MEETING THE COMPUTATIONAL DEMANDS FOR DATA DRIVEN BIOLOGY AT VECTORBASE

A Thesis

Submitted to the Graduate School
of the University of Notre Dame
in Partial Fulfillment of the Requirements
for the Degree of

Master of Science

by

Nathan A. Konopinski

_________________________
Frank Collins, Director

Graduate Program in Biological Sciences
Notre Dame, Indiana
December 2011
TABLE OF CONTENTS

Index of Figures .................................................................................................................. iv

Chapter 1: VectorBase Computer Hardware and Systems Migration ....................... 1
   1.1 Overview ..................................................................................................................... 1
   1.2 Setting Up Network Services ............................................................................... 6
   1.3 SSL Certificates ..................................................................................................... 7
   1.4 FTP .......................................................................................................................... 10
   1.5 E-Mail ...................................................................................................................... 11
   1.6 Secondary Web ....................................................................................................... 14
   1.7 Production Web ....................................................................................................... 17
       1.7.1 Separation of VectorBase and Ensembl Code .............................................. 22
   1.8 Production Database ............................................................................................ 23
   1.9 Production Application ......................................................................................... 24
   1.10 Development and Pre-Production Machines .................................................... 25
   1.11 Network Services ................................................................................................. 26
   1.12 Conclusion ............................................................................................................ 31

Chapter 2: Distributed Computing at VectorBase ...................................................... 33
   2.1 Overview .................................................................................................................. 33
   2.2 Modifications .......................................................................................................... 34
       2.2.1 First Modification ........................................................................................... 35
       2.2.2 Second Modification ....................................................................................... 36
       2.2.3 Third Modification .......................................................................................... 38
       2.2.4 Fourth Modification ....................................................................................... 39

Chapter 3: BLAST at VectorBase .................................................................................. 40
   3.1 Overview .................................................................................................................. 40
   3.2 Original User Interface ......................................................................................... 46
   3.3 New Technologies for New Interface ..................................................................... 48
   3.4 Error Checking ....................................................................................................... 50
   3.5 Locally Run Jobs ..................................................................................................... 51
   3.6 BLAST Workflow ................................................................................................... 53
   3.7 Updated User Interface ......................................................................................... 55
   3.8 Future Work ............................................................................................................ 56

Appendix A: Procedures for adding new machines to VectorBase ......................... 59
Appendix B: Jobs Web Service Code ................................................................. 61
B.1 xgrid.wsdI........................................................................................................61
B.2 server.php........................................................................................................64
Appendix C: BLAST GUI Web Code.................................................................65
   C.1 input.php ..................................................................................................66
   C.2 submit.php..............................................................................................106
   C.3 status.php...............................................................................................111
   C.4 results.php..............................................................................................123

Bibliography......................................................................................................134
INDEX OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Pre-Migration Hardware Configuration</td>
<td>5</td>
</tr>
<tr>
<td>1.2</td>
<td>Post-Migration Hardware Configuration</td>
<td>6</td>
</tr>
<tr>
<td>1.3</td>
<td>Email Configuration: General Tab</td>
<td>12</td>
</tr>
<tr>
<td>1.4</td>
<td>VPN Configuration: L2TP</td>
<td>28</td>
</tr>
<tr>
<td>1.5</td>
<td>VPN Configuration: PPTP</td>
<td>29</td>
</tr>
<tr>
<td>1.6</td>
<td>VPN Configuration: Client Information</td>
<td>30</td>
</tr>
<tr>
<td>3.1</td>
<td>VectorBase Job Submissions by Month</td>
<td>41</td>
</tr>
<tr>
<td>3.2</td>
<td>Country of Origin for Jobs Submitted To VectorBase</td>
<td>42</td>
</tr>
<tr>
<td>3.3</td>
<td>Top Submitting Metropolitan Areas In the US With Associated Universities in Those Areas</td>
<td>43</td>
</tr>
<tr>
<td>3.4</td>
<td>Original BLAST User Interface</td>
<td>46</td>
</tr>
<tr>
<td>3.5</td>
<td>Original User Interface Expanded</td>
<td>47</td>
</tr>
<tr>
<td>3.6</td>
<td>An error scenario in the updated UI</td>
<td>51</td>
</tr>
<tr>
<td>3.7</td>
<td>Workflow of BLAST Job Submissions</td>
<td>53</td>
</tr>
<tr>
<td>3.8</td>
<td>Updated BLAST User Interface</td>
<td>55</td>
</tr>
</tbody>
</table>
CHAPTER 1

VECTORBASE COMPUTER HARDWARE AND SYSTEMS MIGRATION

1.1 Overview

Every computing project faces hardware decisions at one point or another. Whether to maintain older equipment or replace it with the latest and greatest is a common question. In the long run, it could cost more to maintain older equipment. It is possible that the machines become plagued by not only the cost of replacement parts but also downtime while new parts are being delivered and installed. This could affect developer productivity or even site wide downtime in the worst case. Older equipment generally has higher power requirements, costing more in energy bills just to have them plugged in and turned on. On top of consuming more electricity, older machine are also outperformed by newer models. This means it will take longer to get results from the old computers.

VectorBase has made several hardware purchases over the years. Since 2007 there have been two minor upgrades and two major upgrades. The major upgrade was made to replace many machines that had either completely failed or had components that were failing. The following text contains brief
descriptions of the minor upgrades and a detailed description of the first major upgrade. The major upgrade involved a site wide migration to a completely new set of computers.

As of July 2007, VectorBase had obtained three new Xserves to replace their line of production machines. These Xserves were the first with Intel processors to arrive at VectorBase. All previous machines on the project contained the IBM G5 PowerPC chip. The G5 machines produced a lot of heat. This fact is the most likely culprit contributing to the machines failing after only a few years of service. Switching CPU architectures meant that binary files compiled on the G5 machines would not be interchangeable with those on the Intel machines and vice-versa. This in itself was not much of an issue for migration at VectorBase as all the software the site runs is capable of being compiled on either of these architectures using the same configuration options. The source code could be copied over to the new machines and re-compiled.

The operating system all the Xserves ran at the time, Apple's OS X 10.4 Server, was capable of straddling both architectures. This aided in incorporating the new hardware design since the configuration of core networking services such as DHCP, DNS, and LDAP were identical. It also meant there was no new material to learn for administrating the machines and their configuration files were stored in identical locations. Copying source code to the new machines and executing the configure and make commands were the only steps necessary to get the new machines up and running, aside from setting up the core networking
This process was done for MySQL, PostgreSQL, and Apache’s httpd and Tomcat software packages. Once the software was installed, the existing site content data was simply copied over from the previous production machines or the subversion repository. The names given to the new machines were MrLee, the database machine; Tanya, the web server; and Patty, the application server.

VectorBase purchased another Xserve during the following winter. This machine, named Gene, was to be a substitute for running VectorBase jobs (BLAST, ClustalW, HMMER, Exonerate) on Notre Dame’s Center for Research Computing hardware because of stability problems plaguing the CRC’s system. This was a very powerful machine at the time and not only came with Intel processors but also with Apple’s new 10.5 Server operating system, Leopard.

Not much thought was given to the fact that it ran a new OS and could potentially be incompatible with the existing VectorBase infrastructure. All the core networking services were easily set up through a process similar to 10.4 and were perfectly functional. The machine’s primary purpose of running a job queuing and executing system was easily constructed using the Xgrid software that came pre-installed with OS X 10.5 and, conveniently, the previous versions of OS X, 10.4. This meant all VectorBase machines could easily submit jobs to be run on the grid without needing any modification. The only hiccup to the configuration was poor documentation on Apple’s part. The lack of documentation on Xgrid user permissions and details on how to submit complex jobs meant that the grid took slightly longer than anticipated to become functional.
Sadly, this was the last time a hardware upgrade at VectorBase would be so simple. This was also the last machine to use the original naming scheme of characters from the movie, “Dude, Where’s My Car?” The next wave of new hardware came at the beginning of 2009. This time, eight new machines were purchased to replace the aging supporting servers for the site. The pre-production, development, head-node, ftp, mail, and the secondary web machine were all slated for upgrades. This process involved bumping the current production machines down to pre-production and using the new hardware for the live line. It also replaced the three development machines with an experimental one-machine setup. The argument for one machine was that the administration and financial cost for two more machines was unnecessary given the minimal load imposed on the development line. For the new hardware, VectorBase adopted a new server naming scheme based on The CW’s television show, “Gossip Girl”. The names and functions of the new machines are listed in Figure 1.2
Figure 1.1 Pre-Migration Hardware Configuration circa August 2007. Here VectorBase is still using the CRC for job processing, a RAID array for data storage, and an outdated Compaq system for its external DNS server.
1.2 Setting Up Network Services

The first steps in incorporating a new machine into the VectorBase network are fairly standard procedures. First, the MAC address of the network interface card on the new machine is identified and entered into the DHCP service along with a unique IP address on the firewall, Jeff. This allows the new machine to receive the same IP address every time it boots up and connects to the network. The second step is to assign the machine a hostname and enter the hostname and statically assigned IP address in the DNS configuration on the head-node machine. This allows all the other machines on the network to identify and
connect to the new machine by only using its name, without need to remember its actual IP address. The final step is to connect the machine to the LDAP service on the head-node. LDAP is a directory service that provides a centralized location to set user and group permissions as well as user passwords for the entire network. This saves users the time and trouble of needing to log in to all twenty plus machines and changing passwords on each machine if they ever need to update their password. With LDAP, a password change on one machine will propagate to the rest of the servers. This also saves administrators the need to create user accounts or change group permissions across all the machines as well. Detailed instructions on adding new hardware to the VectorBase network can be found in Appendix A.

1.3 SSL Certificates

Configuring machines to connect to LDAP in OS X 10.5 is slightly different than 10.4. Since passwords are involved, LDAP connections are made using encrypted channels. To establish these channels, each machine needs its own unique security certificate. The machine receiving the connection needs a copy of all the other machines certificates. Upon initiation of a secure connection, the machine accepting the connection verifies that the certificate of the incoming machine matches its local copy of the certificate. In 10.4, as long as the LDAP server had a copy of the client’s certificate, the system worked fine. However, on 10.5 the LDAP clients need to have a copy of the server’s certificate. It was also
discovered that additional configuration options needed to be set. These are the necessary steps to set this up:

```bash
# Copy LDAP server’s .pem certificate to client machine
sudo scp gossipgirl:/etc/certificates/gossipgirl.vectorbase.org.pem /System/Library/OpenSSL/certs

# Copy the Certificate Authority’s .pem certificate to the client machine
sudo scp gossipgirl:/etc/certificates/VectorbaseCA/VectorbaseCA.pem /System/Library/OpenSSL/certs

# Identify hash values of these two certificates
openssl x509 -noout -hash -in gossipgirl.vectorbase.org.pem
openssl x509 -noout -hash -in VectorbaseCA.pem

# Create link to the hashed values with .0 at the end
cd /System/Library/OpenSSL/certs
sudo ln -s gossipgirl.vectorbase.org.pem 8c5d1d8b.0
sudo ln -s VectorbaseCA.pem 28d57f7a.0

# Edit /etc/openldap/ldap.conf to have these lines:
DEREF never
REFERRALS off
TLS_REQCERT demand
TLS_CACERTDIR /System/Library/OpenSSL/certs

# Check the SSL box in the Directory Server Utility. Then reboot
```

These certificates prove authenticity between all the machines running on the internal network of VectorBase. Users of the web site have no interaction with these certificates. They are however, forced to accept the authenticity of a different SSL certificate if they wish to access secure content through the https protocol. This web site certificate was created by VectorBase and approved by the VectorBase Certificate Authority. The VectorBase CA is by no means a nationally accredited authority but it does provide the same level of security as any other CA and SSL certificate. Since VectorBase is not an official CA and was
created by the site, using the VecotrBase CA to sign our own SSL certificate makes it a self-signed certificate. All users accepting self-signed certificates are warned by their internet browsers that the site isn’t a trusted resource. A full screen of red typically accompanies the warning and looks very intimidating to users who might not know any better. Earlier generations of web browsers didn't display such warnings and scaring users away wasn’t an issue.

In order to maintain a professional appearance, VectorBase chose to have its web certificate signed by an approved Certificate Authority. After shopping around at several authorities, GoDaddy.com was chosen as the CA to sign the VectorBase certificate. This certificate is valid through May 11, 2015. Upon activation, this newly signed certificate removed all warnings to not trust the site and also removed the amateur level of handling security certificates.

Additional information for the certificate can be viewed with the following command on any of the web machines:

```
openssl x509 -text -in /Volumes/Web/vectorbase/ensemble/conf/ssl/vectorbase.org.crt
```

Information from the GoDaddy certificate that accompanies the Vectorbase certificate can be obtained from this command:

```
echo '' | openssl s_client -connect www.vectorbase.org:443
```
1.4 FTP Server

Switching the production FTP machine to Chuck was the first migration of the group. The FTP machine is the least critical machine on the site. All its files are mirrored on the web server. If it were to go down, users would still be able to download VectorBase files. Apple’s FTP service is not used at VectorBase because of its lack of support for anonymous connections and downloads. Instead, the third party application, pureFTPd, is the FTP server of choice.

The latest version of pureFTPd source code was downloaded from the organization’s web site, http://www.pureftp.org, and configured with the command:

```
./configure --with-tls --with-certfile=/etc/certificates/ --with-virtualchroot --with-pam --with-everything --with-largefile --with-ldap
```

After installation, the OS X specific application, PureFTPd Manager was downloaded from http://jeanmatthieu.free.fr/pureftp/ and installed to facilitate a simplified configuration of the service. Most importantly, the machine’s internal IP address of 192.168.1.40 needs to be masked so it appears to users as the external IP of the web site, 129.74.255.230.

Some users had been submitting complaints because their FTP client programs were able to connect but not transfer files from the FTP server. The users having problems were the ones who were unable to configure the basic options in their FTP clients to switch from Passive Mode to Active Mode.
Previously, VectorBase had only supported Active Mode transfers. Since a new FTP service was being configured, this matter was further investigated. To add support for Passive Mode transfers, the following lines were added to the site’s main firewall configuration on Jeff (/etc/pf.conf):

\begin{verbatim}
ftp_passive     = "9020:9120"

rdr on $ext_if proto tcp from any to $ext_if port $ftp_passive -> $ftpServer
pass in quick on $ext_if proto tcp from any to $ftpServer port { 21, $ftp_passive } modulate state flags S/SA
pass out quick on $int_if proto tcp from any to $ftpServer port { 20, 21, 9020:9120 } keep state
\end{verbatim}

1.5 Email Server

In order to keep the VectorBase email archives in tact, all mail databases from the original email machine, Tommy, were backed up as described in the OS X Migration guide, page 33. The following directories and file were copied to the new Email machine, Blair.

/\var/imap/
/\var/spool/imap/
/\usr/bin/cyrus/bin/ctl_mboxlist

As outlined on pages 40-41 of the guide, the following commands were then run on Blair:

\begin{verbatim}
sudo -u cyrusimap /System/Library/ServerSetup/MigrationExtras/61_migrate_cyrus_db
sudo /usr/bin/cyrus/bin/reconstruct -i
\end{verbatim}
This ensured all email lists, member information, and all emails sent to the lists would be available on the new machine. The remainder of the email system is configuration is identical to the current machine and is as follows in ServerAdmin:

![ServerAdmin Mail Configuration](image)

**Figure 1.3 Email Configuration: General Tab**

The previous instance of the mail system didn’t allow for email to be sent through the VectorBase mail server by clients inside the VectorBase network. Several amendments to the firewall configuration had to be made in order to add this functionality.

```
/etc/pf.conf on Jeff:

rdr on $ext_if proto tcp from any to 129.74.255.228 port $EMAIL -> $mailServer
```
# Enable clients on internal network to access mail.vectorbase.org correctly ( http://www.openbsd.org/faq/pf/rdr.html )
rdr on $int_if proto tcp from any to $ext_if port $SSL_IMAP -> $mailServer port $SSL_IMAP
rdr on $int_if proto tcp from any to $ext_if port $SSL_SMTP -> $mailServer port $SSL_SMTP
rdr on $int_if proto tcp from any to $ext_if port $SSL_SMTPSUB -> $mailServer port $SSL_SMTPSUB
rdr on $int_if proto tcp from any to $ext_if port 82 -> $mailServer port 80
rdr on $int_if proto tcp from any to $ext_if port $SMTP -> $mailServer port $SMTP
no nat on $int_if proto tcp from $int_if to $int_if:network
nat on $int_if proto tcp from any to $mailServer port $SSL_IMAP -> $int_if
nat on $int_if proto tcp from any to $mailServer port $SSL_SMTP -> $int_if
nat on $int_if proto tcp from any to $mailServer port $SSL_SMTPSUB -> $int_if
nat on $int_if proto tcp from any to $mailServer port 80 -> $int_if
nat on $int_if proto tcp from any to $mailServer port $SMTP -> $int_if

The original configuration of the VectorBase email lists had all the web traffic running through port 82 and had the firewall direct the traffic to the Mailman machine. This setup led to many complications and specialized configurations.

To simplify the mailman and firewall configuration, the mailman traffic was treated as the rest of the web traffic and run on port 80. That way it could be routed through a virtual host on the central web server, which forwarded the requests to the Mailman machine. This removed the port 82 link from all the URLs and allowed for administration of the lists on the internal network, which was previously impossible, as well as external. As an added bonus, this allows for easier tracking of the Mailman web traffic.
When VectorBase staff log in to Mailman to administer the mail traffic they must submit their passwords. Previously, all these passwords were transferred on port 80 in clear text. Amendments were made to Mailman and the web server to automatically redirect all Mailman traffic to an https connection in order to encrypt the sensitive information.

1.6 Secondary Web Server

The secondary web server at VectorBase is responsible for services such as the Project Management Wiki, the Developer’s Wiki, the Awstats page, and Subversion. Not only does the machine store all the code changes in the subversion repository but it also runs a web-based front end to the service. The front end provides an easy way to browse through the code and view all the changes that have been made over the years. The machine runs its own instance of the PostgreSQL database server for storing wiki information.

Configuration of the software on this machine was simplified because there was only one OS X specific configuration step. That step was to install Apple’s X Code Developer Tools. The tools contain essential programs for compiling source code on OS X and are included on the OS X installation DVD.

All necessary software was installed through the MacPorts application (http://www.macports.org/). This program is similar to yum or apt-get on Linux systems. It will go out to the internet to download a program and install it on the machine, along with any library or application dependencies it needs. One major
The difference between the Linux tools and MacPorts is that MacPorts will fetch the source code and compile it on the machine where the Linux tools merely download pre-compiled binaries of the software. In that respect, it is much more similar in function to Gentoo Linux's Portage system. Compiling the software from source is why the Developer Tools are a prerequisite of using MacPorts.

The following commands were used to install the required packages through MacPorts:

```bash
port install curl
port install gmp
port install libmcrypt
port install libiconv
port install ssl
port install openldap
port install mysql5
port install postgres84
```

The current version of Apache's httpd source code was downloaded from the Apache site and configured with the command:

```bash
./configure CFLAGS="-arch i386 -03" --prefix=/opt/local/apache2 --with-mpm=worker --with-apr=/opt/local/bin/apr-1-config --with-apr-util=/opt/local/bin/apu-1-config --with-ssl=/opt/local --enable-authnz_ldap=shared --enable-ldap=shared --enable-dav=shared --enable-dav_svn=shared --enable-authz_svn=shared --enable-rewrite=shared --enable-ssl=shared --enable-dav=shared --enable-dav_fs=shared --enable-mime=shared --enable-alias=shared
```

After apache is built and installed, subversion can be installed and linked against the apache build to create the necessary svn module for apache.

```bash
port install subversion +mod_dav_svn
```
The current version of PHP 5.2 was also downloaded from php.net and configured with the command:

```
```

However, when compiled in OS X, changes need to be made to PHP’s source code in order for it to build correctly and without errors. These are the changes that must be made in the file in $phpRoot/ext/iconv/inconv.c

```
diff -Naur php-5.3.0/ext/iconv/iconv.c php/ext/iconv/iconv.c
--- php-5.3.0/ext/iconv/iconv.c 2009-03-16 22:31:04.000000000 -0700
@@ -51,9 +51,6 @@
 #include <gnu/libc-version.h>
 #endif
-#ifdef HAVE_LIBICONV
-#undef iconv
-#endif

 #include "ext/standard/php_smart_str.h"
 #include "ext/standard/base64.h"
@@ -182,9 +179,6 @@
 }
 /* }}} */
-#ifdef HAVE_LIBICONV
-#define iconv libiconv
-#endif

 /* { {{ typedef enum php_iconv_enc_scheme_t */
typedef enum _php_iconv_enc_scheme_t {
```
All the VectorBase specific data on this machine is stored in the /var/vectorbase directory. This includes PostgreSQL, subversion, and the web data. When PostgreSQL is started it needs to be pointed at that directory with the argument “-D /var/vectorbase/postgres_data”. The Apache configuration file in /opt/local/apache2/conf/ points to these directories as well. Its configuration file should work as is without the need for any modifications.

1.7 Production Web

Migration of the Production Web server was fairly similar to the Secondary Web server. The Developer Tools and MacPorts needed to be installed first.

These are the programs that must be installed through MacPorts:

```
port install openssl
port install jpeg
port install libpng
port install freetype
port install gd2
port install libmcrypt
port install apr
port install apr-util
port install postgresql81
port install cyrus-sasl2
port install libxslt
port install mysql5
port install lynx
port install wget
port install ncftp
port install gpgme
port install gsed
port install gawk
port install graphviz-devel
port install gmp
port install curl
```
Once these applications are installed, the Apache web server source code should be downloaded then compiled with this command:

```
./configure CFLAGS="-arch i386 -O3" --with-apr=/opt/local/bin/apr-1-config --with-apr-util=/opt/local/bin/apu-1-config --enable-deflate --enable-so --enable-ssl=shared --with-ssl=/opt/local --enable-proxy=shared --enable-rewrite --sysconfdir=/Volumes/Web/vectorbase/ensembl/conf --with-perl=/usr/bin/perl --enable-speling
```

There are quite a few Perl modules that must be installed for Ensembl and BioMart to function properly. These are the commands to install those modules:

```
cpan
install Bundle::CPAN
#-------- V copy/paste these
install CGI
install HTML::Tagset
install GD
install XML::Parser
install Digest::MD5
install Storable
install XML::Simple
install Parse::RecDescent
install PDF::API2
install OLE::Storage_Lite
install Spreadsheet::WriteExcel
install HTML::Template
install Mail::Mailer
install Math::Bezier
install IO::String
install Image::Size
install GD::Text
install Class::Std
install Log::Log4perl
install Readonly
install XML::DOM
install CGI::Session
install Exception::Class
install List::MoreUtils
install Test::Exception
install Number::Format
```
install IO::Compress::Zip
install File::Temp
install ExtUtils::XSBuilder::ParseSource
install IO::Scalar
install LWP::Parallel
install Data::UUID
install Template::Plugin::Number::Format
install Time::HiRes
install LWP
install DBI
install SOAP::Lite
install MIME::Types
install BSD::Resource
install Hash::Merge
install Compress::Bzip2
install Cache::Memcached
force install Class::DBI
install Class::DBI::Sweet

#--------- following are manual---------
install DBD::Pg  #path to pg_config:
/opt/local/lib/postgresql81/bin/pg_config
install DBD::mysql
  look DBD::mysql
  # MUST use same major version my mysql_config for the database
  you're connecting to. I.E. Don't try and use mysql5 config to connect
  to mysql4 database.
  perl Makefile.PL --testdb=testdb --testhost=mrlee.vectorbase.org
  --testuser=db_public --testpass=limecat --
  mysql_config=/opt/local/bin/mysql_config5 --cflags='-
  I/opt/local/include/mysql5/mysql -fno-omit-frame-pointer'
  make
  make test
  make install
  make clean
  exit
install Apache::Test
  # httpd: /usr/local/apache2/bin/httpd
  # apxs: /usr/local/apache2/bin/apxs
  # skip test suite, it will fail for Darwin = not install
install ModPerl::Registry
  #apxs: /usr/local/apache2/bin/apxs
  # skip test suites
install Apache2::Request  #this one is the only module from the
original required set that breaks now
  # skip test suites
  look Apache2::Request
  perl Makefile.PL --with-apache2-apxs=/usr/local/apache2/bin/apxs
  make
  make install
Once the Perl modules have been compiled and installed, the Apache interface to Perl can be installed. These are the steps to download, compile, and install mod_perl.

```bash
make clean
exit
install Apache2::Upload
   # same procedure as Request
install Bio::Root::Root # all optional modules, default options for the rest
install Bio::Das
install Bio::Das::Lite
   look Bio::Das::Lite
   # comment out lines 861-863 of
   # blib/lib/Bio/Das/Lite.pm
   #   $curl->setopt( CURLOPT_PROXYUSERNAME, $self->proxy_user );
   #   $curl->setopt( CURLOPT_PROXYPASSWORD, $self->proxy_pass );
   #   $curl->setopt( CURLOPT_NOPROXY, join q(,), @{ $self->
>no_proxy } );
perl Makefile.PL
   make
make install
make clean
exit
exit
```

Installation of PHP is nearly identical to the steps listed in the Secondary Web Server section. Once difference is that on the production web machine, PHP needs to be built to support MySQL interaction. On OS X, this is facilitated through the following commands:

```bash
gunzip -d mod_perl-2.0-current.tar.gz
tar -xf mod_perl-2.0-current.tar
cd mod_perl-2.0.*
perl Makefile.PL PREFIX=/usr/local/apache2
MP_APXS=/usr/local/apache2/bin/apxs
make -j10
make install
```
The PHP source code needs to be modified in the same way as the Secondary
Web machine. Then PHP can be configured with this command:

```
```

All the required applications to run the web server have been installed at this
point. The remaining steps are to check out the current version of the web code
from SVN and set a few permissions on those files and directories. This can be
achieved with the following commands:

```
cd /Volumes/Web/vectorbase     # Created by Apache installation
rm -r ensembl                   # Don't want this old conf file
sudo mkdir .svn/             # Don't need to change permissions on .svn/ directory
sudo chgrp -R webdevs .svn/ && sudo chmod -R g+w .svn/  
cd /Volumes/Web/vectorbase/ensembl/biomart
perl ./bin/configure.pl -r conf/vectorbaseDBLocation.xml

# permissions

cd /Volumes/Web/vectorbase/ensembl
sudo chown -R www tmp img tmp cache

cd /Volumes/Web/vectorbase/sections/Forum
sudo chmod 777 cache/

cd /Volumes/Web/vectorbase/ensembl/biomart/conf/templates/
sudo chown -R _www
```
1.7.1 Separation of VectorBase and Ensembl Code

One of the best things to happen to VectorBase’s web machine was the separation of the VectorBase web code from the Ensembl genome browser. Previously, the two had been integrated to provide a unified look to the site. The genome browser was surrounded by the site’s header and footer and presented in the same viewable width as the rest of the site. It had a clean and professional look but took an enormous effort to maintain. Ensembl updates their code every two months. It took a highly experienced VectorBase developer over two weeks to make all the necessary changes to the VectorBase code in order to keep the unified appearance. In the long run the upkeep was just not worth the effort.

Furthermore, Ensembl is written completely in Perl and envelopes the entirety of Apache’s configuration file in Perl code. This prevented the site from utilizing one of the best features of Apache, its ability to run virtual hosts. Virtual hosts allow more than one web site to be run from the same machine, on the same port. Placing the Ensembl code on a different machine enabled VectorBase to run virtual hosts in Apache. This allowed the site to open more of its internal web content to the internet. This included the development and pre-production sites, the project management wiki, and the Mailman mailing lists.

```bash
sudo chmod 777 /Volumes/Web/vectorbase/ensembl/biomart/results
```
1.8 Production Database

Both MySQL and PostgreSQL database servers are run on the database machine. MySQL is utilized to store the Ensembl data and all other data is stored in PostgreSQL. Migration of the database machine was one of the simplest procedures although it takes a bit of time to compile the software and copy the current db data to the machine. To get things up and running, the Developer Tools and MacPorts were first installed on the machine. Then the two database servers are installed through MacPorts:

```
port install mysql5
port install postgresql81
```

Next, the database data from the previous live database machine should be copied to the new machine:

```
scp -rp mrlee:/Volumes/DB /Volumes/
```

The programs must then be configured to point at the directories where they store their respective data. For MySql, the commands are as follows:

```
mkdir -p /opt/local/var/run/mysql5
chown -R _mysql /opt/local/var/run/
mkdir /Volumes/DB/mysql5/logs
chown _mysql /Volumes/DB/mysql5/logs/
```

Now the MySQL server can be started:

```
```
PostgreSQL is a bit simpler as it only requires the data to be copied over and then given the location of that directory when the server is started:

```bash
sudo -u postgres pg_ctl -D /Volumes/DB/postgresql start
```

1.9 Production Application

The application server is responsible for running all of VectorBase’s web based Java applications. These applications are the site’s main search functionality, the Community Annotation Pipeline, CV Search, and at one point an application to handle communications between the web servers and the job controller. The beautiful thing about the application server is that it runs in Java. Java code and applications are independent of machine architecture. OS X 10.4 and 10.5 both run the same versions of the Java Virtual Machine so migration is very easy.

Apache’s Tomcat is the program the runs the App server at VectorBase. The directory the program resides in is /opt/local/tomcat. This directory and its contents were copied from the previous live machine to the new machine. Then several directories were created for storing application data:

```bash
sudo mkdir -p /Volumes/App/job_tmpspace/job_input/BLAST
sudo mkdir -p /Volumes/App/job_tmpspace/job_input/ClustalW
sudo mkdir -p /Volumes/App/job_tmpspace/job_input/HMMER
```
These were the only steps necessary for migrating the production application server.

1.10 Development and Pre-Production Machines

The application, database, and web servers on the site are replicated across three tiers. These tiers are live production, pre-production, and development. Each tier has a machine for each service. Workload on development and pre-production machines is very low. It was decided to consolidate the application, database, and web machines for these two tiers to reduce administrative and financial costs, resulting in one machine for development and one for pre-production. Integrating these machines has several additional benefits. It simplifies management and administration of the machines since there are only two instead of six. It also provides a small speed increase since the machines no longer have to talk to each other across a network. Lastly, funds could be used on acquiring better machines rather than more machines.

The installation and configuration of the new development machine was identical to the procedures of the new production machines. The only
differences between the configurations being all three services were installed on the development machine. The pre-production tier used the previous live db machine, mrlee, as its new home. It followed the configuration steps identical to the web and application machines.

1.11 Network Services Server

By far the most difficult machine to get up and running was the ‘head-node’ server. This machine is responsible for most of the internal core networking services at VectorBase. It runs the internal Domain Name System, which converts machine names to IP addresses; the Virtual Private Networking server that allows developers outside of Notre Dame to connect to the internal network; and the Lightweight Directory Authentication Protocol daemon, which provides a centralized system of managing groups and user accounts.

The DNS server was simple to set up. Merely copying the DNS configuration files from the previous head-node, Zoltan, was enough to provide that service with everything it needed to function. The files copied over were /etc/named.conf and the directory/var/named. Then the DNS service was started in Server Admin and the program converted the files to 10.5 format automatically. However, it was noticed that any DNS changes made to the system
using the System Admin GUI would create errors in the DNS zone files, causing the DNS service to hault. To get around this, administrators need to directly edit the file /var/named/zones/db.vectorbase.org.zone.apple. Then start and stop the DNS service with the command line calls:

```
$ serveradmin stop dns
$ serveradmin start dns
```

If an administrator has already attempted to edit DNS with Server Admin and needs to get the service back online, they should edit the previously mentioned zone file and look for these lines, currently 73-75:

```
_tcp.dc._msdcs.VECTORBASE.ORG. IN  PTR
_ldap._tcp.dc._msdcs.VECTORBASE.ORG.gossipgirl.vectorbase.org.
_tcp.dc._msdcs.VECTORBASE.ORG. IN  PTR
_kerberos._tcp.dc._msdcs.VECTORBASE.ORG.gossipgirl.vectorbase.org.
_tcp.dc._msdcs.VECTORBASE.ORG. IN  PTR
_ldaps._tcp.dc._msdcs.VECTORBASE.ORG.gossipgirl.vectorbase.org.
```

Changes made through the GUI will remove the gossipgirl string from the lines and it will need to be manual inserted back into the lines to get DNS online.

Set up of the VPN server was also fairly straightforward. Since no log files or older information needed to be retained, a few configuration parameters were entered into the new machine to setup the VPN server. The necessary parameters are shown in the following images.
Figure 1.4 VPN Configuration: L2TP
Figure 1.5 VPN Configuration: PPTP
Configuring LDAP on the 10.5 machine was the most frustrating and difficult experience of the entire migration. The LDAP server contains sensitive information, all the users’ passwords for VectorBase. It makes sense for security reasons that Apple would want to prevent the copying and distribution of user passwords, even though the passwords are encrypted in LDAP. Unfortunately, they went a step too far and prevented any LDAP migrations from occurring. According to the migration guide, moving the LDAP settings is as simple as clicking the export LDAP configuration in System Admin. However, this method will export all the user and group settings except for the passwords. Several
methods were documented in the Apple forums on how to migrate LDAP settings with user passwords. None of the methods found worked.

In order to get the new LDAP system up and running, GossipGirl was first downgraded to OS 10.4. This is a procedure Apple goes out of their way to prevent. Apple has made it impossible to install a version of an Apple operating system that is older than the version the machine shipped with. To get around this, the hard drive of the new machine is first placed in an older machine, running 10.4 in this instance. Then the 10.4 installation DVD is loaded and run on the older machine. The hard drive of the new machine is chosen as the installation target.

The reasoning behind the downgrade is that Apple allows the exporting of LDAP configurations, including passwords on 10.4 to 10.4 machines. The settings were then exported from Zoltan and imported on Gossipgirl. Then an OS 10.5 DVD was inserted into GossipGirl and an upgrade install from 10.4 to 10.5 was performed. This was the only method discovered that would kept all the original LDAP information, including the user passwords intact.

1.12 Conclusion

In hindsight, a more efficient option would have been to downgrade the Operating Systems on the new machines from Leopard, the version pre-installed, to Tiger, the version on all of the machines at VectorBase. This would have saved
a lot of time and frustration involved in the migration process. Having servers running Leopard as their OS produced no noticeable performance benefits, nor did it offer any new features that aided in administration of the machines. It was merely a large hurdle to overcome to keep the site running at its previous level. There would have been trouble initially downgrading the new machines from 10.5 to 10.4 as Apple goes out of their way to prevent old operating systems from being installed on new machines. However, a method for circumventing this limitation was discovered and would have taken no more than a day to implement across all the new hardware.

Moving forward, it would be advisable to consolidate parts of the alternate web machine into other machines. The developer’s wiki and project management wiki should be integrated into the main web machine and their associated databases should be moved to the database server.
2.1 Overview

Distributed computing is a way to divide and conquer. It is a method of farming out tasks of a program to run on several machines simultaneously to decrease the overall runtime of a program. VectorBase takes advantage of distributed computing for processing users’ jobs submitted through the Tools section. This includes BLAST, ClustalW, and HMMER programs. Additionally, user submitted gene annotations received in the CAP are run through Exonerate on the distributed computing system.

Hundreds of user jobs are submitted every day to VectorBase and can use up large amounts of system resources. By farming out these tasks, the VectorBase machines aren’t bogged down with processing user jobs in addition to running the main services of the web site. This allows the site to respond quickly and smoothly to all user requests. Most BLAST jobs contain five to fifty tasks that are
farmed out for simultaneous processing, greatly benefiting users of the site by reducing job runtime.

All VectorBase jobs are processed through the backend distributed system in a similar manner. The first step has always been to take user input through the web interface. This usually includes program parameters and input sequences. That data is sent on to a second server, the application machine. There, the users’ data is shaped and reformatted in a way that the distributed computing system will understand. It is then sent on to the distributed system where the job is executed. When it is complete, the results are stored at VectorBase and then displayed to the user through the web interface.

2.2 Modifications

Over the years, the backend system that executes users’ jobs has undergone several changes. Simplicity, reliability, scalability, developer education, and necessity are all reasons that have placed pressures on the system and required changes to be made. Four of these major modifications are described next.
2.2.1 First Modification

The first change made to the system was to convert the software on the application server from an Apache/PHP combination to a Tomcat/Java implementation. The reasoning behind this move was two-fold. One reason was to give the developer in charge of the change experience with programming in Java and deploying web applications on Tomcat. This relatively simple in scope and straightforward project was a good introduction to the topics since the developer had no previous experience with either. The second reason was that the code for the distributed system was the only code using Apache’s httpd and PHP on the application server. All other applications on the server ran through Tomcat. Running httpd along side Tomcat placed additional administrative overhead on the machine such as starting the services on boot, opening two ports on its firewall, and maintaining updates on two software packages. The setup doesn’t make much sense considering both programs are capable of accomplishing the same task.

The conversion from PHP to Java was carried out smoothly. A class was written in Java to translate the incoming data to a format acceptable by the distributed system and send it on to that system. The class was also able to get results and status updates of jobs from the system. Then a descriptor of the class was written. This descriptor was a list of publically viewable functions the class would respond to and the input parameters and output types of those functions.
This descriptor and Java class were fed to Axis, a program running in Tomcat, which generated a WSDL file and allowed the Java class to be interacted with over SOAP. The end result was a setup that provided the same functionality to the web interface as the original PHP/http implementation.

2.2.2 Second Modification

Perhaps the most drastic change to the system came next. At the time, VectorBase was utilizing computation resources of another group on campus to run all its jobs. Outsourcing was convenient since it removed the need to setup and administer a distributed system of computers by VectorBase personnel. There were a substantial amount of computational resources allotted to VectorBase by this group. Unfortunately, the group rarely achieved a twenty-four seven uptime of their resources. They would also periodically make changes to their setup without informing VectorBase. These changes would bring a halt to all job related activity on the site. Rather than continually being subjected to downtime outside of their control, VectorBase chose to implement their own distributed system. After investigating several options, the choice was made to use Apple’s Xgrid.

Employing Xgrid at VectorBase had several advantages. The first being it is a relatively simple system to use. Apple has developed GUI tools that allow for
intuitive administration of the system, machines may be added or removed with ease. It is also easy to see the jobs running across all members of the grid as well as jobs that have already completed and their results. Another advantage is that Xgrid comes pre-installed on Apple machines running the server version of their operation system, which all VectorBase machines do. Setting up the grid was as easy as checking the GUI box to enable Xgrid.

A lack of documentation was the one noticeable downside to using Xgrid. The first graduate student who attempted to set up the system was unable to figure out its specifics or successfully run VectorBase jobs. After some digging on Apple forums, it was discovered that Xgrid jobs are run as one of two Xgrid users. The users are somebody and nobody. The permissions for these users and the files they can access, write, and execute are stored in files under /usr/share/sandbox/xgridagenttd*.s. The default user, nobody, has very limited permissions and is unable to access most of the files and programs on the computer. VectorBase staff were able to execute simple jobs on the system after making the necessary changes to the permissions file for the nobody user.

Poor documentation for Xgrid also extended into the formatting of batch files. Batch files specify how to carry out larger jobs and are the files submitted to Xgrid to run jobs. The specifics could include running multiple tasks for a job, the order to run the tasks in, having tasks run after other tasks complete, directing input files to the programs, and what to do with their outputs. Learning the correct syntax for all these parts of the batch files was eventually pieced together
after scouring various internet forums. One of the more peculiar aspects was learning input files needed to be Base64 encoded and appended to the actual batch file itself.

2.2.3 Third modification

The transition to Xgrid was successfully deployed at VectorBase. The necessary changes were made to the code on the application server to submit to Xgrid. However, after several months of operation a peculiar hiccup in the system emerged. After being operational for a random time interval, the Java application would cease to respond to any Xgrid requests by the web server, be it job submissions, status checks, or retrieving job results. Upon further investigation, it was discovered the Java code would stop responding at the point where it called the Xgrid program from command line. The temporary solution was to restart Tomcat at a short interval but this disrupted other services on the site. For a permanent fix, the code on the application server was abandoned. All VectorBase machines ran OS X server and had Xgrid pre-installed on them. Code was written in PHP and ran directly from the web server that would handle all aspects of the VectorBase job system. Since the site no longer needed to connect to CRC’s grid engine, which required special permissions and obtaining tokens to access it, there was little need to keep the Job application up and running.
2.2.4 Fourth Modification

The aforementioned system has served VectorBase and its users very well over the past few years. There have been zero unplanned interruptions to the distributed system. However, VectorBase is in the process of moving away from OS X. The newly acquired machines have been setup to run Red Hat Enterprise Linux. This operating system does not support Xgrid. This presents the problem of how to allow the web server to continue to communicate with the distributed system. Rather than continue to fight possible outages with the use of the Java application running on Tomcat, the decision was made to set up another instance of Apache’s httpd on the machine that serves as the Xgrid master. This instance allows the site’s primary web servers to communicate over SOAP to Xgrid. It functions in the same way as the original distributed system setup. PHP is used to intercept and interpret the SOAP messages. The two configuration files for this web service are called server.php and xgrid.wsdl. They are stored on gene in /Library/WebServer/xgrid. These files are documented in Appendix B.
CHAPTER 3

BLAST AT VECTORBASE

3.1 Overview

BLAST is a cornerstone of bioinformatics research. The program allows researchers to find the best-fit match between any two segments of sequence. A query sequence is compared against all the known sequences in the organism it was found in to find orthologs or against other organisms to find paralogs. Upon discovering a new sequence of interest, one of the first actions to take is to run a BLAST and see where else the sequence is found. As a result, VectorBase’s BLAST section is the most popular feature of the site. Well over 100 BLAST jobs are submitted through the site every day. The site receives BLAST job submissions from all over the world. The majority of jobs come from major universities in the United States.
Figure 3.1: VectorBase Job Submissions by Month
Figure 3.2 Country of Origin for Jobs Submitted To VectorBase

- United States: 56%
- United Kingdom: 13%
- France: 6%
- Brazil: 4%
- Japan: 2%
- Mexico: 2%
- Other: 17%
For years the user interface to BLAST had remained the same as the day it was created. The only modifications to its appearance were periodically updating query databases or the rare addition of a new organism to BLAST against. Its design employed older web standards that made it difficult to interact with and navigate the interface. With the abundance of new technologies, web standards, and tools available to developers, VectorBase decided to give its BLAST section a facelift.

Figure 3.3 Top Submitting Metropolitan Areas In the US With Associated Universities in Those Areas
Creating an elegant and practical web form for users to interact with BLAST takes some thought. There are many parameters necessary to perform a BLAST job. At VectorBase, users are given the power to adjust some of the more popular parameters of their searches such as program type, e-value, word size, scoring matrix, complexity masking, and the number of returned results. Combining those options with an area to post a query sequence and display the large list of databases available for searching places a high demand on screen real estate. Laying out a form that organizes all these options in an efficient and orderly manner is no trivial task.

Complicating the design even further is VectorBase’s use of a rather small, fixed width resolution to display its content. This poses a difficult task on the designer to cram all the necessary configuration options into the user interface window and present it in a manner that is clearly visible and allows for easy, intuitive navigation. The original design, although perfectly functional, had a few shortcomings in the areas of visibility and easy navigation. Specifically, there was difficulty in viewing and selecting query databases, setting options for the search, the requirement of precise clicks on tiny check boxes, and the need to scroll up and down the page if dealing with multiple datasets and search options. There were also two problematic navigation issues the interface suffered from. One was the fairly often scenario where a user would select multiple search options and specific datasets to search, run the job and not be satisfied with the results. In that case, the user would have to suffer the nuisance of navigating back to the submission form and re-entering all the old parameters as well as the
tweaked ones to run the job again. The second issue was also quite common. If a user entered incorrect data or selected the wrong option on the submission form, the data would first be sent to the server, the server would reply that there was an error with the form, and then the user would have to again re-enter all the data on the form to correct the mistake before the job would run successfully.
3.2 Original User Interface

![Original BLAST User Interface](http://web.archive.org/web/20070728085103/http://www.vectorbase.org/To
ols/BLAST/)

Presented here is the previous edition of the BLAST user interface. Initially, it is hard to tell what is going on with this layout. The developer needed to write in explicit instructions on how to use the interface. Knowing how and where to select databases to BLAST against is difficult to figure out because of the nonstandard form elements that contain the lists. A better choice would have
been to use + and – icons to indicate a drop down list of options for the organism databases. Any kind of familiar form element would aid users and let them know these are expandable lists. All of the advanced configuration options are also hidden from view in an unorthodox manner.

Figure 3.5 Original User Interface Expanded
This is what happens when the organism boxes are clicked. Each one expands a list of searchable databases. The list is so long it spills down the page and the user must scroll up and down the page to view all the form’s contents. Even more frustrating is that each tiny check box must be clicked for each database. Clicking on the text describing the database should be able to check the box but it does not. There is no easy way to select all the databases for a single organism or all the organisms. This means over twenty clicks are necessary to select all the databases for nucleotide searches.

3.3 New Technologies for New Interface

The fact that the BLAST section was unchanged wasn’t necessarily a detriment to the site. Upon initial deployment, the interface used the current layout methods and data submission practices of its day. Over the years, newer technologies have been developed that aid in the ease of user navigation through the interface. Keeping the site up-to-date with the current technologies shows the users the site is continually working to bring them the best experience possible. The two important technologies being utilized in the new UI are CSS and AJAX.

Cascading Style Sheets, CSS, are really more of a benefit for the web developers. They allow the code that controls the layout, colors, and fonts of the
page to be much cleaner and written in a more legible format. Adding new features or fixing bugs is simplified when using CSS since nearly 70% of the old code, which dealt with the layout, has been removed. There are a few aspects of the style sheets users will notice. HTML 5 is a new web standard that brings in fancy design elements to web pages. VectorBase has implemented rounded corners and color gradients from HTML 5 into the style sheets for BLAST.

The AJAX technology is the workhorse behind the new BLAST interface. The acronym stands for Asynchronous JavaScript and XML. It allows the main interface page to send and pull information from other pages on the site when necessary and then update itself with the new data. It can do all of this without the need to refresh itself. The original BLAST system would go to a job progress page after a job was submitted. There it would check the status of the job, wait fifteen seconds, then reload itself so it could check the job status again. With AJAX, users don’t even have to leave the initial submit interface. The main UI page can submit the job and then continually send out requests to get the status of the job from another page. It can then update itself with the data and let the user know how much of the job is completed without the need to reload the page.

Users are now presented with a single page, from which they can control the majority of their BLAST operations. This page queries other pages and updates itself with information retrieved from those queries. Initial results are displayed on the main mage
3.4 Error Checking

One of the largest frustrations with the original interface was the method it used for catching errors in users’ job parameters. Upon submission, the parameters for the job were sent to the web server to be parsed out and checked for correctness. If invalid, the user would be redirected to the submission form with a message describing what caused the error but the form they previously filled out would be empty.

The best method to resolve this inconvenience was to move the error checking to the client’s machine. Client side error checking is done by the user’s web browsing application. Written in JavaScript, the error checking on the interface runs when the user clicks the ‘BLAST!’ button. If there are errors on the form, the user is still notified with a message describing the error, but now the user never leaves the page and all the data that was entered in the form remains intact. As an added feature, the section containing the error is highlighted in red so the user can quickly locate the problem.
Figure 3.6 An error scenario in the updated UI. A large red area draws user’s attention to item that needs correction. User’s database selections remain intact after error was thrown.

3.5 Locally Run Jobs

VectorBase has good relations with the University’s Center for Research Computing department. Previously, all jobs from the site had been sent to execute through their Grid Engine. However, their grid suffered from stability
issues towards the end of 2008, which had a negative impact on the VectorBase site. The decision was made to move away from the CRC cluster and run all jobs locally on VectorBase hardware. After investigating several options, Apple’s Xgrid was chosen for the job.

Xgrid has a lot of advantages for a distributed job system at VectorBase. First, it runs on the OS X operating system, which is what all the VectorBase servers run. Second, it is already installed on all the machines. Third, it is very easy to configure. Lastly, any user may submit a job to the cluster, which is very convenient since all submissions come the web server and thus are run as the web server user, _www.
3.6 BLAST Workflow

This figure illustrates how all the scripts of the BLAST system work. Input.php is the UI script and collects and verifies all user data. Using AJAX, it sends the data to submit.php which takes all the parameters, formulates them into a file XGrid can use, then submits that file to XGrid, which immediately begins running the job. Submit.php then loads processing.php, which queries XGrid on the status of the job. XGrid will reply with the percentage of the job that is completed. Processing.php relays this information to input.php where the user can see the
job progress. Input.php will then continue to update the status of the job from processing.php over AJAX. Once processing.php receives word that the job is completed, it loads results.php. This final page parses the results, stores them in a database, and then returns a table of hits to input.php so the user can see results on the same interface their query form is on. This way, if they are unhappy with their results, then can easily modify their query and BLAST again without the need to navigate to other pages or re-enter data on the form. The full source code for these files has been included in Appendix C.
3.7 Updated User Interface

The updated interface is shown here. Immediately, the eyes are drawn to brightly colored organism datasets that help organize all the searchable databases. Also, databases are already viewable in an area that is the same color as the dataset they belong to. This gives the user a more intuitive idea of where to click to see databases for other organisms. There is no longer a need for clicking to expand and collapse the database views as in the previous interface.
One click will show or select all the databases for an organism. The BLAST program options are clearly visible and do not require a click to view and modify them. Additional functionality was added to account for which databases are selected depending on the type of blast program selected. If a user selects the Anopheles data set, then switches from a blastn to blastp job, the interface will automatically deselect the nucleotide databases and select the peptide database for the user. Maximum E-Value, Word Size, and Complexity Masking are also automatically updated under specific conditions.

3.8 Future Work

There are still many areas to improve upon for the BLAST section at VectorBase. Currently, when a user submits a BLAST search across several databases, each database search is farmed out to a separate processor core. This drastically reduces the job run time for users. However, after the searches are completed, all the results are dumped into one large file and a Perl script parses out all the results details. It would beneficial to have multiple Perl jobs running, each parsing the results from a single database. This would help reduce runtime even further.

Adding new types of query sequences to the interface is another feature that would aid users. The form currently only accepts sequences in FASTA format. While perfectly functional, it limits the users choices. The GenBank format for sequences is one that should be added to the interface.
Another horrific bottleneck in the BLAST system is how results are stored on the VectorBase machines. All the results are stored in a database. This is desirable since it gives a quick and easy way to retrieve results. However, rather than storing individual results parameters in their own fields in the database, past developers created a custom format which resembles XML. They decided to store that entire XML like entry in its own field. This means that each time a user wants to retrieve a BLAST result, the results page needs to re-parse the database entry. This puts a high and unnecessary load on the web server. It also prevents larger jobs from being executed at VectorBase since the resource demand is so high. Currently there is a limit of 20 query sequences. If more sequences are searched with, the parsing of results will cause the web server to crash. A better handling of the results parsing would drastically increase the number of query sequences.

Extrapolating on the previous improvement of allowing more input queries, there needs to be a better way of displaying the results to the users. The current interface works well for a small number of input queries searched against a small number of databases. However, allowing for many input queries searched against many databases leaves a chaotic display of results, causing users to search through their search results. Users should at least be presented with an option to display all their results side by side to see how they compare to each other in addition to the current view of how they compare to results within each database.
APPENDIX A:

PROCEDURES FOR ADDING NEW MACHINES TO VECTORBASE

Locate the MAC address on the new machine.

Log in to Jeff, or primary firewall. Edit /etc/dhcpd.conf and add an entry for the new machine. Make sure the assigned IP address is unique. Run `ps aux | grep dhcpd` to locate the process id of the dhcp daemon. Kill this process with -9 and then restart the dhcp daemon: `/usr/sbin/dhcpd -q em0`

Log in to gossipgirl, or internal DNS machine. Edit the vectorbase zone file. On Gossipgirl this is at /var/named/zones/db.vectorbase.zone.apple. Add an entry for the new machine in this file with the chosen IP address in the previous step. Save the changes to the file then restart the DNS service. On gossipgirl this can be done with the command: `sudo serveradmin stop dns && sudo serveradmin start dns`

At this point the machine can be connected to the VectorBase network and all networking services should work. Power the machine on to configure the OS.

Upon setting up the new operating system, connect the machine to LDAP. Currently, the service is running on gossipgirl, ldap://gossipgirl.vectorbase.org.
If a search base needs to be specified use the following:

dc=gossipgirl,dc=vectorbase,dc=org

Create SSL certificate

This concludes the basic setup for new machines. Further configuration may be needed to poke holes in the firewall for incoming traffic or additional DNS entries made on the internal or external servers depending on the requirements of the new server.
APPENDIX B:

JOBS WEB SERVICE CODE

The following two files are all that is necessary to run the VectorBase jobs web service. The first file is the Web Service Description Language file describing the functions available for the web service as well as their expected inputs and outputs. The second contains all the functionality of the web service. Note that server.php calls the xgrid program from the command line, which is only available on OS X machines.

B.1 xgrid.wsdl

```xml
<?xml version='1.0' encoding='UTF-8' ?>
<definitions name='Xgrid' targetNamespace='http://jobs.vectorbase.org/xgrid.wsdl'
xmlns:tns='http://jobs.vectorbase.org/xgrid.wsdl'
xmlns:soap='http://schemas.xmlsoap.org/wsd1/soap/'
xmlns:xsd='http://www.w3.org/2001/XMLSchema'
xmlns:soapenc='http://schemas.xmlsoap.org/soap/encoding/
xmlns:wsdl='http://schemas.xmlsoap.org/wsd1/'
xmlns='http://schemas.xmlsoap.org/wsd1/'>

<message name='submitBatchRequest'>
  <part name='xgridXml' type='xsd:string'/>
</message>

<message name='submitBatchResponse'>
  <part name='Result' type='xsd:string'/>
</message>

<message name='getStatusRequest'>
</message>
```

60
<part name='jobId' type='xsd:string'/>
</message>
<message name='getStatusResponse'>
  <part name='Result' type='xsd:string'/>
</message>

<message name='getResultsRequest'>
  <part name='jobId' type='xsd:string'/>
</message>
<message name='getResultsResponse'>
  <part name='Result' type='xsd:string'/>
</message>
<operation name='getStatus'>
<soap:operation
soapAction='http://jobs.vectorbase.org/server.php'/>
<input>
<soap:body use='encoded' namespace='urn:xmethods-delayed-quotes'/>
</input>
<output>
<soap:body use='encoded' namespace='urn:xmethods-delayed-quotes'/>
</output>
</operation>

<operation name='getResults'>
<soap:operation
soapAction='http://jobs.vectorbase.org/server.php'/>
<input>
<soap:body use='encoded' namespace='urn:xmethods-delayed-quotes'/>
</input>
<output>
<soap:body use='encoded' namespace='urn:xmethods-delayed-quotes'/>
</output>
</operation>

<definitions>
B.2 server.php

```php
//add all functions here that soap server should handle
function submitBatch($xml)
{
    $filename = genRandomString(6)."_xgrid.xml";
    file_put_contents($filename, $xml);
    $output = shell_exec("/usr/bin/xgrid -h gene -job batch $filename");
    exec("rm $filename");
    preg_match("#jobIdentifier = \(\d+\)#s", $output, $match);
    // return the job id, error text otherwise
    if($match[1])
        $return = $match[1];
    else
        $return = $output;
    return $return;
}

function getStatus($jobId)
{
    $output = shell_exec("/usr/bin/xgrid -h gene -job attributes -id $jobId");
    return $output;
}

function getResults($jobId)
{
    $output = shell_exec("/usr/bin/xgrid -h gene -job results -id $jobId");
    return $output;
}

// initialize soap server
ini_set("soap.wsdl_cache_enabled", "0"); // disabling WSDL cache
$opts = Array();
$opts["compression"] = SOAP_COMPRESSION_ACCEPT | SOAP_COMPRESSION_GZIP;
$server = new SoapServer('xgrid.wsdl', $opts);
$server->addFunction('submitBatch');
$server->addFunction('getStatus');
```

$server->addFunction('getResults');
$server->handle();

function genRandomString($length) {
    $characters = '0123456789abcdefghijklmnopqrstuvwxyz';
    $string = '';  
    for ($p = 0; $p < $length; $p++) {
        $string .= $characters[mt_rand(0, strlen($characters)-1)];
    }
    return $string;
}

?>
APPENDIX C:
BLAST GUI CODE

The following four files are the complete works of the BLAST GUI at VectorBase. A flowchart of their operation is located in section 3.5.
<?php

$BLAST_FORM_NAME = "blast_input_form";
include("blast_errors.php");

//blast_programs.php
$blast_programs["blastn"] = "Nucleotide vs. Nucleotide";
$blast_programs["tblastn"] = "Peptide vs. Translated Nucleotide";
$blast_programs["tblastx"] = "Translated Nucleotide vs. Translated Nucleotide";
$blast_programs["blastp"] = "Peptide vs. Peptide";
$blast_programs["blastx"] = "Translated Nucleotide vs. Peptide";

//this needs to grab the associated ids and their params
if ($_GET['blast_id']){
    $job_query = "select * from job_params where job_id = " . $_GET['blast_id'];
    $job_rs = pg_query(DB::getJOB_DB(), $job_query);
    $redoDatabases = array();
    while($job_row = pg_fetch_assoc($job_rs)){
        var_dump($job_row['argument']);
        if ($job_row['argument'] == "sequence"){
            $redoSequence = $job_row['value'];
        } elseif ($job_row['argument'] == "program"){
            $redoProgram = $job_row['value'];
        }
25 }  
26 elseif($job_row["argument"] == "target_database"){
27     array_push($redoDatabases, $job_row["value"]);
28 }
29 elseif($job_row["argument"] == "wordsize"ollision){
30     $redoWordsize = $job_row["value"]; 
31 }
32 elseif($job_row["argument"] == "numhits_oneline"ollision){
33     $redoNumhits_oneline = $job_row["value"]; 
34 }
35 elseif($job_row["argument"] == "numhits_align"ollision){
36     $redoNumhits_align = $job_row["value"]; 
37 }
38 elseif($job_row["argument"] == "evalue"ollision){
39     $redoEvalue = $job_row["value"]; 
40 }
41 elseif($job_row["argument"] == "scoringmatrix"ollision){
42     $redoScoringmatrix = $job_row["value"]; 
43 }
44 elseif($job_row["argument"] == "masking"ollision){
45     $redoMasking = $job_row["value"]; 
46 }
47 }
48 }
49
50 ?>
51
52 <script language="JavaScript" type="text/javascript">
53 <!--
54     var http_request = false;
55 </script>
var response = '';
var jobId = '';
var selectedOrganism = '';
var detailEST = false;

// auto sets wordsize to 11 for blastn, 3 for anything else
function setWordSize(blast_program){
    if (blast_program == 'blastn'){
        document.<%=BLAST_FORM_NAME;?>.wordsize.selectedIndex = 4;
    } else {
        document.<%=BLAST_FORM_NAME;?>.wordsize.selectedIndex = 0;
    }
}

// set low complexity masking off for blastp
function setLowComplexity(blast_program){
    if (blast_program == 'blastp'){
        document.<%=BLAST_FORM_NAME;?>.dust.selectedIndex = 1;
    } else {
        document.<%=BLAST_FORM_NAME;?>.dust.selectedIndex = 0;
    }
}

function wordSizeCheck(){
    if (getSelectedBlastProgram() == 'blastn' &&
    document.<%=BLAST_FORM_NAME;?>.wordsize.options[0].selected){
        document.<%=BLAST_FORM_NAME;?>.wordsize.selectedIndex = 1;
        alert("Word Size must be at least 5 for nucleotide searches");
    }
    if ((getSelectedBlastProgram() == 'tblastn' || getSelectedBlastProgram() == 'blastp' ||
        getSelectedBlastProgram() == 'blastx') &&
    (document.<%=BLAST_FORM_NAME;?>.wordsize.options[3].selected ||
    document.<%=BLAST_FORM_NAME;?>.wordsize.options[4].selected ||

function setEstOtherDataset(blast_program) {
    var output = '';
    if (blast_program == 'blastp' || blast_program == 'blastx') {
        document.getElementById("other_checkbox").checked = false;
        document.getElementById("other_checkbox").disabled = true;
        document.getElementById("otherLabel").style.color = "Grey";
        document.getElementById("est_checkbox").checked = false;
        document.getElementById("est_checkbox").disabled = true;
        document.getElementById("est_standard").checked = false;
        document.getElementById("est_standard").disabled = true;
        document.getElementById("est_detailed").checked = false;
        document.getElementById("est_detailed").disabled = true;
        document.getElementById("estLabel").style.color = "Grey";
        output = false;
    } else {
        document.getElementById("other_checkbox").disabled = false;
        document.getElementById("otherLabel").style.color = '#<?=DV::$BLASTOCS["llongipalpis"][0];?>';
        document.getElementById("est_checkbox").disabled = false;
        document.getElementById("est_label").style.color = '#<?=DV::$BLASTOCS["llongipalpis"][0];?>';
        document.getElementById("est_standard").disabled = false;
    }
}

// disable the other and est datasets option for peptide searches
function setEstOtherDataset(blast_program) {
    var output = '';
    if (blast_program == 'blastp' || blast_program == 'blastx') {
        document.getElementById("other_checkbox").checked = false;
        document.getElementById("other_checkbox").disabled = true;
        document.getElementById("otherLabel").style.color = "Grey";
        document.getElementById("est_checkbox").checked = false;
        document.getElementById("est_checkbox").disabled = true;
        document.getElementById("est_standard").checked = false;
        document.getElementById("est_standard").disabled = true;
        document.getElementById("est_detailed").checked = false;
        document.getElementById("est_detailed").disabled = true;
        document.getElementById("estLabel").style.color = "Grey";
        output = false;
    } else {
        document.getElementById("other_checkbox").disabled = false;
        document.getElementById("otherLabel").style.color = '#<?=DV::$BLASTOCS["llongipalpis"][0];?>';
        document.getElementById("est_checkbox").disabled = false;
        document.getElementById("est_label").style.color = '#<?=DV::$BLASTOCS["llongipalpis"][0];?>';
        document.getElementById("est_standard").disabled = false;
    }
}
document.getElementById("est_detailed").disabled=false;

// include other and est datasets if select all is checked and a peptide search is not selected
if(document.getElementById("all_checkbox").checked==true) {
    document.getElementById("other_checkbox").checked=true;
    document.getElementById("est_checkbox").checked=true;
}

if(document.getElementById("est_detailed").checked==true){
    document.getElementById("other_checkbox").checked=false;
    document.getElementById("other_checkbox").disabled=true;
}

output=true;
return output;

// look at all datasets, if set is checked select all appropriate (nuc or pep) dbs for that set and blast program type combination
function selectDatasetDBs(blast_program){
    var datasets = document.getElementsByName("datasets[]");
    for(var i=0; i<datasets.length;i++) {
        var pepSet = document.getElementsByName("pepdbs["+datasets[i].value+"][]");
        var nucSet = document.getElementsByName("nucdbs["+datasets[i].value+"][]");
        var traceSet = document.getElementsByName("tracedbs["+datasets[i].value+"][]");
    }
}
//if it's checked then at least one db is selected
if (datasets[i].checked==true){
    if (blast_program == 'blastp' || blast_program == 'blastx'){
        //select pep dbs
        for(var j=0; j<pepSet.length; j++){
            pepSet[j].checked=true;
            pepSet[j].disabled=false;
        }
        //deselect and deactivate nuc dbs
        for(var j=0; j<nucSet.length; j++){
            nucSet[j].checked=false;
            nucSet[j].disabled=true;
        }
        //deselect and deactivate trace dbs
        for(var j=0; j<traceSet.length; j++){
            traceSet[j].checked=false;
            traceSet[j].disabled=true;
        }
    } else{
        //select nuc dbs
        for(var j=0; j<nucSet.length; j++){
            nucSet[j].checked=true;
            nucSet[j].disabled=false;
        }
        //activate trace dbs
        for(var j=0; j<traceSet.length; j++){
            traceSet[j].disabled=false;
        }
        //deselect and deactivate pep dbs
        for(var j=0; j<pepSet.length; j++){
pepSet[j].checked=false;
pepSet[j].disabled=true;
}
}
}else{
    //display correct databases but don’t check anything
    if (blast_program == 'blastp' || blast_program == 'blastx'){
        //select pep dbs
        for(var j=0; j<pepSet.length; j++){
            pepSet[j].disabled=false;
        }
        //deselect and deactivate nuc dbs
        for(var j=0; j<nucSet.length; j++){
            nucSet[j].disabled=true;
        }
        for(var j=0; j<traceSet.length; j++){
            traceSet[j].disabled=true;
        }
    }else{
        //select nuc dbs
        for(var j=0; j<nucSet.length; j++){
            nucSet[j].disabled=false;
        }
        for(var j=0; j<traceSet.length; j++){
            traceSet[j].disabled=false;
        }
        //deselect and deactivate pep dbs
        for(var j=0; j<pepSet.length; j++){
            pepSet[j].disabled=true;
        }
    }
}
```javascript
// these run when a blast program is selected
function blastProgramChecks()
{
    var blastProgram = getSelectedBlastProgram();

    // user has selected a detailed search that disabled all datasets. if peptide search program is selected we need to enable the datasets
    if (document.getElementById("est_detailed").checked == true && (blastProgram == 'blastp' || blastProgram == 'blastx')){
        document.getElementById("agambiae_checkbox").disabled = false;
        document.getElementById("aaegypti_checkbox").disabled = false;
        document.getElementById("iscapularis_checkbox").disabled = false;
        document.getElementById("cquinquefasciatus_checkbox").disabled = false;
        document.getElementById("phumanus_checkbox").disabled = false;
        document.getElementById("all_checkbox").disabled = false;
    }

    setWordSize(blastProgram);
    setLowComplexity(blastProgram);
    selectDatasetDBs(blastProgram);
    setEstOtherDataset(blastProgram);
    // wordSizeCheck(); wordsize is being auto set by function above. shouldn't have to check it here
}
```
function getSelectedBlastProgram()
{
    var blastPrograms = document.getElementsByName('program');
    for(var i=0; i<blastPrograms.length; i++){
        if (blastPrograms[i].checked==true){
            var selectedProgram = blastPrograms[i].value;
        }
    }
    return selectedProgram;
}

function uncheckAllDBs()
{
    var datasets = document.getElementsByName("datasets[]");
    for(var i=0; i<datasets.length; i++) {
        var pepSet = document.getElementsByName("pepdb["+datasets[i].value+"'][]");
        var nucSet = document.getElementsByName("nucdb["+datasets[i].value+"'][]");
        var traceSet = document.getElementsByName("tracedb["+datasets[i].value+"'][]");
        for(var j=0; j<nucSet.length; j++) {
            nucSet[j].checked=false;
        }
        for(var j=0; j<pepSet.length; j++) {
            pepSet[j].checked=false;
        }
        for(var j=0; j<traceSet.length; j++) {
            traceSet[j].checked=false;
        }
    }
}
function toggleAllDatasets() {
    var sets = document.getElementsByName("datasets[]");
    if(document.getElementById("all_checkbox").checked == true) {
        for(var i=0; i<sets.length; i++) {
            document.getElementsByName("datasets[]")[i].checked = true;
            if(document.getElementsByName("program")[4].checked == true ||
                document.getElementsByName("datasets[]")[i].value == 'other' ||
                document.getElementsByName("datasets[]")[i].value == 'est') {
                document.getElementsByName("datasets[]")[i].checked = false;
            }
        }
        // now select all the dbs for the set
        selectDatasetDBs(getSelectedBlastProgram());
    } else {
        for(var i=0; i<sets.length; i++) {
            document.getElementsByName("datasets[]")[i].checked = false;
        }
        uncheckAllDBs();
    }
}

function validateFormOnSubmit(theForm) {
    var reason = "";
    var result = false;
    var errorColor="#fe6969"
wordSizeCheck();

//reset the fasta background to white
//if(theForm.job_id.value="Retrieve BLAST Job ID" || theForm.job_id.value=""){
  // theForm.job_id.style.background = 'White';
  // theForm.input_sequence.style.background = errorColor;
//}

//user has entered data in job_id field and is trying to retrieve a job, doesn’t need a db or fasta sequence
if( theForm.job_id.value!="Retrieve BLAST Job ID" && theForm.job_id.value!=""){
  if (isNaN(theForm.job_id.value)) {
    theForm.job_id.style.background = errorColor;
    reason = "Job ID contains illegal characters\n"
  }else{
    theForm.job_id.style.background = "white";
  }
}

//user is submitting a job, check fasta syntax and ensure at least 1 db is selected
else{
  //check fasta
  var numberOfQueries = 0;
  if(theForm.input_sequence.value.match(/>/gm) != null ){
    numberOfQueries = theForm.input_sequence.value.match(/>/gm).length;
  }
  //no query
  if(theForm.input_sequence.value=="") {
    reason += "<li>Enter a query sequence</li>";
    theForm.input_sequence.style.background = errorColor;
  }
}
// too many queries
else if (numberOfQueries > 20 ) {
    reason += "<li>The maximum number of query sequences is twenty.</li>";
    theForm.input_sequence.style.background = errorColor;
}

// this is the case for a user entering a single query sequence with no >sequence name
else if ( theForm.input_sequence.value.match(/([ARNDCEQGHILKMFPSWYVN] \-)+)/gim)!==null &&
numberOfQueries==0 ) {
    theForm.input_sequence.style.background = 'White';
    theForm.input_sequence.value = ">Untitled Query Sequence\n" +
    theForm.input_sequence.value;
}

// bad fasta sequence
else if ( theForm.input_sequence.value.match(/^[\n\r]*(?:[^\n\r]*[^\n\r]+[ARNDCEQGHILKMFPSWYVN][^\n\r]+)+$/i) ==
null ) {
    reason += "<li>Input query must be in FASTA format</li>";
    theForm.input_sequence.style.background = errorColor;
}

// check for at least one database selected
var total=0;
var datasets = document.getElementsByName("datasets[]");
for(var i=0; i<datasets.length; i++) {
    var pepSet = document.getElementsByName("pepdb\['"+datasets[i].value+"'][]");
    var nucSet = document.getElementsByName("nucdb\['"+datasets[i].value+"'][]");
    var traceSet = document.getElementsByName("trace\['"+datasets[i].value+"'][]");

    for(var j=0; j<pepSet.length; j++){
        if(pepSet[j].checked==true){
            total++;
        }
    }

    for(var j=0; j<nucSet.length; j++){
        if(nucSet[j].checked==true){
            total++;
        }
    }

    for(var j=0; j<traceSet.length; j++){
        if(traceSet[j].checked==true){
            total++;
        }
    }

```javascript
for (var j=0; j<nucSet.length; j++){
    if(nucSet[j].checked==true){
        total++;
    }
}
for (var j=0; j<traceSet.length; j++){
    if(traceSet[j].checked==true){
        total++;
    }
}
//separate check for est dbs
if(document.getElementById('est_checkbox').checked == true){
    total++;
}
var ds1 = document.getElementById("datasetsDiv");
if(total==0){
    ds1.style.borderColor = errorColor;
    ds1.style.borderWidth = 'medium';
    reason += "<li>Select at least one database</li>";
} else{
    ds1.style.borderColor = <div color>
'DV::$BLASTOC[$_SESSION["organism_id"]][5].""';?>
    ds1.style.borderWidth = 'thin';
}
//now check to see if there were errors
if(reason!=""){
    document.getElementById("errorBox").innerHTML='<ul class="errorList">' + reason +
```
function submitBLAST()
{
    //pinwheel cursor
    document.getElementById('blastForm').style.cursor = 'progress';
    //disable button
    document.getElementById('Web:Misc_Help#submit').disabled = true;
    document.getElementById('Web:Misc_Help#submit').value = 'Running';

    // is user looking up old job?
    if (document.getElementsByTagName('input').job_id.value != 'Retrieve BLAST Job ID' &&
        document.getElementsByTagName('input').job_id.value != ''){
        jobId = document.getElementsByTagName('input').job_id.value;
        document.getElementById('jobActivity').innerHTML = '<div class="jobBox" style="clear:both;">Retrieving Job '+jobId+' Information</div>';  
        makeRequest('/sections/Tools/tool_includes/BLAST/processing.php', '?job_id='+jobId, 'GET');
    } else { //user is submitting a new job
        //update the status/results div
        return true;
    }
}
document.getElementById('jobActivity').innerHTML = '<div class="jobBox" style="clear:both; width:98%;">
<h3>Submitting Job</h3>
</div>';

//clear the results link so users aren't confused
document.getElementById('jobResultsLink').innerHTML = '';

//start submit process
makeRequest('/sections/Tools/tool_includes/BLAST/submit.php', getSubmitParams(), 'POST');

//build submit string
function getSubmitParams() {
    var getstr = ''; //"?";
    var j=0;
    var programs = document.getElementsByTagName("program");
    for(var i=0;i<programs.length;i++) {
        if(programs[i].checked==true){
            getstr +="program=" + programs[i].value + ";";
        }
    }

    var k=0;
    var datasets = document.getElementsByTagName("datasets[]");
    for(var i=0; i<datasets.length; i++) {
        var pepSet = document.getElementsByTagName("pepdb[" + datasets[i].value + "][" + i + "]");
        var nucSet = document.getElementsByTagName("nucdb[" + datasets[i].value + "][" + i + "]");
        var traceSet = document.getElementsByTagName("tracedb[" + datasets[i].value + "][" + i + "]");

        for(var j=0; j<pepSet.length; j++){
            if(pepSet[j].checked==true){

            }
getstr += "blastdbs[" + k + "]=" + pepSet[j].value + ";
" + k++;
 }

for(var j=0; j<nucSet.length; j++){
  if(nucSet[j].checked == true){
    getstr += "blastdbs[" + k + "]=" + nucSet[j].value + ";
" + k++;
  }
}

for(var j=0; j<traceSet.length; j++){
  if(traceSet[j].checked == true){
    getstr += "blastdbs[" + k + "]=" + traceSet[j].value + ";
" + k++;
  }
}

// special vars for est searches
if(document.getElementById('est_checkbox').checked == true &&
  document.getElementById('est_detailed').checked == true) {
  getstr += "detailed_est=true&";
}
else if(document.getElementById('est_checkbox').checked == true &&
  document.getElementById('est_standard').checked == true) {
  getstr += "standard_est=true&";
}

// new lines are getting dropped from the input sequence, we need to fix that.
string as placeholder that submit.php will use to reinsert new lines

```javascript
var sequence = document.getElementsByName("input_sequence")[0].value;
var newseq = sequence.replace(new RegExp('\n',"g"), 'Z-z-Z');

getstr += "value=" + document.getElementsByName("evalue")[0].value + ";
getstr += "wordsize=" + document.getElementsByName("wordsize")[0].value + ";
getstr += "scoringmatrix=" + document.getElementsByName("scoringmatrix")[0].value + ";
getstr += "dust=" + document.getElementsByName("dust")[0].value + ";
getstr += "num_results=" + document.getElementsByName("num_results")[0].value + ";
getstr += "input_sequence=" + newseq;

return getstr;
```

```javascript
function makeRequest(url, params, openMethod) {
  //openMethod is POST, GET, or HEAD
  if (window.XMLHttpRequest) {
    // Mozilla, Safari, IE7+...
    http_request = new XMLHttpRequest();
  } else if (window.ActiveXObject) {
    try{
      http_request = new ActiveXObject("Msxml2.XMLHTTP");
    } catch(e) {
      http_request = new ActiveXObject("Microsoft.XMLHTTP");
    }
  }

  if (!http_request) {
    alert('Cannot create XMLHTTP instance, upgrade to a newer browser.');
    return false;
  }
```
http_request.onreadystatechange = function () {
  if (http_request.readyState == 4) {
    if (http_request.status == 200) {
      response = http_request.responseText;
      analyzeResponse(response);
    } else {
      // alert('Unexpected response from server:
      // ' + http_request.responseText + '\nreadyState: ' + http_request.readyState + '\nstatus: ' + http_request.status);
      // unpinwheel cursor
      document.getElementById('blastForm').style.cursor = 'default';
      // enable button
      document.getElementById('Web:Misc_Help#submit').disabled = false;
      document.getElementById('Web:Misc_Help#submit').value = 'BLAST!';
      // update the status/results div
      document.getElementById('jobActivity').innerHTML = '';
      // clear the results link so users aren't confused
      document.getElementById('jobResultsLink').innerHTML = '';
    }
  }
};

if (openMethod == 'GET') {
  http_request.open(openMethod, url + params, true);
  http_request.send(null);
} else {
  http_request.open(openMethod, url + '?'+ params, true);
  http_request.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
  http_request.setRequestHeader("Content-length", params.length);
  http_request.setRequestHeader("Connection", "close");
  http_request.send(params);
function returnOneSubstring(regex, input) {
    regex.exec(input);
    return RegExp.$1;
}

function analyzeResponse(response) {
    //if only numbers are returned it's the job id
    //alert(response);
    if (!isNaN(response)) {
        jobId = response;
        document.getElementById('jobActivity').innerHTML = '<div class="jobBox" style="clear:both;">
<h3>Retrieving Job '+jobId+' Information</h3></div>';
        makeRequest('/sections/Tools/tool_includes/BLAST/processing.php', '?job_id='+jobId, 'GET');
    } else if (response.match(/Retrieving results for job/gm)) {
        document.getElementById('jobActivity').innerHTML = '<div class="jobBox" style="clear:both;">
<h3>Retrieving results for job '+jobId+'</h3></div>';
    }
    //results are done, processing them and storing in the db...
    else if (response.match(/Retrieving results for job/gm)) {
        document.getElementById('jobActivity').innerHTML = '<div class="jobBox" style="clear:both;">
<h3>Retrieving results for job '+jobId+'</h3></div>';
    }
    //parse out all the job's input params. ps substring matching in js is ridiculous
    var program = returnOneSubstring(/program=(\w+)/m, response);
    var wordsize = returnOneSubstring(/wordsize=(\d+)/m, response);
    var numhits_oneline = returnOneSubstring(/numhits_oneline=(\d+)/m, response);
    var numhits_align = returnOneSubstring(/numhits_align=(\d+)/m, response);
    var evalue = returnOneSubstring(/evalue=(\[-\d\.]+)/m, response);
    var scoringmatrix = returnOneSubstring(/scoringmatrix=(\w+)/m, response);
var masking = returnOneSubstring(/masking=\(w\)/m,response);
var sequence = returnOneSubstring(/sequence=((>.*\n[ARNDCEQGHILKMFPSWTYV]\n\n)*+\n)/m,response);
var detailed_est = returnOneSubstring(/detailed_est=(\w+)###/m,response);
var program_desc = returnOneSubstring(/programDesc=(.*###)/m,response);
//var estdbs = returnOneSubstring(/estdbs=(\w+)/m,response);

//set correct blast description
document.getElementById('blastDescription').innerHTML=program + ' - ' + program_desc;

// uncheck all databases
uncheckAllDBs();
// uncheck all datasets
var datasets = document.getElementsByName("datasets[]");
for(var i=0; i<datasets.length; i++) {
    document.getElementsByName("datasets[]")[i].disabled=false;
    document.getElementsByName("datasets[]")[i].checked=false;
}
document.getElementById("all_checkbox").disabled=false;
document.getElementById("all_checkbox").checked=false;

if(detailed_est=='true'){
    //check est dataset and select est_fancy radio button
    document.getElementById('est_detailed').checked=true;
    document.getElementById('est_checkbox').checked=true;
    //disabled everything else
    datasetDisplayToggle('est');
} else {
    var temp=response.match(/target_database=(\d+)###/gm);
}
if(temp){
    re=/target_database=(\d+)###/m
    for(i=0; i<temp.length; i++){
        re.exec(temp[i]);
        document.getElementById(RegExp.$1).checked=true;
    }
}

temp=response.match(/target_dataset=(\w+)###/gm);
if(temp){
    re=/target_dataset=(\w+)###/m
    for(i=0; i<temp.length; i++){
        re.exec(temp[i]);
        document.getElementById(RegExp.$1 + '_checkbox').checked=true;
    }
}

if( response.match(/estdbs=true/gm) ){
    //traditional est search was selected, check est box and est radio button
    document.getElementById('est_checkbox').checked=true;
    document.getElementById('est_standard').checked=true;
}

//update all the parameters to match the job being retrieved
document.getElementById('Web:Misc_Help#paste').value=sequence;
document.getElementById(program).checked=true;
document.getElementById('Web:BLAST#maximum_e-value').value=evalue;
document.getElementById('Web:BLAST#wordlength').value=wordsize;
document.getElementById('Web:BLAST#scoring_matrix').value=scoringmatrix;
document.getElementById('Web:BLAST#masking').value=masking;
//document.getElementById('numhits_oneline').value=numhits_oneline;
document.getElementById('Web:BLAST#number_of_results').value=numhits_align;

selectDatasetDBs(getSelectedBlastProgram());
setEstOtherDataset(getSelectedBlastProgram());

if(detailed_est=='true'){
    makeRequest('/:sections/Tools/tool_includes/BLAST/estResults.php', '?job_id='+jobId,'GET');
} else {
    makeRequest('/:sections/Tools/tool_includes/BLAST/results.php', '?job_id='+jobId,'GET');
}

//set this global variable
detailEST=detailed_est;

//we've been forwarded to the results page, all done here
else if(response.match(/Results/gm) && !response.match(/Retrieving results/gim)){
    if(detailedEST=='true'){
        document.getElementById('jobResultsLink').innerHTML = '<div style="float:right;padding-right:15px;"><a href="/Tools/BLAST/?estResult&job_id='+jobId+'" target="_blank">View results in a new window</a></div>';
    } else{
        document.getElementById('jobResultsLink').innerHTML = '<div style="float:right;padding-right:15px;"><a href="/Tools/BLAST/?job_id='+jobId+'" target="_blank">View results in a new window</a></div>';
    }
    document.getElementById('jobActivity').innerHTML = response;
//unpinwheel cursor
document.getElementById('blastForm').style.cursor='default';

//enable button
document.getElementById('Web:Misc_Help#submit').disabled=false;
document.getElementById('Web:Misc_Help#submit').value="BLAST!";

//reset detailEST
detailEST=false;

}

//ugh oh, errors!
else if (response.match(/error/gim) && !response.match(/Retrieving results/gim) ) {
    document.getElementById('jobActivity').innerHTML = response;

    //unpinwheel cursor
document.getElementById('blastForm').style.cursor='default';

    //enable button
document.getElementById('Web:Misc_Help#submit').disabled=false;
document.getElementById('Web:Misc_Help#submit').value="BLAST!";
}

//no results yet, keep waiting...
else {
    document.getElementById('jobActivity').innerHTML = response;
    makeRequest('/sections/Tools/tool_includes/BLAST/processing.php', '?job_id='+jobId,
'GET');
}

function checkEsts(){
    // est box needs to be checked when either of its radio buttons are clicked
document.getElementById('est_checkbox').checked=true;

//set some defaults for the detailed search
if(document.getElementById("est_detailed").checked){
    document.<?=$BLAST_FORM_NAME;?>.dust.selectedIndex=1;
    document.<?=$BLAST_FORM_NAME;?>.evalue.selectedIndex=3;
}

var activatedDataset='';

function datasetDisplayToggle(organism){
    var dataset = document.getElementById((organism + '_databases'));
    document.getElementById('jobDatasetDescriptionHelp').innerHTML='';

    if(activatedDataset){ var datasetToDeactivate = document.getElementById((activatedDataset + '_databases')); }

    if (dataset.style.display=='none'){
        dataset.style.display='block';
        if(activatedDataset){ datasetToDeactivate.style.display='none'; }  
        activatedDataset=organism;
    }

    //special case for ests
    if(organism=='est'){
        var sets = document.getElementsByName("datasets[]");

        //est box has been unchecked, activate all datasets
        if(document.getElementById("est_checkbox").checked==false){
            var sets = document.getElementsByName("datasets[]");
        }
    }
for(var i=0; i<sets.length; i++) {
    document.getElementsByName("datasets[]")[i].disabled=false;
}
document.getElementById('all_checkbox').disabled=false;
document.getElementById('est_standard').checked=true;
}
else if(document.getElementById('est_detailed').checked==true){
    //uncheck and deactivate all other datasets and select all
    for(var i=0; i<sets.length; i++) {
        if(document.getElementsByName("datasets[]")[i].value!="est") {
            document.getElementsByName("datasets[]")[i].checked=false;
            document.getElementsByName("datasets[]")[i].disabled=true;
            //needs to uncheck all dbs too
            var nucSet =
            document.getElementsByName("nucdbs['"+document.getElementsByName("datasets[]")[i].value+"'][]["");
            for(var j=0; j < nucSet.length; j++) {
                nucSet[j].checked = false;
                }
            }
            var traceSet =
            document.getElementsByName("tracedbs['"+document.getElementsByName("datasets[]")[i].value+"'][]["");
            for(var j=0; j < traceSet.length; j++) {
                traceSet[j].checked = false;
                }
    }
    document.getElementById('all_checkbox').checked=false;
    document.getElementById('all_checkbox').disabled=true;
}else{
    //activate all datasets
    for(var i=0; i<sets.length; i++) {

document.getElementsByName("datasets[]")[i].disabled=false;
}
document.getElementById('all_checkbox').disabled=false;
}
//if est box is checked and neither search option is checked, default to checking
standard so the search doesn't bomb
if(document.getElementById("est_standard").checked==false &
document.getElementById("est_detailed").checked==false){

document.getElementById("est_standard").checked=true;
}
}
}

function datasetCheckToggle(organism){
    var pepSet = document.getElementsByName("pepdbs['"+organism+"'][]");
    var nucSet = document.getElementsByName("nucdbs['"+organism+"'][]");
    var traceSet = document.getElementsByName("tracedbs['"+organism+"'][]");

    if(document.getElementById(organism+_checkbox').checked==true){
//enable all the dbs for this organism
        for(var i=0; i < nucSet.length; i++) {
            if(nucSet[i].disabled == false){
                nucSet[i].checked = true;
            }
        }
        for(var i=0; i < pepSet.length; i++) {
            if(pepSet[i].disabled == false){
                pepSet[i].checked = true;
            }
        }
    }
else {
    //disable all dbs for organism
    for (var i=0; i < nucSet.length; i++) {
        nucSet[i].checked = false;
    }
    for (var i=0; i < traceSet.length; i++) {
        traceSet[i].checked = false;
    }
    for (var i=0; i < pepSet.length; i++) {
        pepSet[i].checked = false;
    }
}

function databaseChecks(organism){
    var pepSet = document.getElementsByTagName("pepdbs["+organism+"][]");
    var nucSet = document.getElementsByTagName("nucdbs["+organism+"][]");
    var traceSet = document.getElementsByTagName("tracedbs["+organism+"][]");
    var atLeastOne = 0;
    for (var i=0; i < nucSet.length; i++) {
        if(nucSet[i].checked == true ) atLeastOne = 1;
    }
    for (var i=0; i < pepSet.length; i++) {
        if(pepSet[i].checked == true ) atLeastOne = 1;
    }
    for (var i=0; i < traceSet.length; i++) {

if (traceSet[i].checked == true) atLeastOne = 1;

}
if (document.getElementById(organism+'_'checkbox').checked==true){
  // the dataset is checked, if no dbs are checked then uncheck dataset
  if (atLeastOne==0) { document.getElementById(organism+'_'checkbox').checked=false; } 
  } else {
  // dataset is unchecked, if there is at least one db selected then check the dataset
  if (atLeastOne==1) { document.getElementById(organism+'_'checkbox').checked=true; } 
}

</script>

<div id="errorBox">
<?
if ($_GET['e5'])
print $blast_errors[5];
if ($_GET['e7'])
print $blast_errors[7];
if ($_GET['e8'])
print $blast_errors[8];
if ($_GET['e9'])
print $blast_errors[9];
if ($_GET['e12'])
print $blast_errors[12];
if ($_GET['e14'])
print $blast_errors[14];
<form method="post" enctype="multipart/form-data" action="javascript:submitBLAST();" id="blastForm" name="<?=$_SERVER['REQUEST_URI'];?>" onsubmit="return validateFormOnSubmit(this)"
818 <div id="ieAlert" style="display:none; margin-top:4px; margin-left:30px; width:90%; border:1px solid #888a00; background-color:#c3c517; font:bold 10px Sans-serif; padding:4px 8px;">
819  IE 7 Users: There is a limitation with IE 7 that prevents successful submission of BLAST jobs. Please consider upgrading to IE 8 or using Firefox.
820 </div>
821 <div id="issueAlert" style="display:none; margin-top:4px; margin-left:30px; width:90%; border:1px solid #888a00; background-color:#c3c517; font:bold 10px Sans-serif; padding:4px 8px;">
822  BLAST Users: There is currently an issue with BLAST jobs, VectorBase staff are aware of the issue and are working to correct the problem. If your job takes more then ten minutes to complete please try refreshing the page. We appreciate your patience and patronage.
823 </div>
824 <!-- wtf is this??
825 <div id="blastTimer" style="display:none; margin-top:4px; margin-left:30px; width:90%; border:1px solid #888a00; background-color:#c3c517; font:bold 10px Sans-serif; padding:4px 8px;">
826  Your query has been running for <script language="JavaScript">QueryTimer()</script></div>
827 -->
828 <div class="jobBox" style="width:98%;">
829  <h1>FASTA Query Sequence</h1>
830  <textarea class="hashelp" id="Web:Misc_Help#paste" name="input_sequence" style="font-family:Sans-serif; font-size:11px; margin:8px; width:97%; height:10em;" onfocus="if(this.value=='')Example Sequence\nCGGTCTATTGTGTTTGGGATCGAATCCATGACGACATGATGTTAAAGTCGTA')this.value=';"(">?
if($_GET['blast_id']) echo $redoSequence;
else echo "Example Sequence\nCGGTCTATTTGGGGATCGAACCCATGACGAGCATGATGTTAAGTCGTA";

if($_GET['blast_id']){
    if($program == $redoProgram) echo " checked ";
} else{
    if($program == 'blastn') echo " checked ";
}

echo $program;
```php
866     echo '</label></td>').'\n";
867     }?
868     </tr>
869     </table>
870     <span id="blastDescription" style="margin:4px 0px; text-align:center;">blastn -
871     </span>
872     <tr>
873     <div id="datasetsDiv" class="jobBox" style="margin-left:6px; width:48em;">" datasets
874     <h1>Datasets</h1>
875     <table style="width:100%;">
876     <tr><td style="width:16em; vertical-align:top;">
877     <div id="datasetsDiv" class="jobBox" style="margin-left:6px; width:48em;">" datasets
878     foreach (ORG::$ENDOWED as $organism){
879     if ($organism != "all" && $organism != "other"){
880     echo '<div class="jobDataset" onclick="datasetDisplayToggle('.$organism.');" style="border-color: #'.$DV::$BLASTOCS[$organism][4].'; background-color: #'.$DV::$BLASTOCS[$organism][5].';">'">
881     echo '<input type="checkbox" id="'.$organism.'_checkbox" name="datasets[]" value="'.$organism.'" style="vertical-align:middle;">'
882     if ($_SESSION["organism_id"] == $organism) echo ' checked';
883     echo ' onclick="datasetCheckToggle('.$organism.');"="/"."\n";
884     echo '<label for="'.$organism.'._checkbox" style="vertical-align:middle; color: #'.$DV::$BLASTOCS[$organism][0].';">"</i>'.ORG::$FN[$organism].'</i></label>).'\n";
885     echo '</div').'\n";
886     }
887     }?
888     <!-- other organisms -->
889     <div class="jobDataset" onclick="datasetDisplayToggle('other');" style="border-color: #'.$DV::$BLASTOCS["llongipalpis"][4];" background-color: #'.$DV::$BLASTOCS["llongipalpis"][5];">"
```
<input type="checkbox" id="other_checkbox" name="datasets[]" value="other" style="vertical-align:middle;" onclick="datasetCheckToggle('other');" />
<label id="otherLabel" for="other_checkbox" style="vertical-align:middle; color: #<?=DV::$BLASTOCS["llongipalpis"]?[0];?>;">Other Organisms</label>

<!-- ESTs -->
<div class="jobDataset" onclick="datasetDisplayToggle('est');" style="border-color: #<?=DV::$BLASTOCS["llongipalpis"]?[4];?>; background-color: #<?=DV::$BLASTOCS["llongipalpis"]?[5];?>;"
     style="vertical-align:middle;" onclick="datasetCheckToggle('est');"/>
<input type="checkbox" id="est_checkbox" name="datasets[]" value="est" style="vertical-align:middle;" onclick="datasetCheckToggle('est');" />
<label id="estLabel" for="est_checkbox" style="vertical-align:middle; color: #<?=DV::$BLASTOCS["llongipalpis"]?[0];?>;">ESTs</label>

<!-- all organisms -->
<div class="jobDataset" style="border-color: #<?=DV::$BLASTOCS["all"]?[4];?>; background-color: #<?=DV::$BLASTOCS["all"]?[5];?>;"
     style="vertical-align:middle;" onclick="toggleAllDatasets();"/>
<input type="checkbox" id="all_checkbox" name="blastAll" onclick="toggleAllDatasets();" style="vertical-align:middle;" />
<label id="allLabel" for="all_checkbox" style="vertical-align:middle; color: #<?=DV::$BLASTOCS["all"]?[0];?>;">Select all</label>

<!-- description column-->
<td style="vertical-align:top;">
<div id="jobDatasetDescription" style="font:normal 12px Sans-serif">
<span id="jobDatasetDescriptionHelp" style="font-family:Sans-serif">Click on an organism to view its databases</span>
</div>
$i=0;
// lists all dbs for each organism. these will be displayed one at a time, when that organism has been clicked
foreach (ORG::$ENDOWED as $organism){
    if ($organism != "all" && $organism != "other"){
        foreach (ORG::$ENDOWED as $organism){
            if ($organism != "all" && $organism != "other"){
                //nuc dbs
                echo '<div id="'. $organism ."_databases" style="display:none; width:100%; border-radius: 12px; background-color: #'. DV::$BLASTOCS[$organism][5].'; border: 1px solid #'. DV::$BLASTOCS[$organism][4].'">';
                echo '<fieldset style="margin-bottom:6px">';
                echo '<legend style="color: #'.DV::$BLASTOCS[$organism][0].'; margin: 0px; padding: 2px 2px 2px 8px;">Nucleotide</legend>';  
                $sql = "select b.display_id, b.blastdb_id, b.file_name, b.sequence_type from blast_databases b, organism o where b.organism_id=o.organism_id and o.short_name='" . $organism . "' and b.residue_type='NUCLEOTIDE' order by b.order_by";
                $dbResult = pg_query(DB::getUI_DB(), $sql);
                while ($row = pg_fetch_assoc($dbResult)){
                    if($row["sequence_type"]=='Trace Read')
                        echo '<li class="altLi"><input type="checkbox" id="'. $row["blastdb_id"]." name="tracedbs["'. $organism ."][" value="'. $row["file_name"]." style="vertical-align:middle;"/>
                        <label style="font-family:Sans-serif;vertical-align:middle; color: #'.DV::$BLASTOCS[$organism][0].'; font-size:10px;" for="'. $row["blastdb_id"] ."">'. $row["display_id"] ."</label></li>";
                    else
                        echo '<li class="altLi"><input type="checkbox" id="'. $row["blastdb_id"]." name="nucdbs["'. $organism ."][" value="'. $row["file_name"]." style="vertical-align:middle;"/>
                        <label style="font-family:Sans-serif;vertical-align:middle; color:";

```php
#'.DV::$BLASTOCS[$organism][0].' font-size:10px;"
for="'.$.row["blastdb_id"][""'."'.$.row["display_id"]["</label></li>'."n"

933     }
934     echo " </ul>");
935     echo "</fieldset>
936     //pep dbs
937     echo '<fieldset';
938     echo ' <legend style="color: #'.DV::$BLASTOCS[$organism][0].' margin: 0px; padding: 2px 2px 2px 8px;">Peptide</legend>';
939     echo ' <ul';
940     $sql = "select b.display_id, b.blastdb_id, b.file_name from blast_databases b, organism o where b.organism_id=o.organism_id and o.short_name='"'.$.organism."' and b.residue_type='PEPTIDE' order by b.order_by";
941     $dbResult = pg_query(DB::getUI_DB(),$sql);
942     while ( $row = pg_fetch_assoc($dbResult)){ // setting the peptide dbs to disabled initially since the default blast program is blastn
943         echo ' <li class="altLi"><input type="checkbox" id="'.$.row["blastdb_id"][""'."'.$.row["display_id"][""'."'.$.row["file_name"][""'."'.$.organism."'] onclick="databaseChecks("'.$.organism."' value="'.$.row["file_name"][""'."'.$.organism."'); style="vertical-align:middle;" disabled="disabled" /><label style="vertical-align:middle; color: #'.DV::$BLASTOCS[$organism][0].' font-size:10px;"
for="'.$.row["blastdb_id"][""'."'.$.row["display_id"]["</label></li>'."n"

945             }
946             echo " </ul>");
947             echo " </fieldset>
948             echo "</div>
949             $i++;
950         }
951     }
952     }
953     //other datasets
```
echo '<div id="other_databases" style="display:none; width:100%; border-radius: 12px; background-color: #.DV::$BLASTOCS["llongipalpis"][5].'; border: 1px solid '#.DV::$BLASTOCS["llongipalpis"][4].'; color: #.DV::$BLASTOCS["llongipalpis"][0].';";'>

//est

//trace reads

$ssql = "select blastdb_id, description, file_name from blast_databases where sequence_type='EST' and organism_id!=9 and organism_id>7 order by order_by";
$dbResult = pg_query(DB::getUI_DB(),$ssql);
while ($row = pg_fetch_assoc($dbResult))
{
  echo '<li class="altLi"><input type="checkbox" id="'.$row['blastdb_id'].'" name="nucdbs["other"]["] onclick="databaseChecks("other")" value="'.$row['file_name'].'" /><label for="'.$row['blastdb_id'].'" style="color: #.DV::$BLASTOCS["llongipalpis"][0].';font-size:10px;"">'.$row['description'].'" /></label></li>"."n";
}

$dbResult = pg_query(DB::getUI_DB(),$ssql);
while ($row = pg_fetch_assoc($dbResult))
{
  echo '<li class="altLi"><input type="checkbox" id="'.$row['blastdb_id'].'" name="tracedbs["other"]["] onclick="databaseChecks("other")" value="'.$row['file_name'].'" /><label for="'.$row['blastdb_id'].'" style="color: #.DV::$BLASTOCS["llongipalpis"][0].';font-size:10px;"">'.$row['description'].'" /></label></li>"."n";
}

$dbResult = pg_query(DB::getUI_DB(),$ssql);
while ($row = pg_fetch_assoc($dbResult))
{
  echo '<li class="altLi"><input type="checkbox" id="'.$row['blastdb_id'].'" name="tracedbs["other"]["] onclick="databaseChecks("other")" value="'.$row['file_name'].'" /><label for="'.$row['blastdb_id'].'" style="color: #.DV::$BLASTOCS["llongipalpis"][0].';font-size:10px;"">'.$row['description'].'" /></label></li>"."n";
}
size:10px;">'. $row["description"] .'</label></li>'. 

973 }  
974  
975 //mRNAseq  
976 echo ' <legend style="color: #'.DV::$BLASTOCS["llongipalpis"][0].'; margin: 0px; padding: 2px; font-size: 8pt; padding-left:8px; padding-top:10px;">mRNAseq</legend>';  
977 echo '<ul>';  
978 $sql = "select blastdb_id, description, file_name from blast_databases where sequence_type='mRNAseq' order by order_by";  
979 $dbResult = pg_query(DB::getUI_DB(),$sql);  
980 while ($row = pg_fetch_assoc($dbResult)){  
981 echo '<li class="altLi"><input type="checkbox" id="'. $row['blastdb_id'] .'" name="nucdbs[other][]" onclick="databaseChecks('other');" value="'. $row['file_name'] .'" /><label for="'. $row['blastdb_id'] .'" style="color: #'.DV::$BLASTOCS["llongipalpis"][0].';font-size:10px;">'. $row['description'] .'</label></li>'. 
982 }  
983  
984 echo "</ul></fieldset></div>";  
985  
986 //EST datasets  
987 echo '<div id="est_databases" style="display:none; width:100%; border-radius: 12px; background-color: #'.DV::$BLASTOCS["llongipalpis"][5].'; border: 1px solid #'.DV::$BLASTOCS["llongipalpis"][4].'; color: #'.DV::$BLASTOCS["llongipalpis"][0].';">';  
988 echo '<fieldset>';  
989 echo ' <legend style="color: #'.DV::$BLASTOCS["llongipalpis"][0].'; margin: 0px; padding: 2px 2px 8px;">Nucleotide</legend>';  
990 echo ' <ul>';  
991  
992 echo '<li class="altLi"><input type="radio" id="est_standard" name="est_results" onclick="checkEsts(); datasetDisplayToggle(\'est\');" checked/>'<label for="est_standard" style="color: #'.DV::$BLASTOCS["llongipalpis"][0].';">All Libraries</span>Search complete species EST datasets (One
<li class="altLi"><input type="radio" id="est_detailed" name="est_results" onclick="checkEsts();datasetDisplayToggle('"est\');" />
<label for="est_detailed" style="color: #DV::$BLASTOCS["llongipalpis"][0]."">Individual Libraries</label></li>

We suggest decreasing the Maximum E-Value for more significant results</li>

?&gt;
</div></td><div class="jobBox" style="width:97%;">
<h1>Options</h1>
<ul>
<li class="altLabel" for="evalue">Maximum E-Value</li>
<select class="jobSmall hashelp" name="evalue" id="Web:BLAST#maximum_e-value">
<? $evalsArray=array("10","1","0.1","1E-3","1E-5","1E-10","1E-20","1E-40","1E-80");
foreach($evalsArray as $option){
    echo "<option value="/" . $option . ""/>
    if($_GET['blast_id']){
if($option == $redoEvalue)
    echo " selected="selected"";
}elseif ($option="1") echo " selected="selected"";
echo ">". $option . "</option>

<?
</select>
</li>

<label class="altLabel" for="wordsize">Word Size</label>
<select class="jobSmall hashelp" name="wordsize" id="Web:BLAST#wordlength"
onchange="wordSizeCheck()">
<?
$sizes = array(3, 5, 7, 8, 11, 15, 30, 60);
foreach($sizes as $size){
echo "<option value="" . $size . "";
    if($_GET['blast_id'] && $size == $redoWordsize) echo " selected="selected"";
elseif($size==11) echo " selected="selected"";
    echo ">". $size . "</option>";}
?>
</select>

<label class="altLabel" for="scoringmatrix">Scoring Matrix</label>
<select class="jobSmall hashelp" name="scoringmatrix" id="Web:BLAST#scoring_matrix">
<?
$sizes = array("BLOSUM45", "BLOSUM62", "BLOSUM80", "PAM30", "PAM70");
foreach($sizes as $size){
echo "<option value="" . $size . "";
    if($_GET['blast_id'] && $size == $redoScoringmatrix) echo " selected="selected"";
elseif($size=="BLOSUM62") echo " selected="selected"";
    echo ">". $size . "</option>";}
?>
</select>
Complexity Masking

Low

Off

Number of Results

$\text{sizes = array(1, 5, 10, 50, 100, 250, 500); ~}$
$\text{$\text{numhitset = 0; ~}$}$
$\text{foreach($\text{sizes as $size){ ~}$}$
$\text{echo "<option value="" . $size . ""; ~}$}$
$\text{if($_GET[\text{\text{'blast_id'}]} & $\text{redoNumhits_align}{ ~}$}$
$\text{echo " selected=""; ~}$}$
$\text{elseif($\text{size==50 & $\text{numhitset ! = 1) echo " selected=""; ~}$}$}
$\text{echo "}" . $\text{size . "}"</option>"; ~}$}
$\text{?></select> ~}$}
$\text{</li> ~}$}
$\text{</ul> ~}$}
$\text{</div> ~}$}
$\text{<div class=\text{\text{jobBox} style=\text{\text{float:left; clear:both; width:97%;}}} ~}$}
$\text{<h1>Job Control</h1> ~}$}
$\text{<input type=\text{\text{text} size=18 name=\text{\text{job_id} class=\text{\text{jobSmall} style=\text{\text{float:left; margin:8px;}}} ~}$}
value="<? if($_GET['jobId']) echo $_GET['jobId']; else echo 'Retrieve BLAST Job ID';?>"
<onfocus="if(this.value=='Retrieve BLAST Job ID')this.value='';" />
<input type="submit" value="BLAST!" class="jobSmall" id="Web:Misc_Help#submit"
style="float:right; margin:8px;"/>
</div>
</td></tr></table>
<form>
<div id="jobResultsLink" style="font: normal 12px Sans-serif;"></div>
<div id="jobActivity" style="float:left; clear:both;width:100%;"></div>
<div id="spacer" style="float:left; clear:both; margin-top:8px;"></div>
<script>
//now that page has loaded, if we are under an organism page, make sure dbs are selected for the
//species dataset, also make that tab viewable
? if ($_SESSION["organism_id"]){
  echo 'datasetCheckToggle('.'$_SESSION["organism_id"]'.');';
  echo 'datasetDisplayToggle('.'$_SESSION["organism_id"]'.');';
}
?
</script>
C.2 submit.php

```php
<?php
SERVER_vars = $SERVER["DOCUMENT_ROOT"] . "/includes/index_header.php";
include ($SERVER_vars);
include_once("Jobs.php");
include("blast_errors.php");

// debug testing for Input.php
/*
  echo "error"
  var_dump($_POST);
  exit();
*/
// submitting a new job
if($_POST["job_id"]=='Retrive BLAST Job ID' || $_POST["job_id"]==''){

  // input.php replaced \n with Z-z-Z string because otherwise line breaks are lost when transferring sequence from input.php to submit.php with ajax,
  $input_sequence_data = preg_replace("#Z-z-Z#","\n",trim($_POST["input_sequence"]));
  // replace 's to html safe 's in fasta identifier lines so the sql insert doesn't break
  $input_sequence_data = preg_replace("'","&#39;",$input_sequence_data);

  // if there's an est search, the est blast dbs will need to be looked up
  if($_POST["detailed_est"]){
    $sql = "select filename from blast_est";
    $dbResult = pg_query(DB::getUI_DB(),$sql);
    while ($row = pg_fetch_assoc($dbResult)){
      $_POST["blastdbs"][] = $row["filename"];
    }
  }
```
else if($_POST['standard_est']){
    $sql = "select file_name from blast_databases where sequence_type='EST'";
    $dbResult = pg_query(DB::getUI_DB(),$sql);
    while ($row = pg_fetch_assoc($dbResult)){
        $_POST['blastdbs'] [] = $row['file_name'];
    }
}

//remove any duplicates the previous est commands may have entered
$_POST['blastdbs'] = array_unique($_POST['blastdbs']);

//submit search
$blast = new BLASTJob($_POST['program'],$_POST['input_sequence_data'],$_POST['blastdbs'],$_POST['wordsize'],$_POST['evaluate'],$_POST['scoringmatrix'],$_POST['dust'],$_POST['num_results'],$_POST['num_results']);

$result = $blast->submit();

try { } catch (JobException $e) {
    $error_rdir = "http://" . $_SERVER['HTTP_HOST'] . "/Tools/BLAST/?error=1&e0=" . urlencode($e->getMessage());
    header("Location: " . $error_rdir);
    print "Error! Please see the error message \a href=" . $error_rdir . ">here</a>.";
    exit();
}

/job_id = $result;

if($job_id == ''){
    echo '<!-- error --> <div class="jobBox" style="margin-right:8px"> <div class="errorText">Unable to get a job id, the cluster may be offline.</div> </div>';
    exit();
$blast_params_array = array("program"=>$_POST["program"],
"sequence"=>$_input_sequence_data,
"databases"=>$_POST["blastdbs"],
"wordsize"=>$_POST["wordsize"],
"numhits_oneline"=>$_POST["num_results"],
"numhits_align"=>$_POST["num_results"],
"evalue"=>$_POST["evalue"],
"scoringmatrix"=>$_POST["scoringmatrix"],
"masking"=>$_POST["dust"],
"searchId"=>$_blast->searchId);

if($_SESSION['logged_in']) $uid = $_SESSION['user_id'];
else $uid = 'Anonymous';

//enter search information into job db
$jobtype_insert_query = "insert into job_params (job_id, argument, value) values ('".$job_id."',
'job_type','".$_POST["program"]."');
pg_query(DB::getJOB_DB(), $jobtype_insert_query);

$jobuser_insert_query = "insert into job_params (job_id,argument,value) values ('".$job_id.
'user_id','".$uid."');
pg_query(DB::getJOB_DB(), $jobuser_insert_query);

$ip_query = "insert into job_params (job_id,argument,value) values ('".$job_id.
'submitter_ip','".$_SERVER["REMOTE_ADDR"]."');
pg_query(DB::getJOB_DB(), $ip_query);

$date_query = "insert into job_params (job_id,argument,value) values ('".$job_id.
'job_submit_date','".date('Y-m-d H:i:s')."');

pg_query(DB::getJOB_DB(), $date_query);

if($_POST['detailed_est']) pg_query(DB::getJOB_DB(), "insert into job_params (job_id, argument, value) values (" . $job_id . ",'detailed_est','true');");
else pg_query(DB::getJOB_DB(), "insert into job_params (job_id, argument, value) values (" . $job_id . ",'detailed_est','false');");

foreach($blast_params_array as $bpa_key => $bpa_value) {
  if($bpa_key == "databases"){
    foreach ($bpa_value as $database){
      $bpa_query = "insert into job_params (job_id, argument, value) values (".$job_id.",
      'target_database','".$database ")."");
      pg_query(DB::getJOB_DB(), $bpa_query);
    }
  }else{
    $bpa_query = "insert into job_params (job_id, argument, value) values (".$job_id.",
    ",".$bpa_key.",',".$bpa_value.")");
    pg_query(DB::getJOB_DB(), $bpa_query);
  }
}

// oh no! no id was returned. app server or xgrid is down
if ($job_id == "") echo '<span class="errorText">No job id was returned! VectorBase is malfunctioning at the moment but should be fixed shortly.</span>';
else echo $job_id;

// a job id was given for job retrieval, lets jump to the status page
else{
  $status_rdr = "http://" . $_SERVER["HTTP_HOST"] . "/Tools/BLAST/?job_id=" .$_POST["job_id"]."&page=status";
108   header("Location: " . $status_rdr);
109 }
110 ?>
C.3 status.php

```php
<?php
include_once("/Volumes/Web/vectorbase/includes/Jobs.php");
include_once("blast_results.php");
include_once('/Volumes/Web/vectorbase/includes/db_connect.php');
include_once('/Volumes/Web/vectorbase/includes/organism_definitions.php');

function vbPack($goodies){
    //print "<br><pre>".serialize($goodies)."</pre><br>";
    //$ser = serialize($goodies);
    //print_r(unserialize($ser));
    return serialize($goodies);
}

function vbUnpack($goodies){
    return unserialize($goodies);
}

//get all job ids for group id
$sql = "select job_id from job_id_groups where group_id=".$_GET["job_id"].";";
$sqlResult = pg_query(DB::getJOB_DB(), $sql);
while ($db_row = pg_fetch_assoc($sqlResult)){
    $jobIds[]=db_row["job_id"];
}
//proceed if there are results/db entries for this job
if($jobIds!=null){
```
//Checking to make sure there are no errors with getting status
//also grab the db name being searched while we're at it
$i = 0;
foreach ($jobIds as $jobId) {
    //first get the db file name
    $sql = "select description from job_id_groups where job_id='".$jobId."';";
    $sqlResult = pg_query(DB::getJOB_DB(), $sql);
    while ($db_row = pg_fetch_assoc($sqlResult)) {
        $jobDBs[$i] = $db_row["description"];
    }
    //use db file name to look up its display name
    $sql = "select display_id from blast_databases where file_name='".$jobDBs[$i]."';";
    $sqlResult = pg_query(DB::getUI_DB(), $sql);
    while ($db_row = pg_fetch_assoc($sqlResult)) {
        $jobDBNames[$i] = $db_row["display_id"];
    }
    $i++;
}

/*
for($i = 0; $i<count($jobIds); $i++) {
    if($jobsStatus[$i] == "Finished") $finished++;
}
$percentDone = round($finished/count($jobIds)*100);
*/

$percentDone = round($finished/count($jobIds)*100);

//echo '<div id="progressBarBorder" style="width:97%; height: 15px; margin: 4px; border: 1px solid #'.DV::$OCS['all'][0].';">'
//echo '<div id="progressBar" style="width:'.$percentDone.'%; height: 100%; vertical-
align:middle; text-align:center; color: white; background-color: #'.DV::$OCS["all"][2].' "$percentDone."\n</div>
60        </div>
61
62        <table class="jobOptionTable">
63        for($i=0; $i<count($jobIds); $i++){
64            echo '<tr class="jobOptionTr"><td class="jobOptionTd">'.$jobDBNames[$i].'"</td><td class="jobOptionTd">';
65            if ($jobsStatus[$i] == "Running" || $jobsStatus[$i] == "Pending") echo "<blink>".$jobsStatus[$i]."</blink>";
66            else echo $jobsStatus[$i];
67            echo "</td></tr>\n";
68        }
69        echo '</table>
70        </div>
71
72 //get all job parameters
73 $sql = "select * from job_params where job_id="$_GET["job_id"].";";
74 $sqlResult = pg_query(DB::getJOB_DB(), $sql);
75 while ($db_row = pg_fetch_assoc($sqlResult)){
76    switch($db_row["argument"]){
77        //case "job_type": $jobOptions["Job type"]=$db_row["value"]; break;
78        //case "user_id": $jobOptions["User ID"]=$db_row["value"]; break;
79        //case "submitter_ip": $jobOptions["Submitter IP"]=$db_row["value"]; break;
80        //case "job_submit_date": $jobOptions["Job submission date"]=$db_row["value"]; break;
81        case "program": $jobOptions["Program Type"]=$db_row["value"]; break;
82        case "wordsize": $jobOptions["Wordsize"]=$db_row["value"]; break;
83        //case "numhits_oneline": $jobOptions["Number of hits on one line"]=$db_row["value"];break;
84        //case "numhits_align": $jobOptions["Number of hits align"]=$db_row["value"]; break;
85        case "evalue": $jobOptions["E-value"]=$db_row["value"]; break;
case "scoringmatrix": $jobOptions['Scoring Matrix'] =$db_row['value']; break;
case "masking": $jobOptions['Masking'] =$db_row['value']; break;
}

//display the job params
echo '<div class="jobBox" style="float:left;">' .
<div class="jobBoxTitle">BLAST Paramaters</div>
<table class="jobOptionTable">
    foreach(array_keys($jobOptions) as $jobOption){
        echo '<tr class="jobOptionTr"><td class="jobOptionTd">'.$jobOption.'</td>' .
        echo '<td class="jobOptionTd">'.$jobOptions[$jobOption].'</td></tr>' .
    }
    echo '</table>
</div>
</div>' .

//jobs are done, show results
if(!$notDone){
    $results = array();
    $results_error = 0;
    $resQuery = "select raw_results from blast_results where blast_id = " . $_GET['job_id'];
    $resRes = pg_query(DB::getJOB_DB(), $resQuery);
    $resRow = pg_fetch_assoc($resRes);
    //Our results are parsed and in the database - let's just pull them out
    if($resRow['raw_results']){
        $hashQuery = "select result_id,hash from blast_hashes where blast_id = " . $_GET['job_id'];
        $hashRes = pg_query(DB::getJOB_DB(), $hashQuery);
        $i = 0;
        while ($hashRow = pg_fetch_assoc($hashRes)){

        }
115  //Strip slashes because we put it in using pg_prepare
116  $results[$i++] = vbUnpack(pg_unescape_bytea(stripslashes($hashRow['hash'])))
117  }
118  }else{
119    // First time getting results - let's parse them and put into the database
120    try {
121      $res = Jobs::getResults($_GET['job_id'], "BLAST");
122    }catch(JobException $e){
123      $results_error=1;
124      if ($e->getJobCode() == JobException::NO_RESULTS)
125      print "<br /><div style="font-weight:bold; padding-left:10px; color:red; font-size:14px;">No results for job ".$_GET['job_id'].". Please report this error to the webmaster.</div>">
126        elseif ($e->getJobCode() == JobException::NO_JOB)
127          print "<br /><div style="font-weight:bold; padding-left:10px; color:red; font-size:14px;">No job with id ".$_GET['job_id'].".</div>">
128            elseif ($e->getJobCode() == JobException::CLIENT) {
129              print "<br /><div style="font-weight:bold; padding-left:10px; color:red; font-size:14px;">Job with id ".$_GET['job_id']." is too large and returned too many results. Please consider using a larger word size or a smaller E-Value.</div>">
130                else print $e->getMessage();
131            }
132        }
133    } catch (Exception $e) {
134      print "<br /><div style="padding-left:10px;">";<a href="/Tools/BLAST/?blast_id=".".$_GET['job_id'].">Run new BLAST with same parameters</a></div>">
135        else print $e->getMessage();
136    }
137    // no error
138    if (!$results_error){
139      //Parsing everything
140      //Insert parsed results
$insertQuery = "insert into blast_results (blast_id, raw_results, parsed_results) values (" . $_GET['job_id'] . "," . addslashes($res->results()) . "," . addslashes($res->parsedResults()) . ");"
pg_query(DB::getJOB_DB(), $insertQuery);

$tempResult = new VectorbaseBlastResult();
$tempHit = new VectorbaseBlastHit();
$tempHsp = new VectorbaseBlastHsp();

$contents = explode("\n", $res->parsedResults());
// 0 = nothing. 1 = result. 2 = hit. 3 = hsp.
$on = 0;
$hspId = 1;
$hitId = 1;
$resId = 1;
$gffNum = 0;
$i = 0;

foreach ($contents as $input) {
    $i++;
    $input = chop($input);
    // If input is a new result/hit/hsp we declare the arrays
    if (substr($input, 0, 2) == "++") {
        $on++;
        if ($on == 1) $tempResult = new VectorbaseBlastResult();
        if ($on == 2) $tempHit = new VectorbaseBlastHit();
        if ($on == 3) $tempHsp = new VectorbaseBlastHsp();
    }
    // If input is end of a result/hit/hsp
    // we iterate to get the next one, and
elseif (substr($input, 0, 2) == '--') {
    if($on == 1) {
        $tempResult->clearEmptyHits();
        $addResultQuery = "insert into blast_hashes (blast_id,result_id,hash) values ($1, $2, $3)"
        pg_prepare(DB::getJOB_DB(), "my_query$i", $addResultQuery); //DB::getJOB_DB(), $addResultQuery);
        if(!pg_execute(DB::getJOB_DB(), "my_query$i", array($_GET['job_id'], $resId, pg_escape_bytea(vbPack($tempResult))))){
            pg_prepare(DB::getJOB_DB(), "my_query", $addResultQuery); //DB::getJOB_DB(), $addResultQuery);
            print "die";
        }
        $results[$resId-1] = $tempResult;
        $resId++;
    }
    if($on == 2) {
        $tempHit->setId($hitId++);
        $tempResult->addHit($tempHit);
    }
    if($on == 3) {
        $tempHsp->setId($hspId++);
        $tempHsp->setParent($tempHit);
        $tempHit->addHsp($tempHsp);
    }
    $on--;
} elseif( substr($input, 0, 8) == '*)&*gff' ){
    } //
(!file_exists($viewPath.'view'.$_GET['job_id'].'.gff'))

$onGFF = 1;

$gffNum = substr($input, 8, 1);

$gfh = fopen($tmpGFF, "w");

if (!$gfh) {
    print "Error!1"; exit; }

$gfh = fopen("/Volumes/Web2/vectorbase/data/job_output/BLAST/".$_GET['job_id'].'_'. $gffNum.'.gff", "w");

if (!$gfh) {print "Error!1"; exit; }

elseif(substr($input, 0, 9) == '*&**/gff') {
    $onGFF = 0;
    fclose($gfh);
}

elseif($on == 0 && $onGFF){
    if(substr($input, 0, 5) == "track") $input .= $_GET['job_id'];

    if (!$write($gfh, $input . "\n")) { print "ha".$input . "Error!2" . $on; exit; }
}

elseif($on == 1){
    if($matches[0] == "res_database_name") $tempResult->setDb($matches[1]);
    if($matches[0] == "res_query_name") $tempResult->setQueryString($matches[1]);
}

elseif($on == 2){
    if($matches[0] == "hit_name") $tempHit->setName($matches[1]);
}

elseif($on == 3){

// We're looking at a gff

}
switch($matches[0]) {
    case "hsp_evalue":
        $tempHsp->setEvalue( $matches[1] );
        break;
    case "hsp_length(hit)":
        $tempHsp->setLength( $matches[1] );
        break;
    case "hsp_score":
        $tempHsp->setScore( $matches[1] );
        break;
    case "hsp_percent_identity":
        $tempHsp->setIdentity( $matches[1] );
        break;
    case "hsp_start(hit)":
        $tempHsp->setHitStart( $matches[1] );
        break;
    case "hsp_end(hit)":
        $tempHsp->setHitEnd( $matches[1] );
        break;
    case "hsp_start(query)":
        $tempHsp->setQueryStart( $matches[1] );
        break;
    case "hsp_end(query)":
        $tempHsp->setQueryEnd( $matches[1] );
        break;
    case "hsp_homology_string"
        $tempHsp->setHomologyString( $matches[1] );
        break;
    case "hsp_hit_string"
        $tempHsp->setHitString( $matches[1] );
        break;
}
```php
    case "hsp_strand(hit)"
        $tempHsp->setStrand($matches[1]);
        break;
        case "hsp_query_string"
        $tempHsp->setQueryString($matches[1]);
        break;
    }
} //end foreach
} //end error check

//End getting results. now display stuff
    echo '<table class="jobOptionTable">

    $iter1 = 0;
    while($iter1<sizeof($results)){
        preg_match("/((\.*\*)\*(\.*))/", $results[$iter1]->getDb(), $db);
        $blastQ = "select o.short_name,b.display_id, b.sequence_type from blast_databases b,
                    organism o where o.organism_id=b.organism_id and b.file_name = "' . $db[3] . "'";
        $bqResults = pg_fetch_assoc(pg_query(DB::getUI_DB(), $blastQ));
        echo '<tr
            <td align="center" valign="top" width="10" rowspan="3" bgcolor="#fcfcfc">&bull;</td>
            <td colspan="2" bgcolor="#ffffff">.' . $results[$iter1]->getQueryString() . ' vs. <i>' . $bqResults["short_name"] . '</i> ' . $bqResults["display_id"] . '</td>
        </tr>
        <tr>
            <td colspan="2" height="2" bgcolor="#f9f9f9"></td>
        </tr>
```
<td bgcolor="#fcfcfc" class="small" style="padding:3px;" width="205"> // TODO
// here's the problem -- there are hits getting passed around without hsp's in them!
// do i fix it in the perl or in the php?
// php is probably easier - do it when we read the hsp in (something like clearEmptyHits()
   // yes!
   print "<b>" . $results[$iter1]-&gt;getNumHsps();
   print " Hit";
   if ($results[$iter1]-&gt;getNumHits() != 1) echo "s ";
   print "</b> on " . $results[$iter1]-&gt;getNumHits() . " unique " . $bqResults["sequence_type"];
   if ($results[$iter1]-&gt;getNumHits() != 1) print "s";
   print ".";
   echo '</td>
<tr bgcolor="#fafafa" class="small" align="right" style="padding-right:4px;padding-left:4px;">'
   if ($results[$iter1]-&gt;getNumHsps() > 0)
      print "<a href="/Tools/BLAST/?result-html_results&job_id=" .
          $_GET['job_id'] . 
          
          '&result='.($iter1+1). "&resNum=" . ($iter1+1) . "&page=res" class="hashelp" name="Web:Misc_Help#view_interactive_results">Interactive Results</b></a>&nbsp;&bull;&nbsp;
     echo '<a class="hashelp" name="Web:Misc_Help#view_raw_blast_results" href="/sections/Tools/tool_includes/BLAST/raw_results.php?job_id='
          
          .$_GET['job_id'] . 
          
          
          
          
          &result='.($iter1+1). '">Raw Results</a>
```php
<tr>
    <td>
        $iter1++;
        if ($iter1!=sizeof($results)) print "<tr><td colspan="3" height="8"></td></tr>
";
    }
    echo '</table>'
</div>
```

```php
//} //end check for results being done

}else{
    //no info for the job id, alert the user
    $rdr = "http://" . $_SERVER["HTTP_HOST"] . "/Tools/BLAST/?error=1&e14=1&id="$_GET['job_id'];
    header("Location: " . $rdr); } else{
```
function vbPack($goodies) {
    return serialize($goodies);
}

function vbUnpack($goodies) {
    return unserialize($goodies);
}

// proceed if there are results/db entries for this job
if ($_GET['job_id'] !== null) {

$results = array();

$resQuery = "select raw_results from blast_results where blast_id = " . $_GET['job_id'];
$resRes = pg_query(DB::getJOB_DB(), $resQuery);
$resRow = pg_fetch_assoc($resRes);

//Our results are parsed and in the database - let's just pull them out
if($resRow['raw_results']){
    $hashQuery = "select result_id,hash from blast_hashes where blast_id = " . $_GET['job_id'];
    $hashRes = pg_query(DB::getJOB_DB(), $hashQuery);
    $i = 0;
    while ($hashRow = pg_fetch_assoc($hashRes)){
        //Strip slashes because we put it in using pg_prepare
        $results[$i++] = unserialize(pg_unescape_bytea(stripslashes($hashRow['hash'])));
    }
}
else{ // First time getting results - parse and place into database
    $res = Jobs::getResults($_GET['job_id'], "BLAST");
    $insertQuery = "insert into blast_results (blast_id,raw_results,parsed_results) values (" . $_GET['job_id'].""," . addslashes($res->results()).""," . addslashes($res->parsedResults()).")";
    pg_query(DB::getJOB_DB(), $insertQuery);
    $tempResult = new VectorbaseBlastResult();
    $tempHit = new VectorbaseBlastHit();
    $tempHsp = new VectorbaseBlastHsp();
    $contents = explode("\n", $res->parsedResults());
    // 0 = nothing. 1 = result. 2 = hit. 3 = hsp.
    $on = 0;
    $hspId = 1;
    $hitId = 1;
$resId = 1;
$gffNum = 0;

$i=0;

foreach($contents as $input) {
    $i++;
    $input = chop($input);
    // If input is a new result/hit/hsp we declare the arrays
    if (substr($input, 0, 2) == '++') {
        $on++;
        if($on == 1) $tempResult = new VectorbaseBlastResult();
        if($on == 2) $tempHit = new VectorbaseBlastHit();
        if($on == 3) $tempHsp = new VectorbaseBlastHsp();
    }
    // If input is end of a result/hit/hsp
    // we iterate to get the next one, and
    // reset the last data structure
    elseif (substr($input, 0, 2) == '--') {
        if($on == 1){
            //Add result to database
            $tempResult->clearEmptyHits();
            $addResultQuery = "insert into blast_hashes (blast_id,result_id,hash) values ($1, $2, $3)"
            pg_prepare(DB::getJOB_DB(), "my_query$i", $addResultQuery); //DB::getJOB_DB(), $addResultQuery);
            if(!pg_execute(DB::getJOB_DB(), "my_query$i", array($_GET['job_id'], $resId, pg_escape_bytea(vbPack($tempResult) )) )){
                //pg_prepare(DB::getJOB_DB(), "my_query", $addResultQuery); //DB::getJOB_DB(), $addResultQuery);
            //this is doing stuff..why>
            //print "die";
$results[$resId-1] = $tempResult;
$resId++; } 

if($on == 2){
    $tempHit->setId($hitId++);
    $tempResult->addHit($tempHit);
} 

if($on == 3){
    $tempHsp->setId($hspId++);
    $tempHsp->setParent($tempHit);
    $tempHit->addHsp($tempHsp);
}

$on--;

elseif( substr($input, 0, 8) == '*&*&*gff') {// && (!file_exists($viewPath.'view'.$_GET['job_id'].'.gff')) )
    $onGFF = 1;
    $gffNum = substr($input, 8);
    $tmpGFF = "/Volumes/Web/vectorbase/data/job_output/BLAST/" . $_GET['job_id'] . "_" . $gffNum . ".gff";
    $gfh = fopen($tmpGFF, "w");
    if (!$gfh) {print "Error!1"; exit; }

    elseif( substr($input, 0, 9) == '*&*&*/gff' ) {
        $onGFF = 0;
        fclose($gfh); 
    } 

    // We're looking at a gff
elseif($on == 0 && $onGFF){
    if(substr($input, 0, 5) == "track") $input .= $_GET['job_id'];
    if (!fwrite($gfh, $input . "\n") { print "ha".$input . "Error!2" . $on; exit; }
    //Write the line to the file!
    //$input
} else {
    $matches = explode('=>', $input, 2);
    if($on == 1){
        if($matches[0] == "res_database_name") $tempResult->setDb($matches[1]);
        if($matches[0] == "res_query_name") $tempResult->setQueryString($matches[1]);
    }
    if($on == 2){
        if($matches[0] == "hit_name") $tempHit->setName($matches[1]);
    }
    if($on == 3){
        switch($matches[0]) {
            case "hsp_evalue":
                $tempHsp->setEvalue( $matches[1] );
                break;
            case "hsp_length(hit)"
                $tempHsp->setLength( $matches[1] );
                break;
            case "hsp_score"
                $tempHsp->setScore( $matches[1] );
                break;
            case "hsp_percent_identity"
                $tempHsp->setIdentity( $matches[1] );
                break;
            case "hsp_start(hit)"
                $tempHsp->setHitStart( $matches[1] );
                break;
        }
    }
break;
    case "hsp_end(hit)"
    $tempHsp->setHitEnd( $matches[1] );
    break;
    case "hsp_start(query)"
    $tempHsp->setQueryStart( $matches[1] );
    break;
    case "hsp_end(query)"
    $tempHsp->setQueryEnd( $matches[1] );
    break;
    case "hsp_homology_string"
    $tempHsp->setHomologyString( $matches[1] );
    break;
    case "hsp_hit_string"
    $tempHsp->setHitString( $matches[1] );
    break;
    case "hsp_strand(hit)"
    $tempHsp->setHitStrand( $matches[1] );
    break;
    case "hsp_strand(query)"
    $tempHsp->setQueryStrand( $matches[1] );
    break;
    case "hsp_frame(hit)"
    $tempHsp->setHitFrame( $matches[1] );
    break;
    case "hsp_frame(query)"
    $tempHsp->setQueryFrame( $matches[1] );
    break;
    case "hsp_query_string"
    $tempHsp->setQueryString( $matches[1] );
    break;
foreach($results as $i => $result){
    //preg_match('/\((\./.*)\)*(.+)/', $result->getDb(), $db);
    $db=$results[$i]->getDb();
    $blastQ = "select o.short_name, b.display_id, b.sequence_type from blast_databases b, organism o
    where o.organism_id=b.organism_id and b.file_name = '" . $db . "';

    if( !pg_fetch_assoc(pg_query(DB::getUI_DB(), $blastQ)) ){
        $blastQ = "select species, short_name from blast_est where filename = '' . $db . "';
        $bqResults[] = pg_fetch_assoc(pg_query(DB::getUI_DB(), $blastQ));
        $organismsSearched[] = ORG::$LTS[$bqResults[$i]["species"]];
        $dbDisplayIds[] = $bqResults[$i]["short_name"];
        $querySequences[] = $results[$i]->getQueryString();
        $dbsSearched[] = $db;
    } else{
        $bqResults[] = pg_fetch_assoc(pg_query(DB::getUI_DB(), $blastQ));
        $organismsSearched[] = $bqResults[$i]["short_name"];
        $dbDisplayIds[] = $bqResults[$i]["display_id"];
        $querySequences[] = $results[$i]->getQueryString();
    }
}
$dbsSearched[] = $db;

if(!array_unique($organismsSearched)){
    //something bad happened if we managed to get here... the job status says done but there are no results!
    echo '<div class="jobBox" style="float:left; clear:both; margin-left:8px; width:98%;" >&lt;h1&gt;Job '.$_GET['job_id'].' Results&lt;/h1&gt;';
    echo '<p>Could not retrieve results! Possible cause: Results too large or improper output generated. &lt;a href="/sections/Tools/tool_includes/BLAST/raw_results.php?job_id='.$_GET['job_id'].'">Job raw output&lt;/a&gt;&lt;/p&gt;&lt;/div&gt;';
    echo ' <!-- error -->';
    exit();
}

$uniqueOrganisms = array_unique($organismsSearched);
$uniqueQueries = array_unique($querySequences);
$hitlessOrganisms[]=NULL;
foreach($uniqueOrganisms as $organism){
    $emptyResult=true;
    foreach($uniqueQueries as $query){
        for ($i=0; $i<sizeof($organismsSearched); $i++){
            if($organism==$organismsSearched[$i] && $query==$querySequences[$i]){;
                if($results[$i]->getNumHsps()!==0){
                    $evalueSortedResults=$results[$i]->getSortedHsps('evalue','asc');
                    $blastHits[$organism][$query][]= new blastResult($dbDisplayIds[$i],$query,$results[$i]->getNumHsps(),$evalueSortedResults[0]->getEvalue(), $i);
                    $emptyResult=false;
                } else {
                    $emptyResult=false;
                }
            }
        }
    }
}
else {
    //something bad happened if we managed to get here... the job status says done but there are no results!
    echo '<div class="jobBox" style="float:left; clear:both; margin-left:8px; width:98%;" >&lt;h1&gt;Job '.$_GET['job_id'].' Results&lt;/h1&gt;';
    echo '<p>Could not retrieve results! Possible cause: Results too large or improper output generated. &lt;a href="/sections/Tools/tool_includes/BLAST/raw_results.php?job_id='.$_GET['job_id'].'">Job raw output&lt;/a&gt;&lt;/p&gt;&lt;/div&gt;';
    echo ' <!-- error -->';
    exit();
}
//find organisms with zero hits for all dbs
if($emptyResult) $hitlessOrganisms[]=$organism;

//display results
echo '<div class="jobBox" style="float:left; clear:both; margin-left:8px; width:98%;"><h3>Job 
'.$_GET['job_id'].' Results</h3></div>','

//this is for the separate results page
if($separate) include_once('displayInputParams.php');

//sort by organism
foreach($uniqueOrganisms as $organism){
    if(!array_search($organism,$hitlessOrganisms) && $organism!=NULL){
        $organismTitle=(ORG::$FN[$organism])?ORG::$FN[$organism]:'mRNAseq Data';
        echo '<div class="jobBox" style='"border: 1px solid ";.DV::$BLASTOCS[$organism][4].'; margin: 2px 0px; margin-left:8px; float:left; clear:both; width:98%;">','
border-width: 0px; border-bottom-width: 1px; border-color:'.$.DV::$BLASTOCS[$organism][4].'; background- color:'.$.DV::$BLASTOCS[$organism][5].';">'.$.organismTitle.'</h1>','

    if($blastHits[$organism]){ //somehow null enetries are getting into this array. make sure it has something here
        foreach($blastHits[$organism] as $query){
            $rowCount=0;
            echo '<table style="margin-left:10px; border-spacing: 0px 3px; width:98%;color:';
<tr><td style="width:33em; font-size:1.1em; font-weight:bold; color: #DV::$BLASTOCS[$organism][0]." > querying
</td><td style="font-weight:bold; text-align:center; color: #DV::$BLASTOCS[$organism][0]." > Hits
</td><td style="font-weight:bold; text-align:center; color: #DV::$BLASTOCS[$organism][0]." > Best E-Value
</td><td style="font-weight:bold; text-align:center; color: #DV::$BLASTOCS[$organism][0]." > Raw Results
</td></tr>

foreach($query as $blastresult){
  if($rowCount%2==1) $color="ffffff";
  else $color=DV::$BLASTOCS[$organism][6];

  $i=$blastresult->getResultNumber();
  echo '<tr';
  echo '<td style="background-color: #'.$color.';">';<a target="new" href="/Tools/BLAST/?result=html_results&job_id='.$_GET['job_id'].'&resNum=(($i+1)).'&page=res" class="hashelp" name="Web:Misc_Help#view_interactive_results">'.strip_tags($blastresult->getDbName()).'</td>
  echo '<td style="text-align:center; color: #'.$color.';">'.($i+1).'</td>
  echo '<td style="text-align:center; color: #'.$color.';">'.($i+1).'</td>
  echo '<td style="text-align:center; color: #'.$color.';' width:8em; background-color: #'.$color.';">';<a class="hashelp" target="new" name="Web:Misc_Help#view_raw_blast_results" href="/sections/Tools/tool_includes/BLAST/raw_results.php?job_id='.$_GET['job_id'].'&result=(($i+1)).'"><img src="/imgs/trans_doc.gif" height="16" width="14"/></a></td>
  echo '</tr';
  $rowCount++;
}
}
278    echo '</table>';  
279 }  
280  
281 echo '</div>';  
282  
283 }  
284 }  
285  
286 //display organisms with no hits  
287 if($hitlessOrganisms[1]!=NULL){  
288   echo '<div class="jobBox" style="float:left; clear:none; width:98%; border: 1px solid #DV::$BLASTOC[$all][4].'; margin: 2px; margin-left:8px;">  
289   <h1>No Results for these organisms</h1>  
290   <ul>';  
291   foreach($hitlessOrganisms as $organism){  
292     echo '<li><i>ORG::$FN[$organism].'</i></li>';  
293   }  
294   echo '</ul></div>';  
295 }  
296  
297 }else echo '<span class="errorText">No job ID was entered. How did you get to this point?</span>';
BIBLIOGRAPHY


Apple 10.4 to 10.5 Migration Guide:

Apple 10.4 to 10.5 Migration Guide Second Edition:

Apache Software Foundation, Apache HTTP Server Project, 2007,
http://httpd.apache.org

PostgreSQL Database Management System, PostgreSQL, 2007,
http://www.postgresql.org/

The MacPorts Project, MacPorts, 2007,
http://www.macports.org/

W3C, Simple Object Access Protocol (SOAP), 2007,
http://www.w3.org/TR/soap/