一种可靠名字服务的设计与实现*)

Design and Implementation of a Reliable Name Service

李亚伟　陈　松　周明天

(电子科技大学计算机学院　成都610054)

Abstract The limitations of traditional name services in constructing dependable systems are discussed in this paper, and TongCos, a reliable name service based on replication of name servers and objects, is presented. TongCos Implements a consistency algorithm founded on VSC model. By introducing a loose binding mechanism between objects and name strings, TongCos guarantees a conspicuous enhancement of reliability without losing scalability in systems.

Keywords Middleware, Single point of failure, VSC, Smart stub

1 引言

随着 Internet 的日益发展,许多应用系统迫切需要能够迁移到 Web 上。而 Internet 不可靠的特性要求中间件平台向上提供可靠服务来保证应用的健壮性。但是,主流的工业标准如 CORBA、J2EE[1]都没有给出构筑可靠系统的规范,系统可靠性的保证只有依赖于应用系统开发者。更为严峻的是,当前主流的中间件平台在实现分布式系统的关键部件——名字服务器时基本上都采用"多客户——单服务器"模式,使名字服务器成为系统的单个失效点(single points of failure)[2]。因此,实现可靠的名字服务已经成为当前中间件领域一个研究热点。

在本文中,我们对 J2EE 中的 JNDI 名字服务接口的体系结构进行了扩展,提出并实现了一个基于 VSC 虚同步模型[3]的名字服务 TongCos。TongCos 支持服务器和对象两个层次上的复制,以集群方式为系统提供了可靠的服务。

我们首先剖析了传统名字服务在构造可靠系统方面的缺陷,并在此基础上引出了 TongCos 的体系结构。详述了 TongCos 实现服务器名字信息一致和复制对象透明切换的关键技术。对国内外相关工作进行了比较。最后给出了结论。

2 TongCos 的体系结构

2.1 传统名字服务的弊端

2.1.1 *单失效点的引入* 传统的名字服务使用单个名字服务器来集中管理对象名字,提供绑定、解绑定、解析等基本操作。典型的例子有 OMG 提出的基于 CORBA 环境的 CosNaming,SUN 提出的基于 JAVA 环境的 RMI Registry。在这些名字服务中,大量的应用服务对象都绑定在同一个名字服务器中。一旦名字服务器进程失效,其中注册的所有服务对象无法被访问(即使它们可用),导致名字服务器成为单个失效点。这是可靠系统不能忍受的。

2.1.2 *紧密的名字绑定机制* 传统名字服务的核心实现都依赖于符号名字与对象物理地址一一对应的紧密绑定机制。这种机制在只有单个对象实例情况下才是非常有效的[4]。然而在可靠系统中,对象名字与其物理地址的绑定关系要复杂得多。系统通常使用对象复制技术来提高可靠性,这对名字服务模型的绑定机制提出了挑战:对象复制要求客户能通过对象名字使用多个对象的拷贝,因此对象的名字与对象物理地址应该是"一对多"的松散绑定关系。而现有名字系统局限于"一对一"紧密绑定的机制,无法支持对象复制。

2.2 TongCos 的体系结构

我们的 TongCos 名字服务器实现了 JNDI-CosNaming 映射服务接口,为基于 RMI-IIOP 通信协议的 EJB 对象提供了名字服务功能。TongCos 从名字服务器的体系结构和对象地址绑定两个抽象层次进行了改进,实现了名字服务器和对象两个级别的复制。图1示出了 TongCos 名字服务的体系结构。

图1中,TongCos 名字服务器 Server1、Server2在管理自己单独的名字空间同时,通过事件通道连接构成服务器组(其下的通信组件为 CMU 在 JavaGroup 中实现的 NAKACK 协议)。一个组中的 TongCos 名字服务器依靠 VSC 层和失效检测层的实现对名字操作的语义进行了扩展,使作用于单个 TongCos 服务器的操作能可靠传播到服务器组中的每个成员,保证了各服务器间名字信息的一致。这样,即使某个名字服务器如 TongCos Server 1崩溃不可用,客户可以从其它的名字服务器如 TongCos Server2中获得绑定在原来 TongCos Server1名字空间的信息,从而消除了名字服务器成为系统单个失效点的可能性。

同时,TongCos 允许位于不同主机上的多个复制对象使用相同名字绑定到 TongCos 名字服务器中,如图1中与名字字符串"object"关联的三个复制对象 object1、object2、object3。TongCos 将这些等同的复制对象组织成对象组(图1中灰色椭圆代表了一个对象组)。TongCos 在建立名字绑定关系时,是将一个符号串与一个对象组进行关联。而客户端使用 Smart Stub 机制,能够透明地切换复制对象,刷新对象引用,对客户屏蔽了复制对象的物理位置,有力地支持了对象复制。

*)本文的工作得到信息产业部和四川省重点基金 DJ8.1.1和 SJ16.1的资助。李亚伟　硕士生,主要研究领域为分布式对象计算、对象中间件。陈　松　硕士生,主要研究领域为分布对象计算、对象中间件。周明天　教授,博士生导师,主要研究领域为计算机网络技术、计算机安全、分布式对象计算。

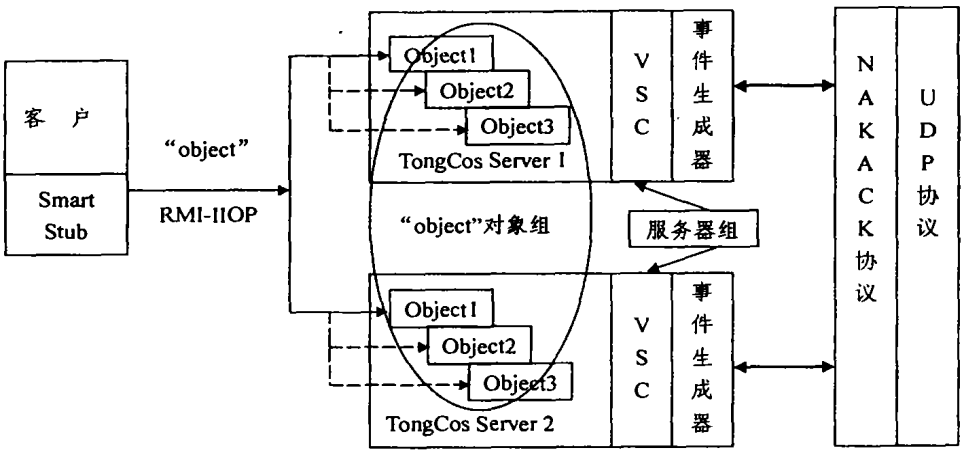

图1 TongCos 的体系结构

3 TongCos 实现的关键技术

3.1 虚同步模型保证名字空间树的完全复制

TongCos 名字服务实现可靠性的关键在于各个服务器均维护一棵完全复制的名字树，形成如图2的名字森林。

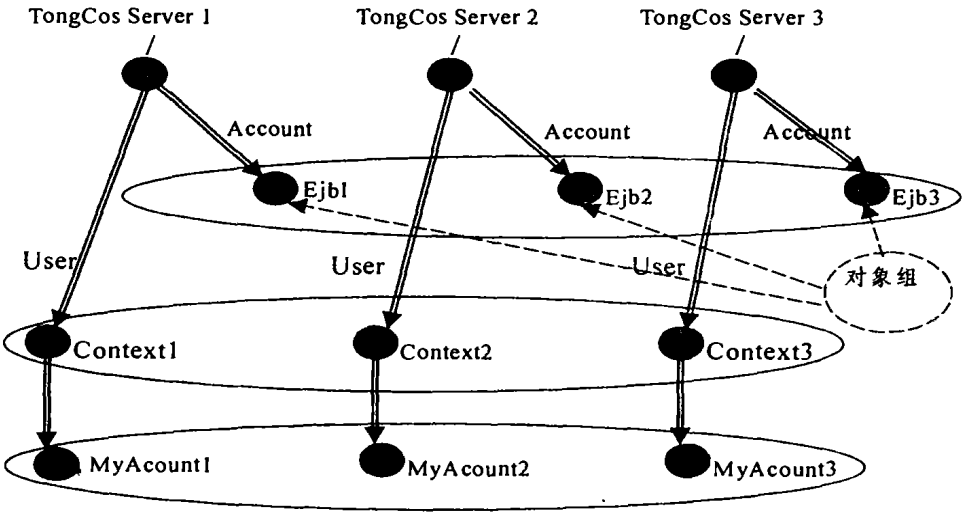

图2 TongCos 名字空间的结构

图2中的名字空间结构要求扩展定义在名字上下文上的操作语义，使单个名字服务器上信息变化的消息传播到其他 TongCos 名字服务器上。由于名字信息量的传播需要相对少的处理，TongCos 服务器间通信协议选择了高效率的 UDP，以提高系统的灵活性和性能。

然而由于 UDP 的不可靠性，简单的数据报传递并不能解决名字信息一致性问题。下面以一个实例来说明。假设一个新的对象 NewAccount 以名字"User/NewAccount"绑定到图2中的 Server 1上。此时，Server1中的名字上下文"User"的 Bind 操作会将该信息依次传递给 Server 2和 Server3。如果 Server1在将信息传递给 Server2之后便崩溃了，Server2和 Server3之间就产生了名字空间的不一致。

为此，我们引入了虚同步模型（virtual synchrony Model）。虚同步模型是由 Birman 等人在 Isis[5]系统中提出的组通信协议模型，其基本思想是通过处理成员间的组播信息和成员变化信息[6]来保证一个进程组内的成员变化能够被其他所有成员以相同顺序认识。

我们为 TongCos 提出了一种层次化的 VSC 协议。它丰富了 VSC 模型，解决了名字服务器之间的一致性问题。其算法实现分为两层：事件生成器层和 VSC 层。事件生成器层对网络信息进行了抽象，为上层生成服务器组中成员的崩溃、加入和离开事件，并向 VSC 层转发事件。VSC 层对成员的加入、离去和崩溃事件进行协调处理，通过对名字信息的调整来达到各名字服务器之间的一致性。

3.1.1 事件生成器的实现 事件生成器由两个守护线程 FaultDetector 和 GroupListener 实现。FaultDetector 维持着一个成员列表 members，并通过发送 HeartBeat 消息来轮询服务器组成员。如果某个轮询对象的响应超时，则判断该对象失效。其 JAVA 伪码如下所示：

```
FaultDetector.run(){
  while(member.size>=2){
    收到响应标志 ack_rcv = false;
    通过事件通道向当前被检测成员 dest 发送一条 heartbeat 消息;
    线程阻塞 timeout 时间;//此间如事件通道穿上来响应消息，则将 ack_rev 置为 false;
    if ( ack_rcv ==false) {
      服务器响应超时，向 VSC 上传一个 Failure 事件;
    }
    dest = memers.next;
  }
}
```

GroupListener GroupListener 用来监听是否有新成员加入和

旧成员离开。其 Java 伪代码如下所示：

```
GroupListener.run(){
  while(IsController){
    receive(msg);
    switch(msg.type){
      case JOIN_GROUP : 向 VSC 上传一个 JOIN 事件;break;
      case LEAVE_GROUP: 向 VSC 上传一个 LEAVE 事件;
break;
      default :break;
    }
  }
}
```

3.1.2 VSC 算法 TongCos 在实现 VSC 算法时定义了两种角色：控制者 Controller 和成员 Member。每个组由唯一的控制者负责在成员发生变化时计算新的状态和控制名字信息的一致性传递。每个成员配合控制者工作，以保证算法的正确性。该算法给出了控制者实现 VSC 算法的主要方法 Adjust。

```
Adjust(NewContext , LeaveContext, FailureContext)
{
  NewContext 为新加入的组成员,LeaveContext 为离开的组成员,
  FailureContext 为失效的组成员;
  计算原来组成员视图中需要收集不稳定消息的组成员集合
  Flush_Context = CurrentContext + LeaveContext - FailureCon-
  text;
  计算新的组成员视图 NewView = CurrentContext +NewContext
  - FailureContext;
  计算需要得到新视图通知的成员集合 Dest = CurrentContext +
  LeaveContext + NewContext-FailureContext;
  发送 getState 消息给 Flush_Context 集合中的每个成员,各成员
  调用自己的 returnState 方法向控制者发送自己的名字信息.控制
  者从各个成员收集的状态信息计算不稳定的名字信息集合(即没
  有在全组内得到一致确定的信息)UnStableMsgSet = ∪Flush_
  Context[i].State-∩Flush_Context[i].State,将 UnStableMsgSet
  向 Flush_Context 集合中所有成员进行可靠广播.而各成员调用
  自己的 SetState 方法刷新名字信息;
  向 Dest 中发送成员视图变化消息 View_Changed,使 Dest_View
  中各成员利用该机会刷新自己的成员视图为 NewView ;
}
```

图3以一个具体实例来说明上述算法在成员关系发生变化的情况下，保证名字信息一致的过程。图3中，TongCos Server s1、s2、s3、s4、s5组成一个名字服务器组，s3为控制器。5个服务器此前维持一致的名字信息(m0)，组视图为(s1,s2,s3,s4,s5)。在 T1时刻，s1执行了一个 bind("user\helloejb",obj)的操作，并分别向 s2、s3、s4、s5组播该消息 m1。在 T2时刻，s4执行了 bind ("user\myejb2")操作，并向 s1,s2,s3,s5组播该消息 m2。假定在 T3时刻，s1崩溃，s4离开服务器组。而此时，组中(s3,s5)没有收到 m1,(s2,s3)没有收到 m2,也无法进行重传和确认，造成了 T3到 T4时刻组间状态的不一致。在 T4时刻，Fauldetector 检测到了成员变化关系，由控制者 s3执行 VSC 调整算法 adjust(null,s1,s4)。s3计算出 UnStableMsgSet 为(m1,m2)，并向新的成员视图(s2,s3,s5)发送 UnStableMsgSet 和新的成员视图信息。最后在 T5时刻，组中形成新的成员视图(s2,s3,s5)，而且组间名字信息达到一致，均为(m0,m1,m2)。到此，算法成功。

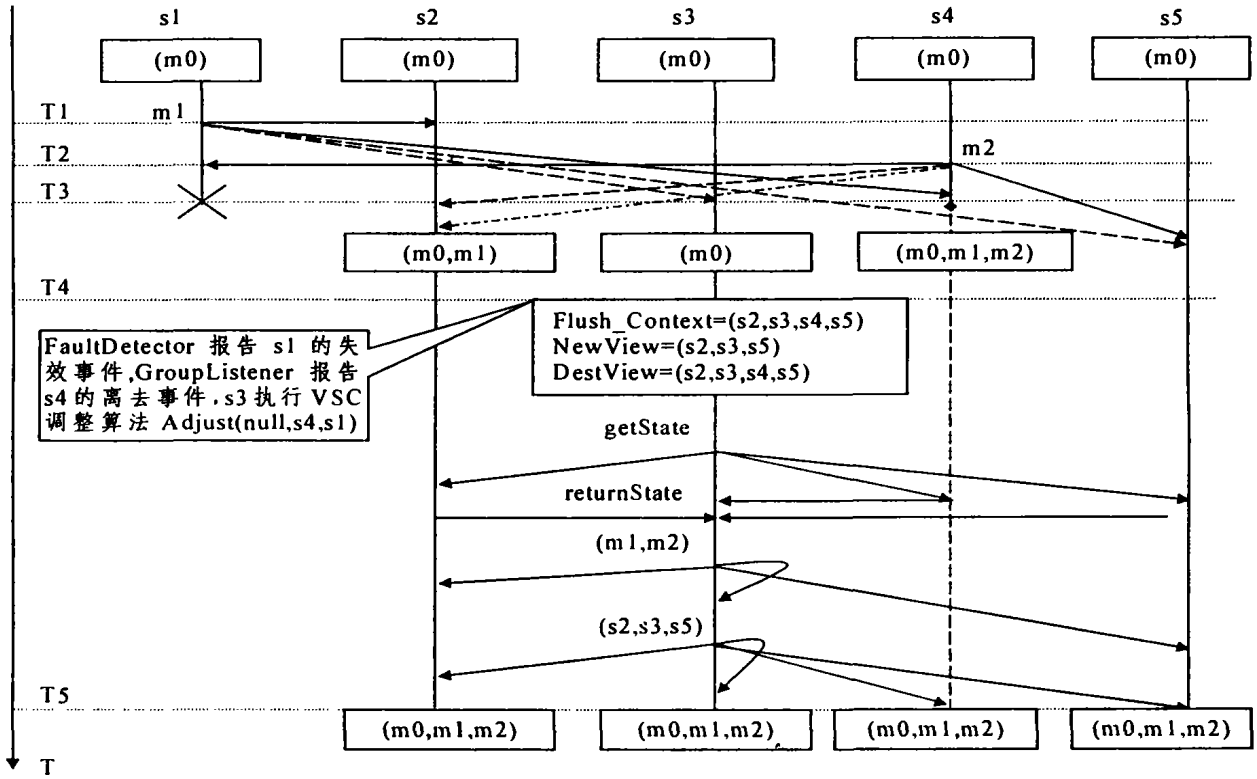

图3 VSC 算法的应用实例

3.2 基于松散绑定机制的对象复制

可靠系统中，应用需要将多个复制对象以相同名字绑定到名字空间中构成一个对象组。一旦对象组中有一个复制对象失效，失效恢复机制可以将客户的请求重发给其他的复制对象，来继续应用逻辑。

TongCos 使用组 ID 和 Smart Stub 相结合的机制来支持对象复制。TongCos 使用对象组 ID 来隔离名字和复制对象。组 ID 是实现对象与名字的松散绑定的关键。客户方的对象存根使用了一种改进的可扩展 Smart Stub 技术，能够使对象组 ID 透明地在多个复制对象之间切换对象引用，实现对象级的容错。

3.2.1 基于 FlyWeight 模式的对象绑定机制 要支持基于组的对象复制，名字服务就必须做到正确地标识所引用对象组并且能有效地管理对象组。然而由于可靠系统中复制对象的数量通常是巨大的，基于紧密绑定机制的名字系统需要为每个单独的对象使用空间来存放整个对象组的标识信

息，往往难以有效管理对象组。

为了有效地管理对象组，TongCos 使用了 Flyweight[7]模式。该模式提供了管理大量对象外部共享状态的方式。TongCos 系统中每个 RMI-IIOP 复制对象的 ID 是由底层的 POA 分配的，其 JAVA 类型信息是独立于各个实例的外部共享状态。因此，采用 JAVA 对象的类型信息作为对象组的 FlyWeight 对象是可行的。

图4示出了 TongCos 用于支持对象复制的绑定机制。

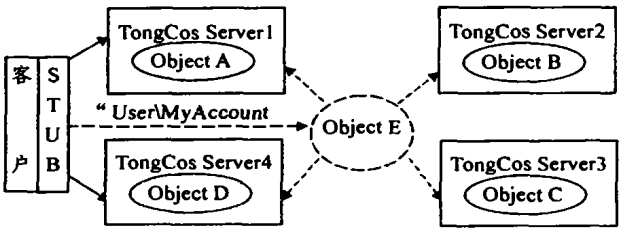

图4 TongCos 的对象绑定机制

图4中，A、B、C、D 是四个位于不同的主机上的四个复制对象，它们分别以相同的名字"User\MyAccount"绑定在四个 TongCos 名字服务器上。TongCos 为它们配了相同的组 ID "Test. Relication. test"。从客户应用的角度看，这四个对象代表着一个逻辑上的单一对象 E。运行时，客户程序地址空间可以不固定地获得该组对象中任意一个的物理地址（图中的实线箭头表示）。而客户应用始终只感觉到在使用标识符为组 ID 的对象 E，物理地址的变化对客户应用来说是完全透明的。通过这种松散绑定机制，TongCos 名字服务为对象复制提供了保障。

3.2.2 可扩展的 Smart Stub 机制实现对象的失效恢复

在众多可靠系统中，对象的失效恢复是通过客户端的 Smart Stub 机制实现的，如 Weblogic[8]的 Smart Stub，但是这种机制要求客户方存根在编译或者部署时刻具有复制对象组的全局知识，系统扩展性较差。

TongCos 提出了一种可扩展的 Smart Stub 方法，它使用反向查询的机制来进行透明的对象地址切换，无需了解对象组的分布。图5描述了这种改进 Smart Stub 的机理。

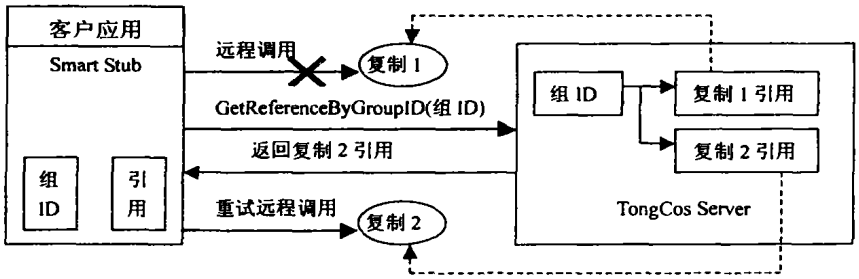

图5 可扩展 Smart Stub 的工作原理

图5中，客户向 TongCos 服务器查询对象时，将同时得到一个组 ID 和组中复制对象1的引用。客户方的 Smart Stub 使用得到的对象地址进行远程调用。当某一次调用失效时，Smart Stub 将拦截 Comm_Exception 异常，并以先前获得的组ID向TongCos名字服务器发起一次getReferenceByGroupID 调用，向 TongCos 服务器查询新的对象引用。TongCos 服务器将根据该组 ID 在其 Flyweight 对象的映射表中查询并返回可用复制对象2的引用。这样 Smart Stub 就可重新发起调用。

可以看出，这种改进的 Smart Stub 机制不需要依赖静态的对象组信息，而是通过名字服务器的反向查询来切换复制对象。即使注册在服务器中的对象组规模发生动态的变化，也可保证能被客户访问，提高了系统的可扩展性。

4 相关工作的比较

Iona 的 OrbixNames[9]通过对象的复制，提供负载均衡服务。通过 OrbixNames，发往同一个名字标识对象的请求可以根据负载的分布被派发到多个复制对象上去，系统的性能得到了的提高。然而 OrbixNames 的复制结构对服务器端的应用对象并不透明——应用需要显式创建管理复制对象组，应用逻辑复杂，可移植性受到削弱。而在名字服务器层次，OrbixNames 并不支持组的管理。因此，其名字服务器依然是系统的单失效点。

BEA 的 Weblogic 支持名字服务器和对象两个级别的可靠复制。WebLogic 的 JNDI 机制通过 IP Multicast 来实现服务集群。这种机制受到硬件的限制，难以应用在不支持多播的网络。此外，Weblogic 使用客户方的 Smart Proxy 技术，要依赖部署时刻的信息来获得对象组的配置信息支持组中的复制对象，当远程调用失效时，Stub 根据配置信息获得新的复制对象的网络端点地址，然后切换复制对象，尝试新的调用。这种机制对对象组的动态变化支持有限，扩展性较差。

FilterFresh[10]是 Baratloo 等人提出的一个容错的 RMI Registry。它采用了类似于 TongCos 名字服务的反向查询机制来提高可扩展性。然而，FilterFresh 使用的反向查询参数是失效对象的过时引用（stale reference）。这就要求名字服务器必须保存系统中大量失效对象的信息。因此，系统空间浪费较大，且保持组成员之间名字信息一致的复杂程度也增大。

结论 具有容错能力的名字服务器在构建基于 Web 的可靠系统中有着重要的意义，尤其对关键业务应用方面更为突出。TongCos 名字服务器在服务器层次上使用 VSC 算法保证了名字信息的强一致，同时在对象层次上实现了透明的对象复制，是构建可靠系统的有力工具。通过比较可以看出，TongCos 不受网络硬件的限制，在复制层次的全面性和系统的可扩展性上均有突出的优势。

我们使用 Web 应用测试工具 WAS 对 TongCos 进行了实际运行的并发客户压力测试，并模拟了服务器失效的场景，取得了令人满意的效果。

参考文献

1 Java™2 Platform Enterprise Edition Specification Version 1.3. Sun Microsystems，2000

（下转第137页）

4. 实验和结果

使用动态允许概率决定算法，每个微时隙都可以更新允许概率 p 的大小。但是该模型的模拟实验耗费的时间很大，不容易得到包丢失率、带宽利用率等结果。我们采用另一种方法。在第2节中得到结论，最优的允许概率应该等于实际竞争状态的终端数目的倒数。所以将算法给出的允许概率和此最优允许概率相比较。设系统中 $F=120, N=150, k=6, 1/\alpha=0.015$，$1/\beta=0.01$。此系统模拟了8M 的信道。终端状态的变化频率是正常的150倍(这是为了模拟大量终端的效果)。结果可见图1。X 轴坐标表示微时隙序列，Y 轴表示允许概率。图中的实线表示动态决定算法计算的允许概率，虚线表示最佳允许概率。可以看出，利用此模型得到的允许概率和最佳允许概率极为接近。并且在一帧中，这种差距会越来越小。

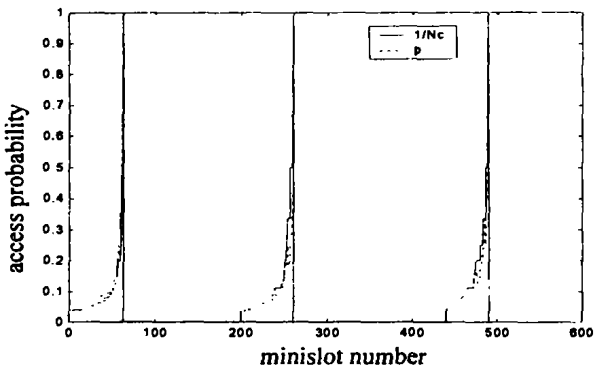

图1　正常情况下的动态允许概率和最优允许概率

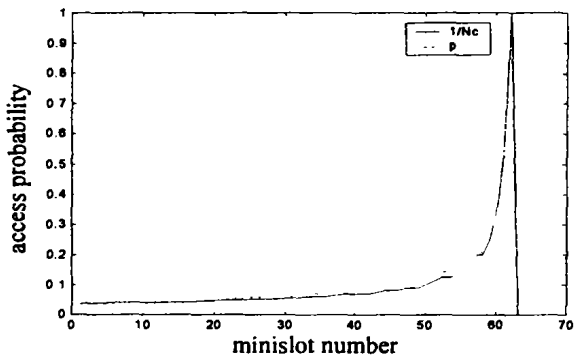

图2　错误分布下的动态允许概率和最优允许概率

同时我们发现，即使某帧开始时刻的竞争态终端数目的分布是错误的，在经过部分微时隙后，计算的允许概率也会同样接近最优允许概率。若在帧开始时，终端数目的分布为平均分布，则结果可见图2。

结论　本文提出的允许概率动态决定算法，进一步优化了 p-persistent 算法。为了实现该算法，我们还建立了一个简单模型，并给出了该模型下的协议。该算法有着广泛的应用，很多使用了分组预约技术和 p-persistent 算法的协议中都可以利用这种算法。虽然在我们的模型中只有语音传输，但在实际应用中往往采用一些其他技术集成数据或其他多媒体数据的传输。通过实验得出，此算法能使得系统允许概率接近最优允许概率。因为在最优允许概率下，每个微时隙竞争成功的极限值为1/e，所以算法能够很好地适应各种各样的系统参数。

参 考 文 献

1 Goodman D J, et al. Packet reservation multiple access for local wireless communications. IEEE Trans. on Commun., 1989, 37: 885～890

2 Wong W, Goodman D J. Integrated data and speech transmission using packet reservation multiple access. In: Proc. ICC'93, Geneva, Switzerland, 1993. 172～176

3 Bianchi G, et al. C-PRMA: the centralized packet reservation multiple access for local wireless communications. In: Proc. Globecom'94, San Francisco, USA, Nov. 1994. 1340～1346

4 Qiu X, Li V O K. Dynamic reservation multiple access (DRMA): A new multiple access scheme for personal communication system. ACM/Baltzer Wireless Networks, 1996, 2: 117～128

5 Khan F, Zeghlache D. Analysis of aggressive reservation multiple access scheme for wireless PCS. In: Proc. ICC'96, Dallas, TX, June 1996. 1750～1755

6 Kim J G, Widjaja I. PRMA/DA: A new media access control protocol for wireless ATM. In: Proc. ICC'96, Dallas, TX, June 1996. 240～244

7 Chan H C B, Leung V C M. Dynamic reservation integrated services multiple access for wireless ATM. In: Proc. IEEE ICUPC'98, Florence, Italy, Oct. 1998. 841～846

(上接第134页)

2 Maffeis S. A Fault-Tolerant CORBA Name Server. In: Proc. of the 15th IEEE Symposium on Reliable Distributed Systems. IEEE Press, Niagara-on-the-lake, Canada, Oct. 1996

3 Birman K P, Joseph T A. Exploiting virtual synchrony in distributed systems. In: Proc. of the ACM Symposium on Operating System Principles, 1987. 123～138

4 van steen M, Hauck F J, Homburg P, Tanenbaum A S. Locating Objects in Wide-Area Systems. IEEE Communications Magazine, Jan. 1998. 104～109

5 Birman K, Joseph T. Reliable Communication in the presence of failures. ACM Trans on Computer Systems, 1987. 47～76

6 史殿习，吴泉源，王怀明，邹鹏. 组通信中虚拟同步协议的研究与设计. 计算机研究与发展，2000, 37(10): 1192～1196

7 Gamma E, Hekm R, Johnson R, Vlissides J. Design patterns Elements of Resuable Object-Oriented Software. Addison Welse, 1995

8 Programming with Webogic JNDI. http://edocs.bea.com/wls/docs60/jndi/jndi.html

9 OrbixNames Programmer's and Administrator's Guide. http://www.iona.com

10 Baratloo A, et al. Filterfresh: Hot Replication of Java RMI Server Objects. In: 4th USENIX Conf. on Object-Oriented Technologies and Systems (COOTS'98)