

Hecatonchires manual

Chenxi Qiu



-
- 这篇文章是提供给Hecatonchires开发人员的开发指南。
 - Email: qiucxnju@gmail.com

CONTENTS

1	Hecatonchires 简介	4
1.1	用途	4
1.2	SlaveNode 原理	4
1.3	MasterNode 原理	5
2	通信协议	6
3	数据结构	7
3.1	SlaveInfo	7
3.1.1	overview	7
3.1.2	构造函数	7
3.1.3	公有函数	7
3.2	SlaveFactory	7
3.2.1	overview	7
3.2.2	构造函数	7
3.2.3	公有函数	8
3.2.4	私有函数	8
3.3	TaskInfo	9
3.3.1	overview	9
3.3.2	构造函数	9
3.4	TaskFactory	9
3.4.1	overview	9
3.4.2	构造函数	9
3.4.3	公有函数	9
4	服务器运行的线程	11
4.1	MasterNode	11
4.1.1	overview	11
4.1.2	构造函数	11
4.1.3	公有函数	11
4.2	MasterListener	11
4.2.1	overview	11
4.2.2	构造函数	11
4.2.3	公有函数	11
4.3	SocketFilter	12
4.3.1	overview	12
4.3.2	构造函数	12
4.3.3	公有函数	12
4.4	TaskProducer	12
4.4.1	overview	12
4.4.2	构造函数	12

	4.4.3	公有函数	12
4.5		TaskDivider	12
	4.5.1	overview	12
	4.5.2	构造函数	13
	4.5.3	公有函数	13
5		客户端运行的线程	14
5.1		SlaveNode	14
	5.1.1	overview	14
	5.1.2	构造函数	14
	5.1.3	私有函数	14
	5.1.4	公有函数	14
5.2		HeartBeatSender	14
	5.2.1	overview	14
	5.2.2	构造函数	14
	5.2.3	公有函数	14
5.3		TaskListener	15
	5.3.1	overview	15
	5.3.2	构造函数	15
	5.3.3	公有函数	15

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。

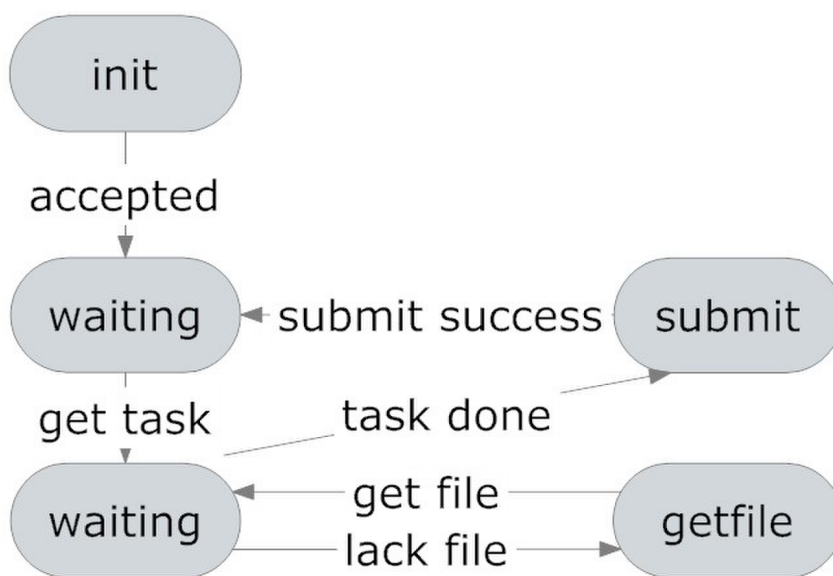
运行时，在服务器上运行一个叫MasterNode的线程，而在若干个client上分别运行一个SlaveNode线程。MasterNode主要负责：管理所有的slave，包括身份验证、ip管理、心跳管理、查找任务、分配任务。当client的SlaveNode启动后，它需要等待Master分配给他任务，完成任务后把结果发给服务器。并且在这过程中，slave要每隔一段时间发送一个心跳给master，以保证master知道哪些slave在等待，而哪些slave已经关闭。

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。

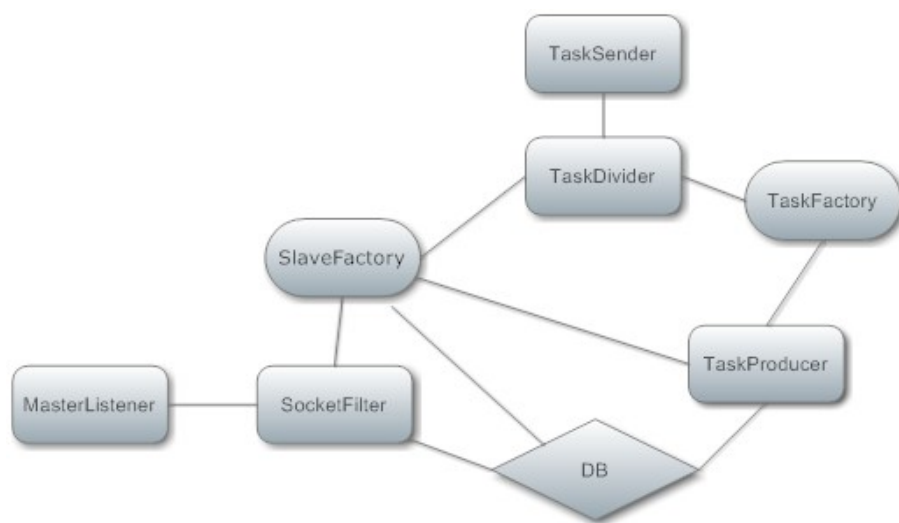
HeartBeatSender负责每隔一段时间，跟Master做一次心跳以确认双方的信息是否正确。如果HeartBeatSender获得Master通知发现双方存在不一致问题，则会强行终止SlaveNode。

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是单线程的，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。



如上图所示，slave首先处于init状态，当slave成功调用register后，进入waiting状态，当TaskListener接收到master的任务后，进入working状态。working的原理是把task需要的文件放入一个安全的目录中并调用shell指令make。如果发现slave缺少这些文件，会进入getfile状态，向master请求文件，当该文件请求到后，继续working。当评测结束后，进入submit状态向master上传文件。当所有文件上传后，report给master告诉master这个任务处理完成了，然后进入waiting状态。

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SlaveFactory，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、文件接收、文件下载、结果上传五种，其中，注册请求、心跳确认、结果上传三种都需要调用SlaveFactory的接口而结果上传需要调用数据库。

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SlaveFactory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检测到。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 通信协议

slave向master发起的通信包括：注册、心跳、文件请求、上传文件、评测结束五种。其中，除了注册是SlaveNode发送给SocketFilter的以外，其他都是TaskListener发送给SocketFilter的。master向slave发起的通信直接是分派任务，是TaskDivider发送给TaskListener的。

slave发送给master的通信第一行包含一个type参数，其中type的取值为：{TYPE_REGISTER, TYPE_HEARTBEAT, TYPE_GETFILE, TYPE_SUBMIT, TYPE_CLOSE}。（具体数值定义在Param中）

当type为TYPE_REGISTER时，说明当前slave是要进行注册，包内第二行和第三行分别是注册的用户名和密码。如果注册成功，返回“OK”，否则，返回“Error”。

当type为TYPE_HEARTBEAT时，说明slave要进行一次心跳。包只有一行也就是type信息，master会检查是否有对应ip的注册信息，如果成功heartbeat，返回“OK”，否则返回“Error”。

当type为TYPE_GETFILE时，slave要读取文件。第二行包含需要读取的文件名。master会返回整个文件。

当type为TYPE_SUBMIT时，slave向master上传文件。第二行是文件名。后面有若干行，每一行以0或者1结尾，1代表不是结尾，0代表是结尾。当发现某行最后为0时，表示传输完成，返回“OK”。

当type为TYPE_CLOSE时，说明评测结束，将调用report函数把结果存入数据库，并且把slave的状态从“工作中”切换到“等待中”。

而分配任务的时候，包的第一行有两个数字a, b，分别代表输入文件个数和输出文件个数。接下来有 $a + b$ 行，从2到 $1 + a$ 行，每行两个字符串，以空格分开，分别表示其真实目录位置和评测时重命名的名字。从 $2 + a$ 行到 $1 + a + b$ 行，每行有两个字符串，以空格分开，分别表示其在评测时的名字和其存放到master后的名字。其中，第 $2 + a$ 行必须是themis的输出文件。

3 数据结构

3.1 SlaveInfo

3.1.1 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。

3.1.2 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。

3.1.3 公有函数

long getTime()

获取当前时间

boolean isTimeOut()

返回是否超时

3.2 SlaveFactory

3.2.1 overview

SlaveFactory类用来储存并管理所有slave的信息。

3.2.2 构造函数

SlaveFactory()

产生一个新的SlaveFactory

3.2.3 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。

`TaskInfo getTask(String ip)`

返回`ip`标示的slave正在处理的任务信息，不存在slave或者这个slave没有在工作，返回null。

`boolean setUnwork(String ip)`

把`ip`标示的slave的状态设置为“等待中”，清空其正在处理的TaskInfo信息然后把它让入等待队列中。如果不存在slave或者其状态不是“工作中”，则返回false，否则返回true。

`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。

`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 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，则先调用`remove(ip)`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。

`void recycle(Stack<TaskInfo> tasks)`

回收所有处理失败的TaskInfo并存储在tasks中。

3.2.4 私有函数

`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 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；如果这个ip正在等待，则从等待队列中删除。

3.3 TaskInfo

3.3.1 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。

3.3.2 构造函数

TaskInfo(int Level, int Index, String Data)

3.4 TaskFactory

3.4.1 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。

3.4.2 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，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。

3.4.3 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将会被重置。

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。

4 服务器运行的线程

4.1 MasterNode

4.1.1 overview

MasterNode类负责启动、管理、关闭服务器上的线程。

4.1.2 构造函数

MasterNode()

初始化MasterNode，创建SlaveFactory、TaskFactory两个数据结构和MasterListener、TaskProducer、TaskDivider三个线程。

4.1.3 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()

，运行MasterListener、TaskProducer、TaskDivider线程。

4.2 MasterListener

4.2.1 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。

4.2.2 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。

4.2.3 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。如果这时发现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。

4.3 SocketFilter

4.3.1 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。

4.3.2 构造函数

SocketFilter(Socket s, SlaveFactory sf, MasterNode m)

SocketFilter通过传递socket, SlaveFactory, MasterNode来初始化。

4.3.3 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，说明一个新的slave到master注册，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。如果是GETFILE说明slave缺少文件，读取文件本地文件发送过去。如果是SUBMIT，接收slave发送过来的文件并保存在指定位置。如果是CLOSE，说明一个slave评测完成，保存结果到数据库，这里需要调用report函数，这个还没有实现。

4.4 TaskProducer

4.4.1 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。

4.4.2 构造函数

TaskProducer(MasterNode m, SlaveFactory s, TaskFactory t)

TaskProducer通过传递TaskFactory, SlaveFactory, MasterNode来初始化。

4.4.3 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依然未滿，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。

4.5 TaskDivider

4.5.1 overview

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

4.5.2 构造函数

`TaskDivider(MasterNode m, SlaveFactory s, TaskFactory t)`

`TaskDivider`通过传递`TaskFactory`，`SlaveFactory`，`MasterNode`来初始化。

4.5.3 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`SlaveFactory`里面把对应`IP`设置为工作中，初始化一个`TaskSender`线程发送。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。

5 客户端运行的线程

5.1 SlaveNode

5.1.1 overview

SlaveNode类负责启动、管理、关闭客户端上的线程。

5.1.2 构造函数

MasterNode()

初始化SlaveNode。

5.1.3 私有函数

boolean Register()

向master注册，如果注册成功，返回true，否则，返回false。

5.1.4 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()

首先调用Register函数向master注册，如果注册失败，退出，如果注册成功，启动HeartBeatSender和TaskListener两个线程。

5.2 HeartBeatSender

5.2.1 overview

HeartBeatSender主要是用于不断向master发送心跳，使得master一直知道当前slave处于活动状态，当发现master拒绝slave的心跳时，关闭客户端。

5.2.2 构造函数

HeartBeatSender(SlaveNode s)

传递管理其的SlaveNode，设置stopFlag为false。

HeartBeat()

这个函数是进行一次心跳，如果成功，返回true，否则返回false。首先，如果无法连接上socket，则返回false。如果master返回值不是“OK”则打印返回值到logger并且返回false。

5.2.3 公有函数

void SetStopFlag()

把StopFlag设置为true，但自己永远不调用，只由SlaveNode线程调用。

void run()

不断进行心跳，如果HeartBeat函数调用返回false则关闭slave。

5.3 TaskListener

5.3.1 overview

TaskListener是用于处理任务的线程，其工作原理是建立socket等待master发送任务，当接受到master的任务过后，补全处理任务所需要的文件，然后计算，上传结果和文件，所有任务结束后接着等待下一次任务。

5.3.2 构造函数

TaskListener(SlaveNode s)

传递管理其的SlaveNode，设置stopFlag为false。

5.3.3 公有函数

void SetStopFlag()

把stopFlag设置为true。

boolean submit(String filename, String newname)

上传sandbox文件中名为filename的文件，并告知master把这个文件保存并命名为newname，当出现错误的时候返回false，否则返回true。

boolean cp(String orgname, String servername, String newname)

把本地名为orgname的文件拷贝到sandbox中，并命名为newname。当本地丢失orgname的文件时，向master请求名为servername的文件并重新命名为orgname。否则，继续拷贝。当出现任何问题是，返回false，否则，返回true。

boolean report(String dir)

把sandbox中名为dir文件里面的评测结果报告给master，如果出错，返回false。

boolean DoTask (ArrayList<String> data, int insize, int outsize)

这个函数的作用是处理一次任务，insize代表输入文件的个数，outsize代表输出文件的个数，data前insize个文件名是评测输入文件名，后outsize是处理后产生的文件名。注意前insize个文件的顺序可以打乱，但是outsize中的第一个必须是themis.out的输出文件，因为后面会对这个文件调用report函数函数运行时，首先，重新创建sandbox文件夹，准备好评测环境。接着调用cp函数把需要评测的文件拷贝到sandbox文件夹中。对sanbox调用make，接着把outsize个文件依次调用submit传输给master，最后调用report告知master评测完成。

void run()

首先TaskListener会注册一个ServerSocket用来等来master的任务。当收到任务后，调用DoTask处理任务，如果出现异常，关闭整个slavenode，否则，等待下一次任务。