计算机科学 2002Vol. 29№. 9

分布式系统中的全局状态算法及实现*)

An Algorithm of Global State in a Distributed System

陈 宁 王忠仁 丁香荣

(电子科技大学计算机学院 成都 610054)

Abstract Global state, an important concept in a distributed system, helps to solve problems such as cocurrency, mutual exclusion and deadlock, etc. in a distributed system. This paper, after attempting to present readers the importance and necessity of using global state in a distributed system, gives an in-depth discussion of an algorithm.

Keywords Global state, Distributed snapshot, Middleware, Stable property

1. 引言

全局状态是分布式系统中的一个关键概念,分布式系统同样需要处理在一个集中式环境中出现的如并发、互斥、死锁等问题,在分布式系统中对这些问题的处理变得更为复杂。比如在一个交易中,由于缺少全局状态信息,对可能出现的失败提供一个回滚所需的检查点(checkpoint)就变得很困难。本文将对使用全局状态的必要性及其在基于消息中间件中使用的算法进行讨论。

协同工作的进程间往往需要知道对方的状态,在本地系统中,一个进程可以通过访问对方的进程控制块而获取其现在的状态信息。而对于一个远程进程,由于网络的延误,它获得的状态信息往往是远程进程在过去某个时刻的状态。这个时间上的延迟再加上各系统的时钟不同步使得进程间的协同工作变得更为复杂,可用下面简单示例来说明其复杂性。

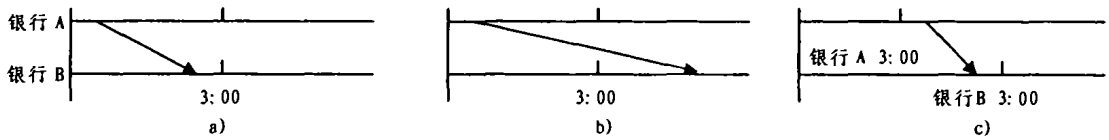

图 1 两个银行间转帐示意图

假设某客户在两家银行开有帐户,并在银行 A 存有 100 元,现他将此存款转入银行 B 去,转帐过程发生在下午 3:00 前后,由于银行间有一段距离,转帐过程导致一定的时延。若我们需要在下午 3:00 统计银行帐户余额,可能会出现图 1 所示几种情况。

图中水平直线代表了一个与时间相关的进程,带有箭头的直线表示两个进程间的消息传递,这里表示转帐的消息传递。

图 1a)表示在 3:00 以前银行间的转帐过程已经结束,在 3:00 时进行统计,某客户在银行 A 已无存款,而在银行 B 有 100 元存款;图 1b)表示,在 3:00 钟进行统计时,转帐过程还未结束,即银行 A 已转出而银行 B 还未收到,统计的结果是该客户在两银行的存款都为零;图 1c)表示两家银行系统时间不同步(银行 A 的时钟比银行 B 快)银行 A 在它的系统时间 3:00 刚过向银行 B 转帐,而银行 B 收到转帐时,他的系统时间还不到 3:00,若两银行在各自的系统时间 3:00 进行统计,在两银行上都有 100 元存款。显而易见,图 1a)是正确的,而后两者都是错误的。

2. 分布式系统中的全局状态算法

通过上例可以看出,在一个分布式系统中,对所有的结点发送和接收的消息进行全面的观察,了解全局状态才能避免错误出现。这里,为了更深入讨论上述问题,我们引入一种通过分布式快照获取全局状态的算法:全局状态构造算法 GSRA (Global State Recording Algorithm)。使用这个算法,一个分布式系统可以获得一组分布式快照,并从中确定当前分布式系统所处的全局状态,检测当前系统是否拥有某种稳定的特征(stable property),因而能使系统在交易过程中提供检查点以支持在交易失败时进行回滚操作。首先,需要对以下几个概念进行定义:

• 通道(channel):两个进程间进行信息交换而建立起来的路径或渠道,称之为通道。为简便起见,我们假设通道是单向的,若要双向通讯,可建立两个通道。

• 状态(state):状态是指与进程相连的通道上接收到和发送了的消息序列的现状。

• 全局状态(global state):全局状态是该系统中所有进程状态的总和。

• 标记(Marker):标记是一种特殊消息,用它来表示系统正在进行快照。系统中所有进程都要将标记从它连接的所有通道中通过一次且仅一次。

• 稳定特性(Stable Property):用严格的数学语言来表示是:令 S 为分布式系统 D 的一个全局状态。令 Y 为 S 的一

*)其研究结果已应用于东方通科技公司的中间件"TongLINK/Q"产品中。陈 宁 硕士研究生,主要研究方向为计算机系统结构、网络应用技术、分布式系统等。王忠仁 教授,主要研究方向为计算机系统结构与分布式系统等。丁香荣 东方通科技公司技术总监,主要研究方向为网络应用技术与分布式系统等。

种特性:Y(S),则称 Y 为 D 的一种稳定特性当且仅当 Y(S)→Y(S'),其中,S' 是所有从 S 能够到达的任何全局状态。可以简单理解为一个分布式系统一旦处于一个稳定的状态并可一直保持稳定,说明该系统具有稳定特性。

· 快照(Snapshot):对进程而言快照是进程某一个时刻的状态。每一个快照都包含了从上一次进行快照以来该进程所接收到和发送出去的消息序列的所有记录;对系统而言快照是一个全局状态。这个全局状态是通过一定时间对系统进行记录而获得的,因此它可能并不是系统当前所存在的真正状态。事实上,它仅仅是系统可能存在的一种状态;系统从快照开始时可能进入过这种状态,也可能没有并且永远不会进入这样一个状态。

· 分布式快照(Distributed Snapshot):系统中所有进程快照的集合。

在讨论 GSRA 算法前我们还得对系统做如下假设:①进程数目和存在的通道数目有限;②作为通讯使用的通道拥有无限的无错缓冲;③在通道上被接收的消息顺序与其被发送时的顺序应一致;④系统中参与 GSRA 的进程没有公用内存和时钟。

在描述 GSRA 算法时使用如下约定:

P:进程,它可以是消息的发送进程,也可以是消息的接收进程。

C:通道,它可以是连接进程的发送通道,也可以是连接进程的接收通道。

S:状态,进程的状态除了它自身的信息外还包括发送或接收的消息。

M:标记,其含义如前对标记的定义。

GSRA 算法由系统中需要获得全局状态的某一个进程首先启动,这个启动进程被称为发送者,其它所有响应发送者的进程都被称为接收者(启动者自己也可以是接收者)。首先,发送者对每一个与之相连接的发送通道送出一个标记,接收到标记的接收者同样对每一个与它相连接的发送通道送出一个标记,要保证每一个通道有且只有一个标记通过。发送者启动后就对每一个与之连接的接收通道上接收到的消息进行记录,直到它在所有的接收通道上都收到一个标记,它也就完成了它自己对系统进行的快照。GSRA 算法描述如下:

发送者(进程 P1):

(1)记录 P1 的状态 S1;

(2)对 P1 相连接的发送通道 C1,在发送其它任何消息之前先发送一个标记 M。

接收者(进程 P2 在通道 C1 上接收到一个标记 M):

如果 P2 尚未记录自己的状态,即以前未接收到标记 M,则:

(1)记录 P2 的状态 S2;

(2)将 C2 的状态置为空;

(3)对 P2 相连接的发送通道 C2,在发送其它任何消息之前先发送一个标记 M。

如果 P2 已经记录了自己的状态,即以前已接收到标记 M,则:

C2 的状态置为自上次 P2 的状态被记录以来收到的所有消息。

需要再次强调的是 GSRA 获得的系统全局状态是个"虚拟状态";真正的系统在记录完成时的真实状态与 GSRA 所记录的状态是有区别的,这是由于 GSRA 使用的标记在传输过程中的延迟和时钟不同步造成的。

还必须用其它算法来确定 GSRA 记录的虚拟状态是否满足一定稳定特性条件。

从效果上来说,GSRA 算法所得到的全局状态同进程在接收到一个标记后停止发送任何其它消息,直到对整个系统状态的记录完成为止所得到的系统状态是等价的。该算法的思想在于它使用一种虚拟的方式将发送者在启动 GSRA 算法时仍然存在于消息通道中的消息"推"(push)到目的进程中,使目的进程在记录自己的状态时能包含这些消息。对于在一个标记被收到后发送出去的消息,GSRA 不做任何处置(它完全忽略这些消息以保证算法能有效、快速地执行)。

GSRA 保证其快照所记录的状态是一个一致的全局状态;也就是说,如果系统当前具有稳定特征,则该特征一定会在这个快照中得以体现。

3. 一个 GSRA 算法示例

作为一个使用 GSRA 的简化示例,考虑下面的情况:有三个进程 P1、P2、P3,刚开始时,每个进程的状态为拥有 $500,进程间所有通道均为空。这个系统中的一个稳定特征是它们总共拥有数额为 $1500。这个系统的初始状态如图 2a)所示,此时三个进程记录如表 1a)所示(表 1 中,空的单元表示尚未记录或不用记录,NULL 表示将相关状态置空,下同)。

图 2b)表示 P1 在发送 $10 到 P2 后决定启动 GSRA,它首先记录自己当前的状态(拥有 $490),然后在发送通道 C1 上发送一个标记 M。在 P1 启动 GSRA 的同时,P2 通过通道 C2 向 P1 发送了 $20,通过通道 C3 向 P3 发送了 $10。系统状态见表 1b)。

图 2c)中,当进程 P2 接收到来自 P1 的 $10 时,它将自己的状态改变为拥有 $480,然后它将从 C1 上又接收到标记 M。按照 GSRA 算法,此时它将记录自己的状态(拥有 $480),然后在所有的发送通道 C2 和 C3 上发出标记。与此同时,我们假设 P3 通过通道 C4 向 P1 送出 $25(P3 发送这个消息是在 P2 接收到标记之前或之后都无关紧要)。系统的当时状态见表 1c)。

图 2d)表示,当 P3 接收到来自 P2 的 $10,然后它又收到从 P2 送来的标记。由于 P3 是第一次收到标记,它将记录自己的状态(拥有 $485),并将同它相连的接收通道状态置空。然后它将在自己的发送通道上送出标记。与此同时,我们假设 P1 在通道 C1 上向进程 P2 送出 $20。注意,按 GSRA 算法的规定,通道 C1 上已经接收过标记,P2 在接收到从 P1 送来的这个消息将不会被记入它的状态中,系统状态见表 1d)。

接下来的过程中,P2 收到来自 P1 的 $20,它将更新自己的状态为 $500。注意,P2 更新的是它的实际状态,而对已经记录的状态而言,这个值仍然是 $480;按照 GSRA 算法,P2 新接收到的 $20 不能被加到被记录的状态中去。

与此同时,P1 从通道 C2 上接收到来自 P2 的 $20,由于在 C2 上已接收到标记,这个 $20 将被 GSRA 算法记录到 P1 中通道 C2 的状态中(而不是 P1 的状态中。按照 GSRA 算法,在一个进程发送了标记后所有收到的消息应该记录到其相连的通道状态上,而不是进程的状态中)。

P1 接收完来自 P2 的 $20 后,将从 C2 上接收到由 P2 所发送的标记。这表明通道 2 中的所有消息已经被"推"到进程

P1，按 GSRA 算法，P1 将停止记录任何从 C2 上接收到的消息到 C2 的状态中。我们用图 2e)和表 1e)来表示。

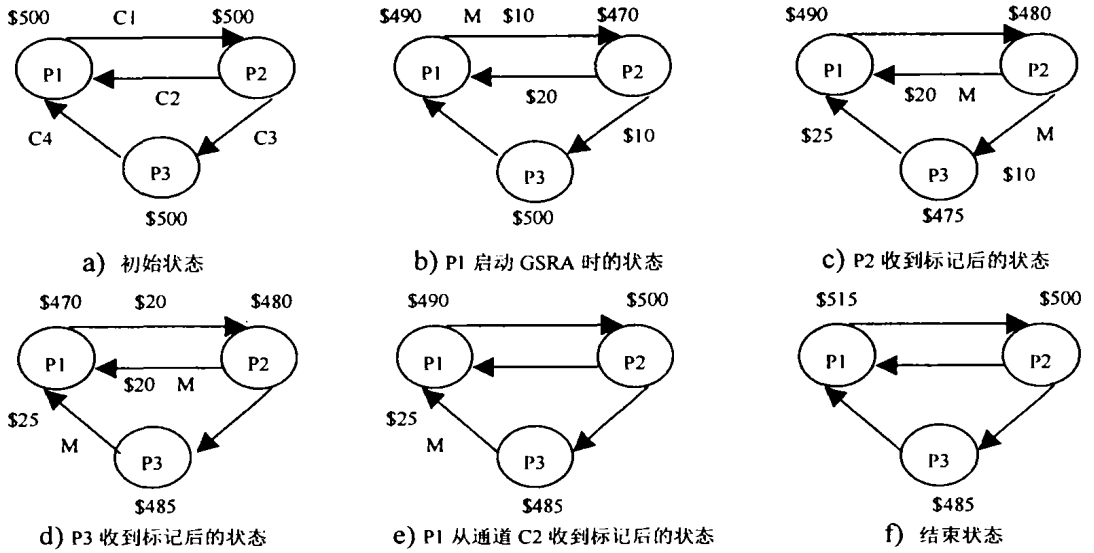

图 2 一个 GSRA 算法示例

表 1 GSRA 示例

表 1a) 初始状态

进程	状态	C1	C2	C3	C4
P1	500	NULL	NULL	NULL	NULL
P2	500				
P3	500				

表 1b) P1 启动 GSRA 时的状态

进程	状态	C1	C2	C3	C4
P1	490		NULL		NULL
P2					
P3					

表 1c) P2 收到标记后的状态

进程	状态	C1	C2	C3	C4
P1	490		NULL		NULL
P2	480	NULL			
P3					

表 1d) P3 收到标记后的状态

进程	状态	C1	C2	C3	C4
P1	490		NULL		NULL
P2	480	NULL			
P3	485			NULL	

表 1e) P1 从通道 C2 收到标记后的状态

进程	状态	C1	C2	C3	C4
P1	490		NULL		NULL
P2	480	NULL			
P3	485			NULL	

表 1f) 结束状态

进程	状态	C1	C2	C3	C4
P1	490		{$20}		{$25}
P2	480	NULL			
P3	485			NULL	

在最后一步中 P1 从通道 C4 上接收到来自 P3 的 $25，同上，这个 $25 将被记录到 P1 中通道 C4 的状态中，同时，P1 将自己的状态更新为 $505(这个更新不会更改 P1 所记录的自己的状态)。当 P1 接收到来自 P3 的标记后，P1 将停止记录任何从 C4 收到的消息到 C4 的状态中。由于 P1 所含的所有接收通道都收到了标记，整个 GSRA 算法结束，见图 2f)和表 1f)。

表中所显示的记录就是 P1 发起运行 GSRA 算法所得到的整个系统的快照；从这个快照中我们可以看到如下几点：

(1)稳定特征存在于该快照中：将 P1、P2、P3 的状态和通道 C1、C2、C3、C4 的状态相加，我们得到 $1500。这同系统最初时的总数一致。

(2)快照所反映的系统状态并不是算法完成时系统所具有的状态(从最后的状态示意图中可以看出)。这也说明了 GSRA 算法所得到的全局状态只是一个"虚拟"全局状态，而不是一个真正的现实状态。

GSRA 算法并不关心如何利用所得到全局状态，它只负责获得系统的一个快照，至于如何利用所获得的快照，则由系统其它算法来决定。

4. GSRA 算法的实现

TongLINK/Q(东方通科技公司的中间件产品名称)中，为确保一个交易的正确实施，当交易失败后能安全地回滚。TongLINK/Q 的事件管理子系统(Even Management Subsystem，EMS)负责建立一个系统全局状态的快照。

由于消息中间件不存在一种可以实施的同一的规范，对消息中间件的设计和实现随不同的中间件提供商将有所不同。在实现 TongLINK/Q EMS 过程中，没有可遵循的设计和实现规范，TongLINK/Q EMS 只是在参照分布式事件规范、Java 消息规范中的订阅/发布模型后针对 TongLINK/Q 自己应用的需要而设计的。此外 TongLINK/Q 的事件管理子系统还参照 GSRA 算法实现了对全局状态的记录，以便系统和应用在需要时使用这种被记录的全局状态。

TongLINK/Q 的 EMS 工作流程大致如图 3 所示，从图中可见，EMS 要负责完成的功能有：

接收本地对事件的订阅；接收本地事件生产者发布的事件；接收远地 EMS 的事件通知；对事件进行存储(如死信事

件等等)；对本地应用进行事件通知；转发事件通知到远地应用(通过远地 EMS)等。

EMS 实现了分布式事件规范中的所谓三方对象的功能。EMS 通过制定同事件相关的策略(policy)来实现对事件的过滤。此外，由于它使用了事件队列这个概念，它拥有了对事件的有限存储功能，因此 EMS 实现了分布式事件规范中的存储-转递代理、事件过滤器、事件邮箱等第三方对象所需要的功能。

TongLINK/Q EMS 的模块结构包括：事件队列；EMS 核心程序；事件生产者；事件消费者(订阅者)。

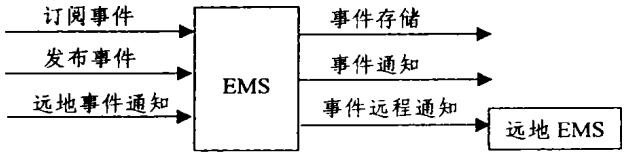

图 3　TongLINK/Q 事件子系统工作流程

TongLINK/Q EMS 将事件生产者分为两大类，TongLINK/Q 系统自身，建立在 TongLINK/Q 之上的其它程序。为此，TongLINK/Q 使用了一个生产者对象对事件生产者进行描述：

```
struct tlqProducerObject{
    short obType; //生产者类型，是核心生产者或应用程序?
    char producerName[NAMELEN]; //生产者标示
    //其它相关属性
};
```

上述结构中，生产者标识用来建立一个同生产者表的映射。生产者生产事件时负责将自己的标识递交给 EMS 核心，EMS 核心将通过这个标识查找生产者表确定与该生产者相关的属性，比如确定是否允许该生产者生产指定类型的事件。

类似地，TongLINK/Q EMS 将事件主要分为两类：系统事件、用户事件。TongLINK/Q EMS 的系统事件是指那些由系统产生的标志系统状态发生某种变化的事件。一般而言，系统事件只能由核心生产者产生。用户事件则是用户应用程序之间为了通知对方状态变化而发布的事件。

TongLINK/Q EMS 为每一种事件指定了一个事件类别；为了方便扩展，TongLINK/Q EMS 将类别为 0-255 的保留为系统事件标示；256 以上的是用户自己发布的事件类型。同事件相关的对象如下：

```
struct tlqEventHead
{
  int eventID; //128 位
  short eventType; //事件类型，0-255 系统保留
  struct tlqProducerObject *producer; //生产者
  time_t stamp; //生产事件邮戳
}
struct tlqEventObject
{
  struct tlqEventHead evHead;
  int nLen; //事件附带信息长度
  char evData[1]; //事件附带信息
}
```

TongLINK/Q EMS 将生成的事件对象加入到事件列表中；事件列表表项中包含了事件的应用记数(有多少订阅者订阅了这个事件)、事件的生存周期(事件能够在事件列表中存在的时间长度)、与事件相关的策略等等信息。事件列表的表项结构如下：

```
struct tlqEventEntry
{
  int refCount; //订阅者数目
  int timeToLive; //生命周期
  //与事件相关的策略
  struct tlqEventObject *event;
  //其它相关信息
}
```

为了提高效率，事件列表是由上述表项组成的一个数组。EMS 核心会定期对这个列表进行检查，将那些超过生命周期仍未被接收的事件清除(一些重要的事件，比如死信事件，可能被写入磁盘)。

系统维护一个订阅者表和一个生产者表，订阅者表的作用是记录对事件进行订阅的订阅者信息；这些信息可能包括订阅者类型、订阅者对事件的请求方式等等。EMS 认为有两种订阅者：本地订阅者和远地订阅者，对本地订阅者而言，EMS 允许的订阅方式有：通知模式(push)和查阅模式(pull)。对通知模式的订阅者，它在订阅事件时将提供对它通知的手段(比如对 WINDOWS 而言，这可能是一个消息 ID 和窗口句柄对，对 UNIX 而言则可能是一个管道或消息队列标识)。

应用程序提出订阅请求，它传递给 EMS 核心的信息包括：事件名称、订阅方式、自己的相关信息(包括通知方式，如果它采用通知模式的话)。

EMS 核心收到订阅请求时，它需要对事件表进行检查，确保应用程序订阅的该类事件存在(事件表不同于事件列表，前者用以标示系统所目前所能提供的事件类型，后者则为系统目前已经收到的事件)；如果事件不存在，EMS 将返回失败。

EMS 确认事件存在后，还要检查对该事件是否应用了某种策略；如果该事件存在某种策略(比如只允许具有某种特征的应用程序订阅该事件；同事件相关的策略是在事件发布者注册事件时确定)，它需要对订阅者进行检查，确保订阅者能满足这些策略，否则它会返回失败。在通过检查后，EMS 核心将创建一个订阅者表项，将它初始化为应用程序提供的信息，然后返回成功。

与订阅过程相关的是事件的发布过程，应用程序在发布事件时，先向本地 TongLINK/Q EMS 提出请求；EMS 接收到应用发布事件请求时，首先对事件表和生产者表进行检查，确保发布的事件是有效的；如果事件有相关的策略，这意味着 EMS 还要对发布事件的应用程序进行认证，确保它符合与事件相连的策略。

接下来 EMS 将通过事件表查询阅者表；订阅者表中可以寻找出当前订阅了这类事件的应用程序的信息；EMS 通过它来对订阅了事件的本地或远地应用进行通知；事件表和订阅者表之间的关系可大致描述为：每个事件表中的表项(代表了一类事件)都有一个指针指向订阅了该事件的订阅者列表；对每一类事件都有一个订阅者列表，EMS 通过这个列表获知哪些应用程序订阅了当前事件。

如果 EMS 发现某个订阅者表明它希望以“查阅方式”订阅事件，则 EMS 会在事件列表中申请一个表项，将该事件放入列表中，并根据订阅者提供的信息或根据系统指定的缺省值为事件指定一个生命周期。当事件列表中的事件生命周期变为 0 或其引用记数为 0 时，这个事件将被 EMS 从事件列表中删除。

对一个远地订阅者，本地 EMS 将直接对远地订阅者所在的 EMS 进行通知，再由远地 EMS 通知远地的订阅者。

由于 TongLINK/Q EMS 允许使用应用定义的事件，因此 TongLINK/Q EMS 需要提供一种机制允许应用注册事件和注销事件。这个过程类似于 WINDOWS 的用户注册消息过

(下转第 35 页)

4 算法与实验结果分析

根据分类超曲面的思想，我们给出以上算法实现过程，当同类样本点在有限个连通分支分布时，学习算法与分类算法的算法复杂度都是多项式的。通过实验我们可以看到基于分类超曲面的海量数据分类法，对解决非线性分类问题是有效的，具有较好的通用性，从实验结果看分类准确率高，特别是查全测试准确率高。同时，采用小规模样本构造超曲面，对大规模样本进行分类的结果表明，基于分类超曲面的直接分类方法有很好的推广能力。对噪声的干扰此方法虽然不能消除，但可以把噪声控制在局部范围。事实上，这种分类方法能有效地解决在有限区域分布的海量数据的非线性分类问题，而实际数据往往具备在有限个连通分支分布的特征。

结论 本文基于 Jordan 曲线定理，提出了一种通用的基于超曲面的直接分类方法，并由此提出了分类超曲面的思想。通过实验，可以证明采用基于分类超曲面的分类方法，对非线性数据进行分类是完全可行的，而且在处理大规模样本数据（10^7）时，分类速度和正确率都可以得到保证。另外，无须考虑矩阵的复杂计算，因而可以大大节省计算资源，有效提高了分类效率。

应当指出，本文所讨论方法是对直接解决非线性分类问题的一种尝试，此方法的一个前提是样本点的分布具有保证数据集在有限个连通分支分布的特征。因此，这种方法在处理如此分布的海量数据集时，有较好的效果。

参 考 文 献

1 Vapnik V N. Support Vector Method for Function Approximation, Regression Estimation and Signal Processing. Neural Information Processing Systems, Vol. 9. MIT Press, Cambridge, MA

2 Vapnik V N. The Nature of Statistical Learning Theory. New York: Springer-Verlag, 1995

3 Zhang Ling, Zhang Bo. A Geometrical Representation of McCulloch-Pitts Neural Model and Its Applications. IEEE Trans. on Neural Networks, 1999, 10(4): 925～929

4 张学工. 关于统计学习理论与支持向量机. 自动化学报, 2000, 26(1)

5 张学工译. 统计学习理论的本质. 北京：清华大学出版社, 2000

6 张文生, 丁辉, 王珏. 基于邻域原理计算海量数据支持向量的研究. 软件学报, 2001, 12(5)

7 边肇棋等. 模式识别(第二版). 北京：清华大学出版社, 2000

8 Vapnik V N. Statistical Learning Theory. J. Wiley, New York, 1998

9 Widrow B, Winter R G. Layered neural nets for pattern recognition. IEEE Trans. on Acoustics, Speech and Signal Processing, 1988, 36(3): 1109～1118

10 Burges C J C. A tutorial on support vector machines for pattern recognition. Data Mining and Knowledge Discovery, 1998, 2(2)

11 Widrow B, Hoff M. IRE Wescon Convension Record. Part 4, Institute of Radio Eng., New York, 1960. 96～104

12 Fulton W. Algebraic Topology A First Course, Springer-Verlag 1995

（上接第 29 页）

程。应用程序通过调用 tlqRegisterEvent()API 提出注册请求，EMS 核心会自动负责为新注册的事件分配一个类型号（事件类型）；随后的应用可以通过这个类型号或事件名称（在注册事件时提供）对事件进行发布和订阅。

对事件的注销是一个相反的过程。事件被注销后，EMS 会试图通知所有订阅了该事件的应用程序，然后清除同该类事件相关的资源。

EMS 实现了前述 GSRA 算法，EMS 将 GSRA 算法中的标记作为一个特殊的系统事件，当某个 EMS 按系统请求需要对系统状态进行快照时（比如一个交易的开始），它将启动 GSRA 算法。EMS 负责对这个特殊事件的响应。系统中任何一个 EMS 接收到 GSRA 标记时，将按照 GSRA 算法记录自己的状态，然后发送标记到其它相关节点。需要注意的是在两个 EMS 系统之间存在通讯通道是双向的，因此它实际代表了 GSRA 中的两个通道。本文将不对 TongLINK/Q EMS 具体实现细节进行详细讨论，结果表明，由于 TongLINK/Q EMS 采用 GSRA 算法，系统开销小，灵活方便。由于 TongLINK/Q EMS 只是在参照分布式事件规范、Java 消息规范中的订阅/发布模型后针对自身需要而设计的，严格来讲 TongLINK/Q EMS 还不是一个完全的分布式系统，也不是完全满足 GSRA 算法的理想条件，例如，在 TongLINK/Q 中，作为通讯使用的通道实际上容量有限，也不能保证无错缓冲，因此，在采用 GSRA 算法时还有一定限制。

参 考 文 献

1 Bernstein P A. Middleware: A Model for Distributed Services. Communications of the ACM, 1996, 39(2): 86～97

2 Stephen Williams Global State Recording Algorithm: GSRA from http: // courses. cs. vt. edu/～cs5204/fall00/Summaries/GlobalState/global_state. html

3 Eckerson W W. Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications. Open Information Systems, 1995, 3(20)

4 Richard S. Middleware Demystified. Datamation, 1995, 41(6): 41～45

5 Distributed Event Specification, Revision 1. 0 Beta. Released on: July 17, 1998, Copyright 1998 Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, CA 94303, U. S. A

6 Java™ Message Service Specification ("Specification"), Version: 1. 0. 2. Released on: 12/17/99, Copyright 1999 Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, CA 94303, U. S. A

7 RFC 1862. Report of the IAB Workshop on Internet Information Infrastructure. Oct. 12-14, 1994

8 RFC 2543. SIP: Session Initiation Protocol

9 RFC 2768. Network Policy and Services: A Report of a Workshop on Middleware

10 RFC 2969. Wide Area Directory Deployment - Experiences from TISDAG

11 TUXEDO System 6 Course for Application Developers. BEA Systems, Inc., Dec. 1996

12 MQSeries Family June Product Announcement White Paper. IBM Inc, June 1999

13 Stevens W R. Advanced Programming in the UNIX Environment. 机械工业出版社, 1999

14 东方通科技公司. TongLINK/Q 技术白皮书. 1999