基于MPI的动态负载平衡算法的研究

郭 静,祝永志,王延玲
(曲阜师范大学 计算机科学学院,山东 日照 276826)

摘 要:MPI是目前集群系统中最重要的并行编程工具,它采用消息传递的方式实现并行程序间通信。在MPI并行程序设计中实现负载平衡有着重要的意义,可以减少运行时间,提高MPI并行程序的性能。为了解决同构集群中动态负载均衡问题,提出了一种在MPI并行程序中实现的方法,可有效地根据节点的负载情况在节点间迁移任务。

关键词:动态负载均衡;集群;消息
中图分类号:TP311.5 **文献标识码**:A **文章编号**:1673-629X(2009)05-0150-04

Research on Dynamic Load Balancing Algorithm Based on MPI

GUO Jing, ZHU Yong-zhi, WANG Yan-ling
(College of Computer Science, Qufu Normal University, Rizhao 276826, China)

Abstract: MPI is the most important parallel programming tool in cluster currently. It implements communication in parallel program by message. Implementing load balance in MPI parallel program is very important. It may reduce running time and improve performance of MPI parallel program, aiming at solving the dynamic balancing problem in homogeneous cluster system, proposes an implementing method in MPI parallel program that can transfer tasks between nodes effectively by node's load. The experiments prove the availability and practicability of the algorithm in parallel computing task.

Key words: dynamic load balance; cluster; message

0 引 言

集群系统具有高可扩展性和高可用性等特点,而处理节点间的负载平衡问题是系统获得高性能的主要障碍,也是科研领域的热点之一。负载均衡算法按方式不同有静态和动态之分,静态算法与系统当前状态无关,盲目性大且效率低;动态算法根据系统当前状态平衡节点上的负载,实时性好,实用价值高。许多负载均衡的研究工作围绕异构集群系统展开,但在同构集群系统上的实际应用也很多。

MPI是目前集群系统中最重要的并行编程工具[1],它采用消息传递的方式实现并行程序间通信。MPI具有移植性好、功能强大、效率高等多种优点,而且有多种不同的免费、高效、实用的实现版本,常见的如MPICH、LAM、IBM MPL,几乎所有的并行计算机厂商都提供对MPI的支持,成为了事实上的并行编程标准。在MPI并行程序设计中实现负载平衡有着重要的意义,可以提高MPI并行程序的性能。

文中主要在探讨同构集群系统中的动态负载算法的基础上提出了一种基于MPI的新负载均衡算法。主要研究同构集群的动态负载均衡问题,并提出了一种基于MPI并行程序的任务调度算法,并通过实验证明了可靠性。

1 MPI并行程序概述

MPI是一个消息传递接口的标准,用于开发基于消息传递的并行程序,其目的是为用户提供一个实际可用、可移植、高效灵活的消息传递接口库。MPI采用单程序流多数据流的编程模型,即每个进程执行同样的MPI程序。MPI程序可以通过MPI_Comm_rank函数获取当前进程的编号,有了该编号,不同的进程就可以将自身和其他进程区分开来,各个进程就可以执行不同的任务,从而实现进程间的并行和合作。

MPI为消息传递和相关操作提供了功能强大而又丰富的库函数,其中有上百个甚至几百个函数调用接口,但最常用的6个函数就可以构成的MPI子集,

收稿日期:2008-09-02
基金项目:山东省高等学校实验研究项目基金(2005-400);曲阜师范大学科研项目(XJ0734)
作者简介:郭 静(1981-),女,硕士研究生,主要研究方向为分布式计算;祝永志,教授,硕士生导师,主要研究方向为网络与分布式系统。

实现所有的并行程序功能。

下面用C语言的调用格式描述:

MPI_INIT: 启动MPI计算

MPI_FINALIZE: 结束MPI计算

MPI_COMM_SIZE: 确定进程数

MPI_COMM_RANK: 确定进程标识符

MPI_SEND: 发送一条消息

MPI_RECV: 接受一条消息

用下面的举例表示MPI应用程序的基本结构:

```
#include"mpi.h"
main(argc,argv)
int argc;
char * argv;
{
int count;
int myid
/* 初始化 */
MPI_Init(&argc,&argv); //启动MPI
MPI_Comm_size(MPI_COMM_WORLD,&count) //找出进程的个数
MPI_Comm_rank(MPI_COMM_WORLD,&myid) //指出当通信组中进程标识号
/* 并行计算 */
if(myid=0) //MASTER
{compution and communicating;
if(fetal error happens) MPI-About();}
else //SLAVE
{
computing and communication;
if(fetal error happens)MPI-About();
}
/* 计算结束 */
MPI_Finalize();
}
```

2　动态负载平衡算法中的决策方式

在负载平衡算法的两种决策中,集中式方式专设一台主机(也称中心点)收集各节点负载情况,继而做出全局决策,该方式在理论上可以得到最优效果。但因为中心节点通讯量大,易成为通信瓶颈,所以目前大多数系统不采用该方式;在分布式方式中,各节点只与一定范围内的其他节点交流负载信息,继而做出决策,该方式比较合理,被很多系统采用。

另外,也有使用处理机的负载变化率来对处理机进行动态负载预测[2]。

文中提出的算法采用分布式决策方式。

3　动态负载平衡算法的分类、设计决策和使用的参数

(1)系统的规模。系统中处理器的数目是影响负载平衡决策的一个参数。

(2)转移的负载门限。系统中触发任务转移的负载门限是一个关键参数,因为选择不当会导致系统不平衡和任务转移的连锁反应。

(3)任务大小。一般来说,转移一个运行时间太短的任务是不恰当的。类似的,太大的进程或者涉及到大量数据和文件的进程最好在本地处理器上执行[3]。

(4)负载平衡的视界。一个节点能够在其邻接节点范围内为一个任务寻找可能的目标节点,在其上运行该任务。这个邻接节点范围的直径称为视界(horizon)。这个参数设置了寻找目标节点过程中探查的邻接节点的数量。

4　动态负载均衡算法

4.1　负载状态

为评价节点的负载状态,通常从以下各因素中的一个或综合多个方面考虑负载状态的定义[4]:

(1) CPU队列长度;

(2) CPU利用率;

(3) 可用内存大小;

(4) 上下文切换速度等。

T. Kunz的研究表明[5],采用不同因子,效率差别较大,而最简单的因子,即CPU队列长度,就能获得教高的效率。文中提出的算法采用CPU队列长度作为负载评价因子,并作如下定义:

定义1:同构集群系统中节点的处理能力,即单位时间内CPU能处理的最多进程数用C描述,负载阈值用A描述,通常取C的10%,当前CPU队列中的进程数用Q描述,则节点负载状态S可由如下函数判定:

$$f(Q)=\begin{cases}\text{heavy} & Q>C+A\\ \text{normal} & C-A\leqslant Q\leqslant C+A\\ \text{light} & Q<C-A\end{cases}$$

其中,heavy表示节点处于重载状态,需要迁出部分进程;normal表示节点处于适载状态,既不发送进程到其他节点,也不接受迁移进程;light表示节点处于轻载状态,可以接受来自重载节点的外部迁移进程。

迁移进程满足的时间条件:

Horchol-Balter和Downey[6]认为同构节点间适合迁移的进程的最小运行时间为:

$$AGE_{min}=\frac{f+u/b}{Qi-Vs/Vd\cdot(Qj+1)}$$

其中,Vs和Vd分别表示迁出进程的源节点和接受进

程的目标节点的处理速度;Qi 表示迁移前源节点上的进程数,Qj 为迁移前目标节点上的进程数;进程迁移代价为 f + u/b,f 为抢占式迁移的固定迁移代价,u 为迁移进程的内存开销,b 为内存传输带宽。

4.2 文中提出的任务调度算法

在并行计算中多个任务可能在同一时间运行,因此使用怎样的任务调度策略就显的非常重要。文中提出的任务调度策略思想是:在 MPI 并行程序执行过程中根据各计算节点的当前负荷,在计算节点间动态迁移任务,将负载重的节点上的任务迁移到负载轻的节点上,尽量平衡各计算节点的负载,从而减少程序运行时间。但动态负载平衡在实现上比较复杂,需要一些额外的通信代价。

一般来说,动态负载平衡需要解决以下三个问题[7]:

a. 如何收集系统当前负载的信息;

b. 决定是否进行任务迁移以及向何处迁移;

c. 如何进行迁移。

对应地,动态负载平衡方法也应该包含三个部分:

1) 信息收集:制定衡量节点负载信息的负载指标,以及信息收集的方式;

2) 迁移决策:基于任务和节点负载,判断是否要把一个任务迁移到其它计算节点上运行;

3) 迁移执行:对于适合迁移到其它计算节点上的任务,选择任务迁移的目的节点。

上述三个部分之间是以不同的方式相互作用的,"迁移执行"利用"信息收集"提供的负载信息,仅当任务被"迁移决策"判断为适合迁移之后才行动。因此,对于动态负载平衡,需要一个进程收集各个节点的负载信息,确定哪个节点需要进行任务迁移,以及向哪个节点迁移,此进程称为调度进程。在 MPI 并行程序中,可以让编号为 0 的进程作为调度进程,其他进程作为计算进程。因为调度进程相对计算量较少,所以可以让调度进程与某个计算进程运行在同一个计算节点上。具体实现方法如下:

(1) 每个计算进程完成一个任务,发送消息通知调度进程其剩余任务数 Q。

if(Q>C+A)

{节点状态函数 f(Q) = heavy;等待调度进程的消息。迁移出任务数为(Q-C-A)/2 ;}

else if(Q<C-A)

{节点状态函数 f(Q) = light;等待其他计算进程的任务迁移消息。迁移进任务数为[(C-A-Q)/2, (C+A-Q)/2];}

else if(C-A≤Q≤C+A)

{节点状态函数 f(Q) = normoal;继续执行}

(2) 调度进程接受计算进程 i 的消息。得到计算进程 i 的当前剩余的任务数 Qi,迁移进程所需要的时间为 Ti。

if(f(Qi) = light)

{登记此进程为空闲进程,可以迁进某些任务;}

else if(f(Qi) = heavy) && f(Qj) = light && Ti≤AGE_{min})

{将计算进程 i 的任务的一半迁移到计算进程 j 中}

else if(f(Qi) = normal)

{继续执行;}

(3) 空闲进程 i 接受调度进程发送过来的计算进程的编号 j 的进程。

if (f(Qj) = light) && Ti≤AGE_{min})

{调度进程向计算进程 j 发送任务迁移消息,迁移任务数为[(C-A-Q)/2 ,(C+A-Q)/2]}

else if (f(Qj) = normal)

{继续等待;}

(4) 若任意一个计算进程 K 对应的负载状态函数 f(Qk) = light,则并行程结束。

5 实验及性能分析

5.1 实验环境

实验环境是由 10 台 PC 组成,软件和硬件配置如下:

1) 每台 PC 机的硬件配置如下:

CPU: Intel(R) Pentium(R) 4 CPU 3.20GHz 3.20GHz

主板: 华硕 A7V8X

内存: 256MB

硬盘: 80GB

2) 网络配置:

交换机:16 口 100Mb/s

网卡: 100Mbps 自适应网卡×16

超五类双绞网络线缆、接头:适量

3) 每台 PC 机的软件配置如下:

操作系统:Linux RadHat9.0

并行实验环境:MPICH

5.2 实验数据

实验采用典型的表序问题为例,表序问题指表中每个元素 k 指定一个它在表列中的次第号(rank(k)),rank(k)可看作元素 k 到表尾的距离。为此每个元素 k 都为一个指向下一个元素的指针 next(k)。如果 k 是表中最后一个元素,则 next(k) = k。具体算法如下:

Begin

(1) for all k∈L par-do

(1.1) P(k) = next(k)

(1.2) if P(k)≠k then distance(k) = 1

```
        else distance(k) = 0
        end if
    end for
(2) repeat⌈logn⌉times
    (2.1) for all k∈L par - do
          if P(k)≠P(P(k)) then
          (I) distance(k) = distance(k) +
           distance(P(k))
          (II) P(k) = P(P(k))
          end if
    end for
    (2.2) for all k∈L par - do
          rank(k) = distance(k)
        end for
    end repeat
end
```

利用上述并行程序所得实验数据如下:

(1) 当数据规模固定,例如:表中元素有 200 个,处理机不同时运行时间见表 1。

表 1　数据规模固定时 DNBLI 和 NBLB 的运行时间

处理机数	DNBLI 的运行时间(秒)	NBLB 的运行时间(秒)
1	4.407	4.109
2	3.055	2.850
4	1.976	1.401
6	1.230	0.985
8	1.089	0.889
10	0.865	0.791

(2) 当系统规模固定时,例如:10 台 PC 在不同数据规模不同是所得运行结果见表 2。

表 2　系统规模固定时 DNBLI 和 NBLB 的运行时间

数据规模	DNBLI 的运行时间(秒)	NBLB 的运行时间(秒)
50	0.237	0.221
100	0.656	0.561
150	0.837	0.781
200	0.955	0.823
250	1.236	1.056
300	2.245	2.012
350	3.656	3.321
400	5.213	4.981

根据表 1,2 可以得到如下曲线图(见图 1、图 2)。

5.3　数据分析

(1) 当数据规模固定时,如图 1 所示,NBLB 的时间优势比较明显,这说明随着系统规模的增大,NBLB 能得到更多的时间效益。

(2) 当数据规模较小时,各算法的时间开销差别不大;当规模逐渐增大时,MBDBA 比其它算法表现出更好的时间性能,如图 2 所示。

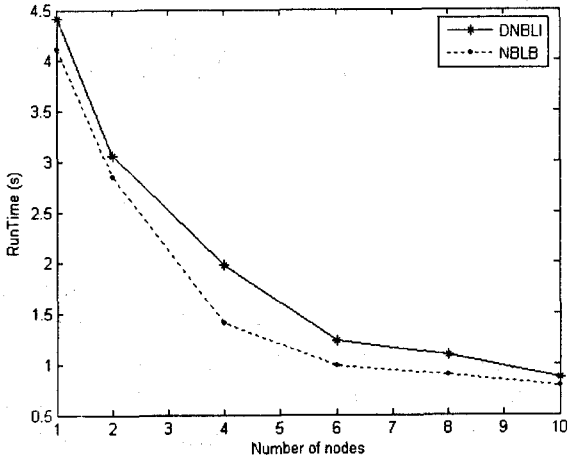

图 1　数据规模固定时 DNBLI 和 NBLB 的运行时间对比

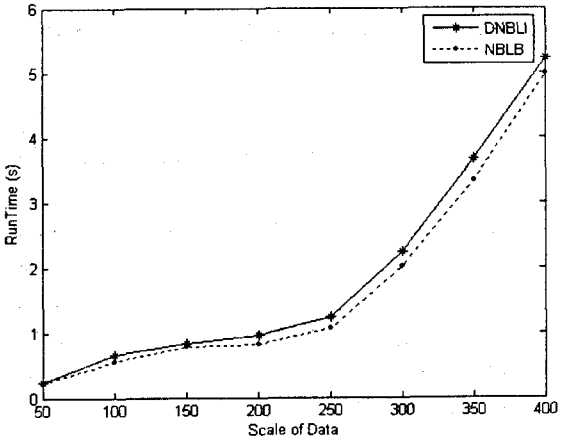

图 2　系统规模固定时 DNBLI 和 NBLB 的运行时间对比

注:DNBLI 是由 Franco、Zambonelli[8] 提出的动态负载算法,NBLB 是文中提出的任务调度算法。

6　结束语

对集群系统的动态负载均衡算法进行了研究,提出了一种任务调度策略,通过实验与 Franco、Zambonelli 提出的动态负载算法 DNBLI 进行比较,证明具有可用性和更高时间效益。

参考文献:

[1] 祝永志,赵　岩,魏榕晖.基于 MPICH 的 Beowulf 集群系统构建与性能评测[J].计算机工程与应用,2006(14):132 - 133.

[2] 杨永健,鞠九滨,曹晓东.基于非完全信任确认机制的负载均衡预测方法[J].计算机工程,2006,32(7):109 - 111.

[3] 陈国良.并行计算——结构·算法·编程[M].北京:高等教育出版社,2004:104 - 109.

[4] 刘　滨,石　峰.基于消息传递机制的动态负载平衡算法

(下转第 191 页)

间周期 T 的时间间隔机制。测量数据分析程序在数据存储模块中,按照采集时间大于前次分析时间与采集时间小于或等于当前分析时间作为查询条件,确定是否在两个时间点中测量系统取得了新的测量数据。如果没有取得新数据则表明 DHT 路由表中信息可能在进行更新,系统测量过程已经被禁用,因此,对应的分析过程将停止,如果取得新数据则表明系统稳定的按照周期 T 在对网络状况进行测量,因此,对应的分析过程启动。测量数据分析程序在完成分析后将分析结果回存入数据存储模块,用于数据发布层进行查询与发布。

3.3 数据发布层

测量系统的性能数据发布接口主要是实现请求-回应机制,其主要是作为与 DHT 网络中其他节点查询其中一个节点所维护的性能数据的接口。在具体实现时使用消息机制在节点间进行通讯,消息定义出所进行的操作以及操作所针对的 PT 类型,这里的操作主要是查询数据,包括查询性能数据、测量数据以及数据条数。其运作过程为:请求查询的 DHT 节点向网络中被查询节点发送一个包含 PT 类型的查询消息;被查询节点接收到上述消息后,根据查询的性能类别与数据条数,调用数据查询函数对性能进行查询;查询函数返回最近 n 次的数据,其中 n 为查询请求节点发送的数据条数;被查询节点最后将查询所得数据返回给请求节点。

4 结束语

在深入研究 P2P 技术和网络测量领域的已有研究成果的基础上,设计与实现了架构在 DHT 层上的针对 DHT 网络的三层架构网络测量系统。该系统能对 DHT 网络中节点间网络性能以及节点自身性能进行测量、存储、分析和发布,为 DHT 网络的优化利用提供决策支持。测量系统参考 GMA 系统架构,具体设计与实现了如下内容:设计提取 DHT 节点所维护的 DHT 路由表中节点信息的方式;对测量工具的调用与管理方式;测量采样过程的触发与控制机制;测量数据的存储系统;测量性能的发布机制。

在现有的 P2P 系统特别是 P2P 文件分享系统中存在多对多的通讯与服务情况,单纯考虑 DHT 节点间一对一的通讯与测量还存在不足。基于小世界理论的 DHT 思想保证了查找的可达性问题,但长链路上的性能不是其关注重点,如何综合考虑保证整条长链路上的性能也将是需要进一步考虑的问题。同时,伴随着测量工具的引入,测量分析程序、系统内的时间控制与触发机制都有改进空间。

参考文献:

[1] 罗杰文. Peer-to-Peer 综述[EB/OL]. 2006-08-25. http://www.intsci.ac.cn/users/luojw/P2P/index.html.

[2] Stoica I, Morris R, Karger D. Chord: a scalable peer-to-peer lookup service for internet applications[C]//Proceedings of ACM SIGCOMM 2001. San Deigo, CA:[s.n.], 2001: 149-160.

[3] Ratansamy S, Francis P, Handley M, et al. A scalable content-addressable network[C]// Proceedings of ACM SIGCOMM 2001. San Deigo,CA:[s.n.], 2001:161-172.

[4] Rowstron A, Druschel P. Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems [C]//IFIP/ACM International Conference on Distirbuted Systems Platforms (Middleware). Heidelberg:[s.n.], 2001: 329-350.

[5] Zhao Y B, Huang L, Stribling J, et al. Tapestry: a resilient global-scale overlay for service deployment[J]. IEEE Journal on Selected Areas in Communications,2004,22(1):41-53.

[6] Gribble S D, Brewer E A, Hellerstein J M, et al. Scalable, Distributed Data Structures for Internet Service Construction [C]//In Proceedings of OSDI 2000. San Diego:[s.n.], 2000:68-76.

[7] 刘　琼,徐　鹏,杨海涛,等. Peer-to-Peer 文件共享系统的测量研究[J]. 软件学报, 2006,17(10):2131-2140.

[8] Paxson V, Mahdavi J, Adams A, et al. An Architecture for Large-Scale Internet Measurement[J]. In IEEE Communications,1998,36(8):48-54.

[9] Tierney, Aydt R, Gunter D, et al. A Grid Monitoring Architecture[EB/OL]. 2002-01-16. http://www-didc.lbl.gov/GGFPERF/GMA-WG/papers/GWD-GP-16-2.

[10] Latter T, Kutzko M, Engelhardt S, et al. Network Performance Advisor[EB/OL]. 2005-09. http://dast.nlanr.net/projects/advisor/.

(上接第 153 页)

研究[J]. 计算机工程,2007(5):58-60.

[5] Kunz T. The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme[J]. IEEE Trans. on Software Eng,1991,17(7):725-730.

[6] Balter M H, Downey A B. Exploiting Process Lifetime Distributions for Dynamic Load Balancing[J]. ACM Transactions on Computer Systems,1997,15(3):253-285.

[7] 陆克中,林晓辉. MPI 并行程序设计的负载平衡实现方法[J]. 微计算机信息,2007(3):226-227.

[8] Zambonelli F. Exploiting Biased Load Information in Direct-neighbour Load Balancing Policies[J]. Parallel Computing, 1999,25(6):745-766.