文章编号:1001 - 9081(2008)02 - 0371 - 03

基于 ProActive 的容错调度器设计与实现

梁正友, 孙 宇
(广西大学 计算机与电子信息学院, 南宁 530004)
(zhyliang@gxu.edu.cn)

摘 要:在分布式计算系统中保证并行应用计算的正确性及提高计算系统中动态资源的利用率是一个重要的研究问题。在原有的基于 ProActive 的并行计算平台上,引入呼吸通信机制、故障节点发现机制和子任务重新调度机制,设计和实现了一个容错调度系统。实验表明该调度器在部分节点出现故障的情况下,能保证并行计算的正确性,并具有较好的性能。

关键词:并行计算;调度;容错;ProActive
中图分类号: TP393 **文献标志码**:A

ProActive-based fault-tolerant task-scheduler

LIANG Zheng-you, SUN Yu
(*School of Computer and Electronic Information, Guangxi University, Nanning Guangxi* 530004, *China*)

Abstract: It is an important research issue to ensure the computation correctness for parallel application and enhance the using rate of dynamic computing resource in distributed computing system. Based on the previous ProActive-based parallel computing system, a ProActive-based Fault-tolerant Task-scheduler was developed, which combined the breathe mechanism, fault-discover mechanism and subtask reschedule mechanism. Experiments show that the Fault-tolerant Task-scheduler has good performance and ensures the computation correctness even if when some computing resources fail.

Key words: parallel computing; schedule; fault-tolerant; ProActive

0 引言

网格计算利用动态的网络计算资源获得取高性能、高吞吐量或协同计算。但由于网络资源的动态性,计算资源可随时加入或退出计算系统。同时,随着系统规模的不断扩大,系统内部发生故障的概率也会呈指数增长。在计算过程中,并行应用程序所在的节点一旦发生故障或计算节点强行退出,就会导致整个并行计算的彻底失败,故障之前的大量计算不可再用,使得应用程序不得不从头开始执行;最差的情况下,甚至可能导致整个计算系统的崩溃。为了减少由于系统故障产生的不良后果,就必须采用相应技术来提高网格计算系统的可靠性和并行计算结果的正确性。容错技术就是提高系统可靠性的手段之一。

容错技术是指在计算机内部出现故障的情况下,计算机系统仍然能正确地完成指定的任务[1]。检查点与卷回恢复技术是一类重要的容错技术。根据设置检查点与恢复机制的不同,检查点与恢复机制被分为两大类:基于检查点的卷回恢复和基于日志的卷回恢复。

ProActive[2]是一个网格网络环境下开发并行、分布和并发计算、移动计算的基于 Java 的开发工具和环境,目前被广泛地应用于开发基于 Java 的高性能计算。ProActive 提供了两种不同的容错协议(CIC 和 PML),这些容错协议可无缝的嵌入应用中。

本文在原有的并行计算平台 WPHPC[3,4]上进行改进,设计和实行了一个容错调度系统。与 ProAcitve 提供的容错机制不同,我们将调度和容错结合起来,在调度器而不是应用层实行容错,使得原有的调度系统具有容错性,提高系统的可靠性,同时又能实现负载均衡,提高计算系统的效率。实验表明所设计的容错系统运行正确,具有良好的性能。

1 容错调度器的设计与实现

1.1 WPHPC

WPHPC[3,4]是我们先前设计实现的一个基于 ProActive 的并行计算平台,WPHPC 具有支持异构、可移植性好、易于扩展、配置简单的特点;WPHPC 设计了一个高效的调度器,使得并行应用实例能够在 WPHPC 上进行分配调度,实现了系统的负载平衡,提高了实例的执行效率和负载均衡。但是 WPHPC 也存在着不足,比如:节点的故障将会影响应用实例的运行结果的正确性,甚至是影响整个系统的稳定性。针对 WPHPC 存在的不足,我们在 WPHPC 系统上进行改进,增加相应的容错模块,并且对原有的调度器、用户代理等模块做相应的调整和改进,构成一个新的容错调度模块,并且在新的容错调度模块上设计和实现一个并行计算系统。我们在容错调度模型的设计中主要考虑以下几个重要因素:1) 故障节点的发现。当计算节点发生故障或者强制退出之后,服务器可以自动的发现故障或退出的节点。2) 故障节点的任务恢复。当计算节点出现故障或退出之后,服务器尽快地在其他正常节点上恢复运行故障节点上的计算任务。3) 发现和恢复的时效性。服务器发现故障节点和恢复故障节点的时间应该较短,不影响计算任务在系统上的执行。

1.2 容错调度器设计

针对之前讨论的几个要设计中要考虑的因素,我们在

收稿日期:2007 - 08 - 20;**修回日期**:2007 - 10 - 20。 **基金项目**:广西教育厅科研项目([2006]26 号);广西大学博士启动基金项目。
作者简介:梁正友(1968 -),男,广西天等人,副教授,博士,主要研究方向:网格计算、并行分布式计算; 孙宇(1980 -),女,广西南宁人,硕士,主要研究方向:网格计算。

WPHPC 基础上设计一个容错调度器,如图1。

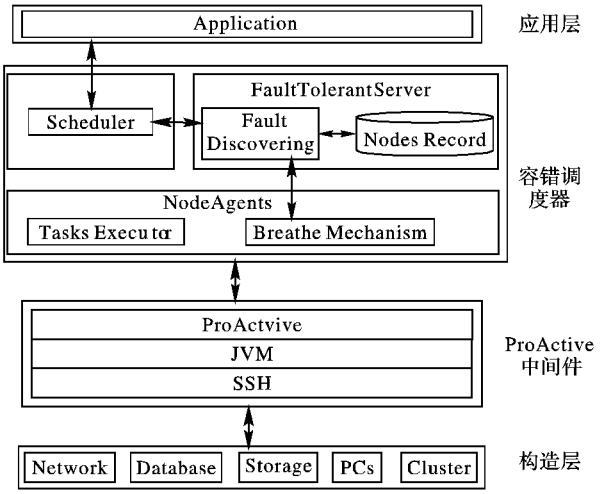

图1 系统结构

第一层:构造层,由高速互联的 PC 工作站群或 Cluster 组成,提供平台所需的计算和存储设备,它们提供了执行计算程序的环境。

第二层:基于 ProActive 的中间层,在这一层中有三个小层,分别是提供连接的 SSH 层、JVM 层和 ProActive 层。

1)SSH 连接层(SSH):采用 SSH 作为连接协议,为计算节点的动态部署提供安全的部署连接。

2)JVM 层(JVM):为 ProActive 的通信及运行提供 Java 环境支持。由于 ProActive 是一个纯 Java 的中间件,需要 JVM 为其提供运行环境支持,同时也为系统的跨平台提供支持。

3)ProActive 层:基于 Java 的并行分布式计算支持包。支持用户开发 Java 并行分布式应用程序。

第三层:容错调度器层,为应用程序在系统上的运行提供有效的任务调度和执行。它主要由在服务器端的调度器、容错服务器和计算节点上的节点代理构成。目前它支持基于 ProActive 的并行计算应用的调度。

第四层:应用层,基于 ProActive 并行应用程序[3,4]。

并行应用程序执行的工作流程如下:

1)服务器节点通过基于 ProActive 的中间层连接实现远程计算节点的启动与计算节点上的用户代理(NodeAgent)的部署;

2)用户代理收集各自节点的信息,并返回给调度器;

3)用户在客户端运行编写的应用程序向服务提交计算任务,并通过门户网站提交参数信息;

4)调度器(Scheduler)收到用户请求,根据用户提交的信息选择用户指定的调度算法进行任务调度,生成任务的调度信息;

5)调度器根据调度信息,将各子任务分配到各计算节点;

6)各个用户代理负责本节点上的子任务的执行;

7)当容错服务器(FaultTolerantServer)发现系统中有节点出现故障,则通过调度器重新调度分配子任务到其他计算节点;

8)当用户代理负责的子任务全部执行完毕之后,用户代理并将执行结果送给调度器;

9)调度器对结果进行归约生成执行结果;

10)将结果以文件的形式存放在服务器上,用户可通过门户网站,查看和下载执行结果。

1.3 容错调度关键技术

任务的调度和执行由运行在服务器端的容错调度模块完成。容错调度模块被设计成为并行计算系统中的一个中间模块。容错调度模块中的容错技术,它可以有效地保证系统的可靠性和任务执行的正确性。该模块实现故障节点的检测以及故障节点上子任务的恢复和重新调度的功能。模型由3个功能模块组成:

节点代理(NodeAgent) 保证并行实例在计算系统上的运行。节点代理执行调度器分配的任务,并且提交任务的运行结果;每个节点代理除了能够执行任务之外,还必须具备呼吸通信机制,实现节点与服务器的呼吸通信。

容错服务器(FaultTolerantServer) 保证了并行应用实例在计算系统上的运行结果的正确性。容错服务器在有效时间内发现故障节点,更新可达节点列表;还包含了一个存储器,用来存储了各个计算节点上的任务集合以及可达节点列表。

调度器(Scheduler) 为并行应用实例在计算系统上的运行提供了有效的任务调度,包括任务的初始分配调度以及出故障后的容错调度。调度器的关键就是调度算法实现。

1.3.1 节点的呼吸通信机制

计算节点在每一个 TTL 时刻利用自身的呼吸通信机制,向容错服务器发送呼吸消息,告知服务器该节点依然存活。其中,TTL 是检测点时刻点,用来检查节点是否存活,这个时刻与容错服务器的 TTL 相一致;呼吸消息的格式为(NodeAgentID、UUID),NodeAgentID 是用户代理的全局唯一序列号;UUID 是节点代理发送消息的全局唯一 ID。NodeAgentID 来唯一标识消息的来源节点;根据 UUID 来唯一标识这个信息。

1.3.2 故障节点的发现机制

发现机制的基本思想是:不断接收来自计算节点的呼吸通信消息,并且通过呼吸通信消息来判断系统中的节点是否可达。容错服务器在每一个 TTL 时刻都去检查各个节点发过来的呼吸通信消息。由于节点 i 在不同时刻发送的消息的全局唯一 ID 号(UUID)不同,由此可以判断我们接受到的消息是否与过去接受的消息相同。若接受到的消息的 UUID 与前一次相同,则认为该节点暂时不可达,并且记录下收到同一个消息的次数(SN)。当接受到同一个消息的次数达到一定的数值时,我们认为这个时候消息的发送端节点 i 出现故障,则要更新可达节点列表。

1.3.3 容错调度的算法

假设有 m 个子任务运行在 n 个节点的并行计算系统中,对于每个子任务都有其对应的副版本。为了便于容错调度算法的设计和分析,本文假设如下的任务模型:

定义1 设分布式系统中处理机节点个数为 n,则可达节点的集合可以定义 $\phi = \{N1, N2, \cdots, Nn\}$,初始时可达节点集合是节点的集合。每个节点可以表示为 (Ci, Pi, P_UNi),Ci 代表了 Ni 节点的计算能力,$Pi = \{ST\alpha, ST\beta, \cdots, ST\gamma\}$ 表示分配到标志 Ni 节点上的子任务的集合,$P\_UNi = \{ST\alpha, ST\beta, \cdots, ST\gamma\}$ 表示未完成的子任务的集合。

定义2 设要处理的子任务数为 m,子任务的计算量相当并且互相独立,那么子任务的集合可以定义为 $\varphi = \{ST1, ST2, \cdots, STm\}$,每个子任务都是一个二元组 $STi = \{ST\_Mi, ST\_BKi\}$,分别表示这个子任务的基版本和副版本。

定义3 当容错服务器发现节点失败之后,通知调度器故障的节点 ID,并且更新的可达节点列表,之后由调度器负

责子任务副版本的重新调度。容错调度方法与初始调度策略相一致,分为池调度、静态调度和动态调度三种[4]。本文仅介绍基于容错的池调度算法。

池调度基本思想:将相互独立的各个子任务,加入到任务池中,依次给每一个计算节点分配一些子任务去进行计算,子任务计算结束后,返回执行结果。计算节点从任务池中取出新的子任务进行计算,直到所有的任务执行完毕。如定义 1 所设,m 为总的子任务数,n 为计算节点的个数,s 为任务池中剩余的子任务数。

若系统中节点 j 故障,容错服务器将节点 j 未完成子任务 *P_UNj* 的副版本直接加入任务池中,更新可达节点列表,等待再次的池调度。容错的池调度算法描述如下:

1. 将用户程序产生的子任务加入任务池;
2. 给每一个节点分配$\lceil \frac{m}{2n} \rceil$个子任务;
3. while (任务还未完成){
   3.1. if (节点的子任务执行完毕){
   返回计算结果}
   3.2. else {
   从任务池中取$\lceil \frac{s}{n} \rceil$个子任务分配到该节点;}
   /* 对节点 j 上的子任务 *P_UNj* 进行容错调度 */
   3.3 if (节点 j 出故障){
   3.3.1 调整子任务计数器;
   3.3.2 获得不可达节点的子任务 *P_UNj* 相对应的副版本;
   3.3.3 将子任务的副版本创建为活动对象;
   3.3.4 将创建的活动对象加入到事先定义好的任务池中;将子任务的副版本放在最后调度;
   3.3.5 更新新的可达节点列表 ϕ;}
   }
4. 收集各结果,返回最后结果;

2　实验和性能分析

为了测试容错调度器的性能,我们在容错使用矩阵乘应用实例来进行性能测试。矩阵乘应用实例由两个 420 × 420 的矩阵进行矩阵乘,分别在拥有 $n = \{1,11\}$ 个计算节点的并行计算原型上进行测试。其中,各个机器的性能使用测试软件 nbench 测试得到。并行计算原型上的各个计算节点的运行环境如表 1 所示。

表 1　机器配置参数

机器	性能基准	CPU /GHz	内存 /MB
A	5.975	2.0	512
B	9.493	2.4	512
C	9.600	2.4	512
D	9.539	2.4	512
E	9.496	2.4	512
F	9.527	2.4	512
G	9.647	2.8	512
H	9.596	2.8	512
I	9.572	2.8	512
J	9.579	2.8	512
K	9.764	2.4	512

测试时,分别测量容错调度算法在 n 个计算节点上无故障节点时应用实例的运行时间(n 节点无故障运行时间)、在 $(n-1)$ 个计算节点上无故障节点时应用实例的运行时间(($n-1$) 节点无故障运行时间),以及在 n 个计算节点上有一个节点故障后应用实例的运行时间(n 节点容错运行时间)。实验结果如图 2 所示。

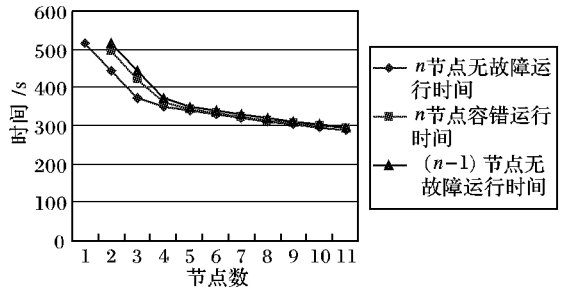

图 2　池调度容错调度算法运行时间对比

从图 2 可见容错调度算法是有效的,而且出现故障节点之后,计算任务能够在短时间内被合理地重新调度到各个计算节点上,有效地实现了负载平衡。从结果我们也可以看出,如果具有 n 个节点的系统中有一个节点发生故障之后,应用实例的执行效率优于或者等于 $n-1$ 个节点共同执行这个应用实例所得到的执行时间。

表 2　引入容错机制带来的时间花费

节点数	*Tnb*	*Tb*	百分比/%
1	371	380	2.4
2	242	249	2.9
3	206	211	2.4
4	196	197	2.0
5	187	190	1.6
6	180	183	1.7
7	175	177	1.7
8	170	172	1.2
9	163	165	1.2
10	160	162	1.3
11	158	159	0.6

引入容错机制后,调度器需要额外的时间花销。为了测试容错机制带来的开销,实验分别在有 1 ~ 11 台计算节点的系统原型上,用本文开发的容错调度器与 WPHPC 分别运行一个具有 30 个子任务的 420 × 420 矩阵乘来分别对比:没有容错机制时运行矩阵乘所需要的时间(*Tnb*),有容错机制时运行矩阵乘所需要的时间(*Tb*),以及呼吸机制占整个执行时间的百分比。实验结果如表 2 所示,实验表明引入容错机制带来的额外开销很小,而且随着计算节点的增加减小趋势。

3　结语

本文在原有的基于 ProActive 的并行计算平台上,引入呼吸通信机制、故障节点发现机制和子任务重新调度机制,设计和实现了一个容错调度系统。实验表明该调度器在部分节点出现故障的情况下,能保证并行计算的正确性,并具有较好的性能。目前本文的工作仅适用于独立子任务的并行计算类型。

参考文献:

[1] GREVE F, HURFIN M, LE NARZUL J P. OPEN EDEN: a portable fault tolerant CORBA architecture[C]// 2nd International Symposium on Parallel and Distributed Computing. Slovenia: IEEE, 2003:88 - 95.

[2] CAROMEL D. ProActive[EB/OL]. [2007 - 03 - 01]. http://www.sop.inria.fr/oasis/ProActive/2007.

[3] 董明刚,梁正友. Windows 下基于 ProActive 的并行计算平台的关键技术研究[J]. 计算机工程,2006,32(19): 105 - 107.

[4] 董明刚,梁正友. 基于 ProActive 的并行计算任务调度器的研究[J]. 计算机工程,2007,33(7):73 - 74.