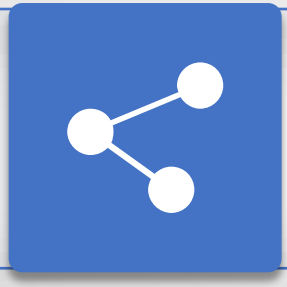
分布式算法仿真平台基本介绍

清华大学自动化系 Tsinghua University

严 虎 Mail：yanh20@mails.tsinghua.edu.cn Wechat：w2624507753
王奕凡 Mail：wangyifa17@mails.tsinghua.edu.cn Wechat：Wyf_1995_



目的

建立一套能模拟分布式环境的虚拟软件系统，用来仿真硬件平台，为用户提供分布式程序的测试环境，验证分布式算法的可行性以及对分布式系统中的潜在问题进行发掘与研究。

研究意义

仿真硬件平台

该虚拟系统在构建方式、通信以及任务执行方面与硬件系统基本一致，能有效地模拟硬件平台。

验证系统功能

搭建软件平台后，系统能对各种未来硬件平台需要执行的任务进行测试，验证分布式算法在系统中的可行性与效率。

高容错率

软件系统能提供相对更高的容错率，在实现硬件系统前，相应软件系统的构建能更好地发掘系统中的错误，避免硬件连接时产生不可挽回的损失。

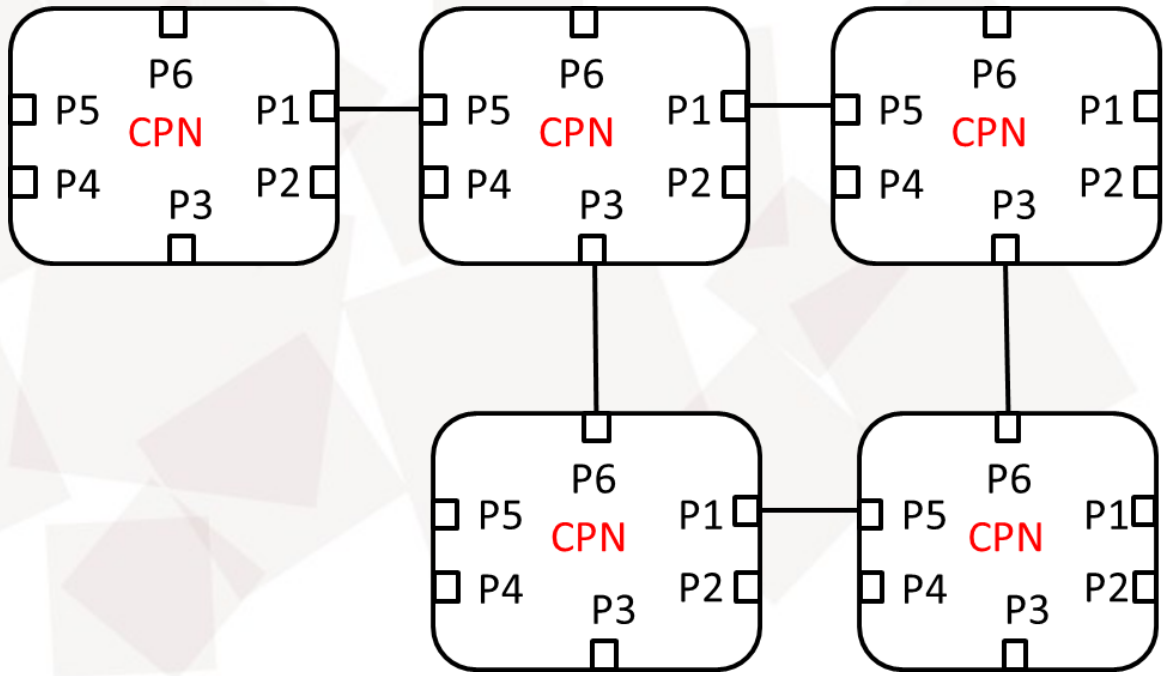
规模与可靠性

可利用云服务器大幅度提高软件系统模拟的节点个数，避免因规模扩大而产生的问题。

平台简介

硬件平台

- 平台由计算节点CPN通过数据线相互连接而成。每个计算节点具有独立的CPU、存储单元，六个可以和其他计算节点进行双向数据通信的端口（P1，P2，...P6）以及若干与外围设备进行通信的I/O端口。



[5个CPN节点连接组成的分布式计算平台]

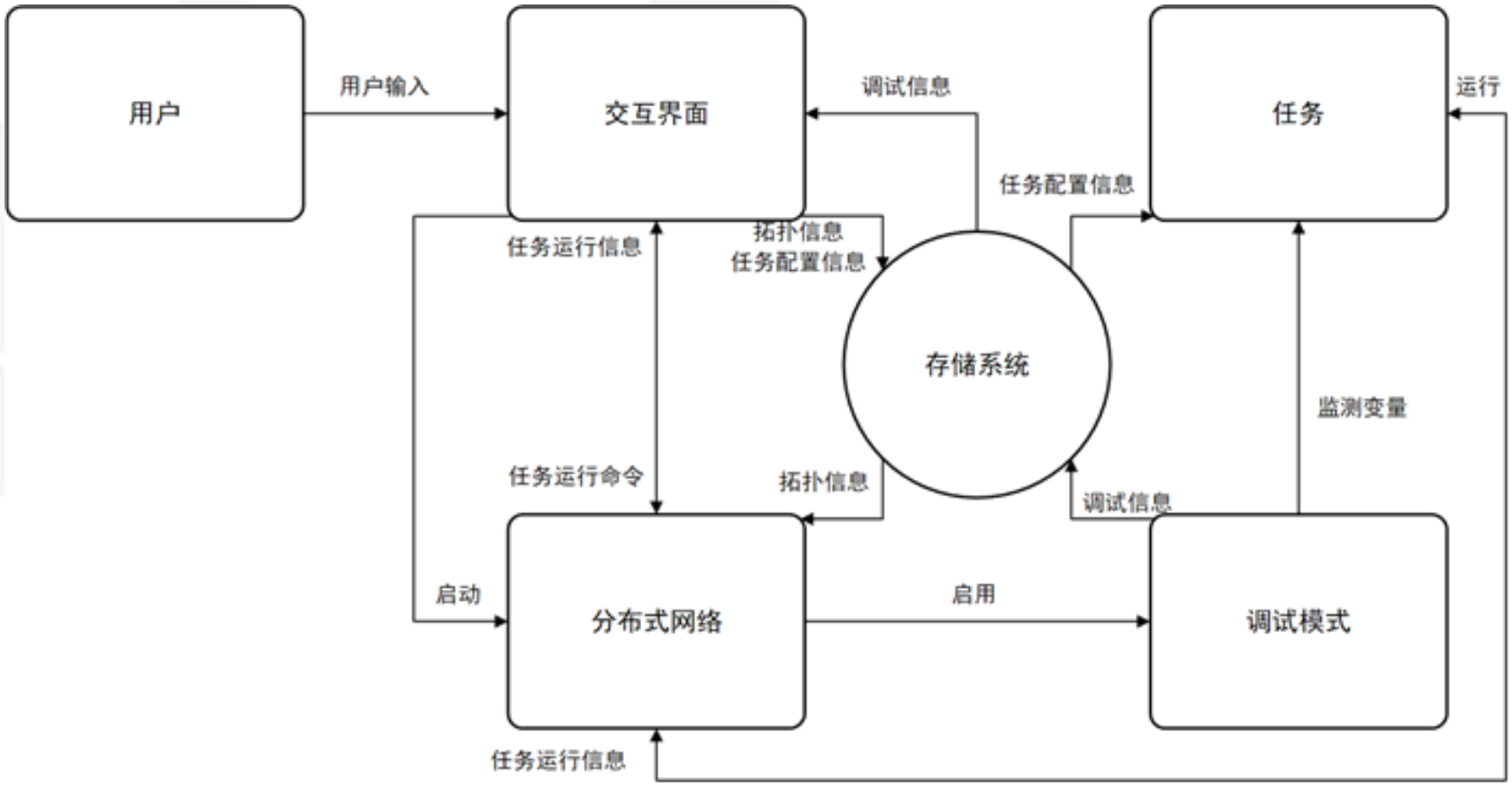
硬件环境

Windows/ Linux/ Mac OS X操作系统计算机一台

软件环境

Python3.7.6 64位

系统架构


用户
用户输入
交互界面
调试信息
任务
运行
任务配置信息
任务运行信息
拓扑信息
任务配置信息
存储系统
监测变量
任务运行命令
拓扑信息
调试信息
启动
分布式网络
启用
调试模式
任务运行信息

模块简介

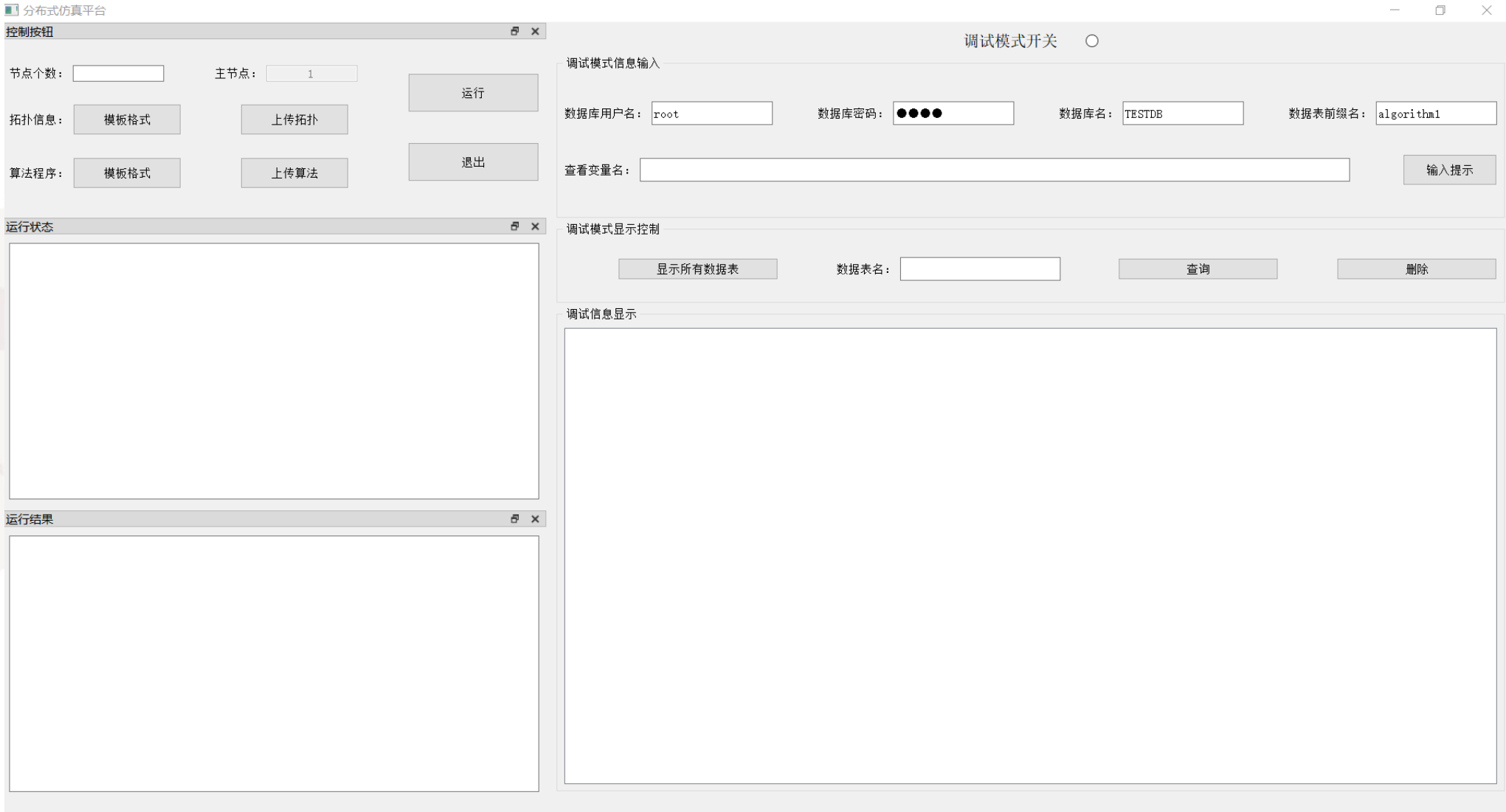
- 节点进程：利用进程仿真节点，通过导入的拓扑信息(JSON格式)进行初始化，存储有进程ID、进程信息、路由表等信息，主要分为通信、外部模块、任务计算三个模块。

- 通信模块：每个进程有6个TCP服务器的子线程，节点利用这些服务器来与邻居进行通信，每个服务器具有独立的端口。设计目的：CPN特性保证节点邻居上限为6，服务器一一对应的通信方式模拟实际的硬件连线。

- 外部交互模块：每个进程有1个与外界交互的任务服务器子线程，用来接收外界任务请求，执行相应任务并返回数据。

- 任务计算模块：通过导入外部的具体分布式算法algorithm.py，可在仿真平台中运行该算法，验证该算法的正确性与可靠性。

平台界面


分布式仿真平台
控制按钮
节点个数：
主节点：
1
运行
拓扑信息：
模板格式
上传拓扑
退出
算法程序：
模板格式
上传算法
运行状态
运行结果
调试模式开关
调试模式信息输入
数据库用户名：
root
数据库密码：
数据库名：
TESTDB
数据表前缀名：
algorithm1
查看变量名：
输入提示
调试模式显示控制
显示所有数据表
数据表名：
查询
删除
调试信息显示

用户输入信息

- 节点个数
- 拓扑信息(JSON格式)
- 算法程序(可执行的py程序)

平台输出信息

- 运行状态：输出算法运行的情况，各个节点的信息反馈，算法错误信息以及用户算法中需要输出的自定义信息(用作调试)。
- 运行结果：主节点收到的每个节点的输出信息，若算法有误则输出为空。

任务流程

拓扑信息

拓扑信息

- 拓扑信息于节点初始化时给出，用来提供给节点其邻居情况
- 拓扑为JSON数据格式，具体为每个节点描述组成的数组。
- 单个节点需要包含如下六个属性：
  1. "ID": 节点编号
  2. "IP": 节点IP
  3. "PORT": 节点各服务器的端口数组
  4. "adjID": 邻接节点数组
  5. "adjDirection":邻居节点对应的端口编号
  6. "datalist": 初始化数据

```
{
    "ID": 2,
    "IP": "localhost",
    "PORT": [
        10007,10008,10009,10010,10011,10012,10013
    ],
    "adjID": [
        3,
        1,
        4
    ],
    "adjDirection": [
        1,
        5,
        3
    ],
    "datalist": []
},
```

拓扑示例

拓扑模板格式

```
[
    {
        "_comment": "_comment中的文字仅作为注释，实际导入中可删去。拓扑请使用json数据格式进行导入，具体形式为每个节点描述组成的数组。",
        "_comment2": "单个节点具体格式如下所示，ID为1~n的编号；IP为该节点IP；PORT为给该节点指定的6个通信服务器与1个任务服务器的端口数组；adjID为该节点邻接节点数组，要求对称性；adjDirection为邻居节点对应的端口编号，需在1~6中取值；datalist为该节点初始化时需传入的数据，格式自定。接下来为实例，指定一个正方形中的连接关系"
    },
    {
        "ID": 1,
        "IP": "localhost",
        "PORT": [10000, 10001, 10002, 10003, 10004, 10005, 10006],
        "adjID": [2, 3],
        "adjDirection": [1, 2],
        "datalist": {}
    },
    {
        "ID": 2,
        "IP": "localhost",
        "PORT": [10007, 10008, 10009, 10010, 10011, 10012, 10013],
        "adjID": [1, 4],
        "adjDirection": [3, 2],
        "datalist": {}
    },
```

拓扑示例


拓扑输入
"datalist": {}
},
{
"ID": 3,
"IP": "localhost",
"PORT": [10014, 10015, 10016, 10017, 10018, 10019, 10020],
"adjID": [1, 4],
"adjDirection": [4, 1],
"datalist": {}
},
{
"ID": 4,
"IP": "localhost",
"PORT": [10021, 10022, 10023, 10024, 10025, 10026, 10027],
"adjID": [2, 3],
"adjDirection": [4, 3],
"datalist": {}
}
]
确定
取消

拓扑格式生成

- 节点规模较小时，可以采用手动添加的方式，直接生成各个节点的拓扑
- 节点规模大时，可根据具体拓扑提炼规律，通过程序生成拓扑，再进行JSON格式化

算法设计

算法设计要点

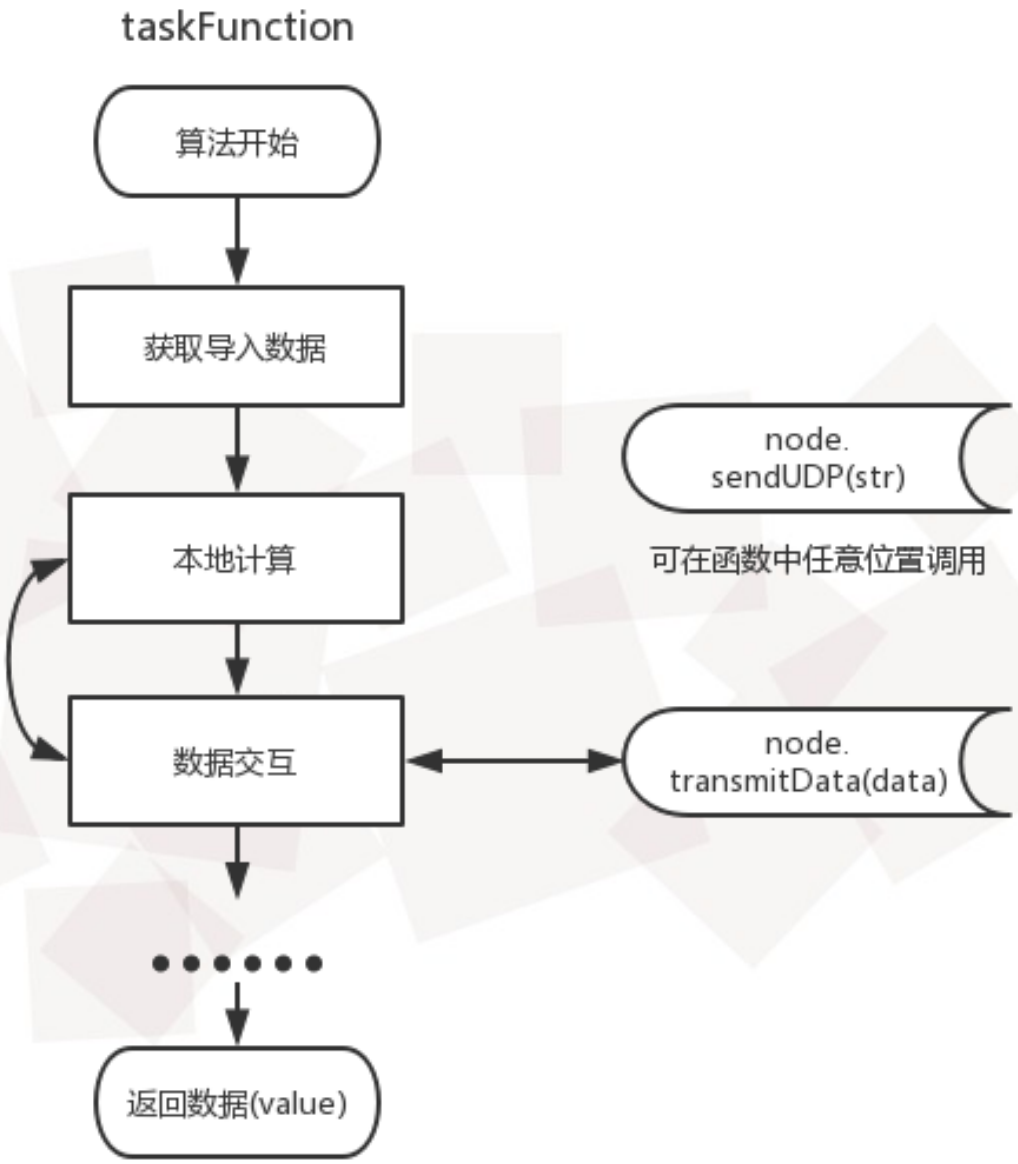
- 无全局信息
- 各节点运行算法相同
- 算法适应性要求高，需能在任意拓扑运行
- 算法为同步算法

algorithm.py语法规定

所有编程语句均应符合python语法，程序包括4部分

1. (可选)模块引入
2. (可选)定义辅助函数
3. 定义任务主函数。语法为 def taskFunction(node, id, adjDirection, datalist): 其中形参node代表节点类，ID表示节点编号，adjDirection代表节点邻居对应的方向，datalist表示初始化时节点的输入信息
4. 算法主体。用户可以通过node.transmitData(data)进行数据交互，data为需要交互传出的数据，接收的数据为2元数组，第一部分为邻接节点对应的端口编号数组，第二部分为对应的具体数据数组，格式与data完全相同。还可以通过node.sendUDP(str)将字符串类型的数据打印到GUI中的运行状态中，作为一种调试工具使用。
5. 返回结果。return value，其中value为需要传递给根节点的值

设计流程

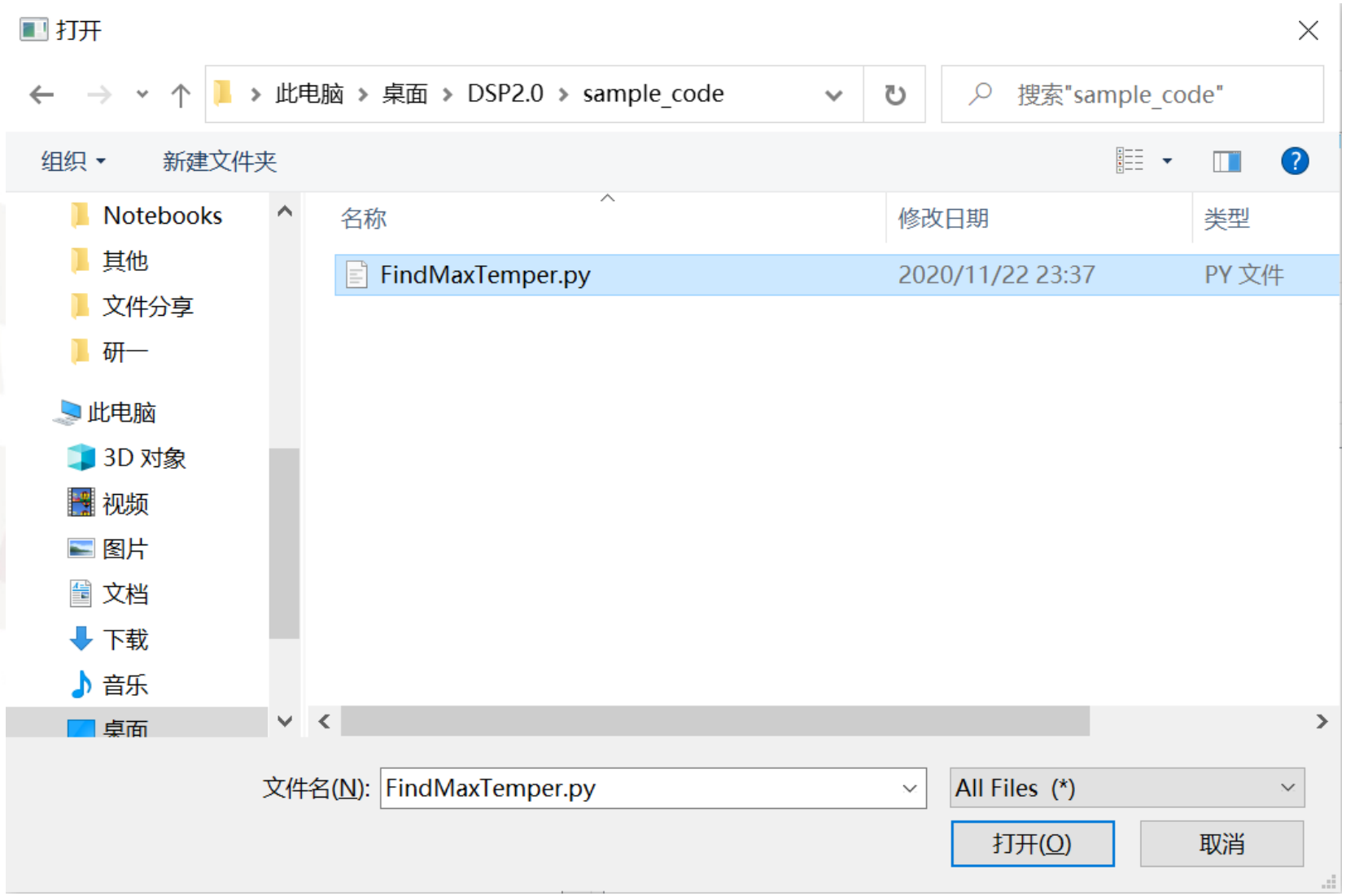

taskFunction
算法开始
获取导入数据
本地计算
数据交互
返回数据(value)
node.
sendUDP(str)
可在函数中任意位置调用
node.
transmitData(data)

算法模板

算法模板格式

```
# import需求模块
# 用户自定义函数区
"""
用户编码区域
"""
# 定义算法主函数taskFunction(self, id, adjDirection, datalist)，四个形参分别为节点类，节点ID，节点邻居对应的方向以及初始化时键入的数据
def taskFunction(self, id, adjDirection, datalist):
    # 在算法设计中，可与邻居节点进行通信，函数为self.transmitData(data)，data为需通信数据，可为字符串、对象、数组等JSON格式能解析的数据类型
    # 返回数据feedback = self.transmitData(data)为二元数组，第一分量表示返回邻居方向adjDirection，第二分量表示相应数据，与第一分量中方向一一对应型
    # 通信的异步函数为self.sendDataToDirection(direction, data)，direction为需要传递数据的方向，data为需通信数据，可为字符串，对象，数组等JSON格式能解析的数据类型，调用该函数时可通过self.adjData获取邻居传递过来的数据信息，内容与方向adjDirection一一对应
    # 异步通信函数请勿与同步通信函数同时使用，使用异步通信函数时需要考虑到程序执行的异步性与延时，慎用！
    # 调用异步函数时可以使用self.syncNode()函数使得程序实现同步
    # 在算法设计中，可在运行状态区域打印出自己调试所需要的算法运行信息，函数为self.sendUDP(str)，str为字符串格式的数据
    value = []    # value为返回值，可为任意格式
    """
    用户编码区域
    """
    return value
```

算法导入


打开
此电脑 › 桌面 › DSP2.0 › sample_code
搜索"sample_code"
组织
新建文件夹
Notebooks
其他
文件分享
研一
此电脑
3D 对象
视频
图片
文档
下载
音乐
桌面
名称
修改日期
类型
FindMaxTemper.py
2020/11/22 23:37
PY 文件
文件名(N): FindMaxTemper.py
All Files (*)
打开(O)
取消

FindMaxTemper.py 文件示例

sample_code > FindMaxTemper.py

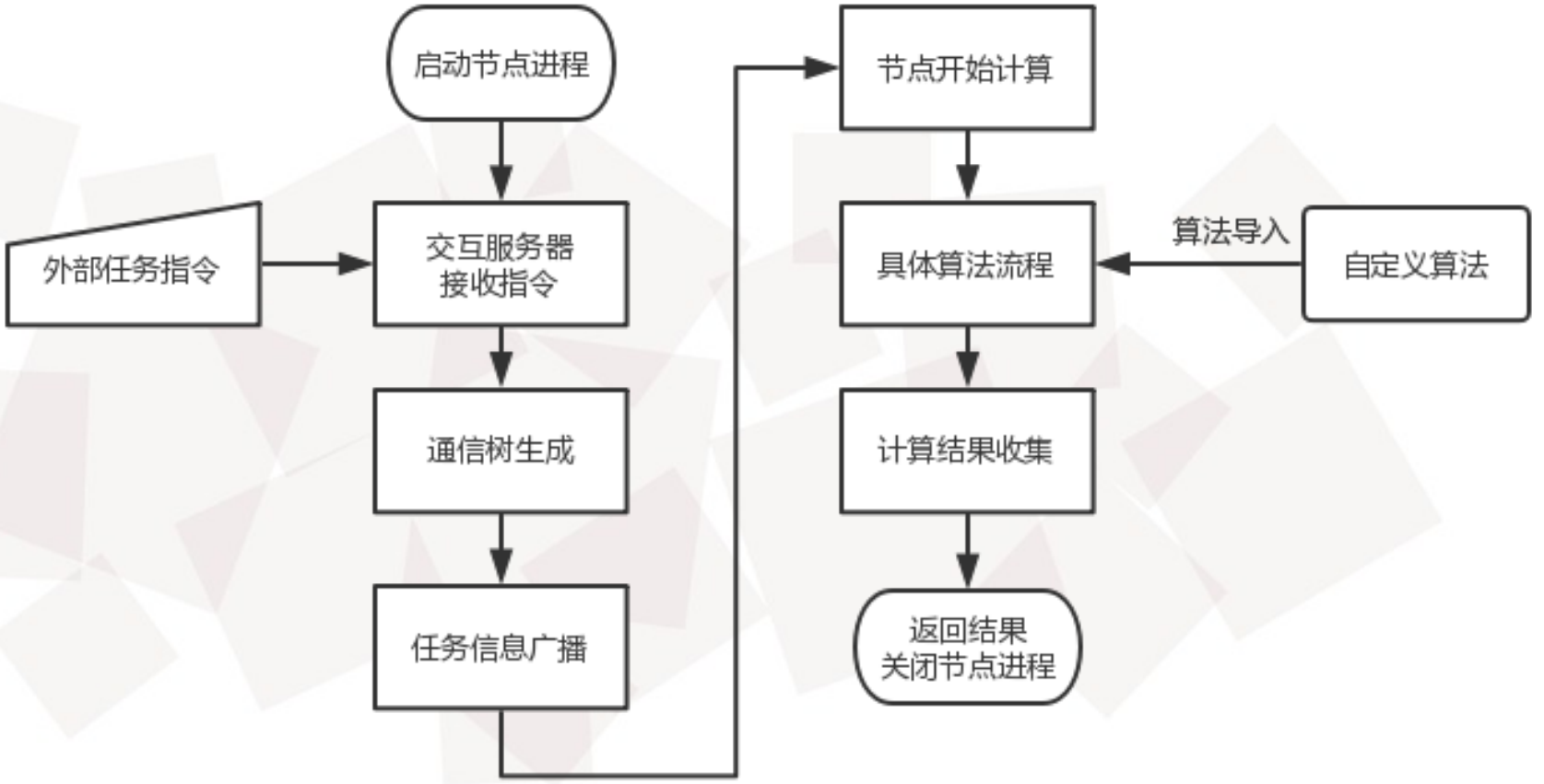
```python
# import需求模块
import random
# 用户自定义函数区
"""
用户编码区域
"""

# 定义算法主函数taskFunction(self,id,adjDirection,datalist)
# 四个形参分别为节点类，节点ID，节点邻居对应的方向以及初始化时键入的数据
def taskFunction(self, id, adjDirection, datalist):
    temp = round(random.uniform(0,30))
    self.sendUDP("初始温度为: " + str(temp))
    maxtemp = temp
    for m in range(10):
        data = self.transmitData(maxtemp)
        adjDirection = data[0]
        adjData = data[1]
        for i in range(len(adjData)):
            if adjData[i] > maxtemp:
                maxtemp = adjData[i]
        self.sendUDP("第" + str(m+1) + "步完成")
    value = maxtemp
    # value为返回值，可为JSON能解析的任意格式
    return value
```

算法设计优势

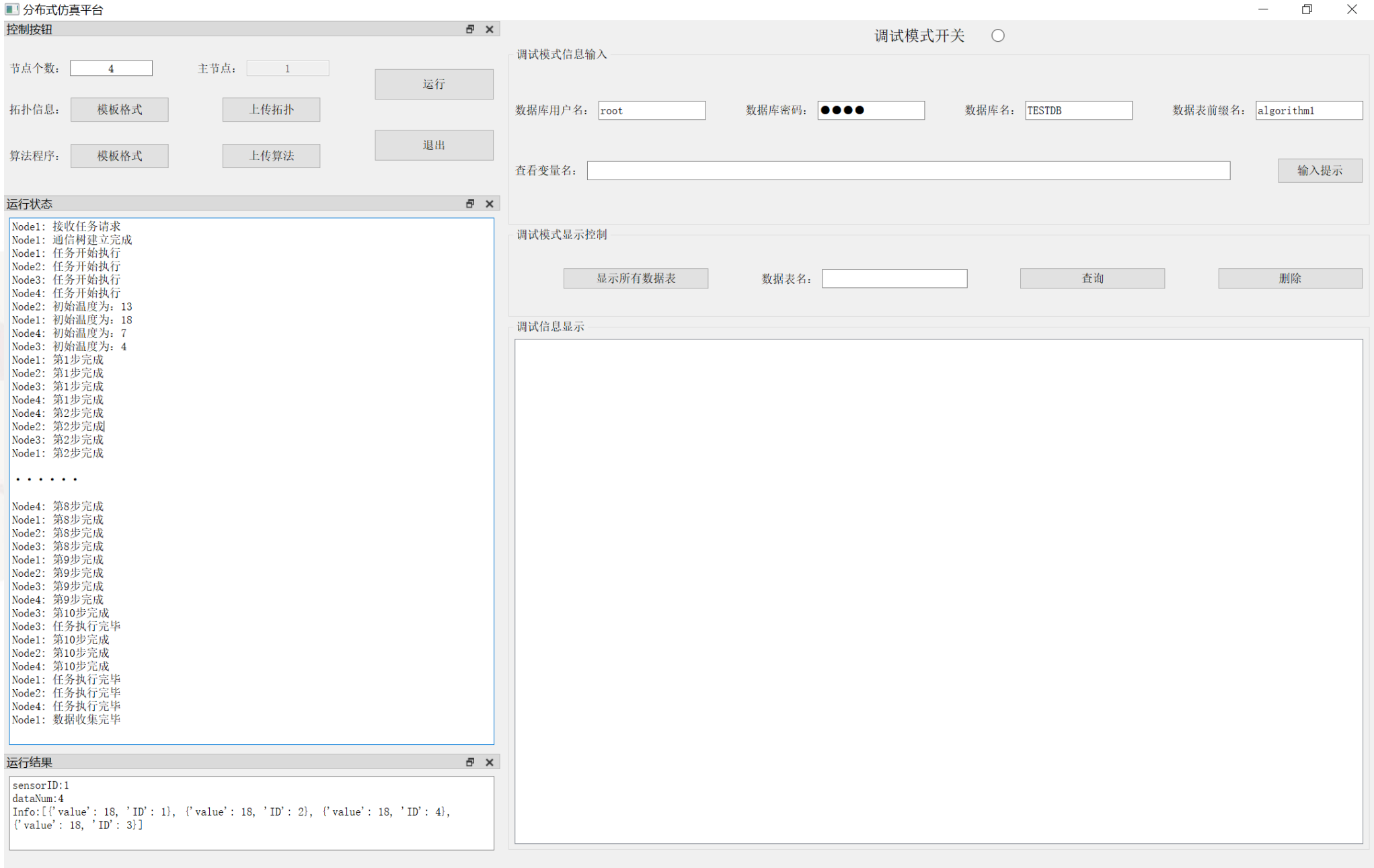
- 算法与仿真平台独立，只需将算法文件导入即可运行
- 无需考虑底层的拓扑结构与节点间的通信，只需在计算层面上考虑
- 支持节点规模的扩展，能充分验证算法的效率与可行性
- 可通过运行状态查看程序错误信息以及需要输出的信息，便于调试

任务执行

内部流程


启动节点进程
外部任务指令
交互服务器
接收指令
通信树生成
任务信息广播
节点开始计算
具体算法流程
算法导入
自定义算法
计算结果收集
返回结果
关闭节点进程

执行结果


分布式仿真平台
控制按钮
节点个数：
4
主节点：
1
运行
拓扑信息：
模板格式
上传拓扑
退出
算法程序：
模板格式
上传算法
运行状态
Node1：接收任务请求
Node1：通信树建立完成
Node1：任务开始执行
Node2：任务开始执行
Node3：任务开始执行
Node4：任务开始执行
Node2：初始温度为：13
Node1：初始温度为：18
Node4：初始温度为：7
Node3：初始温度为：4
Node1：第1步完成
Node2：第1步完成
Node3：第1步完成
Node4：第1步完成
Node4：第2步完成
Node2：第2步完成
Node3：第2步完成
Node1：第2步完成
Node4：第8步完成
Node1：第8步完成
Node2：第8步完成
Node3：第8步完成
Node1：第9步完成
Node2：第9步完成
Node3：第9步完成
Node4：第9步完成
Node3：第10步完成
Node3：任务执行完毕
Node1：第10步完成
Node2：第10步完成
Node4：第10步完成
Node1：任务执行完毕
Node2：任务执行完毕
Node4：任务执行完毕
Node1：数据收集完毕
运行结果
sensorID:1
dataNum:4
Info:[{'value': 18, 'ID': 1}, {'value': 18, 'ID': 2}, {'value': 18, 'ID': 4}, {'value': 18, 'ID': 3}]
调试模式开关
调试模式信息输入
数据库用户名：
root
数据库密码：
数据库名：
TESTDB
数据表前缀名：
algorithm1
查看变量名：
输入提示
调试模式显示控制
显示所有数据表
数据表名：
查询
删除
调试信息显示

谢谢聆听！