第 33 卷 第 6 期 计 算 机 工 程 2007 年 3 月
Vol.33 No.6 Computer Engineering March 2007
·软件技术与数据库· 文章编号：1000—3428(2007)06—0083—03 文献标识码：A 中图分类号：TP311

一种基于中间件的容错系统的研究与设计

姚 兰[1]，桂 勋[2]，巨军让[1]
（1. 成都信息工程学院控制工程系，成都 610225；2. 西南交通大学电气工程学院，成都 610031）

摘 要：随着硬件容错技术的成熟，软件容错技术成为提高系统可靠性的热点问题。直接开发容错应用是非常困难的，鉴于中间件为应用系统提供了良好的开发环境，该文研究和设计了一个基于中间件的容错系统模型，提出了一种新的节点容错结构构造方法，为解决冗余、失效检测和恢复等容错的关键技术问题形成了一套较完整的解决方案。采用马尔科夫过程求出系统的可靠度，验证了系统设计的合理性和可靠性。

关键词：中间件；容错技术；失效检测；可靠性

Research and Design of Fault Tolerant System Based on Middleware

YAO Lan [1], GUI Xun [2], JU Junrang [1]
(1. Department of Control Engineering, Chengdu University of Information Technology, Chengdu 610225;
2. School of Electrical Engineering, Southwest Jiaotong University, Chengdu 610031)

【Abstract】 With the rapid development and mature of hardware techniques, software fault tolerance become the main reason of system dependability. However, fault tolerance system is hard to be developed directly. As well all know the development of system greatly benefits from the middleware for its convenient development environment and transparent communication mechanisms. So the paper puts forward a new kind of software fault tolerant structure based on middleware, and forms a complete solution project including some key techniques of fault tolerance, such as redundancy, fault detection and recovery, finally makes out its reliable degree which can verify the feasibility and reliability of this system.

【Key words】 Middleware; Fault tolerant technique; Fault detection; Reliability

1 软件容错现状分析

随着硬件容错技术的成熟，软件错误成为影响系统可靠性的最主要因素[1]。过去 20 年中，在软件容错方面进行了可观的大量研究，软件错误主要采用恢复块、N版本程序设计和一致性恢复块 3 种典型的软件容错技术。纵观这些典型软件容错技术，都是在集中式模式下基于进程研究如何增强系统可靠性和可用性，存在容错粒度过粗和进程检查点开销过大等问题，并且也不能满足分布式应用系统的容错需求[2～4]。因此，修改存在的容错机制或开发新容错机制势在必行。

但是直接开发容错应用是非常困难的，因为开发者不仅要处理复杂的应用逻辑，还要面对纷繁的容错逻辑。鉴于中间件为应用系统提供了良好的开发环境和多种通信方式，基于中间件设计一种新的系统模型和软件容错结构，不仅尽可能多地屏蔽容错实现和管理细节，也屏蔽了底层的通信细节，使设计人员能够专注于业务逻辑开发，从软件工程角度来说是一个很自然的想法。本文基于中间件研究和设计了一种新的软件容错结构，提出了一种新的节点容错结构构造方法，在系统层和任务层给出了切实可行的容错策略。

2 基于中间件的容错系统模型

2.1 相关概念

(1)任务

应用程序可认为是任务的集合。在应用设计阶段，首先合理地把应用程序分解成若干个任务，再把各个任务封装成周期性任务。充分利用任务周期性的特点，就不需要采用其它措施实现多机间任务的同步管理。也就是说，不同主机的一个任务的副本从相同初始状态开始，按同样的调度顺序接收同样的消息序列，则它们将抵达相同的状态。任务的基本周期如图 1 所示。

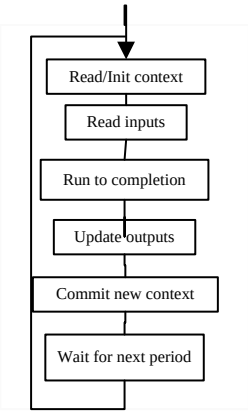

图 1 任务的基本周期

每个任务都有相关的 context，它是本周期内变化的静态存储变量，通过 context 与下一个周期任务进行信息交互。context 每次任务执行完成后被发布给任务组中的其它成员，以维护任务组成员的一致性。容错应用设计阶段，设计人员在分解和设计任务时，需要合理地确定 context 数据结构。

(2)任务组

资源冗余是实现容错的基本手段，本文采用任务冗余的

作者简介：姚 兰(1980－)，女，硕士、讲师，主研方向：测控技术；桂 勋，博士；巨军让，教授
收稿日期：2006-03-24 **E-mail：**dancyyao@cuit.edu.cn

方法实现容错。实现资源冗余的模式有主动复制、热被动复制和冷被动复制，鉴于热被动复制有资源消耗少和恢复过程快的特点[5]，本文采用热被动复制模式实现任务复制。

互为冗余的任务互称副本，一个任务的不同副本分布在系统中不同主机，其中由一个 primary 和多个 backup 构成一个任务组，组中的每个 backup 被称作组的成员。一个任务组被用户当作独立的逻辑实体看待。组中任务的状态完整性必须同步，并且必须在持久存储介质上设置检查点，当 primary 失效时，backup 可以取而代之，保证系统能够正确运行。

每个任务的冗余参数包括：任务备份数目(最小备份数目和最大备份数目)，backup 所在的主机位置。

根据这些信息，每个任务组都会有一组相关的容错属性，包括：primary 的位置和状态，backup 的位置和状态，primary 执行需要的 context(来自上个周期)，primary 的临时信息(运行周期起始和结束的时间)。

2.2 容错系统模型

应用程序可以认为是由多个相互交换消息的任务构成，将分解后的任务指派到系统中的各个主机，根据任务划分给定的任务间的数据约束关系，合理地安排好每个主机上任务的执行顺序，由各个主机上相互独立的任务协同运行，共同完成应用程序设定的功能。应用程序中有些任务是关键任务，需要对这些关键任务进行备份容错。同一个任务不同主机上的备份以一个组的形式进行管理。

如图 2 是系统模型，其中：

(1)初始化：应用程序的配置信息；

(2)初始化：节点本地任务组的配置信息；

(3)主任务发送 context 到各个任务组成员；

(4)周期性生存信号的相互监测。

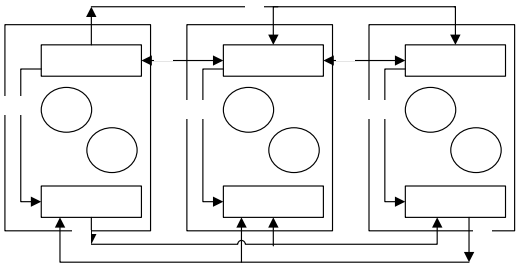

图 2 系统模型

在该模型中，包含 3 个节点：N1, N2, N3。两个任务：T1 和 T2。T1 的 primary 在节点 N1(T1p)，T2 的 primary 在节点 N3(T2p)，T1b 和 T2b 分别为 backup。节点 N1 首先启动，负责系统初始化，把系统配置信息发送到网络中的所有节点。系统配置信息对于整个系统正常运行来说至关重要的。初始化完成后，各个节点上采用循环调度算法开始执行各个任务。

应用程序正常运行时，冗余模块周期性发送生存信号监测节点状态。主任务节点的复制模块负责发布任务的 context，备份任务节点的备份模块负责订购相关的 context，并且要周期地设置检查点。备份任务节点如果不能在时限接收到 context，就认为发生了主任务节点失效。

一旦发现故障，冗余模块检查故障节点的任务信息，更新相关数据，选举出一个新的主任务代替故障节点的原主任务，并动态进行系统配置，更新复制模块的信息，由复制模块进行真正任务主备切换，刷新 context 信息表。如图 3 所示，T1 正常运行结束后，节点 N2 节点通过 context 接收情况，或生存信号检测到节点 N3 故障，则节点 N2 的 T2 由 backup 切换到 primary，系统中就只剩下节点 N1 和节点 N2 协同完成应用程序。

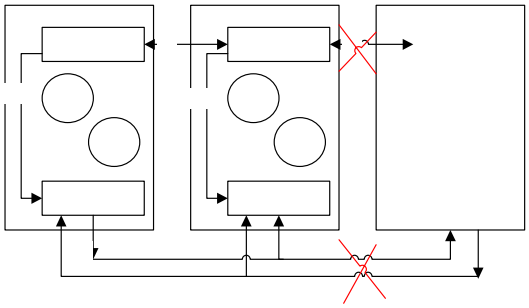

图 3 系统故障

3 节点容错结构

本文设计结构中，应用程序的任务并不进行直接交互，而是通过容错模块的控制任务进行数据交换，任务通过不同的通信路径与控制任务交换信息[6]。

节点容错结构如图 4 所示。应用程序启动时，系统首先创建一个控制任务 appliControl，由 appliControl 统一管理系统中的任务，负责各个任务的创建并封装为周期性任务，同时建立各个任务的通信机制，并建立一个应用程序信息表(appli_tab)和一个任务信息表(tasks_tab)，appli_tab 记录应用程序的基本信息，如系统中节点数目、节点 ID 和任务数目等信息，tasks_tab 记录任务的基本信息，包括任务的备份数目和任务所在主机的位置等信息，并把这些表格信息通过中间件发送给容错模块。

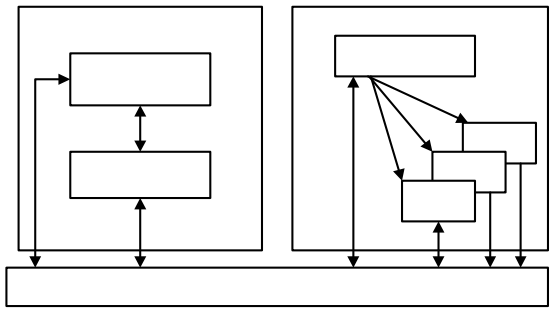

图 4 节点容错结构

任务被创建后可能处于以下两个状态之一：活跃或休眠。任务一般处于休眠状态，当调度时间到，控制任务就会激活该任务，该任务取回执行需要的数据，从而由休眠状态转为活跃状态处理数据，遵照应用逻辑执行适当的流程控制。

根据从中间件收到的信息，冗余模块建立任务组信息表(tasks_groups_tab)，并发送给复制模块。复制模块对各个任务组成员进行真正的管理，同时创建成员的状态信息表，如 task_context_tab、task_control_tab 和 task_shared_data_tab，其中 task_context_tab 周期地记录各个任务运行产生的 context，task_control_tab 记录各个任务运行的状态信息。

正常时，主任务在时限内提交 context，由复制模块通过中间件以可靠的组播方式把 context 发送给任务组中的其它成员，同时，还要设置检查点，把当前的 context 保存在 task_context_tab，以便节点失效时能通过检查点恢复故障前的状态。如果任务运行时限到了，备份任务节点的复制模块还未接收到 context，就通知冗余模块进行节点故障检测。冗余模块确定是节点故障，就把本节点的 backup 切换到

primary，新 primary 从最近的一个检查点恢复原节点失效前的状态，同时进行系统的动态配置，把系统重配置信息发送给网络中的各个节点，如果确定不是节点故障，就由复制模块采取默认的操作。

该容错结构实现了节点层和任务层的双重容错，提高了容错性，增强了可靠度，以下是容错策略。

(1)任务层容错

如果复制管理模块监测到某个任务发生超时，即 primary 没有按时发送 context，就通知冗余模块进行确认，进一步确认后就在其任务组中选举一个 backup 代替 primary，并通过信息表 task_context_tab，从最近的一个检查点恢复 primary 失效前的状态，以新 primary 方式进行工作。

(2)节点层容错

冗余模块周期地发送生存信号到其它节点。如果不能按时接收到生存信号，就认为节点故障，从节点顺序表中选举出 backup 节点替代。如果找不到 backup 来替代，就采取默认的方式结束整个应用程序的操作。

4 主要功能模块

4.1 冗余模块

冗余模块具体内容有应用程序配置的初始化、系统节点的监测(利用生存信号进行节点故障监测)和节点故障时实施应用程序的动态配置。

从图 5 可见，该模块由一个控制任务和 3 个特殊任务组成，控制任务负责应用程序周期任务的控制，特殊任务负责生存信号、节点故障监测和消息发布，并且维护 4 个信息表：nodes_tab，appli_tab，tasks_tab 和 tasks_group_tab。nodes_tab 包含冗余管理节点处理的所有信息，如节点静态数据结构和节点状态控制数据，在应用程序启动之前就创建完毕，随后启动生存信号监测和消息发布管理。应用程序启动时，创建 appli_tab，它包含系统配置数据(节点数量及其命名)和与任务相关的数据(任务数量和任务描述)。根据以上这些信息，开始创建任务描述和任务组信息表，称为 tasks_tab 和 tasks_group_tab，其中 tasks_tab 记录任务的基本信息，包括任务的备份数目和任务所在主机的位置等信息，tasks_group_tab 记录各个任务组的信息，比如一个任务组的备份数目，不同备份所在主机的位置和状态。

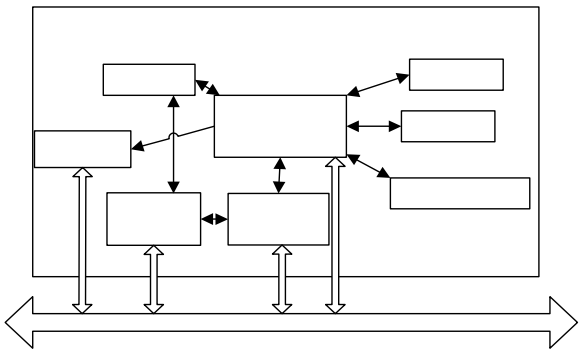

图 5 冗余模块的结构

这些信息表被传输给复制模块，由它来处理与任务组相关的信息表的初始化。处理完毕后，就通知冗余模块，冗余模块就发布 ACK 给应用程序，应用程序才开始创建执行具体的任务。

正常时，冗余模块除了周期发送生存信号，并不参于系统其它操作。当 node_crash_detector 检测到节点故障时，就发送生存信号询问消息进行故障确认，故障得到确认后，通知 notification_manager, 由它通知 appli_controller 确定新的 primary 和更新相关数据库，并把信息发布给复制模块。

4.2 复制模块

复制模块具体内容有监测运行在节点上的任务、周期地传送 context 和设置检查点、维护备份任务的状态以及维护与 context 相关的信息表。

如图 6 所示，该模块包含有若干相互协作的控制任务，一个主要的控制任务 task_group_manager 处理本节点任务的全局管理，其它特殊控制任务负责设置检查点、记录 context 等重要数据和任务故障通知的执行。同样， 它也需要维护几个用于实现任务同步的信息表，如任务组信息表 task_groups_tab 、任务控制信息表 task_control_tab 、task_contexts_tab，其中， task_groups_tab 记录每个任务组的成员信息； task_control_tab 记录当前执行任务的状态信息，如主或备状态、开始或结束、循环周期和运行时限等；task_contexts_tab 记录每个任务的 context 对象。

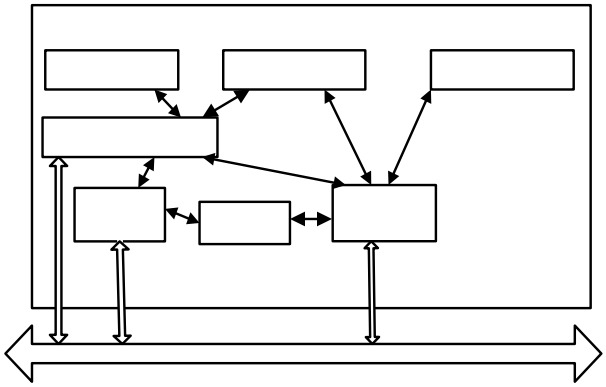

图 6 复制模块的结构

从图 6 可见，控制任务 checkpoint_manager 可对 task_control_tab 和 task_contexts_tab 进行操作，处理的信息主要是 context。一方面，它从应用程序接收数据并存在相应的信息表中，同时把数据通过中间件发布到任务组的所有成员；一方面以同样的方式周期地接收其它节点的数据，对本地信息表进行更新，以备故障时可以及时进行任务恢复。

同时，checkpoint_manager 还要负责管理任务运行的时间。如果看门狗定时器 watchdog 监测到超时，可还没有从应用程序或其它节点收到 context，通知冗余模块采用生存信号进行节点故障确认。故障确认后，复制模块通知冗余模块收集必要信息，重新选举新的主任务，复制模块这时才真正把任务从被动状态切换到主动状态。新任务的主备切换完成后，由定时器启动下一个新的周期。

5 可靠性分析

马尔科夫模型是最常用的分析容错冗余系统的工具。对马尔科夫模型进行定量分析，可以提供系统可靠性设计方案的比较和评价。

根据系统模型做如下假设，节点 N1 和 N3 正常工作，节点 N2 处于等待工作状态。节点处于正常工作状态的失效率为 λ_1，节点处于等待工作状态的失效率为 λ_2。用 $X(t)$ 来表示系统的状态，则系统有如下几种状态形式，如式(1)所示。然后，根据系统所处的状态，可得到系统的状态转移图，如图 7 所示。

(下转第 143 页)