A SECURE WEB APPLICATION PROVIDING PUBLIC ACCESS TO HIGH-PERFORMANCE DATA INTENSIVE SCIENTIFIC RESOURCES

ScalaBLAST Web Application

Darren Curtis, Elena Peterson and Christopher Oehmen
Pacific Northwest National Laboratory, PO Box 999 MSIN: K7-28, Richland, WA 99352, U.S.A.

Keywords: Scalable, Grid Computing, Access Control, Web Application, Security, ScalaBLAST, Portable Batch System.

Abstract: This work presents the ScalaBLAST Web Application (SWA), a web based application implemented using the PHP script language, MySQL DBMS, and Apache web server under a GNU/Linux platform. SWA is an application built as part of the Data Intensive Computer for Complex Biological Systems (DICCBS) project at the Pacific Northwest National Laboratory (PNNL). SWA delivers accelerated throughput of bioinformatics analysis via high-performance computing through a convenient, easy-to-use web interface. This approach greatly enhances emerging fields of study in biology such as ontology-based homology, and multiple whole genome comparisons which, in the absence of a tool like SWA, require a heroic effort to overcome the computational bottleneck associated with genome analysis. The current version of SWA includes a user account management system, a web based user interface, and a backend process that generates the files necessary for the Internet scientific community to submit a ScalaBLAST parallel processing job on a dedicated cluster.

1 INTRODUCTION

Biological sequence analysis is a computational methodology for helping characterize newly sequenced genomes. The fundamental task in sequence analysis is a comparison of an uncharacterized gene or protein to a collection of sequences from previously sequenced genomes. Through sequence analysis, one can build evidence for the *functional* role of a newly sequenced gene or protein. New genomes are being sequenced at an exponentially increasing rate resulting in a sequence data 'avalanche'.

Efficiently managing sequence analysis on this growing dataset is a challenge that must be addressed using high performance computing. However, requiring the biology community to use high-end hardware is not practical. We instead strive to deliver high-end computing capacity for large-scale sequence analysis through convenient, intuitive and secure interfaces to advanced hardware and software.

ScalaBLAST is a high-performance sequence analysis implementation, (Oehmen & Nieplocha, 2006), which accommodates very large databases and which scales linearly to as many as thousands of processors on both distributed memory and shared memory architectures. This paper represents a web application front-end to ScalaBLAST (SWA) that allows users to take advantage of the high-performance sequence alignment running on a 64 processor cluster from a public web-site and have full access to the results. Several issues are addressed by the web application including:

- Security
- User account management system
- Server side processing which includes:
  - File management that allows the user to upload data files and download results
  - Script generation to create jobs control files
  - Notification that sends email when jobs start and finish
  - Maintaining a history of the user's jobs

The SWA is currently in production at http://www.biopilot.org.

Curtis D., Peterson E. and Oehmen C. (2008).
A SECURE WEB APPLICATION PROVIDING PUBLIC ACCESS TO HIGH-PERFORMANCE DATA INTENSIVE SCIENTIFIC RESOURCES - ScalaBLAST Web Application.
In *Proceedings of the Fourth International Conference on Web Information Systems and Technologies*, pages 244-251
DOI: 10.5220/0001521902440251

2 MOTIVATION AND DESIGN GOALS

The primary motivation for developing SWA is to provide the bioinformatics scientific community with access to high-performance sequence alignment via a web application.

2.1 Provide Scientists Internet Access to High Performance ScalaBLAST Application

ScalaBLAST is an application that runs on a variety of shared and distributed memory multiprocessor architectures. The source code is available to scientists upon request but most scientists do not have access to large multiprocessor systems. ScalaBLAST is an optimized implementation of the popular BLAST (Altschul, Gish, Miller, Myers, & Lipman, 1990) (Altschul et al. 1997) algorithm used by the scientific community for sequence analysis. Providing scientists with access to the ScalaBLAST application on high-performance systems at PNNL allows them to perform large-scale sequence analysis which may be beyond the reach of conventional BLAST installations.

2.2 Provide a Web Based Interface to ScalaBLAST

ScalaBLAST has a command line interface with over 20 command line options. Most of the options have default values that are commonly used and can be considered "advanced" options. The user interface has been greatly simplified so that most users only need to select the "Program Type" of Nucleotide (blastn) or Protein (blastp), "Output Format" of Text file, Zipped file, or Tar file, "Database" to run their query against, and the query file they want to upload.

2.3 Provide a User Interface Consistent with Existing BLAST User Interfaces

There are several software applications available that provide a basic graphical or web-based user interface to BLAST. We used user interface elements that were consistent with tools and applications like BLAST (NCBI, 2007), BatchBLAST (Harvard, 2007), and GSC Batch Server (Washington University, 2007). Users familiar with these BLAST user interfaces will be able to take advantage of the performance improvements of ScalaBLAST without having to learn another user interface. This approach helps bridge the gap between existing tools and high performance tools without forcing the user to learn a new system.

3 RELATED WORK

There are many user account management systems ranging from publicly available scripts on websites to portal and content management frameworks. In addition, there are several BLAST tools and websites to upload data and perform queries. We have addressed many challenges associated with integrating these independently designed components for scripting, scheduling, launching and monitoring user tasks in SWA.

3.1 Publicly Available Scripts

Many public domain scripts written in PHP do not provide adequate user account management. The user registration and login pages pass the user's password in unencrypted ASCII text. These scripts often store the password in a database as unencrypted ASCII text so that email can be sent to the user with their password. Some of the scripts use .htaccess and .htpasswd files to implement access control which is not very flexible for system administrators or application developers. Some of the scripts require the use of cookies which forces the user to have cookies enabled to use the application. Some scripts are older and do not work well with PHP 5.X so they were not used. An exhaustive analysis of these scripts is beyond the scope of this paper.

3.2 Drupal

Drupal (VanDyk & Westgate, 2007) is a comprehensive content management framework to create customized web sites. The development of Drupal-based applications is a great solution for many applications but it can be difficult to migrate or rewrite existing web applications or PHP libraries to Drupal Modules. Drupal requires more advanced programming experience and is beyond the skill level of many junior programmers that could use the techniques mentioned here to successfully implement an application similar to SWA.

3.3 BLAST Web-Based Applications

Most of the BLAST web-based applications are interfaces to BLAST so they do not take advantage of the performance improvements of ScalaBLAST. The Harvard BatchBLAST application gives the user the ability to perform multiple BLAST queries at the same time which is an improvement over BLAST but not as significant as the performance gains of ScalaBLAST.

4 DESIGN AND IMPLEMENTATION

4.1 Development Tools and Environment

Our intent is to demonstrate the success of our approach to SWA using multiple tools to develop web-based applications. We do not advocate any given tool or endorse any vendor. The purpose of discussing software tools is instead to present a comprehensive approach to deal with the challenges of developing web-based applications. We used the PhpED IDE (NuSphere, 2007) for writing PHP, JavaScript, HTML, SQL, and XML files. PhpED helps the developer by providing code completion, code highlighting, and other functionality. The biggest reason we used PhpED was for the integrated PHP debugger and embedded versions of Mozilla and Internet Explorer. We used Firefox 2.0 for debugging JavaScript and the Firefox Firebug add-on to debug CSS layout issues. For use case testing we used the Firefox Selenium IDE add-on to generate the initial Selenium test files.

During the development process we took the time to use tools like Adobe DreamWeaver CS3 to check spelling and do a site-wide checks to make sure all the links on the pages worked correctly. Unfortunately, DreamWeaver still does not have an automated site-wide check for spelling so each page had to be loaded and checked for spelling.

Selenium (OpenQA, 2007) is the tool we used for automated testing. There are several API libraries available for languages like Java, PHP, Python, PERL, etc, which all produce HTML commands to interface with Selenium.

Test cases written for Selenium use simple HTML tables. An example test case for typing “hello world” on the Google search engine might look like:

```
<table>
<tr><td>Google</td></tr>
<tr><td>open</td>
<td>http://www.google.com</td></tr>
<tr><td>type</td>
<td>q</td><td>"hello world"</td></tr>
<tr><td>click</td><td>btnG</td></tr>
</table>
```

The first row of the table is the title for the test case. The rest of the rows are the actions used to mimic what a user might do. The first column of the action row is the command. The second and third columns are for any arguments to the command. The second column is typically a URL to open or the name or id of the HTML element that is to be controlled. The third column is typically used to specify a value. In the example above, the first action is to ***open*** the Google web page. The second action is to ***type “hello world”*** in the HTML input field that is named ***q***. The last command is to do a left mouse ***click*** on the input button named ***btnG***.

There are some limitations to using Selenium. For example, to upload a file to the server, you cannot specify the file to be uploaded. The reason is that browsers ignore any file you specify in an input field with a type of filename. This is a security safeguard so that hackers cannot use javascript to grab files from your system when you visit their web page. Selenium is very full-featured but it cannot be used to test features that are added to thwart Internet bots. It cannot be used to test a user registration page that uses the technique of displaying an image with distorted text that the user has to interpret and input before they can submit a form.

Another challenge is testing the dynamic generation of URLs. Our Google test case fills out the search form and submits the form for processing. The resulting URL on the Google page is `http://www.google.com/search?num=100&hl=en&rls=GGLD%2CGGLD%3A2003-41%2CGGLD%3Aen&q=%22hello+world%22` which is generated. Trying to add an action that can verify that the resulting page had the correct URL is challenging.

Normal testing issues include setup and teardown of tests that insert or delete records in databases. For example, adding a user to the system using the user account registration page requires the test to remove the user prior to starting the test. Care must be taken when testing web pages that require username and password to use the page.

Rather than use the PHP API for Selenium, we took the approach of using the libraries we wrote to embed PHP into our HTML tables. The action to

check that the test is currently at the first page of the web site looks like:

```
<td>verifyLocation</td>
<td><?= getServerRootUrl() ?></td>
```

This allows us to test our internal functions and our higher level integration tests at the same time.

4.2 Security

The primary goal of any user access control and authorization system is to protect the user's password to mitigate risk of compromising the user's data or the application server. We approach this using several design philosophies.

4.2.1 Protected and Public Access

Most web sites have content that is available to the public as anonymous users. The public content should not require a user to obtain a user account to view the information. The protected content should only be accessible after the user has registered with the system and has used the proper credentials to access the protected areas of the system.

4.2.2 Protecting Directories

Directories can be protected by server-level access control (e.g. Apache httpd.conf), virtual server access control, .htaccess file overriding the server access control, or index.html redirects to an error, login, or site index page. One problem with server access control is that a system or web administrators could modify the server configuration files and unintentionally turn off access restrictions to your web application and expose your application to the public. Another problem is that the administrator could modify the server configuration files to no longer allow an .htaccess to override the server level access control which would block valid users from your application. We chose to put an index.html in each protected directory that redirects the user to the front page of the web site.

The index.html pages in protected directories use an HTML meta tag such as:

```
<meta http-equiv="refresh"
content="5;URL=/" />
```

As an additional security measure, we also include JavaScript code in the HTML head section such as:

```
<script type="text/javascript">
  window.location.href='/';
</script>
```

4.2.3 Protecting PHP Source Code

The web application is organized using a directory structure that separates the PHP source code from the web pages that use PHP. This can be done a number of different ways but we use the following directory structure where `/path` is the physical directory on the disk and `<N>` is the name of the site (e.g. biopilot.org):

```
/path/conf
/path/data
/path/lib
/path/www/Root/<N>/
```

The directory, `/path/www/Root/<N>/` is configured as the top level directory of the web site. In Apache httpd.conf this may look like:

```
Alias /dev/ /path/dev/www/Root/<N>/
Alias /tst/ /path/tst/www/Root/<N>/
Alias /prd/ /path/prd/www/Root/<N>/
```

The directory structure and alias allows for a development site, testing site, and production site on the same server. By putting all the PHP source code in `/path/lib`, the code cannot be accessed through the web browser using source code viewing tools (Source Viewer, 2007). This is very important if you have to put a username and password in your source code to access a database. The PHP code and HTML at the top of every page is based on a common page template.

4.2.4 Protecting Web Pages

We use a page template for every page on the web site. The template has a conditional section that specifies a web page as a document or an application. The first few lines of the template include some PHP code to include the PHP libraries and classes that determine if the page is public or private. This code is executed before the `<!DOCTYPE>` and `<html>` tags so that it is redirected to the login page if the page is protected and the user has not successfully logged in yet.

```
 1 <?php
 2 $WEBENV_DIR =
   preg_replace("/\/www\/Root\/.*/",
   '/lib',
   $_SERVER['SCRIPT_FILENAME']);
 3 require("$WEBENV_DIR/webenv.php");
 4 $title = 'PUT TITLE HERE';
 5 $doc   = 'document';
 6 // Put Extra PHP Code Here
 7 ?>
 8 <!DOCTYPE>
 9 <html>
10 <head>
11 <title><?= $title ?></title>
12 <!-- Stylesheets start -->
```

```
13 <!-- Stylesheets stop -->
14 <!-- Head JavaScript start -->
15 <!-- Head JavaScript stop -->
16 </head>
17 <body>
18 <div id="page">
19 <div id="container">
20 <div id="main">
21 <!-- content start -->
22 <h2><?= $title ?></h2>
23 <!-- content stop -->
24 </div> <!--main -->
25 </div> <!--container-->
26 <? vInclude($doc, 'navigation')?>
27 <? vInclude($doc, 'footer')?>
28 </div> <!--page-->
29 <!-- Tail JavaScript start -->
30 <!-- Tail JavaScript stop -->
31 </body>
32 </html>
```

Line #2 may look complicated but it is simply finding the absolute path to the main PHP library for this site. Line #3 performs a PHP require on the main PHP library, webenv.php, which knows if the document is protected or public based on the directory where the file is located. In addition, simply moving a document from a public directory into a protected directory causes the behavior of the file to be protected. Indentation and several parts of the template such as site specific CSS references and JavaScript have been left out for simplicity. The site specific CSS references and JavaScript would be located in the `<head>` section of the file. The author would add any document specific CSS references between lines 12-13 and any document specific JavaScript between lines 14-15. Line #26 will call a function to include the appropriate navigation for the web page based on whether the value of `$doc` on line #5 is `'document'` or `'application'`.

4.2.5 Protecting Databases

The web application may be running on the same system as the web server or another system. The web application should be designed to access the database using a hostname, port number, username, and password that are stored in a configuration file in case any of those values change. Many developers make the mistake of putting these values in their source code. The database administrator should limit access to the web application by granting access to only the database that is used by the application.

NOTE: If the system administrator installs MySQL in a directory other than the default location or configures MySQL (e.g. `/etc/mysql/my.conf`) to use a datadir other than /var/lib/mysql, then it is very important to modify the mysql.default_sock value in the php.ini file to the actual location of the mysql.sock named pipe file.

4.2.6 Protecting Passwords

The user's password needs to be protected. The SWA uses several techniques to make sure that the user's password is never sent across the network in clear ASCII text. In addition, SWA requires the user to choose a good password with minimum requirements. A good minimum requirement is at least on number, one uppercase letter, no common words, no username (forwards or backwards), and a length of 8 characters.

Some public domain scripts encrypt the password in JavaScript before submitting the HTML form for processing on the server. The most common mistake is that the developer has a password (and maybe a confirm password) input field and a hidden input field used to store the encrypted password on a web page. The user types their password and clicks on the Submit button. The form calls a JavaScript function that encrypts the password from the password input field and assigns it to the encrypted input field before submitting the form to the server. The problem is that the password input field still has the password in clear ASCII text and it is sent to the server as part of the form. This problem is easily corrected if the JavaScript function deletes the text from the password input field and double checks to make sure the password input field is empty before submitting the form.

The SWA stores the encrypted version of the password from the form in the database. All encrypting is done in the client's browser using JavaScript functions. This eliminates the need for any encryption in PHP or MySQL. The server side processing is reduced to comparing the encrypted form of the password from the browser with the value stored in the database.

If the user forgets their password, the user puts their email address in a form and submits it for processing. If the email address is in the SWA database, an email is sent to the user with a link containing an id and temporary key. The user can use the link to set a new password. A password is never stored in clear ASCII text so it cannot be emailed to the user. There is still the risk of the user's email being compromised. If a hacker can get into a user's email, they can use the SWA form to request the forgotten password and use the link in the email to set the password and then use the user's account. SWA stores the IP address of every login which could be used to track down the hacker.

4.2.7 Protecting User Data and the System

The user is placing their trust in the system that it will protect their data from other users and outside hackers. The SWA software does not inherently trust the user and takes precautions to protect the system from potential security risks.

There is a "hard" limit placed on the maximum file size that a user can upload to the server which is currently 10 MB.. The file upload routines use the standard security measures provided by the PHP libraries. Several additional precautions are taken to ensure that the uploaded file is protected. The file is uploaded to a temporary directory. A directory is created for each job submitted by the user and the data file is moved to a unique filename (e.g. `/path/user/uniqJobName/data.input`). This helps eliminate the security risk of a user submitting a job with a filename `` `mail hacker@bad.com < /etc/password` ``. **NOTE:** The original filename that the user put in the HTML form is saved in a database table that stores all information about the job being submitted. The files that are uploaded and generated are stored in a directory structure that is not web accessible. The only way to get files into that directory structure is for SWA to put, generate, or get them from the protected directory area.

4.3 User Account System

The user account management component allows a user to register for an account, update their information (password, email, organization), and reset their password if they forgot it. It also allows an administrator to allow users to self-register or require user registrations to be reviewed and activated. The component provides the ability for the user to login and use the protected applications and logout from the system. These HTML form pages were easy to implement using JavaScript and PHP libraries to implement the Security component and the page template mentioned in the section, Protecting Web Pages.

4.4 Web-Based User Interface

The user interface was designed to help the user put in only the information they need to submit a new job or view a job's current status or a previous job's history. It was designed to be intuitive for users of common sequence analysis web-based tools, but has the added value of allowing for large-scale sequence analysis tasks.

4.4.1 Submit New Job

The SWA user interface makes it very easy for users to submit a new job by specifying the job title, BLAST program type (e.g. Protein-protein (`blastp`)), file to upload, output format (e.g. tabular text vs. conventional results), and database to use. The job title is used when sending email to the user to help them identify which job is being referenced. It is also used to help the user reference a job from the "Job History" web page.

SWA helps the user by dynamically changing the default values in the user interface based on which program type is selected. For instance, if blastp is selected, the databases are limited to the selection of protein databases available. This reduces the possible errors that could occur if the user selects the Protein-protein program and a Nucleotide-Nucleotide database. The user also has the ability to override the default ScalaBLAST options such as filtering, word size, probability matrix, alignments, descriptions, etc. The user submits their job and is taken to the "Job History" page rather than an output view page because the jobs can take hours or days. The SWA does not assume any default values for the program type, the database, or output type. An invalid assumption by the program could result in a significant delay for the user and wasted CPU cycles on a large multiprocessor system or cluster.

All error checking is done in JavaScript on the client side to avoid round-trip delays. As an extra level of protection, the same error checking is done on the server side in case a hacker figures out how to submit the form bypassing the JavaScript checks.

4.4.2 Job History

After a job has been submitted, the SWA takes the user to the "Job History" tab on the user interface. The user can see all their input values and the page has a simple timer that refreshes the page every N minutes where N is computed based on the size of the job. If the number of queries in the job is small, the user interface refreshes every 5 minutes. If the number of queries is large and the job is estimated to take several hours or days, the user interface refreshes every other hour. If the output file exists when the page is refreshed then a link is provided for the user to download their output. The job history page allows the user to select any job that they have previously submitted and view all the input parameters to ScalaBLAST. In addition, metadata for the job is available such as the time/date the job started and finished, how many

queries were in the input file, the standard output and standard error messages from ScalaBLAST, and when the output was retrieved by the user.

4.5 Server Side Processing

4.5.1 User Environment

One significant problem that we encountered was having an Apache 2.X web server running from the user account, apache, with minimal permissions. Most system and web administrators try to limit the capabilities of this account running the web server.

We had to configure the web server so that the apache user could create directories, files, and run programs from PHP. In addition, the compute nodes in the cluster had to share a common directory structure so that jobs could be distributed to many compute nodes and still have them access the user input file and create output files. The ScalaBLAST application parses the input file and creates N input files, one for each compute node. Each compute processor creates a single output file.

The controlling script that starts the job makes calls to job scheduling software and other legacy applications that use environment variables. To minimize the impact of future upgrades to the Apache web server, we assumed that the apache user account does not have a user environment or environment variables. Environment variables necessary for the software applications to run are embedded in scripts that are generated by the SWA.

Another issue that we encountered was the need for developers to test their code. The problem was that the files created by the SWA were owned by the apache user and apache group. The solution we chose was to add the developers to the apache group and make all directories and files group writeable.

4.5.2 File Management

Though many aspects of file management were addressed in the security sections, there is an additional need to clean up data files after the user has downloaded them or after they have "expired". Files that have completed and are older than two weeks are deleted by a cron job that runs nightly.

4.5.3 Script Generation

The approach used to provide an interface between the web application and submitting a ScalaBLAST job was to automate what a ScalaBLAST user would do. To protect the user from accidentally submitting the same job multiple times the web application redirects the user to the job history page. If the user goes back to the new job page, the values are now empty. A directory is created for each user that submits a job. A unique directory is created for each job that is submitted under the user's directory. The web application uses PHP to generate several files:

- A shell script to execute, send email to the user that the job has started, and submit the job to the Portable Batch System (PBS) (OpenPBS, 2007) queue.
- A PBS script with directives for the number of nodes and processors to use.
- A shell script that monitors the job directory and job queue for output and error files created by PBS, packages the output files, gather runtime statistics, update the job history database, and send email to user that the job has finished.
- A PERL script to calculate the statistics about the job.

The reason for multiple scripts is because the compute nodes do not have all the necessary programs such as email or PERL or MySQL.

4.5.4 Miscellaneous

The deployment goal for SWA was a production level service that user's can rely on while still making it easy to understand and use. Part of the effort was implemented with policy decisions such as not using default values in web forms. This provided an additional benefit of minimizing automated bots from creating bogus accounts.

A utility library was written to make it easy for the developers to use utility function calls instead of PHP function calls. The call to `getSelf()` was used rather than `$_SERVER['PHP_SELF']`. This made it easier for web page designers to use a few simple functions rather than having to learn and understand PHP programming practices.

A function was written to let developers hide email address from spam harvesters (Raz, 2007). Page designers were able to use the PHP function getMailTo() on a web page with an argument of a real email address without worrying about it being harvested. Since the function executed on the server side before downloading the page, it generates a JavaScript function in the page with a random key and a function call to the function using the random key. When the end user viewing the web page moves their mouse over the link generated by getMailTo(), they see the actual email address of the user. If the link is clicked, the browser performs the normal function of generating an email message. If the user viewed the HTML source, the embedded

JavaScript and function call makes it more difficult to find the email address. Since the key to encrypt and decrypt the email address is randomly generated each time the page is loaded, spam harvesters find email addresses.

4.6 Future Work

Enhancements could be made to SWA that would provide more context-sensitive help and examples. As more data is gathered from actual users we may need to enhance the user interface and job history to allow users to run previous jobs again or we may need to modify default values used by ScalaBLAST. When more features are added to SWA which complicated the design, we may revisit using a more extensive development framework like Drupal. In addition, we are expanding this approach to other high-performance applications in bioinformatics and computational biology, such as a high-performance peptide identification tool called Polygraph (Cannon et al. 2005) that will run from the same web portal on the same cluster. Because of the extensibility of this framework this addition will be accomplished in a fraction of the time.

5 CONCLUSIONS

The SWA system provides secure yet public web-based access to a high-performance sequence alignment tool, ScalaBLAST. It was written to be extensible and flexible while staying consistent with BLAST applications currently in use. This project used innovative design to integrate many pieces of existing technology such as PHP, MySQL, JavaScript, job launching, job monitoring, user-notification, file management, et al. This system is fully operational and can be found at http://www.biopilot.org.

ACKNOWLEDGEMENTS

The research described in this paper was supported in part by the US Department of Energy, Office of Advanced Scientific Computing Research through the "Data Intensive Computing for Complex Biological Systems" project at the Pacific Northwest National Laboratory, a multiprogram national laboratory operated by Battelle for the US Department of Energy under Contract DE-AC06-76RL01830. The authors would like to thank Leigh Williams for helping design the web page template and working with scientists to generate most of the public content on the web site.

REFERENCES

Altschul, S., Gish, W., Miller, W., Myers, E. & Lipman, D. (1990). "Basic Local Alignment Search Tool", J. Molecular Biology. 215, p403-410.

Altschul, S., Madden, T., Schaffer, A., Zhang, J., Zhang, Z., Miller, W. & Lipman, D. (1997). "Gapped BLAST and PSI-BLAST", Nucleic Acids Research. 25, p3389-3402.

Cannon, W., Jarman, K.H., Webb-Robertson, B., Baxter, D., Oehmen, C., Jarman, K.D., Heredia-Langner, A, Auberry, K., Anderson, G. (2005). Comparison of Probability and Likelihood Models for Peptide Identification From Tandem Mass Spectrometry Data. J. Proteome Res. 4(5), p1687-98.

Harvard. (2007). BatchBLAST: A Java Software With Graphical User Interface To Blast Multiple Sequences Against Multiple Databases In Batch Mode [online]. Available: http://www.hip.harvard.edu/informatics/programs/JAVA%20BLAST%20Parser.html [Accessed 15 October 2007].

NCBI - National Center for Biotechnology Information. (2007). BLAST: Basic Local Alignment and Search Tool [online]. Available: http://www.ncbi.nlm.nih.gov/BLAST [Accessed 23 August 2007].

NuSphere. (2007). PHP IDE - Integrated Development Environment for PHP [online]. Available: http://www.nusphere.com/products/phped.htm [Accessed 15 October 2007].

Oehmen, Christopher & Nieplocha, Jarek. (2006). "ScalaBLAST: A Scalable Implementation of BLAST for High-Performance Data-Intensive Bioinformatics Analysis". IEEE Transactions on Parallel and Distributed Systems. 17 (8), p740-749.

OpenPBS. (2007). OpenPBS: Portable Batch System [online]. Available: http://www.openpbs.org/ [Accessed 15 October 2007].

OpenQA. (2007). OpenQA: Selenium [online]. Available: http://www.openqa.org/selenium [Accessed 15 October 2007].

Raz, Uri. (2007). How do spammers harvest email addresses? [online]. Available: http://www.private.org.il/harvest.html [Accessed 15 October 2007].

Source Viewer. (2007). Source Viewer [online]. Available: http://source-viewer.softswot.qarchive.org [Accessed 15 October 2007].

VanDyk, John K. & Westgate, Matt. (2007). Pro Drupal Development. New York: Apress. p1-10.

Washington University in St. Louis, Genome Sequencing Center. (2007). GSC: BLAST Server [online]. Available: http://genome.wustl.edu/tools/blast [Accessed 15 October 2007].