A Tiny Queuing System for Blast Servers

Colas Schretter*and Laurent Gatto†

December 9, 2005

Introduction

When multiple Blast [4] similarity searches are run simultaneously against large databases and no parallel computing facility is at hand, the IO-bound process requires to limit the number of concurrent runs. 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 OpenPBS [1], Maui [2] or Sun Grid Engine [3] 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-like systems. The features of our solution are:

- it only relies on shell scripts and the crontab service
- it is easy to install, use and extend
- it provides e-mail notifications

The set of scripts of a real-case application are given and commented. The example presented has been limited (*i.e.* in the number of Blast parameters) for didactical purposes. This document should help anyone to adapt those scripts and extend the system for further requirements.

Framework

In the following sections, we describe each functionality by providing the scripts and their dependencies. First, we describe how users submit jobs 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. A general deployment diagram describing the queuing system is given in figure 1. The requirements for the queuing system are a UNIX-like operating system, a running webserver and the necessary Blast programs and databases.

*cschrett@ulb.ac.be
†lgatto@ulb.ac.be

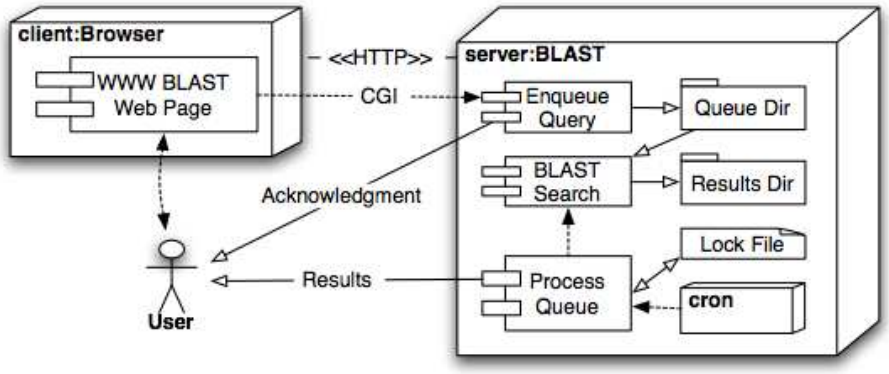

Figure 1: Deployment diagram of the queuing system for BLAST searches.

Job Submission

The queueing system manages two sets of files:

1. BLAST searches ready to be run as soon as resources are freed
2. result files of recently finished searches

The search results are saved in a web accessible directory (*i.e.* `/var/www/localhost/htdocs/`). Each file is uniquely identified by its exact generation time and can be consulted by the user. The *ready-to-be-run* searches (consisting of a `blastall` command and the input queries) are stored in the queue directory. The queuing blast file are identified by their exact generation time and the users contact email adress. Note that the `blast/queue` and `blast/results` directories have to be respectively writable and readable by the apache user.

We use a modified WWWBLAST web page as entry to the queuing system. The only modification consists in the addition of an HTML text field for the users email adress. When posted, the form invokes a shell cgi script `queue_blast_jobs.sh` located in a directory containing server scripts, typically `/var/www/localhost/cgi-bin/`. This script will parse the user's blast parameters and add the appropriate file to the blast jobs queue. To avoid a possible delay if resources are free, the `process_blast_jobs.sh` script (see below) is immediately launched (independently of cron).

Listing 1: queue_blast_jobs.sh

```
#! /bin/sh

# Authors: Colas Schretter and Laurent Gatto {cschrett,lgatto}@ulb.ac.be, 2004-2005

# change to the results and queue directories
cd /var/www/localhost/htdocs/blast

# unique temporary file identified by the ID of the running process
TMPFILE=/tmp/queue_blast_post.$$

# saving the cgi post
cat > $TMPFILE

# parsing the cgi post file to extract blast parameters
```

```
DATALIB=$( cat $TMPFILE | grep -Z -A 2 "DATALIB" | tail -n 1 | tr -d '\r' )
PROGRAM=$( cat $TMPFILE | grep -Z -A 2 "PROGRAM" | tail -n 1 | tr -d '\r' )
EMAIL=$( cat $TMPFILE | grep -Z -A 2 "EMAIL" | tail -n 1 | sed s/@/[at]/ | tr -d '\r'
    )
DATE=$( date +%Y%m%d%H%M%N )

# creating a new file in the queue
# the size of the query input is limited to 1000 lines
echo "echo \"$( cat $TMPFILE | grep -Z -A 1000 "SEQUENCE" | sed -e 1,2d -e /---/,'$'d
    )\" | blastall -p $PROGRAM -d $DATALIB -o ../results/$DATE.html -T T" > queue/
    $DATE.$EMAIL.blast

# remove cgi post
rm $TMPFILE

# output an HTML page
echo "Content-type: text/html"
echo
echo "<h2>Your BLAST job has been enqueued</h2>"
echo "<p>The status of your request will be notified by email when resources are
    available</p>"
echo "<p>You results will be availables at http://yourblastserver.org/blast/results/
    $DATE.html as soon as possible.</p>"

# try to run the BLAST immediately
./process_blast_jobs.sh
```

Job Watcher and Launcher

A crontab daemon periodically runs the `process_blast_jobs.sh` script that checks if files are stored in the queue directory and eventually runs the BLAST searches.

```
MAILTO=""
* * * * * /var/www/localhost/htdocs/blast/process_blast_jobs.sh
```

In this example, BLAST queries will be detected in less than a minute. The script first checks if the necessary ressources are available. When an application is running behind the queuing system, a lock file is created as a flag and the script exits. If no lock file is present, `process_blast_jobs.sh` first locks the system. For all the jobs in the queues, it then chooses the oldest job file, recovers the users email adress and the future result file url and launches the similarity search. When a BLAST search finishes, the script mails the user the result url. When all the jobs in the queue are done, the system is unlocked to allow the next searches to be performed.

Listing 2: process_blast_jobs.sh

```
#!/bin/sh

# Authors: Laurent Gatto and Colas Schretter {lgatto,cschrett}@ulb.ac.be, 2004-2005

# change to the working directory
cd /var/www/localhost/htdocs/blast/queue/

if [ -e ./locked ]
then
    # Resources are not available
    exit 0
```

```
fi

# lock de system
touch ./locked

# process all enqueued files reverse sorted by timestamp
for JOB in $( ls -1tr *blast )
do
        # get the results' url and the user's email
        EMAIL=$( basename $JOB .blast | sed -e s/^[0-9]*\.// -e s/\\[at\\]/@/ )
        HTML=${JOB%%.*}.html

        sh $JOB
        rm $JOB

        # mail user that the results are ready
        echo "Dear user,
your BLAST results are available at
http://yourblastserver.org/blast/results/$HTML

Enjoy" | mail -s "BLAST results" $EMAIL

done

# job(s) is/are done; resources are freed
rm ./locked
```

Possible extensions

It is further possible to distribute the computation charge among several independent and dedicated servers. A possibility would be to have a web server as the entry-point for job submissions, that manages the queue and result directories. In order to ensures interactive performances to any visitor, the resources of this server are exclusively reserved for dynamic web page generation and mailing capabilities. Resource-consuming tasks are not allowed to run on the web server. Therefore, executables are launched on a third party application server. The shell cgi scripts `queue_blast_jobs.sh` and `process_blast_jobs.sh` would, in such case, still be located on the webserver, but the BLAST search would be run over `ssh` on the application server and the results would be copied back to the webserver[1]:

```
ssh wwwrun@application_server "$JOB"
scp -q wwwrun@application_server:~/results/$OUTPUT  $QUEUEDIRS/results/.
```

It is further possible to define two or more simultaneous jobs by allowing two or more `locked` files to be created. Different `locked` files may be associated with different application hosts. `process_blast_jobs.sh` would check for the existence of `locked_1` to `locked_n` before starting a new search.

References

[1] `http://www.openpbs.org`.

[1] ssh and scp need to be automated with ssh keys to avoid password requests

[2] http://www.clusterresources.com.

[3] http://gridengine.sunsource.net.

[4] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *J Mol Biol*, 215(3):403–410, October 1990.