

分布并行计算在网格环境下的一种新实现*

王振宇, 林伟伟, 齐德昱

(华南理工大学 计算机科学与工程学院, 广东 广州 510641)

摘 要: 介绍了开放网格服务结构 OGSA 的标准实现 Globus Toolkit 3 的系统结构、编程模型以及分布式并行的支撑环境,并以大规模矩阵相乘为例给出了该环境下分布并行计算的实现方法和试验结果。讨论了任务分布、系统通信和容错机制等关键问题。

关键词: 分布并行计算; OGSA; Globus 工具

中图法分类号: TP393. 03 文献标识码: A 文章编号: 1001-3695(2006) 01-0206-04

A New Implementation Method of Distributed Parallel Computing on Grid Computing Environments

WANG Zhen-yu, LIN Wei-wei, QI De-yu

(College of Computer Science & Engineering, South China University of Technology, Guangzhou Guangdong 510641, China)

Abstract: Grid computing provides some new solutions for many complex problems. A standard implementation of OGSA, called Globus Toolkit (GT3), is discussed in its architecture, programming model, and construction of distributed parallel environment. Some of key issues, such as job distributing, system communications and fault tolerance are analyzed. An example of large-scale multiplication of matrixes is presented, and the test result is attached.

Key words: Distributed Parallel Computing; OGSA; Globus Toolkit

1 引言

网格计算开始于科学研究领域,于 20 世纪 90 年代中期提出,用来表述一种适用于高端科学和工程的分布式计算体系结构。它涉及计算密集型、数据密集型、知识密集型和协作密集型等应用领域,如天文计算、生物医学数据分析、高能物理的联合计算、知识集成等^[1]。当前,网格计算在商业计算上得到应用,如 IBM 的“按需计算”是以网格计算为基础。在分布式计算技术方面,CORBA, J2EE / RMI, DCOM 等尽管早已得到应用,但仍存在一些本质问题,未完全达到平台无关的目标,也不容易穿过防火墙,未实现真正意义的互连互通。从 1997 年起,开放源码的 Globus Toolkit 2 成为网格计算的事实标准,在世界范围内上千个网格中应用。2002 年由 Globus 小组和 IBM 提出的开放网格服务体系(Open Grid Service Architecture, OGSA)成为 GGF 标准。该标准统一了 Globus 和 Web 服务,向外提供网格服务^[2]。Globus Toolkit 3(GT3) 是 OGSA 的首个实现,建立在 WSDL, SOAP 和 WS-Inspection 等万维网标准上,支持分布式状态管理、轻量级检查和发现以及异步通知。

OGSI(Open Grid Service Infrastructure) 是构建 OGSA 的基础设施^[3],网格服务规范在 Web 服务的基础上定义网格服务的标准接口和行为。本文以 OGSA/ GT3 为背景,分析实现分布并行计算的关键问题,并给出了在 GT3 环境下矩阵乘的实现和结果分析。

2 GT3 软件系统结构

GT3 软件系统结构由 GT3 核、基础服务、用户定义服务组成^[3],如图 1 所示。GT3 核是网格服务架构的基本模块,包括 OGSI 参考实现:提供 OGSI 定义的接口、消息和网格行为,方便同 Web 服务引擎和宿主平台互操作; 安全基础设施:提供基本的网络安全,包括消息和传输层的保护、端到端的互相认证和授权; 系统级服务:日志服务、系统管理、句柄解析、路由等服务。

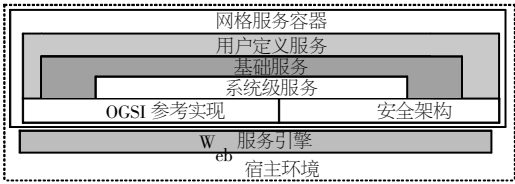


图 1 GT3 系统结构模型

基础服务建立在 GT3 核之上,提供信息服务、数据管理服务、作业执行服务等。用户定义服务作为应用层服务,用来发掘 OGSI 引用实现和安全基础架构提供的功能。这些服务可以和其他高层服务协同工作,如元调度器、资源分配管理器、协作监控服务等。GT3 引入网格服务容器,该运行环境提供网格服务持续管理、生命周期管理、实例管理等功能,可借助现有的宿主运行环境实现,如 J2EE EJB 容器。Web 服务引擎负责管理从网格服务客户端到服务端的 XML 消息,包括解码、拆包(Unmarshalling)、类型映射、把调用分发给服务实例。GT3 选用 Apache Axis 作为 Web 服务引擎。GT3 在安全基础架构中支持传输层安全和消息级安全,提供系统级服务。

2.1 服务端框架结构组件模型

GT3 服务端框架的参考实现模型如图 2 所示。Apache Axis 框架作为 Web 服务引擎处理 Web 服务, 如 SOAP 消息处理、JAX-RPC 处理器和 Web 服务配置等。Globus 容器框架提供实例句柄管理、实例仓库、生命周期管理等功能, 管理有状态的 Web 服务。Pivot Handlers 创建 Web 服务实例和调用服务上的操作。RPCURIHandler 是 Axis 框架的一个 Java 提供者的特殊情形的实现, 它处理封装的消息, 通过服务实例句柄与实例仓库通信, 发现 Web 服务, 调用服务上的操作。通常, Web 服务在工厂所在的宿主环境下创建, 有时因为负载平衡或访问权限等问题需要在不同的宿主机环境下创建 Web 服务(对客户透明), 由 GT3 框架处理路由问题。

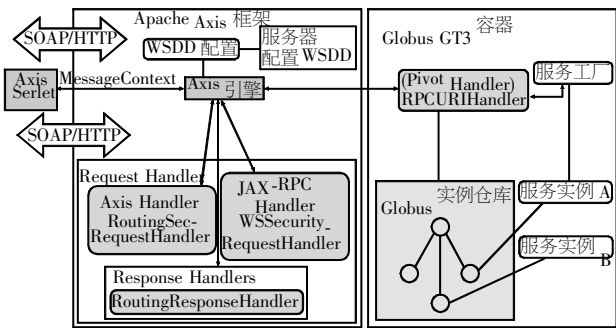


图 2 GT3 服务端框架的参考实现模型

2.2 客户端框架结构组件模型

GT3 以 JAX-RPC 作为客户端编程模型, 缺省实现是 Apache AXIS, 该框架提供运行时 JAX-RPC 引擎, 负责从客户端到来的消息处理和分发, 如图 3 所示。

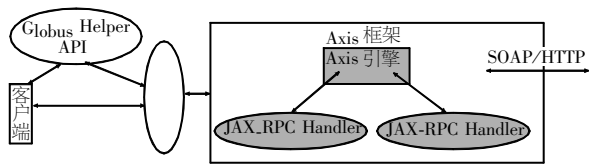


图 3 客户端框架的系统结构组件

3 Globus Toolkit 编程模型

GT3 内核采用 Java 实现, 基于 Web 服务和 Java 编程环境提供 OGSi 功能。GT3 编程模型包括网格服务和客户端编程。

3.1 网格服务编程模型

网格服务编程模型如图 4 所示, 包括网格服务基础接口 (Base) 及实现、网格服务回调机制、操作提供者、工厂及回调、服务数据。

GridServiceBase 接口继承 GridService, ServiceProperties, GridServiceCallback 接口, 其中只有 GridService 接口通过 WSDL 向外暴露给客户端, 其他接口用来控制网格服务的行为。这些基础类实现服务数据管理、操作提供者管理、服务实例生命周期管理等。GT3 提供 GridServiceBase 接口的两个缺省实现:

(1) GridServiceImpl。它构成临时的网格服务的基类, 这些服务由 OGSi 工厂机制创建。

(2) PersistentServiceImpl。它是持久的网格服务的基类, 这些服务不是 OGSi 工厂机制创建的。

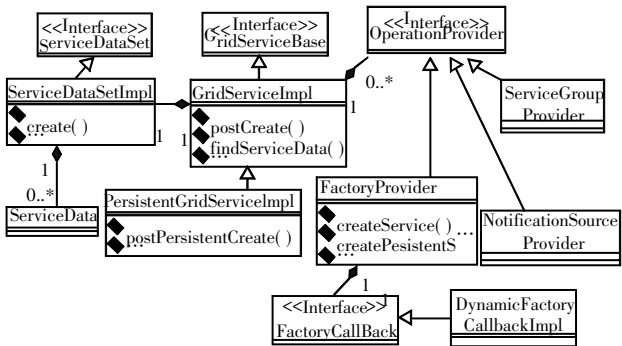


图 4 网格服务编程模型

创建网格服务有两种方式: 服务继承 GridServiceImpl 或者 PersistentServiceImpl, 实现向客户端暴露的接口; 服务继承 GridServiceImpl 或者 PersistentServiceImpl, 使用操作提供者实现向客户端暴露的接口。

Java 不支持多继承, GT3 引入操作提供者 (Operation Providers) 作为网格服务操作的动态代理模型。操作提供者在部署时被配置到服务中, 并且使 GridServiceBase 在运行时能够动态增加操作提供者, 动态配置服务行为。这样, 开发者在操作提供者中编写业务逻辑, 由网格服务的实现类把客户端的服务请求委托给操作提供者。GT3 为开发者缺省提供三种设计模式, 即 Factory Provider, ServiceGroup Provider 和 Notification-Source Provider。

GT3 提供工厂回调机制为增加定制的工厂类创建特定的网格服务, 例如定制的工厂可以与 EJB Home 协作在远程宿主环境下创建 EJB 服务, 如同本地创建一样。服务数据是服务向外暴露的状态信息, 服务数据集 (ServiceDataSet) 为网格服务实例存放所有的服务数据, 服务数据元素通过 findServiceData 或订阅操作向外暴露。

3.2 客户端编程模型

GT3 客户端以 JAX-RPC 编程模型为基础。缺省时 GT3 提供 Apache AXIS 客户端框架、工具和运行时, Globus 为客户端提供帮助类, 简化 GSH 到 GSR 的解析和 GSR 的自省。通过构造服务定位器, 对 JAX-RPC 编程模型的扩展如图 5 所示。

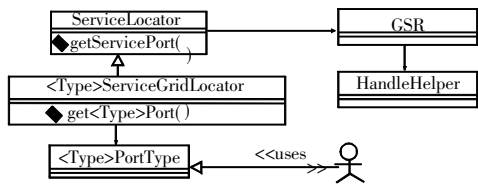


图 5 客户端编程模型

客户程序从注册表获得网格服务的 GSH, 将该句柄传递给 ServiceGridLocator, 后者调用句柄解析服务, 把 GSH 解析为 GSR, 创建一个代理或桩, 该代理向外暴露 JAX-RPC 格式的网格服务接口; 客户端用 JAX-RPC 模型调用代理提供的方法。JAX-RPC 规范为客户端提供三种调用机制: 静态桩、动态代理和 DII (Dynamic Invocation Interface)。

4 基于网格的分布并行计算

由多个提供网格服务的计算节点组成虚拟并行机, 完成由网格客户端应用提交的并行计算任务。每个网格节点是并行计算节点, 提供一个或一些可以并行执行子任务的并行网格服

务。网格客户端应用通过 Globus Toolkit 并发地调用这些网格服务实例。由于每个并行节点用 Globus Toolkit 构建,支持从微机到巨型机的任何机种,从 Windows 到 UNIX 和 Linux 等,方便异构机间的网络并行,充分地利用现有的网络资源完成分布并行计算。

与传统意义上的并行机和网络并行实现方式相比,利用网格基础设施作为分布并行计算的编程环境的优点是:充分利用互联网上的资源实现大型问题的分布并行计算,解决处理机、操作系统等方面的异构性,充分利用现有的网络设施以及实现的简单性。

4.1 几个关键问题

(1) 任务分布。一些复杂的计算任务能够分解成相互协作的子任务,每个子任务的实现通过定义一个网格服务来完成。网格客户端启动多个线程,并发调用网格服务来并行执行计算任务。每个网格服务可以运行在物理网格计算节点上,即每个任务被分布到一台计算机。基于网络的分布式并行计算环境把 Globus 网格节点扩展到广域网络中,利用空闲的网络资源提升应用的运行速度。

(2) 系统通信。系统通信涉及两个方面: 网格服务之间通信,包括并行计算节点之间的通信,主要完成并行计算的各网格节点上的网格服务之间的数据通信和同步信息交流; 网格服务与客户之间通信。网格服务提供 JAX-RPC 机制、通知机制和服务数据,实现网格服务之间通信和网格服务与客户的通信。通知机制有 Pull 方法和 Push 方法^[4,5], Push 方法允许数据和通知一起传输,通信效率较高,对于并行计算这种性能优先的应用,建议采用这种方式。

(3) 容错机制。基于网络的分布式计算环境由若干台计算机用 Internet 连接而成,每台计算机是自治的,与集中式系统相比系统更为强壮,一个单元或资源(软件或硬件)的故障不影响其他资源的正常功能。但由于 Internet 通信的不可靠性,需要提供容错措施。理论上可以通过全局的检查点实现网络的容错^[6],但实现难度大,较为可行的方法是通过应用程序自身来实现容错。Globus 使用监听心跳的措施跟踪每个网格节点的失效情况,达到容错的目的。

4.2 实现分布并行计算的方法

在 GT3 下开发分布并行应用步骤如下:

(1) 分解并行任务,创建多个可并行的网格服务。一个网格服务完成一个或多个并行任务。

(2) 通过继承 Thread 类或实现 Runnable 接口来创建一个分布线程。

(3) 分布线程的 run() 方法中创建一个网格服务实例,即部署并行计算代码到网格节点中。

(4) 创建网格客户应用程序。它负责管理启动多个分布线程。

(5) 多个并行网格服务执行并行任务。网格服务之间以通知机制来完成并行任务之间消息和数据的通信。

(6) 网格服务节点返回计算结果给网格客户应用。

5 大规模矩阵相乘计算实例及测试结果

5.1 大规模矩阵相乘的实现

采用分治算法求解两个矩阵相乘问题: $C = A \times B$, 其中 A

和 B 分别是 $n \times n$ 和 $n \times n$ 矩阵, C 是 $n \times n$ 矩阵。矩阵相乘的串行计算的复杂性为 $O(n^3)$, 而在异构并行计算环境下以并行程序来完成的计算复杂性依赖处理机的个数、任务映射策略和数据通信量等。为实现矩阵相乘并行操作,将其进行划分并映射到不同处理器。下面将矩阵 A 和 B 分别划分为行块子矩阵和列块子矩阵的并行计算方法。假设 $n = \overline{n} \times p$, 其中 p 为处理机个数。

$A = \begin{bmatrix} A_0^T & A_1^T & \dots & A_{p-1}^T \end{bmatrix}^T, B = \begin{bmatrix} B_0 & B_1 & \dots & B_{p-1} \end{bmatrix}$, 这时 $C = (C_{i,j}) = (A_i \times B_j)$, 其中 $C_{i,j}$ 是 $\overline{n} \times \overline{n}$ 矩阵。每台处理机计算出一个 $C_{i,j}$, 最后合并 p 台处理机的结果。简化的矩阵相乘的网格服务实现如下所示:

```
public class MatrixImpl extends GridServiceImpl implements MatrixPortType
{
    public float[ ][ ] MatrixImpl( float[ ][ ] a, float[ ][ ] b)
    {
        ...; super( " Sub Matrix Service" );
        ... // 矩阵赋值;
        for ( i = 0; i < n; i ++ ) { // 矩阵相乘;
            for ( j = 0; j < n; j ++ ) { c[ i ][ j ] = 0;
                for( k = 0; k < n; k ++ )
                    c[ i ][ j ] += a[ i ][ k ] * b[ k ][ j ];
            }
        }; return c; // 矩阵相乘 // 返回子矩阵相乘结果
    }
}
```

简化的矩阵相乘客户端应用程序如下所示:

```
class ThreadMatrix extends Thread{ // 任务分布线程
    MatrixServiceGridLocator MatrixServiceLocator = new MatrixServiceGridLocator();
    ThreadMatrix( URL GSH, long starttime, float [ ][ ] x) {
        try{ ...;
            locator = MatrixFactory. createService();
            MatrixServiceGridLocator MatrixLocator = new MatrixServiceGridLocator();
            MatrixPortType Matrix = MatrixLocator. getMatrixService ( locator);
            ... // 矩阵赋值
        } catch( Exception e) { ... }
    }
    public void run() // 在线程体中执行服务
    {
        try{ // Get current value through remote method MatrixImpl '
            float [ ][ ] r = Matrix. MatrixImpl();
        } catch( Exception e) { ... }
    }
    public float[ ][ ] get() { return r; } // 返回子矩阵相乘结果
}

public class MatrixClient // 执行并发任务的客户端程序
{
    public static void main( String[ ] args) {
        try{
            ... /* 矩阵初始化, 根据客户机数量分解矩阵, 确定子任务的大小 * /
            ...; // 取得多个网格服务句柄
            URL GSH = new java. net. URL( args[ 0 ] );
            URL GSH1 = new java. net. URL( args[ 1 ] );
            ...; // 定义多个线程
            ThreadMatrix t2 = new ThreadMatrix( GSH1, start);
            ThreadMatrix t1 = new ThreadMatrix( GSH, start); ...
            t1. start(); t2. start(); ...; // 启动线程, 并行运行服务
            t1. join(); t2. join(); ... // 等待线程执行结束
            float [ ][ ] out1 = t1. get(); float [ ][ ] ou2 = t2. get();
            // 返回服务执行的结果
            ...; /* 合并多个任务(服务)执行的结果, 输出矩阵相乘结果 * /
        } catch( Exception e) { ... }
    }
}
```

5.2 测试环境与结果

测试环境是表 1 中计算机组成的异构网络环境。求解问题运行时间如表 2 所示, 求解加速比如图 6 所示。

表 1 系统配置表

配置 机器	CPU	内存	操作系统	网格计算环境
主机 1	Intel Pentium 2.8GHz	512MB	RH Linux 9.0	Globus Toolkit 3.2.1
主机 2	Intel Celeron 2.2GHz	256MB	Windows 2000	Globus Toolkit 3.2.1
主机 3	Intel Celeron 2.2GHz	256MB	Windows 2000	Globus Toolkit 3.2.1
主机 4	Intel Celeron 2.2GHz	256MB	Windows 2000	Globus Toolkit 3.2.1

表 2 矩阵相乘运行时间表(单位:s)

n 主机数	400	500	1 000	2 000
单机环境	1.813	4.187	58.188	802.750
两台主机	3.125	4.216	32.163	414.157
三台主机	2.946	3.672	24.782	271.154
四台主机	2.845	3.491	18.167	211.056

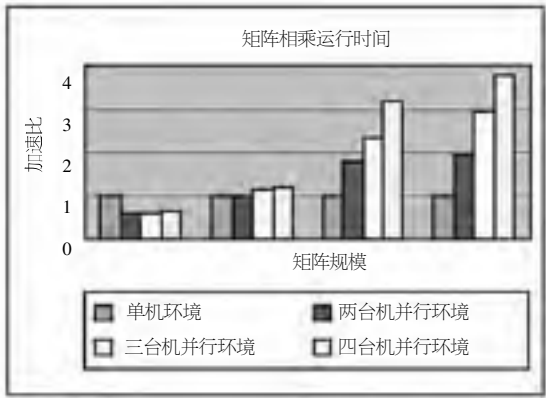


图 6 矩阵相乘性能加速比图

结果表明,随着问题的规模增大,在 GT3 环境下,并行求解速度比单机求解的速度有很大的提高。由于问题求解扩展到以 Globus 网格服务节点为并行节点的广域网中,系统具有很好的可伸缩性,为提高问题求解的速度提供了新的途径。

6 结论

本文介绍了 GT3 软件的系统架构、编程模型,讨论了 GT3 网格计算环境下实现分布式计算的方法和任务分布、系统通信、容错机制等问题。开放网格结构能够充分利用互联网上分散的、闲置的、异构的资源,为实现超大问题的大规模分布并行计算开辟新的途径,并且能够简化并行计算的实现。以大规模矩阵相乘为例给出 GT3 环境下实现分布并行计算的方法。对试验结果分析表明,随着问题复杂性增加,GT3 下分布式并行的性能得到提高。

参考文献:

[1] an Foster, Carl Kesselman. 网格计算(第 2 版)[M]. 金海,等. 北京:电子工业出版社,2004.

[2] 徐志伟,冯百明,李伟,等. 网格计算技术 [M]. 北京:电子工业出版社,2004.

[3] Joshy Joseph, Craig Fellenstein. Grid Computing [M]. Prentice Hall, 2003.

[4] Grid Services Programming and Application Enablement [EB/OL]. <http://www.ibm.com/redbooks>, 2004.

[5] Chao-Cheng Wen, Yuan-Sun Chu, Kim-Joan Chen. Causal-ordered Communication of Grid Computing on Internet[C]. Proceedings of the IEEE International Conference on Communications, 2004. 20-24.

[6] Jin Liang, Tong WeiQin, Tang JianQuan, *et al.* A Fault-tolerance Mechanism in Grid [C]. Proceedings of the IEEE International Conference on Industrial Informatics, 2003. 21-24.

作者简介:

王振宇(1967-),男,副教授,博士,研究方向为分布式计算、操作系统等;林伟伟(1980-),男,博士研究生,主要研究方向为网络计算、计算机体系结构。

(上接第 205 页)

参考文献:

[1] essier Russell, Burleson Wayne. Reconfigurable Computing for Digital Signal Processing: A Survey[J]. Journal of VLSI Signal Processing, 2001, 28(2): 7-27.

[2] Compton Katherine, Hauck Scott. Reconfigurable Computing: A Survey of Systems and Software[J]. ACM Computing Surveys, 2002, 34(2): 171-210.

[3] Kessal L, Abel N, Demigny D. Real-time Imaging Processing with Dynamically Reconfigurable Architecture [J]. Real-time Imaging, 2003, 57(9): 297-313.

[4] M Khalid, J Rose. A Hybrid Complete-graph Partial-crossbar Routing Architecture for Multi-FPGA Systems[J]. ACM/SIGDA International Symposium on FPGAs, 1998, 32(4): 45-54.

[5] J Vuillemin, P Bertin. Programmable Active Memories: Reconfigurable Systems Come to Age[J]. IEEE Trans. VLSI Syst., 1996, 4(1): 56-69.

[6] Bondalapati K, Prasanna V. Reconfigurable Computing: Architectures, Models and Algorithms[J]. Current Science, 2000, 78(7): 828-837.

[7] Bondalapati K. Modeling and Mapping for Dynamically Reconfigurable Hybrid Architectures[D]. Ph D Thesis, University of Southern California, 2001.

[8] Bondalapati K, Prasanna V. The Hybrid System Architecture Model (HySAM) of Reconfigurable Architectures[R]. Technical Report, Department of Electrical Engineering-System, University of Southern California, 1998.

[9] 刘勇,何永孟,杨根庆. CDMA 通信中一种新的基于 MMSE 多用户检测的自适应功率控制算法[J]. 电路与系统学报, 2003, 8(3): 104-108.

[10] 贾鹏,李松. FPGA 的动态可重配置技术[J]. 军事通信技术, 2001, 22(2): 52-54.

[11] Hauck S, Fry T W. The Chimaera Reconfigurable Functional Unit [C]. IEEE Symposium on Field-Programmable Custom Computing Machines, 1997. 87-96.

[12] Chameleon Systems, Inc. CS2000 Advance Product Specification [EB/OL]. <http://www.chameleonsystems.com>, 2000/2004.

[13] ilinx Application Note[EB/OL]. <http://www.xilinx.com>, 1996/2004.

作者简介:

刘勇(1978-),男,湖北武汉人,博士研究生,主要研究方向为嵌入式系统、信号处理、可重配置计算;李华旺(1973-),男,江西人,计算机研究室主任,副研究员,博士,主要研究方向为信号处理、计算机系统;尹增山(1971-),男,山东人,项目部部长,研究员,博士,主要研究方向为信号处理、卫星导航;杨根庆(1952-),男,上海人,主任,总设计师,研究员,博士生导师,主要研究方向为计算机系统、信号处理。