基于软件抗衰的分布式负载均衡策略*)

高 炜 杨 群 许满武

(软件新技术国家重点实验室 南京大学计算机科学与技术系 南京 210093)

摘 要 随着网络的迅速发展,服务器集群技术得到了广泛的应用,对负载均衡策略的研究也变得越来越必要,但当前的分布式负载均衡策略始终存在性能和开销不能兼顾的问题。本文将软件抗衰思想引入负载均衡策略设计,根据系统内的均衡程度来确定均衡过程的起止时机,在一定程度上解决了这一矛盾。文内给出了相应的实现算法。

关键词 软件抗衰,分布式,负载均衡,平均负载率,任务迁移

Distributed Load Balancing Strategy Based on Software Rejuvenation

GAO Wei YANG Qun XU Man-Wu

(State Key Lab for Novel Software Technology, Department of Computer Science and Technology, Nanjing University, Nanjing 210093)

Abstract With the rapid development of network, the technique of server clusters is widely used. Accordingly the research on load balancing techniques becomes more and more important and necessary. But at present there always exists the conflict between cost and performance in distributed load balancing strategies. This paper introduces the idea of software rejuvenation in the design of load balancing strategy. According to the degree of balancing in the whole system, the new strategy decides when and which machine in the cluster runs the balancing process. It partly resolves the conflict, there after, the relevant algorithm has been shown.

Keywords Software rejuvenation, Distributed, Load balancing, Rate of average load, Job migrating

1 问题的提出

负载均衡技术是伴随着网络规模的不断扩大,用户与业务量不断增加这一过程而发展起来的。对某一服务来说,单一结点很难满足用户需要,因此增加服务结点协同处理成为了解决这一问题的主要途径。负载均衡系统的任务就是将任务合理地分配到各结点上,实现整个系统的平衡性,提高系统的处理能力和服务质量。

按照拓扑结构的不同,负载均衡策略可以划分为两种:集中式调度策略和分布式调度策略,简称集中式策略和分布式策略。其中集中式策略实际应用较为广泛,它的缺点也很明显:中心结点很容易成为系统瓶颈,一旦崩溃后,则整个系统不可用。本文的讨论集中于后者。

分布式策略中不存在中心结点,系统中任意结点都可以发起均衡过程,一个结点的崩溃不再对整个系统造成很大的影响。由于技术的限制,当前的分布式调度研究一般是建立在如下假设上的:(1)任务是易于迁移的,(2)负载大小是可以量化的。以下讨论都基于这两点假设[1]。

最简单的分布式策略是让每一个结点都拥有集中式调度的中心结点功能,需要频繁的结点间通信来维持所有服务器结点的状态列表。由此所带来的就是大量网络资源的耗费以及额外的系统开销,难以在实际中应用。目前分布式策略主要有以下几种:设立多个子中心结点的层次式调度策略,基于随机选择任务移动结点的概率调度策略;根据负载变化差额而基于梯度模型的策略;接收者驱动、发送者驱动策略以及自适应的契约策略等[1~3]。

以上算法主要可分为3类:

① 层次式调度策略实际上是对集中式策略的改进,不是完全的分布式策略,后面不再对其进行讨论;

② 自适应的契约策略可以达到全局范围内的负载均衡(在此称之为全局性策略),对每个到达的任务都需要发起均衡过程,询问其余结点的处理能力、负载量等数据,额外通信开销太大,在稍大的系统中难以应用;

③ 对于其它几种策略(在此称之为局部性策略),存在的主要问题是只着眼于本结点的情况,不能对全局作出调整,而且仅在某些特定情况下才发挥作用,对系统性能改善不大;有时由于均衡过程频频发生或是迁移任务形成环,甚至进一步降低了系统的性能。因此,虽然不需要过多的结点间通信,但实际应用价值同样有限。

鉴于以上讨论,我们希望能够提出一种策略,使得它既能在全局范围内实现负载均衡,又使得为此需要的通信开销能被接受。为此,我们借鉴软件抗衰的思想来对负载均衡策略进行设计。

2 抗衰思想的引入

2.1 软件抗衰简介

软件衰老又称为"软件老化"(aging),是指长时间运行的软件系统随着运行时间的增长,运行性能逐渐下降并最终变为不可用的过程[4]。另一方面,随着近年来软件系统变得越来越庞大,人们对其可用性、稳定性和运行性能也不断地提出越来越

*)本项目得到国家自然科学基金(60273035)、江苏省自然科学基金(BK2002080)、江苏省科技攻关项目(BE2003064)资助。高 炜 硕士生,主要研究方向为软件方法学、软件抗衰、自主计算;杨 群 博士生,主要研究方向为软件方法学、自主计算、软件自愈;许满武 教授、博导,主要研究方向为软件方法学、新型程序设计。

高的要求。由此，软件抗衰成为研究的热点[5]。

软件抗衰是指为预防系统突然发生故障而预先采取的措施。它是一种“前摄”的容错技术，主要通过适时、适度地消除系统内部错误的运行状态来完成。抗衰的基本思想是在系统(或构件)性能下降到一定程度时(或之前)对其应用抗衰策略，从而使性能恢复到良好状态。软件抗衰技术是利用相关的数学理论分析软件系统运行时的状态参数，适时恢复衰老构件至“干净”状态，被证明是一种有效的解决软件衰老问题的方法。

2.2 负载均衡系统中抗衰思想的引入

一般来说，我们可以用三元组(S,Q,P)来简单表示一个采用了抗衰策略的系统，其中S为该系统，Q为系统提供的服务，P则为抗衰策略。系统的性能可用Q的质量q来表示。可以看出，抗衰策略P不是时时被应用的，在q下降之前并不需要进行抗衰。如果抛开实际应用意义，将负载均衡系统视为一个完整的系统而从软件抗衰的角度去衡量其性能的话，则对于一个应用均衡策略B和抗衰策略P的负载均衡系统S'，可以表述为前述三元组(S',B,P)，系统性能由B的质量b，即负载的均衡水平表示，即在各结点负载相当时表示均衡系统性能良好，而负载相差过大时表示系统性能衰退，需要应用抗衰策略P抗衰。

B是任意的负载均衡策略，而对于负载均衡系统S而言，抗衰策略P实质上是系统粒度上的负载均衡策略P'。换言之，将B无限简化至忽略不计，就可以构造一个应用系统策略P'的均衡系统S'，只有在系统整体负载情况相差过大时才进行全局性的均衡。由此，抗衰思想被引入了负载均衡系统的设计之中。初步来看，这样设计出的策略减少了均衡发生的次数，也考虑到了全局的均衡情况，是一种可行的解决方案。

3 基于抗衰思想的策略

基于前述的考虑，我们提出了基于抗衰思想的分布式负载均衡策略，它的核心思想如下：

① 策略的目的是试图在全局范围内均分负载；

② 以系统内负载的平衡状况作为均衡过程的起止条件；

③ 各结点只在均衡过程中才相互传递自身信息。

由于以上几点叙述相对较为简略，还难以和其他策略直接进行优劣比较。因此，下面我们对策略进行精化，利用具体的算法来进行性能分析。

3.1 相关定义

根据策略思想，均衡起止条件为系统内的负载平衡情况。自然地，我们首先需要制定量化指标来对其进行衡量，此外还需要引入一些概念以方便描述精化后的策略。

① 定义负载率k和系统平均负载率K。

$$k=c/t*100\%$$

(c:结点当前负载；t:结点最大处理能力)

$$K=C/T*100\%$$

(C:系统总负载；T:系统最大处理能力)

k的期望$E(k)=K$，其方差$D(k)$的大小标志了整个系统内的均衡情况。另外，从单个结点的角度出发，k和K的差异过大，代表系统内最少在局部产生了不均衡状况。

② 定义任务迁移树T、前驱/后继结点。

任务迁移树：对任意一次均衡过程通过一定的算法构造一棵由系统全部结点构成的树T，其根结点为发起均衡过程的结点，均衡过程中消息的流动和任务的迁移只能沿着树进行，此即为任务迁移树。

前驱/后继结点：对某次均衡中的某一结点而言，在当前T内该结点的父母/子女结点称为该结点的前驱/后继结点。

定义这几个概念的目的是希望能通过一定的算法确定负载迁移的方向与数量，避免随机性迁移任务可能引起的效率低下状况。同时，由于每次均衡过程都要重新构造T，因此对于有结点失效或是新增结点的情况，对系统整体不会构成较大的影响。

任务迁移树的构造是算法的重点之一，根据考虑因素的不同，其构造过程和构造结果也不尽相同。一般来说，在复杂的异地网络环境中，我们可以根据一定的指标(如物理上的距离或是结点间的跃点数等)将其中我们关注的若干个结点抽象成带权无向图$G=(V,E)$的形式，因此这一步工作就转化为由无向图G构造其生成树T。

考虑异地网络环境(即抽象为带权无向图)的情况。这一工作可以通过构造深度优先(DFS)/广度优先(BFS)生成树或是构造最小代价生成树来进行[6]。直观上来看，由于考虑到减小单个结点失效对整个系统的影响以及时间上的因素，将任务迁移树构造为BFS生成树是一种比较合适的方案。而采用最小代价生成树的方式则适用于网络资源相对较为紧张且对时间要求不太严格的场合。对这一部分的研究将是日后主要工作之一。

对于本地网络集群的情况，需要考虑因素相对较少，迁移树的设计更为自由。例如我们的实验系统中采用的完全K叉树方案：首先将所有本地结点组织为循环链表，对于结点I开始的均衡过程，组织一棵完全K叉树$T(i)$，其根结点为I，整棵树在链表中是分层次依次排列的。这样不需额外数据，只要知道发起结点，通过简单的计算就能确定树的结构。以一个9结点采用完全3叉树的系统为例，$T(0)$和$T(4)$的结构如图1所示。

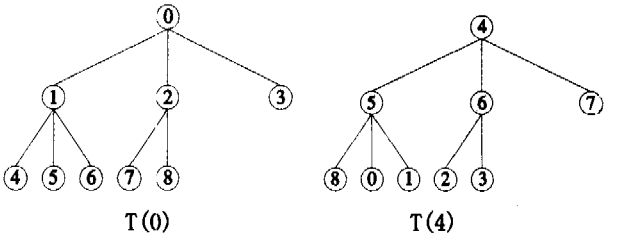

图1 任务迁移树示例

以上涉及的概念和算法均在文[6]中有详细叙述，由于篇幅所限，此处不再赘述。

3.2 基本算法

对于一个应用了软件抗衰的系统而言，正确判断系统性能何时发生衰退是抗衰措施得以发挥作用的基础。如前所述，在负载均衡系统中，$D(k)$的大小是系统均衡与否的标志。但考虑到相应的代价，单个结点不可能时刻获取当前系统内结点的所有相关信息，也即无法时刻计算当前的$D(k)$，只能主要通过自身状况来判断。对单个结点而言，本地统计的k和K达到一定的差距即可认为系统内存在负载不均衡状况，可以发起均衡过程。k是本地数据，易于统计。至于计算K和确定其他一些参数需要的相关数据，通过均衡过程中结点间信息传递来进行。

很自然的想法是均衡过程中收集所有结点数据，计算出新的K值，此即基本算法，其步骤如下：

① 各结点根据初始K值确定均衡过程起止的临界值，并

持续统计自身的 k 值。

② 最先达到临界值的结点发起均衡过程，该结点收集其余所有结点的数据，统计出新的 K 值。

③ 发起均衡结点根据 K 值计算出各结点的负载迁移量以及迁移方向，并发送给其余各个结点。

④ 各结点在收到消息之后更新 K，按消息中所带迁移信息进行负载的迁移。

以上各步中，消息流动和任务迁移方向均按任务迁移树进行。即各结点只能和前驱或后继进行通信，任务可能会进行多次迁移。但迁移次数最多是迁移树的高度，且绝不会形成环。此外，考虑到多个结点可能同时到达临界值，可以采用令牌环或 Bully 算法[7]等方式来保证同一时刻只有一个结点发起均衡过程。

需要注意的是，不仅只有负载过重时需要发起均衡调整 K 值，负载过轻时也同样需要，否则一旦负载达到峰值后再回落，将可能永远无法再发起均衡过程。但由于对两种情况的处理极为类似，后面的讨论中仅以重载情况为例。

在基本算法中，发起均衡结点实际上临时承担了集中式调度的中心结点的任务。与自适应契约策略比较，一次通信的通信量多于后者，但仍在一个数量级上，均衡次数则明显下降（根据任务总量的不同，减少程度不同，可能会降低若干个数量级），总体来说优于后者。

3.3 改进算法

精确地统计 K 值，需要所有的结点都参与每一次均衡过程，大量的通信不可避免。这样的策略与其余的全局性策略比较，仍没有特别明显的优势，因此我们希望能对基本算法进行改进，设法进一步减少通信开销。

依然从软件抗衰的角度去考虑。传统软件抗衰研究的着眼点集中在软件系统层次，检测和抗衰的对象都是整个软件系统。但事实上，软件系统中只有部分存在衰老现象，抗衰可以只针对有需要的一部分进行。同时，对一个系统而言，可能有多种力度不同的抗衰策略。考虑到具体实现与开销等实际情况，力度最强的策略未必是综合效果最好的，因此需要选择最恰当的策略应用在实际系统中[8]。根据以上考虑，我们设计了如下的改进算法：

① 各结点根据初始 K 值确定均衡过程起止的阈值，并持续统计自身的 k 值。

② 最先达到阈值的结点发起均衡过程，各结点在均衡过程中互相交换信息，以各自重新计算最新的系统平均负载率。

③ 根据一定的算法确定参与均衡的结点，并在该范围内实现负载均衡（根据算法以及实际负载量的不同，均衡过程可能是局部的，也可能扩展到所有结点）。

④ 对于在一次均衡过程中没有参加任务迁移的结点，可以在均衡过程的最后更新 K 值，或者是定时由某一结点广播 K 值，来达到基本统一估计系统负载水平的目的。

和基本算法比较，差别主要有以下两点：首先，改进算法中一次均衡过程不一定牵涉到所有的结点。当认为在一定范围内的严重不均衡状况已经消除时，均衡消息或是任务迁移不再沿迁移树继续传播，均衡过程终止。其次，改进算法设法采用估计值来计算，只和部分结点进行通信，从有限的数据中估计系统的大约负载状况，通过调整负载达到相对均衡的目的。

整个均衡过程流程如下：结点达到均衡开始条件后，重新计算系统负载，往后继发送平衡消息 Balance（包含自身所估计的系统负载，也可包含自身负载等信息）。收到消息的结点重新计算系统负载，判断自身能否满足迁移要求。若可以，则向前驱发回回应 Response；若不能满足，则在平衡消息中附加自身信息后，继续向后继发送 Balance（若本身已经是叶结点，则向前驱发送失败消息 Failure）。发起均衡结点在收到所有后继结点的回应（Response 或 Failure）后，调整估计系统负载，进行负载的迁移。

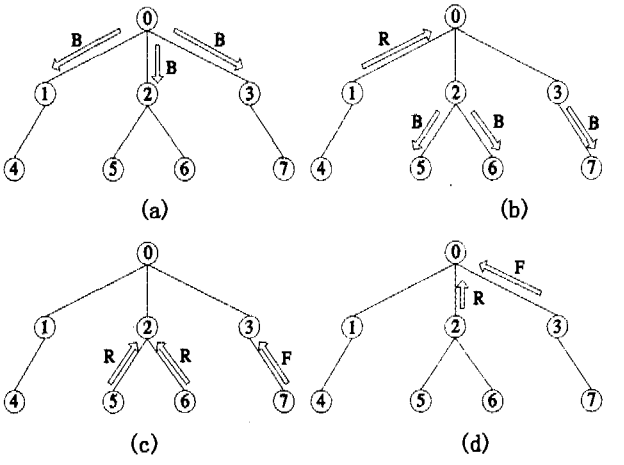

图 2 均衡过程示意图

图 2 描述了 8 结点构成的系统一次均衡过程中的消息流动状况，其中 0～7 表示结点，B/R/F 分别表示消息 Balance/Response/Failure。结点 1 因为能直接满足结点 0 的初始要求，从而不再向后传播 Balance，而是直接返回 Response。结点 7 由于不能满足结点 3 的要求而返回 Failure，导致结点 3 也向前返回 Failure。结点 0 在收到所有后继结点的消息（无论是 R 还是 F）后，根据消息中包含的数据计算并迁移实际负载。

每次消息的传递或是任务的迁移都有可能做出对自身估计系统负载的调整，根据具体算法有所不同。此外，在改进算法中，迁移的任务只能是沿任务迁移树由根向叶单向流动。

4 具体实现方案

前述的基本/扩展算法在实际应用中根据不同的要求，会设计出不同的估计系统负载的函数。同时，相应的 Balance 和 Response 等消息所包含的信息也不完全一致，可能包含以前经过结点的负载量、当前处理能力等等其他数据，相应的性能也会有所差异。在我们的实验系统高性能视频点播系统中，根据扩展算法采用了 2 种方案，分别进行了设计及实验。根据实际情况，也为简单起见，这里忽略了结点最大处理能力的差异，即用负载量 C 的大小直接代表负载率 k，系统平均负载量 L 代表系统平均负载率 K。

4.1 乐观/悲观方案

根据系统特性以及侧重点的不同，我们制定了乐观和悲观 2 种方案。

乐观方案：试图将发起结点超过原系统平均负载的部分负载按权值均匀地分摊在每一个结点上，认为只有自身负载较重，其余结点的负载都较轻，适用于系统内各结点差异较大的场合。

悲观方案：试图把超过系统平均负载的部分平均分摊在最邻近（即直接后继）的结点上。悲观策略认为，既然自身达到了

均衡过程的阈值，则整个系统的负载都应该较重，适用于各结点情况与负载特性均类似的场合。

从另一个角度说，乐观方案注重于均衡性，希望更多的结点参与，从而在全局内达到较好的均衡性。悲观方案则注重于减少通信开销，希望通过尽可能少的结点参与就达到局部内均衡的目标。

4.2 确定相关数据

(1)定义函数 S

对结点 p、q 定义 $S(p,q)$。乐观方案中，$S(p,q)$ 的值为在当前 q 结点开始的均衡过程模式下以 p 结点为根的子树的所有结点数；对叶结点，$S(p,q)=1$。悲观方案中，$S(p,q)$ 的值为 p 的所有直接子女数+1。

乐观方案与悲观方案的差别主要表现在此处，其余部分的计算是相同的。

(2)确定均衡起止阈值 f_{start}、f_{end}

我们简单地取 $f_{start}=4K$，$f_{end}=3/2K$。即当结点自身负载大于等于系统负载的 4 倍时发起均衡过程，而在一次均衡过程中，自身负载小于系统负载的 3/2 时，不再继续向后继结点传播均衡消息。

此外，为避免负载普遍过重时发起不必要的均衡过程，进一步增加系统负担，还要设定绝对上限。当负载超过上限时，该结点不再主动发起均衡，其值根据具体机器而定，我们设为 3/4A，A 为本结点最大处理能力。

(3)确定 Balance/Response/Failure 等消息中携带的数据

前面已经提到，由于采用了循环链表的表示方式，只需要发起结点的标识 P 即可根据链表构造出任务迁移树。此外，为了能够计算估计系统负载，还需要消息来源结点(设为结点 I)的当前负载 Ci、估计系统负载 Li 以及迁移负载量 M。

(4)确定计算估计系统负载函数

由于负载的迁移在均衡过程的最后才实际进行，因此结点自身统计的 Ci 在均衡过程中只是个计算值，不一定为真正的负载量，只有在均衡过程开始前和结束后两者才相等。

在结点 N 发起的均衡过程中，结点 I 需要迁移的总负载量是超出新计算的预计系统负载的部分，迁移给后继结点 J 的负载量为：

$M=\lfloor S(j,n)*[(Ci-Li)/(S(i,n)-1)]\rfloor$(乐观情况)

$M=\lfloor (Ci-Li)/(S(i,n)-1)\rfloor$(悲观情况)

其中 $\lfloor\ \rfloor$ 为取整符号。

当结点 J 在结点 N 发起的均衡过程中收到来自结点 I 的 Balance 消息时，依次做下列计算：

$Cj=Cj+M$;

$Lj=Li$;

$Lj=Lj+(Cj-Li)/S(j,n)$

当结点 J 收到来自后继结点 K 的 Failure 消息时，

$Cj=Cj+M$

在均衡过程的最后，由发起均衡结点计算所有参与均衡结点的平均估计系统负载，作为新的估计系统负载。

4.3 实验结果

为了能够直观地了解策略的效果以及两种算法的差异，下面以一个 6 结点的系统来进行策略的实施。T 构造为完全二叉树，发起均衡结点均为结点 3，即 T 如 3 图。

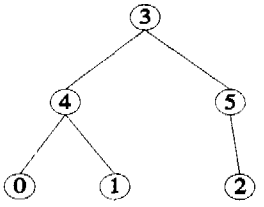

图 3 实验系统的任务迁移树

表 1～3 给出了系统负载较轻/一般/较重的情况下分别根据两种方案发起一次均衡过程后的数据，$C(I)$ 表示结点 I 的负载量，$L(e)$ 表示估计系统平均负载，$L(f)$ 表示实际系统平均负载，T_0 为初始时刻，T_{pe}/T_{op} 为采用悲观/乐观方案进行一次均衡后的时刻。初始 $L(e)$ 均为 16。

表 1 系统轻载情况

	C(0)	C(1)	C(2)	C(3)	C(4)	C(5)	L(e)	L(f)
T_0	12	12	8	64	20	12	16	21
T_{pe}	12	12	8	32	36	28	32	21
T_{op}	19	18	8	24	31	28	28	21

表 2 系统中载情况

	C(0)	C(1)	C(2)	C(3)	C(4)	C(5)	L(e)	L(f)
T_0	12	60	8	64	20	60	16	37
T_{pe}	12	60	30	32	36	54	38	37
T_{op}	18	60	34	24	38	50	36	37

表 3 系统重载情况

	C(0)	C(1)	C(2)	C(3)	C(4)	C(5)	L(e)	L(f)
T_0	60	60	48	64	32	60	16	54
T_{pe}	60	60	70	32	48	62	42	54
T_{op}	71	70	74	24	35	50	40	54

与各结点的初始数据相比较，可以得出以下几点结论：(1)对结点 0 和结点 1 的调整较小，而其余结点的负载变化较明显，这是各结点在任务迁移树中的位置不同引起的，说明迁移树的构造方式对具体的均衡结果影响很大。(2)对于中等负载的情况，估计系统负载与实际负载相当接近，但在其余两种情况下系统负载的调整都有明显的迟滞现象，在一定程度上会增加均衡发生的次数。这一现象可以通过调整估计系统负载的函数加以修正，但无法完全避免。(3)无论是悲观策略还是乐观策略在各种情况下都起到了确实的均衡效果，对估计系统负载作出了相应的调整，这说明策略是合理且有效的。

5 策略性能分析

除了直接衡量均衡程度的指标 $D(k)$ 之外，对分布式负载均衡来说，均衡所需通信量的大小以及时间的长短也是影响策略效率的重要参数。根据实际情况，我们有以下观点：与建立一次通信的开销相比，每次通信消息里携带的数据量大小的影响要小得多。因此，在一次通信携带数据量不是十分巨大的情况下，通信量的大小我们通常可以近似用通信次数这一指标来衡量。由于在同样的负载状况下，不同的策略在一段时间内发起的均衡次数并不相同，所以除了单次均衡的通信量和通信时间外，总计通信量也应被纳入考虑范围。

考虑共 n 个结点，总负载量为 m 的情况(为简单起见，用

任务数代替负载量),对几类策略进行比较(表4)。

表4 几类策略的比较

	单次通信量	单次通信时间	总计均衡次数	均衡程度指标 D(k)
自适应契约策略	O(n)	O(1)	m	$O(1/n^2)$
局部策略	O(1)~O(n)	O(1)	O(1)~O(m)	$O(m^2/n^2)$
基于抗衰均衡策略	O(n)	O(logn)	O(logm)	$O(1/n^2)$[注]

注:实际为 $O(u^2/n^2)=O(1/n^2)$ $u=f_{start}/K$ 为常数。

将基于抗衰均衡策略和自适应契约策略进行比较,单次通信量、总计通信时间(等于单次通信时间 * 均衡次数)相当(在 $m\gg n$ 时此指标也明显占优),总通信量大大少于后者,即系统开销优于自适应算法;系统均衡程度虽不及自适应算法,但 $D(k)$ 仍在同一数量级上,表示确实得到了有效的均衡。

局部策略本身情况相对复杂,这类策略受到系统特性的影响很大,在开销这一指标上难以直接比较。一般来说,对于自身负载普遍较低的系统,局部策略的均衡次数、通信量都较小,最理想的情况下趋近于0;而对于负载较高的系统,局部策略发起均衡的次数会明显上升,某些极端情况下甚至逼近无穷大。不论哪种情况,基于抗衰均衡策略的开销都是相对稳定的,前一种情况是局部策略占优,后一种情况则反之。在系统均衡程度上,基于抗衰的均衡策略是明显优于局部策略的。

综合来看,基于抗衰思想的负载均衡策略部分吸收了两类分布式策略各自的优点,在全局均衡以及减少开销间取得了一定的平衡,初步达到了前文所提到的目标。

结束语 我们讨论了一种基于软件抗衰的分布式动态负载均衡策略,通过结点自身状况和有限的通信来推测系统的均衡情况以及发起均衡的时机及范围,部分解决了分布式负载均衡策略中全局性和效率性不能兼顾的问题。但是,我们提出的具体算法适用范围有限,对系统和负载特性有一定的要求。此外,对任务迁移树的设计等问题尚未进行更深入的研究。在以后的实验和研究中,我们会不断对策略本身和具体算法进行进一步的完善。

参考文献

1 Harvey D J. Load balancing techniques for distributed processing environment:[Ph. D Thesis]. Arlington: The University of Texas, 2001

2 Harvey D J, Biswas R, Das S K. Dynamic load balancing for adaptive meshes using symmetric broadcast networks. In: 12th Intl. Conf. on supercomputing, Melbourne, Australia, 1998. 417~424

3 Teo Y M, Ayani R. Comparison of Load Balancing Strategies on Cluster-based Web Servers. Transactions of the Society for Modeling and Simulation, 2001

4 Huang Y, Kintala C, Kolettis N, et al. Software Rejuvenation: Analysis, Modules and Application. In: Proc. of 25th Symposium on Fault Tolerant Computer Systems, Pasadena, California, June 1995. 381~390

5 An architectural blueprint for autonomic computing. IBM and autonomic computing

6 陈本林,陈佩佩,吉根林. 数据结构. 南京:南京大学出版社,1998

7 Tanenbaum A S. Distributed operating systems. 北京:清华大学出版社,1997

8 朱广蔚,谭建,杨献春,等. 在构件的多个实现间动态选择的算法. 计算机应用研究,2004,19(9):20~24

9 Willebeek-Lemair H, Reeves A P. Strategies for dynamic load balancing on highly parallel computers. IEEE Transactions on Parallel and Distributed Systems, 1993, 4(9): 979~993

10 Buyya R. High Performance Cluster Computing Architectures and Systems. Prentice Hall PTR, NJ, USA, 1999, 1: 340~363

11 Eager D L, et al. A comparison of receiver-initiated and sender-initiated adaptive load sharing. Performance Evaluation, 1986, 6: 53~58

12 Lin FCH, Keller R M. The gradient model load balancing method. IEEE Transactions on Software Engineering, 1987, SE-13(1): 32~38

(上接第241页)

结论 本文提出四参数四点细分方法,给出了 C^0 到 C^2 连续的充分条件;在理论上证明了采用本文的方法可造型满足高阶光滑的曲线,并把该方法推广到曲面造型上。利用该方法进行山地模拟,用曲面网逼近地形的整体形状,然后控制最终曲面网格的显示精度,结合有效的纹理生成技术,真实感显示地表细节。充分利用细分方法中参数可以取不同取值,递归产生曲面,从而产生具有不同地貌特征的地形。该方法有效、实用且简单,效果是令人满意的,为山地造型提供了一种新的方法。

参考文献

1 Dyn N, Levin D, Gregory J A. A 4-point interpolatory subdivision scheme for curve design. Computer Aided Geometric Design, 1987, 4(4): 257~268

2 Chao Yuan. Necessary and sufficient condition for the continuous Limit curves and surfaces of 4-point interpolation subdivision scheme. Journal of Computer Aided Eesign & Computer Graphics, 2003, 15(8): 961~966

3 Jin Jian-Rong, Wang Guo-Zhao. A non-uniform 4-point interpolatory subdivision scheme to construct curve. A Volume High School Application Mathematical Transaction, 2000, 15(1): 97~100

4 Jiang Lei, Guo De-Gui. An improvement to the four-point interpolation subdivision scheme. Journal of Yantai University, 2002, 15(3): 171~176

5 Wang Zheng-Xuan, Pang Yun-Jie. A recursive algorithm based four-point interpolation scheme for curve design and its application to rendering of fractals. Journal of CAD & CG, 1997, 9(3): 223~227

6 Wang Jing, Qian Xiao-Yuan. Dimensionality estimation of the fractal interpolatory curve generated by 4-point interpolatory subdivision scheme. Journal of Gansu University of Technology, 2003, 29(3): 120~122

7 Hassan M F, et al. An interpolating 4-point C2 ternary stationary subdivision scheme. Computer Aided Geometric Design, 2002, 19(1): 1~18

8 Dyn N. Subdivision schemes in computer-aided geometric design. In: Lighted W, ed. Advances in numerical analysis, Clarendon Press, 1992, 2: 36~104

9 Cararetta A S, Dahmen W, Micchelli C A. Stationary subdivision [J]. Memoirs of the American Mathematical Society, 1991, 93(453): 1~186

10 Zhao Hong-Qing, Peng Guo-Hua, Ye Zheng-Lin. Study a New Subdivision Scheme for Curve. Journal of Software, 2004, 15(suppl): 246~251