A Tiny Queuing System for Blast Servers

Colas Schretter*and Laurent Gatto†

16th November 2005

Introduction

With the aim to assists researchers for a rapid and automated oligonucleotide design, we developed an online multi-users bioinformatic application. People could submit several design jobs and order results through a web portal. Therefore, several submission could occur independently in the same time.

Some design jobs need to launch a similarity search across a nucleotide Blast database. This IO-bound process requires to limit the number of such queries executed in parallel on the Blast server. Therefore we setup a queuing system to ensure that, at any time, the executed jobs do not require more than the available IO resources.

Existing queuing systems like PBS or OpenPBS (`http://pbs.mrj.com`), Maui (`http://www.clusterresources.com`) or Sun Grid Engine (`http://gridengine.sunsource.net`) could be installed to solve these requirements.

However, we developed a set of shell scripts that work together to add lightweight queuing abilities to any UNIX systems. The features of our solution are:

- it only rely on shell scripts and the crontab service
- it uses e-mail notification
- it can be set up with users privileges only
- it supports distant execution of jobs through SSH
- it is easily to install, use and configure

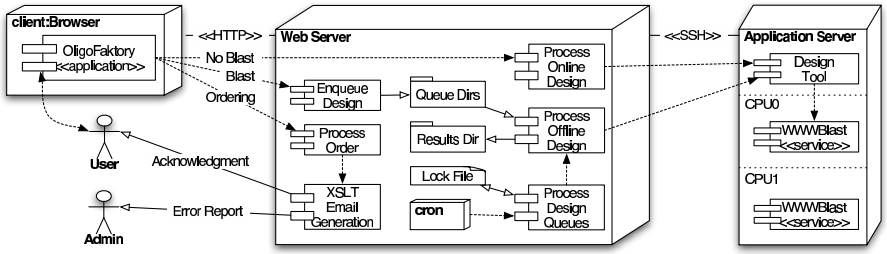
The set of scripts of a real-case application are given and commented. This document should help anyone to adapt those scripts and extend the system for further requirements.

*cschrett@ulb.ac.be
†lgatto@ulb.ac.be

Framework

In the following sections, we describe each functionality by providing the scripts end their dependencies. First, we describe how any user could submit a job to the system. Then, we explain how we implemented the queuing mechanism. We setup an active system loop that watches for the presence of new job requests and a locking mechanism to limit resources usages.

We have chosen to distribute the computation charge among several independent and dedicated servers. In our application, a web server is the entry-point to job submissions, it manages the queues and results directories. In order to ensures interactive performances to any visitor of our web server, we wanted to reserve the resources of this server exclusively for dynamic web page generation. It will also be in charge of the mailing capabilities. Resource-consuming tasks are not allowed to run on the web server. Therefore, executables are launched on a third party application server, parameters and results are stored on the web server.



Job Submission

To submit a job, one just needs to create a file in a conventional queue directory. For our application, the file is the XML input for a design application, or an order to process. However, we could consider any executable files, like complex shell scripts [1].

In our case, the directory structure for queues look like:

```
/var/lib/wwwrun/queues/
   ./order/
   ./longoligo/
   ./optiamp/
   ./sirna/
   ./analysis/
```

When an application is running behind the queuing system, a lock file is created as a flag. This file does not exists while the system is idle. Results of designs applications are copied in:

```
/var/lib/wwwrun/results/
```

Moreover, results or error reports will be sent by mail to users.

[1] In this case, the queuing system should take the identity of the file's owner before executing it.

Job Watcher

A crontab daemon periodically checks if files are stored in the queue directories.

The `crontab` file, **executes** the `process_design_queues.sh` and `process_order_queue.sh` scripts at given times or time intervals.

```
MAILTO=""
0 6, 12, 17 * * * /var/lib/wwwrun/process_order_queue.sh
* * * * * /var/lib/wwwrun/process_design_queues.sh \
optiamp longoligo analysis sirna
```

In this example, ordering jobs are performed at 6:00, 12:00 and 17:00 each day. Design queries will be detected in less than a minute.

Offline Design Job Launcher

The core of the system relies in the scripts launched by the cron daemon. These scripts will check if the resources are available, for instance by looking for the presence of a lock file. Several jobs could be executed in parallel, if we allow more than one lock file. However, for a computation intensive BLAST query, we chose to limite one job at a time.

The `process_design_queues.sh` script will

1. check if ressources are available,
2. lock the system and
3. for each xml file (*.i.e* each job waiting to be run), **call** `process_offline_design.sh` with the appropriate parameters.

Listing 1: process_design_queues.sh

```
#!/bin/sh

echo "This script process all xml files stored in the tool    "
echo "queue directories. The result is moved to a conventional"
echo "results directory. A notification e-mail with a direct  "
echo "access url is sent to user or an error e-mail is sent to"
echo "the administrator and the user.                          "
echo

if [ -z "$1" ]
then
    echo "This script needs at last a tool name in parameter"
    exit -1
fi

# exit if the resources are not available (a design is already running)
if [ -e queues/locked ]
then
    echo "Resources are not available (a design is already running)"
    exit 0
```

```
fi

cd /var/lib/wwwrun

# lock de design queue
touch queues/locked

# parse argument list
while [ $# -ge 1 ]
do
    tool=$1 #extract the tool (directory) to use for design
    echo "File list to process:"
    echo queues/$tool/*.xml
    if test -f queues/$tool/*.xml
    then
        # process each files of the todo list
        for X in queues/$tool/*.xml
        do
            echo "Call process_offline_design.sh $X $tool :"
            ./process_offline_design.sh $X $tool
        done
    else
        echo "No xml file to process"
    fi
shift
done

# job is done; resources are released
rm queues/locked
```

The `process_offline_design.sh` script

1. sets variables (like file names, email addresses...) and
2. launches the computation over `ssh` on the application server
3. notifies the user that the job is done or the user and the administrator that an error occured.

Listing 2: process_offline_design.sh

```
#!/bin/sh

echo "This script process a given xml file with a given java "
echo "tool. The result is moved to a conventional results    "
echo "directory. A notification e-mail with a direct access  "
echo "url is sent to user or an error e-mail is sent to the  "
echo "administrator and the user.                            "
echo

if [ -z "$2" ]
```

```
then
    echo "This script needs a file name and a tool name in parameters"
    exit -1
fi

# set usefull variables:
filename=$1      # full path file name
tool=$2          # extract the tool name to use for design
file_id=`basename $filename .xml`
designer_email=`cat $filename | grep "email" | awk -F '"' '{print $4}'`
       #  Use " as input field separator to be replaced
administrator_email=administrator@yourdomain.org

# avoid concurrent access collisions
mv $filename results
# we will work with the file stored in the results directory
filename=results/`basename $filename`
# launch the appropriate design tool:
ssh wwwrun@appserver "java -jar $tool.jar" \
       <$filename 2>log/$file_id.log >log/$file_id.out

if [ "$?" != 0 ] # test if no error occurred
then
    echo "Error: failed to run $tool on $filename!"
    # we keep the log in case of errors
    cat log/$file_id.log
    echo "sending a notification e-mail to \
    <$designer_email> and <$administrator_email>"
    echo "Dear OligoFaktory User!
          This e-mail has been generated.
          An error occurred at the execution of $tool on your design
          query of id '$file\_id'! In most of the case, this mean that
          your input data contains integrity errors. The following
          error message could help developers to understand the exact
          cause:
          `cat log/$file_id.log`" \
          | mail -s "OligoFaktory Design ERROR (id=$file_id)" \
          -r $administrator_email $administrator_email
else
    rm log/$file_id.log                 # we erase empty log file
    mv log/$file_id.out $filename       # we move output file
    echo "Design of $tool on $filename done!"
    echo "sending a notification e-mail to <$designer_email>"
    # send a notification e-mail to the customer:
    echo "
      Dear OligoFaktory User!

      This e-mail has been generated.
      Your results sheet corresponding to $tool design of id '$file_id'
      is available on the OligoFaktory web site. You can access directly
      to the results with the following URL:
      http://ueg.ulb.ac.be/oligofaktory/results.jsp?id=$file_id
```

```
    Have a nice day.
  " \
  | mail -s "OligoFaktory Design (id=$file_id)" -a $filename \
  -r \$administrator\_email \$designer\_email
fi
```

Online Design Job Launcher

We provide the opportunity to directly run a design, independently of the queuing management. This is particularly useful when the web server wants to process a query file during an interactive user's session. Therefore, a waiting splash screen will be drawn during the design and the user will not receive any notification in his mailbox.

The `process_online_design.sh` script performis similar tasks than `process_offline_design.sh` except that all output is displayed on the users screen.

Listing 3: process_online_design.sh

```
#!/bin/sh

echo "This script processes a given xml file with a given java tool."
echo "The result is moved to a conventional results directory."
echo

 if [ -z "$2" ]
then
    echo "this script needs a file name and a tool name in parameters"
    exit -1
fi

# set useful variables:
filename=$1     # full path file name
tool=$2         # extract the tool name to use for design
file_id=`basename $filename .xml`

# avoid concurrent access collisions ...
mv $filename results
# ... and use the file stored in the results directory
filename=results/`basename $filename`

# launch the appropriate design tool:
ssh wwwrun@appserver "java -jar $tool.jar" \
        <$filename 2>log/$file_id.log >log/$file_id.out

if [ "$?" != 0 ] # test if no error occurred
then
    # we keep the log and output files in case of errors
    echo "Error: failed to run $tool on $filename!"
    cat log/$file_id.log >&2
    exit -1
else
```

```
    cat log/$file_id.log >&2
    rm log/$file_id.log             # we erase empty log file
    mv log/$file_id.out $filename   # we move output file
    echo "Design of $tool on $filename done!"
    exit 0
fi
```

Order Job Launcher

The ordering service yields the development of a supplementary script. Ideally, the third party facility should GET the orders from queue and process the available files, when they want and like they want. However, for technical and human reasons, we developed a PUT procedure. Periodically, we process orders, and sent two formated e-mails:

1. A formated e-mail is sent to the ordering facility to launch effective synthesis.
2. A human readable confirmation e-mail is sent to the customer.

The `process_order_queue.sh` program (**called by** `crontab`) **runs** `results_to_customer.xsl` and `results_to_isogen.xsl` that make use of two XSL style-sheet to generate the content of these two e-mails.

Listing 4: process_order_queue.sh

```
#!/bin/sh

echo "This script process all xml files stored queues/order/"
echo "The results are put in queues/order/processed/"
echo "A notification e-mail of the ordering list is sent to user."
echo "A formated order e-mail is sent to the ordering facility."
echo ""

echo "file list to process:"
echo queues/order/*.xml
cd /var/lib/wwwrun

if test -f queues/order/*.xml
then
    # process each files of the order list:
    for X in queues/order/*.xml
    do
        # move the file to avoid concurrent access collisions:
        mv $X queues/order/processed/
        # we will work with the file stored in the processed directory:
        X=queues/order/processed/`basename $X`
        # set useful variables:
        file_id=`basename $X .xml`
        customer_email=`cat $X | grep "email" | awk -F '"' '{print $4}'`
                #  Use " as input field separator to be replaced
        order_email=administrator@yourdomain.org
```

```
        # send an e-mail to the automated synthesis queue:
        echo "sending a formated e-mail to <$order_email>"
        xsltproc results_to_order.xsl $X \
                | mail -s "Order from OligoFaktory ('$X')" \
                -r $customer_email $order_email
        # send a confirmation e-mail to the customer:
        echo "sending a notification e-mail to <$customer_email>"
        xsltproc results_to_customer.xsl $X \
                | mail -s "Oligo Order Confirmation (id=$file_id)" \
                -r $order_email $customer_email
    done
else
    echo "no xml file to process"
fi
```

The `results_to_production.xsl` script will notify the ordering facility to launch effective synthesis.

Listing 5: results_to_production.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version=
    "1.0">
  <xsl:output method="text" />
  <xsl:template match="/oligofaktory">
\&lt;DESCRIPTION\&gt;
\&lt;OLIGOORDER\&gt;
\&lt;KDNR\&gt; <xsl:value-of select="entry[@name='customerNumber']/@value
    "/>
\&lt;UNI\&gt; <xsl:value-of select="entry[@name='organisation']/@value"/>
\&lt;INSTITUT\&gt;
\&lt;AK\&gt;
\&lt;STREET\&gt; <xsl:value-of select="entry[@name='street']/@value"/>
\&lt;PLZ\&gt; <xsl:value-of select="entry[@name='zip']/@value"/>
\&lt;CITY\&gt; <xsl:value-of select="entry[@name='city']/@value"/>
\&lt;COUNTRY\&gt; <xsl:value-of select="entry[@name='country']/@value"/>
\&lt;NAME\&gt; <xsl:value-of select="entry[@name='name']/@value"/>
\&lt;PHONE\&gt; <xsl:value-of select="entry[@name='telephone']/@value"/>
\&lt;FAX\&gt;\&lt;EMAIL\&gt; <xsl:value-of select="entry[@name='email']/
    @value"/>
\&lt;ORDERNO\&gt; <xsl:value-of select="entry[@name='purchaseNumber']/
    @value"/>
\&lt;DATE\&gt; <xsl:value-of select="entry[@name='date']/@value"/>
\&lt;DELIVERY\&gt; Standard
\&lt;COMMENT\&gt;
<xsl:value-of select="entry[@name='billAdress']/@value"/>
Comment: <xsl:value-of select="entry[@name='comments']/@value"/>
    <!-{}- parse leaf nodes of the sequence hierarchy -{}->
    <xsl:apply-templates select="//oligo" />
\&lt;OLIGOORDEREND\&gt;
```

```
  </xsl:template>
  <xsl:template match="oligo">
\&lt;OLIGO\&gt; <xsl:value-of select="@name"/>
\&lt;SEQUENCE\&gt;<xsl:value-of select="dna"/>
\&lt;SCALE\&gt; <xsl:value-of select="//entry[@name='scale']/@value"/>
\&lt;DOC\&gt; Standard
\&lt;CLEAN\&gt; <xsl:value-of select="//entry[@name='purification']/
    @value"/>
\&lt;MODIFICATION\&gt;
5'= <xsl:value-of select="//entry[@name='tailModification5']/@value"/>
\&lt;MODIFICATION\&gt;
3'= <xsl:value-of select="//entry[@name='tailModification3']/@value"/>
\&lt;MODIFICATION\&gt;
INT= <xsl:value-of select="//entry[@name='internalModification']/@value"/
    >
\&lt;MODIFICATION\&gt;
<xsl:if test="//entry[@name='phosphorothioate']/@value = 'true'">PTO</
    xsl:if>
\&lt;COMMENT\&gt;
\&lt;OLIGOEND\&gt;
  </xsl:template>
</xsl:stylesheet>
```

The `results_to_customer.xsl` script will notify the custommer that the order has effectively been send.

Listing 6: results_to_customer.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version=
    "1.0">
  <xsl:output method="text" />
  <xsl:template match="/oligofaktory">
Dear Customer,Thank you for your order.
This order confirmation was generated by the oligo ordering software
to enable verification and storage of your data.

You will receive another e-mail to confirm that your order was received
by our oligo ordering administration and we will process it a.s.a.p!

We received the following order from you:

Scale : <xsl:value-of select="entry[@name='scale']/@value"/>
Clean : <xsl:value-of select="entry[@name='purification']/@value"/>
Modification (5') :
<xsl:value-of select="entry[@name='tailModification5']/@value"/>
Modification (3') :
<xsl:value-of select="entry[@name='tailModification3']/@value"/>
Modification (Internal) :
<xsl:value-of select="entry[@name='internalModification']/@value"/>
Phosphorothioate :
```

```
<xsl:value-of select="entry[@name='phosphorothioate']/@value"/>
    <!-{}- parse leaf nodes of the sequence hierarchy -{}->
    <xsl:apply-templates select="//oligo" />

Comment:
    <xsl:value-of select="entry[@name='comments']/@value"/>
  </xsl:template>
  <xsl:template match="oligo">
Oligo Name : <xsl:value-of select="@name"/>
Sequence : <xsl:value-of select="dna"/>
  </xsl:template>
</xsl:stylesheet>
```

Conclusions

We described a lightweight solution to the implementation of a queuing system on UNIX systems. The described framework has been successfully implemented as a primer design portal (`http://ueg.ulb.ac.be/oligofaktory`) able to launch and manage several design applications.