doi:10.11918/j.issn.0367-6234.2015.01.011

虚拟时间及其在数据竞争检测中的应用

禹 振, 苏小红, 王甜甜, 马培军

(哈尔滨工业大学 计算机科学与技术学院, 150001 哈尔滨)

摘 要: 为将虚拟时间机制应用于数据竞争检测,提出描述虚拟时间3种基本实现形式的统一模型. 先建立分布式执行的抽象模型,在此模型下统一描述虚拟时间的3种基本实现形式,即标量时间系统、向量时间系统和矩阵时间系统,并以向量时间系统和矩阵时间系统为例介绍虚拟时间的4种优化技术,最后讨论将虚拟时间应用到共享内存并发系统的数据竞争检测中需要解决的问题以及4个应用实例. 结果表明,提出的模型能统一描述虚拟时间的不同实现形式,并能降低基于虚拟时间检测数据竞争的应用难度.

关键词: 虚拟时间;逻辑时间;向量时钟;并发缺陷;数据竞争

中图分类号: TP311 **文献标志码**: A **文章编号**: 0367-6234(2015)01-0068-07

Virtual time and its application to data race detection

YU Zhen, SU Xiaohong, WANG Tiantian, MA Peijun

(School of Computer Science and Technology, Harbin Institute of Technology, 150001 Harbin, China)

Abstract: Aiming at applying virtual time mechanism to data race detection, a model is proposed to uniformly describe different implementation forms of virtual time. We firstly establish an abstract model for a distributed execution. Based on this model, we give a unified description on virtual time's three basic implementation forms: scalar time system, vector time system and matrix time system. Furthermore, we take vector time system and matrix time system for examples to illustrate four optimization techniques for virtual time. At last, we discuss the problems to be solved when applying virtual time to data races detection in shared-memory concurrent systems and four application examples. The model proposed in this paper unifies different implementation forms of virtual time and reduces the difficulty of applying virtual time to data race detection.

Keywords: virtual time; logical time; vector clock; concurrency bug; data race

虚拟时间能精确刻画并发事件的发生顺序,为并发计算的性质分析[1]、缺陷检测[2]和故障恢复与调试[3]奠定基础,因此无论在非共享内存的分布式系统还是共享内存的多核/多处理器并发系统中都得到了广泛应用.

虚拟时间起源于分布式系统. 与内存分离的分布式系统不同,共享内存的多线程并发系统容易遭遇数据竞争[4-5],即对某一共享内存单元,存在来自不同线程的2个无序访问,且有1个为写访问. 检测数据竞争的关键在于检测对共享内存单元的2个访问是否是无序的. 虚拟时间可以为内存访问事件分配时钟,从而2个访问之间的顺序关系能通过比较时钟获得.

现已有针对不同应用场景的多种时钟机制. 本文略去各种时钟机制的应用背景,将它们统称为虚拟时间;建立抽象的分布式执行模型,统一描述虚拟时间的3种基本实现形式及相关优化与改进技术;讨论将虚拟时间应用到共享内存并发系统的数据竞争检测中需要解决的关键问题,并以4个应用实例为例展示基于虚拟时间的数据竞争检测系统的时钟分配和竞争检测机制.

1 问题描述

图1所示的分布式系统由3台异构计算机组

收稿日期: 2014-04-03.
基金项目: 国家自然科学基金(61173021;61202092).
作者简介: 禹 振(1987—),男,博士研究生;
苏小红(1966—),女,教授,博士生导师;
马培军(1963—),男,教授,博士生导师.
通信作者: 禹 振,yuzhen_3301@aliyun.com.

成，每个计算机上运行一个进程，进程之间消息的下标表示消息发生的顺序. 在分布式系统中，由于时钟震荡频率不同，不同处理器在同一时间的机器时间可能不同，而在不同时间的机器时间又可能相同，这就导致难以为不同处理器或者进程中发生的事件确定相对顺序. 例如，假设图 1 中进程 p_1、p_2 和 p_3 各自所在机器的震荡频率分别为 6、8 和 10，则基于局部时钟的通信图如图 2 所示. 其中当 p_3 给 p_2 发送消息 msg_3 时，由于 p_2 的时钟走的比 p_3 的慢，所以接收到 msg_3 时，本机的时钟 56 比 msg_3 附带的发送时钟 60 小. 这是不合理的，因为事件的接收时间应迟于其发送时间.

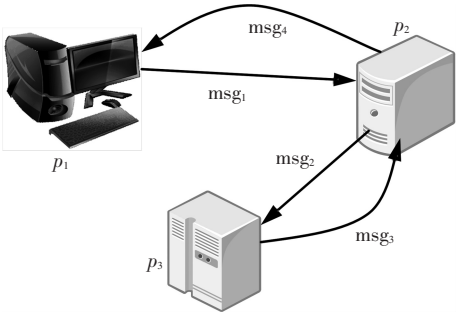


图 1　分布式系统示例

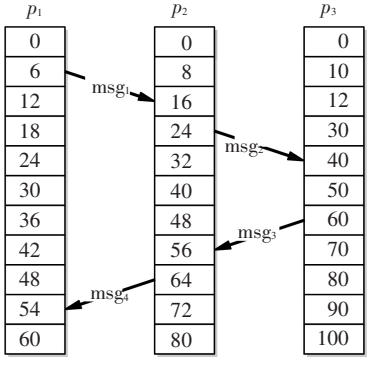


图 2　局部时钟的通信图

为解决这一问题，研究者们提出虚拟时间[6-11]，又称逻辑时间，作为分布式系统的全局时间，使得系统中的事件可以通过比较其发生时被赋予的时钟来确定事件的先后顺序.

2　模型与概念

一个分布式计算，即分布式程序，由 n 个异步执行的进程 p_1，p_2，…，p_i，…，p_n 构成. 每个进程执行 3 类操作：发送消息、接收消息和本地计算. 进程之间由通信网络互连、不共享内存、只通过传递消息进行通信.

定义 1（分布式执行的抽象模型）　单个进程 $p_i(1 \leqslant i \leqslant n)$ 的执行产生事件序列 $e_i^1, e_i^2, \cdots, e_i^x, e_i^{x+1}, \cdots$，记为 $E(p_i)$. $E(p_i)$ 中的事件具有自然序，即事件按照其发生时的顺序构成全序关系. 整个分布式计算的执行对应的事件集合 $\boldsymbol{E} = \{e \mid e \in E(p_i), i = 1,2,\cdots,n\}$，事件之间的顺序关系由 happens-before[10] 描述.

对于任意的事件 $e \in \boldsymbol{E}$，$\mathrm{Dsp}(e)$ 表示事件 e 的进程号，$\mathrm{Usp}(e, \mathrm{Dsp}(e))$ 表示事件 e 在事件序列 $E(p_{\mathrm{Dsp}(e)})$ 中的序号. 假如事件 e_i^x 是消息发送操作，则其发送的消息表示为 msg_i^x. 对于任意消息 msg_i^x，$\mathrm{Snd}(msg_i^x, i)$ 表示进程 p_i 发送 msg_i^x 的操作，$\mathrm{Rcv}(msg_i^x, j)$ 表示进程 p_j 接收进程 p_i 发送的 msg_i^x 的操作. 本文定义 happens-before 关系来为 $\boldsymbol{E}$ 中事件排序.

定义 2（happens-before 关系）　对于事件 $\varepsilon, \omega \in \boldsymbol{E}$，$\varepsilon$ 和 ω 具有 happens-before（简称 hb）关系，记为 $\varepsilon \rightarrow \omega$，当且仅当：1）$\mathrm{Dsp}(\varepsilon) = \mathrm{Dsp}(\omega) = i$，且 $\mathrm{Usp}(\varepsilon, i) < \mathrm{Usp}(\omega, i)$；2）$\mathrm{Dsp}(\varepsilon) = i$，$\mathrm{Dsp}(\omega) = j, i \neq j$，且存在消息 msg_i^x，使得 $\varepsilon = \mathrm{Snd}(msg_i^x, i)$ 和 $\omega = \mathrm{Rcv}(msg_i^x, j)$；3）存在 $\varphi \in \boldsymbol{E}$，使得 $\varepsilon \rightarrow \varphi$ 且 $\varphi \rightarrow \omega$.

hb 关系是反自反的、反对称的和可传递的. 如果两个事件不具有 hb 关系，则称之为并发事件. 按照 hb 关系对 $\boldsymbol{E}$ 中事件进行排序，则得到全体事件的一个偏序图.

定义 3（抽象虚拟时间系统）　一个虚拟时间系统的构成为：事件集合 $\boldsymbol{E}$、虚拟时间 $\boldsymbol{T}$ 和映射函数 C. 其中 C 是从 $\boldsymbol{E}$ 到 $\boldsymbol{T}$ 的映射 $\boldsymbol{T} = C(\boldsymbol{E})$，表示为集合 $\boldsymbol{E}$ 中的每个事件 e 分配一个虚拟时钟 $C(e)$，所有事件的虚拟时钟构成虚拟时间 $\boldsymbol{T}$.

定义 4（弱连贯和强连贯[12]）　对于一个虚拟时间系统和 $\boldsymbol{E}$ 中具有 hb 关系的两个事件 ε 和 ω，如果 $\varepsilon \rightarrow \omega \Rightarrow C(\varepsilon) < C(\omega)$，则称该虚拟时间系统为弱连贯的；如果 $\varepsilon \rightarrow \omega \Leftrightarrow C(\varepsilon) < C(\omega)$，则称之为强连贯的.

在虚拟时间系统的 3 种基本实现形式中，标量时间系统是弱连贯的，而向量时间系统和矩阵时间系统都是强连贯的.

3　虚拟时间系统

3.1　标量时间系统

标量时间系统由文献[6]提出. 在标量时间系统中，初始时刻各个进程 $p_s(1 \leqslant s \leqslant n)$ 的标量时钟为 0，进程 p_s 的当前时钟记为 C_s，消息 msg 的时钟为 C_{msg}，C_{msg} 为消息发送进程在发送 msg 时的时钟. 若 e_s^x 表示进程 p_s 正要执行的事件，则事件 e_s^x 的时钟根据如下规则更新：

1) 如果该事件不是消息接收事件，则 $C_s = C_s + d$（d 通常取 1），然后 $C(e_s^x) = C_s$；

2) 如果该事件是消息 msg 接收事件，则 $C_s = \max(C_s, C_{msg})$，$C_s = C_s + d$，然后 $C(e_s^x) = C_s$.

图 3 是基于标量时间系统的通信图. 其中，p_2 根据接收到的消息 msg_3 所携带的时钟 60，将自己的时钟从 56 改为 61；类似地，p_1 在接收到消息 msg_4 时，将自己的时钟改为 70. 标量时间不能保证如果两个事件的标量时钟存在大小关系，则两个事件具有 hb 关系. 例如图 3 中，假设 p_1 在标量时钟 18 时发生事件 a，p_2 在标量时钟 24 时发生事件 b，则虽然 $C(a) < C(b)$，但并不存在 $a \rightarrow b$. 因此标量时间系统是弱连贯的.

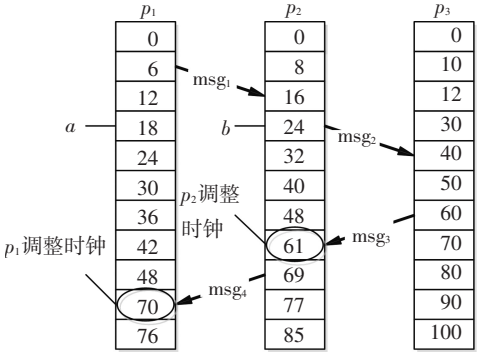


图 3 标量时间系统的通信图

3.2 向量时间系统

向量时间系统由文献[7－8]提出. 假设分布式计算中共有 n 个进程参与，则初始时刻各个进程 $p_v(1 \leqslant v \leqslant n)$ 的向量时钟为[0, 0, …, 0]（共 n 个 0）. 进程 p_v 的当前时钟记为 $\boldsymbol{C}_v$，它是一个 n 维向量，其第 v 维分量 $\boldsymbol{C}_v[v]$ 表示进程 p_v 的执行进展，其第 k 维分量 $\boldsymbol{C}_v[k]$ 表示进程 p_v 对进程 p_k 的最新知识，即 p_v 是否了解 p_k 发生了多少事件，p_v 是否了解 p_k 的执行进展. 消息 msg 携带消息发送进程发送消息时的时钟 C_{msg}，e_v^x 表示进程 p_v 正要执行的事件. 事件 e_v^x 的时钟根据如下规则更新：

1) 如果该事件不是消息接收事件，则仅更新 $\boldsymbol{C}_v$ 的第 v 维分量：$\boldsymbol{C}_v[v] = \boldsymbol{C}_v[v] + d$（$d$ 通常取 1），然后 $C(e_v^x) = \boldsymbol{C}_v$；

2) 如果该事件是消息 msg 的接收事件，则对于 $1 \leqslant k \leqslant n$，$\boldsymbol{C}_v[k] = \max(\boldsymbol{C}_v[k], C_{msg}[k])$；并且 $\boldsymbol{C}_v[v] = \boldsymbol{C}_v[v] + d$（$d$ 通常取 1），然后 $C(e_v^x) = \boldsymbol{C}_v$.

图 4 中，初始时刻 3 个进程 p_1，p_2 和 p_3 的向量时钟都为[000]. 当各自的第 1 个事件发生后，3 个进程的时钟分别变为[100]，[010] 和[001]. 当进程 p_1 的第 2 个事件发生时，根据向量时钟更新规则 1)，该事件被分配时钟[200]；由于该事件是消息发送事件，故发送的消息 msg 上携带时钟[200]. 当进程 p_2 接收到消息 msg 时，根据向量时钟更新规则 2)，p_2 的消息接收事件被分配时钟[220].

向量时间系统是强连贯的，任意 2 个事件 a 和 b，如果 $C(a) \leqslant C(b)$，则 $a \rightarrow b$；反之，如果 $a \rightarrow b$，则一定有 $C(a) \leqslant C(b)$. 在向量时间系统中，仅根据事件的向量时钟就可以判断两个事件是否具有 hb 关系. 例如图 4 中的事件 a 和 b，它们的向量时钟分别是[340] 和[232]，因为既不存在 $C(a) \leqslant C(b)$，又不存在 $C(b) \leqslant C(a)$，故它们是并发事件.

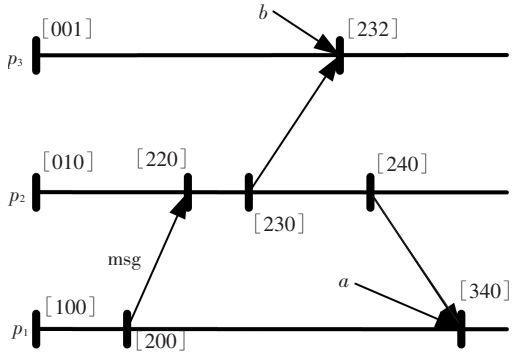


图 4 向量时间系统的通信图

3.3 矩阵时间系统

矩阵时间系统由文献[9－11]提出. 假设分布式计算中共有 n 个进程参与，则初始时刻各个进程 $p_m(1 \leqslant m \leqslant n)$ 的矩阵时钟为 $n \times n$ 的 0 矩阵. 进程 p_m 的当前时钟 $\boldsymbol{C}_m$ 为一个 $n \times n$ 矩阵，其中：元素 $\boldsymbol{C}_m[m, m]$ 表示进程 p_m 的执行进展，$\boldsymbol{C}_m[m, k]$（$1 \leqslant k, m \leqslant n$）表示进程 p_m 对进程 p_k 执行进展的最新知识，$\boldsymbol{C}_m[k, l]$（$1 \leqslant k, l, m \leqslant n$）表示进程 p_m 最近所知道的对进程 p_k 对进程 p_l 执行进展的最新知识的知识. 消息 msg 携带消息发送进程发送消息时的时钟 C_{msg}，e_m^x 表示进程 p_m 正要执行的事件，其时钟根据如下规则更新：

1) 如果该事件不是消息接收事件，则仅更新 $\boldsymbol{C}_m$ 的第 m 行第 m 列元素：$\boldsymbol{C}_m[m, m] = \boldsymbol{C}_m[m, m] + d$（$d$ 通常取 1），然后 $\boldsymbol{C}(e_m^x) = \boldsymbol{C}_m$；

2) 如果该事件是消息 msg 的接收事件，且该消息来自进程 $p_j(1 \leqslant j \leqslant n, j \neq k)$，则 $1 \leqslant k \leqslant n$，$\boldsymbol{C}_m[m, k] = \max(\boldsymbol{C}_m[m, k], C_{msg}[j, k])$；$1 \leqslant k, l \leqslant n$，$\boldsymbol{C}_m[k, l] = \max(\boldsymbol{C}_m[k, l], C_{msg}[k, l])$；$\boldsymbol{C}_m[m, m] = \boldsymbol{C}_m[m, m] + d$（$d$ 通常取 1），然后 $\boldsymbol{C}(e_m^x) = \boldsymbol{C}_m$.

图 5 中，初始时刻 3 个进程 p_1，p_2 和 p_3 的矩阵时钟都为 $\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$. 当各自的第 1 个事件发生

后，3 个进程的时钟分别变为 $\begin{pmatrix}1&0&0\\0&0&0\\0&0&0\end{pmatrix}$，$\begin{pmatrix}0&0&0\\0&1&0\\0&0&0\end{pmatrix}$ 和 $\begin{pmatrix}0&0&0\\0&0&0\\0&0&1\end{pmatrix}$. 当进程 p_1 的第 2 个事件发生时，根据矩阵时钟更新规则 1)，该事件被分配时钟 $\begin{pmatrix}2&0&0\\0&0&0\\0&0&0\end{pmatrix}$；由于该事件是消息发送事件，故发送的消息 msg 上携带时钟 $\begin{pmatrix}2&0&0\\0&0&0\\0&0&0\end{pmatrix}$. 当进程 p_2 接收到消息 msg 时，根据矩阵时钟更新规则 2)，p_2 的消息接收事件被分配为时钟 $\begin{pmatrix}2&0&0\\2&2&0\\0&0&0\end{pmatrix}$.

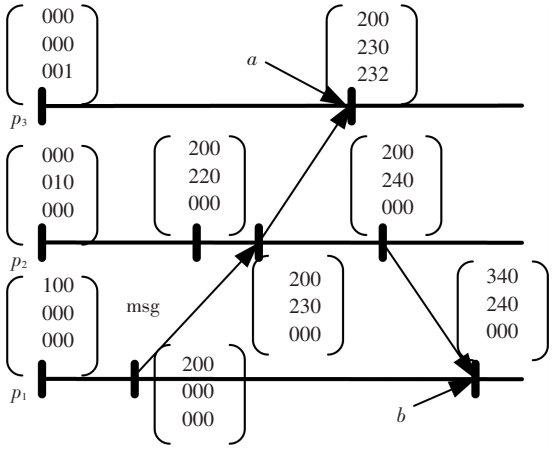


图 5　矩阵时间系统的通信图

矩阵时间系统是强连贯的，任意事件 a 和 b，如果 $a \to b$ 当且仅当 $C(a) \leqslant C(b)$. 例如图 5 中的事件 a 和 b，它们的矩阵时钟分别是 $\begin{pmatrix}2&0&0\\2&3&0\\2&3&2\end{pmatrix}$ 和 $\begin{pmatrix}3&4&0\\2&4&0\\0&0&0\end{pmatrix}$，既不存在 $C(a) \leqslant C(b)$，又不存在 $C(b) \leqslant C(a)$，因此它们是并发事件.

4　虚拟时间的改进与优化技术

4.1　丢弃过时时钟

当某个事件发生时，并不需要与所有已发生的事件进行时钟比较，而只需要与一些事件进行比较即可，因为另一些事件在该事件发生前或者发生时已经“过时”.

4.1.1　列投影技术[12]

在某个矩阵时间系统中，事件 e_m^x 的时钟 $\boldsymbol{C}(e_m^x)$ 为 $n \times n$ 的矩阵. 如果 $\min\limits_{k=1\cdots n}(\boldsymbol{C}(e_m^x)[k, l]) = t$，则表示：1) 进程 p_m 知道所有进程 p_k（包括 p_m 自己）都知道进程 p_l 已进展到 t，或者说执行了 t 个事件；2) 此后任何发生的事件 e 的时钟 $C(e)[l, l] \geqslant t$；3) 进程 p_l 的第 $1 \sim t$ 个事件可以丢弃，其与未来发生的事件的 hb 关系已经确定.

图 6 使用列投影技术丢弃过时时钟，矩阵下方的横线表示对矩阵的每一列求取最小值. 进程 p_3 的第 2 个事件 e_3^2 发生时，被赋予矩阵时钟 $\begin{pmatrix}2&0&0\\2&3&0\\2&3&2\end{pmatrix}$. 由于 $\min\limits_{k=1,2,3}(\boldsymbol{C}(e_3^2)[k, 1]) = 2$，因此 p_3 知道所有进程 p_1，p_2，p_3 都知道进程 p_1 已执行了 2 个事件. 此后任何事件 e 的时钟的第 1 行第 1 列肯定 $\geqslant 2$，且 $e_1^2 \to e$. 因此 p_1 的前 2 个事件 a 和 b 在今后的时钟比较时可以忽略.

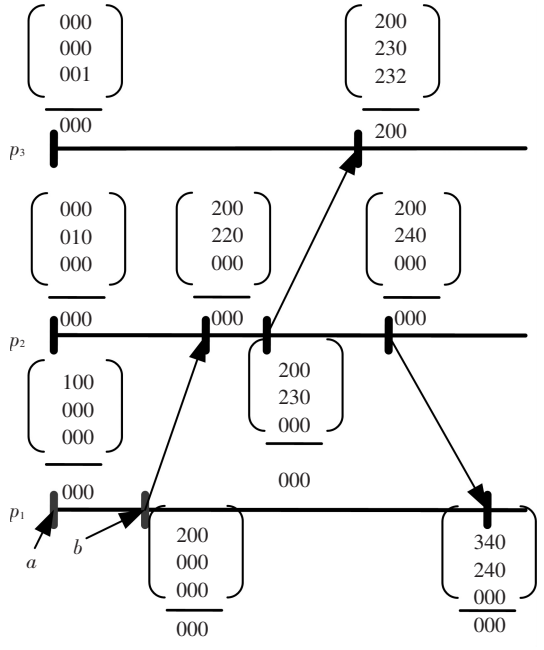


图 6　列投影技术丢弃过时时钟

4.1.2　探询（Snooping）技术[13]

列投影技术对过时信息的计算依赖于事件的矩阵时钟，而后者只能根据进程内执行进展或者进程间消息传递而更新，速度较慢.这就导致列投影技术对系统全局进展的最新知识较滞后. 鉴于此，探询技术不再被动等待进程间的消息传递，而是主动探询各个进程的最新进展，从而获得对系统全局进展的实时知识.

图 7 使用探寻技术对图 4 所示的通信图进行过时时钟丢弃. 进程 p_1 的第 2 个事件的时钟在图 4 中是[200],在图 7 中的 snooping 时钟是 $\begin{pmatrix}2&0&0\\0&1&0\\0&0&1\end{pmatrix}$. 探寻技术在计算事件 b 的 snooping 时钟时,主动去询问进程 p_2 和 p_3 的当前时钟,分别得到[010]和[001],然后结合 p_1 的当前时钟构成 b 的 snooping 时钟. 一旦得到事件的 snooping 时钟,探询技术就采用列投影技术来计算可被丢弃的时钟. 在 p_3 的第 2 个事件发生时,探寻技术丢弃进程 p_1 的前 2 个事件的时钟,这与列投影技术一样;在 p_1 的第 3 个事件发生时,探寻技术丢弃 p_2 的前 3 个事件的时钟,这是列投影技术无法做到的.

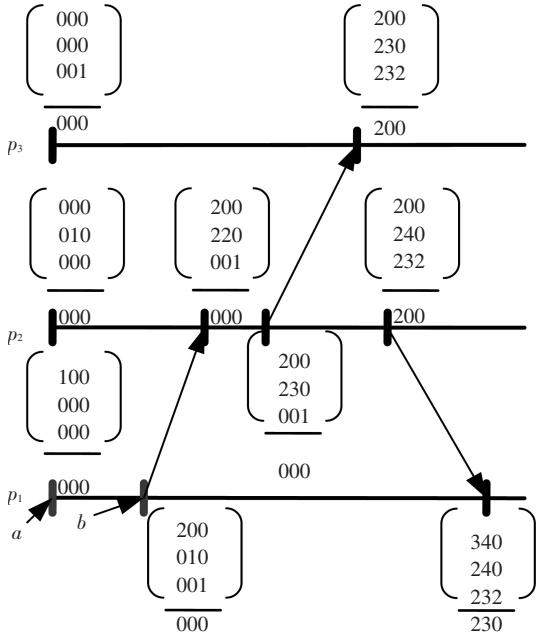


图 7 探询技术丢弃过时时钟

4.2 减少消息带宽

文献[14]发现如果从上次向进程 p_j 发送消息以来,进程 p_i 的时钟向量的第 $i_1, i_2, \cdots, i_{n_1}$ 个分量变为 $v_1, v_2, \cdots, v_{n_1}$,则下次 p_i 向 p_j 发送消息时,只需发送发生变化的分量 $<(i_1, v_1), (i_2, v_2), \cdots, (i_{n_1}, v_{n_1})>$. 进程 p_j 接收到该压缩格式的向量时钟后,如图 8 所示更新自己的时钟:对于 $k = i_1, i_2, \cdots, i_{n_1}$, $C_j[k] = \max(C_j[k], v_k)$.

图 8 使用 Singhal 技术传递消息. 其中 p_3 传递第 3 个消息给 p_2 时,消息上附带的信息是 $<(3, 4), (4, 1)>$. 这是由于上次 p_3 给 p_2 传递消息时,p_3 的时钟是[0020],而当前 p_3 时钟是[0041],向量时钟的第 3 项从 2 变为 4,第 4 项从 0 变为 1. p_2 接收到该消息后,将自己的时钟从[1320]变为[1441].

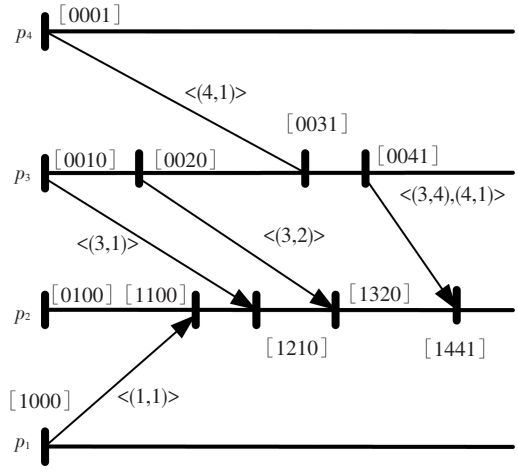


图 8 使用 Singhal 技术传递消息

4.3 可折叠时钟

可折叠时钟[15](accordion clock)只维护对于当前活跃的进程的知识. 对于尚未启动或者已经结束的进程,可折叠时钟不维护对于它们的任何信息. 可折叠时钟的时钟长度随着分布式计算中进程的启动和结束而伸展与收缩,从而总是维护对于"必要"进程的知识.

可折叠时钟的分量具有形式 (s, pid),其中 s 表示进程 pid 的本地时钟. 图 9 使用可折叠时钟为分布式计算中的事件进行排序. 其中,初始时刻 3 个进程的 p_1, p_2 和 p_3 的可折叠时钟都为空. 当各自的第 1 个事件发生后,3 个进程的时钟分别变为[(1, 1)],[(1, 2)]和[(1, 3)]. 当进程的第 2 个事件发生时,该事件被分配时钟[(2, 1)];由于该事件是消息发送事件,故发送的消息 msg 上携带时钟[(2, 1)]. 当进程 p_2 接收到消息 msg 时,p_2 的消息接收事件被分配时钟[(2, 1), (2, 2)].

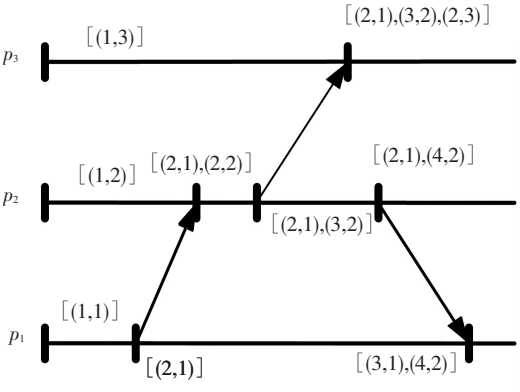


图 9 可折叠时间系统的通信图

5　虚拟时间的数据竞争检测

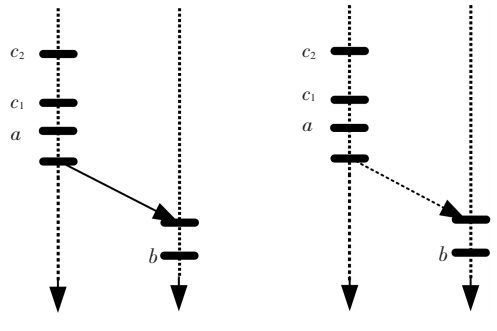
将虚拟时间应用在数据竞争检测中，面临 3 个问题：1）哪些事件是消息发送和接收事件；2）哪些事件需要为其分配时钟；3）哪些事件需要进行并发检测. 已有研究[16-19]将以下事件作为消息发送/接收事件：fork/线程入口点，线程退出点/join，unlock/lock，signal/wait；为以下事件分配时钟：消息发送/接收事件，内存访问事件；对以下事件进行并发检测：内存访问事件.

本文介绍了基于向量时间的 4 个数据竞争检测系统，即 Dij$^+$系统，FastClock 系统，Weaker-than 系统和 Threadset 系统.

5.1　Dij$^+$系统

Dij$^+$系统为每个共享变量 x 维护 2 个时钟向量 $\boldsymbol{aw}_x$ 和 $\boldsymbol{ar}_x$，分别代表各个线程对 x 的最新写访问和读访问. Dij$^+$如下更新共享变量的 2 个时钟向量：在 t 的某个时间帧中，a 是对 x 的第 1 个访问，如果 a 是读访问，则 $\boldsymbol{ar}_x[t] = C_t[t]$；如果 a 是写访问，则 $\boldsymbol{aw}_x[t] = C_t[t]$. Dij$^+$ 在 t 执行完 a 后，检测数据竞争：是否存在某个线程 u，使得如果 a 是读访问，则 $\boldsymbol{aw}_x[u] > C_t[u]$；如果 a 是写访问，则 $\boldsymbol{aw}_x[u] > C_t[u]$ 或者 $ar_x[u] > C_t[u]$.

Dij$^+$能保证若 x 上存在数据竞争，则至少一个会被检测出来. 图 10 解释了 Dij$^+$的时钟更新机制，其中所有事件都是对 x 的写访问事件. 在图 10(a)中，来自不同线程的两个事件 a 和 b 具有 hb 关系，则按照 hb 定义，c_1 和 c_2 也与 b 具有 hb 关系，因此无需保存 c_1 和 c_2 对 x 的访问信息. 在图 10(b)中，a 和 b 是并发的，而且 c_1 和 c_2 也可能与 b 是并发的；这种情况下 Dij$^+$可能漏检 c_1（或 c_2）和 b 构成的数据竞争，但能保证检测到 a 和 b 这一对数据竞争.



(a) a 先发生于 b　　(b) a 和 b 并发

图 10　Dij$^+$时钟更新机制的原因

5.2　Fastclock 系统

若有 n 个线程，则 Dij$^+$对线程 t 的每个读写操作，需要耗费 $O(n)$ 时间来检测它是否与其他线程的读写操作存在数据竞争. 假如事先知道某个线程 w 对共享变量 x 的访问是到当时为止所有线程对 x 的最新访问，则当线程 t 对 x 进行访问 a 时，只需进行以下比较来检测数据竞争：$\boldsymbol{aw}_x[w] > C_t[w]$ 或者 $\boldsymbol{ar}_x[w] > C_t[w]$. 此比较的时间复杂度是常量级的，即 $O(1)$. 因此 $\boldsymbol{aw}_x$ 和 $\boldsymbol{ar}_x$ 只需存储最新读写共享变量的线程的本地时钟和线程号，称为 epoch（形式为 $c@t$），存储空间也由 $O(n)$ 降为 $O(1)$.

FastClock 时钟更新机制如下：如果多个读访问并发访问 x，则 $\boldsymbol{ar}_x$ 是 epoch 形式的向量时钟，以同时记录多个读访问；如果多个读访问顺序访问 x，则 $\boldsymbol{ar}_x$ 是标量时钟；如果多个读访问并发访问 x 后，遇到对 x 的写访问，则 $\boldsymbol{ar}_x$ 从向量时钟变为变量时钟；无论多个写访问顺序或者并发访问 x，$\boldsymbol{aw}_x$ 都是标量时钟. FastClock 竞争检测机制与 Dij$^+$类似.

5.3　Weaker-than 系统

对于给定的 2 个事件 p 和 q，以及将来发生的任何事件 r，如果 IsRace(q, r) 蕴含 IsRace(p, r)，则说事件 p 相对于事件 q 来说受到（锁）的保护更弱（weaker-than），更易于与其他事件构成数据竞争. 其中 IsRace(e_1, e_2) 用于判断 2 个访存事件 e_1 和 e_2 是否构成数据竞争对. 对于 p 和 q，Weaker-than 只保留更弱者 p，而丢弃 q，并且将来发生的事件 r，都将与 p 进行 hb 关系比较，而不与 q 进行比较. Weaker-than 系统与 Dij$^+$ 系统一样，保证如果共享变量上 x 上存在数据竞争，则它至少会检测到其中一个.

5.4　Threadset 系统

与 Dij$^+$类似，Threadset 为每个共享变量 x 维护一个并发访问集 $\boldsymbol{S}_x$，其中保存着 epoch 形式的时钟. 与 Dij$^+$不同的是，$\boldsymbol{S}_x$ 不区分共享变量的读访问与写访问，因此可能将 2 个读访问报告为数据竞争. $\boldsymbol{S}_x$ 使用可折叠时钟表示，其大小随并发访问的数目增大或者缩小. $\boldsymbol{S}_x$ 中使用 hb 关系来检测两个访问是否并发：当线程 t 读访问或者写访问 x 时，Threadset 将 $(t, C_t(t))$ 添加到 $\boldsymbol{S}_x$ 中，然后检查是否存在 $(u, k) \in \boldsymbol{S}_x$，使得 $k \leqslant C_t(u)$；如果存在的话，则说明 (u, k) 代表的访问与本访问具有 hb 关系，故从 $\boldsymbol{S}_x$ 中删除 (u, k)，$\boldsymbol{S}_x$ 中剩余元素都是有可能相互并发的访问. 但单独使用 $\boldsymbol{S}_x$，Threadset 不能判断是否有数据竞争发生，还需要 x 的锁集信息 L_x. Threadset 结合 hb 算法和锁集算法[20-21] 来检测数据竞争.

6 结 论

1)提出虚拟时间和虚拟时钟的概念,给出虚拟时间系统的形式化定义,对虚拟时间和虚拟时钟的概念作出区分.

2)建立抽象的分布式执行模型,在同一框架下统一描述虚拟时间的3种基本实现形式和相关改进优化技术.

3)总结将虚拟时间应用到数据竞争检测时需要解决的关键问题,以4个数据竞争检测器为例说明基于虚拟时间的数据竞争检测系统的时钟分配和竞争检测机制.

4)本模型清晰准确地刻画虚拟时间的不同实现形式,有助于降低基于虚拟时间检测数据竞争的应用难度.

参考文献

[1] 罗军, 王宏, 李文生. 基于向量时钟模型的 NoSQL 最终一致性的研究[J]. 计算机工程与应用, 2013, 49(23): 100-102.

[2] 宋庆祯. 开放式协同环境下分布式事务提交和恢复机制的研究[D]. 广西:广西大学, 2005.

[3] 刘磊, 黄河, 唐志敏. 支持多核并行程序确定性重放的高效访存冲突记录方法[J]. 计算机研究与发展, 2011, 49(1): 64-75.

[4] LU S, PARK S, SEO E, et al. Learning from mistakes: a comprehensive study on real world concurrency bug characteristics[J]. ACM SIGPLAN Notices, 2008, 43(3): 329-339.

[5] 霍玮,于洪涛,冯晓兵,等. 静态检测中断驱动程序的数据竞争[J]. 计算机研究与发展, 2011, 48(12): 2290-2299.

[6] LAMPORT L. Time, clocks, and the ordering of events in a distributed system[J]. Communications of the ACM, 1978, 21(7): 558-565.

[7] MATTERN F. Virtual time and global states of distributed systems[J]. Parallel and Distributed Algorithms, 1989, 1(23): 215-226.

[8] FIDGE C. Logical time in distributed computing systems[J]. Computer, 1991, 24(8): 28-33.

[9] FISCHER M J, MICHAEL A. Sacrificing serializability to attain high availability of data in an unreliable network[C]//Proceedings of the 1st ACM SIGACT-SIGMOD symposium on Principles of database systems. New York: ACM, 1982: 70-75.

[10] WUU G T J, BERNSTEIN A J. Efficient solutions to the replicated log and dictionary problems[J]. Operating systems review, 1986, 20(1): 57-66.

[11] SARIN S K, LYNCH N A. Discarding obsolete information in a replicated database system[J]. IEEE Transactions on Software Engineering. 1987, 13(1): 39-47.

[12] RAYNAL M, SINGHAL M. Logical time: Capturing causality in distributed systems[J]. Computer, 1996, 29(2): 49-56.

[13] DE BOSSCHERE K, RONSSE M. Clock snooping and its application in on-the-fly data race detection[C]// Proceedings of the 3rd International Symposium on Parallel Architectures, Algorithms, and Networks. Piscataway, NJ: IEEE, 1997: 324-330.

[14] SINGHAL M, KSHEMKALYANI A. An efficient implementation of vector clocks[J]. Information Processing Letters, 1992, 43(1): 47-52.

[15] CHRISTIAENS M, DE BOSSCHERE K. Accordion clocks: Logical clocks for data race detection[C]// Proceedings of the 7th International European Conference on Parallel Processing. Berlin, Heidelberg: Springer, 2001: 494-503.

[16] CHOI J D, LEE K, LOGINOV A, et al. Efficient and precise datarace detection for multithreaded object-oriented programs[J]. ACM SIGPLAN Notices, 2002, 37(5): 258-269.

[17] FLANAGAN C, FREUND S N. FastTrack: efficient and precise dynamic race detection[J]. ACM Sigplan Notices, 2009, 44(6): 121-133.

[18] POZNIANSKY E, SCHUSTER A. MultiRace: efficient on-the-fly data race detection in multithreaded C++ programs[J]. Concurrency and Computation: Practice and Experience, 2007, 19(3): 327-340.

[19] YU Y, RODEHEFFER T, CHEN W. Racetrack: efficient detection of data race conditions via adaptive tracking[J]. ACM SIGOPS Operating Systems Review, 2005, 39(5): 221-234.

[20] ZHOU P, TEODORESCU R, ZHOU Y. HARD: Hardware-assisted lockset-based race detection[C]// Proceedings of the 13th International Symposium on High Performance Computer Architecture. Piscataway, NJ: IEEE, 2007: 121-132.

[21] XIE Xinwei, XUE Jingling, ZHANG Jie. Acculock: Accurate and efficient detection of data races[J]. Software: Practice and Experience, 2013, 43(5): 543-576.

(编辑 张 红)