Midterm Deliverable

Summary of Project

In the beginning of the semester, Wilson came to us with a problem: How can we create a scalable solution capable of supporting up to hundreds of people using Sequence Server to query NCBI Blast databases. Moreover, Wilson needed us to also include:

1. A web client which can be used by educators and students to run the NCBI BLAST Basic Local Alignment Search Tool algorithm.
2. The client would specifically like for us to provide the wurmlab/sequenceserver local web front-end version of the BLAST tool, which delivers the user interface and analysis capabilities they require. In other words he needed us design this with the same BLAST interface the staff we're already used to.
3. Finally the client needs the hosted blast implementation to support multiple classes of at least 100 students all using the BLAST curriculum concurrently (courtesy of Wilson's slides), with an average run time of 2-3 minutes per job. As well as some assurance that the ~100 jobs will finish at approximately the same time.

Team Frankenstein aimed to create a 'classroom cluster' model, meaning that the teacher is the master node, and the class becomes a cluster of workers. Essentially the teacher will download a docker file containing sequenceserver onto a Cyverse master CCTools instance. Once the IP address is available and the teacher has selected a port number, this information will be shared with the students. The students download a docker work file on their local machine containing CCTools and the selected database, and then set their machine up to talk to the teachers master Cyverse VM. This process essentially creates the class as a set of workers, whom are able to submit jobs to the class.

Team Frankenstein's solution sets the stage for a completely offline solution, assuming the students and master machine have Docker already installed. This is a route that we have not explored, but wanted to make the user aware that this is an option.

Description of Project

Code Availability

Github Repository - https://github.com/GEABlast/frankenstein

DockerHub - https://hub.docker.com/r/jforstedt/sswq1

Full Report on Wiki - https://wiki.cyverse.org/wiki/display/A2/Midterm+Deliverable

Installing and Running Instructions

To begin this process, the master is going to have to create a Cyverse master instance, which can be a 'tiny1' size or bigger, although any 'tiny1' sized instance will suffice for this solution. Once the instance is up and running, the teacher needs to download the master docker file using the 'wget' command. Once the master docker file is downloaded, the professor can look at the IPv4 of the instance found on the Cyverse Launch page and share that with students via email, verbal communication, etc. The Master docker file will have downloaded SequenceServer, a test blast DB, and the CCTools 7.0.19 tools suite. It will operate over the SeqServer default port 4567, which must also be shared with the students via email, verbal communication, etc. The installation to download the docker file can be downloaded and followed here.

On the student side, they will go through a similar process that includes downloading the docker file which includes CCTools 7.0.19 and the test

DB. This process will be done on the local machine, and the professor must help them set up their machine to talk on the correct port/IP address. Once the class has all been hooked up to the same Master instance, they are now able to submit BLAST queries through the master instance which has sequence server, and thus a que, that jobs will be filed through. Submitted jobs will then be distributed across the classroom, using the classroom's local resources to accomplish jobs.

This essentially allows for a scalable solution that is as efficient as the resources on hand allow. Their are admittedly shortcomings to our solution that have potential for improvement. One aspect is that our benchmarking times in a live use case would be varied because the worker cluster efficiency relies on computational resources in the classroom. The more students we have as workers, the quicker the times and vice versa. Another shortcoming is that this solution is meant to use small databases; because we are using student local machines, synching large databases across the local wifi network may cause networking issues/failure. The last caveat is that we are assuming the students and professor can follow our directions to input the port/IP address; in other words, this is not a download and done solution, there is some user input required.

We have also included a tutorial for how to create and upload your own custom database as seen in Reetu & Friends solution, should the instructor trust the classroom bandwidth to handle it. It can be seen on the wiki here. This solution uses iRODS, which is what Cyverse uses to move data to/from their data storage. We have made a tutorial for how to download iRODS which can be followed here. (INSERT LINK)

Team Members Contribution

Josh Forstedt – Linked independent master and worker VMs running docker containers together, running BLASTEasy's modified 1.0.12 Sequence Server with WorkQueue and bench-marking.

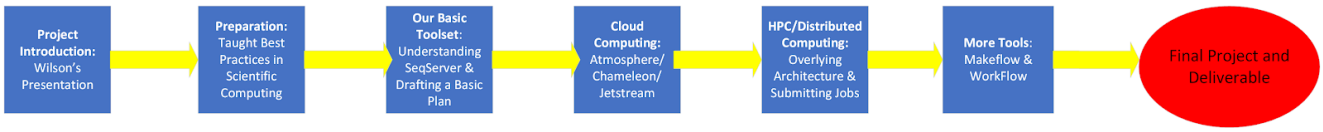
Derek Caldwell - keeping track of documentation, final presentation, bench-marking, progress presentations in class, and attending class collaboration meetings.

Nasser Albalawi - contributed in creating VMs to test bench-marking and Creating" Master Machine", created docker image for the master, having hands on makeflow and work queue attending class meeting, attending workshops, contributed with writing and presentation.

Asiedu Owusu-Kyereko - Contributed on making the docker container. Contributed with write-up, bench-marking, and presentation.

Special Thanks to John Xu, Sateesh Peri, and Team BLASTEasy.

Project Timeline



Bench-marking

For the following bench-marking portion, we had the entire team of 5 people run BLAST queries through our solution. Derek could not get an instance up and running in time, so he was considered to be the 'outlier', submitting jobs as a non-worker. This is why there is no core count next to his name in the graph. Nonetheless, Josh was able to start up two Ubuntu instances (4 core each) and satisfy the fifth worker requirement.

For the first nucleotide bench-marking, we took the random generator and had it create sequences from 1000- 500000 sequences long. What we found in the beginning was expected; the smaller the queries the quicker the times. We ended up not being able to break the solution or crash the browser; everything ran fine for the nucleotide sequences. This was not the case with the protein sequences.

For all benchmarks (nucleotide and protein) there were Three x 4 Core VMs and Two x 8 Core VMs as dedicated workers (28 threads) with a 2 Core VM set as the master node.

Times are in seconds and for simultaneous randomly generated (non-similar) runs. Times are for the rendered results to appear on the end-user's screen, not backend server complete times.

For this benchmark we each used a simple stopwatch.

Nucleotide Sequence Length	Nasser	Josh	Jaeden	Ace	Derek	**Average**
1000	0.55	0.4	1	2	0.19	**0.83**
2000	0.28	0.67	1.8	1.87	0.14	**0.95**
5000	2.49	1.32	2.1	2.5	0.17	**1.72**
15000	2.44	2.9	3.2	3.5	2.9	**2.99**
50000	7.04	6.41	7.5	7	7.6	**7.11**
100000	15.49	15.57	15.5	15.5	15.59	**15.53**
500000	56.1	56.43	56.6	57	56.8	**56.59**

To benchmark the protein sequences, we used an online random protein sequence generator that only gave us the ability to go from 1 - 100 sequences. In this test, we experienced increasingly diversified times depending on our devices. Those with more RAM in their machines had quicker times, so we recognize by the bench-marking process that something is running or processing on the local memory. The following table illustrates our results.

Protein 1000 (length) x n	Nasser	Josh	Jaeden	Ace	Derek	**Average**
n = 1	2.65	3.02	2.71	3.18	8.37	**3.99**
n = 10	17.49	17.47	16.82	21	53.35	**25.23**
n = 50	88.8	89.4	90	125	202	**119.04**
n = 100	181	183.4	185	325.3	468	**268.54**

Derek was running his benchmarks on a 4 year old Surface tablet and had much higher response times (skewing our average). Josh was running hardware monitor on his machine and noticed a spike (>20%) in CPU activity before results were rendered. This would indicate that time can vary greatly depending on the local CPU power (on 1 thread). It would be worth testing if John's solution of integrating nginx alleviates this bottleneck.

Presentation:

Google Slides - https://docs.google.com/presentation/d/1SOUsKjVtZrnL7GM0E0m_KDS_d3FSJZITMnHaoGh_rFs/edit?pli=1#slide=id.g73ce69d719_2_5

Post-Mortem Analysis

The Good: As far as the things that went right throughout this project, I think the team learned about some interesting tools and how they can be theoretically applied. We tried our best to help each other out if there were questions, and the support from other teams/classmates was crucial to our success.

What Could Have Been Done Better: I think the teams collective lack of experience held us back; without Josh we really do not have any solid command line experience. Maybe adding one more grad student to our team would have increased our chances at getting this done in a more timely fashion. Maybe we could have met more, either online or in person, to do the homework so we could get more out of the class. We could also have communicated more frequently throughout the project instead of the week of the due date.