分布式计算的并发测量

贺乐天 孙永强

(上海交通大学计算机科学与工程系 上海 200030)

摘要 本文提出了一种分布式计算的并发测量方法,这些测量使用基于局部时钟的区域向量时钟,延迟向量和计算高度等概念量化了并发计算时间和同步延迟时间,不用定义原子事件和划分事件粒度;测量机制独立于被测量的程序,容易计算,管理简单,同时考虑了处理机间的性能差异.

关键词 分布式计算,并发测量,区域向量时钟,同步延迟.

中图法分类号 TP302

顺序程序中常用时间和空间复杂度来表示计算的效率.并行程序中常用加速比和计算成本来表示计算的效率.在分布式程序和算法中,则常用计算时间和消息数量来描述计算的效率.计算时间和消息数量是量的测量,不能回答如计算是否较好地分布了和是否因为同步延迟而影响了计算效率等质的问题.Charron-Bost[1,2]提出用计算中的一致切割(Consistent Cut)数量与理想情况下的一致切割数量的比值来表示并发度和通过量化同步延迟(Synchronization Delay)来测量并发度两种方法,Fidge[3]用由于并发而节省的逻辑时间与假定给出无限资源情况下能节省的逻辑时间的比值来计算并发度,Raynal 等[4]发展了用同步延迟测量并发度的思想,使用向量时钟[5,6]及其扩展提出了计算并发度的公式.

计算中一致切割的数量是不可计算的[4],因而文献[1]的方法在实际应用中是不可行的.文献[2~4]的方法使用原子事件定义,假设了所有事件消耗单位时间,不符合测量的实际情况,而且这些方法需要在被测量的程序中实现,有较大的局限性.本文提出的并发测量方法能回答上面的质的问题,这些测量容易计算,独立于被测量的程序,适合于对实际分布式计算的分析和测量.本文的测量与传统量的测量一起,较好地刻画了分布式计算.

1 分布式计算

1.1 分布式计算的模型

一个分布式系统是 n 个顺序进程的集合$\{P_1,\ldots,P_n\}$,进程间仅以消息传递的方式通讯,通讯是异步的,消息延迟是不确定的有限时间,通讯是可靠的,系统中没有全局时钟.文献中常用原子事件来抽象计算中有意义的事件,原子事件分为 3 类:消息发送事件、消息接收事件和内部事件.Lamport 用 happened before 关系[7]说明了原子事件间的关系是严格的偏序关系,又称为因果关系[8,9],向量时钟[5,6]刻画了这种因果关系.原子事件简化了计算的形式描述和验证.但由于程序性质的千差万别,对事件粒度的划分存在着很不一致的标准,而且原子事件的定义依赖于程序实现,管理复杂,使用原子事件来测量分布式计算难以客观地反映实际计算,其有效性受到限制.

进程间的逻辑关系只有在消息传递时才能改变,进程在消息传递之间是顺序的.我们放弃对内部事件的识别,只考虑消息传递事件和由这些事件划分的进程区域.常常用进程时空图[7]来直观地表示分布式计算,如图 1 所示,水平方向为进程时间轴,轴上的圆点表示消息事件,轴间的箭头线表示消息传递.下面形式地定义区域和区域间的关系.

定义 1.1. A_i 表示 P_i 上由消息传递事件划分的区域集合,$A=A_1\cup A_2\cup\ldots\cup A_n$ 是计算中所有区域的集合.对 A_i 中的区域按发生的先后排序,使其呈现为 $A_i=\{a_{i,1},a_{i,2},\ldots\}$.

定义 1.2. A 上的因果关系:$\rightarrow\subseteq A\times A$ 是满足如下条件的最小传递关系:

(1) $a_{i,j},a_{i,k}$是 P_i 的两个区域,$j<k$,则 $a_{i,j}\rightarrow a_{i,k}$;

(2) 区域 s 的右边界是消息发送事件,区域 r 以相应的接收事件为左边界,则 $s\rightarrow r$.

---

* 作者贺乐天,1971 年生,博士,主要研究领域为分布式计算,分布式算法.孙永强,1931 年生,教授,博士导师,主要研究领域为软件理论,并行处理.

本文通讯联系人:孙永强,上海 200030,上海交通大学计算机科学与工程系

本文 1996-10-15 收到原稿,1996-11-19 收到修改稿

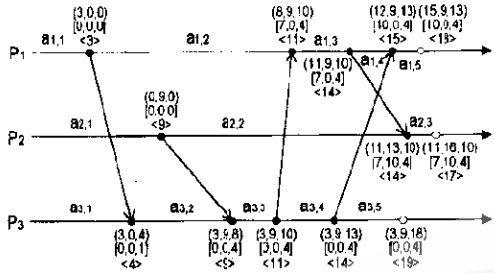

图1　进程时空图

定义 1.3. A 上的并发关系：$\| \subseteq A \times A$ 是：$\forall a, a' \in A, a \| a' \text{iff} \neg(a \rightarrow a') \wedge \neg(a' \rightarrow a)$.

设各进程在终止时都有一个终止事件，如图1所示白点表示的事件，称该事件前的区域为终止区域. 与 happened before 关系相同，$\rightarrow$ 是严格的偏序关系，但 $\|$ 是非传递的. 图1中，$a_{1,1} \rightarrow a_{1,3}$，$a_{2,1} \rightarrow a_{1,5}$，$a_{1,2} \| a_{2,1}$，$a_{1,2} \| a_{3,3}$，但 $a_{2,1} \rightarrow a_{3,3}$.

下面进一步细化并发测量模型.

1.2　消息原语

异步通讯时，消息发送是非阻塞的，而消息接收可能是阻塞的，也可能是非阻塞的. 不失一般性，假定消息接收是阻塞的，并定义如下的消息原语：消息发送 $send(m)$、阻塞接收 $brev(m)$ 和消息接收 $deliver(m)$. $send(m)$ 为进程向其它进程发送消息 m，$brev(m)$ 为进程阻塞等待从其他进程传来的消息 m，$deliver(m)$ 则表示进程实际接收消息 m.

这3个消息原语是计算中需要识别的事件，进程时空图上标出了消息发送和接收事件，阻塞接收存在于右边界为消息接收的区域中. 这样，右边界为消息发送的区域完全是进程的局部计算，而右边界为消息接受的区域则包含局部计算和阻塞接收两个部分. 我们将阻塞接收的部分区域称为同步延迟区域，如图2所示.

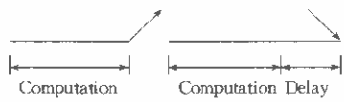

图2　两种进程区域

1.3　局部时钟

为了量化局部计算量和同步延迟量，设进程 P_i 除了有一局部时钟 T_i 外，还有一个对计算计时的计算时钟 CT_i 和一个对阻塞计时的同步延迟时钟 BT_i，定义时钟上的如下操作：

(1) $reset(T)$：初始化时钟，即 $T_i=0$；

(2) $enable(T)$：开启时钟，即让时钟滴嗒计时；

(3) $disable(T)$：关闭时钟，即让时钟停止计时.

要求时钟值在开启状态时是严格单调的，在连续的两个关闭操作之间是可区分的.

协议 1.4. 计算中各时钟遵循如下的协议：

(1) 计算开始时：$reset(CT)$，$reset(BT)$，$enable(CT)$，$disable(BT)$；

(2) 执行 $brev(m)$ 时：$disable(CT)$，$enable(BT)$；

(3) 执行 $eliver(m)$ 时：$enable(CT)$，$disable(BT)$；

(4) 进程终止时：$disable(CT)$，$disable(BT)$.

分布式系统的各处理机间存在着性能差别和时钟偏移，定义每个处理机的性能参数 $\rho_i (0<\rho_i \leqslant 1)$ 表示各处理机的相对计算速度；这一性能参数可通过基准测试程序得到. 为简化下面的描述，假设局部时钟 CT_i 和 BT_i 都是被参数 ρ_i 归整后的时钟，即 $CT_i := (\rho_i \times CT_i)$ 等. 要求对时钟的操作不受计算中和计算外操作（如维护向量时钟等）的影响.

1.4　区域向量时钟、延迟向量和计算高度

为了在计算中跟踪进程上的实际计算量和同步延迟量及为了测量到达计算中的某区域右边界时最少应消耗的计算时间，需要在计算中最长的区域链上统计计算时间，我们提出了基于局部时钟的区域向量时钟 W 和延迟向量 U 及定义了计算高度 HT 的概念.

协议 1.5. P_i 上保持两个 n 维的向量 W_i，U_i 和一个变量 HT_i，遵循如下的协议（$OldCT_i$ 为临时变量）：

(1) 进程初始时：

$W_i[i] := CT_i$，$W_i[k] := 0$ $(1 \leqslant k \leqslant n,\ k \neq i)$；

$U_i[i]:=BT_i, U_i[k]:=0$ $(1\leqslant k\leqslant n, k\neq i)$;

$HT_i:=0, OldCT_i:=CT_i$;

(2) 执行 $send(m)$时:$W_i[i]:=CT_i; HT_i:=HT_i+CT_i-OldCT_i, OldCT_i:=CT_i$;

让被发送出去的消息带上向量 W_i,U_i 和变量 HT_i;

(3) 执行 $deliver(m)$时,设消息携带的向量为 $W(m)$和 $U(m)$,携带的变量为 $HT(m)$,则:

$W_i[i]:=CT_i, W_i[k]:=\max(W_i[k], W(m)[k]), 1\leqslant k\leqslant n$;

$U_i[i]:=BT_i, U_i[k]:=\max(U_i[k], U(m)[k]), 1\leqslant k\leqslant n$;

$HT_i:=HT_i+CT_i-OldCT_i, OldCT_i:=CT_i, HT_i:=\max(HT_i, HT(m))$;

(4) 进程终止时:$W_i[i]:=CT_i; U_i[i]:=BT_i; HT_i:=HT_i+CT_i-OldCT_i$.

约定区域右边界处的向量 W 为区域向量时钟,下面称到达区域时都指到达区域右边界.区域向量时钟可以理解为:$W_i[i]$为到达该区域时 P_i 上执行的计算量(时间),$W_i[j]$为到达该区域时 P_j 上至少应执行的计算量.区域向量时钟刻画了区域间的因果关系.延迟向量与区域是对应的,可以理解为:$U_i[k]$为到达区域时 P_k 上的延迟时间.HT_i 在逻辑意义上等价于 *Lamport* 逻辑时钟.[7]用整数表示时钟值,图 1 中分别用(...)、[...]和⟨...⟩标出了区域向量时钟、延迟向量和计算高度.

2 并发度测量和其他测量

2.1 并发测量

区域向量时钟和延迟向量量化了计算中实际的计算量和同步延迟量.为了测量计算的并发程度,需要计算实际的并发计算量和理想情况下的并发计算量.

设 a 是进程 P_i 上的区域,$W(a)$,$U(a)$和 $HT_i(a)$分别表示 a 的区域向量时钟、延迟向量和区域高度,用 W_i,U_i 和 HT_i 分别表示 P_i 终止时的区域向量时钟、延迟向量和计算高度,C 表示整个计算.我们象文献[4]中一样定义如下的概念:

(1) 区域 a 的重量 $wt(a)=\Sigma_{k=1,\ldots,n}W(a)[k]$表示从计算开始到达 a 时所执行的计算量;

(2) 区域 a 的容量 $vol(a)=\Sigma_{k=1,\ldots,n}(W(a)[k]+U(a)[k])$表示理想情况下从计算开始到达 a 时最多能执行的计算量;

(3) 计算 C 的高度 $ht(C)=\max_{k=1,\ldots,n}(HT_k)$表示计算中最长因果路径上的计算量;

(4) 计算 C 的重量 $wt(a)=\Sigma_{k=1,\ldots,n}W_k[k]$表示从计算开始到终止所执行的计算量;

(5) 计算 C 的容量 $vol(a)=\Sigma_{k=1,\ldots,n}(W_k[k]+U_k[k])$表示理想情况下从计算开始到终止最多能执行的计算量;

由实际的并发计算量和理想情况下的并发计算量可以计算出区域和整个计算的并发度,分别用 $\alpha_a(a)$和 $\alpha(C)$表示,有如下的公式:

$$\alpha_a(a)=(wt(a)-HT_i(a))/(vol(a)-HT_i(a));$$

$$\alpha(C)=(wt(C)-ht(C))/(vol(C)-ht(C));$$

即 $$\alpha_a(a)=(\Sigma_{k=1,\ldots,n}W(a)[k]-HT_i(a))/(\Sigma_{k=1,\ldots,n}(W(a)[k]+U(a)[k])-HT_i(a));$$

$$\alpha(C)=(\Sigma_{k=1,\ldots,n}W_k[k]-\max_{k=1,\ldots,n}HT_k)/(\Sigma_{k=1,\ldots,n}(W_k[k]+U_k[k])-\max_{k=1,\ldots,n}HT_k).$$

上面的两个公式中,分子表示实际的并发计算量,分母表示理论上可能的最大并发计算量,当计算中只有一个进程时,上面的公式因分母为 0 而失去意义.计算区域并发度的意义在于当每个区域的并发度都最大化时,整个计算的并发度同样最大化了.我们的测量与其他作者提出的测量[1~4]是可以比较的,$\alpha=1$ 表示完全并发的计算,$\alpha=0$ 则表示完全顺序化的计算.图 1 中,α_a 在 $a_{1,1}$和 $a_{2,1}$上没有意义,$\alpha_a(a_{3,1})=0.75$,$\alpha_a(a_{3,2})=0.73$,$\alpha_a(a_{1,2})=0.59$,$\alpha_a(a_{1,4})=0.58$,$\alpha_a(a_{2,2})=0.49$,$\alpha(C)=0.56$.由这些结果可以看出区域 $a_{1,4}$和 $a_{2,2}$的并发度较低,这为调节整个计算提供了有益的信息.

2.2 其他测量

使用前面的模型和定义,还可以计算很多有趣的测量.如在进程 P_i 的区域 a 上:

$\varphi_1(a)=W(a)[i]/(W(a)[i]+U(a)[i])$:表示 P_i 上到 a 时计算时间与总时间的比值,称为局部计算效率;

$\varphi_2(a)=U(a)[i]/(W(a)[i]+U(a)[i])$:表示 P_i 上到 a 时由于同步延迟损失的计算时间与总时间的比值,可称为局部计算的损耗率;

$\varphi_3(a)=W(a)[i]/\Sigma_{k=1,\ldots,n}W(a)[k]$:表示 P_i 上执行的计算与为产生 a 所需要的计算的比值;

$\varphi_i(a)=HT(a)/\Sigma_{k=1,\cdots,n}W(a)[k]$：表示 P_i 上到 a 时最长因果路径的计算时间与总计算时间的比值，该比值与并发度成反比.

在整个计算 C 上，可以测量：

$\Phi_1(C)=\Sigma_{k=1,\cdots,n}W_k[k]/\Sigma_{k=1,\cdots,n}(W_k[k]+U_k[k])$：表示计算中实际计算时间与总时间的比值，可称为计算效率；

$\Phi_2(C)=\Sigma_{k=1,\cdots,n}U_k[k]/\Sigma_{k=1,\cdots,n}(W_k[k]+U_k[k])$：表示计算中同步延迟时间占总计算时间的比值，可称为计算的损耗率；

$\Phi_3(C,i)=W_i[i]/\Sigma_{k=1,\cdots,n}W_k[k]$：表示进程 P_i 上执行的计算与整个计算的比值，可称为进程 P_i 在计算中的比重；

$\Phi_4(C)=HT(C)/\Sigma_{k=1,\cdots,n}W_k[k]$：表示计算中最长因果路径上的计算时间与总计算时间的比值，该比值与并发度成反比.

$\Phi_5(C)=\Sigma_{k=1,\cdots,n}W_k[k]/n$：表示计算的平均负载；

$\Phi_6(C)=\Sigma_{k=1,\cdots,n}|W_k[k]-\Phi_5(C)|/n$：表示计算的平均负载偏差；

$\Phi_7(C)=(\Phi_5(C)-\Phi_6(C))/\Phi_5(C)$：表示计算的负载均衡程度；

考虑各处理机间的差异，定义系统的计算能力 $\rho=\Sigma_{k=1,\cdots,n}\rho_k$ 和如下的测量：

$\Phi'_5(C)=\Sigma_{k=1,\cdots,n}W_k[k]/\rho$：表示在负载平衡状况下，性能参数为 1 的处理机上的负载；

$\Phi'_6(C)=\Sigma_{k=1,\cdots,n}|W_k[k]-\Phi'_5(C)\times\rho_k|/\rho$：表示实际计算的平均负载偏差；

$\Phi'_7(C)=(\Phi'_5(C)-\Phi'_6(C))/\Phi'_5(C)$：表示实际计算的负载均衡程度；

上面的测量为分布式计算提供了多种角度的理解，有助于更好地分析和把握计算的特性并通过这些测量调整计算.使用区域向量时钟等概念，还可以想象出其他的测量.

3　各种测量方法的比较

本节回顾了现有的测量方法，并简要地同我们的方法进行了比较.

3.1　Charron-Bost 基于一致切割数量的方法[1]

分布式计算 C 的一致切割 Cut 是事件集合 E 的有限子集：$Cut\subseteq E$ 使得 $e\in Cut, e'\rightarrow e$ 则 $e'\in Cut$. 计算包含的一致切割数越多，则计算的并发度越大. 并发度 m 定义为：$m(C)=(\mu-\mu')/(\mu'-\mu')$. 其中 μ 是计算的一致切割数量，μ' 和 μ' 是包含同样进程数量和事件数量的完全顺序化和完全并发化计算的一致切割数量. 测量 m 定义在整个计算 C 上，从理论上刻画了计算 C 的并发度，然而 μ 是不可计算的.

3.2　Fidge 的测量方法[3]

测量并发度的公式为：$\beta=(\rho-\tau)/(\rho-1)$. 其中 ρ 和 τ 分别表示在计算的当前点之前必须发生的事件的最小数量和最长因果路径上的时间数量. 公式中的分子表示由于并发而节省的逻辑时间量，分母表示若给出无限数量的计算资源能节省的时间量. β 可在事件和整个计算上计算，形式上 β 与 α 分子相同，但 β 没有考虑同步延迟对计算的影响.

3.3　基于同步延迟的测量方法[2,4]

Charron-Bost[2]首先提出用同步延迟来测量并发度，Raynal 等[4]用锥和柱抽象发展了这一思想. 使用传统的向量时钟[5,6]和扩展了 Lamport 逻辑时钟[7]的向量时钟，Raynal 等定义了事件和计算的高度 ht、重量 wt 和容量 vol. 计算并发度的公式为：

$$a(a)=(wt(a)-ht(a))/(vol(a)-ht(a)),\ a(C)=(wt(C)-ht(C))/((n-1)\times(ht(C)).$$

形式上，$a(a)$ 的计算公式与我们的 α_a 完全相同，但两者的测量机制完全不同. Raynal 等认为进程必须等待计算终止后才能终止，因而 C 的容量为 $n\times ht(C)$. 我们认为，进程完成局部计算后的情形是计算 C 的未知世界，因而 C 的容量应该是实际进程时间的累加.

4　并发测量与时钟方法

4.1　事件计数方法的局限性

原子事件的定义取决于对程序行为的理解和粒度划分. 有意义的事件随应用目的、任务性质和计算的不同阶段的不同而不同. 这样，必须仔细地标识不同进程和不同阶段的事件，给程序开发带来了很大的负担. 忽略事件的粒度差别，对它们计以单位时间，在实际分析和测量中，难以获得合理的结论.

传统逻辑时钟[5~7]使用原子事件的计数，实现这些时钟需要在程序内维护一个(组)全局变量，要修改本来的程序. 在复杂的计算中，管理这一变量要付出很高的代价. 在系统调用时，往往难以准确及时地更新这一变量. 这一变量

甚至会影响程序的本来行为.[8]

使用逻辑时钟的并发测量方法[2~4],虽然可以获得逻辑上的结果,但由于原子事件划分的困难和不一致,这些测量得到的并发度仅仅是原子事件间的可比性,在平衡负载算法中,由此得到的结论不能找到真正的性能瓶颈和平衡方案.

4.2 基于局部时钟的测量

基于局部时钟来测量并发度不需要去识别计算的原子事件和划分行为粒度,适合于任何性质的进程.本文中的区域向量时钟、延迟向量和计算高度只关心消息事件的时间,可由进程或系统的通讯模块去管理这些量,独立于程序.这给分布式应用的监控、调试和分析带来了方便.我们给每个处理机一个性能参数来减小由处理机间的性能差异和时钟偏移带来的测量误差.区域向量时钟刻画了因果关系,能为需要保证因果一致性的应用(如因果序通讯[10])提供系统级支持.

实现本文的测量机制需要两个额外的时钟——计算时钟和延迟时钟.各进程维持测量机制所需的计算很少,设计算中消息事件数目为 k,执行协议 1.4 和 1.5 的时间复杂度为 $O(k)$.各进程上需要 $O(2n+1)$的空间来保存区域向量时钟、延迟向量和计算高度,相应地消息上也需要附着 $O(2n+1)$的信息量.当计算中的进程数较多时,压缩向量的算法[8,11]可以用来降低消息上的空间复杂度,但带来了较大的计算复杂度.

传统向量时钟的复杂性一方面在于已经证明了不可能有更小的时钟来刻画因果关系[12],另一方面在于其不能独立于程序,实现困难,管理复杂.[13]区域向量时钟使用系统中的固有资源(时钟,通讯层),独立于程序,降低了向量时钟方法在应用中的实际复杂度.

5 结 论

基于原子事件计数的并发测量方法,在应用中难以合理地定义事件和划分事件粒度,而且必须在程序内部来实现和管理,不能独立于被测量的程序.本文提出的并发测量方法使用基于局部时钟的区域向量时钟,延迟向量和计算高度量化了计算时间和延迟时间;测量机制独立于被测量的程序,容易计算,管理简单,同时考虑了处理机间的性能差异.

本文提出的区域向量时钟能刻画计算的因果关系,降低了向量时钟方法在应用中的实际复杂度.我们认为,基于原子事件计数的向量时钟方法适合于对程序的逻辑描述和验证,基于局部时钟的区域向量时钟适合于对实际计算的分析和测量.

参考文献

1 Charron Bost B. Combinatorics and geometry of consistent cuts: applications to concurrency theory. In: Proceedings of the International Workshop on Distributed Algorithms. LNCS 392. 1989. 45~56

2 Charron-Bost B. Measure of parallelism of distributed computations. In: Proceedings of the 6th Annual Symposium on Theoretical Aspects of Computer Science. LNCS 349, 1989. 434~445

3 Fidge C J. A simple run-time concurrency measure. In: Proceedings of the 3rd Australian Transputer and OCCAM User Group Conference. 1990. 92~101

4 Raynal M, Mizuno M, Neilsen M L. Synchronization and concurrency measure for distributed computations. In: Proceedings of the 12th International Conference on DCS. 1992. 700~707

5 Fidge C J. Logical time in distributed computing systems. IEEE Computer, 1991,24(8):28~33

6 Mattern F. Virtual time and global states in distributed systems. In: Proceedings of the Workshop on Parallel and Distributed Algorithms. North-Holland, 1989. 215~226

7 Lamport L. Time, clocks, and the ordering of events in a distributed system. Communications of the ACM, 1978,21(7):558~565

8 Reinhard S, Mattern F. Detecting causal relationships in distributed computations: in search of the holy grail. Distributed Computing, 1994,7:149~174

9 He Le-tian, Sun yong-qiang. Characterizing causal relationships in distributed computations. Computer Engineering, 1996,22(2):1~5

10 Birman K, Schiper A, Stephenson P. Lightweight causal and atomic group multicast. ACM Terminal Operations Control System, 1991,9(3):272~314

11 Singhal M, Kshemkalynai A. An efficient implementation of vector clocks. Information Processing Letters, 1992,43:47~52

12 Charron-Bost B. Concerning the size of logical clocks in distributed systems. Information Processings Letters, 1991,39:11~16

13 Cheriton D R, Skeen D. Understanding the limitations of causality and totally ordered communications. ACM Operating Sys-

tems Reviews, 1993. 44～57

Concurrency Measures for Distributed Computations

HE Le-tian　SUN Yong-qiang

(*Department of Computer Science and Engineering　Shanghai Jiaotong University　Shanghai*　200030)

Abstract　This paper presents concurrency measures for distributed computations. These measures quantify concurrent computation time and synchronization delay using concepts such as local clocks based regional vector clocks, synchronization delay vector, and height of computation. The authors use no atomic events, thus need not devide event granularity. The measuring mechanism is independent of programs being measured. Their measures are independent of the programs to be measured, are feasible to be computed, simplify the management, and also take into account the performance of individual processors.

Key words　Distributed computation, concurrency measure, regional vector clock, synchronization delay.

Class number　TP302