# Hecatonchires manual

Chenxi Qiu


- *Chenxi Qiu is with the Department of Computer Science and Technology, Nanjing University, PR China.*
- *Email:qiucxnju@gmail.com*

# CONTENTS

# 1 HECATONCHIRES简介

## 1.1 用途

Hecatonchires是metis项目的一个子项目，这个项目是用java编写的，主要用于解决metis中实时排名产生大量计算任务的问题。Hecatonchires是一个采用Master/Slave架构的分布式计算程序。

运行时，在服务器上运行一个叫MasterNode的线程，而在若干个client上分别运行一个SlaveNode线程。MasterNode主要负责：管理所有的slave，包括身份验证、ip管理、心跳管理、查找任务、分配任务。当client的SlaveNode启动后，它需要等待Master分配给他任务，完成任务后把结果发给服务器。并且在这过程中，slave要每隔一段时间发送一个心跳给master，以保证master知道哪些slave在等待，而哪些slave已经关闭。

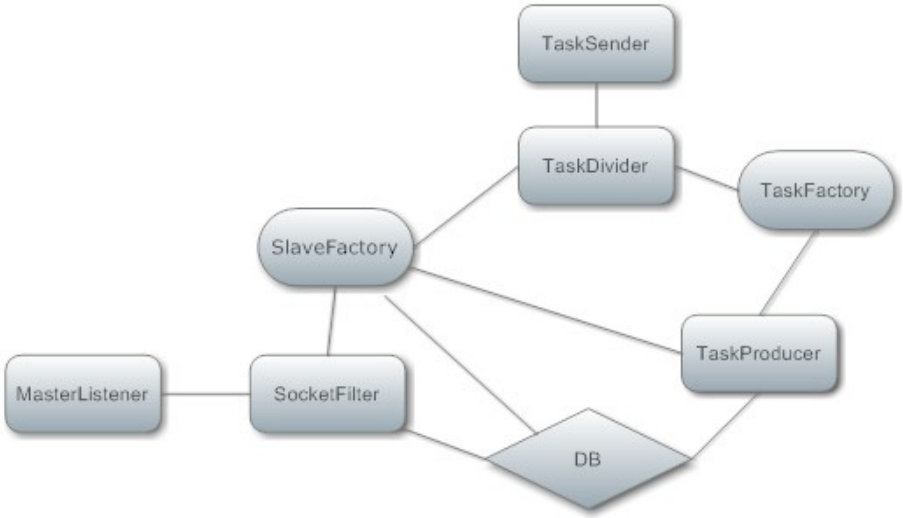
## 1.2 SlaveNode原理

SlaveNode的结构比较简单，他主要包含两个线程TaskListener和HeartBeatSender。当启动的时候，SlaveNode会首先自动给Master发送注册请求，接着创建TaskListener和HeartBeatSender。

HeartBeatSender负责每隔一段时间，跟Master做一次心跳以确认双方的信息是否正确。如果HeartBeatSender获得Master通知发现双方存在不一致问题，则会强行终止SlaveNode。

TaskListener会开启一个ServerSocket监听Master发给他的信息，一旦接受了Master的任务，TaskListener会马上开始进行处理，完成后上传结果给服务器。这里TaskListener是但线程的，所以他在处理这个任务的时候是无法监听到任何socket，而这也是不应该发生的。

## 1.3 MasterNode原理




MasterNode的整体结构如上图所示。其中方框代表一类线程，圆圈代表一个数据结构，菱形代表数据库。

其中，所有的Slave的信息都存储在SlaveFacltory，这些信息包括ip地址、认证id、id级别(id级别请参照后面章节的介绍)、心跳时间等，并由其统一提供接口进行访问。所有的任务的信息保存在TaskFactory中，包括评测优先级和传输内容等等。SlaveFactory的所有的函数都是串行

的，以维持数据的一致性。所以若非必要，应该尽量减少对其函数的调用次数，TaskFactory与之相同。

MasterListener主要是管理一个ServerSocket，这个ServerSocket固定监听一个接口（接口号请参照“参数设置”章节）。当监听到一个Socket请求后，MasterListener将创建SocketFilter线程，并将这个socket交给SocketFilter线程来处理。

SocketFilter主要是根据Socket中的信息来确定不同的通信请求，包括注册请求、心跳确认、结果接收、文件下载4种，其中，注册请求、心跳确认、结果接收三种都需要调用SlaveFactory的接口而结果接收需要调用数据库。

TaskProducer主要是用于产生任务并把它放入TaskFactory中。TaskProducer工作时首先会查看SlaveFtory中是否有需要被回收的任务，如果有，则把他们全部回收，然后查看数据库，看数据库中是否有任务需要评测并且之前没有被TaskProducer侦测到。然后把他们插入TaskFactory中。

TaskDivider用于把任务分发给正在等待任务的slave。首先，TaskDivider会从TaskFactory中提取一个Task，然后查看SlaveFactory，并获取到这个SlaveFactory的ip地址，接着其会创建一个TaskSender的线程将这个task和ip地址转交给TaskSender。

TaskSender把task对应的数据发送给处理这个task的id。

# 2 数据结构

## 2.1 SlaveInfo

### *2.1.1 overview*

SlaveInfo类用来存储每个slave的信息，其内容如下：
String ip：标示这个slave的ip地址。
String id：标示这个slave认证使用的用户名。
int level：标示这个slave的优先等级，level越低将越先获取任务。
int state：标示这个slave的状态，状态定义在Param中。
long lastHeartBeat：标示这个slave的上一次心跳时间。
int index：标示这个slave加入等待序列的顺序号，当level相同时，index越低将越先获取任务。
TaskInfo taskInfo：保存这个slave正在处理的任务的信息。

### *2.1.2* 构造函数

SlaveInfo(String Ip, String Id, String Password, int Level, int Index)
state被初始化为Param.STATE_WAITING
taskInfo被初始化为null
lastHeartBeat使用SlaveInfo.getTime()得到。

### *2.1.3* 公有函数

long getTime()
获取当前时间

boolean isTimeOut()
返回是否超时

## 2.2 SlaveFactory

### *2.2.1 overview*

SlaveFactory类用来储存并管理所有slave的信息。

### *2.2.2* 构造函数

SlaveFactory()
产生一个新的SlaveFactory

#### 2.2.3 公有函数

boolean heartBeat(String ip)
ip标示的Slave进行了一次心跳，若这个slave存在在这个SlaveFactory中则更新这个slave的上一次心跳的时间并返回true，反之返回false。

boolean register(String ip, String id, String password)
向SlaveFactory中注册一个slave。参数分别为其ip，认证id，认证密码。这时将检查数据库，如果这个id和密码不匹配，返回false。否则如果这个ip之前注册过，则先删除这个ip对应的slave信息。接着把这个slave新的信息插入SlaveFactory中。

String getSlave()
返回一个有效的slave对应的ip。该函数首先把所有等待中的slave按照¡level,index¿（参照SlaveInfo的定义）的顺序排序，并从小到大检查所有的slave如果发现最小的slave出现超时的情况，则将删除这个slave的信息并检查下一个。当找到一个满足条件地slave后，把它的状态设置为“调度中”，把它从等待队列中删除，返回ip。如果不存在这样的slave则返回null。

boolean setWork(String ip, TaskInfo taskInfo)
把ip对应的slave的状态设置为工作中，若这个slave不在则直接返回false，如果这个slave的状态不是“调度中”，则将把这个slave删除并返回false。如果正常，则把slave设置为工作中并把这个taskInfo存在这个slave对应的SlaveInfo中并返回true。

int getLevel(String id, String password)
获取id对应的账户的用户等级，如果账号密码匹配失败则返回-1。

void recycle(Stack¡TaskInfo¿ tasks)
回收所有处理失败的TaskInfo并存储在tasks中。

#### 2.2.4 私有函数

void insert(String ip, SlaveInfo slaveInfo)
向数据结构中插入这个SlaveInfo的信息，这里并不会检查SlaveInfo的正确性，所以插入前请保证这个SlaveInfo的state为“等待中”。若数据结构中已经存在ip则直接退出。
void remove(String ip)
从数据结构中删除ip对应的slave的信息。如果ip不存在则直接退出。如果这个ip正在工作，则把这个ip正在处理的任务存放在回收站中；如果这二个ip正在等待，则从等待队列中删除。

### 2.3 TaskInfo

#### 2.3.1 overview

TaskInfo保存评测需要的信息：
String data：存储这个任务所需要传输给slave的数据。

int level: 存储这个任务的评测等级。
int index: 存储这个任务加入等待序列的时间。

TaskInfo是有序的，首先按照level排序，如果level相同则按index排序，所以必须至少保证index是单调递增的，如果index重复可能导致TaskInfo从TaskFactory中丢失。

#### *2.3.2* 构造函数

TaskInfo(int Level, int Index, String Data)

### 2.4 TaskFactory

#### *2.4.1 overview*

TaskFactory保存所有的TaskInfo，并把他们按照一颗树来组织。必须保证TaskInfo是两两不同的。并且这个TaskFactory的大小是有限制的，具体限制定义在Param中。

#### *2.4.2* 构造函数

TaskFactory()
清空整个数据结构，并把index设置为0，并用这个单调增加的数来构造TaskInfo的序。因为index的序是有限的，只能所以如果index自由增大到一定量，服务器必须重启，但是可以保证的是如果服务器客户端正常，index是不会达到上限的。

#### *2.4.3* 公有函数

int getSize()
返回TaskFactory中有多少任务在排队。

int getNeedSize() 返回TaskFactory中还可以插入多少任务。

boolean insert(TaskInfo taskInfo)
像TaskFactory中插入一个TaskInfo，如果TaskFactory已经满了，则返回false并停止向队列中加入这个任务，反之则把任务加入等待序列并返回true。注意这个TaskInfo的index将会被重新设置。

boolean insert(int level, String data)
函数作用同上一个函数。

TaskInfo getTask()
函数返回一个正在排队中任务，并将其从等待队列中删除。如果等待队列为空，则返回null，否则返回队列中第一个任务。

# 3 服务器运行的线程

## 3.1 MasterNode

### *3.1.1 overview*

MasterNode类负责启动、管理、关闭服务器上的线程。

### *3.1.2* 构造函数

MasterNode()

初始化MasterNode，并创建其控制的线程的类。

### *3.1.3* 公有函数

void SetStopFlag()

把MasterNode的StopFlag设置为false并对其管理的所有的线程调用void SetStopFlag()。

void run()

创建SlaveFactory、TaskFactory、MasterListener、TaskProducer、TaskDivider，并且把SlaveFactory传递给MasterListener、TaskProducer、TaskDivider线程，把TaskFactory传递给TaskProducer、TaskDivider线程。

## 3.2 MasterListener

### *3.2.1 overview*

MasterListener用于监控指定的port（定义在Param中），每收到一个socket请求，就创建一个SocketFilter线程处理这个socket。所以当突然有大量socket请求时系统可能因为创建太多的线程而崩溃，所以现在的解决方法是让SocketFilter这个线程的生存周期尽量短。这是在后来的工作中需要被改进的。

### *3.2.2* 构造函数

MasterListener(SlaveFactory s, MasterNode m)

通过传递SlaveFactory和MasterNode来构造这个类，并且试图绑定ServerSocket，如果绑定失败，则调用MasterNode的SetStopFlag()函数。

### *3.2.3* 公有函数

void SetStopFlag()

把StopFlag设置为true。

void run()

循环忙等待socket请求，当收到一个socket请求后，创建SocketFilter线程来处理这个socket请求。

如果这时返现StopFlag已经被设置为false了，则退出，否则继续忙等待。所以即使StopFlag被设置为false，这个线程也不会马上停止，只会在收到socket请求后才会停止。

## 3.3 SocketFilter

### *3.3.1 overview*

SocketFilter是用于处理socket请求的类，它现在采用的是block的通信机制，所以需要保证slave及时发送数据，任何通信一致性上的问题都可能导致这个线程死掉，所以这个线程处理是Master上最脆弱的一环。这时在后面的工作中需要优化的。

### *3.3.2* 构造函数

SocketFilter(Socket s, SlaveFactory sf, MasterNode m)
SocketFilter通过传递socket，SlaveFactory，MasterNode来初始化。

### *3.3.3* 公有函数

void run()
首先SocketFilter会读取Socket信息中的type类型，如果发现是REGISTER，则调用SlaveFactory的register函数，以注册一个新的Slave。如果是HEARTBEAT则调用slaveFactory的heartBeat函数更新这个ip的心跳信息，防止这个ip对应的slave因为过期被回收。其他类型还有待实现。

## 3.4 TaskProducer

### *3.4.1 overview*

TaskProducer的作用是向TaskFactory中插入TaskInfo。TaskProducer会优先回收处理失败并保存在SlaveFactory中的TaskInfo，如果回收过后TaskFactory仍然没满，则访问数据库，从数据库中提取任务。

### *3.4.2* 构造函数

TaskProducer(MasterNode m, SlaveFactory s, TaskFactory t)
TaskProducer通过传递TaskFactory，SlaveFactory，MasterNode来初始化。

#### 3.4.3 公有函数

void GetTaskFromDB(Stack¡TaskInfo¿ tasks, int need)
从数据库中提取不多于need个任务，并保存在tasks中。在这里并没有实现。

void run()
循环运行，直到StopFlag被设置为true。每次循环会从SlaveFactory中回收TaskInfo，并把他们插入TaskFactory中，如果插入完成过后TaskFactory依然未满，则从数据库中提取任务加入TaskFactory中。每次循环结束后，sleep一段时间。

### 3.5 TaskDivider

#### 3.5.1 overview

TaskDivider是负责任务分发的。这里暂时保证只有一个TaskDivider。

#### 3.5.2 构造函数

TaskDivider(MasterNode m, SlaveFactory s, TaskFactory t)
TaskDivider通过传递TaskFactory，SlaveFactory，MasterNode来初始化。

#### 3.5.3 公有函数

setStopFlag()
将线程的停止标记设置为true。

void SendTask(String ip, TaskInfo taskInfo)
将任务的数据发给对应ip。

void run()
首先从SlaveFactory中提取一个空闲的ip，如果有，则再提取一个任务。如果ip和任务都提取成功，则调用SendTask函数。否则，说明SlaveFactory或者TaskFactory为空，则等待一段时间防止占用这两个数据结构的时间。