Galaxy based BLAST submission to distributed national high throughput computing resources

Soichi Hayashi[1]
Indiana University
2709 E. Tenth Street, Bloomington, IN 47408-2671
hayashis@iu.edu

Sandra Gesing
University of Notre Dame, sandra.gesing@nd.edu

Rob Quick
Indiana University, rquick@iu.edu

S. Teige
Indiana University, steige@iu.edu

Carrie Ganote
Indiana University, cganote@iu.edu

Le-shin Wu
Indiana University, lewu@iu.edu

Elizabeth Prout
Indiana University, echism@iu.edu

To assist the bioinformatic community in leveraging the national cyberinfrastructure, the National Center for Genomic Analysis Support (NCGAS) along with Indiana University's High Throughput Computing (HTC) group have engineered a method to use the Galaxy to submit BLAST jobs to the Open Science Grid (OSG). OSG is a collaboration of resource providers that utilize opportunistic cycles at more than 100 universities and research centers in the US. BLAST jobs make a significant portion of the research conducted on NCGAS resources, moving jobs that are conducive to an HTC environment to the national cyberinfrastructure would alleviate load on resources at NCGAS and provide a cost effective solution for getting more cycles to reduce the unmet needs of bioinformatic researchers. To this point researchers have tackled this issue by purchasing additional resources or enlisting collaborators doing the same type of research, while HTC experts have focused on expanding the number of resources available to historically HTC friendly science workflows. In this paper, we bring together expertise from both areas to address how a bioinformatics researcher using their normal interface, Galaxy, can seamlessly access the OSG which routinely supplies researchers with millions of compute hours daily. Efficient use of these results will supply additional compute time to researcher and help provide a yet unmet need for BLAST computing cycles.

1

Speaker

The International Symposium on Grids and Clouds (ISGC) 2013
March 23-28, 2013
Academia Sinica, Taipei, Taiwan

1. Introduction

1.1 BLAST

BLAST (Basic Local Alignment Search Tool), is one of the most common tools in the field of bioinformatics for conducting similarity searches of DNA and protein sequences [11]. The BLAST program attempts to quantify the similarity of a query string to a subject string using a heuristic approach.

Assessing sequence similarity is critical in bioinformatics - in order to find potential instances of a known gene in an unknown genome, to annotate potential genes with an unknown sample against a known genome, to infer gene homology between two samples, and to test the strength of a transcriptome assembly with a search for highly conserved sequences are a few examples. BLAST approaches the problem of sequence similarity and pattern search with a seed-and-extend algorithm, in which a short match (called a word) must occur with a score greater than a certain threshold; the sequence in both directions from the match is then extended [12].

1.2 NCGAS [14,17]

The National Center for Genome Analysis Support is an NSF-funded organization dedicated to providing bioinformatics consultation, infrastructure, and training to a national audience. With expertise in RNA-Seq and gene expression analysis, proteomics, meta-genomics-transcriptomics, software installation and development, and techniques for studying novel organisms, NCGAS is set to cover a wide range of support options for biologists in the field of genomics. NCGAS offers its services to NSF-grant-funded research in exchange for acknowledgment of support in publications, though opportunities for more intensive consultation and authorship are available.

1.3 Galaxy [8, 9, 10]

The Galaxy project is an open, web-based graphical user interface for data-intensive biomedical research developed and maintained by Penn State and Emory University funded in part by NSF, NHGRI, and the Huck Institutes of the Life Sciences. Galaxy is a flexible Python framework designed as a toolbox. An administrator of a Galaxy instance provides the configuration files for the available command line tools or web services. Users can select, parameterize and invoke such tools and web services via pre-configured input masks and combine them via drag-and-drop mechanisms to a workflow. Galaxy is widely used in the biomedical community and increases the usability of available applications via an intuitive user interface. Thus, it flattens the learning curve for the data analysis. NCGAS provides a Galaxy instance as an interface for access to its high-performance computing environment.

1.4 Use and user demand

The technologies for analyzing and creating data in the field of genomics and proteomics have been enhancd over the last decades and the increase of available data is reflected in publicly available databases. Figure 1 shows rapidly growing number of base pairs and users in recent years published by NCBI and the National Institutes of Health.

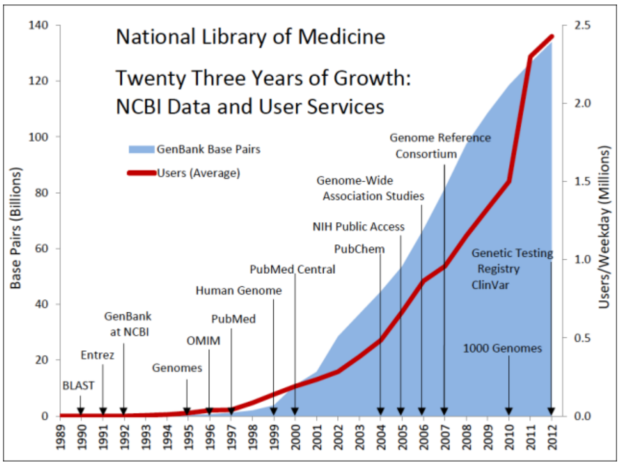

Figure 1 - Growth of the National Institutes of Health, National Library of Medicine [13]

In the past, bioinformatics researchers have tackled the issue of increasing data volume by purchasing additional resources or enlisting collaborators doing the similar type of research. However, we expect that it will be difficult to sustain the necessary increase of resource capacity to meet the research goals simply by traditional means explored thus far.

The Open Science Grid (OSG) [5] is a collaboration of resource providers at more than 100 universities and research centers in the US and abroad. OSG allows member institutions to share each other's computing resources to solve computationally challenging problems by joining heterogeneous cluster systems into a single coherent grid computing platform. OSG also allows researchers to "opportunistically" use OSG computing resources that are not in use by the owners at a given moment.

OSG currently provides about 2 million CPU hours daily. Although most OSG resources are used to run programs submitted by the resource owners ("contracted" use), however, roughly 1/10 of total CPU hours are consumed by job submitted by researcher who does not own any resources themselves. This is due to various resource owners allowing their unused computing cycles to be used by someone else, namely people from the "OSG" virtual organization.

Any researcher from any part of the world can join the OSG virtual organization and, thus, can benefit from such unused computing cycles.

The HPC cluster behind the NCGAS Galaxy instance is a large-memory machine with 512GB of RAM and 32 CPUs per node. BLAST can account for nearly the entire monthly resource usage for Galaxy during spikes. These jobs could be run more efficiently on OSG, leaving room for users with large-memory jobs.

1.5 OSG vs. local resources

Most BLAST applications are CPU intensive, and are usually bottlenecked by lack of enough CPU resources. Backend clusters currently in use by NCGAS Galaxy such as Indiana University Mason cluster are designed to handle high-memory applications such as genome assembly, or large-scale phylogentic applications which require large amounts of memory as the size of the input increases [7]. Such systems tend to perform poorly due to a lack of enough CPU resources, and and the optimization of BLAST runs is beneficial for all applications, whose execution would be hampered by extensive BLAST runs.

OSG's opportunistic computing cycles can provide thousands of CPU cores at a time, thus properly distributed BLAST jobs can then be efficiently executed on OSG with no additional cost to the bioinformatics researchers.

2. Methods

2.1 Galaxy functionality

Galaxy provides a web-based user interface to available tools and web services with pre-configured input masks for parameters and input data. The entire history can be exported as a “workflow”, or an automated version of the tools that were run. This allows replicating the entire process, perhaps with new and subsequent sets of data as the project matures. Workflows can be published and shared, capturing the results of an analysis in a concise, transparent and reproducible way. Additionally, Galaxy supports building workflows outside the history panel via drag-and-drop by combining tools without the need to first run the jobs.

The Galaxy web application can be run in multiple threads that divide the responsibilities of the software into logical tasks and lighten the load from a single large application. A job is sent to an internal component called a job handler that deals with requests to create or manipulate Galaxy jobs. A Galaxy instance can be configured to support local jobs, various batch systems or cloud systems via related job handlers.

A large memory or computational job, such as genome assembly, would take all available system resources from the local host and Galaxy would become unresponsive to other user requests. In this case, it is desirable to have a job runner that can export the job to run on a cluster. The tool configuration specifies which parameters are passed to the job runner to be used when submitting the job, such as resources requested, the destination queue, environment setup and any flags that may be required. Job runners exist for PBS/Torque, Grid Engine via DRMAA, HTCondor, local runner, a command line based job runner, and a lightweight runner with a client-server model. For BLAST on OSG, we chose to extend the command line job runner to include staging files and to send jobs to osg-xsede where a dedicated osg-blast workflow submission system is installed.

The command line job runner provides a plugin for connecting to a PBS/Torque environment which we extended to be compatible with OSG. Instead of using qsub/qstat/qdel commands that are part of PBS/Torque, we defined a set of very simplistic shell scripts to pass information between OSG and Galaxy.

In the OSG plugin, the process ID of the osg-blast workflow instance is stored in a file in the remote working directory and used to stop a job when it is deleted from Galaxy. When the job handler brings up an active job, it will send a request for that job’s state.

When a user clicks the execute button on the Galaxy’s tool page, the newly created job is added to the list of active “watched” jobs. When the job ends, it is popped off the list and no longer tracked. The job handler requests the status of a job by querying the job runner associated with that job. The job runner implements a status check function. The OSG plugin makes a call to the osg-xsede server requesting the status of a job given a job ID - the server looks up the status of that job and returns one of the following:

- TESTING - The job is currently in testing state, meaning a small number of queries are being run to calibrate the distributed job with an ideal configuration
- RUNNING - The queries (and potentially the databases) have been distributed and the job is running on the OSG.
- FAILED - An error occurred on the server. This will be marked as a failed job in Galaxy.
- ABORTED - The job was aborted on osg-xsede. This will be marked as a deleted job in Galaxy.
- COMPLETED - The BLAST job has finished, but the results have not yet been merged.
- SUCCESS - The results are merged and ready to be staged back to Galaxy.

2.2 HTC-enabling BLAST & osg-blast prototype

We started working on osg-blast workflow submission system prototype in the spring of 2013. The initial goal of this prototype was simply to determine if we could run BLAST in the high-throughput computing environment using a small input queries (70k fasta queries) and NCBI's NR/NT databases. We have hand picked several OSG sites which allowed access for OSG virtual organization and provided necessary disk space (10-20G) and memory (>2G) and enough available slots (or CPU cores). After staging NR/NT databases on our submit host (osg-xsede.grid.iu.edu) with databases downloaded from NCBI's public FTP server (ftp://ftp.ncbi.nlm.nih.gov/blast/db/), we installed the databases to each OSG site on OSG_DATA partitions. The NCBI tool makeblastdb converts genomic sequences into a BLAST database. makeblastdb can split the BLAST DB such that each partition is about 1G compressed. BLAST can natively run queries on all database partitions or on a specific partition. This capability can be used to distribute BLAST jobs by running the same set of input queries on all database partitions and later merging all output files into a single output file by grouping hits for each input query sorted by e-value for all results.

"e-value" generated by Blast as part of its output reflects the probability to find an input query in given random gene sequences. It is an important indicator of the statistical importance for each search hit. e-values in blast results depends on the size of the database among other variables, therefore, in order to correct the e-value calculation while searching on a single database partition, we must specify "-dbsize" parameter on blast command to be the total size of the entire database. The total size of the database can be found inside .pal or .nal file generated by makeblastdb tool depending on the type of database specified (either neucleotide, or protein.)

We then created python scripts which 1) split the input queries into multiple blocks with no more than 5000 fasta sequences each 2) generate a HTCondor workflow to submit jobs for each input blocks (block 1 - block 14), and for each input blocks queue jobs for all database partitions (nr.00 - nr.16, see Figure 2).

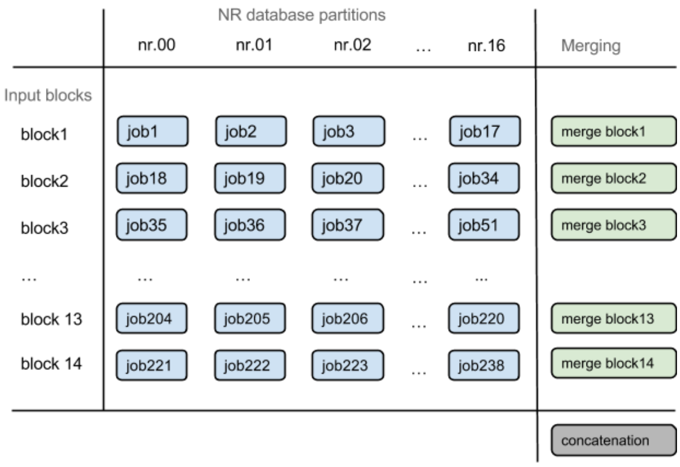

Figure 2 - Distributed BLAST workflow

The generated HTCondor workflow also merges outputs from each input blocks as soon as all jobs pertaining to a specific input blocks completes. Then, when all block merging is completed, all merged outputs are concatenated into a single output file.

The result was validated by executing BLAST locally and running all input files against all database partitions at once. The output from OSG matched as expected the output from the local execution except for some statistical parameters related to the database size, as well as some extra hits with higher e-values most likely due to blast being submitted against partitioned database instead of as a whole.

2.3 OSG crawler and installing BLAST databases on OSG

Our next step was to identify other OSG sites where we can install BLAST databases, and install them automatically if possible. For this, we have written a script which crawls all OSG sites and gather following information.

- Does the site allow execution from OSG virtual organization?
- Does the site provide OSG_DATA partition and do I have write access to it?
- Does the site provide OS / library to allow execution of BLAST ?
- Is BLAST database already installed?

Installation of BLAST databases on various OSG sites turned out to be more difficult than anticipated, due to lack of consistency among OSG resources. For example, some sites include VO name as part of OSG_DATA (i.e. /app/osg) and some does not. Some sites allow world-writable access but others require us first contact the site admin to create a VO specific subdirectory before we could install databases. Some sites had OSG_DATA available on CE but not on WN, and some sites even published different OSG_DATA locations for CE and WN. These issues were exasperated by some site admins claiming that these issues were intentional rather than misconfigurations. Many sites are often used by other high-priority / contracted jobs that we had to repeat the installation workflow over the course of several weeks in order to complete installation on most sites. Another important issues was that OSG only recommends each site to provide up to 10G of disk space for OSG_DATA. Since NR/NT databases are about 10G each; compressed, we found many sites where we could barely install a single BLAST database. We hoped to have at least half a dozen such databases available on most OSG sites.

These challenges were compounded by the fact that NCBI updates their BLAST DB regularly (a few times a month), and the size of BLAST DB is constantly growing as we mentioned earlier. Thus, installing the BLAST databases on OSG_DATA or OSG_APP was not feasible nor robust enough for our use case, which led us to explore an alternative.

2.4 Distributing BLAST database via wget / squid

Another way of making the BLAST DB accessible for each job was to download a part of database used by each job via “wget”. This method solves the problem of frequently updating the BLAST database across OSG sites and keeping them all synchronized. Additionally, it reduces the disk space required to run BLAST on each cluster, at the expense of increased network traffic at the beginning of each job execution.

In order to lessen the amount of network traffic required, we have decided to rely on squid proxies. Squid proxy works as a cache between our submit host and each WN, and they are commonly installed on most OSG clusters. In theory, the same database partition will only be downloaded once per each cluster, and all WNs inside the cluster will share the same database partition from the cache instead of downloading from our submit host individually.

The wget / squid approach worked well for the most part. However some clusters were often misconfigured or not operational although the clusters advertised that the squid servers were available. Thus, the job script needed to test the squid server before using it. Some site also had an extremely low throughput between their squid server and the rest of the Internet, due to how their campus network was configured. We even had to blacklist some sites from executing our BLAST workflow altogether.

2.5 Clustering jobs that simultaneously use the same database partition

During the previous phase of prototyping, a critical issue was discovered with the way we were submitting our jobs. As we have depicted in the diagram (Distributed BLAST workflow) above, we wanted to simultaneously submit all jobs that belongs to the same input blocks across different database partitions. This way the workflow can start merging outputs as soon as all jobs pertaining to the single input block have been completed.

However, this meant that each job running in OSG will most likely require different database partitions as the workflow progresses. In this situation, squid proxy will cause a high rate of cache-miss, and the throughput of the entire cluster will suffer. In one case, combined with sub-optimal configuration of the squid proxy, one large cluster was completely rendered offline due to the overload of their proxy server, which was used as a key component of their infrastructure.

We needed a way to ensure that jobs that used the same database partition would somehow cluster together in order to reduce the chance of cache-miss. Currently, neither HTCondor nor glidein provides any capability to hint certain jobs to be submitted together at or near a particular cluster, however, we could accomplish a similar effect simply by submitting and queueing jobs close to each other that requires the same database partitions. In other words, instead of submitting jobs grouped by each input block, we decided to submit jobs grouped by each database partition.

This simple change solved our squid problem. However, it postpones the merging of the results for each input block until almost all jobs are completed. Previously, merging of output files could happen concurrently while jobs for other input blocks were still running. Since merging of the output files needs to be done on the submit host, for large jobs with tens of thousands of output files, this new submission approach would add a nontrivial amount of CPU / disk IO load on the submit host and add extra few hours of processing time at the end of the workflow.

2.6 Input block size

In order to implement a generic BLAST workflow submission sub-system that can be invoked by another system such as Galaxy, it is important to understand that we can not rely on submitter to provide any technical configurations or provide any hint as to how long each query will take to execute or how much memory and CPU resources are required.

We have observed that, even with the identical input query and database, execution time and resource requirement of the job would increase dramatically simply by making a few adjustments on BLAST input parameters (such as -evalue or -max_target_seqs). There is no known method to estimate resource requirement simply by observing the input query / database / BLAST input parameters before job submission.

Since we can not change the number of database partitions at runtime, the only parameter that we could adjust is the number of input query sequences per each input block. During the prototyping phase, we had to first determine for each run the optimal input block size by a trial and error approach such that each job would run for an approximate 1 - 2 hours and without exceeding the maximum memory available on most OSG resources (2 GB). The optimal block size could be as low as 10 up to over 10000.

If the input block size is too low, there will be too many jobs submitted with each job running only a fraction of a second with a lot of overhead wasted by queueing and staging the job. If the input block size is too high, then each job will either timeout and never complete, or use too much memory is and be preempted by the cluster's WMSs.

These experiences led us to believe that the only method to properly determine the optimal block size is to submit a few test jobs using a small amount of sample sequences (10-25) taken from the actual input queries, and measure their execution time, maximum memory used, etc.. We could then extrapolate the optimal block size from the test results. Running a test job could also avoid submitting thousands of jobs if all jobs are to fail simply due to user error in input parameter, or invalid input queries.

2.7 HTCondor and node-osg

Another unique aspect of creating a workflow sub-system is that, not only must we handle issues introduced by unpredictable user input, we must also handle issues caused by BLAST application itself and/or by OSG's heterogeneous cluster environments.

A BLAST application could fail for a variety of reasons. It could simply crash randomly (such as the infamous "NCBI C++ Exception", or a simple segmentation fault), it could die due to bad input queries, corrupted database, error with the BLAST engine, out of memory, etc. A job could also fail if an OSG site is not able to download the BLAST database due to a network issue, disk space or file permission issue, preemption by other higher priority jobs, or simply landing on a site where BLAST simply does not perform well for whatever the reason.

HTCondor provides features for error handling such as configuring the workflow to run only on sites that meets certain minimum resource requirements, or determining when to resubmit a job v.s. abort the entire workflow given a range of return code from the job wrapper script and a complex set of logic statements. However, we needed a much more robust workflow management / error handling functionality that could be easily implemented by us instead of relying completely on features provided by HTCondor.

For example, we needed a feature such as counting the number of job failures on a particular site, and if the count reaches a certain threshold, modify the HTCondor submit option at runtime to prevent subsequent jobs from being resubmitted on that site and automatically avoid submitting jobs for other users until OSG operations group can diagnose and troubleshoot the issue.

In order to acconmplish this, we have implemented our workflow system using Node.js based HTCondor wrapper called node-osg. Node.js is a programing platform built on Google

Chrome's V8 JavaScript runtime. Node.js's event-driven, non-blocking I/O model allows us to create applications that can handle real-time, data-intensive tasks easily and efficiently [16]. node-osg interfaces with HTCondor command line tools and monitor HTCondor job logs and fire Node.js events, which will allow us to implement necessary dynamic, event-driven workflow capabilities.

With HTCondor's native capabilities augmented by node-osg, we then implemented our first production BLAST workflow submission system.

3. osg-blast (v2) production workflow submission system

3.1 OASIS to distribute BLAST database

OASIS (OSG Application Software Installation Service) is an instance of CernVM File system (CVMFS)[6] developed by CERN. CVMFS is widely used by EGI community and many other grid infrastructure communities around the world, such as WLCG (worldwide LHC computing grid) and WeNMR (worldwide e-infrastructure for NMR and structural biology.) CVMFS provides a scalable, read-only, distributed software distribution system through multi-tiered web servers and squid caches. OSG VOs are enabled to upload their software via OASIS and the software will become automatically available on all clusters that currently have OASIS mounted on their CE/WNs.

Although OASIS was designed to distribute applications, not data, we have decided to use OASIS to host our BLAST databases for the following reasons:

- It allows us to easily add / update various BLAST databases commonly used by the bioinformatics community. Updated database will be propagated to all OASIS mounted clusters almost immediately.
- CVMFS is highly scalable, supported by CERN, and widely adopted by major OSG clusters available through osg-xsede submit host.
- OASIS provides us enough disk space to provide several dozen large BLAST databases.
- Squid cache used by CVMFS is a required part of OASIS installation, and the fact that a site has adopted OASIS often means they have a functioning, well-configured squid server; contrary to experience gained in section 2.4.

On the OASIS login host, where we publish our BLAST databases, we have written a set of scripts to download databases published by NCBI, which comes pre-partitioned into roughly 1G in size and do some minor restructuring for osg-blast v2. Currently, we publishes about a dozen popular databases published by NCBI and Flybase. Anyone with access to OSG OASIS can access these databases outside of osg-blast.

osg-blast v2 will submit jobs with various HTCondor requirements ensuring that the jobs are submitted on a site that supports OASIS. Each time we update our database, or add a new database, we will update the REVISION ID so that the job will only run on site that has fresh content propagated (it usually take about a day or so until most OASIS sites has a fresh content available).

Although OASIS is widely available via osg-xsede submit host, it is still in the early adoption phase and not all OSG clusters have OASIS mounted on them. Also, OSG currently recommends each site to have at least 20G of shared squid cache space. This means that, a single BLAST workflow with a large database (such as NT database with 17 partition with 300M-800M each) could almost invalidate the entire squid cache by the time this workflow finishes. Our experience so far is quite positive, however we will need to monitor our usage and determine if OASIS should be used for our BLAST workflow in the long term.

3.2 osg-xsede / Glidein WMS [15]

We have decided to use the osg-xsede submit host to stage our BLAST workflow submission system. osg-xsede is "glidein enabled" submit host. Glidein WMS is a high level workflow management system developed by the USCMS VO, which provides a glue between various OSG clusters and a submit host, such that any job submitted on the local queue will automatically be resubmitted to available OSG resources where our job can be executed immediately.

Glidein WMS identifies unused computing cycles from various OSG clusters on osg-xsede pools using the OSG VO, which is a special VO dedicated to opportunistic use of OSG resources. osg-xsede allows us to simplify our task of identifying available resources in OSG and submitting our jobs manually to each cluster.

The number of job slots available on osg-xsede therefore differs from moment to moment, depending on amount of opportunistically available resources in OSG, and number of jobs currently queued locally at osg-xsede. We typically see 5k-10k job slots (running jobs) available on osg-xsede, and therefore osg-blast v2 workflow would run up to 5-10k jobs in parallel if all available slots are allocated to osg-blast jobs.

3.3 User Database

In addition to the use case of applying a common database like NR/NT, user-defined databases are also supported by osg-blast. Since a user database could be different for each job submission, hosting it on OASIS does not make sense in this case. Instead, user provided fasta sequence will be converted to BLAST database in an appropriate format, then copied to our submit host's public_html directory so that each BLAST job can then download it through local squid proxy. For each user database, md5 sum is created for security reasons and used as directory name containing the user database to obsure URLs. Large user databases will be split into partition just like the OASIS hosted databases.

3.4 osg-blast v2 overview

osg-blast loads a BLAST configuration and starts submitting jobs using the node-osg module, which interfaces with HTCondor command line tools. osg-blast will block until the workflow is finished, failed, or aborted by an external user either via command line or via Galaxy.

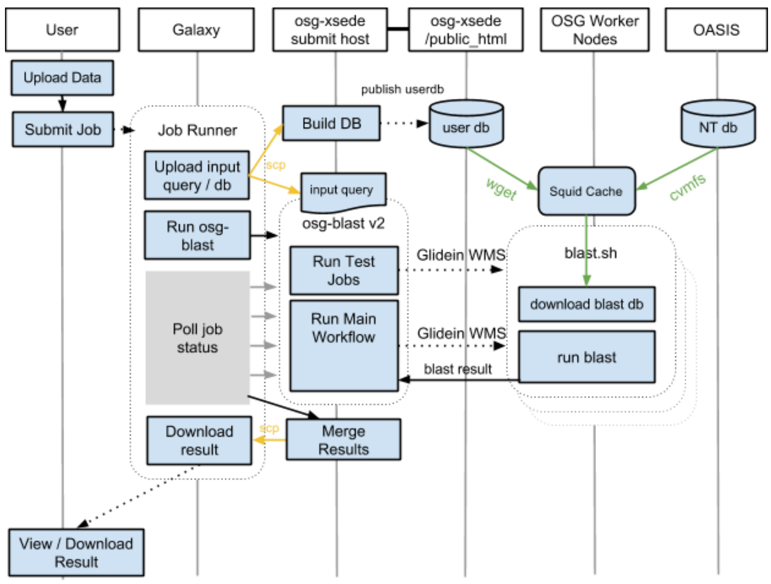

Figure 3 - Overview of osg-blast workflow system

A script staged and executed by Galaxy's job runner is responsible for building and publishing databases using a script provided by osg-blast v2. As above sample URL shows, we use /local-scratch/public_html to publish user databases through the apache server running on osg-xsede submit host.

"blast" and "blast_opts" are the BLAST command and BLAST options provided by the user via the Galaxy UI for osg-blast.

osg-blast v2 executes in 2 main stages; testing and main workflow stage. The goals of testing stage are: 1) to make sure user has provided a correct input file, a database if applicable, and input parameters; 2) analyze the resources required to execute the search, and the time it takes to process each input sequences. osg-blast v2 will currently submit 5 different test jobs, using 25 input queries each, and on different database partitions.

Once all test jobs are completed, execution time and other resource metrics reported by HTCondor can be used to calculate the optimal input block size for the main workflow. The current implementation of osg-blast v2 only takes into consideration the average execution time and it then extrapolates the best block size such that each job of the main workflow will be executed for roughly 90 minutes.

We have created the node-readblock module, which can read a large file delimited by some token. We have used this module to load input fasta sequences which is delimited by newline character followed by ">" and split it into separate files according to the block size determined during the test stage. For each jobs submitted, osg-blast v2 will use symlinks to each input block in order to save disk space locally at the submit host.

osg-blast v2 uses the same strategy we used in our prototype in order to correct e-value calculations and subsequent merging of the output files. We currently provide a merging script for XML output format and tabular output format. In the future, we intend to provide a merging script for most BLAST output formats. A script staged and executed by the Galaxy job runner is responsible for executing an appropriate merging script since the user selects the output format in Galaxy.

3.5 osg-blast v2 error handling

osg-blast sends the user-specified BLAST executable as well as a wrapper script, which tests local execution environment, downloads / copies and uncompresses an OASIS or a user provided database, and finally executs BLAST on the WN. The wrapper script is also responsible for detecting various error conditions, and relaying the exit code from BLAST executable. For most HTC applications, the wrapper script usually makes decisions as to whether or not certain error conditions are fatal, or recoverable. For osg-blast, we wanted such a decision to be made by the osg-blast application based on various contextual information available to osg-blast. The standard return codes of the wrapper script has been extended with a large number of different return codes for each distinct event and error condition.

We have implemented various high level workflow logics such as following.

- Terminating the workflow if the total number of resubmission across all test jobs exceeds certain number of times.
- Outputting a detailed error report to be delivered to OSG operations group if a job was held with unknown subcode.

- If jobs are held during the test stage, abort the workflow.

The currently implemented high-level workflow logics are a good start and the implementation is designed to be easily extendable. The design and the use of node-osg assures the scalability for additional logic to create a robust and reliable BLAST workflow submission system in the near future.

It is important to note that, HTCondor supports simple logic such as the keeping up with number of time each job is retried, aborting the workflow if it exceeds certain amount of retries, or adding an execution timeout and resubmitting it. We rely on HTCondor to implement such functionalities in order to avoid "re-inventing the wheel" and implement more complex logic on the workflow application.

3.6 monitoring & alerting

As part of OSG operations center; a group dedicated to handle communication and hosts various critical services for Open Science Grid, we have implemented a robust monitoring / alerting system, which consists of nagios, munin, rootmail, hardware alarms, and various service monitoring script installed throughout our infrastructure. All alerts will be sent to our alert bus, then to our alert processor which parses and analyze various alert messages and finally to staff's email inbox or are texted to a mobile device.

osg-blast v2 currently logs all critical events or unhandled held events encountered to a separate log file named gocalert.log. This log file is then monitored and any new content will be automatically sent to GOC alert bus. Once it is delivered to GOC alert bus, appropriate GOC staff will be notified and the problem will be troubleshooted.

osg-blast v2 generates workflow statistics as well as any exceptions or held events occurred on each cluster during the workflow execution. This information allows us to identify sites that are causing frequent issues or sites that becomes unsuitable for running BLAST. So far, we have already identified various OASIS and squid related sites issues as well as potential issue with Glidein WMS installed on osg-xsede using our monitoring & alerting infrastructure.

4. Results

Quarry is Indiana University's primary Linux cluster computing environment for research and research instruction use. It is commonly used by IU's bioinformatics researchers to run BLAST applications and, therefore, we have used it to compare and benchmark scaling performance between osg-blast and quarry

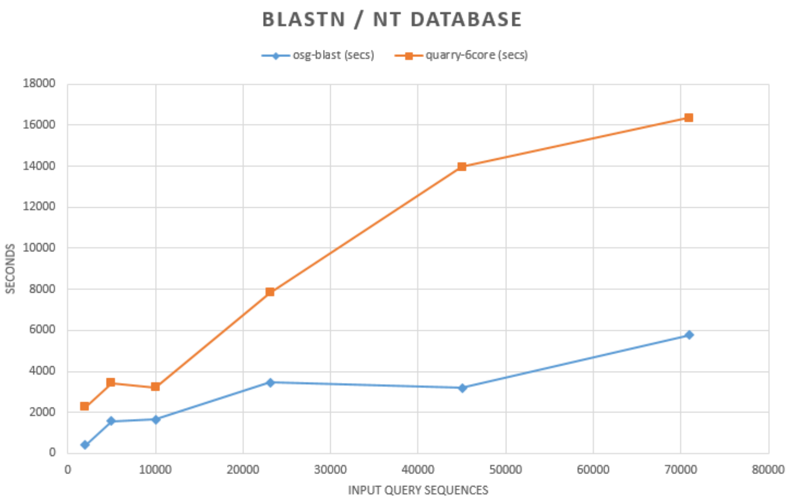

Figure 4 - Average execution time comparison between osg-blast and blastn on Quarry

Although it is difficult to perform an apple-to-apple comparison between BLAST on Quarry and osg-blast, Figure 4 shows what a typical user might see in term of the total execution time as a function of input query size. As the slope of the graph above for osg-blast indicates, osg-blast can scale better as the size BLAST query increases. However, due to the use of OSG's opportunistic computing cycles, osg-blast's execution time may vary depending on how many resources are available[2] at any given time.

5 Discussion

5.1 Galaxy / OSG interface

The Galaxy job submission interface currently lacks capabilities to display detailed status and progress information. A user must download the stdout/stderr log manually to find the current status and progress information.

Importing of data to Galaxy is very slow, and this needs to be improved so that users can stage input query / database more easily. We are currently discussing the use of tools such as bittorrent/sync in order to make the data import and export much easier and faster for our users.

We have noticed that Galaxy does not handle the stopping of uploaded jobs well. Currently, the staging step of files to and from Galaxy to osg-xsede run as a local process to Galaxy; if the Galaxy server is restarted, or if the job is stopped during this phase, there is no graceful way to handle it.

During the staging step Galaxy should check to make sure the job is not canceled by the user. The job should restart the staging process if Galaxy is rebooted. Currently, scp is the mechanism for staging files - this should be moved to gridftp or torrent.

Scp is single-threaded and does not provide partial download recovery - a more robust system would be able to recover from a connection error and achieve higher speeds for uploads and downloads.

Minor changes to the BLAST tool UI would allow advanced options and would change the name of the history item from "blastn on db" to a more informative title.

The debugging and troubleshooting of the Galaxy front end were greatly aided by loading the entire Galaxy server as a project into Eclipse. This represented a significant amount of work, but an IDE would be highly recommended for any development of a complex software suite. A multi-threaded web server that continuously passes control of the logic between python, javascript, cheetah, and system libraries can be cumbersome to track without the use of a debugging harness or development environment. Variables and data structures can be explored and breakpoints can be added to determine the flow of the program at runtime.

5.2 osg-blast v2

The use of OASIS for hosted BLAST databases and wget/squid from osg-xsede for user database could become an issue if the size of BLAST database continues to grow in the future. If this is the case, we could explore distributing our database via SRM although this approach will require a lot of manual data movements. Another possibility is to employ bittorrent/sync here as well in order to automate the database distribution and taking advantage of wn-to-wn connectivity.

[2] Users can visit http://osg-flock.grid.iu.edu/monitoring/condor/condor_31day.png to find out how many jobs are currently queued on osg-xsede; one of OSG's submit hosts for opportunistic jobs.

osg-blast v2 currently only uses processing time from test jobs to extrapolate the correct block size for main workflow jobs. We will update this algorithm so that it will also use other information such as memory / disk space used, and standard deviation of the execution times, in order to better calculate the optimal block size.

We will implement more custom workflow logic in order to implement different resubmission strategies and make the system more robust and error tolerant. For example, if a job keeps failing due to timeout or lack of memory, then we could split the input query in half for that particular input block and resubmit it individually at runtime.

Currently, osg-blast v2 can merge XML and tabular output formats. We will need to implement mergers for other output formats. One alternative strategy is to always use XML output format internally, but we will convert the final merged XML file into another common BLAST formats.

6. Conclusions

We have shown that the need for acquiring more computing resources for BLAST runs will continue to increase in coming years. By integrating OSG BLAST into Galaxy portal, existing bioinformatics researchers can take advantage of OSG's opportunistic computing cycles without retraining themselves to become HTC experts. Running BLAST on OSG is proven to be technically possible although running as a sub-system of Galaxy, it requires much higher robustness and reliability assured by a higher degree of engineering. In order to achieve this, we have shown various methods and approaches we took to find the right mix of technologies currently available within distributed-HTC community combined with our own event driven workflow submission system using node-osg and tapping into an existing OSG operations group's capabilities.

We have successfully implemented our first production version of Galaxy/OSG submission system and demonstrated its capability to execute BLAST using OSG's opportunistic resources that can exceed the performance of a typical BLAST execution system implemented with a campus cluster backend. However, we must continue investigating and incooporating new technologies that can provide us even better reliability and performance in order to cope with an expected growth of the BLAST database and input query size.

Acknowledgements

The authors would like to acknowledge the support of Bill Barnett, Tom Doak, and Rich LeDuc at Indiana University for their guidance and occasional bioinformatics lessons during this project. Ruth Pordes provided access to an audience to help vet this project in the form of the OSG Council. Other valuable contributions were made by Derek Weitzel at the Holland Computing Center at University of Nebraska Lincoln, Chander Seghal at Fermi National Accelerator Laboratory, Mats Rynge at the Information Science Institute of the University of Southern California, and the entire OSG Operations Center staff including Alain Deximo, Kyle Gross, Tom Lee, Vince Neal, Chris Pipes, and Michel Tavares.

References

[1] Jacques Lagnel, Costas S. Tsigenopoulos, Ioannis Iliopoulos, *NOBLAST and JAMBLAST: New Options for BLAST and a Java Application Manager for BLAST results*, Bioinformatics (2009) 25 (6): 824-826. doi: 10.1093/bioinformatics/btp067

[2] Daniel Xavier de Sousa, Sergio Lifschitz, Patrick Valduriez, *BLAST Parallelization on Partitioned Databases with Primary Fragments* [http://vecpar.fe.up.pt/2008/hpdg08_papers/4.pdf]

[3] Pruitt KD, Tatusova T, Maglott DR., *NCBI Reference Sequence (RefSeq): a curated non-redundant sequence database of genomes, transcripts and proteins* [PMCID:PMC539979] (http://www.ncbi.nlm.nih.gov/pubmed/15608248)

[4] TrEMBLstats Current Release Statistics [http://www.ebi.ac.uk/uniprot/TrEMBLstats]

[5] Open Science Grid / Gratia Accounting [http://gratiaweb.grid.iu.edu/gratia/]

[6] Buncic, Predrag, Jakob Blomer, Pere Mato, Carlos Aguado Sanchez, Leandro Franco, and Steffen Klemer. CernVM-a virtual appliance for LHC applications. In Proceedings of the XII International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT08). 2008.

[7] Stamatakis *A. Phylogenetic Models of Rate Heterogeneity: A High Performance Computing Perspective*. In Proc. of IPDPS2006, HICOMB Workshop, Proceedings on CD, Rhodos, Greece, April 2006.

[8] Goecks, J, Nekrutenko, A, Taylor, J and the Galaxy Team. *Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences.* Genome Biol. 2010 Aug 25;11(8):R86.

[9] Blankenberg D, Von Kuster G, Coraor N, Ananda G, Lazarus R, Mangan M, Nekrutenko A, Taylor J. *Galaxy: a web-based genome analysis tool for experimentalists. Current Protocols in Molecular Biology*. 2010 Jan; Chapter 19:Unit 19.10.1-21.

[10] Giardine B, Riemer C, Hardison RC, Burhans R, Elnitski L, Shah P, Zhang Y, Blankenberg D, Albert I, Taylor J, Miller W, Kent WJ, Nekrutenko A. *Galaxy:a platform for interactive large-scale genome analysis.* Genome Research. 2005 Oct; 15(10):1451-5.

[11] Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ. *Basic local alignment search tool.* J Mol Biol. 1990 Oct 5;215(3):403-10.

[12] Altschul, S. F., Madden, T.L., Schaffer, A. A. et al. (1997), *Gapped BLAST and PSI-BLAST: A new generation of protein database search programs*, Nucleic Acids Res., Vol. 25, pp. 3389–3402

[13] DEPARTMENT OF HEALTH AND HUMAN SERVICES. NATIONAL INSTITUTES OF HEALTH. National Library of Medicine (NLM). FY 2014 Budget

[14] LeDuc, R., Wu, L.-S., Ganote, C., Doak, T., Blood, P., Vaughn, M., and Williams, B. (2013) *National Center for Genome Analysis Support Leverages XSEDE to Support Life Science Research*. Proceedings of XSEDE 13, San Diego CA. 7/22/2013.

[15] [Sfiligoi2009] Sfiligoi, I., Bradley, D. C., Holzman, B., Mhashilkar, P., Padhi, S. and Wurthwein, F. (2009). *The Pilot Way to Grid Resources Using glideinWMS*, 2009 WRI World Congress on Computer Science and Information Engineering, Vol. 2, pp. 428–432. doi:10.1109/CSIE.2009.950.

[16] Node.js [http://nodejs.org/]

[17] LeDuc, R., Vaughn, M., Fonner, J.M., Sullivan, M., Williams, J., Blood, P.D., Taylor, J., and Barnett, W. (2013*) Perspective: Leveraging the National Cyberinfrastructure for Biomedical Research*,

Journal of the American Medical Informatics Association. Published on line 8/20/2013: doi:10.1136/amiajnl-2013-002059.