基于 Python 和 C／C++的分布式计算架构

高 远

（上海大学，上海 200333）

摘 要：综合 Python 和 C/C++的优势建立了一种半自动部署的分布式计算架构。其主要创新之处在于：运用 SWIG 技术对 Python 和 C/C++进行原子算法级别的桥接，并通过 RPYC 进行节点间的传递同步。该架构在开发上多维度地运用了各种现有技术，充分发挥了 Python 作为脚本语言的优势而又不失运算效率。在同步算法上，该架构引入了一个以运算步骤数位核心的、用于同步的数据读写适配器算法，易用且有效率地同步操作。

关键词：SWIG；RPYC；原子算法

中图分类号：TP301 **文献标识码**：A **文章编号**：1672-7800(2012)006-0017-02

0 引言

Python 作为一种被广泛应用的脚本编程语言，其语法简单、语义清晰、平台兼容性好、模块丰富。同时由于 Python 作为“胶水语言”的设计初衷，因此经常被用于连接 C/C++等编译型语言编写的模块。近年来随着分布式编程模式被广泛应用，Python 的并行化也日益被人们所重视。

从实现方式来看主要有 3 类：①以 ParallelPython 和 RPYC 为代表。其使用纯 Python 作为系统实现，主要通过模块文本的传递和 Pickle 序列化参数传递等方式实现并行；②以 MPI 的 Python 封装作为实现，在编程方式上沿袭 MPI 的模式；③以 Cython 为代表，通过自动化编译实现 Python 模块的并行化，其内在机制在于通过编译器加入 OpenMP 支持。

实际使用中，这 3 者各有利弊：第一种方式运行效率相对较低，但可以充分发挥 Python 的优势；第二种方法虽然效率上优于完全基于 Python 的分布式实现，但受到 MPI 已有模式的约束，不能充分发挥 Python 内嵌数据结构的效能；第三种方式效率相对较高，但语法上比较晦涩，同时只能多线程并行，不能多 CPU 并行。出于实际应用的需要，本文拟建立一种综合各方优点、基于 Python 的分布式编程架构。该架构主要解决如下 3 个问题：其一，数据的高效传递；其二，混合 Python 和 C/C++代码；其三，数据同步问题。

脚本语言在数据传递上的效率上虽不如 C/C++的 MPI，但总体传输时间和传输效率仍旧和需传输的信息量成正比例关系，这主要是由于 Python 内建的传输机制主要仍然基于封装 socket 等传统的传输方式。因此随着机器性能和网络传输速度的提升，传输效率的高效性对基于 Python 的分布式系统是有保障的。混合代码运算主要是为保证分布式计算架构在原子级别运算的效率。Python 作为脚本语言语法易用、语义简单，相应地由于其弱类型和动态解释特征而造成其在原子级别运行效率相对不足。Python 是按模块化运作的，因此使用 C/C++等高效编译型语言编写 Python 扩展模块以提升原子算法的运行效率。为便于在分布式系统上自动达成这一目标，故使用 SWIG 作半自动编译。

1 SWIG 的半自动化编译

SWIG 是一个旨在帮助使用 C 或者 C++编写的软件能与其它各种高级编程语言进行嵌入联接的开发工具，其既支持脚本语言，也支持非脚本语言。原子算法对执行效率的要求是最高的，故使用 C/C++作为原子算法的开发语言，并通过 Python 调用 SWIG 生成接口文件，由 SWIG 处理生成 C/C++代码，进而通过 C/C++编译器编译生成模块文件以供 Python 解释器导入（import）。这一过程主要由 4 个步骤组成：设计算法、明确输入输出、序列化复杂对象、Python 调用 shell 命令编译 SWIG 接口文件、部署到相应节点位置。

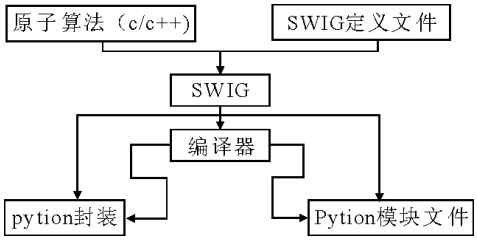

图 1 SWIG 半自动化编译架构

作者简介：高远(1984－)，男，上海人，上海大学硕士研究生，研究方向为分布式计算。

1.1 设计算法,明确输入输出

原子算法的本质是一个对输入数据的变换处理过程,设计上要功能单一化。同时由于最终编译过程是发生在运算节点的,因此原子算法的整体篇幅也要进行控制,以降低传输时间。以计算阶乘为例:

首先编写 SWIG 接口文件,main.i:

```
/* main.i */
%module Sam
%{
/* Put header files here or function declarations like below */
extern int fact(int n);
%}
extern int fact(int n);
```

为了实现阶乘功能,用 c 语言编写实现文件 main.c

```
/* File : main.c */
int fact(int n)
{
  if (n <= 1) return 1;
    else return n* fact(n-1);
}
```

1.2 序列化复杂对象

Python 支持类实例等复杂的对象,Python 定义了 7 种基本数据类型:对象、整型、浮点型、字符串、元组、列表和字典。对于将 C/C++数据类型转为 Python 对象可以使用 Python C API:Py_BuildValue,其原型为:PyObject* Py_BuildValue(const char *format,...);若要将 Python 对象转为 C/C++数据类型可以使用 Python C API:PyArg_Parse,其原型为:int PyArg_Parse(PyObject *args, const char *format,...)。为实现 Python 分布式计算的 OOP 特性,对于对象等复杂数据类型在架构上也必须加以支持,而以上方法只是对简单对象(数据类型)直接的转换,对于复杂对象的转换可以采取两种方法:其一是将 Python 对象的定义和运行状态序列化,以字符串方式处理,用 exec()加以运行;其二是用 import 指令导入相应定义和状态。为了在 C/C++程序段内可以使用 Python 的对象/类定义和过程,可以使用 Python C API 在 C/C++程序段内运行 Python 解释器。在 C/C++程序段内使用 Python 对象有一定效率损失,故因慎重对待,尽可能少用。

1.3 Python 调用 shell 命令编译生成 SWIG 接口文件

对于 c 语言的 SWIG 命令比较容易(默认接口文件和生成的文件在同一目录下):swig-python main.i,进而通过 Python 引入 os 模块,使用 os.system()函数调用对其进行编译,生成模块封装文件 main_wrap.和 Sam.py(该文件和 SWIG 定义文件内的模块名同名)。

1.4 部署到相应节点位置

由于模块封装文件(main_wrap.c)和模块实现文件(main.c)都是源码文本文件,因此可以简单地通过 RPYC 传递分发到各个运算节点,并进行同步编译器编译。

SWIG 生成的模块封装并不针对某种特定编译环境,这里使用 gcc 作为编译环境,其编译命令行为:

```
gcc-shared main.c main_wrap.c
  -ID:  Python25  include -LD:  Python25
Libs -lPython25 -o _Sam.pyd
```

同上,使用 os.system()函数调用对其进行编译,生成相应的 Python 扩展模块文件。

2 通讯层的实现

通讯层的作用旨在解决数据的同步问题和数据的高效传递。通过对不同 Python RPC 的比较,RPYC 易用而不失效率,符合架构需要。

为实现数据的同步需使用读写适配器(read/write adapter)算法,首先引入读写适配器的核心数据结构,其数学描述如下:序偶 L:L=<processor_id,obj_name,obj_value,step,r|w>。其中 processor_id 为不同进程的标示,obj_name 为需要读写同步的对象名称,obj_value 为对象值,step 为步骤数,r|w 为读写指示。读写同步操作的实现如下:若系统按照 C/S 实现则在服务端建立一个“读写适配器”,适配器按照 step 的先后按照 r|w 进行对象和对象值的传递派发;若系统实现时没有明确的服务端,则向读写目标发送相应的序偶,由目标解释器进程处理。

3 分布式模块的优化

原子算法是使用 C/C++实现,并使用 Python 调用的。由于 Python 有 GIL,因此线程安全可以保证。而同时在分布式模块的编写中可以进一步地使用 OpenMP 对多线程进行优化,同时可以基于分批和分离 TCP 连接的方法进行算法优化。

参考文献:

[1] Python Software Foundation. Python 编程语言的首页[EB/OL]. http://www.python.org/,2011.

[2] Vitalii Vanovschi. Parallel Python 主页[EB/OL]. http://www.parallelpython.com/.

[3] Tomer Filiba. RPYC 主页[EB/OL]. http://rpyc.sourceforge.net/,2012.

[4] DALCIN L D. Parallel distributed computing using Python[J]. Advances in Water Resources,2011(9).

[5] Cython 主页[EB/OL]. http://www.cython.org/,2012.

[6] BEAZLEY D M. Automated scientific software scripting with SWIG [J]. Future Generation Computer Systems,2003(5).

[7] 徐咏梅. Python 网络编程中的远程调用研究[J]. 电脑编程技巧与维护,2011(18).

[8] 蒋锦林,谢华,肖寅东. 提高 Python 处理网络数据速度的算法研究[J]. 自动化信息,2011(9).

(责任编辑:杜能钢)