广义超立方体和它的任务分配问题*

毛国君 王 薇 杨名生

(北京工业大学计算机科学与工程系 北京 100022) (大连理工大学工程力学研究所 大连 116023)

摘要 广义超立方体EH(extended hypercube)作为组织大型分布式多处理机系统的拓扑结构,在使用阈值和阈长两种控制参量的阈值选择策略中表现出许多良好的性质.文章探讨了EH中的若干性质,这些性质为实现高效稳定的任务分配算法提供了理论基础.最后,该文详尽地讨论了在一个EH中,基于使用阈值和阈长的启发式选择策略、多叉树状态信息组织方法以及动态阈值修改措施的任务分配算法和它的性能.

关键词 广义超立方体(EH),任务分配,阈值/阈长,状态信息/CIT树,阈值修改.

中图法分类号 TP338

随着并行计算系统中处理单元PE(processor element)数目的增加,组织这些PE的拓扑结构显得越来越重要.就任务分配而言,一个优秀的拓扑结构是产生高效算法的基础.在这方面,超立方体(Hypercube)已经显示出它的优势.有关超立方体中的任务分配和容错转移方案近几年来被广泛地讨论.[1,2]简言之,一个n维的超立方体$H(n)$具有$k=2^n$个节点和$(k\log_2 k)/2$条连线,使得两个PE间的最短路径不超过$\log_2 k$.虽然和全互连式的拓扑结构相比,超立方体的连线数目从$o(k^2)$减到$o(k\log_2 k)$,但是当节点数目k很大时,超立方体中节点间连线数目以及最大最短路径仍然很大.为此,J. M. Kumar等提出了一种称为广义超立方体EH(extended hypercube)的结构.[3]在EH中,所有的PE被分成若干组,在组内采用超立方体结构来组织它们,而组与组之间的联系则通过增加网络控制节点NC(network controller)来层次式地控制.这种结构为组织大型的分布式计算处理系统提供了新的途径.

负载均衡是分布式系统中任务分配的关键问题,而利用阈值方法进行启发式的负载均衡是目前广泛使用的策略.[4,5]传统的阈值策略把节点分成接受者和发送者两种类型,使得接受者这种欠载的节点成为接受任务的候选者,而发送者这种超载的节点成为转移任务的候选者.但是系统中的颠簸问题[4]必须引起注意.一种常见的颠簸现象表现为,当一个接受者刚刚接受一个任务,并由此而成为一个发送者时,这个节点自身又到达一个任务,因为它已经成为发送者了,而不得不转移这个后到达的任务.如果这种现象频繁发生,系统的效率就在这种颠簸中下降.文献[6]在使用阈值的同时,增加了一个称为阈长的控制参量,这就在一定程度上减少了这种颠簸现象.

阈值的确定有两种基本的方法[6]:静态方法和动态方法.静态方法是事先为系统的所有节点一次性地给出合适的阈值,它只能建立在对系统中各节点情况的静态分析基础上.例如,在异构式系统中,对处理能力大的节点应该给一个大一点的阈值.动态方法则根据系统的动态信息加以调整.例如,在总负载大时每个节点应该分担更多的任务,阈值相应要大些.阈长的确定也可以通过静态或动态方法给出,一般不宜过大,因为过大的阈长可能降低系统的负载均衡程度.[6]

本文给出EH的若干性质,这些性质基于使用阈值及阈长两种控制参量的阈值选择策略.最后讨论了建立在这些性质基础上的一个动态任务分配算法以及它的性能.

1 广义超立方体和它的节点编号

EH拓扑结构是建立在超立方体结构基础上的一种规则的递归式层次结构.对于一个n维$l+1$层广义超立方体$EH(n,l)$,当$l=0$时,它退化成一个n维超立方体$E(n)$.当$l=1$时,它包含1个NC和2^n个PE.在第0层上的2^n个PE构成一个超立方体$E(n)$,而在第1层上的NC和所有第0层上的PE都有连线(称这个NC为这些PE的父亲节

---

* 作者毛国君,1966年生,讲师,主要研究领域为分布式系统,人工智能.王薇,女,1972年生,讲师,主要研究领域为分布式系统,数据库.杨名生,1939年生,教授,博士导师,主要研究领域为分布式系统,计算机图形学.

本文通讯联系人:毛国君,北京 100022,北京工业大学计算机科学与工程系

本文1997-03-18收到原稿,1997-06-09收到修改稿

点).图1给出了$EH(3,1)$的结构示意图.一般地,当$l>1$时,在第0层上的PE被划分成$2^{(l-1)n}$组,组内的PE以$E(n)$加以组织,而在第1层上为第0层上的每个$E(n)$配备一个NC(称为它们的父亲节点)以构成一个$E(n,1)$,并且所有第1层上的NC(共$2^{(l-1)n}$个)又被分成$2^{(l-2)n}$组,组内的2^n个NC也以$E(n)$加以组织.如此进行下去便可以形成一个$EH(n,l)$.事实上,一个$EH(n,l)$可以递归地定义为2^n个$EH(n,l-1)$加上一个和这些$EH(n,l-1)$均相连接的NC.图2给出了$EH(3,2)$的拓扑结构.

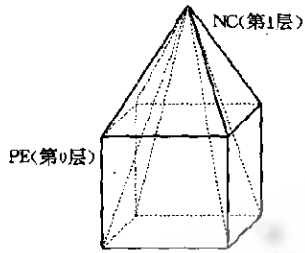

图1　$EH(3,1)$结构示意

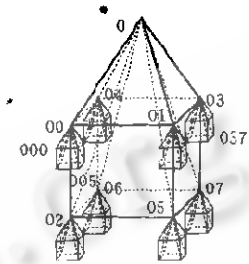

图2　$EH(3,2)$结构示意及部分节点编码

从上面的叙述可以看出,EH是靠横纵两种关系维系在一起的规则拓扑结构.从横向上看,同一层上的节点(PE或NC)被分成若干组.组内的节点通过超立方体形式加以组织,可以直接相连(称为相邻)或间接相连.从纵向上看,上一层的节点又和它所控制的下一层的节点存在父子关系.推而广之,祖先和子孙的概念在此也适用.EH要进行任务分配,需要节点间的通讯及任务传输等,这些工作都建立在对节点间关系的存储和利用上,而合理的节点编号方法对于存储和利用这些关系是很重要的.我们使用了下面这些编码原则来形成对$EH(n,l)$的不等长节点编号.

(1) 对最上一层(第l层)的NC的编号至少需要n个bit;

(2) 除最上一层以外,任一层上的NC节点的编号由它父亲节点的编号加上至少n个bit来形成;

(3) 某一层上的同组的超立方体中的相邻两个节点(PE或NC)有且仅有一个bit不同.

遵循这些原则,$EH(3,2)$的部分节点编号如图2所注,其中的节点编号用八进制形式给出.这样的节点编号为形成两个节点间的任务传输路径提供了方便.例如,在图2中,编号为000和005的两个PE在同一个超立方体内,它们对应的二进制数通过异或运算(⊕)得到的二进制结果为$(000000000)_2 \oplus (000000101)_2 = (000000101)_2$,其中有两个bit为"1",这正反映了这两个PE间的最短路径是2.最短路径可以从一个节点出发,通过每次对一个异或结果所指示的位(异或结果中为"1"的位)进行取反来得到.例如,000和005有两条最短路径:(000,004,005)和(000,001,005).如果两个PE不在同一个超立方体内,则必须都先向上搜索,而且总能在某一层上找到它们的祖先在同一个超立方体内,然后形成路径.例如,000和037不在同一个超立方体内,但它们的父亲00和03在同一个超立方体内,于是可以形成传输路径:(000,00,01,03,037)或(000,00,02,03,037).另外,NC的瓶颈问题需要加以考虑.我们知道,一个NC要直接或间接地控制若干个PE.NC所在的层次越高所控制的PE越多.如果一个NC难以招架它控制的所有PE所提出的要求,这就造成所谓的NC瓶颈问题.所以在形成传输路径时,应该尽量在低层内进行.例如,000和037有一个可能的传输路径:(000,00,0,03,037),但是由于它比前面的两条路径牵涉到更高层次的NC,所以应该优先考虑前面的两条路径.

2　EH的性质及它的任务分配思想

在EH中,PE是真正执行具体任务的节点,而NC则是为了进行任务分配特别是实现启发式负载均衡而配备的.一般地,每个NC除了记录它本身的包括负载在内的状态信息以外,还应该记录它的子孙的状态信息.一个NC的负载就是它的所有儿子节点的负载之和,最终是它所直接或间接控制的所有PE的负载之和.如上所述,PE的阈值和阈长可以通过静态的或动态的方法来确定.NC的阈值和阈长通过下面的定义给出.

定义 2.1. 一个NC节点的阈值或阈长分别为其所有儿子节点(PE或NC)的阈值或阈长之和.

定义 2.2. 若一个NC节点的负载小于它的阈值和阈长之差,则这个NC为接受者;若一个NC的负载大于它的阈值和阈长之和,则这个NC为发送者;既不是接受者也不是发送者的节点被称为负载适中节点,简称为OK节点.

定理 2.1. 在EH中,如果一个NC是接受者,那么至少存在它的一个儿子节点(PE或NC)是接受者.

证明:设NC^*是$EH(n,l)(l>0)$中的第i层$(1\leqslant i\leqslant l)$的一个NC,它所对应的阈值和阈长记为$T_i^*$和$\alpha_i^*$,它的负

载记为 $Load_i^*$. 用反证法证明.

假设 NC^* 的所有节点都不是接受者，即对任意 $p(p=1,2,\ldots,2^n)$，有 $Load_{l-1}^p \geqslant T_{l-1}^p - a_{l-1}^p$，其中 $Load_{l-1}^p$，T_{l-1}^p 和 a_{l-1}^p 分别对应 NC^* 的第 p 个儿子的负载、阈值和阈长. 于是

$$Load_i^* = \sum_{p=1}^{2^n} Load_{l-1}^p \geqslant \sum_{p=1}^{2^n} (T_{l-1}^p - a_{l-1}^p) = \left(\sum_{p=1}^{2^n} T_{l-1}^p\right) - \left(\sum_{p=1}^{2^n} a_{l-1}^p\right) = T_i^* - a_i^*.$$

因此，NC^* 不是一个接受者，和命题的前提矛盾. 故定理 2.1 成立. □

推论 2.1. 在 EH 中，如果一个 NC 是一个接受者，那么至少存在它的一个子孙 PE 是接受者.

定理 2.2. 在 EH 中，如果一个 NC 是一个发送者，那么至少存在它的一个儿子节点(PE 或 NC)是发送者.

定理 2.2 的证明和定理 2.1 类似，不再赘述.

推论 2.2. 在 EH 中，如果一个 NC 是一个发送者，那么至少存在它的一个子孙 PE 是发送者.

定理 2.3. 在 EH 中，如果一个 NC 是一个 OK 节点，那么至少存在它的一个儿子节点不是发送者；同样地，如果一个 NC 是一个 OK 节点，那么至少存在它的一个儿子节点不是接受者.

证明：只证"如果一个 NC 是一个 OK 节点，那么至少存在它的一个儿子节点不是发送者"，另外一部分类似. 设 NC^* 是一个 OK 节点，而且 $Load_i^*$，T_i^*，a_i^*，$Load_{l-1}^p$，T_{l-1}^p 和 a_{l-1}^p 同定理 2.1 约定. 用反证法证明.

假设 NC^* 的所有儿子都是发送者，则对任意的 $p(p=1,2,\ldots,2^n)$，有

$$Load_{l-1}^p > T_{l-1}^p + a_{l-1}^p,$$

于是

$$Load_i^* = \sum_{p=1}^{2^n} Load_{l-1}^p > \sum_{p=1}^{2^n} (T_{l-1}^p + a_{l-1}^p) = \left(\sum_{p=1}^{2^n} T_{l-1}^p\right) + \left(\sum_{p=1}^{2^n} a_{l-1}^p\right) = T_i^* + a_i^*.$$

因此，NC^* 是发送者，这与命题的前提矛盾. 故定理 2.3 成立. □

推论 2.3.1. 在 EH 中，如果一个 NC 是一个 OK 节点，那么至少存在它的一个子孙节点 PE 不是发送者；同样地，如果一个 NC 是一个 OK 节点，那么至少存在它的一个儿子节点不是接受者.

证明：用数学归纳法. 只证"如果一个 NC 是一个 OK 节点，那么至少存在它的一个子孙节点 PE 不是发送者". 对 $EH(n,l)$ 的第 i 层$(1 \leqslant i \leqslant l)$的 NC_i^*，有

(1) 当 $i=1$ 时，NC_i^* 的儿子节点由 PE 组成，由定理 2.3 知，推论 2.3.1 成立；

(2) 当 $i>2$ 时，假设对于第 $i-1$ 层的所有是 OK 节点的 NC 而言，推论 2.3.1 的结论都成立. 现看第 i 层的 NC_i^*，因为 NC_i^* 是 OK 节点，由定理 2.3 知，存在第 $i-1$ 层的一个 NC 节点 NC_{i-1}^* 不是发送者，即要么 NC_{i-1}^* 是接受者，要么是 OK 节点. 假如 NC_{i-1}^* 是接受者，由定理 2.1 知，可以找到 NC_{i-1}^* 的一个子孙 PE 是接受者(当然不是发送者). 假如 NC_{i-1}^* 是一个 OK 节点，由归纳假设知，可以找到一个 NC_{i-1}^* 的一个子孙 PE 不是发送者.

由(1)(2)知，推论 2.3.1 成立. □

推论 2.3.2. 在 EH 中，如果一个 NC 的所有儿子(或者所有子孙 PE)都是接受者或都是发送者，那么这个 NC 是接受者或是发送者.

由定理 2.3 的证明不难看出推论 2.3.2 是正确的.

如上所述，EH 中的 NC 节点的主要任务之一是帮助形成任务转移的决定. 当一个任务到达某 PE 时，它若不能本地处理就需要转移，实现这种转移必须要解决转移到何处的问题. 这一般需要两个过程，首先向上搜索这个 PE 的祖先集合以试图找到一个合适的 NC(记为 NC^*)，然后再从 NC^* 向下搜索以找到一个它的合适的子孙 PE(记为 PE^*). NC^* 最好是一个接受者，因为由推论 2.1 知道，此时一定能向下搜索到 PE^* 是接受者，于是欠载的 PE^* 接受这个任务. 如果到达任务的 PE 的祖先全不是接受者，只好找一个 OK 节点作为 NC^*，此时由推论 2.3.1 知，也能从 NC^* 向下搜索找到 PE^* 是接受者或 OK 节点，这样的 PE^* 也作为转移点使用. 如果到达任务的 PE 的祖先全是发送者，说明系统的总负载已经和当前的阈值及阈长不再适应(原因见文献[6])，这时，进行阈值修改是必要的(后面讨论). 由此可见，EH 中的这种任务分配思想是基于它的良好特性的.

3　状态信息的组织方法

状态信息的组织与使用是为了能作出更好的分配决策. 集中式和分散式是分布式系统中组织状态信息的两种基本方法.[4]集中式方法就是采用一个称为"中心节点"的节点来负责收集整个系统的状态信息. 由于任何一个节点的状

态信息(负载等)发生变化时都必须告诉中心节点,因此,状态信息的交换频繁.分散式方法则每个节点都可以收集系统中的一些节点的状态信息,必要时作出对整个系统的状态信息的估计.它不如集中式方法那么准确,但是减少了节点间的信息交换,进而有望提高系统的执行效率.

根据EH的拓扑结构特点,我们采用如下的层次式方法来组织状态信息.一个PE节点只收集它自己的执行进程与任务等待情况、存储与设备使用情况以及负载、阈值和节点类型等所有必要的状态信息.一个NC节点不仅记录本节点的状态信息,而且记录它的子孙节点的状态信息以及它们之间的联系.为此,在设计模拟算法时,我们为每个PE或NC设计一棵称为控制信息树CIT(control information tree)的完全多叉树.CIT上的数据结点(只列出一些重要数据域)如下:

No	Level	Load	Type	Th	Alpha	FC	NS

其中No为此数据结点对应的PE或NC的节点编号;Level为其对应PE或NC在EH中的层次,当Level=0时,此数据结点对应一个PE,否则对应一个NC;Load为其对应PE或NC的负载;Type为0,1,2时,分别对应于接受者,OK节点和发送者;Th和Alpha是其对应的PE或NC的阈值和阈长;FC(first child)是指向这个数据结点的第1个儿子结点的指针;NS(next sibling)是指向这个数据结点的同层右邻兄弟结点的指针.

从逻辑上说,对于$EH(n,l)$,在第i层($1\leqslant i\leqslant l$)的每个NC节点上,它的CIT是一棵$i+1$层的完全多叉树.显然,当EH的层次较大时,对高层上节点的CIT的搜索越来越困难,因此需要找到一种有效的对CIT树的搜索方法.文献[7]给出了一种三叉树的称为"儿子 兄弟链表"的存储方法,并且详细地讨论了它的各种基本操作的高效性.这种方法不难推广到一般的多叉树中.上述的CIT树的数据结点的FC和NS两个数据域就是为这种存储方法而设置的.简言之,在这种多叉树的存储结构下,向下搜索儿子意味着通过FC找到第1个儿子,而寻找其他儿子只需要通过NS链来进行,不需要回溯到父结点.

4 状态信息的维护

状态信息随着系统任务的到达和释放处于动态变化之中,与之相对应,必须要有有效的动态维护方法.一致性、时效性和可操作性是动态维护的基本原则.[4]我们只选择两个修改相关的CIT树起点,即当且仅当有新任务到达某个PE或有已经完成的任务从某PE上释放时,才进行状态信息的维护.

当有一个任务到达某个PE^*时,根据PE^*的节点类型不同,有3条途径来处理这个任务,每种情况都需要进行状态信息的维护.(1) PE^*是接受者或OK节点,则进行本地处理,维护状态信息从PE^*开始向上修改,完成对其自身以及它的所有祖先节点包括负载及节点类型等在内的状态信息的维护.(2) PE^*是发送者而且在PE^*的祖先集合中找到了一个NC^*是接受者或OK节点,按第2节所述的方法可以从NC^*向下找到一个可以接受此任务的PE^{**},状态信息的维护从这个PE^{**}开始向上进行.(3) PE^*是发送者,而且PE^*的所有祖先都是发送者,此时进行阈值修改是必要的.(4) 当一个任务从某个PE^*上释放时,也必须从PE^*向上修改其自身以及它的所有祖先的CIT树上的相关信息.另外,这种情况也存在着阈值修改的可能性.这是因为当系统的总负载过小,而系统中各节点的阈值相对过大时,系统中的大多数节点变成了非发送者,系统中节点间的合作很少进行,因而系统的负载均衡很差.

阈值修改以系统总负载为根据是合乎逻辑的.我们知道,当系统的总负载增大时,每个PE应该分担更多的任务,因而阈值应该大些;反之,当系统的总负载减少时,每个PE应该分担较少的任务,因而阈值应该小些.发生上面(3)的情况一般是因为,在某一时期内系统的任务到达率大大高于它的任务释放率,此时增大阈值可以增加系统的任务容纳量,改善系统在高负载环境下的忍耐力.当然阈值不能无休止地增加下去.当系统在一个相当长的时期中任务到达率始终大于任务释放率,而导致所有的PE处于几乎满负荷运转状态时,一些任务的等待成为必然.在(4)的情况下,如果在某一时期内系统的任务释放率大大高于任务到达率,而导致系统的总负载下降很快,减少阈值可以使任务更均匀地分布到各PE中去,因而提高系统的任务执行效率.本文仅考虑所有PE使用相同阈值和阈长的同构式EH,而且使用平均工作量MWL(mean work load)[6,7]的概念来作为一个节点的负载以及阈值和阈长的度量标准,即理论上说,一个节点的负载是所有它的任务的工作量之和除以MWL所得的商数.在任务可分割的情况下,如果以MWL为将一个大任务划分成若干子任务的话,那么一个节点的负载就是它所包含的子任务数.同样地,一个节点的阈值和阈长也是以MWL为单位来度量的(我们使用整数表示).我们知道,目前对一个节点的负载的普遍定义是执行它的所有任务所需的时间,而在实际操作上,则往往需要进行启发式的估算[5],这是因为到达一个节点的任务很难预先精确知道它所花费的执行时间.把一个大的任务划分成若干子任务来进行任务分配是分布式多处理机系统的常用方法.本文涉及到

的“任务”、“任务数”都是相对于这样的子任务概念而言的.

当需要增加阈值时，如果取 PE 上的新阈值为 $t_0=\lceil L_{top}/2^{nl}\rceil$（当 $L_{top}/2^{nl}$ 为整数时，$t_0=L_{top}/2^{nl}$；否则，$t_0=int(L_{top}/2^{nl})+1$），其中 L_{top} 为 $EH(n,l)$ 系统的总负载. 下面的定理 4.1 保证了在上面(3)的情况下使用这种阈值修改方法以后，再进行任务分配不会再发生(3)的情况，进而保证了算法的稳定性.

定理 4.1. 在 $EH(n,l)$ 中，设当前系统的总负载为 L_{top}，如果此时所有的 PE 具有相同的阈值 $t_0=\lceil L_{top}/2^{nl}\rceil$，相同的阈长 $\alpha_0\geqslant 0$，那么最上一层的 NC 节点肯定不是一个发送者.

证明：对 $EH(n,l)$，设 $Load_l$，T_l 和 α_l 代表最上一层 NC 的负载、阈值和阈长，由 EH 的结构特点，知

$$Load_l=L_{top}=(L_{top}/2^{nl})\ 2^{nl}\leqslant(\lceil L_{top}/2^{nl}\rceil)2^{nl}=t_0\ 2^{nl}\leqslant t_0\ 2^{nl}+\alpha_0\ 2^{nl}=T_l+\alpha_l.$$

所以，最上一层的 NC 不是一个发送者.

当任务释放并进行信息状态维护到最上一层时，如果发现总负载减少很多，减少阈值将提高系统效率. 我们在设计算法时，使用 $T_l-L_{top}>2^{nl}$ 作为阈值修改条件. 同样地，使用 $t_0=\lceil L_{top}/2^{nl}\rceil$ 作为 PE 上的新阈值（有关符号意义同前）. 下面的定理 4.2 保证了在新的阈值条件下，增加了系统对任务释放率长期高于任务到达率环境下的适应性.

定理 4.2. 在 $EH(n,l)$ 中，设当前系统总负载为 L_{top}，如果此时修改所有的 PE 具有相同的阈值 $t_0=\lceil L_{top}/2^{nl}\rceil$，且使用相同的阈长 $\alpha_0\geqslant 1$，那么最上一层的 NC 节点不是一个接受者.

证明：$Load_l$，T_l 和 α_l 意义同定理 4.1，则

如果 $L_{top}/2^{nl}$ 为整数，有

$$Load_l=L_{top}\geqslant int(L_{top}/2^{nl})\ 2^{nl}=t_0\ 2^{nl}=T_l;$$

否则，有

$$Load_l=L_{top}\geqslant int(L_{top}/2^{nl})2^{nl}=(int(L_{top}/2^{nl})+1)2^{nl}-2^{nl}=t_0\ 2^{nl}-2^{nl}\geqslant t_0\ 2^{nl}-\alpha_0\ 2^{nl}=T_l-\alpha_l.$$

因此，不论 $L_{top}/2^{nl}$ 是否为整数，都有 $Load_l\geqslant T_l-\alpha_l$，故最上一层 NC 不是一个接受者.

推论 4.1. 在 $EH(n,l)$ 中，设当前系统总负载为 L_{top}，如果此时修改所有的 PE 具有相同的阈值 $t_0=\lceil L_{top}/2^{nl}\rceil$，且使用相同的阈长 $\alpha_0\geqslant 1$，那么有最上一层 NC 一定是一个 OK 节点.

由定理 4.1 和 4.2 很容易得出这个推论.

5　实验与讨论

为了验证上述 EH 中任务分配思想的可用性，我们对 $EH(3,2)$ 进行了模拟实验. 模拟算法 ETA 描述如下：

当一个任务到达 $PE1$ 时，处理的主要过程为

```
Void TaskArrive(PE1)
{
    PE2=WhoDoTask(PE1);
    if (PE2 =0xFF){
                PathMake(PE1,PE2);
                UpModify(PE2,TaskSize);
                }
    else{
          ThresholdModify();
          TaskArrive(PE1);}
}
```

当一个任务从 PE1 释放时，处理的主要过程为

```
Void TaskFinish(PE1)
{
     UpModify(PE1,-TaskSize);
     if (abs(Tl-Ltop)>2^nl))
          ThresholdModify();
}
```

其中函数 WhoDoTask(PE1)用于寻找能接受此任务的 PE 节点（见第 2 节）. 当找不到合适节点来接受此任务时，返回 0xFF；PathMake(PE1,PE2)形成 PE1 到 PE2 的传输路径（见第 1 节）；UpModify(PE2,TaskSize)用于从 PE2 自下而上进行状态信息的更新（见第 4 节），当任务到达时，给 PE2 及其它的所有祖先节点的负载加上 TaskSize，当任务释放时则减去 TaskSize；ThresholdModify()则按定理 4.1 或 4.2 所提供的方法进行阈值修改.

为了验证算法的稳定性和高效性，我们同时进行了对比实验. 第 1 个对比算法 ETA1 是只使用阈值而不使用阈长参量来实现阈值选择策略，同时使用如上所述的阈值修改方法. 第 2 个对比算法 ETA2 是不使用阈长参量，同时也不进行阈值修改(分别取固定阈值 1,2,3,4). 以平均探寻 NC 节点数 MPN 作为比较指标(当某个 PE 有任务到达时，每向上测试一次它的祖先 NC 为一次探寻，通过统计被探寻的 NC 节点数目和任务的到达次数，即可得到 MPN 的值). 在 500 次的随机模拟任务到达节点的实验中，得出了如图 3 所示的 ETA，ETA1 和 ETA2 三个算法随着系统总负载增大时 MPN 的变化情况(其中 MWL=1).

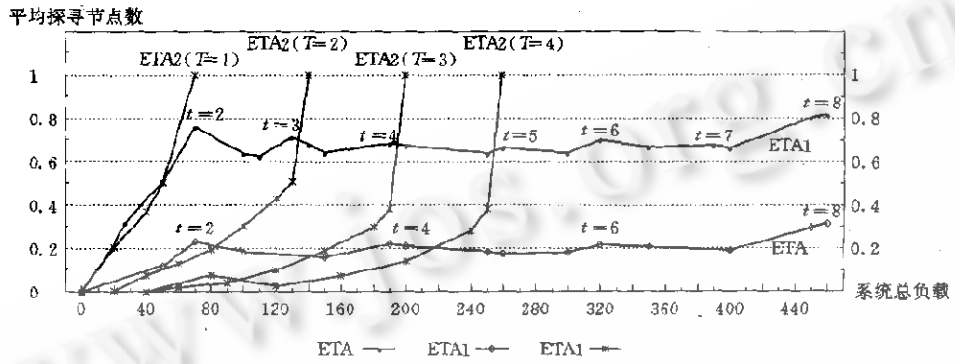

图3　ETA，ETA1，ETA2性能比较图

对图 3 加以分析得出结论如下：

(1) 阈长的引进减少了 MPN. 如图 3 所示，ETA1 的 MPN 在 0.7 左右摆动，而 ETA 则在 0.2 左右摆动，即 ETA 平均向上探寻 NC 的数目还不足 ETA1 的 1/3，这就有效地抑制了 NC 的瓶颈问题. 在图 3 中，ETA1 共修改阈值 7 次，而 ETA 则只修改了 4 次. 阈值修改一般要涉及到全系统的所有节点，不宜过于频繁. 由于 ETA 在阈值修改后维持时间比 ETA1 要长，所以系统的稳定性要高.

(2) 阈值修改方法的使用增加了系统对负载的承受力. 在图 3 中，如果 ETA2 使用 $t=1$ 的固定阈值，当负载超过 64 时，新到达的任务就会被拒绝. 虽然使用大的固定阈值可以增加系统对负载的承受力(当 $t=4$ 时总负载可以达到 256)，但是随着总负载增大 MPN 增加很快，而且大的固定阈值必然对负载均衡影响很大.

(3) ETA 有两个固有的缺点，克服这些缺点以改进它的性能是我们下一步的工作：(1) 它仍然使用发送者启动策略，这种策略随着系统的总负载增大，匹配节点的效率下降.[4,5]如果采用双向启动策略[6](发送者和接受者都能实施启动行为)，可能延缓 MPN 的增加速度. 但是，在 EH 中使用双向启动策略必然增加状态信息的维护量，因而需要找到更有效的方法. (2) 阈值修改在全局范围内强迫式统一进行，即当阈值修改期间，所有节点中断现行工作，待所有节点的状态信息更新后才能恢复原来工作. 我们设想将来实现分散式异步进行阈值修改，即只修改最上层 NC 节点的 CIT 树，待以后再向下和其它节点打交道时，再顺便把阈值修改后的新状态信息传递下去. 这种想法难度在于如何有效地控制状态信息的延迟更新所带来的消极影响.

致谢　对大连理工大学王秀坤教授对本文工作的支持和帮助表示衷心感谢.

参考文献

1　Woo J, Sahni S. Load balancing in a hypercube. In: Los Alamitos ed. Proceedings of the 5th International Parallel Processing Symposium. Anaheim, California: IEEE Computer Society Ar., 1991. 525～530

2　Schmidit V M. Efficient parallel communication with the nCUBE 2S processor. Parallel Computing, 1994,20(4):509～513

3　Kumar J M, Patnaik L M. Extended hypercube: a hierarchical interconnection network of hypercubes. IEEE Transactions on Parallel and Computer, 1992,13(1):45～47

4　Shivaratri N G, Krueger P, Singhal M. Load distributing for locally distributed systems. IEEE Computer, 1992,25(2):33～44

5　Bark A B, Shioh A. A distributed load-balancing policy for a multicomputer. Software——Practice and Experience, 1993,15(9):901～913

6　毛国君，杨名生等. 分布式系统中的双向启动自适应任务分配算法. 计算机学报，1996,19(7):514～519

(Mao Guo-jun, Yang Ming-sheng *et al*. A bi-intiated adaptive algorithm for task allocation in distributed systems. Chinese Journal of Computers, 1996,19(7): 514~519)

7 毛国君,杨泽非.一种三叉树的存储结构及其基本操作的实现.计算机研究与发展,1994,31(5):62~65
(Mao Guo-jun, Yang Di-fei. A storage structure for trinary tree and the implementation of its basic operations. Computer Research & Development. 1994,31(5):62~65)

Extended Hypercube and Its Task Allocation

MAO Guo-jun

(*Department of Computer Science and Engineering Beijing Proctology University Beijing* 100022)

WANG Wei YANG Ming-sheng

(*Research Institute of Engineering Mechanics Dalian University of Technology Dalian* 116023)

Abstract EH(extended hypercube) is a kind of topologic architecture to organize large-scale distributed multiprocessor systems. Extended threshold strategy, which uses threshold-length parameter besides threshold, provides a new method to increase the stability in a distributed system. In this strategy, there are many good properties in EH. In this paper, some properties of EH are given at first, then a task allocation algorithm for EH is designed, and finally the performance of this algorithm is discussed in detail.

Key words EH(extended hypercube), task allocation, threshold/threshold-length, state information/control information tree, threshold modification.