# 一个基于 Java 虚拟机的分布式计算模型*)

齐德昱　谢景明
(华南理工大学计算机科学与工程学院　广州 510641)

**摘　要**　针对单个 JVM 的性能缺陷问题,分析了实现分布式 JVM 的关键技术,提出了一个基于 Spaces 的分布式虚拟机集成模型,该模型将执行代码和数据分离,通过异步协作机制和动态装载类技术,将多个 Java 作业透明地调度到不同的 JVM 资源上并行执行,实现了单一系统映象。

**关键词**　类装载器,单一系统映象,虚拟机,分布式计算

## A Distributed Computing Model Based on Java Virtual Machine

QI De-Yu　XIE Jing-Ming
(College of Computer Engineering and Science, South China University of Technology, Guangzhou 510641)

**Abstract**　It is difficult to improve the performance of a single java virtual machine. Some key technologies on distributed java virtual machine are analyzed firstly. Then a distributed java virtual machine model based on spaces is proposed to solve the above problem, which separates data from executable codes. Jobs are scheduled to resources in asynchronous mode and executed by dynamically loading classes, which implement the single system image in a distributed computing environment.

**Keywords**　ClassLoader, Single system image, Java virtual machine, Distributed computing

## 1　引言

Java 的跨平台性促进了基于 Java 应用的发展,而为 Java 程序提供运行环境的 Java 虚拟机(JVM, Java Virtual Machine)正成为 Java 系统性能的关键。改善 Java 运行环境的途径有很多,当前的研究大多集中在改善 JVM 性能上,例如提出了 JIT、HotSpot 等编译优化技术。事实上,与处理机类似,除了改善 JVM 本身外,如果将多个分布的 JVM 集成在一起,形成一个多 JVM 相互协作的环境,这也是提高 Java 性能的手段。特别是在单个 JVM 的性能提高达到极限时,采用分布式 JVM 是必然的趋势。这方面的研究在国内外都是处于初步阶段,其理论研究的空间很大,对发展我国自主的 Java 环境技术,占领未来的 Java 市场具有重要的战略意义与实践意义。

国内外对 Java 分布式环境的研究主要有两大方向,一类是 Java 多线程程序被分布到扩展的 JVM 上计算,如 cJVM[4]、JESSICA2[3]等;另一类是使用标准的 JVM 对 Java 多线程程序进行分布并行处理,如 JDSM[5]、JavaSplit[2]等。这些研究的路线集中在引入新的 API,对 Java 字节码的多线程重写或者对 JVM 的内部进行并行化改造等方面。

为了以一种灵活的机制实现复杂的分布式计算,我们提出了一个新的面向 Java 虚拟机环境的分布式计算模型——DJVMS(Distributed Java Virtual Machine Based on Spaces)。该模型利用 Spaces 的异步通信机制对 JVM 资源进行有效的协作,支持将多个 Java 作业透明地分发给不同的 JVM 资源,并通过类装载器 JVMSpace 实现新任务的动态执行。

## 2　关键技术

JVM 能够屏蔽不同宿主平台在物理硬件和操作系统上的差异,向上提供与实现无关的操作接口。利用该特性,DJVMS 的目标是以 JVM 为中间介质,利用网络上分散、独立、异构的资源进行分布式计算,从而提供一个低代价高性能的解决方案。这比把一个 Java 程序分配到一个单一的 JVM 执行要复杂得多。以下几个因素是实现 DJVMS 的关键。

(1)JVM 集群模型与协议方面:需要研究采用何种拓扑模型,何种通信/互操作模式来对各个 JVM 资源进行有效的组织与协作。

(2)Java 字节码的迁移与调度:要实现任务的分布并行执行,就必须将相应的 Java 字节码迁移到目标 JVM 上,这种迁移需要耗费带宽和时间。如何实现对任务的合理调度是提高系统性能的关键。

(3)JVM 运行时数据的分布共享:需要研究如何使 JVM 运行时的数据保持逻辑上的统一,以保证计算的正确性。

(4)系统的稳定性:由于通信网络和计算资源具有不可预测性,需要一种容错机制来保证某个 JVM 资源的加入或者崩溃,不会影响系统的正常运转。

为了实现系统的高可移植性和可扩展性,DJVMS 不对 JVM 以及 Java 应用程序进行任何的更改,而是设计一种新的机制来实现单一系统映象。主要的思想是:把作业实现计算功能的代码与计算所需的源数据分离,源数据分成独立可并行执行的任务,从而避免对 Java 字节码的并行化改造;并且引入异步通信机制协调各个 JVM 资源的计算,向上屏蔽 JVM 资源的分布性,以避免对 JVM 内部的并行化改造。

*)基金项目:粤港关键领域重点突破项目(2005A10307007),广东省重大科技专项(2005B10101003)。齐德昱　教授,博士生导师,主要研究方向为分布式计算、计算机体系结构;谢景明　博士生,主要研究方向为网格计算,计算机体系结构。

## 3 DJVMS 模型

耶鲁大学的 David Gelernter 在 Linda 系统提出了元组空间(Tuple Space)的思想,该系统利用 Space 为并行计算提供全局的通信缓存[1]。DJVMS 模型对此进行了扩展,通过多个 Space 将分布在各个地方的 JVM 资源连接起来(参考图 1)。Space 在 DJVMS 中的作用是为作业提供公共的存储和共享服务。应用的参与者集中在分布的 Space 中,作业提交者把作业提交给 Space,任务计算节点从 Space 获取任务信息,参与者之间的协作由各个 Space 的异步通信机制实现。Space 服务为应用参与者提供相同的读、写、取、通知等操作作业的接口,即应用参与者获取 $Space_1$ 服务与获取 $Space_2$ 服务的方法相同。

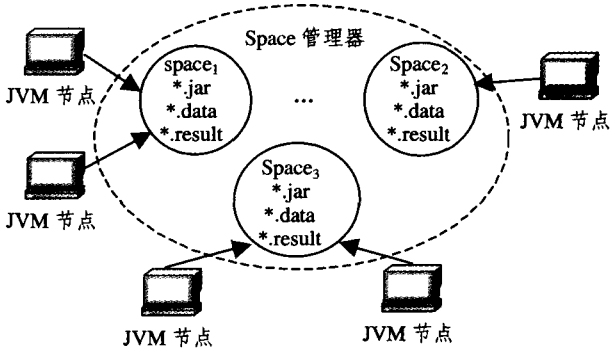


图 1 基于 Spaces 的 JVM 计算模式

**定义 1** 将一个安装有标准 JVM 且可以通过网络访问的资源称为 JVM 节点。

从类型上分,JVM 节点涵盖了计算机、内嵌芯片、移动设备等安装有 JVM 且有计算处理能力的资源,它们都能够参与 Space 的协作计算。

**定义 2** 在 DJVMS 模型中,一个 Java 作业按下列方式分解:

$JOB=(JarSet, DataSet, ResultSet)$,其中:

$JarSet=\{jar_1, jar_2, \cdots, jar_m\}$

$DataSet=\{data_1, data_2, \cdots, data_n\}$

$ResultSet=\{result_1, result_2, \cdots, result_n\}$

这里将实现计算的功能和提供计算的数据分离,*JarSet* 集合包含提供计算具体实现的 jar 文件,jar 文件是 Java 压缩包,它包含了计算实现所涉及的各种相关类;*DataSet* 集合包含为 jar 文件提供计算所需的 data 源数据文件,每个 data 文件代表一个计算任务;*ResultSet* 集合包含计算结果 result 文件,每个 data 任务对应一个计算结果。

从功能上看,DJVMS 由 Space 管理器,Space 服务和 JVM 节点三部分组成。分布在多个物理位置不同的 Space 服务由一个 Space 管理器以平面的方式统一管理。不同 Space 服务的关系是平等的,能够直接进行相互通信。当一个 Space 服务 A 出现过载时,它首先向 Space 管理器发出作业迁移请求,然后 Space 管理器根据平台内的其它 Space 服务的负载情况,返回最合适接受作业迁移的 Space 服务 B 的信息。这样 A 就可以作为一个作业提交者向 B 发出作业执行请求。这种内部的分布式作业迁移机制对于 JVM 节点来说是透明的。此外,根据 JVM 节点的分布情况,Space 管理器能够动态地创建或者撤销 Space 服务。即 Space 服务能够随着分布式计算环境的变化而动态变化。例如,在 DJVMS 环境建立的初期,若 A 地的 JVM 节点数量比较少时,可以由位于 B 地的 Space 服务为这些节点提供任务分配服务。但随着 A 地越来越多的 JVM 节点参与,为了减少网络上的通信开销,Space 管理器将采用评估算法,判断专门在 A 地新建立一个 Space 服务的必要性。相反,由于某些原因,A 地参与的 JVM 节点可能会大幅减少,为了降低 Space 管理器的管理花费,Space 管理器也会评估是否有必要撤销 A 地的 Space 服务。对于一个即将被撤销的 Space 服务,当其完成所负责的任务后,原属该服务的 JVM 节点将被重新部署到相邻地区的其它 Space 服务。因此,DJVMS 是一个能够动态适应环境变化的计算模型,实现了通过内部的动态调整提高计算的效率。

Space 服务起到了连接 JVM 节点和 Space 管理器的中间桥梁作用。一个 Space 服务向下可以管理多个 JVM 节点,而每个 JVM 节点只受一个 Space 服务的管理。参与 JVM 资源首先连接平台的 Space 管理器,Space 管理器根据当前各个 Space 服务的负载、JVM 资源的处理能力以及地理位置等因素,将 JVM 资源引导到相应的 Space 服务。当某个 Space 服务出错或者过载,Space 管理器能够根据调配算法重新将该服务辖下的 JVM 节点切换到其它 Space 服务。

每个 Space 服务能够管理若干个不同参与者提交的 Java 作业集,并以多线程的方式提供任务调度服务(参考图 2)。Space 服务主要包含了两种类型的请求队列,一种是 JVM 节点对 jar 文件下载的请求,另一种是 JVM 节点对 data 文件执行的请求。为了提高资源的负载均衡能力,不同的 Space 服务允许包含相同的 jar 文件。但为了避免重复计算,不允许将属于同一个作业集的 data 数据重复放在不同的 Space 服务。Space 服务根据作业的要求与 JVM 节点的性能情况,决定 JVM 节点应参与计算的作业。任务计算节点首先需要下载 jar 文件,然后根据 Space 服务的调度安排,再从 Space 服务下载 data 数据进行计算。当 JVM 节点完成任务处理后,将结果返回给 Space 服务。

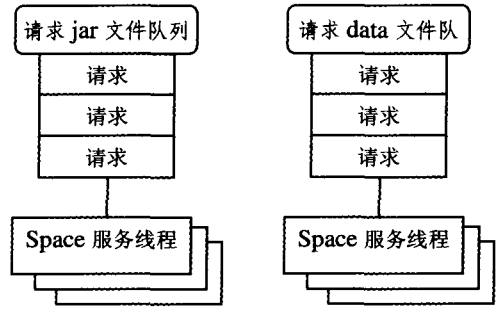


图 2 Space 服务的逻辑处理过程

## 4 分布式类装载器的实现

### 4.1 类装载器的体系

类的装入过程分为装入、链接和初始化三个过程。装入阶段负责查找所需的类文件,并将其以字节码形式装入;链接阶段负责验证字节码、准备类所需的数据结构以及装载被该类引用的其它类;初始化阶段执行所有包含在类中的静态初始块。经过上面的三个过程后,类被完全装入,处于可使用状态。

JVM 缺省的类装载器只能够从本地文件系统装载类文件,并不适合直接应用于 DJVMS 环境。为了使 JVM 节点能从 Space 服务下载类文件并动态运行,本文专门设计了

JVMSpace 类装载器(见图 3)。

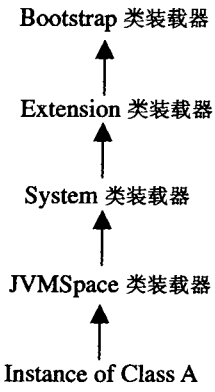


图 3 JVMSpace 类装载器的所在层次

类装载器是按委托层次的形式组织的:Bootstrap 类装载器为根节点,其它类装载器都有一个父装载器。Bootstrap 负责装载 JVM 提供的基本运行类,如 java. lang. * 等;Extension 类装载器负责装载 JRE 扩展目录中的类;System 类装载器负责装载 CLASSPATH 变量所指定的类;JVMSpace 类装载器是 System 类装载器的儿子,负责装载从 Space 服务发送过来的类文件,并在运行时动态地调用执行。

采用委托模型是出于安全的考虑,以避免加载不安全的类。当类装载器 JVMSpace 被请求装载某个类时,首先在缓存查找该类是否与曾经装入的类相同,若相同则返回上次返回的类,否则委托 System 装载器进行装载,如果 System 装载器还不能够装载,则向 Extension 类装载器发出装载请求,这是一个递归过程。如果最后所有的上层父装载器都无法装载该类,则由 JVMSpace 类装载器在自己所指定的路径上进行装载。

### 4.2 主要实现方法

由于 JVM 节点参与分布式计算的时间是不限定的,为了避免 JVM 节点重复向 Space 服务提出同样的下载请求,加重 Space 服务器的负担,影响整个分布式计算的效率,Space 服务首先将相关的 jar 文件下载到 JVM 节点,然后再由 JVMSpace 类装载器动态地从本地文件系统装载所需的类文件。Space 服务的 jar 文件分为两种,一种是属于业务计算的 jar 文件,被保存在一个专门的 SpaceLib 目录下,另一种是业务计算所依赖的第三方 jar 文件,被保存在%JAVA_HOME%/jre/lib/ext 目录下。

自定义类装载器主要有两种途径,一种是继承 CLASSLOADER 类,另一种是继承 URLCLASSLOADER 类。URLCLASSLOADER 相对来说功能更强大,不但能够访问本地文件系统,而且还能够调用网络上的类文件和包文件。JVMSpace 类装载器通过对 URLCLASSLOADER 的继承,实现了对类的动态装载。而且在预先不了解对象的情况下利用反射机制获得对象的内部信息,灵活地调用执行类的各种函数。

在进行分布式计算时,jar 文件的业务处理功能对所有参与任务计算的 JVM 节点是透明的,JVM 节点通过 JVMSpace 类装载器装载所需的类,并调用所需的 data 源数据进行处理。下面给出 JVMSpace 类装载器的部分实现代码。

```
import java. net. * ;
import java. io. * ;
import java. lang. reflect. * ;
public class JVMSpaceClassLoader extends URLClassLoader{
  public JVMSpaceClassLoader(URL [] urls){
      super(urls);
  }
  …其它类构造函数
public Class loadClass (String className, boolean resolve) throw
ClassNotFoundException{
  Class c=findLoadedClass(className); //检查该类是否已经被装
  载
  if(c==null){
   try {
      c=super. loadClass(className,resolve);
      }catch (Exception e){
        e. printStackTrace();
      }
   }
   if (c==null)
      throw new ClassNotFoundException (className);
   return c;
  }
  …其它类装载函数
  //利用反射机制执行装载类的 main 函数
public void InvokeMain(Class c,String [] args)throws Exception{
  Class [] mainParamType={args. getClass()}; //获得参数的类型
  Method m=c. getMethod("main",mainParamType); //搜索类的
  main 主函数
  m. setAccessible(true); //禁止对反射对象进行访问检查
  int mods=m. getModifiers(); //获得反射对象的标识值
  if( m. getReturnType ( )! = void. class || ! Modifier. isStatic
  (mods)|| ! Modifier. isPublic(mods)){
    throw new NoSuchMethodException("在类中找不到主函数");}
  try {
    Object [] mainParams={args}; //获得 main 函数所传递的参数
    m. invoke(null,mainParams); //执行 main 函数
    } catch(IllegalAccessException e){
    System. out. println("不能成功执行函数");
     }
  }
…其它利用反射机制调用装载类函数的代码
…
}
```

JVM 节点完成的计算结果 result 文件可以由 XML 数据格式进行保存,通过定期或主动两种方式将结果返回给 Space 服务。各个作业提交节点可根据自己所定义的数据规范 Schema 对存放在 Space 服务的结果数据进行验证读取。同时也可以采用其它数据格式,这对参与计算的 JVM 节点和 Space 服务来说是透明的。

**结束语** 本文对分布式 JVM 的集成模型和协议进行了研究,旨在提供一种新的面向 Java 的透明分布式计算环境,以迎接日益增长的 Java 应用新需求和新一代网格体系结构的建设。DJVMS 利用异步通信协作机制和动态类装载技术,建立了一个具有可扩展性和可伸缩性的分布式计算环境,使安装有 JVM 的资源能够灵活地加入该计算环境,并在逻辑上形成一个虚拟的整体,透明地完成任务计算。本文的下一步工作是把 JVM 节点作为一个标准的网格服务,这些服务能够高效地组合在一起为应用程序提供网格执行环境。

## 参 考 文 献


1 Gelernter D. Generative communication in Linda, ACM Transactions on Programming Languages and Systems, 1985, 7(1): 80～112

2 Factor M, Schuster A, Shagin K. JavaSplit: A Runtime for Execution of Monolithic Java Programs on. Heterogeneous Collections of Commodity Workstations. In: Proceedings of IEEE International Conference on Cluster Computing, 2003. 110～117

3 Zhu W, Wang C, Lau CM. JESSICA2: a distributed Java Virtual Machine with transparent thread migration support. In: Proceedings of 2002 IEEE International Conference on Cluster Computing, 2002. 381～388

4 Aridor Y, Factor M, Teperman A. cJVM: a single system image of a JVM on a cluster. In: Proceedings of the International Conference on Parallel Processing, 1999. 4～11

5 Sohda Y, Nakada H, Matsuoka S. Implementation of a portable software DSM in Java. In Java Grande/ISOPE'01, 2001. 163～172