一种基于令牌的新的互斥算法分析与设计

李云鹤
（茂名学院计算机系　广东茂名 525000）

摘　要　在对现有典型分布式系统中互斥算法研究的基础上，本文依据令牌技术，提出了一种分布式系统中解决互斥问题的新算法。文中对算法的设计思想及实现过程进行了详细描述，同时对其性能进行了严格的理论证明和分析，该算法能有效地提高系统的通信效率。
关键词　令牌，分布式系统，互斥，临界区

Study on Distributed Mutual Exclusion Algorithms Based on the Token

LI Yun-He
(Department of Computer, Maoming College, Guangdong Maoming 525000)

Abstract　A new algorithm which is used in distributed system is proposed in this paper based on researching of existing algorithms and token technology. The algorithm's design, realization and performance proof have been described in this paper. The analyzed result proofs that this algorithm can effectively reduce the system's traffic.
Keywords　Token, Distributed, Mutual, Critical resistance

分布式计算系统是若干独立自治的计算机系统的集合，是计算机网络的高级发展阶段，是近年来计算机科学技术领域中备受青睐、发展迅速的一个方向。在分布式系统中，经常出现多个进程请求访问同一个临界资源的问题，为了协调访问，保证访问的正确性（无死锁，无饥饿现象），分布式操作系统必须处理相关进程之间的同步与互斥。互斥是分布式系统设计的关键问题。定义基本的操作来解决共享资源的多个并发进程的冲突问题，已成为分布式系统设计的主要问题。

1　典型互斥算法

1.1　基于非令牌的互斥算法

1.1.1　集中式算法

在分布式系统中获得互斥的最直接方法是仿照单处理机系统的方法，选一个进程为协调者。每个要求进入临界区的进程向协调者发出请求，协调者对所有的请求进行排队并根据一定的规则如时间戳（Time Stamp）授予许可。若某一时刻临界区中已有一个进程，协调者便不能同意新的请求，发出拒绝应答。当占有临界区的进程从临界区退出时，向协调者发送释放互斥消息，允许其它进程进入临界区。

可见，该算法保证了互斥的实现，协调者仅能让某一进程在某一时刻进入临界区。该算法也很公平，因为允许请求的顺序同它们接收的顺序一致，没有进程在永远等待。这种方案也容易实现，每访问一次临界区只需 4 条消息（请求、允许、拒绝、释放），它不仅仅能管理临界区，也可用于更普遍的资源管理。该算法也存在缺点，如协调者为一个单点故障，如果它发生了故障，那么整个系统将会瘫痪。

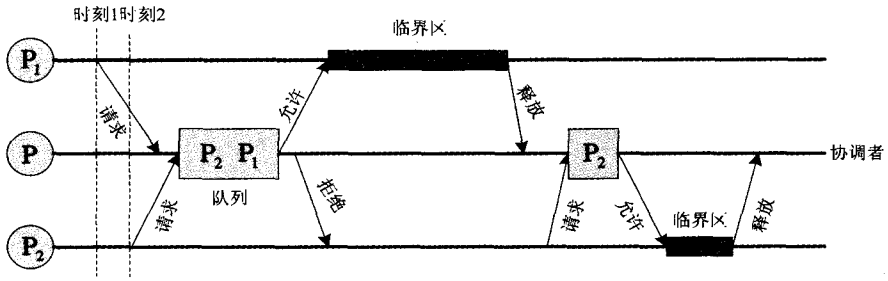

图 1　集中式算法的实现实例

1.1.2　Lamport 算法

假设所有事件都是全序的，当进程欲进入临界区时，应向其他进程广播请求，当进程收到此请求时，①若不在临界区中，也不想进入临界区，向发送者发送 OK；②若已在临界区中，不回答，负责将发送者排入请求队列；③若要进入临界区，尚未进入，则要比较时间戳，取小的进入，即发来的时间戳小，发 OK，否则同②。如所有回答均为 OK，该请求进程可进入临界区；当从临界区退出，向请求队列中所有进程发 OK，释放它们。有 3 个进程 P_1，P_2，P_3。其中 P_1 和 P_2 请求临界区（假设 P_1 在 P_2 之前请求）。Lampot 算法的具体实现过程如图 2 所示。

该算法需要进行 $2(N-1)$ 个消息交换：$N-1$ 个消息用于进程 P_i 通知所有其他进程对临界区的请求，另外 $N-1$ 个消息用于传达同意信息（Agreement）。一个进程只有收到所有其他进程的许可后才能访问临界区。经典的 Lamport 算法利用了时戳方式，通过使用带时戳的消息交换来进行分散式的资源管理和同步。Lamport 算法需要 $3(n-1)$ 个消息来保证 n 个进程集的互斥。

李云鹤　讲师，硕士，主要研究方向：数据结构与算法，嵌入式计算机应用。

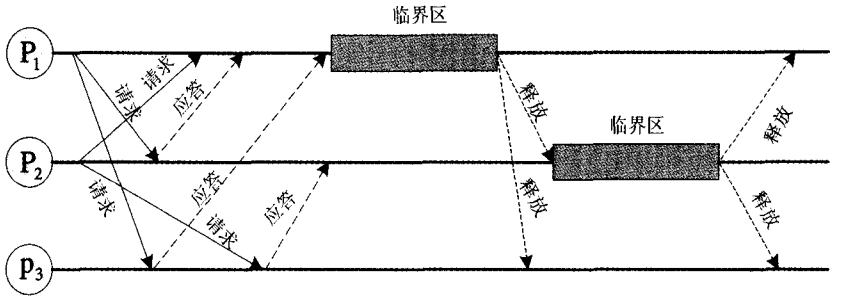

图 2　Lamport 算法的实现实例

为了提高算法的效率，作为改进，一个请求进程可以在收到预先定义的进程子集的许可信号后访问临界区，而不是必须收到所有其他进程的许可。而只需请求一个进程子集的许可。假设 R_i 和 R_j 分别是进程 P_i 和 P_j 的请求子集，要求 $R_i \cap R_j = \Phi$。由于每个进程授予一个许可信号给一个请求进程，因此互斥自动得以执行。当进程 P_i 请求临界区时，它只给 R_i 中的进程发送请求消息。当进程 R_j 收到一个请求消息时，如果它自从上一次释放临界区后还没有发出过回答消息给任何进程，它就发出一个回答消息。否则，请求消息被放入队列中。注意，批准请求的手续不是基于时戳，而是基于每个请求的到达时间，也就是说，基于时戳的强公平性在这里不被执行。然而，如果每个通道的通信延迟是有限的，就不会发生饥饿现象。进程 P_i 只有在收到 R_i 中的所有进程的回答消息后才能访问临界区。在释放临界区时，P_i 只给 R_i 中的进程发送释放消息。

考虑另外两种 R_i 的选择。在集中式互斥中，P_c（$c \in \{1, 2, \cdots, n\}$）作为唯一的仲裁者，有：$R_i : \{P_c\} 1 \leqslant i \leqslant n$，另一种极端是完全分布式的互斥算法，其中请求进程向所有其它进程请求许可：$R_i : \{P_1, P_2, \cdots, P_n\}$　$1 \leqslant i \leqslant n$。

这样容易造成死锁，因为进程有可能被其它进程锁住。死锁问题的一种解决方法是：如果某个进程请求的时戳比其它等待同一个锁请求的时戳大，则要求该进程放弃对锁的申请。另一种保守的方法保证进程只有在没有更小时戳的其它请求时才对请求进程给予回答，它类似于 Lamport 的方法，然而这种情况需要的消息交换的数量较大。

1.2　基于令牌的互斥算法

在算法中引入令牌，其思想为：所有的进程组成一个逻辑环，环中每个进程都要知道谁在它的下一个位置。而令牌代表了一个控制点，它在环上传递，一个进程当且仅当拥有令牌时可以访问临界区。当请求进程没有令牌时，算法需要 N 条消息；当请求进程持有令牌时，算法不需要产生任何消息。其具体描述如下：

如果进程 P_i（$0 \leqslant i \leqslant n-1$）连接成一个环，同样使用令牌代表一个动态的单一控制点使该令牌绕环传递，则实现了一个简单的互斥算法。

$P(i : 0, \cdots, n-1) :: =$[receive token from $P((i-1) \bmod n)$;
　消费资源;
　send token to $p((i+1) \bmod n)$
]

分布式-互斥 :: = || $P(i : 0, \cdots, n-1)$

算法的正确性是显而易见的，但也存在一些问题。比如，令牌一旦丢失，它必须重新生成。实际上，检测令牌丢失是很困难的，因为在网络上，令牌两次出现的时间是不定的，一个小时没有发现令牌并不意味着它丢失了，也许某个进程还在使用它。当进程崩溃时，该算法也会出现麻烦。如果需要进程在接收到令牌后发回确认消息，当相邻进程试图传递给它令牌却没有成功时，它就会检测到死进程。这时就将死进程从进程组中移出，它的下一个进程就会从令牌持有者手中接收到令牌。

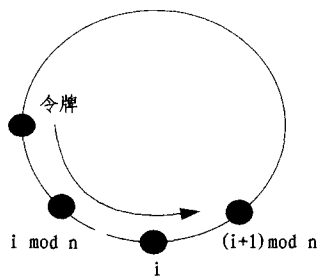

图 3　令牌算法示意图

1.3　典型互斥算法的性能比较

互斥算法的性能由以下参数衡量：①平均消息数；②同步延迟，以一个进程离开临界区到下一个进程进入该临界区之间的间隔来衡量；③反应时间，以一个进程发出请求到该进程离开该临界区之间的间隔来衡量。

以上三个参数中，①和②用于衡量一个给定的互斥算法的性能，而第③个参数更多地取决于系统中符合和每个进程在临界区执行时间的长短。典型互斥算法的性能比较如表 1 所示。

表 1　典型互斥算法的性能比较

	基于非令牌的互斥算法		基于令牌的互斥算法
	集中式	Lamport	
消息数	4	2(N−1)	1～∞
时间延迟	2	2(N−1)	0～N−1
缺点	协调者发生故障，系统会崩溃	任何进程发生故障，系统都会崩溃	令牌丢失，系统崩溃

可见，集中式算法最简单，也最有效。令牌环算法中的消息数是可变的，如果每个进程总是想进入一个临界区，则令牌的每次传递将导致一次进出临界区。平均每次进入临界区人口都有一条消息。在其他极限情况下，令牌也许将长达几小时沿环传递而没有一个进程使用它，在这种情况下，每次进出临界区消息数趋向无穷大。

本文提出了一种新的基于令牌的算法，能有效地降低通信量和保证系统安全性。

2　新令牌算法的设计

2.1　算法的设计思想

在普通的令牌算法中，令牌持有者不断变化，为了保证希

望进入临界区的进程发出的请求消息能够被令牌持有者接收并处理，算法要求进程在发送请求消息时以广播的形式发送，因此进程每次对临界区的请求都要发送 $N-1$ 条请求消息，而其中 $N-2$ 条消息不是被令牌持有者接收并处理，成为额外的无用消息，降低了算法的性能。因此，可以针对普通令牌算法中的以下两点进行改进：①减少广播消息的次数；②减少无用消息的次数。

2.2 算法的实现过程

在新的令牌算法中定义了以下三种消息：①令牌消息，即令牌，其中包含了一个请求队列 Q；②请求消息，其中包含了发送请求消息的进程的标识；③通知消息，其中包含了被选出的管理进程的标识。新算法的详细描述如下：

(1) 令牌包含一个请求队列 Q，令牌是按照 Q 中的请求的顺序从一个进程转交到另一个进程。

(2) 系统中有一个管理进程，管理进程由系统中所有进程按照一定的规则担任。所有的请求消息都发给管理进程，管理进程接收这些请求消息并将请求消息加入到本地的消息队列 P 中。

(3) 系统初始化的时候，随机地任命一个进程作为管理进程，而管理进程将创建一个令牌。这时候令牌的请求队列 Q 为空，若管理进程希望进入临界区，则直接进入，若有其他进程发送请求消息到管理进程，管理进程将在一定时间 t 内收集各请求消息并且将它们加入到本地请求队列 P 中。过了时间 t 后，管理进程队列 P 中的请求消息按优先级排序并附加到令牌的队列 Q 的末尾，接着，任命队列 Q 中最后一个请求的发送者作为新的管理进程，广播通知消息，最后，将令牌传给队列 Q 中的第一个请求的发送者，并清空本地的请求队列 P。

(4) 当进程得到令牌的时候，它就进入临界区，当它退出临界区的时候，把令牌转交给 Q 中第一个请求的发送者，并把第一个请求从 Q 中删除。

(5) 当管理进程得到令牌的时候，管理进程将已在本地请求队列 P 中的请求消息排序并追加到令牌的请求队列 Q 中，清空请求队列 P。若此时 Q 不为空，则由管理进程将令牌转交给 Q 中第一个请求消息的发送者，任命 Q 中最后一个请求消息的发送者作为新的管理进程，并广播通知消息；若此时 Q 为空，则由管理进程保留令牌，并且继续作为管理进程，之后管理进程的行为与(4)中相同。

(6) 当管理进程正处于临界区的时候，如果收到请求消息，则直接把请求消息加入到令牌的请求队列 Q 的末尾。

(7) 每个进程都包含一个 managers 数组，用于追踪管理进程的变化情况。当进程收到管理进程广播的通知消息的时候，把通知消息中包含的管理进程的标识加入到 managers 数组中，并开始将请求消息发送给管理进程。

(8) 当进程收到请求消息的时候，如果发现自己不是管理进程，则会将收到的请求消息转发给管理进程。

3 新令牌算法的性能分析

算法的性能参数“反应时间”，主要取决于系统的负载和每个临界区执行时间的长短，所以在此只对算法性能中的参数“平均消息数”和“同步延迟”进行分析。

3.1 平均消息数

假设消息在无转发的情形下，系统中有 N 个进程，管理进程执行完临界区后，会将之前收集的请求消息加入到令牌的 Q 队列中，然后选出新的管理进程并广播通知消息，接着令牌在 Q 中的请求消息的发送者间依次传递，直到传递到新的管理进程，以上过程是循环周期性地发生的。在一个周期内，假设管理进程离开临界区的时候，它已经收集的请求消息数为 $R_i(1\leqslant i\leqslant N)$ 条，则 R_i 个进程从发出请求到进入临界区的过程中一共发送了 R_i 条请求消息，$(N-1)$ 条通知消息以及 R_i 条令牌消息，因此在 n 个周期内，平均消息数为：

$$\overline{M}=\frac{\sum_{i=1}^{n}(2R_i+N-1)}{\sum_{i=1}^{n}R_i}=2+\frac{\sum_{i=1}^{n}(N-1)}{\sum_{i=1}^{n}R_i}$$

又因为 $1\leqslant i\leqslant N$，可知：$3\leqslant\overline{M}\leqslant N+1$。

假设进程 m 是当前的管理进程，且已经执行完临界区，它之前收集的请求按优先级大小排列为 $q_1,q_2,\cdots,q_i$，这些请求的发送者分别为 $p_1,p_2,\cdots,p_i$，那么进程 m 会选择 p_i 为新的管理进程，然后广播通知消息，最后把令牌传给 p_1。如果之后进程 m 收到了进程 p_j 的请求面，它会将 q_j 转发给 p_i，只要 q_j 在 p_i 得到令牌并且执行完临界区之前到达 p_i，就不需要被再次转发。因为 q_i 执行完临界区之前的时间为 i 次令牌消息的传递时间与 i 次临界区执行时间之和，大于 1 次消息的传递时间，所以 q_j 不会被再次转发，也就是说，一般情况下，请求消息只会被转发一次。

在一个周期内，假设管理进程离开临界区的时候，它已经收集的请求消息数为 $R_i(1\leqslant i\leqslant N-1)$，最坏的情况下，这 R_i 个请求消息都经过了一次转发才到达管理进程，则 R_i 个进程从发出请求消息到进入临界区的过程中一共发送了 R_i 条请求消息，$(N-1)$ 条通知消息，R_i 条令牌消息，以及 R_i 条转发消息，因此在 n 个周期内平均消息数为：

$$\overline{M'}=\frac{\sum_{i=1}^{n}(3R_i+N-1)}{\sum_{i=1}^{n}R_i}=3+\frac{\sum_{i=1}^{n}(N-1)}{\sum_{i=1}^{n}R_i}$$

亦可得知：$4\leqslant\overline{M'}\leqslant N+2$

综合可得：$3\leqslant\overline{M}\leqslant N+2$ 且仅在极端低请求负荷的情况下，改进算法的平均消息数稍多；而占大部分的其他情况下，特别是请求负荷高的情况下，改进算法的平均消息数比普通算法少得多，且请求负荷越高，差距越大。

3.2 同步延迟

每当进程离开临界区的时候，将令牌直接转交给令牌 Q 队列中的第一个请求的发送者，因此同步延迟为一个令牌消息的发送时间。

结束语 本文依据令牌算法的特点，提出了一种新的分布式系统中基于令牌技术的互斥算法。该算法能有效地降低系统中的通信量，特别是在请求负荷高的情况下，改进算法的平均消息数比普通算法少得多，且请求负荷越高，差距越大。

参 考 文 献

1 Wu Jie. Distributed Operating System Design[M]. 北京：机械工业出版社，2001

2 Wu J. 高传善，等译. 分布式系统设计[M]. 北京：机械工业出版社，2001

3 Goscinski A. A Synchronization Algorithm for Processes with Dynamic Priorities in Computer Networks with Node Failures [J]. Information Processing Letters，1989，32(3)：129～136

4 Andrews T. Distributed Operating Systems[M]. 北京：电子工业出版社，1997

5 Advantech Co. Ltd. PCL-730 User's Manual (5th Edition) [M]. Taiwan：Advantech Co. Ltd . 1998

6 汤了赢. 计算机操作系统[M]. 西安：西安电子科技大学出版社，1992

7 鞠九滨. 分布计算系统[M]. 北京：高等教育出版社，1997

8 尹俊文，邹鹏，王广芳. 分布式操作系统[M]. 长沙：国防科技大学出版社，2000