计算机科学1998Vol. 25 №. 5

支持分布式多媒体的对象管理系统 DMOMS*)

A Distributed Multimedia Object Management System

高瀚昭 杨国良 曾庆凯 谢 立

(南京大学计算机系计算机软件国家重点实验室 南京 210093)

摘 要 In this paper, a distributed multimedia object management system is proposed and implemented. It is divided to two parts: Distributed Multimedia Object (DMO) and support system (MPVM). On contrast to other systems, this system improves synchronization and consistency of multimedia data, simplifies the development of applications.

关键词 Distributed multimedia, Synchronization, Consistency

1. 引言

近年来,在计算机领域发展最快的当属多媒体技术和网络通信技术。然而,在分布式网络下开发多媒体应用程序不可避免地会遇到一些的问题[1]:(1)对不同媒体数据(如文本,图片,声音等)如何进行组织;(2)如何开发简单而有效的工具对数据进行管理;(3)多媒体数据的本地处理方法;(4)如何在网络各节点分布数据;(5)与现有标准及开放平台的兼容性。

近几年来围绕这一领域也开展了很多研究。H W. Peter Beadle 进行了多点间多媒体通信的实验[2],Yahya Y. Al-Salqan 和 Carl K. Chang 分析了媒体数据间的时间关系并提出了基于 Petri 网的同步机制[3],T. Helbig 和 K. Rothermel 也提出了基于分布式多媒体环境的同步体系[4]。但这些研究往往集中在解决以上问题的一个方面,而没有将它们构成一个完整的系统。

Stuart Wray 等提出了 Medusa 环境[5],采用了模块的设计思路,其结构如图1所示。这一结构的缺点在于应用程序还是要和各个组件模块分别连接、并没有减轻应用程序的设计负担。为此,我们提出了分布式多媒体对象管理系统(Distributed Multimedia Object Management System,简称 DMOMS)的概念,目的在于让用户不必关心数据是如何组织的,位于本地还是其它机器上,其具体分布状况如何,等等。

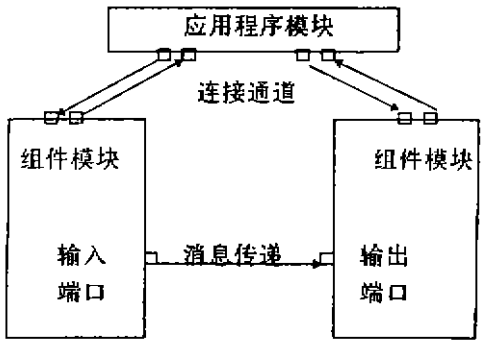

图1 Medusa 环境体系示意图

DMOMS 是基于对象的分布式多媒体系统 FDM 的一部分。该模型根据多媒体数据处理的特点,将分布式多媒体系统分为基本存取层、媒体存取层、分布式多媒体对象层、多媒体表现层和应用层,如图2所示。其中,基本存取层完成对设备/文件的基本 I/O 操作,媒体存取层将数据流划分为不同媒体种类,以媒体为单位进行存取。DMOMS 是其中的分布式多媒体对象层,基于媒体存取层之上,将相互有依赖关系的多种媒体合并为一个多媒体对象进行存

*) 本论文工作得到国家863计划"实用化软件开发工具"项目和江苏省科委"分布式多媒体技术研究"项目的支持。高瀚昭、杨国良 研究生,研究方向为分布式多媒体;曾庆凯 副教授、研究方向为分布式多媒体;谢立 教授,博士导师,研究方向为并行分布处理。

取。

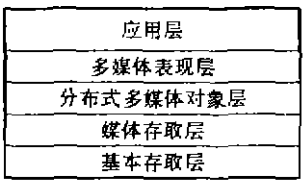

图2 基于对象的分布式多媒体系统 FDM 示意图

2. DMOMS 的结构

DMOMS 的基本结构如图3所示。与图1相比，图3中的调用程序只需在本地构造分布式多媒体对象并调用它的相应方法即可。分布式多媒体对象将各个方法转化为一组命令发给本地构件管理器，本地构件管理器再把命令划分为本地命令和远地命令，本地命令转化为对本地构件的方法调用，远地命令则转发给远地的构件管理器执行。这样就使下层的结构与调用者无关，真正做到了网络透明。

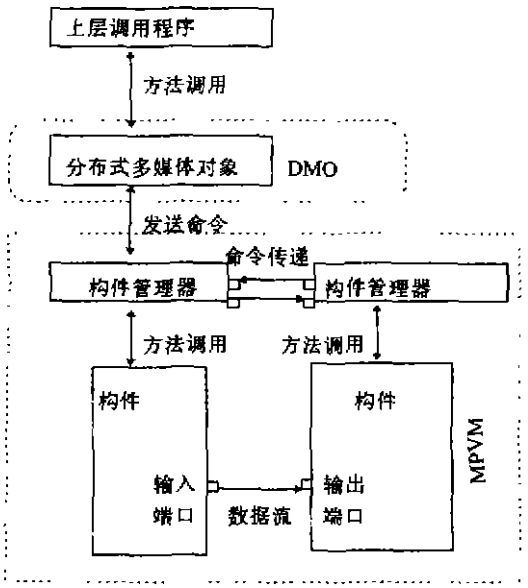

图3 DMOMS 的基本结构

DMOMS 分为两大部分：分布式多媒体对象(Distributed Multimedia Object，DMO)和多媒体并行虚拟机(Multimedia Parallel Virtual Machine，MPVM)。

本文将先介绍 DMO 的概念和特点，然后介绍 MPVM 的功能。

3. 分布式多媒体对象 DMO

DMO 层是与用户交互的一层，用户首先创建 DMO 并通过填写媒体描述表向 DMO 提交任务的资源描述，再通过调用 DMO 的相应方法来提交任务的动作描述。

DMO 的功能是将下层的构件对象相连接构成一条数据通道来完成多媒体数据的各种分布式操作。为此，DMO 向下层提供了流速控制的接口，通过它接受下层构件对象的反馈信息和发出控制消息。同样，DMO 也向上层调用者提供一组方法方便地对 DMO 进行管理。此外，对于由多个媒体流组成的 DMO，还需解决媒体间的同步关系。

下面将先给出媒体描述表的格式，然后提出我们的同步策略，最后给出 QoS 的调节办法。

3.1 媒体描述表(MediaTable)

媒体描述表的格式如下所示：

MediaTable = {MediaID, MediaType, MediaSource}

其中，MediaID 是媒体流的标识，以后就通过此标识访问该媒体流；MediaType 是媒体流的类型，通常有 Video，Audio，Text，Image 等，不同类型的媒体流由不同的构件处理；MediaSource 是媒体流的源名称，包括所在机器名和设备/文件名。这三项可指明一条媒体流，而如果 DMO 是由多个媒体流合成的，媒体描述表中就需要指明多个域，各条媒体流分别对应一个域。

3.2 媒体同步策略

同步分为两个层次，即用户级同步和系统级同步，分别对应层次同步模型和点同步模型。

用户级同步是由用户通过人机交互对象控制媒体对象的表现同步，这是一种广义上的同步。由于 DMO 提供的方法调用并不直接面向用户，因此我们的工作重点是在系统级同步层。

系统级同步不含用户交互信息，用于内容对象内部各个同步数据流的同步。它们将按约定的同步策略实现同步的组织。这一级的同步较用户级同步粒度更细。不仅仅在启停位置要求同步，在整个媒体播放过程中，都要求同步。

常见的点同步模型有两种：时间轴同步和参考点同步。时间轴同步是指先确定一根时间轴，所有的同步对象都以这一时间轴为参照系给出相应的时间坐标。它的优点是各同步对象之间相对独立，但它只能表现媒体间简单的同步关系，不能完成复杂的同

步操作表示。参考点同步是指将媒体数据分为一系列离散的单元，每个单元的位置称为参考点，这样，将不同媒体流的相应参考点连接起来就实现了流间同步。它的优点是理论上可表达一切同步关系，但在实际应用中，往往需要以某一条数据流作为主同步对象，于是当这一对象发生差错时往往会影响其它以它为参照的数据流，代价也是高昂的。

为更好地描述媒体间的同步关系，我们融合了时间轴同步和参考点同步的优点，提出了同步标记法。与参考点同步类似，我们也将媒体数据分为很多子单元，并将同步标记加在每个子单元上，然而，这种同步标记是依赖于时间轴的，这样就解决了当主参考媒体丢失时整个同步系统的崩溃。同时，由于同步标记的粒度与参考点同步相当，因此它的表现力也不是问题。

由于在分布式环境下，没有统一的全局时钟，所以需要引入相对时间系统，即把多媒体对象创建的时间作为时间系统的零点，并在它创建构件对象时，也将当前的相对时间坐标传递给构件对象，一个媒体单元在相对时间系统中的位置就作为它的同步标记（Synchronized Token）。每个媒体单元都有一个ST，在播放时，有相同ST的媒体单元必须同步。

那么，一条媒体流划分为媒体单元后可表示如下：

$\{M_iST_1, M_iST_2, M_iST_3, \cdots, M_iST_n\}$

其中，i是这条媒体流的MediaID，n是这条媒体流划分的媒体单元的总数。于是，流间同步关系的描述可以用同步关系描述表SyncTable表示：

$SyncTable = \{\{M_1ST_1, M_2ST_1, \cdots, M_iST_1\}, \{M_1ST_2, M_2ST_2, \cdots, M_iST_2\}, \cdots, \{M_1ST_n, M_2ST_n, \cdots, M_iST_n\}\}$

表中每一项对应的媒体单元在播放时均需保证同步。当然，在不需要同步的情况下，任意一项中的M_iST_j均可为空，表示该媒体单元不参与此项的同步，只需顺序播放即可。这样的表示法可以充分表现出多媒体对象中各个媒体流之间的同步关系。

这样，当接收方从发送方接收到同步标记为M_iST_a和M_jST_b的媒体单元时，首先查照SyncTable，若$\{M_iST_a, M_jST_b\} \subseteq x, x \in SyncTable$，则这些媒体单元在播放时要求同步，这些媒体单元组成的集合叫做同步集合。然后，在演示过程中，由输出构件定时向DMO返回各输出媒体单元的ST，DMO通过反馈的ST值判断正在播放的媒体单元是否同步，如果不同步，则通知相应构件对象进行流速控制，以期达到要求。

3.3 QoS的调节

在分布式多媒体系统中，为解决用户的静态需求和系统资源的动态分配间的矛盾，引入了QoS的概念。

QoS参数可分为三层：用户层、应用层和系统层，用户层参数是指通过给用户一些代表不同等级QoS参数的示例（如高、中、低）让其选择来确定应用将采用的QoS服务质量；应用层QoS参数是指媒体数据的表现属性，如对视频流而言，是帧频，调色板，颜色数，图象大小等；系统层参数是指资源的需求，分本地级（CPU优先级，外围设备，和缓存分配）和通信级（包传输率，最大延迟，数据包大小等）。

DMO层作为DMOMS的上层，向调用者提供用户层QoS参数并将用户层QoS需求转换为应用层QoS参数传递给下层构件。

在实际系统中，上层应用程序在运行前须设定QoS参数，并与下层资源管理器协商资源的分配，成功后方开始运行。在运行中，各构件也不断向DMO反馈运行情况，需要时由DMO通知各构件进行QoS重协商。

4. MPVM设计

MPVM的功能是提供一组基于操作系统的抽象接口，屏蔽各机器间的差异，使一个分布式的环境抽象为一台并行虚拟机。为此，MPVM需要提供存取服务，资源管理，进程通信，数据传输等功能。下面将分三方面介绍MPVM：构件的扩充和管理；多媒体资源的管理；通信机制的改进。

4.1 构件的扩充和管理

构件的功能是提供基本的输入输出和数据传输等，它们构成了DMOMS的基础。DMO正是通过将构件组装成流水线结构来实现分布式多媒体数据的处理过程。可是，如果让DMO直接管理每一个构件，那么DMO的负担太大，也不符合网络透明性的思想，于是，我们提出了构件管理器的概念，即在每一台机器上运行一个构件管理器，专门负责本地构件的管理。

为进一步提高网络透明性和降低DMO负担，DMO也只需和本地的构件管理器进行交互。构件管理器接到DMO的命令后，若是本地命令，则将其转化为一组对构件的方法调用，否则将其转发给远地的构件管理器。

4.2 多媒体资源的管理

对于一般的资源管理要求，PVM 系统已提供了一套信号量机制进行处理。但是，很多多媒体应用（如 CSCW）需要广播形式的共享访问，这是信号量机制无法处理的。

为此，我们提出了资源管理器的概念，用来管理各媒体资源的登录和分配，每一个资源对应一个资源管理器，所有的对资源的输入/输出访问都通过资源管理器来完成。这样，就可以用广播方式解决顺序存取设备的共享问题。

资源管理器维护一张资源状态表 $RST=\{(R_i, S_i)|i=1,2,\cdots\}$，其中，$R_i$ 为一正整数，表示第 i 个使用者的权限级，$S_i=\{True, False\}$ 表示第 i 个使用者是否锁定该资源，若锁定该资源则 $S_i=True$，反之则 $S_i=False$。此外，定义变量 c 为计数器，统计当前使用者总数。变量 m 为主用户标识，即 (Rm, Sm) 对应的用户为主用户，可对该资源进行完全的控制；其他用户则为共享用户，只能对资源进行共享操作而不能改变它的存取状态。

资源管理器接收到输入/输出请求 (Ru, Su) 后，首先查询 RST：

1 若 $RST=\varnothing$，则 $RST=\{(Ru, Su)\}$，且令 $c=1; m=1$
2 若 $RST\neq\varnothing$，比较 Ru 与 Rm
(1)若 $Ru>Rm$，则 $RST=RST\cup\{(Ru, Su)\}; c=c+1; m=c$
(2)若 $Ru=Rm$，则 $RST=RST\cup\{(Ru, Su)\}; c=c+1$
(3)若 $Ru<Rm$，
①若对于 $\forall Ri, ((Ri, Si)\in RST)\wedge(Si=True)$，均有 $Ru\geq Ri$，则 $RST=RST\cup\{(Ru, Su)\}; c=c+1$
②若 $\exists Ri, ((Ri, Si)\in RST)\wedge(Si=True)$，使得 $Ru<Ri$，则拒绝访问请求

在资源输入/输出过程中，资源管理器将输出的数据流用广播方式同步发送给主用户和共享用户，但只有主用户才能改变当前的输入/输出状态。

当某个使用者中途退出，若其不是主用户，则简单地将其从资源状态表中删除，否则需要将表中权限最高的用户且最先访问资源的用户作为新的主用户并通知该用户。

4.3 通信机制的改进

多媒体数据的通信和普通数据通信相比，有数据量大，实时性强，精确性低，数据需要同步等特点，针对以上特点，我们可以归纳出对多媒体网络协议的几点要求：

(1)速度优先，只需对数据的格式说明等关键信息进行校验，大量的数据块只需"尽力传递"。

(2)流量控制，收发双方必须提供流量控制机制来限制网络流量。

(3)同步控制。同样，收发双方应能提供一套机制对同步的数据流进行处理。

现在通用的传输协议有 TCP（传输控制协议）和 UDP（用户数据报协议）两种。

传输控制协议 TCP 提供了很高的可靠性。为保证可靠性，它采用了确认与超时重传、流量控制、拥塞控制等技术，导致效率不够高。

用户数据报协议 UDP 的优点在于效率高，因为它并未增加多少系统开销。它的缺点是不提供可靠性服务，对于报文丢失、重复、失序等情况必须由应用程序自己解决。

为此，我们提出了 EUDP 的设计思想，它以 UDP 协议为基础，综合 TCP 中的流量控制技术，再加入同步控制技术，在不影响效率的前提下提供了必要的控制功能，它对 UDP 协议作了以下改进：

(1)预传数据流说明，并在说明信号位作标志，收件方据此进行数据校验，若不符则进行重传。

(2)用滑动窗口进行流量控制，其机制与 TCP 协议相同。

(3)在每个数据报的头标附加时间坐标，收件方根据时间坐标进行取舍及同步处理，并适当调整滑动窗口大小。

5. DMOMS 应用举例

下面举例说明应用程序如何利用 DMOMS 开发分布式多媒体应用程序。

当应用程序需要在本地播放一个远地摄像机的摄入数据时，它首先填写媒体描述表，其中 MediaID =1表示1号媒体流，MediaType =1表示是视频流，MediaSource =(dest, device)，dest 远地摄像机所连的机器名，device 是摄像机的设备名。当用户根据此表创建 DMO 时，DMOMS 中的 DMO 层便分析该命令并向本机的构件对象管理器发出"VideoOutput"和"VideoInput(dest)"两条命令，构件对象管理器随即启动本地的视频输出构件和接收构件，并把启动远地构件的命令发给远地的构件管理器，由远地构件管理器负责启动视频输入构件和发送构件。分布式多媒体对象得到所有构件启动成功的消息后，返回 DMO 创建成功的消息，用户就可调用 DMOMS 的 Play()方法进行播放，接到方法调用后，DMO 再向本地构件管理器发出"VideoStart"和"VideoStart(dest)"两条命令，构件管理器以同样的方式将这两条命令分别发给这四个构件，由构件负责完成剩下的数据的读发收放等操作。同时，构件也将当前的 QoS 信息及同步标记发给构件管理器，再转发到分布式多媒体对象，由分布式多媒体对象

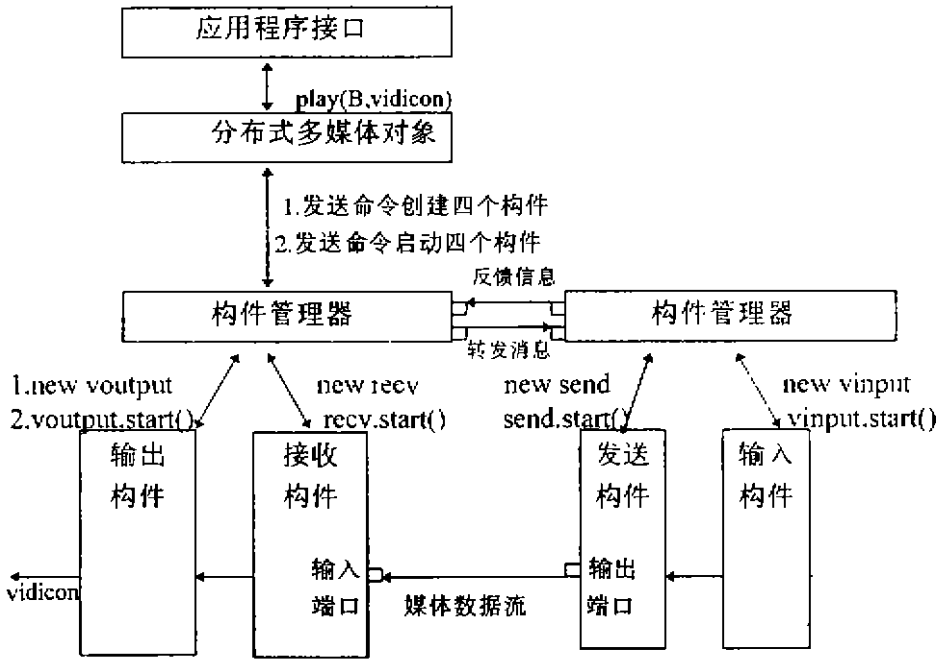

图4 DMOMS 工作流程

进行同步和 QoS 的协调工作。整个工程流程如图4所示。

参 考 文 献

[1] Giacomo Bucci et al. ,Sharing Multimedia Data Over a Client-Server Network,IEEE MultiMedia Fall 1994

[2] H. W. Peter Beadle,Experiments in Multipoint Multimedia Telecommunication,IEEE Multimedia Summer 1995

[3] Vahya Y. Al-Salqan,Carl K. Chang,Temporal Relations and Synchronization Agents,IEEE Multimedia 1996

[4] Tobias Helbig,Kurt Rothermel,An Architecture for a Distributed Stream Synchronization Service,Interactive Distributed Multimedia Systems and Services,Springer-Verlag Berlin heidelberg 1996

[5] Stuart Wray et al. , Networked Multimedia: The Medusa Environment, IEEE MultiMedia Winter 1994

（上接第113页）

表1

Machine	Processor	Spec rating	MIPS
DEC2100	R2000	8.7 Specint89	~11
DEC3100	R3000	11.8 Specint89	~15
DEC5000/125	R3000	16.1 Specint92	~25

结论 从以上种种迹象表明，外展式核心能够安全、有效地复合硬件资源，在低级的外展式核心界面上，可以通过操作系统库完成传统操作系统所做的一些抽象。我们可得出以下的结论：①外展式核心可以被有效地创建。②低级的硬件资源可以被安全地复合，并提供给应用使用。③传统操作系统完成的一些抽象可以在应用级有效地实现。④应用可以根据具体情况创建自己的抽象。我们相信外展式核心这样一种体系结构有其发展的天地，也具有发展前途。然而，外展式核心是一个新型的系统，还存在着许多不足，有许多问题值得我们去研究，去探索。

参 考 文 献

[1] Dawson R. Engler et al. , Exokernel: An Operating System Architecture for Application-level Resource Management,M. I. T. Laboratory for Computer Science

[2] Dawson R. Engler,M. Frans Kaashoek,Exterminate all Operating System Abstractions,Same to[1]

[3] Dawson R. Engler et al. ,The Operating System Kernel as a Secure Programmable Machine,Same to[1]

[4] M. Frans Kaashoek et al. , Application Performance and Flexibility on Exokernel Systems,Same to[1]

[5] Dawson R. Engler et al. , AVM: Application-level Virtual Memory,Same to[1]