网格计算环境下的I/O支持机制的分析、改进与实现*)

王 彬 徐国市 罗 引 许卓群 丁文魁
(北京大学信息科学技术学院计算机科学技术系 北京100871)

摘 要 在广域的网格计算环境下,计算所在站点常常远离相关的数据站点和用户控制台。因而网格计算环境下启动管理远处计算应用就要不可避免地考虑为其提供I/O支持(包括传入输入文件、传出输出文件,标准输入、标准输出和标准出错等3个特殊文件的处理)。网格计算环境的系统,如Condor,Legion和Globus等,都给出了各自的I/O解决方案。这些解决方案满足了启动远程计算对I/O机制的大多数的需要,已获得广泛的使用。但这些解决方案在设计时没有考虑到两个实际中的I/O需要——用户在计算过程中实时的标准输入和中间结果文件的实时传走,因而不能解决这两个实际需要。所以,本文提出了在网格计算环境下远程计算的I/O支持系统的改进的设计方案,并在Globus Toolkit的I/O解决方案基础之上给出了实现的参考方案。改进后的文件I/O机制能够更好地满足了广域的网格计算环境下计算应用的需要。

关键词 I/O,标准输入,网格计算,Globus toolkit

Analysis, Improvement and Implementation of I/O Support Mechanisms in Grid Computing Environments

WANG Bin XU Guo-Shi LUO Yin XU Zhuo-Qun DING Wen-Kui
(Department of Computer Science and Technology, School of Electronic Engineering and Computer Science, Peking University 100871)

Abstract In Grid computing environments, computation is often performed at a site distant from data needed and from user console. As a result, it inevitably involves providing I/O support for submission and management of computations at remote sites. Many Grid software providers, such as Condor, Legion, Globus, have implemented their own I/O solutions in their released packages. These solutions meet most requirements for I/O mechanisms and have seen wide application. But two important user requirements are not sufficiently addressed when designing these solutions, namely, run-time stdin from user console and real-time stage-out of scratched results files during the course of computation. Regarding these deficiencies, the paper puts forward an improved solution to I/O support mechanisms for remote job execution in Grid Computing environments. In addition, on the base of Globus Toolkit, the paper presents a reference implementation. Enhanced file I/O mechanisms will better serve the need of computing applications in Grid computing environments.

Keywords I/O, Stdin, Grid computing, Globus toolkit

1 引言

Ian Foster在其经典论文"网格剖析"中提出网格计算问题就是在动态的、多机构的虚拟组织中协调资源共享和协同解决问题,其核心是在一组参与节点(资源提供者和消费者)中协商资源共享管理的能力,利用协商得到的资源池共同解决一些问题[1]。

网格计算的本质就是在最广泛的环境下实现大规模的资源共享(large-scale resource sharing)[2]。能够提供资源或者服务在加入网格之后成为了提供者,对资源服务有需求的用户通过计算网格使用网格上的提供者提供的服务或资源,成为消费者。一个常见的网格计算的样式是对计算能力的共享。空闲的具有计算能力的资源加入网格,计算资源消费者通过网格在计算资源上面运行自己需要的计算。而在广域的网格计算环境下,计算资源直到提交计算的时候才能确定,计算常常在远离与自己相关的数据文件的站点进行。因而网格计算环境下启动管理远处计算应用就要不可避免地考虑为其提供I/O支持(包括传入输入文件、传出输出文件,标准输入、标准输出和标准出错等3个特殊文件的处理)。

本文专注于广域的网格计算环境下为运行远程计算提供支持的数据I/O机制,即研究为资源管理、进程管理器服务的数据I/O。并不是研究单纯的数据传输、管理(Data Grid)。

经过对网格计算环境下的计算I/O进行了细致的调研和考察,我们总结了以下六个I/O支持机制用户要求:

1. 计算所需的源数据文件位于由用户指定的某个站点,需要在计算之前移动到计算服务所在站点;

2. 按照用户指定要求,计算结果文件可能需要转移到用户指定的某个站点;

3. 计算所需的参数文件由用户在提交任务时设置,生成后位于提交任务站点,需要在计算之前移动到计算服务所在

*)本文研究得到国家自然科学基金项目#60173004资助。王 彬 博士研究生,主要研究领域:网格计算、并行计算、数据库系统等。徐国市 博士研究生,主要研究领域:网格计算、并行计算与并行编译。罗 引 硕士研究生,研究方向:网格计算,Web服务。许卓群 教授,博士生导师。研究方向:人工智能,网格计算,并行计算。丁文魁 教授。研究方向:系统软件、并行计算、编译系统。

站点；

4. 在计算进行的过程中在提交任务站点能够实时地看到计算任务进程的标准输出和标准出错，同时还要将这次计算进程的标准输出和标准出错信息存储在数据中心的一个文件里，以备以后研究参考之用；

5. 在计算进行的过程中用户希望能够在提交任务的本地对计算进行一定的干涉，来影响程序流，例如，可以通过标准输入；

6. 在计算进行过程中用户希望能够得到计算进程生成的中间结果文件，用以可视化实时模拟输出。

带着这些要求，我们考察了广域的网格计算环境的主要的中间件或平台提供者，如 Globus，Condor，Legion 等等。发现这些解决方案，满足了我们的大部分要求。但对于需求5和需求6在设计时都没有进行很好的支持。经过研究、设计和试验，我们提出了在网格计算环境下远程计算的 I/O 支持系统的改进的设计方案，并在目前网格计算界影响最大的 Globus 计划开发的 GlobusToolkit 所提供的 I/O 解决方案基础之上给出了实现的方案，同时进行了对比、分析。分析表明本文提出的 I/O 改进设计方案更好地满足了广域的网格计算环境下计算应用的需要。

本文作出了以下一些贡献：

1) 分析了网格环境下的启动和管理远程计算所需的 I/O 支持问题，并比较了目前网格环境的几家主要中间件系统提供者的解决方案的优缺点；

2) 基于已有解决方案的缺憾，本文提出一个改进的 I/O 支持设计方案。这个方案里，首次提出了网格计算环境的数据 I/O 中实现真正的运行时的标准输入的问题和中间结果的实时传走问题；

3) 在 Globus Toolkit 中间件的基础之上，给出了运行时的标准输入和中间结果的实时传走的参考实现方案。

2 背景

通常来说，任何一个计算应用程序的输入和输出，即涉及到的数据 I/O 至少要包括：

普通文件 I/O(文件 I/O)：输入的数据文件(可能多个)、输出的数据文件(可能多个)；

特殊文件 I/O(标准 I/O)：标准输入、标准输出、标准出错；

不包括管道、FIFO、套接字等，因为它们是视程序内部代码而定的。

正常的情况下，计算、数据和启动计算的用户终端都在同一站点。用户在终端启动一个计算应用程序后，计算应用从本地计算机读取输入数据文件，计算过程中从启动计算的终端设备的标准输入(默认是键盘)读入用户输入的信息，把程序的标准输出和标准出错信息打印到启动计算的终端设备的输出设备(一般是显示器)上，计算应用生成的输出数据文件写到本地计算机上。

而在广域网上的网格计算环境下的计算应用程序对 I/O 的需求也是如此。但由于计算环境的特殊性，实现这些需求就与通常环境下的 I/O 就大不相同了。

在网格计算环境下，计算资源直到需要计算的时候才能确定，计算常常在远离与自己相关的数据文件的站点(输入数据文件所在和输出数据文件需要保存的站点)进行，计算也远离提交控制计算应用的用户终端。在很多情况下，这是必需的。这种情况下，如何进行计算呢？

2.1 普通文件 I/O 问题

显而易见，对于解决输入 I/O 问题，即普通文件 I/O 问题，有两种解决方案[3]：

1)将计算应用移动到数据所在站点；

2)将数据移动到计算应用所在的站点。

但实际上很少采用1)，因为在存储数据的站点(数据资源)的计算能力通常不够强，而且这样效率低下(数据资源既忙于管理维护数据，又得负责计算，顾此失彼)，扩展性不好(与数据中心数据有关的计算都在数据中心计算，无法达到一定规模)。所以一般的解决方案是采用2)，将输入数据移动到计算所在站点，计算之后把结果文件移回到数据资源或指定的地点。

2.2 标准 I/O 问题

有两种传统的方法：

1)将输出信息以文件的形式存储到磁盘上，待计算结束后通过 FTP 协议或类似的协议把整个文件传回；

2)将标准输出/标准出错直接重定向到一个 TCP 流上，TCP 流与用户自己的计算机相连接。

两种方法都有缺点：第一种方法不能做到实时不断更新输出的需要。而且许多应用程序通过不断地打印一定的输出信息来说明正常运行。此外，远处的计算机有可能没有足够的磁盘空间来保存所有的输出。而且有些情况下，计算应用需要运行非常长的时间。第二种方法，则使得应用程序依赖于网络的反复变化。因为在广域的网格计算环境下，网络是不可靠的\不稳定的，所以不能指望在计算进行的过程中，计算站点和用户自己计算机之间保持一个持续的 TCP 连接(长连接)，很有可能断开。

3 相关工作

3.1 Condor

Condor 计划始于1985年，由美国 Wisconsin-Madison 大学计算机系的一个小组负责开发至今。Condor 计划的目标是在分布环境下分属不同所有者的计算资源组合之上开发、实现、部署和评估支持高通量计算的机制和策略。Condor 开发组已经开发出软件工具，使得科学家和工程师能够增加他们的计算生产量。

Condor 对于数据 I/O 的早期解决方案：如果可能的话，尽可能使用共享文件系统，否则，Condor 自动传回已发生变化的文件；Condor 在不改变程序源代码的情况下，通过重新与 Condor 库链接，使得对本地的系统调用替换成 RPC，来实现异地 I/O，而对于标准 I/O，则把它们对应成文件。例如下面是一个 Condor 提交任务的描述文件[4]：

```
# Example condor_submit input file
# (Lines beginning with # are comments)
Universe = vanilla
Executable = /home/wright/condor/my_job.condor
Input = my_job.stdin
Output = my_job.stdout
Error = my_job.stderr
Arguments = -arg1 -arg2
InitialDir = /home/wright/condor/run_1 Queue
```

可以看出它把标准输入、标准输出和标准出错分别映射到 my_job.stdin my_job.stdout my_job.stderr 三个文件。

最近 Condor 又提出了 Stork，用于在异构的网格环境解决数据放置问题(Data Placement activities)。它的一个主要目标就是支持尽可能多的存储系统和文件传输协议，目前它

已经支持 FTP、NeST (chirp)、GridFTP、SRB 和 SRM 等等。通过与 Condor 的 DAGMan 相结合，很好地解决了自动地传输输入输出数据的问题。但 Stork 解决的是常规的文件 I/O，没有触及到标准 I/O(stderr，stdin，stdout)[3]。

3.2 Legion

Legion 是由美国 Virginia 大学计算机系一个小组从1993年开始开发的一种面向对象的网格操作系统。其目标是依据一些可靠原则实现一个高度可用的、高效的和可扩展的系统。

Legion 是一种基于对象的网格操作系统。它向其他真正的操作系统一样，提供了资源管理、调度和其他系统层次的任务，但是，是在一个更加宽阔的范围上。Legion 从一开始就注意可扩展性、易于编程、容错、安全和站点自治[5]。

为了解决普通文件 I/O 的问题，Legion 引进了它自己的特殊的 I/O 调用。在 Legion 系统里，通过特殊的 I/O 方法就可以支持对数据的全局访问。Legion 提供了 shell 命令(例如"legion_cp"，是 Unix 的拷贝命令的 Legion 版本)从而实现了全局访问。首先，Legion 把二进制执行程序和计算所需的输入数据文件拷贝到选定的计算站点。接着，在 Legion 系统的监视之下计算开始了。一旦计算结束，Legion 把输出的数据文件拷贝用户指定的站点[6]。

对于标准 I/O，Legion 提供了一个 TTY 对象(legion_tty <ttyobj>)[5]。如图1所示它能够将运行时的标准输出和标准出错信息输出重导向到用户提交任务的中央控制台(或多个控制台)，可以动态地连接和断开。典型情况下，用户启动一个与当前 xterm 或窗口相关联的 TTY 对象。之后，用户在开始启动一个计算应用的时候将 TTY 对象的名字传递给计算应用。然后，计算应用的标准输出和标准出错就通过这个 TTY 对象导向到用户的窗口。同时别的窗口还可以与用户共享这个 TTY 对象，这样当计算应用运行时，这些窗口和用户的窗口都得到了输出信息。还可以把这个 TTY 对象与一个文件共享，这种情况下，标准输出和标准出错也被导向到一个文件。

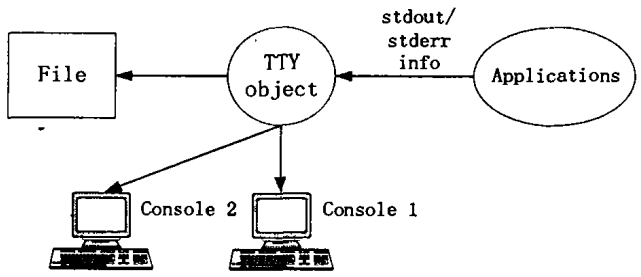

图1 Legion 里的 TTY 对象

实际运行时 TTY 对象非常有用。数十年的并行和分布式的研究已经表明了用户在运行时仍然希望通过标准出错/标准输出的信息来调试他们的程序。而且，对于一个在远处运行的程序能够获得其输出到本地，是很方便的。

3.3 Globus

Globus 计划是网格计算界最有影响的项目。是由美国 Argonne 国家实验室牵头，在南加州大学、芝加哥大学等多家大学和研究机构参与下进行的。其主要研究目标就是研发创建计算网格所需的基础技术。

Globus Toolkit 是 Globus 项目开发的能在各种平台上运行的网格计算工具软件 。Globus Toolkit 是网格计算协议和服务的主流实现，也是公认的事实上的标准(defacto standard)。

Globus Toolkit 为远程计算提供了 I/O 支持方案，可以概括为"GRAM+GASS"方案。

GRAM (Globus Resource Allocation Managment) 是 Globus Toolkit 资源管理的核心，它处理提交的资源请求以在远处执行作业任务，分配请求的资源，并管理运行着的作业任务[7]。

Globus 计划开发组认为，在广域的网格计算环境下，数据访问 I/O 具有访问形式简单，一致性要求较低(simple access patterns and coherency requirements)等特点。

基于对网格计算环境下的文件 I/O 访问的需求和特点，Globus 开发者设计并实现了一个数据移动和访问服务，称为 GASS(Global Access to Secondary Storage)[8]。GASS 就是专门为资源管理的文件 I/O 服务的。GASS 与 GRAM 的关系是：GRAM 启动和管理一个远程的计算任务，GASS 则是为远程计算的任务访问移动数据提供支持。

一个应用程序如果想在远处的资源上启动计算，必须向相应的 GRAM 发出请求，这个请求是以 RSL 格式表达的，其中指定各种参数(如可执行文件名称、参数和资源要求等等)。利用 GASS 服务的能力，就可以很轻松地扩展 GRAM API，允许对 RSL 语言中的 executable\file_stage_in \stdin\stdout \stderr\file_stage_out 等参数使用 URL。这样通过在 GRAM 的实现中加入 GASS API 的调用，就用 GASS 把一个基于 URL 的可执行文件和标准输入和其它输入数据文件取到了计算所在地的 GASS cache，并重导向 stdout，stderr 到提交任务的站点，把计算之后的结果文件转移。下面是一个向 GRAM 提交计算任务的描述文件：

```
&
(file_stage_in = (https://data_site: port/dir/of/input_data_file /
dir/of/input_data_file@incompute_site))
(file_stage_out = (/dir/of/outputdatafilein@compute_site https://
data_site:port/dir/of/output_data_file))
(executable=/dir/of/executable)
(arguments=arg1 arg2 arg3)
(stdout=https://user_submission_site:port/dev/stdout)
(stderr=https://user_submission_site:port/dev/stderr)
```

图2详细说明了这个解决方案的实现。

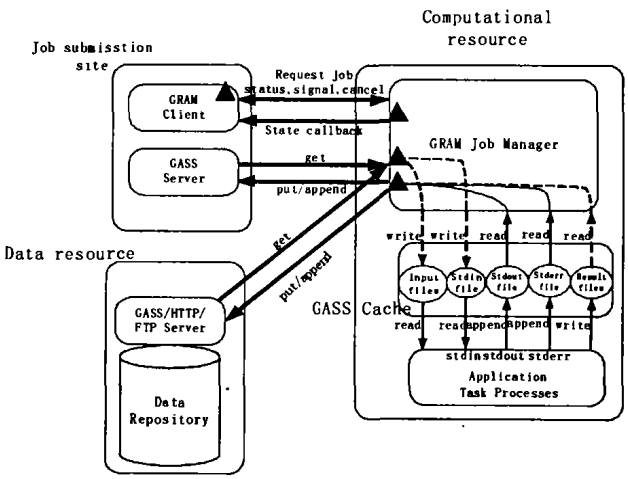

图2 Globus 的"GRAM+GASS"解决方案

注：其中，每一根连线代表一个 I/O 数据连接，箭头仅代表此连接的数据流向，而黑三角则代表这次数据通讯是由哪一方请求发起的，实线表示此 I/O 连接在计算应用进行的时候是连接着的，而虚线的连接则表示在计算应用进行时不存在，只是在计算应用进行之前或之后进行。

下面对图2中的各个 I/O 连接进行说明。

GRAM Client 与 GRAM Job Manager 之间的有两个连接，负责控制、状态信息的传输，用 GRAM 协议通讯。其中一个用于 GRAM Client 向 Job Manager 控制、查询计算应用任务，Job Manager 收到请求之后，执行请求的操作，再给与适当的回答反馈；另一个用于 Job Manager 向 GRAM Client 返回计算应用任务的状态变化；

GRAM Job Manager 与 GASS Server 之间有两个连接，负责有关的数据传输，用 GASS 协议进行通讯。其中一个用于 Job Manager 向 GASS Server 进行 get 操作，用于将计算应用的输入数据文件和标准输入取过来；另一个用于 Job Manager 向 GASS Server 进行 put 或 append 操作，用于将计算应用的输出、诊断数据及结果文件传送回去；

计算应用的标准输出、标准出错和标准输入重定向到本地 GASS Cache 的三个文件，分别用追加和只读的方式打开；

在计算应用进行时 Job Manager 定时检查对应于标准输出和标准出错相应的本地 GASS Cache 文件，已决定是否读取信息，并发送信息；

在计算应用进行之前，Job Manager 执行操作把从 GASS Server 那里 get 的输入数据写为本地文件，供计算应用任务做标准输入之用；

在计算应用进行之后，Job Manager 执行操作把计算生成的结果文件用 put 的方式传送到 GASS Server 那里。

4 已有方案的缺憾和不足

已有的解决方案基本上都实现了本文引言提出的 I/O 支持要求(1～4)，但对于要求5和要求6，没能提供足够的支持。

通过本文第3节的分析可以看出，主要的网格中间件系统对于普通文件 I/O 都给予了考虑，可以容易地实现对文件 I/O 的支持。

对于标准输出和标准出错，Condor，Legion 和 Globus 都给出了实现不同的支持。在 Legion 中，通过一个 TTY 对象把标准输出和标准出错重导向到一个或多个用户控制台和(或)多个文件。在 Globus Toolkit 中，标准输出和出错被暂时储存在 GASS 缓存里，然后传递到用户指定的一个或多个用户控制台和(或)多个文件。但在 Condor 里，用户恐怕不能期望看到实时的标准输出和出错信息，而只能等到远程的计算结束之后才能得到类似日志的磁盘文件。

但对于标准输入，已有的解决方案都没有真正意义地进行解决。在 Condor 和 Globus 中，都是把标准输入视为计算之前就可以确定的一个磁盘文件，在计算过程之中是不会改变的。而事实上，标准输入与普通的输入数据文件是有区别的，因为用户通过标准输入进行的输入是可能随着计算任务的进展而发生变化，所以不可能在计算之前就固定住了。把标准输入等同于一般输入数据文件处理，就丧失了许多程序的灵活性。

例如下面一段 C 语言程序：

```
int main(int agrc,char * * argv)
{
...
fprintf (stdout," Time consumed in parallel computing: %f seconds,%d iterations,tolerence is %f in Processor %d\n",timeuse,num,tolerence,me);
fprintf(stdout,"Please input your directions:(yes or no,yes to continue,no to quit");
if(fgets(buf,sizeof(buf),stdin)!=NULL)
{
if(strcmp(buf,"no\n")==0) exit(0);
}
...
...
}
```

在程序中，当计算进行到一定程度(可以是一定的时间之后，或进行了一定次数的迭代)之后，向标准输出打印出计算进行的进展情况(已用时间，迭代次数，收敛系数，节点号)，然后由用户决定是否继续算下去，用户在标准输入输入"yes"表示继续进行，"no"表示对计算不满意，退出，放弃这次计算。

在这里例子程序里，计算任务的标准输入在计算启动之前是不能确定的，应该视计算的进展而定，所以这样的计算程序在现有的 Globus 和 Condor 系统里是不能正常运行的。

而且缺少了实时标准输入支持的 I/O 支持系统是不完整的，因为它是单向的、半交互的。I/O 流的方向是只能从计算任务流到用户，而不能从用户流到计算任务。用户只能了解到远处计算任务的运行信息，但不能实时地对计算的流程施加影响。

在网格环境下的高性能计算周期较长，计算过程产生了大量的中间结果，是科学工作者非常需要的，既可以用于实时的计算过程仿真，有可以用于计算后的分析回放。这就需要在科学计算进行的同时对中间计算结果进行保存，并传递到指定站点(例如，用于实时可视化仿真输出)[9]。而在已有解决方案的设计中没有考虑到对中间计算结果保存和传输的支持。

5 网格计算环境下 I/O 解决方案的改进设计

考虑到已有的网格环境 I/O 支持机制的缺憾，我们提出了一个改进的解决方案。它包括两部分，即添加对运行时标准输入支持和对实时中间文件的移走的支持。

5.1 运行时标准输入的支持

首先考虑两个添加对运行标准输入的支持方法：

1) 在计算之前，创建一个稳定的从用户控制台到远处计算站点的 TCP 套接字连接。这个连接贯穿整个计算过程始终，负责把用户的输入直接传送到远处的计算站点上的计算进程。

2) 把在已有的用于传送标准输出和出错的通讯连接里添加对传送标准输入的内容。

上述1)虽然已成功地应用在并行计算环境，但不适用于广域的网格计算环境。并行计算环境，例如 MPI 等，在用户输入 MPIRUN 命令，启动 MPIRUN 进程后，通常是在每一个执行并行子任务的站点启动一个进程管理器之类的守护进程，它的一个职责就是负责捕获其看护的并行子任务的标准输出和标准出错，并把它们传送到0号节点的进程管理器，然后 0号节点的进程管理器把它们传给 MPIRUN 进程的终端上。对于标准输入，一般是 MPIRUN 进程把它传送到0号节点所在的并行子任务，再由其进行广播给其他节点[10]。这样做的一个前提是各个计算节点距离较近，有十分可靠的网络连接，例如在一个集群里或一个特定组织内部里。但是，在广域的网格计算环境，网络连接具有不确定性，不能指望用户控制台和远程的计算站点之间的网络连接在整个计算过程保持稳定和可靠。在某个时刻，网络的某个部分出现中断或 down 掉是很正常的。

上述2)是不可行的，因为它忽略了标准输入和标准输出出错的差别。在一个用于传递标准输出或出错的连接里，是由远处的计算站点负责发起数据传输的连接的，但在传输标准输入的连接里，却是由用户控制台负责发起数据传输的连接。

在排除了以上两个方案之后，我们提出了如下解决方案：

使用短连接来传送标准输入信息，一次连接只处理一次数据传输。用户控制台一发现收到用户的输入就请求与计算站点进行连接，传送完毕后关闭连接。

采用短连接避免了对网格计算环境不可靠网络的依赖。而用户控制台作为连接的起始方确保了对标准输入信息的及时传送。

我们提出的这一方案实现了用户控制台和远处站点上的计算任务完全的双向的交互。用户可以根据计算任务的实际进展来实施运行时干预。

但是这一方案也增加了系统的管理和通讯负担。

并不是所有的计算任务都需要实时的标准输入传送支持。为了减轻系统不必要的通讯和管理开销，我们可以把标准输入的实时传入设为可选，在用户的任务提交描述文件里把标准输入作为一个任选项。如果用户没有指定这一项，则默认为不需要实时标准输入支持。

5.2 实时中间结果文件的移走的支持

为了提供对实时中间结果移走的支持，我们提出了与5.1节传输标准输入几乎相同的方案：

使用短连接来传送实时的中间结果文件，远处的计算站点一旦发现了一个新的中间结果文件就负责启动一次连接，传送结束后，连接关闭。

可以在作业提交描述文件里添加一项(Middle_Result_Files)用以标注中间结果文件的名称和传送目的地(URLs)。而在远程计算站点，解析作业描述文件一旦发现这一选项，就立即启动运行时中间结果文件的传送机制。当然，这是一个可选项，而不是必选项。如果不必传送中间结果文件，用户可以选择不启动这个机制以节省系统的开销。

6 参考实现

我们选择 Globus Toolkit 作为基础给出了改进的 I/O 支持方案的实现。选择 Globus Toolkit 是因为它已经成为网格计算的事实上的标准，并且开放源代码，很多网格项目都是基于它的，有着遍及世界的广泛的开发者队伍。

6.1 对于计算应用标准输入的解决方案

需要对 GRAM 协议、RSL、GRAM Client 和 GRAM Job Manager 进行适当的添加和修改：

1) GRAM 协议中，GRAM Client 向 Job Manager 请求的通道加上对传递标准输入信息的接口。

2) RSL：如果用户想把一个事先已有的数据文件 url/dir/filename1、stdin(并且在计算过程中保持不变)作为标准输出，则采用原来的机制不变，stdin = url/dir/filename1.stdin；若用户想在计算时施加自己的控制，则把 stdin =url/dev/stdin。"/dev/stdin"这个特殊文件作为是否启动实时的标准输入机制的标志。

3) Job Manager 看到"/dev/stdin"之后，就会知道这次运算用户想在计算进行的时候得到用户的标准输入信息。在启动计算应用之前，先以追加的方式打开一个本地文件 Stdin file，然后将计算应用的标准输入重定向到这个本地文件。Job Manager 的监听服务发现有客户端的请求信息，经分析发现是标准输入类型，则马上把输入的标准输入信息追加地写到 Stdin file。

4) GRAM Client 一端：在提交任务之后，定时地查询标准输入，采用非阻塞的读操作，一旦发现有数据，则向 Job Manager 发送传递标准输入数据请求，把用户传递的标准输入信息发送过去。

6.2 对计算应用程序中间结果存储和转移的支持的解决方案

需要对 GRAM RSL Parser、GRAM Client 和 GRAM Job Manager 进行适当的添加：

1) 在 RSL 里面增加一项 Middle_Result_Files，用于在提交任务时指定。指定的内容包括中间结果文件的命名规则、转移地点的 URL、文件大小等等。然后对 RSL parser 程序进行修改，使其能够对 process_files 的信息解析出来；

2) Job Manager 看到提交的 RSL 中的关于"Middle_Result_Files"信息之后，就会启动本机制，在计算任务执行的时候，定时地检查是否有中间结果文件生成，已经发现，就把它们传送到指定的地点去；

3) GRAM Client 只需在用户有保存转移动中间结果文件要求时，在要提交的 RSL 请求中加上 Middle_Result_Files 即可。

改进后的文件 I/O 解决方案可以如图3所示。

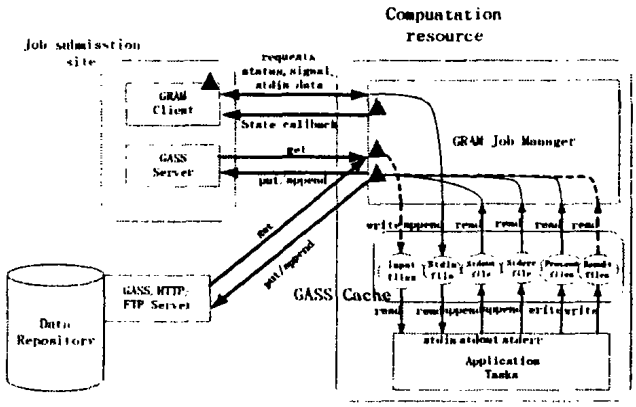

图3 改进后的资源管理文件 I/O 解决方案

如图3所示，计算应用的传输标准输入数据的各个环节的连线都变成了实线，在运行时是畅通的，可以在计算进行时实时地获得标准输入信息。另一方面，在 GASS Cache 里面添加了过程结果文件，它的 I/O 通道可以一直通到数据中心或提交站点，可以在计算的时候实时地传输计算的中间结果。

结论 网格计算环境的 I/O 支持机制的最终目的就是使得用户在远程执行的计算任务就如同或近似于计算是在本地执行一样。已有的网格计算环境中间件提供者已对大部分 I/O 用户需求提供了很好的支持，但没能解决好运行时标准输入和中间结果文件的传送。

改进后的 I/O 机制能够更好地满足广域的网格计算环境下计算应用的需要，用户可以在计算过程中施加自己的干预，控制程序流；同时，通过计算的中间结果，用户对正在进行的计算有了更多更清楚的了解，而不是对计算的进展一无所知。

参 考 文 献

1 Foster I, Kesselman C, Tuecke S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International J. Supercomputer Applications, 2001, 15(3)

(下转第37页)

由于请求到达速率 λ 的大小反映了当前被请求流媒体对象的"访问热度"。对于服务器上的不同流媒体文件，我们对 λ 取不同的值来观察算法性能。图4.a)，4.b)，4.c)和4.d)为在不同批处理间隔时，对同一个流媒体对象在 $\lambda=0.5$，$\lambda=1.0$，$\lambda=2.0$，$\lambda=4.0$时的性能。可以看出，在相同的请求到达速率时，随着批处理间隔的增大，缓存代理需要从源服务器获取的补丁数量迅速减少，算法 IC 及 ICBR 的骨干网带宽流量显著低于 OBP。随着请求速率的增加，IC 和 ICBR 的性能趋于接近。

图5.a)，5.b)，5.c)和5.d)是请求不同播放长度的流媒体时骨干所需传输的 Patch 数量。显然，IC 算法和 ICBR 算法均显著优于 OBP 并且随着请求到达速率的增加，IC 算法和 ICBR 算法趋于接近。

结论 如上我们对 IC 算法和 ICBR 算法的性能与 OBP 算法在不同播放长度的流媒体对象，不同请求到达速率和不同批处理间隔时进行了比较。当采用批处理补丁调度策略时，骨干网络上传输的数据为 RC 数据和 PC 数据，对于给定的流媒体对象(T,b 与 λ 给定)，其 RC 流量固定为 T/I。上述结果表明：1)对流媒体对象的后缀采用 IC 及 ICBR 算法进行动态缓存和释放大大降低了骨干传输补丁块的块数，从而有效地降低了骨干带宽的消耗，其性能显著优于 OBP 算法。2)当缓存因容量有限而采用 ICBR 算法对缓存进行管理时，由于骨干网链路上仍需传输部分重复的补丁数据，因此所需传输的补丁块数略大于 IC 算法。ICBR 算法牺牲了部分骨干流量，但是却获得了更高的缓存效率。3)当请求到达速率较大($\lambda>5.5$) 时，会导致每个批处理间隔内均有至少一个请求到达时，此时算法 IC 的性能则等于算法 ICBR 的性能，但都显著优于 OBP 算法。4)(9)式为在每个补丁窗口内平均所需从服务器获得的补丁块数，由于请求到达速率较大时 ($\lambda>5.5$)，算法 IC 的性能接近于算法 ICBR，因此，(9)式也表明了此时系统所需的平均缓存容量。

上述结论说明，对于在存储和传输都占据较大资源量的流媒体对象在 Internet 上的传输，采用带前缀缓存的批处理补丁调度策略并在缓存代理上采用 IC 或 ICBR 算法对补丁数据块进行有效的缓存不仅可以有效降低骨干网带宽和流媒体服务器的负载，同时在保持较低的客户端延迟的前提下，也有效地降低了客户端资源需求(减少了客户端需同时接收流的并发通道个数)和提高了客户端 QoS，从而可更好地支持 VOD 等视频应用。

存储型 A/V 数据是实现 VOD 等应用的数据载体，它是指将视频节目进行编码压缩后存放在媒体服务器上，以按需传输的模式进行的静态的，可按照流式播放的多媒体数据。由于存储型 A/V 数据的本身固有特性和基于包分发的 IP 网络的体系结构及带宽和服务器资源的限制，通过端对端链路的组播来传输存在很大困难，因而一定程度上阻碍了这些应用的发展。而结合应用层组播算法和缓存复制协议的内容分送网络 CDN 进行流媒体数据的分发可以获得骨干网络资源的大量节省，同时也可以有效降低流媒体服务器的负载和提高客户端 QoS，从而更好地支持 VOD 等视频应用。下一步，我们将专注于：(1)研究适合存储型 A/V(Stored A/V Data)数据传输的应用层组播协议及更高效的缓存算法；(2)研究多缓存代理协作体结构和相关算法；(3)结合应用层组播协议和缓存算法研究 CDN 流媒体分发体系结构；(4)设计实现流媒体缓存代理服务器。

致谢 本文工作得到了中国科学院数学所周龙骧教授、中国科学院应用数学研究所安鸿志教授的帮助与指导，其中一些关键结果的推导是与安鸿志教授共同讨论得出的，在此向他们表示衷心的感谢。

参考文献

1 Hua K A, Sheu S. Skyscraper Broadcasting: A New Broadcasting Scheme for Metropolitan VOD systems. In: Proc. of ACM/SIGCOMM, 1997, 27(4): 89～100
2 Viswanathan S, Imielinski T. Metropolitan Area Video-on-Demand Service using Pyramid Broadcasting. Multimedia Systems, 1996, 4(4): 197～208
3 Dan A, Sitaram D, Shahabuddin P. Scheduling Policies for an On-Demand Video Server with Batching. ACM Multimedia, Oct. 1994. 15～23
4 Hua K A, Cai Y, Sheu S. Patching: a multicast technique for true on-demand services. In: Proc. of ACM Multimedia, Sept. 1998. 191～200
5 Cai Y, Hua K A, Vu K. Optimizing patching performance. In: Proc. of ACM/SPIE Multimedia Computing and Networking, Jan. 1999. 204～215
6 Sen S, Rexford J, Towsley D. Proxy prefix caching for multimedia streams. In: Proc. of IEEE INFOCOM'99, 1999
7 Almeida J, Eager D, Vernon M. A hybrid caching strategy for streaming media files. In: Proc. SPIE/ACM Conf. on Multimedia Computing and Networking, Jan. 2001
8 Hofmann M, Ng E, Guo K, Paul, Zhang H. Caching techniques for streaming multimedia over the internet: [Technique Report BL011345-990409-04TM]. Bell Labs. Apr., 1999
9 Sen S, Rexford J, Towsley D F. Proxy Prefix Caching for Multimedia Streams. In: Proc. of IEEE INFOCOM, 1999(3): 1310～1319
10 向哲，钟玉琢，冼伟铨. 一种基于周期合并策略的流调度算法. 软件学报，2001，12(8)：1183～1189
11 White P P, Crowcroft J. Optimised Batch Patching with Classes of Service. ACM SIGCOMM Computer Communication, 2000, 30: 21～28
12 Frossard P, Verscheure O. Batch Patch caching for Streaming Media. IEEE Communications letters, 2002, 6(4)

(上接第18页)

2 Foster I, Kesselman C. The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1999
3 Kosar K. Managing and Scheduling Data Placement (DaP) Requests. in Talk - Condor Week, March, 2002
4 Couvares P, Tannenbaum T. Condor Tutorial. In: 1st EuroGlobus Workshop, June 2001
5 Legion: The Grid OS Architecture and User View. in 2nd International Global Grid Forum Meeting, 2001
6 White B, Grimshaw A, Nguyen-Tuong A. Grid-based file access: The Legion I/O Model. In: Proc. of the Ninth IEEE International Symposium on High Performance Distributed Computing, Pittsburgh, PA, August 2000. IEEE Computer Society Press
7 Czajkowski K, et al. A Resource Management Architecture for Metacomputing Systems. In: Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, 1998. 62～82
8 Bester J, et al. GASS: A Data Movement and Access Service for Wide Area Computing Systems. Sixth Workshop on I/O in Parallel and Distributed Systems, May, 1999
9 Wang Bin, et al. A Grid-computing-based Solution to Science&Engineering Computing and its Implemenation. ISTM/2003: 5th International Symposium on Test and Measurement, 2003, 6(1): 581～586
10 Ralph B, et al. Components and interfaces of a process management system for parallel programs. Parallel Computing Volume: 2001, 27(11): 1417～1429
11 Grimshaw A S, Wulf W A. The Legion Vision of a Worldwide Virtual Computer, Comm. of the ACM, 1997, 40(1)
12 Thain D, et al. The Kangaroo Approach to Data Movement on the Grid. In: Proc. of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10), San Francisco, California, August 2001. 7～9