实时中间件的优先级映射*

郭长国+, 王怀民, 邹 鹏, 王 锋

(国防科学技术大学 计算机学院 网络与信息安全研究所,湖南 长沙 410073)

Priority Mapping in Real-Time Middleware

GUO Chang-Guo+, WANG Huai-Min, ZOU Peng, WANG Feng

(Institute of Network Technology and Information Security, School of Computer Science, National University of Defence Technology, Changsha 410073, China)

+ Corresponding author: E-mail: cgguo@163.net; rtcorba@sohu.com

http://www.nudt.edu.cn

Received 2002-07-25; Accepted 2002-10-11

Guo CG, Wang HM, Zou P, Wang F. Priority mapping in real-time middleware. *Journal of Software*, 2003,14(6):1060~1065.

http://www.jos.org.cn/1000-9825/14/1060.htm

Abstract: As a specification of middleware, real-time CORBA (common object request broker architecture) defines uniform CORBA priority which is independent of any operating system. The CORBA priority is carried with the CORBA invocation and is used to ensure that all threads subsequently executing on behalf of the invocation run at the appropriate native priority, which is mapped from that CORBA priority. How to map many CORBA priority levels to fewer native priority levels and maintain the order of these priorities is very important. The mapping problem is common in real-time middleware. The priority mapping mechanism of real-time CORBA is introduced briefly. The two methods of how to map middleware priority to native operating system priority are presented, the two methods are static mapping method and dynamic mapping method. Static mapping method is simple to be implemented, but it can break the interoperability, portability and the strict order of middleware's priorities. Dynamic mapping method is complex to be implemented, but it can maintain the strict order of middleware's priorities.

Key words: real-time CORBA (common object request broker architecture); priority mapping

摘 要: 实时 CORBA(common object request broker architecture)规定了独立于操作系统的 CORBA 优先级,每个请求都可以携带 CORBA 优先级,而执行请求的所有线程都使用该 CORBA 优先级映射后的本地操作系统优先级.如何将范围较大的 CORBA 优先级映射到范围较小的操作系统优先级,并且尽量保持 CORBA 优先级的序关系,避免优先级翻转是实时 CORBA 需要解决的重要问题.该问题在实时中间件中具有普遍意义.简要介绍了实时中间件中的优先级映射机制,提出了中间件优先级到本地操作系统优先级的静态散列法和动态映射法,并给出了动态映射法的实现算法.散列法实现简单,但会影响中间件的互操作和可移植性,难以保证优先级映射的

* Supported by the National Natural Science Foundation of China under Grant No.90104020 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2001AA113020 (国家高技术研究发展计划(863)); the National Grand Fundamental Research 973 Program of China under Grant No.G1999032703 (国家重点基础研究发展规划(973))

第一作者简介: 郭长国(1973－),男,河南武陟人,博士,主要研究领域为分布计算技术,数据库技术.

严格序关系;动态映射法实现复杂,但能严格保证优先级映射的序关系.

关键词: 实时 CORBA;优先级映射

中图法分类号: TP311 **文献标识码**: A

OMG(object management group)制定的具有固定优先级调度能力的实时 CORBA[1](common object request broker architecture)为分布式实时系统的开发制定了相关的标准.使用实时 CORBA,发送请求的一方(客户)可以为请求规定优先级,ORB(object request broker)在传输和执行请求时,将按照请求优先级的高低区别加以对待:优先处理高优先级的请求.由于典型的分布式应用通常涉及多个计算节点,涉及多种异构的操作系统平台,而且不同的操作系统往往具有不同的优先级能力,如何在不同的操作系统平台上协调优先级是实时 CORBA 的一个重要内容.比如 Solaris 上的应用可能为请求规定优先级 100,如果服务方运行于 Windows 平台,它如何处理这个优先级?因为 Windows 平台只有 32 个优先级.实时 CORBA 规定请求的优先级使用 CORBA 优先级,它独立于任何操作系统的优先级,CORBA 优先级具有 2^{15} 个级别,但是 CORBA 优先级如何映射到通常级别少得多的操作系统优先级,并且保持优先级之间的序关系?

实时 CORBA 中的优先级映射问题在实时中间件的优先级映射中具有普遍性.本文探讨了实时中间件中的优先级映射机制,提出了实现优先级映射的散列法和动态映射法,并分析和比较了各种方法的优缺点.

CORBA 体系结构和实时 CORBA 可参见文献[2],由于实时 CORBA 中将调度的最小单元指定为线程,所以若无特殊指明,本文中的优先级均指线程优先级.

1 实时 CORBA 的优先级映射

为了屏蔽操作系统的差异,实时 CORBA 将优先级区分为两种:CORBA 优先级和本地优先级.CORBA 优先级是独立于任何具体操作系统的优先级,本地优先级是具体操作系统的优先级.本地优先级的特性由具体的操作系统决定,比如 Windows2000 和 Solaris 操作系统就具有完全不同的优先级模式.本地优先级的差异主要体现在优先级级别个数上的不同、优先级序关系的不同、优先级高低与优先级值的关系不同(有的操作系统优先级值越小,级别越高).

基于 CORBA 优先级和本地优先级,实时 CORBA 定义了优先级传播机制:

客户方规定请求优先级时使用 CORBA 优先级,假设为 x.CORBA 优先级 x 随请求传播到服务方,实时 CORBA 中称为优先级的客户传播模式(client propagated priority model).服务方在执行请求时,首先将 CORBA 优先级 x 映射为本地操作系统的优先级 w,然后在本地优先级 w 下执行请求.在请求执行过程中,如果需要访问另一个服务(即出现了请求的嵌套),并且该服务也使用客户广播模式,则嵌套的请求也携带 CORBA 优先级 x.

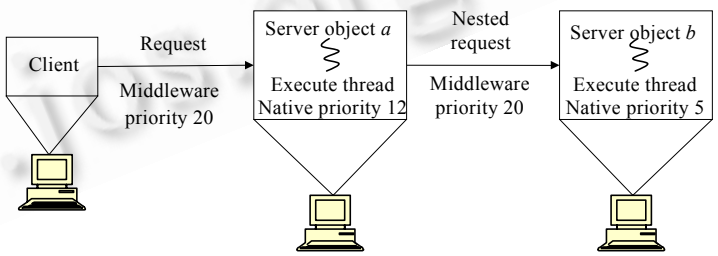

Fig.1 Priority mapping in real-time CORBA

图 1 实时 CORBA 优先级映射

图 1 给出了一种传播与映射实例,客户和服务对象 a 以及对象 b 分别位于 3 个不同的计算节点,客户规定请求的 CORBA 优先级 x=20,在对象 a 中执行时,映射为本地线程的优先级 w_a=12,而在对象 b 中执行时,映射为本地线程的优先级 w_b=5.在请求的传输过程中都使用 CORBA 优先级 x.

2 优先级映射方法

本文提出优先级映射的方法,一种是静态方法,称为散列法,另一种是动态映射法.

2.1 系统定义

为了描述实时中间件的优先级映射机制,首先定义系统.中间件优先级使用全序结构 P_m 表示:

$$P_m=\langle\{m_1,m_2,\ldots,m_k\},\prec_m\rangle,\text{记 } P_c=\{m_1,m_2,\ldots,m_k\},\prec_m=\{\langle m_i,m_j\rangle|m_i\in P_c,m_j\in P_c,\text{且 } i<j\}.$$

本地优先级使用全序结构 P_l 表示:

$$P_l=\langle\{n_1,n_2,\ldots,n_l\},\prec_n\rangle,\text{记 } P_n=\{n_1,n_2,\ldots,n_l\},\prec_n=\{\langle n_i,n_j\rangle|n_i\in P_n,n_j\in P_n,\text{且 } i<j\}.$$

n_1 是 P_n 中的最小元,记为 $\Im$;n_l 是 P_n 中的最大元,记为 $\aleph$.

在本文的讨论中,均假设 $k>l>1$.在不导致混淆的情况下,为便于表述,我们使用 $\prec$ 表示 $\prec_n$ 和 $\prec_m$,根据上下文能够确定 $\prec$ 的确切含义.本文也使用 x,y 表示 P_c 的元素.

优先级映射 f 是定义在 P_m 上的函数,即:

$f:P_m\to P_l$,我们定义:

$\forall m_i\in P_c,\forall m_j\in P_c$,如果 $m_i\prec m_j\Rightarrow f(m_i)\prec f(m_j)$,我们就称 f 严格保序;

$\forall m_i\in P_c,\forall m_j\in P_c$,如果 $m_i\prec m_j\Rightarrow f(m_i)\prec f(m_j)$或者$f(m_i)=f(m_j)$,我们称 f 部分保序关系,并且将$f(m_i)\prec f(m_j)$或者$f(m_i)=f(m_j)$记为$f(m_i)\succcurlyeq f(m_j)$.

优先级映射必须解决保序性:f 至少部分保序,最好能够严格保序,这样才能够避免发生优先级翻转[3],即低优先级的活动阻塞高优先级的活动.

2.2 散列法

在散列法中,将 P_c 分成不相交的 l 段,记为 $D_1,D_2,\ldots,D_l$.

$|D_i|\geq 1,1\leq i\leq l$;

$\forall x\in D_i(1\leq i\leq l),\forall y\in D_j(1\leq j\leq l)$,如果 $i<j$,则 $x\prec y$.

f 将每一段中的优先级都映射为一个相同的操作系统优先级,并且不同的段映射为不同的优先级:

$f(x)=n_i,x\in D_i$.

根据 f 定义,可知散列法有如下结论:

定理 1. $\forall x\in D_i(1\leq i\leq l),\forall y\in D_j(1\leq j\leq l)$,如果 $i<j$,则$f(x)\prec f(y)$.

f 是部分保序的,所以可能会产生优先级翻转.

散列法的缺点是受制于操作系统的本地优先级个数.比如针对 Windows 平台,通常情况下只有 7 个优先级,最多也只有 16 个优先级,对实时应用的实现非常不便.

根据 D_i 的不同选择方法,散列法可以分为平均散列法和区段散列法.

2.2.1 区段散列法

在区段散列法中,我们只取 P_c 中的一段中间件优先级,将它一一映射到 P_n.即在 P_c 中选取一个区间$[a,a+l]$,其中,$0\leq a\leq k-l$,即 $D_i=\{x|x\in P_c,x=a+(i-1)\},1\leq i\leq l,|D_i|=1$.

f 定义如下:

$$f(x)=\begin{cases}\Im, & x<a\\ n_i, & x=a+i-1,1\leq i\leq l.\\ \aleph, & x\geq a+l\end{cases}$$

根据 f 定义有:

定理 2. $\forall x,\forall y\in[a,a+l]$,如果 $x<y$,则$f(x)\prec f(y)$,即区段散列法在其区段$[a,a+l]$中的映射是严格保序的.

根据定理,应该在区段$[a,a+l]$中选择中间件优先级,这样才能够避免发生优先级翻转.

区段法的优点是映射简单.其缺点是互操作性和可移植性的损失.因为不同的中间件可能选择不同的区段,导致不同中间件的优先级不能互操作.

2.2.2 平均散列法

定义 $B=\left\lfloor\frac{k}{l}\right\rfloor$,$\lfloor a\rfloor$ 表示对 a 按照四舍五入取整,比如 $\lfloor 2.3\rfloor=2,\lfloor 4.6\rfloor=5$.在平均散列法中,取$|D_1|=|D_2|=,\ldots,|D_{l-1}|=B$.

$\forall x\in P_c$,如果 $x<B*l$,则 $x\in D_i \Leftrightarrow B*(i-1)\le x<B*i, 1\le i\le l$,$\forall x\in P_c$,如果 $x\ge B*l$,则 $x\in D_l$.

即从 0 开始,每 B 个 P_c 划分为一段.这时的映射定义如下:

$$f(x)=\begin{cases} n_i, & x<B*l, B*(i-1)\le x<B*i, 1\le i\le l \\ n_i, & x\ge B*l \end{cases}.$$

根据f的定义,有:

定理 3. $\exists x,y\in P_c, x\prec y$,但是$f(x)=f(y)$,即平均散列法只能部分保序.

这是平均散列法的主要缺点,因此在使用平均散列法时,应该选择不同段中的优先级.

定理 4. $\forall m_i\in P_c, 1\le i<k-B, f(m_i)\prec f(m_{i+B})$,即在选择中间件优先级时,可以按照跨度为 B 来选择.

平均散列法的优点是提高了系统的互操作性和可移植性.如果所有的服务方中间件都采用散列法,则所有的中间件优先级都有对应的本地优先级,但是这是以牺牲严格的优先级关系为代价的.比如客户对两个重要性不同的请求 q_1 和 q_2,客户为 q_1 规定 CORBA 优先级 x,为 q_2 规定中间件优先级 y,$x\prec y$,但在某个映射f下可能出现$f(x)=f(y)$,即执行时无法区分请求的重要性差异.各种散列法如图 2 所示.

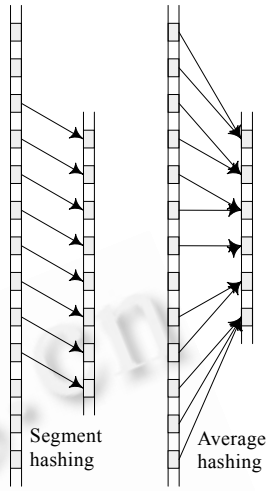

Fig.2 Static mapping

图 2 静态映射法

2.3 动态映射法

在静态情况下,不可能找到严格保序的映射,为了严格保序,本文提出一种动态映射法.

动态映射法的基本思想如下:假设系统有 N 个本地优先级,在某一时刻,系统需要有 M 个中间件优先级级别需要映射.如果 $M<N$,则在该时刻,可以将全部的中间件优先级映射到本地优先级上,如果 $M>N$,则在该时刻从 M 个中间件优先级中选择 N 个中间件优先级级别(为保序,我们选择优先级高的前 N 个),将它们映射到本地优先级,其他中间件优先级在 $M-N$ 中的线程则被暂停执行.当需要映射的中间件优先级级别发生变化时,需要重新调整映射,所以这种方法是一种动态映射法.

我们使用 S 表示系统的状态序列,系统在运行过程中分别经历 $s_1,s_2,\ldots$,即 $S=s_1,s_2,\ldots$

对任意的系统状态 s_i,如果发生下列事件中的任意一个,则系统迁移到下一个状态 s_{i+1}:① 系统中有新线程加入,② 系统中有线程退出(结束),③ 系统中有线程的中间件优先级发生变化,④ 系统中有线程的状态发生变化(由 Ready 到 Wait,或者相反).

动态映射就是在每一个系统状态 s_i 下,使用一个和状态相关的映射 f_{s_i} 来完成中间件优先级到本地优先级的映射,一般来讲,s_i 不同 f_{s_i} 也不同.为刻画系统状态,我们定义如下:

系统中的线程定义为 $t_i, i=1,2,\ldots; t_i=\langle c_i,l_i,h_i\rangle, c_i\in P_c$ 表示 t_i 的中间件优先级,$l_i\in P_n$,表示 t_i 的操作系统优先级,h_i 表示线程状态,$h_i\in\{r,w,s\}$,r 表示 Ready,w 表示 Wait,s 表示暂停(suspend).我们使用 R 表示系统中 Ready 的线程和暂停的线程集合.

在 R 上定义操作 C,$C(R)$表示 R 中所有线程的中间件优先级集合,即

$$C(R)=\{x\mid x\in P_c\}, \forall x\in C(R), \exists t_i\in R, c_i=x; \forall t_i\in R, c_i\in C(R).$$

在 $C(R)$上定义操作 Z,$Z(x)$表示按照 $\prec$ 关系确定的 x 在 $C(R)$中的位置,$Z(x)=j$,表示 x 在 $C(R)$中是第 j 个低的中间件优先级,即 $Z(x)=j \Leftrightarrow \exists Y\subset C(R), \forall y\in Y, y\prec x; \forall r\in(C(R)-Y), x\succ r; |Y|=j-1$.

在系统的执行状态 s_i 下:

对 $\forall t_i=\langle c_i,l_i,h_i\rangle\in T$,当 $|C(R)|\le l$ 时,

$$f_{s_i}(t_i)=\begin{cases} \langle c_i, \Im, h_i\rangle, & h_i=w \\ \langle c_i, n_j, r\rangle, & m_i\in\{r,s\}, Z(c_i)=j \end{cases}.$$

该映射函数表示当需要的中间件优先级级别比本地优先级级别少时,可以将所有非 Wait 线程投入运行.由于 Wait 线程不具备获得 CPU 的条件,所以将它们的优先级设置为系统中最低的本地优先级.

对 $\forall t_i=\langle c_i,l_i,h_i\rangle\in T$,当 $|C(R)|>l$ 时,

$$f_{s_i}(t_i)=\begin{cases}\langle c_i,\Im,h_i\rangle, & h_i=w\\ \langle c_i,\Im,s\rangle, & Z(c_i)\le|C(R)|-l\\ \langle c_i,n_j,r\rangle, & Z(c_i)-(|C(R)|-l)=j>0\end{cases}.$$

该映射函数表示,当需要的中间件优先级级别比本地优先级级别多时,将中间件级别按序排列,位于前 l 的所有非 Wait 线程投入运行,并保证它们之间的优先级序关系;暂停位于 l 之后所有非 Wait 线程.由于 Wait 线程不具备获得 CPU 的条件,所以将它们的优先级设置为系统中最低的本地优先级.

动态优先级需要在中间件中设置一个调度线程(进程),它通常运行在系统的最高优先级,并监控系统状态,当状态发生迁移时,按照映射的方法,重新调整系统的优先级映射.

假设在状态 s_i 下,任意两个线程 $t_i=\langle c_i,l_i,h_i\rangle,t_j=\langle c_j,l_j,h_j\rangle$,在映射 f_{s_i} 下有

$$f_{s_i}(t_i)=\langle c_i,l_i^{f_{s_i}},h_i^{f_{s_i}}\rangle, f_{s_j}(t_j)=\langle c_j,l_j^{f_{s_i}},h_j^{f_{s_i}}\rangle.$$

动态映射有如下结论:

命题 1. 如果 $c_i=c_j$,并且 $h_i\neq w,h_j\neq w$,则 $l_i^{f_{s_i}}=l_j^{f_{s_i}}$,并且 $h_i^{f_{s_i}}=h_j^{f_{s_i}}\neq w$.

即两个中间件优先级相等的非 Wait 线程在任意状态下,它们映射后的本地优先级相同.

命题 2. 如果 $c_i\prec c_j,h_i\neq w,h_j\neq w$,并且 $h_i^{f_{s_i}}=r$,则 $l_i^{f_{s_i}}\prec l_j^{f_{s_i}}$,并且 $h_j^{f_{s_i}}=r$.

即两个非 Wait 线程在任意状态下,如果中间件优先级低的线程映射后可以执行,则优先级高的线程也可以执行,并且它们严格保序.

命题 3. 如果 $c_i\prec c_j,h_i\neq w,h_j\neq w$,并且 $h_j^{f_{s_i}}=s$,则 $l_i^{f_{s_i}}=l_j^{f_{s_i}}=\Im$,并且 $h_i^{f_{s_i}}=s$.

即两个非 Wait 线程在任意状态下,如果中间件优先级高的线程映射后被暂停,则优先级低的也被暂停.

命题 4. 如果 $c_i\prec c_j,h_i=h_j=r$,并且 $h_j^{f_{s_i}}=h_i^{f_{s_i}}=r$,则 $l_i^{f_{s_i}}\prec l_j^{f_{s_i}}$.

即两个 Ready 线程在任意状态下,如果映射后它们都可以执行,则它们严格保序.

根据以上命题,我们容易得到:

定理 5. 动态优先级映射是严格保序的.

定理 5 可以表述为:任意两个线程 $t_i=\langle c_i,l_i,h_i\rangle,t_j=\langle c_j,l_j,h_j\rangle$,在 t_i 和 t_j 的整个执行过程中,$c_i\prec c_j$,则不存在状态 s_i,在该状态下 $h_j^{f_{s_i}}=h_i^{f_{s_i}}=r$,并且 $l_i^{f_{s_i}}=l_j^{f_{s_i}}$ 或者 $l_j^{f_{s_i}}\prec l_i^{f_{s_i}}$;也不存在状态 s_k,在该状态下 $h_j^{f_{s_k}}=s,h_i^{f_{s_k}}=r$.

动态映射算法不受操作系统优先级个数的限制,能够保持优先级之间的高低关系,也很好地保持了中间件的互操作性和可移植性,其缺点是调度线程需要消耗计算资源.

散列法可以认为是动态法的一个特例.

在动态映射法具体实现时,可以使用如下算法:

算法 1. 动态映射的调度算法.

记 T 为 P_c 的集合,并且定义如下两个关于 T 的函数:

min(T)是集合 T 中优先级最小的元素,即如果 $j=\min(T)$,则 $\forall x\in T$,均有 $j\le x$.

max(T)是集合 T 中优先级最大的元素,即如果 $j=\max(T)$,则 $\forall x\in T$,均有 $j\ge x$.

```
1.  T=∅
2.  for (ORB 中的每一个线程 K){
3.    if (K 的操作系统线程状态是 Ready)
4.        T=T∪{K 的 CORBA 优先级};
5.    else
6.        将 K 对应的操作系统优先级设置为 ℑ ;//最低
7.  }
8.  if (|T|≤l){
9.  for (i=1;i≤|T|;i++)//从低到高依次设置{
```

```
10.    j=min(T);
11.    将 CORBA 优先级为 j 的线程的操作系统优先级设置为 n_i;
12.    T=T−{j};
13.  }
14.  }else {//|T|>1
15.  for (i=0;i<n;i++){//从高到低依次设置
16.    j=max(T);
17.    将 CORBA 优先级为 j 的线程的操作系统优先级设置为 n_{l−i};
18.    T=T−{j};
19.  }
20.  Suspend 其他所有线程;
21.  }
```

3 比 较

表 1 从不同的方面总结了各种优先级映射方法的优缺点.每种映射方法都有各自的优缺点,根据不同的系统特性,可以选择不同的映射方法.散列法适合操作系统优先级较多的平台,比如基于 Pthread 的 Solaris 平台等等,在散列法中,区段散列方法使用得比较多.在 Windows 等操作系统优先级比较少的平台中可以使用动态映射方法,某些支持动态全局优先级调度的系统也使用动态调度方法,比如 URI(university of Rhode island)的动态实时 CORBA 系统[4].在实际系统中,往往混合使用多种方法,比如在我们实现的实时 ORB 中,在 Solaris 平台上使用平均散列法,在 Windows 平台上使用动态映射法,为了使用户可以定制应用,我们的实时 ORB 除了提供缺省的优先级映射方法以外,还提供重载优先级映射的机制,以便用户可以根据自己的需求定制自己的映射方法.

Table 1 Priority mapping comparison

表 1 映射方法对比

	Segment hashing method	Average hashing method	Dynamic mapping method
Range of CORBA priority	Part	All	All
Number of native priority	Limited	Limited	No limited
Order of priority	Strict order	Partial order	Strict order
Interoperability and portability	Breaking (interoperability and portability)	Maintain (interoperability and portability)	Maintain (interoperability and portability)
Strict order of priority mapping	Choose the priorities among segment	Choose the priorities according to the span B	Always maintain (strict order)
Expense of efficiency	Yes	Yes	No
Implementation	Simple	Simple	Complex

4 结 论

本文根据实时 CORBA 的优先级映射问题,提出了具有普遍意义的实时中间件优先级映射模型以及散列法和动态法两种映射机制,分析了各种机制的优缺点和映射的序关系,特别提出了能够严格保序的动态优先级映射机制.本文对实时中间件的优先级映射实现具有指导意义.

References:

[1] Object Management Group. The common object request broker: architecture and specification, 2.4.1 ed. 2000.

[2] Schmidt DC, Khans F. An overview of the real-time CORBA specification. IEEE Computers, 2000,33(6):56~63.

[3] Sha L, Rajkumar R, Lehoczky JP. Priority inheritance protocols: An approach to real-time synchronization. IEEE Transactions on Computers, 1990,39(9):1175~1184.

[4] Wolfe VF, DiPippo LC, Ginis R, Squadrito M, Wohlever S, Zykh I, Johnston R. All expressing and enforcing timing constraints in a dynamic real-time CORBA system. Real-Time Systems Journal, 1999,16(2-3):253~280.