HOWTO-For Setting Up Biopipe

Shawn Hoon

shawnh@fugu-sg.org

This is a HOWTO written in DocBook (SGML) for the installation of the biopipe, its workings and definition.We will describe how to load and run a simple blast pipeline.

Table of Contents

Introduction

The bioperl pipeline framework is a flexible workflow system that complements the bioperl package in providing job managment facilities for high throughput analysis. This system is heavily inspired by the EnsEMBL Pipeline system. This section describes the design of the bioperl pipeline. Some features of the current system include:

- Handling of various input and output data formats from various databases.
- A bioperl interface to non-specific loadsharing software (LSF,PBS etc) to ensure that the various analysis programs are run in proper order and are successfully completed while re-running those that fail.
- A flexible pluggable bioperl interface that allows programs to be ’pipeline-enabled’.

Setting up bioinformatics pipeline is not trivial. This tutorial introduces some aspects of biopipe through setting up a very siimple blast pipeline. It is hope that through this tutorial, two objectives are achieved:

- Iron out installation issues using a simple example.
- Familiarization with the biopipe system and introduction of the XML system we have develop to ameliorate some of the complexities involve in setting up.

Installation

The following sections descibes how to install your pipeline from start to finish

System Requirements

Note these are the requirements needed for running the simple blast example. Each pipeline will come with dependencies that depend on the pipeline design. Note that Biopipe is a framework and depending on your input and output sources, your analysis, it will greatly affect what you need to install.

MySQL

The database persistent design of the pipeline means that all information necessary to run the pipeline is stored in a database. The Biopipe was mainly designed to work with mySQL although it should work with other DBMS system but this has not been verified. MySQL[1] is an open source system that has proven to be robust enough for high throughput projects like genome annotation.

Perl Libraries

- bioperl-pipeline

- bioperl-live
- bioperl-run

Available thru anonymous cvs:

```
    cvs -d :pserver:cvs@cvs.open-bio.org:/home/repository/bioperl check-
out bioperl-live
    cvs -d :pserver:cvs@cvs.open-bio.org:/home/repository/bioperl check-
out bioperl-run
    cvs -d :pserver:cvs@cvs.open-bio.org:/home/repository/bioperl check-
out bioperl-pipeline
```

This is bleeding edge stuff so it is recommended that you use main trunk code for all three packages.

Note the schema for biopipe has moved to bioperl-pipeline/sql/schema.sql for convenience

- XML::SimpleObject

This module is required for the XML loading script to work. It works with XML::Parser which is built on XML::Parser::Expat. Some people have encountered problems installing SimpleObject. This may have something to do with LibXML which is another library that comes with XML::SimpleObject. Some error one might find:

```
      [shawnh@sashimi XML-SimpleObject0.51]$ perl Makefile.PL
      NOTE: XML::SimpleObject requires XML::Parser. If you have XML::LibXML, you
      can install XML::SimpleObject::LibXML instead.

      Checking for XML::Parser ...
      OK
      Checking if your kit is complete...
      Looks good
      NOTE: XML::SimpleObject::LibXML requires XML::LibXML. If you have XML::Parser
      you can install XML::SimpleObject instead.

      Checking for XML::LibXML ...
      WARNING from evaluation of /home/shawnh/_download/XML-SimpleObject0.51/LibXML/Mak
      Can’t locate XML/LibXML.pm in @INC (@INC contains: /home/shawnh/cvs_src/bioperl-
run/ /home/
      shawnh/cvs_src/ensembl/modules/ /home/shawnh/cvs_src/ensembl-compara/modules/ /ho
      cvs_src/bioperl-live/ /home/shawnh/cvs_src/bioperl-pipeline/ /home/shawnh/cvs_src
d
      b/ /home/shawnh/cvs_src/ensembl-pipeline/modules/ /home/shawnh/download/ /usr/lib
      .0/i386-linux /usr/lib/perl5/5.6.0 /usr/lib/perl5/site_perl/5.6.0/i386-
linux /usr/lib/perl5
      /site_perl/5.6.0 /usr/lib/perl5/site_perl .) at (eval 20) line 8.
```

The simple solution we found was to simply remove the LibXML directory before running perl Makefile.PL As there have been some significant difficulties with this, a sample installation session is as follows:

```
      [shawnh@sashimi _download]$ tar -xvf XML-SimpleObject0.51.tar
      XML-SimpleObject0.51
      XML-SimpleObject0.51/Changes
      XML-SimpleObject0.51/ex.pl
      XML-SimpleObject0.51/LibXML
      XML-SimpleObject0.51/LibXML/ex.pl
      XML-SimpleObject0.51/LibXML/LibXML.pm
      XML-SimpleObject0.51/LibXML/Makefile.old
      XML-SimpleObject0.51/LibXML/Makefile.PL
      XML-SimpleObject0.51/LibXML/test.pl
      XML-SimpleObject0.51/Makefile.old
      XML-SimpleObject0.51/Makefile.PL
      XML-SimpleObject0.51/MANIFEST
      XML-SimpleObject0.51/README
      XML-SimpleObject0.51/SimpleObject.pm
      XML-SimpleObject0.51/test.pl
      [shawnh@sashimi _download]$ cd XML-SimpleObject0.51
      [shawnh@sashimi XML-SimpleObject0.51]$ ls
      Changes  ex.pl  LibXML  Makefile.old  Makefile.PL  MANIFEST  README  Sim-
pleObject.pm  test.pl
      [shawnh@sashimi XML-SimpleObject0.51]$ rm -rf LibXML/
      [shawnh@sashimi XML-SimpleObject0.51]$ perl Makefile.PL
      NOTE: XML::SimpleObject requires XML::Parser. If you have XML::LibXML, you
      can install XML::SimpleObject::LibXML instead.

      Checking for XML::Parser ...
      OK
      Checking if your kit is complete...
      Warning: the following files are missing in your kit:
              LibXML/LibXML.pm
              Please inform the author.
              Writing Makefile for XML::SimpleObject
              [shawnh@sashimi XML-SimpleObject0.51]$ make
              mkdir blib
              mkdir blib/lib
              mkdir blib/lib/XML
              mkdir blib/arch
              mkdir blib/arch/auto
              mkdir blib/arch/auto/XML
              mkdir blib/arch/auto/XML/SimpleObject
              mkdir blib/lib/auto
              mkdir blib/lib/auto/XML
              mkdir blib/lib/auto/XML/SimpleObject
              mkdir blib/man3
              scp SimpleObject.pm blib/lib/XML/SimpleObject.pm
              cp ex.pl blib/lib/XML/ex.pl
              Manifying blib/man3/XML::SimpleObject.3pm
              u[shawnh@sashimi XML-SimpleObject0.51]$ su
              Password:
              [root@sashimi XML-SimpleObject0.51]# make install
              Installing /usr/lib/perl5/site_perl/5.6.0/XML/SimpleObject.pm
              Installing /usr/lib/perl5/site_perl/5.6.0/XML/ex.pl
              Installing /usr/share/man/man3/XML::SimpleObject.3pm
              Writing /usr/lib/perl5/site_perl/5.6.0/i386-linux/auto/XML/SimpleObject/.
              Appending installation info to /usr/lib/perl5/5.6.0/i386-
linux/perllocal.pod
```

Binaries

NCBI Blast Package which is available at this ftp site[2]

The Pipeline XML Format

This section describes the various sections of XML with Biopipe

To describe this, we will use the demo xml template, blast_file_pipeline.xml. You may find this in the bioperl-pipeline/xml/templates directory.

XML Organization

```
<pipeline_setup>
  <database_setup>
  <iohandler_setup>
  <pipeline_flow_setup>
  <job_setup>(optional)
</pipeline_setup>
```

<database_setup>

This specifies the databases that the pipeline connects to and the adaptor modules that intefaces with them.

<iohandler_setup>

This specifies the method calls that will be used by the pipeline to access the databases. These methods are contained in the modules specified by the database setup section above.

<pipeline_flow_setup>

This specifies the analysis and rules of the pipeline. Analysis refer to the runnables that will be used in this pipeline while the rules specify the order in which these analysis are to be run, including any specific pre-processing actions that are to be carried out.

<job_setup>

This is an optional part that allows specific inputs to be inserted. Usually, this is done using DataMongers and Input Create modules.

You will need to modify some parts of this XML file to point files to non-default places.

The Simple Blast Pipeline

Use Case

```
Given a file of sequences, split the files into smaller chunks, and blast
it against the database over a compute farm. Blast results files are stored
into a given results  directory, with one result file per blast job.
```

This is a simple blast pipeline demo that allows one to pipeline a bunch of blast jobs. It is stripped bare, assuming that the user has sequences in files and simply wishes to parallalize the blast jobs. It doesn't utilize one of the main features of blast which is to allow inputs from different database sources.

Configuring the Pipeline

ANALYSIS 1: DataMonger

This involves a DataMonger Analysis using the *setup_file_blast* module. The datamonger will split the input file specified into a specified number of chunks. It will create a blast job in the pipeline for each chunk. It will also create the specified working directory for storing these files and format the db file for blasting if you are blasting against itself. If you are blasting against a different database file, you can specify the formatting of the db as part of the analysis parameters.

```
270     <analysis id="1">
271       <data_monger>
272         <input>
273           <name>input_file</name>
274         </input>
275         <input_create>
276            <module>setup_file_blast</module>
277            <rank>1</rank>
278             <argument>
279                 <tag>input_file</tag>
280                 <value>/data0/shawn_tmp/blast.fa</value>
281                 <type>SCALAR</type>
282             </argument>
283             <argument>
284                 <tag>chop_nbr</tag>
285                 <value>5</value>
286                 <type>SCALAR</type>
287             </argument>
288             <argument>
289                 <tag>workdir</tag>
290                 <value>/tmp/blast_dir/</value>
291                 <type>SCALAR</type>
292             </argument>
293             <argument>
294                 <tag>result_dir</tag>
295                 <value>/tmp/blast_dir/blast_result/</value>
296                 <type>SCALAR</type>
297             </argument>
298          </input_create>
299       </data_monger>
300     </analysis>
301
```

line 276: This species the particular DataMonger to use that will prepare your file for paralization. For this case, we will use setup_file_blast which will chop up your input file specified below into smaller chunks.

line 280: This specifies the name of the input file that will be split into smaller chunks. Modify this accordingly.

line 285: This specifies the number of files to split the input file into which wil equal the number of blast jobs. You will want to chose a reasonable number that will utilize your compute farm best.

line 290: This specifies the working directory in which the blast chunks will be stored.

line: 295: This specifies where the blast result files will be stored.

The Simple Blast Pipeline

ANALYSIS 2: Blast

```
303     <analysis id="2">
304       <logic_name>Blast</logic_name>
305       <runnable>Bio::Pipeline::Runnable::Blast</runnable>
306       <db>family</db>
307       <db_file>/data0/Fugu_rubripes.pep.fa</db_file>
308       <program>blastall</program>
309       <program_file>/usr/local/bin/blastall</program_file>
310       <parameters>-p blastp -e 1-e05 -formatdb 1 -result_dir /data0/shawn_tmp/
311     </analysis>
```

This set of xml tags specify the blast analysis to run.

Line 305: This specifies the pipeline to use the *Bio::Pipeline::Runnable::Blast* runnable.

Line 306: This is the name of the database to blast against

Line 307: This is the path to the database file to blast against.

Line 308: This is the name of the blast program.

Line 309: This is the location of the blast program

Line 310: These are the blast parameters as well as parameters to the Blast runnable.

- *-p blastp* This is a blastall parameter to specify using the blastp alignment program.
- *-e 1-e05* Another blastall parameter to return hits with scores < 0.00001
- *-formatdb 1* A Bio::Pipeline::Runnable::Blast parameter that tells it to format the db file specified in line 307 before commencing blasting.
- *-result_dir /data0/shawn_tmp/blast_result/* This tells the Runnable where to store the blast output.

Loading the Pipeline

The xml is loaded to create the pipeline using the Xml2DB.pl script. The script is located in the bioperl-pipeline package under *bioperl-pipeline/xml/*. Other xml templates are found in *bioperl-pipeline/xml/templates* This script assumes that you have a mysql database installed.

Run the script with a -h to see the options available:

```
shawnh@pulse1 ~/src/bioperl-pipeline/xml> perl Xml2Db.pl -h
******************************
```

```
*Xml2DB.pl
******************************
This script configures and creates a pipeline based on xml definitions.

Usage: Xml2DB.pl -dbhost host -dbname pipeline_name -dbuser user -dbpass pass-
word -schema /path/to/biopipeline-schema/ -p pipeline_setup.xml

Default values in ()
-dbhost host (mysql)
-dbname name of pipeline database (test_XML)
-dbuser user name (root)
-dbpass db password()
-schema The path to the bioperl-pipeline schema.
        Needed if you want to create a new db.
        (../sql/schema.sql)
-verbose For debugging
-p      the pipeline setup xml file (required)
```

An example session:

```
shawnh@pulse1 ~/src/bioperl-pipeline/xml> perl Xml2Db.pl -dbname blast_pipe -
dbuser root -p templates/blast_file_pipeline.xml
A database called test_xml already exists.
Continuing would involve dropping this database and loading a fresh one us-
ing templates/blast_file_pipeline.xml.
Would you like to continue? y/n [n] y
Dropping Databases
Creating test_xml
   Loading Schema...
Reading Data_setup xml   : templates/blast_file_pipeline.xml
Doing DBAdaptor and IOHandler setup
Doing Pipeline Flow Setup
Doing Converters..
Doing Analysis..
Doing Rules
Doing Job Setup...
Loading of pipeline test_xml completed
shawnh@pulse1 ~/src/bioperl-pipeline/xml>
```

At this point, you have a pipeline database called blast_pipe that is ready for running.

Configuring the PipeConf

Hangon. You may want to configure any pipeline management parameters before running. This is done via the PipeConf.pm module located at *bioperl-pipeline/Bio/Pipeline/PipeConf.pm*. Various parameters may be set here:

```
%PipeConf = (

    # You will need to modify these variables

    # working directory for err/outfiles
    NFSTMP_DIR => '/tmp/',

    # database specific variables

    DBI_DRIVER => ’mysql’,
    DBHOST     => ’mysql’,
    DBNAME     => ’test_xml’,
    DBUSER     => ’root’,
    DBPASS     => ”,

    # Batch Management system module
    # Currently supports PBS and LSF
    # ignored if run in local mode
    BATCH_MOD   =>  ’LSF’,

    # farm queue
    QUEUE      => ’normal3’,

    # no of jobs to send to Batch Management system at one go
    BATCHSIZE  => 3,

    #bsub opt
    BATCH_PARAM => ’-C0’,

    # no of jobs to fetch at a time and submit
    MAX_INCOMPLETE_JOBS_BATCHSIZE => 1000,

    # no of completed jobs to fetch at a time and create next jobs
    MAX_CREATE_NEXT_JOBS_BATCHSIZE => 5,


    # number of times to retry a failed job
    RETRY       => ’5’,

    # path to runner.pl, use by the BatchSubmission objects
    # to look for runner.pl. If not supplied it looks in the default
    # directory where PipelineManager lies
    RUNNER     => ”,

    #sleep time (seconds) in PipelineManager before waking up and looking for jobs to run
    SLEEP      => 30,
    #number of jobs to fetch at a time
    FETCH_JOB_SIZE => 100,
```

Running your pipeline

To run the pipeline, cd to the *bioperl-pipeline/Bio/Pipeline/* directory.

Run the PipelineManger.pl with the -h option to check the options available:

```
shawnh@pulse1 ~/src/bioperl-pipeline/Bio/Pipeline> perl PipelineManager.pl -
h
************************************
*PipelineManager.pl
************************************
This is the central script used to run the pipeline.
```

```
Usage: PipelineManager.pl

Options:
Default values are read from PipeConf.pm

     -dbhost The database host name (localhost)
     -dbname The pipeline database name
     -dbpass The password to mysql database
     -flush  flush all locks on pipeline and remove any that exists.
             Should only be used for debugging or development.
     -batchsize The number ofjobs to be batched to one node
     -local     Whether to run jobs in local mode
                (on the node where this script is run)
     -number    Number of jobs to run (for testing)
     -queue     Specify the queue on which to submit jobs
     -verbose   Whether to show warning during test and setup
     -help      Display this help
```

If you run the script in local(-l) mode, the script will not be batched to LSF or PBS. Jobs are executed sequentially in this way This is usually a recommended way of testing your pipeline before batching all the jobs. You may also specify the number of jobs to run with the -n option.

Once you are ready to run on the node, you may reset your jobs by rerunning the Xml2DB script as described in the last section. Submit the jobs to the compute nodes now by running the PipelineManger without the -l option:

```
shawnh@pulse1 ~/src/bioperl-pipeline/Bio/Pipeline> perl PipelineManager.pl -
dbname test_xml -f
///////////////Starting Pipeline//////////////////////
Fetching Analysis From Pipeline test_xml
2 analysis found.
Running test and setup..

//////////// Analysis Test ////////////
Checking Analysis 1 DataMonger ok
Checking Analysis 2 Blast ok

///////////////Tests Completed////////////////////////

Fetching Jobs...
Fetched 1 incomplete jobs
Fetched 0 completed jobs
opening bsub command line:
 bsub -o /usr/users/shawnh/tmp/test_xml_DataMonger.1038684370.167.out -
e /usr/users/shawnh/tmp/test_xml_DataMonger.1038684370.167.err -q nor-
mal3 -C0  /usr/users/shawnh//src/bioperl-pipeline//Bio/Pipeline/runner.pl -
dbname test_xml -host mysql -port 3306 -dbuser root 1
Going to snooze for 3 seconds...
Waking up and run again!
Fetching Jobs...
Fetched 0 incomplete jobs
Fetched 0 completed jobs
Going to snooze for 3 seconds...
Waking up and run again!
Fetching Jobs...
Fetched 0 incomplete jobs
Fetched 0 completed jobs
Going to snooze for 3 seconds...
Waking up and run again!
```

At any one time, you may check in the job table for the status of your jobs. The following shows an example mysql session

```
  shawnh@pulse1 ~/src/bioperl-pipeline/Bio/Pipeline> mysql -u root test_xml
  Reading table information for completion of table and column names
  You can turn off this feature to get a quicker startup with -A

  Welcome to the MySQL monitor.  Commands end with ; or \g.
  Your MySQL connection id is 3940238 to server version: 3.23.36

  Type 'help;' or '\h' for help. Type '\c' to clear the buffer

  mysql> select * From job;
  +--------+-----------+------------+----------+------------------
------------------------------------+------------------------------
-------------------------+----------------------------------------
------------+----------+---------+--------------------+-----------
--+
  | job_id | process_id | analysis_id | queue_id | stdout_file
ject_file                                         | status    | stage   | time
  +--------+-----------+------------+----------+------------------
------------------------------------+------------------------------
-------------------------+----------------------------------------
------------+----------+---------+--------------------+-----------
--+
  |      2 | NEW       |          2 |   437257 | /usr/users/shawnh/tmp/test_xml_Blast
MITTED | RUNNING | 2002-12-01 11:59:21 |          0 |
  +--------+-----------+------------+----------+------------------
------------------------------------+------------------------------
-------------------------+----------------------------------------
------------+----------+---------+--------------------+-----------
--+
  1 row in set (0.00 sec)

  mysql>
```

Here we can see that one blast job has been created and is currently having status *SUBMITTED* and stage *RUNNING*. The status available are NEW|SUBMITTED|FAILED|COMPLETED and for stage are BATCHED|READING|RUNNING|WRITING. Once a job is completed, the jobs will be move to the completed_jobs table. If the job fails, you may view the error log specified by stderr_file. Currently stderr_file and stdout_files are removed only if a job is completed.

Viewing Your Results

You may now check the *resultdir* specified in the Blast Analysis to see if your blast results are stored properly.

Congratulations at this point you have managed to setup the biopipe successfully. It is hoped that you have a feel of how the biopipe works. You may now try out more complex examples of through the other XML templates that we have. More documentation for this will come in the future.

Notes

1. http://www.mysql.com
2. ftp://ftp.ncbi.nih.gov/blast/executables/