

## 基于自适应调度的网格容错机制研究

陈欢欢, 刘宴兵

(重庆邮电大学 计算机科学与技术学院, 重庆 400065)

**摘 要:** 在研究现有网格计算容错技术的基础上, 结合蚂蚁算法的自适应优势改进现有容错技术 Migol 的任务调度策略, 构建一个效率更高的网格计算自适应容错框架. 通过仿真实验, 验证了该模型在保证系统稳定性的同时能提高系统资源利用率.

**关键词:** 容错; 自适应; 任务调度; 蚂蚁算法

**中图分类号:** TP393

**文献标识码:** A

**文章编号:** 1000-7180(2010)02-0145-04

## Research of Fault - Tolerance Grid Based on Adaptive Scheduling

CHEN Huan-huan, LIU Yan-bing

(College of Computer Science, Chongqing University of posts and telecommunications, Chongqing, 400065, China)

**Abstract:** Based on the current research of fault tolerance technology, this paper improves the scheduling strategy of Migol by utilizing the advantages of ant algorithm. The simulation results show that the model has good performance on guaranteeing the stability of system and improving utility of resource.

**Key words:** fault - tolerance; adaptive; job scheduling; ant algorithm

### 1 引言

网格计算<sup>[1]</sup>环境中资源的异构性、动态性和通信的不确定性相比传统的分布式环境和并行计算环境更为复杂. 网络传输, 资源节点发生故障都会引起节点上任务运行的失败, 导致整个网格系统的效率和服务质量的下降. 为保证任务顺利完成, 网格系统必需采取一定的错误处理机制来解决任务运行时出现的问题<sup>[2-3]</sup>, 即网格容错服务. 鉴于网格环境与以往的网络有很大的不同, 网格环境下的容错服务必须满足以下几点: 可扩展性、正确性、可用性、及时性、低负载、开放性.

目前, 对网格环境下容错机制的研究中, 容错的方法有很多种<sup>[3]</sup>, 重试、副本、替代作业, 检查点等, 可以根据不同的需求采用不同的容错方法, 但这些方法大都不够灵活, 不能更好的适应网格动态, 复杂的环境. Migol<sup>[4]</sup>是一种基于检查点技术的网格计算

容错体系, 能有效地缩短任务执行时间. 文中在继承 Migol 容错优势的基础上结合蚂蚁算法的自适应特性, 提出一种新的网格容错模型并构建出相应的网格计算自适应容错中间件. 形成效率更高的网格计算自适应容错中间件. 仿真结果表明: 该模型能更好地保证系统稳定性并减少任务执行时间.

### 2 网格框架模型

#### 2.1 容错框架

Migol 是在利用检查点技术的基础上开发的支持容错的、面向服务的、可靠的网格中间件. 为了达到系统稳定、任务成功执行的目标, Migol 提供了各种不同的 OGSA (Open Grid Service Architecture) 服务来保证运行的正确性和可靠性, 即使某个节点发生故障, 它也能根据内在的恢复模块来恢复任务的执行. Migol 的核心是 GSO (Grid Service Object), 它定义了两种模式来描述网格应用: 普通模式和可扩

展模式. 一个网格服务对象(GSO)存储了任务所有的相关信息, 其中包括: 对象的 ID、资源需求、检查点文件等. 所有正在运行的网格服务对象(GSO)都存放在 AIS(Application Information Service).

图 1 显示了 Migol 的整体框架以及各个模块之间的交互关系. 用户发送服务请求之后, 任务代理服务器接收到这个任务并把它在 AIS(Application Information Service)注册一个 GSO, 然后根据任务的需求查找所有符合要求的资源, 再按照一定的规则把任务提交到其中某个资源上. 资源的查找是通过 RMDs(Resource Monitoring and Discovery Service)实现的, 它收集了当前所有的可用资源以及资源详细信息, 比如: CPU 的个数、CPU 处理能力、网络带宽等. 任务代理服务器根据任务需求查找到符合要求的资源后通过 ARS(Advance Reservation Service)把任务提交到其中的资源上.

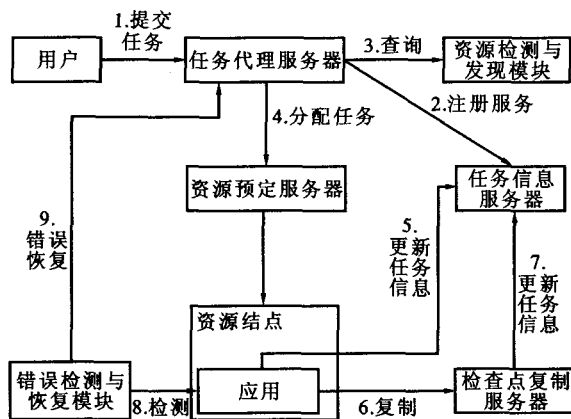


图 1 Migol 容错结构

要实现任务的容错必须做到两点: 错误检测与错误恢复. 错误检测协议的实现都是基于超时语义的, 有两大类错误检测器, 一种是基于“push”模型(也称作心跳策略), 另外一种是基于“pull”模型的. 在“push”模型中, 被检测的对象按照一定的时间间隔向错误检测器发送心跳消息, 当错误检测器在指定的时间内没有收到心跳消息, 系统认为该对象出现故障. “pull”模型(也称作“ping”策略)中被检测对象是被动的而错误检测器是主动的. 错误检测器定时发送查询消息查询对象的状态, 被检测对象收到查询消息后返回应答消息. 错误检测器超过一定时间间隔没有收到应答消息就判定被检测对象出错.

为减少通讯过程中的信号量, Migol 依靠“push”型的 MRS(Monitoring and Restart Service), AIS 中资源每隔一段时间向 MRS 发送资源信息, 一旦 MRS 发现某个节点发生异常立即反馈到任务代理

服务器, 任务代理服务器启动错误恢复机制. 对于错误恢复, Migol 应用的是作用在应用层上的检查点机制. 在系统正常运行的适当时刻设置检查点(Checkpoint), 把系统当时的运行状态保存到检查点文件中, 并对进程进行相关的跟踪和记录. 系统发生故障后, 调用检查点文件, 将相关进程回卷(Roll-back)到故障前系统一致性状态, 经过状态恢复后从该检查点处重新执行, 而不是从程序开始处执行, 从而节省了大量重复计算的时间. 检查点技术能大量节省任务恢复时间, 但检查点文件必须是正确的, 可靠的. 为了确保检查点信息在资源失效的情况下的正确性和可靠性, 系统必须通过 CRS(Checkpoint Replication Service)把检查点信息存放到稳定的存储节点上.

## 2.2 检查点复制服务

在错误的恢复过程中, 检查点<sup>[5]</sup>文件承担保存数据、恢复程序运行的重要任务, 是网格系统稳定而且高效地实现容错的关键. 如何成功有效建立检查点文件也关系到整个系统的运行效率. 图 2 显示了与 Migol 相对应的 CRS 体系结构.

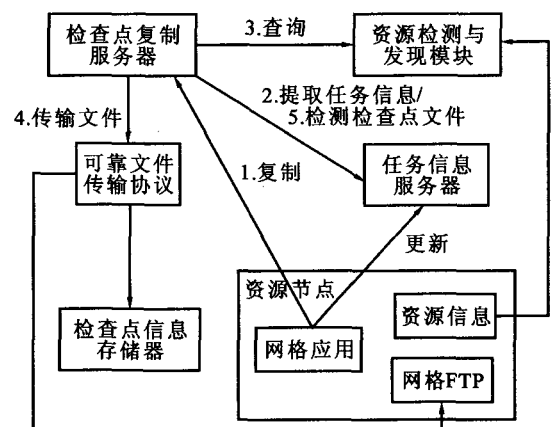


图 2 CRS 结构

CRS 提供了自动、可靠的文件复制服务, 并且能自动发现和选择存储检查点信息的存储资源. 资源节点处理任务时, 检查点复制服务器每隔一段时间都会根据任务信息服务器中的任务信息创建检查点, 检查点的文件名是根据任务信息建立的可唯一标识的文件名. 对于同一个任务, 新的检查点文件建立后会覆盖旧的检查点文件. 与此同时错误检测模块检测任务在资源上的运行情况. 检测到错误后, 任务代理服务器从 GSO 中读取与该任务相对应的检查点文件, 将相关进程回卷(Rollback)到故障前系统的状态(检查点), 经过状态恢复后从该检查点处重

新执行,而不是从程序开始处执行,从而节省了大量重复计算时间,充分提高网格的可用性。

### 3 自适应资源调度

#### 3.1 调度算法思想

根据网格动态的特点,结合蚂蚁算法<sup>[5]</sup>在资源选择上的优势,利用蚂蚁算法作为 Migol 容错框架中资源预定服务器的调度算法,提高任务执行效率。

针对网格环境中的每一个资源节点,都要对它本身的计算能力进行评估。主要依据以下几个参数:CPU的个数  $n$  及处理能力  $Tp$ (MIPS)、网络带宽  $Tb$ (Mb/s)、处理任务的成功率  $Tr$ ,以及资源的单位定价等因素,把这些因素综合起来确定资源的信息素浓度。这样就可以对资源节点进行相对全面的评估,避免只考虑单个因素的片面性。资源节点信息素的初始化公式如下:

$$T_i(0) = a * Tp(0) + b * Tb(0) + c * Tr(0) \quad (1)$$

其中,  $Tr = T_{\text{success}} / T_{\text{all}}$  表示资源节点处理任务的成功率,初始值为0。  $a + b + c = 1$ ,  $a$ 、 $b$ 、 $c$ ,分别表示CPU处理能力、网络带宽,成功率在该资源节点信息素中所占的比重。

根据每个资源节点的信息素,可以按下式求得在该节点上分配任务的概率:

$$p_i(t) = \frac{[T_i(t)]^\alpha * [\eta_i]^\beta}{\sum_{i=1}^m \{[T_i(t)]^\alpha * [\eta_i]^\beta\}} \quad (2)$$

其中,  $i$  表示资源  $i$ ,  $m$  表示所有的资源数。  $T_i(t)$  表示时间  $t$  时资源的信息素。  $\eta_i = T_i(0)$  表示资源节点的固有属性,  $\alpha$  表示资源信息素的重要性,  $\beta$  表示资源固有属性的重要性。可以根据不同的任务需求来调整  $\alpha$  和  $\beta$  的值。

资源完成对一个任务的处理后,它的信息素会发生如下变化:  $T_s^{n+1} = T_s^n + \Delta T_s$ , 该资源被分配时,  $\Delta T_s = -k$ ,  $k$  是资源处理任务的消耗量。该资源成功处理任务时,  $\Delta T_s = Ce * k$ ,  $Ce$  是奖励参数,取正值。该资源处理任务失败时,  $\Delta T_s = -Ce * k$ ,  $Ce$  是惩罚参数,取负值。

在蚂蚁的寻路过程中,蚂蚁寻找信息素最多的路径,并不担心这条路上的蚂蚁太多而走其他的路。把任务分配给资源时却不能这样,把过多的任务分配给同一个资源就会造成该资源任务繁重,而其他的资源又处于空闲状态,这并不符合资源利用最大化的原则。为了解决这个问题,需要在任务调度时增

加一个负载均衡因子<sup>[6]</sup>  $\lambda$ , 为任务寻找最优资源的同时也尽量将其分配到其他的资源节点上。同时网格资源是动态的,随时会发现其它的资源,已有资源也可能失效。为确保系统能更好地适应网格动态环境,有效利用资源,要做到以下几点:

(1) 如果发现新的资源,根据式(1)和式(2)计算出新节点的任务分配概率。

(2) 如果发现某个资源失效,把该节点的任务分配概率设置为0。

(3) 在 AIS 中把所有的资源按照任务分配概率的大小依次从大到小排列。在负载均衡因子许可的情况下把任务提交到概率最大的资源。

#### 3.2 调度流程

基于上面蚂蚁算法的思路和公式,网格任务调度流程的步骤如下:

(1) 检测并发现资源。

(2) 用公式  $T_i(0) = a * Tp(0) + b * Tb(0) + c * Tr(0)$  初始化网格中各个资源的信息素。

(3) 用  $p_i(t) = [T_i(t)]^\alpha * [\eta_i]^\beta / \sum_{i=1}^m \{[T_i(t)]^\alpha * [\eta_i]^\beta\}$  计算各个资源被分配任务的概率,把资源列表按概率从大到小排序。

(4) 提交任务:检查概率最大的资源上运行的任务数  $n_k$ 。如果  $n_k < \lambda$ ,把任务提交到概率最大的资源上,如果  $n_k > \lambda$ ,把该任务提交到下一个大概率的资源。

(5) 检测资源上任务的完成情况,建立检查点文件。如果任务成功,执行(6);如果任务失败,执行(7)。

(6) 返回结果,用  $TS^{n+1} = TS^n + \Delta TS$  更新资源信息素。

(7) 启动错误恢复机制,调用检查点文件,回卷任务,用  $TS^{n+1} = TS^n + \Delta TS$  更新资源信息素。

(8) 重复(3)~(7),直到所有的任务都成功执行,返回正确的结果为止。

### 4 仿真及结果分析

根据以上对网格容错模型和蚂蚁算法的分析,利用仿真平台 Gridsim 进行仿真,蚂蚁算法中各个参数可以有不同的设置,执行的结果也有很大的不同,文中在参考其他文献的基础上结合仿真实验结果,把各个参数设置为:  $a = 0.4$ ,  $b = 0.3$ ,  $c = 0.3$ ,  $\alpha = 0.5$ ,  $\beta = 0.5$  资源成功处理任务时:  $Ce = 0.003$ , 资源处理任务失败时:  $Ce = -0.002$ 。模拟创建 10 个资源,产生不同的任务数,分别在文中所介绍的自适

应容错方法(Adaptive - Migol)和原先的容错方法(Migol)下执行,任务执行时间如图 3 所示.

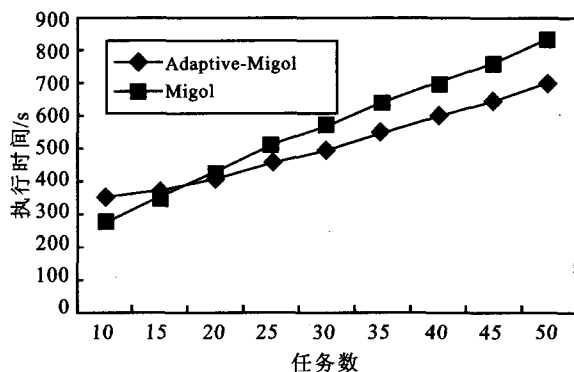


图 3 任务执行时间

可以看出,在任务数不多的情况下 Adaptive - Migol 处理任务的时间要比 Migol 的时间长,任务逐渐增多后 Adaptive - Migol 的处理时间就比 Migol 所需的时间少,这是因为自适应资源选择需要对资源信息素进行计算并且对资源的优劣进行排列这需要一定的时间,在任务数不多的情况下这些时间所占的比重比较大,影响也大.这也说明自适应资源选择算法并不适用于任务数少的任务,资源选择的优势也体现不出来.随着任务数增加,资源选择的时间在总时间中占的比重逐渐变小,这些时间对总时间的影响也逐渐变小.由于任务总是被分配到相对最优的资源上运行,每个任务的执行时间相比较也要比 Migol 所需的时间少,在任务量大的情况下,自适应资源选择的优势就能明显的体现出来<sup>[7-8]</sup>.

图 4 显示了任务在 Adaptive - Migol 和 Migol 上运行的成功率.可以看出:Adaptive - Migol 处理任务的成功率更高.这是因为自适应资源算法选择了相对最优的资源,这些资源也相对稳定,从而使得任务的处理成功率更高.任务数比较大并且对成功率要求高的任务就可以选择此自适应调度算法,能较好的减少执行时间并提高任务执行成功率.

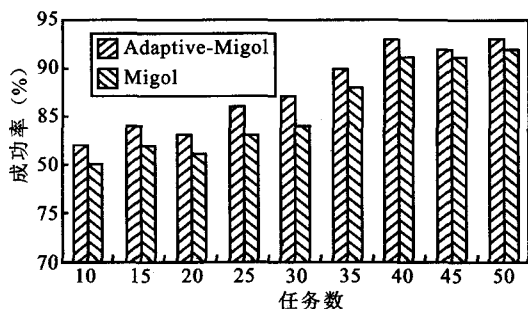


图 4 处理任务成功率

## 5 结束语

文中研究了网格环境下的容错问题,根据网格资源动态的特点,提出了 Migol 容错框架下结合蚂蚁算法在选择资源方面的优势形成了自适应容错体系,最后从任务的执行时间和任务的处理成功率两个角度对此算法进行分析比较.仿真表明该容错体系可以更好地适应网格复杂的环境,有效地提高处理任务的成功率.下一步工作就是根据网格用户不同的需求调整蚂蚁算法的各个参数,找到参数的最优组合,进一步提高成功率.

## 参考文献:

- [1] 都志辉,陈渝. 网格计算[M]. 北京:清华大学出版社, 2002:3-55.
- [2] 李军国,黄昱,邹键. 一种中间件服务容错配置管理方法[J]. 计算机学报, 2007, 30(10):1696-1704.
- [3] 孙广中,肖锋,熊曦. MapRduce 模型的调度及容错机制研究[J]. 微电子学与计算机, 2007, 24(9):178-180.
- [4] Ander Luckow, Bettina Schnor. Migol: a fault tolerant service framework for MPI applications in the grid[C]// Future Generation Computer Systems - The International Journal of Grid Computing: Theory, Methods and Application. Italy, 2008, 24(2):142-152.
- [5] Andre Luckow, Bettina Schnor. Adaptive checkpoint replication for supporting the fault tolerance of applications in the grid[C]// Seventh IEEE International Symposium on Network Computing and Applications. Cambridge, 2008, 2(8):299-306.
- [6] Fatos Xhafa, Javier Carrtero. Design and evaluation of tabu search method for job scheduling in distributed environments[C]// IPDPS 2008.. IEEE International Symposium on Parallel and Distributed Processing. USA, Florida, 2008, 3(8):1-8.
- [7] 赵晨阳,杨裔,刘礼. 基于蚂蚁算法的网格作业调度研究[J]. 微电子学与计算机, 2007, 24(9):78-81.
- [8] Ruay Shiung Chang, Jih Sheng Chang, Po Sheng Lin. Balanced job assignment based on ant algorithm for computing grids[C]// 2007 IEEE Asia - Pacific Services Computing Conference. Japan, Tsukuba Science City, 2007, 6(7):291-295.

## 作者简介:

陈欢欢 男, (1984-), 硕士研究生. 研究方向为网格计算.

刘宴兵 男, (1971-), 博士, 教授. 研究方向为网格计算、网络路由.