BioQueue Documentation

Release 1.0.0

Li Yao

Mar 09, 2018

Contents

BioQueue is a lightweight and easy-to-use queue system to accelerate the proceeding of bioinformatic workflows. Based on machine learning methods, BioQueue can maximize the efficiency, and at the same time, it also reduces the possibility of errors caused by unsupervised concurrency (like memory overflow). BioQueue can both run on POSIX compatible systems (Linux, Solaris, OS X, etc.) and Windows.

CHAPTER 1

How does BioQueue work?

One very conspicuous characteristic of data analyses in bioinformatics is that researchers usually need to analyze large amount of data by using the same protocol and other researchers may need to use this protocol to analyze their own data too. So improving the reusability of protocol is very crucial. To achieve this goal, BioQueue explicitly differentiates two concepts. One concept is “protocol”, which is a chain of steps consisting of software and its parameters that define the behavior of the analysis. When a “protocol” is assigned with specific experimental variables, like input files, output files or sample name, the protocol will turn into a runnable “job”. So to analyze data with BioQueue, you need to create a protocol first.

BioQueue is based on a chain of checkpoints which not only provide the support for reentrancy (The ability of a program to continue its execution from where it lefts off if interrupted, without restarting from the beginning of a process), but also estimate the resources (CPU, memory and disk usage) required by following steps. And if the system resources are abundant for the next step, BioQueue will execute it, otherwise the step has to wait until there are enough resources.

CHAPTER 2

Contents

2.1 Get Started

2.1.1 Prerequisites

BioQueue can store data on SQLite, which means users can set up BioQueue without an extra database software. However, to achieve a higher performance, we suggest users to install MySQL. For Windows users, download the MySQL Installer or Zipped binary from MySQL. For POSIX compatible systems (like Ubuntu) users, running the following command should be enough to install MySQL server:

```
sudo apt-get install mysql-server mysql-client
sudo apt-get install libmysqld-dev
mysql -u root -p
CREATE DATABASE BioQueue;
CREATE USER 'bioqueue'@'localhost' IDENTIFIED BY 'YOURPASSWORD';
GRANT ALL PRIVILEGES ON BioQueue . * TO 'bioqueue'@'localhost';
```

Please replace 'YOURPASSWORD' with your own password for the database!

Since BioQueue is written in Python 2.7, please make sure that you have installed Python and pip. The following instructions are for Ubuntu 14.04, but can be used as guidelines for other Linux flavors:

```
sudo apt-get install build-essential
sudo apt-get install python-dev
sudo apt-get install python-pip
```

2.1.2 Installation

First of all, clone the project from github (Or you can download BioQueue by open this link):

```
git clone https://github.com/liyao001/BioQueue.git
Or
wget https://github.com/liyao001/BioQueue/zipball/master
```

NOTE:Download archives rather than use git makes it more difficult to stay up-to-date with BioQueue code because there is no simple way to update the copy.

Then navigate to the project's directory, and run the `install.py` script (All dependent python packages will be automatically installed):

```
cd BioQueue
python install.py
```

When running `install.py`, this script will ask you a few questions including:

1. CPU cores: The amount of CPU to use. Default value: all cores on that machine.
2. Memory (Gb): The amount of memory to use. Default value: all physical memory on that machine.
3. Disk quota for each user(Gb, default value: all disk space on that machine).

If you decide to run BioQueue with MySQL, the script will ask a few more questions:

1. Database host: If you install MySQL server on your own machine, enter *localhost* or *127.0.0.1*.
2. Database user: user name of the database. *bioqueue* by default.
3. Database password: password of the database.
4. Database name: Name of the data table.
5. Database port: *3306* by default

Start the queue

Run `bioqueuepy` script in the *BioQueue/worker* folder:

```
python worker/bioqueue.py
```

For Linux/Unix users, BioQueue can run in the background by running `bioqueue_daemon.py` instead of `bioqueue.py`:

```
python worker/bioqueue_daemon.py start
```

Start web server

Run the following command to start the webserver:

```
python manage.py runserver 0.0.0.0:8000
```

This will start up the server on **0.0.0.0** and port **8000**, so BioQueue can be accessed over the network. If you want access BioQueue only in local environment, remove **0.0.0.0:8000**. If you want to use BioQueue in a production environment, we recommend you to use a proxy server instead of `manage.py` (see detail at FAQ).

Start FTP server

BioQueue provides a FTP server to make it more convenient to transfer files, to run this server:

```
python worker/ftpserver.py
```

This step is optional, if you run command above, the FTP server will listen **20001** port by default. For Linux/Unix users, BioQueue FTP service can run in background by run `ftp_daemon.py` instead of `ftpserver.py`:

```
python worker/ftp_daemon.py start
```

2.1.3 Useful informations

1. To stop the queue, the webserver or the FTP server, just hit Ctrl-c in the terminal from which BioQueue is running. If you run the queue or FTP server in the background, hit:

   ```
   python worker/bioqueue_daemon.py stop
   python worker/ftp_daemon.py stop
   ```

2. To get a better performance, moving the webserver to Apache or nginx is a good idea.

2.2 Protocol

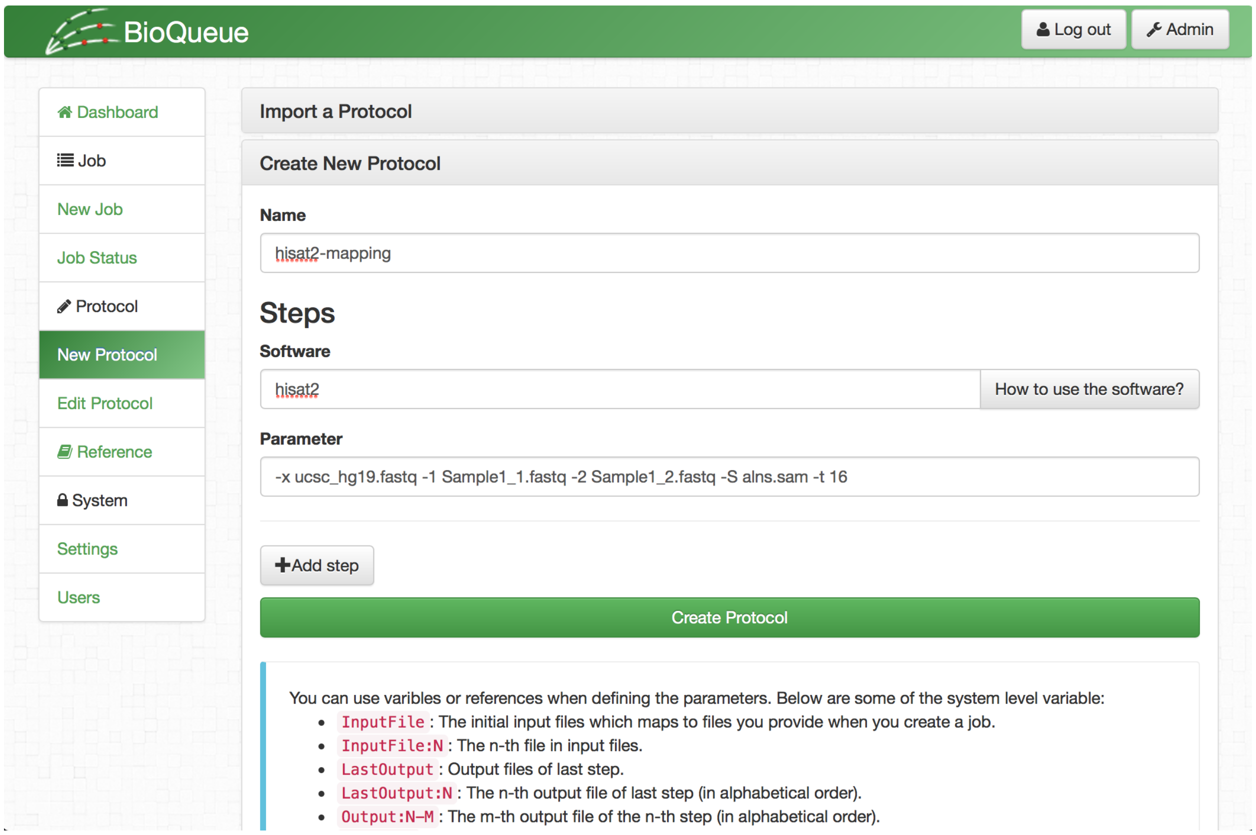
Like wet experiments, a protocol (or workflow, pipeline in other software) in bioinformatics is built from several linked steps that consume inputs, process and convert data and produce results. To analyze data, you need to create a protocol to tell BioQueue what it should do for you.

2.2.1 Create a Protocol

To create a new protocol, you can either click `Create Protocol` button at the homepage or click `New Protocol` at the sidebar. Since neither programming nor software engineering is included in a typical biological curriculum, making the syntax as easy as possible is very necessary. When creating a protocol, BioQueue implements the shell command-like syntax rather than a domain-specific language (DSL). Basically, each step of the protocol consists of software and its parameter (which can easily get from the software's documentation). For example, if you want to use HISAT2 to map reads produced by next-generation sequencing, the original command may like this:

```
hisat2 -x ucsc_hg19.fastq -1 Sample1_1.fastq -2 Sample1_2.fastq -S alns.sam -t 16
```

In BioQueue, you need to enter the software ("hisat2") into `Software` textbox, and then enter "-x ucsc_hg19.fastq -1 reads_1.fastq -2 reads_2.fastq -S alns.sam -t 16" into `Parameter` textbox.



Name: hisat2-mapping

Steps

Software: hisat2

Parameter: -x ucsc_hg19.fastq -1 Sample1_1.fastq -2 Sample1_2.fastq -S alns.sam -t 16

You can use varibles or references when defining the parameters. Below are some of the system level variable:

- `InputFile` : The initial input files which maps to files you provide when you create a job.
- `InputFile:N` : The n-th file in input files.
- `LastOutput` : Output files of last step.
- `LastOutput:N` : The n-th output file of last step (in alphabetical order).
- `Output:N-M` : The m-th output file of the n-th step (in alphabetical order).

Actually, a typical protocol contains many steps, so you can click the `Add Step` button to add more step. After you added all steps, click `Create Protocol` button.

2.2.2 Make Protocol Reusable

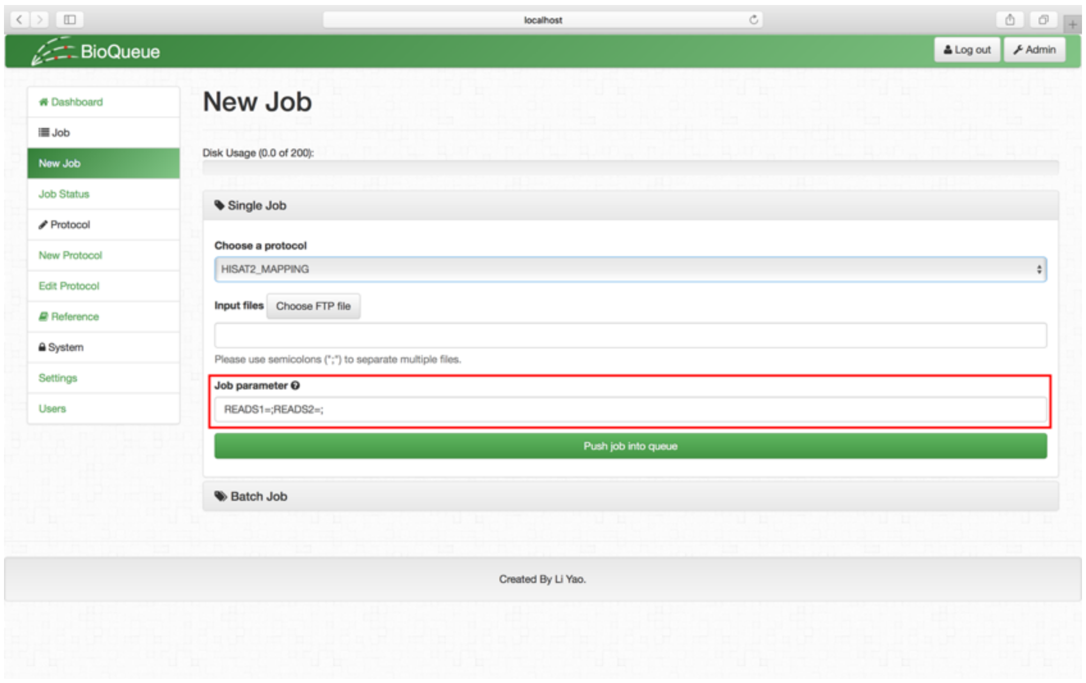
Though BioQueue explicitly separates the concepts of "protocol" and "job", a protocol can convert into a runnable job without assigning any experimental values. However, we strongly recommend you replace those values with wildcards (strings embraced with braces, like "{ThreadN}") to make the protocol reusable and reproducible. For instance, if you want to utilize HISAT2 to map reads, you can replace "Sample1_1.fastq" with {READS1}, and replace "Sample1_2.fastq" with {READS2}. So, the protocol may like this:

```
hisat2 -x ucsc_hg19.fastq -1 {READS1} -2 {READS2} -S alns.sam -t 16
```

And then you should tell BioQueue the value of the two wildcards when you create a job by entering:

```
READS1=Sample1_1.fastq;READS2=Sample1_2.fastq;
```

into `Job Parameter` textbox.



New Job

Disk Usage (0.0 of 200):

Single Job

Choose a protocol: HISAT2_MAPPING

Input files: Choose FTP file

Please use semicolons (";") to separate multiple files.

Job parameter: READS1=;READS2=;

Push job into queue

Batch Job

Created By Li Yao.

Note: BioQueue will automatically load all the experimental variables into the `Job parameter` textbox.

And if you want to analyze one more sample (for example Samples), you just need to type:

```
READS1=Sample2_1.fastq;READS2=Sample2_2.fastq
```

Otherwise, you will have to create a new protocol.

2.2.3 Mapping Files Between Steps

In most cases, a step denotes how to create output files from input files. Since a protocol usually consists of many steps, making mapping of files flexible enough is a very important issue. So BioQueue provides three types of file mapping methods to do this.

The first method is to write the file name directly. For some tools, the output files have standard names. One example is STAR, when it finished mapping RNA-SEQ reads, it would produce those files:

1. Log.out
2. Log.progress.out
3. Log.final.out
4. Aligned.out.sam
5. SJ.out.tab

So, if you want to use the mapped SAM file in the following steps, you can enter "Aligned.out.sam" straightforwardly.

The second method is to use the **"Suffix"** wildcard. For example, if you want to use the SAM file produced by the previous step, you can enter *"{Suffix:sam}"*. If you would like to use the **"Suffix"** wildcard to map file from any steps before rather than the last step, **"Suffix:N-FileSuffix"** might be very helpful ("N" means the n-th steps).

The third method is to use the **"Output"** family wildcards. Here is a table of those wildcards:

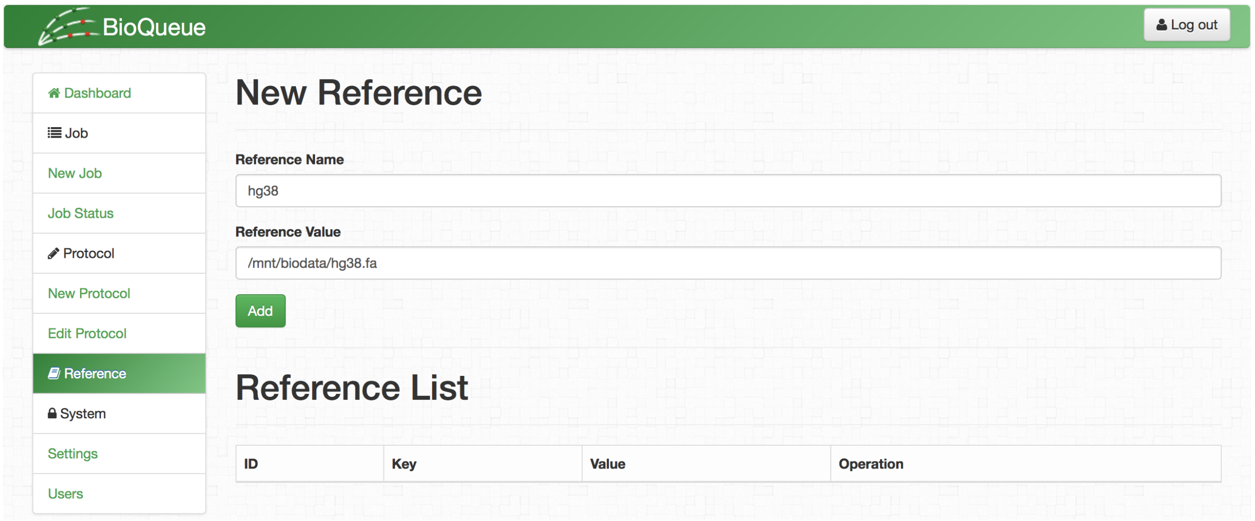
Wildcard	Description
InputFile	The initial input files which maps to files you provide when you create a job.
InputFile:N	The n-th file in input files.
LastOutput	Output files of last step.
LastOutput:N	The n-th output file of last step (in alphabetical order).
Output:N-M	The m-th output file of the n-th step (in alphabetical order)+.
AllOutputBefore	All output files before this step.

2.2.4 More About Wildcards

There are two main types of wildcards in BioQueue: pre-defined wildcards and user-defined wildcards (experimental variables and reference). Below is a table of pre-defined wildcards:

Wildcard	Description
Suffix:X	Output file(s) with a "X" suffix in the last step.
Suffix:N-X	Output file(s) with a "X" suffix in the n-th step.
Suffix:N-X-M	The m-th output file with a "X" suffix in the n-th step.
InputFile	The initial input files which maps to files you provide when you create a job.
InputFile:N	The n-th file in input files.
LastOutput	Output files of last step.
LastOutput:N	The n-th output file of last step (in alphabetical order).
Output:N-M	The m-th output file of the n-th step (in alphabetical order).
History:N-X	Output file "X" in the n-th job.
AllOutputBefore	All output files before this step.
Job	Job ID.
Workspace	Local storage path for the job.
ThreadN	The number of CPUs in the running node.

Now, let's have a look at the user-defined wildcards. As we mentioned before, BioQueue suggests users use wildcards to denote experimental variables in protocols, like sample name. This type of user-defined wildcards need to be assigned as `Job parameter` when creating jobs. In bioinformatics, there are some data that can be cited in different protocols, such as reference genome, GENCODE annotation, etc. So, in BioQueue, biological data that may be used in multiple protocols is called a "reference". This is the other type of user-defined wildcards and it is defined at the `Reference` page.



New Reference

Reference Name: hg38

Reference Value: /mnt/biodata/hg38.fa

Reference List

ID	Key	Value	Operation

Use Reference

The aim of implementing the concept of "reference" is to reduce the redundancy of protocols (see the table below), consequently, references are available to all protocols of the user. We recommend creating references for genome files, SNP annotation files and gene annotation files, etc. Let's suppose you have a human reference genome file "hg38.fa" in the folder "/mnt/biodata/reference_genome", thus you can type "HG38" in `Reference Name` textbox and assign the value "/mnt/biodata/reference_genome/hg38.fa" into the `Reference Value` textbox.

Here is a table showing how the usage of reference can reduce the redundancy of protocols.

	Without Reference	With Reference
One step in protocol A	hisat2-build `/mnt/biodata/reference_genome/hg38.fa` genome	hisat2-build `{HG38}` genome
One step in protocol B	java -jar gatk.jar -T HaplotypeCaller -R `/mnt/biodata/reference_genome/hg38.fa` -I input.bam -dontUseSoftClippedBases -stand_call_conf 20.0 -stand_emit_conf 20.0 -o output.vcf	java -jar gatk.jar -T HaplotypeCaller -R `{HG38}` -I input.bam -dontUseSoftClippedBases -stand_call_conf 20.0 -stand_emit_conf 20.0 -o output.vcf

Note: Don't forget to add braces before you use a reference in any of your protocol, like `{HG38}`!

Shortcuts for Creating Reference

To facilitate our users to create a reference with ease, we provide some shortcuts. Both uploaded files and job output files can be set as a reference by clicking the `Create a reference` button on either `Sample Pool` page, which can be accessed through clicking the `Sample Pool` button in index, or `Job Status` page (When you click on the output folder, you will be able to see the button in the new dialog.)

2.2.5 Create a Protocol with Ease

To help general biologists to create a protocol with ease, we provide auxiliary functions which cover the entire process.

1. Knowledge Base

We set up a knowledge base on our open platform, so when our users need to search the usage information about a certain software, they can click the `How to use the software?` button.

2. Autocomplete

We provide an autocomplete widget to provide suggestions about pre-defined wildcards and user-defined references. Here is a demo:

In the demo, {HISAT2_HG38} is a user-defined reference, which refers to the path of hg38 indexes for HISAT2. While {InputFile:1}, {InputFile:1} and {ThreadN} are pre-defined wildcards.

2.2.6 Edit Steps

When you need to change parameters of a certain step, you should click `Edit Protocol` at the sidebar. Then you move mouse to `Operation` column where the protocol locates in, and click the `Edit Protocol` label.

When the steps' table shows up, you can click the parameter of the step. Now you can edit the parameter. Once you click any place at that page, your changes will be saved automatically.

2.2.7 Share Protocol With Peer

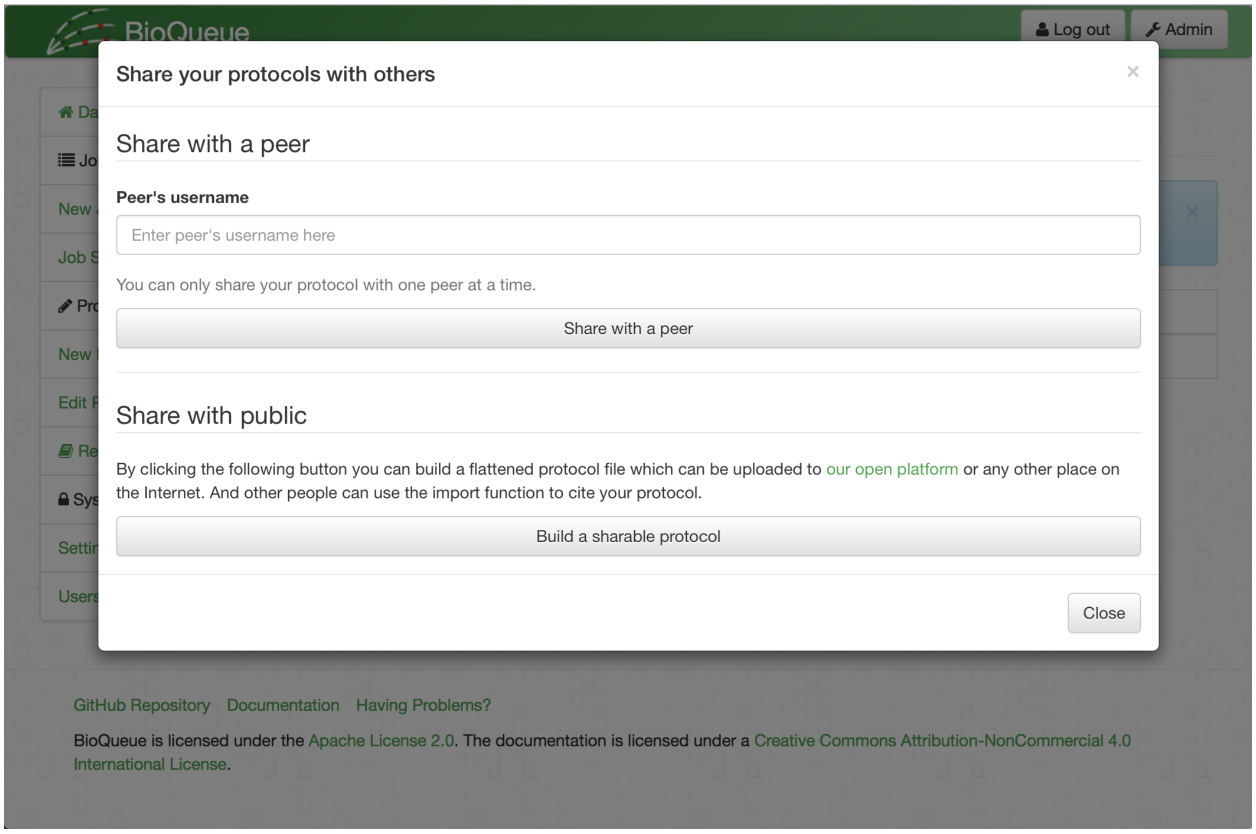
We know the importance of making computational analysis in life sciences:

1. Easily to get started for researchers who do not have a strong background in computer science (accessibility);
2. Easily to reproduce the experimental results;

So, protocols written by BioQueue can be shared with a peer who are using the same platform, and BioQueue can generate a portable protocol file which can be published on the Internet.

To share a protocol with a peer, you need to open the `Edit protocol` page, and choose `Share` in the `Operation` column.

Then enter username of the peer you want to share with, and click `Share with a peer`.



You can only share your protocol with one peer at a time.

By clicking the following button you can build a flattened protocol file which can be uploaded to our open platform or any other place on the Internet. And other people can use the import function to cite your protocol.

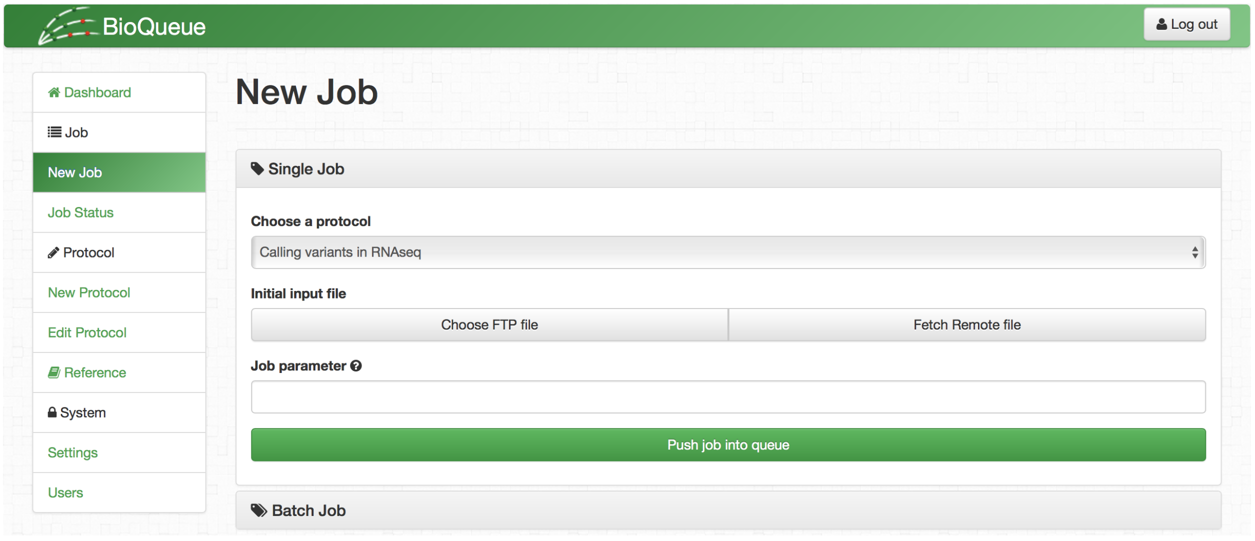
BioQueue is licensed under the Apache License 2.0. The documentation is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

To share a protocol with the public, you need to open the same dialog, and click the `Build a sharable protocol` button, then a protocol file would be generated. You can publish this protocol on BioQueue Open Platform or any other web forums.

2.3 Job

2.3.1 Create a Job

To create a new job, you can either click the `Create New Job` button at the homepage or click the `New Job` link at the sidebar.



Firstly, you need to choose a protocol for this job. To make a protocol reusable, you might have placed some wildcards in your protocol, so you need to assign values to those wildcards, like input files, sample name, etc. As mentioned before, BioQueue provides two pre-defined wildcards `InputFile` and `InputFileN` for mapping input files to a protocol. Those two wildcards are defined in `Input files` textbox. In BioQueue, there are three types of input files:

1. FTP file: Files uploaded through BioQueue FTP server. To make sure the safety of your data, BioQueue will hide the store path of those files. When you need to use those files, you can click the `Choose FTP file` button, then select files you need. Or, you can manually type `{Uploaded:FILENAME}` into `Input files` textbox.
2. Local file: For users who only use BioQueue in local environment, there's no need to transfer files through FTP. So, you can put your files at some place, and then use the absolute path of the file, like `/mnt/biodata/RNAseq/sample1_1.fastq`.
3. Remote file: For security, BioQueue will not fetch remote files by default. So to use these files, you may need to add a download step in your protocol, like `wget {InputFile}`.

When you want to add more than one file into `Input files` textbox, do not forget to add a semicolon at the end of each file. For example:

```
{Uploaded:ERR030878_1.fastq.bz2};{Uploaded:ERR030878_2.fastq.bz2};
```

If you use `{InputFile}` in a step, then it will be replaced by `user_ftp_store_path/ERR030878_1.fastq.bz2 user_ftp_store_path/ERR030878_2.fastq.bz2`, `{InputFile1}` will be replaced by `user_ftp_store_path/ERR030878_1.fastq.bz2`, and `{InputFile2}` will be replaced by `user_ftp_store_path/ERR030878_2.fastq.bz2`.

For user-defined wildcards, you can declare them in the `Job parameter` textbox by entering `wildcard=value;` . For instance, in our demo protocol, there are two user-defined wildcards (SAMPLENAME and SEQMACHINE), if you want assign `ERR030878` to `SAMPLENAME`, and `HighSeq2000` to `SEQMACHINE`, enter the following text into `Job parameter` textbox:

```
SAMPLENAME=ERR030878;SEQMACHINE=HiSeq2000;
```

In the end, click the `Push job into queue` button.

2.3.2 Create Batch of Jobs

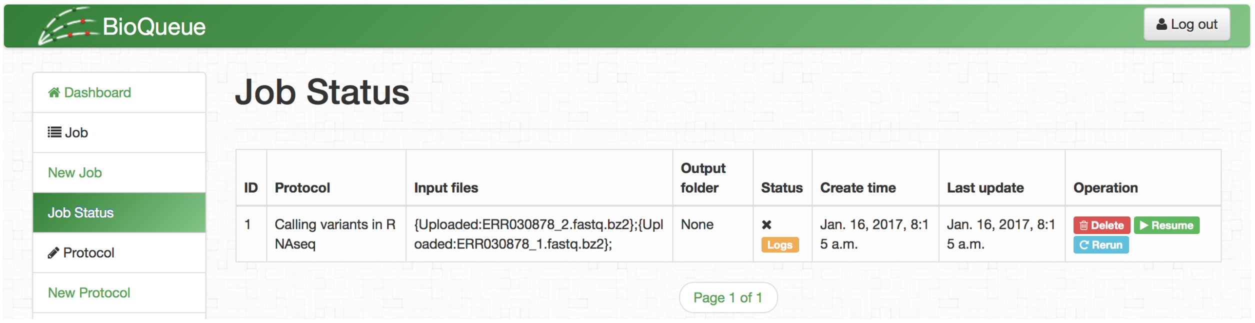
In most cases, we need to analyze multiple samples with the same protocol. And it's usually tedious to create jobs one by one. So BioQueue provides a very convenient way to do it. You can provide a file containing three columns to describe the jobs you want to create. Here are those three columns:

1. Protocol Id, which can be found in the `Edit Protocol` page.
2. Input files.
3. Job parameter.

Then click `Upload and create` button. Note: columns should be separated by `Tab (\t)`

2.3.3 Reentrancy

When we manually terminate a job or some errors occur that cause the termination of a job, we might want to restart the job by directly running the step where the job was terminated rather than rerun all the steps, which is so-called reentrancy. In BioQueue, there are checkpoints before the execution of each step, so when click the `Resume` tag in the `Operation` column, the job will be automatically restart from the step where the termination occurs.

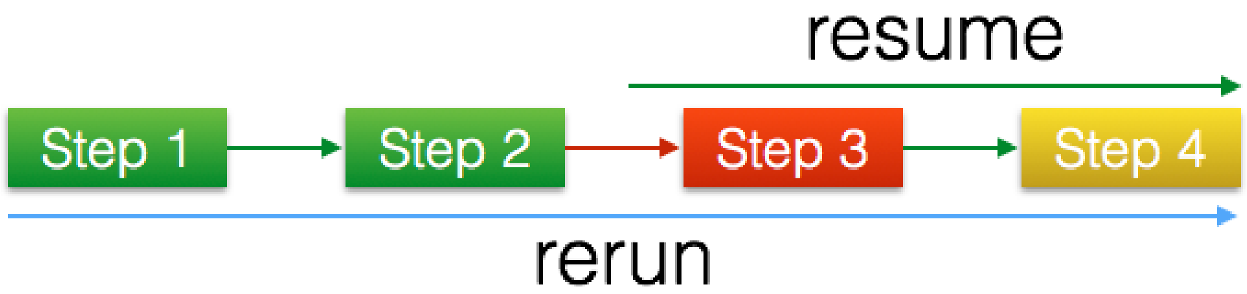


Job Status

ID	Protocol	Input files	Output folder	Status	Create time	Last update	Operation
1	Calling variants in RNAseq	{Uploaded:ERR030878_2.fastq.bz2};{Uploaded:ERR030878_1.fastq.bz2};	None	✖ Logs	Jan. 16, 2017, 8:15 a.m.	Jan. 16, 2017, 8:15 a.m.	Delete Resume Rerun

Page 1 of 1

Here is a diagram showing the difference between `rerun` and `resume`.



2.4 Examples

2.4.1 Transcript-level expression analysis of RNA-seq experiments with HISAT, StringTie

This protocol is created according to this nature protocol paper:

Software	Parameter
hisat2	-p {ThreadN} `--dta` -x {HISAT2_HG38} -1 {InputFile:1} -2 {InputFile:2} -S {EXP}.sam
samtools	sort -@ {ThreadN} -o {EXP}.sorted.bam {EXP}.sam
stringtie	-p {ThreadN} -G {GENCODE_HG38} -o {EXP}.gtf -l {EXP} {EXP}.sorted.bam

Reference settings

Reference Name	Reference Value
HISAT2_HG38	The folder stores hg38 genome index for hisat2, this is generated by `hisat2-build`
GENCODE_HG38	Path of a gene annotation file, like `/mnt/biodata/gencode_hg38_v23.gtf`

Job Initial

Input files:

```
{Uploaded:ERR033015_1.fastq};{Uploaded:ERR033015_2.fastq};
Or /mnt/biodata/samples/ERR033015_1.fastq;/mnt/biodata/samples/ERR033015_2.fastq
```

Job parameter:

```
EXP=ERR033015;
```

2.4.2 Calling variants in RNAseq (STAR-gatk)

This protocol is created according to gatk's Best-Practices provided by Broad Institute:

Software	Parameter
STAR	–genomeDir {STAR_HG38} –readFilesIn {InputFile:1} {InputFile:2} –runThreadN {ThreadN}
STAR	–runMode genomeGenerate –genomeDir 2pass –genomeFastaFiles {HG38} –sjdbFileChrStartEnd SJ.out.tab –sjdbOverhang 75 –runThreadN {ThreadN}
STAR	–genomeDir 2pass –readFilesIn {InputFile:1} {InputFile:2} –runThreadN {ThreadN}
java	-jar {picard} AddOrReplaceReadGroups I=Aligned.out.sam O=rg_added_sorted.bam SO=coordinate RGID={RGID} RGLB={RGLB} RGPL={RGPL} RGPU={RGPU} RGSM={RGSM}
java	-jar {picard} MarkDuplicates I=rg_added_sorted.bam O=dedupped.bam CREATE_INDEX=true VALIDATION_STRINGENCY=SILENT M=output.metrics
java	-jar {gatk} -T SplitNCigarReads -R {HG38} -I dedupped.bam -o split.bam -rf ReassignOneMappingQuality -RMQF 255 -RMQT 60 -U ALLOW_N_CIGAR_READS
java	-jar {gatk} -T HaplotypeCaller -R {HG38} -I input.bam -dontUseSoftClippedBases -stand_call_conf 20.0 -stand_emit_conf 20.0 -o output.vcf
java	-jar {gatk} -T VariantFiltration -R {HG38} -V output.vcf -window 35 -cluster 3 -filterName FS -filter "FS > 30.0" -filterName QD -filter "QD < 2.0" -o output.hard.filtered.vcf

Reference settings

Reference Name	Reference Value
STAR_HG38	The folder stores hg38 genome index for star, this is generated by command `STAR --runMode genomeGenerate`
HG38	Path of a reference genome file, like `/mnt/biodata/hg38.fa`
picard	Path of picard.jar, like `/mnt/biosoftware/picard.jar`
gatk	Path of GenomeAnalysisTK.jar, like `/mnt/biosoftware/GenomeAnalysisTK.jar`

Job Initial

Input files:

```
{Uploaded:ERR033015_1.fastq};{Uploaded:ERR033015_2.fastq};
Or /mnt/biodata/samples/ERR033015_1.fastq;/mnt/biodata/samples/ERR033015_2.fastq
```

Job parameter:

```
RGID=4;RGLB=lib1;RGPL=illumina;RGPU=unit1;RGSM=20;
```

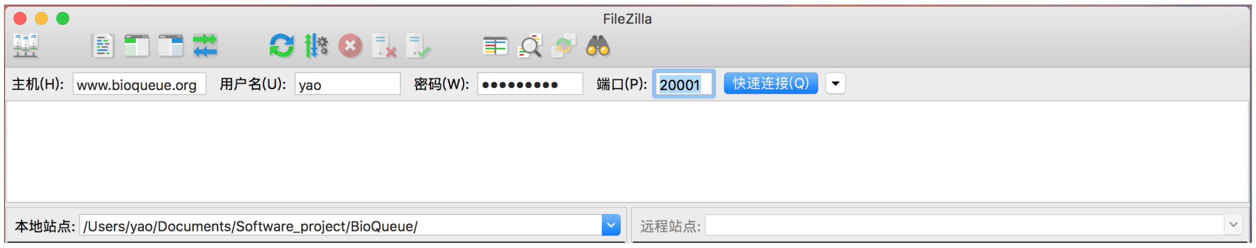
2.5 FAQ

2.5.1 Upload file

When uploading datasets, the files are usually cached in memory till the upload is finished. In bioinformatics research, files are large in most cases, so it may eat up memory. To avoid this risk, BioQueue provide a FTP service. The administrator can start the service by running:

```
python ftp_daemon.py start
Or
python ftpserver.py
```

And users can use a ftp client (FileZilla, the free FTP solution) to access this service. Files will be uploaded to user's upload folder, and those files can be selected by click the `Choose FTP file` in the `Create Job` page. Note: the user name and password are identical to those in BioQueue web platform. The default port is **20001** not **21**.



主机(H): www.bioqueue.org 用户名(U): yao 密码(W): ••••••••• 端口(P): 20001

本地站点: /Users/yao/Documents/Software_project/BioQueue/

远程站点:

2.5.2 How to update BioQueue

We will bring new features and fix bugs when we release a new version. So we recommand you to keep your instance as new as possible. If you have an BioQueue repository and want to update it, there are several ways to do so.

1. Run update.py in worker folder

We provide a python script named as `update.py` in `worker` folder, which will check updates for both BioQueue's source code and dependent packages:

```
python worker/update.py
```

Also, for Linux/Unix users, BioQueue update service can run in background by run `update_daemon.py` instead of `update.py`:

```
python worker/update_daemon.py start
```

This service will check for update everyday.

2. Click Update button in `Settings` page

We also provide an update button in the `Settings` page, clicking the button, BioQueue will call `update.py` to update your instance.

3. git pull

You can also use git pull command to update BioQueue's source code, but this command won't update the dependent packages!

4. NOTE

The update service relies on git, so please make sure that you have installed git and you cloned BioQueue from GitHub.

2.5.3 Use BioQueue with Apache in Production Environment

To host BioQueue with Apache, you must first install the `mod_wsgi` Apache module. On Ubuntu, you can install this as follows:

```
sudo apt-get install libapache2-mod-wsgi
```

If you have not installed Apache, before running command above, you need to setup Apache server by running this command:

```
sudo apt-get install apache2
```

You can then find an example VirtualHost file located at `deploy/000-default.conf`. Copy this file to `/etc/apache2/sites-available/` and restart the Apache server by running (on Ubuntu):

```
sudo /etc/init.d/apache2 restart
```

Note: For virtualenv users, please replace `/usr/lib/python2.7/dist-packages` with `/path/to/venv/lib/python2.7/site-packages`.

2.5.4 Cannot install MySQL-python?

By default, BioQueue will use a python package called MySQL-python to connect to MySQL server. However, it may be hard to install it especially for non-root users. The alternative solution is to use PyMySQL (a pure python mysql client). We provide a python script in BioQueue's `deploy` folder to help you to complete the switch. So for most of our users, the following command should be enough to solve this problem:

```
python deploy/switch_from_MySQLdb_to_PyMySQL.py
```

However, if you want to try it yourself, here is the protocol:

1. Remove `MySQL-python==1.2.5` from `prerequisites.txt` in `deploy` folder.
2. Copy the python code and paste them into the begining of `manage.py` and `worker >> __init__.py`:
3. Rerun `install.py`.

Code:

```
try:
    import pymysql
    pymysql.install_as_MySQLdb()
except ImportError:
    pass
```

2.5.5 Cannot access BioQueue due to firewall settings

Sometimes, the firewall installed by the administrators of the cluster may block web access outside the cluster. Under this circumatance, build a tunnel by hiring `ssh` should solve the problem. Here is an example:

```
ssh -N -L ${PORT}:localhost:${PORT} ${USER_NAME}@{REMOTE_ADDRESS}
```

Please replace `${PORT}`, `${USER_NAME}`, `${REMOTE_ADDRESS}` with your own answers. If you use BioQueue's default settings, `${PORT}` should be replaced with `8888`.

2.5.6 Turn on E-mail Notification

We know that keeping an eye on watching those time-consuming jobs is very tedious, so BioQueue provides an E-mail notification for changes among job status. By default, e-mail notification is silent. To turn on this push service, you need to fill in a form in `Settings >> Notification`. If the `Mail host` textbox is left to be blank, then the service will be silent, otherwise BioQueue will send a mail to you when a job is finished or an error is raised. And then you can configure it as a mail client.

2.6 Cluster Specification

When running on cluster, BioQueue corporates with the original DRMs and associates them to allocate proper resources for jobs. In this page, we will introduce:

1. How to use BioQueue in cluster
2. How to develop new cluster plugins for BioQueue
3. Problems you may encounter with

2.6.1 How to use BioQueue on clusters

The usage of BioQueue on clusters is identical to local or clouds. The only thing you need to do extraly is to tell BioQueue you are using a cluster. Here is the protocol.

1. Install BioQueue following the protocol mentioned before.
2. Run BioQueue web server.
3. Login to BioQueue and open `Settings`.
4. Click `Cluster Settings` in the page and fill in the form. By default, the value for `Cluster engine` is `Run on local / cloud` and all options for clusters are disabled. Once you choose a cluster engine (For example, TorquePBS), the cluster model for BioQueue will be activated. To turn it off, change cluster engine back to `Run on local / cloud`.
5. Click `Save changes` to save your changes.
6. *Start the queue*.

In `Cluster Settings` section, we provide some options. Here is a more detailed explanation for them.

Option	Default	Description
CPU cores for single job	1	Specify the number of virtual processors (a physical core on the node or an “execution slot”) per node requested for this job.
Physicial memory	No limit	Maximum amount of physical memory used by any single process of the job.
Virtual memory	No limit	Maximum amount of virtual memory used by all concurrent processes in the job.
Destination	Default server	Defines the destination of the job. The destination names a queue, a server, or a queue at a server.
Wall-time	No limit	Maximum amount of real time during which the job can be in the running state.

For example, when BioQueue submits job on a cluster managed by TorquePBS, the options defined above will be translated into Torque parameters like this:

1. -l ppn: CPU cores BioQueue predicts the job will take, if the prediction model has not been generated, the ppn option is equal to `CPU cores for single job`.
2. -l mem: The physical memory BioQueue predicts the job will use, if the prediction model has not been generated, the mem option is equal to `Physicial memory`.
3. -l vmem: The virtual memory BioQueue predicts the job will use, if the prediction model has not been generated, the vmem option is equal to `Virtual memory`.
4. -q: The destination defined in `Destination`, for example, if the cluster has five queues: high, middle, low, FAT_HIGH and BATCH, the destination should be one of them.
5. -l walltime: Maximum amount of real time during which job can be in the running state defined in `Wall-time`. For example, 24:00:00.

2.6.2 How to develop new cluster plugins for BioQueue

The support for clusters depends on the corresponding plugins. All python files in the folder `worker>>cluster_models` will be seen as a plugin. BioQueue can work with these plugins directly and users do not need to modify any code of BioQueue. However, limited by the clusters we can access, BioQueue now just provides build-in plugins for TorquePBS (carefully tested under production environment) and HTCondor (not tested under production environment). So we hope our users who have the privilege to access other types of DRMs can involve in the development of cluster plugins. Here we provide a detailed documentation for developing new plugins for BioQueue.

1. Coding conventions

1. **Required**: All plugins files are written in Python or at least provide a wrapper written in Python. The file name should be identical to the DRM's name and other supplementary files should use the DRM's name with different suffix. Take TorquePBS for example, the plugin file should be named as *TorquePBS.py* and if you use an extra file as the template for job script, the name of it should be *TorquePBS.tpl*.
2. Suggested: Function name or variable name should follow the style guide for Python code stated in PEP 8. In brief, both function name and variable in function should be lowercase.
3. Suggested: Two blank lines between two function block are expected.

2. Functions should be implemented in the plugin

The plugin must provide three functions for BioQueue to call. When you develop a new plugin, **REMEMBER TO FOLLOW THE PARAMETER LIST WE PROVIDE BELOWE!!**

submit_job

This function enables BioQueue to submit a job to the cluster via a DRM. The prototype of the function is:

```
submit_job(protocol, job_id, job_step, cpu=0, mem='', vrt_mem='', queue='', log_file='
↪', wall_time='', workspace='')
```

param protocol string, the command a job needs to run, like "wget http://www.a.com/b.txt"

param job_id int, job id in BioQueue, like 1

param job_step int, step order in the protocol, like 0

param cpu int, cpu cores the job will use

param mem string, allocated physical memory, eg. 64G.

param vrt_mem string, allocated virtual memory, eg. 64G.

param queue string, job queue

param log_file string, path to store the log file

param wall_time string, cpu time

param workspace string, the initial directory of the job, all output files should be stored in the folder, or the users will not be able to see them

return int, if success, return job id in the cluster, else return 0

Note: BioQueue will assign '' to both mem and vrt_mem if the user doesn't define the max amount of pyhsical memory or virtual memory a job can use and there are no prediction model to predict the amount of resource the job will occupy.

Note: protocol here is just a single step in the protocol defined in BioQueue

query_job_status

This function provides BioQueue an interface to query the status of a job. The prototype of the function is:

```
query_job_status(job_id)
```

param job_id int, job id in the cluster

return int, job status

If the job has completed, the function should return 0. If the job is running, it should return 1. If the job is queuing, it should return 2. If an error occurs during the execution of a job, it should return a negative number.

cancel_job

The function allows BioQueue to terminate the execution of a job. The prototype of the function is:

```
cancel_job(job_id)
```

param job_id int, job id

return if success, return 1, else return 0

3. Share the plugin with everyone

To share your plugin with other people, please fork BioQueue at github, and copy the plugins files into `worker>>cluster_models`. Then you can start a pull requests. Once we receive your pull requests, we will validate it as soon as possible. After that your plugin will be available for everyone.

2.6.3 Problems you may encounter with

1. Install python 2.7 or high and pip without root privilege

Cluster users usually do not have root privilege, and the python installation may be out-of-date. So it may be hard for biologists to configure the python environment for BioQueue, here we provide a helper script in *deploy/python_pip_non_root.sh*. This shell script will download source code of Python 2.7.13 from Python.org and compile it on the machine. You can run the script by running:

```
cd deploy
chmod +x python_pip_non_root.sh
./python_pip_non_root.sh
```

If the compile process failed, you can download a pre-built binary from ActiveState straightforwardly. NOTICE: the pre-built binary from ActiveState cannot be used in production environment.

After installation, you can add the bin directory to your PATH environment variable for quicker access. For example, if you use the Bash shell on Unix, you could place this in your ~/.bash_profile file (assuming you installed into /home/your_name/bin):

```
export PATH=/home/your_name/bin:$PATH
```

then save the .bash_profile file and run:

```
source ~/.bash_profile
```

Now you should be able to run BioQueue with the new Python.

2. Cannot run BioQueue with sqlite on clusters

Before answer the question, we highly recommand that all users use MySQL rather than SQLite. When running BioQueue on a cluster with Network File System (NFS), you may get an error message like:

```
django.db.utils.OperationalError: disk I/O error
```

The reason for this error is that SQLite uses reader/writer locks to control access to the database, while those locks are unimplemented on many NFS implementations (including recent versions of Mac OS X). So the only solution is to use a database software like MySQL.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search