异构型分布式计算机系统技术*)

薛　行　孙钟秀　（210008 南京大学计算机科学系）

摘　要

The purpose of a heterogenous computer system (HCS) is to allow users of computers with different hardware and software configurations to share resources. This paper first examines the various kinds of heterogeneity that can occur in a HCS and their influencces on the implementation of distributed resource sharing. The state of the art and techniques used in heterogenous system interconnection, data sharing, and processor sharing are then discussed in detail. Some contemporary systems that can accommodate heterogeneity are presented as examples.

异构型分布式计算机系统（HCS）由互连的不同硬件或软件系统组成，一个典型的HCS可配置以不同操作系统，用不同网络连接，具有不同类型的机器。

随着计算机技术的发展以及分布、互连和协同操作计算机系统的广泛使用，HCS越来越显示出其重要性，并得到更多的应用。即使用户的初衷是建立一个同构型系统，但往往最终也会自然地演化成异构型系统，这主要有以下几个原因。

1．分布式计算机系统已成为资源共享的重要方式。随着分布式系统资源的增多，其它用户也希望加入系统，共享其资源。这些新用户系统往往跟系统中现有硬、软件不同。

2．硬件性能的提高和其价格的下降。当扩展一个分布式计算机系统时，人们往往会购买或开发新型的计算机系统，而不是采用系统中已有的设备类型，保持系统的同构性。因此，异构型是不可避免的。

3．把不同的硬、软件系统结合在一个计算机系统中可以得到较高的性能价格比。在这样的系统中，配置有一些专为某种操作设计的具有特殊系统结构的处理器，普通工作站可以共享这样的功能。

然而，由于硬、软件系统上的差异，与同构型相比，互连不同的子系统，建立一个能够在各机器之间提供资源共享的异构型分布式计算机系统要困难得多。除了一般分布式系统要解决的共同问题之外，还有一些异构型系统特定的问题[1]，诸如通信、命名、数据表示、访问权限验证、远程执行等。下面，本文将讨论连接机制，数据共享和处理器共享的当前研究现状和异构型的处理技术，指出各种方案的得失，并给出系统实例。尽管同构型分布式系统的许多问题（例如控制资源共享的同步机制）也同样是异构型系统的问题，但我们的重点是异构型，所以在此不作专门讨论，有兴趣的读者可以参考〔2〕〔3〕。

一、异构型的种类

对于异构型分布式计算机系统的资源共享可能出现以下几种类型的异构性。

*)国家自然科学基金资助项目

1. 操作系统异构

各操作系统提供的系统调用功能和形式各不划一。此外，各操作系统的通信机制也不尽相同，有的用复杂的传输协议，有的则使用远程过程调用。如果不同系统之间要进行协同操作来提供资源共享，则必须有一种公共的通信方式。

操作系统异构性特别表现在文件系统异构。这种异构主要有三个方面。首先，不同文件系统用不同方法给文件命名。例如，在UNIX中，文件系统是层次型的，路径名的每个元素由除了字符"/"外的任意ASCII字符组成，"/"用来分隔路径名中的各个元素。MS-DOS的文件系统虽然也是层次型的，但路径名的第一个元素是驱动器名，并且文件名的形式是××××××××.×××。因此UNIX程序可以生成和使用任何MS-DOS文件名，但反之却不然。如果简单地把这两个系统容纳在一个系统中共享文件，则许多操作将会失败，因为UNIX下认为是完整的路径名，现在被MS-DOS认为缺少驱动器名。另外，文件名的长度也受到限制。

其次，不同文件系统对文件保护的方法和程度也不相同。例如，UNIX把用户分成三类：文件所有者、同组用户和其它用户，对文件进行保护；而MS-DOS则根本没有文件保护设施。因此，很难要求一个文件系统在共享另一个不同文件系统的文件时能够按那个系统的保护方法进行文件保护。

最后，不同文件系统具有不同的文件模式。有些文件系统，如UNIX只提供一种文件模式，即字节流模式；而有的，如VMS，则提供多种记录结构文件模式。

2. 硬件异构

通过网络互连的机器在硬件上主要有以下三个差异。首先，机器的指令集可能不同。这意味着一种机器的可执行代码程序不能在另一种不兼容的机器上运行。其次，各种机器的数据表示形式可能互不兼容。例如，DUAL机使用的M68000CPU的字长是16位，并且可按字节单位寻址。如果一个字的高字节的地址是i的话，低字节就是i+1；而IBM PC系列使用的Intel8086或80286也是按字节单位寻址的，但一个字里两个字节的顺序跟M68000刚好相反。因此，IBM PC机上运行的程序不能对DUAL机上的程序所产生的数据直接进行操作。

第三，尽管机器的类型可能相同，其硬件配置也可能互不兼容。例如，某一台可能比其它机器具有更多的内存或外存，而另一台则可能多配置了某种外设等等。

3. 程序设计语言异构

不同程序设计语言的语法和语义不同。更为重要的是它们用不同的方法在文件中存贮数据。例如，Pascal写的程序用十进制字符串的形式在文件中存放整型数；而使用UNIX系统调用的C语言程序则以二进制形式在文件中存放整型数。因而，不同语言书写的程序之间不能直接共享数据文件。

二、连接机制

分布式系统中各个组成部份之间的协作以各个结点之间必须进行连接为基础，提供资源共享。连接包含两个部份：在低层，通信机制提供各结点进程间通信的手段；在高层，服务接口定义各组成部份间通信的语义。本节将讨论连接所含的这两个部份的异构性处理技术。

2.1 通信机制

在用户层，各种系统一般都在通信网络基础上提供远程过程调用（RPC）作为进程之间通信的手段。RPC使得进程通信的语义类似于通常高级程序设计语言中的局部过程调用，它比信件传递方式更好地抽象了进程间的通信[4]。

尽管各个系统采用的都是RPC机制，但在具体设计和实现上有很大差别。首先是数据表示的问题。各机器和各个程序设计语言间同种类型数据的长度及字节顺序和结构型数据的安排各不相同，使得远程过程的调用方和被调用方不能正确地解释调用参数并返

回值。其次，各个系统的网络设施所采用的传输协议不尽相同，有UDP、TCP、XNS等。而RPC机制在底层是通过网络设施的传输协议实现的。第三，各种语言的语义和数据类型系统不完全相同，难以在不同语言之间保持RPC的语义。因此，不同系统的现有RPC设施之间不易直接相互通信。

目前，对于数据表示主要有两种解决方法：1）定义和使用一种标准的数据表示形式；2）在建立连接（即调用方和被调用方进行初次通信）时，指定一种表示形式。对于传输协议的问题也可以有类似的方法。

SUN RPC[5]采取了第一种方法。SUN公司定义了一种公共数据表示形式，称为外部数据表示（XDR）。当进行远程过程调用时，调用方的桩过程（stub）首先把调用参数转换成XDR定义的标准形式发送给目的机器。然后，由被调用方的桩过程进行解码，转换成对应数据类型在本机的表示形式。调用结果则按相反方向送回。目前SUN XDR已被许多公司（包括IBM和DEC）所接受。这种方法的好处是简单和统一，但存在效率的问题。一个典型的例子是两个系统采用不同于标准形式的相同数据表示，在进行通信时，将不得不进行不必要的编码和解码。

在传输协议方面，Washington大学的HRPC和DEC公司的SRC RPC采用了第二种方法[6]。它们在桩过程和底层传输机制之间定义一个接口。这个接口由若干个过程组成，可以对应于常用的传输协议以不同方式实现。然后，在建立连接时增加一个机制确定在特定的一对调用双方之间应该使用哪一种传输协议，并把应使用哪一种过程实现告诉桩过程。从而，使同一个桩过程可以使用多种传输协议。

对于解决不同语言（语法、语义）和数据类型的系统所引起的异构性，通常是通过实现桩过程，屏蔽具体细节，使RPC看上去就像该语言通常的过程调用。桩过程比较复杂，既要针对不同语言做不同实现，又要实现RPC的控制，底层数据的打包、拆包和跟传输层交互。所以，通常随RPC系统一起还提供一个桩过程生成器，自动生成桩过程。Horus系统[7]支持多种语言间的RPC，其方法是在规格说明中指出不同源语言及其不同机器上数据表示形式的差异，将其跟其它接口描述一起交给桩过程生成器，然后由生成器为调用双方生成能够处理这些差异的桩过程。虽然目前通过桩过程已能够解决不同语言语法上差异的问题，但要使所有现存程序设计语言在RPC的语义上相互兼容还相当困难，这个问题有待进一步探讨。

除上述系统外，还有一些RPC机制在不同程度上支持异构型。Matchmaker[8]系统通过给信件加上标记，告诉接收者信件体使用什么类型的数据表示解决数据表示不同的问题。它还可支持多种程序设计语言。Apollo的NCS/RPC[9]支持数据表示和传输协议的异构性。RPC信件是自定义的，包含了所送数据的类型信息。桩过程用Berkley UNIX的Socket方式的传输接口定义，可用不同方式实现。目前只支持UDP/IP和Apollo Domain两种传输协议。

2.2 服务接口

顾客/服务员模式是分布式系统中一种广泛使用的资源共享模式，它具有良好的模块性，服务员能够较好地向顾客隐匿实现服务的细节，包括实现策略，服务员机与顾客机在系统结构上的差异，资源的特性等。

为了使顾客和服务员之间能够有效地协作，必须定义一个接口。在一些早期的分布式系统中，这种服务的接口通常就是服务员结点上操作系统本身的服务功能。对同构型系统来说，采用这种接口确实是一种简单有效的方法。然而，对一个异构型的系统这种方法就不合适了，因为它要求所有顾客都遵循一个操作系统的标准，服务员操作系统难以满足顾客的各种需要。

一种极端的做法是允许每类顾客确定它自己与分布式系统服务的接口，满足每一类

顾客的需要，但服务员必须支持每一种接口是显然不可行的。另一种连接不同系统的方法是定义一组非常简单的基本功能作为服务接口，构成所有顾客系统功能的最小公分母，一个顾客系统的远程服务可以要求要多个原语来完成。虽然这种方法能解决异构型顾客和服务员系统之间的通信问题，但在进行机间原语调用信息交换上通信开销比较大。

DEC公司的NCL[10]和南京大学的ZGL系统[11]借鉴高级程序设计语言的概念，即用一种标准高级语言编写的程序可以在不同机器上编译生成不同的机器代码，设计了一种网络服务语言，这种语言与具体系统无关，并且允许顾客可以详细描述它想从服务员那里得到的服务。当顾客要求某种服务时，它就根据所要求的服务生成一个或多个标准网络服务语言的表达式（程序）。然后，把这些表达式送给服务员，服务员解释并完成这些表达式所要求的服务。

采用上述方法的基础是定义一种合适的网络命令语言。这种语言必须能够描述一个异构型分布式计算机系统所需要的和能够提供的所有服务功能。考虑到系统的效率，远程服务表达式还应该保持一定大小的粒度。在NCL和ZGL中粒度大小在系统调用一级。

三、数据共享

分布式文件系统是分布式系统数据共享的基本设施。一个异构型分布式文件系统应该提供不同机器之间的文件共享，使得现有应用程序可以通过其局部操作系统的标准接口访问本地文件和远程文件，对访问不同操作系统支撑下的远程文件不需做特别处理。然而，由于前述的异构性，要达到这个目标还有许多问题，如文件命名，文件组织，文件保护和文件模式。

3.1 文件命名和文件组织

分布式系统主要有三种命名方法。第一，也是最简单的一种是用文件所在的机器名加上文件在该机上的路径名指出远程文件名。IBIS和Washington大学的HFS[12]采用了这种方法。这种方法的好处是易于容纳异构型系统。用户以机器:局部文件名的形式指出要访问的远程文件。分布式文件系统可以不必涉及每台机器上文件系统的结构和文件名的形式，它把要访问的文件名发到提供这个文件的机器上，由该机的文件系统解释路径名。这种实现比较简单，但是这种方式没有提供位置透明性和位置无关性，是一个很大的缺陷。

第二种方式以SUN公司的NFS[6]为代表，允许用户把其它机器上的子目录安装到本机文件树的某个子目录上，可以按照访问本地文件的方式访问远程文件。这种方法具有较好的网络透明性和位置透明性，但实现起来比前一种方法复杂些，要求异构型分布式文件系统扩充不同的文件系统，提供一种统一的语义，在不同文件系统的文件名之间进行转换。SUN NFS定义了一个网络文件共享协议并在各种不同的局部文件系统上定义了一个屏蔽异构性的虚拟文件系统VFS。虚拟文件系统VFS之间通过网络文件共享协议进行文件共享。SUN NFS目前已能够在UNIX，VMS，MS-DOS等文件系统之间共享文件。

第三种命名方式是全系统使用一个单一的全局的名字空间，具有较高的网络透明性，有利于实现全系统范围的资源共享。然而，这种系统的实现比较复杂，尤其是对异构型系统来说，协调各种不同系统和修改现有操作系统均十分困难。Locus[13]，Sprite[14]等采用了这种方式，只提供基于UNIX的系统间的文件共享。Andrew[15]可以容纳UNIX和MS-DOS，但使用MS-DOS的机器必须通过一个专门的PC服务器参加到系统中来共享资源。

3.2 文件保护

分布式文件系统必须防止用户对文件做未经授权的访问，访问权限验证机制通常跟局部操作系统紧密相关，往往各不相同。有的采用访问权限表，有的采用权标(Capability)，要在这样一种跟低层操作系统有关的

机制上容纳异构比较困难。目前的方法主要有：1）异构型分布式文件系统保存一张访问权限表。每当用户访问远程文件时，由系统通过这张表进行访问权限验证。这种方法的优点是实现简单，但是要求为系统中每一个可能被共享的文件都保存一个访问权限信息。当系统比较庞大时，进行访问权限验证的开销就比较大。2）利用系统中各机器局部操作系统现有的保护机制。因为大多数文件系统中都设有一个保护类别"其它用户"，包括除了文件所有者外的所有其它用户。例如，在UNIX中，用户分成三个保护类别：文件所有者，同组用户和其它用户。我们可以把其它用户这一类扩充到分布式文件系统的用户，把所有远程访问的用户都归入"其它用户"一类。SUN NFS和Washington大学的HFS采用了这种方法，它的优点是易于容纳异构型的文件系统且实现简单，但在文件的保护方面有潜在的不安全因素。

3.3 文件模式

在设计异构型分布式文件系统中最为困难的要数文件模式，因为在异构型环境中，不同机器支持不同的文件结构，有的是没有结构的字节流，有的是高度结构化的记录文件。那么，不同结构的文件系统之间如何进行文件共享呢？一个异构型分布式文件系统应该支持什么样的文件操作呢？

SUN NFS的方法是只提供一种基本的文件类型，即字节流。不同系统之间的文件共享在这种基本类型的基础上进行。Washington大学的HFS功能较强，它定义了一种基本的记录类型和对这种类型的一组操作。在此基础上可以扩充成许多不同类型的文件模式[12]。HFS在每台机器上实现把远程顾客给出的文件操作请求转换成对局部相应类型的文件操作。因此，当一个VMS顾客要求在UNIX机器上读一个包含10个长整型数的记录时，将会翻译成从文件中读40个字节。

四、处理器共享

分布式计算机系统中资源共享的另一个重要方面是处理器共享，工作站用户能够利用系统中空闲的工作站或处理器池中的处理器，提高处理效率。另外，普通工作站可以共享一些专门为某种操作设计的具有特殊结构的处理器。处理器共享也是开发分布并行处理软件的基础。

由于异构型分布式计算机系统中机器硬件和操作系统等的异构性，在异构型系统中进行处理器共享远比同构型系统困难。在本节，将讨论异构型系统中处理器共享的途径及有关技术。

4.1 任务远程执行

处理器共享主要有两种模式。一是动态进程迁移，二是任务远程执行。动态进程迁移指通过系统中各处理器之间动态地迁移正在运行的进程，达到全系统范围内处理器的负载平衡，从而充分利用系统的处理能力。任务远程执行指在该模式下应用程序把远程执行的要求（命令），而不是要在远程执行的程序，交给计算机服务器，服务器根据顾客的要求使用局部或者共享文件系统中适合自己系统结构的二进制装入模块和有关的数据进行操作，然后把计算结果返回给顾客。因此，计算机服务器可以在很大程度上向顾客工作站隐匿实现细节，包括机器类型和支撑操作系统上的差异。

现在许多系统提供了支持远程执行的工具。例如，UNIX系统的rsh，rcp，rlogin和rdist等命令使用户可以在UNIX机器组成的网络中进行远程操作。Locus，Apollo Domain，Newcastle Connection，VAX群集系统和Eden等支持对网络处理器和存贮设施的透明访问。但是这些工具和服务都是在同构型的硬件或操作系统的环境中实现的。南京大学的ZGL系统[16]支持工作站运行UNIX和MS-DOS两种不同操作系统，允许在命令和程序级通过远程执行进行处理器共享，应用程序可以在执行中动态地申请一个或多个计算机服务器执行远程命令。Washington大学的HCS[6]也支持异构型工作站之间的任务远程

执行。此外，HCS还提供了一个高层服务描述语言THERE，需要进行远程执行的用户可用THERE语言定义顾客和服务员间交换的信息和创建远程执行服务所需环境的步骤。顾客和服务员机器上各有一个不同版本的THERE程序，当用户给出一个远程执行请求时，顾客机上的THERE语言解释器将解释相应的THERE程序，并把远程执行要求发给服务员机上的解释器，后者再根据服务员一侧的THERE程序中的定义，跟顾客机上的解释器交互，创建所需的环境。最后，把服务程序调入执行。

任务远程执行模式的优点是实现相对简单，易于在异构型机器之间共享处理器，比动态进程迁移的方法开销小，比较适合于由工作站和个人计算机组成的分布式系统。但另一方面，这种方法共享的程度比较低，需要应用程序显式指出需要远程执行的命令。另外，计算机服务器必须事先将顾客可能要求执行的命令进行编译、连接好，并存放在局部或共享文件系统中，否则将无法提供服务。

4.2 动态进程迁移

动态进程迁移比任务远程执行模式具有更好的透明性，并且能够更为有效地利用全系统的处理器资源。但是，它的实现也更为复杂。目前同构型系统中的进程迁移技术已趋于成熟。然而，异构型系统中的进程迁移，也就是在具有不同硬、软件配置（包括不同处理器结构，机器指令集和操作系统）的机器间动态移动进程难度较大，研究也刚刚开始。目前文献报道的技术主要有源代码解释执行和重编译这两种方法。

4.2.1 源代码解释执行 进程迁移跟任务远程执行不一样，交给计算机服务器执行的不是命令而是程序。由于各处理器的结构、指令集各不相同。不能简单地把一台机器上的程序动态地移到另一台机器上执行。考虑到高级程序设计语言通常与具体物理机器无关，源代码解释执行法[17]是在每台机器上配置一个高级语言的解释器，由解释器直接解释执行高级语言程序的源代码。当需要迁移时，把这个进程的源代码连同解释器保存的状态和运行现场数据一起移到目的机，由目的机上的解释器恢复该进程的现场，并从迁移点接着执行。但是，结果为每一种高级语言都配备一个解释器及有关的迁移机制，则实现的代价太大，而且直接解释执行高级语言的开销也很大，影响到系统的效率。因此，一种改进方法是把高级语言编译成一种机器无关的中间语言，由每台机器上的解释器加以解释执行，并在此基础上实现进程迁移。为了保证能够正确执行迁移后的进程并且使进程已迁移的事实对与其协作的进程透明，除进程本身的迁移外，还要实现位置透明及位置无关的进程通信机制和进程环境的建立。进程通信机制的工作与同构型系统相同，这里不再赘述。进程环境的建立涉及到该进程打开的文件，当前缺省路径等。解决的方法通常是在迁移进程时附上关于环境的描述信息，然后由目的机和源机上的解释器交互，为迁移后的进程创建环境，就像上一节Washington大学的HCS所做的一样。由于源机和目的机上的操作系统可能不同，其接口和功能也就不一样。在有的系统中，环境描述信息中还包含关于操作系统接口和功能的描述，由目的机上的解释器将其映射到本机操作系统的接口。

4.2.2 重编译 Theimer 和 Hayes在[18]中提出了通过重编译实现异构型进程迁移的方法，其思想是将高级语言程序编译成二进制装入模块执行。迁移时，系统把该进程的进程状态和数据以高级语言过程的形式嵌入该进程源代码迁移点的位置，然后在目的机上编译和执行这个程序。编译后将首先执行系统嵌入的那个过程，由这个过程重新构造进程的状态和该进程在源机上所有全局数据和堆数据，恢复堆栈的状态，并接着从迁移点往下执行。这种方法的优点是把不

同机器进程状态和数据表示之间的差异转换成高级语言的过程表示，通过高级语言的编译程序来解决。另外，由于这种方式下执行的是编译生成的机器指令，其效率要比解释执行高。当然，重新编译和连接所花的时间是额外的开销，但这可以通过事先在目的机上将该程序的源程序编译好，当进程迁移来时，只对嵌入过程涉及的模块进行编译，然后连接生成可执行程序。还有一种方法是指定某种中间语言，在此基础上进行重新编译。

参考文献

[1] Notkin, D., et al., Heterogeneous computing environments: Report on the ACM SIGOPS Worksop on Accommodating Heterogeneity, Commu. of ACM, 30 (2), Feb. 1987

[2] Tanenbaum, A.S. and Van Renesse, R., Distributed operating systems, ACM Comput. Surv., 17 (4), Dec. 1985

[3] Levy, E. and Silberschatz, A., Distributed file systems: concepts and examples, ACM Comput. Surv., 22 (4), Dec. 1990.

[4] Birrel, A.D. and Nelson, B.J., Implementing remote procedure calls, ACM Trans. Comput. Syst. 2(1), Feb. 1984

[5] Sun Microsystems, Inc., Network programming, Sun Microsystems, Revision A, May 1988

[6] Notkin, D., et al., Interconnecting heterogeneous computer systems, Commu. of ACM, 31(3), March 1988

[7] Gibbons, P.B., A stub generator for multi-language PRC in heterogeneous environments, IEEE Trans. Softw. Eng. SE-13(1), Jan. 1987

[8] Jones, M.B. and Rashid, R.F., Mach and Matchmaker: Kernel and language support for object-oriented distributed systems, Proc. of OOPSLA86, Sept. 1986

[9] Dineen, T.H., et al., The network computing architecture and system: An environment for developing distributed applications, Proc. of the USENIX Conf., 1987

[10] Falcone, J.R., A programmable interface language for heterogeneous distributed systems, ACM Trans. Comput. Syst., 5(4), Nov. 1987

[11] Sun, Z.X., et al., Developing a heterogeneous distributed operating system, ACM Operating System Review, 22(2), Apr. 1988

[12] Pinkerton, C.B., et al., A heterogeneous distributed file system, Proc. of the 10th Intl Conf. on Distributed Computing Systems, July 1990

[13] Poper, G. and Walker, B., The LOCUS distributed system architecture, MIT Press, 1985.

[14] Nelson, M., et al., Caching in the Sprite network file system, ACM Trans. Comput. Syst., 6(1), Feb. 1988

[15] Satyanarayanan, M., Scalable, secure, and highly available distributed file access, IEEE Computer, 23 (5), May 1990

[16] 薛行，孙钟秀，ZGL 分布式操作系统的设备共享，计算机学报，1991年2月

[17] Attardi, G., et al., Techniques for dynamic software migration, Proc. of 5th Annual Esprit Conf., Nov. 1988

[18] Theimer, M.M. and Hayes, B., Heterogeneous process migration, Proc. of 11th Conf. on Distributed Computing Systems, June 1991