DEVELOPER GUIDE FOR ABCGrid

Last update: Feb-10-2006, version: 0.1

Ying Sun
Center for Bioinformatics
College of Life Sciences
Peking University
Beijing 100871, P.R.China
suny@mail.cbi.pku.edu.cn

Architecture

ABCGrid is a server/client application consists of three independent applications: ABCMaster (manage job I/O), ABCWorker(process job) and ABCUser(send commands).

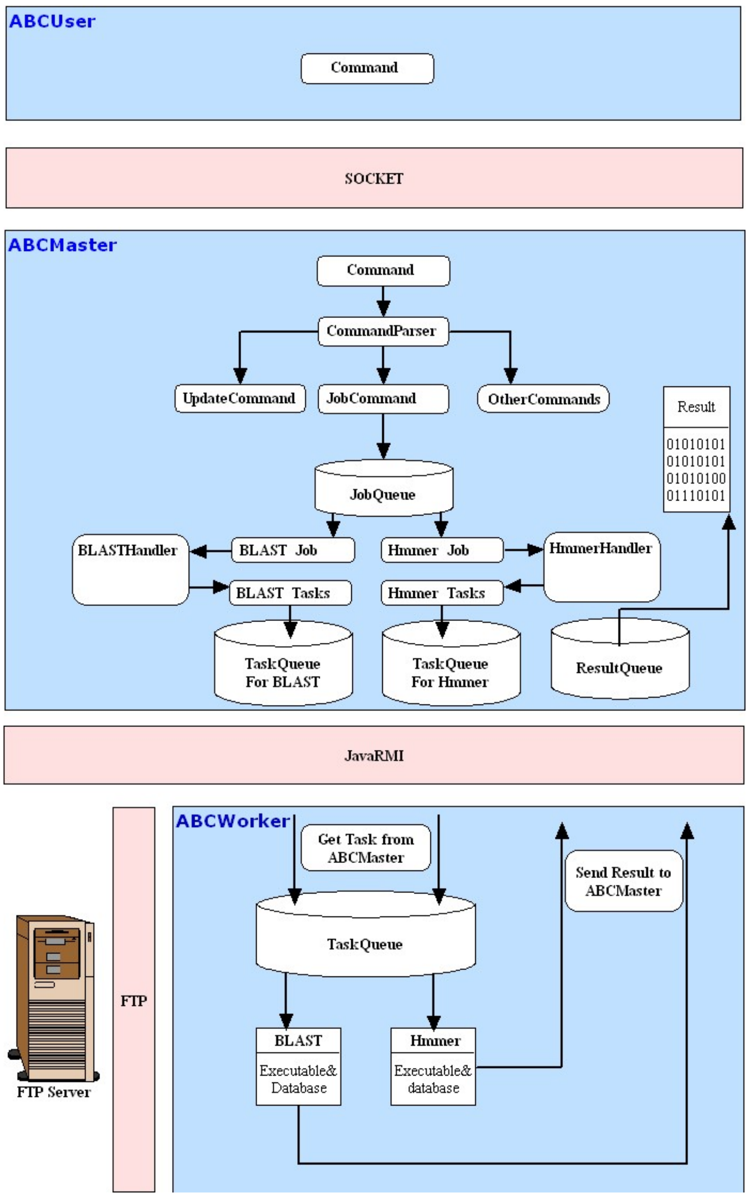

Figure. 1 The architecture of ABCGrid.

Tutorials

1. How to add support to new bioinformatics application

I take NCBI_BLAST's blastall as an example to demonstrate how to add a plug-in in ABCGrid, which is used to support the execution of "blastall" jobs by appending a package and a parser class in ABCMaster's source code.

First, let us see how to add support step by step.

1) Create a package named "NCBI_BLAST" under the package of ***pku.cbi.abcgrid.master.service.*** It's full name should be ***pku.cbi.abcgrid.master.service*.NCBI_BLAST**
2) Create a class named "blastall" under package "NCBI_BLAST" and derived this class from class ***pku.cbi.abcgrid.master.service.AbstractApp***, it's full name should be ***pku.cbi.abcgrid.master.service*.NCBI_BLAST.blastall**
3) Implement the code of class blastall.

Please see ABCMaster's source code for details of class blastall. In fact, blastall is derived from class NCBI_BLASTBase which is a higher layer of abstraction of all blastall-like applications share similar parameter specifications.

Before going into programming details, it will be helpful to understand how does ABCGrid process a job. Suppose user "demo" (whose priority is 10) inputs a command in a console window:

>>>r blastall –p blastp –i test.fasta –d nr –o test.blastp –m 9

First, ABCMaster recognizes this is a "run" (**r**) command issued from user "demo". Next, by using the parameter followed "**r**" (Here it is ***blastall***), ABCMaster will try to find a parser class named "blastall". Next, ABCMaster call class blastall's method to spawn a Job object and decompose that job object into several task objects (represented by class Task). So, basically, the parser class should do two things: 1) Spawns a Job object according to command string and 2)Decomposes a job into several tasks. OK, now let us see how to do these two things. You may noticed that all parser class have a common parental interface: ***pku.cbi.abcgrid.master.service.App***

```
/*interface pku.cbi.abcgrid.master.service.App */
package pku.cbi.abcgrid.master.service
import pku.cbi.abcgrid.master.Job;
import pku.cbi.abcgrid.master.Task;
import java.util.List;
public interface App
{
     public String getServiceName();
     public String getAppName();
    public Job parse(String user,String jobcmd,int priority);
    public List<Task> fragment(Job job);
}
```

Interface ***App*** defined four methods, namely *getServiceName(),getAppName(),parse()*

and *fragment()*. Let's check them one by one.

1. *String getServiceName()*
Since class blastall is derived from interface ***App***, it must implement all four methods. Thus, *getServiceName()* should return the name of the service, in this scenario, it is "NCBI_BLAST", the name of class blastall's parental package.

2. *String getAppName()*
Similarly, this method should return the name of the application, it is "blastall", the name of class "blastall".

3. *Job parse(String user,String jobcmd,int priority)*
This method has three parameters. Given above example, they are "demo", "blastall –p blastp –i test.fasta –d nr –o test.blastp –m 9" and 10 sequentially. This method is used to parse the command string into a job object. Then, what exactly is job? In ABCGrid, a job (class **Job**) consists of tasks (class **Task**), each task has task input (class **TaskInput)**, and a task input is a fragment of job input file. You must have clear understanding on the usage of blastall in order to parse its parameters. Basically, we care about five kinds of parameters:

1. The executable file
2. Input file(s)
3. Output file(s)
4. Database
5. Auxiliary file(s)

Other parameters are simple "tag parameters" which should be keep untouched.
Then, for command "blastall –p blastp –i test.fasta –d nr –o test.blastp –m 9"

1. The executable file: blastall
2. Input file: test.fasta
3. Output file: test.blastp
4. Database: nr
5. Auxiliary file: N/A

"-p blastp" and "-m 9" are just "tag parameters".
Since this command is executed in worker node, which has different directory structure, we must make some special translations which could be understand by ABCWorker. Then, here comes the translations rules:

1. Inserts "#" into the head of executable file name
2. Inserts "@" into the head of database name
3. Inserts "%" into the head of auxiliary file name

After applying translation rules, the original command:
"**blastall –p blastp –i test.fasta –d nr –o test.blastp –m 9**" **i**s translated into "**#blastall –p blastp –i test.fasta –d @nr –o test.blastp –m 9**". And the transferred command is sent to ABCWorker to execute. In ABCWorker, the command is translated again, each file's full path is mapped to the local path of the worker node. Finally, the ABCWorker will execute a command like:

"**/PATH/TO/NCBI_BLAST/executable/blastall -p blastp -i test.fasta -d /PATH/TO/NCBI_BLAST/db/nr -o test.blastp -m 9**". Because each task is executed in a separated working directory, that means I/O files are stored in different directory, you need not worry about the concurrent problem. By default, ABCGrid only use one CPU per worker node. If you want to use ALL CPUs, append parameter "-a" when you start the ABCWorker. For example,
"**java -jar worker.jar -h xxx.xx.xx.xx -a**" will use ALL CPUs of this worker node.

4. *List<Task> fragment(Job job)*
This method is used to decompose a job into several tasks. This work is done using package **abcgrid.cbi.pku.fragment.** ABCGrid does not depend on file name extension but a class **FormatDetector** to detect the format of input file. At present, FASTA, SwissProt, Genbank, EMBL and PDB are supported. If the file's format is one of them, corresponding class under **abcgrid.cbi.pku.fragment** is used to decompose the input file into several pieces of data. Each TaskInput takes a piece of data.

Note: An abstract class, ***AbstractApp***, derived from ***App***, provides a skeleton to implement a parser class and hide many details. Generally, you should derive your class from ***AbstractApp***.

For more details, please reference ABCMaster's source code.

2. How to add support to new input file format

Input file format include FASTA, EMBL, GenBank, etc. At present, ABCGrid supports FASTA, EMBL, GenBank, SwissProt and PDB. However, if you want to support other formats, you can write your own handler. Your handler class should derive from class **pku.abcgrid.master.fragment.BaseFragment**.