分布式计算的快照算法*

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 快照算法是很多分布式应用的基础. 在假设了全局时钟的情况下,本文说明了计算通道状态的多种方案. 使用一致割作为虚拟的全局时刻,说明了这些方案适用于现有的快照算法. 文中还提出了一个适合于各种通讯模型的快照算法.

关键词 分布式计算,快照算法,一致割,全局状态,通讯模型.

中图法分类号 TP316

分布式计算的全局状态是所有进程和通道局部状态的集合. 由于缺乏全局时钟,不可能同时记录这些局部状态,分布式快照[1]被提出来构造系统的一致全局状态. 在各种不同的通讯模型下,现已提出了很多的快照算法[1~5],这些算法在消息复杂度、空间复杂度、计算复杂度和对原有计算的影响及灵活性上有较大的差别.

在假设了全局时钟的情况下,本文说明了计算通道状态的多种方案. 使用一致割作为虚拟的全局时刻,说明了这些方案适用于现有的快照算法. 我们还提出了一个适合于各种通讯模型的快照算法.

1 系统模型和定义

一个分布式系统 $D=(P,C)$,其中 $P=\{p_1,p_2,\ldots,p_n\}$ 是进程的集合,n 是进程数,$C=\{c_{i,j};\forall\ 1\leqslant i,j\leqslant n,i\neq j\}$ 是进程间通道的集合. 系统中没有全局时钟,进程间仅以消息传递的方式通讯,通讯是可靠的,通讯延迟是不确定的有限时间.

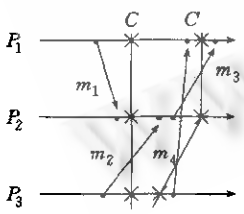

图1 进程时间图

进程的局部计算是由局部状态和原子事件组成的序列,事件区分为 3 类:发送事件、接收事件和原子事件. 用 $E=E_1\cup E_2\cup\ldots\cup E_n$ 表示计算中所有事件的集合,用 $send(m)$ 和 $recv(m)$ 表示消息 m 的发送和接收. 事件间的关系(用$\rightarrow$表示)是一个严格的偏序关系[6],又称因果关系,是满足如下条件的最小传递关系:①e,e' 是同一进程上先后发生的事件,则 $e\rightarrow e'$;②$send(m)\rightarrow recv(m)$. 常常将分布式计算表示为进程时空图[6](图 1).

---

* 作者贺乐天,1971 年生,博士生,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,主要研究领域为计算理论,新型语言.

本文通讯联系人:贺乐天,上海 200030,上海交通大学计算机科学与工程系

本文 1996-09-19 收到修改稿

分布式快照是进程和通道局部快照的集合. 割是事件集合 E 的子集,包含每个进程快照点前的所有事件. 一致割是那些正确反映了因果关系的割,即:若 $e \rightarrow e'$,$e' \in C$,则 $e \in C$. 图 1 中,C 是一致割,C' 则是不一致的. 一致割可看作分布式计算中虚拟的全局时刻.

在快照点时可以记录进程和通道的局部状态,为保证全局状态一致,局部快照点构成的割应是一致的. 分别用 GS,s_i 和 $cs_{i,j}$ 表示全局状态、进程 p_i 的状态和通道 $c_{i,j}$ 的状态,则 $GS=\{s_i:1\leqslant i\leqslant n\}\cup\{cs_{i,j}:\forall\ 1\leqslant i,j\leqslant n,i\neq j\}$.

2　通道状态计算方案

假设系统中有一个初始进程 P_{init},其作用是启动快照算法,收集各进程和通道的状态及终止算法. 假设存在全局时钟,各进程可约定在某一时刻 t,在执行下一步操作之前同时记录其局部状态. 显然,在时刻 t 和 t 之后发生的事件不可能影响到 t 之前的事件,在时刻 t 的局部快照点构成的割是一致的. 我们可用多种方案计算时刻 t 时通道中的消息.

方案 1.[3] 每个进程 p_i 维护 2 个消息队列数组 $sent_i$ 和 $recd_i$,其中 $sent_i[j](j\neq i)$ 和 $recd_i[j](j\neq i)$ 分别记录 p_i 在通道 $c_{i,j}$ 上发送的所有消息和在通道 $c_{j,i}$ 上收到的所有消息. 在快照点时,p_i 记录局部状态 s_i,向 p_{init} 发送消息 $(s_i,sent_i,recd_i)$. p_{init} 收到所有进程的消息后计算通道 $c_{i,j}$ 的状态:$cs_{i,j}=sent_i[j]-recd_j[i](i\neq j)$. 得到的全局状态 GS 是一致的. 本方案需要 n 条消息,p_{init} 需要 $n\times(n-1)$ 次队列差运算来计算通道状态,但各进程需要很大的空间来储存消息历史.

方案 2.[4] 消息在发送时被打上时间邮戳. 每个进程 p_i 记录发送和接收到的消息数 $count_i$(初值为 0,接收消息时 $count_i=count_i-1$,发送消息时 $count_i=count_i+1$). 在快照点时,p_i 记录局部状态 s_i 和 $count_i$ 的当前值,向 p_{init} 发送消息 $(s_i,count_i)$,此后 p_i 每收到一条时戳小于 t 的通道 $c_{j,i}$ 上的消息 m,便向 p_{init} 发送消息 (j,i,m). p_{init} 收到所有进程发来的 $(s_i,count_i)$ 后,计算正在传输的消息数 $in\_transmit$(初值为 0):

$$in\_transmit=in\_transmit+\Sigma_{i=1}^{n}count_i.$$

当 p_{init} 收到一条 (i,j,m) 消息时,修改 $in\_transmit$ 和通道 $c_{i,j}$ 的状态 $cs_{i,j}$(初值为 $\varnothing$):

$$in\_transmit=in\_transmit-1,cs_{i,j}=cs_{i,j}\cup\{m\}.$$

当 $in\_transmit=0$ 时,计算 GS 就得到一个一致全局状态. 本方案需要(n+快照时通道中的消息数量)条消息,各进程仅需一个整数计数器,p_{init} 上的计算较简单.

方案 3. 消息在发送时被打上时间邮戳. 进程 p_i 维护向量 $sent_i$,其中 $sent_i[j](j\neq i)$ 表示 p_i 在通道 $c_{i,j}$ 上发送的消息数,$sent_i[i]$ 表示 p_i 在所有输入通道上接收到的消息数. 在快照点时,p_i 记录局部状态 s_i,向 p_{init} 发送消息 $(s_i,sent_i)$,p_i 记录和计数所有收到的在 t 之前发送的消息,计数器为 $recd_i$. p_{init} 收到所有进程的 $(s_i,sent_i)$ 消息后,对每个进程 p_i 计算发往 p_i 的传输中的消息数 $in\_transmit_i$:

$$in\_transmit_i=\Sigma_{j=1,\ldots,n,j\neq i}sent_j[i]-sent_i[i].$$

然后,p_{init} 向进程 p_i 发送消息 $(in\_transmit_i)$. p_i 接收消息 $(in\_transmit_i)$,当 $recd_i=in\_transmit_i$ 时,p_i 将记录的通道状态 $\{cs_{j,i}:\forall\ 1\leqslant j\leqslant n,j\neq i\}$ 传递给 p_{init},计算 GS 也可得到一致的全局状态. 本方案需要 $3n$ 条消息,进程需要一定的空间来保存通道中的消息,p_{init} 上的计算也

很简单. 使用类似的计数向量,文献[4]用控制环绕收集通道状态.

全局时间在上面3个方案中的作用是:便于各进程同时取局部状态和区分快照前后发送的消息. 为了区分快照前后的消息,在消息上附着一位的信息就足够了.[3]

定理 2.1. 在一致割下,方案1/2/3记录了所有通道的状态.(证明略)

这样,只要快照算法的快照点构成了一致割,假设了全局时钟的通道状态计算方案就可以用在这些算法中,因此,研究快照算法的核心在于构造一致割.

3 快照算法

进程按编号顺序(为简化描述,设 p_{init} 是计算进程)构成一个有向环路,并定义:

$$succ(p_i)=\text{if } (i=n) \text{ then } p_1 \text{ else } p_{i+1}.$$

算法 3.1. 快照由 p_{init} 向自己发送 *token* 消息开始;任一进程 p_i 在收到 *token* 消息时,若未取局部快照,则记录局部状态,并标记所有随后发出的消息,然后向 $succ(p_i)$ 传递 *token* 消息,若接收到 *token* 消息时,进程已取局部快照,则仅向 $succ(p_i)$ 传递 *token* 消息;任一进程 p_i 在收到一条被标记的消息前,若未取局部快照,则记录其局部状态,并标记所有随后发出的消息,否则,仅接收该消息. 当 p_{init} 收到其它进程传来的消息时,使用前述的任一方案计算通道状态.

定理 3.2. 各进程收到 *token* 消息或第1条被标记消息时的快照点构成的割 C 是一致的.

证明:假设某进程 p_i 在通道 $c_{j,i}$ 上收到一条消息 m,$recv(m)\in C$,$send(m)\notin C$. 进程 p_j 在发送 m 前已取局部快照(否则 $send(m)\in C$),由上面的算法,进程在局部快照后要标记发出的消息,故 m 是一条被标记的消息. 进程 p_i 在收到一条被标记的消息前应取局部快照,故 p_i 在收到 m 前将开始快照,这样 $recv(m)\notin C$,与假设矛盾. □

为取得一致割,算法需要 n 条 *token* 消息和在消息上附上一位的标记,可适用于各种通讯模型. *token* 沿有向环路顺序传递,将遍历所有的进程并回到 p_{init},所有进程都已进行了局部快照,算法是可终止的. 计算通道状态的3种方案都适用于算法3.1,消息复杂度为 $O(n)$(方案2除外). 对于非FIFO的系统,Taylor[5]说明了快照算法或者会拟制原有的计算,或者需要在计算原有的消息上附着控制信息,算法3.1属于后一类. *token* 的传递可采用其它的图遍历结构(如生成树等).

Chandy等的快照算法[1]适合于通讯的系统. Lai等[3]的算法适合于非FIFO通讯的系统,用第1种方案计算通道状态,需要一些措施来保证终止.[3,4]Mattern[4]用第2种方案改进了文献[3]的空间复杂度. 算法3.1使用方案3时,较之传统算法[1],消息复杂度大大降低,空间和计算复杂度基本相同;较之文献[3]中的算法,不需要附加的措施来保证终止,空间复杂度大大降低,消息复杂度和计算复杂度基本相同.

4 结 论

使用一致割作为虚拟的全局时刻,假设了全局时钟的计算通道状态的多种方案适用于现有的快照算法. 本文提出了一个消息复杂度较传统算法[1]小和空间复杂度较文献[3]中算

法小,适合于各种通讯模型的快照算法.我们可以在假设全局时钟的情况下设计算法,然后寻求全局时钟的替代物,将全局时钟的算法应用到分布式系统中去,本文的思想具有方法论意义.

致谢　审稿人和编辑对本文初稿的改进和建议大大改善了本文的面貌,在此谨致感谢.

参考文献

1 Chandy K M, Lamport L. Distributed snapshots: determining global states of distributed systems. ACM Trans. Comput. Systems, 1985,**3**(**1**):63～75.

2 Acharya A, Badrinath B R. Recording distributed snapshots based on causal order of message delivery. Inform. Proces. Lett., 1992,**44**:317～321.

3 Lai T H, Yang T H. On distributed snapshots. Inform. Proces. Lett., 1987,**25**:153～158.

4 Mattern F. Efficient algorithms for distributed snapshots and global virtual time approximation. Journal of Parallel and Distributed Computing, 1993,**18**:423～434.

5 Taylor K. The role of inhibition in asynchronous consistent-cut protocols. In: Bermond J-C, Raynal M eds. Proc. of the 3rd Intl. Workshop on Distributed Algorithms, LNCS 392, Berlin / New York: Springer-Verlag, 1989. 280～291.

6 Lamport L. Time, clocks and ordering of events in a distributed systems. Comm. ACM, 1978,**21**(**7**):558～565.

ON SNAPSHOT ALGORITHMS IN DISTRIBUTED COMPUTATIONS

HE Letian　SUN Yongqiang

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

Abstract　Snapshot algorithms are fundamental for many distributed applications. This paper shows the several schemes for computing states of channels under the assumption of global clock. Taking consistent cut for the virtual global instant, the authors show that these schemes are suitable for the existing snapshot algorithms. A new algorithm is also presented, which is applicable for the various communication models.

Key words　Distributed computation, snapshot algorithm, consistent cut, global state, communication model.

Class number　TP316