

网格环境下一种基于 T-RAG 图的关联任务迁移算法

张 千, 刘 昶

(中国石油大学 计算机与通信工程学院, 山东 东营 257062)

摘 要: 网格计算中的大型应用程序往往被分解为多个关联任务. 关联任务提交到网格环境后, 由于网格环境的异构性和不确定性, 当任务执行发生异常时, 需要对该任务实施迁移. 通过分析任务的迁移代价, 以 Total-Copy 算法为基础提出了基于 T-RAG 图的关联任务迁移算法. 实验结果表明, 对于大规模的关联任务, 该算法具有较高的协调性能, 并且提高了关联任务的执行效率.

关键词: 进程迁移; 关联任务; 网格

中图分类号: TP31

文献标识码: A

文章编号: 1000-7180(2010)08-0057-04

A Migration Algorithm of Related Tasks Based on T-RAG Graph in Grid

ZHANG Qian, LIU Chang

(College of Computer and Communication Engineering, China University of Petroleum, Dongying 257062, China)

Abstract: A large-scale application program is always broken down into a number of tasks that contain much relation in grid environment. In order to improve the efficiency, the task needs to be migrated after submitting to the grid. Through analyzing the migration cost of selecting the task, a migration algorithm of related tasks of T-RAG graph based on Total-Copy algorithm was proposed, experimental results show that for large-scale related tasks, this algorithm has higher coordination, improves the efficiency of the execution of the related tasks.

Key words: process migration; related tasks; grid

1 引言

由于网格环境的异构性和不确定性, 可能会导致某些节点出现故障, 任务执行发生异常. 进程迁移可以将任务迁移到其他节点, 使任务能够继续执行. 因此, 进程迁移成为提高网格系统整体性能和增强系统容错能力的关键技术之一.

目前对迁移机制的研究, 通常限制在集群的范围内, 并且仅适用于独立的作业进程. 多数迁移系统采用较为普遍的 Total-Copy 算法, 该算法的核心在于如何将进程的信息传送到目的节点, 且针对的任务也是普通的进程, 在进程的选择方面采取最久运行时间的策略, 并没有对关联任务的迁移进行分析和研究.

本文在 Total-Copy 算法的基础上, 提出了一种基于 T-RAG 图的关联任务迁移算法, 实验结果验证了该算法在一定程度上提高了关联任务的执行效率.

2 Total-Copy 迁移算法

Total-Copy 算法具体流程如下: 源节点向目的节点发送请求, 目的节点同意之后, 便开始进程的迁移. 进程迁移的信息主要包括以下内容: 进程控制信息、进程执行状态信息、进程地址空间的信息等等. 首先源节点将进程的控制状态和运行状态发送到目的节点, 随后是相关的文件描述符. 源节点完成信息的传送后, 向目的节点发送一条消息, 通知目的节点开始对迁移进程进行恢复. 在进程迁移过程中, 该算

法对消息的处理,采取对源节点和目的节点进程冻结的策略,等待迁移完成之后恢复进程之间的通信.算法流程如图 1 所示.

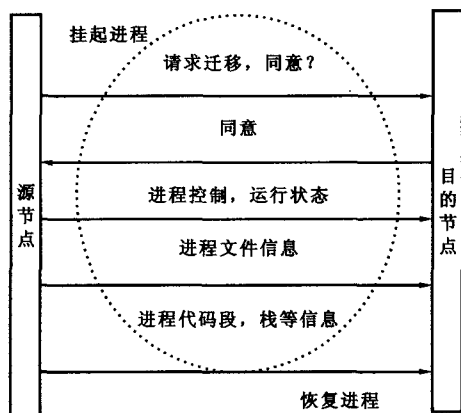


图 1 Total-Copy 算法流程图

进程迁移本身的目的就是为任务的执行开辟一种高效率的方式——即更换任务的执行主机,使任务在更合适的一种环境下运行.因此任务的迁移可以简单的概括为选择进程,执行迁移,寻找目的主机,恢复进程.

由图 1 可知,Total-Copy 算法只适合单一任务进程的迁移,且该算法的核心是如何将进程的相关信息发送到目的节点.

对于关联任务而言,由于任务之间频繁的依赖关系,使得关联任务的迁移更为复杂.本文就关联任务产生大量数据依赖的问题,针对进程迁移中待迁进程和目的节点的选择进行了详细分析,在分析 T-RAG 图的基础上,提出了改进的 Total_Copy 算法.

3 基于 T-RAG 图的关联任务迁移算法

3.1 关联任务分析

为了更好地分析关联任务,引入以下定义:

定义 1 任务资源分配图(T-RAG)

T-RAG = (V, E, WA, WE) 是一个四元组^[1]. V 是节点的集合 $V = \{(s_u, m_j) \mid u \leq K; j \leq N\}$, 其中 (s_u, m_j) 表示任务 s_u 映射到资源 m_j 上; $E = \{e_{ij} \mid i > 0, j \leq K; i \neq j\}$ 表示任务之间的优先级关系和通信关系. WA = (W, A) 是一个二元组,表示节点的权值,其中 $W = \{w(s_u) \mid 0 < u \leq K\}$ 表示任务 s_u 的计算量的集合, $A = \{a(m_i) \mid 0 < i < N\}$ 表示资源 m_i 的计算能力集合. WE = (C, B) 是一个二元组,表示边的权值,其中 $C = \{C_{uv} \mid 0 < u, v \leq K\}$ 表示具有依赖关系的任务 s_u 与 s_v 之间的数据传输量的集合, $B = \{b_{ij} \mid 0 < i, j < N\}$ 表示

部署了依赖任务 s_u 与 s_v 的资源 m_i 与 m_j 间的网络连接带宽的集合.一个 T-RAG 示意图如图 2 所示.

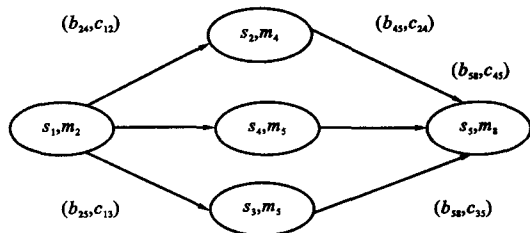


图 2 任务-资源分配图

定义 2 关键路径

由图 2 可知,从起始任务到任务结束,可以有 multiple 路径,每条路径所消耗的时间为该路径上各个节点的计算时间与传输时间的和,其中运行时间最大的为关键路径,关键路径上的节点称为关键节点,路径上的任务称为关键任务.关键任务构成的集合记为 KEY(s_i).

定义 3 关联任务集和关联度

若任务 s_i 与 s_j 有依赖关系 c_{ij} ,则 s_i 为 s_j 的前续任务, s_j 为 s_i 的后继任务.一个任务 s_i 所有的前续任务和后继任务的总和就构成了该任务的关联任务集,记为 RE(s_i).集合中元素的个数称为该任务的关联度,记为 RED(s_i).

定义 4 动态任务和静态任务

在节点上已经执行的任务称为动态任务.由动态任务构成的集合称为动态任务集,记为: DY(s_i).尚未执行的任务称为静态任务,其构成的集合记为 ST(s_i).

定义 5 任务之间的关系

由图 2 可知,不同的任务之间可能存在 3 种关系,①父子关系:任务 s_u 和 s_v 在同一条路径上,并且两个任务直接相连;②独立关系:任务 s_u 和 s_v 不在任何一条路径上;③祖先关系:任务 s_u 和 s_v 在同一条路径上,但是两者并非直接相连.

3.2 迁移代价分析

通常活跃进程的迁移代价主要包括两部分,即固定迁移代价和资源使用代价.

固定迁移代价包括进程系统状态的封装,数据的传输,状态的解包时间.通常状况下,进程的固定迁移代价,是可以估算的一个固定常数^[2].

资源的使用代价主要是指进程地址空间的转移代价.在目前的研究当中,内存的转移代价主要由进程地址空间的状态存储量和网络之间的带宽决定,因此内存的转移代价可以表示为: $RP = M/B$ ^[3],

M 是内存存储量, B 是节点之间的网络带宽。

I/O 迁移代价,即文件的 I/O 问题.在进程未来的执行过程中,进程还会进行和当前已经完成次数相当的本地 I/O 和远程 I/O^[4].为了进行简化,我们认为节点内部进行 I/O 读写操作时,读写一个数据块的时间开销相同,不同进程对不同的节点进行远程 I/O 时,读写一个数据块的时间开销也相同.进程 p 从节点 i 迁移到节点 j 的 I/O 迁移代价,本文采用式(1)^[5]:

$$C_{i/o}(i, p, j) = LO_{(i/o)}(p) \times C_{out} + [RT_{(i/o)}(P) - TW_{(i/o)}(i, p, j)] \times C_{out} - [LO_{(i/o)}(p)C_{in} + RT_{(i/o)}(P) \times C_{out}] \quad (1)$$

式中, C_{in} 为节点内部 I/O 操作读写一个数据块的时间开销, $LO_{(i/o)}(p)$ 为本地 I/O 操作读写的文件块数目. C_{out} 为节点间读写一个数据块的所消耗的时间, $RT_{(i/o)}(P)$ 为进程 p 进行的节点之间远程 I/O 操作数据块数目, $TW_{(i/o)}(i, p, j)$ 为节点 i 与节点 j 之间的 I/O 操作数据块数目。

进程未来的运行过程中,还会与当前已经完成次数相当的本地 IPC 和远程 IPC 操作^[6].节点内不同进程的通信代价相同,节点间不同进程的通信代价相同.因此,进程 p 从节点 i 迁移到节点 j 的进程通信代价可以表示为式(2):

$$C_{ipc}(i, p, j) = LO_{(ipc)}(p) \times IPC_{out} + [RT_{(ipc)}(P) - TW_{(ipc)}(i, p, j)] \times IPC_{out} - [LO_{(ipc)}(p) \times IPC_{in} + RT_{(ipc)}(P) \times IPC_{out}] \quad (2)$$

式中, IPC_{in} 为节点内部通信开销, $LO_{(i/o)}(p)$ 为本地进程间通信次数. IPC_{out} 为节点间的通信开销, $RT_{(i/o)}(P)$ 为进程 p 进行的节点之间的通信次数, $TW_{(i/o)}(i, p, j)$ 为节点 i 与节点 j 之间的通信次数。

3.3 待迁进程的选择

关联任务的完成与否主要取决于关键路径上任务的完成情况.也就是说,关键路径上任务的执行效率决定了整个应用程序的完成情况.关键路径上的任务具有执行时间长,计算量大的特性.如果选择关键任务进行迁移,就必须充分考虑进程的剩余执行时间和迁移时间的开销问题。

关于进程生命周期的研究,Horchol 和 Downey 曾经指出,在负载较高的情况下,对于运行时间较长的进程,进程剩余执行的生命时间很大概率上与当前的运行时间相等.根据这一特性引入以下理论:设某进程继续执行的概率参数为 λ ,随机变量 ξ 表示进程继续运行的时间,我们可以将进程的剩余执行

时间 ξ 看作是满足当前执行时间期望值 T_E 的一个指数分布,其密度函数为式(3):

$$f_{\xi}(t) = \lambda e^{-\lambda t} \quad (3)$$

式中, $\lambda = 1/T_E$,那么进程继续执行时间大于 t 的概率就可以表示为式(4):

$$G_{\xi}(t) = \int_t^{+\infty} \lambda e^{-\lambda t} dt = e^{-\lambda t} \quad (4)$$

在目的主机没有确立之前,无法准确地计算关键任务的迁移开销,为了较好地比较进程继续执行时间和迁移时间的关系,引入进程平均迁移开销概念。

进程的平均迁移开销 $\overline{C_{total}}(p)$ 可以表示为式(5):

$$\overline{C_{total}}(p) = \overline{C_{ipc}}(p) + \overline{C_{i/o}}(p) + RP \quad (5)$$

式中, RP 表示该进程的内存转移代价, $\overline{C_{ipc}}(p)$ 是该进程的平均 IPC 开销, $\overline{C_{i/o}}(p)$ 是该进程的平均 I/O 开销。

在选择进程进行迁移时,首先判断的对象是关键任务,若关键任务正在执行,则根据提出的以上策略进行分析,判断是否对该任务进行迁移.当不存在关键任务时,对它的非关键任务,采用以下方式处理.由于在对关联任务进行调度时,为了节省部分开销,充分利用资源,部分具有父子关系的任务很有可能分配到同一个节点上.因此,对于出现以上情况的进程,暂且先不考虑.在运行时间相差不大的情况下,优先考虑其它的进程进行迁移。

本文中,源节点发送的迁移请求都是通过客户端发送给网格迁移服务中心,然后通过服务中心完成关联任务的迁移。

待迁进程的选择算法如下:

(1) 每经过一个时间周期 T ,计算网格中各个节点的负载状况;

(2) 高负载节点对节点内的任务集合 S 进行轮询;

(3) 判断 s_i 是否属于 $K(s_i)$;

(4) 如果 $s_i \in K(s_i)$,并且 $s_i \in ST(s_i)$,向迁移服务中心发送迁移目标节点请求。

(5) 迁移服务中心接收源节点信息,查询,并获取目的节点信息。

(6) 挂起该进程,打包进程状态,向目的节点发送进程状态信息。

(7) 如果 $s_i \in ST(s_i)$,计算该进程的平均迁移代价 $\overline{C_{total}}(p)$,令 $t = \overline{C_{total}}(p)$,计算 $G_{\xi}(t)$ 的概率值,若 $G_{\xi}(t) > 0.5$,则执行迁移。

(8) 若 $G_i(t)$ 不满足, 计算节点上所有进程的运行时间, 将没有执行完毕而运行时间最长, 且在该节点上没有父子依赖的进程, 发送到目的节点.

(9) 传输完毕后重新计算节点负载.

3.4 目的节点的确立

进程迁移中对目的节点的选择依据, 主要是减少迁移之后的通信开销, 尽可能地将远程通信变成本地通信, 这对于频繁通信的关联任务而言, 可以有效地提高整个应用程序的执行效率. 在对目的节点的选择过程中, 重点考虑与被迁任务有父子关系的任务所在的节点, 只有当预迁移任务的前续任务未执行完毕时, 才会选择前续任务所在节点. 当低负载队列中出现较多这种情况时, 通过节省的通信开销 $Save_c$ 来进行比较^[7], 可表示为式(6):

$$Save_c = TW_{(ipc)}(i, p, j) \times (IPC_{out} - IPC_{in}) + TW_{(i/o)}(i, p, j) \times (RT_{out} - LO_{in}) \quad (6)$$

目的节点选择过程如下:

- (1) 计算节点的负载状况;
- (2) 查询低负载节点队列, 并获取关联任务表信息, 是否存在和预选进程相关的父子进程的节点;
- (3) 判断是否存在和预选任务 s_i 存在父子关系的节点, 有则计算节省的通信开销 $Save_c$, 选择具有最大值的节点;
- (4) 若不存在, 则计算进程迁移的 I/O 代价和进程通信代价 IPC , 并最终计算节点总的迁移开销, 选择最佳目的节点;

(5) 服务器发送消息给目的节点开始消息发送;

(6) 目的节点恢复进程状态后, 重新计算节点负载.

3.5 算法流程

(1) 源节点负载较高, 向迁移中心服务器发送迁移请求, 服务器查看关联任务表, 根据进程的选择策略选出待迁移进程, 并将进程控制信息和状态打包.

(2) 迁移中心根据迁移代价的分析选择一个合适的目的节点, 源节点与目的节点协商.

(3) 同意接收迁移请求后, 挂起迁移的进程, 源节点对进程的控制状态和运行状态进行整理并传送.

(4) 源节点传送文件描述符和脏的文件缓存块, 并将所有的代码页、堆和栈页传送.

(5) 源节点完成信息的传送后, 向目的节点发送一条消息, 通知目的节点开始对迁移进程进行恢复.

4 算法性能分析

本文在网格环境下对该算法进行了实验, 由于只考虑迁移对关联任务的影响, 因此在设计实验的关联任务图时, 已经选取具有最小完成时间的 T-RAG 图.

为了较好地测试算法的性能, 本文分别在网格环境, Condor 和 Pbs 集群系统测试有关应用程序. 由于 Condor 和 Pbs 集群系统本身提供了相应的迁移机制, 本文将提出的迁移算法与这两个系统进行对比, 主要从执行任务总数和最大关联任务数进行考虑. 各系统的实验环境及参数如表 1 所示.

表 1 实验环境参数图

网格系统	GridShed	Condor	Pbs
节点数量	3 个 Pbs 节点; 3 个 Condor 节点	6 个 Condor 节点	6 个 Pbs 节点
总任务数	Min: 10; Max: 100	Min: 10; Max: 100	Min: 10; Max: 100
最大关联度	3-7	3-7	3-7

不同的总任务 V 下, 最大关联度一定的情况下, 各个系统对关联任务执行情况的对比如图 3 所示.

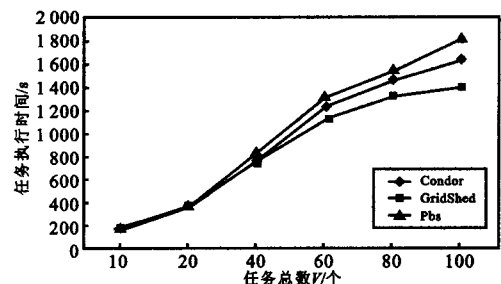


图 3 任务最大关联度一定下各系统执行状况图

从图 3 可以看出, 在任务总数较少的情况下, Pbs 较 Condor 的执行时间较短是因为 Condor 有一套较为复杂的迁移机制, 使得任务的执行时间花费较长; 但随着任务数量的不断增加, 关联任务之间的关系更加复杂时, 本文提出的算法用时较少.

任务总数 V 一定, 最大关联度逐渐增加的情况下, 各系统的执行效果如图 4 所示.

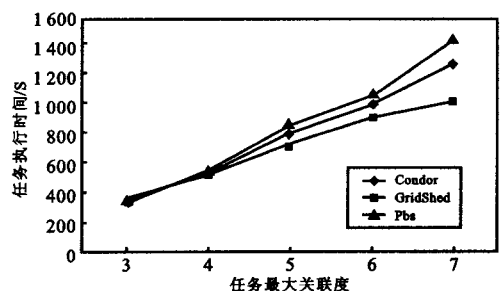


图 4 总任务数一定下各系统执行状况图

(下转第 65 页)

质量需求.

参考文献:

- [1] Neal Leavitt. Are web services finally ready to deliver[J]. IEEE Computer, 2004, 37(11):14-18.
- [2] 陈宁江. 面向Web服务的层次型动态失效检测体系[J]. 计算机工程, 2009, 35(17):94-95.
- [3] Naohiro Hayashibara, Makoto Takizawa. Performance evaluation of the accrual failure detector[C]// Proceeding of the 26th IEEE international on Distributed Computing Systems Workshops (ICDCSW06). USA: Washington, 2006, 32(3):213-221.
- [4] 陈诚. 面向容灾的失效检测技术与实现[D]. 长沙: 国防科学技术大学, 2008.
- [5] 栾兰, 刘淑芬, 张欣佳. 基于检测点失效检测算法的研究与改进[J]. 吉林大学学报: 理学版, 2008, 46(4):681-

686.

- [6] Chen Wei Toueg S, Aguilera M K. On the quality of service of failure detectors[J]. IEEE Transactions on Computers, 2002, 51(5): 561-580.
- [7] Marin Bertier, Olivier Marin. Performance analysis of a hierarchical failure detector[C]// Proceedings of the 2003 International Conference on Dependable Systems and Networks (DSN'03). San Francisco, 2003, 21(3): 241-251.

作者简介:

陈宁江 男, (1975-), 博士, 副教授, CCF 会员. 研究方向为软件工程、网络分布计算等.

李昌武 男, (1982-), 硕士研究生. 研究方向为软件工程、中间件和服务计算等.

俞闽敏 女, (1986-), 硕士研究生. 研究方向为中间件、数据集成等.

(上接第60页)

由图4可知,随着任务最大关联度的增大,及任务之间的联系越复杂,本系统的执行效率较Condor和Pbs都有一定提高.

5 结束语

本文研究了网格环境下关联任务的迁移问题,提出了一种适合关联任务的迁移算法.该算法与传统集群系统的迁移机制相比,执行效率有一定的提高.算法侧重于待迁进程的选择和目的节点的确定,在迁移过程中要不断地访问数据库,提取关联任务的相关信息,尤其是关联任务数量较大时,对数据库的访问会占用一部分迁移的时间开销,因此,如何降低迁移的时间开销是下一步研究的重点.

参考文献:

- [1] 陈廷伟,张斌,郝宽文.基于任务-资源分配图优化选取的网格依赖任务调度[J].计算机研究与发展,2007,44(10):1741-1750.

- [2] 许丽娟.基于计算网络的集群系统负载平衡和进程迁移机制的研究与实现[D].上海:上海大学,2005.

- [3] Ferrai D, Zhou S. An empirical investigation of load indices for load balancing applications[M]. Englewood Cliffs, NJ: Prentice-Hall, 1988.

- [4] Kemal Efe, Margaret A Schaar. Performance of co-scheduling on a network of work stations[C]// USENIX Winter Conference. IEEE, 1993:525-531.

- [5] 陈廷伟,张斌,郝宽文.网格依赖任务调度系统的设计与实现[J].小型微型计算机系统,2008,29(3):455-460.

- [6] 杨永健,孙永雄,李秋秋,等.网格计算中一种负载均衡聚类匹配迁移算法[J].微电子学与计算机,2006,23(10):119-121.

- [7] 许封元,房至一,朱维平.进程迁移对负载平衡影响的实验[J].吉林大学学报:理学版,2006,44(6):925-930.

作者简介:

张千 女, (1982-), 硕士, 讲师, CCF 会员. 研究方向为云计算.