BLASTer: A hub-based tool for bioinformatics

B. Cotton[1], **C. Thompson**[1], **and B. Raub**[1]
[1]Purdue University, West Lafayette, IN, USA

Abstract— *Basic Local Alignment Search Tool (BLAST) is a widely used sequencing tool in the bioinformatics community. BLAST searches are highly-parallelizable, lending themselves to computation on a high-throughput resource. In this paper, we describe the BLASTer tool developed by Purdue University to present an easy-to-use, web-based interface for performing BLAST searches. BLASTer uses the large HTCondor pool at Purdue University to dramatically shorten run times compared to desktop usage. Using the HUBzero platform, BLASTer can be made available for research and instruction to users worldwide. This paper describes the platform, the development process, early success stories, and efforts to grow the user community.*

Keywords: BLAST, SaaS, bioinformatics

1. Introduction

Since its initial release in 1990 [1], the Basic Local Alignment Search Tool (BLAST) has been a key part of bioinformatics research. The various programs that compose the BLAST package allow researchers to search for nucleotides or proteins in a database of known strings. The tens of thousands of citations indicate the popularity and broad utility of BLAST. It has been used in genome sequencing in a range of organisms, from e. coli [2] to humans [11].

Although the BLAST programs are freely available from the National Center for Biotechnology Information website [9], barriers to entry still exist. Long-running searches require the dedication of computing resources, often the desktop computer that a graduate student would otherwise be using for other work. Dedicated scientific computing resources alleviate this problem, but are expensive. Even if such a resource is available, it might not be approachable. Not all researchers are familiar with command line tools, presenting a challenging learning curve to those used to point-and-click interfaces. Other web-based tools, such as that offered by NCBI, alleviate this problem, but the resource pool is potentially small, large searches are often unsupported, and users generally do not have the ability to search against custom databases.

Our goal in creating the BLASTer tool was to address these issues. We wanted a tool that would easily handle large searches, both in terms of computation and storage requirements. Though powerful, the tool needed to be approachable by users. This is particularly important for use in an educational setting, where the goal is to teach the science, not the tool.

BLASTer combines existing components into a tool unique in its capability and capacity. BLASTer is powered by a large opportunistic computing resource that provides thousands of available cores. Users receive 10 gigabytes of storage, with 5 terabytes of centralized storage for NCBI databases and other common files. Future versions will incorporate visualization capabilities via Blast2Go.

Table 1: BLASTer compared to NCBI's web-based offering

	BLASTer	NCBI
Computation time	Unlimited	1 hour
Storage	10 GB default	10k characters

In addition to enabling single users, the BLASTer tool was intended to enable collaboration. Collaborators on a project should be able to share custom databases with each other. Researchers who wish to make their data publicly available should have the ability to easily publish files for use by others. Developing the user community allows mutually-beneficial knowledge sharing.

2. Computational Environment

2.1 Community Clusters

As part of its community cluster program [3], Purdue University opportunistically harvests idle cycles [10]. With millions of core hours per year available, a workflow that can make use of a high-throughput computing paradigm [6] is well-suited. Although work has been done to enable the use of MPI with BLAST [4], inter-node communication is not necessary to parallelize searches. BLAST queries can be separated and sent to individual nodes [7]. This enables BLAST searches to make ready use of opportunistic resources.

In addition to the nearly 50,000 cores provided by the community cluster program, this shared infrastructure brings additional benefits. A common application suite across the clusters ensures that the same version of the BLAST programs is on any given compute node. Shared storage allows the NCBI's standard databases to be available to all users without requiring file transfer. The BLASTer team can update BLAST binaries and NCBI databases without effecting currently-running jobs. These features mean that users only need to upload their search queries, saving effort and reducing the time to results.

2.2 HUBzero

In order to provide an easy-to-use interface for users that is accessible around the world, we decided to host BLASTer on the HUBzero Platform for Scientific Collaboration [8]. HUBzero provides a Java-based web interface for applications. The back end is a Linux server, allowing developers to make use of existing shared libraries and toolkits. The hub provides an interface to submit jobs directly into computational resources, leaving developers free to focus on application and workflow development.

The HUBzero platform also addresses other needs. Users can self-register and begin using BLASTer immediately. There is no application or allocation process that needs to be followed. Built-in feedback mechanisms allow users to quickly request help or register wishes for the application. Wishes can be commented and voted on by other users, giving developers a good way to gauge interest from the user community. Papers, data sets, and other files can be easily published by users. An in-hub user group allows the creation of wiki pages and provides a forum tool to enhance the community experience.

2.3 DiaGrid

DiaGrid is a HUBzero implementation dedicated to offering scientific applications to broad communities. Most DiaGrid applications, including BLASTer, are available for immediate use to the general public. DiaGrid applications can employ a high-throughput or high-performance paradigm by virtue of access to Purdue's HTCondor and community cluster resources. Existing applications on DiaGrid include SubmitR, a tool for submitting R jobs, and CryoEM, a tool for cryogenic electron microscopy. Tools to support molecular dynamics with GROMACS and climate modeling with the Community Earth System Model are under active development.

3. BLASTer Implementation

The HUBzero platform provides abstraction layers so that application developers do not need to be familiar with the target job scheduling system. A client-server system called "submit" receives job characteristics and files from the application and selects the appropriate venue. In the case of BLASTer, submit uses the Pegasus workflow engine [5] to submit jobs to the HTCondor resources. As jobs complete, Pegasus provides the output to submit, which sends output to the application in turn. This arrangement is presented in Figure 1.

3.1 GUI and Engine Development

To understand the design of BLASTer, one must first understand the requirements of the BLAST application suite. Instead of a single overall program, BLAST comprises a collection of independent executables, each of which

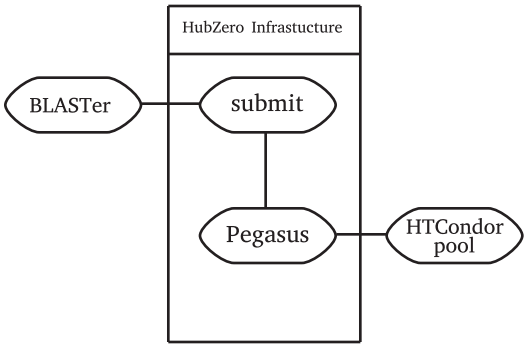

Fig. 1: The architecture of BLASTer and the HUBzero infrastructure.

uses a different search algorithm. A large portion of the BLAST community has also embraced analysis tools such as Blast2Go in production workflows. Without these additional tools, many of these users have told us they find no value in a new BLAST environment. BLASTer was designed to account for this wide range of requirements by abstracting job parameter entry, execution mechanics, and data inspection within the GUI.

The requirement of running multiple independent executables and various third-party tools makes it necessary for BLASTer to support arbitrary job descriptions. This is handled through a system of Job Profiles. Each Profile is built from a collection of Parameters organized into Parameter Groups as fits to the particular task at hand. Individual Parameter represents a particular argument of the job and has a value type. Simple value types such as integers and booleans are mapped to standard GUI widgets like text fields and checkboxes. Complex and novel value types can be added and represented with custom GUI components. At runtime, BLASTer will dynamically map a view of the Parameter list in the Profile using these widgets. Through this system any type of job can be represented to the user for option entry. Regardless of the argument format of the underlying programs, the user is presented with the same experience of an intuitive entry form customized to each job type.

To support the variety of execution needs of the many BLAST applications, an API is defined in BLASTer for common job control tasks. This abstracts all concrete details of execution away from the GUI and the user. Each type of job has a corresponding Job Engine following this API to provide access to common operations like submission, cancellation, and status retrieval. To add a new tool or even just a new execution method of an existing tool, developers only need to implement a new Engine which conforms to this API (and define a matching Job Profile for it). This allows BLASTer to hide from the user what is actually happening within the job through a set of common controls . An overall

Engine Manager tracks all available Engines in the system and handles routing commands from the GUI to the correct Engine for each job as well as funneling status updates back to the GUI. This system gives BLASTer the flexibility it will need to adapt to the changing and evolving BLAST application suite.

Just as important as the parameterization and execution of jobs is the effective communication of status and output. BLASTer tackles these problems by creating a standardized status reporting format in the Engine API as well as allowing the definition of custom GUI Job Viewer panels for each type of job performed. Just as each Profile lists which Engine to use for a job, they also list which Viewer represents a job. BLASTer uses this value at runtime to dynamically load the appropriate view for jobs a user has selected for closer inspection. With standardized status reports from the various Engines, BLASTer can also quickly communicate information to the user about any job they have created. Together, status reports and dynamic Viewers give the user both a bird's eye and ground level view of their work.

The primary mission of BLASTer is to abstract away the details of BLAST-related tools in an intuitive manner so users can focus on the science of their tasks. The design of BLASTer uses Job Profiles to dynamically support any set of arguments a task may require, Job Engines to handle any type of execution needs, and Job Viewer panels to communicate any format of data to the user. Via these three mechanisms, BLASTer can achieve its mission to execute these tasks on behalf of the user for the wide variety of BLAST applications.

3.2 Back end development

Back end development for BLASTer focused on making the best use of the computing resources available. With help from Rick Westerman, a Bioinformatics Specialist at Purdue University, we were able to gauge what a "typical" BLAST job would consist of and how the data could be parallelized. Westerman's Bioinformatics group pulls in gigabytes of raw sequence data daily and run BLAST jobs on their own servers. A typical job for this group consists of a 'fasta' file that contains biological sequences. Files are typically around 110,000 sequences, with each sequence containing between 200 and 12,215 bases, totaling about 66 million bases. A BLAST job of this size would take 8 or more hours to complete with the servers that the Bioinformatics group owns.

There are two methods of splitting data for execution. The first method splits the input file into smaller files and tests all files against a single database. The second method splits the database into parts and tests a single input file accordingly. The latter method requires MPI. With our intent to use HTCondor to harvest idle cycles, splitting the input file was clearly the most suitable option.

We assumed that a majority of the typical BLAST jobs would be roughly the size of Westerman's input file or smaller, so that was used as a baseline for splitting files. The goal was to achieve complete results within four hours of submission in order to minimize job preemption. On the community clusters, jobs arriving on a node from the primary (PBS) scheduler evict all HTCondor jobs on the node. Jobs that are evicted are automatically rescheduled by HTCondor. Although BLAST jobs respond well to rescheduling, because they must re-start from the beginning it is desirable to minimize the number of preemptions.

Testing was first done on the cluster nodes themselves to determine the optimal splits for the fastest completion times. Initial tests created jobs of 100 splits, 500 splits and 1,000 splits (1,100 sequences, 220 sequences and 110 sequences per split, respectively). With each test, the results were obtained faster and faster yet still not below the four hour threshold.

Beyond 2,000 splits, we observed a decay in completion time, suggesting we had surpassed the upper limit of file splits. This upper limit could have been for a few different reasons. When a job is submitted, all submissions are scheduled through one machine so the queue gets quite lengthy. Once all of the jobs are queued and running, each mini BLAST job compares its input file to the same database creating a bottleneck in file reads. The optimal split for such a job we determined to be about 85 sequences per file, thus equating to about 1,300 total splits.

Selecting 1500 splits per search provided us with a complete job that could run consistently within our allotted four hours. This splitting technique was further confirmed to be accurate with help from Professor Jianxin Ma and his graduate student Meixia Zhao in the College of Agriculture that had search times cut down from days to hours. Professor Andrew Dewoody and his graduate students from the Department of Biology also had similar speedups with their data analysis.

Table 2: BLAST job completion times

Sequences	Bases	Cluster wall time	HTCondor wall time
40K	15M	5:30	2:30
43K	16M	5:20	2:40
105K	49M	7:15	3:30
145K	72M	8:20	3:20

More recently, some users have used BLASTer to search against input files containing more than 700,000 sequences. The time required for each individual split to complete ran upwards of 15 hours, resulting in considerable delays as splits were repeatedly evicted from nodes. In order to accomodate these larger input files, BLASTer now splits files based on the method described in Table 3. Optimizing the balance between scheduler overhead and the node eviction rate is the primary challenge, and is the subject of ongoing testing.

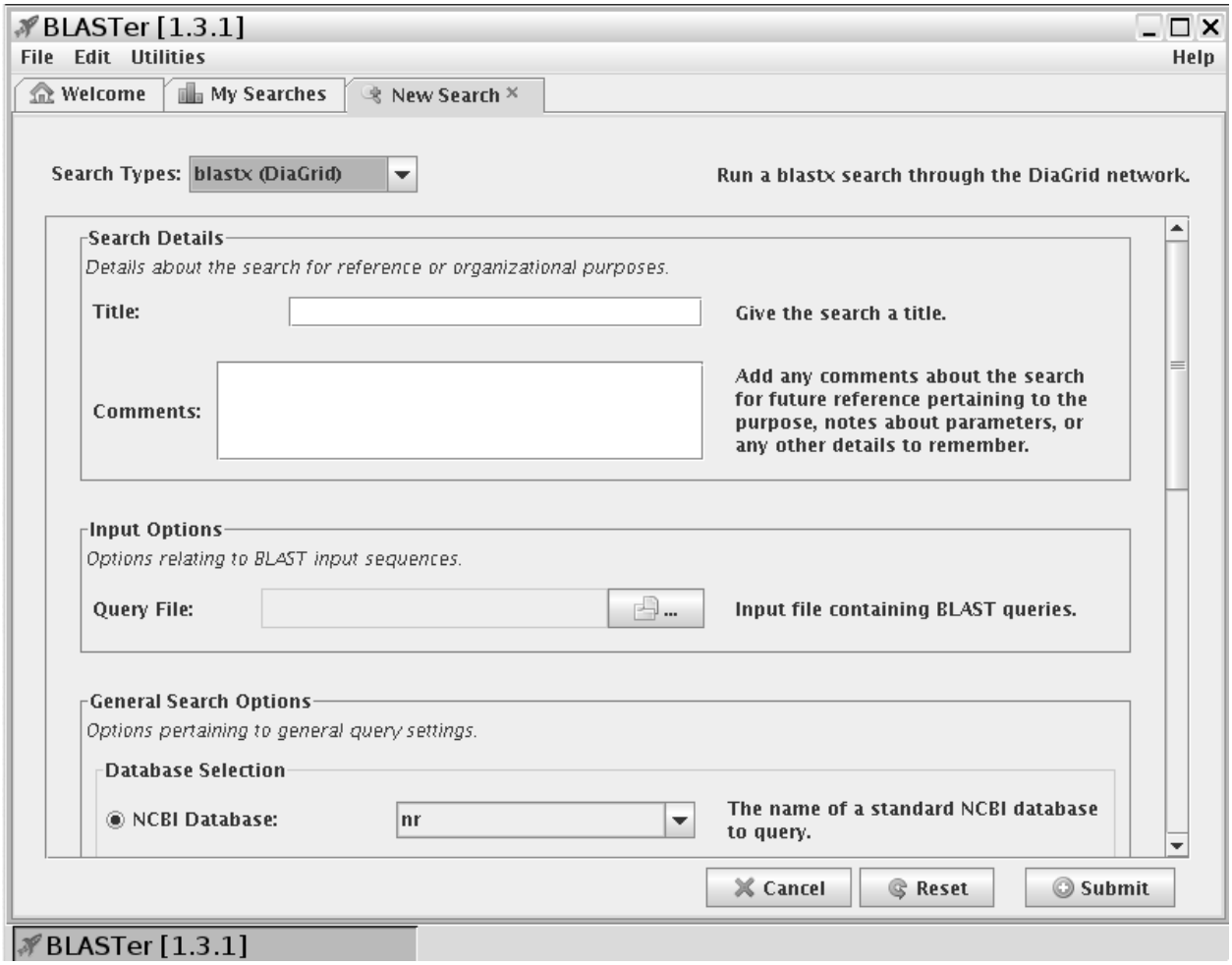

Fig. 2: BLASTer's configuration window for new searches.

Table 3: BLASTer search splitting

Sequences	Split size
<128k	85
128-750k	85 + (#sequences/750k)
>750k	160

4. Future Work

BLAST was selected as an early DiaGrid application because of the immediately-available local user base. The ability to have a core group of researchers that could help us with testing during development that we could also help by speeding up their research was key. One research group found they could complete a search in three hours that took three days on a desktop in their lab.

In BLASTer's first year, nearly 100 users have completed over a million BLAST runs. BLASTer clearly has technical merit, but much work remains to be done. Additional user-requested features are being implemented, including better management of databases and the ability to send output to the Blast2Go visualization tool. Improvements to the job splitting methodology are being investigated as well, to avoid over-splitting small jobs and ensure large jobs are not under-split.

As the BLASTer application approaches a feature-complete state, focus will shift toward engaging and growing the user community. Currently, users make light use of HUBzero's community features. Almost all wishes have been entered by DiaGrid developers in response to in-person conversations with users and wishes are rarely voted on. We expect that adding more non-Purdue users will help drive use of the community tools.

We have already created a comprehensive User Guide that focuses on walking a user through the basics of creating and managing a search. With local users, conducting periodic workshops with live demonstrations provides an easy way to reinforce the documentation and provide customized help. As the user community grows geographically, self-service training will be necessary. We plan to create short tutorial videos to provide a visual reference for the instructions in

the User Guide.

To date, usage has been driven mostly by the faculty, staff, and students at Purdue University. In order to have ongoing success, BLASTer usage should expand beyond the borders of a single campus. The development team will work to engage with potential users by presenting BLASTer at relevant conferences. Encouraging current users to recommend BLASTer to their colleagues at other institutions will also be a key part of the outreach strategy. Encouraging greater use of the hub's community tools should help make users more engaged and serve to make the community self-reinforcing.

5. Acknowledgments

The authors would like to acknowledge Rick Westerman for providing test cases, Professors Jianxin Ma and Andrew Dewoody and their respective research groups for being early testers, and Mike McLennan and the HUBzero team - especially Steve Clark - for their help in running the DiaGrid hub.

References

[1] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, D. J. Lipman, et al. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

[2] F. R. Blattner, G. Plunkett, C. A. Bloch, N. T. Perna, V. Burland, M. Riley, J. Collado-Vides, J. D. Glasner, C. K. Rode, G. F. Mayhew, et al. The complete genome sequence of Escherichia coli K-12. *science*, 277(5331):1453–1462, 1997.

[3] A. G. Carlyle, S. L. Harrell, and P. M. Smith. Cost-effective HPC: The community or the cloud? In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 169–176. IEEE, 2010.

[4] A. Darling, L. Carey, and W.-c. Feng. The design, implementation, and evaluation of mpiblast. *Proceedings of ClusterWorld*, 2003, 2003.

[5] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.-H. Su, K. Vahi, and M. Livny. Pegasus: Mapping scientific workflows onto the grid. In *Grid Computing*, pages 131–140. Springer, 2004.

[6] M. Livny, J. Basney, R. Raman, and T. Tannenbaum. Mechanisms for high throughput computing. *SPEEDUP journal*, 11(1):36–40, 1997.

[7] D. R. Mathog. Parallel blast on split databases. *Bioinformatics*, 19(14):1865–1866, 2003.

[8] M. McLennan and R. Kennell. HUBzero: A platform for dissemination and collaboration in computational science and engineering. *Computing in Science & Engineering*, 12(2):48–53, 2010.

[9] National Center for Biotechnology Information. BLAST+ Download. ftp://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/LATEST/.

[10] P. M. Smith, T. J. Hacker, and C. X. Song. Implementing an industrial-strength academic cyberinfrastructure at purdue university. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–7. IEEE, 2008.

[11] J. C. Venter, M. D. Adams, E. W. Myers, P. W. Li, R. J. Mural, G. G. Sutton, H. O. Smith, M. Yandell, C. A. Evans, R. A. Holt, et al. The sequence of the human genome. *Science Signalling*, 291(5507):1304, 2001.