第35卷 第21期 计 算 机 工 程 2009年11月
Vol.35 No.21 Computer Engineering November 2009

·开发研究与设计技术· 文章编号：1000—3428(2009)21—0264—03 文献标识码：A 中图分类号：TP393

基于.NET Remoting的动态负载平衡模型

谢红薇，吉妙通

(太原理工大学计算机与软件学院，太原 030024)

摘 要：在对.NET Remoting技术和传统的分布式负载平衡算法深入研究的基础上，提出一种排序队列和哈希映射矩阵相结合的负载平衡策略，给出一个分布式环境下的动态负载平衡模型。一方面可提高系统吞吐量，另一方面可缩短任务请求的响应时间。模型采用模块化设计方法，使其具有部署灵活性和容错性，并应用滑动窗口机制提高模型的负载平衡指标可信度。

关键词：.NET Remoting技术；分布式；负载平衡；排序队列；哈希映射

Dynamic Load Balancing Model Based on .NET Remoting

XIE Hong-wei, JI Miao-tong

(College of Computer and Software, Taiyuan University of Technology, Taiyuan 030024)

【Abstract】Based on intensive study of .NET Remoting technology and traditional distributed load balancing algorithms, this paper proposes a load-balancing algorithm of combining sequence queue and hash mapping matrix, gives a dynamic load balancing model based on distributed environment, increases system throughput, and shortens the response time of task requests. The model is modular in design, enhancing its deployment flexibility and fault-tolerant. Sliding window mechanism improves the load-balancing index's credibility.

【Key words】.NET Remoting technology; distributed; load balancing; sequence queue; hash mapping

1 概述

分布式环境已经成为并行计算及分布式服务的一个非常重要的平台，随着分布式应用需求的迅速增长，原有的C/S, B/S模式和技术已经不能胜任。在此背景下，Microsoft推出了.NET Remoting技术，.NET Remoting 是Microsoft .NET进行分布式应用开发的面向对象体系，是传统DCOM技术的替代，它是建立在公共语言运行时(Common Language Runtime, CLR)之上的进程间的通信，较DCOM更容易实现和配置，安全性更高，支持更多的通信协议。.NET Remoting还支持HTTP, SOAP, Web服务描述语言(WSDL)以及XML此类的开放标准[1]，从而达到了协同工作的目标，是实现分布式计算和分布式系统的一种理想技术。分布式计算的最大优势是将负载分布，因此，负载平衡便成为这类系统实现的关键技术之一，在不同的环境和限制条件下，负载平衡机制和策略有很大不同，其中的许多问题都是理论和实践上具有挑战性的难点。

文献[2]提出了一种自适应负载均衡算法，它通过降低集群中每个CPU发送、接收的消息数量来减少负载平衡开销，即D-R算法。对以计算为主的任务来说，此算法效率比较高，但由于只是通过对CPU信息的收集来进行判断，因此局限了它的使用范围。文献[3]提出将遗传算法和阈值策略、信息交换原则相结合的动态负载平衡模型，在调度策略上采用了集中式控制的方法，其缺点是一旦主控计算机失效，将造成整个系统的崩溃。文献[4-5]中提出的负载平衡模型使用有向无环图(Directed Acyclic Graphs, DAG)描述并行任务，根据DAG划分任务粒度，而采用紧耦合的结构使得模型缺乏灵活性。本文在深入研究各种负载平衡策略和模型优缺点的基础上，结合.NET Remoting技术，提出一个新的动态负载平衡模型。

2 .NET Remoting技术概述

.NET Remoting依托于.NET平台，位于.NET框架(.NET Framework)第2层公共语言运行时之上，.NET Remoting提供了一种允许一个应用程序域(Application Domain)中的对象与另一个应用程序域中的对象进行交互的框架。应用程序域是.NET框架中一个重要技术改进，减轻了运行应用程序的系统开销，它将不同应用程序进行隔离，但同时还需要能彼此通信。.NET Remoting体系结构如图1所示[1]。

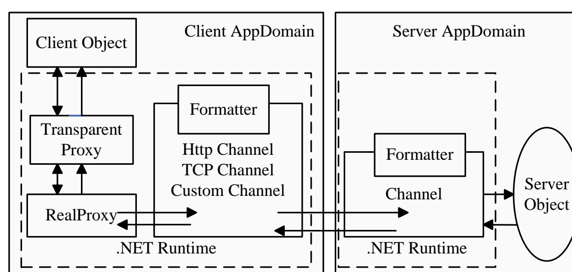

图1 .NET Remoting体系结构

在图1中，服务端应用程序域(Server AppDomain)由传输通道、序列化格式程序和Server Object组成；客户端应用程序域(Client AppDomain)由Client Object、代理、传输通道和序列化格式程序组成。

3 模型负载平衡策略

负载平衡策略设计的好坏直接决定了分布式系统在负载

基金项目：山西省自然科学基金资助项目(20051035)

作者简介：谢红薇(1962－)，女，教授、博士研究生，主研方向：人工智能，并行计算；吉妙通，硕士研究生

收稿日期：2009-04-25 **E-mail：**xiehongwei@tyut.edu.cn

平衡上的表现。基本的负载平衡策略主要有：轮询法，最小连接法，散列法和加权法，其各有优缺点，实际应用中几种策略往往需要结合使用。

3.1 负载信息收集策略

对于负载指标，有多种不同的选择，为适应不同的任务类型，选取 $procUtil \times a + userUtil \times b + memUtil \times c + diskPer \times d$ 作为负载平衡的综合指标($a+b+c+d=1$)，a, b, c, d 根据不同任务类型设置不同的权重 *weight*。各参数说明如下：

(1)*procUtil* 为 CPU 在内核模式(Kernel Mode)下的利用率。

(2)*userUti* 为 CPU 在用户模式(User Mode)下的利用率。

(3)*memUtil* 为内存利用率。

(4)*DiskPer* 为磁盘对某一请求数据读写完成所消耗的时间百分比。

负载信息的收集采用阶段性信息交互原则，同时结合本模型自身特点，加入负载信息处理机制：设定一个滑动窗口及窗口上限容量。避免所收集的负载信息的不可靠性，防止发生“抖动”。

表 1 是一台计算机在一段时间内其 CPU 利用率的数据报告(这里设定其窗口最大容量为 5)。初始状态滑动窗口没有数据，随着时间变化，所存储负载数据增多，但在滑动窗口容量没有达到上限时并不计算其平均负载，在时间为 15 s 时，滑动窗口容量达到上限，此时进行结果汇总：

(62+30+23+25+26)/5=33.2

表 1　滑动窗口机制

时间/s	滑动窗口(max=5)	负载量计算
0	[0]	0=0
3	[26]	0+26=26
6	[25 26]	25+26=51
9	[23 25 26]	23+25+26=74
12	[30 23 25 26]	30+23+25+26=104
15	[62 30 23 25 26]	62+30+23+25+26=166
18	[18 62 30 23 25]	18+62+30+23+25=158
21	[35 18 62 30 23]	35+18+62+30+23=168

负载总量除以滑动窗口容量上限值得到其平均负载后将结果进行汇报。在下一时刻，新的负载 18 进入，从而将 26，即时间为 3 时进入的负载数据从窗口中删除，此时计算结果为

(18+62+30+23+25)/5=31.6

滑动窗口机制的应用提高了负载信息报告的真实性和可靠性。

对于单个节点机上所收集的负载计算步骤如下。

(1)计算每个负载指标参数所对应采集的数据总和：

$$sum_{Vi} = \sum_{j=1}^{m} Vi \quad (1)$$

其中，*Vi* 为负载收集所采集的负载指标参数：*procUtil*, *userUtil*, *memUtil*, *diskPer*；*m* 为滑动窗口容量的最大值，$1 \le i \le 4$，$1 \le j \le m$。

(2)计算每个负载指标参数所对应采集的数据平均负载：

$$WA_{Vi} = sum_{_{Vi}} / m \times weight_{Vi} = ((\sum_{j=1}^{m} Vi)/m) \times weight_{Vi} \quad (2)$$

其中，$\sum_{i=1}^{n} weight_{Vi} = 1$；$weight_{Vi}$ 是负载指标 *Vi* 所对应的权重。

(3)计算节点机上的综合平均负载指标：

$$machineLoad = \sum_{i=1}^{n} WA_{Vi} \quad (3)$$

对以上 3 步进行归纳，得到下面的总的平均负载计算公式：

$$machineLoad = \sum_{i=1}^{n} ((\sum_{j=1}^{m} Vi)/m \times weight_{Vi}) \quad (4)$$

负载信息收集策略算法描述如下：

```
Begin：
  Start 信息收集定时器启动;
  While(信息采集量 i < n)
  If(滑动窗口容量 > = maxNum)
  Then {
        Lock(滑动窗口 i);
        由式(2)计算负载 WAV i;
        UnLock(滑动窗口 i);
        Store(WAVi);
  }
  Else 由式(1)计算 SumV i;
  End While;
  定时器暂停;
  由式(3)计算本节点机平均负载;
  向负载信息管理进程发出软中断;
  Report(本节点机平均负载);
  Clear(负载信息计算缓冲区);
  恢复定时器操作;
End
```

3.2 负载平衡调度策略

本模型对于所收集的负载信息进行决策调度算法采用了 2 种结构：排序队列和哈希映射矩阵。

排序队列表示为 $m \times n$ 矩阵，结构如下所示：

$$\boldsymbol{SL} = \begin{pmatrix} V_{11} & V_{12} \cdots & V_{1n} \\ I_{21} & I_{22} \cdots & I_{2n} \\ \vdots & \vdots & \vdots \\ I_{m1} & I_{m2} \cdots & I_{mn} \end{pmatrix}$$

矩阵第 1 行 V_{1j} 表示各节点机的负载信息数据值，V_{11} 到 V_{1n} 所代表的数据不重复，按字典序排列，从而避免每次查找节点机最小负载时，都需重新计算负载信息数据值的最小值。采用在构建数据过程中进行排序的方法，提高了其运行效率；I_{ij} 表示各节点机信息，包括节点机计算机名、IP 地址等信息。矩阵中同一列 *j* 表示不同计算机 I_{ij} 所报告的负载数据为 V_{1j}。

哈希映射以各节点机的计算机名为关键字进行。哈希映射的特点在于其在查找效率方面有很好的时间复杂度，其结构如下所示：

$$\boldsymbol{MAP} = \begin{pmatrix} N_{11} & M_{12} & L_{13} \\ N_{21} & M_{22} & L_{23} \\ \vdots & \vdots & \vdots \\ N_{k1} & M_{k2} & L_{k3} \end{pmatrix}$$

它是一个 $k \times 3$ 的矩阵，$N_{i1}(1<i \le k)$表示不同计算机名，作为关键字，它们的唯一性是保证计算正确的前提。$M_{i2}(1<i \le k)$存储的是一个关键值对序列(节点机名，负载值)，$L_{i3}(1<i \le k)$是和前面的排序队列 ***SL*** 的结合点，其存储的是 N_{i1} 在 ***SL*** 矩阵中对应的 I_{ij} 所在列的地址。哈希映射可以快速更新 ***SL*** 中的负载信息数据，保证负载信息的实时性。

负载平衡模型负载信息管理调度算法描述如下：

```
Begin：
 注册 Remoting 远程 SingleCall 并配置信道;
 初始化 SL 和 MAP;
 Start 启动定时器，触发负载信息接收进程;
 If(接收到中断请求) Then
 {
```

```
响应负载报告请求并缓存此节点机信息;
Lock(SL);
Lock(MAP);
在 SL 中查找 If(machineLoad == V1j)
Then {
    Store(请求节点机信息到 SL 中的 Iij);
}
Else {
    查找 machineLoad 在 SL 的 V1j 中应插入的位置(升序);
    Store(machineLoad 到 SL 中指定位置);
}
在 MAP 中查找 If(machineName 不在哈希表中)
Then {
    以 machineName 为关键字做哈希映射到 Nk1，在 MAP 中新增存储空间;
    存储请求节点机的键值对序列到 Mk2;
    存储请求节点机在 SL 中 V1j 的位置到 Lk3;
}
Else {
    查找 machineName 在 MAP 中的映射地址 Nk1;
    以 Nk1 对应的 Lk3 为对象查找 SL 中对应信息并对变更的信息进行删除修改;
    更新 Nk1 对应的 Mk2 和 Lk3 的信息;
}
Unlock(SL);
Unlock(MAP);
}
End
```

同时在模型中加入了负载平衡容错机制，设计出针对本模型的容错算法，负载平衡容错策略描述如下：

(1)给定节点机负载报告延时最大值，超过此上限值仍无响应时，则认为此节点机不可用，将其记入错误队列 *DeadQueue* 中。

(2)对排序队列 ***SL*** 和哈希映射进行扫描，查找其中存入的超时节点机信息并将其从中删除。

(3)当 *DeadQueue* 中的节点机达到设定阈值时，探测进程被激活，对错误队列 *DeadQueue* 中的节点机重新进行检测，节点机可用则重新加入负载调度，并从 *DeadQueue* 中将此节点机删除。

(4)探测进程检测后 *DeadQueue* 中节点机数目超过调度设定阈值，则认为负载平衡调度已经没有意义，不再进行负载调度，避免不必要的系统开销和资源浪费。

4 负载平衡模型结构

结合第 3 节提出的负载平衡策略，依托于.NET 平台，构建出基于.NET Remoting 技术的动态负载平衡模型，总体框架结构如图 2 所示。

模型构建在.NET Remoting 框架之上，不同应用程序域之间的通信通过.NET Remoting 的通信信道。位于不同节点机的模块之间的信息交互采用组播(Multicast)的方式。各模块主要功能如下：

(1)ShareLibrary，包含各个模块都需要的一些基本功能，主要有读取 XML 配置文件、解析读取 XML 配置文件信息、程序异常报告、程序运行出现异常时向用户显示出错信息。

(2)LMS(Load Monitoring Server)，监控处理负载信息，并及时响应 SLB(Server Load Balancer)。其内部模块 Collector Worker 对 LRS(Load Reporting Server)的负载报告响应接收，并交由 Load Collection 进行存储处理，其中在 Collector Worker 中加入了容错机制，即 3.2 节所提出的负载容错策略。Reporter Worker 根据 Load Collection 的变化对 SLB 进行更新。

(3)LRS(Load Reporting Server)，由 CounterInfo 采集节点机负载信息，经由 Reporting Worker 向 LMS 汇报。

(4)LBR(Load Balancing Request)，当客户有任务请求时，向 SLB 发出请求，并等待 SLB 回复，然后给客户端任务分配相应的节点机。

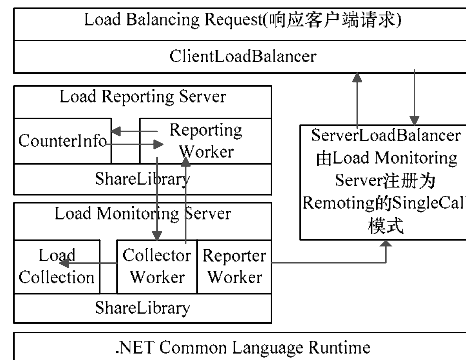

图 2 负载平衡模型框架

Server Load Balancer 注册为.NET Remoting 的 SingleCall 模式。在此模式下，运行库将建立一个新对象为每个客户端请求服务，并在完成请求后释放这个对象。在处理服务器端负载平衡请求时，这个模式非常有效。

5 结束语

提高分布式计算效率的一个关键因素就是实现负载平衡，在考虑负载平衡算法的同时也要注意其不同的应用环境，把不同的算法和技术结合起来使用。

本文提出的动态负载模型以当前广泛应用的.NET Remoting 技术为基础，利用排序队列特定的数据结构和快速响应能力、哈希映射矩阵的高查找效率和映射存储特点，使用松耦合的模块化设计方法，同时为了避免广播所带来的通信开销和网络资源浪费，采用组播通信的方式，和.NET Remoting 技术的相结合使得该模型具有较广泛的适用性。

参考文献

[1] Barnaby T. Distributed .NET Programming in C#[M]. 黎 媛, 王小峰, 译. 北京: 清华大学出版社, 2004.

[2] 王 玥, 蔡皖东, 段 琪. 一种自适应动态负载均衡算法[J]. 计算机工程与应用, 2006, 42(21): 121-123.

[3] Zomaya A, Member S, Teh Y H. Observations on Using Genetic Algorithms for Dynamic Load-balancing[J]. IEEE Trans. on Parallel and Distributed Systems, 2001, 12(9): 889-911.

[4] Chen Chung-Kai, Chang Yu-Hao, Chen Cheng-Wei. Efficient Switching Supports of Distributed .NET Remoting with Network Processors[C]//Proc. of the International Conference on Parallel Processing. Oslo, Norway: [s. n.], 2005: 350-357.

[5] Berger E, Browne J C. Scalable Load Distribution and Load Balancing for Dynamic Parallel Programs[C]//Proc. of the 1st Workshop on Cluster-based Computing. Austin, Texas, USA: [s. n.], 1999: 471-512.

编辑 顾逸斐