恢复算法相比,该算法将故障区域局部化,仅对故障敏感节点进行回卷,从而有效地降低了系统开销.

关键词　分布式程序设计,检查点,回卷,故障恢复.

中图法分类号　TP311

扩充的面向图结构的分布式程序设计模型(extended graph-oriented model,简称ExGOM)提供了一个支持动态配置的系统框架.它支持多种动态配置,包括系统运行时任务图的伸缩、系统运行时系统模块的替换与升级以及系统运行过程中由于若干主机结点故障或其他原因(如负载、效率等)所导致的各进程的重新映射.对于第3种情形,无论是迁出进程,还是在主机正常后恢复进程运行,都需要适当地保存进程的状态,用以故障恢复,这种方法也被称为“检查点”方法.目前,国内外都有大量关于检查点算法的研究[1～4],其中大致可分为同步方法与异步方法两类.在同步方法下,所有进程协调其检查点动作以获得一个全局一致的状态.在异步方法下,每个进程各自独立地进行检查点动作,故障恢复时进程需协作回卷至一个全局一致状态.采用消息传递模型的分布式系统,各进程除需记下进程状态外,还需记下其传递的消息.本文给出的回卷算法和故障恢复采用异步检查点方式.

1　系统模型

为了更好地说明问题,本节将简要介绍回卷和故障算法运行其上的ExGOM模型.

ExGOM下的分布式应用程序由3部分组成[5]:树型逻辑结构图、若干应用程序以及从应用程序到图中节点再到主机的映射关系.用户在该模型下编写分布式应用程序时首先定义一个树型逻辑结构图,该逻辑图结构反映了该应用下进程间的初始静态关系,用户也可以通过虚拟节点或边定义潜在的进程关系.实节点意味着分配其上的进程一旦启动该应用即被生成并运行;虚节点上的进程则由其他进程动态创建.如果一条边的两个端点均为实节点,则该边为实边,否则称为虚边.可见,虚边表明了一个进程与一个动态创建的进程间的潜在的通信关系.然后用户在定义的图结构之上,利用图上提供的多种操作编写程序.逻辑图结构及映射关系都在该应用

---

* 本文研究得到国家863高科技项目基金(No.863-306-ZT02-03-01)和香港理工大学研究基金资助.作者**柳颖**,女,1973年生,博士,主要研究领域为分布式系统,并行计算,容错计算.**陈道蓄**,1947年生,教授,主要研究领域为分布式系统,并行计算,计算机网络.**谢立**,1942年生,教授,博士生导师,主要研究领域为并行计算与分布式处理.**曹建农**,1960年生,博士,助教,主要研究领域为分布式系统,容错计算.

本文通讯联系人:柳颖,南京210093,南京大学计算机软件新技术国家重点实验室

本文1998-09-15收到原稿,1999-02-02收到修改稿

© 中国科学院软件研究所　http://www.jos.org.cn

的配置文件中定义. 当执行一个 ExGOM 应用时,运行系统读取配置信息,构造动态执行图 DEG(dynamic execution graph),根据映射关系创建各进程并启动运行. 此后由运行系统负责维护动态执行图.

每个主机有一个运行系统的核(kernel). 每个核都维护着一个 DEG. 正常情况下,各 DEG 是一致的. 运行系统的核结构如图 1 所示. 其中 SLG(static logical graph)是用户定义的初始逻辑图结构,FSG(fault-sensitive graph)是当故障发生后从 DEG 得到的故障敏感图.

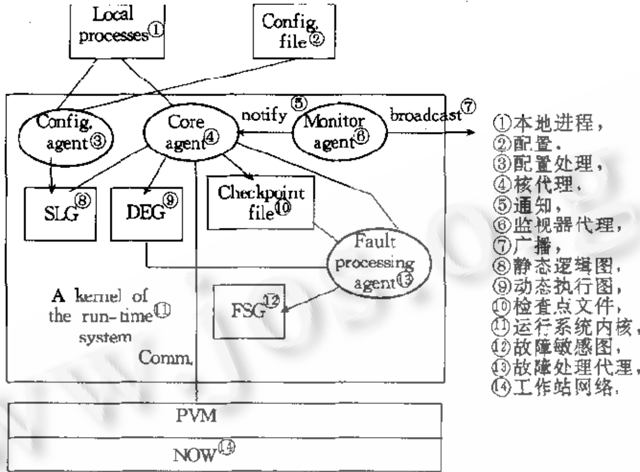

Fig. 1 Core structure of run-time system
图1 运行系统的核结构

2 故障敏感图

当系统监控器监测到某台主机故障时,就通知本地的运行系统,运行系统于是检查映射表,找出运行在该主机上的所有进程,在 DEG 上标注出来. 图 2 给出了一个标有故障进程的 DEG.

FSG 是一个树型结构,它的顶点是某一个故障节点(当某一个节点上的进程为故障进程时,该节点就是故障节点). 故障节点在 DEG 中的父节点和孩子节点都被添加进 FSG,FSG 中各节点间的边也都被加进 FSG. 假定每个 FSG 片段至多有两个故障节点(这可在定义配置文件时做到). 由图 2 可得到图 3 所示的两个 FSG 片段,从图 3 我们也可以看到多个 FSG 中的节点有可能相互交叠.

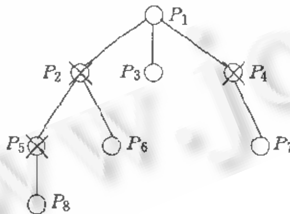

Fig. 2 DEG with fault processes tagged
图2 标有故障进程的DEG

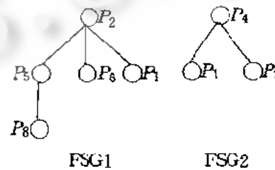

Fig. 3 FSG derived from DEG
图3 由DEG获得的FSG

采用 FSG 是因为我们考虑到当一个分布式应用中的某一进程发生故障后,其他相对独立于该进程的进程应可以继续执行下去而无需真正回卷,这也是本文稍后给出的回卷算法和故障恢复的基本思想. 运行系统收到监控器发来的故障通知消息后即挂起本地进程,将其状态设为"ABNORMAL". 此后故障主机将重启,其上的运行系统也将重启. 如果某本地进程正在发送消息,则在其发送动作完毕后被挂起. 所发送的消息由本地运行系统保存在一个队列里,称为 TEM-QUEUE. 此外,由于各运行系统非同时终止正常操作而转入故障恢复操作,一个运行系统有可能收到其他运行系统转发的消息,这些消息也需保存. 这样,非故障敏感的进程一旦收到"NORMAL"的消息后无需真正回卷就可继续执行. 详细过程将在后两节给出.

© 中国科学院软件研究所 http://www.jos.org.cn

3 回卷算法

3.1 定 义

对采用消息传递模型的分布式应用来说,应用可看成是由事件驱动的,即一进程每接收到一条消息就从一种状态转换到另一种状态.文献[1]中将接收消息定义为事件,$e_{i,j}$表示进程 P_i 的第 j 个事件,$s_{i,j}$表示事件 $e_{i,j}$后的状态,$SENT_{i,j}(e)$表示到事件 e 发生为止进程 P_i 发送到进程 P_j 的消息总数,$RECD_{i,j}(e)$表示到事件 e 发生为止进程 P_i 接收到来自 P_j 的消息总数.运行系统为每一个本地进程保存一系列 $SENT$ 与 $RECD$ 参数对.每发送或接收一条消息,相应的参数增加 1,并且与消息一起保存到稳定的存储介质(如硬盘、磁带等)上.这一方法增加了系统空间上的开销.相对于较为廉价的存储介质而言,我们认为这一方法是可行的,但在今后的工作中我们仍将研究如何有效地降低系统的这一开销的问题.

故障发生后进程所处的状态各不一样.回卷算法的目的即是要找到一个全局的一致状态.我们定义一个全局的一致状态是这样的一个事件集合,每个进程在该集合中有一个事件相对应,任意两个事件 e_i 与 e_j 之间满足 $SENT_{i,j}(e_i)\geqslant RECD_{j,i}(e_j)$以及 $SENT_{j,i}(e_j)\geqslant RECD_{i,j}(e_i)$.这里,要求消息传递是有序的.

3.2 回卷算法

回卷算法由发生短暂故障后重启的运行系统启动.该运行系统首先从其他正常的运行系统处获得逻辑图结构的拷贝,然后通过系统记录的动态信息得到 DEG,最后由 DEG 得到若干 FSG.下面,我们分别给出发生故障的运行系统上执行的回卷算法(算法 1)和正常的运行系统上执行的回卷算法(算法 2).

算法 1.

```
while 尚有 FSG 未处理 loop
  rb_i ← FSG 根节点上故障进程 P_i 的最近事件;
  if FSG 中另有一个故障进程 P_k then
    rb_k ← P_k 的最近事件;
    if SENT_{k,i}(rb_k) < RECD_{i,k}(rb_i) then
      P_i 回卷直至事件 e',使得 SENT_{k,i}(rb_k) ≥ RECD_{i,k}(e');
      rb_i ← e';
    else if SENT_{i,k}(rb_i) < RECD_{k,i}(rb_k) then
      P_k 回卷直至事件 e',使得 SENT_{i,k}(rb_i) ≥ RECD_{k,i}(e');
      rb_k ← e';
  广度遍历树 FSG:
    if 所遍历的进程 P_j 是一个正常进程 then
        设 P_i 为 P_j 所在节点的父节点上的进程;
      if P_i 为故障进程且其回卷状态为 rb_i then
          向 P_j 运行其上的运行系统发送 UPDATE(SENT_{i,j}(rb_i), P_j);
end loop
```

算法2.

```
while 接收到消息 loop
  if 该消息为 UPDATE(SENT_{i,j}(rb_i), P_j)消息 then
    从稳定存储介质中得到 P_j 的最近事件赋给 rb_j;
    if SENT_{i,j}(rb_i) < RECD_{j,i}(rb_j) then
        P_j 回卷直至事件 e',使得 SENT_{i,j}(rb_i) ≥ RECD_{j,i}(e');
        rb_j ← e';
end loop
```

设 e_i 为进程 P_i 的最近事件,e_j 为进程 P_j 的最近事件.可证明 $SENT_{i,j}(e_i)<RECD_{j,i}(e_j)$与 $SENT_{j,i}(e_j)<RECD_{i,j}(e_i)$不可能同时成立.证明如下:

证明:为简便起见,我们分别将 $SENT_{i,j}(e_i)$与 $RECD_{j,i}(e_j)$简写为 $SENT(e_i)$与 $RECD(e_j)$.假设 $SENT(e_i)<RECD(e_j)$,这就意味着在 P_i 发送消息 m 后故障发生.$SENT(e_i)$未更新,而 P_j 收到了消息 m 且 $RECD(e_j)$已

© 中国科学院软件研究所 http://www.jos.org.cn

更新.在此之前,P_i 与 P_j 的状态是一致的.如果 $SENT(e_j)<RECD(e_i)$,则意味着 P_j 在事件 e_j 发送消息 m' 但却未及更新 $SENT(e_j)$,而 P_i 接收到了消息 m' 且更新了 $RECD(e_i)$.根据我们对事件的定义及 e_i,e_j 分别为 P_i 与 P_j 的最近事件的前提条件,m' 必须在 P_j 收到 m 后发送.根据消息有序发送的假定,m' 在事件 e_i 后到达 P_i,而 P_i 在事件 e_i 后发生故障,因此,P_i 不可能收到消息 m' 并更新 $RECD(e_i)$.故 $SENT_{i,j}(e_i)<RECD_{j,i}(e_j)$ 与 $SENT_{j,i}(e_j)<RECD_{i,j}(e_i)$ 不可能同时成立.

由于其不能同时成立,算法中的单方向的调整就可以保证最终所得为一个全局一致状态.

4 故障恢复

对正常的运行系统而言,恢复工作首先要处理保存在 TEM_QUEUE 中的消息.处理过程如下(算法3):

算法3.

while TEM_QUEUE 非空 loop
　　if 消息是发往本地进程 then 启动该进程运行处理消息,随后挂起该进程;
　　if 消息是发往一故障进程 then 丢弃;
　　if 消息是发往一远程正常进程 then 进行常规处理;
　end loop
　向所有其他正常的运行系统广播 TEM_FINISH 消息;
　　if 收到所有来自其他正常运行系统的 TEM_FINISH 消息
　　then 转而执行算法4; else 等待.

故障运行系统重启后即执行回卷算法,然后从所得到的一致状态处启动其上进程运行,并设其状态标志为"*NORMAL*".算法4主要给出了恢复运行的前故障运行系统如何处理来自本地进程的接收与发送请求以及如何处理"*NORMAL*"消息.*

设 P_f 为任一原故障进程,e_f 为其回卷点;P_n 为某个 FSG 上的任一正常进程,e_n 为其回卷点,e_{nl} 为其最近事件;P_u 为任一不在任何 FSG 上的正常进程.

算法4.

if 消息是由 P_f 发往另一故障进程 then 进行常规处理;
if 消息是由 P_f 发往 P_n then
　本地运行系统对其进行常规处理,然后转交给远程的运行系统;远程运行系统首先检查其状态标志:
　　if 状态标志为 *ABNORMAL* then
　　　if $RECD(e_n)\leqslant SENT(e_f)$ then
　　　　if $e_n\neq e_{nl}$ then 将 e_n 更新为下一个事件并丢弃所收到的消息;
　　　　else 将 P_n 状态标志设为 *NORMAL*,启动其正常运行,并向所有相邻进程发 *NORMAL* 消息;
if P_f 请求由 P_f 发送的消息 then 进行常规处理;
if P_f 请求由 P_n 发送的消息 then
　if P_n 的状态标志为 *ABNORMAL* then
　　本地运行系统将请求及 $RECD(e_f)$ 转交给远程运行系统;远程运行系统接到请求后检查 $SENT(e_n)$ 与 $RECD(e_f)$;
　　　if $SENT(e_n)>RECD(e_f)$ then
　　　　从保存的目的地为 P_f 的发送消息队列中找到第 $RECD(e_f)+1$ 条消息发送给 P_f
　　　　　if 队列为空且 $e_n==e_{nl}$ then
　　　　　　将 P_n 状态标志设为 *NORMAL*,启动其正常运行,并向其邻接进程广播 *NORMAL* 消息;

* 正常进程与故障进程相对应,但其状态标志并非为"NORMAL".

© 中国科学院软件研究所　http://www.jos.org.cn

if 消息是发往某本地进程 P_k 的 *NORMAL* 消息 then
  if P_k 的状态标志为 *ABNORMAL* then
    if P_k 在某一 FSG 上 then
      将其状态标志设为 *NORMAL*,启动其正常运行,并向所有不在任一 FSG 上的邻接进程广播 *NORMAL* 消息.

通过上述算法,一个正常进程将知道故障进程何时已正常运行,并启动自身继续执行.

5 结 论

回卷算法与故障恢复目前已有大量的研究,研究者提出了各种方法[1~4,6~8].本文旨在为我们早些时候提出的扩充的支持动态配置的面向图结构的分布式程序设计模型(ExGOM)提供对暂时性主机故障的容错处理.回卷算法与故障恢复建立在故障敏感图基础之上,这使得相应算法无需对所有进程都进行回卷处理,非故障敏感的进程不必回卷,从而减少了恢复工作的开销.

暂时性主机故障的容错处理可运用到永久性主机故障的容错处理中.例如,通过冗余实现容错的系统可以及时地或是定期地将检查点信息复制到对等主机.故障发生后,由对等主机进行恢复.此时,对等主机可采用暂时性主机故障的容错处理策略.

参考文献

1 Venkatesan S, Tony Tong-ying Juang, Sridhar Alagar. Optimistic crash recovery without changing application messages. IEEE Transactions on Parallel and Distributed Systems, 1997,8(3):263~270
2 Wong K E, Franklin M. Checkpointing in distributed computing systems. Journal of Parallel and Distributed Computing, 1996,35(1):67~75
3 Sunondo Ghosh, Melhem R, Daniel Mosse. Fault-Tolerance through scheduling of aperiodic tasks in hard real-time multiprocessor systems. IEEE Transactions on Parallel and Distributed Systems, 1997,8(3):272~284
4 Wei Xiao-hui, Ju Jiu-bin. Checkpointing algorithms in distributed systems. Chinese Journal of Computers, 1998,21(4):367~375
(魏晓辉,鞠九滨.分布式系统中的检查点算法.计算机学报,1998,21(4):367~375)
5 Liu Ying, Xie Li, Cao Jian-nong. GOM: a graph-oriented model for distributed programming. Chinese Journal of Computers, 1998,21(1):18~25
(柳颖,谢立,曹建农.面向图结构的分布式程序设计模型GOM.计算机学报,1998,21(1):18~25)
6 Pankaj Jalote. Fault Tolerance in Distributed Systems. Englwood Cliffs, NJ: PTR Prentice Hall, Inc., 1994
7 Smith S W, Johnson D B. Minimizing timestamp size for completely asynchronous optimistic recovery with minimal rollback. In: Proceedings of the 15th IEEE Symposium on Reliable Distributed Systems. 1996. 66~75
8 Plank J S. Improving the performance of coordinated checkpointers on networks of workstations using RAID techniques. In: Proceedings of the 15th IEEE Symposium on Reliable Distributed Systems. 1996. 76~85

Rollback Algorithm and Crash Recovery Based on Fault-Sensitive Graphs

LIU Ying[1] CHEN Dao-xu[1] XIE Li[1] CAO Jian-nong[2]

[1](*State Key Laboratory for Novel Software Technology Nanjing University Nanjing* 210093)
[2](*Department of Computing Hong Kong Polytechnic University Hong Kong*)

Abstract Extended graph-oriented distributed programming model (ExGOM) provides a system architecture to support dynamic configuration. Dynamic configuration involves system expansion and shrink during execution, upgrading while running, and reconfiguration after a fault occurs. One problem in reconfiguration is how to recover the system to the consistent states that exist just before the occurrence of faults. This paper is focused on this problem and proposes an asynchronous rollback algorithm and a crash recovery mechanism based on fault-sensitive graphs. The issue of multiple faulty processes on a single transient faulty host is addressed. Compared with other asynchronous rollback and recovery algorithms, the algorithm presented in this paper localizes the region of faults. Only fault-sensitive nodes are rolled back. This results in a minimized system overhead.

Key words Distributed programming, checkpoint, rollback, crash recovery.

© 中国科学院软件研究所 http://www.jos.org.cn