**用户使用手册**

# DCR简介

DCR(Decompose, Computation, Reduce)是一个基于C++的编程模型，也是一个运行框架，与Map Reduce有类似之处，但是又有其特点。

## 1.1 DCR的使用范围

DCR模型，仅仅支持非常易于并行的可分解问题。

原始问题：

其中：*R*为结果，*S*为输入，*C*为计算过程

如果该问题可以化解为以下过程，则称其为**子空间并行模型**

1. 原始问题空间*S*可以划分为n个子集{S1,…, Sn}，一般而言满足，

2. 对于每个子集的计算可以并行化，，其中C’是每个子集的计算函数；

3. 最终结果可以简单规约

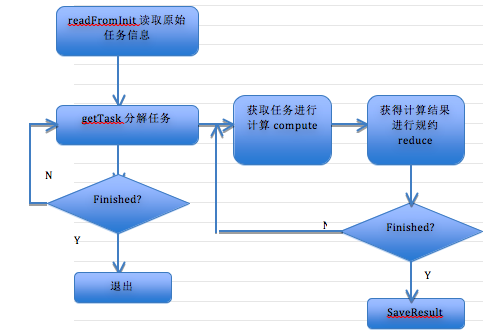
上述描述过程中有以下节点需要注意：

1、第2步中，第i个子空间的计算仅仅依赖于子空间Si，不需要和其他子空间交换信息；

2、第3步中，每个结果的规约是串行完成的，有可能在没有完成整个空间计算时（例如找到方程的根）就结束整个空间的计算。

## 1.2 DCR的计算流程

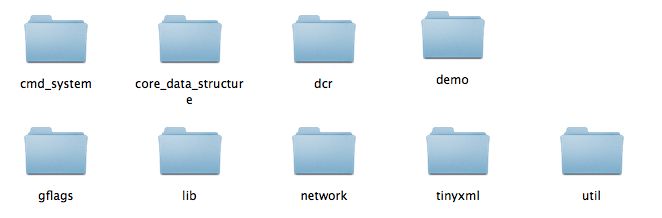
DCR的计算流程如下图所示：



系统读取原始任务文件，分解器对原始任务文件进行分解得到计算任务，计算任务被分发到各个计算节点上去计算并将计算结果返回调度节点进行规约，最后保存计算完成的结果。

# 2.开发

## 2.1系统的代码组织



**cmd\_system** 中主要存放管理节点的代码(目前还未实现管理节点)

**core\_data\_structure** 中主要存放系统核心数据结构包括：计算几点表，内存池（即原来系统中的vq, rq）等代码

**dcr** 文件夹中主要存放计算逻辑基类和类注册系统的代码，以及程序的入口main函数

**demo**文件夹中存放用户实现的应用的代码， 比如积分运算，blast运算等

**gflags** 文件夹中是google开源的参数解析库的头文件，用于解析程序启动时传如的参数

**tinyXML** 中存放的是开源XML 解析库的头文件

**lib** 中存放的是gflags 和 tinyXML编译出来的库文件

**network** 中存放的是系统网络通信模块的代码

**until** 中存放的是系统中使用的一些工具类，比如智能指针，日记系统等

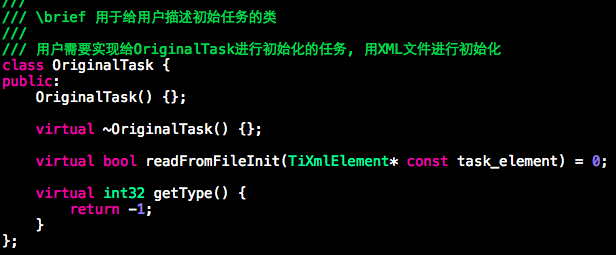
## 2.2用户要实现的基类以及基类的函数

用户在实现具体应用的时候应该继承一下类以及并实现父类提供虚函数.用户在实现自己的子类的时候可以有自己的成员函数和数据成员，但是框架只会调用子类实现的虚函数，不会调用子类自己定义的成员函数，所以用户定义的函数只能通过虚函数调用。

子类中不能定义类成员或者STL中的容器对象。

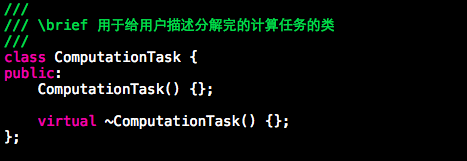
用户用框架进行编程首先要继承和实现以下基类（基类的定义在头文件dcr\_base\_.h中）：

1.原始任务



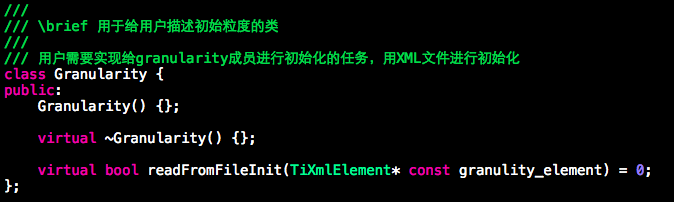
定义用户原始计算任务, 子类需要实现readFromFileInit从XML文件中读取原始任务信息。

2. 计算任务



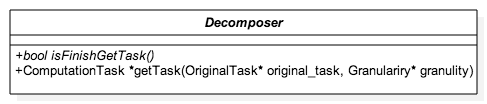
计算任务, 没有提供接口，主要用于用户定义计算任务的数据成员，由分解器通过getTask接口产生。

3.粒度



定义对原始计算任务进行分解的粒度，子类需要实现readFromFileInit从XML文件中读取粒度信息。

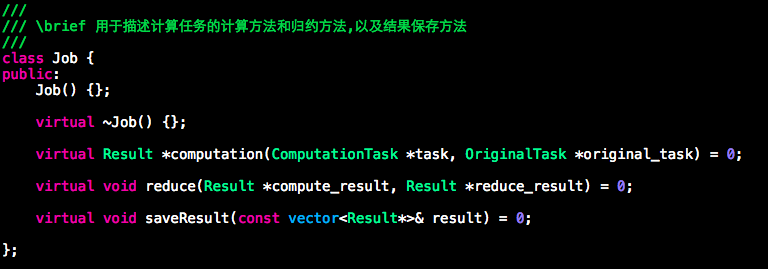
4.分解器



分解器， 接口getTask用于定义用户如何根据OriginalTask 和Granularity分解得到计算任务ComputationTask。

接口isFinishGetTask 用于定义一个原始任务是否已经分解完成

5.描述计算方法和规约方法的类



工作类，主要用于定义如何进行计算和规约以及最后结果如何保存的类。

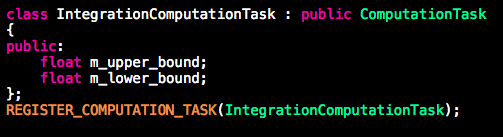
接口computation用于定义用ComputaionTask 和 OriginalTask的信息计算得到Result

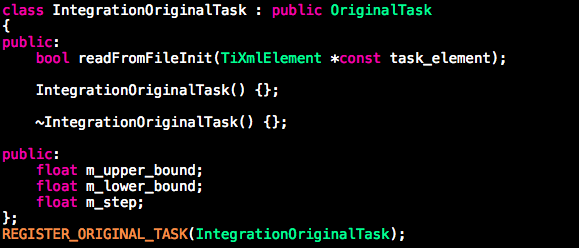
接口reduce用于定义如何把每个ComputationTask计算得到的结果进行规约。

接口saveResult用于定义如何把一个任务文件中的所有任务保存到XML中。

## 2.3注册类

用户继承以上基类实现子类后在每个子类的后面要增加一个宏定义，对实现的类进行注册， 如图所示:



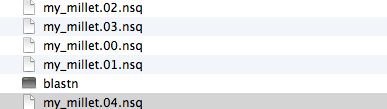


## 2.4以blast为例进行说明

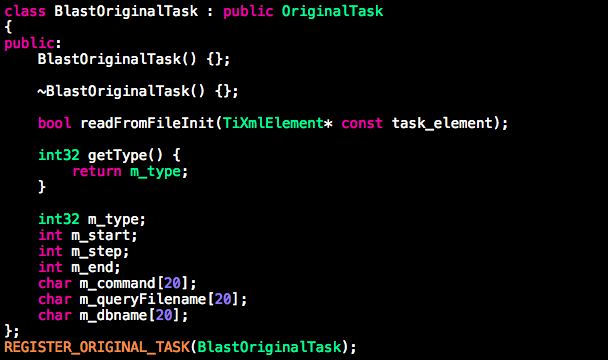
blast 是一个基因序列查询程序， 能够根据提交的基因序列在指定的基因库中进行查找，并输出预支最为相识的基因序列的序列号和相识度。

要对基因序列的查询过程进行并行化，由于blast查询程序的比较复杂，我们不通过更改blast源代码的方法对blast进行并行，而是通过将数据库进行切分，让blast 程序同时在不同数据库分片上进行查询来实现对blast查询的并行化。

1.首先将测试的核酸数据库（my\_millet）分片成00到04共五个数据库，并命名如下所示：

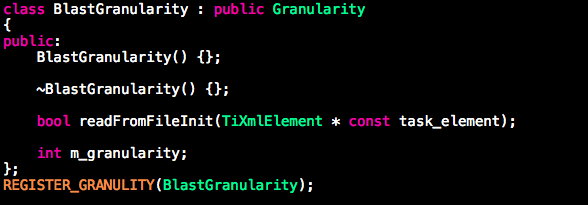


2. 对原始任务信息和分解粒度进行的定义也就是OriginalTask和Granularity子类的定义。如下图所示：

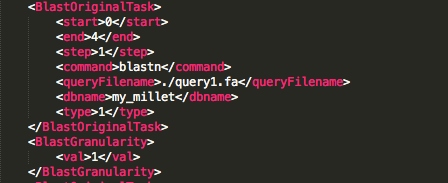


用户需要提供的任务信息为1. blast 程序的名称(m\_command)2.查询文件的文件名(m\_queryFilename)3.要查询的数据库的名称(m\_dbname)4.查询数据库分片的起始编号(m\_start) 5. 查询数据库分片的结束编号(m\_end)。

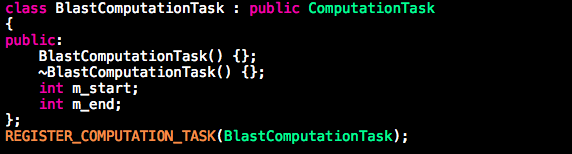
用于需要提供对任务文件进行分解的粒度的信息为粒度的大小，也就是在每个分解得到的计算任务中blast程序需要对多少个数据库分片进行查询， 在这里定义粒度的值为1。



写一个用于测试的任务文件， 如下图所示：

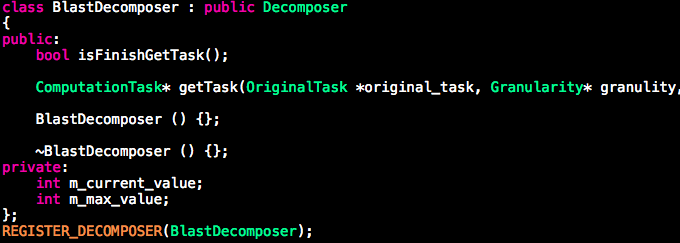


3.对分解得到的计算任务进行定义。



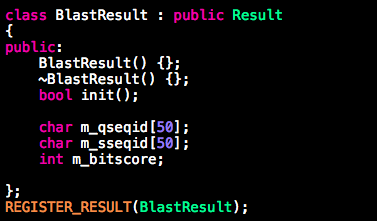
计算任务的信息为计算任务对应的数据库分片的序号(m\_start)

4. 分解器如何由原始任务和粒度分解得到一个计算任务进行描述。如下图所示：



m\_current\_value 用于保存当前分解的数据库分片的序号， getTask 的过程为，通过originalTask 得到数据库分片的起始序号m\_start，然后不断得通过用粒度的值进行叠加得到新的数据库分片的序号，产生并放回一个计算任务， 并将当前分解得到的序号保存在m\_current\_value中。当m\_current\_value的值已经大于数据库分片的最大序号的时候可以判断分解结束。

5.定义计算结果信息。



blast查询输出的信息为1.查询的基因序列的序号（m\_qseqid）2.得到相识度最高的基因序列的序号（m\_sseqid）。3.最大相似度是多少（m\_bitscore）。

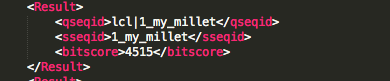
6.对如何利用OriginalTask 的信息和ComputationTask得到的Result 进行描述。

这里我们在Job 的computation 函数中 fork 出一个子进程，然后利用在OriginalTask 的得到的blast可执行文件的名称，以及查询文件的名称，要查询的数据库的名称，还有在computation数据库分片的序号作为参数，用execve执行查询， 并将查询输出的结果通过管道返回到父进程， 并将从管道得到的信息复制给最好要放回的result.

7.对如何利用每个数据库分片计算得到的结果进行规约进行描述。

在Job的reduce函数中对于每个数据库分片计算得到的结果我们都会与当前保存的相识度最高(也就是m\_bitscore最大)的result进行比较，如果比当前保存的最大bitscore还要大，则用这个结果替换掉原来保存的结果。

8.实现Job中saveResult，将最终结果保存到一个XML文件中。 如下图所示：



# 3.测试调试

## 3.1日志系统

系统在初始化的时候会预先创建3个文件输出流，log文件的路径在系统启动的时候通过参数指定。 用户在编写程序的时候可以通过LOG(ERROR)<<”error”的形式输出错误信息error文件中， LOG(INFO) << “info” 输出打印信息到info文件中，通过LOG(WARN) 输出警告信息到warn文件中.

日志的文件名由节点名+用户名+文件创建时间+进程号组成。如下图所示:

Macintosh HD:Users:JJZ:Desktop:2E67F1BE-44A4-4A3E-B237-913455A4DEB0.png

## 3.2串行框架

串行验证框架主要是用于帮助用户对程序进行调试，只要用户在串行验证框架上能够调试正确就能保证，串行验证框架主要是绕过并行计算框架的核心数据结构(内存池，队列等)以及网络通信，把整个框架的计算逻辑走一遍。

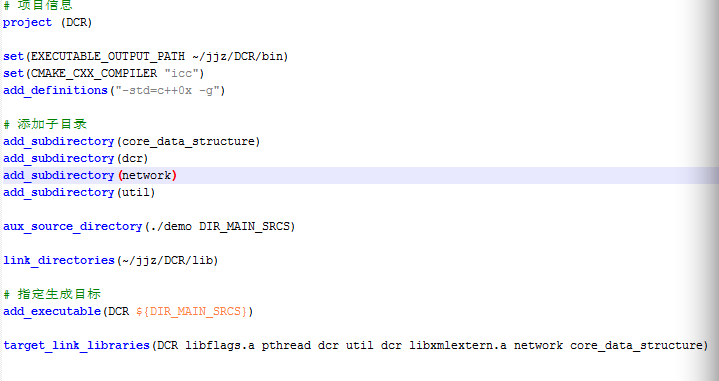
用户可以先使用串行验证框架把程序调试通过，再将用户定义的程序用并行框架进行运行。区别在于串行验证框架在启动的时候没有指定系统运行在调度节点还是计算节点的参数(is\_scheduler\_node)。

# 4.部署

## 4.1 Cmake

CMake是一个比make更高级的编译配置工具，它可以根据不同平台、不同的编译器，生成相应的Makefile或者vcproj项目。  
通过编写CMakeLists.txt，可以控制生成的Makefile，从而控制编译过程。

下图是系统的CmakeLists.txt 文件， 安装完Cmake 之后可以将Cmake 放置在系统源代码目录下， 执行cmake ./ 就可以生成系统的makefile文件，执行make 可以将系统进行编译。Cmake 的安装和cmakeList的编写可以参照网上教程。



## 4.2 TinyXML

TinyXML是一个轻量级的C++XML解析库。 关于TinyXML使用的文章有很多，而且官方也有完整的文档，系统中也有已经编译好的libxmlextern.a 静态库。如果需要重新编译得到静态库文件可以去网上下载源代码按照官方文档说明进行编译， 并将编译的到的库文件放置到lib文件夹中，将头文件放置到tinyXML文件夹中。

## 4.3 gFlags

写服务程序时，如果需要提供命令行参数。传统的方法是手工解析argv参数，或者使用getopt函数。两种方法都比较费劲。使用Google gflags可以大大简化命令行参数处理。

其实系统中已经有编译好的库，在此说明如何编译获取gflags依赖库， 以防以后需要用到。

从官方地址<http://code.google.com/p/google-gflags/>下载gflags并安装。比如我下载的是1.5版本

wget  <http://google-gflags.googlecode.com/files/gflags-1.5.tar.gz>

tar zxvf gflags-1.5.tar.gz

cd gflags

./configure –prefix=$HOME/google-library

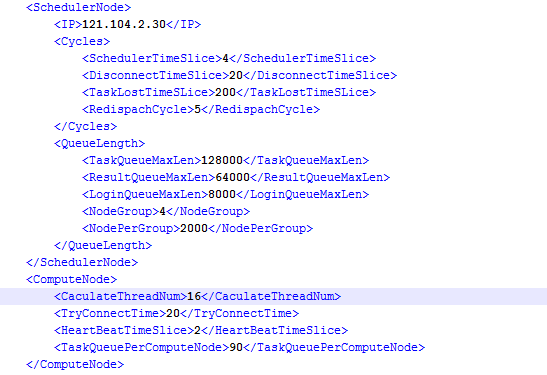
make j=4 & make install

这样可以得到gflags 编译得到的静态库文件libflags.a 将libflags.a 放到系统lib文件夹下, 将gflags 的头文件放到gflags文件夹下。

# 5.运行

## 5.1配置config 文件

在运行系统之前需要先写一个config.xml文件对系统进行配置，系统中默认的config.xml 如下所示



config 配置的参数主要分为调度节点的参数和计算节点的参数.

用户在使用的时候主要需要配置的参数是

1.调度节点的IP，要根据调度节点的IP填写。

2.计算节点中主要需要配置的是任务队列的长度taskQueuePerComputeNode, 这个参数要根据应用的任务计算时间长度进行配置，如果任务计算时间较短则需要增大任务队列的长度才能重复利用计算节点的性能。

3.cacluateThreadNum, 这个参数设置了计算节点开启的计算线程的数量，需要根据计算节点的性能和计算任务的情况具体问题具体分析。

## 5.2运行的参数

配置完config文件后，就可以启动程序了。由于程序需要的参数比较多，所以建议使用脚本或者一个启动程序将系统启动起来。

1. 启动调度节点

./bin/DCR 是可执行文件

参数is\_scheduler\_node 表示该节点是否为调度节点，调度节点为true, 计算节点为false

参数

original\_task\_name,

decomposer\_name,

result\_name,

computation\_task\_name,

granulity\_name,

job\_name 的值为用户定义的子类的类名

参数task\_file\_name 为用户任务文件的文件名

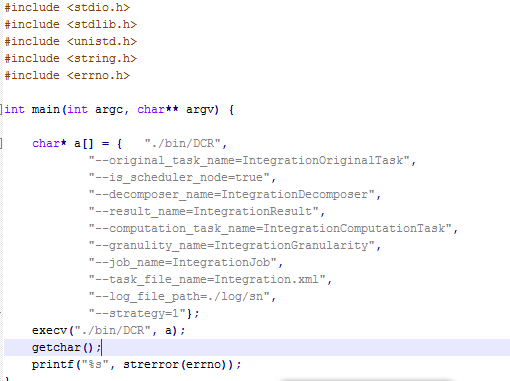
参数 log\_file\_path 则是用户希望系统日志文件保存的路径

参数 strategy 则是用户希望系统采用那种策略分发任务，1表示按照原来口令破解系统的分发策略，即随机分发， 2表示用户将根据类型信息将类型相同的任务分到同意节点上， 使用与IO密集型应用（比如blast）

1. 启动计算节点

使用的参数与调度节点一致， 只是参数is\_scheduler\_node的值为false

积分启动程序例子:



# 6.天河系统上的使用说明

## 6.1如何运行

yhrun -n 1 -N 1 -p work -w cn30 ./sn.o

-p 表示程序运行在哪个节点分区

-n 表示任务要在一个节点上运行有多少个实例(一般情况下设为1)

-N 表示任务要运行在多少个节点上

-w 表示要运行的节点的名称， 如果不提供这个选项，则由系统自动选择节点运行任务

最后一个参数是可执行文件名

## 6.2如何查看节点信息

用yhi 可以查看系统有哪些节点处于空闲状态

用yhq 可以查看系统现在都有哪些程序运行在哪些节点上

## 6.3如何取消作业

通过yhq 获取系统现在都有那些程序运行，然后可以通过yhkill –n sn.o 的方法取消sn.o 的运行，也可以通过yhkill –u hnlg 取消用户hnlg运行的所有程序

## 6.4如何让作业在后台运行

可用用screen 命令创建一个虚拟终端运行程序

或者用yhrun -n 1 -N 1 -p work -w cn30 ./sn.o& 在程序名后面加上&就可以将程序放在后台运行