异构 Beowulf 系统负载均衡技术的研究与实现

李丙锋,祝永志,魏榕晖
(曲阜师范大学 计算机科学学院,山东 日照 276826)

摘 要:负载均衡技术是并行计算系统的关键技术之一,其主要思想是将计算任务合理分配到各节点,以避免由于某个节点的计算速度慢而导致的整个系统的性能瓶颈。为了使系统达到更优的结果,提出了一种基于 MPICH 的负载均衡模型,构建了一个异构 Beowulf 并行计算系统,并且用一个适合异构环境的矩阵分块相乘算法进行了性能测试。实验结果表明,此算法比使用传统的矩阵相乘算法执行效率更高。从而证明了系统能够把计算任务高效合理地分配到各个节点。

关键词:Beowulf;负载均衡;MPICH;矩阵分块相乘
中图分类号:TP311.1 **文献标识码**:A **文章编号**:1673-629X(2008)07-0060-03

Implementation of Load Balancing Technology on Heterogeneous Beowulf System

LI Bing-feng, ZHU Yong-zhi, WEI Rong-hui
(College of Computer Science, Qufu Normal University, Rizhao 276826, China)

Abstract: Load balancing technology is one of the key technologies of the parallel computing system, and its main idea is computing tasks will be assigned to each node reasonably, to avoid the bottleneck of the whole system due to a node of slow. To enable the system to better results, presents an MPICH-based load balancing model of a heterogeneous Beowulf parallel computing systems, and with a heterogeneous environment suitable for the block matrix multiplication algorithm was tested. Experimental results show that the algorithm is more efficient than traditional matrix multiplication algorithm implementation. To prove that the system can assign tasks to each nodes reasonably.

Key words: Beowulf; load balancing; MPICH; matrix block multiplication

0 引 言

负载均衡(Load Balancing)是并行计算一个重要研究领域。在异构集群系统中,由于节点类型和资源类型的多样性,负载均衡问题就更为复杂。

所谓异构集群系统,是由高速互连网络连接起来的多台不同类型的计算机组成的高性能计算系统。系统中各个节点性能不同,甚至体系结构也不相同。异构并行计算与一般多机系统中的并行任务调度最主要的不同之处在于要对所分配的节点进行选择(Machine Selection),挑选一个最能满足该任务的计算要求的节点[1]。

集群系统的异构性包含两个方面:类型异构和资源异构。类型异构是指计算节点具有不同的指令集结构(ISA)和不同操作系统(OS)类型;资源异构是指计算节点的资源(主要为 CPU 资源、内存和 I/O 资源)拥有量的不同[2]。文中主要研究资源异构。

1 基于 MPICH 的并行计算负载均衡模型

MPI(Message Passing Interface)是当前最流行的消息传递并行程序设计的标准之一[3]。用 MPI 编写程序时,负载均衡是程序员必须考虑的问题。负载均衡的主要方法是将计算任务按节点的处理能力合理分配到各节点。如何分配,往往是困扰开发人员的问题。总括而言,有两种方案可供选择:静态分配和动态分配。

静态任务分配一般采用二次均分法,这种方法是将所有的总任务首先按照参与计算的节点数量进行均分,然后将不能整除的任务数再一次分配给各节点,以保证各节点的工作量大致一样。这种方法的思想和实现代码都比较简单,易于实现,比较适用于一般同构的并行计算环境。但是这种方法的缺点是显而易见的,

收稿日期:2007-10-30
基金项目:山东省高等学校实验研究项目基金(2005-400)
作者简介:李丙锋(1979-),男,山东临清人,硕士研究生,研究方向为分布式计算;祝永志,教授,硕士生导师,研究方向为网络与分布式系统。

即如果参与计算的各节点能力相差较大,则计算时间符合木桶原则,取决于最慢那台计算机的处理时间,即使其他较快的节点处理完数据,仍然要等待最慢的机子返回数据才能得到最终结果。这样使得并行数据处理的加速比大大降低。

最好的方案是在分配任务前,服务节点能够知道哪个节点运算能力强一些,就把多一点的任务分配给它,同时将相对较少的任务分配给性能低一点的节点。真正做到动态分配,即任务的分配不是固定的,是根据节点的不同而不同,以达到最优的数据处理效果。

文中建立了一个负载均衡模型。利用一个 Client/Server 程序[4],Client 程序位于各个节点上,Server 位于服务节点上。首先,由服务节点调用各节点的 Client 程序取得每个节点的性能参数,服务节点将各个节点按照综合处理能力排序,然后将任务按性能比值分配给每个节点。

此模型主要分为三大模块:Client 模块、Sched 模块、Server 模块(如图 1 所示)。

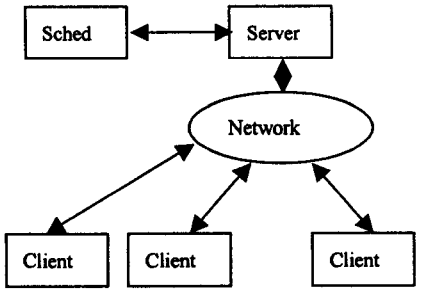

图 1　模型结构图

(1)Client 模块。

Client 程序的作用在于调用相应的 API,取得每个节点的性能参数(CPU 主频、内存大小、响应时间等),并将性能参数传回给服务节点。

(2)Sched 模块。

进行任务分配,并根据调度算法实现资源分配[5]。算法描述如下:

a.权重计算,计算节点的权重主要考虑以下参数:CPU 资源、内存资源和响应时间,节点权重的计算公式如下:

$$w_i = R_1 \times K_i \times L_{CPU}(p_i) + R_2 \times \frac{L_M(p_i)}{BASE_M} + R_3 \times \frac{BASE_{Resp}}{L_{Resp}(p_i)}$$

其中 $L_f(p_i)$ 表示节点 p_i 当前某一参数值在以上公式中依次表示:CPU 利用率、可用内存数和节点响应时间。K_i 为处理器系数(根据节点的处理器类型及数量确定)。$BASE_f$ 为某一参数的基准值,依次为基准内存数和基准响应时间,这些基准值可以根据集群系统的实际情况统计确定。R_i 是为每个参数设定的一个可调常量,用以区分各个参数的重要程度,其中 $\sum R_i = 1$。由于对于不同类型的应用系统各个参数的重要程度是不同的,在使用时可根据具体的应用对各参数所占的比重进行调节。

b.按节点权重从大到小的顺序排列。

c.采用加权轮转法对用户请求进行调度,即根据节点权重的大小按照轮转的方式将用户请求分配到各节点去,权重越高的节点分配到的任务就越多。每个节点所分配到的任务数是按其权重占权重总和的比例来确定的。

(3)Server 模块。

Server 模块部署于服务节点上,主要功能是负责各种信息的处理和反馈;新资源加入、资源失效等;负责 MPICH 作业和串行作业的提交、完成和取消。

(4)系统运行流程。

系统启动后,部署于各计算节点上的 Client 模块作为守护进程始终在后台运行。用户递交作业,Server 接收到控制台请求后将作业放入队列,然后调用 Sched 模块,Sched 模块向各节点广播一个获取节点性能参数的 Socket 消息,得到各种参数后,将各处理节点按综合处理能力权重排序,根据各节点的负载情况对作业进行任务分配,再由 Server 程序把各个子任务发送给各节点并开始计算,计算结果最终发送给 Server。

程序使用 C++ 调用 MPICH 库函数开发[6,7]。开发工具采用 Redhat Linux9.0 下的 KDeveloper。

2　矩阵相乘算法

矩阵运算是数值计算中最重要的一类运算,特别是在线性代数和数值分析中,它是一种最基本的运算。一个 $m \times n$ 阶矩阵 $\boldsymbol{A} = [a_{ij}]$ 乘以一个 $n \times k$ 的矩阵 $\boldsymbol{B} = [b_{ij}]$ 就可以得到一个 $m \times k$ 的矩阵 $\boldsymbol{C} = [c_{ij}]$,它的元素 c_{ij} 为 $\boldsymbol{A}$ 的第 i 行向量与 $\boldsymbol{B}$ 的第 j 列向量的内积。常用的算法有分块矩阵乘法(Block Matrix Multiplication),适当旋转矩阵元素的方法(如 Cannon 乘法),或采用适当复制矩阵元素的办法(如 DNS 算法),或采用流水线的办法使元素下标对准[8]。

上述算法适用于同构的并行计算系统,只有在各个处理节点的 CPU 处理能力、主存大小相当,并且节点的硬件结构、操作系统相同的情况下才能得到较高的加速比和效率。文中使用了简单的粗粒度的矩阵分块相乘算法,以适用于异构集群系统。

算法思想如下:一个 $m \times n$ 阶矩阵 $\boldsymbol{A}$ 乘以一个 $n \times k$ 的矩阵 $\boldsymbol{B}$,对 $\boldsymbol{A}$ 按行进行分块,分为 p 块(p 为处理器个数,即节点个数),根据上述 Server 服务节点提供的

各个节点的综合处理能力权重进行分配，设共有 p 台节点，权重比值为 $w_1 : w_2 : \cdots : w_p$，则对 A 进行如下分块：

A_1 有 $R_1 = (m/w_1 + \cdots w_p) \times w_1$ 行

A_2 有 $R_2 = (m/w_1 + \cdots w_p) \times w_2$ 行

…

A_p 有 $R_p = (m/w_1 + \cdots w_p) \times w_p$ 行

对 B 进行相应的列分块，分别有 $L_1, L_2, \cdots, L_p$ 行。

第 i 次计算时，处理器并行计算 $C_{ij} = A_i \times B_j$，其中 $i, j = 1,2,\cdots,p$。比如，第一次运算时，各处理器的存储内容如表 1 所示。此时各处理器并行计算 $C_{1j} = A_1 \times B_j$，其中 $i = 1,2,\cdots,p$。

表 1 第一次运算时处理器存储内容

处理器编号	存储内容	
p_1	A_1	B_1
p_2	A_2	B_2
…	…	…
p_p	A_p	B_p

由于对 A 的分块是按照各处理节点的权重大小按比例分配的，所以能够把计算负载均匀合理地分配到各个节点。设一次乘法和加法运算时间为一个单位时间，经计算，$A_i \times B_1, A_i \times B_2, \cdots, A_i \times B_p$ 的计算时间比为 $R_i \times n \times L_1 : R_i \times n \times L_2 : R_i \times n \times L_p = w_1 : w_2 : \cdots : w_p$。因此各个节点内每次分块相乘所花费的计算时间之比，正好等于事先得到的各个处理节点的权重比值。

3 性能测试与分析

为体现系统的异构性，实验采用四种不同配置普通微机共十六台，主要配置情况如表 2 所示，其中配置最高的一台机器兼做服务节点和计算节点。都采用 RTL 8139 100M 网卡，网络互连使用 Cisco2600 交换机，操作系统为 Redhat Linux9.0，并行计算环境为 MPICH2－1.0.5。

表 2 实验机器配置情况

类型	CPU	内存
1	赛扬 900MHz	128M
2	赛扬 1.7GHz	256M
3	奔腾 2.0GHz	256M
4	奔腾 3.0GHz	512G

为了测试负载均衡模型以及算法的有效性，用两个 1000 阶的矩阵相乘，进行了三次实验：

(1)不使用负载均衡算法。按照第 3 部分的算法直接编写程序，分别把矩阵分为 $p = 1,2,\cdots,16$ 块（p 为节点个数）。

(2) 使用负载均衡算法。分别把矩阵分为 $p = 1, 2,\cdots,16$ 块，把计算任务使用负载均衡算法自动把大小不同的任务分配到各个节点。

(3) 使用传统的矩阵分块乘法。分别把矩阵分为 $p = 1,2,\cdots,16$ 块。

实验依次用 1,2,…,16 台机器分别按上述三种方法进行实验，配置高的机器优先选用，实验结果如图 2 所示。

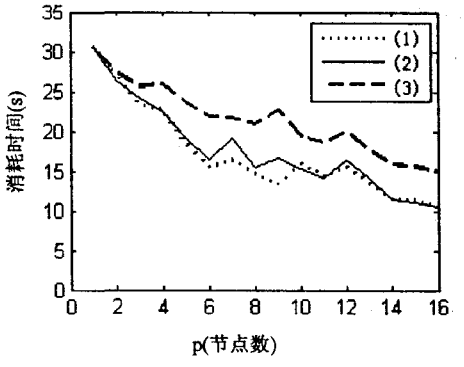

图 2 实验结果

节点数为 1 时，由于没有并行性，所以花费时间相等。节点数增多时，花费时间逐渐减少。由于(1)是直接编写程序进行负载分配的，所以花费的时间最少；(2)采用了负载均衡算法，花费的时间和(1)相差不多，这说明负载均衡算法能够把不同大小的任务高效合理地分配到各个计算节点；而(3)是把大小不同的任务平均分配到处理能力不同的各个计算节点，所以花费的时间更多。

4 结束语

对基于 MPICH 的负载均衡技术进行了讨论，提出的这种 Client/Server 的模型，简单且适用于多台异构计算机进行并行计算。并使用了适合异构环境下矩阵相乘算法对系统进行了性能测试，从运行效果看，达到了预期的目的。

参考文献：

[1] 陆鑫达. 异构并行计算及其调度策略[J]. 计算机工程，1997,23(特刊):37－39.

[2] 蒋 江,张民选,廖湘科. 异构集群系统中一种基于资源的负载平衡算法的设计与模拟[J]. 小型微型计算机系统，2003,24(4):626－627.

[3] Group W. A high performance portable implementation of the message passing interface[J]. Journal of Parallel Computing，1996,22(6):789－828.

[4] 陈华平,计永昶,陈国良. 分布式动态负载平衡调度的一个通用模型[J]. 软件学报，1998,9(1):25－29.

(下转第 65 页)

文中提出的服务发现框架中的服务与服务请求都是由OWL-S描述的,从而实现基于语义的Web服务的发现成为可能。

Web服务的发现过程实质上就是服务请求者从众多Web服务中找到满足其需求的服务过程。首先,提取每个用户请求合约输入输出参量、服务非功能参量,与已有Web服务描述相比较,即进行服务相似度计算,然后根据相似度进行排序,最后按相似度高低的顺序返回给用户[9]。

基于语义的服务发现算法主要包含两个步骤:首先基于功能的服务匹配,计算机服务相似度,得到语义上符合用户需求的所有候选服务。文中服务匹配的服务相似度计算方法similation()主要采用文献[10]中的参量相似度计算方法。参量相似度对可供度量的参量,如输入参量、输出参量、服务前置条件参量和服务结果参量等参数进行相似度计算。这些参数的相似度可以从语义、类型和数值上进行比较;然后基于非功能的服务匹配,在候选服务中根据用户的服务请求参数QOS与服务的非功能描述的匹配,找出最能满足用户请求的有序服务集。

```
match (WSreq,services[])|
for each service[i] in services[] do
  if(numOfInputs( WSreq) ! = numOfInputs(service[i]) then
    return NULL;
for each service[i]in service[] do
  find some service[i] in service with
    similation(Wsreq_input, service[i]_input)>=min1 and
    similation(Wsreq_output, service[i]_output)>=min2
  endfind;
  rank[]=service[i];
endfor
for each service[j] in rank[] do
    if(similation(WSreq_QOS,service[j]_QOS)<min3) then
      delete service[j] from rank[];
    endif;
  endfor;  sort(rank[]);
  return(rank[]);
}
```

其中,min1、min2和min3分别是输入相似度、输出相似度和QOS相似度的最低值,可由用户根据实际情况来定义。

4　结束语

Web服务的大量涌现对服务发现提出了挑战。目前基于关键字和基于框架的服务搜索机制已经不能很好地满足用户需要。在分析相关研究的基础上,提出了一个基于语义的Web服务发现框架,根据服务提供的语义选取适合的Web服务,并给出了相应的算法,从而提高查准率。

今后,将进一步完善与实现该框架与算法,达到Web服务发现的自动化。

参考文献:

[1] Berardi D. Automatic Service Composition Models, Techniques and Tools[D]. Roma: Roma University, 2005.

[2] Brainz S M. A Semantic Web Services[J]. IEEE Intelligent System, 2002,17(1):76-77.

[3] Gonzalez C J, Tastour D, Bartolini C. Description logics for matchmaking of services[C]//IN: Proceeding soft Workshop on Application of Description Logic. Vienna, Austria: [s. n.], 2001:581-586.

[4] 张　钋.基于语义的网络服务匹配机制的研究与实现[D].北京:清华大学,2005.

[5] Papazoglou M P, Georgakopoulos D. Service-Oriented Computing[J]. Service Oriented Computing Communications of the ACM, 2003,46(10): 25-28.

[6] 黄　莺.信息服务描述的方法与语言[J].现代图书情报技术,2005(5):53-56.

[7] McGuinness. Web Ontology Language (OWL)[EB/OL]. Web-Ontology (WebOnt) Working Group[EB/OL]. 2002. http://www.w3.org/2001/sw/WebOnt/, 2002.

[8] Payne T R, Paolucci M, Sycara K. Advertising and matching DAML Service descriptions[C]//In: SWWS. Amsterdam: IOS Press, 2001:411-430.

[9] Xu Bin, Zhang Po, Li Juan-Zi, et al. A semantic Matchmaker for Ranking Web Services[J]. Journal of Computer Science & Technology, 2006,21(4):574-581.

[10] 吴　健,吴朝晖,李　莹,等.基于本体论和词汇语义相似度的web服务发现[J].计算机学报,2005,28(4):595-602.

(上接第62页)

[5] Willebeek-LeMair M H, Reeves A P. Strategies for dynamic load balancing on highly parallel computer[J]. IEEE Transactions on Parallel and Distributed System, 1993,4(9):979-993.

[6] 奎　因.并行程序设计C、MPI与OpenMP[M].北京:清华大学出版社,2005.

[7] Gropp W. Using MPI-2: Advanced Features of the Message-Passing Interface[M]. [s.l.]: MIT Press, 1999.

[8] 陈国良.并行算法实践[M].北京:高等教育出版社,2004.