Journal of Computer Applications
计算机应用,2013,33(2):323 -325,360
ISSN 1001-9081
CODEN JYIIDU
2013-02-01
http://www.joca.cn

文章编号:1001 -9081(2013)02 -0323 -03
doi:10.3724/SP.J.1087.2013.00323

具有 $O(n)$ 时间复杂度的分布式请求集生成算法

武　鹏[1*],李美安[2]
(1.山西工程职业技术学院 计算机工程系,太原 030009;　2.内蒙古农业大学 计算机与信息工程学院,呼和浩特 010018)
(*通信作者电子邮箱 and202058@163.com)

摘　要:在大规模完全分布式系统的互斥问题上,快速生成请求集是必要的。在基于松弛差集的相关原理上,引入了二次松弛差集的概念。经分析相关概念及定理,将原本"求差"的过程变为"求和"的过程;进而利用"求和"步骤间的递推关系,大大减少了求和步骤,使整个算法的时间复杂度控制在 $O(n)$。与时间复杂度同为 $O(n^2)$ 的其他经典算法相比,生成的请求集长度仍保持在 $2\sqrt{n}$ 的数量级。

关键词:分布式互斥;请求集;松弛差集;时间复杂度
中图分类号:TP301.6;TP316.4　　**文献标志码**:A

Quorum generation algorithm with time complexity of $O(n)$

WU Peng[1*], LI Meian[2]
(1. *Department of Computer Engineering, Shanxi Engineering Vocational College, Taiyuan Shanxi* 030009, *China*;
2. *College of Computer and Information Engineering, Inner Mongolia Agriculture University, Hohhot Inner Mongolia* 010018, *China*)

Abstract: It is necessary to generate the quorums as soon as possible in large-scale fully distributed system for its mutual exclusion problem. Based on the theory of relaxed cyclic difference set, the definition of second relaxed cyclic difference set was proposed. After researching the new concepts, the subtraction steps in previously classical methods can be changed into summation steps. Furthermore, a lot of summation steps can be cut down by the recurrence relation deduced from the summation steps. The time complexity of this algorithm is just only $O(n)$ and the size of the symmetric quorums is still close to $2\sqrt{n}$.

Key words: distributed mutual exclusion; quorum; relaxed cyclic difference; time complexity

0　引言

在全分布式系统中,各节点完全对等,从而消除了主从式分布式系统单点失效的缺点。但系统中的任意节点要进入临界区,必须要向系统的其他 $n-1$ 个节点通信[1]。Maekawa[2]发现,某节点要进入临界区时,不必向其他 $n-1$ 个节点请求,而只需要向其请求集发出请求即可,并指出对称请求集的下限为 $\sqrt{n}$。但对任意节点的系统,快速生成大小接近 $\sqrt{n}$ 的对称请求集是很困难的事。Sanders[3]解决了 Maekawa 算法的死锁问题,增加了 Maekawa 互斥算法的实用性。Luk[4]开创了松弛差集理论,并结合该理论提出了可生成最优请求集(请求集节点接近 $\sqrt{n}$)的算法。虽然该算法生成的请求集符合文献[2]中关于对称请求集的定义,但该算法的时间复杂度是指数形式。至今为止,尚未发现时间复杂度为多项式的最优请求集生成算法。至此,人们更多的研究能生成接近最优请求集的算法。李美安等[5]提出的算法生成的请求集长度在 $2\sqrt{n}$ 数量级,时间复杂度为 $O(n^2)$。文献[6-9]中的算法生成的请求集长度控制在 $\sqrt{n}\sim 2\sqrt{n}$,但时间复杂度仍在 $O(n^2)$ 以上。经过实验测试,当系统节点规模超过 10 万时,文献[6-9]的算法单机运行时间均在 10 min 以上,这在大规模完全分布式系统中是不可接受的。Ng 等[10]提出的算法将请求集长度降到了 $n^{0.63}$,但当 n 较大时,生成的请求集长度远大于 $2\sqrt{n}$。这样,当全分布式系统的节点数大规模增加时,节点进入临界区所花费的消息量就会显著增加。文献[11-12]都对 Maekawa 算法进行了改进,但未给出适合大规模全分布式系统的对称请求集生成算法。目前,在大规模全分布式系统中,人们对能快速生成对称请求集的算法少有研究。如果该类算法的时间复杂度降到 $O(n)$,且生成的请求集长度没有显著增加,那么,就能在不明显增加节点进入临界区所需的消息量的条件下,快速生成所需的请求集。这样的算法显然更快、更优。

1　分布式系统模型

设系统的节点数为 n,并从 0 到 $n-1$ 对节点编号,第 i 个节点的 ID 号为 $i-1$,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,节点间完全依靠消息进行通信,并且消息通信时间延迟无法预知。

2　对称请求集的相关概念

2.1　对称请求集满足的条件

Maekawa[2]提出了对称请求集满足的四个条件:

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:$\forall i\in[0,n-1]$,$S_i\in R_i$,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:$\forall i\in[0,n-1]$,$|\{R_j|S_i\in R_j$,且$j\in[0,n-1]\}|=k$,

收稿日期:2012-08-29;修回日期:2012-10-22。

作者简介:武鹏(1981 -),男,山西太原人,助教,硕士,主要研究方向:分布式系统;　李美安(1973 -),男,四川大竹人,教授,博士,主要研究方向:分布式操作系统、分布式算法。

即任一节点都属于 k 个请求集。

其中:用 S_n 表示包含 n 个节点的分布式系统;S_i 表示系统中 ID 号为 i 的节点;R_i 表示节点 S_i 的请求集;k,n 等为常数。

满足上述条件 A1 ~ A4 的请求集称为对称请求集,能够生成对称请求集的算法称为对称请求集生成算法;且满足以上四个条件的请求集中的元素个数有下界,下界约为$\sqrt{n}$。

2.2 松弛差集概念

Luk[4] 提出了以下松弛差集的相关概念和定理。

定义 1 循环请求集。$\forall S_i \in S_n$,令 R_i 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-shift(R_i,k) mod (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-shift(R_i,k) mod (n) 表示将 R_i 中的元素依次取模(循环)左移 k 位。

定义 2 松弛循环差集。设 SUB 为有限集合 $D = \{x_i \mid x_i \in [0,n-1]\}$ 的一个子集,且$|D| = n$,$|SUB| = m$。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 $x = |x_i - x_j|$,或者 $x = n - |x_i - x_j|$,则称 SUB 为 D 的松弛循环差集,以下简称松弛差集。

定理 1 循环请求集与松弛差集等价。

定理 2 循环请求集为对称请求集。

2.3 SUB 为松弛差集的充分必要条件

笔者的前期研究[7] 中给出了集合 SUB 为松弛差集的充分必要条件,如定理 3。

定理 3 SUB 为松弛差集的充分必要条件是:$D = \{x \mid x = |x_i - x_j|$ 或 $x = n - |x_i - x_j|, x_i, x_j \in SUB\}$。其中:$SUB \subset D$,$D = \{x_i \mid x_i \in [0,n-1]\}$。

3 算法分析与描述

3.1 相关概念与分析

定义 3 二次松弛差集,以下简称 S_SUB,$S\_SUB = \{x \mid x = x_{i+1} - x_i, x_{i+1}, x_i \in SUB\}$。

Luk 提出的 SUB 集中元素是系统节点的标号,标号大的节点比标号小的节点数值大。所以 S_SUB 集中元素为依次递增的自然数列,即 S_SUB 为 SUB 集中元素后项减前项的差所构成的集合。结合 S_SUB 的概念,可以给出 SUB 为松弛差集的充要条件的另一种表述,如定理 4。

定理 4 SUB 为松弛差集的充要条件为:$D = \{x \mid x = sum(x_i,x_j)$ 或者 $x = n - sum(x_i, x_j)$, $x_i \leqslant x_j$ 且 $x_i, x_j \in S\_SUB\}$。其中:$D = \{x_i \mid x_i \in [0,n-1]\}$;$sum(x_i, x_j)$ 为在 S_SUB 集合中,从 x_i 到 x_j 的所有元素之和,i 和 j 可以相等,如 i 和 j 相等,$sum(x_i, x_i) = x_i$。

证明 由定义 3 可得,S_SUB 为 SUB 集中元素后项减前项的差所构成的集合,定理 3 的任意$|x_i - x_j|$ 为对应的 S_SUB 集合相应 $sum(x_i', x_j')$ 之和,即定理 3 与本定理同真假。定理 3 在文献[7] 已证,所以本定理亦为真。

3.2 算法分析

在定理 4 中,给出了不同于文献[7] 的充要条件。所以只需构造一个 S_SUB,使其满足该充要条件,则由 S_SUB 所确定的 SUB 集为松弛差集。而 S_SUB 集中的元素是由 SUB 集中元素依次后项减前项构成的集合,所以由 S_SUB 集导出对应的 SUB 集是十分方便的。

3.3 算法思想

从小到大扫描 D 集合的元素,取出一个未被表示的元素 d_i,将 d_i 加入到 S_SUB 集中(假设此时 $S\_SUB = \{x_1,x_2,\cdots,x_m\}$),并将该元素作为当前 S_SUB 集的最后一个元素。i 从 1 到 m,计算所有的 $sum(x_i, d_i)$。将由 d_i 加入 D 后引起的 D 中表示了的元素置为被表示状态,直至 D 中所有元素都被表示。

3.4 算法数据结构

1) SUB 队列用数组实现。

2) S_SUB 队列用数组实现,该队列中元素的值满足 SUB 对应数组元素后项减前项的差。如:$S\_SUB[0] = SUB[1] - SUB[0]$。

3) D 数组,长度为 n,$D[i] = 1$ 表示第 $i+1$ 号节点已被 SUB 表示(即满足 2.3 节中的充分必要条件);$D[i] = 0$ 则表示第 $i+1$ 号节点未被 SUB 表示。

3.5 算法描述

用 C 语言伪代码描述如下:

```
Get_SUB (n, &q, *SUB)
        /* n 为系统节点总数,q 的值返回 SUB 队列的长度。*/
1)   { 为 S_SUB 队列分配足够的空间
2)     p = sum = 0,SUB[0] = D[0] = q = 1;
3)     D[1,2,…,n-1] = 0;
4)     for (x = 1;x < n;x ++)
5)     { if (D[x] == 1) continue;
         else
6)       { S_SUB[p] = x;
7)         p ++;
8)         D[x] = 1;
9)         D[n-x] = 1;
10)        sum = S_SUB[p-1];
11)        for (i = p-2;i >= 0;i ++)
12)        { sum = S_SUB[i] + sum;
13)          if ((sum%n) == 0) continue;
       /*① 防止 D[n-(sum%n)] 越界。② 当 sum%n = 0,*/
          /*D[0] 不在 D[1] - D[n-1] 范围内,不考虑。*/
            else
14)           { D[sum%n] = 1;
15)             D[n-(sum%n)] = 1;
              }
          }
        }
      }
16)   for (i = 1;i <= p;i ++)
17)   { if ((SUB[i-1] + S_SUB[i-1])%n == 0);
18)     SUB[i] = n;
        else
19)     SUB[i] = (SUB[i-1] + S_SUB[i-1])%n;
20)   q ++;
      }
    }
```

4 算法实例与时间复杂度分析

4.1 算法实例介绍

以 $n = 13$ 时举例:

1) 初始化:$S\_SUB = \{\}$,$SUB = \{1\}$。此时 D 数组如表 1 所示。

表 1 初始化后 D 数组的状态

元素	值	元素	值	元素	值	元素	值
$D[1]$	0	$D[4]$	0	$D[7]$	0	$D[10]$	0
$D[2]$	0	$D[5]$	0	$D[8]$	0	$D[11]$	0
$D[3]$	0	$D[6]$	0	$D[9]$	0	$D[12]$	0

2)*S_SUB* 队列加入元素 1 后:*S_SUB* = {1},*SUB* = {1}。此时 *D* 数组如表 2 所示。

表 2　*S_SUB* 队列加入 1 后 *D* 数组的状态

元素	值	元素	值	元素	值	元素	值
D[1]	1	*D*[4]	0	*D*[7]	0	*D*[10]	0
D[2]	0	*D*[5]	0	*D*[8]	0	*D*[11]	0
D[3]	0	*D*[6]	0	*D*[9]	0	*D*[12]	1

3)*S_SUB* 队列加入元素 2 后:*S_SUB* = {1,2},*SUB* = {1}。此时 *D* 数组如表 3 所示。

表 3　*S_SUB* 队列加入 2 后 *D* 数组的状态

元素	值	元素	值	元素	值	元素	值
D[1]	1	*D*[4]	0	*D*[7]	0	*D*[10]	1
D[2]	1	*D*[5]	0	*D*[8]	0	*D*[11]	1
D[3]	1	*D*[6]	0	*D*[9]	0	*D*[12]	1

4)*S_SUB* 队列加入元素 4 后:*S_SUB* = {1,2,4},*SUB* = {1}。此时 *D* 数组如表 4 所示。

表 4　*S_SUB* 队列加入 4 后 *D* 数组的状态

元素	值	元素	值	元素	值	元素	值
D[1]	1	*D*[4]	1	*D*[7]	1	*D*[10]	1
D[2]	1	*D*[5]	0	*D*[8]	0	*D*[11]	1
D[3]	1	*D*[6]	1	*D*[9]	1	*D*[12]	1

5)*S_SUB* 队列加入元素 5 后:*S_SUB* = {1,2,4,5},*SUB* = {1}。此时 *D* 数组如表 5 所示。

表 5　*S_SUB* 队列加入 5 后 *D* 数组的状态

元素	值	元素	值	元素	值	元素	值
D[1]	1	*D*[4]	1	*D*[7]	1	*D*[10]	1
D[2]	1	*D*[5]	1	*D*[8]	1	*D*[11]	1
D[3]	1	*D*[6]	1	*D*[9]	1	*D*[12]	1

6) 由 *S_SUB* 队列导出 *SUB* 队列:*S_SUB* = {1,2,4,5},*SUB* = {1,2,4,8,13}。

7) 各节点得到自己的对称请求集。根据文献[4] 中证明的有关循环请求集与松弛差集等价的相关定理,系统中的各节点可以根据 *SUB* 来导出各节点的请求集,可得如下公式:

假设系统各节点标号依次为 1,2,…,*n*,*SUB* = {S_1, S_2, …,S_m},则标号为 *p* 的节点的请求集为:

{1 + (S_1 + *P* − 2) mod (*n*),1 + (S_2 + *P* − 2) mod (*n*), …,1 + (S_m + *P* − 2) mod (*n*)}

该例中,系统节点标号从 1 到 13,已求得的 *SUB* 为{1,2,4,8,13},则标号为 $p \in [1,13]$ 的节点的请求集为:

{1 + (1 + *P* − 2) mod (13),1 + (2 + *P* − 2) mod (13), 1 +(4 +*P* −2) mod (13),1 + (8 +*P* −2) mod (13),1 + (13 + *P* − 2) mod (13)}

得到各节点的请求集如下:

节点 1 的请求集:{1,2,4,8,13};
节点 2 的请求集:{2,3,5,9,1};
节点 3 的请求集:{3,4,6,10,2};
节点 4 的请求集:{4,5,7,11,3};
节点 5 的请求集:{5,6,8,12,4};
节点 6 的请求集:{6,7,9,13,5};
节点 7 的请求集:{7,8,10,1,6};
节点 8 的请求集:{8,9,11,2,7};
节点 9 的请求集:{9,10,12,3,8};
节点 10 的请求集:{10,11,13,4,9};
节点 11 的请求集:{11,12,1,5,10};
节点 12 的请求集:{12,13,2,6,11};
节点 13 的请求集:{13,1,3,7,12}。

经验证,上述给出的各节点的请求集完全符合 Maekawa[2] 提出的对称请求集的四个条件。

4.2　算法时间复杂度分析

因为 *D* 数组长度为 *n* − 1,3.5 节伪代码的第 5) ~ 10)行的赋值语句执行次数最大不超过 4(*n* − 1)。在计算“新加入 *S_SUB* 队列的元素 *x*” 与“*S_SUB* 队列中其他元素” 的 *sum* 和时(*sum* 定义见定理 4),假设此时 *S_SUB* 队列 = {x_1,x_2,…, x_m,*x*},即计算 *sum*(x_1, *x*),*sum*(x_2, *x*),…,*sum*(x_m, *x*)。因为各 *sum* 和有如下递推关系:x_1 + *sum*(x_2, *x*) = *sum*(x_1, *x*), x_2 +*sum*(x_3, *x*) = *sum*(x_2, *x*),…, x_{m-1} + *sum*(x_m, *x*) = *sum*(x_{m-1}, *x*),x_m + *x* = *sum*(x_m, *x*),所以 *i* 从 1 到 *m*,计算所有的 *sum*(x_i, *x*) 的过程中,可按上述递推公式从后往前计算。这样,总的计算次数为 *m*,而本算法最终得到的 *S_SUB* 的长度大约为 $2\sqrt{n}$ 的数量级,所以算法的执行过程中,*m* 从 1 到 $O(2\sqrt{n})$。所以 3.5 节伪代码中计算 *sum* 和及相关对 *D* 数组赋值的语句(第 11) ~ 15) 行) 的执行次数大约为:$3(1 + 2 + \cdots + 2\sqrt{n}) = [3(2\sqrt{n}+1)2\sqrt{n}]/2 = 6n - 3\sqrt{n} = O(n)$。因为 $|S\_SUB| \approx 2\sqrt{n}$,所以执行 3.5 节伪代码的第 16) ~ 20) 行中的赋值语句的次数大约为 $2\sqrt{n}$。综合以上分析可得该算法时间复杂度为 $O(n)$。

4.3　与经典算法及经典常数的比较

本文算法与经典算法及经典常数的比较如表 6 所示。本算法生成的请求集长度为 $2\sqrt{n}$ 数量级,请求集长度和时间复杂度都优于文献[5]。对比文献[4],本文算法的请求集长度虽不及文献[4],但时间复杂度仍优于文献[4],更适合大规模全分布式系统的请求集生成问题。

表 6　本文算法生成的请求集长度与其他算法及常数的比较

n	$2\sqrt{n}$	请求集长度		
		文献[6] 算法	文献[5] 算法	本文算法
13	8	5	5	5
31	12	7	7	8
57	15	10	9	11
91	20	13	14	14
133	24	16	19	19
183	28	18	22	21
241	32	22	26	25
307	36	25	31	30
381	40	27	35	34
463	44	30	40	36
5 000	144	110	175	151
10 000	202	162	257	230
时间复杂度		$O(n^2)$	$O(n^2)$	$O(n)$

5　结语

通过引入二次松弛差集的概念,利用“求和” 步骤间的递推关系,减少了计算步骤。最终算法的时间复杂度为 $O(n)$,与经典算法相比,生成的对称请求集的长度仍为 $2\sqrt{n}$ 的数量级。尤其是与文献[5] 的算法对比,在生成的请求集长度与其大致相当的情况下,时间复杂度由 $O(n^2)$ 降到了 $O(n)$。当系统节点数很大时,本文算法能很快地生成对称请求集,对解决大规模全分布式系统的互斥问题有很好的帮助。

(下转第 360 页)

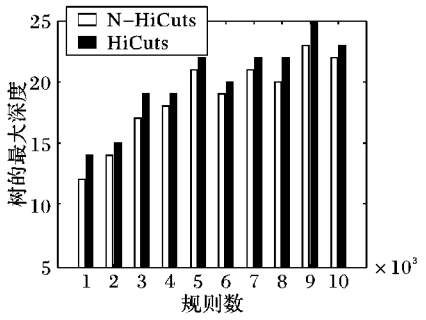

图 4 决策树最大深度

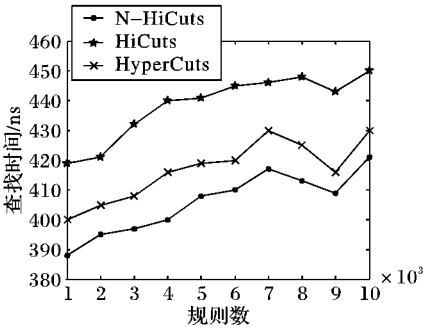

图 5 决策树查找时间曲线

图 6 是使用 N-HiCuts 算法、HiCuts 算法以及 HyperCuts 算法内存空间占用的比较。从图 6 中可以看出,随着规则数的增加,三种算法的存储空间占用都大大增加;但是由于 N-HiCuts算法采用了非均匀分割将规则集均匀地划分到子集中去,使决策树平衡性得到改进从而叶子节点深度落差减小。因此N-HiCuts算法的空间占用远远小于 HiCuts 算法,比经过改进的决策树层次减小的 HyperCuts 算法略小。

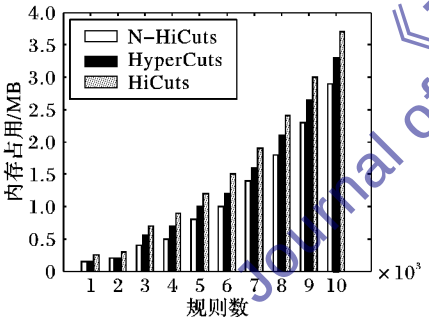

图 6 决策树存储空间占用

5 结语

数据包分类技术是整个网络运行体系中的重要支撑功能,近年来更是在许多网络服务当中得到了广泛的应用。本文通过对规则集特征分析及对 HiCuts 算法的研究,发现规则集分布并不均匀,规则集中的某些域只有有限的几种取值,据此提出 N-HiCuts 包分类算法。该算法对规则集中的某些规则分布不均匀域采用文中所提到的非均匀分割的方式对域进行分割,对规则集分布较均匀的域采用文中的等分函数对该域进行等分切割。非均匀切割使划分后各节点规则集数目趋于平均,从而增强了决策树的平衡性,降低了决策树树的深度。与多维包分类算法中具有代表性 HiCuts 算法相比,N-HiCuts算法在查找的时间效率和空间占用上都有很大提高。

参考文献:

[1] 单福勇. 基于 HiCuts 算法的 Linux IPv6 防火墙研究[D]. 大连: 大连海事大学, 2009.

[2] 翟钰, 武舒凡, 胡建武. 防火墙包过滤技术发展研究[J]. 计算机应用研究, 2004, 21(9): 144 - 146.

[3] GUPTA P, McKEOWN N. Algorithms for packet classification [J]. IEEE Network, 2000, 15(2): 24 - 32.

[4] 王萌, 王玲. 包分类算法在防火墙中的应用研究[J]. 通信技术, 2011, 44(5): 57 - 59.

[5] 赵国锋, 陈群丽. 基于 Hash 和 AQT 的类决策树包分类算法研究[J]. 通信技术, 2010, 43(2): 210 - 215.

[6] 谭俊璐. 基于决策树规则分类算法的研究与应用[D]. 广州: 暨南大学, 2010.

[7] 李振强, 张圣亮, 马严, 等. 多决策树包分类算法[J]. 电子与信息学报, 2008, 30(4): 975 - 978.

[8] 钱萌, 董小明, 胡昊然, 等. 基于统计决策树的包分类算法[J]华东理工大学学报: 自然科学版, 2008, 34(3): 432 - 437.

[9] CHANG Y-K, LIN Y-S, SU C-C. A high-speed and memory efficient pipeline architecture for packet classification [C]// FCCM 2010: 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines. Piscataway: IEEE, 2010: 215 - 218.

[10] 高蕾, 谭明峰, 龚正虎. IP 报文分类算法综述与评价[J]. 计算机工程与科学, 2008, 28(3): 70 - 73.

[11] GUPTA P, McKEOWN N. Classifying packets with hierarchical intelligent cuttings [J]. IEEE Micro, 2000, 20(1): 34 - 41.

(上接第 325 页)

参考文献:

[1] RICART G, AGRAWALA A K. An optimal algorithm for mutual exclusion in computer networks [J]. Communications of the ACM, 1981, 24(1): 9 - 17.

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

[3] SANDERS B A. The information structure of distributed mutual exclusion algorithms [J]. ACM Transactions on Computer Systems, 1987, 5(3): 284 - 299.

[4] LUK W-S, WONG T-T. Two new quorum based algorithms for distributed mutual exclusion [C]// Proceedings of the 17th International Conference on Distributed Computing Systems. Washington, DC: IEEE Computer Society, 1997: 100 - 106.

[5] 李美安, 刘心松, 王征. 一种基于循环编码的高性能分布式互斥算法[J]. 电子学报, 2005, 33(8): 1397 - 1402.

[6] 李美安, 刘心松, 王征. 一种基于松弛循环差集的对称分布式互斥算法[J]. 四川大学学报: 工程科学版, 2005, 37(4): 115 - 118.

[7] 武鹏, 李美安, 裴喜春. 改进的分布式互斥请求集生成算法[J]. 计算机应用, 2010, 31(S1): 243 - 244, 250.

[8] 李美安, 林岚, 陈志党. 基于局部递归的动态多点初始化请求集生成算法[J]. 计算机应用, 2012, 32(3): 606 - 608.

[9] 李美安, 林岚, 陈志党. 基于折半加一的分布式循环请求集生成算法[J]. 计算机工程, 2012, 38(14): 59 - 61.

[10] NG W K, RAVISHANKAR C V. Coterie templates: a new quorum construction method [C]// Proceedings of the 15th International Conference on Distributed Computing Systems. Washington, DC: IEEE Computer Society, 1995: 92 - 99.

[11] CAO G H. A delay-optimal quorum-based mutual exclusion algorithm for distributed systems [J]. IEEE Transactions on Parallel and Distributed Systems, 2001, 12(12): 1256 - 1268.

[12] THIARE O, FALL P A. Using Maekawa's algorithm to perform distributed mutual exclusion in quorums [J]. Advances in Computing, 2012, 2(4): 54 - 59.