一种基于松弛循环差集的高性能分布式互斥算法

李美安,刘心松,王 征
(电子科技大学计算机学院 8010 研究室,四川成都 610054)

摘 要: 基于竞争的分布式互斥算法以请求集为基础. 对称的请求集才能产生对称、公平的分布式互斥算法. 本文首先证明了循环请求集与松弛循环差集具有等价性,并在此基础上提出了一种基于松弛循环差集的对称请求集生成算法. 在提出动态令牌和请求集重构概念的基础上,本文将 Maekawa 类分布式互斥的同步时间降为 T,节点容错能力达到 $N-1$,一次临界区执行所需交换的消息数降为 $2m-3m$, m 是请求集大小.

关键词: 松弛循环差集; 分布式; 互斥; 算法
中图分类号: TP393 **文献标识码**: A **文章编号**: 0372-2112 (2007) 01-0058-06

A High Performance Distributed Mutual Exclusion Algorithm Based on Relaxed Cyclic Difference Set

LI Mei-an, LIU Xin-song, WANG Zheng
(8010 *R&D Group, University of Electronic Science and Technology of China, Chengdu, Sichuan* 610054, *China*)

Abstract: A new quorum generation algorithm has been presented. A symmetric quorum can be generated based on this algorithm. And the size of the quorum is about 2 *sqrt(N). In the basis on the message of transfer, the synchronization delay time of the distributed exclusion has been reduced to T. Through the quorum re-constructing, the sites fault-tolerance of Maekawa type distributed mutual exclusion algorithm has been increased to N-1. The message number exchanged of the CS execution will be reduced to 2m-3m, our distributed mutual exclusion algorithm in this paper has reduced the message complexity of Maekawa type distributed mutual exclusion algorithm m messages, m is the quorum size.

Key words: relaxed cyclic difference set; distributed; mutual exclusion; algorithm

1 引言

基于竞争的分布式互斥算法主要以 Lamport[1] 提出的逻辑时戳 (Logical Timestamp) 和 Maekawa[2] 提出的请求集 (Quorum) 概念为基础,以优化度量分布式互斥算法的三个性能指标为目标. 如优化消息复杂度的算法[3~11],优化可靠性的算法[12,13],以及优化同步延迟的算法[14,15].

Luk 在文献[16]中提出并证明了基于循环对称区组设计和循环差集生成对称请求集的可能性,并列出了节点数为 4 到 111 的分布式系统的循环基. 但是文中没有明确提出请求集的生成算法. WEE 在文献[17]中提出一种不限制请求集相交节点数的无结构请求集生成算法,将分布式互斥的消息复杂度降到了 $O(N^{0.63})$. 但当 $N>250$ 时,其请求集长度大于 $2\sqrt{N}$.

Maekawa 算法的消息复杂度为 $O(\sqrt{N})$,一次临界区执行所需交换的消息数为 $3m-5m$. Guohong[14] 提出一种与请求集生成算法无关的分布式互斥算法,将分布式互斥算法的同步时间降低到 T,一次临界区执行所需交换的消息数为 $3m-6m$.

本文将在证明循环请求集与循环松弛差集等价的基础上,提出一种消息复杂度较低,容错性能较高,且同步延迟小的对称分布式互斥算法,其请求集的生成算法简单,容易实现,理论上它可以处理任意规模分布式系统的对称请求集生成问题. 其互斥算法的一次临界区执行所需交换的消息数为 $2m-3m$.

2 系统模型

设分布式系统的节点数为 N,节点从 0 到 $N-1$ 编号. 假定系统的节点与通信均可靠. 各节点没有共享存储器和共同的物理时钟,节点依靠消息传递进行异步通信. 消息通信有限延迟但延迟时间无法预知.

3 对称请求集生成算法

3.1 对称请求集满足的条件

用 S^N 表示包含 N 节点的分布式系统, S_i 表示系统中 ID

收稿日期:2005-02-03;修回日期:2006-09-27

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

号为 i 的节点, R_i 表示节点 S_i 的请求集, k, n 等是常数.

Maekawa 在文献[2]中,提出了对称请求集必须满足的四个条件. 即

A1　$\forall i, j \in [0, N-1], R_i \cap R_j \neq \varnothing$. 即任意两个节点的请求集交集不为空.

A2　$\forall i, j \in [0, N-1], S_i \in R_i$. 即任意节点的请求集包含该节点本身.

A3　$\forall i, j \in [0, N-1], i \neq j, |R_i| = |R_j| = k$. 即每个节点的请求集长度相同,都包含 k 个节点.

A4　$\forall i \in [0, N-1], |\{R_j | S_i \in R_j, j \in [0q, N-1]\}| = k$ 即任一节点都属于 k 个请求集.

3.2　基本思想

用 $R\text{-}shift(R_i, k)$ 操作表示将集合 R_i 中的元素右移 k 位, $L\text{-}shift(R_i, k)$ 操作表示将集合 R_i 中的元素左移 k 位,则 $R\text{-}shift(R_i, k) \bmod(N)$ 操作表示以 N 为模将集合 R_i 中的元素循环右移 k 位, $L\text{-}shift(R_i, k) \bmod(N)$ 操作表示以 N 为模将集合 R_i 中的元素循环左移 k 位. 定义

$$R\text{-}shift(R_i, k) = R_{i+k} = \{(r_j + k) | j \in [1, n], k \in [1, N-1]\} \quad (1)$$

$$L\text{-}shift(R_i, k) = R_{i-k} = \{(r_j - k) | j \in [1, n], k \in [1, N-1]\} \quad (2)$$

则有

$$R\text{-}shift(R_i, k) \bmod(N) = \{(r_j + k) \bmod(N) | j \in [1, n], k \in [1, N-1]\} \quad (3)$$

$$L\text{-}shift(R_i, k) \bmod(N) = \{(r_j - k) \bmod(N) | j \in [1, n], k \in [1, N-1]\} \quad (4)$$

定义 1　循环请求集

$\forall S_i \in S^N$,令 $R_i = \{r_j | j \in [1, n]\}$ 表示 S_i 的请求集,如果有 $S_i \in R_i$,且 $\forall S_j \in S^N, j = (i + k) \bmod(N)$,都有 $R_j = R\text{-}shift(R_i, k) \bmod(N)$,且 $|R_i \cap R_j| \neq 0$,则称 R_j 和 R_i 分别是节点 S_i 和 S_j 的循环请求集. 其中 r_k 表示请求集 R_i 中第 k 个节点的 ID 号, n 表示请求集包含的节点数.

称 R_i 为系统 S^N 的一个循环请求基. S^N 的任意节点的请求集都能够由该请求基循环右移或左移生成.

定义 2　循环差集

设 $SUB = \{x_i | i \in [1, n]\}$ 是有限集合 D 的一个子集,且 $|D| = N$, $|SUB| = k$. 如果 $\forall x \in D$,都存在 λ 个数对 (x_i, x_j), $x_i, x_j \in SUB$,使得 $x = (x_i - x_j) \bmod(N)$,则称集合 SUB 是 D 的循环差集,并用 (N, k, λ) 表示.

定义 3　松弛循环差集

设 $SUB = \{x_i | i \in [1, n]\}$ 是有限集合 D 的一个子集,且 $|D| = N$, $|SUB| = k$. 如果 $\forall x \in D$,至少存在一个有序数对 (x_i, x_j), $x_i, x_j \in SUB$,使得 $x = (x_i - x_j) \bmod(N)$,则称 SUB 为 D 的松弛循环差集,简称松弛差集.

定理 1　循环请求集是对称请求集

证明:设 $R_i = \{r_j | j \in [1, n]\}$ 表示节点 S_i 的循环请求集. 因为 $\forall S_j \in S^N$,如果 $j = (i + k) \bmod(N)$,都有 $R_j = R\text{-}shift(R_i, k) \bmod(N)$,且 $|R_i \cap R_j| \neq 0$,因此循环请求集满足对称请求集的条件 A1;同时,因为 $|R_i| = n$,所以, $\forall S_j \in S^N, j \in [0, N-1]$, $|R_j| = n$,因此循环请求集满足对称请求集的条件 A3;遍取 S^N 中所有元素的请求集, R_i 的分量 $r_1, r_2, \cdots, r_n$ 中每个元素依次取 $[0, N-1]$ 中的元素各一次,请求集中 n 个元素共取 $[0, N-1]$ 中每个元素各 n 次. 即 $[0, N-1]$ 中每个元素属于 n 个不同的请求集,即循环请求集满足对称请求集的条件 A4. 又 $S_i \in R_i, j = (i + k) \bmod(N)$, $R_j = R\text{-}shift(R_i, k) \bmod(N)$,因此必有 $S_j \in R_j$,即任意节点的请求集包含节点本身,满足对称请求集的条件 A2,因此循环请求集是对称请求集. 证毕.

定理 2　循环请求集与松弛差集等价

证明:只须证明循环请求集是松弛差集,且松弛差集是循环请求集即可. 设 $R_i = \{r_i | i \in [1, n]\}$ 表示节点 S_i 的循环请求集,则 S^N 是一个有限集,且 $|S^N| = N$, $|R_i| = n$, $\forall S_i, S_j \in S^N$,如果 $j = (i + k) \bmod(N)$,有 $i = (j - k) \bmod(N)$, $j, k \in [0, N-1]$. 因为 $R_j = R\text{-}shift(R_i, k) \bmod(N)$, $|R_i \cap R_j| \neq 0$,因此 $|R_i \cap R_j| \geq 1$,且 $\exists r_k, r_k' \in R_i$,使得 $(r_k + k) \bmod(N) \in R$, $r_k' = (r_k + k) \bmod(N)$. 即 $k = (r_k' - r_k) \bmod(N)$. 又 $k \in [0, N-1]$, $r_k, r_k' \in R_i$ 表示节点的 ID 号,因此 $k \in [0, N-1]$ 也表示节点的 ID 号,即 S^N 中的任何节点都能由 R_i 中的元素的模 N 差表示. 因此循环请求集是松弛循环差集. 设 $D = S^N$, $R_{x_i} = SUB = \{x_j | j \in [1, n]\}$ 是有限集合 D 的一个循环差集,则 $|D| = |S^N| = N$, $|R_i| = |SUB| = n$. $\forall x \in D$,至少存在一个数对 (x_i, x_j), $x_i, x_j \in SUB$,使得 $x = (x_i - x_j) \bmod(N)$. 因此 $x_j = (x_i + x) \bmod(N)$,令 $R_{x_j} = \{(x_i + x) \bmod(N) | i \in [1, n], x \in [0, N-1]\}$,则 $R_{x_j} = R\text{-}shift(R_{x_i}, x) \bmod(N)$,且因为 $x_i, x_j \in SUB$, $x_j = (x_i + x) \bmod(N)$,因此 $(x_i + x) \bmod(N) \in R_{x_i} \cap R_{x_j}$,即 $|R_{x_i} \cap R_{x_j}| \neq 0$, $S_{x_i} \in R_{x_i}$. 因此松弛差集是循环请求集,循环请求集与松弛差集等价. 证毕.

3.3　请求集生成算法

(1) 数据结构

(a) 请求基向量 Q^N. 它是一个集合,其初始状态为 $Q^N = \{\varnothing\}$. 当请求集生成算法执行完毕后,它含有 n 个元素,表示系统 S^N 的一个请求基. $Q^N[i]$ 表示 Q^N 的第 i 个元素.

(b) 系统状态向量 A^N. 它含有 N 个分量,分量 $A^N[i]$ 标记系统 S^N 的对应节点 S_i 的状态. $A^N[i] = 1$ 表明 S_i 已被请求基 Q^N 中已有元素的模 N 差表示, $A^N[i] = 0$ 表明请求基 Q^N 中已有元素的模 N 差还不能表示节点 S_i.

(c) 重复记录字 T_j. $\forall j \in [0, N-1]$, $\forall i, j \in [1, n]$, $T_{i,j} = |\{k | A^N[(Q^N[i] + j - Q[k]) \bmod(N)] = 1\}|$,表示如果将元素 $(Q^N[i] + j) \bmod(N)$ 放入请求基 Q^N,新生成的模 N 差使 A^N 中元素被重复表示的次数.

(d) 向量状态字 S. 系统状态向量 A^N 的状态按其包含 1 的数量可以分为 $N + 1$ 种. 但由于松弛循环差集的对称性,即如果 $x = (x_i - x_j) \bmod(N)$,则存在 $x' = (x_j - x_i) \bmod(N)$, $x' \in [0, N-1]$,且 $x' = N - x$. 因此 A^N 的状态可以简化为 $[N/2] + 1$ 种. 其中 $[N/2]$ 表示大于 $N/2$ 的最小整数. 为简化算法的描述,将 A^N 的状态进一步简化为两种,并用状态集合 $\{s_0, s_1\}$

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

表示. 状态 s_0 表示 A^N 的分量不全为 1,状态 s_1 表示 A^N 的分量全为 1. 在状态字 s 中,状态 s_0, s_1 分别用数字 0 和 1 表示.

(2)算法描述

(a) **初始化**. $Q^N = \{\varnothing\}$,**任选** $S_i \in S^N$,**令** $A^N[i] = 1$, $Q^N = Q^N + \{i\}$, $\forall j \in [0, N-1], j \neq i$, $A^N[j] = 0$, $T = N$, $S = s_0$;

(b) **如果** $S = s_0$,**则** $\forall j, k \in [0, n-1]$,**令** $i = \min\{j \mid A^N[j] = 0\}$, $T_{k,i} = \min(T_{j,i})$, $Q^N = Q^N + \{i + k\}$, $\forall j, k \in [0, n-1]$, $A^N[(Q^N[j] - Q^N[k]) \bmod (N)] = 1$. **如果** $\forall i \in [0, N-1]$, $A^N[i] = 1$,**则令** $S = s_1$,**结束. 否则转** *(b)*.

(3)算法实例

设一个包含 20 个节点的分布式系统,节点 ID 从 0 到 19. 初始化时指定 Q^{20} 中包含节点 S_0. 图 1 *(a)* ～ *(d)* 描述了算法执行过程中各数据结构的变化.

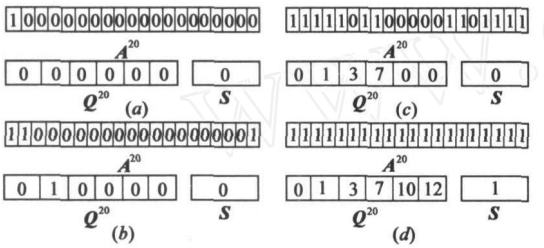

图 1 *(a)*初始化;*(b)*算法运行第一步;
*(c)*算法运行第三步;*(d)*算法运行第五步

初始化结束后,判断状态向量的状态字 $S = s_0$,因此继续执行算法,得到第一步的各数据结构状态. 直到第五步结束,判断状态向量的状态字 $S = s_1$,算法结束. 由 Q^{20} 知,20 个节点分布式系统的循环请求基包含 6 个节点,且 $R_0 = \{S_0, S_1, S_3, S_5, S_7, S_{10}, S_{12}\}$.

4 互斥算法

本算法的基本思想是:改变经典 Makawa 类算法的请求、应答、释放三段消息传送方式. 当请求到达某一节点时,如果其等待队列中有请求,则向其优先级高于本请求的前驱请求的请求节点发送转移(transfer)消息,如果其等待队列中没有请求,则向该请求的请求节点发送应答消息. 节点收到转移(transfer)消息,比较各转移消息携带请求的优先级,找出本节点执行临界区请求的下一个请求.

4.1 消息类型与数据结构

request:从节点 S_i 发送到节点 S_j 的 *request*$(i; j; req_i)$ 消息表示节点 S_j 请求节点 S_j 同意其进入临界区. req_i 表示 *request* 消息的时戳.

reply:从节点 S_i 发送到节点 S_j 的 *reply*$(i; j)$ 消息表示节点 S_j 同意节点 S_j 进入临界区.

release:从节点 S_i 发送到节点 S_j 的 *release*$(i; j; req_k)$ 消息表示节点 S_j 结束临界区执行,要求节点 S_j 删除其等待队列中优先级(时戳)不低于 req_k 的所有请求.

inquire:从节点 S_i 发送到节点 S_j 的 *inquire*$(i; j)$ 消息表示节点 S_j 想查询节点 S_j 是否正在执行临界区.

token:从节点 S_i 发送到节点 S_j 的 *token*$(i; j)$ 消息表示节点 S_j 已经执行完它的临界区,并将执行临界区的权力转给其下一个优先级的节点 S_j.

transfer:从节点 S_i 发送到节点 S_j 的 *transfer*$(i; j; req_k)$ 消息表示节点 S_j 收到一个优先级较低的请求并将该请求通知节点 S_j.

reconstruct:当节点 S_i 在请求进入临界区期间请求集中的节点出现故障,且所有节点的请求集都不可用时, S_i 向系统的所有活跃节点广播 *reconstruct* 消息,要求所有节点重构请求集.

为保证分布式互斥的实现,一个节点需要保存如下数据结构.

lock:是每个节点保存的一个数对 $(sn; j)$ (也表示成 req_j). j 是节点 S_i 的节点号, sn 消息的序号. *lock* 被初始化为 (max;max),max 是一个比任何消息序号都大的数.

failed:是一个布尔变量. 每次新发送一个请求时将其初始化为 0,当 S_i 收到一个 *inquire* 消息时,将 *failed* 置为 1.

fail_site:用于保存需转移令牌的最高优先权请求.

replied:是一个大小为 m 的布尔矢量(m 是请求集大小). 该矢量在每次发送新请求时将各分量初始化为 0,当 S_i 收到消息 *reply*$(j; i)$ 时将 *replied*$[j]$置为 1.

req_q:用于按优先级从高到低顺序存放等待请求.

inq_set:用于存放那些限于应答消息达到查询消息.

tokened:一个长度为 m 布尔矢量. 当收到 *token*$(j; i)$ 消息时,将 *tokened*$[j]$置 1.

trans_stack:用于存放 S_i 收到的 *transfer*$(i; j; req_k)$ 消息对应的次优请求,其优先级最高的次优请求位于栈顶. 当每次临界区执行完后将动态令牌传送给位于栈顶的次优请求并清空该栈.

4.1 分布式互斥算法

A 请求临界区

A1 当 S_i 请求进入临界区

S_i 发送 *request*(i, j, req_i) 给 $S_j \in R_i$;

清空 $trans\_stack_i$, inq_set_i, $failed\_site_i$;

置 $failed_i = 0$; $replied_i[\,] = 0$, $tokened_i = 0$, $token\_keep_i = 0$, $token\_sby_i = 0$, $lock_i = (\max, \max)$;

A2 当 S_i 收到 *request*(j, i, req_j):

```
if (lock_i = (max,max)) {
  lock_i = req_j;
  S_i 发送 reply(i, j) 给 S_j;
} else {
  case req_q_i = = ∅ ∧ req_j < lock_i :
       lock_i = req_j;
       S_i 发送 reply(i, j) 给 S_j;
  case req_q_i = = ∅ ∧ req_j > lock_i :
       S_i 发送 reply(i, j) 给 S_j;
  case req_q_i ≠ ∅ ∧ req_j > head(req_q_i) :
       将 request(j, i, req_j) 插入 req_q_i;
       令 S_k, S_k′ 表示 request(j, i, req_j) 在 req_q_i 中的前驱和后继;
       发送 transfer(i, k, req_j, req_k) 给 S_k;
       if (S_k ≠ ∅) {
          发送 transfer(i, j, req_k′, req_j) 给 S_j;
```

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

发送 $inquire(i,k)$ 给 S_k';}

case $req\_q_i \neq \varnothing \wedge req_j > head(req\_q_i) < lock_i$:

$lock_i = req_j$;

令 S_k 代表 req_q_i 中的第一个请求;

发送 $transfer(i,j,req_k,req_j)$ 给 S_j;

发送 $inquire(i,k)$ 给 S_k;

}

A3 当 S_i 收到 $inquire(j,i)$:

if{

$(((replied_i == 1) \vee (tokened_i == 1)) \wedge failed_i == 1)$){

令 $request_m$ 代表 $inquire$ 与 $fail\_site_i$ 中较高优先级的请求;

令 $fail\_site_i = request_m$;

$failed_i = 1$;

}else {

令 $request_k$ 代表 $inquire$ 中较高优先级的请求;

if($request_k$ 是等待队列中的首请求){

发送 $reply(i,k)$ 给 S_k;}

令 $request_m$ 代表 $inquire$ 与 $fail\_site_i$ 中较高优先级的请求;

令 $fail\_site_i = request_m$;

$failed_i = 1$;

}

A4 当 S_i 收到:$transfer(j,i,req_k,req_m)$:

if $req_m \in req\_q_i$){

令 $request_k$ 代表 req_q_i 与 $tran\_stack_i$ 中较高优先级的请求;;

$tran\_stack_i = request_k$;

}else{

if ($token\_keep_i == 0$){

丢弃 $request_k$;

}else{

发送 $token(i,k)$; $token\_keep_i = 0$;

发送 $release$ 给备份节点;}

}

A5 当 S_i 收到 $reply(j,i)$:

$replied_i[j] = 1$;

if ($replied_i[j] == 1$, $\forall j \in R_i$){

$tokened_i = 1$;}

A6 当 S_i 收到 $token(j,i)$:

令 $request_k$ 代表 $token(j,i)$ 的相应节点;

if ($request_k \in req\_q_i$){

$tokened_i = 1$;

执行临界区;

}else{

退出临界区;

}

}

B 执行临界区

当所有 $tokened_i == 1$ 或收到令牌时 S_i 能执行临界区

C 释放临界区

C1 当 S_i 退出临界区:

$replied_i[\,] = 0$;

if ($fail\_site_i \neq \varnothing$){

令 $request_k = fail\_site_i$;

发送 $token(i,k)$ 给 S_k;

$failed_i = 0$;

清空 $tran\_stack_i$;

}else{

if ($tran\_stack_i == \varnothing$){

($token\_keep_i = 1$);

发送 $release(i,k,req_k)$ 给 $S_i \in R_i$;

}else{

$request_k = tran\_stack_i$;

发送 $token(i,k)$ 给 S_k;

$failed_i = 0$;

清空 $tran\_stack_i$;}

}

C2 当 S_i 收到 $release(i,k,req_k)$:

if ($release(i,k,req_k).type == 0$){

清除所有优先级高于 req_k 的请求;

if(S_i 是 S_j 的备份节点){

$token\_sby_i = 1$;}

}else{

if ($token\_sby_i == 1$){

$token\_sby_i = 0$}

}

D 节点容错操作

if(所有请求给都不可用){

发送 $reconstruct$ 给 $S_i \in S$;

重构系统请求集;

}else{

搜索一个可用请求集 R_k;

发送 $request(i,j,req_i)$ 给 $S_j \in R_k$;}

if(S_i 已知 S_j 不可用且 S_i 是 S_j 的备份节点且 $token\_sby_i == 0$){

$token\_sby_i = 1$;}

5 性能分析

5.1 消息复杂度

设 $R \subseteq R_i$, R 是 R_i 的子集,若 $\forall i,j \in R$, $i < j$, $T = |\{(i,j) | (j-i) \bmod (N) = l, l \in [0, N-1]\}| = 1$,即 R 是 R_i 中,模 N 差不同的元素的集合.令 $k = |R|$,则 $\forall i,j \in R$, $i < j$, $|\{l | (j-i) \bmod (N) = l, l \in [0, N-1]\}| = k(k-1)+1$.显然 $k(k-1)+1 \leq N$.设 S^N 中有 $k(k-1)+1$ 个元素被 R 中元素的模 N 差表示,则 S^N 中有 $N-k(k-1)+1$ 个元素未被 R 中元素的模 N 差表示.由差集的对称性,在请求集 R_i 中增加一个元素至少能使 S^N 中的两个元素被表示,因此 $N-k(k-1)+1$ 个未被 R 中元素的模 N 差表示的元素将至多能被 $m = k+(N-k(k-1)+1)/2$ 个元素表示.又因当 $k(k-1)+1 = N$ 时, N 个元素能被 k 个元素的模 N 差表示,因此当 $k(k-1)+1 < N < (k+1)k+1$ 时, k 个元素的差能表示 $k(k-1)+1$ 个元素,且 $k \approx \sqrt{N}$, $N-k(k-1)+1 \leq 2k$,因此 $m \leq 2k \approx 2\sqrt{N}$.因此,本文请求集生成算法生成的请求集上限 m 为 $2\sqrt{N}$.

在轻负载时,任意一次临界区请求到达请求集各节点时,其等待队列为空,因此一次临界区执行需要 $m-1$ 次请求, $m-1$ 次应答及 $m-1$ 次释放消息.因此轻负载时一次临界区执

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

行所需交换的消息数为 $3(m-1)$. 重负载时,如果节点收到的请求都没有乱序,则一次临界区执行需要 $m-1$ 次请求, $m-1$ 转移,一次令牌发送. 此时一次临界区执行所需交换的消息数为 $2(m-1)+1=2m+1$. 而在节点收到的请求优先级在等待队列中既不是最高也不是最低时,需要最多的消息数. 这时需要 $m-1$ 次请求, $m-1$ 转移, $m-1$ 查询,一次令牌发送. 此时一次临界区执行所需交换的消息数为 $3(m-1)+1=3m-2$. 因此该算法的消息复杂度为 $O(3m)$. 较一般 Makawa 类算法在重负载时的消息复杂度降低了近一倍. 由于 $m\approx 2\sqrt{N}$, 因此,本算法消息复杂度可写成 $O(6\sqrt{N})$.

5.2 同步延迟

本文互斥算法由于采用了转移应答和令牌消息,因此将系统中的请求组成了一个类似于 RA 算法的动态逻辑令牌环,使系统中的请求能够按 Lamport 逻辑时戳决定的优先级顺序依次完成. 同时,转移应答消息改变了经典 Maekawa 算法要求一个请求释放以后才能对次优请求发送应答的约束,次优请求发起节点收到转移释放消息后能够将其当作请求控制节点(两个请求集的交集)发送的应答消息. 因此,次优请求发起节点进入临界区只须等待一次消息(转移应答消息)传递的时间,即同步延迟为 T. 而经典 Maekawa 算法的次优请求发起节点进入临界区必须等待两次消息(释放,应答)传送时间,因此其同步延迟为 $2T$.

5.3 节点容错性能

本算法采取了动态令牌的方式转移节点进入临界区的权力,因此并不要求请求节中的所有节点都能正常工作. 如果是因为请求集中节点故障造成节点不能访问临界区,本算法将采用重新寻求请求集并重新发送请求的方式实现节点失效的容错. 在系统生成的请求集都不可用时,本算法采用了重构请求集得以保证算法继续运行. 因此,如果不对临界区访问延时进行限制,本算法在除请求节点外的任何数量节点失效情况下都能保证互斥访问的顺利完成. 因此,本算法的节点容错能力为 $N-1$.

6 死锁证明

定理 算法无死锁.

证明. 假定算法可能出现死锁,即在一系列请求节点中,因为他们都在等待一或者多个应答,从而造成循环等待使得没有一个节点能够进入临界区. 在该等待循环中,必然存在一个节点 S_i,它的请求优先级最高. 在算法开始时,假定 S_i 等待 S_j 的应答. 而 S_j 给了 S_k 应答,同时 S_k 正在等待 S_i 的应答. 根据算法 A2, S_j 发送 inquire 消息给 S_k, S_k 发送 fail 消息给 S_j,因此 S_j 发送应答消息给 S_i, S_i 得到进入临界区的权利. 因此不存在死锁. 在算法运行过程中,也假定 S_i 等待 S_j 的应答. 而 S_j 给了 S_k 应答,同时 S_k 正在等待 S_i 的应答. 在这种情况下,假定令牌被节点 S_m 持有. 而 S_m 不是 S_i, S_j 和 S_k 中的一个. 如果. S_k 的请求比 S_i 的请求早到达 S_m,而当 S_m 退出临界区时 S_j 的请求还未到达 S_m,则 S_m 将令牌发送给 S_k,而当 S_k 退出临界区时,它将令牌按算法 C1 发送给相应节点. 因此不存在死锁. 综合上述两种情况,本文算法不存在死锁.

7 结语

利用松弛循环差集与互斥循环请求集的等价性,能够生成较短的分布式系统对称请求集,并且其实现简单,高效. 利用转移消息及动态令牌,虽然增加了消息种类,但明显降低了 Makawa 类互斥算法的同步延迟和消息复杂度,提高了算法的时间效率. 同时,在节点失效时重发请求及重构请求集,提高了算法及系统的容错性能,使算法的节点容错能力达到 $N-1$.

参考文献:

[1] L Lamport. Time, clocks and ordering of events in distributed systems[J]. Comm ACM, 1978, 21(7): 558 - 565.

[2] M Maekawa. A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems [J]. ACM Trans Computer Systems, 1985, 3(2): 145 - 159.

[3] J Helary. A general scheme for token- and tree-based distributed mutual exclusion algorithms [J]. IEEE Trans. Parallel and Distributed Systems, 1994, 5(11): 1185 - 1196.

[4] M Naimi. An improvement of the Log(n) distributed algorithm for mutual exclusion[A]. Proc. Seventh Int'l Conf. Distributed Computing System[C]. IEEE, 1987. 371 - 375.

[5] K Raymond. Tree-based algorithm for distributed mutual exclusion[J]. ACM Trans Computing Systems, 1989, (2): 61 - 77.

[6] A Kumar. Hierarchical quorum consensus: a new algorithm for managing replicated data [J]. IEEE Trans. Computers, 1991 (9): 996 - 1004.

[7] Y C Kuo, S T Huang. A geometric approach for constructing coteries and k-coteries[J]. IEEE Trans. Parallel and Distributed Systems, 1997, 8(4): 402 - 411.

[8] K Ogata, K Futatsugi. Formally modeling and verifying Ricart & Agrawala distributed mutual exclusion algorithm [A]. Quality Software, 2001. Proceedings. Second Asia-Pacific Conference on[C]. IEEE, 2001. 357 - 366.

[9] Jiannong Cao, Jingyang Zhou. Wu J, An efficient distributed mutual exclusion algorithm based on relative consensus voting [A]. Parallel and Distributed Processing Symposium [C]. IEEE, 2004: 51.

[10] R Baldoni, A Virgillito. A distributed mutual exclusion algorithm for mobile ad-hoc networks [A]. Computers and Communications, Proceedings. ISCC 2002. Seventh International Symposium on[C]. IEEE, 2002. 539 - 544.

[11] T Harada, M Yamashita. Transversal merge operation: a non-dominated coterie construction method for distributed mutual exclusion[J]. Parallel and Distributed Systems, IEEE Transactions on, 2005, 2(2): 183 - 192.

[12] S Rangarajan. A fault-tolerant algorithm for replicated data management [J]. IEEE Trans Parallel and Distributed Systems, 1995, 6(12): 1271 - 1282.

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

[13] M Bearden. A fault-tolerant algorithm for decentralized on-line quorum adaptation [A]. Fault-Tolerant Computing, 1998. Twenty-Eighth Annual International Symposium on [C]. IEEE, 1998: 262 - 271.

[14] C Guohong. A delay-optimal quorum-based mutual exclusion algorithm for distributed systems [J]. IEEE Trans. Parallel and Distributed Systems, 2001, 12(12): 1256 - 1268.

[15] C Guohong. A delay-optimal quorum-based mutual exclusion scheme with fault-tolerance capability [A]. Distributed Computing Systems, Proceedings. 18th International Conference on [C]. IEEE, 1998. 444 - 451.

[16] Wai-Shing Luk. Two new quorum based algorithms for distributed mutual exclusion [A]. Distributed Computing Systems, 1997, Proceedings of the 17th International Conference on [C]. IEEE, 1997: 100 - 106.

[17] W K Ng, C V Ravishankar. Coterie templates-a new quorum construction method [A]. Distributed Computing Systems, 1995, Proceedings of the 15th International Conference [C]. IEEE, 1995. 92 - 99.

作者简介:

李美安　男,1973 年生于四川大竹,电子科技大学计算机科学与工程学院博士生,主要研究方向为分布式操作系统、分布式计算、宽带网络与通信等. E-mail: limeian@sohu.com

刘心松　男,1940 年生,教授,博士生导师,研究方向为分布式计算、宽带网络与通信等. E-mail: Liu_xs@sohu.com

著名微电子专家童勤义教授去世

著名微电子专家童勤义教授于 2006 年 9 月 21 日在美国去世,享年 68 岁. 童勤义教授 1938 年 12 月 16 日出生于浙江慈溪,1962 年毕业于清华大学无线电系,后分配到东南大学工作. 1980 年赴英国爱丁堡大学进修. 历任东南大学讲师、副教授、教授,1985 年担任博士生导师,曾任国务院学位委员会第二届学科评议组成员. 1991 年到美国 DUKE 大学进行科学研究. 曾先后出版《VLSI 物理学导论》(电子工业出版社,1988)、《微电子系统导论》(东南大学出版社,1990)、《Semiconductor Wafer Bonding》(John Wiley & Sons,1999)等著作. 1988 - 1995 担任国际期刊(Sensors and Actuators)编委及国际会议 TRANSDUCER 领导委员会委员等国际会议职务. 为国内微电子、微传感器事业发展培养了大批高级技术人才. 我们深切怀念他.

© 1994-2010 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net