二、远程过程调用(RPC)

前面，我们学习了通过客户／服务器结构来设计分布式操作系统中进程间的通信，通过寻址、阻塞／非阻塞、缓冲／非缓冲以及可靠与非可靠等原语来实现分布式系统中进程间的通信。

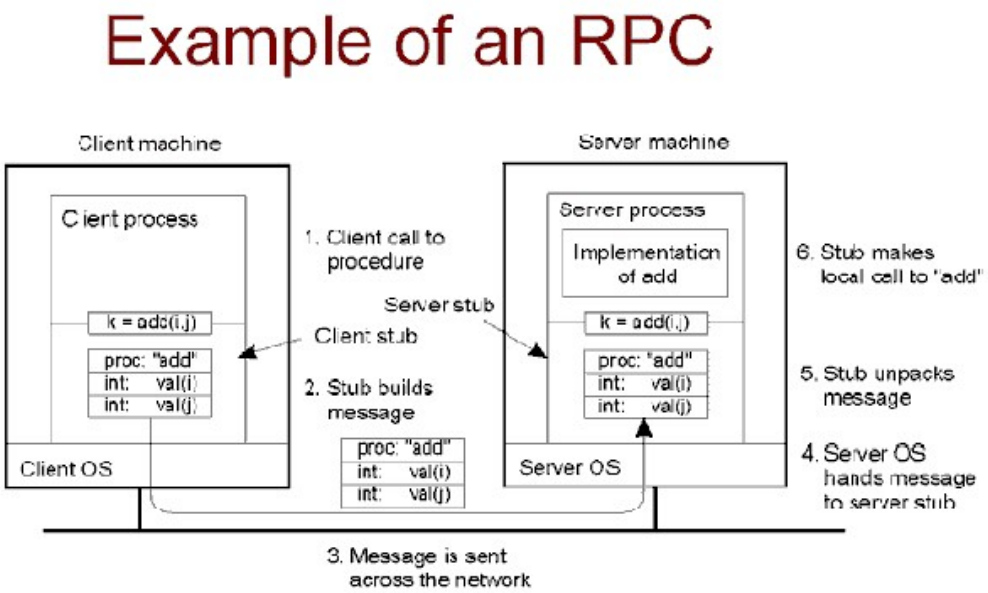
分布式系统的主要特点是能够将一台机器上的一个任务分解到系统中其他的机器上运行，实现多个 CPU 的协同工作。远程过程调用 RPC 就是实现这一特点的有效方法之一。这一节我们主要介绍

1.什么是 RPC

RPC 的基本思想

(1984 年,Birrell,Nelson)允许程序调用位于其他机器上的过程，当机器 A 上的一个进程调用机器 B 上的过程时，在 A 上调用进程被挂起，在 B 上执行被调用过程，过程所需的参数以消息的形式从调用进程传送到被调用过程，被调用过程处理的结果也以消息的形式返回给调用进程。而对程序员来说，根本没有看出消息传递过程和 I/O 处理过程，这种方式称为远程过程调用。remote procedure call---RPC

RPC 例子



传统的过程调用

count=read(fd,buf,nbytes)

操作系统隐藏了具体的写数据，对程序员来说也看不到这一过程。

参数传递：

按值传送(call-by-value)

按地址传送(call-by-reference)

拷贝／恢复(call-by-copy/restore)

2.RPC 的透明性(transparent)

RPC 透明性

客户－调用进程所在的机器

服务器－被调用过程所在的机器

RPC 透明性的思想使得远程过程调用尽可能象本地调用一样，即调用进程应该不知道被调用过程是在另一台计算机上执行，反过来也是如此，被调用过程也应该不知道是由哪台机器上的进程调用。

RPC 透明性的实现

客户代理(client stub)

将参数封装成消息，

请求内核将消息送给服务器

调用 receive 原语进入阻塞状态

当消息返回到客户时，内核找到客户进程，消息被复制到缓冲区，并且客户进程解除阻塞

客户代理检查消息，从中取出结果，并将结果复制给它的调用进程

服务器代理(server stub)

调用 receive 原语，处于阻塞状态，并等待消息的到来当消息到达后，代理被唤醒

将消息分解，并取出参数

以通常方式调用服务器的过程

调用结束并获得控制后，把结果封装成消息

调用 send 原语发送给客户重新调用 receive 等待下一个消息

RPC 的工作步骤

客户进程以通常方式调用客户代理

客户代理构造一个消息，并自陷进入内核

客户内核发送消息给远程内核(服务器)服务器

内核把消息送给服务器代理服务器代理从消息中分解出参数，度调用服务器过程

服务器过程完成其工作，并返回给代理

服务器代理封装结果，并自陷内核

远程内核发送消息给客户内核

客户内核将消息传给客户代理

代理分解消息取出结果，返回给调用进程。

Steps of a Remote Procedure Call

1. Client procedure calls client stub in normal way
2. Client stub builds message, calls local OS
3. Client's OS sends message to remote OS
4. Remote OS gives message to server stub
5. Server stub unpacks parameters, calls server
6. Server does work, returns result to the stub
7. Server stub packs it in message, calls local OS
8. Server's OS sends message to client's OS
9. Client's OS gives message to client stub
10. Stub unpacks result, returns to client

3.参数传递(Parameter Passing)

客户代理的功能之一是取出参数，将它们封装成消息，然后发送给服务器代理。表面上看，好像很简单，但实现起来并不是如此。下面我们将讨论 RPC 系统中参数传递的有关问题。

参数整理(Parameter Marshaling)

将参数封装成消息的工作

参数传递中存在的问题

Marshalling

- Problem: different machines have different data formats
  - Intel: little endian, SPARC: big endian
- Solution: use a standard representation
  - Example: external data representation (XDR)
- Problem: how do we pass pointers?
  - If it points to a well-defined data structure, pass a copy and the server stub passes a pointer to the local copy
- What about data structures containing pointers?
  - Prohibit
  - Chase pointers over network
- Marshalling: transform parameters/results into a byte stream

系统中不同机器的字符集可能不同

分布式系统中客户机与服务器可以是不同类型的，例如

IBM 工作站　EBCDIC 字符集 IBM PC 机　ASCii 字符集

系统中不同机器的数据存储方式可能不同

Intel CPU 整数从右到左存储(little endian),SPARC CPU 整数从左到右存储(big endian)

浮点数的位数可能不同
布尔值的表示
网络传送按字节

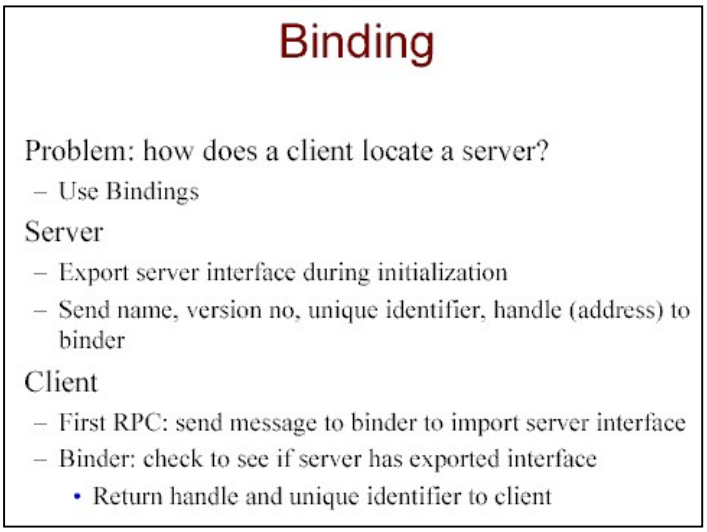
解决方案

设置一个基本类型的标准，正则表(canonical form)，描述字符集类型，数据存储方式及长度(位数)等。过程所需的参数，客户代理在进行参数整理时按 canonical form 转换为标准类型，然后封装成消息发送。服务器代理收到消息后，也根据该标准映射到本地机器的字符集和数据类型。

参数按地址传递的解决
这是一个比较复杂的问题，
方法 1 ：copy/restore 方法
方法 2 ：指出是输入参数(in parameter)还是输出参数(out parameter)　指针参数的解决
动态传送：指针值存入寄存器，通过寄存器间接寻址
用户定义类型的数据传递

4.动态联编(Dynamic Binding)

问题的提出
透明性
可靠性
迁移性
动态联编
联编(binder):
一个程序，功能：
登记(register)
查找(lookup)
撤销(unregister)
服务器：export,启动时 register,unregister
客户：lookup
动态性：服务器启动时，export for register ,关闭时，export for unregister
服务器故障时，定期轮询(客户代理)对无响应的过程 unregister.
联编表(list)



name	version	handle	unique id
read	3.1	1	1
write	3.1	1	2

close	**3.1**	**1**	**3**

动态联编的灵活性(flexible)

均衡工作量(load balancing):支持多服务器(support multiple servers)，把客户均衡地分布于各个服务器上；

容错性(fault tolerance):定期转询服务器(poll the server perodically)，对无响应的过程unregister，到达一定程度的容错性

支持权限:服务器可以指定由哪些用户使用，这样联编对非授权的用户拒绝接受。

动态联编的缺点(disadvantages)

花费系统时间:the extra overhead of exporting and importing interfaces costs time.

客户进程往往执行时间短，但每次每个进程要重新 import to binder

瓶颈(bottleneck):use multiple binders

5.RPC 表示错误的语义(Semantics in the Presence of Failures)

RPC 的设计目标是隐藏通信过程，使得远程过程调用像本地调用一样，但也有一些另外，如不能处理全局变量，事实上，只要客户和服务器能正常工作，RPC 就应该可以正常工作。下面的问题是当客户或服务器出错误时的处理方法。

RPC 可能出现的错误及处理方法

1) 客户不能找到服务器(client cannot locate the server)

客户不能找到合适的服务器，可能原因：服务器可能关闭或服务器软件升级。

这种错误目前尚无好的办法，需要指出，我们不能试图通过返回错误代码来实现，因为代码可能刚好是一个正常的返回值。

从客户到服务器的消息丢失(lost request)

客户发出的请求到达服务器之前丢失，服务器根本不能响应。

解决方法:超时重传机制

2) 从服务器到客户的应答丢失(lost reply)

丢失应答非常难以处理，简明的解决方法是依赖于超时重传机制，发出的请求在一个合理的时间内没有应答，就再发送一个请求。这种方法的问题是，客户内核无法确定为什么没有应答，是否请求或应答丢失？或许只是服务器速度慢？

同一有效性(idempotent)：服务器上有些操作可以安全地重复执行多次，而对数据不影响，如果某一请求的操作具有这一属性，则称为同一有效性(ide 不 mpotent)。多数请求都不具有同一有效性。

解决方案

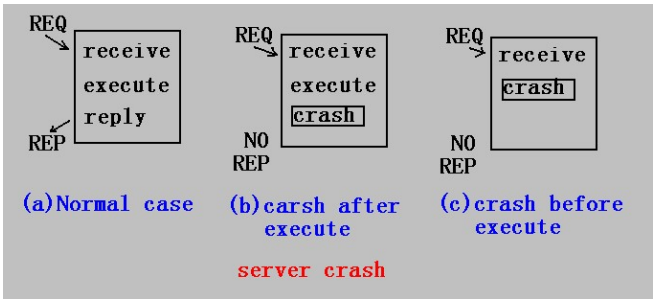
客户内核给每个请求一个序列号，服务器内核则保留每个请求最近接收到的序列号。这样，服务器就可以通过这个序列号来区别一个请求是重发的还是原来的，对重传的请求拒绝响应。另外，也可以在消息中增加一个位来提示该请求消息是原来的还是重发的，对于原请求可以安全地进行处理，对于重发的请求，则处理要十分小心（服务器保留一个副本，是一种有效的办法）。

Failure Semantics

1 *Client unable to locate server*: return error
2 *Lost request messages*: simple timeout mechanisms
3 *Lost replies*: timeout mechanisms
- Make operation idempotent
- Use sequence numbers, mark retransmissions

4 *Server failures:* did failure occur before or after operation?
- At least once semantics (SUNRPC)
- At most once
- No guarantee
- Exactly once: desirable but difficult to achieve

5 *Client failure:* what happens to the server computation?
- Referred to as an *orphan*
- *Extermination*: log at client stub and explicitly kill orphans
  - Overhead of maintaining disk logs
- *Reincarnation*: Divide time into epochs between failures and delete computations from old epochs
- *Gentle reincarnation*: upon a new epoch broadcast, try to locate owner first (delete only if no owner)
- *Expiration*: give each RPC a fixed quantum *T*; explicitly request extensions
  - Periodic checks with client during long computations



3) 服务器接收了请求后崩溃(server crash)

重传机制的语义

至少一次语义(at least once semantice)：客户等等直到服务器重启并再次执行操作：客户继续重试，直到获得一个应答为止。可保证 RPC 至少执行一次。

至多一次语义(at most once semantics)：立即放弃报告失败。保证 RPC 至多执行一次，但可能没有执行。

仅有一次语义(exactly once semantics)：

4) 客户发送请求后崩溃(client crash)

最后,我们介绍客户在发送给服务器请求后而在应答收到之前崩溃的情况.

孤报(orphan)及其存在的问题

孤报(orphan):a computation is active and no parent is waiting for result.Such unwanted computation is call Orphan.孤报(orphan)

存在的问题：

浪费 CPU 时间;

Orphan 可能锁住某些文件或占用有价值的资源;

当客户机重启时,来自 Orphan 的应答造成应答混乱.

孤报(orphan)的解决方法

1981 年 Nelson 提出了四种的解决方法

消灭(extermination)，思想：客户代理在发送 RPC 消息后,代理进行事务登录(log),记录发送的请求,重启时,检查事务登录,Orphan 被撤消。缺点：磁盘空间浪费;orphan 可能引起新的 RPC 导致更多的 orphan,客户重启后,无法找到它们; 网络分区:网络被分成两个独立的部分,客户所在的另一个部分中的 orphan 仍能活动(不是一中可靠的方法)

"再生"(reincarnation)，思想：将时间按顺序分成时间段，每一段的一个序号，当客户重启时，广播一个消息告诉所有的主机一上新的时间段的开始。机器收到这样的广播消息后， 所有的过程计算被撤消。对于已发出的应答，由于消息中含有时间段序号而客户可以很容易地区别它们。缺点：那些有效的计算也被删除

gettlereincarnation(合理再生)，思想：当机器收到新的时间段的广播广消息后，每台机器检查是否有远程计算，如果有则试着找出它们的客户，若能找到客户(主人)，则继续它的计算，否则将远程计算撤消。缺点：系统开销大。

期满(expiration)，思想：每个 RPC 都给对方一个标准的时间量 T，来作为它的工作期限，如果它在 T 时间内不能完成工作，客户必须重新请求，以保证每个 RPC 就可以在 T 时间内完成。当有客户崩溃后，在客户机重起之前等待一个时间量 T,这样可保证所有的 Orphan 已发送。缺点：时间量 T 的取值比较复杂。

6.RPC 的实现

RPC 协议选择(protocol selection)是选择面向连接的还是非连接的协议

面向连接的协议的优点：通信实现容易：内核发送消息后，它不必关心它是否会丢失，也无须处理确认，而这些方面都由支持连接的软件来实现。缺点：性能较差（需要额外的软件）。在单一建筑物和校园内使用的分布式系统可以采用面向连接的协议。

是标准通用的还是 RPC 专用的协议

要使用自定义的 RPC 协议就得完全自己设计一些分布式系统使用 IP(or UDP)作为基本协议，原因是：IP/UFP 协议已经设计，可节约相当可观的工作消息包可在几乎所有的 Unix 系统中传送和接收 IP/UDP 消息支持许多现存的网络总之，IP/UDP 很容易使用，并且适合现存的 Unix 系统和网络关于性能，IP 是无连接的协议，在它之上可建立起可靠的 TCP 协议（面向连接的），IP 协议支持网关的报文分组，使报文从一介网络进入另一个网络（物理网络）。预言，一个高性能的 RPC 协议可望面世。

确认机制(Acknowledgement)

1) stop-and-wait protocol:

思想：逐包确认。特点:一个包丢失了，可独立重传；容易实现。

2) blast protocol

思想:一个消息的所有包都发送完成后等待一个确认，而不是一个一个确认。特点:报文丢失时，有两种选择:

全部放弃：等待重传整个消息的所有包，容易实现；

选择重传：请求丢失包重传，占用网络的带宽少，对局域网这种方法较少使用，广域网络上普遍采用）。

流量控制(flow control)

通常网络接口芯片可以接收连续不断到来的包，但是由于芯片中缓冲区的限制，总是对连续到来的包的个数有个数量限制"超载"(overrun):一个消息到来而接收它的机器无法接收，以至于到来的消息丢失。超载是一个很严重的问题，它比因噪声等其他破坏引来的包丢失普遍得多。

以上两种确认机制对超载问题有所不同

stop-and-wait protocol：超载的可能性小，因为第二个包在没有收到明确的确认之前不能被收送。（在拥有多个发送用户的情况下可能存在）

blast protocol：接收方超载是很可能的：由于网络接口芯片引起的超载（因接收方处理一个进程而来不及处理到来的消息包）。

解决方法：

忙等待：CPU 空操作，应用短延时的网络环境；

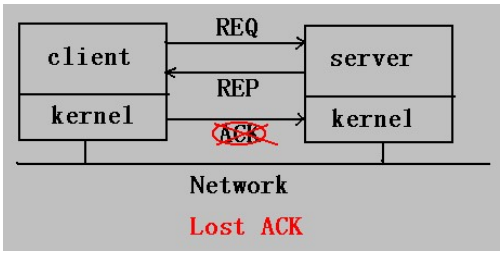
中断：发送进程挂起，CPU 重新调度，应用在长延时网络环境；

由于网络接口芯片缓冲区容量引起的超载，解决方法：

设有 n 个缓冲容量，发送进程每连接发送 n 个消息包时，便等待一个确认（由协议完成）确认丢失问题(lost ACK)

图中"ACK"包丢失，造成的问题。

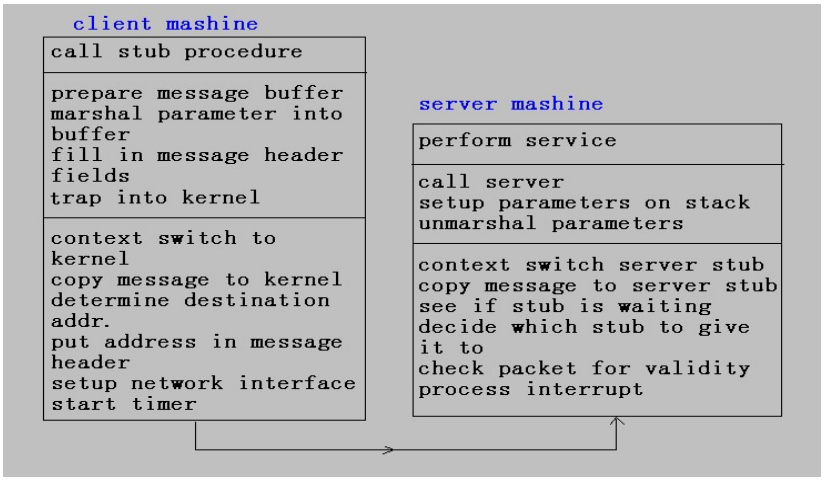
解决：对确认(ACK)进行确认（ACK），"超时确认"。



临界路径(critical path)

分布式系统是否成功依赖于它的性能的好坏，而系统性能的好坏又依赖于它的通信速度，通信速度与系统的具体实现有关，下面我们进一步讨论从客户到服务器执行一个 RPC 的过程。

临界路径(critical path)：每个 RPC 的指令执行的顺序是从客户调用客户代理，自陷进入内核，消息传送，服务器中断，服务器代理，最后到达进行请求处理并返回应答的服务器。RPC 的这一系列步骤称为（从客户到服务器的）临界路径。临界路径(critical path)图示



客户调用 stub procedure

申请一个缓冲区用来整理外出的消息，有些系统有一定数量的缓冲区，也有一些是一个缓冲区池，从中选择一个合适的供服务器使用。

参数整理

参数整理成适合的格式，并与消息一起插入消息缓冲区中。以备传送，自陷进入内核。

切换进入内核

内核获得控制，保存 CPU 寄存器及内存映像，建立新的内存映像。

拷贝消息到内核

因为用户和内核是不连接的，内核必须明确地把拷贝到内核缓冲区。

填入目标地址

将其拷贝到网络接口，到此客户临界结束。

原则上，启动计时器不属于计算 RPC 时间的部分，启动计时机后，系统有两种方式：忙等待和重新调度。

在服务器端，当字节到达后，被存入板上缓冲区或主存，当消息包的所有字节都到达后，服务器将形成一个中断。

检查消息包的有效性，并决定将其送给一个代理，若没有等待的代理则放弃或保存至缓冲区。

假定有一个等待的代理，那么，消息被拷贝到代理并切换到服务器代理，恢复寄存器及主存映像，代理调用 receive 原语，分解参数，建立服务器的调用环境，进行请求处理。

临界路径(critical path)的开销－拷贝

在考虑临界路径的时间开销问题时，其中最重大的部分是拷贝

copy1:客户代理－＞内核

copy2:内核－＞网络接口板

copy3:网络接口板－＞目标机器

copy4:网络接口板－＞内核

copy5:内核－＞服务器代理。

定时管理(timer)

所有的协议都是以通过通信介质交换消息为目的的，而事实上，所有的系统中，消息都有可能丢失，或许是因为噪声或是超载。所以，多数协议都设置定时器，当消息发

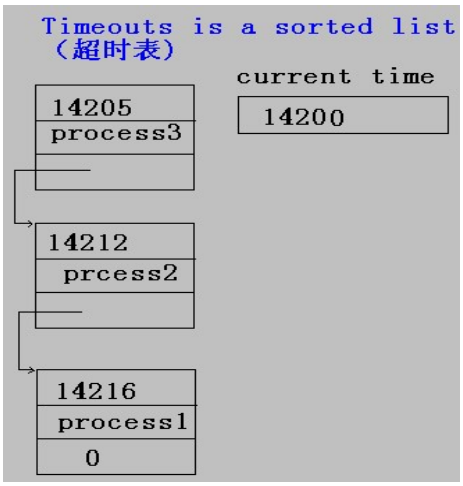
送出去后，期待应答（确认）的到来，若没有消息到来而定时器期满(expiration),则重发。这个过程可以重复几直到发送方放弃。

设置定时器：

建立一个数据结构来指定定时器何时期满，以及期满时如何处理。

多个进程的定时器组织成列表方式

当一个确认或应答在超时之前到来，列表中查出对应进程所在的项，将它删除，实际上，只有少数的进程可能超时，但每一个进程都要进入超时表后再删除，这些大量的工作多数是多余的。另外，定时器也不需要精确的时间，但定时太短引起过多的超时，定时太长则对包丢失的情况又过多的等待。



实现方法：在 PCB 中增加一个字段，称为定时器，当发出一个 RPC 时，将允许延迟的时间加上当前的时钟的值并存于 PCB 中定时器字段，如果不需要超时重传的，其值规定为 0，这样，内核定期扫描 PCB 链表，如果定时器值非 0 且小于当前时间，则该进程超时。

7.RPC 与消息传递通信的比较

RPC 结构性好，使用方便;消息传递通信更灵活，但结构性差 RPC 只有一个返回，而消息传递通信可以向多个客户返回。RPC 返回的结果或参数的值最好是少量的，消息传递通信可适合于大批量数据的传递。